Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations | Graduate School

2022

The l1-norm regularized l1-norm best-fit line problem and applications

Xiao Ling

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

 Part of the Operations Research, Systems Engineering and Industrial Engineering Commons, and the Philosophy of Science Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/6972

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

©Xiao Ling, May 2022

All Rights Reserved.

THE ℓ^1-NORM REGULARIZED ℓ^1-NORM BEST-FIT LINE PROBLEM AND APPLICATIONS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

XIAO LING

Doctor of Philosophy in Systems Modeling and Analysis

Director: Paul Brooks,

Professor, Department of Information Systems

Virginia Commonwealth University

Richmond, Virginia

May, 2022

Dedication

This dissertation is wholeheartedly dedicated to the memory of my grandparents.

Acknowledgements

I would like to first thank my advisor, Paul Brooks. I am deeply grateful for his mentorship and openness to my interests while also providing guidance and for generously sharing his ideas, skills, knowledge, and resources. I am constantly inspired by his diligence, persistence, and sharpness. He always held me to the highest standards with confidence in me, for which I will forever be grateful. I am convinced that my academic journey will not be possible without Prof. Brooks.

Besides, I would like to thank Prof. Bui, his work immediately leads to one of my projects. I appreciate the elegant concepts he has exposed me to. Moreover, thanks to Prof. Larson and Prof. Cano for the optimization concepts and CUDA concepts you have taught me, I would not be able to finish the 2 projects without your excellent lectures. I also thank them for their participation in the thesis committee and for their valuable time and helpful comments.

In addition to my advisors, I would like to thank some excellent people for opening my eyes to this wonderful academic journey. You inspired me and directly lifted my life trajectory. Reetam Majumder and Robert Clark thank you for your encouragement that motivated me to continue my doctoral studies.

Finally, I would like to thank my father for educating me in *"Scientia potentia est"*. I believe this is why I remain persistent in seeking knowledge.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Abstract

THE ℓ^1-NORM REGULARIZED ℓ^1-NORM BEST-FIT LINE PROBLEM AND APPLICATIONS

By Xiao Ling

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2022.

Director: Paul Brooks,
Professor, Department of Information Systems

The best-fit subspace or low-rank approximation of a data matrix revolves around the norm approximation technique. ℓ^2-norm criterion is probably the most widely used norm for fitting subspaces. As the computational power increases, the ℓ^1-norm analogue has recently gained attention from the academic community. It is widely agreed that the ℓ^1 norm is insensitive to outliers, compared to its ℓ^2 variant. Because of the polyhedral structure interrelated with linear programming (LP), the ℓ^0 norm is commonly relaxed into the ℓ^1-norm problem to induce sparsity in models. In this work, we examine the role of the ℓ^1-norm approximation in several domains. Our focus is on the underlying characterization of the ℓ^1-norm subspace, the parallel implementation of the algorithm, and its applications in the field of computer vision.

Chapter 2 develops a sparse and outlier insensitive method to fit a one-dimensional subspace that can be used as a replacement for eigenvector methods such as principal component analysis. The method is insensitive to outlier

observations by formulating procedures as optimization problems that seek the best-fit line according to the ℓ^1 norm. It is also capable of producing sparse principal components with an additional penalty term to take sparseness into account. Our algorithm has a worst-case time complexity of $O(m^2 n \log n)$. This chapter also presents the results of the implementation of this algorithm in the parallel and heterogeneous environment of NVIDIA CUDA and discusses the behavior of the algorithm in various settings. Our goal is to demonstrate the scalability and efficiency of our new approach.

Chapter 3 proposes an image denoising approach based on an ℓ^1-norm best-fit line algorithm. The denoising process is expressed as a best-fit subspace estimation problem, where the best-fit subspaces are derived in a ℓ^1-norm minimization framework. This new approach is competitive with existing approaches in terms of objective error metrics and visual fidelity and has the advantage that it can be implemented in parallel for large-scale applications. Numerical experiments illustrate that the technique can be successfully applied to the classical case of additive Gaussian noise. The performance of our approach is experimentally verified on a variety of images and noise levels.

Chapter 4 describes a method for denoising data using kernel principal component analysis (KPCA) that is able to recover preimages of the intrinsic variables in the feature space using a single line search along the gradient descent direction of its squared projection error. This method combines a projection-free preimage estimation algorithm with an ℓ^1-norm kernel PCA (KPCA). Those two stages provide distinct advantages to other KPCA preimage methods in the sense that it is insensitive to outliers and computationally efficient. The method can enhance the results of a range of unsupervised learning tasks such as denoising, clustering, and dimensionality reduction. Numerical experiments in the Amsterdam Library of

Object Images demonstrate that the proposed method performs better in terms of mean squared error than the ℓ^2-norm analogue as well as on toy synthetic data. The proposed method is applied to different data sets and the performances are reported.

CHAPTER 1

INTRODUCTION

Subspace fitting or low rank matrix approximation, which is the basis for applied problems such as pattern recognition, signal processing, and computer vision, is one of the main topics in data science. In this introduction, we describe the idea of ℓ^1-norm projection, low rank approximation, two sparse outlier-insensitive subspace estimation approaches, applications in computer vision, kernel principal component analysis, and the preimage Problem.

1.1 The ℓ^1-norm projection

Given a vector $v \in \mathbb{R}^m$, the ℓ^1-norm projection of $x \in \mathbb{R}^m$ onto the line $\{\alpha v : \alpha \in \mathbb{R}\}$ can be found by solving the following optimization problem

$$\min_{\alpha \in \mathbb{R}} \|x - v\alpha\|_1 \tag{1.1}$$

where x is a data point in $\mathbb{R}^m$ and $\|x\|_1$ or ℓ^1-norm of x is $\sum_{i=1}^m \|x_i\|$. (1.1) can be further cast as the following constrained linear program (LP):

$$\min_{\substack{\alpha \in \mathbb{R} \\ \lambda^+, \lambda^- \in \mathbb{R}^m}} \sum_{j=1}^{m} (\lambda_j^+ + \lambda_j^-), \tag{1.2}$$

subject to:

$$v_j \alpha + \lambda_j^+ - \lambda_j^- = x_j, j = 1, \ldots, m,$$

$$\lambda_j^+, \lambda_j^- \geq 0, j = 1, \ldots, m,$$

$$\alpha \quad \textit{unrestricted}.$$

The dual of (1.2) is

$$\max_{\pi \in \mathbb{R}^m} \sum_{j=1}^{m} \pi_j x_j, \tag{1.3}$$

subject to:

$$\sum_{j=1}^{m} \pi_j v_j = 0,$$

$$-1 \leq \pi_j \leq 1, j = 1, \ldots, m.$$

Given $v = (3, 2, 1, 5)^T$, the projection of $x = (2, -1, -3, 1)^T$ preserves the third coordinate (the element in v equal to 1). Therefore, we have $\alpha = -3$ and the distance is

$$\|x - v\alpha\|_1 = |2 + 9| + | - 1 + 6| + | - 3 + 3| + |1 + 15| = 32.$$

The reconstruction can be derived as

$$v\alpha = -3 \cdot (3, 2, 1, 5)^T = (-9, -6, -3, -15)^T.$$

Finally, the Euclidean distance from the origin to the projection or the length of the reconstruction is

$$\|v\alpha\| = \sqrt{81 + 36 + 9 + 225} = \sqrt{351}.$$

For two points $x_1, x_2 \in \mathbb{R}^m$, if they use the same unit directions with signs to project onto a line $\{\alpha v : \alpha \in \mathbb{R}\}$, then the optimal dual solutions for the projection linear program are the same. To prove this, we rewrite (1.2) in vector and matrix notation.

$$\min \mathbf{c^T x} \tag{1.4}$$

subject to:

$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{c} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ \vdots \end{pmatrix}$, $\mathbf{x} = \begin{pmatrix} \alpha \\ \lambda_1^+ \\ \lambda_1^- \\ \vdots \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}$, and $\mathbf{A} = \begin{pmatrix} v_1 & 1 & -1 & 0 & 0 & \cdots \\ v_2 & 0 & 0 & 1 & -1 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \end{pmatrix}$.

Let us partition x as $\begin{pmatrix} \mathbf{x_B} \\ \mathbf{x_N} \end{pmatrix}$, where $\mathbf{x_B}$ is the vector of m basic variables and $\mathbf{x_N}$ be the vector of $m+1$ non-basic variables. Similarly, divide $\mathbf{A}$ into $\mathbf{A} = (\mathbf{A_B}, \mathbf{A_N})$, where $\mathbf{A_B}$ is a matrix $m \times m$ corresponding to m basic variables, and $\mathbf{A_N}$ is a matrix $m \times (m+1)$ corresponding to the $m+1$ non-basic variables. Therefore, the optimal solution to (1.4) will be $\mathbf{x_B} = \mathbf{A_B^{-1}b}$. Any feasible basic solution will have α as basic because it is unrestricted in sign. The remaining $m-1$ basic variables will be from each pair $(\lambda_j^+, \lambda_j^-)$ for $j \neq \hat{j}$ for some preserved directions. Given that two points use the same unit directions with signs, $\mathbf{A_B}$ in two LPs will be the same. The dual solution $\pi_\mathbf{B}^\mathbf{T} = \mathbf{c_B^T A_B^{-1}}$ shall also be the same as each other.

1.2 Principal Component Analysis, SharpEL and Kernel Principal Component Analysis

Principal component analysis (PCA) is one of the most popular methods in subspace estimation. Given a data matrix $X \in \mathbb{R}^{n \times m}$ with zero column means, where n and m denote the number of points and the dimension of the original input space. PCA provides a rank r approximation on criterion ℓ^2, by solving the problem for each value of r.

$$\min_{U,V} \|X - AV^T\|_2^2 \quad s.t. V^T V = I_r \tag{1.5}$$

where, $V \in \mathbb{R}^{m \times r}$ is the principal subspace whose columns are the basis (principal components), $A \in \mathbb{R}^{n \times r}$ is the coordinates of projected points. (1.5) can be inter-

preted as minimizing the sum of squared distances of points from their projection in a subspace of dimensions $r < m$. The solution to (1.5) is the solution of following maximization problem.

$$\max_V \|XV\|_2^2 = \max_V V^T \Sigma V \quad s.t. V^T V = I_r [64] \tag{1.6}$$

where $\Sigma = \frac{X^T X}{n}$ is the covariance matrix of X. The columns of solution V are the eigenvectors of Σ. In another words, $\lambda v = \Sigma v$, v must lie in the span of all points $x_i, i = 1 \cdots n$. Thus, $\lambda x_i^T v = x_i^T \Sigma v$ is always true for each point. The kernel principal component analysis (KPCA) is PCA in a higher-dimensional space $\mathcal{F}$ constructed by the feature map $\Phi : \mathbb{R}^m \mapsto \mathbb{R}^N (N \gg m)$. Given that the data mapped into $\mathcal{F}$ with Φ, KPCA (of ℓ^2-norm) is naturally the problem to find the eigenvectors of $\bar{\Sigma} = \frac{\Phi(X)^T \Phi(X)}{n}$[73]. In a similar vein, $\lambda \Phi(x_i)^T v = \Phi(x_i)^T \bar{\Sigma} v$. For all feature representation $\Phi(x_i)$, $\lambda \Phi(x_i)^T \alpha^T \Phi(x_i) = \Phi(x_i)^T \frac{\Phi(X)^T \Phi(X)}{n} \alpha^T \Phi(x_i)$ is always true,where α are linear combination coefficients. Most often, the map Φ is implicitly defined; therefore, a predefined function should be chosen as representations with respect to $\Phi(x)$. [7] first remarked on the connection between a kernel and the dot product in another space such that $k(x, x')_{\mathcal{X}} = \langle \Phi_i(x), \Phi_i(x') \rangle_{\mathcal{F}}$, where $k(.,.)$ is a similarity function giving rise to a Gram matrix K in $\mathcal{F}$. Therefore, KPCA can be solved by $n\lambda K\alpha = K^2 \alpha$. [39] introduces the ℓ^1-norm kernel principal component analysis problem,

$$\max_v \sum_{i=1}^{n} \|\Phi(x_i)^T v\|_1 \quad s.t. \|v\|_2 = 1 \tag{1.7}$$

where v is principal component in $\mathcal{F}$ (kernel principal component) and $\Phi(x_i)$ is x_i's feature representation. The solution can be approximated by a modified Power iteration method [29] in a finite number of steps.

The problem (1.5) can be modified into the following ℓ^1-norm minimization

problem. Consider the optimization problem to find an ℓ^1-norm best-fit one-dimensional subspace:

$$\min_{v,\alpha} \sum_{i \in n} \|x_i - v\alpha_i\|_1. \tag{1.8}$$

where x_i, $i \in n$ are given points in $\mathcal{R}^m$. An approximate solution vector v^* determines a line through the origin corresponding to the best-fit subspace. An algorithm called SharpEL is proposed to approximate v by calculating a series of weighted medians [10]. Each element of a component v can be calculated independently by sorting several lists of ratios. SharpEL can be used to generate the basis for robust principal component analysis. An optimal solution to (1.5) is an optimal solution to (1.6). However, the same relationship does not hold for (1.7) and (1.8) because of the use of the ℓ^1 norm.

1.3 Low-Rank Approximation

The best-fit subspace problem is closely related to the low-rank approximation problem. There are growing needs for a robust and sparse alternative of PCA, and this has led to an active research area in low-rank approximation by leading scholars. In this section, we will describe the general idea of the low-rank approximation and some popular methods for solving the problems. We define the nuclear norm of L, $\|L\|_*$, as the sum of the singular values of L. Minimizing $\|L\|_*$ encourages L to be low rank. Low-Rank approximation is based on the assumption that a matrix X can be decomposed into a low-rank matrix L and a sparse matrix S, namely $X = L + S$. Minimizing the ℓ^1 norm of S encourages sparsity. For a matrix $A \in \mathcal{R}^{m \times n}$, $\|X\|_{2,1} = \sum_{i=1}^{m} \sqrt{\sum_{j=1}^{n} X_{ij}^2}$. The $\|X \cdot\|_{2,1}$ can force L to have zero columns corresponding to outliers.

Given a collection of centered data points $X \in \mathbb{R}^{n \times m}$, the low-rank approxima-

tion is a minimization mathematical problem, as in

$$\min_{L} \|X - L\|_p \qquad s.t.\ rank(L) \leq r$$

where $\|\cdot\|_p$ can be ℓ^1, ℓ^2 norm, etc, under the assumption that a data matrix X can be decomposed into a low-rank matrix L. The problem is, in general, convex. For example, the low-rank approximation problem Principal Component Pursuit (PCP) problem and Low-Rank and Block-Sparse Matrix Decomposition (LRBS) can be formulated as:

$$\text{PCP} : \min_{L,S} \|L\|_* + \lambda\|S\|_1 \quad s.t. \qquad L + S = X. \tag{1.9}$$

$$\text{LRBS} : \min_{L,S} \|L\|_* + k\lambda\|S\|_{2,1} + k(1-\lambda)\|L\|_{2,1} \quad s.t. \qquad L + S = X. \tag{1.10}$$

where L is low-rank matrix and S is the sparse matrix. The (1.9) and (1.10) problem are convex and can be solved by various methods. An augmented Lagrange multiplier (ALM) algorithm used in [17] and an alternating direction method (ADM) algorithm used in [83]. ALM and ADM methods use the following Lagrangian function of (1.9):

$$\mathcal{L}(L, S, Z) = \|L\|_* + \lambda\|S\|_1 + Z^T(X - L - S) + \frac{\beta}{2}\|X - L - S\|_F^2, \tag{1.11}$$

where β is a positive penalty parameter. The difference between ALM and ADM is that ALM minimizes (1.11) with respect to L and S, by setting $(L^{j+1}, S^{j+1}) = \arg\min \mathcal{L}(L, S, Z^j)$ and updating the Lagrangian multiplier matrix $Z^{j+1} = Z^j + \beta(X - L^j - S^j)$[51]. [59] introduces a related variant ALM scheme. In contrast, ADM sets $L^{j+1} = \arg\min \mathcal{L}(L, S^j, Z^j)$, $S^{j+1} = \arg\min \mathcal{L}(L^j, S, Z^j)$ and $Z^{j+1} = Z^j + \beta(X - L^j - S^j)$. The form of L^{j+1} and S^{j+1} usually have closed form solutions [97, 49, 16, 48]. (1.10) can be solved by ALM by minimizing the following Lagrangian

function of (1.10)[82]:

$$\mathcal{L}(L,S,Z) = \|L\|_* + k\lambda\|S\|_{2,1} + k(1-\lambda)\|L\|_{2,1} + Z^T(X-L-S) + \frac{\beta}{2}\|X-L-S\|_F^2. \quad (1.12)$$

The problem (1.9) is convex and can be solved by ALM-based algorithms. [44] developed an updating scheme with closed-form solutions at each ALM iteration. An ALM variant called DNDP-ALM proposed in [18] improved the original optimization problem and incorporated noise into the constraints. [3, 78] solves PCP by taking advantages of the multi-block structure. [47] introduce the exact and inexact ALM methods that achieves a good performance in solving the PCP problem. [66] proposed an algorithm approximately an order of magnitude faster than inexact ALM to construct a sparse component of the same quality. [77] described a simple and almost parameter-free algorithm by reformulating the PCP as an unconstrained nonconvex program and then performing alternating minimization scheme. A parallel splitting ALM method was introduced in [52]. [30] described an ADM algorithm is able to achieve global convergence under standard assumptions. [53] proposes the first linear time algorithm for exactly solving very large PCP problems. A scalable algorithm proposed in [50] is able to generate suboptimal solution to PCP. [31] combines an ADM with a Gaussian back substitution procedure [100] to solve the PCP. MATLAB LRSLibrary [79] provides various algorithms and variants to the PCP problem. [22] presented a method for robust principal component analysis (RPCA) that can be used for automatic learning of subspace for data. [67] propose a simple alternating minimization algorithm for solving a minor variation on the original Principal Component Pursuit (PCP). Under the same assumption to PCP, similar problem formulations have been studied.

[95] described a problem $\min_{L,S} \|L\|_* + \lambda\|S\|_{1,2}$ called Outlier Pursuit can be efficiently solved by proximal gradient algorithm ([16]). [36] presented a more robust and less biased nonconvex formulation and solved using augmented Lagrange multiplier framework. [101] described a novel low-rank and sparse decomposition problem called Go Decomposition (GoDec).

The rank-one components can be obtained by the singular value decomposition factorization of L, which will be used to compare with the Algorithm 1 results in later experiments.

1.4 Related Approaches to Sparse Robust Subspace Estimation

For comparison, we also examined a different approach, robust sparse principal component analysis [57] (RSPCA), a method that maximizes ℓ^1-norm variance. We will explain the relationship and examine the difference between RSPCA and our algorithm the ℓ^1-norm regularized ℓ^1-norm best-fit lines (Algorithm 1, in Chapter 2). Taking a collection of centered data points $X \in \mathbb{R}^{n\times m}$, the two problems used to derive heuristic algorithms RSPCA [57] and Algorithm 1 are formulated as follows,

$$\text{RSPCA: } \max_{\substack{v^T v=1 \\ \|v\|_1<t}} \|Xv\|_1, \tag{1.13}$$

$$\text{Algorithm 1: } \min_{v,\alpha} \|X - \alpha v^T\|_1 + \lambda\|v\|_1, \tag{1.14}$$

where v is the optimal i^{th} one-dimensional best-fit subspace. One may notice that αv^T is a rank one matrix in the (1.14) problem formulation. Both optimizations achieve robustness and sparseness by applying the ℓ^1 norm to the objective functions, along with additional penalties. The formulations show that the two approaches attack the problem from different perspectives. The RSPCA objective

function maximizes the ℓ^1-norm of ℓ^2-norm projections. On the other hand, the objective goal of Algorithm 1 is to minimize the reconstruction error along with a penalty. λ is a parameter that controls the sparsity of the solution through the ℓ^1 norm of the solution. Thus, we define that this penalty is the ℓ^1-norm regularized by λ. Note that the solutions to (1.13) and (1.14) are different, because they are not dual to each other.

Both approaches use heuristic approximation algorithms, the difference being that an iteration algorithm is used to approximate a reasonable sparse local maximum solution v in RSPCA. On the other hand, Algorithm 1 is carried out by several independent sortings; therefore, it is deterministic, scalable, and suitable for parallel or distributed implementations. However, the one-dimensional subspace fitting algorithm for RSPCA is not scalable, which means that it is not suitable for big data. Furthermore, the greedy algorithm optimizes the projection directions one by one, making it easy to get stuck in a local solution.

The computational complexity of the RSPCA approximation algorithm is around $O(nm \log m)k$ and the Algorithm 1 approximation algorithm is $O(m^2 n \log n)$, where k is the number of iterations for convergence. An important issue for RSPCA is that random initialization does not guarantee convergence to a better local optimal. Therefore, additional computational complexity will be involved if a good initialization is required, such as a PCA input v.

As we will demonstrate, Algorithm 1 has some distinct advantages in that it does not depend on initialization, is deterministic and scalable. More importantly, the Algorithm 1 can be processed in parallel for increased efficiency, which is suitable for implementation in a distributed or parallel framework.

1.5 Applications in Computer Vision

The best-fit subspace or low-rank approximation algorithms have been widely used in the field of computer vision, such as background subtraction (modeling) and image denoising. Background subtraction is the process of separating the foreground objects (non-static objects) from the background scene. And image denoising is the process of separating the noise with true pixels as much as possible.

The connection between image denoising and ℓ^1 norm optimization algorithms can be traced to a method called sparse coding. The basic idea of sparse coding is to choose a small number of components in a dictionary (i.e. principal components) learned by some matrix decomposition frameworks to estimate the signal of interest [45, 76, 81]. Low-rank approximation also finds its applications in image denoising [33, 34, 99]. A group of neighborhood patches is the input matrix used to learn a dictionary using low-rank approximation techniques. Here, we first introduce several popular patch-based image denoising methods, which will be used as benchmarks in later experiments. We first define a grayscale image or a frame as a two-dimensional data matrix with each entry ranging from 0 to 255. The higher the value, the brighter this pixel is. Given a corrupted data matrix $X = \hat{X} + E$ with observations stacked in rows, where $\hat{X}$ represents an implicit true signal usually of low rank and E represents noise. $\hat{X}$ are the denoised patches (a collection of subregions of an image) for image denoising. NLM [14] outlined an approach that is considered the foundation of many patch-based methods for image denoising as an alternative to the pixel-wise bilateral filter [6, 25, 86, 5]. The method replaces a target patch with an average of similar patches in a specified search window. In extension, Deledalle et al. [23] expressed the problem as a weighted maximum likelihood estimation problem (PPB). Aharon et al. [1] presented an iterative dic-

tionary learning algorithm (K-SVD) by seeking the sparse linear combination of columns (atoms) of an adaptive dictionary that is learned by solving a low-rank approximation problem[26]. Later in 2007, Dabov et al. [21] proposed the widely adopted BM3D denoising method. In this method, the grouped patches in three dimensions are transformed into a wavelet domain and then filtered by a Wiener filter to estimate the denoised patches. The results of computational experiments have shown that K-SVD and BM3D are among the best denoising algorithms in terms of the peak signal-to-noise ratio (PSNR) and visual fidelity. Deledalle et al. [23] simplified dictionary generation by employing PCA in collections of overlapping patches with excellent results using Patch based Global PCA (PGPCA) and Patch based Local PCA (PLPCA) [24]. PGPCA constructs a dictionary by carrying out PCA on the whole compilation of patches extracted from the image. PLPCA constructs local dictionaries by performing PCA on a small group of patches extracted from a series of predefined windows of the image.

In background subtraction applications, pixels that have not changed or have changed gradually are considered as background, and pixels that have changed dramatically are considered as foreground objects. The low-rank approximation algorithms find their applications in background subtraction in the sense that low-rank subspaces naturally represent a gradual change over points (frames). [17] demonstrates the effectiveness of the Principal Component Pursuit in its background subtraction experiment. Practically, pixels that have not changed could be part of moving objects, for example, if they have the same color as the background. Similarly, pixels that have changed could be part of the background when illumination changes occur, for example.

For image denoising and background subtraction applications, the number of features is usually large (e.g., the resolution of an image or a frame). Therefore, a

scalable algorithm will be more efficient in terms of running time. Moreover, [87] points out challenging situations in background subtraction such as illumination changes, intermittent moving foreground moving background and etc. A robust method is more suitable for tackling such challenges. In light of this, we incorporate Algorithm 1 into background subtraction and an image denoising framework to tackle those challenges.

1.6 Kernel Principal Component Analysis and the Preimage Problem

The last chapter describes a method that applies a kernel ℓ^1 norm principal component analysis to a novel preimage estimation procedure called projection-free kernel principal component analysis (PFKPCA_2). Kernel principal component analysis (KPCA) seeks the best-fit linear subspace embedded with some underlying structures in a higher-dimensional space. Suppose that we are given a point (input, pattern, or observation) $x \in \mathcal{X} \subseteq \mathbb{R}^m$, with an image $\Phi(x)$ in the feature space $\mathcal{F}$ a higher-dimensional space constructed by the feature map $\Phi : \mathbb{R}^m \mapsto \mathbb{R}^N (N \gg m)$. Most often, the map Φ is implicitly defined; therefore, a predefined function should be chosen as representations with respect to $\Phi(x)$. [7] first remarked on the connection between a kernel and the dot product in another space such that

$$k(x, x')_{\mathcal{X}} = \langle \Phi_i(x), \Phi_i(x') \rangle_{\mathcal{F}}, \tag{1.15}$$

where $k(.,.)$ is a similarity function giving rise to a Gram matrix K in $\mathcal{F}$. In one example, [91] used a predefined kernel for the dot product in a classifier decision function. For kernels satisfying Mercer's condition, there exists an implicit Φ [74]. Equation 1.15 enables us to compute dot products in the feature space by means of the inputs without any knowledge of Φ. Accordingly, it was shown that the normalized feature principal score of point x is a linear combination of feature repre-

sentations in the work by [71]. The discussion so far has concentrated on PCA in feature space. However, in some applications, such as denoising or compressing, the reconstruction of the KPCA results in the input space would be of primary interest. This is the so-called preimage problem as put forward in [72]. It is studied as finding an approximated x in the input space such that the function

$$f(z) = \|\Phi(z) - P_n\Phi(x)\|^2 \tag{1.16}$$

is minimized, where P_n is a projection operator operating on the top n eigenvectors. With a small modification to the formulation 1.16, they used the gradient method to find the preimage. Moreover, Schölkopf et al. [72, 74] pointed out that an exact preimage does not exist in general. [58] later proposed the fixed-point iteration algorithm, which suffers from numerical instability and is limited by radial basis kernels. [42] proposed a non-iterative method inspired by the connection between kernel PCA and metric multidimensional scaling sacrificing efficiency. [69] modified the preceding method only involving the feature distance. In a similar vein, [32] proposed a more efficient method without computing the distance in both spaces. [15] described a projection-free method based on the observation that the true manifold is typically nearly parallel to the level curve of the function $\|\Phi(\cdot) - P_n\Phi(\cdot)\|^2$. This algorithm does not try to attack the nonlinear optimization problem itself, rather it simply mimics the line search in two spaces. In Chapter 4, we give a geometric interpretation that provides some insights on the relation between the error surface and the preimages. We then develop a method based on the kernel ℓ^1-norm principal components to increase insensitivity to outliers.

CHAPTER 2

THE ℓ^1-NORM REGULARIZED ℓ^1-NORM BEST-FIT LINES

2.1 Motivation

Subspace estimation can be used for dimension reduction by projecting data in a high-dimensional feature space to a low-dimensional subspace. It sheds light on a broad range of tasks from computer vision to pattern recognition. Conventional principal component analysis (PCA), hereafter referred to as ℓ^2-PCA, is a widely used technique to finding a best-fit subspace. ℓ^2-PCA produces the linear combinations of the original features such that the combinations capture maximal variance. ℓ^2-PCA can be computed via the singular value decomposition (SVD) of the data matrix. The ℓ^2 metric is sensitive to outliers in the data matrix. A solution to this disadvantage is replacing the ℓ^2 metric with an ℓ^1-norm analog [40, 17, 13, 56, 37]. Another drawback of PCA is that it is difficult to interpret the principal components (PCs) without domain knowledge, for example, in movie recommendation data a linear combination of adventure, historical, and action might come up with western genre. But, difficulty arises when the dimension increases. To help with interpretation, we can encourage sparsity in the PC loadings. There have been many works applying ℓ^1-regularization to a variety of problems since [85] proposed the LASSO method for regression problems. [92] demonstrated the significance and efficacy of the ℓ^1-regularization as a vehicle of inducing sparsity. A simple and intuitive definition of sparsity of data is the number of nonzero entries in the dataset, quantified by ℓ^0 norm.

In this chapter, we propose an algorithm with modest computational require-

ments for ℓ^1 regularization with the traditional squared ℓ^2-norm cost replaced by the ℓ^1 cost. Consider the optimization problem to find an ℓ^1-norm regularized ℓ^1-norm best-fit one-dimensional subspace:

$$\min_{v,\alpha} \sum_{i \in N} \|x_i - v\alpha_i\|_1 + \lambda \|v\|_1, \tag{2.1}$$

where x_i, $i \in N$ are points in $\mathbb{R}^m$. An optimal vector v^* determines a line through the origin corresponding to the best-fit subspace. For each point x_i, the optimal coefficient α_i^* specifies the locations of the projected points $v\alpha_i$ on the line defined by v^*. Due to the nature of the ℓ^1 norm, some components of v will be reduced to zero if λ is large enough. Therefore, our proposed method simultaneously generates both a best-fit and a sparse line in m dimensions, which makes it suitable for large or high-dimensional data. The method can be extended to the problem of fitting subspaces. The problem in (2.1) is non-linear, non-convex, and non-differentiable. Therefore,we adapt the approximation algorithm of [13] to the regularized problem.

2.2 Related Works

Boscovitch outlined an algorithm with complexity $\mathcal{O}(n^2)$ for data in two dimensions by evaluating a simple ratio at each point to find a line that best fits the n points in the least absolute deviation (LAD) context in 1760. Later in 1887, Edgeworth came up with a famous weighted median solution enlightened by Laplace. [37, 38] presented an alternative minimization algorithm using weighted median and convex quadratic programming with random initialization. [12] described a non-convex polynomial-time algorithm for finding an ℓ^1-norm best-fit hyperplane using LP. [80] propose a polynomial-time algorithm to approximate the ℓ^1-low-rank subspace. [10] demonstrate an equivalence between their approach, that of

[90], and that of [13]. [28] recently showed that finding an ℓ^1-norm best-fit line is NP-hard.

To the best of our knowledge, the ℓ^1-regularized ℓ^1-norm best-fit line problem has not been directly attempted. However, ℓ^1-regularized LAD regression(ℓ^1-LAD) is a quite active area. ℓ^1-LAD is a special case of optimal subspace fitting to data. In the regression context, regularization can support variable selection, while error measurement is designed to be insensitive to outliers in the response variable. [92] is the first attempt to combine LAD and Lasso in the regression sense. The weighted version of the ℓ^1-LAD methods, such as [93] studied a near-oracle performance method to fit an ℓ^1-LAD. Recently, [61] showed us an iterative algorithm using the parametric simplex method to solve this problem.

2.3 Problem Formulation

In this section, we will extend the sorting method introduced in [13] to the setting where we add a penalty for sparsity. First, we introduce four sets of goal variables $\epsilon_{ij}^+, \epsilon_{ij}^-$ and ζ_j^+, ζ_j^-. The optimization problem in (2.1) can be recast as the following constrained mathematical program.

$$\min_{\substack{v\in\mathbb{R}^m,\alpha\in\mathbb{R}^n \\ \epsilon^+,\epsilon^-\in\mathbb{R}_+^{n\times m}, \\ \zeta^+,\zeta^-\in\mathbb{R}_+^m}} \sum_{i\in N}\sum_{j\in M}(\epsilon_{ij}^+ + \epsilon_{ij}^-) + \lambda\sum_{j\in M}(\zeta_j^+ + \zeta_j^-), \tag{2.2}$$

subject to:

$$\begin{aligned} v_j\alpha_i + \epsilon_{ij}^+ - \epsilon_{ij}^- &= x_{ij}, i \in N, j \in M, \\ v_j + \zeta_j^+ - \zeta_j^- &= 0, j \in M, \\ \epsilon_{ij}^+, \epsilon_{ij}^-, \zeta_j^+, \zeta_j^- &\geq 0, i \in N, j \in M. \end{aligned}$$

Proposition 1. *The formulation* (2.2) *is equivalent to* (2.1).

Proof. The presence of absolute values in the objective function can be avoided by replacing each $x_{ij} - v_j\alpha_i$ with $\epsilon_{ij}^+ - \epsilon_{ij}^-, \epsilon_{ij}^+, \epsilon_{ij}^- \geq 0$ and each v_j with $\zeta_j^+ - \zeta_j^-, \zeta_j^+, \zeta_j^- \geq 0$, and these become the constraints. The new objective function $\sum_{i=1}^n \sum_{j=1}^m |\epsilon_{ij}^+ - \epsilon_{ij}^-| + \lambda|\zeta_j^+ - \zeta_j^-|$ can be replaced with $\sum_{i=1}^n \sum_{j=1}^m (\epsilon_{ij}^+ + \epsilon_{ij}^-) + \lambda(\zeta_j^+ + \zeta_j^-)$. This linear program will have an optimal solution with at least one of the values in $\epsilon_{ij}^+, \epsilon_{ij}^-$ and ζ^+, ζ^- is zero respectively. In that case, $x_{ij} - v_j\alpha_i = \epsilon_{ij}^+$, if $x_{ij} - v_j\alpha_i > 0$, and $x_{ij} - v_j\alpha_i = -\epsilon_{ij}^-$, if $x_{ij} - v_j\alpha_i < 0$. $v_j = \zeta_j^+$, if $v_j > 0$, and $v_j = -\zeta_j^-$, if $v_j < 0$. Any feasible solution for (2.2) generates an objective function value which is the same as that of (2.1) using the same vlaues for v and α, and vice-versa. Therefore, an optimal solution to (2.1) generates a feasible solution for (2.2) and vice-versa. □

An optimal solution to (2.2) will be a vector $v^* \in \mathbb{R}^m$, along with scalars α_i^*, $i \in N$. For each point i, the feature j, two pairs (ϵ_{ij}^{+*},ϵ_{ij}^{-*}) reflect the distance along each unit direction j between the point and its projection. The pairs $(\zeta_j^{+*}, \zeta_j^{-*})$ provide the difference from zero for each coordinate of v^*.

Proposition 2. [11] *Let $v \neq 0$ be a given vector in $\mathbb{R}^m$. Then there is an ℓ^1-norm projection from the point $x \in \mathbb{R}^m$ on the line defined by v that can be reached using at most $m-1$ unit directions.*

Proof. A proof is in [11]. □

2.4 Estimating an ℓ^1-norm regularized ℓ^1-norm best-fit line

The modification is to impose the preservation of one of the coordinates,$\hat{j}$, in the projections of the n points originated in the work of [90, 20]. This means that each point will use the same $m-1$ unit directions to project onto the line defined by v. The modification will give us a linear program. By modifying the mathematical program in (2.2), we can obtain an estimation of an ℓ^1 regularized ℓ^1-norm best-fit

line.

By the proof of Proposition 2, if $x_{\hat{\jmath}} \neq 0$, then $v_{\hat{\jmath}} \neq 0$[11]. Therefore, we can set $v_{\hat{\jmath}} = 1$ and set $\alpha_i = x_{i\hat{\jmath}}$ to preserve $\hat{\jmath}$ without loss of generality for the error term, though the regularization term is affected. We will optimize over lines defined by v preserving a direction $\hat{\jmath}$ with $v_{\hat{\jmath}}$.

The remaining components can be found by solving an LP.

$$z_{\hat{\jmath}}(\lambda) = \min_{\substack{v\in\mathbb{R}^m, v_{\hat{\jmath}}=1 \\ \epsilon^+,\epsilon^-\in\mathbb{R}^{n\times m}, \\ \zeta^+,\zeta^-\in\mathbb{R}^m}} \sum_{i\in N}\sum_{j\in M}(\epsilon_{ij}^+ + \epsilon_{ij}^-) + \lambda\sum_{j\in M}(\zeta_j^+ + \zeta_j^-), \tag{2.3}$$

subject to:

$$\begin{aligned} v_j x_{i\hat{\jmath}} + \epsilon_{ij}^+ - \epsilon_{ij}^- &= x_{ij}, i \in N, j \in M; j \neq \hat{\jmath}, \\ v_j + \zeta_j^+ - \zeta_j^- &= 0, j \in M, \\ \epsilon_{ij}^+, \epsilon_{ij}^-, \zeta_j^+, \zeta_j^- &\geq 0, i \in N, j \in M. \end{aligned}$$

Each of the n data points generates $2m - 1$ constraints in this LP. By solving these m LPs and selecting the vector v from the solutions associated with the smallest values of the objective function m, we will have the ℓ^1-norm regularized ℓ^1-norm best-fit line under the assumption that all points project by preserving the same coordinate and $v_{\hat{\jmath}}$=1. The following lemma describes how to generate solutions to the LPs by sorting several ratios.

Lemma 1. *For data $x_i \in \mathbb{R}^m$, $i \in N$, and for a $\lambda \in \mathbb{R}$, an optimal solution to* (2.3) *can be constructed as follows. If $x_{i\hat{\jmath}} = 0$ for all i, then set $v = 0$. Otherwise, set $v_{\hat{\jmath}} = 1$ and for each $j \neq \hat{\jmath}$,*

- *Take points x_i, $i \in N$ such that $x_{i\hat{\jmath}} \neq 0$ and sort the ratios $\dfrac{x_{ij}}{x_{i\hat{\jmath}}}$ in increasing order.*
- *If there is a $\tilde{\imath}$ where $\left|sgn\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right)\lambda + \sum_{\substack{i\in N: \\ i<\tilde{\imath}}}|x_{i\hat{\jmath}}| - \sum_{\substack{i\in N: \\ i>\tilde{\imath}}}|x_{i\hat{\jmath}}|\right| \leq |x_{\tilde{\imath}\hat{\jmath}}|$, then set $v_j =$*

$\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}$.

- *If no such $\tilde{\imath}$ exists, then set $v_j = 0$.*

Proof. The problem is separable into m independent small sub-problems, one for each column j. For a given j, we can introduce goal variables ϵ_i^+, ϵ_i^-, ζ^+, and ζ^- to transform the problem for finding v_j into a linear program of the form

$$\min_{\substack{v_j,\epsilon^+,\epsilon^-,\lambda \\ \zeta^+,\zeta^-}} \sum_{i\in N}(\epsilon_i^+ + \epsilon_i^-) + \lambda(\zeta^+ + \zeta^-), \tag{2.4}$$

$$s.t. v_j x_{i\hat{\jmath}} + \epsilon_i^+ - \epsilon_i^- = x_{ij}, i \in N, \tag{2.5}$$

$$v_j + \zeta^+ - \zeta^- = 0. \tag{2.6}$$

Under the assumption that all points used the same unit direction $\hat{\jmath}$ to project onto a best-fit regularized line, we can set $v_{\hat{\jmath}} = 1$ in (2.1) and obtain the following optimization problem

$$\min_{v,\alpha,\lambda} \sum_{i=1}^{n}\sum_{j=1}^{m} |x_{ij} - v_j x_{i\hat{\jmath}}| + \lambda \sum_{j=1}^{m} |v_j| \tag{2.7}$$

$$= \min_{v,\alpha,\lambda} \sum_{i=1}^{n}\sum_{j=1}^{m} |x_{i\hat{\jmath}}| \left| \frac{x_{ij}}{x_{i\hat{\jmath}}} - v_j \right| + \lambda \sum_{j=1}^{m} |v_j|. \tag{2.8}$$

We will show that the solution for v_j stated in Theorem 2.1 is optimal by constructing a dual feasible solution with the same objective function value. Suppose that the ratios $\frac{x_{ij}}{x_{i\hat{\jmath}}}$, $i \in N$, are sorted in increasing order. The primal objective function value (2.4) becomes

$$\begin{array}{rl} \sum_{i\in N} |x_{i\hat{\jmath}}| \left| \frac{x_{ij}}{x_{i\hat{\jmath}}} - \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}} \right| + \lambda \left| \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}} \right| & \text{if } v_j \neq 0, \\ \sum_{i\in N} |x_{ij}| & \text{if } v_j = 0. \end{array}$$

The dual linear program to (2.4) is

$$\max_{\pi,\gamma} \sum_{i \in N} \frac{x_{ij}}{x_{i\hat{j}}} \pi_i, \tag{2.9}$$

$$s.t. \sum_{i \in N} \pi_i + \gamma = 0, \tag{2.10}$$

$$-|x_{i\hat{j}}| \leq \pi_i \leq |x_{i\hat{j}}|, i \in N, \tag{2.11}$$

$$-\lambda \leq \gamma \leq \lambda. \tag{2.12}$$

Suppose there is an $\tilde{\imath}$ satisfying (1). Then let $\gamma = -\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{j}}}\right)\lambda$ and let

$$\pi_i = \begin{cases} |x_{i\hat{j}}| & \text{if } i > \tilde{\imath}, \\ -|x_{i\hat{j}}| & \text{if } i < \tilde{\imath}, \\ -\gamma - \sum_{i \neq \tilde{\imath}} \pi_i & \text{if } i = \tilde{\imath}. \end{cases}$$

This solution satisfies complementary slackness. To show that the solution is dual feasible, we need to show that $\pi_{\tilde{\imath}}$ satisfies the bounds in (2.11) (all other bounds and constraints are satisfied):

$$\begin{aligned} |\pi_{\tilde{\imath}}| &= |-\gamma - \sum_{i \neq \tilde{\imath}} \pi_i|, & (2.13) \\ &= \left|\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{j}}}\right)\lambda + \sum_{i:i<\tilde{\imath}} |x_{i\hat{j}}| - \sum_{i:i>\tilde{\imath}} |x_{i\hat{j}}|\right|, & (2.14) \\ &\leq |x_{\tilde{\imath}\hat{j}}|. & (2.15) \end{aligned}$$

The inequality is due to (1). The dual solution has the following objective function

value.

$$\begin{aligned}
\textstyle\sum_{i\in N} \frac{x_{ij}}{x_{i\hat{\jmath}}}\pi_i &= \textstyle\sum_{i:i>\tilde{\imath}} \frac{x_{ij}}{x_{i\hat{\jmath}}}|x_{i\hat{\jmath}}| - \sum_{i:i<\tilde{\imath}} \frac{x_{ij}}{x_{i\hat{\jmath}}}|x_{i\hat{\jmath}}| + \\
&\quad \textstyle\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\left(-\gamma - \sum_{i\neq\tilde{\imath}}\pi_i\right), \\
&= \textstyle\sum_{i:i>\tilde{\imath}} |x_{i\hat{\jmath}}| \left(\frac{x_{ij}}{x_{i\hat{\jmath}}} - \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) + \\
&\quad \textstyle\sum_{i:i<\tilde{\imath}} |x_{i\hat{\jmath}}| \left(-\frac{x_{ij}}{x_{i\hat{\jmath}}} + \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) + \\
&\quad \lambda \text{sgn}\left(\textstyle\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}, \\
&= \textstyle\sum_{i\in N} |x_{i\hat{\jmath}}| \left|\frac{x_{ij}}{x_{i\hat{\jmath}}} - \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right| + \lambda \left|\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right|,
\end{aligned}$$

which is the same as the objective function value for the corresponding primal solution, and is therefore optimal. Now suppose that there is no $\tilde{\imath}$ satisfying (1). Note that if (1) is satisfied for some $\tilde{\imath}$ with $\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) = +$, then

$$\lambda \geq \sum_{i:i>\tilde{\imath}} |x_{i\hat{\jmath}}| - \sum_{i:i\leq\tilde{\imath}} |x_{i\hat{\jmath}}|, \tag{2.16}$$

$$\lambda \leq \sum_{i:i\geq\tilde{\imath}} |x_{i\hat{\jmath}}| - \sum_{i:i<\tilde{\imath}} |x_{i\hat{\jmath}}|. \tag{2.17}$$

If (1) is violated for each $\tilde{\imath}$, then for each $\tilde{\imath}$ either the lower bound (2.16) or the upper bound (2.17) for λ is violated. If for a given $\tilde{\imath}$, the lower bound (2.16) is violated, then $\lambda < \sum_{i:i>\tilde{\imath}} |x_{i\hat{\jmath}}| - \sum_{i:i\leq\tilde{\imath}} |x_{i\hat{\jmath}}|$. This implies that the upper bound (2.17) is satisfied. If we now consider point $\tilde{\imath}+1$, then the upper bound is the same as the lower bound for $\tilde{\imath}$ and is therefore satisfied. So λ must violate the lower bound for $\tilde{\imath}+1$, and we can consider $\tilde{\imath}+2$ and so on. Then lower bound is violated for all points with $\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) = +$, in particular the largest, and so $\lambda < 0$, contradicting the choice of λ. A symmetric argument holds for $\tilde{\imath}$ with $\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) = -$. Therefore, $\lambda > \sum_{i:i\geq\tilde{\imath}} |x_{i\hat{\jmath}}| - \sum_{i:i<\tilde{\imath}} |x_{i\hat{\jmath}}|$, for every $\tilde{\imath}$ with $\text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) = +$ and $\lambda > \sum_{i:i\leq\tilde{\imath}} |x_{i\hat{\jmath}}| -$

$\sum_{i:i>\tilde{\imath}} |x_{i\hat{\jmath}}|$ for every $\tilde{\imath}$ with sgn $\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right) = -$. In particular,

$$\lambda > \left| \sum_{i:\frac{x_{ij}}{x_{i\hat{\jmath}}}<0} |x_{i\hat{\jmath}}| - \sum_{i:\frac{x_{ij}}{x_{i\hat{\jmath}}}>0} |x_{i\hat{\jmath}}| \right|. \tag{2.18}$$

A dual feasible solution is to set

$$\pi_i = \begin{cases} |x_{i\hat{\jmath}}| & \text{if} \frac{x_{ij}}{x_{i\hat{\jmath}}} > 0, \\ -|x_{i\hat{\jmath}}| & \text{if} \frac{x_{ij}}{x_{i\hat{\jmath}}} < 0, \end{cases}$$

and $\gamma = \left| \sum_{i:\frac{x_{ij}}{x_{i\hat{\jmath}}}<0} |x_{i\hat{\jmath}}| - \sum_{i:\frac{x_{ij}}{x_{i\hat{\jmath}}}>0} |x_{i\hat{\jmath}}| \right|$. Note that $|\gamma| < \lambda$ by the development above. The dual objective function is $\sum_{i\in N} |x_{ij}|$, which is the same as the primal objective function value when $v_j = 0$, and is therefore optimal. □

Given a penalty λ, an optimal solution to (2) with the preservation of one coordinate $\hat{\jmath}$ requires the sorting of $(m-1)$ lists of ratios according to the lemma (1). Thus, m independent LPs or sortings are used to find the lowest value of the objective function. Therefore, for a penalty λ, it requires sorting $m(m-1)$ lists of ratios in total, each costing $(n \log n)$ running time. Algorithm 1 yields an $O(m^2 n \log n)$ running time limit in the worst-case scenario.

Proposition 3. *For a given λ and data $x_i \in \mathbb{R}^m$, $i \in N$, Algorithm 1 finds an optimal solution to* (2.3).

Proof. For each fixed coordinate, Algorithm 1 finds an optimal solution according to Lemma 1. From among those solutions, Algorithm 1 picks the one with the smallest combination of error plus regularization term. □

Algorithm 1 finds the best solution that preserves each coordinate $\hat{\jmath}$ and $v_{\hat{\jmath}}$=1 for any value of λ. Algorithm 2 seeks the intervals constructed by successive break-

Algorithm 1 Estimating an ℓ^1-norm regularized ℓ^1-norm best-fit line v^* for given λ.

Input: $x_i \in \mathbb{R}^m$ for $i = 1, \ldots, n$. λ.
Output: v^*

Set $z^* = \infty$
for $\hat{\jmath} \in M$ **do**
  Set $v_{\hat{\jmath}} = 1$.
  for $j \in M : j \neq \hat{\jmath}$ **do**
    Set $v_j = 0$.
    Sort $\left\{ \frac{x_{ij}}{x_{i\hat{\jmath}}} : i \in N, x_{i\hat{\jmath}} \neq 0 \right\}$.
    for $\tilde{\imath} \in N : x_{\tilde{\imath}\hat{\jmath}} \neq 0$ **do**
      if $\operatorname{sgn}\left(\frac{x_{ij}}{x_{i\hat{\jmath}}}\right) \lambda \in (\sum_{i:i>\tilde{\imath}} |x_{ij}| - \sum_{i:i\leq\tilde{\imath}} |x_{i\hat{\jmath}}|, \sum_{i:i\geq\tilde{\imath}} |x_{ij}| - \sum_{i:i<\tilde{\imath}} |x_{i\hat{\jmath}}|]$ **then**
        Set $v_j = \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}$.
      end if
    end for
  end for
  set $z = \sum_{i\in N} \sum_{j\in M} |x_{ij} - v_j x_{i\hat{\jmath}}| + \lambda \sum_{j\in M} |v_j|$
  if $z < z^*$ **then**
    Set $z^* = z$, $v^* = v$
  end if
end for
return v^*

points, λs at which the solution is going to change. Algorithm 2 does not determine which coordinate $\hat{\jmath}$ is best to preserve for each interval.

Proposition 4. *For data $x_i \in \mathbb{R}^m$, $i \in N$, Algorithms 2 and 3 generate the entire solution path for* (2.3) *under the assumption that all points are projected, preserving the same unit direction and $v_{\hat{\jmath}}$=1 for the preserved direction $\hat{\jmath}$.*

Proof. For each fixed coordinate $\hat{\jmath}$ and each coordinate j of v, Algorithm 2 finds the breakpoints where the conditions of Lemma 1 are satisfied. Algorithm 3 iterates through each interval for λ from Algorithm 2 and finds the intervals where preserving $\hat{\jmath}$ minimizes the objective function value. □

It is necessary to "merge" the intervals for each possible preserved coordinate $\hat{\jmath}$

Algorithm 2 Find all major breakpoints.

Input: $x_i \in \mathbb{R}^m$ for $i = 1, \ldots, n$.

Output: Ordered breakpoints for the penalty Λ and solutions $v^{\hat{\jmath}}(\lambda)$ for each choice of preserved coordinate $\hat{\jmath}$, and each $\lambda \in \Lambda$.

Set $\Lambda = \{0, \infty\}$.
for $\hat{\jmath} \in M$ **do**
  Set $v_{\hat{\jmath}}^{\hat{\jmath}} = 1$.
  for $j \in M : j \neq \hat{\jmath}$ **do**
    Set $\lambda^{\max} = 0$.
    Sort $\left\{ \frac{x_{ij}}{x_{i\hat{\jmath}}} : i \in N, x_{i\hat{\jmath}} \neq 0 \right\}$.
    for $\tilde{\imath} \in N : x_{\tilde{\imath}\hat{\jmath}} \neq 0$ **do**
      Set $\lambda = \text{sgn}\left(\frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}\right)\left(\sum_{i:i>\tilde{\imath}} |x_{i\hat{\jmath}}| - \sum_{i:i<\tilde{\imath}} |x_{i\hat{\jmath}}|\right) - |x_{\tilde{\imath}\hat{\jmath}}|$
      if $\lambda + 2|x_{\tilde{\imath}\hat{\jmath}}| > 0$, **then**
        Set $\Lambda = \Lambda \cup \max\{0, \lambda\}$.
        Set $v_j^{\hat{\jmath}}(\max\{0, \lambda\}) = \frac{x_{\tilde{\imath}j}}{x_{\tilde{\imath}\hat{\jmath}}}$.
      end if
      if $\lambda + 2|x_{\tilde{\imath}\hat{\jmath}}| > \lambda^{\max}$, **then**
        Set $\lambda^{\max} = \lambda + 2|x_{\tilde{\imath}\hat{\jmath}}|$.
      end if
    end for
    Set $\Lambda = \Lambda \cup \{\lambda^{\max}\}$.
    Set $v_j^{\hat{\jmath}}(\lambda^{\max}) = 0$.
  end for
end for
Sort Λ.
return Λ, $\{v_j^{\hat{\jmath}}(\lambda) : j \in M, \hat{\jmath} \in M, \lambda \in \Lambda\}$

and determine when the preservation of each coordinate results in the lowest value of the objective function. Therefore, we need Algorithm 3 to check each consecutive interval for λ from Algorithm 2 to determine if changing the preserved coordinate $\hat{\jmath}$ can reduce the objective function value, which may result in new breakpoints that were not discovered using Algorithm 2.

Algorithm 3 Solution Path for ℓ^1-norm Regularized ℓ^1-norm best-fit line

Input: A ordered set of breakpoints for the penalty $(\lambda^k : k = 1, \ldots, K)$ and solutions $v^{\hat{\jmath}}(\lambda^k)$ for each choice of preserved coordinate $\hat{\jmath}$, and each $k = 1, \ldots, K$.

Output: Breakpoints for the penalty Λ and solutions $v^*(\lambda)$ for each $\lambda \in \Lambda$.

Set $\Lambda = \emptyset$.
for $k = 1, \ldots, K-1$ **do**
  for $\hat{\jmath} \in M$ **do**
    Set $z^{\hat{\jmath}}(\lambda^k) = \sum_{i \in N} \|x_i - v^{\hat{\jmath}} x_{i\hat{\jmath}}\|_1 + \lambda^k \|v^{\hat{\jmath}}(\lambda^k)\|_1$.
  end for
  for $j \in M$ **do**
    $\beta_L = \max \left\{ \frac{z^j(\lambda^k) - z^{\hat{\jmath}}(\lambda^k)}{\|v^{\hat{\jmath}}(\lambda^k)\|_1 - \|v^j(\lambda^k)\|_1} : \hat{\jmath} \in M, \|v^j(\lambda^k)\|_1 < \|v^{\hat{\jmath}}(\lambda^k)\|_1 \right\}$.
    $\beta_U = \min \left\{ \frac{z^{\hat{\jmath}}(\lambda^k) - z^j(\lambda^k)}{\|v^j(\lambda^k)\|_1 - \|v^{\hat{\jmath}}(\lambda^k)\|_1} : \hat{\jmath} \in M, \|v^j(\lambda^k)\|_1 > \|v^{\hat{\jmath}}(\lambda^k)\|_1 \right\}$.
    if $\left| \left\{ \hat{\jmath} : z^j(\lambda^k) > z^{\hat{\jmath}}(\lambda^k), \|v^j(\lambda^k)\|_1 = \|v^{\hat{\jmath}}(\lambda^k)\|_1 \right\} \right| = 0$, **then**
      if $0 < \beta_L < \beta_U$ and $\lambda^k + \beta_L \leq \lambda^{k+1}$, **then**
        Set $\Lambda = \Lambda \cup \{\lambda^k + \beta_L\}$.
        Set $v^*(\lambda^k + \beta_L) = v^j(\lambda^k)$.
      else if $\beta_L \leq 0 < \beta_U$, **then**
        Set $\Lambda = \Lambda \cup \{\lambda^k\}$.
        Set $v^*(\lambda^k) = v^j(\lambda^k)$.
      end if
    end if
  end for
end for
return Λ, $\{v^*(\lambda) : \lambda \in \Lambda\}$

2.5 Synthetic Experiments

In this section, we shall first shift our attention by analyzing a toy sample, trying to understand the complete solution path in terms of breakpoints and coordinate preservation, that is, how breakpoints affect solutions by changing preserved coordinates $\hat{\jmath}$. Next, we conducted simulation studies to evaluate the performance of the Algorithm 1 against some classic low-rank approximation algorithms. Two measurements, discordance$(1-|v^T v|)$ and ℓ^0 norm of the solution vector v are used to evaluate the competing methods.

2.5.1 A Toy Example

Let us first consider five points $(4, -2, 3, -6)^T$, $(-3, 4, 2, -1)^T$, $(2, 3, -3, -2)^T$, $(-3, 4, 2, 3)^T$, $(5, 3, 2, -1)^T$. Algorithm 2 generates breakpoints $\{1, 3\}$ for $\hat{j} = 1$, $\{4, 6\}$ for $\hat{j} = 2$, $\{0, 2\}$ for $\hat{j} = 3$ and $\{3, 5, 11\}$ for $\hat{j} = 4$. The collection of breakpoints is $\{0, 1, 2, 3, 4, 5, 6, 11\}$. We now illustrate that the optimal solution (under the assumption that all points preserve the same coordinate) might change due to the existence of additional breakpoints between successive breakpoints generated from Algorithm 2. Algorithm 3 iterates by preserving $j = 1, 2, 3$ to find the lowest objective value over the lambda interval $(3.5, 4]$, giving rise to an additional breakpoint 3.5. The value of the objective function comprises the error term $(\sum \|x - v_j x_{ij}\|_1)$ and the penalty term $(\|v_j\|_1)$, both fixed over each interval for each coordinate j. Algorithm 2 finds all possible breakpoints without filtering comparable larger objective function values, which is assessed in Algorithm 3. In other words, Algorithm 3 further narrows the breakpoint intervals of Algorithm 2 by evaluating m objective function values. The complete solution path is summarized in Table 1.

Table 1.: Solution Path for toy example. The best-fit line is varying over the four lambda intervals.

λ	$z^*(\lambda)$	$v^*(\lambda)$
(0.0, 3.0)	(34.5, 42.0)	(-0.7,0.3,-0.5,1.0)
(3.0, 3.5)	(42.0, 42.9)	(-0.7,0.3,0.0,1.0)
(3.5, 11)	(42.9, 52.0)	(1.0,0.0,0.0,-0.2)
(11, ∞)	(52.0, ∞)	(1.0,0.0,0.0,0.0)

The objective function $z_{\hat{j}}$ is a linear function with respect to λ over a certain interval for each preserved $\hat{j}$, in the sense that the intercept is $\sum \|x_{ij} - v_j x_{ij}\|_1$, and $\|v_{\hat{j}}\|_1$ is a slope. Algorithm 3 gives $38.8 + 1.2\lambda$, $35 + 2.5\lambda$, $46 + 1\lambda$, and $36 + 2\lambda$ represented by four lines in Figure 1. Geometrically speaking, intersection points

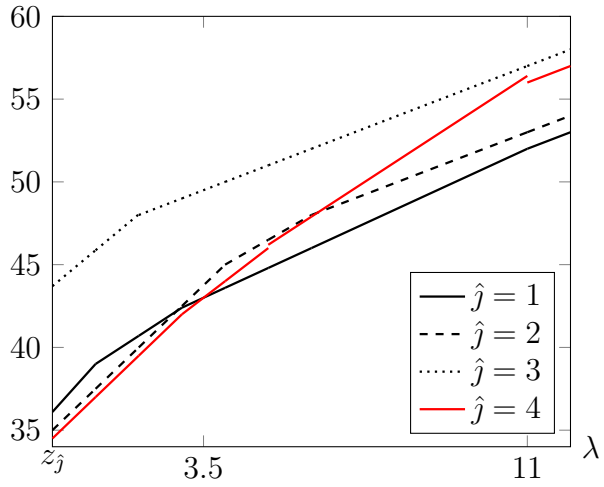

Fig. 1.: Geometric Interpretation of Breakpoints

represent the value of λs, where it is preferable to preserve the coordinate $\hat{j}$ instead of another. There is an intersection that occurs between $38.8 + 1.2\lambda$ and $36 + 2\lambda$ in the interval (3,4], resulting in an additional breakpoint that changes the preserved direction from $\hat{j} = 1$ to $\hat{j} = 4$. This result is consistent with that of Table 1, where the first two solutions preserve $\hat{j} = 4$ and the last two solutions preserve $\hat{j} = 1$. The computational results at every stage are summarized in Table 2.

Remark 1. *For some lambda values, the normalized vector* $\frac{v^*}{\|v^*\|}$*, where* v^* *is the best-fit one-dimensional subspace generated by Algorithm 1, does not minimize the objective function* (2.3).

Proof. To justify the remark 1, we use the toy example in Section (2.5.1). Given $\lambda = 3.49$, Algorithm 1 produces the best-fit $v^* = (1, 0, 0, -0.2)^T$ with the object function 2.1 42.91. Under the same λ, one solution $v = (-0.67, 0.33, 0, 1)^T$ yields a slightly larger value 43.0. In contrast, evaluating the objective function $\sum_{i \in N} \|x_i - w\frac{\alpha_i}{w_{\hat{j}}}\|_1 + \lambda\|w\|_1$ at $w = \frac{v^*}{\|v^*\|}$ and $w = \frac{v}{\|v\|}$ produces 42.91 and 41.61. This example shows that we cannot always obtain an optimal solution by setting one of the coordinates to 1. □

The successive dimension can be computed by applying the Algorithm 1 to the data projected in the null space of the acquired subspace.

Table 2.: The computation results of algorithm 1 over all breakpoints

	$\hat{j}$	j	x_j	$x_{\hat{j}}$	$\frac{x_j}{x_{\hat{j}}}$	λ_-	λ_+	(0,1]	(1,2]	(2,3]	(3,4]	(4,5]	(5,6]	(6,11]
1	1	2	4	-3	-1.33	-17	-11							
2	1	2	4	-3	-1.33	-11	-5							
3	1	2	-2	4	-0.5	-5	3	-0.5	-0.5	-0.5				
4	1	2	3	5	0.6	-13	-3							
5	1	2	3	2	1.5	-17	-13							
6	1	3	-3	2	-1.5	-17	-13							
7	1	3	2	-3	-0.67	-13	-7							
8	1	3	2	-3	-0.67	-7	-1							
9	1	3	2	5	0.4	-9	1	0.4						
10	1	3	3	4	0.75	-17	-9							
11	1	4	-6	4	-1.5	-17	-9							
12	1	4	-2	2	-1	-9	-5							
13	1	4	3	-3	-1	-5	1	-1						
14	1	4	-1	5	-0.2	1	11		-0.2	-0.2	-0.2	-0.2	-0.2	-0.2
15	1	4	-1	-3	0.33	-17	-11							
16	2	1	4	-2	-2	-16	-12							
17	2	1	-3	4	-0.75	-12	-4							
18	2	1	-3	4	-0.75	-4	4	-0.75	-0.75	-0.75	-0.75			
19	2	1	2	3	0.67	-10	-4							
20	2	1	5	3	1.67	-16	-10							
21	2	3	3	-2	-1.5	-16	-12							
22	2	3	-3	3	-1	-12	-6							
23	2	3	2	4	0.5	-2	6	0.5	0.5	0.5	0.5	0.5	0.5	
24	2	3	2	4	0.5	-10	-2							
25	2	3	2	3	0.67	-16	-10							
26	2	4	-2	3	-0.67	-16	-10							
27	2	4	-1	3	-0.33	-10	-4							
28	2	4	-1	4	-0.25	-4	4	-0.25	-0.25	-0.25	-0.25			
29	2	4	3	4	0.75	-12	-4							
30	2	4	-6	-2	3	-16	-12							
31	3	1	-3	2	-1.5	-12	-8							
32	3	1	-3	2	-1.5	-8	-4							
33	3	1	2	-3	-0.67	-4	2	-0.67	-0.67					
34	3	1	4	3	1.33	-8	-2							
35	3	1	5	2	2.5	-12	-8							
36	3	2	3	-3	-1	-12	-6							
37	3	2	-2	3	-0.67	-6	0							
38	3	2	3	2	1.5	-4	0							
39	3	2	4	2	2	-8	-4							
40	3	2	4	2	2	-12	-8							
41	3	4	-6	3	-2	-12	-6							
42	3	4	-1	2	-0.5	-6	-2							
43	3	4	-1	2	-0.5	-2	2	-0.5	-0.5					
44	3	4	-2	-3	0.67	-8	-2							
45	3	4	3	2	1.5	-12	-8							
46	4	1	5	-1	-5	-13	-11							
47	4	1	2	-2	-1	-11	-7							
48	4	1	-3	3	-1	-7	-1							
49	4	1	4	-6	-0.67	-1	11	-0.67	-0.67	-0.67	-0.67	-0.67	-0.67	-0.67
50	4	1	-3	-1	3	-13	-11							
51	4	2	4	-1	-4	-13	-11							
52	4	2	3	-1	-3	-11	-9							
53	4	2	3	-2	-1.5	-9	-5							
54	4	2	-2	-6	0.33	-7	5	0.33	0.33	0.33	0.33	0.33		
55	4	2	4	3	1.33	-13	-7							
56	4	3	2	-1	-2	-13	-11							
57	4	3	2	-1	-2	-11	-9							
58	4	3	3	-6	-0.5	-9	3	-0.5	-0.5	-0.5				
59	4	3	2	3	0.67	-9	-3							
60	4	3	-3	-2	1.5	-13	-9							

2.5.2 Evaluation of Effectiveness

In this section, we demonstrate the effectiveness of Algorithm 1 compared to principal component pursuit [17] (PCP), augmented lagrange multiplier[82] (LRBS) and alternating direction multiplier method [84] (ADM). For each configuration, a total of 10 replications are performed. For each replication, we create datasets with n observations $\in \mathbb{R}^m$ including nC outliers. Therefore, the values of n and m are the number of rows and the number of columns of input data. The values of nC and mC are the number of rows and columns contaminated. Each element of the "true" v is sampled from a Uniform distribution (-1,1) and v is normalized for all replications. α is sampled from a Uniform distribution (-100,100). Synthetic data are generated by $\alpha v^T + \epsilon$, where ϵ is the noise sampled from a Laplacian distribution Laplace (0, 1) with the probability density function $f(\epsilon|0, 1) = 0.5e^{-|\epsilon|}$. There is a link between the median and the Laplace distribution in the sense that the maximum likelihood estimator of location parameter for a list of independent and identically distributed samples following the Laplacian distribution is the sample median [46]. Outlier observations are created by sampling the first five coordinates from a Uniform (100,150) distribution. All free parameters for PCP, LRBS, and ADM are set by default. The default value λ for Algorithm 1 is chosen as the average value of all breakpoints.

As can be seen in Table 3, Algorithm 1 and PCP produce accurate estimation in terms of low discordance over all configurations. (The cosine of the angle between two unit vectors v_{est} and v_{true} is equal to their dot product. Therefore, a smaller discordance implies a smaller angle between two unit vectors.) However, the precision of PCA, LRBS and ADM decreases significantly compared to that of Algorithm 1, when contamination is introduced into the data. In terms of ℓ^0 in Table

Table 3.: The standard deviation of the discordance(1-$|\mathrm{v}_{est}^T \mathrm{v}_{true}|$) to the true line is subscript below the mean. $-_-$ stands for values less than 0.001.

n	m	nC	mC	PCA	PCP	LRBS	ADM	Algorithm 1
1000	100	0	0	$-_-$	$-_-$	$-_-$	$-_-$	$-_-$
1000	100	100	5	$0.9_{.05}$	$-_-$	$-_-$	$0.9_{.13}$	$-_-$
10000	100	0	0	$-_-$	$-_-$	$-_-$	$-_-$	$-_-$
10000	100	1000	5	$0.8_{0.1}$	$-_-$	$0.87_{.09}$	$0.84_{.11}$	$-_-$
1000	1000	0	0	$-_-$	$-_-$	$-_-$	$-_-$	$-_-$
1000	1000	100	5	$0.9_{.04}$	$-_-$	$0.96_{.05}$	$0.9_{.08}$	$-_-$
1000	2000	0	0	$-_-$	$-_-$	$-_-$	$-_-$	$-_-$
1000	2000	100	5	$0.9_{.06}$	$-_-$	$0.99_{.04}$	$0.98_{.08}$	$-_-$
5000	2000	0	0	$-_-$	$-_-$	$-_-$	$-_-$	$-_-$
5000	2000	1000	5	$0.9_{.02}$	$-_-$	$0.98_{.02}$	$0.97_{.02}$	$-_-$

4, the solutions of PCA and PCP do not exhibit any sparsity, whereas Algorithm 1 produces more sparser solution for a given λ without sacrificing much precision in presence of outliers. We will explore the effect of λ on sparsity in late experiment. On the contrary, ADM and LRBS have some degree of sparsity along with a large discordance.

Table 4.: The standard deviation of the ℓ^0 of solutions in percent is subscript below the mean.

n	m	nC	mC	PCA	PCP	LRBS	ADM	Algorithm 1
1000	100	0	0	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$96.8_{0.8}$
1000	100	100	5	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$99.8_{0.4}$	$97.2_{0.8}$
10000	100	0	0	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$97.2_{2.4}$
10000	100	1000	5	$100_{0.0}$	$100_{0.0}$	$99.9_{0.3}$	$99.9_{0.3}$	$96.2_{0.8}$
1000	1000	0	0	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$98.3_{0.8}$
1000	1000	100	5	$100_{0.0}$	$100_{0.0}$	$98.5_{1.3}$	$98.8_{1.6}$	$91.7_{0.9}$
1000	2000	0	0	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$90.2_{0.8}$
1000	2000	100	5	$100_{0.0}$	$100_{0.0}$	$99.1_{1.4}$	$98.4_{1.9}$	$91.4_{1.0}$
5000	2000	0	0	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$100_{0.0}$	$90.2_{0.7}$
5000	2000	1000	5	$100_{0.0}$	$100_{0.0}$	$94.8_{3.7}$	$90.6_{12.0}$	$89.9_{0.6}$

Positive regularization terms λ are used to control the level of sparsity in the solution. And there shall be a maximum lambda beyond which all elements are 0 except one of the elements is 1. In Figure 2, the lambda behaviors are illustrated in two settings, varying columns by fixed row number (right) and varying rows by fixed column number (left). As can be seen, with the same number of rows, Algorithm 1 is more sensitive to data with a larger number of columns in terms of the increasing rate of discordance and the decreasing rate of ℓ^0. With the same number of columns, Algorithm 1 is more sensitive to data with a smaller number of rows. It also shows that a λ less than the intersection point (between the discordance curve, and the ℓ^0-norm curve) will lead to a fairly accurate solution with a certain level of sparsity. Figure 3 further shows a discordance curve and the ℓ^0 curve intersects around 30% ℓ^0 levels in five data sets of different rows and columns. Furthermore, Figures 3 illustrate that Algorithm 1 is more sensitive to λ when working on a smaller data set in all dimensions.

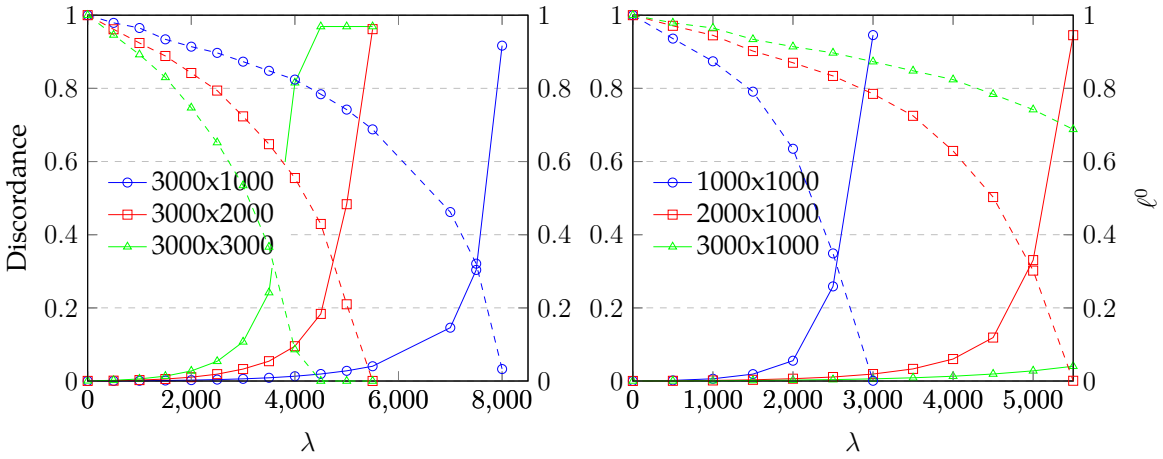

Fig. 2.: Lambda behavior in 3000 rows $\times$ varying columns on the left. Lambda behavior in varying rows $\times$ 1000 columns on the right. The discordance is read on the left y-axis for solid lines and ℓ^0 on the right y-axis for dashed lines.

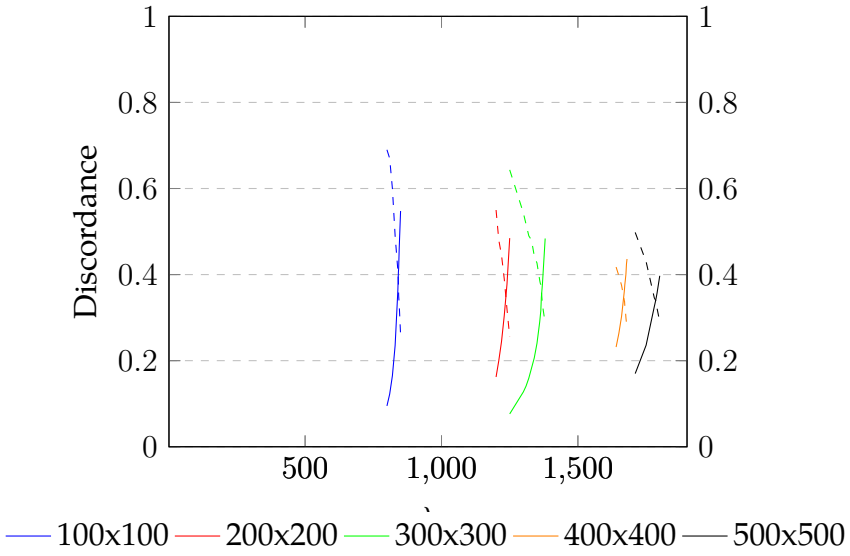

Fig. 3.: Discordance and ℓ^0 curve with respect to λ.

2.6 Implementing Algorithms 1 and 2 on NVIDIA Graphical Processing Units

In this section, we briefly discuss the implementation of Algorithms 1 and 2 on NVIDIA CUDA, a general-purpose parallel computing platform. We recognize that Algorithms 1 and 2 can be implemented in a parallel framework such as CUDA, in the sense that sorting one of the m lists is independent of the others.

2.6.1 Introduction

Recent distributed parallel computing technologies offer a solution for handling big data by increasing overall throughput (number of jobs or tasks executed per unit of time). Apache Spark [98] and CUDA [70] are the popular means of parallel computing platforms. Apache Spark architecture provides a user-friendly programming paradigm to deal with large data within a cluster of nodes. Its fault-tolerant mechanism and high-throughput capability ensure that large data processing continues. Users create a Spark application (master node) that connects to a cluster manager, which in turn allocates resources (worker node). The worker

nodes are responsible for performing computations and storing partitioned data. A driver is responsible for monitoring, scheduling, analyzing, and delivering task instructions to all workers. All tasks run in parallel.

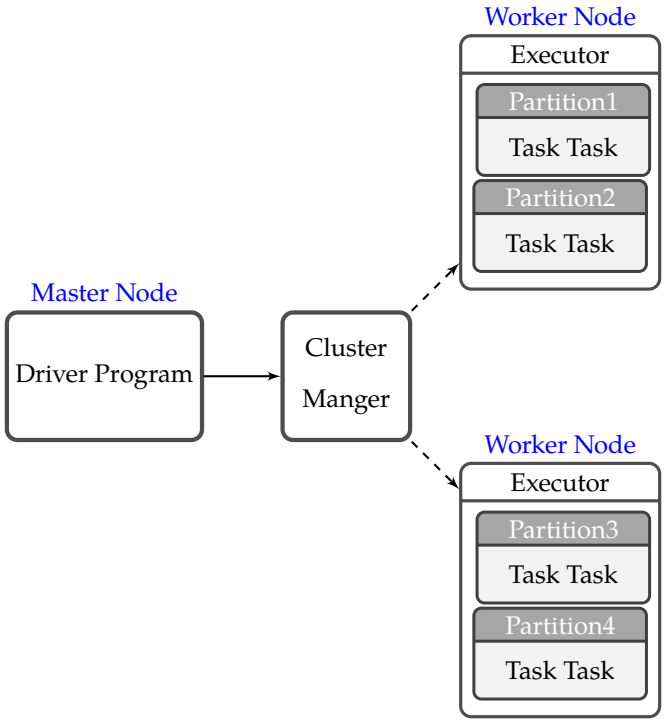

Fig. 4.: The architecture of a Spark Application

CUDA compute platform provides a scalable programming paradigm that extends C, C++, Python, and Fortran to be capable of executing parallel algorithms within thread groups on GPUs. CUDA application is heterogeneous in the sense that parallel and sequential abstractions coexist in one application,namely kernel and host. Users initialize a C/C++ application(host on CPU) connecting to a kernel interface, which in turn allocates the resources on the GPU. Devices are responsible for performing computations and partitioning the cache. A kernel is a C++/C function with the qualifier `__global__`. It will run independently on GPU threads with a unique ID, as illustrated in Figure 5. In this toy application, 2 blocks with 4

threads are launched for parallel computation on each element of the vector, which will be cached on the GPU. The memory location of each element can be indexed with three built-in variables `threadIdx.x`, `blockIdx.x` and `blockDim.x`. We will dive into more details in the next section.

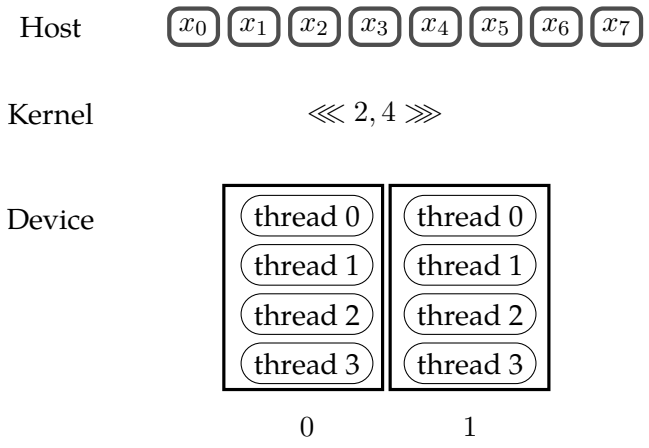

Fig. 5.: One dimensional decomposition using blocks and threads. The formula will map each thread to an element in the vector.

2.6.2 Computational Speedup Results

In this section, we run our CUDA application on a NVIDIA GeForce RTX 3060 laptop GPU with 3840 cores and 6 gigabytes of graphics memory. We run CPU implementations on an Intel 8-core I9 processor along with 40 gigabytes of memory. Figure 6 shows a snippet of this application that computes the quotients between the k^{th} column and all columns and stores the result in the vector *d_out* through three built-in variables `threadIdx.x`, `blockIdx.x`, `blockDim.x` and `gridDim.x`. `threadIdx.x` is the index of each element in one block, and `blockIdx.x` is the index of each block in CPU memory. `gridDim.x` (the number of blocks) and `blockDim.x` (the number of threads per block) for this practical task are specified in $\lll 128, 128 \ggg$. This tells runtime to create 128 copies of kernel and running them

in parallel. Each of these parallel invocations is a block. The code for Algorithms 1 and 2 is in the Appendix.

```
//Device code declaration of the matrix column-wise division kernel
1.__global__ void mykernel(float* d_out,float* d_in,int rows,int
↪  cols,int k) {
2. int idx=threadIdx.x+blockIdx.x*blockDim.x; //handle the data at
↪  this index
3.   while (idx < rows*cols) {
4.     d_out[idx]=d_in[idx]/d_in[idx%rows+k*rows]; //Each thread
↪  computes one quotient
5.     idx+= gridDim.x*blockDim.x;}}
//Host code
6.int main() {mykernel<<<128,128>>>(d_out,d_in,rows,cols,k) //invoke
↪  kernel}
```

Fig. 6.: Kernel Definition

We then run this CUDA implementation with 10 replications for each size and average over the runtime. Table 5 gives the speedup overview for 121 different input sizes. It shows up to 16.57 speedup over the R implementation and implies an increasing speedup as the size increases.

2.6.3 Solution Path with Varying Dimensions and Lambdas

In this section, we first evaluate the behavior of the solution of Algorithm 1 under different regularization parameters λ and input dimensions in terms of norm ℓ^0 and discordance. We also evaluated the space requirements for the number of breakpoints generated by Algorithm 2. All experiments were carried out on a CUDA GPU.

Tables 6 show the average elapsed time to compute Algorithm 1 on 10 replications for each size. For example, Algorithm 1 of a 5000×1000 matrix takes about 26 seconds and 127 seconds for a 1000×5000 matrix on the GPU. Since the run-

Table 5.: Speedup results for a matrix of dimension row index $\times$ column header. A value greater than 1 demonstrates the efficacy of the implementation of Algorithm 1.

	100	200	300	400	500	600	700	800	900	1000	2000
100	0.83	1.62	2.28	3.09	3.17	3.67	4.69	4.56	4.64	5.01	5.35
200	1.70	3.17	3.95	5.10	5.72	5.85	5.90	6.46	5.65	6.13	7.13
300	2.61	3.84	5.45	6.09	6.48	6.02	6.79	7.31	6.01	6.45	7.86
400	3.60	4.88	6.12	7.02	6.76	7.81	6.51	7.11	7.47	8.15	9.59
500	3.26	6.02	6.90	6.87	8.07	6.88	7.49	8.24	8.88	9.55	11.03
600	3.77	6.19	6.44	7.91	7.16	7.75	8.64	9.34	8.26	8.91	10.99
700	4.76	6.41	7.34	6.83	7.77	8.74	9.72	8.60	9.34	9.70	11.78
800	5.06	7.24	8.03	7.50	8.55	9.53	8.67	9.47	9.99	10.63	11.82
900	4.95	6.56	6.63	8.07	9.22	8.40	9.37	10.00	10.74	11.26	12.54
1000	5.42	7.01	7.16	8.62	9.84	9.17	9.92	10.65	11.32	12.00	13.52
2000	6.57	8.81	9.77	11.42	12.75	12.51	13.86	13.79	14.67	15.30	16.57

ning time of Algorithm 1 is directly proportional to m^2 and n, the matrix of larger columns requires more computation time than the matrix of fewer columns. This can also be illustrated by the time in terms of input rows, with 5000 columns being the steepest line in Figure 7. Lastly, Figures 8 and Table 7 illustrate that the number of breakpoints generated by Algorithm 2 is directly proportional to n and m^2.

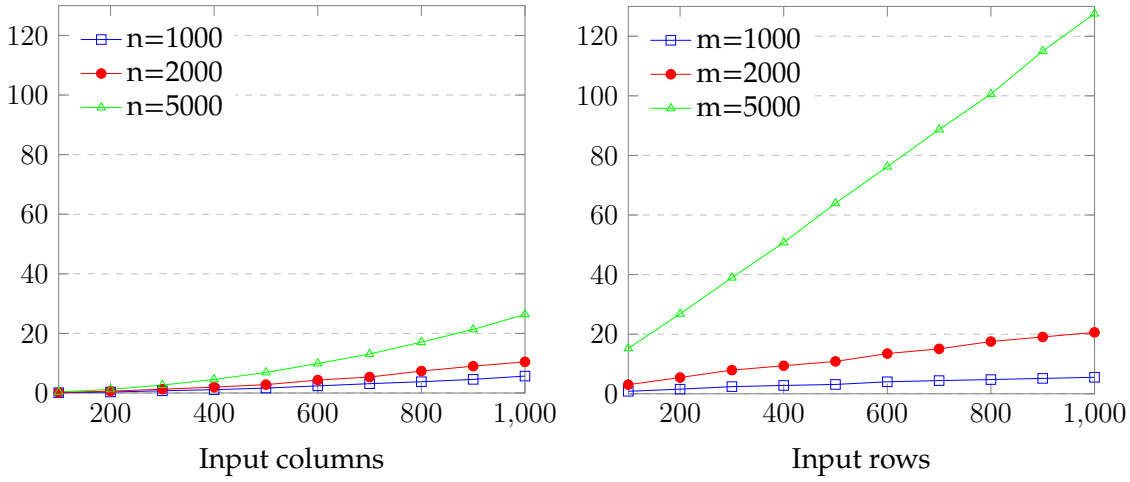

Fig. 7.: Algorithm 1 running time

Table 6.: Average and standard deviation time in seconds for 10 replications for each dataset with varying number of columns with fixed number of rows at 1000, 2000, and 5000 in the left table. Average and standard deviation time in seconds over 10 replications, varying the number of rows with the fixed number of columns at 1000,2000, and 5000 in the right table.

	1000	2000	5000
100	0.098 $_{0.008}$	0.184 $_{0.013}$	0.372 $_{0.015}$
200	0.329 $_{0.016}$	0.606 $_{0.031}$	1.209 $_{0.027}$
300	0.743 $_{0.012}$	1.269 $_{0.038}$	2.658 $_{0.064}$
400	1.118 $_{0.344}$	1.973 $_{0.049}$	4.536 $_{0.065}$
500	1.649 $_{0.062}$	2.802 $_{0.069}$	6.897 $_{0.168}$
600	2.392 $_{0.019}$	4.347 $_{0.115}$	9.895 $_{0.202}$
700	3.127 $_{0.085}$	5.351 $_{0.060}$	13.078 $_{0.352}$
800	3.782 $_{0.013}$	7.369 $_{0.203}$	17.075 $_{0.317}$
900	4.574 $_{0.021}$	9.004 $_{0.205}$	21.325 $_{0.272}$
1000	5.671 $_{0.116}$	10.451 $_{0.284}$	26.380 $_{0.412}$

	1000	2000	5000
100	0.836 $_{0.013}$	3.053 $_{0.004}$	15.223 $_{0.010}$
200	1.540 $_{0.012}$	5.458 $_{0.016}$	26.837 $_{0.034}$
300	2.397 $_{0.009}$	7.942 $_{0.006}$	38.974 $_{0.041}$
400	2.790 $_{0.012}$	9.411 $_{0.018}$	50.777 $_{0.057}$
500	3.134 $_{0.017}$	10.859 $_{0.004}$	63.932 $_{0.070}$
600	4.021 $_{0.004}$	13.488 $_{0.015}$	76.215 $_{0.047}$
700	4.421 $_{0.010}$	15.079 $_{0.017}$	88.683 $_{0.061}$
800	4.768 $_{0.006}$	17.530 $_{0.017}$	100.569 $_{0.068}$
900	5.160 $_{0.007}$	19.070 $_{0.025}$	115.010 $_{0.277}$
1000	5.524 $_{0.004}$	20.612 $_{0.023}$	127.561 $_{0.074}$

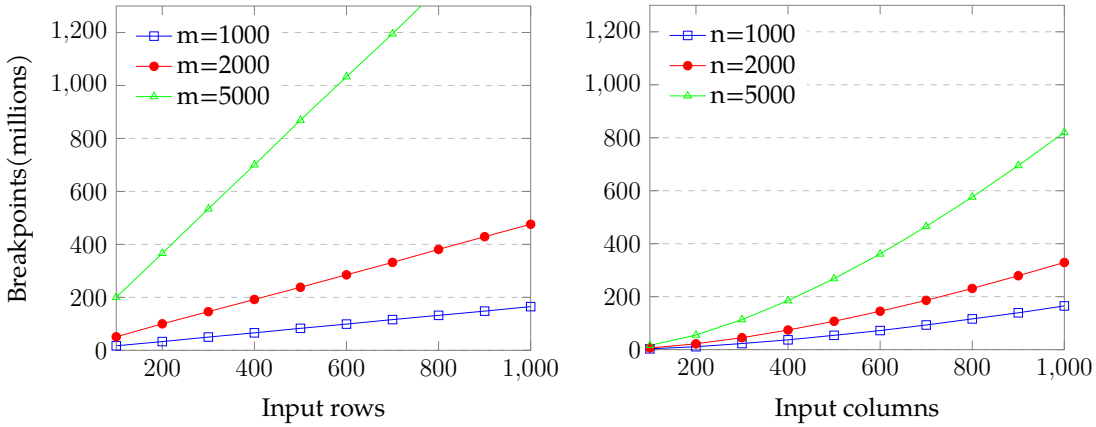

Fig. 8.: Number of breakpoints from Algorithm 2.

2.7 Background Modeling Applications

We summarize the existing research efforts on background subtraction and image denoising. Computer vision has been an active research field for the past several decades. Many applications such as activity recognition, object detection, and automated video surveillance in this research field need in the first place to extract

Table 7.: Average and standard deviation of breakpoints in millions over 10 replications varying the number of columns with the fixed number of rows at 1000, 2000, and 5000 in the left table. Average and standard deviation of breakpoints in millions over 10 replications varying the number of rows with the fixed number of columns at 1000, 2000, and 5000 in the right table.

	1000	2000	5000
100	3 $_{0.09}$	6 $_{0.19}$	16 $_{0.48}$
200	11 $_{0.32}$	22 $_{0.62}$	55 $_{1.52}$
300	23 $_{0.51}$	45 $_{0.85}$	113 $_{2.35}$
400	37 $_{0.90}$	74 $_{1.59}$	185 $_{4.29}$
500	54 $_{1.24}$	107 $_{2.36}$	268 $_{5.94}$
600	72 $_{1.35}$	145 $_{2.59}$	361 $_{7.63}$
700	93 $_{1.35}$	186 $_{2.63}$	465 $_{7.70}$
800	116 $_{1.18}$	231 $_{2.86}$	576 $_{7.80}$
900	139 $_{0.91}$	279 $_{2.86}$	695 $_{7.00}$
1000	165 $_{1.29}$	329 $_{3.04}$	820 $_{6.85}$

	1000	2000	5000
100	17 $_{0.77}$	51 $_{1.69}$	200 $_{8.12}$
200	33 $_{1.34}$	100 $_{2.60}$	366 $_{15.64}$
300	50 $_{1.48}$	146 $_{4.17}$	534 $_{16.63}$
400	66 $_{1.69}$	192 $_{6.16}$	700 $_{26.84}$
500	83 $_{1.72}$	238 $_{7.88}$	869 $_{27.42}$
600	99 $_{1.67}$	285 $_{7.67}$	1033 $_{27.32}$
700	116 $_{1.85}$	332 $_{8.24}$	1195 $_{29.47}$
800	132 $_{1.34}$	381 $_{7.28}$	1365 $_{35.68}$
900	148 $_{1.21}$	429 $_{9.86}$	1534 $_{35.67}$
1000	165 $_{1.29}$	476 $_{10.17}$	1705 $_{41.92}$

objects (foreground objects) from scene backgrounds. The widely used approach is the background subtraction method. The background subtraction paradigm is essentially a pipeline consisting of initialization of the background model, foreground detection, and background maintenance, as shown in Figure 9. In this paradigm, we first use N frames to estimate a scene background and combine it with the *N+1* frame to get the *N+1* foreground and background. Next, the *N+1* background enters the background maintenance process and the *N+1* foreground enters as a foreground result. These two procedures are executed recursively as time passes.

The simplest way to generate a background at the initial stage is to acquire a ground truth image, which is obtained after removing foreground objects with a semiautomatic method. Some video scenes need a robust or adaptive model to handle illumination changes or dynamic backgrounds when there are waving trees, rippling water, or fluttering catkins [88]. The problem that many background sub-

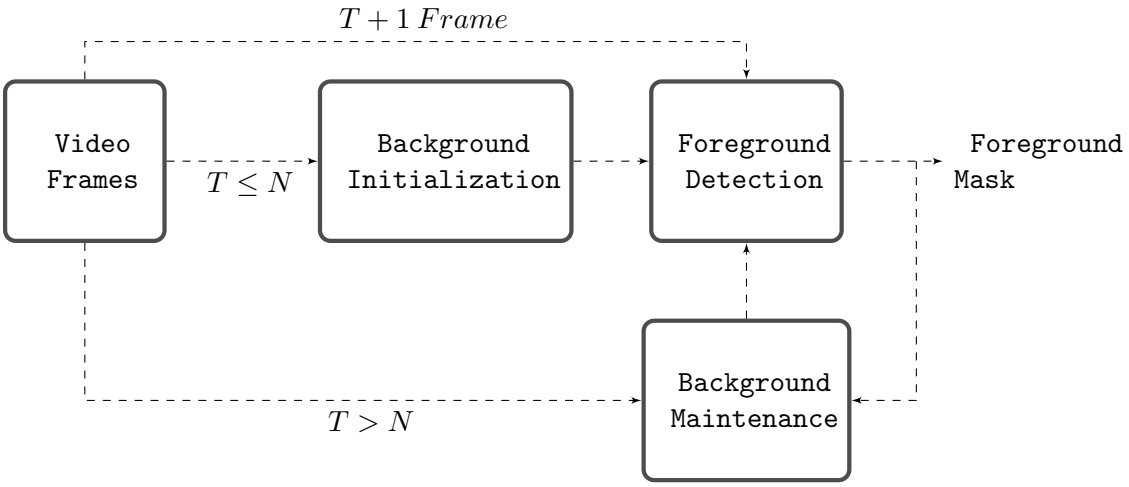

Fig. 9.: Background subtraction pipeline. N is the number of frames that are used for background initialization. T is the T time sequence frame. [8]

traction methods are susceptible to outliers represented by illumination changes or intermittent motion has existed for years. There are many background modeling methods designed to handle these situations. They can be classified into the following categories: basic models, statistical models, and subspace learning models [9]. In the basic model, central tendency parameters such as mean and median were proposed to be representations of the background in a scene [43]. Then, a classification rule is applied on the absolute difference between the background image and the current frame. Pixels are classified as background and foreground according to this rule [8]. In statistical models, each pixel is modeled as a probabilistic distribution. Some classical models, such as the Gaussian model proposed by [94], model the background of the scene as a texture surface; each pixel on the texture surface follows a Gaussian distribution. Then the standard score for each pixel is computed and pixels away from the mean by a threshold are marked as foreground. Nonparametric statistical models such as kernel density estimation (KDE) were developed to handle background with fast variations. The idea is to compute the weighted average of the gray scale values in a window around the es-

timated pixel, and the weight is defined by the kernel functions such as polynomial and Gaussian kernel. However, KDEs require significant computing time and data storage.

Subspace learning model is based on the assumption that the low-rank matrix represents the static background and the sparse matrix captures foreground objects. To understand this, we first convert each frame to a vector and then stack them by row to a data matrix. Moreover, if we stack all the backgrounds by row as a matrix, it will be a low-rank matrix because each row is almost identical. In a similar vein, the matrix consisting only of foreground pixels is sparse. Thus, a video sequence can roughly be decomposed into a low rank and a sparse matrix based on [17]. [4] work implies that the set of images of a convex Lambertian object with distant illumination can be approximated accurately using a low-dimensional subspace. We now present a robust and parallel computed background subtraction method and demonstrate its effectiveness on intermittent motion video, in which foreground objects stop for a while and then move away.

2.7.1 Background Subtraction Methods

We first parse a video into a $n \times m$ two-dimensional data set, where n represents the number of frames and m represents the number of pixels of each frame. The MP4 sample comes from scenebackgroundmodeling.net. A video file can be parsed into a certain number of frames. Each frame can be further converted into a two-dimensional grayscale pixel matrix and then converted to a vector with pixel values per column. Figure 10 illustrates this process for each frame. We need to transfer video frames into a matrix organized into named columns before continuing our algorithm. The very first step is to load the video frames with APIs.

We illustrate the performance of our algorithm against PCA in a video se-

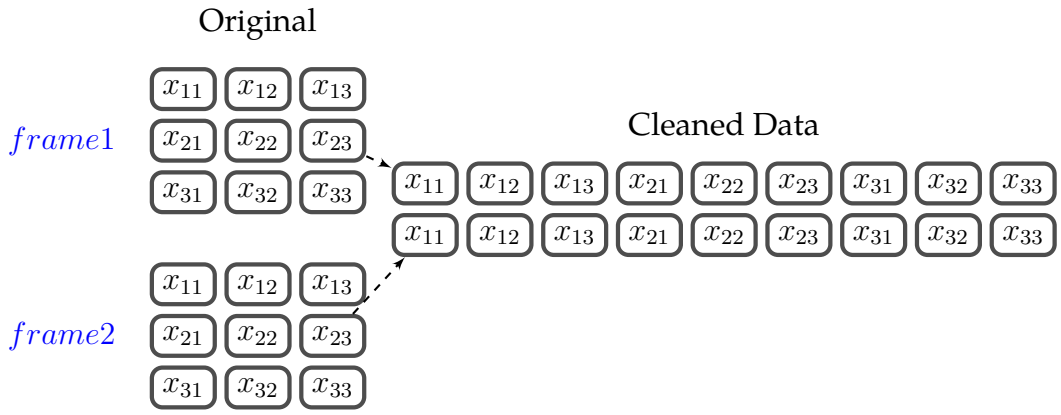

Fig. 10.: Data Initialization

quence from [35]. The original sequence is a scene in which a person stops for a while and then moves away. There are 258 frames with resolution 140 $\times$ 140. Each frame is converted to a vector of grayscale values and then stacked as rows. Therefore, the input matrix X, is in $258 \times 19,600$. Figure 11 shows the results for the $160^{th}, 200^{th}$ and 258^{th} frames for our algorithm and PCA.

Both algorithms did not distinguish the foreground from the background in frame 160 because the foreground remains still in the first 145 frames. We notice that the foreground is completely separated by our approach in column (b) for frames 200 and 258, however, there remain ghost artifacts for PCA in column (c). Similarly, we observe more artifacts of the background for PCA in column (e) than for our algorithm in column (d).

2.7.2 Deep Learning Comparison

In this section, we compare the performance metrics of the Algorithm 1 applied to a video sequence with that of a neural network method called self-organized background subtraction (SOBS) proposed by [54]. The basic idea of SOBS is assigning a weight vector $w \in \mathcal{R}^{n \times n}$ to each pixel in initial background model (first

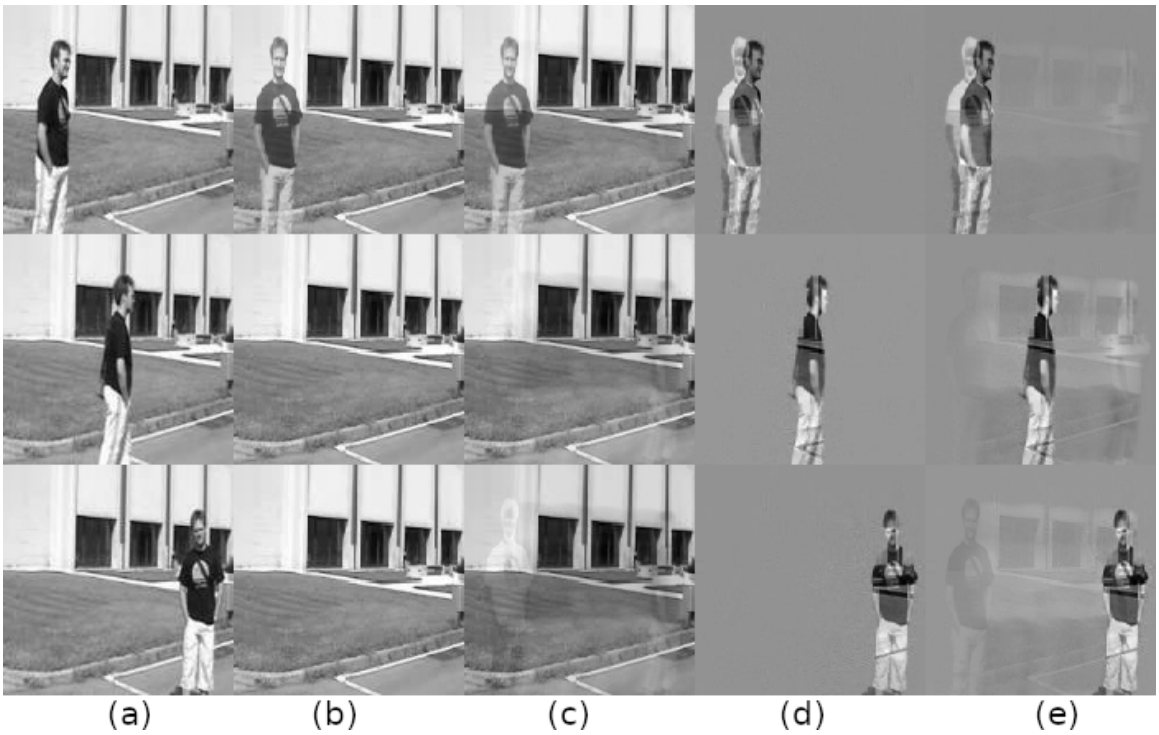

Fig. 11.: From top to bottom, the rows consist of $160^{th}, 200^{th}$ and 258^{th} frames. Column (a) is the original frame; (b) and (d) show the results of our algorithm; and (c) and (e) show the results of PCA. Columns (b) and (c) are background images and (d) and (e) are foreground images.

frame) (see Figure 12). The subtraction method is between the incoming pixel values and their corresponding pixel model. If a weight vector has been found as a match using similarity metric to determine, the incoming pixel value is considered as background pixel or foreground otherwise. Background modeling construction is self-organized in the sense that the process is automated and each learned pattern is independent. SOBS can detect forground pixels very well [2], which can be demonstrated on a higher rate of true positive in following experiment.

The data set contains raw highway video frames, and the binary foreground truth and shadow masks come from the autonomous transportation agents for on-scene networked incident management systme of highway traffic (ATON)[89]. Matlab code is provided by [75]. To quantify performance metrics, we use the fol-

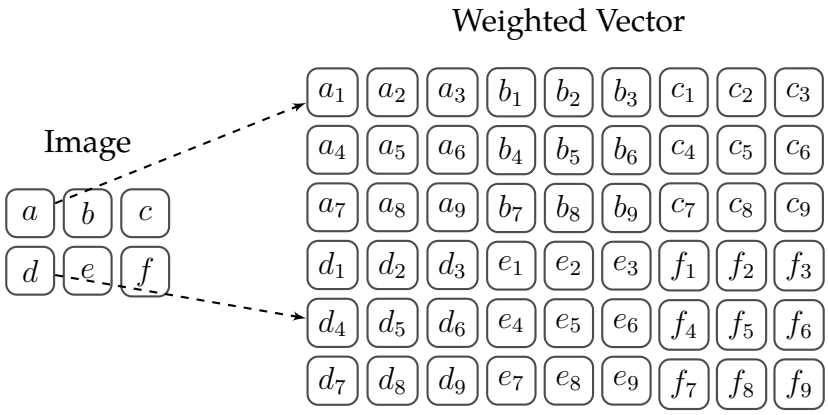

Fig. 12.: A toy image of resolution 2× 3 (Left) and its neuronal map(right).

lowing four metrics in [75] to construct a confusion matrix. True positive (TP) is the percentage of foreground pixels detected correctly. False positive (FP) is the percentage of background pixels that are incorrectly detected as foreground pixels. True negative (TN) is the percentage of background pixels detected correctly. False negatives (FN) are the percentage of foreground pixels incorrectly detected as background pixels.

	P	N
T	72.6	96.4
F	3.6	27.4

	P	N
T	70.1	97.2
F	2.8	29.9

Table 8.: Background modeling confusion matrix. The table on the left shows the performance metrics of the SOBS method. The right one shows the performance metrics of the Algorithm 1.

Table 8 shows that SOBS is slightly better at detecting the foreground pixels than Algorithm 1. However, Algorithm 1 is better at detecting background pixels than SOBS. This fact can be illustrated in Figure13, the results of the algorithm 1 in the third column show more details (black pixels in the foreground) in the foreground than the SOBS results (second column). Moreover, the background in

the second column contains some anomalies.

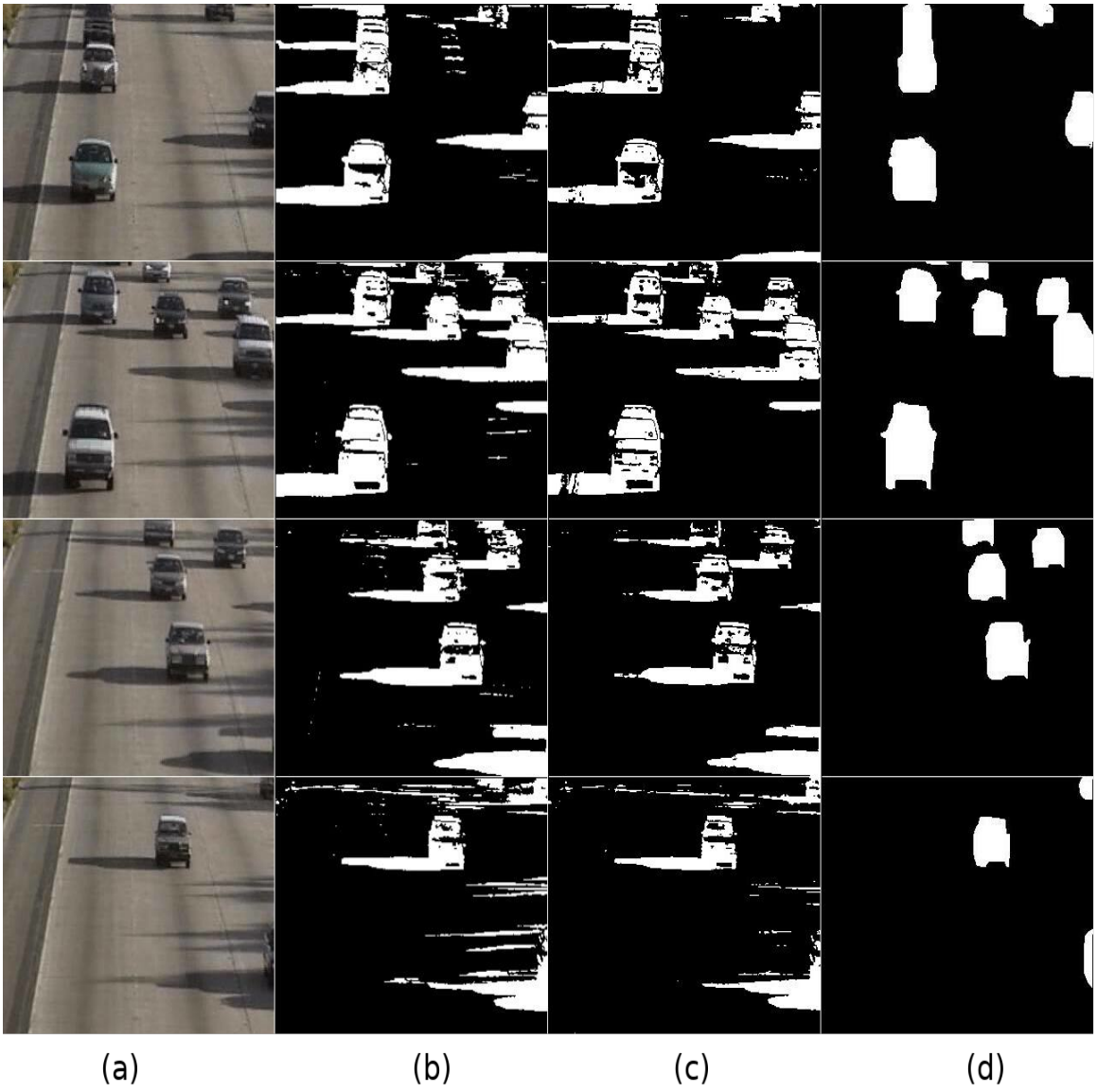

Fig. 13.: Estimated binary background and foreground. From left to right, the rows consist of raw, SOBS, Algorithm 1, and ground truth frames. From top to bottom, 33^{th}, 321^{th},417^{th} and 438^{th} frames.

2.8 Conclusion

This article discusses a method to estimate a best-fit line using linear program relaxation techniques and its connection to another algorithm based on the calcu-

lation of adjusted weighted medians. In addition, our algorithms can be processed in parallel for increased efficiency, which may enable a wide range of new practical applications.

2.9 Discussion

The least squares principal component analysis (LSPCA) is a gradient-based principal component regression. Unlike traditional principal component regression, LSPCA simultaneously finds an orthogonal subspace and fits a hyperplane of the ambient space of span(Y, XL^TL). One can argue that the ℓ^1-norm regularization could provide this framework with robustness. One obtains straightforwardly as follows,

$$\min_{L,\beta} \|Y - XL^T\beta\|_F^2 + \lambda\|X - XL^TL\|_1 \tag{2.19}$$

A similar gradient descent algorithm can be used to find the approximations L^*, β^*. By fixing L, one can obtain the optimal $\beta^* = (XL^T)^+Y$, problem 1 can be transferred to

$$\min_{L,\beta} \|Y - XL^T(XL^T)^+Y\|_F^2 + \lambda\|X - XL^TL\|_1 \tag{2.20}$$

The key aspect to note here is the regularization term is non-differentiable w.r.t L. Nonetheless, the gradient can still be computed without closed form by Automatic Differentiation in some general purpose programming languages.

[63] introduced an ADM based framework to find the sparsest element in a Bernoulli-Gaussian subspace given its basis. This sparse vector could be learned

by solving an alternative relaxation

$$\min_{q,x} \frac{1}{2}\|Yq - x\|_2^2 + \lambda\gamma \log(1 + \frac{|x|}{\gamma})$$
$$s.t. \|q\|_2 = 1$$

where, $\lambda > 0$ is a penalty parameter and an additional parameter $\gamma > 0$ to control the shape of the threshold operator S. γ is considered to affect the sensitivity of the threshold operator to the λ. We can still apply the ADM to this relaxation in a sense that

$$\begin{aligned} x^{k+1} =& S_{\lambda,\gamma}[Yq^{(k)}] \\ q^{(k+1)} =& \frac{Y^T x^{(k+1)}}{\|Y^T x^{(k+1)}\|_2}, \end{aligned} \tag{2.21}$$

where $S_{\lambda,\gamma}(x)$ is given by

$$\begin{cases} \texttt{sign}(x)\frac{\gamma}{2}((\frac{|x|}{\gamma} - 1) + \sqrt{(\frac{|x|}{\gamma} - 1)^2 - \frac{4}{\gamma}(\lambda - |x|)} & |x| \geq \lambda \\ 0 & |x| \leq \lambda \end{cases}$$

It is noted that λ in the threshold function of [63] control the sparsity of solutions. In addition, α can be used to attenuate the magnitude of the nonzero entries $x_0(i) = \Theta(1/\sqrt{\theta p})$. Hence, $x_0(i)$ will be more bigger than most of the other entries in each row of Y. In another word, α can control the sensitivity of each row of Y biased towards the first standard basis.

CHAPTER 3

IMAGE DENOISING VIA PATCH-BASED ℓ^1-NORM PRINCIPAL COMPONENT ANALYSIS

3.1 Introduction

Image denoising or image filtering is one of the fundamental problems in the computer vision field for the production of high resolution digital images and is the first evidence for the development of image inverse problems. Conventional image denoising methods exploit thresholding of the wavelet transformed coefficients, followed by reconstructing or smoothing image pixel values. Substantial progress was made in the first patch-based framework called nonlocal means (NLM) proposed by Buades et al. [14]. Since then, the majority of image denoising methods have been patch-based, and many of them have produced encouraging results compared to those of pixel-based methods. Each patch consists of a group of sub-regions within a search window of the noisy image, which is then used to estimate the true sub-region. Among all patch-based methods, PCA-based has drawn a lot of attention. The general idea for PCA-based patch-based methods is looking for a linear combination of eigen charactristics extracted from collections of patches.

3.2 Denoising Scheme

Often a signal is contaminated by multiple noise sources, however, the normalized sum of independent random variables follows a normal distribution according to the central limit theorem. Thus, the image is contaminated by an independent and identically distributed additive Gaussian noise holds under the assumption

that all sources of noise are independent. On this condition, a noisy image can be modeled as $x = \hat{x} + \xi$, where $\hat{x}$ is the underlying true image and ξ follows a normal distribution with zero mean.

The goal of a denoising method is to separate the noise from the true image signals as much as possible. The denoising pipeline consists of dictionary learning, hard thresholding, and aggregation. Figure 14 shows the whole process to estimate an underlying noise-free image. Suppose an image is an $n \times n$ array of pixel values. In our patch-based denoising, $(n-m+1)^2$ patches of the original pixel array are collected and stacked by rows. This data matrix is used to derive an $m^2 \times m^2$ dictionary of "clean" signals. Each denoised patch can be estimated by a linear combination of the columns of this dictionary, respectively. The denoised image can finally be obtained by aggregating all estimated patches.

Conventional PCA produces best-fit subspaces as measured by the sum-of-squared distances of points to their projections [65]. PCA also produces linear combinations of the original features such that the combinations capture maximal variance. A PCA can be computed via the singular value decomposition (SVD) of the data matrix. However, computing the SVD is slow and computationally expensive, especially for a larger input matrix. Moreover, SVD is sensitive to outliers. A possible modification to reduce the sensitivity to outliers is replacing the squared ℓ^2-norm criterion with an ℓ^1-norm criterion.

3.2.1 Dictionary Learning

The dictionary will be trained using the SharpEL algorithm [10]. SharpEL is an iterative method to find successive orthogonal components based on an ℓ^1-norm criterion. Each element of one component can be independently calculated by sorting several lists of ratios. Consider the optimization problem to find an ℓ^1-

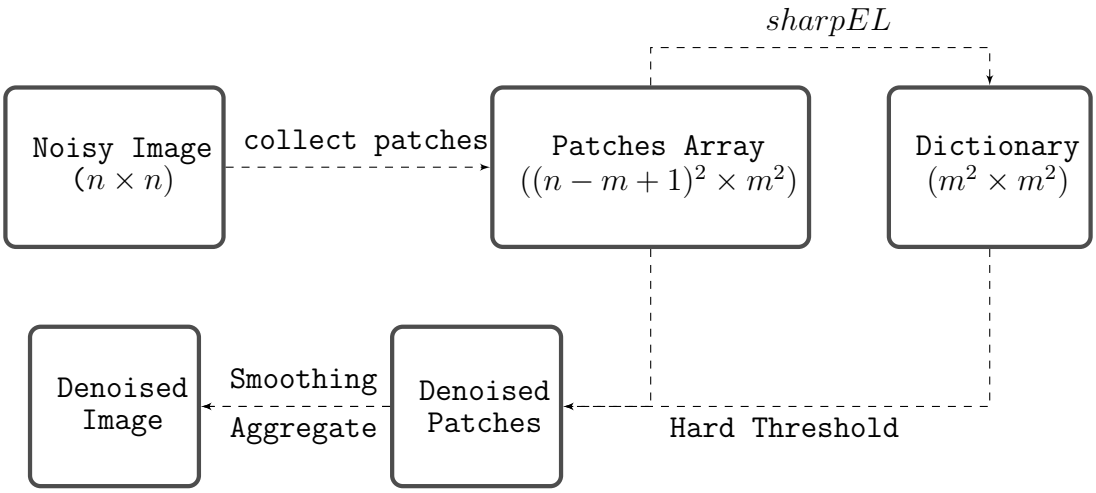

Fig. 14.: SharpEL denoising scheme by hard-thresholding

norm best-fit one-dimensional subspace:

$$\min_{v,\alpha} \sum_{i \in N} \|x_i - v\alpha_i\|_1, \tag{3.1}$$

where x_i, $i \in N$ are given points in $\mathbb{R}^d$. An optimal vector v^* determines a line through the origin corresponding to the best-fit subspace. For each point x_i, the optimal coefficient α_i^* specifies the locations of the projected points $v^*\alpha_i^*$ on the line defined by v^*. Exact solution is an NP-hard problem [28]. Brooks and Dulá [10] introduced an approximation algorithm based on the sorting of certain ratios. The problem (3.1) is recast as the following constrained linear program by imposing the preservation of one of the directions $\hat{\jmath}$. Each point will use the same $d-1$ unit directions to project onto the line defined by v. The idea of having all points use the same $d-1$ unit direction was also proposed by [90] and [20]. For the sake of simplicity, we set $v_{\hat{\jmath}} = 1$ and $\alpha_i = x_{i\hat{\jmath}}$ to impose this assumption without restricting the line defined by v:

$$\min_{\substack{v \in \mathbb{R}^d, v_{\hat{\jmath}}=1, \\ \epsilon^+, \epsilon^-}} \sum_{i \in N} \sum_{j=1}^{d} (\epsilon_{ij}^+ + \epsilon_{ij}^-), \tag{3.2}$$

subject to:

$$v_j x_{i\hat{j}} + \epsilon_{ij}^{+} - \epsilon_{ij}^{-} = x_{ij}, i \in N, j \in \{1, \ldots, d\} : j \neq \hat{j},$$
$$\epsilon_{ij}^{+}, \epsilon_{ij}^{-}, i \in N, j \in \{1, \ldots, d\}.$$

Each of the data points generates $d - 1$ constraints in this LP. By solving these d LPs and selecting the vector v from the solutions associated with the smallest of the d objective function values, we will have the estimation for ℓ^1-norm best fit line. Brooks and Dulá [10] showed that the LP can be solved directly by sorting $d - 1$ ratios. With d choices for $\hat{j}$ and $d-1$ ratios for each to sort, there are $d(d-1)$ sortings. These sortings can be calculated in parallel.

This method can be extended by iteratively projecting data onto a subspace orthogonal to the estimates for v to fit a k-dimensional subspace[10].

Suppose an observed patch x_i has been contaminated by a random variable ξ following $\mathcal{N}(0, \sigma)$:

$$x_i = \hat{x}_i + \xi, i = 1, \cdots, (n - m + 1)^2, \tag{3.3}$$

where $\hat{x}$ is the estimate of the "true value" of that patch in the underlying image. In this work, the first patch is an area of size $m \times m$ at the top left corner of the noisy image. The remaining patches are extracted in the same manner by right sliding one pixel each time. Therefore, there are a total of $(n-m+1)^2$ patches of size $m \times m$ for a image of $n \times n$ in this experiment.

The dictionary is obtained by applying SharpEl to the patch array, where $N = (n - m + 1)^2$ so that each point corresponds to a patch and $d = m^2$ so that each feature corresponds to a pixel in a patch. The dictionary is given by all m^2 components v that are iteratively derived by using SharpEl, by projecting the data into the sub-

space that is orthogonal to the components recovered so far after each iteration[55].

3.2.2 Hard Thresholding and Aggregating

Suppose we are given a collection of patches of size $m \times m$ extracted from a noisy image. If we stack the patches as rows of a matrix, each column can be considered as a feature of all patches. Under the assumption that noise spreads in all directions uniformly, each estimated true patch is the projection to a subspace of a certain dimension:

$$\hat{x}_i = \sum_{k=1}^{m^2} (e_k^T x_i) e_k \tag{3.4}$$

where e_k represents k^{th} component. Thus, the dot product $e_k^T x_i$ is a representation in that direction. However, some components e_k are less relevant to a good estimation for each patch in terms of PSNR due to the fact that the dictionary is learned from the entire collection of patches. The common approach to address this issue is by hard thresholding those representations, that is, the components with small magnitude representation value will be discarded. This leads to our general formula of estimation for each patch:

$$\hat{x}_i = \bar{x}_i + \sum_{k=1}^{m^2} \eta(e_k^T (x_i - \bar{x}_i)) e_k \tag{3.5}$$

where $\bar{x}_i$ is the median pixel value of i^{th} patch and $\eta(x) = x \cdot \mathbb{1}(\lambda < |x|)$. The optimal choice of parameter λ can be found via standard tuning techniques. Experiments demonstrate that there exists a quadratic relationship between λ and PSNR in Figure 15. Although a true mapping of $f(\lambda)$ = PSNR would not be possible in advance, a small increment on the λ in each iteration can eventually approximate the optimal result in terms of PSNR. The search for a choice for λ can also be parallelized. With

the estimate of λ, the expression (3.5) is applied to each patch. Once the whole collection of patches is denoised, it remains to estimate the true pixel values. Since patches are mostly overlapping each other, the pixel values in most areas will have multiple estimations. The pixels in the overlapping area are then averaged to get the final estimation.

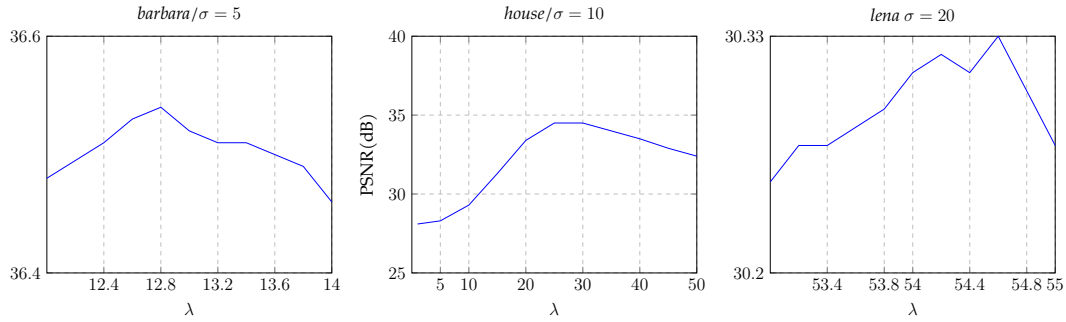

Fig. 15.: PSNR as a function of choices of λ.

3.3 Experiment Results

In this section, we first apply our method to four standard benchmark images synthesized by adding Gaussian noise with $\sigma = 5, 10, 15$, and 20. Additionally, we present the results obtained from our method along with some popular patch-based methods. The objective metric PSNR was calculated by $20\log(\frac{255}{\sqrt{MSE}})$, where MSE is the mean square error between the noise-free pixel values and their estimations. We also present one of the results for different configurations for subjective assessment in Figure 16 and 17.

For a given image of size $n \times n$ with each patch size of $m \times m$, the time complexity for procedures SharpEL, thresholding and aggregation with a given $\lambda$are $\mathcal{O}(m^4(n-m+1)^2\log(n-m+1)^2)$, $\mathcal{O}(m^4(n-m+1)^2)$ and $\mathcal{O}(n^2(n-m+1)^2)$ respectively. Previously, we made assumption that the noise-free signal lies in a subspace

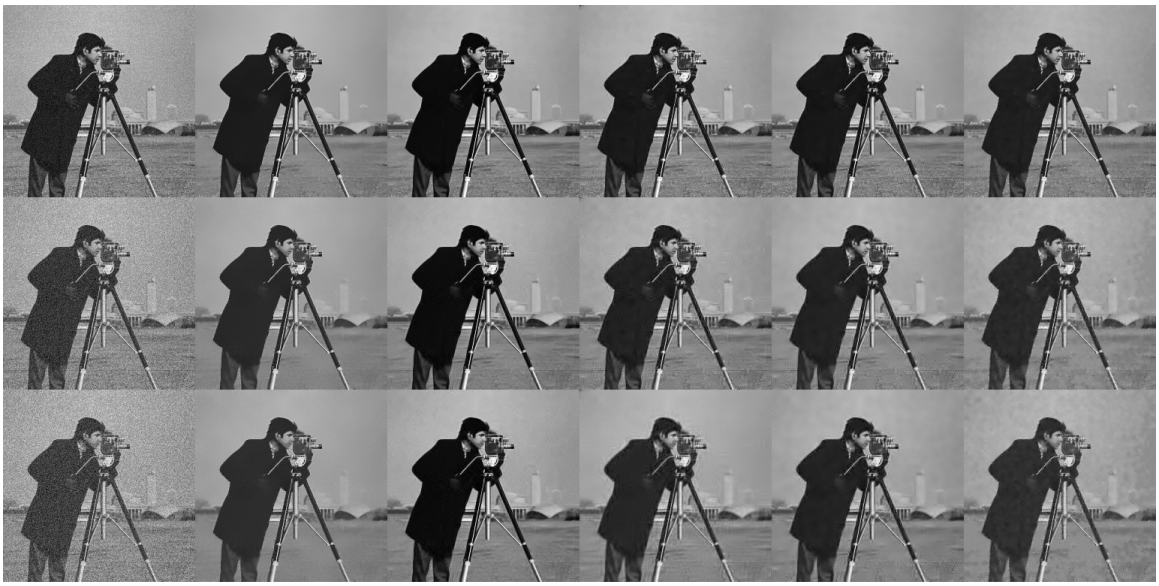

Fig. 16.: From top to bottom, rows of images are corrupted by Gaussian noise with σ= 10, 20 and 25. Column (a) stores the noisy images, (b) shows the NML results, (c) shows BM3D results, (d) shows PB-PCA results, (e) shows the results of KSVD and (f) shows the results of SharpEL.

constructed by a certain combination of m^2 orthogonal components. Therefore, m shall be sufficiently large to hold the textural patterns in the image. On the other hand, a large choice of m will increase the time complexity dramatically. In Pyatykh et. al's work[62], they suggest 4×4, 5×5, and 6×6 patch sizes. The optimal threshold parameters have been chosen for each configuration as those maximizing PSNR. All sample grayscale images are fixed in the size of 256 × 256 pixels. For each configuration, we fixed the patch size in 5 × 5 resolution. The input matrix for learning the dictionary, therefore, is in 63,504 × 25. Table 9 shows the PSNR results for each configuration.

For $\sigma = 5$, the SharpEl-based procedure is less than 3.8% worse than the best method for each of the four images. The PSNR for SharpEl for $\sigma = 10$ is 8.0% worse than the best method for the cameraman image and more competitive on the other

images. For $\sigma = 15$, the performance is at most 8.5% worse than the best method. For $\sigma = 20$ the performance is at most 7.0% worse than the best method. In general, the performance range for SharpEl is 1.8 to 8.5% worse than the best method and is typically around 5-6% worse. Figure 16 indicates that SharpEl's denoised images are competitive with existing methods and better than the noisy originals. The performance of SharpEl, coupled with the possibility of parallelization, indicates that it is a viable method.

Fig. 17.: From top to bottom, rows of images are corrupted by Gaussian noise with σ= 10, 20, 30, and 50. Column (a) stores the noisy images, (b) shows the NML results, (c) shows BM3D results, (d) shows PB-PCA results, (e) shows the results of SharpEL.

		NLM[14]	BM3D[21]	PGPCA[24]	K-SVD[1]	PLPCA[24]	PPB[23]	SharpEI[10]
$\sigma = 5$	barbara	37.3	38.8	37.8	38.3	38.6	36.1	37.3
	cameraman	36.9	38.3	37.5	37.9	37.9	35.1	37.6
	lena	37.9	39.5	38.6	39.1	39.3	36.2	38.7
	house	37.8	39.8	38.5	39.1	39.4	37.3	38.3
$\sigma = 10$	barbara	33.4	35.0	33.9	34.2	34.5	33.7	32.2
	cameraman	32.5	34.1	33.2	33.4	33.5	33.7	33.8
	lena	33.9	35.8	34.7	35.0	35.2	34.0	34.4
	house	34.8	36.6	35.1	35.4	35.6	35.0	34.5
$\sigma = 15$	barbara	31.3	32.9	31.6	31.9	32.2	31.7	30.8
	cameraman	30.3	31.8	30.4	31.0	30.7	29.5	30.1
	lena	31.8	33.6	32.3	32.7	32.7	31.8	31.9
	house	33.2	34.8	33.3	33.5	33.7	33.4	31.8
$\sigma = 20$	barbara	29.7	31.5	30.3	30.4	30.6	30.3	29.3
	cameraman	29.1	30.3	29.2	29.4	29.4	28.6	29.2
	lena	30.2	32.2	30.9	31.0	31.0	30.4	30.3
	house	31.8	33.7	31.7	32.2	31.8	31.8	31.5
$\sigma = 30$	barbara	27.5	29.6	28.5	28.2	28.8	28.9	27.4
	cameraman	27.2	28.5	27.1	27.2	27.3	28.1	27.6
	lena	28.0	30.1	28.9	28.4	29.1	29.5	29.0
	house	29.4	32.0	29.9	30.1	30.1	31.2	29.4
$\sigma = 40$	barbara	25.9	28.2	27.2	26.6	27.0	27.4	26.3
	cameraman	25.5	27.1	25.7	25.9	25.8	26.7	26.6
	lena	26.4	28.5	27.4	27.1	27.2	27.8	27.1
	house	27.5	30.6	28.7	28.2	28.3	29.6	29.4
$\sigma = 50$	barbara	24.7	27.2	26.1	25.2	25.7	26.0	25.7
	cameraman	24.1	26.1	24.6	24.7	24.6	25.7	25.1
	lena	25.1	27.6	26.3	25.7	25.9	26.5	26.5
	house	26.0	29.6	27.7	27.0	28.0	28.0	27.1
$\sigma = 60$	barbara	23.7	26.4	25.3	24.4	24.7	25.0	25.6
	cameraman	23.0	25.3	23.9	23.7	23.6	24.7	24.4
	lena	24.0	26.8	25.5	24.8	24.9	25.4	26.3
	house	24.9	28.7	26.8	25.7	25.8	26.5	26.6

Table 9.: Results in PSNR(dB) of the patch-based schemes.

3.4 Conclusion

This chapter describes an image denoising scheme based on a best-fit subspace algorithm and hard thresholding. The novelty of our method is the integration of a ℓ^1-norm best subspace estimation algorithm into the patch-based sparse dictionary image denoising framework. This algorithm can be processed in parallel for increased efficiency, which may enable a wide range of new practical applications. The experiment results demonstrate that this scheme achieves competitive results in terms of PSNR when compared to several state-of-the-art patch-based methods.

CHAPTER 4

KERNEL ℓ^1-NORM PRINCIPAL COMPONENT ANALYSIS FOR DENOISING

4.1 Introduction

Ordinary principal component analysis (PCA) is a commonly used method to uncover an underlying pattern of data by linear combinations of variables. The linear combinations can be obtained by means of matrix factorization into a canonical form. In a typical PCA, the data points (we will use patterns) are projected onto a subspace such that the first basis vector, (principal component), presents the greatest variance, the second PC presents the second greatest variance, and so on. PCA is an efficient method that requires no parameter tuning and is designed for discovering linear patterns. However, notwithstanding its success, PCA suffers two major weaknesses: its inability to resist outliers and its inability to capture nonlinear patterns. Since the loss function of the PCA is in terms of squared error, the result is sensitive to outliers (see Figure 18a). As a consequence, ℓ^1 norm based PCA is attracting considerable interest in the past decades [12, 56, 17, 19, 40]. Another major issue of PCA is that difficulties arise in recovering nonlinear patterns (see Figure 18b). [60] and [96] have pointed out that the components corresponding to smaller eigenvalues have the same importance as those with larger eigenvalues in nonlinear cases. To extract the nonlinear pattern in the input space $\mathcal{X}$, kernel-based methods have attracted considerable interest. Typically, those methods comprise two stages: a module (KPCA) that carries out a standard PCA to recover the linear patterns in $\mathcal{F}$ and a learning algorithm (the preimage problem) designed to recover nonlinear patterns in $\mathcal{X}$. The necessary algorithms to solve the

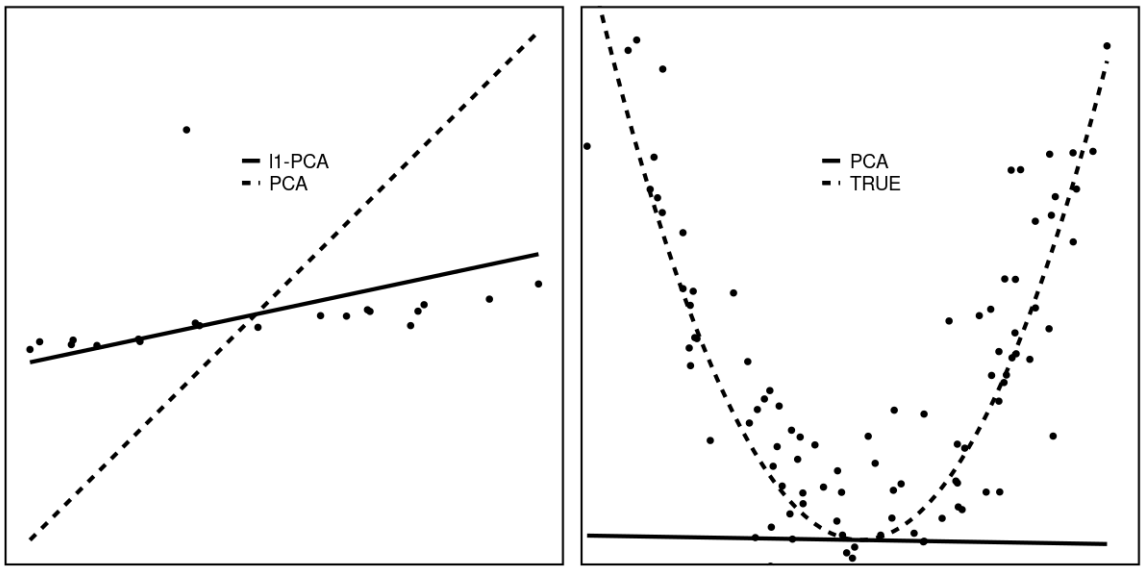

Fig. 18.: Schematic diagram illustrating how outliers and nonlinearities can distort the principal component. a the outlier drags the PC component away from the data. b the PC not even close to the true pattern.

above problems are of great interest for a broad range of applications from denoising to classification. The main purpose of this paper is to investigate the effect of incorporating an ℓ^1-norm PCA into a projection-free KPCA denoising method called PFKPCA$_1$. The work is motivated by the empirical observation by [15] that the level sets of $f(x) = \|\Phi(x) - P\Phi(x)\|^2$ are almost parallel to the underlying pattern, where $\Phi(x)$ is the representation of x in the feature space and $P\Phi(x)$ is an orthogonal projection of $\Phi(x)$. In this investigation, we provide an explanation of the impact of the choice of the similarity measure (kernel) on the denoising performance from a geometric perspective and provide some insights thereof. Then we compare PFKPCA$_1$ with that of [15] and show that it performs well in terms of denoising datasets such as spirals and pictures with changing temperatures. The difference between our algorithm and the method proposed by Bui et al. [15] is the use of an ℓ^1-norm KPCA proposed by Kim and Klabjan [39]. We will see the

potential benefits of this approach.

The remaining is organized as follows; in the next section, we provide more introductions to KPCA and the preimage problem. Then in Section 3, we formulate (4.3) using kernels and ℓ^1-norm minimization. Experiments on synthetic and computer vision data are examined in (Section 4), followed by a geometric interpretation of our findings (Section 5).

4.2 An ℓ^1-norm Basis in KPCA

To perform a line search in feature space, one needs to find the gradient with respect to K_z. Therefore, the kernel function has to be differentiable. We use the radial basis function kernel $k(z, z') = e^{-\frac{\|z-z'\|^2}{2\sigma^2}}$ in the following experiments, a commonly used kernel in support vector machine. Note that a wide selection of kernels such as polynomial, exponential, or sigmoid are compatible with this algorithm. We now try to reformulate (1.16) to a function of kernel K.

$$\|\Phi(z) - P_n\Phi(x)\|^2 = \langle\Phi(z), \Phi(z)\rangle - 2\langle\Phi(z), P_n\Phi(x)\rangle + \langle P_n\Phi(x), P_n\Phi(x)\rangle, \quad (4.1)$$

$P_n\Phi(x)$ is the projection onto n-dimensional subspace P of the feature space. [41] shows that the column space of P is $\Phi(X)U\Lambda^{-1/2}$, where U and Λ is eigenvectors and values of K. Thus, one can compute $P_n\Phi(x)$ without knowing explicit mapping Φ. We argue that P constructed by ℓ^1-norm basis could mitigate the influence of outliers. [39] developed a fixed point algorithm seeking such basis. One can derive

the projections onto P by sequentially updating of K,

$$\begin{aligned}\langle \Phi(z), P_n\Phi(x)\rangle &= \langle \Phi(z), \sum_{k=1}^{n}\langle \Phi(x), v_k\rangle v_k\rangle \\ &= \sum_{k=1}^{n}\langle \Phi(z), v_k\rangle\langle \Phi(x), v_k\rangle \\ &= \sum_{k=1}^{n}(\frac{K_z^k c_k K_x^k c_k}{c_k^T K^k c_k})\end{aligned}$$

where K_z^k is row z of kernel matrix K^k associated with k^{th} basis v_k and c_k has all entries in $\{-1, 1\}$. Likewise, $\langle P_n\Phi(z), P_n\Phi(z)\rangle = \sum_{k=1}^{n}(\frac{K_z^k c_k K_z^k c_k}{c_k^T K^k c_k})$. Thus, the error surface can be written as $k(z,z) - \sum_{k=1}^{n}(\frac{K_z^k c_k K_z^k c_k}{c_k^T K^k c_k})$. To find its derivative with respect to z, we need to reduce K_z^k in terms of K_z^1 (the original centralized kernel matrix). Each $K^{k:k>1}$ can be interpreted as the null space projection of its predecessor such as

$$\begin{aligned}K^k &= K^{k-1} - \frac{K^{k-1}c_{k-1}(K^{k-1}c_{k-1})T}{(c_{k-1})^T K^{k-1} c_{k-1}} \\ &= K^{k-1}\left(I - \frac{c_{k-1}(K^{k-1}c_{k-1})^T}{(c_{k-1})^T K^{k-1} c_{k-1}}\right) \\ &= K^1 \prod_{i=1}^{k-1}\left(I - \frac{c_i(K^i c_i)^T}{(c_i)^T K^i c_i}\right). \end{aligned} \tag{4.2}$$

Let $A_{k:k>1} = \prod_{i=1}^{k-1}(I - \frac{c_i(K^i c_i)^T}{(c_i)^T K^i c_i})$ and $A_1 = I$. Thus,

$$\|\Phi(z) - P_n\Phi(z)\|^2 = k(z,z) - K_z^{1T}\left(\sum_{k=1}^{n}\frac{A_k c_k (A_k c_k)^T}{c_k^T K^k c_k}\right)K_z^1 \tag{4.3}$$

$$= k(z,z) - K_z^T C C^T K_z. \tag{4.4}$$

This expression is similar to the original expansion in [15] except $C = [\frac{A_1 c_1}{\sqrt{c_1^T K^1 c_1}}, \cdots, \frac{A_n c_n}{\sqrt{c_n^T K^n c_n}}]$. It is a differentiable convex function, provided that kernel function is differentiable. Its gradient ∇f at point z is $\nabla k(z,z) - 2K_z^T C C^T \nabla K_z$.

Convexity guarantees there exists $\alpha > 0$ such that $-\nabla {f_z}^T \alpha < 0$. To this end, we have the necessary conditions for the line search. Bui et al. [15] conjectured that the true pattern is parallel to one of the level sets of the $f(x)$. Since the gradient direction is orthogonal to the level set curve, one can move close to the preimages by traveling along the negative gradient by an appropriate amount which can be determined by a line search. The algorithm is summarized as below.

Algorithm 4 Projection-free Kernel ℓ^1-norm PCA Algorithm

Input: $\mathcal{X}_c = \mathcal{X} + \epsilon$,

1: Compute Kernel K and its principal subspace C using the KPCA method of [39].
2: Construct error surface $\|\Phi(.) - P_n\Phi(.)\|^2$ over $\mathcal{X}_c$ with K and C.
3: **for** $z \in \mathcal{X}_c$ **do**
4: Find the 1^{st} steepest descent direction d at z on the surface.
5: Find the first stationary point αd along this direction.
6: Move the point by same α step along the direction d in the input space
7: $\hat{x} = z + \alpha d$.
8: **end for**

Output: $\hat{\mathcal{X}} \approx \mathcal{X}$.

4.3 Experiment Results

In this section, we run both algorithms on synthetic data sets and two image data sets contaminated by additive Gaussian noise and attempt to get some favorable properties of Algorithm 1. In all cases, we use the radial basis function kernel $k(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$ and the parameter σ^2 has been set to half of the data variance. We denote [15]'s method as PFKPCA$_2$ and Algorithm 1 as PFKPCA$_1$. The performance is assessed using mean squared error (MSE) and the distance between the preimage and its nearest true image $\mathrm{d}_{\mathcal{M}}$. MSE is the mean of the squares of errors between the estimated values and the true values of the pixels. Note that a preimage of higher MSE lying on the true manifold is preferred to a preimage with lower MSE that does

not lie on the true manifold. Therefore, we consider a low $\mathrm{d}_{\mathcal{M}}$ value as the primary measure of performance, followed by a low MSE as a secondary measure.

For image and video data, the peak signal-to-noise ratio (PSNR) in decibels (**dB**) is used to measure the image (8-bit RGB) denoising quality defined as $20 \log \frac{255}{\sqrt{\texttt{MSE}}}$. Typical values for the PSNR in lossy image and video compression are between 30 and 50 dB. We illustrate the proposed and existing approaches using one baseline image by varying different noise levels for denoising.

4.3.1 Spiral Data

We first use synthetic datasets similar to the one used by [39] to evaluate their denoising performance. The data consists of Gaussian contaminated ($\sigma = 0.1$) noisy spirals of 1 and 3 cycles with an additional 4 outliers in the middle. The two-dimensional spiral data of the 1 cycle coordinates are $(e^{0.14x}\cos(x)+\epsilon, e^{0.14x}\sin(x)+\epsilon)$, where ϵ is sampled from a normal distribution (0,0.01) and x is a series of 158 numbers ranging from -19 to -12.7 by step 0.04. 4 outliers coordinates are sampled from uniform distributions (-0.01,0) and (-0.03,-0.01). The spiral data of the 3 cycle have the same setting except that x is a series of 476 numbers ranging from -19 to 0. We apply both methods by varying the number of components preserved up to 4, respectively.

Figure 19 contains the results for one pair of spiral datasets. There is a clear discrepancy between the results in rows 1 and 2 and in rows 3 and 4 in the absence of outliers in the PFKPCA_2 results. The preimages indicate that PFKPCA_2 treats the outliers as normal points and incorporates them into the spiral, while PFKPCA_1 leaves the outliers in the middle which is their true position. The smaller $\mathrm{d}_{\mathcal{M}}$ values in Table 10 confirm that the preimages of PFKPCA_1 are closer to the true manifold than those of PFKPCA_2 over five pairs of spiral instances. In the case of preserving the

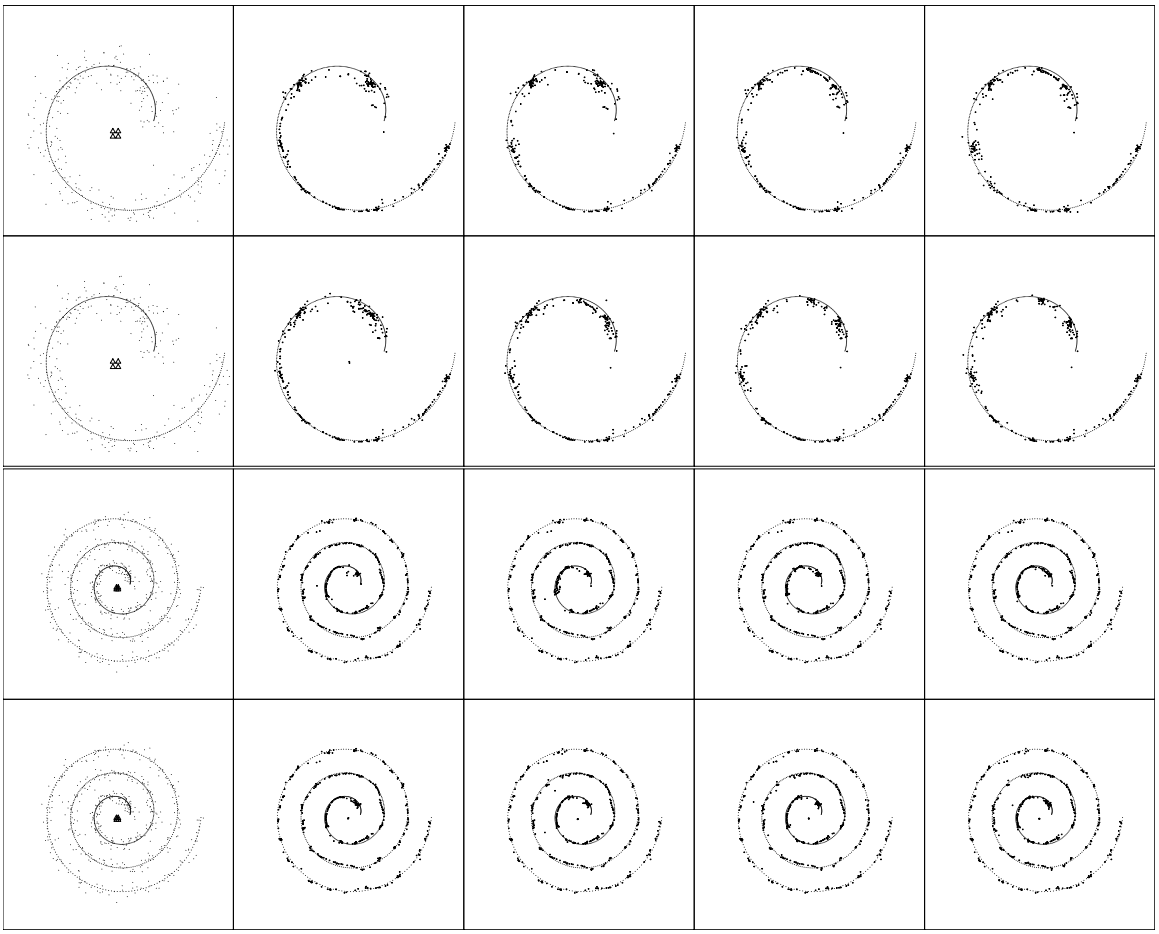

Fig. 19.: PFKPCA$_2$ reconstructions with preserving the top four components (odd rows) versus those of PFKPCA$_1$ (even rows). Rows 1 and 2 are results for the 1-cycle spiral, and rows 3 and 4 are results for the 3-cycle spiral. The first column shows the raw data along with the true curve. The following six columns are reconstructions based on the preservation of 1, . . ., and 4 components.

first component of the 1-cycle spiral, the mean and standard deviation for $d_{\mathcal{M}}$ of PFKPCA$_2$ is more than twice that of PFKPCA$_1$. The mean values for $d_{\mathcal{M}}$ and MSE are always smaller for PFKPCA$_1$ than for PFKPCA$_2$ and the standard deviations are smaller with few exceptions. The evidence suggests that PFKPCA$_1$ is less sensitive to outliers than PFKPCA$_2$.

Table 10.: Two contaminated spirals of Cycles 1 and 3 are denoised by preserving 1, ..., and 4 components, respectively. Each setting was replicated 5 times to obtain the mean and standard deviation of the two measures.

		MSE(‰)				$d_{\mathcal{M}}$(‰)			
		1	2	3	4	1	2	3	4
1-cycle	PFKPCA$_2$	$20_{4.1}$	$19_{3.2}$	$19_{1.4}$	$18_{1.4}$	$1.4_{0.8}$	$0.9_{0.5}$	$0.8_{0.3}$	$1.2_{0.3}$
	PFKPCA$_1$	$17_{2.5}$	$16_{1.6}$	$18_{2.0}$	$16_{1.6}$	$0.6_{0.3}$	$0.6_{0.3}$	$0.6_{0.3}$	$0.6_{0.3}$
3-cycle	PFKPCA$_2$	$24_{2.9}$	$26_{3.1}$	$24_{2.9}$	$24_{2.7}$	$6.0_{2.9}$	$9.0_{4.0}$	$6.0_{3.8}$	$6.0_{3.0}$
	PFKPCA$_1$	$20_{1.6}$	$20_{2.3}$	$22_{2.7}$	$21_{2.3}$	$4.0_{1.2}$	$5.0_{4.9}$	$5.0_{2.6}$	$4.0_{1.5}$

4.3.2 Clustering Example

In the next example, we demonstrate that denoising with the proposed method can be a helpful preprocessing step for clustering. The data are generated in a manner similar to an example of [72]. There are initially three point sources at location (0,0.7), (0.5,0.1), and (-0.5,-0.1). 100 random points with $\sigma = 0.1$ Gaussian noise are scattered around each point. We attempt to reduce the intracluster and intercluster variance simultaneously. Figure 20 shows both algorithms successfully force all points to three sources and the best results using the first 2 components. This suggests that our proposed preimage estimation algorithm can also be used as a preprocessing step for clustering. Both methods appear to produce line and circle patterns as the number of components preserved increases beyond 2. When preserving three components, preimage from PFKPCA$_1$ exhibits less noise for the top right clusters, but more noise for the bottom left cluster when compared to PFKPCA$_2$. These results are similar to that of [72].

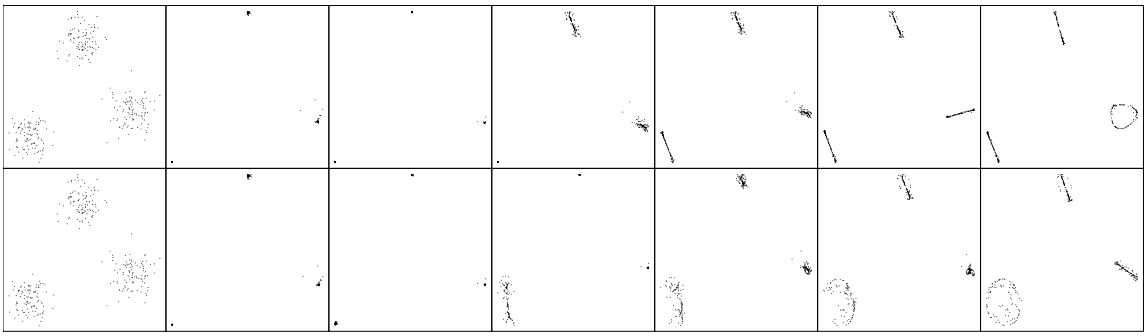

Fig. 20.: The first column of the image contains two copies of the original data with noise added. The next 6 columns contain preimages produced by PFKPCA$_2$ (first row) and PFKPCA$_1$ (second row) when preserving 1, 2, . . ., 6 components.

4.3.3 Object Images with Changing Illumination Color Temperature

We have used 3 objects from the Amsterdam Library of Object Images (ALOI) [27] illumination color collection to evaluate the denoising performance of PDFPCA$_1$. Each image is of size 100×100 in frontal view under 12 illumination color temperatures (measured in degrees of Kelvin), resulting in objects illuminated under a reddish to white illumination color. For experimental purpose, all 12 images are stacked in rows of matrix to which 7 additive Gaussian noises with variance 5, 10, 15, 20, 25, 30, and 35 are applied in each experiment.

Figure 21 shows one example of 12 images for each object in corrupted and denoised conditions. The images denoised by PFKPCA$_1$ (third, sixth, and ninth rows) tend to be more consistent and sharp. There are discernible differences in the results in columns 3 and 4 between the two methods. The quantitative results for all 12 images of each object are summarized in Table 11. PFKPCA$_1$ has a higher average PSNR value for all noise levels and objects. However, the standard deviations for PFKPCA$_1$ is larger than that of PFKPCA$_2$ for 15 out of 21 object-noise variance configurations. PFKPCA$_1$ provides an average performance between 0.1 and 1.6 dB higher than that of PFKPCA$_2$.

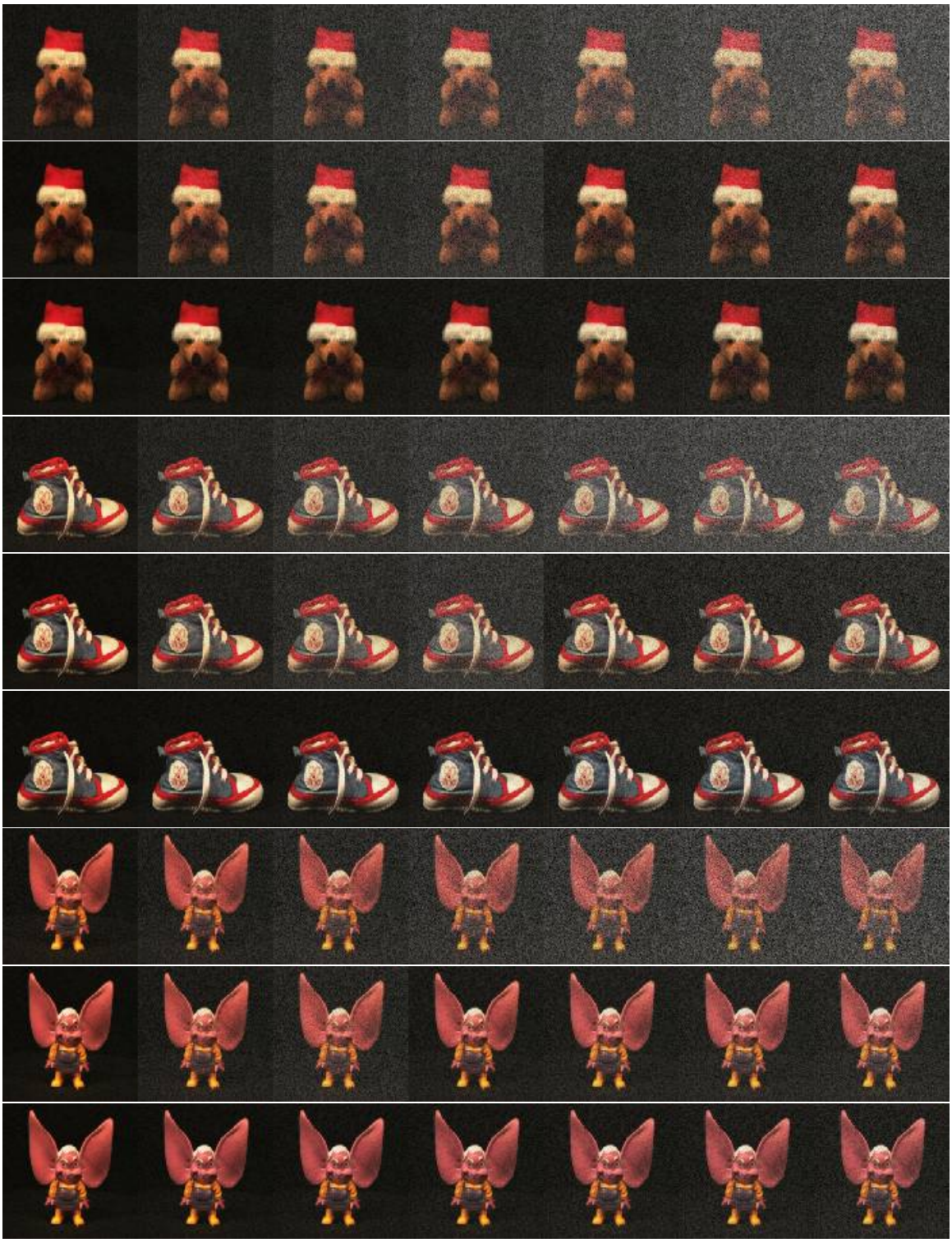

Fig. 21.: For each of the three objects, one example from 12 images is shown in rows 1, 4, and 7 with noise added (variance 5, 10, 15, 20, 25, 30, and 35). Rows 2, 5, and 8 contain the denoised example images from PFKPCA$_2$ and rows 3, 6, and 9 contain those from PFKPCA$_1$.

Table 11.: PSNR comparison between two methods. The PSNR for each object with different noise realizations are averaged over 12 images. The standard deviation is in the subscript.

object	method	5	10	15	20	25	30	35
bear	PFKPCA$_2$	$39.4_{2.3}$	$34.9_{2.6}$	$32.0_{2.7}$	$29.9_{2.6}$	$28.6_{2.0}$	$27.1_{2.0}$	$26.0_{1.9}$
	PFKPCA$_1$	$39.7_{2.9}$	$35.2_{2.7}$	$32.7_{2.5}$	$30.8_{2.4}$	$29.4_{2.3}$	$28.2_{2.3}$	$27.1_{2.2}$
shoe	PFKPCA$_2$	$38.7_{2.7}$	$34.5_{2.8}$	$31.1_{3.2}$	$29.1_{3.0}$	$28.5_{1.8}$	$27.1_{1.6}$	$26.0_{1.5}$
	PFKPCA$_1$	$39.4_{3.2}$	$35.0_{2.9}$	$32.5_{2.6}$	$30.7_{2.4}$	$29.3_{2.3}$	$28.1_{2.2}$	$27.1_{2.1}$
fairy	PFKPCA$_2$	$38.9_{2.4}$	$34.4_{2.8}$	$31.8_{3.1}$	$30.7_{2.3}$	$29.2_{2.2}$	$27.8_{2.1}$	$26.6_{2.0}$
	PFKPCA$_1$	$39.7_{2.9}$	$35.2_{2.6}$	$32.7_{2.5}$	$30.9_{2.4}$	$29.4_{2.4}$	$28.2_{2.4}$	$27.1_{2.3}$

4.4 Geometric Interpretation

We provide an explanation of poor denoising performance for noise points in cases when the tangent plane of the level curve L_c of the function $f(z) = \|\Phi(z) - P\Phi(z)\|$ for a noise point is not parallel to the tangent plane for the true manifold at the true point.

We illustrate the inconsistency of denoising performance in cases when the tangent plane for true manifold at the true point is not parallel to the level curve L_c of the function $f(z) = \|\Phi(z) - P\Phi(z)\|$.

The function $f(z)$ is differentiable, the steepest descent $-\nabla f$ at a point z is orthogonal to any L_c of f at that point. By virtue of the assumption, the line search along the first steepest descent will reach the representation on the true manifold, provided that the level curve at z parallel to the manifold ideally. It is certainly possible that L_c is not parallel to the true manifold. Figure 22 presents a graphic illustration of this situation. For the sake of illustration, a Gaussian noising parabola and corresponding $\|\Phi(\cdot) - P\Phi(\cdot)\|$ surface is also shown in the Figure 23. This situation is mostly happening under the pointy area of error surface $f(z)$. Immediately from the loss function, the error surface is determined by the kernel matrix

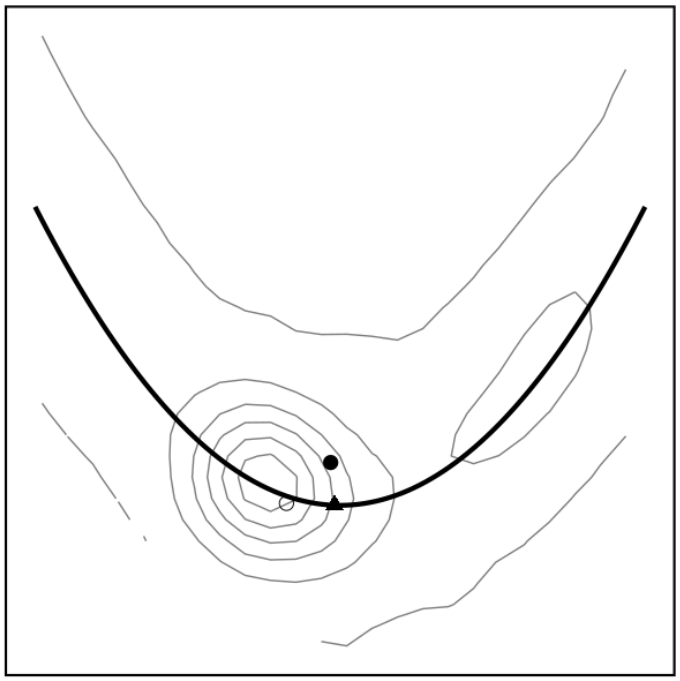

Fig. 22.: The example of bad denoising performance. The L_c are not parallel to the underlying pattern(thick parabola). The noisy point (solid circle) was drag towards preimage (open circle) along the 1st steepest descent, which is far away from true point (triangle).

K. The above reasoning illustrates an important point pertaining to the construction of the error surface associated with K. Indeed, some differentiable regularizations which smooth the pointy area could potentially improve the preimage quality around those areas.

4.5 Conclusions

This article incorporates a kernel ℓ^1-norm principal component analysis into a novel preimage estimation procedure using kernel tricks that can be used for a wide range of machine learning tasks such as handwriting recognition and image denoising. We demonstrate that the kernel trick is also applicable to the kernel ℓ^1-norm principal component analysis in section 3. The new method was able to provide more consistent denoised results compared to the projection-free kernel

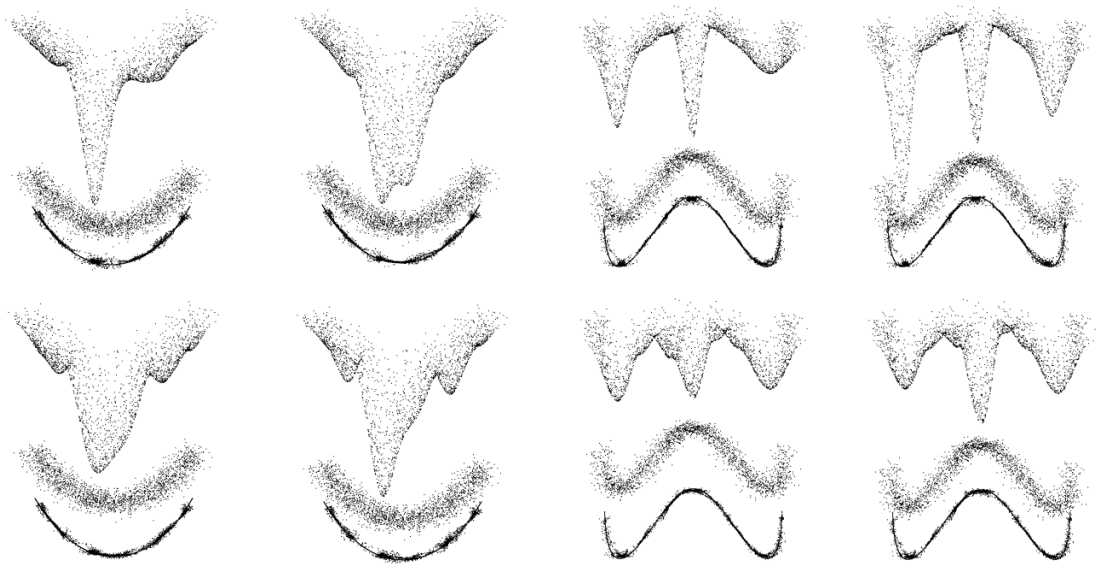

Fig. 23.: Four plots on left are based on sample of 2000 from a polynomial of degree 2. Plots on right are based on sample of 2000 points from a polynomial of degree 4. Three layers are exhibited. Middle layer consists of contaminated points. Error surface of $\|\Phi(\cdot) - P\Phi(\cdot)\|$ is on the top layer, and corresponding preimages along with true line (thick curve) are at bottom. Top row represents preserving first 1 and 2 L_2 PCs. Second row represents preserving first 1 and 2 L_1 PCs.

principal component analysis. We see a couple of potential future projects as follows: (1) We only consider radial basis function kernel for experiment in this paper; however, the proposed method is also compatible with other kernels, such as exponential, polynomial, and sigmoid kernels. (2) The proposed method could be applied to the data with outliers and nonlinearity.

4.6 Discussion

We can extend least squared PCA into Hilbert space, namely, kernel least absolute supervised PCA. Consider the formulation for least absolute supervised PCA,

$$\min_{V,\gamma} \|Y - XV^T\gamma\|_F^2 + \lambda\|X - XV^TV\|_F^2 \tag{4.5}$$

problem 4.5 can be extended into high-dimensional feature space with user-chosen kernels.

$$\min_{V,\gamma} \|Y - \Phi(X)V^T\gamma\|_F^2 + \lambda\|\Phi(X) - \Phi(X)V^TV\|_F^2$$

$$\min_{V,\gamma} \|Y - \Phi(X)V^T\gamma\|_F^2 + \lambda(\texttt{diag}(K) - 2\langle\Phi(X), V\rangle\langle\Phi(X), V\rangle + \langle\Phi(X)V^TV, \Phi(X)V^TV\rangle$$

where V is the analog of L in feature space and Φ is an implicit mapping. One may attempt to use the kernel trick in [72] that V lies on the **span**$(\Phi(X_i))$, $V = W\Phi(X)$. Then we will have the formulation as below,

$$\min_{\gamma} \|Y - KW\gamma\|_F^2 + \lambda(\texttt{diag}(K) - KWW^TK^T) \tag{4.6}$$

where K is the kernel matrix. In this case, W is fixed to eigenvectors of K. Therefore, formulation 4.6 really is just the kernel principal component regression introduced in [68].

An interesting approach is using the nonlinear kernel trick introduced in [41]. Suppose the column space of Π is $\Phi(X)U\Lambda^{-\frac{1}{2}}$, where U, Λ is eigenvectors and associated eigenvalues of K. By Lemma 1 in [41],

$$\min_{\eta,B} \|Y - ZB^T\eta\|_F^2 + \lambda(\texttt{diag}(K) - 2\langle Z, B\rangle\langle Z, B\rangle + \langle ZB^TB, ZB^TB\rangle \tag{4.7}$$

where $\Phi(X)V^T = ZB^T$, $\Phi(X)V^TV = ZB^TB$, and $Z = \Lambda^{\frac{1}{2}}U^T$. Formulation 4.7 is the same as 4.5, the LSPCA in 4.5 can directly be applied to Z.

Appendix A

CODE

Algorithm 1

```
#include <thrust/sort.h>
#include <thrust/iterator/zip_iterator.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include <thrust/scan.h>
#include <thrust/execution_policy.h>
#include <thrust/gather.h>
#include <thrust/functional.h>
#include <thrust/tuple.h>
#include <thrust/reduce.h>
#include <thrust/random.h>
#include <thrust/for_each.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/sequence.h>
#include <thrust/binary_search.h>
#include <thrust/inner_product.h>
#include <thrust/fill.h>
#include <thrust/copy.h>
#include <thrust/transform_reduce.h>
#include <thrust/set_operations.h>
#include <limits>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <istream>

using namespace thrust::placeholders;

__global__ void mykernel(float* d_out,float* d_in,int rows,int
↪  cols,int k){
    int idx=threadIdx.x+blockIdx.x*blockDim.x;
    while (idx < rows*cols) {
      d_out[idx]=d_in[idx]/d_in[idx%rows+k*rows];
      idx+= gridDim.x*blockDim.x;
    }
```

```
}

struct s_rat
{
  template <typename T1, typename T2>
  __host__ __device__
  bool operator()(const T1 &t1, const T2 &t2)
  {
    if  (thrust::get<1>(t1) < thrust::get<1>(t2)) return true;
    if  (thrust::get<1>(t1) > thrust::get<1>(t2)) return false;
    if  (thrust::get<0>(t1) < thrust::get<0>(t2)) return true;
    return 0;
  }
};

struct lb
{
    const float tot;
    lb(float _tot): tot(_tot){}
    __host__ __device__
   float operator()(float& x) {
   return tot-2.0*x;
  }
};

typedef thrust::tuple<int,float,float,float> floatTup;
struct int_pred
{
  const float reg;
  const float tot;
  int_pred(float reg,float tot): reg(reg),tot(tot){}
  __host__ __device__
   bool operator() (const floatTup& tup )
    {
        const float x = thrust::get<1>(tup);
        const float y = thrust::get<2>(tup);
        const float z = thrust::get<3>(tup);
        return (z<0? (!(reg <= -x && -y< reg)) : (!(y >=reg &&
         ↪  x < reg)));
    }
};


struct xminusvx
    {
        const float *m_vec1;
        const float *m_vec2;
        const float *m_A;
        float *m_result;
```

```
        size_t v1size;
        xminusvx(thrust::device_vector<float> const&
        ↪  A,thrust::device_vector<float> const& vec1,
        ↪  thrust::device_vector<float> const&
        ↪  vec2,thrust::device_vector<float>& result)
        {
          m_vec1 = thrust::raw_pointer_cast(vec1.data());
          m_vec2 = thrust::raw_pointer_cast(vec2.data());
          m_result = thrust::raw_pointer_cast(result.data());
          m_A=thrust::raw_pointer_cast(A.data());
          v1size = vec1.size();
        }

        __host__ __device__
        void operator()(const size_t  x) const
        {
            size_t i = x%v1size;
            size_t j = x/v1size;
            m_result[i + j * v1size] = fabs(m_A[i+j*v1size] -
            ↪  m_vec1[i] * m_vec2[j]);
        }
    };

template <typename T>
struct absv
{
__host__ __device__ T operator()(const T &x) const
  {
    return (x < T(0)) ? -x : x;
  }
};


void rc_find(FILE *fp,int* rows,int* cols)
{
    *rows = 0;
    int i,j;
    *cols = 0;
    while((i=fgetc(fp))!=EOF)
    {
        if (i == ' ') {
            ++j;
        }
        else if (i == '\n')
        {
            (*rows)++;
            *cols=j+1;
            j = 0;
        }
```

```
    }
    fclose(fp);
}

int main(int argc,char **argv){

    char *problem  = (char *) malloc ((100) * sizeof (char));
    strcpy(problem,argv[1]);
    int lamb = atof(argv[2]);
    float *h_in, *d_in,*d_out;

    FILE *getrc = fopen(problem,"r");
    int rows, cols;
    rc_find(getrc,&rows,&cols);
    int N = rows*cols;
    h_in = (float *) malloc (N*sizeof (float));
    FILE *data = fopen(problem,"r");
    for (int j=0;j<rows;j++) {
      for (int i=0;i<cols;i++){
        fscanf(data, "%f",&h_in[i*rows+j]);
    }}
    fclose(data);

    cudaMalloc((void **) &d_in,N*sizeof(float));
    cudaMalloc((void **) &d_out,N*sizeof(float));
    cudaMemcpy(d_in,h_in,N*sizeof(float),cudaMemcpyHostToDevice
);
    float zopt = 9999999999999;
    int alpha;
    thrust::device_vector<float> vopt(cols);

    for (int k = 0;k<cols;k++)
    {
        mykernel<<<128,128>>>(d_out,d_in,rows,cols,k);
        thrust::device_vector<float> ratio(d_out, d_out+N);
        thrust::device_vector<float> xjhat(d_in,d_in+N);
        thrust::device_vector<float> l_lamada(N);
        thrust::device_vector<float> r_lamada(N);
        thrust::device_vector<int> index(ratio.size());
        thrust::device_vector<int> jhat(ratio.size());
        thrust::sequence(index.begin(), index.end());
        thrust::sequence(jhat.begin(), jhat.end());
        thrust::transform(index.begin(), index.end(),
        ↪  index.begin(), _1/rows);
        thrust::transform(jhat.begin(), jhat.end(),
        ↪  jhat.begin(), _1%rows+k*rows);
        auto myit = thrust::make_zip_iterator(thrust::make_tupl
        ↪  e(ratio.begin(),
        index.begin(), jhat.begin()));
```

```
thrust::sort(myit, myit+N, s_rat());
thrust::gather(thrust::device,jhat.begin(),
↪  jhat.end(),xjhat.begin(),xjhat.begin());
thrust::device_vector<float> abs_xjhat(N);
abs_xjhat = xjhat;
thrust::device_vector<float> inc(N);
thrust::device_vector<float> exc(N);
thrust::transform(abs_xjhat.begin(),abs_xjhat.end(),
    abs_xjhat.begin(),absv<float>());
thrust::exclusive_scan_by_key(index.begin(),
↪  index.end(), abs_xjhat.begin(), exc.begin(),0.0,thr
↪  ust::equal_to<int>(),thrust::plus<float>());
thrust::inclusive_scan_by_key(index.begin(),
↪  index.end(), abs_xjhat.begin(), inc.begin(),thrust:
↪  :equal_to<int>(),thrust::plus<float>
↪  ());
thrust::transform(inc.begin(),inc.end(),l_lamada.begin(
↪  ),lb(thrust::reduce(abs_xjhat.begin(),
abs_xjhat.begin()+rows)));
thrust::transform(exc.begin(),exc.end(),r_lamada.begin(
↪  ),lb(thrust::reduce(abs_xjhat.begin(),
abs_xjhat.begin()+rows)));

typedef thrust::device_vector<float>::iterator fit;
typedef thrust::device_vector<int>::iterator iit;
typedef thrust::tuple<iit,fit,fit,fit> tup;
typedef thrust::zip_iterator<tup>  zip_it;
zip_it v = thrust::remove_if(thrust::make_zip_iterator(
↪  thrust::make_tuple(index.begin(),l_lamada.begin(),r
↪  _lamada.begin(),ratio.begin())),
↪  thrust::make_zip_iterator(thrust::make_tuple(index.
↪  end(),l_lamada.end(),r_lamada.end(),
ratio.end())),int_pred(lamb,thrust::reduce(abs_xjhat.be
↪  gin(),abs_xjhat.begin()+rows)));

tup endTuple = v.get_iterator_tuple();
index.erase(thrust::get<0>(endTuple),index.end());
l_lamada.erase(thrust::get<1>(endTuple),l_lamada.end());
r_lamada.erase(thrust::get<2>(endTuple),r_lamada.end());
ratio.erase( thrust::get<3>(endTuple),ratio.end());

thrust::device_vector<float>vstar(cols);
thrust::device_vector<int>v_keys(cols);
thrust::sequence(v_keys.begin(),v_keys.end());
thrust::set_union_by_key(index.begin(),index.end(),v_ke
↪  ys.begin(),v_keys.end(),ratio.begin(),vstar.begin(),
v_keys.begin(),vstar.begin());
```

```
        thrust::device_vector<float> vx(N);
        thrust::device_vector<float>
        ↪  vec1(d_in+k*rows,d_in+k*rows+rows);
        thrust::device_vector<float> A(d_in,d_in+N);
        thrust::for_each_n(thrust::device, thrust::counting_ite
        ↪  rator<size_t>(0),(N),xminusvx(A,vec1,vstar,vx));
        float z = thrust::reduce(vx.begin(),vx.end()) +
        ↪  lamb*thrust::transform_reduce(vstar.begin(),vstar.e
        ↪  nd(),absv<float>(),0.0,thrust::plus<float>());

        if (z <= zopt) {
            zopt = z;
            vopt = vstar;
            alpha = k;
        }
    }
    float norm = std::sqrt(thrust::inner_product(vopt.begin(),v
    ↪  opt.end(),vopt.begin(),0.0f));
    thrust::transform(vopt.begin(),vopt.end(),vopt.begin(),_1/=
    ↪  norm);

    free(h_in);
    cudaFree(d_in);
    cudaFree(d_out);

    return 0;
}
```

Algorithm 2

```
#include <thrust/sort.h>
#include <thrust/iterator/zip_iterator.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include <thrust/scan.h>
#include <thrust/execution_policy.h>
#include <thrust/gather.h>
#include <thrust/functional.h>
#include <thrust/tuple.h>
#include <thrust/reduce.h>
#include <thrust/random.h>
#include <thrust/for_each.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/sequence.h>
#include <thrust/binary_search.h>
#include <thrust/fill.h>
#include <thrust/copy.h>
#include <thrust/transform_reduce.h>
```

```
#include <thrust/remove.h>
#include <thrust/set_operations.h>
#include <limits>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <thrust/extrema.h>
#include <thrust/unique.h>
#include <string>
#include <thrust/count.h>
#include <thrust/merge.h>

using namespace thrust::placeholders;
__global__ void mykernel(float* d_out,float* d_in,int rows,int
↪  cols,int k){
    int idx=threadIdx.x+blockIdx.x*blockDim.x;
    while (idx < rows*cols) {
      d_out[idx]=d_in[idx]/d_in[idx%rows+k*rows];
      idx+= gridDim.x*blockDim.x;
    }
}
struct s_rat
{
  template <typename T1, typename T2>
  __host__ __device__
  bool operator()(const T1 &t1, const T2 &t2){
    if  (thrust::get<1>(t1) < thrust::get<1>(t2)) return true;
    if  (thrust::get<1>(t1) > thrust::get<1>(t2)) return false;
    if  (thrust::get<0>(t1) < thrust::get<0>(t2)) return true;
    return false;
  }
};

struct lb
{
    const float tot;
    lb(float _tot): tot(_tot){}
    __host__ __device__
   float operator()(float& x) {
   return tot-2.0*x;
  }
};

 struct lb1
{
    const float tot;
    lb1(float _tot): tot(_tot){}
    __host__ __device__
```

```
   float operator()(float& x) {
   return 2.0*x-tot;
  }
};

 struct lbs
{
    __host__ __device__
   float operator()(float& x,float& y) {
   return x+2*y;
  }
};

struct is_neg
{
  __host__ __device__
  bool operator()(const float x)
  {
    return (signbit(x) || x==0.0);
  }
};

struct is_pos
{
  __host__ __device__
  bool operator()(const float x)
  {
    return (signbit(-x));
  }
};

template <typename T>
struct absv
{
  __host__ __device__ T operator()(const T &x) const
  {
    return (x < T(0)) ? -x : x;
  }

};

struct is_k
{
  const int tot;
    is_k(int _tot): tot(_tot){}
  __host__ __device__
  bool operator()(const int x)
  {
    return (x == tot);
```

```
  }
};


template <typename Iterator>
void print_range(Iterator first, Iterator last)
{
    typedef typename std::iterator_traits<Iterator>::value_type
    ↪  T;

    thrust::copy(first, last,
    ↪  std::ostream_iterator<T>(std::cout<< std::setw(6) <<
    ↪  std::fixed<< std::setprecision(1), " "));
    std::cout << "\n";
}


template <typename Iterator>
void writefile(Iterator first, Iterator last)
{
    typedef typename std::iterator_traits<Iterator>::value_type
    ↪  T;
    std::ofstream output;
          output.open("breakpoints");
    thrust::copy(first, last,
    ↪  std::ostream_iterator<T>(std::cout<< std::setw(6) <<
    ↪  std::fixed<< std::setprecision(1), " "));

}

void rc_find(FILE *fp,int* rows,int* cols)
{
    *rows = 0;
    int i,j;
    *cols = 0;

    while((i=fgetc(fp))!=EOF)
    {
            if (i == ' ') {
                ++j;
            }
            else if (i == '\n') {
                (*rows)++;
                *cols=j+1;
                j = 0;
            }
    }
    fclose(fp);
}
```

```
int main(int argc,char **argv){

    char *problem  = (char *) malloc ((100) * sizeof (char));
    strcpy(problem,argv[1]);
    float *h_in, *d_in,*d_out;

    FILE *getrc = fopen(problem,"r");
    int rows, cols;
    rc_find(getrc,&rows,&cols);
    int N = rows*cols;


    h_in = (float *) malloc (N*sizeof (float));
    FILE *data = fopen(problem,"r");
    for (int j=0;j<rows;j++) {
      for (int i=0;i<cols;i++){
        fscanf(data, "%f",&h_in[i*rows+j]);
    }}
    fclose(data);

    cudaMalloc((void **) &d_in,N*sizeof(float));
    cudaMalloc((void **) &d_out,N*sizeof(float));

    cudaMemcpy(d_in,h_in,N*sizeof(float),cudaMemcpyHostToDevice
);


    typedef thrust::device_vector<float> Vector;

    float minupp=99999999;
    float maxlow=0;
    float finalavg=0;
    float finalsize=0;
    for (int k = 0;k<cols;k++) {
    mykernel<<<128,128>>>(d_out,d_in,rows,cols,k);


    thrust::device_vector<float> ratio(d_out, d_out+N);
    thrust::device_vector<float> xjhat(d_in,d_in+N);
    thrust::device_vector<float> l_lamb(N);
    thrust::device_vector<float> r_lamb(N);
    thrust::device_vector<int> index(ratio.size());
    thrust::device_vector<int> jhat(ratio.size());
    thrust::sequence(index.begin(), index.end());
    thrust::sequence(jhat.begin(), jhat.end());
    thrust::transform(index.begin(), index.end(),
     ↪  index.begin(), _1/rows);
```

```
thrust::transform(jhat.begin(), jhat.end(), jhat.begin(),
↪  _1%rows+k*rows);
auto myit = thrust::make_zip_iterator(thrust::make_tuple(ra
↪  tio.begin(), index.begin(),
↪  jhat.begin()));
thrust::sort(myit, myit+N, s_rat());
thrust::gather(thrust::device,jhat.begin(),
↪  jhat.end(),xjhat.begin(),xjhat.begin());
thrust::device_vector<float> abs_xjhat(N);
abs_xjhat = xjhat;
thrust::device_vector<float> inc(N);
thrust::device_vector<float> exc(N);
thrust::transform(abs_xjhat.begin(),abs_xjhat.end(),abs_xjh
↪  at.begin(),absv<float>());
thrust::exclusive_scan_by_key(index.begin(), index.end(),
↪  abs_xjhat.begin(), exc.begin(),0.0,thrust::equal_to<int
↪  >(),thrust::plus<float>());
thrust::inclusive_scan_by_key(index.begin(), index.end(),
↪  abs_xjhat.begin(),
↪  inc.begin(),thrust::equal_to<int>(),thrust::plus<float>
↪  ());
thrust::transform(inc.begin(),inc.end(),l_lamb.begin(),lb(t
↪  hrust::reduce(abs_xjhat.begin(),abs_xjhat.begin()+rows)
↪  ));
thrust::transform(exc.begin(),exc.end(),r_lamb.begin(),lb1(
↪  thrust::reduce(abs_xjhat.begin(),abs_xjhat.begin()+rows
↪  )));

thrust::device_vector<int> se(ratio.size());
thrust::device_vector<float> lambdas(ratio.size());
thrust::sequence(se.begin(), se.end());
thrust::gather_if(se.begin(),se.end(),ratio.begin(),r_lamb.
↪  begin(),l_lamb.begin(),is_neg());


typedef Vector::iterator           Iterator;
thrust::transform(l_lamb.begin(),l_lamb.end(),abs_xjhat.beg
↪  in(),lambdas.begin(),lbs());
Vector r1(ratio.size());
thrust::remove_copy_if(lambdas.begin(), lambdas.end(),
↪  index.begin(),r1.begin(), is_k(k));
Iterator iter = thrust::remove_if(r1.begin(),
↪  r1.end(),is_neg());
r1.resize(iter-r1.begin());
Iterator iter2 = thrust::unique(r1.begin(),r1.end());
r1.resize(iter2-r1.begin());

typedef thrust::device_vector<float>::iterator ft;
```

```
    ft min = min_element(r1.begin(), r1.end());
    ft max = max_element(r1.begin(), r1.end());
    if (*min < minupp) minupp = *min;
    if (*max > maxlow) maxlow = *max;
    float avg = reduce(r1.begin(),r1.end());
    float sam = r1.size();
    finalavg= avg + finalavg;
    finalsize=sam +finalsize;
    }
    std::cout << minupp << " " << maxlow << " " <<
    ↪  finalavg/finalsize << std::endl;


    cudaFree(d_in);
    cudaFree(d_out);
    free(h_in);
    return 0;

}
```

Kernel ℓ^1-norm PCA

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "type.h"
#include <mkl.h>

int pfkpca (IOINFOptr ioinfo,ENTITYINFOptr entityinfo,
↪  PROBLEMINFOptr probleminfo);
int pfkpca (IOINFOptr ioinfo,ENTITYINFOptr entityinfo,
↪  PROBLEMINFOptr probleminfo)
{
 int i,j,l,m,z;
 int numattributes_m = entityinfo->numattributes_m;
 int numentities_n = entityinfo->numentities_n;
 double *points_XT = entityinfo->points_XT;
 double *xbzx;
 double sum,sum1,sum2,sum4, A;
 double* alpha;
 double h = 1.2;
 double alphastar;
 double xhat[numattributes_m];
 int jstar=0;
 double *kone;
 double *kx;
 double *kx_tilde;
 double **jkx;
```

```
double **jkx_tilde;
double f,p;
double konesum =0.0;
char output[40];
char err[30];

kx=(double *)malloc(numentities_n*sizeof(double));
kone = (double *)calloc(numentities_n,sizeof(double));
kx_tilde = (double *)calloc(numentities_n,sizeof(double));
xbzx =  (double *)calloc(numattributes_m,sizeof(double));
jkx=(double **)calloc(numentities_n,sizeof(double *));
jkx_tilde=(double **)calloc(numentities_n,sizeof(double *));
for (i=0;i<numentities_n;++i)
{
 jkx[i]=(double *)calloc(numattributes_m,sizeof(double));
 jkx_tilde[i]=(double *)calloc(numattributes_m,sizeof(double));
}

double *Y,*jt,*C,*zmiusx,*jkxcolumsum,*maxA,*zmiusxalpha,*gpri
↪  me,f_exh,gp_exh;
jt= (double *)calloc((entityinfo->numentities_n*entityinfo->nu
↪  mattributes_m),sizeof(double));
C=  (double *)calloc((entityinfo->numentities_n*entityinfo->nu
↪  mattributes_m),sizeof(double));
Y =  (double *)calloc(numentities_n,sizeof(double));
zmiusx = (double *)calloc((entityinfo->numentities_n*entityinf
↪  o->numattributes_m),sizeof(double));
zmiusxalpha = (double *)calloc((entityinfo->numentities_n*enti
↪  tyinfo->numattributes_m),sizeof(double));
alpha =(double *)malloc(600*sizeof(double));
jkxcolumsum = (double *)calloc(numattributes_m,sizeof(double));
maxA =  (double *)calloc(numentities_n,sizeof(double));
gprime = (double *)calloc(numattributes_m,sizeof(double));

p = exp(-1.0/((probleminfo->var)));
konesum=0.0;
for(i=0;i<numentities_n;++i)
{
 kone[i] =0.0;
 for (j=0;j<numentities_n;++j) kone[i] += entityinfo->KK[i][j];
 konesum += kone[i];
}

for (z=0;z<numentities_n;z++)
{
 for (i = 0;i<numentities_n;++i)
 for (j = 0;j<numattributes_m;++j)
```

```
 zmiusx[i*numattributes_m+j] = points_XT[z*numattributes_m+j]
  ↪  -points_XT[i*numattributes_m+j];

for (j =0,sum2=0.0;j<numentities_n;++j)
{
 kx[j]=1.0;
 for (l=0;l<numattributes_m;++l) kx[j] *= pow(p,zmiusx[j*numa
  ↪  ttributes_m+l]*zmiusx[j*numattributes_m+l]);
 sum2 += kx[j];
}
for (i=0;i<numentities_n;++i)
 kx_tilde[i]=kx[i] - (kone[i]+sum2)/numentities_n+konesum/(nu
  ↪  mentities_n*numentities_n);

for (i=0;i<numattributes_m;++i)
{
 jkxcolumsum[i] =0.0;
 for (j = 0,sum1=0.0;j<numentities_n;++j)
 {
  jkx[j][i] = kx[j] *2*log(p)*(zmiusx[j*numattributes_m+i]);
  jkxcolumsum[i] += jkx[j][i];
 }
 for (j=0;j<numentities_n;++j) jt[j*numattributes_m+i]=jkx[j]
  ↪  [i]-jkxcolumsum[i]/numentities_n;
}

for (i=0;i<numentities_n;++i) Y[i] = 0.0;
cblas_dgemv(CblasRowMajor,CblasNoTrans,numentities_n,numentit
 ↪  ies_n,1.0,entityinfo->a,numentities_n,kx_tilde,1,0.0,Y,1);
sum = cblas_ddot(numentities_n,kx_tilde,1,Y,1);
 ↪
f = 1-(2*sum2/numentities_n)+konesum/(numentities_n*numentiti
 ↪  es_n)-sum;
if ( f > 10e-9)
{
 memcpy(xbzx,jkxcolumsum,sizeof(double)*numattributes_m);
  ↪
 cblas_dgemm(CblasRowMajor,CblasTrans,CblasNoTrans,numattribu
  ↪  tes_m,numentities_n,numentities_n,1.0,jt,numattributes_m
  ↪  ,entityinfo->a,numentities_n,0.0,C,numentities_n);
 cblas_dgemv(CblasRowMajor,CblasNoTrans,numattributes_m,numen
  ↪  tities_n,2.0,C,numentities_n,kx_tilde,1,2.0/numentities_
  ↪  n,xbzx,1);
 sum4 = cblas_dnrm2(numattributes_m,xbzx,1);
 for (i = 0;i<numattributes_m;++i) xbzx[i] = xbzx[i]/sum4;
  ↪
```

```
cblas_dgemv(CblasRowMajor,CblasNoTrans,numentities_n,numattr
↪  ibutes_m,1.0,points_XT,numattributes_m,xbzx,1,0.0,maxA,1
↪  );
A = maxA[0];
for (i=0;i<numentities_n;++i) if (maxA[i]> A && i !=z) A =
↪  maxA[i];
A = A-maxA[z];
int r = 0;
alpha[r] =0.0;
while(alpha[r]< A )
{
 alpha[r+1] = alpha[r] + A*h/499.0;
 r = r + 1;
};
alpha[r]=A;
alphastar=f;
for (jstar=1;jstar<r+1;++jstar)
{
 for (l= 0;l<numentities_n;++l){
 for (j=0;j<numattributes_m;++j)
 zmiusxalpha[l*numattributes_m+j] =
 ↪  zmiusx[l*numattributes_m+j]+alpha[jstar]*xbzx[j];}
 sum1 =0.0;
 for (j=0;j<numentities_n;++j)
 {
  kx[j]=1.0;
  for (l=0;l<numattributes_m;++l)
   kx[j] *= pow(p,zmiusxalpha[j*numattributes_m+l]*zmiusxalp
   ↪  ha[j*numattributes_m+l]);
      sum1 += kx[j];
 }

 for (m=0;m<numentities_n;++m)
  kx_tilde[m]=kx[m] - (kone[m]-sum1)/numentities_n+konesum/(
  ↪  numentities_n*numentities_n);

 memset(Y,0,sizeof(double)*numentities_n);
 cblas_dgemv(CblasRowMajor,CblasNoTrans,numentities_n,nument
 ↪  ities_n,1.0,entityinfo->a,numentities_n,kx_tilde,1,1.0,
 ↪  Y,1);
 sum = cblas_ddot(numentities_n,kx_tilde,1,Y,1);
 f_exh = (1-(2*sum1/numentities_n)+konesum/(numentities_n*nu
 ↪  mentities_n)-sum);
 if (f_exh > alphastar)
 {
  if(jstar>1)
  {
   for (l= 0;l<numentities_n;++l)
    for (j=0;j<numattributes_m;++j)
```

```
        zmiusxalpha[l*numattributes_m+j] =
         ↪  zmiusx[l*numattributes_m+j]+alpha[jstar-1]*xbzx[j];

      for (j =0,sum1=0.0;j<numentities_n;++j)
      {
       kx[j] =1.0;
       for (l=0;l<numattributes_m;++l)
        kx[j] *= pow(p,zmiusxalpha[j*numattributes_m+l]*zmiusxa
         ↪  lpha[j*numattributes_m+l]);
              sum1 += kx[j];
      }
      for (j=0;j<numentities_n;++j)
       kx_tilde[j]=kx[j] - (kone[j]-sum1)/numentities_n+konesum
        ↪  /(numentities_n*numentities_n);

      for (m=0;m<numattributes_m;++m)
      {
       jkxcolumsum[m]=0.0;
       for (j = 0;j<numentities_n;++j)
       {
        jkx[j][m] =   kx[j]
         ↪  *2*log(p)*(zmiusxalpha[j*numattributes_m+m]);
        jkxcolumsum[m] += jkx[j][m];
       }
       for (j=0;j<numentities_n;++j) jt[j*numattributes_m+m]=
        ↪  jkx[j][m]-jkxcolumsum[m]/numentities_n;
      }
      memcpy(gprime,jkxcolumsum,sizeof(double)*numattributes_m)
       ↪  ;
      memset(C,0,numattributes_m*numentities_n*sizeof(double));
      cblas_dgemm(CblasRowMajor,CblasTrans,CblasNoTrans,numattr
       ↪  ibutes_m,numentities_n,numentities_n,1.0,jt,numattrib
       ↪  utes_m,entityinfo->a,numentities_n,0.0,C,numentities_
       ↪  n);
      cblas_dgemv(CblasRowMajor,CblasNoTrans,numattributes_m,nu
       ↪  mentities_n,-2.0,C,numentities_n,kx_tilde,1,-2.0/nume
       ↪  ntities_n,gprime,1);
      gp_exh = cblas_ddot(numattributes_m,gprime,1,xbzx,1);
      jstar =(gp_exh > 0.0)? jstar-1:jstar;
      break;
     }

    }
    alphastar = f_exh;
   }

   if (jstar > 1)
   {
```

```
    for (l=0;l<numattributes_m;++l)
     xhat[l] = points_XT[z*numattributes_m+l]+0.5*(alpha[jstar-
      ↪ 1]+alpha[jstar])*xbzx[l];
   }
   else
   {
    for (l=0;l<numattributes_m;++l)
     xhat[l] =points_XT[z*numattributes_m+l]+(alpha[jstar])*xbz
      ↪ x[l];
   }
  }
  else
  {
   for (l=0;l<numattributes_m;++l) xhat[l] =
    ↪ points_XT[z*numattributes_m+l];
  }
 }

        free(Y);
        free(jt);
        free(C);
        free(zmiusx);
        free(jkxcolumsum);
        free(maxA);
        free(zmiusxalpha);
        free(points_XT);
        free(kx);
        free(kone);
        free(kx_tilde);
        free(xbzx);
        free(jkx);
        free(jkx_tilde);
        free(gprime);
        return 0;
}
```

VITA

Xiao Ling, originally from Shanghai, China, graduated with a Bachelor of Electrical Engineering degree, a Master of Science in Statistics.

Bibliography

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[2] Bayar Azeez and Fattah Alizadeh. Review and classification of trending background subtraction-based object detection techniques. In *2020 6th International Engineering Conference "Sustainable Technology and Development"(IEC)*, pages 185–190. IEEE, 2020.

[3] Jianchao Bai, Jicheng Li, Fengmin Xu, and Hongchao Zhang. Generalized symmetric ADMM for separable convex optimization. *Computational Optimization and Applications*, 70(1):129–170, 2018.

[4] Ronen Basri and David W. Jacobs. Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003. ISSN 01628828. doi: 10.1109/TPAMI.2003.1177153.

[5] Weiss Ben. Fast median and bilateral filtering. *ACM SIGGRAPH 2006*, pages 519–526, 2006.

[6] Eric Bennett and Leonard McMillan. Video Enhancement Using Per-Pixel Virtual Exposures. *ACM Transactions on Graphics*, 24(3):845–852, 2005.

[7] Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

[8] Thierry Bouwmans. Traditional and recent approaches in background modeling for foreground detection: An overview. *Computer Science Review*, 11-12: 31–66, 2014.

[9] Thierry Bouwmans, Fatih Porikli, Benjamin Höferlin, and Antoine Vacavant. *Background Modeling and Foreground Detection for Video Surveillance*. Chapman and Hall/CRC, London, 2015. ISBN 1482205378.

[10] James Paul Brooks and José H Dulá. Estimating L1-norm best-fit lines for data. *Optimization Online*, 2017.

[11] James Paul Brooks and José H Dulá. Approximating l1-norm best-fit lines. *Optimization Online*, 2019.

[12] James Paul Brooks, José H Dulá, and Edward L Boone. A pure L_1-norm principal component analysis. *Computational Statistics Data Analysis*, 61:83–98, 2013. ISSN 0167-9473.

[13] James Paul Brooks, José H Dulá, Amy L. Pakyz, and Ronald E. Polk. Identifying hospital antimicrobial resistance targets via robust ranking. *IISE Transactions on Healthcare Systems Engineering*, 7(3):121–128, 2017. ISSN 24725587. doi: 10.1080/24725579.2017.1339148.

[14] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Non-local means denoising. *Image Processing On Line*, 1:208–212, 2011.

[15] Anh Tuan Bui, Joon-Ku Im, Daniel W Apley, and George C Runger. Projection-free kernel principal component analysis for denoising. *Neurocomputing*, 357:163–176, 2019.

[16] Jian Feng Cai, Emmanuel J Candès, and Zuo wei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[17] Emmanuel Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):1–37, 2011. ISSN 0004-5411.

[18] Dan Song Cheng, Jian Zhe Yang, Jun Wang, Da Ming Shi, and Xiao Fang Liu. Double-noise-dual-problem approach to the augmented lagrange multiplier method for robust principal component analysis. *Soft Computing*, 21(10): 2723–2732, 2017.

[19] Flavio Chierichetti, Ravi Kumar, Prabhakar Raghavan, and Tamas Sarlos. Are web users really Markovian? In *Proceedings of the 21st International Conference on World Wide Web*, pages 609–618. ACM, 2012. ISBN 1450312292.

[20] Flavio Chierichetti, Sreenivas Gollapudi, Ravi Kumar, Silvio Lattanzi, Rina Panigrahy, and David P. Woodruff. Algorithms for ℓ_p low-rank approximation. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 806–814, 2017.

[21] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[22] Fernando De la Torre and Michael J Black. Robust principal component analysis for computer vision. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 362–369. IEEE, 2001.

[23] Charles-Alban Deledalle, Loïc Denis, and Florence Tupin. Iterative weighted

maximum likelihood denoising with probabilistic patch-based weights. *IEEE Transactions on Image Processing*, 18(12):2661–2672, 2009.

[24] Charles-Alban Deledalle, Joseph Salmon, Arnak S Dalalyan, et al. Image denoising with patch-based PCA: local versus global. In *British Machine Vision Conference*, volume 81, pages 425–455, 2011.

[25] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 257–266, 2002.

[26] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[27] Jan-Mark Geusebroek, Gertjan J Burghouts, and Arnold WM Smeulders. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

[28] Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1-norm low-rank matrix approximation. *Mathematics of Operations Research*, 43:1072–1084, 2018.

[29] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[30] Deren Han, Xiaoming Yuan, Wenxing Zhang, and Xingju Cai. An adm-based splitting method for separable convex programming. *Computational Optimization and Applications*, 54(2):343–369, 2013.

[31] Bingsheng He, Min Tao, and Xiaoming Yuan. Alternating direction method with gaussian back substitution for separable convex programming. *SIAM Journal on Optimization*, 22(2):313–340, 2012.

[32] Paul Honeine and Cédric Richard. A closed-form solution for the pre-image problem in kernel-based machines. *Journal of Signal Processing Systems*, 65 (3):289–299, 2011.

[33] Hui Ji, Sibin Huang, Zuowei Shen, and Yuhong Xu. Robust video restoration by joint sparse and low rank matrix approximation. *SIAM Journal on Imaging Sciences*, 4(4):1122–1142, 2011.

[34] Xixi Jia, Xiangchu Feng, and Weiwei Wang. Adaptive regularizer learning for low rank approximation with application to image denoising. In *2016 IEEE International Conference on Image Processing*, pages 3096–3100. IEEE, 2016.

[35] Pierre-Marc Jodoin, Lucia Maddalena, Alfredo Petrosino, and Yi Wang. Extensive benchmark and survey of modeling methods for scene background initialization. *IEEE Transactions on Image Processing*, 26(11):5244–5256, 2017.

[36] Zhao Kang, Chong Peng, and Qiang Cheng. Robust pca via nonconvex rank approximation. In *2015 IEEE International Conference on Data Mining*, pages 211–220. IEEE, 2015.

[37] Qi Fa Ke and Takeo Kanade. Robust subspace computation using L1 norm. Technical report, Carnegie Mellon University, 2003.

[38] Qi Fa Ke and Takeo Kanade. Robust L1-norm factorization in the presence of outliers and missing data by alternative convex programming. In *2005*

IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), pages 739–746, 2005.

[39] Cheolmin Kim and Diego Klabjan. A simple and fast algorithm for ℓ_1-norm kernel PCA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42 (8):1842–1855, 2019.

[40] Nojun Kwak. Principal component analysis based on L_1-norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008. ISSN 01628828. doi: 10.1109/TPAMI.2008.114.

[41] Nojun Kwak. Nonlinear Projection Trick in Kernel Methods: An Alternative to the Kernel Trick. *IEEE Transactions on Neural Networks and Learning Systems*, 24(12):2113–2119, 2013.

[42] James T Kwok and Ivor W Tsang. The pre-image problem in kernel methods. *IEEE Transactions on Neural Networks*, 15(6):1517–1525, 2004.

[43] Andrew H S Lai and Nelson Hon Ching Yung. A fast and accurate scoreboard algorithm for estimating stationary backgrounds in an image sequence. In *1998 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 241–244 vol.4, 1998. doi: 10.1109/ISCAS.1998.698804.

[44] Chul Lee and Edmund Y Lam. Computationally efficient truncated nuclear norm minimization for high dynamic range imaging. *IEEE Transactions on Image Processing*, 25(9):4145–4157, 2016.

[45] Amos Lev, Steven W Zucker, and Azriel Rosenfeld. Iterative enhancemnent of noisy images. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(6): 435–442, 1977.

[46] Yinbo Li and Gonzalo R Arce. A maximum likelihood approach to least absolute deviation regression. *EURASIP Journal on Advances in Signal Processing*, 2004(12):1–8, 2004.

[47] Zhouchen Lin, Minming Chen, and Yi Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[48] Zhouchen Lin, Risheng Liu, and Zhixun Su. Linearized alternating direction method with adaptive penalty for low-rank representation. *Advances in Neural Information Processing Systems*, 24, 2011.

[49] Guang can Liu, Zhou chen Lin, Yong Yu, et al. Robust subspace segmentation by low-rank representation. In *International Conference on Machine Learning*, 2010.

[50] Guangcan Liu and Shuicheng Yan. Active Subspace: Toward Scalable Low-Rank Learning. *Neural Computation*, 24(12):3371–3394, 2012. doi: 10.1162/NECO_a_00369.

[51] Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery of subspace structures by low-rank representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):171–184, 2012.

[52] Jing Liu, Yong Rui Duan, and Tong Hui Wang. A Parallel Splitting Augmented Lagrangian Method for Two-Block Separable Convex Programming with Application in Image Processing. *Mathematical Problems in Engineering*, 2020, 2020.

[53] Risheng Liu, Zhouchen Lin, Siming Wei, and Zhixun Su. Solving principal component pursuit in linear time via ℓ_1 filtering. *arXiv preprint arXiv:1108.5359*, 2011.

[54] Lucia Maddalena and Alfredo Petrosino. A self-organizing approach to background subtraction for visual surveillance applications. *IEEE Transactions on Image Processing*, 17(7):1168–1177, 2008.

[55] Panos P Markopoulos, George N Karystinos, and Dimitris A Pados. Optimal algorithms for L1-subspace signal processing. *IEEE Transactions on Signal Processing*, 62(19):5046–5058, 2014.

[56] Panos P Markopoulos, Mayur Dhanaraj, and Andreas Savakis. Adaptive L1-norm principal component analysis with online outlier rejection. *IEEE Journal on Selected Topics in Signal Processing*, 12(6):1131–1143, 12 2018. ISSN 19324553. doi: 10.1109/JSTSP.2018.2874165.

[57] Deyu Meng, Qian Zhao, and Zongben Xu. Improve robustness of sparse PCA by L1-norm maximization. *Pattern Recognition*, 45(1):487–497, 2012.

[58] Sebastian Mika, Bernhard Schölkopf, Alexander J Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel PCA and de-noising in feature spaces. *Advances in Neural Information Processing Systems*, 11, 1998.

[59] Kerui Min, Zhengdong Zhang, John Wright, and Yi Ma. Decomposing background topics from keywords by principal component pursuit. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 269–278, 2010.

[60] Milan Paluš and Ivan Dvořák. Singular-value decomposition in attractor reconstruction: pitfalls and precautions. *Physica D: Nonlinear Phenomena*, 55 (1-2):221–234, 1992.

[61] Haotian Pang, Han Liu, Robert J Vanderbei, and Tuo Zhao. Parametric simplex method for sparse learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[62] Stanislav Pyatykh, Jürgen Hesser, and Lei Zheng. Image noise level estimation by principal component analysis. *IEEE Transactions on Image Processing*, 22(2):687–699, 2012.

[63] Qing Qu, Ju Sun, and John Wright. Finding a sparse vector in a subspace: Linear sparsity using alternating directions. *IEEE Transactions on Information Theory*, 62(10):5855–5880, 2016. doi: 10.1109/TIT.2016.2601599.

[64] Robert Reris and James Paul Brooks. Principal component analysis and optimization: A tutorial. 2015.

[65] Robert A. Reris and James P. Brooks. Principal component analysis and optimization: A tutorial. In *Proceedings of the 14th INFORMS Computing Society Conference*, pages 212–225, 2015.

[66] Paul Rodriguez and Brendt Wohlberg. Fast principal component pursuit via alternating minimization. In *2013 IEEE International Conference on Image Processing*, pages 69–73. IEEE, 2013. ISBN 1479923419.

[67] Paul Rodríguez and Brendt Wohlberg. Fast principal component pursuit via alternating minimization. In *2013 IEEE International Conference on Image Processing*, pages 69–73, 2013. doi: 10.1109/ICIP.2013.6738015.

[68] Roman Rosipal, Mark Girolami, Leonard J Trejo, and Andrzej Cichocki. Kernel PCA for feature extraction and de-noising in nonlinear regression. *Neural Computing & Applications*, 10(3):231–243, 2001.

[69] Yogesh Rathi Samuel, Samuel Dambreville, and Allen Tannenbaum. Statistical shape analysis using kernel PCA. In *SPIE, Electronic Imaging.(2006*. Citeseer, 2006.

[70] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[71] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[72] Bernhard Schölkopf, Sebastian Mika, Alex Smola, Gunnar Rätsch, and Klaus-Robert Müller. Kernel PCA pattern reconstruction via approximate pre-images. In *International Conference on Artificial Neural Networks*, pages 147–152. Springer, 1998.

[73] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10 (5):1299–1319, 1998.

[74] Bernhard Schölkopf, Sebastian Mika, Chris JC Burges, Philipp Knirsch, K-R Muller, Gunnar Ratsch, and Alexander J Smola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5): 1000–1017, 1999.

[75] Kamal Sehairi, Fatima Chouireb, and Jean Meunier. Comparative study of

motion detection methods for video surveillance systems. *Journal of Electronic Imaging*, 26(2):023025, 2017.

[76] Sreelekshmy Selvin, SG Ajay, B Ganga Gowri, V Sowmya, and KP Soman. ℓ_1 trend filter for image denoising. *Procedia Computer Science*, 93:495–502, 2016.

[77] Yuan Shen, Hongyu Xu, and Xin Liu. An alternating minimization method for robust principal component analysis. *Optimization Methods and Software*, 34(6):1251–1276, 2019. ISSN 1055-6788.

[78] Yuan Shen, Yannian Zuo, and Aolin Yu. A Partial PPa S-ADMM for Multi-Block for Separable Convex Optimization with Linear Constraints. *Optimization Online*, 2020.

[79] Andrews Sobral, Thierry Bouwmans, and El-hadi Zahzah. LRSlibrary: Low-rank and sparse tools for background modeling and subtraction in videos. *Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing*, 2016.

[80] Zhao Song, David P Woodruff, and Pei Lin Zhong. Low rank approximation with entrywise ℓ1-norm error. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2017. ISBN 9781450345286. doi: 10.1145/3055399.3055431.

[81] Anuj Srivastava, Ann B Lee, Eero P Simoncelli, and S-C Zhu. On advances in statistical modeling of natural images. *Journal of Mathematical Imaging and Vision*, 18(1):17–33, 2003.

[82] Gongguo Tang and Arye Nehorai. Robust principal component analysis

based on low-rank and block-sparse matrix decomposition. In *2011 45th Annual Conference on Information Sciences and Systems*, pages 1–5. IEEE, 2011.

[83] Min Tao and Xiao ming Yuan. Recovering Low-Rank and Sparse Components of Matrices from Incomplete and Noisy Observations. *SIAM Journal on Optimization*, 21(1):57–81, 2011.

[84] Min Tao and Xiaoming Yuan. Recovering low-rank and sparse components of matrices from incomplete and noisy observations. *SIAM Journal on Optimization*, 21(1):57–81, 2011.

[85] Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[86] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 839–846. IEEE, 1998.

[87] K Toyama, J Krumm, B Brumitt, and B Meyers. Principles and practice of background maintenance. In *The Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 255–261, 1999.

[88] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 255–261. IEEE, 1999.

[89] Mohan Trivedi, Shailendra Bhonsle, and Amarnath Gupta. Database architecture for autonomous transportation agents for on-scene networked inci-

dent management (ATON). In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 4, pages 664–667. IEEE, 2000.

[90] Nicholas Tsagkarakis, Panos P Markopoulos, and Dimitris A Pados. On the L1-norm approximation of a matrix by another of lower rank. In *Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016*, pages 768–773. Institute of Electrical and Electronics Engineers Inc., 1 2017.

[91] Vladimir N Vapnik. *Nature of Statistical Learning Theory*. Springer, 2013. ISBN 0387945598.

[92] Hansheng Wang, Guodong Li, and Guohua Jiang. Robust regression shrinkage and consistent variable selection through the LAD-Lasso. *Journal of Business Economic Statistics*, 25(3):347–355, 2007.

[93] Lie Wang. The L1 penalized LAD estimator for high dimensional linear regression. *Journal of Multivariate Analysis*, 120:135–151, 2013.

[94] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[95] Huan Xu, Constantine Caramanis, and Sujay Sanghavi. Robust PCA via Outlier Pursuit. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL `https://proceedings.neurips.cc/paper/2010/file/fe8c15fed5f808006ce95eddb7366e35-Paper.pdf`.

[96] Lei Xu, Erkki Oja, and Ching Y Suen. Modified Hebbian learning for curve and surface fitting. *Neural Networks*, 5(3):441–457, 1992.

[97] Jun Feng Yang and Yin Zhang. Alternating direction algorithms for ℓ_1-problems in compressive sensing. *SIAM Journal on Scientific Computing*, 33 (1):250–278, 2011.

[98] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[99] Yongqin Zhang, Ruiwen Kang, Xianlin Peng, Jun Wang, Jihua Zhu, Jinye Peng, and Hangfan Liu. Image denoising via structure-constrained low-rank approximation. *Neural Computing and Applications*, 32(16):12575–12590, 2020.

[100] Yin qiang Zheng, Guang can Liu, Shigeki Sugimoto, Shui Cheng Yan, and Masatoshi Okutomi. Practical low-rank matrix approximation under robust l1-norm. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1410–1417. IEEE, 2012.

[101] Tianyi Zhou and Dacheng Tao. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011.