Sparse Multivariate Reduced-Rank Regression with Covariance Estimation

by

Khalif Halani

B.Sc., University of Victoria, 2015

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Statistics and Actuarial Science
Faculty of Science

SIMON FRASER UNIVERSITY
Fall 2016

Approval

Name: **Khalif Halani**

Degree: **Master of Science (Statistics)**

Title: ***Sparse Multivariate Reduced-Rank Regression with Covariance Estimation***

Examining Committee: **Chair:** Dr. Tim Swartz
Professor

Dr. Jinko Graham
Senior Supervisor
Professor

Dr. Farouk Nathoo
Supervisor
Associate Professor
Department of Mathematics and Statistics
University of Victoria

Dr. Brad McNeney
Internal Examiner
Associate Professor

Date Defended: December 14, 2016

Abstract

Abstract
Multivariate multiple linear regression is multiple linear regression, but with multiple responses. Standard approaches assume that observations from different subjects are uncorrelated and so estimates of the regression parameters can be obtained through separate univariate regressions, regardless of whether the responses are correlated within subjects. There are three main extensions to the simplest model. The first assumes a low rank structure on the coefficient matrix that arises from a latent factor model linking predictors to responses. The second reduces the number of parameters through variable selection. The third allows for correlations between response variables in the low rank model. Chen and Huang propose a new model that falls under the reduced-rank regression framework, employs variable selection, and estimates correlations among error terms. This project reviews their model, describes its implementation, and reports the results of a simulation study evaluating its performance. The project concludes with ideas for further research.

Keywords: Multivariate Regression; Reduced-Rank; Covariance Estimation; Variable Selection; LASSO; Simulation

Acknowledgements

I would like to thank Dr. Jinko Graham for her support and advice throughout my time at SFU. Without her patience and encouragement I would not be where I am today. I would also like to thank all of the professors who taught me during my time at SFU. Dr. Steve Thompson, Dr. Boxin Tang, Dr. Luke Bornn, Dr. Rachel Altman, and Dr. Dave Campbell, I learned so much from all of you and and cannot thank you enough for your time. Thank you to Dr. Brad McNeney and Dr. Farouk Nathoo for agreeing to be part of my committee and for the great feedback. Kelly, Charlene, and Sadika, thank you so much for your help.

My time at SFU would not have been the same if it were not for the many friends I made in the program. In particular, Trevor Thomson, Steven Wu, Matt van Bommel, JinCheol Choi, thank you for making these last few years so enjoyable.

To my parents and my sister: You have always been there for me. I appreciate all the love and support you have given me. Without you this would not have happened.

Table of Contents

Approval **ii**

Abstract **iii**

Acknowledgements **iv**

Table of Contents **v**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Methods **3**
2.1 Reduced-Rank Model 3
2.2 Covariance Estimation and Variable Selection 5
2.3 Algorithm 6

3 Evaluation **11**
3.1 Simulation Design 11
3.2 Variable Selection and Prediction 12
3.3 Timing 16

4 Concluding Remarks **17**
4.1 Future Work 18

Bibliography **19**

Appendix A R Code **21**

List of Tables

Table 3.1 Results from 30 simulations. Reported numbers are means with sd's in parentheses 13

Table 3.2 Timing results from 30 simulations. Reported numbers are means (in seconds) with sd's in parentheses 16

List of Figures

Figure 3.1 Mean Prediction Error vs CV Error with $p = 100$, $p_0 = 20$, $q = 20$, and $\rho_e = 0.9$ 13

Figure 3.2 Mean Prediction Error vs CV Error with $p = 60$, $p_0 = 20$, $q = 60$, and $\rho_e = 0$ 14

Figure 3.3 Comparison of estimated and actual precision matrices from a simulation run with $p = 100$, $p_0 = 20$, $q = 20$, and $\rho_e = 0.9$ 15

Chapter 1

Introduction

Multivariate multiple linear regression is a technique to infer linear relationships between response variables, $\mathbf{Y} = \{Y_1, \ldots, Y_q\}$, and a set of input variables, $\mathbf{X} = \{X_1, \ldots, X_p\}$. This is similar to multiple linear regression, but with the change that there are now multiple response variables. In fact, the model used to estimate these linear relationships is a simple extension of the multiple linear regression model.

Let $\mathbf{X}$ be the $n \times p$ matrix of predictor variables, $\mathbf{Y}$, the $n \times q$ response matrix, and $\mathbf{E}$, the $n \times q$ error matrix. The multivariate regression model is then,

$$\mathbf{Y} = \mathbf{XC} + \mathbf{E}, \tag{1.1}$$

where $\mathbf{C}$ is the $p \times q$ matrix of regression parameters, and each row of $\mathbf{E}$ is from the $MVN(0, \Sigma_e)$ distribution. Under this model, the q observations from a single subject have covariance matrix Σ_e; however, the observations from different subjects are assumed uncorrelated. Thus, the least squares estimates of the regression parameters, $\hat{\mathbf{C}}$, can be obtained by performing q separate univariate regressions.

Let $\hat{c}_{(i)}$ be the least squares estimate obtained from the ith response,

$$\hat{c}_{(i)} = (\mathbf{X'X})^{\mathbf{-1}}\mathbf{X'Y}_{(\mathbf{i})}; \tag{1.2}$$

then the collection of these separate estimates gives us,

$$\begin{aligned} \hat{\mathbf{C}} &= \left\{\hat{c}_{(1)}|\hat{c}_{(2)}|\ldots|\hat{c}_{(q)}\right\} \\ &= (\mathbf{X'X})^{\mathbf{-1}}\mathbf{X'}\left\{\mathbf{Y}_{(1)}|\mathbf{Y}_{(2)}|\ldots|\mathbf{Y}_{(q)}\right\} \\ &= (\mathbf{X'X})^{\mathbf{-1}}\mathbf{X'Y}. \end{aligned} \tag{1.3}$$

From this result, we can see that “multivariate regression is a procedure that has no true multivariate context” [Izenman, 2013].

To add multivariate context, three main extensions have been proposed. The first is to assume a low rank structure on the coefficient matrix that arises from a latent factor model linking the predictors to the responses. The second is to reduce the number of parameters in the model through variable selection. The third extension is to allow for correlations between the response variables [Chen and Huang, 2016].

If we assume a low rank structure on the coefficient matrix $\mathbf{C}$ such that $rank(\mathbf{C}) = r \leq min(p, q)$. (Recall that the rank of a matrix is defined as the dimension of the vector space spanned by its columns.) The coefficient matrix can then be decomposed in the following way,

$$\mathbf{C} = \mathbf{B}\mathbf{A}'$$

where $\mathbf{B}$ is a $p \times r$ matrix and $\mathbf{A}$ is an $q \times r$ matrix. This is called the reduced-rank regression model.

By assuming this low-rank structure of the coefficient matrix, we effectively reduce the number of parameters to be estimated from pq to $(p+q)r$. Under this formulation we can think of $\mathbf{XB}$ as underlying predictive factors and $\mathbf{A}'$ as their coefficients. This model gives us a nice framework from which to work, but does not perform variable selection. Moreover, some reduced rank regression models still assume that the errors are independently and identically distributed. In the article *Sparse reduced rank regression with covariance estimation* [Chen and Huang, 2016], the authors propose a new model and a corresponding algorithm to fit this model that falls under the reduced-rank regression framework, but also employs variable selection and estimation of the correlation among error terms. In this report, I review this method, describe its implementation, and evaluate its performance through a simulation study. The project concludes with ideas for further research and a discussion of this method’s potential application in the area of imaging genetics.

Chapter 2

Methods

2.1 Reduced-Rank Model

In the multivariate linear regression model, we have response variables, $Y_1, Y_2, \ldots, Y_q$ and multiple predictor variables, $X_1, X_2, \ldots, X_p$. We assume a linear relationship between the response variables and the predictor variables such that,

$$Y_j = \sum_{k=1}^{p} X_k c_{kj} + \epsilon_j,\ j = 1, 2, \ldots, q \tag{2.1}$$

where ϵ_j is the error term and the collection of the ϵ_j's are assumed to be jointly $MVN(0, \Sigma_e)$ distributed. Thus the regression model can be written,

$$\mathbf{Y} = \mathbf{XC} + \mathbf{E}, \tag{2.2}$$

where $\mathbf{Y}$ is the $n \times q$ matrix of responses, $\mathbf{X}$ is the $n \times p$ matrix of predictors, $\mathbf{C}$ is the $p \times q$ coefficient matrix, and $\mathbf{E}$ is the $n \times q$ error matrix, each row of which is drawn from the $MVN(0, \Sigma_e)$ distribution.

The negative log-likelihood of the unknown parameters in the model can be written as,

$$l(\mathbf{C}, \Sigma_e) \propto \frac{1}{n} \mathrm{tr} \left[(\mathbf{Y} - \mathbf{XC})' \Sigma_e^{-1} (\mathbf{Y} - \mathbf{XC}) \right] - \log |\Sigma_e^{-1}| \tag{2.3}$$

or

$$l(\mathbf{C}, \mathbf{\Omega}) \propto \frac{1}{n} \mathrm{tr} \left[(\mathbf{Y} - \mathbf{XC})' \mathbf{\Omega} (\mathbf{Y} - \mathbf{XC}) \right] - \log |\mathbf{\Omega}|, \tag{2.4}$$

where $\mathbf{\Omega} = \Sigma_e^{-1}$. Minimizing this equation over the unknown parameters, $(\mathbf{C}, \mathbf{\Omega})$, we can obtain the maximum likelihood estimates, $\hat{\mathbf{C}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$ and $\hat{\mathbf{\Omega}} = \mathbf{Y}'[1 - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}']\mathbf{Y}$ [Tso, 1981]. The maximum likelihood estimate for $\mathbf{C}$ is actually equal to the estimate given by performing q separate linear regressions and does not depend on the error covariance structure. If, however, we impose some structure on $\mathbf{C}$ this will not be the case.

Reduced-rank regression works by imposing a rank constraint on the coefficient matrix such that,

$$rank(\mathbf{C}) = r \leq min(p, q). \tag{2.5}$$

By doing so, we can greatly reduce the number of free parameters that need to be estimated for the coefficient matrix.

With the imposition of the rank constraint on the coefficient matrix we can further decompose the coefficient matrix, $\mathbf{C}$, into the product of two rank-r matrices,

$$\mathbf{C} = \mathbf{B}\mathbf{A}', \tag{2.6}$$

where $\mathbf{B}$ has dimension $p \times r$ and $\mathbf{A}$ has dimension $q \times r$. Thus we are able to reduce the number of free parameters from pq, assuming a full-rank coefficient matrix, to $(p+q)r$ in the reduced-rank setting. Under this formulation, the negative log-likelihood of the reduced-rank regression is,

$$l(\mathbf{B}, \mathbf{A}, \mathbf{\Omega}) \propto \frac{1}{n} \mathrm{tr}\left[(\mathbf{Y} - \mathbf{X}\mathbf{B}\mathbf{A}')'\mathbf{\Omega}(\mathbf{Y} - \mathbf{X}\mathbf{B}\mathbf{A}')\right] - \log|\mathbf{\Omega}|. \tag{2.7}$$

In this setting, in order to ensure identifiability, additional constraints on $\mathbf{A}$ and $\mathbf{B}$ are required. For example, without placing constraints on these matrices, if $\mathbf{F}$ is an $r \times r$ invertible matrix and $\mathbf{B}\mathbf{A}' = \mathbf{C}$, then we have $\mathbf{B}\mathbf{F}\mathbf{F}^{-1}\mathbf{A}' = \mathbf{G}\mathbf{H}' = \mathbf{C}$, where $\mathbf{G} = \mathbf{B}\mathbf{F}$ is a $p \times r$ matrix and $\mathbf{H}' = \mathbf{F}^{-1}\mathbf{A}'$ is an $r \times q$ matrix. One way to formulate reduced rank regression is as a two-step procedure [Davies and Tso, 1982]. In the first step, we use the singular-value decomposition of the $n \times q$ fitted response matrix, $\hat{\mathbf{Y}}_{OLS}$, from ordinary least squares regression of $\mathbf{Y}$ on $\mathbf{X}$ to obtain $\hat{\mathbf{Y}}_r$, a rank-r approximation. In the second step, we find the coefficient matrix, $\mathbf{C}$ such that $\mathbf{X}\mathbf{C} = \hat{\mathbf{Y}}_r$. This formulation can be used to derive a solution $\mathbf{C} = \mathbf{A}\mathbf{B}'$ such that $\mathbf{A}'\mathbf{\Omega}\mathbf{A} = I_r$ and $\mathbf{B}'\mathbf{S}_x\mathbf{B}$ is diagonal, where $\mathbf{S}_x$ is the sample covariance of $\mathbf{X}$ [Chen and Huang, 2012].

In addition to reducing the number of parameters to be estimated, the reduced-rank formulation is described as "finding the interaction between the q response variables as a group and the predictors $\mathbf{X}$" [Chen and Huang, 2016]. One possible explanation of this statement is as follows [SAS, 2016]. Conceptually, reduced-rank regression selects factors underlying $\mathbf{Y}$, represented by linear combinations of the response variables, that account for as much variation as possible in the OLS predictions, $\hat{\mathbf{Y}}_{OLS}$. These factors are projected onto the column space of $\mathbf{X}$ to obtain the factors underlying $\mathbf{X}$. One particular procedure for performing the estimation in the reduced-rank regression [Davies and Tso, 1982] does this explicitly. This procedure uses the singular value decomposition (SVD) of the unconstrained OLS predictions, $\hat{\mathbf{Y}}_{OLS} = \sum_{i=1}^{\tau} \lambda_i u_i v_i' = UDV'$, where τ is the rank of $\hat{\mathbf{Y}}_{OLS}$, to form a reduced-rank projection matrix $\mathbf{P}_r = \sum_{i=1}^{r} v_i v_i'$, where $r < \tau$. This reduced-rank projection matrix is then used to compute the reduced-rank coefficient matrix. This is clearly very different from unconstrained multivariate regression which "has no true multivariate context" [Izenman, 2013].

2.2 Covariance Estimation and Variable Selection

The reduced-rank model clearly has some advantages over the classic multivariate regression model. However, it is still quite limited. Reduced rank regression does not address the issue of consistent covariance estimation for high-dimensional data. In high-dimensional settings, standard estimators of covariances are not consistent (i.e. the elements of the estimated covariance matrix do not converge in probability to their population counterparts) and can lead to invalid conclusions [Cai et al., 2016]. Also, the reduced-rank regression does not provide a method for variable selection. Both of these issues are important in many applications. In order to address these issues, Chen and Huang propose a penalized version of the reduced-rank regression model that aims to jointly perform variable selection and covariance estimation. These goals are achieved through the incorporation of two separate penalizations on the likelihood equations. The first is a row-wise penalty on $\mathbf{B}$ which allows us to select predictive variables. The second is an element-wise lasso penalty on the precision matrix $\mathbf{\Omega}$.

By imposing these penalties, it is possible to avoid overfitting when the sample size is small. That is, by introducing some bias we may be able to reduce the variance of our predictions. The element-wise lasso penalty on the precision matrix provides the necessary regularization to estimate the precision matrix, while the row-wise penalty on $\mathbf{B}$ allows us to select the most important variables in the model. These properties are desirable, especially when there are a large number of predictors, of which only a small subset may be relevant.

In order to achieve the first goal of variable selection we can consider a form of penalized regression from Chen and Huang [2012] which results in the following objective function:

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{B}} \ \frac{1}{n}\mathrm{tr}\left[(\mathbf{Y} - \mathbf{XBA}')'(\mathbf{Y} - \mathbf{XBA}')\right] + \sum_{j=1}^{p} \lambda_j ||\mathbf{B}^{\mathbf{j}}||_2, \\ &\text{s.t. } \mathbf{A}'\mathbf{A} = \mathbf{I}_r, \end{aligned} \tag{2.8}$$

where $\mathbf{B}^j$ is the jth row of $\mathbf{B}$ and λ_j is a tuning parameter that encourages row-wise sparsity. This form of penalized regression is very similar to the group-wise lasso penalty proposed by Yuan and Lin [2006]. In the case of many predictors, we can greatly reduce the computational burden of determining penalty parameters by setting all λ_j's equal. This gives us

$$\begin{aligned}
&\min_{\mathbf{A},\mathbf{B}} \frac{1}{n}\mathrm{tr}\left[(\mathbf{Y}-\mathbf{XBA'})'\mathbf{\Omega}(\mathbf{Y}-\mathbf{XBA'})\right] + \lambda\sum_{j=1}^{p}||\mathbf{B^j}||_2, \\
&\text{s.t. } \mathbf{A'A} = \mathbf{I}_r.
\end{aligned} \tag{2.9}$$

It is important to note that the solution to this optimization problem is not unique, or rather, it is only unique up to an $r \times r$ orthonormal matrix (i.e. $\mathbf{QQ'} = \mathbf{Q'Q} = \mathbf{I}$). For example, if $(\hat{\mathbf{A}}, \hat{\mathbf{B}})$ is a solution to the optimization and $\mathbf{Q}$ is an $r \times r$ orthonormal matrix, letting $\tilde{\mathbf{A}} = \hat{\mathbf{A}}\mathbf{Q}$ and $\tilde{\mathbf{B}} = \hat{\mathbf{B}}\mathbf{Q}$, it follows that $\hat{\mathbf{B}}\hat{\mathbf{A}}' = \tilde{\mathbf{B}}\tilde{\mathbf{A}}'$ since $\mathbf{QQ'} = \mathbf{I}$ [Chen and Huang, 2012]. The penalty term is also unchanged since the length of a row of $\mathbf{B}$ is not affected by post-multiplying by $\mathbf{Q}$.

In order to incorporate the correlation structure into this model, Chen and Huang [2016] use a penalized Gaussian Likelihood to estimate the precision matrix [Yuan and Lin, 2007]. This likelihood penalizes all the off-diagonal entries in the precision matrix, encouraging sparsity. For a fixed value of $(\mathbf{A}, \mathbf{B})$, the precision matrix $\mathbf{\Omega}$ is the solution to

$$\min_{\mathbf{\Omega}} \mathrm{tr}(\mathbf{\Sigma}_R\mathbf{\Omega}) - \log|\mathbf{\Omega}| + \lambda_1\sum_{j\neq j'}|\omega_{j'j}|, \tag{2.10}$$

where $\mathbf{\Sigma}_R = \frac{1}{n}(\mathbf{Y}-\mathbf{XBA'})'(\mathbf{Y}-\mathbf{XBA'})$ is the sample covariance matrix of the errors, $\omega_{j'j}$ is the (j', j) entry in the precision matrix $\mathbf{\Omega}$ and λ_1 is the tuning parameter that controls the sparsity of the solution.

Combining these two penalties into the reduced-rank regression model, we get the following penalized likelihood,

$$\begin{aligned}
&\min_{\mathbf{A},\mathbf{B},\mathbf{\Omega}} \frac{1}{n}\mathrm{tr}\left[(\mathbf{Y}-\mathbf{XBA'})'\mathbf{\Omega}(\mathbf{Y}-\mathbf{XBA'})\right] - \log|\mathbf{\Omega}| + \lambda_1\sum_{j\neq j'}|\omega_{j'j}| + \lambda_2\sum_{j=1}^{p}||\mathbf{B^j}||_2, \\
&\text{s.t. } \mathbf{A'\Omega A} = \mathbf{I}_r.
\end{aligned} \tag{2.11}$$

2.3 Algorithm

Due to the complexity of the problem, solving this equation can be very difficult. We can, however, break the problem down into two separate parts and solve each separately given the results of the other part. That is, for a given coefficient matrix $\mathbf{C} = \mathbf{BA}'$ we can optimize the precision matrix, $\mathbf{\Omega}$, with the appropriate penalty, λ_1, and for a given precision matrix, $\mathbf{\Omega}$, we can solve the reduced-rank regression problem incorporating the row-wise penalty, λ_2. The algorithm proposed by Chen and Huang [2016] alternates between these two tasks.

Given a coefficient matrix $\mathbf{C} = \mathbf{BA}'$ we can optimize the following criterion to solve for the precision matrix,

$$\min_{\mathbf{\Omega}} \ \mathrm{tr}(\mathbf{\Sigma}_R \mathbf{\Omega}) - \log|\mathbf{\Omega}| + \lambda_1 \sum_{j \neq j'} |\omega_{j'j}| \tag{2.12}$$

where $\mathbf{\Sigma}_R = \frac{1}{n}(\mathbf{Y} - \mathbf{XBA}')'(\mathbf{Y} - \mathbf{XBA}')$. This is the same problem as that of learning the structure in an undirected Gaussian graphical model. The graphical lasso (GLASSO) [Friedman et al., 2008] is a popular algorithm for learning this structure using l_1 regularization to control the number of zeroes in $\mathbf{\Omega}$. However, convergence of the GLASSO algorithm can be troublesome and “the converged precision matrix might not be the inverse of the estimated covariance” [Mazumder and Hastie, 2012]. This is due to the fact that the GLASSO algorithm is actually targeting the covariance matrix rather than its inverse (the precision matrix). Because our model seeks to identify a sparse precision matrix, we instead use the related DP-GLASSO algorithm [Mazumder and Hastie, 2012] which directly targets the precision matrix.

For a given precision matrix $\mathbf{\Omega}$ we then minimize over $(\mathbf{A}, \mathbf{B})$,

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{B}} \ \frac{1}{n}\mathrm{tr}\left[(\mathbf{Y} - \mathbf{XBA}')\mathbf{\Omega}(\mathbf{Y} - \mathbf{XBA}')'\right] + \lambda_2 \sum_{j=1}^{p} ||\mathbf{B}^{\mathbf{j}}||_2, \\ &\text{s.t. } \mathbf{A}'\mathbf{\Omega}\mathbf{A} = \mathbf{I}_r. \end{aligned} \tag{2.13}$$

If we let $\tilde{\mathbf{A}} = \mathbf{\Omega}^{1/2}\mathbf{A}$ we can rewrite the equation as

$$\begin{aligned} &\min_{\tilde{\mathbf{A}},\mathbf{B}} \ \frac{1}{n}\mathrm{tr}\left[(\mathbf{Y}\mathbf{\Omega}^{1/2} - \mathbf{XB}\tilde{\mathbf{A}}')(\mathbf{Y}\mathbf{\Omega}^{1/2} - \mathbf{XB}\tilde{\mathbf{A}}')'\right] + \lambda_2 \sum_{j=1}^{p} ||\mathbf{B}^{\mathbf{j}}||_2 \\ &= \min_{\tilde{\mathbf{A}},\mathbf{B}} \ \frac{1}{n}||\mathbf{Y}\mathbf{\Omega}^{1/2} - \mathbf{XB}\tilde{\mathbf{A}}'|| + \lambda_2 \sum_{j=1}^{p} ||\mathbf{B}^{\mathbf{j}}||_2 \\ &= \min_{\tilde{\mathbf{A}},\mathbf{B}} \ \frac{1}{n}||\tilde{\mathbf{Y}} - \mathbf{XB}\tilde{\mathbf{A}}'|| + \lambda_2 \sum_{j=1}^{p} ||\mathbf{B}^{\mathbf{j}}||_2, \\ &\text{s.t. } \tilde{\mathbf{A}}'\tilde{\mathbf{A}} = \mathbf{I}_r, \end{aligned} \tag{2.14}$$

where $\tilde{\mathbf{Y}} = \mathbf{Y}\mathbf{\Omega}^{1/2}$. This has the same form as equation (2.8) and can be solved in the manner proposed by Chen and Huang [2012].

In order to minimize equation (2.14) we use an algorithm that iterates between optimizing $\tilde{\mathbf{A}}$ for a fixed $\mathbf{B}$, and optimizing $\mathbf{B}$ for a fixed $\tilde{\mathbf{A}}$. For a fixed $\mathbf{B}$, equation (2.14) can be expressed as

$$\min_{\tilde{\mathbf{A}}} ||\tilde{\mathbf{Y}} - \mathbf{X}\mathbf{B}\tilde{\mathbf{A}}'||, \quad \text{s.t. } \tilde{\mathbf{A}}'\tilde{\mathbf{A}} = \mathbf{I}_r. \tag{2.15}$$

This minimization problem can be viewed as trying to find an orthogonal matrix $\tilde{\mathbf{A}}$ that transforms $\mathbf{XB}$ to fit $\tilde{\mathbf{Y}}$ as closely as possible. Such a problem is known as an "orthogonal Procrustes Problem"[Gower and Dijksterhuis, 2004] and has the solution $\tilde{\mathbf{A}} = \mathbf{U}\mathbf{V}'$ where $\mathbf{U}$ and $\mathbf{V}$ are from the singular value decomposition of $\mathbf{Y}'\mathbf{X}\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{V}'$ [Chen and Huang, 2012].

For a fixed $\tilde{\mathbf{A}}$, optimizing $\mathbf{B}$ is difficult because the function in equation (2.14) is convex, but non-differentiable [Chen and Huang, 2016]. In order to overcome these difficulties, Chen and Huang [2012] propose the use of a subgradient method [Bertsekas, 1999] and provide an algorithm that employs blockwise coordinate descent [Friedman et al., 2007] to update $\mathbf{B}$ one row at a time, holding the rest of the matrix fixed. The subgradient method is a standard way to minimize a non-smooth convex function.

When minimizing equation (2.14) for a fixed $\tilde{\mathbf{A}}$ with respect to a row $\mathbf{B}^l$ of $\mathbf{B}$, the objective function is non-differentiable at the vector $\mathbf{B}^l = 0$ because the penalty term involves the non-differentiable function $||\mathbf{B}^l||_2 = \sqrt{\sum_{i=1}^{q}(\mathbf{B}_i^l)^2}$. To see why $||\mathbf{B}^l||_2$ is non-differentiable, consider the simplest case of $q = 1$. Then $||\mathbf{B}^l||_2$ reduces to the absolute value function $|\mathbf{B}_1^l|$ which is not differentiable at zero. The subgradient generalizes the derivative to functions such as $||\mathbf{B}^l||_2$ which are not differentiable at all points. A subgradient at a point can be viewed as the slope of any tangent line to the function at that point that lies at or below the function at points everywhere else on its domain [Wikipedia, 2016b].

At points where the function is differentiable, subderivatives are equal to the derivatives and unique. At points where the function is non-differentiable, subderivatives are non-unique. A point $\mathbf{B}^l$ is a minimum of the function if and only if zero is a subgradient of the function at $\mathbf{B}^l$ [Gordon and Tibshirani]. For $||\mathbf{B}^l|| \neq 0$, the subgradient is unique and is equal to $2\mathbf{X}_l\left(\sum_{k\neq l}^{p}\mathbf{X}_k\mathbf{B}^k - \mathbf{Y}\mathbf{A}\right) + \lambda_2\mathbf{B}^l/||\mathbf{B}^l||$ [Chen and Huang, 2012]. For $||\mathbf{B}^l|| = 0$, the subgradients are of the form $2X_l\left(\sum_{k\neq j}^{p}X_kB^k - YA\right) + \lambda_2 s_l$, where $||s_l||$ is any vector with Euclidean length $||s_l|| < 1$ [Chen and Huang, 2012, Gordon and Tibshirani].

We wish to find the $\mathbf{B}^l$ which has zero as its subgradient. If $\mathbf{B}^l$ is a differentiable point, this means we check whether zero is its derivative. If $\mathbf{B}^l$ is a non-differentiable point, this means we check whether zero is in its set of subgradients. Chen and Huang [2012] refer to these checks as "solving the subgradient equations". That is, we solve for $\mathbf{B}^l$ in

$$2\mathbf{X}_l'\left(\sum_{k\neq l}^{p}\mathbf{X}_k\mathbf{B}^k - \mathbf{Y}\tilde{\mathbf{A}}\right) + \lambda_2 s_l = 0, \quad l = 1, 2, ..., p, \tag{2.16}$$

where $s_l = \mathbf{B}^l/||\mathbf{B}^l||$ if $||\mathbf{B}^l|| \neq 0$ or s_l is a vector satisfying $||s_l|| < 1$ if $||\mathbf{B}^l|| = 0$.

For $||\mathbf{B}^l|| = 0$, we can solve for s_l in the subgradient equations and find

$$s_l = -\frac{2}{\lambda_2}\mathbf{X}_l'\left(\sum_{k\neq l}^{p}\mathbf{X}_k\mathbf{B}^k - \mathbf{Y}\tilde{\mathbf{A}}\right) = -\frac{2}{\lambda_2}\mathbf{X}_l'\mathbf{R}_l, \tag{2.17}$$

where $\mathbf{R}_l = \mathbf{Y}\tilde{\mathbf{A}} - \sum_{k\neq l}^{p}\mathbf{X}_k\mathbf{B}^k$ and then check whether $||s_l|| < 1$. If

$$||s_l|| = \frac{2}{\lambda_2}||\mathbf{X}_l'\mathbf{R}_l|| < 1, \tag{2.18}$$

then zero is in the set of subgradients for $\mathbf{B}^l = 0$; i.e., $\mathbf{B}^l = 0$ is the minimizer of equation (2.14) for fixed $\tilde{\mathbf{A}}$ with respect to $\mathbf{B}^l$.

For $||\mathbf{B}^l|| \neq 0$, we have the subgradient (gradient) equation,

$$2\mathbf{X}_l'\left(\sum_{k\neq l}^{p}\mathbf{X}_k\mathbf{B}^k - \mathbf{Y}\tilde{\mathbf{A}}\right) + \lambda_2\frac{\mathbf{B}^l}{||\mathbf{B}^l||} = 0, \tag{2.19}$$

which can be transformed into

$$-2\mathbf{X}_l'(\mathbf{R}_l - \mathbf{X}_l\mathbf{B}^l) + \lambda_2\frac{\mathbf{B}^l}{||\mathbf{B}^l||} = 0. \tag{2.20}$$

Solving for $\mathbf{B}^l$ we get

$$\mathbf{B}^l = \left(\mathbf{X}_l'\mathbf{X}_l + \frac{\lambda_2}{2||\mathbf{B}^l||}\right)^{-1}\mathbf{X}_l'R_l. \tag{2.21}$$

This equation involves $||\mathbf{B}^l||$ on the right-hand side. However, by taking the norm on both sides, we obtain $||\mathbf{B}^l|| = (||\mathbf{X}_l'\mathbf{R}_l|| - \lambda_2/2)||\mathbf{X}_l||^2$. By substituting this expression for $||\mathbf{B}^l||$ into equation (2.21), we obtain

$$\mathbf{B}^l = \frac{1}{\mathbf{X}_l'\mathbf{X}_l}\left[1 - \frac{\lambda_2}{2||\mathbf{X}_l'\mathbf{R}_l||}\right]\mathbf{X}_l'\mathbf{R}_l. \tag{2.22}$$

In summary, we first check if $\mathbf{B}^l = 0$ by checking if equation (2.18) to see if $\frac{2}{\lambda_2}||\mathbf{X}_l'\mathbf{R}_l|| < 1$. If $\mathbf{B}^l \neq 0$, then $\frac{2}{\lambda_2}||\mathbf{X}_l'\mathbf{R}_l|| \geq 1$ in equation (2.18) and $\mathbf{B}^l$ is given by equation (2.22). These two cases can be combined to obtain a general expression for $\mathbf{B}^l$:

$$\mathbf{B}^l = \frac{1}{\mathbf{X}_l'\mathbf{X}_l}\left[1 - \frac{\lambda_2}{2||\mathbf{X}_l'\mathbf{R}_l||}\right]_+ \mathbf{X}_l'\mathbf{R}_l, \tag{2.23}$$

where $\mathbf{R}_l = \mathbf{Y}\tilde{\mathbf{A}} - \sum_{k\neq l}^{p} \mathbf{X}_k\mathbf{B}^k$ and "+" indicates the positive part of a real number.

By iterating over these steps of estimating $\mathbf{\Omega}$, $\mathbf{A}$, and $\mathbf{B}$ until the objective function is minimized, we can easily obtain estimates for both our coefficient matrix $\mathbf{C} = \mathbf{BA}'$ and precision matrix $\mathbf{\Omega}$ and arrive at the following overall algorithm.

Algorithm 1: Algorithm to solve the Sparse Reduced Rank Regression with Covariance Estimation

```
Input  : X, Y, λ1, λ2
Output: A, B, Ω
initialization;
while objective function (2.11) not converged do
    For fixed (A, B), estimate Ω via DP-GLASSO algorithm;
    while objective function (2.14) not converged do
        For fixed B solve Ã by SVD;
        For fixed Ã while B not converged do
            for each row l in B do
                estimate B^l using equation (2.23);
            end
            check for convergence of B;
        end
        check for convergence of objective function (2.14);
    end
    check for convergence of objective function (2.11)
end
```

Use of subgradients in this method guarantees convergence to an arbitrarily close approximation to the minimum value [Bertsekas, 1999]. In the implementation of this algorithm, $\mathbf{B}$ was initialized to be the first r columns of the right singular vectors of $\mathbf{X}$, the nearest possible matrix of rank r to X [Wikipedia, 2016a]. $\mathbf{\Omega}$ was initialized as $\mathbf{S}_Y^{-1}$ or an identity matrix, I, if $\mathbf{S}_Y$ was not invertible.

Chapter 3

Evaluation

Following Chen and Huang [2016], we performed a simulation study to evaluate the performance of the method under a variety of circumstances. Specifically, we evaluated the performance in terms of the prediction error and variable selection ability of the method. For variable selection we consider both the sensitivity (the proportion of non-zero elements that are correctly included) and the specificity (the proportion of zero elements that are correctly excluded) of the rows of $\mathbf{B}$.

3.1 Simulation Design

Data for the simulation study were generated using the model $\mathbf{Y} = \mathbf{XBA}' + \mathbf{E}$. Each component of this model is generated in the following way.

- The columns of the $n \times p$ matrix $\mathbf{X}$ are generated from $N(0, \mathbf{\Sigma}_x)$ with $\mathbf{\Sigma}_x(i,j) = 0.5^{|i-j|}$.
- The first p_0 rows of the $p \times r$ matrix $\mathbf{B}$ are generated from $N(0,1)$ and the rest of the $p - p_0$ rows are set to be zero.
- The entries of the $q \times r$ matrix $\mathbf{A}$ are generated from $N(0,1)$
- The entries in the $n \times q$ matrix $\mathbf{E}$ are generated from $N(0, \sigma^2 \mathbf{\Sigma}_e)$ with $\mathbf{\Sigma}_e(i,j) = \rho_e^{|i-j|}$ and σ^2 is chosen so that the signal to noise ratio is 1.

Chen and Huang [2016] state that the signal to noise ratio they wish to be equal to 1 is $trace(\mathbf{C}'\mathbf{\Sigma}_x\mathbf{C})/trace(\mathbf{E}'\mathbf{E})$. However, this equation seems to be different from the actual signal to noise ratio which can be expressed as

$$\frac{trace(\mathbf{C}'\mathbf{\Sigma}_x\mathbf{C})}{trace\left(\frac{1}{n}\mathbb{E}(\mathbf{E}'\mathbf{E})\right)}. \tag{3.1}$$

Evaluating the expectation we get

$$\frac{trace(\mathbf{C}'\mathbf{\Sigma}_x\mathbf{C})}{trace\,(\sigma^2\mathbf{\Sigma}_e)}. \tag{3.2}$$

Therefore, by setting $\sigma^2 = trace(\mathbf{C}'\mathbf{\Sigma}_x\mathbf{C})/trace(\mathbf{\Sigma}_e)$, we can ensure that the signal to noise ratio is one-to-one.

In order to test the algorithm under a variety of circumstances, following Chen and Huang [2016], we set $n = 50$, $r = 3$, let $\rho_e = 0,\ 0.5,$ or 0.9, and considered 3 different combinations of p, p_0, and q, $(p, p_0, q) = (30, 10, 10)$, $(100, 20, 20)$ and $(60, 20, 60)$. Tuning parameters, λ_1 and λ_2, were selected in the same manner as Chen and Huang [2016], using a validation set of size 50. A test set of size 1000 was used to evaluate prediction accuracy. This setup follows closely that of Chen and Huang [2016]. My interpretation of these authors' statements is that a dataset of size $n = 1100$ is generated from the process detailed above, 100 of which are used for the training and validation process, while the remaining 1000 are used as a held-out test to evaluate the prediction performance of the algorithm.

For each set of runs in the simulation (for each combination of $p,\ p_0,\ q,$ and ρ_e), I first performed a coarse grid search over values of λ_1 and λ_2, performing 2-fold cross-validation (CV) at each pair of fixed values. That is, I split the size 100 training and validation set into 2 equally sized parts, train the model on the first half and calculate its prediction error on the second half, and then train the model on the second half and calculate its prediction error on the first half. Then the average of these two prediction errors forms the CV error for that fixed value of (λ_1, λ_2). Once the CV error has been obtained over this coarse grid of (λ_1, λ_2), the pair with the lowest CV error value is chosen as the starting point of a quasi-Newton optimization algorithm to find the optimal (λ_1, λ_2) that minimized the prediction error for each run of the simulation at the same values of $p,\ p_0,\ q,$ and ρ_e.

3.2 Variable Selection and Prediction

From Table 3.1, we can see that the algorithm does a very good job at excluding irrelevant variables in the model (specificity ≈ 1 for all simulations) and does a decent job at selecting the appropriate variables to include in the model. In particular, in the presence of strongly autocorrelated error, the algorithm yields better sensitivity, while still retaining the high specificity, than in the presence of uncorrelated error. I believe that this may be due to under-penalization of the precision matrix when correlation is not present, however more investigation into this would be needed in order to say anything conclusive.

p	p_0	q	ρ_e	Prediction Error	Sensitivity	Specificity
30	10	10	0	22.923 (41.117)	0.317 (0.137)	1(0)
			0.5	37.869 (36.412)	0.447 (0.201)	0.997 (0.013)
			0.9	41.847 (28.14)	0.527 (0.22)	0.972 (0.055)
100	20	20	0	71.491 (47.701)	0.377 (0.159)	0.992 (0.018)
			0.5	90.002 (70.805)	0.347 (0.149)	0.993 (0.018)
			0.9	81.361 (47.577)	0.428 (0.169)	0.99 (0.018)
60	20	60	0	39.54 (37.378)	0.083 (0.065)	1 (0)
			0.5	49.073 (31.933)	0.435 (0.167)	0.99 (0.021)
			0.9	56.529 (48.589)	0.34 (0.193)	1 (0)

Table 3.1: Results from 30 simulations. Reported numbers are means with sd's in parentheses

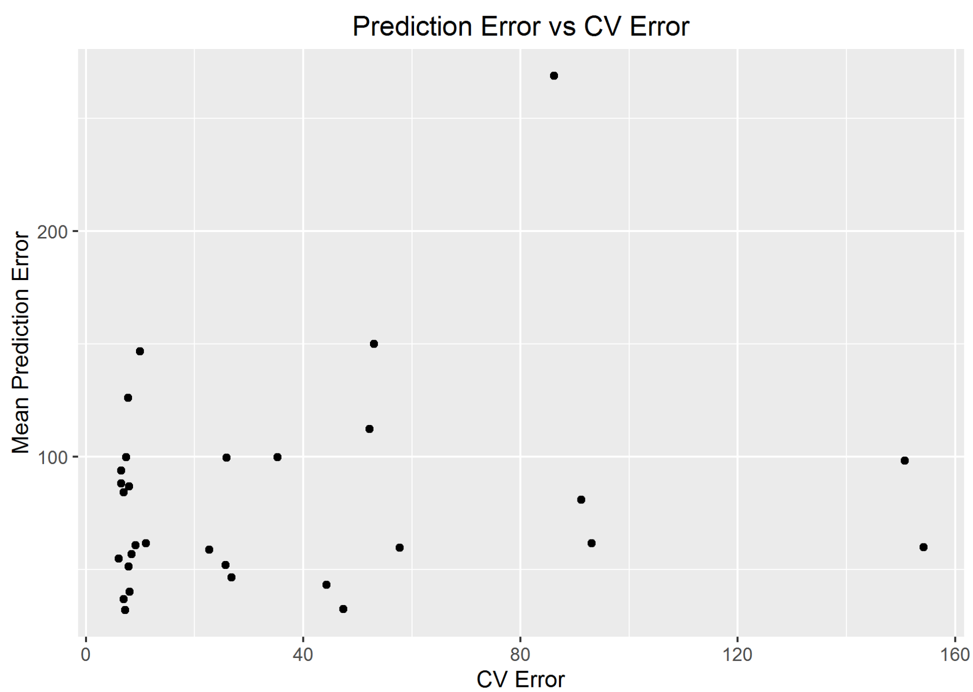

Figure 3.1: Mean Prediction Error vs CV Error with $p = 100$, $p_0 = 20$, $q = 20$, and $\rho_e = 0.9$

The prediction error, on the other hand, seems to suffer badly. In Figures 3.1 and 3.2 we observe that the relationship between the cross-validation error and the prediction error does not have the positive linear relationship that we would expect. In fact, in Figure 3.1 we can see that, despite the small CV error for a number of the simulation runs, the prediction error on those runs is highly variable. This could be due to the fact that 2-fold cross-validation is being used, which will be more variable than higher order k-fold cross-validation [Hastie et al., 2013]. Also, the small size of the training and validation set relative to the test set (100 vs 1000) suggests that (i) the test set will contain more observations at the tail ends of the distribution from which the data were generated, and (ii) the estimates

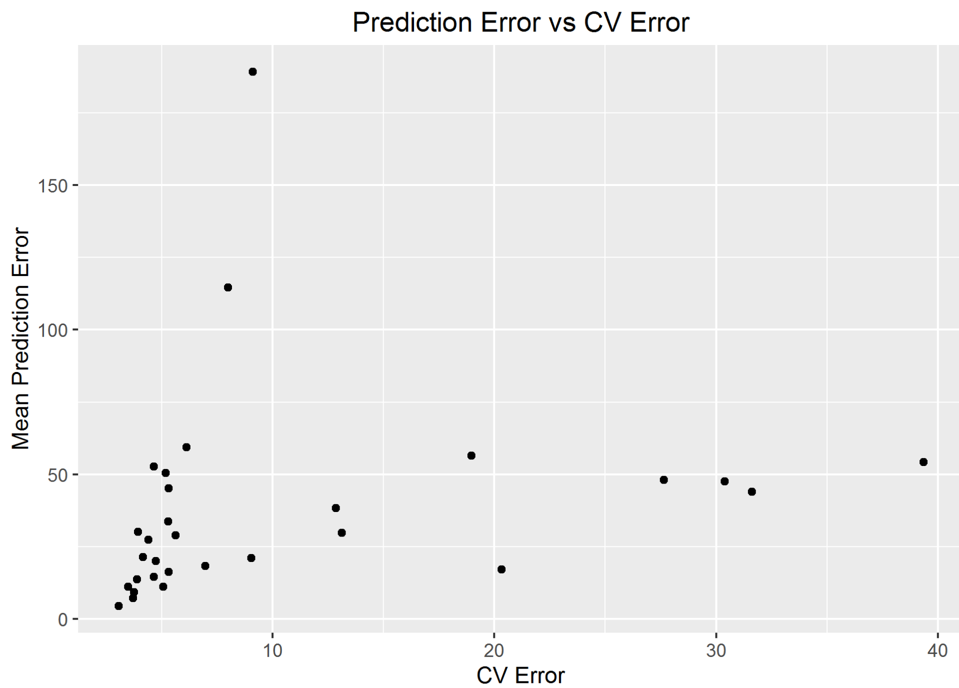

Figure 3.2: Mean Prediction Error vs CV Error with $p = 60$, $p_0 = 20$, $q = 60$, and $\rho_e = 0$

of the tuning parameters may not be well determined.

The results in Table 3.1 differ from those produced in Chen and Huang [2016]. This could be due to a number of reasons. Firstly, the choice of σ^2 in the simulation differs from that in Chen and Huang [2016], which could lead to differences in the data generated. Secondly, the convergence of $\mathbf{B}$ in the algorithm can be problematic, with the algorithm occasionally selecting all variables as non-zero and then pushing the coefficients to large values. Because of this problem, in the implementation of the algorithm, I chose to check for this situation and abort the algorithm if this happened. I believe this happens when the tuning parameter λ_2 (corresponding to the penalty term for $\mathbf{B}$) is too small. The third possible reason for this problem, is that because the tuning process is very time consuming, I sought to automate the tuning process and used an optimization algorithm which does not guarantee that we have found the best (λ_1, λ_2). In my experience, manually tuning the algorithm using progressively finer grid searches yields much better results.

From Figure 3.3, we see that the algorithm does a fairly good job at correctly identifying the structure of the precision matrix, especially in the presence of strongly autocorrelated errors. This, of course, depends on the appropriate selection of the tuning parameter λ_1, corresponding to the penalty term for $\mathbf{\Omega}$, and the algorithm can, if λ_1 is not selected carefully, select all or none of the entries in the precision matrix. Chen and Huang [2016] allude

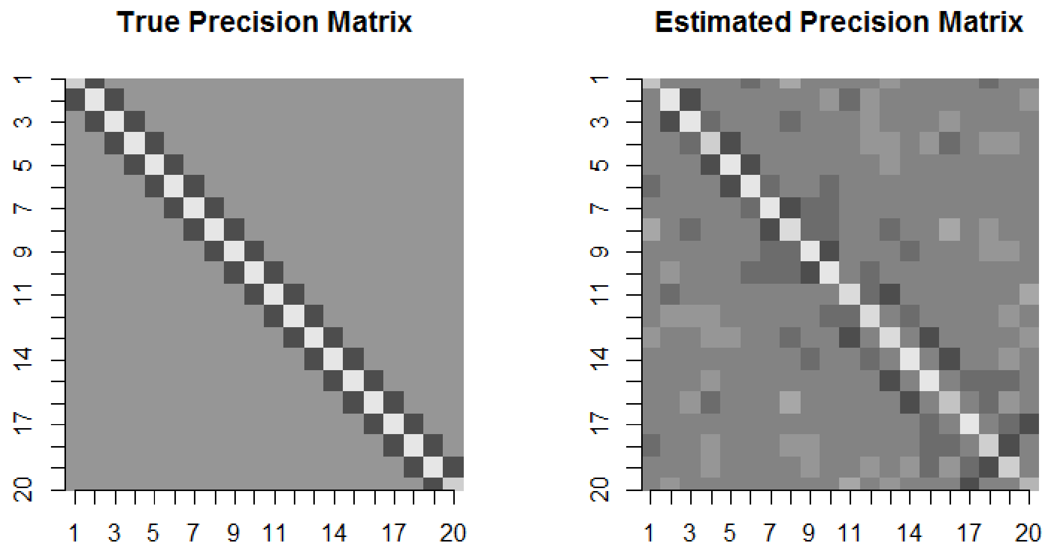

Figure 3.3: Comparison of estimated and actual precision matrices from a simulation run with $p = 100$, $p_0 = 20$, $q = 20$, and $\rho_e = 0.9$

to this behavior when they say "The sensitivity and the specificity are undefined (noted as "-") when all elements are zero and nonzero respectively". Note that their statement is referring to the sensitivity and specificity for the elements of $\mathbf{\Omega}$, which we have not considered in our simulation study. We did not consider sparsity of $\mathbf{\Omega}$ in our evaluation, because it is not of particular interest in the application that motivated this work. Our motivation is an imaging genetics study involving a moderate number, q, of correlated imaging phenotypes. In this application, the number of genetic markers, p, far exceeds the sample size, n, but q is well below n. We expect sparsity in $\mathbf{B}$ because it reflects the genetic association, which is typically sparse. However, we do not expect sparsity in $\mathbf{\Omega}$ because the imaging phenotypes are known to be correlated. Hence sparseness in the elements of $\mathbf{\Omega}$ was of less interest than sparseness in the rows of $\mathbf{B}$.

The simulation design chosen by Chen and Huang could be improved upon in that, rather than random assignment of the entries in $\mathbf{B}$ and $\mathbf{A}$, one could fix the entries and thus gain more insight into the performance of the algorithm. Also, the tuning method used in Chen and Huang [2016] uses a validation set of size $n = 50$ and training set of size $n = 50$. This is similar to performing 2-fold cross-validation and can lead to a more variable CV error than a higher order k-fold cross validation tuning process [Hastie et al., 2013].

The data generating process currently does not account for the tuning parameters but it would be nice to incorporate them, since the performance of the algorithm is highly dependent on the appropriate selection of λ_1 and λ_2. This idea would lend itself better to a Bayesian hierarchical model approach as in Kyung et al. [2010]. However, in utilizing such an approach we may lose some of the speed of the Chen and Huang algorithm.

3.3 Timing

p	p_0	q	ρ_e	Tuning Time	Model Fit Time
30	10	10	0	102.076 (110.636)	1.165 (0.766)
			0.5	101.312 (208.353)	1.444 (1.469)
			0.9	93.436 (94.585)	1.349 (1.043)
100	20	20	0	860.131 (1700.269)	18.749 (14.431)
			0.5	1484.486 (3753.604)	14.92 (11.933)
			0.9	543.897 (504.389)	19.148 (18.805)
60	20	60	0	1049.746 (1055.286)	6.939 (4.856)
			0.5	801.185 (964.388)	4.495 (3.257)
			0.9	617.814 (797.179)	4.767 (3.253)

Table 3.2: Timing results from 30 simulations. Reported numbers are means (in seconds) with sd's in parentheses

In Table 3.2 we can see that fitting the final model, after λ_1 and λ_2 have been chosen via cross-validation, is very fast. The tuning process, on the other hand, takes a significant amount of time and is highly variable. As the dimensionality of the problem increases, the time needed to both tune and fit increases. One possible reason for the high variability in the time needed for the tuning is the use of a quasi-Newton optimization algorithm to "automate" the tuning process. At each iteration of the optimization algorithm, it must compute an approximation to the gradient and Hessian in order to update the parameter values. The optimization algorithm must compute these numerically. I believe that it is this computation that accounts for the long amount of time the tuning took in performing the simulations. Alternative tuning methods, such as adaptive, manual tuning on a fixed grid of (λ_1, λ_2) values, may result in shorter tuning times. Because of the relatively long tuning times, a Bayesian approach [Kyung et al., 2010], in which the tuning is "built-in" to this model, may be competitive.

Chapter 4

Concluding Remarks

The work done by Chen and Huang provides a useful extension to the standard multivariate regression framework that satisfies a number of needs. The model proposed provides dimension reduction through the use of the reduced-rank regression framework, it performs shrinkage and variable selection through its use of the LASSO penalty on the rows of the coefficient matrix, and it performs sparse inverse covariance estimation which allows the model to account for a potentially large number of correlated errors, even when the sample size is small compared to the number of response variables. Their work also gives us a fast algorithm to estimate the unknown parameters in the model. However, as with any model that incorporates a tuning parameter, appropriate tuning of the model can be difficult.

Though the simulation design could be improved upon and manual tuning of the model is preferred to the use of a quasi-Newton optimization algorithm to “automate” the tuning process, this simulation study gives us a good picture of the potential advantages of this model and corresponding algorithm. From the results of the simulation study, we can see that the model does a very good job at excluding unimportant variables and a decent job at identifying important variables. It also performs very good covariance estimation when there are strongly correlated errors. The main drawbacks of this method are that the tuning of the model can take some time and can be quite sensitive to small changes in the tuning parameters and that we cannot obtain measures of uncertainty around the parameter estimates that we obtain. The first drawback of this method can be addressed by taking the time to carefully tune the model. As for the second drawback, unfortunately, there exists no easy way to obtain uncertainty measures. In fact, it has been shown that bootstrap estimates for the standard errors of the lasso estimator are inconsistent [Kyung et al., 2010]. A Bayesian hierarchical framework could yield both parameter estimates and uncertainty measures, but might sacrifice the fast computation that a manually-tuned Chen and Huang algorithm could provide.

4.1 Future Work

There are several avenues for further investigation that we were not able to pursue due to time constraints. As mentioned before, a better simulation design could give us more information about the model's strengths and weaknesses and may give us some insight into why the algorithm performs better in the presence of strong correlation. This project did not investigate the implications of choosing r, the assumed rank of the coefficient matrix. Misspecification of the rank is subject to a bias-variance trade off in that choosing a low rank can introduce bias, while choosing a rank that is too high would increase the variance of the predictions. It is possible that, since we have already incorporated a penalization on the rows of $\mathbf{B}$ and thus on the rows of $\mathbf{C}$, the reduction in variance gained by this penalization may counteract the increase in variance from choosing too high a rank. It is unclear what the implications of mis-specifying r would be and thus further investigation would be needed. A comparison of the performance of this algorithm to a Bayesian hierarchical algorithm could be another area for future investigation. The overall picture that we get from the simulations so far is very encouraging. In future simulation studies, it would be interesting to examine more systematically the performance of the precision-matrix estimator $\hat{\mathbf{\Omega}}$ through measures such as:

- its mean square error: $\frac{1}{q}\sum_{i=1}^{q}\sum_{j=1,j\neq i}^{q}(\hat{\mathbf{\Omega}}_{ij} - \mathbf{\Omega}_{ij})^2$, and
- its sensitivity and specificity (as defined by Chen and Huang; i.e., for the sensitivity, the proportion of non-zero off-diagonal elements that are correctly included and, for the specificity, the proportion of zero off-diagonal elements that are correctly excluded.)

This method has many potential applications because of its low computational burden and its easily interpreted structure. For situations in which the data has high dimensionality and small sample size, such as in imaging genomics, this method could potentially provide insight into which variables are important to the model. The application that motivated this work was an analysis of 56 brain imaging phenotypes and their association with 510 single-nucleotide polymorphisms (SNPs) in 33 candidate genes for Alzheimer's disease [Greenlaw et al., 2016]. In this data, there is spatial correlation in the phenotypes as well as strong correlation between left- and right-hemisphere versions of phenotypes for the same brain structure. We expect a sparse set of SNPs to be associated with the imaging phenotypes based on previous experience with low signal in genetic association studies [Carbonetto and Stephens, 2012]. We also expect the SNPs within genes to be correlated, thus extensions of sparse reduced-rank regression that allow for this group structure at the gene level is another interesting area for future research. Application of this algorithm to the data in Greenlaw et al. [2016] is a potential avenue for future work.

Bibliography

Dimitri P. Bertsekas. *Nonlinear programming.* Athena Scientific, Belmont, Mass, 2nd edition, 1999. ISBN 9781886529007;1886529000;.

T. Tony Cai, Zhao Ren, and Harrison H. Zhou. Estimating structured high-dimensional covariance and precision matrices: Optimal rates and adaptive estimation. *Electron. J. Statist.*, 10(1):1–59, 2016. doi: 10.1214/15-EJS1081. URL `http://dx.doi.org/10.1214/15-EJS1081`.

Peter Carbonetto and Matthew Stephens. Scalable variational inference for bayesian variable selection in regression, and its accuracy in genetic association studies. *Bayesian Anal.*, 7(1):73–108, 03 2012. doi: 10.1214/12-BA703. URL `http://dx.doi.org/10.1214/12-BA703`.

Lisha Chen and Jianhua Z. Huang. Sparse reduced-rank regression for simultaneous dimension reduction and variable selection. *Journal of the American Statistical Association*, 107(500):1533–1545, 2012. doi: 10.1080/01621459.2012.734178. URL `http://dx.doi.org/10.1080/01621459.2012.734178`.

Lisha Chen and Jianhua Z. Huang. Sparse reduced-rank regression with covariance estimation. *Statistics and Computing*, 26(1):461–470, 2016. ISSN 1573-1375. doi: 10.1007/s11222-014-9517-6. URL `http://dx.doi.org/10.1007/s11222-014-9517-6`.

P. T. Davies and M. K-S. Tso. Procedures for reduced-rank regression. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 31(3):244–255, 1982. ISSN 00359254, 14679876. URL `http://www.jstor.org/stable/2347998`.

Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *Ann. Appl. Stat.*, 1(2):302–332, 12 2007. doi: 10.1214/07-AOAS131. URL `http://dx.doi.org/10.1214/07-AOAS131`.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. doi: 10.1093/biostatistics/kxm045. URL `http://biostatistics.oxfordjournals.org/content/9/3/432.abstract`.

Geoff Gordon and Ryan Tibshirani. Subgradient method. URL `https://www.cs.cmu.edu/~ggordon/10725-F12/slides/06-sg-method.pdf`. [Online; accessed 7-November-2016].

J. C. Gower and Garmt B. Dijksterhuis. *Procrustes problems*, volume 30.;30;. Oxford University Press, Oxford;New York;, 2004. ISBN 0198510586;9780198510581;.

Keelin Greenlaw, Elena Szefer, Jinko Graham, Mary Lesperance, and Farouk S. Nathoo. A bayesian group sparse multi-task regression model for imaging genetics. October 2016. URL https://arxiv.org/pdf/1605.02234.pdf.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2013.

Alan Julian Izenman. *Modern Multivariate Statistical Techniques.* Springer, 2013.

Minjung Kyung, Jeff Gill, Malay Ghosh, and George Casella. Penalized regression, standard errors, and bayesian lassos. *Bayesian Anal.*, 5(2):369–411, 06 2010. URL http://projecteuclid.org/euclid.ba/1340218343.

Rahul Mazumder and Trevor Hastie. The graphical lasso: New insights and alternatives. *Electron. J. Statist.*, 6:2125–2149, 2012. doi: 10.1214/12-EJS740. URL http://dx.doi.org/10.1214/12-EJS740.

Yi Lin Ming Yuan. On the non-negative garrotte estimator. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 69(2):143–161, 2007. ISSN 13697412, 14679868. URL http://www.jstor.org/stable/4623260.

SAS. Proc pls: Regression methods :: Sas/stat(r) 9.22 user's guide, 2016. URL https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_pls_sect014.htm. [Online; accessed 23-November-2016].

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

M. K.-S. Tso. Reduced-rank regression and canonical analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, 43(2):183–189, 1981. ISSN 00359246. URL http://www.jstor.org/stable/2984847.

Wikipedia. Principal component analysis — wikipedia, the free encyclopedia, 2016a. URL https://en.wikipedia.org/wiki/Principal_component_analysis. [Online; accessed 16-December-2016].

Wikipedia. Subderivative — wikipedia, the free encyclopedia, 2016b. URL https://en.wikipedia.org/w/index.php?title=Subderivative&oldid=710770961. [Online; accessed 7-November-2016].

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2005.00532.x. URL http://dx.doi.org/10.1111/j.1467-9868.2005.00532.x.

Ming Yuan and Yi Lin. Model selection and estimation in the gaussian graphical model. *Biometrika*, 94(1):19–35, 2007. ISSN 00063444. URL http://www.jstor.org/stable/20441351.

Willín Álvarez and Victor J. Griffin. Estimation procedure for reduced rank regression, plssvd. *Statistics, Optimization and Information Computing*, 4(2):107–117, 2016.

Appendix A

R Code

```
################# main algorithm ###############

Algorithm_1 = function(Y,X,l1,l2,r){
  # Main algorithm function to perform Cov-SRRR
  # Input:
  # Y is nxq matrix of responses
  # X is nxp matrix of covariates
  # l1 is L1 penalty on all of the entries in the precision matrix
  # l2 is a real number that controls the row-wise sparsity in B
  # r is an integer denoting the assumed rank of the coefficient matrix
  # Output: a list contining the following elements
  # A - an rxq coefficient matrix
  # B - a  pxr factor matrix
  # O - a  qxq precision matrix
  svd_x = svd(X)
  B = svd_x$v[,1:r] %>% as.matrix(ncol = r)

  svd_results = svd(t(Y)%*%X%*%B)
  A_tilde = svd_results$u%*%t(svd_results$v)

  O = diag(nrow = ncol(Y))

  A = solve(chol(O))%*%A_tilde

  obj_4_val = obj_fun_4(Y,X,A,B,O,l1,l2)
  obj_7_val = obj_fun_7(Y,X,A_tilde,B,O,l2)




  obj_4_newval = 10000000
  obj_7_newval = 10000000
```

```
B_new        = B

max_iter = 100
iter1 = 1
iter2 = 1
iter3 = 1
obj_4_diff = 10000
while(obj_4_diff>0.001 && iter1<max_iter){
  # run DP-GLASSO to estimate precision matrix for fixed A,B
  Sigma_R = calc_Sigma_R(Y,X,A,B)
  dp_results = dpglasso(Sigma = Sigma_R, rho = l1)
  O = dp_results$X
  obj_7_diff = 10000
  while(obj_7_diff>0.001 && iter2<max_iter){
    # do SVD to estimate A for fixed B
    svd_results = svd(t(Y)%*%X%*%B)
    A_tilde = svd_results$u%*%t(svd_results$v)
    B_diff     = 10000
    while(B_diff>0.001 && iter3 < max_iter){
      # For fixed A estimate B
      for(l in 1:nrow(B)){
        # update B row by row
        sum_XB = 0
        X_l = as.matrix(X[,l])
        for(k in 1:nrow(B)){
          # calculate summation part of R_l
          if(k!=l){
            sum_XB = sum_XB + as.matrix(X[,k])%*%t(as.matrix(B[k,]))
          }
        }
        R_l = Y%*%A_tilde - sum_XB
        # check whether row of B should be set to zero
        check = 1-(l2/(2*(sqrt(sum((t(X_l)%*%R_l)^2)))))
        if(check>0){
          B_new[l,] = (1/(t(X_l)%*%X_l))%*%check%*%t(X_l)%*%R_l
        }else{
          B_new[l,] = rep(0,ncol(B))
        }
      }
      #check whether B has converged
      B_diff = sum(abs((B_new-B)))
      B = B_new
      # print(paste("iter3 =", iter3))
      iter3 = iter3+1
      if(sum(B[,1]==0)==0){
        return(list(A = A,B = B,O = O))
      }
```

```
      }
      iter3 = 1
      #set A = O^(-0.5)A_tilde
      # A = solve(chol(O))%*%A_tilde
      #check whether objective function 7 has converged
      obj_7_newval = obj_fun_7(Y,X,A_tilde,B,O,l2)
      obj_7_diff = (obj_7_val - obj_7_newval)
      obj_7_val = obj_7_newval
      # print(paste("iter2 =", iter2))
      iter2 = iter2+1
    }
    A = solve(chol(O))%*%A_tilde
    iter2 = 1
    #check whether objective function 4 has converged
    obj_4_newval = obj_fun_4(Y,X,A,B,O,l1,l2)
    obj_4_diff = (obj_4_val - obj_4_newval)
    obj_4_val = obj_4_newval
    # print(paste("iter1 =", iter1))
    iter1 = iter1+1
  }
  # Sigma_R = calc_Sigma_R(Y,X,A,B)
  # dp_results = dpglasso(Sigma = Sigma_R, rho = l1)
  # O = dp_results$X
  return(list(A = A,B = B,O = O))
}

obj_fun_4 = function(Y,X,A,B,O,l1,l2){
  # Function to evaluate objective function 4 from Chen and Huang 2016
  # Inputs:
  # Y is nxq matrix of responses
  # X is pxq matrix of covariates
  # A is qxr matrix of coefficients
  # B is pxr matrix of predictive factors
  # O is qxq precision matrix
  # l1 is L1 penalty on all of the entries in the precision matrix
  # l2 is a real number that controls the row-wise sparsity in B
  # Output:
  # negative log-likelihood value
  n = nrow(Y)
  Sigma_R = calc_Sigma_R(Y,X,A,B)
  return(sum(diag(Sigma_R%*%O)) - log(det(O)) +
           l1*sum(abs(O[row(O)!=col(O)])) + l2*sum(sqrt(B^2)))
}


obj_fun_7 = function(Y,X,A_tilde,B,O,l2){
  # Function to evaluate objective function 7 from Chen and Huang 2016
```

```
  # Inputs:
  # Y is nxq matrix of responses
  # X is pxq matrix of covariates
  # A is qxr matrix of coefficients
  # B is pxr matrix of predictive factors
  # O is qxq precision matrix
  # l2 is a real number that controls the row-wise sparsity in B
  # Output:
  # negative log-likelihood value
  root_O = chol(O)
  # A_tilde = root_O%*%A
  Y_tilde = Y%*%root_O
  n = nrow(Y)
  p = nrow(B)
  B_sum = 0
  for(j in 1:p){
    B_sum = B_sum + sqrt(sum(B[j,]^2))
  }
  result = (1/n)*sqrt(sum((Y_tilde-X%*%B%*%t(A_tilde))^2)) + l2*B_sum
  return(result)
}

calc_Sigma_R = function(Y,X,A,B){
  # Function to calculate sample covariance
  # Inputs:
  # Y is nxq matrix of responses
  # X is pxq matrix of covariates
  # A is qxr matrix of coefficients
  # B is pxr matrix of predictive factors
  # Output:
  # Sigma_r the sample covariance of the errors as determined by the model
  # Y = XBA' + E
  n = nrow(Y)
  Sigma_R = (1/n)*(t(Y-X%*%B%*%t(A))%*%(Y-X%*%B%*%t(A)))
  return(Sigma_R)
}

generate_data = function(n,p,p0,q,rho_e,seed = FALSE){
  if(is.numeric(seed)){
    set.seed(seed)
  }
  r = 3
  Sigma_x = matrix(nrow = p, ncol = p)
  Sigma_e = matrix(nrow = q, ncol = q)


  # generate Sigma_x with AR(0.5) structure
```

```
  for(i in 1:p){
    for(j in 1:p){
      Sigma_x[i,j] = 0.5^(abs(i-j))
    }
  }

  X = rmvnorm(n,sigma=Sigma_x)
  # generate Sigma_e with AR(rho_e) structure
  for(i in 1:q){
    for(j in 1:q){
      Sigma_e[i,j] = rho_e^(abs(i-j))
    }
  }

  # Generate B matrix first p0 rows from N(0,1) the rest p-p0 rows all 0
  data = c(rnorm((p0*r)),rep(0,(p-p0)*r))
  B = matrix(data, nrow = p, ncol = r,byrow=T)

  # Generate A matrix from N(0,1)
  data = rnorm((q*r))
  A = matrix(data, nrow = q, ncol = r)

  # calculate coefficient matrix from C = BA'
  C = B%*%t(A)

  # calculate correct sigma^2 factor to ensure 1-to-1 signal to noise ratio
  sigma_2 = sum(diag(t(C)%*%Sigma_x%*%C))/(sum(diag(Sigma_e)))

  # Generate error matrix
  E = rmvnorm(n,sigma = sigma_2*Sigma_e)

  # Calculate response Y=XC+E
  Y = X%*%C + E
  return(list(Y = Y, X = X, A = A, B = B))
}

do_cv = function(params,Y,X,num_folds){

  l1 = params[1]
  l2 = params[2]
  if(length(params)==3){
    r = params[3]
  }else{
    r = 3
  }

  fold_size = nrow(Y)/num_folds
```

```
  fold_err = numeric(length = num_folds)
  for(k in 1:num_folds){
    Y_in = Y[-c(((k-1)*fold_size+1):(k*fold_size)),]
    X_in = X[-c(((k-1)*fold_size+1):(k*fold_size)),]
    Y_out = Y[c(((k-1)*fold_size+1):(k*fold_size)),]
    X_out = X[c(((k-1)*fold_size+1):(k*fold_size)),]
    res = Algorithm_1(Y_in,X_in,l1,l2,r)
    fold_err[k] = pred_err(Y_out,X_out,res$A,res$B)
    # message(paste(k,"folds done!"))
  }
  mean_err = mean(fold_err)
  # print(paste("MSE:",mean_MSE))
  return(mean_err)
}
```