SPARSE AND ORTHOGONAL SINGULAR VALUE DECOMPOSITION

by

ROHAN KHATAVKAR

B.S., Institute of Chemical Technology, 2007
M.S., University of Georgia, 2009

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2013

Approved by:

Major Professor
Dr. Kun Chen

Copyright

Abstract

The singular value decomposition (SVD) is a commonly used matrix factorization technique in statistics, and it is very effective in revealing many low-dimensional structures in a noisy data matrix or a coefficient matrix of a statistical model. In particular, it is often desirable to obtain a sparse SVD, i.e., only a few singular values are nonzero and their corresponding left and right singular vectors are also sparse. However, in several existing methods for sparse SVD estimation, the exact orthogonality among the singular vectors are often sacrificed due to the difficulty in incorporating the non-convex orthogonality constraint in sparse estimation. Imposing orthogonality in addition to sparsity, albeit difficult, can be critical in restricting and guiding the search of the sparsity pattern and facilitating model interpretation. Combining the ideas of penalized regression and Bregman iterative methods, we propose two methods that strive to achieve the dual goal of sparse and orthogonal SVD estimation, in the general framework of high dimensional multivariate regression. We set up simulation studies to demonstrate the efficacy of the proposed methods.

Key words and Phrases: bregman iteration, multivariate regression, orthogonality constraint, singular value decomposition, sparsity.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am grateful to the Department of Statistics at Kansas State University for providing the platform for my master's program. Without the teaching support and excellent grooming by all the professors I wouldn't have come along so far. They say life is all about timing and I am lucky to have my major professor, Dr. Kun Chen, guide me at the right time. I consider myself privileged to have worked under him and develop a keen sense of interest and motivation in High Dimensional Statistics and Optimization theory in general. I find myself wanting to learn more and explore both fundamental and cutting edge topics in my research area. I take inspiration from Dr. Chen's hard work, leadership, and above all learn the art of engaging others in one's passionate interest.

I would like to sincerely thank my committee members, Dr. Weixing Song and Dr. Gary Gadbury, for their insightful comments, support and guidance. Last but not the least I thank my fellow graduate students in the department for their help and emotional support.

Chapter 1

Introduction

High dimensional models have captured the imagination of many in recent times owing to the ease with which large amounts of data can be collected, stored and processed by modern computing facilities. It isn't a surprise that high dimensional models arise in diverse fields of scientific research such as: genomic/genetic studies, economics, marketing, finance, image processing, medicine, etc. With large amounts of data at our disposal, the challenge is to develop efficient tools that can extract the consistent patterns from the noisy observations. In statistical language, high dimensional methods aim at revealing paramount aspects of the underlying true model: low dimensionality and sparsity, for example, so that the resulting estimator facilitates ease of interpretation and improved predictive accuracy. Several improvements over the traditional least squares method have been proposed under the penalized estimation framework, e.g., Lasso (Tibshirani, 1996), adaptive Lasso (Zou, 2006), MCP (Zhang, 2010), and SCAD (Yuan and Lin, 2006).

A motivating example which involves imposing linear constraints on Lasso finds its application in gross-exposure portfolio optimization problem (Fan *et al.*, 2012). In the world of finance an important goal is to minimize portfolio risk of large number of assets. Let $\mathbf{w}$ be the vector of portfolio weights and $\hat{\mathbf{\Sigma}}$ denote the estimated covariance matrix for the returns on the assets in the portfolio. A common portfolio optimization problem is to minimize the portfolio risk, $\mathbf{w}^\top \hat{\mathbf{\Sigma}} \mathbf{w}$. Fan *et al.* (2012) approximately solved the portfolio optimization problem using a gross-exposure parameter c (tuning parameter) via constraining the asset

weights:

$$\mathbf{w}^\top \hat{\mathbf{\Sigma}} \mathbf{w}, \ \ \|\mathbf{w}\|_1 \leq c, \quad \text{subject to } \mathbf{w}^\top \mathbf{1} = 1.$$

More generally, it can be shown that the above optimization problem is a special case of a constrained Lasso problem:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} w_j |\beta_j| \qquad \text{subject to } \mathbf{A}\boldsymbol{\beta} = \mathbf{b},$$

where $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_p) \in \mathbb{R}^{h \times p}$, $\mathbf{b} \in \mathbb{R}^{h \times 1}$, $\mathbf{w} = (w_1, ..., w_p)^\top$ are some predetermined weights, and λ is a tuning parameter controlling the degree of l_1 penalization. Here we assume that the coefficient vector $\boldsymbol{\beta}$ satisfies h known linear constraints. For the portfolio optimization problem the matrix $\mathbf{A}$ will have at least one row to ensure that the weights $\mathbf{w}$ sum up to one. Additional constraints may be placed on expected return, industry weightings, etc. The constrained Lasso problem can be used to exactly solve the above mentioned portfolio optimization.

Another motivating example is the multivariate reduced rank regression problems for which several high dimensional methods have been recently proposed. Based on singular value decomposition (SVD), the reduced rank estimator of a coefficient matrix of rank r^* can be written as a sum of r^* unit rank matrices. Each of the unit rank matrices, also called SVD layers, is proportional to the outer product of the left and right singular vectors. Chen *et al.* (2012) proposed a sparse reduced rank regression method to induce sparsity in the aforementioned left and right singular vectors for easier interpretation and improved prediction performance. The method was shown to be selection consistent, asymptotically normal and possess oracle properties. The usefulness of such novel ideas is apparent when studying microarray gene expression data (Lee *et al.*, 2010). A typical microarray gene expression data is high dimensional low sample size (HDLSS), wherein the expression levels of thousands of genes are measured for a small number of subjects. The goal in some applications is to recognize sets of *biologically relevant* genes that are significantly expressed for certain cancer types. The bi-clustering via sparse singular value decomposition (Lee

et al., 2010; Chen *et al.*, 2012) is a novel approaches of low rank matrix approximation for simultaneously selecting significant genes and relevant subject groups that possibly represent different cancer types. Chen *et al.* (2012) demonstrated that the microarray gene expression data can be more effectively interpreted under a reduced rank regression framework by incorporating the prior cancer type information for supervised learning of the gene clusters and their contrasts across the different types of cancer.

Often the orthogonality among the estimated left and right singular vectors is relaxed to facilitate an efficient search for their sparsity patterns. However, if we desire to retain the orthogonality among the estimated sparse left and right singular vectors then we delve into the realm of imposing constraints on the regularization procedure in addition to gaining sparsity. Therefore, we strive to devise a methodology to achieve the dual goal: 1) sparsity within the estimated left and right singular vectors, 2) ensuring that the estimated left and right singular vectors are orthogonal within themselves. In this report we propose two methods that aim at obtaining sparse and orthogonal SVD for high dimensional multivariate regression problems. The orthogonality constraint is incorporated or enhanced using Bregman iterative methods and we show the usefulness of Coordinate Descent Algorithms (CDA) to locally solve challenging non-convex optimization problems.

The rest of the report is organized as follows. In Chapter 2, we briefly review existing penalized regression methods for shrinkage estimation and variable selection. In Chapter 3, we propose the Bregman Coordinate Descent Algorithm (BCDA) to solve the constrained adaptive Lasso problem and demonstrate its usefulness by conducting simulation studies. In Chapter 4, we propose two methods to perform sparse and orthogonal singular value decomposition. Both methods use the BCDA to impose constraints needed to promote or enhance orthogonality. We also demonstrate the usefulness of these two methods by conducting simulation studies. Finally, we wrap up by discussing the important learnings from the report and suggesting some possible directions for future research.

Chapter 2

Literature Review

2.1 Introduction

We start our journey by introducing some basic concepts of penalized regression and eventually applying it to sparse singular value decomposition (SSVD) of the coefficient matrix in a multivariate regression model. To understand the motivation for penalized regression, consider the familiar linear regression model:

$$y_i = x_{i1}\beta_1 + ... + x_{ip}\beta_p + \epsilon_i, \quad 1 \leq i \leq n, \quad 1 \leq j \leq p, \tag{2.1}$$

where n is the number of observed response $y_i \in \mathbb{R}$ and predictor $\mathbf{x}_i \in \mathbb{R}^p$, ϵ_i is the random error assumed to be independently and identically distributed with mean mean zero and some constant variance, and $\beta_1, ..., \beta_p$ are the regression coefficients. We assume, without loss of generality, the response are centered, i.e., $\sum_{i=1}^n y_i = 0$, and the predictors are centered and standardized, i.e., $\sum_{i=1}^n x_{ij} = 0$, $\sum_{i=1}^n x_{ij}^2 = n$, for $1 \leq j \leq p$. Therefore, equation (2.1) has no intercept term. Equivalently we can express the above expression in terms of matrix notation as:

$$\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times p}\boldsymbol{\beta}_{p\times 1} + \boldsymbol{\epsilon}_{n\times 1}.$$

where $\mathbf{X}_{n\times p}$ denotes the design matrix whose n rows are the predictors $\mathbf{x}_i \in \mathbb{R}^p$, $\mathbf{y}_{n\times 1}$ is the response vector, $\boldsymbol{\epsilon}_{n\times 1}$ is the error vector, and $\boldsymbol{\beta}_{p\times 1}$ is the regression coefficient vector. The

ordinary least squares estimate (OLS), $\hat{\boldsymbol{\beta}}_{LS}$, minimizes the objective function:

$$\hat{\boldsymbol{\beta}}_{LS} = \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

In order to obtain a unique OLS estimate it is imperative that the $p \times p$ matrix, $\mathbf{X}^\top\mathbf{X}$, is invertible. Therefore, when $p > n$, the OLS estimate is not unique. Apart from the case when $p > n$, there are two well-known issues that demand improvement over OLS estimates (Hastie *et al.*, 2008): 1) the least square estimates, albeit unbiased, may have large variance thereby hampering the accuracy of prediction, and 2) the interpretation using the least square estimates is not terse, especially when the model has large number of predictors. Therefore, a smaller and most important subset of predictors needs to be extracted to achieve a reasonable interpretation.

The variable selection methods, such as forward and backward stepwise selection, that are intended to improve interpretation often yield estimates that have high variance due to the process being inherently discreet: the variables are either preserved or discarded from the overall model (Hastie *et al.*, 2008). Penalized regression methods can be implemented when confronted with above mentioned shortcomings of least squares estimates. In the penalized framework, some form of constraint is exerted on the parameters, for example, shrinking of the regression coefficients toward zero. Shrinkage methods are usually designed to be continuous unlike the discrete process of subset selection. We start with describing ridge regression, a traditional shrinkage method, followed by more effective regularization methods such as the Lasso and adaptive Lasso.

2.2 Ridge Regression

The ridge regression criteria enforces a constraint on the l_2 norm of the coefficient vector, $\boldsymbol{\beta}$. Similar to the least squares criteria, the goal in ridge regression is to minimize the residual sum of squares, although, subject to the squared l_2 norm of the coefficient vector being less

than a fixed threshold value, s. The ridge regression criterion is expressed as:

$$\hat{\boldsymbol{\beta}}^{ridge} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \quad \text{subject to } \|\boldsymbol{\beta}\|_2^2 \leq s.$$

Equivalently it can also be expressed as:

$$\hat{\boldsymbol{\beta}}^{ridge} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \quad \lambda \geq 0,$$

where λ is the tuning parameter which controls the amount of shrinkage. Note that λ and s have one-to-one relationship between them. Higher the value of the tuning parameter, larger is the shrinkage enforced.

An intuitive way to think about the ridge regression setup is to consider a modified version of the objective function for the usual regression set up:

$$L(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}} \|\mathbf{y}^* - \mathbf{X}^*\boldsymbol{\beta}\|_2^2, \tag{2.2}$$

where $\mathbf{y}^* = \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_{p\times 1} \end{pmatrix}_{(n+p)\times 1}$ and $\mathbf{X}^* = \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I}_{p\times p} \end{pmatrix}_{(n+p)\times p}$ are the new response and predictors respectively. The solution of objective function (2.2) is the ridge estimator given as:

$$\hat{\boldsymbol{\beta}}^{ridge} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_{p\times p})^{-1}\mathbf{X}^\top\mathbf{y}.$$

The ridge regression estimator is a linear function of the response, $\mathbf{y}$, and is unique for $\lambda > 0$ even if $\mathbf{X}$ is not full rank. This is true because adding a positive constant, λ, to the diagonal entries of $\mathbf{X}^\top\mathbf{X}$ makes it a nonsingular matrix and hence it has a unique inverse. From a practical point of view, we can compute the ridge estimator over a range of values for the tuning parameter, $\lambda \in [\lambda_{min}, \lambda_{max}]$, to obtain a solution path:

$$\{\hat{\boldsymbol{\beta}}^{ridge}(\lambda) : \lambda \in (\lambda_{min}, \lambda_{max})\}.$$

Using information criteria such as BIC we can select an optimal value for the tuning parameter to get the final ridge estimator. Even though the ridge estimator has desirable

qualities that make it useful in cases when $p > n$, it has certain limitations as far as variable selection is concerned. The constraint on the l_2 norm of $\boldsymbol{\beta}$ provides shrinkage, however, the estimated parameters are still not shrunk to exactly zero. The best way to understand this phenomenon is by considering a simple case when we have only two parameters, β_1 and β_2. If we visually portray such a situation then it is not difficult to see why ridge estimators do not give exact zero solutions. In Figure 2.1, the solution for the ridge estimator is the first

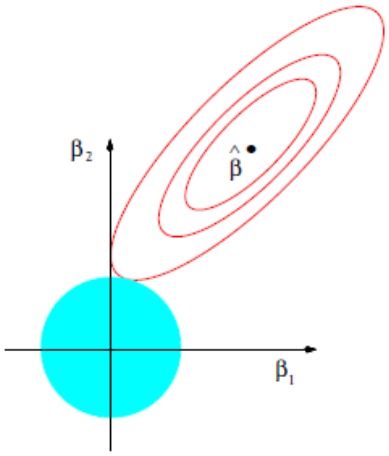

Figure 2.1: *Intersection of contours of the error function (centered at the full least squares estimate, $\hat{\beta}$) as shown with red ellipses with the solid blue area of the constraint region, $\beta_1^2 + \beta_2^2 \leq t^2$ (Hastie et al., 2008).*

place that the contour of the error function (ellipse) touch the constraint region (shown by the blue disk). To obtain a zero solution for either β_1 or β_2 the contour of the red ellipse must touch the blue disk at a single point. Therefore, the probability of estimating the parameters as exactly zero is zero. To address this issue it is worth thinking about different shapes of the constraint areas such that we have a better chance of obtaining zero solutions for the estimates. The next section builds upon this chain of thought.

2.3 Lasso

A simple strategy in penalized regression set up is to think about ways to minimize the residual sums of squares subject to such a constraint on the coefficients that can perform variable selection and shrinkage. The constraint region for the ridge regression method is disk shaped for $p = 2$ case (and an ellipsoid for $p > 2$ case) which makes it impossible for the elliptical contours to hit the constraint region where either of the parameter is zero as shown in Figure 2.1. The Lasso penalty, which stands for "least absolute shrinkage and selection operator", gives advantages of both shrinkage and variable selection by shrinking some coefficients and estimating the others to be exactly zero (Tibshirani, 1996). For the Lasso penalty the constraint region is a rotated square (or a diamond) for $p = 2$ case (which becomes a rhomboid for $p > 2$ case) as shown in Figure 2.2. Just like the ridge estimator

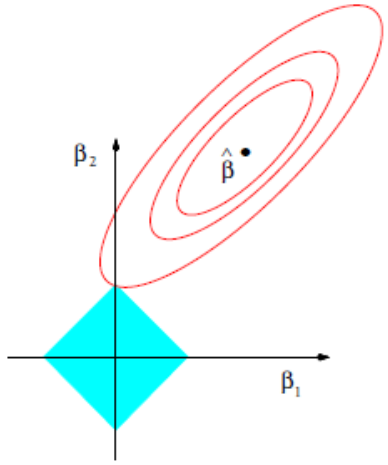

Figure 2.2: *Intersection of contours of the error function (centered at the full least squares estimate, $\hat{\beta}$) as shown with red ellipses with the solid blue area of the constraint region, $|\beta_1| + |\beta_2| \leq t$ (Hastie et al., 2008).*

the Lasso solution is the first place where the ellipse touches the rotated square. Note that the advantage of choosing the constraint space as a rotated square for $p = 2$ case is that the contour of the error function (shown by red ellipse) has a non zero probability of hitting the rotated square at one of the corners implying a zero solution for β_1 or β_2. It is easy to

imagine that for $p > 2$ cases the very design of the rhomboid will allow for many corners, flat edges and plain faces, thereby having the chance of the contour of error function to hit the rhomboid where some parameters can be estimated to be zero. To understand why the constraint area for Lasso is a rhomboid let us examine objective function for the Lasso problem using the same set up where the response and predictors are centered and the predictors are standardized:

$$\hat{\boldsymbol{\beta}}^{Lasso} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \quad \text{subject to } \|\boldsymbol{\beta}\|_1 \leq t.$$

Equivalently it can also be expressed as:

$$\hat{\boldsymbol{\beta}}^{Lasso} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1, \quad \lambda \geq 0, \tag{2.3}$$

where λ is the tuning parameter which controls the amount of shrinkage. $\|.\|_1$ denotes the $l1$ norm. Note that λ and t have one-to-one relationship between them. Higher the value of the tuning parameter, larger is the shrinkage enforced.

To obtain the Lasso solution we will use the "Coordinate Descent Algorithm" (CDA). CDA starts with an initial estimator for the parameters in the objective function; least squares or ridge estimator, for example, and then optimize the given objective function with respect to a single parameter at a time while keeping the remaining parameters fixed. This single parameter optimization step is performed for all the parameters in the objective function, thereby completing one cycle of optimization process. The entire optimization process is then repeated several times until the resulting estimators converge at which the algorithm terminates. CDA are simple to implement, stable and useful in cases when $p > n$.

So for the Lasso optimization problem let's start with an initial estimator, $\tilde{\boldsymbol{\beta}}$, and minimize the objective function 2.3, denoted as $L(\boldsymbol{\beta}; \lambda)$. As per the CDA let us fix all the parameters at their respective initial estimator except the j^{th} parameter, β_j, where $1 \leq j \leq p$. The fixed $p-1$ parameters are denoted as $\tilde{\beta}_k$, $k \neq j$. Define $\tilde{\mathbf{r}}_j = \mathbf{y} - \sum_{k \neq j} \tilde{\beta}_k \mathbf{x}_k$ as the partial residual with respect to the j^{th} parameter. Using this notation we can focus on

optimizing $L(\boldsymbol{\beta}; \lambda)$ with respect to β_j since the other parameters are fixed:

$$L_j(\beta_j; \lambda) = \frac{1}{2n}\|\tilde{\mathbf{r}}_j - \mathbf{x}_j\beta_j\|_2^2 + \lambda|\beta_j|.$$

We can look at the above objective function from the point of view of a new model with respect to a single parameter β_j, response $\tilde{\mathbf{r}}_j$, and predictor $\mathbf{x}_j$. Denote $\tilde{\beta}_j^{LS} = \mathbf{x}_j^\top \tilde{\mathbf{r}}_j / \mathbf{x}_j^\top \mathbf{x}_j$ as the least squares solution for β_j to express the above equation as:

$$\begin{aligned}
L_j(\beta_j; \lambda) &= \frac{1}{2n}\|\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS} + \mathbf{x}_j\tilde{\beta}_j^{LS} - \mathbf{x}_j\beta_j\|_2^2 + \lambda|\beta_j| \\
&= \frac{1}{2n}\|(\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS}) - (\mathbf{x}_j\beta_j - \mathbf{x}_j\tilde{\beta}_j^{LS})\|_2^2 + \lambda|\beta_j| \\
&= \frac{1}{2n}\|\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS}\|_2^2 + \frac{1}{2n}\|\mathbf{x}_j\beta_j - \mathbf{x}_j\tilde{\beta}_j^{LS}\|_2^2 - \frac{1}{n}(\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS})^\top(\mathbf{x}_j\beta_j - \mathbf{x}_j\tilde{\beta}_j^{LS}) + \lambda|\beta_j| \\
&= \frac{1}{2n}\|\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS}\|_2^2 + \frac{\mathbf{x}_j^\top\mathbf{x}_j}{2n}(\beta_j - \tilde{\beta}_j^{LS})^2 - \frac{1}{n}(\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS})^\top\mathbf{x}_j(\beta_j - \tilde{\beta}_j^{LS}) + \lambda|\beta_j|.
\end{aligned}$$

Note that the first term of the above equation is a constant and hence can be ignored as it is irrelevant for the optimization process. The cross term is zero since the error vector, $\tilde{\mathbf{r}}_j - \mathbf{x}_j\tilde{\beta}_j^{LS}$ is orthogonal to the space spanned by the predictor, $\mathbf{x}_j$. We can also multiply both sides of the objective function by a constant, $\frac{n}{\mathbf{x}_j^\top\mathbf{x}_j}$. Since multiplying by a constant does not affect the value of the optimizer we can simply express $L_j(\beta_j; \lambda)$ as:

$$L_j(\beta_j; \lambda) = \frac{1}{2}(\beta_j - \tilde{\beta}_j^{LS})^2 + \lambda^*|\beta_j| + const,$$

where $\lambda^* = n\lambda/\mathbf{x}_j^\top\mathbf{x}_j$. Tibshirani (1996) showed that the minimizer of the above objective function, $\hat{\beta}_j^{Lasso}$, is given by the following soft threshold operator:

$$\hat{\beta}_j^{Lasso} = sign(\tilde{\beta}_j^{LS})(|\tilde{\beta}_j^{LS}| - \lambda^*)_+ = \begin{cases} \tilde{\beta}_j^{LS} - \lambda^*, & \text{if } \tilde{\beta}_j^{LS} > \lambda^* \\ 0, & \text{if } |\tilde{\beta}_j^{LS}| \leq \lambda^* \\ \tilde{\beta}_j^{LS} + \lambda^*, & \text{if } \tilde{\beta}_j^{LS} < -\lambda^*. \end{cases} \tag{2.4}$$

After solving for the j^{th} optimizer, $\hat{\beta}_j^{Lasso}$, let's say that the next parameter in line is β_m. Before proceeding we have to update the value of the residual so that, $\tilde{\mathbf{r}}_m = \mathbf{y} - \sum_{k\neq m}\tilde{\beta}_k\mathbf{x}_k$, is the current residual. $\hat{\beta}_m^{Lasso}$ can be obtained similarly and the process can be repeated for each parameter (updating the residual at every step) until we complete one entire cycle of optimization to obtain $\hat{\boldsymbol{\beta}}^{Lasso}$ as the Lasso estimator.

The algorithm will terminate after few cycles when convergence of the Lasso estimator is reached. Similar to the ridge regression set up we can obtain Lasso estimators for a grid of values of λ. We can choose the maximum value for the tuning parameter, λ_{max}, as the smallest possible value for which all coefficients will be estimated as zero. If the design matrix, $\mathbf{X}$, is not rank deficient then we can select the minimum value for the tuning parameter, λ_{min}, as zero. Otherwise choose the minimum value as $\lambda_{min} = \epsilon\lambda_{max}$ for a small value of ϵ, e.g., $\epsilon = 0.001$. We can express λ_{max} as:

$$\lambda_{max} = \max_{1\leq j\leq p}|(\mathbf{x}_j^\top \mathbf{x}_j)^{-1}\mathbf{x}_j^\top \mathbf{y}|.$$

λ_{max} can be viewed as absolute value of the least square estimate of that single predictor which will be the first to enter the model, therefore, having the largest absolute least square value. We can commence the optimization process starting with λ_{max} as the tuning parameter and obtain Lasso estimators for few values between λ_{max} and λ_{min}. To speed up the computation we can use Lasso estimator from a previous λ value as the initial estimator for a next λ value in line. For practical purpose we can select Lasso estimator given by an optimal lambda value using the BIC criteria:

$$\text{BIC}(\lambda_k) = \log\frac{\|\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_k^{Lasso}\|_2^2}{n} + \frac{log(n)df(\lambda_k)}{n}, \qquad \lambda_{min} \leq \lambda_k \leq \lambda_{max}, \tag{2.5}$$

where $\hat{\boldsymbol{\beta}}_k^{Lasso}$ is the Lasso estimator for the k^{th} value in the tuning parameter sequence and $df(\lambda_k)$ is the number of nonzero coefficients in $\hat{\boldsymbol{\beta}}_k^{Lasso}$. The optimal tuning parameter can be selected as the one corresponding to minimum value of BIC. The next section will introduce an improved version of Lasso called as "Adaptive Lasso".

2.4 Adaptive Lasso

Lasso is a popular method for variable selection and prediction but under some situations Lasso is not consistent as far as variable selection is concerned (Zou, 2006). Selection of the important variables (as per the true model) and good prediction accuracy / model estimation

are the two fundamental goals in penalized regression. Let's consider the regression set up as discusses earlier where the response and predictors are centered so that the intercept term is excluded from the model. Using the set up given by Zou (2006) let the set $\mathcal{A} = \{j : \beta_j^* \neq 0\}$ denote be the set of nonzero parameters in the true regression model and the cardinality of this set is $|\mathcal{A}| = p_0 < p$. Thus the true model is sparse since $p_0 < p$. The different methods of estimation such as ridge regression, Lasso, etc., are given a generic name: "fitting procedure", denoted by δ. A highly desirable property for a excellent fitting procedure is to behave as an *oracle*. An *oracle* procedure performs as if the true model is known in advance. As explained by Fan and Li (2001), if $\hat{\boldsymbol{\beta}}(\delta)$ is the estimator obtained by some *oracle* procedure, δ, then $\hat{\boldsymbol{\beta}}(\delta)$ should satisfy the following asymptotic properties:

1) Correct variable selection, $\{j : \hat{\beta}_j \neq 0\} = \mathcal{A}$ with high probability.
2) Optimal rate of estimation, $\sqrt{n}(\hat{\boldsymbol{\beta}}(\delta)_{\mathcal{A}} - \boldsymbol{\beta}_{\mathcal{A}}^*) \xrightarrow{D} N(0, \Sigma^*)$, where Σ^* is the covariance matrix for the true model.

In addition to the above two properties, an additional desirable property for *oracle* procedures is continuous shrinkage. Previous studies have shown that the Lasso procedure fails to behave as an *oracle* in certain situations. Meinshausen and Bühlmann (2006) pointed at a conflict in the Lasso procedure where the optimal λ chosen for Lasso to achieve good prediction also leads to inconsistent variable selection, therefore, allowing for many redundant variables in the estimated model. They derived an irrepresentable condition under which the Lasso is consistent in variable selection. Zou (2006) illustrated situations when Lasso can be inconsistent for variable selection via solid examples and then proposed a modified version of Lasso called "adaptive Lasso".

Unlike the Lasso where each coefficient in the l_1 penalty term is given equal weight, adaptive Lasso uses data driven adaptive weights so that the redundant coefficients are penalized more than the ones important for variable selection. The objective function for

adaptive Lasso is given as:

$$L(\boldsymbol{\beta};\lambda) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\sum_{i=1}^{p} w_i|\beta_i|,$$

where $\mathbf{w} = (w_1, ..., w_p)^\top$ is the the vector of weights chosen based on some root-n-consistent estimator of the true model, for example $\hat{\boldsymbol{\beta}}^{LS}$ (or $\hat{\boldsymbol{\beta}}^{ridge}$ if $p > n$). The weights can then be defined as:

$$w_i = |\hat{\beta}_i^{LS}|^{-\gamma}, \qquad \gamma \geq 0.$$

To find the optimizer for the adaptive Lasso problem, $\hat{\boldsymbol{\beta}}^{adLasso}$, let us define a $p \times p$ diagonal matrix, $\mathbf{W}$ whose diagonal elements are given by the weight vector, $\mathbf{w}$. Using this diagonal matrix of the weights we can express the objective function for adaptive Lasso as:

$$L(\boldsymbol{\beta};\lambda) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\mathbf{W}^{-1}\mathbf{W}\boldsymbol{\beta}\|_2^2 + \lambda\sum_{i=1}^{p} w_i|\beta_i|, \qquad \lambda \geq 0.$$

If we define $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{W}^{-1}$ and $\tilde{\boldsymbol{\beta}} = \mathbf{W}\boldsymbol{\beta}$ so that $\tilde{\beta}_i = w_i\beta_i$ then,

$$\begin{aligned} L(\boldsymbol{\beta};\lambda) &= \frac{1}{2n}\|\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}}\|_2^2 + \lambda\sum_{i=1}^{p}|\tilde{\beta}_i| \\ &= \frac{1}{2n}\|\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}}\|_2^2 + \lambda\|\tilde{\boldsymbol{\beta}}\|_1. \end{aligned} \tag{2.6}$$

The above objective function is similar to the one for Lasso (equation 2.3) except that $\mathbf{X}$ and $\boldsymbol{\beta}$ are now defined as $\tilde{\mathbf{X}}$ and $\tilde{\boldsymbol{\beta}}$ respectively. Keeping this distinction in mind, the optimizer for (2.6), $\hat{\tilde{\boldsymbol{\beta}}}^{adLasso}$, can be obtained similar to the Lasso optimizer (2.4). Finally, $\hat{\boldsymbol{\beta}}^{adLasso}$ can be obtained as:

$$\hat{\boldsymbol{\beta}}^{adLasso} = \mathbf{W}^{-1}\hat{\tilde{\boldsymbol{\beta}}}^{adLasso}.$$

The optimization for adaptive Lasso is convex and, therefore, we can use CDA to obtain a global optimizer just as we did for the Lasso problem. Similar to Lasso, we can obtain adaptive Lasso estimators over a grid of λ values. λ_{max} for adaptive Lasso can be derived similar to Lasso. Since the maximum value of the tuning parameter corresponds to the

smallest value of λ for which all coefficients are estimated to be zero, we can show λ_{max} for adaptive lasso as:

$$\lambda_{max} = \max_{1 \leq j \leq p} |\frac{\mathbf{x}_j^\top \mathbf{y}}{n w_j}|.$$

The computational algorithm for Lasso and adaptive Lasso is same. In fact, we can develop the algorithm for adaptive Lasso and just set $\gamma = 0$ for Lasso. This is due to fact that when $\gamma = 0$, the diagonal weight matrix, $\mathbf{W}$, is simply an identity matrix and we go back to Lasso framework. Zou (2006) suggested using $\gamma = 2$ for practical use and also proved that the adaptive Lasso is an *oracle* procedure.

An intuitive way to understand why adaptive weights helps to perform better variable selection and estimation compared to Lasso (thereby giving adaptive Lasso the properties of an *oracle* procedure) we can imagine the constraint region for adaptive Lasso when $p = 2$. Suppose the true model is such that $\beta_1 = 0$ and $\beta_2 \neq 0$. If the weights are defined as $w_1 = |\hat{\beta}_1^{LS}|^{-\gamma}$ and $w_2 = |\hat{\beta}_2^{LS}|^{-\gamma}$ then we can see that $w_1 \to \infty$ and $w_2 \to 0$. The adaptive Lasso constraint area, $w_1|\beta_1| + w_2|\beta_2| \leq t$, where t has one to one correspondence with λ, will then be a distorted version of the constraint region for Lasso in Fig (1.3.1). The distortion will be such that the rotated square for Lasso is stretched in the horizontal direction and shrunk in the vertical direction. Therefore, we are promoting the chances of contour of the error function (red ellipse) to first touch the constraint area where β_1 will be estimated to be zero. Therefore, cleverly chosen weights for adaptive Lasso can achieve better variable selection as compared to Lasso.

Before we conclude this section it is noteworthy to mention few ingenious concepts under the framework of penalized regression: group Lasso and non-convex penalties such as MCP and SCAD. Yuan and Lin (2006) proposed the group Lasso method to address the issue of selecting grouped variables. The group Lasso method extends the basic concept of Lasso for group selection problems that naturally arise in several practical situations such as multiple factor ANOVA. The combination of l_1 norm (that encourages sparsity) and l_2 norm (that combines the grouped coefficients) gives group Lasso the ability to enforce sparsity

at the group level. Apart from convex penalties such as Lasso, non-convex penalties such as smoothly clipped absolute deviation (SCAD) (Fan and Li, 2001) and minimax concave penalty (MCP) (Zhang, 2010) are proposed with appealing theoretical properties. One challenge with non-convex penalties is efficient computation. See, e.g., Breheny and Huang (2011) proposed an algorithm using local linear or quadratic approximation and coordinate descent for solving the non-convex SCAD and MCP problems.

Chapter 3

Linear Constrained Lasso

3.1 Lasso with Equality Constraints

We reach the next important step in our journey where we deal with situations which demand the Lasso/adaptive Lasso solution to satisfy certain linear constraints. We shall then apply the methodology developed here to solve the sparse and orthogonal singular value decomposition problem in the next chapter.

Consider the general constrained minimization problems of the following form,

$$\min_{\mathbf{u}}\{J(\mathbf{u}) : H(\mathbf{u}) = 0\}, \tag{3.1}$$

where $\mathbf{u} \in \mathbb{R}^p$, $J(\mathbf{u})$ and $H(\mathbf{u})$ are both convex functions and $\min_{\mathbf{u}\in\mathbb{R}^p} H(\mathbf{u}) = 0$. In order to solve (3.1), consider an unconstrained Lagrangian form,

$$\min_{\mathbf{u}} J(\mathbf{u}) + \mu H(\mathbf{u}), \tag{3.2}$$

where μ is the relaxation penalty parameter. For small μ, the penalty function may not accurately enforce the constraint. To satisfy the constraint, a common approach is to use continuation method and let $\mu \to \infty$ or be extremely large. However, a large μ may make the unconstrained problem hard to solve, especially when $J(\mathbf{u})$ is not differentiable. Besides, in many applications, μ must be increased with very small increments during the continuation procedure, which makes the optimization inefficient (Goldstein and Osher, 2009).

An alternative to the aforementioned conventional approach is Bregman iteration method. The fundamentals of Bregman iteration were laid out by Bregman (1967) who proposed to solve a sequence of unconstrained problems for finding common point of intersection of convex sets which could be used to approximate solutions of problems that arise in linear and convex programming. Bregman iterative algorithms have been widely used in compressed sensing particularly for solving constrained minimization problems such as basis pursuit. We provide a brief description of Bregman iteration method, and refer the interested reader to, e.g., Bregman (1967) and Goldstein and Osher (2009) for details. For a convex function $J(\mathbf{u})$, the Bregman distance at point $\mathbf{v}$ is defined as

$$B_J^S(\mathbf{u}, \mathbf{v}) = J(\mathbf{u}) - J(\mathbf{v}) - < S, \mathbf{u} - \mathbf{v} >,$$

where S is the subgradient of J at $\mathbf{v}$. It can be shown that $B_J^S(\mathbf{u}, \mathbf{v}) \geq 0$ and $B_J^S(\mathbf{u}, \mathbf{v}) \geq B_J^S(\mathbf{w}, \mathbf{v})$ for any $\mathbf{w}$ on the line segment between $\mathbf{u}$ and $\mathbf{v}$, i.e, $\mathbf{w} \in \{(1-t)\mathbf{u}+t\mathbf{v} : 0 \leq t \leq 1.\}$ (Goldstein and Osher, 2009). Bregman (1967) showed that (3.1) can be solved by iteratively solving

$$\begin{aligned}
\mathbf{u}^{(s+1)} =& \underset{\mathbf{u}}{\text{argmin}} B_J^S(\mathbf{u}, \mathbf{u}_s) + \mu H(\mathbf{u}) \\
=& \underset{\mathbf{u}}{\text{argmin}} J(\mathbf{u}) - < S^{(s)}, \mathbf{u} - \mathbf{u}^{(s)} > + \mu H(\mathbf{u}), \\
S^{(s+1)} =& S^{(s)} - \nabla H(\mathbf{u}^{(s+1)}).
\end{aligned}$$

Here for simplicity we have assumed that H is differentiable and used ∇H to denote its derivative. Bregman iterative method is guaranteed to converge and can be very fast in several applications. In particular, when the constraint is linear, the method can be further simplified (Goldstein and Osher, 2009). Consider

$$\min_{\mathbf{u}} \{J(\mathbf{u}) : \mathbf{A}\mathbf{u} = \mathbf{b}\}, \tag{3.3}$$

where $\mathbf{A} \in \mathbb{R}^{m \times p}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{u} \in \mathbb{R}^p$. The problem can be solved by iteratively solving

$$\begin{aligned}
\mathbf{u}^{(s+1)} =& \min_{\mathbf{u}} J(\mathbf{u}) + \frac{\mu}{2} \|\mathbf{A}\mathbf{u} - \mathbf{b}^{(s)}\|_2^2, \\
\mathbf{b}^{(s+1)} =& \mathbf{b}^{(s)} + \mathbf{b} - \mathbf{A}\mathbf{u}^{(s+1)}.
\end{aligned}$$

Inspired by the Bregman iterative algorithm we can solve the familiar Lasso problem under linear constraints. Consider the following constrained (adaptive) Lasso problem,

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2+\lambda\sum_{j=1}^{p} w_j|\beta_j| \qquad \text{subject to } \mathbf{A}\boldsymbol{\beta}=\mathbf{b}, \tag{3.4}$$

where $\mathbf{y}\in\mathbb{R}^n$, $\mathbf{X}\in\mathbb{R}^{n\times p}$, $\mathbf{A}=(\mathbf{a}_1,\ldots,\mathbf{a}_p)\in\mathbb{R}^{h\times p}$, $\mathbf{b}\in\mathbb{R}^{h\times 1}$, $\mathbf{w}=(w_1,...,w_p)^\top$ are some predetermined weights, and λ is a tuning parameter controlling the degree of l_1 penalization. Here we assume that the coefficient vector $\boldsymbol{\beta}$ satisfies h known linear constraints. Henceforth we denote the problem as ConLasso$(\boldsymbol{\beta};\mathbf{y},\mathbf{X},\mathbf{A},\mathbf{b},\lambda\mathbf{w})$. While the problem here is motivated by the orthogonality constrained optimization of the reduced rank regression set up (as elaborated in the introduction), we note that constrained Lasso arises in many problems including fused Lasso, generalized Lasso, monotone curve estimation, etc; for further discussion on this matter, we refer to an unpublished manuscript on the linear equality or inequality constrained Lasso by Gareth M. James, Courtney Paulson and Paat Rusmevichientong.

We construct an augmented Lagrangian function of the Lasso objective function (3.4) for incorporating the equality constraints,

$$f(\boldsymbol{\beta},\mathbf{c};\lambda,\mu)=\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2+\lambda\sum_{j=1}^{p} w_j|\beta_j|+\frac{\mu}{2}\|\mathbf{A}\boldsymbol{\beta}-\mathbf{b}-\frac{\mathbf{c}}{\mu}\|_2^2, \tag{3.5}$$

where the third term is added to penalize the violation of the linear constraints $\mathbf{A}\boldsymbol{\beta}=\mathbf{b}$, and $\mu\geq 0$ is a fixed Bregman parameter. For a given $\mathbf{c}$ vector, this function can be efficiently minimized by coordinate descent with respect to $\boldsymbol{\beta}$, as the problem is separable in each β_j, $j=1,\ldots,p$. By the KKT optimality condition, it can be shown that each one-dimension updating step is simply a scaled soft-thresholding operation, induced by the hybrid of the l_1 and l_2 regularization. Once $\boldsymbol{\beta}$ gets updated, we can then update $\mathbf{c}$ to further enforce the linear constraints. Therefore, the constrained Lasso problem can be solved by alternating minimization between $\mathbf{c}$ and $\boldsymbol{\beta}$ until convergence, which combines the ideas of Bregman iteration and coordinate descent. Note that in practice it is not necessary to completely

solve (3.5) for updating $\boldsymbol{\beta}$ as it is only one inner step of the proposed iterative algorithm. We hence propose a Bregman coordinate Descent algorithm (BCDA) as follows.

Bregman Coordinate Descent Algorithm (BCDA)

Initialization: $\boldsymbol{\beta}^{(0)} \in \mathbb{R}^p$, $\mathbf{c}^{(0)} = \mathbf{0}$, $\mu^{(0)} = 1$, and $\rho \geq 1$.

For cycle $s = 0, 1, 2, ...$

1. Solve

$$\boldsymbol{\beta}^{(s+1)} = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} w_j |\beta_j| + \frac{\mu^{(s)}}{2}\|\mathbf{A}\boldsymbol{\beta} - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}\|_2^2$$

by coordinate descent. For each $j = 1, ..., p$, the updating formula is given by

$$\hat{\beta}_j^{(s+1)} \longleftarrow \frac{\mathcal{S}\left((\mathbf{y} - \sum_{i \neq j} \beta_i \mathbf{x}_i)^\top \mathbf{x}_j + \mu^{(s)}\{(\frac{\mathbf{c}^{(s)}}{\mu^{(s)}} + \mathbf{b})^\top \mathbf{a}_j - (\sum_{i \neq j} \beta_i \mathbf{a}_i)^\top \mathbf{a}_j\}, \lambda w_j\right)}{\mathbf{x}_j^\top \mathbf{x}_j + \mu^{(s)} \mathbf{a}_j^\top \mathbf{a}_j},$$

where $\mathcal{S}(t, \lambda) = \text{sgn}(t)(|t| - \lambda)_+$ is the soft-thresholding operator.

2. $\mathbf{c}^{(s+1)} = \mathbf{c}^{(s)} - (\mathbf{A}\boldsymbol{\beta}^{(s+1)} - \mathbf{b})\mu^{(s)}$
3. $\mu^{(s+1)} = \mu^{(s)}\rho$

Repeat until convergence.

Note that with every cycle the parameter μ may be incremented since it is multiplied by $\rho > 1$, which may improve the speed of convergence in practice. However, this is not essential as we can simply set $\rho = 1$. To understand the updating formula for each β_j we have to solve the optimization problem (3.5) using coordinate descent algorithm such that all parameters are fixed at their initial estimators except the j^{th} parameter. Using the initialization: $\boldsymbol{\beta}^{(0)} \in \mathbb{R}^p$, $\mathbf{c}^{(0)} = \mathbf{0}$, and $\mu^{(0)} = 1$, the Bregman estimator for the $s + 1^{th}$ cycle,

$\hat{\beta}_j^{s+1}$, for j^{th} parameter is the minimizer of the following objective function:

$$
\begin{aligned}
f_j(\beta_j, \mathbf{c}; \lambda, \mu) &= \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} w_j|\beta_j| + \frac{\mu^{(s)}}{2}\|\mathbf{A}\boldsymbol{\beta} - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}\|_2^2 \\
&= \frac{1}{2}\|\mathbf{y} - \sum_{k\neq j} \mathbf{x}_k\beta_k - \mathbf{x}_j\beta_j\|_2^2 + \lambda w_j|\beta_j| + \frac{\mu^{(s)}}{2}\|\mathbf{a}_j\beta_j + (\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}})\|_2^2.
\end{aligned}
$$

Denoting $\mathbf{r}_j = \mathbf{y} - \sum_{k\neq j} \mathbf{x}_k\beta_k$ we further simply:

$$
\begin{aligned}
f_j(\beta_j, \mathbf{c}; \lambda, \mu) &= \frac{1}{2}\|\mathbf{r}_j - \mathbf{x}_j\beta_j\|_2^2 + \frac{\mu^{(s)}}{2}\mathbf{a}_j^\top\mathbf{a}_j\beta_j^2 + \mu^{(s)}\beta_j\mathbf{a}_j^\top(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}) + \lambda w_j|\beta_j| \\
&= \frac{1}{2}(\mathbf{r}_j^\top\mathbf{r}_j - 2\beta_j\mathbf{r}_j^\top\mathbf{x}_j + \beta_j^2\mathbf{x}_j^\top\mathbf{x}_j) + \frac{\mu^{(s)}}{2}\mathbf{a}_j^\top\mathbf{a}_j\beta_j^2 + \mu^{(s)}\beta_j\mathbf{a}_j^\top(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}) \\
&\quad + \lambda w_j|\beta_j| + \text{const.}
\end{aligned}
$$

Since multiplying by a constant does not affect the estimated solution we multiply both sides by $1/\mathbf{x}_j^\top\mathbf{x}_j$ and simply express $f_j(\beta_j, \mathbf{c}; \lambda, \mu)$ as:

$$
\begin{aligned}
f_j(\beta_j, \mathbf{c}; \lambda, \mu) &= \frac{1}{2}(\beta_j^2 - 2z_j\beta_j) + \frac{\mu^{(s)}\mathbf{a}_j^\top\mathbf{a}_j}{2\mathbf{x}_j^\top\mathbf{x}_j}\beta_j^2 + \frac{\mu^{(s)}\beta_j\mathbf{a}_j^\top}{\mathbf{x}_j^\top\mathbf{x}_j}(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}) \\
&\quad + \lambda w_j|\beta_j| + \text{const}, \quad (\text{where } z_j = \mathbf{r}_j^\top\mathbf{x}_j/\mathbf{x}_j^\top\mathbf{x}_j) \\
&= \frac{1}{2}(\beta_j^2 - 2z_j\beta_j + zj^2 - z_j^2) + \frac{\mu^{(s)}\mathbf{a}_j^\top\mathbf{a}_j}{2\mathbf{x}_j^\top\mathbf{x}_j}\beta_j^2 + \frac{\mu^{(s)}\beta_j\mathbf{a}_j^\top}{\mathbf{x}_j^\top\mathbf{x}_j}(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}) \\
&\quad + \lambda w_j|\beta_j| + \text{const} \\
&= \frac{1}{2}(\beta_j - z_j)^2 + \frac{\mu^{(s)}\mathbf{a}_j^\top\mathbf{a}_j}{2\mathbf{x}_j^\top\mathbf{x}_j}\beta_j^2 + \frac{\mu^{(s)}\beta_j\mathbf{a}_j^\top}{\mathbf{x}_j^\top\mathbf{x}_j}(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}}) \\
&\quad + \lambda w_j|\beta_j| + \text{const}.
\end{aligned}
$$

Denote $\lambda^* = \lambda w_j/\mathbf{x}_j^\top\mathbf{x}_j$, $\mu^* = \mu^{(s)}\mathbf{a}_j^\top\mathbf{a}_j/2\mathbf{x}_j^\top\mathbf{x}_j$, and $\nu = \frac{\mu^{(s)}\mathbf{a}_j^\top}{\mathbf{x}_j^\top\mathbf{x}_j}(\sum_{k\neq j} \mathbf{a}_k\beta_k - \mathbf{b} - \frac{\mathbf{c}^{(s)}}{\mu^{(s)}})$ so

that:

$$
\begin{aligned}
f_j(\beta_j, \mathbf{c}; \lambda, \mu) &= \frac{1}{2}(\beta_j - z_j)^2 + \mu^* \beta_j^2 + \lambda^* |\beta_j| + \nu \beta_j + \text{const} \\
&= \frac{1}{2}\beta_j^2 - \beta_j z_j + \frac{z_j^2}{2} + \mu^* \beta_j^2 + \lambda^* |\beta_j| + \nu \beta_j + \text{const} \\
&= (\frac{1}{2} + \mu^*)\beta_j^2 - (z_j - \nu)\beta_j + \lambda^* |\beta_j| + \text{const} \\
&= (\frac{1}{2} + \mu^*)[\beta_j^2 - 2(\frac{z_j - \nu}{1 + 2\mu^*})\beta_j + (\frac{z_j - \nu}{1 + 2\mu^*})^2 - (\frac{z_j - \nu}{1 + 2\mu^*})^2] + \lambda^* |\beta_j| + \text{const} \\
&= (\frac{1}{2} + \mu^*)[\beta_j - (\frac{z_j - \nu}{1 + 2\mu^*})]^2 + \lambda^* |\beta_j| + \text{const}.
\end{aligned}
$$

Dividing both sides by the constant $(1 + 2\mu^*)$ we simply express $f_j(\beta_j, \mathbf{c}; \lambda, \mu)$ as:

$$
f_j(\beta_j, \mathbf{c}; \lambda, \mu) = \frac{1}{2}(\beta_j - \frac{z_j - \nu}{1 + 2\mu^*})^2 + \frac{\lambda^*}{1 + 2\mu^*}|\beta_j| = \mathcal{S}[\frac{z_j - \nu}{1 + 2\mu^*}; \frac{\lambda^*}{1 + 2\mu^*}],
$$

where $\mathcal{S}(t, \lambda) = \text{sgn}(t)(|t| - \lambda)_+$ is the soft-thresholding operator. Expressing μ^*, λ^*, and ν in their orginal form it can be easily shown that:

$$
\hat{\beta}_j^{(s+1)} = \frac{\mathcal{S}\left((\mathbf{y} - \sum_{i \neq j} \beta_i \mathbf{x}_i)^\top \mathbf{x}_j + \mu^{(s)}\{(\frac{\mathbf{c}^{(s)}}{\mu^{(s)}} + \mathbf{b})^\top \mathbf{a}_j - (\sum_{i \neq j} \beta_i \mathbf{a}_i)^\top \mathbf{a}_j\}, \lambda w_j\right)}{\mathbf{x}_j^\top \mathbf{x}_j + \mu^{(s)} \mathbf{a}_j^\top \mathbf{a}_j}
$$

3.2 Simulation

We compare the prediction error, estimation error, sparsity, and conformance to linear constraint for five methods: constrained adaptive Lasso (linear constraint and adaptive weights), adaptive Lasso (adaptive weights only), constrained Lasso (linear constraint but no adaptive weights), Lasso (no linear constraint or adaptive weights), and the ordinary least squares. The Bregman parameter which governs the linear constraint is set as $\mu = \rho = 1$. The sparsity parameter, following the suggestion by Zou (2006), is set as $\gamma = 2$. The simulation is done for two models: 1) low dimensional model where $n = 50$ and $p = 20$, and 2) high dimensional model where $n = 50$ and $p = 100$. For both models the covariate matrix $\mathbf{X}$ is constructed by generating its n rows as i.i.d. samples from $\text{MVN}(\mathbf{0}, \Gamma)$, where

$\Gamma = (\Gamma_{ij})_{p\times p}$ and $\Gamma_{ij} = 0.5^{|i-j|}$. The true signal, $\boldsymbol{\beta}$, is constructed as follows:

$$\boldsymbol{\beta} = [\text{unif}(\mathcal{A}, 5), \text{rep}(0, 10)]^\top \qquad \text{(Model 1)}$$
$$\boldsymbol{\beta} = [\text{unif}(\mathcal{A}, 5), \text{rep}(0, 90)]^\top \qquad \text{(Model 2)}.$$

The notation $\text{unif}(\mathcal{A}, b)$ denotes a vector of length b whose entries are i.i.d. samples from the uniform distribution on the set of real values $\mathcal{A}$; we use $\mathcal{A} = [-1, -2] \cup [1, 2]$. The notation $\text{rep}(a, b)$ denotes a vector of length b, whose entries are all equal to a. The linear constraints are such that the true signal, $\boldsymbol{\beta}$, is orthogonal to the row vectors of matrix $\mathbf{A}_{2\times p}$. Therefore, the vector $\mathbf{b}$ is a zero vector. Using the notation $\mathbf{A}[i,]$ for the i^{th} row, the matrix $\mathbf{A}$ is constructed as:

$$\mathbf{A}[1,] = [\text{rep(0,10)}, \text{ rep(1,5)}, \text{ rep(0,5)}]^\top \qquad \text{(Model 1)}$$
$$\mathbf{A}[2,] = [\text{rep(0,10)}, \text{ rep(0,5)}, \text{ rep(1,5)}]^\top \qquad \text{(Model 2)}.$$

Finally the vector $\mathbf{y}$ is constructed as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e}$, where the entries of $\mathbf{e}$ are i.i.d. samples from $N(0, \sigma^2)$. We define the signal to noise ratio (SNR) as $\|\boldsymbol{\beta}^\top \Gamma \boldsymbol{\beta}\|_2/\sigma$. Therefore, the standard deviation σ can be chosen to fix SNR at a certain level. The experiment was replicated 100 times for each SNR = 5, 10, and 15 to cover decent range of signal strengths relative to the noise level.

Tables 3.1 and 3.2 summarize the simulation results for low and high dimensional models respectively. The averaged false positive rates (FPR) and false negative rates (FNR) are reported. The estimation error (Er-Est) and prediction error (Er-Pred) for a single run is defined as:

$$\text{Er-Est} = 100\frac{\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2}{n},$$
$$\text{Er-Pred} = 100\frac{\|\mathbf{X}\hat{\boldsymbol{\beta}} - \mathbf{X}\boldsymbol{\beta}\|_2^2}{n},$$

where $\hat{\boldsymbol{\beta}}$ is the estimated solution for a particular method and SNR setting. The conformance to orthogonality (linear constraint) for a single run is measured as:

$$\text{constraint} = \|\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{b}\|_2^2.$$

We report the average and standard deviation from all 100 runs for FPR, FNR, Er-Est, Er-Pred, conformance to orthogonality for each method and SNR setting within both low and high dimensional set up. Each simulation run was conducted for 100 lambda values and the optimal solution was chosen using BIC (equation 2.5). The maximum lambda for Lasso (or constrained Lasso) and adaptive Lasso (or constrained adaptive Lasso) was computed as explained in the Lasso and adaptive Lasso section respectively.

Table 3.1: *Constrained Adaptive Lasso: Low Dimensional Model (n=50, p=20)*

SNR	FPR	FNR	Er-Est	Er-Pred	Constraint
			Constrained Adaptive Lasso		
5	5.80%	0.00%	0.87 (0.97)	20.38 (12.67)	0.00
10	1.48%	0.00%	0.21 (0.18)	4.74 (2.77)	0.00
15	0.56%	0.00%	0.09 (0.07)	2.30 (0.98)	0.00
			Adaptive Lasso		
5	9.43%	0.00%	1.12 (0.71)	24.61 (13.67)	0.13 (0.11)
10	4.77%	0.00%	0.31 (0.22)	6.51 (3.85)	0.00
15	2.32%	0.00%	0.13 (0.11)	2.24 (0.91)	0.00
			Constrained Lasso		
5	28.23%	0.12%	1.42 (0.94)	30.21 (14.79)	0.00
10	11.94%	0.00%	0.43 (0.22)	8.21 (3.69)	0.00
15	4.81%	0.00%	0.19 (0.12)	3.43 (2.02)	0.00
			Lasso		
5	35.13%	0.11%	1.62 (1.03)	31.71 (16.01)	10.82 (12.58)
10	15.11%	0.00%	0.37 (0.31)	8.62 (3.79)	2.33 (2.64)
15	6.42%	0.00%	0.19 (0.12)	3.84 (2.02)	0.68 (0.93)
			Least Squares		
5	100.00%	0.00%	1.94 (1.01)	36.32 (15.84)	0.23 (0.18)
10	100.00%	0.00%	0.52 (2.65)	8.83 (54.93)	0.12 (0.11)
15	100.00%	0.00%	0.19 (0.12)	3.82 (1.84)	0.00

From tables 3.1 and 3.2 we can infer that the constrained adaptive Lasso method outperforms rest of the methods. The constrained adaptive Lasso method is able to achieve relatively low FPR, FNR, Er-Est, Er-Pred, and constraint values due to its ability to cleverly use data driven weights that promote sparsity and enforce orthogonality, courtesy Bregman iterative method. Therefore, it can be greatly useful for high dimensional models, particu-

larly when the SNR is on the lower side, as the orthogonality condition restricts and guides the search for optimal sparse solution. Adaptive Lasso, the unconstrained counterpart of

Table 3.2: *Constrained Adaptive Lasso: High Dimensional Model (n=50, p=100)*

SNR	FPR	FNR	Er-Est	Er-Pred	Constraint
			Constrained Adaptive Lasso		
5	16.83%	10.52%	10.31 (7.85)	75.81 (48.69)	0.00
10	13.62%	6.87%	6.42 (8.01)	33.82 (37.03)	0.00
15	10.91%	3.87%	3.53 (5.75)	19.42 (27.89)	0.00
			Adaptive Lasso		
5	18.79%	10.94%	11.92 (8.02)	78.51 (50.04)	0.21 (0.28)
10	12.82%	6.52%	6.52 (6.87)	31.68 (36.81)	0.09 (0.27)
15	10.66%	4.91%	4.61 (6.73)	21.52 (30.82)	0.07 (0.23)
			Constrained Lasso		
5	29.53%	1.21%	8.93 (4.85)	87.01 (34.03)	0.00
10	16.42%	0.11%	2.72 (2.84)	21.22 (7.85)	0.00
15	8.91%	0.18%	1.42 (2.01)	9.42 (3.91)	0.00
			Lasso		
5	28.41%	3.11%	10.48 (5.82)	93.11 (37.01)	0.13 (0.16)
10	18.01%	0.59%	2.91 (3.02)	19.68 (6.79)	0.11 (0.11)
15	10.64%	0.00%	1.31 (1.03)	8.91 (3.79)	0.00
			Least Squares		
5	100.00%	0.00%	25.61 (5.03)	91.77 (35.01)	0.51 (0.51)
10	100.00%	0.00%	23.63 (5.02)	22.81 (11.02)	0.56 (0.57)
15	100.00%	0.00%	22.56 (5.01)	10.53 (3.83)	0.51 (0.59)

constrained adaptive Lasso, slightly underperformed as far as sparsity, estimation and prediction is concerned. However, with respect to linear constraints, constrained adaptive Lasso is far better than adaptive Lasso. If we do not use adaptive weights, as in the case of Lasso and constrained Lasso, we observe relatively high FPR which reflects the over-selection property of the l_1 regularization. However, the over-selection is not so bad when the SNR is on the higher side. As expected the constrained Lasso conforms to the orthogonality condition much better than the Lasso. Clearly the ordinary least squares solution does not perform variable selection (or satisfaction of the linear constraint) as indicated by the high FPR and high estimation / prediction error.

Chapter 4

Sparse and Orthogonal SVD

4.1 Introduction

Given n observations of the response variable $\mathbf{y}_i \in \mathbb{R}^q$ and predictor variable $\mathbf{x}_i \in \mathbb{R}^p$, we consider the multivariate regression model:

$$\mathbf{Y} = \mathbf{XC} + \mathbf{E}, \tag{4.1}$$

where $\mathbf{Y} = (\mathbf{y}_1, ..., \mathbf{y}_n)^\top$, $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)^\top$, $\mathbf{C}$ is a $p \times q$ coefficient matrix, and $\mathbf{E} = (\mathbf{e}_1, ..., \mathbf{e}_n)^\top$ is a random $n \times q$ matrix; the error vectors are assumed to be independently and identically distributed (i.i.d.) with mean vector $\mathrm{E}(\mathbf{e}_i) = \mathbf{0}$ and covariance matrix $\mathrm{Cov}(\mathbf{e}_i) = \Sigma$, a $q \times q$ positive-definite matrix. We assume the variables are centered so that there is no intercept term. It is assumed that $\mathbf{C}$ may admit certain low-dimensional matrix structures, the exploitation of which may mitigate the curse of dimensionality, facilitate model interpretation and improve predictive accuracy. Many desirable low-dimensional properties of $\mathbf{C}$ can be revealed via examining its singular value decomposition (SVD), which can be written as:

$$\mathbf{C} = \mathbf{UDV}^\top = \sum_{k=1}^{r} d_k \mathbf{u}_k \mathbf{v}_k^\top = \sum_{k=1}^{r} \mathbf{C}_k, \tag{4.2}$$

where $r = \min(p, q)$, $\mathbf{U} = (\mathbf{u}_1, ..., \mathbf{u}_r)$ consists of orthonormal left singular vectors, $\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_r)$ consists of orthonormal right singular vectors, $\mathbf{D} = \mathrm{diag}(\{d_k, k = 1, \ldots, r\})$ is a

$r \times r$ diagonal matrix with nonnegative singular values $d_1 \geq ... \geq d_{r^*} > 0$ and $d_{r^*+1} = ... = d_r = 0$ on its diagonal, and $\mathbf{C}_k = d_k \mathbf{u}_k \mathbf{v}_k^\top$.

This SVD representation could be used to parsimoniously reveal some complex dependence structures between $\mathbf{Y}$ and $\mathbf{X}$. Firstly, it shows that $\mathbf{C}$ is composed of r^* orthogonal unit-rank layers with decreasing singular values, i.e. there are r^* distinct additive channels of decreasing importance relating the responses to the predictors. Secondly, for each layer k, the elements in $\mathbf{u}_k$ can be viewed as the predictor effects and the elements in $\mathbf{v}_k$ the response effects. In order to conduct variable selection and facilitate interpretation, it is then desirable that the left and right singular vectors be *sparse*, i.e. each pathway relating the responses to the predictors may only involve a few responses and predictors. In addition to sparsity, it is also desirable to enforce orthogonality among the left/right singular vectors which holds true for the coefficient matrix of the underlying true model. Imposing orthogonality, albeit difficult, is critical in restricting and guiding the search of the optimal sparsity pattern.

We propose two methods that strive to achieve the dual goal of sparse and orthogonality: 1) *Sparse Orthogonal SVD via Sequential Extraction Algorithm* (SOSVD via SEA) which uses only one regularization parameter for each pair of singular vectors, and 2) *Sparse Orthogonal SVD via Block Extraction Algorithm* (SOSVD via BEA) which allows for two regularization parameters so that different levels of sparsity can be imposed on the left and right singular vectors. Both methods use the Bregman iterative algorithm to impose linear constraints needed to promote orthogonality.

4.2 SOSVD via SEA

Suppose that the rank of $\mathbf{C}$ has been correctly identified, i.e., $\text{rank}(\mathbf{C}) = r^*$ so that $\sum_{k=1}^{r^*} d_k \mathbf{u}_k \mathbf{v}_k^\top$ represent the the SVD of $\mathbf{C}$: sum of r^* unit rank matrices with decreasing singular values. We can estimate $\mathbf{C}$ in equation (4.1) by minimizing the following objective

function with respect to the triplets $(d_k, \mathbf{u}_k, \mathbf{v}_k)$ for $k = 1, ..., r^*$:

$$\frac{1}{2}\|\mathbf{Y} - \mathbf{X}\sum_{k=1}^{r^*} d_k \mathbf{u}_k \mathbf{v}_k^\top\|_F^2 + \sum_{k=1}^{r^*} \lambda_k \sum_{i=1}^{p}\sum_{j=1}^{q} w_{ijk}|d_k u_{ik} v_{jk}|,$$
$$\text{subject to } \mathbf{U}^\top\mathbf{U} = \mathbf{I}, \mathbf{V}^\top\mathbf{V} = \mathbf{I}. \tag{4.3}$$

Note that $\|\mathbf{u}_k\| = \|\mathbf{v}_k\| = 1$ where $\|.\|$ denotes the l_2-norm. Further, $w_{ijk} = w_k^{(d)} w_{ik}^{(u)} w_{jk}^{(v)}$ are the data driven weights to promote sparsity (similar to adaptive lasso problem) and low dimension recovery. $\lambda_k s$ are the regularization parameter controlling the degree of penalization of the distinct layers. Similar to adaptive lasso, the second term in equation (4.3) (also called penalty term) is proportional to the weighted l_1-norm of an SVD layer $d_k \mathbf{u}_k \mathbf{v}_k^\top$. Note that the penalty term has a multiplicative form so that the objective function can also be expressed as:

$$\frac{1}{2}\|\mathbf{Y} - \mathbf{X}\sum_{k=1}^{r^*} d_k \mathbf{u}_k \mathbf{v}_k^\top\|_F^2 + \sum_{k=1}^{r^*} \{\lambda_k (w_k^{(d)} d_k)(\sum_{i=1}^{p} w_{ik}^{(u)}|u_{ik}|)(\sum_{j=1}^{q} w_{jk}^{(v)}|v_{jk}|)\},$$
$$\text{subject to } \mathbf{U}^\top\mathbf{U} = \mathbf{I}, \mathbf{V}^\top\mathbf{V} = \mathbf{I}.$$

The above expression shows that only one regularization parameter is needed for each pair of singular vectors. A simple approach to achieve the dual goal of estimating sparse and orthogonal r^* layers of $\mathbf{C}$, as alluded by Chen *et al.* (2012), is by sequentially performing sparse unit rank regression for each of the r^* layers such that every k^{th} layer is orthogonal to the previous $k-1$ layers; implying the orthogonality (and sparsity) of the left/right singular vectors. At each step of the sequential process, the data matrix $\mathbf{Y}$ is replaced by the residual matrix that is obtained by subtracting previously estimated layer of the original matrix. Therefore, estimation of the k^{th} layer boils down to sparse unit rank regression, case when $r^* = 1$, while imposing orthogonality with respect to previous $k-1$ layers.

The sequential approach can be used as a simple and practical algorithm for obtaining sparse estimate of $\mathbf{C}$ in 4.1. For estimating the k^{th} layer, the problem can be expressed as

minimizing the following objective function with respect to k^{th} triplet $(d_k, \mathbf{u}_k, \mathbf{v}_k)$:

$$\frac{1}{2}\|\mathbf{Y}_k - \mathbf{X}d_k\mathbf{u}_k\mathbf{v}_k^\top\|_F^2 + \lambda_k \sum_{i=1}^{p}\sum_{j=1}^{q} w_{ijk}|d_k u_{ik} v_{jk}|, \text{ subject to } \mathbf{U}_{1:k-1}^\top \mathbf{u}_k = \mathbf{0}, \mathbf{V}_{1:k-1}^\top \mathbf{v}_k = \mathbf{0},$$
$$\mathbf{u}_k^\top \mathbf{u}_k = 1,\ \mathbf{v}_k^\top \mathbf{v}_k = 1. \tag{4.4}$$

Here $\mathbf{Y}_k = \mathbf{Y} - \sum_{r=1}^{k-1} \mathbf{X}\hat{\mathbf{C}}_r$ denote the k^{th} residual matrix obtained by subtracting the previous $k-1$ estimated layers from the original matrix $\mathbf{Y}$. The columns of $\mathbf{U}_{1:k-1}$ and $\mathbf{V}_{1:k-1}$ denote the first $k-1$ left and right singular vectors respectively. As explained earlier $\|\mathbf{u}_k\| = \|\mathbf{v}_k\| = 1$ and $w_{ijk} = w_k^{(d)} w_{ik}^{(u)} w_{jk}^{(v)}$ are the data driven weights to promote sparsity. To construct the weights we start with some initial consistent estimator of $\mathbf{C}$, rank r^* reduced rank least squares estimator of $\mathbf{C}$, for example. Let $\tilde{d}_k$, $\tilde{\mathbf{u}}_k$, and $\tilde{\mathbf{v}}_k$ denote the constituents that build up the k^{th} layer of the initial estimator. As suggested by Zou (2006), the weights can be constructed as:

$$\left.\begin{aligned} w_k^{(d)} &= |\tilde{d}|^{-\gamma}, \\ \mathbf{w}_k^{(u)} &= (w_{1k}^{(u)}, ..., w_{pk}^{(u)})^\top = |\tilde{\mathbf{u}}_k|^{-\gamma}, \\ \mathbf{w}_k^{(v)} &= (w_{1k}^{(v)}, ..., w_{qk}^{(v)})^\top = |\tilde{\mathbf{v}}_k|^{-\gamma}, \end{aligned}\right\} \tag{4.5}$$

where γ is a predetermined parameter. As suggested by Zou (2006), we can choose $\gamma = 2$. The notation $|.|^{-\gamma}$ is defined element-wise for the vector under consideration. Inspired by Chen *et al.* (2012) we propose to solve the objective function (4.4) by a block co-ordinate descent algorithm in which two overlapping blocks of parameter, $(d_k, \mathbf{u}_k)$ and $(d_k, \mathbf{v}_k)$, are alternatively updated until convergence, with either $\mathbf{v}_k$ or $\mathbf{u}_k$ held fixed.

For fixed $\mathbf{u}_k$, the minimization of (4.4) with respect to the block $(d_k, \mathbf{v}_k)$ becomes minimization with respect to $\check{\mathbf{v}}_k = \text{diag}(d_k \mathbf{w}_k^{(v)})\mathbf{v}_k$ of the following objective function (referred to as $\mathbf{V}$-step):

$$\frac{1}{2}\|\mathbf{y}_k - \mathbf{X}_k^{(v)}\check{\mathbf{v}}_k\|_2^2 + \lambda_k^{(v)} \sum_{j=1}^{q} |\check{\mathbf{v}}_{jk}|, \quad \text{subject to } \mathbf{V}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(v)})^{-1}\check{\mathbf{v}}_k = \mathbf{0}, \tag{4.6}$$

where the expression $\text{diag}(\mathbf{m})$ denotes a diagonal matrix with entries of the vector $\mathbf{m}$ on its diagonal, $\mathbf{y}_k = \text{vec}(\mathbf{Y}_k)$, $\mathbf{X}_k^{(v)} = \text{diag}(\mathbf{w}_k^v)^{-1} \otimes (\mathbf{X}\mathbf{u}_k)$, and $\lambda_k^{(v)} = \lambda_k w_k^{(d)} (\sum_{i=1}^{p} w_{ik}^{(u)} |u_{ik}|)$.

Note that the definition of $\check{\mathbf{v}}_k$ can be used to show that the condition $\mathbf{V}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(v)})^{-1}\check{\mathbf{v}}_k = \mathbf{0}$ implies $\mathbf{V}_{1:k-1}^\top \mathbf{v}_k = \mathbf{0}$ which ensures that every k^{th} estimated left singular is orthogonal to the previously estimated $(k-1)$ left singular vectors. Solving the $\mathbf{V}$-step, given by equation (4.6), is necessarily a lasso problem with respect to $\check{\mathbf{v}}_k$ in addition to the orthogonality constraint. Therefore, we invoke the constrained adaptive lasso algorithm, referred to as ConLasso$(\boldsymbol{\beta} = \check{\mathbf{v}}_k; \mathbf{y} = \mathbf{y}_k, \mathbf{X} = \mathbf{X}_k^{(v)}, \mathbf{A} = \mathbf{V}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(v)})^{-1}, \mathbf{b} = \mathbf{0}, \lambda = \text{rep}(\lambda_k^{(v)}, q))$.

Similarly for fixed $\mathbf{v}_k$, the minimization of (4.4) with respect to the block $(d_k, \mathbf{u}_k)$ becomes minimization with respect to $\check{\mathbf{u}}_k = \text{diag}(d_k \mathbf{w}_k^{(u)})\mathbf{u}_k$ of the following objective function (referred to as $\mathbf{U}$-step):

$$\frac{1}{2}\|\mathbf{y}_k - \mathbf{X}_k^{(u)}\check{\mathbf{u}}_k\|_2^2 + \lambda_k^{(u)} \sum_{i=1}^{p} |\check{\mathbf{u}}_{ik}|, \quad \text{subject to } \mathbf{U}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(u)})^{-1}\check{\mathbf{u}}_k = \mathbf{0}, \tag{4.7}$$

where $\mathbf{X}_k^{(u)} = \mathbf{v}_k \otimes \mathbf{X}\text{diag}(\mathbf{w}_k^u)^{-1}$, and $\lambda_k^{(u)} = \lambda_k w_k^{(d)} (\sum_{j=1}^{q} w_{jk}^{(v)} |v_{jk}|)$. Again, the definition of $\check{\mathbf{u}}_k$ can be used to show that the condition $\mathbf{U}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(u)})^{-1}\check{\mathbf{u}}_k = \mathbf{0}$ implies $\mathbf{U}_{1:k-1}^\top \mathbf{u}_k = \mathbf{0}$ which ensures that every k^{th} estimated right singular is orthogonal to the previously estimated $(k-1)$ right singular vectors. Thus, solving $\mathbf{U}$-step, given by equation (4.7), is again a lasso problem with respect to $\check{\mathbf{u}}_k$ in addition to the orthogonality constraint. Therefore, we invoke the constrained adaptive lasso algorithm, referred to as ConLasso$(\boldsymbol{\beta} = \check{\mathbf{u}}_k; \mathbf{y} = \mathbf{y}_k, \mathbf{X} = \mathbf{X}_k^{(u)}, \mathbf{A} = \mathbf{U}_{1:k-1}^\top \text{diag}(d_k \mathbf{w}_k^{(u)})^{-1}, \mathbf{b} = \mathbf{0}, \lambda = \text{rep}(\lambda_k^{(u)}, p))$.

Empowered by the constrained adaptive lasso algorithm to the solve the $\mathbf{U}$ and $\mathbf{V}$ step and owing to the multi-convex structure of the objective function (4.3) we propose the *sparse unit rank regression algorithm* for estimating the k^{th} layer of $\mathbf{C}$. We denoted the algorithm as SURR$(\hat{\mathbf{C}}_k; \mathbf{X}, \mathbf{Y}_k, \mathbf{A}_{k-1}^{(u)}, \mathbf{b}_{k-1}^{(u)}, \mathbf{A}_{k-1}^{(v)}, \mathbf{b}_{k-1}^{(v)}, \lambda_k)$. The matrix $\mathbf{A}_{k-1}^{(u)}$ is a $(k-1) \times p$ matrix whose rows are the previously estimated $k-1$ left singular vectors. Similarly, the matrix $\mathbf{A}_{k-1}^{(v)}$ is a $(k-1) \times q$ matrix whose rows are the previously estimated $k-1$ right singular vectors. The vectors $\mathbf{b}_{k-1}^{(u)}$ and $\mathbf{b}_{k-1}^{(v)}$ are zero vectors each of length $k-1$. For a fixed value of the tuning parameter, λ_k, the SURR algorithm is given below.

SURR for the k^{th} orthogonal layer

Initialization: Start with rank r^* reduced rank least squares estimator of $\mathbf{C}$ and let $\tilde{\mathbf{u}}_k$ denote the initial non-zero estimator of the k^{th} right singular vector.

1. **V**-step: Given $\mathbf{u}_k = \tilde{\mathbf{u}}_k$, minimize the objective function (4.6) to obtain $\check{\mathbf{v}}_k$. Therefore, $\hat{\mathbf{v}}_k = \text{diag}(\hat{d}_k \mathbf{w}_k^{(v)})^{-1} \check{\mathbf{v}}_k$ and $\hat{d}_k = \|\text{diag}(\mathbf{w}^{(v)})^{-1} \check{\mathbf{v}}_k\|_2$.

2. **U**-step: Given $\mathbf{v}_k = \check{\mathbf{v}}_k$, minimize objective function (4.7) to obtain $\check{\mathbf{u}}_k$. Therefore, $\hat{\mathbf{u}}_k = \text{diag}(\hat{d}_k \mathbf{w}_k^{(u)})^{-1} \check{\mathbf{u}}_k$ and $\hat{d}_k = \|\text{diag}(\mathbf{w}^{(u)})^{-1} \check{\mathbf{u}}_k\|_2$.

3. Repeat steps (1) and (2), until $\hat{\mathbf{C}}_k = \hat{d}_k \hat{\mathbf{u}}_k \hat{\mathbf{v}} \top_k$ converges implying $\frac{\|\hat{\mathbf{C}}_c - \hat{\mathbf{C}}_p\|_F}{\|\hat{\mathbf{C}}_p\|_F} < \epsilon$, where ϵ is the level of tolerance, $\epsilon = 10^{-6}$, for example. $\hat{\mathbf{C}}_c$ denotes the current fit and $\hat{\mathbf{C}}_p$ denotes the previous fit.

Similar to the lasso / adaptive lasso problem, we can estimate the solution path for any k^{th} layer of $\mathbf{C}$ for a grid of 100 λ_k values in a decreasing order and equally spaced on a log-scale, denoted as $[\lambda_k^{\max}, \lambda_k^{\min}]$. $\lambda_k^{\max}$ is the smallest value of the tuning parameter at which all coefficients are estimated as zero. $\lambda_k^{\min}$ can be chosen as a small value at which the model has excessive number of non-zero coefficients or model simulation becomes numerically unstable (Chen *et al.*, 2012). Also, owing to the continuous nature of the solution path, we can choose the initial estimate for any particular λ_k as the estimate obtained from the previous λ_k value in the sequence of the tuning parameter. This strategy is a key feature of the coordinate descent algorithm to speed up computation.

Finally, we propose the *SOSVD via SEA* algorithm for estimating all r^* layers of $\mathbf{C}$ as summarized below.

SOSVD via SEA

Initialization: Start with rank r^* reduced rank least squares estimator of $\mathbf{C}$ denoted as $\tilde{\mathbf{C}} = \sum_{k=1}^{r^*} \tilde{\mathbf{C}}_k$ and let $\tilde{\mathbf{C}}_k = \tilde{d}_k \tilde{\mathbf{u}}_k \tilde{\mathbf{v}}_k^\top$ denote its k^{th} layer.

1. **Layer 1**: Start with the initial estimator $\tilde{\mathbf{C}}_1$ and estimate the first layer $\hat{\mathbf{C}}_1$ by solving SURR$(\hat{\mathbf{C}}_1; \mathbf{X}, \mathbf{Y}, \mathbf{A}_0^{(u)} = \text{rep}(0,p)^\top, \mathbf{b}_0^{(u)} = 0, \mathbf{A}_0^{(v)} = \text{rep}(0,q)^\top, \mathbf{b}_0^{(v)} = 0, \lambda_1)$. Note that the first layer need not satisfy any orthogonality constraint.

   For $k = 2, 3, ..., r^*$

2. **Layer k**: Start with the initial estimator $\tilde{\mathbf{C}}_k$ and estimate the k^{th} layer $\hat{\mathbf{C}}_k$ by solving SURR$(\hat{\mathbf{C}}_k; \mathbf{X}, \mathbf{Y} = \mathbf{Y}_k, \mathbf{A}_{k-1}^{(u)} = (\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, ..., \hat{\mathbf{u}}_{k-1})^\top, \mathbf{b}_{k-1}^{(u)} = \text{rep}(0, k-1), \mathbf{A}_{k-1}^{(v)} = (\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, ..., \hat{\mathbf{v}}_{k-1})^\top, \mathbf{b}_{k-1}^{(v)} = \text{rep}(0, k-1), \lambda_k)$.

Note that if $\mathbf{X}$ is a identity matrix in equation (4.1) then we revert to unsupervised learning framework. The next section discusses SOSVD via BEA algorithm, which allows for two regularization parameters so that different levels of sparsity can be imposed on the left and right singular vectors.

4.3 SOSVD via BEA

Suppose the rank of $\mathbf{C}$ has been correctly identified, i.e., rank$(\mathbf{C}) = r^*$. To obtain both sparse and orthogonal left/right singular vectors of $\mathbf{C}$, we propose to estimate $\mathbf{C}$ by solving the following optimization problem with respect to the triplet $(\mathbf{U}, \mathbf{D}, \mathbf{V})$, with $\mathbf{U} \in \Re^{p \times r^*}$, $\mathbf{V} \in \Re^{q \times r^*}$ and $\mathbf{D}$ an $r^* \times r^*$ diagonal matrix,

$$\min_{(\mathbf{U},\mathbf{D},\mathbf{V})} \left\{ J(\mathbf{U}, \mathbf{D}, \mathbf{V}) \equiv \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\mathbf{U}\mathbf{D}\mathbf{V}^\top\|_F^2 + \lambda_1 \|\mathbf{W}^{(u)} \circ \mathbf{U}\mathbf{D}\|_1 + \lambda_2 \|\mathbf{W}^{(v)} \circ \mathbf{V}\mathbf{D}\|_1 \right\}$$
$$\text{subject to } \mathbf{U}^\top\mathbf{U} = \mathbf{I}, \mathbf{V}^\top\mathbf{V} = \mathbf{I} \tag{4.8}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $\|\cdot\|_1$ the l_1 norm, and $\circ$ the Hadamard product. Also, $\mathbf{W}^{(u)} = (\mathbf{w}_1^{(u)}, ..., \mathbf{w}_{r^*}^{(u)})$ and $\mathbf{W}^{(v)} = (\mathbf{w}_1^{(v)}, ..., \mathbf{w}_{r^*}^{(v)})$ denote the data driven weights which are constructed similar to (4.5).

We combine the ideas of coordinate descent and Bregman iteration methods for solving

(4.8). The main challenge is how to effectively incorporate the orthogonality constraints in l_1-penalized optimization. Our proposed algorithm admits a nested block-wise coordinate descent structure, and we show that within each block the problem boils down to solving a Lasso problem with certain linear equality constraints originated from the orthogonality condition. In the following, we first propose an efficient Bregman iterative coordinate descent algorithm (BCDA) for solving a constrained Lasso criterion. Based on BCDA, we solve the main problem (4.8) for each fixed set of tuning parameters.

We now consider solving the sparse regression problem (4.8). When $r^* = 1$ and $\mathbf{X}$ is an identity matrix, it can be recognized that the problem reduces exactly to a rank-one sparse SVD criterion with respect to a triplet $(d, \mathbf{u}, \mathbf{v})$, which was proposed by Lee *et al.* (2010) for microarray bi-clustering. Lee *et al.* (2010) developed an iterative algorithm to solve the problem, in which two blocks of parameters $(d, \mathbf{u})$ and $(d, \mathbf{v})$ are alternately updated until convergence, with either $\mathbf{v}$ or $\mathbf{u}$ held fixed. Here we adopt the same idea to our general regression setting, minimizing criterion (4.8) by alternately updating two overlapping blocks of parameters $(\mathbf{U}, \mathbf{D})$ and $(\mathbf{V}, \mathbf{D})$, referred to as the $\mathbf{U}$-step and $\mathbf{V}$-step, respectively. For fixed $\mathbf{V}$, the $\mathbf{U}$-step is to solve

$$\min_{\mathbf{U},\mathbf{D}} \left\{ \frac{1}{2}\|\mathbf{y} - \mathbf{X}^{(u)}\mathrm{vec}(\mathbf{UD})\|^2 + \lambda_1 \|\mathbf{W}^{(u)} \circ \mathbf{UD}\|_1 \right\} \text{ subject to } \mathbf{U}^\top \mathbf{U} = \mathbf{I}. \tag{4.9}$$

where $\mathbf{y} = \mathrm{vec}(\mathbf{Y})$ and $\mathbf{X}^{(u)} = \mathbf{V} \otimes \mathbf{X}$. On the other hand, for fixed $\mathbf{U}$, the $\mathbf{V}$-step is to solve

$$\min_{\mathbf{V},\mathbf{D}} \left\{ \frac{1}{2}\|\mathbf{y}^t - \mathbf{X}^{(v)}\mathrm{vec}(\mathbf{VD})\|^2 + \lambda_2 \|\mathbf{W}^{(v)} \circ \mathbf{VD}\|_1 \right\} \text{ subject to } \mathbf{V}^\top \mathbf{V} = \mathbf{I}. \tag{4.10}$$

where $\mathbf{y}^t = \mathrm{vec}(\mathbf{Y}^\top)$ and $\mathbf{X}^{(v)} = (\mathbf{XU}) \otimes \mathbf{I}_q$.

Both the $\mathbf{U}$-step and the $\mathbf{V}$-step can be solved by a blockwise iterative BCDA algorithm. We use the $\mathbf{U}$-step for illustration. The main idea is that the objective (4.9) can be viewed as with respect to $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^\top, \cdots, \boldsymbol{\beta}_{r^*}^\top)^\top = (d_1\mathbf{u}_1^\top, \cdots, d_r\mathbf{u}_{r^*}^\top)^\top = \mathrm{vec}(\mathbf{UD})$, and it can be shown that updating each $d_k\mathbf{u}_k$ with others held fixed is exactly a linearly constrained Lasso problem. We emphasis that the nonlinear scale constraint $\mathbf{u}_k^\top \mathbf{u}_k = 1$ is completed

avoided as the optimization variable is $d_k\mathbf{u}_k$ rather than $\mathbf{u}_k$. The algorithm for solving the $\mathbf{U}$-step with any fixed λ_1 is given below.

Iterative BCDA for Solving U-Step

Initialization: $\mathbf{U}^{(0)} = (\mathbf{u}_1^{(0)}, \ldots, \mathbf{u}_{r^*}^{(0)})$ satisfying $\mathbf{U}^{(0)\top}\mathbf{U}^{(0)} = \mathbf{I}$; $\mathbf{D}^{(0)} = \text{diag}\{d_k^{(0)}, k = 1, \ldots, r^*\}$.

For $s = 1, 2, \ldots$

1. Compute the current residual $\mathbf{r} = \mathbf{y} - \mathbf{X}^{(u)}\boldsymbol{\beta}^{(s)}$.
2. For each $k = 1, \ldots, r^*$,
   (a) $\mathbf{X}_k \longleftarrow \mathbf{v}_k \otimes \mathbf{X}$;
   (b) Update $\mathbf{r} \longleftarrow \mathbf{r} + d_k^{(s)}\mathbf{X}_k\mathbf{u}_k^{(s)}$;
   (c) $\mathbf{A}_k \longleftarrow \mathbf{U}_{-k}^{\top}$; $\mathbf{b} = 0$;
   (d) Solve ConLasso$(\boldsymbol{\beta}_k; \mathbf{r}, \mathbf{X}_k, \mathbf{A}_k, \mathbf{b}, \lambda_1\mathbf{w}_k^{(u)})$.
   (e) $d_k^{(s+1)} \longleftarrow \|\boldsymbol{\beta}_k\|$; $\mathbf{u}_k^{(s+1)} \longleftarrow \boldsymbol{\beta}_k/\|\boldsymbol{\beta}_k\|$;
   (f) Update $\mathbf{r} \longleftarrow \mathbf{r} - d_k^{(s+1)}\mathbf{X}_k\mathbf{u}_k^{(s+1)}$.

Repeat until convergence.

Finally, our sparse SVD regression algorithm is structured as follows.

SOSVD via BEA

Initialization: obtain $\mathbf{V}^{(0)} \in \Re^{q\times r}$, and $\mathbf{V}^{(0)\top}\mathbf{V}^{(0)} = \mathbf{I}$.

For $k = 1, 2, ...$

1. **U** step: solve (4.9) by iterative BCDA with $\mathbf{y} = \text{vec}(\mathbf{Y})$ and $\mathbf{X}^{(u)} = \mathbf{V}^{(k-1)} \otimes \mathbf{X}$.
2. **V** step: solve (4.10) by iterative BCDA with $\mathbf{y}^t = \text{vec}(\mathbf{Y}^\top)$ and $\mathbf{X}^{(v)} = (\mathbf{X}\mathbf{U}^{(k)}) \otimes \mathbf{I}_q$.
3. **D** step: solve the least squares problem:

$$\mathbf{d}^{(k)} = \arg\min_{\mathbf{d}} \|\mathbf{y} - \mathbf{Z}\mathbf{d}\|^2$$

where $\mathbf{y} = \text{vec}(\mathbf{Y})$ and $\mathbf{Z} = (\mathbf{z}_1, ..., \mathbf{z}_r)$ with $\mathbf{z}_h = \text{vec}(\mathbf{X}\mathbf{u}_h^{(k)}\mathbf{v}_h^{(k)\top})$.

Repeat until convergence.

4.4 Tuning Parameter Selection

For the SOSVD via BEA algorithm, to efficiently determine the optimal set of tuning parameters $\lambda = (\lambda_1, \lambda_2)$ and hence the optimal solution, we consider a Bayesian Information criterion (BIC) (Schwarz, 1978), which is commonly used for model selection in regression analysis especially in the case $q = 1$. However, the classical BIC may fail and lead to excessive overfitting in high-dimensional models (An *et al.*, 2008). We hence modify the BIC for the setting of $p \geq n$ according to An *et al.* (2008). Denote $(\hat{\mathbf{U}}^{(\lambda)}, \hat{\mathbf{V}}^{(\lambda)}, \hat{\mathbf{D}}^{(\lambda)})$ as the fitted value of $(\mathbf{U}, \mathbf{V}, \mathbf{D})$ with the tuning parameter being λ. Define

$$\text{BIC}(\lambda) = \begin{cases} \log(\text{SSE}(\lambda)) + \frac{\log(qn)}{qn} df(\lambda) & \text{when } p < n \\ \log(\text{SSE}(\lambda)) + \frac{2\log(pq)}{qn} df(\lambda) & \text{when } p \geq n \end{cases}$$

where $\text{SSE}(\lambda) = \|\mathbf{Y} - \mathbf{X}\hat{\mathbf{C}}^{(\lambda)}\|_F^2$ stands for the sum of squared error, and $df(\lambda)$ is the model degrees of freedom. We consider

$$\hat{df}(\lambda) = \sum_{k=1}^{r^*} \left\{ \frac{r_x}{p} \sum_{i=1}^{p} \text{I}(\hat{u}_{ik}^{(\lambda)} \neq 0) + \sum_{j=1}^{q} \text{I}(\hat{v}_{jk}^{(\lambda)} \neq 0) - r^* \right\}$$

where r_x is the rank of the design matrix $\mathbf{X}$, $\mathrm{I}(\cdot)$ is the indicator function. When $\lambda = 0$, the model reduces to a reduced-rank regression problem and $\hat{df}(0)$ becomes its effective number of free parameters.

For the SOSVD via SEA method, to select the optimal solution in the solution path we again consider BIC. Denote $(\hat{d}_k, \hat{\mathbf{u}}_k, \hat{\mathbf{v}}_k)$ as the fitted value of the k^{th} triplet $(d_k, \mathbf{u}_k, \mathbf{v}_k)$ with the tuning parameter being λ_k. Define

$$\mathrm{BIC}(\lambda_k) = \begin{cases} \log(\mathrm{SSE}(\lambda_k)) + \frac{\log(qn)}{qn} df(\lambda_k) & \text{when } p < n \\ \log(\mathrm{SSE}(\lambda_k)) + \frac{2\log(pq)}{qn} df(\lambda_k) & \text{when } p \geq n \end{cases}$$

where $\mathrm{SSE}(\lambda_k) = \|\mathbf{Y_k} - \hat{d}_k \mathbf{X} \hat{\mathbf{u}}_k \hat{\mathbf{v}}_k^\top\|_F^2$ stands for the sum of squared error for the k^{th} layer. The model degrees of freedom, $df(\lambda_k)$, is expressed as:

$$\hat{df}(\lambda_k) = \sum_{i=1}^{p} \mathrm{I}(\hat{u}_{ik} \neq 0) + \sum_{j=1}^{q} \mathrm{I}(\hat{v}_{jk} \neq 0) - 1$$

The loss of 1 degree of freedom is due to the two constraints ($\|\mathbf{u}_k\|_2 = 1$ and $\|\mathbf{v}_k\|_2 = 1$) and one free parameter d_k (Chen *et al.*, 2012). To illustrate the advantage of enforcing orthogonality for sparse SVD recovery using the prosed SEA and BEA algorithm we conduct a simulation study at different signal to noise ratio in the following section.

4.5 Simulation

The simulation study is performed under the supervised framework of multivariate reduced rank regression problem. We compare the estimation, prediction and sparse SVD recovery performance of the ordinary least squares estimator (OLS), the reduced rank regression estimator (RRR), the proposed SOSVD via BEA (BEA) under $\mu = \rho = 1$, and the estimator proposed by Chen *et al.* (2012) based on an iterative exclusive extraction algorithm (IEEA). The IEEA performs sparse reduced-rank estimation locally in the vicinity of some good initial estimator, and the orthogonality condition is not enforced when searching for the sparsity pattern in the SVD. In both BEA and IEEA methods, the penalization can be made data adaptive; we hence also consider their adaptive versions, denoted as AdBEA and

AdIEEA, respectively, in which the adaptive weights are constructed based on an initial reduced rank regression. See details in Chen *et al.* (2012). Further we run the simulation for two additional set up: 1) the proposed SOSVD via SEA ($\mu = 1, \rho = 1.1$) using adaptive weights (AdSEAorth), and 2) the adaptive version of sequential extraction algorithm proposed by Chen *et al.* (2012) to solve 4.3 without the orthogonality constraint (AdSEA). We have implemented all the methods in R (R Development Core Team, 2012). All computation was done on computers with 3.4 GHz CPU, 8 GB RAM and Linux operating system.

We consider two simulation models of different dimensions. In both setups, the covariate matrix $\mathbf{X}$ is constructed by generating its n rows as i.i.d. samples from MVN$(\mathbf{0}, \Gamma)$, where $\Gamma = (\Gamma_{ij})_{p\times p}$ and $\Gamma_{ij} = 0.5^{|i-j|}$. In Model I we set $p = q = 25$, $n = 100$ and $r^* = 3$. The SVD of the 25×25 rank-3 coefficient matrix $\mathbf{C}$ is given by $\sum_{k=1}^{3} d_k \mathbf{u}_k \mathbf{v}_k^\top$, where $d_1 = 20$, $d_2 = 15$, $d_3 = 10$, and the $\mathbf{u}_k$s and $\mathbf{v}_k$s are generated as follows,

$$
\begin{aligned}
\check{\mathbf{u}}_1 &= [\text{unif}(\mathcal{A}_u, 5), \text{rep}(0, 20)]^\top, \\
\check{\mathbf{u}}_2 &= [\text{rep}(0, 3), \check{u}_{4,1}, -\check{u}_{5,1}, \text{unif}(\mathcal{A}_u, 3), \text{rep}(0, 17)]^\top, \\
\check{\mathbf{u}}_3 &= [\text{rep}(0, 8), \text{unif}(\mathcal{A}_u, 2), \text{rep}(0, 15)]^\top, \\
\mathbf{u}_k &= \check{\mathbf{u}}_k / \|\check{\mathbf{u}}_k\| \text{ for } k = 1, 2, 3;
\end{aligned}
$$

$$
\begin{aligned}
\check{\mathbf{v}}_1 &= [\text{unif}(\mathcal{A}_v, 5), \text{rep}(0, 20)]^\top, \\
\check{\mathbf{v}}_2 &= [\text{rep}(0, 5), \text{unif}(\mathcal{A}_v, 5), \text{rep}(0, 15)]^\top, \\
\check{\mathbf{v}}_3 &= [\text{rep}(0, 10), \text{unif}(\mathcal{A}_v, 5), \text{rep}(0, 10)]^\top, \\
\mathbf{v}_k &= \check{\mathbf{v}}_k / \|\check{\mathbf{v}}_k\| \text{ for } k = 1, 2, 3.
\end{aligned}
$$

The notation unif$(\mathcal{A}, b)$ denotes a vector of length b whose entries are i.i.d. samples from the uniform distribution on the set of real values $\mathcal{A}$; we use $\mathcal{A}_u = \pm 1$, $\mathcal{A}_v = [-1, -0.3] \cup [0.3, 1]$. The notation rep$(a, b)$ denotes a vector of length b, whose entries are all equal to a. The notation $\check{u}_{a,k}$ denotes the ath entry of $\check{\mathbf{u}}_k$. In Model II we set $p = 100$, $q = 25$, $n = 50$ and $r^* = 3$. All settings are the same as in Model I except that additional 75 noise predictors are

added, i.e., each 25×1 left singular vectors $\mathbf{u}_k$ from Model I is appended with 75 zeros. The matrix $\mathbf{Y}$ is then generated by $\mathbf{Y} = \mathbf{XC} + \mathbf{E}$, where the entries of $\mathbf{E}$ are i.i.d. samples from $N(0, \sigma^2)$. We define the signal to noise ratio (SNR) as $\|d_3 \mathbf{X}\mathbf{u}_3\mathbf{v}_3^\top\|_2/\|\mathbf{E}\|_2$, and the error standard deviation σ is chosen to make SNR at certain level. The experiment was replicated 100 times for each SNR $= 0.25$, 0.5, 1 or 2, covering a full range of signal strengths relative to the noise level.

Our simulation setup characterizes a realistic high-dimensional multivariate dependence structure, i.e., three subsets of response variables are related to three subsets of predictors, and there exist many useless predictors that are not related to any response as well as some irrelevant response variables that can not be explained by any predictor. The sets of important predictors may overlap with each other, and their effects on the corresponding response variables are allowed to be varying, as reflected by the distinct entries of each $\mathbf{v}_k$ vector, and some of the effects can be quite small. These make the estimation challenging.

Tables 4.1 and 4.2 summarize the simulation results for Models I and II, respectively. The model accuracy is measured by the average scaled mean squared error from all runs, i.e., $\mathrm{Er}(\hat{\mathbf{C}}) = 100\|\mathbf{C} - \hat{\mathbf{C}}\|_F^2/(pq)$ for estimating $\mathbf{C}$, and $\mathrm{Er}(\hat{\mathbf{Y}}) = 100\mathrm{tr}\{(\mathbf{C} - \hat{\mathbf{C}})^\top \Gamma (\mathbf{C} - \hat{\mathbf{C}})\}/(nq)$ for predicting $\mathbf{Y}$, where $\mathrm{tr}(\cdot)$ denotes the trace of a matrix. The averaged false positive rates (FPR) and false negative rates (FNR) in recovering the sparsity pattern in the SVD of $\mathbf{C}$ are reported. The orthogonality of the estimated SVD is measured by $\mathrm{ORT} = \|\hat{\mathbf{U}}^\top\hat{\mathbf{U}}\|_1 + \|\hat{\mathbf{V}}^\top\hat{\mathbf{V}}\|_1 - 2r^*$, averaged from all runs. For each method, the average computation time per iteration ans tuning parameter setting is also reported.

The sparse SVD methods greatly outperform both OLS and RRR, owning to their capability of conducting simultaneous latent and original variable selection for dimension reduction. By enforcing the orthogonality of the estimated SVD, the BEA method achieves better performance than the IEEA method in every category. The improvement can be substantial when the SNR is low or the model dimension is high; in these challenging cases, the orthogonality condition is critical in restricting and guiding the search of the optimal

Table 4.1: *SOSVD: Model I (p=q=25, n=100)*

	OLS	RRR	BEA	IEEA	AdBEA	AdIEEA	AdSEAorth	AdSEA
				SNR=0.25				
Er($\hat{\mathbf{C}}$)	34.16 (19.51)	6.17 (3.37)	2.89 (2.19)	3.79 (2.21)	0.92 (0.74)	0.93 (0.81)	2.10 (2.68)	2.32 (3.18)
Er($\hat{\mathbf{Y}}$)	5.24 (2.99)	1.07 (0.64)	0.49 (0.35)	0.63 (0.38)	0.18 (0.15)	0.18 (0.16)	0.42 (0.56)	0.43 (0.55)
FPR	100.00%	100.00%	2.01%	3.83%	0.20%	0.69%	2.44%	3.30%
FNR	0.00%	0.00%	0.11%	0.04%	0.04%	0.04%	0.45%	0.00%
ORT	0.00	0.00	0.10 (0.08)	0.11 (0.09)	0.04 (0.04)	0.07 (0.10)	0.01 (0.002)	0.16 (0.19)
				SNR=0.5				
Er($\hat{\mathbf{C}}$)	17.52 (9.56)	3.11 (1.72)	1.27 (0.98)	1.98 (1.28)	0.43 (0.29)	0.46 (0.31)	0.87 (1.98)	0.51 (0.53)
Er($\hat{\mathbf{Y}}$)	2.66 (1.46)	0.54 (0.32)	0.21 (0.16)	0.32 (0.20)	0.08 (0.05)	0.09 (0.06)	0.17 (0.45)	0.10 (0.09)
FPR	100.00%	100.00%	1.49%	2.99%	0.08%	0.48%	0.62%	0.53%
FNR	0.00%	0.00%	0.04%	0.00%	0.00%	0.00%	0.20%	0.00%
ORT	0.00	0.00	0.06 (0.04)	0.08 (0.05)	0.03 (0.03)	0.04 (0.05)	0.01 (0.00)	0.04 (0.05)
				SNR=1				
ErC	8.94 (4.76)	1.56 (0.87)	0.63 (0.41)	0.95 (0.52)	0.21 (0.14)	0.23 (0.16)	0.23 (0.38)	0.21 (0.13)
Er(Y)	1.37 (0.72)	0.27 (0.15)	0.11 (0.07)	0.16 (0.09)	0.04 (0.03)	0.05 (0.03)	0.04 (0.07)	0.04 (0.02)
FPR	100.00%	100.00%	0.99%	2.04%	0.13%	0.54%	0.06%	0.07%
FNR	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
ORT	0.00	0.00	0.04 (0.02)	0.05 (0.04)	0.02 (0.01)	0.03 (0.05)	0.00 (0.00)	0.02 (0.02)
				SNR=2				
Er($\hat{\mathbf{C}}$)	4.38 (2.48)	0.77 (0.45)	0.32 (0.24)	0.49 (0.28)	0.10 (0.07)	0.11 (0.08)	0.10 (0.06)	0.11 (0.06)
Er($\hat{\mathbf{Y}}$)	0.66 (0.37)	0.13 (0.08)	0.05 (0.04)	0.08 (0.05)	0.02 (0.01)	0.02 (0.02)	0.02 (0.01)	0.02 (0.01)
FPR	100.00%	100.00%	0.42%	1.04%	0.04%	0.32%	0.00%	0.00%
FNR	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
ORT	0.00	0.00	0.02 (0.02)	0.03 (0.03)	0.01 (0.01)	0.02 (0.02)	0.00 (0.00)	0.01 (0.01)
Time	0.01	0.03	0.41	0.14	0.18	0.15	0.04	0.03

sparsity pattern. Note that both BEA and IEEA tend to select slightly more linkages than needed but rarely miss important linkages, especially in Model I, which reflects the over-selection property of l_1 regularization. Adopting adaptive penalization further boosts the performance of the sparse SVD methods in general. While AdBEA and AdIEEA both may substantially outperform their nonadaptive counterparts, the former achieves the lowest estimation/prediction errors and FPR/FNR rates in most cases among all the methods considered. As expected, using the BEA / AdBEA methods bears a higher computational cost than the IEEA / AdIEEA methods, but the runtime is still manageable for practical usage.

As pointed by Chen *et al.* (2012) the sequential extraction algorithm need not produce SVD layers of $\mathbf{C}$ and, therefore, is not suitable to recover desired SVD structure of $\mathbf{C}$. The sequential extraction algorithm is rather suitable for the fitted-value SVD of $\mathbf{C}$ (refer to remark on Lemma 1.3 of Chen *et al.* (2012) for more details). Thus, it is not surprising that the AdSEA method under performs compared to the BEA and IEEA methods. However,

Table 4.2: *SOSVD: Model II (p=100, q=25, n=50)*

	OLS	RRR	BEA	IEEA	AdBEA	AdIEEA	AdSEAorth	AdSEA
				SNR=0.25				
Er($\hat{\mathbf{C}}$)	36.03 (11.37)	18.76 (2.79)	5.16 (3.66)	8.45 (5.98)	3.82 (3.62)	4.63 (4.96)	7.71 (5.43)	8.69 (6.07)
Er($\hat{\mathbf{Y}}$)	61.45 (21.01)	28.68 (4.80)	5.91 (4.16)	10.17 (8.16)	4.55 (3.89)	5.49 (6.44)	9.65 (7.04)	12.60 (8.58)
FPR	100.00%	100.00%	3.88%	3.46%	2.78%	1.52%	2.76%	13.45%
FNR	0.00%	0.00%	19.22%	29.74%	15.15%	20.56%	9.03%	5.00%
ORT	0.00	0.00	0.19 (0.20)	0.24 (0.35)	0.16 (0.15)	0.26 (0.31)	0.03 (0.02)	0.55 (0.28)
				SNR=0.5				
Er($\hat{\mathbf{C}}$)	26.65 (6.45)	16.96 (2.44)	2.27 (2.78)	4.03 (5.15)	1.97 (2.38)	2.12 (3.14)	4.92 (4.28)	5.54 (5.72)
Er($\hat{\mathbf{Y}}$)	44.06 (11.89)	25.45 (3.38)	2.63 (2.63)	4.66 (6.02)	2.34 (2.65)	2.40 (3.34)	7.19 (5.15)	7.60 (7.53)
FPR	100.00%	100.00%	3.08%	3.48%	2.45%	1.59%	2.01%	9.43%
FNR	0.00%	0.00%	7.93%	12.22%	7.85%	10.96%	7.73%	2.76%
ORT	0.00	0.00	0.13 (0.11)	0.19 (0.18)	0.16 (0.17)	0.31 (0.37)	0.03 (0.04)	0.46 (0.45)
				SNR=1				
Er($\hat{\mathbf{C}}$)	19.17 (3.34)	15.50 (2.42)	0.88 (1.07)	1.70 (2.88)	0.81 (1.47)	0.66 (1.39)	4.90 (4.42)	4.76 (5.01)
Er($\hat{\mathbf{Y}}$)	31.15 (6.14)	23.60 (2.81)	1.02 (1.10)	2.05 (3.77)	0.94 (1.64)	0.78 (1.51)	5.68 (5.21)	4.77 (5.01)
FPR	100.00%	100.00%	1.84%	3.70%	1.97%	1.65%	1.30%	7.45%
FNR	0.00%	0.00%	2.96%	3.37%	2.59%	2.78%	6.93%	3.48%
ORT	0.00	0.00	0.07 (0.08)	0.2 (0.26)	0.08 (0.10)	0.26 (0.35)	0.03 (0.04)	0.39 (0.35)
				SNR=2				
Er($\hat{\mathbf{C}}$)	17.31 (1.85)	15.21 (2.27)	0.46 (0.33)	1.14 (0.87)	0.52 (1.7)	0.62 (1.59)	1.95 (2.59)	3.38 (4.04)
Er($\hat{\mathbf{Y}}$)	27.12 (3.35)	22.62 (2.52)	0.53 (0.34)	1.32 (0.89)	0.60 (1.99)	0.72 (1.86)	2.18 (2.98)	4.10 (4.73)
FPR	100.00%	100.00%	1.49%	4.50%	2.11%	2.43%	0.56%	5.96%
FNR	0.00%	0.00%	0.00%	0.19%	1.22%	1.44%	5.00%	2.27%
ORT	0.00	0.00	0.05 (0.04)	0.23 (0.19)	0.07 (0.08)	0.25 (0.24)	0.02 (0.05)	0.33 (0.37)
Time	0.01	0.08	0.76	0.27	0.68	0.32	0.08	0.05

if orthogonality is enforced using the same sequential fitting procedure, the results are improved as shown by the AdSEAorth method. The AdSEAorth method, albeit not as effective as its BEA counterparts, performs best in conforming to orthogonality constraint. Since the AdSEAorth method uses $\rho = 1.1$ (compared to BEA methods that use $\rho = 1$) the orthogonality condition is imposed more strictly with every updating cycle. Overall, the appeal of the sequential fitting strategy is its simplicity and computational efficiency which can make it more suitable for solving more complicated objective functions than the one under consideration in this report. The computation time for the AdSEA and AdSEAorth methods is quite low as compared to AdBEA and AdIEEA methods, which makes the sequential fitting procedure an attractive alternative for estimating models with rank higher than 3. For improving the AdSEAorth procedure, one can iteratively perform the sequential extraction procedure. Similar to the IEEA method, each time the previous sparse estimates can be used as initial values to refine the estimation, until a convergence

is reached. Since the SEA is generally computationally efficient, the computational cost of iteratively performing SEA will be reasonable for application purposes.

Chapter 5

Discussion and future work

In this report we demonstrated the usefulness of imposing the difficult non-convex orthogonality constraint for estimating sparse and orthogonal SVD of a noisy data matrix or a coefficient matrix of a reduced rank multivariate regression model. When confronted with situations that are more likely in real life applications,high dimensional model and low signal to noise ratio, for example, we demonstrate the effectiveness of the proposed SOSVD via SEA and SOSVD via EEA methods to restrict and steer the hunt for optimal sparsity to facilitate model interpretation. We also show the versatility of the Coordinate Descent Algorithm and Bregman iterative methods to solve various formulations of non-convex optimization problems.

There are quite a few exciting ideas for future research that can build upon the current work. We have focused on the adaptive lasso method for inducing sparsity. It will be interesting to develop efficient algorithms for the current methods using other established sparsity-inducing penalties, such as the elastic net penalty (Zou *et al.*, 2004), adaptive grouping penalty (Wang and Zhu, 2008), the smoothly clipped absolute deviation (SCAD) penalty (Fan and Li, 2001), and minimax concave penalty (MCP) (Zhang, 2010). Since SVD is one of many decompositions of a matrix, it may be worthwhile to consider other matrix decomposition forms such as QR decomposition. In special cases when the coefficient matrix is square one may consider Jordan decomposition, Schur decomposition, Cholesky decomposition, LU decomposition and Takagi's factorization.

Bibliography

An, H., Huang, D., Yao, Q. and Zhang, C.-H. (2008) Stepwise searching for feature variables in high-dimensional linear regression. *Technical Report, Department of Statistics, London School of Economics*.

Bregman, L. (1967) The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, **7**, 200 – 217.

Breheny, P. and Huang, J. (2011) Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, **5**, 232–253.

Chen, K., Chan, K.-S. and Stenseth, N. C. (2012) Reduced rank stochastic regression with a sparse singular value decomposition. *Journal of the Royal Statistical Society: Series B.*, **74**, 203–221.

Fan, J. and Li, R. (2001) Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, **96**, 1348–1360.

Fan, J., Zhang, J. and Yu, K. (2012) Vast portfolio selection with gross-exposure constraints. *Journal of the American Statistical Association*, **107**, 592–606.

Goldstein, T. and Osher, S. (2009) The split bregman method for l1-regularized problems. *SIAM J. Img. Sci.*, **2**, 323–343.

Hastie, T., Tibshirani, R. and Friedman, J. H. (2008) *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer.

Lee, M., Shen, H., Huang, J. Z. and Marron, J. S. (2010) Biclustering via sparse singular value decomposition. *Biometrics*, **66**, 1087–1095.

Meinshausen, N. and Bühlmann, P. (2006) High-dimensional graphs and variable selection with the lasso. *Ann. Statist.*, **34**, 1436–1462.

R Development Core Team (2012) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Schwarz, G. (1978) Estimating the dimension of a model. *The Annals of Statistics*, **6**, 461–464.

Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, **58**, 267–288.

Wang, S. and Zhu, J. (2008) Variable selection for model-based high-dimensional clustering and its application to microarray data. *Biometrics*, **64**, 440–448.

Yuan, M. and Lin, Y. (2006) Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **68**, 49–67.

Zhang, C.-H. (2010) Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, **38**, 894–942.

Zou, H. (2006) The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, **101**, 1418–1429.

Zou, H., Hastie, T. and Tibshirani, R. (2004) Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, **15**.

Appendix A

R code

A.1 Code for constrained adaptive lasso

```
library(MASS)
library(Matrix)
frobenius.square = function (x) { sum(x*x) }
frobenius = function (x) { sqrt(sum(x*x)) }
l1 = function(x) { sum(abs(x)) }
l2 = function(x) { sum(x*x) }
bregman.lasso = function (x,y, A, lambda, b=rep(0, nrow(A)),mu=1, max.iter = 100,
nu=1.1, inner.conv=1e-5)
{ # input the design matrix x and then response vector y, and mu
n= nrow(x)
p = ncol(x)
# gamma=2 is suggested
ols = abs (ginv(t(x) %*% x) %*% t(x) %*% y)
ols[ols==0] = 0.000001
wts=as.vector(ols^-0 ) # can use positive value instead of 0
betaold=numeric(p) # creating a beta vector
betanew=numeric(p) # creating a beta vector
```

```
betaold = ginv(t(x)%*%x) %*% t(x) %*%y
ck=0
ckplus1=0
counter=0
repeat
{
counter=counter+1
betanew = betaold # store betaold in betanew
error = y - x %*% betanew # r is the residual
for(j in 1 : p)
{
r = error + betaold[j]*x[,j]
z = sum(r*x[,j]) + mu * ( sum(((ck / mu)+b)*A[,j]) - sum( ( A[,-j]%*%betanew[-j] ) *( A[,j]
) ) ) # find the z tilda
lambdastar = lambda * wts[j]
betanew[j] = sign(z) * (abs(z) -lambdastar) * identity(abs(z) ¿ lambdastar) / ( mu*sum(A[,j]*A[,j])
+ sum(x[,j]*x[,j] ) ) # the jth element of betanew is updated using the soft thresholding
error = error - (betanew[j]-betaold[j]) * x[,j] # updating error for next iteration or next j.
}
beta_difference = (betaold-betanew)
constraint = A%*%betanew - b
difference = sqrt( sum(beta_difference*beta_difference) ) + sqrt(l2(constraint))# magnitude
of the difference
if (difference > inner.conv)
{
betaold = betanew
ckplus1 = ck - (A%*%betanew - b)*nu
```

```
ck = ckplus1
mu = mu*nu
}
if ((difference < inner.conv) | counter > max.iter) {break}
} #end of repeat loop
as.vector(betanew)
}
```

A.2 Code for SOSVD via SEA

```
# The function surr.ortho performs unit rank SSVD for any particular layer to be
#orthogonal to previous layers
surr.ortho = function(x,Y, Au, bu=rep(0, nrow(Au)),Av, bv=rep(0, nrow(Av)), gamma,
mu,nu,inner.iter=100, max.iter=50,inner.conv=1e-5, ustar, vstar, dstar,lambdamax,
lambdamin,lambdalength)
# ustar, vstar and dstar are initial estimators
{ n = nrow(x)
p = ncol(x)
q = ncol(Y)
qn = ncol(Y) * nrow(Y)
wd = abs(dstar)^-gamma
wu = vector()
wu = abs(ustar)^-gamma
wv = vector()
wv=abs(vstar)^-gamma
wd = as.numeric (wd) # gamma=2
k= exp( seq (log(lambdamax),log(lambdamin),length=lambdalength))
k=sort(k, decreasing=TRUE)
```

ulambda = matrix(NA, nrow=length(ustar), ncol=length(k)) # ulambda matrix stores all the descended u vectors for each lambda

vlambda=matrix(NA, nrow=length(vstar), ncol=length(k)) # vlambda matrix stores all the descended v vectors for each lambda

dlambda=numeric(length(k)) # dlambda vector stores all the descended d entries for each lambda

dlambda [1:length(k)]=NA

y = as.vector(Y) #vectorizing matrix Y

bic = numeric(length(k))

bic[1:length(k)]=NA

for (h in 1:length(k)) {

Cp = dstar * (ustar %*% t(vstar))

lambda = k[h]

counter=0

repeat {

counter = counter + 1

if (counter==1) dhat = dstar

XV = kronecker(diag($(wv)^{-1}$) , x %*%ustar) # getting the design matrix using kronecker

lambda_v = lambda* wd * sum (wu * abs(ustar))

tuning = lambda_v

lasso_beta = vector()

lasso_beta = bregman.lasso(XV,y, A=Av %*% diag($(dhat*wv)^{-1}$) , lambda=tuning, b=bv,mu)

vtick=vector()

vtick = lasso_beta

dhat = $(\text{sum} (\text{diag}((wv)^{-1} \%*\% \text{vtick}) * (\text{diag}((wv)^{-1}) \%*\% \text{vtick})))^{0.5}$

if (sum(abs(vtick))==0) {break} # if the vtick obtained has all zero entries then break

vhat=diag($(dhat*wv)^{-1}$) %*% vtick

```
# We now fix v as vhat and get the objective function
XU = kronecker(vhat, x %*% diag( (wu) ^-1 ) )
lambda_u = lambda * wd * sum (wv * abs(vhat)) # use vhat from previous step
lasso_beta = vector() # store the list output in a list called output
tuning = lambda_u
lasso_beta = bregman.lasso(XU,y, A=Au %*% diag( (dhat*wu) ^-1 ), lambda=tuning, b=bu,mu)
utick=vector()
utick=lasso_beta
if (sum(abs(utick))==0) {break} # if the utick obtained has all zero entries then break out
dhat = (sum ( ( diag( (wu) ^-1 ) %*% utick) * ( diag( (wu) ^-1 ) %*% utick) )) ^0.5
uhat=diag( (dhat*wu) ^-1 ) %*% utick
# Checking for convergence
Cc = dhat * (uhat %*% t(vhat))
error = frobenius(Cc-Cp) / frobenius(Cp)
if (error < 0.000001 | counter>max.iter) {break}
else {
Cp=Cc
ustar=uhat
vstar=vhat
}
} # end of repeat loop
if ( ( sum(abs(vtick)) >0.1) && ( sum(abs(utick)) >0.1) ) # iff the vtick and utick vectors
are non zero then update the ustar, vstar and dstar values for next lambda
{
ustar=uhat
vstar=vhat
dstar = dhat
```

```
ulambda[,h] = ustar
vlambda[,h]=vstar
dlambda[h]=dstar
# Compute bic for each lambda
SSE = frobenius(Y-dlambda[h]* (x %*% ulambda[,h] %*% t(vlambda[,h])))
df = sum(ulambda[,h]!=0) + sum(vlambda[,h]!=0) -1
if(p≥q) bic[h] = log (SSE) + (log(qn) * df)/qn
if(p<q) bic[h] = log (SSE) + 2*(log(qn) * df)/qn
}
C_est = dlambda[which.min(bic)]* (ulambda[,which.min(bic)] %*% t(vlambda[,which.min(bic)]))
list (C_est = C_est, d= dlambda[which.min(bic)], u = ulambda[,which.min(bic)] ,dpath=dlambda,
upath=ulambda, vpath=vlambda, v = vlambda[,which.min(bic)],bicpath=bic, lambda=k,
lambda.chosen=k[which.min(bic)],lambda.number=which(k==k[which.min(bic)]) )
}
# The following function uses all the above functions to perform SOSVD via SEA
sea.ortho = function (x, Y, rank.appx,mu,nu, gamma,lambdamin,
lambdamax,lambdalength=100)
{
n = nrow(x)
p = ncol(x)
q = ncol(Y)
qn = ncol(Y) * nrow(Y)
# If rank.appx is greater than rank of X or q then stop and display an error message.
# One way to determine rank of X is couting number or non zero singular values
#(or values greater than 10^−8) of the SVD of X.
if (rank.appx> min(length(svd(x)$d > 1e-8),q)) stop ("Please provide a lower value for
rank.appx")
```

```
ini=RRR(Y,x,nrank=rank.appx) # initial estimator is reduced rank least squares estimator
uinitial = matrix(NA, nrow=p, ncol=rank.appx)
vinitial = matrix(NA, nrow=q, ncol=rank.appx)
dinitial = numeric(length=rank.appx)
for (i in 1:rank.appx)
{
uinitial[,i] = ini$U[,i]
vinitial[,i] = ini$V[,i]
dinitial[i] = ini$D[i,i]
}
# lets define vectors and matrices to store the final descended values for each layer
ufinal = matrix(NA, nrow=p, ncol=rank.appx)
vfinal = matrix(NA, nrow=q, ncol=rank.appx)
dfinal = numeric(length=rank.appx)
dfinal[1:rank.appx]=NA
# Following loop will estimate each layer
for (r in 1:rank.appx)
{
ustar = uinitial[,r] # Assigning initial values for u, v and d
ustar[abs(ustar)<0.05]=1e-20
vstar = vinitial[,r]
vstar[abs(vstar)<0.05]=1e-20
dstar = dinitial[r]
if (r==1) #For the first layer we don't need any orthogonality constraint.
#So set the Av and Au matrices to be zero.
{
Au = t(rep(0,p))
```

```
Av = t(rep(0,q))
a = surr.ortho(x=x, Y=Y, Au=Au, Av=Av, ustar=ustar, vstar=vstar, dstar=dstar,mu=mu,
nu=nu,gamma=gamma,lambdamin= lambdamin, lambdamax=lambdamax,
lambdalength=lambdalength)
}
if (r!=1)
{
Y = Y - x %*% C_est
# We sequentially perform sparse unit rank regression, each time with the data matrix Y
# replaced by the residual matrix that is obtained by substracting previously estimated
#layers from the original data matrix
Au = t(ufinal[,1:(r-1)]) # For layers afer the first we need to enforce orthogonality.
#So Au takes on previously descended u vectors
Av = t(vfinal[,1:(r-1)]) # Same foe v vectors
a = surr.ortho(x=x, Y=Y, Au=Au, Av=Av, ustar=ustar, vstar=vstar, dstar=dstar, mu=mu,
nu=nu,gamma=gamma,lambdalength=lambdalength,lambdamin= lambdamin,
lambdamax=lambdamax )
}
ufinal[,r] = a$u
vfinal[,r] = a$v
dfinal[r] = a$d
C_est = a$C_est
} # End of for loop where r goes from 1 to rank.appx
list(Cfinal = ufinal %*% diag(dfinal, length(dfinal)) %*% t(vfinal), U=ufinal, V = vfinal,
D = diag(dfinal, length(dfinal)))
}
```

```
# Given the data matrix Y, covariate matrix X, number of layers to be extracted, the
# SOSVD via SEA can be performed as follows
sosvd = list()
sosvd=sea.ortho(X,Y, rank.appx=3,mu=1,nu=1.1, gamma=2, lambdamin=1e-30,
lambdamax=1e-1,lambdalength=500)
```