Parameter Selection of Sparse Functional Principal Component Analysis with fMRI data

Joo Yoon Han

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2016

Reading Committee:

Noah Simon

Andrew Zhou

Program Authorized to Offer Degree:
School of Public Health

University of Washington

Abstract

Parameter Selection of Sparse Functional Principal Component Analysis with fMRI data

Joo Yoon Han

Chair of the Supervisory Committee:
Assistant Professor, Noah Simon
Department of Biostatistics

With the advent of high throughput biotechnologies, it is increasingly common for the number of features measured on each subject to vastly exceed the number of subjects in modern biomedical studies. In this manuscript we focus on these high dimensional issues for brain imaging data.

Principal component analysis (PCA) is commonly used to reduce dimension of the data and to examine the major patterns. However, with such high dimensional data, PCA is inconsistent (Johnstone & Lu 2009). Moreover, when the underlying patterns are smooth and sparse, PCA will not be able to properly detect patterns. Sparse and smooth PCA may be of interest for high dimensional data, where the principal components are linear combinations of a subset of the features (with coefficient values that are spatially smooth). Specifically for fMRI data, where data are collected across time and regions of brain, smooth principal components can show major patterns in time. Also finding specific brain regions that are associated with the major patterns may be of interest. Allen (2013) introduced an optimization problem for this scenario, sparse and functional principal component analysis (SFPCA), which encourages both row and/or column factors are sparse and smooth.

We apply SFPCA to brain imaging data $\mathbf{X}_{n\times p}$ with n regions and p time points, where the row factors are sparse and the column factors are smooth. SFPCA problem involves three regularization parameters: sparsity parameter, smoothing parameter and number of components. The main goal of this thesis is to develop an automated method to select those regularization parameters involved in the SFPCA problem. The method is based on cross-validation; however, cross-validation with an unsupervised problem is not trivial. We leverage the time structure of brain imaging data in estimating held-out time-points in the test set. We also define the cross-validated proportion of variance explained for our problem and use it to select appropriate number of components (and regularization parameters for those components). We search for the regularization parameters sequentially, component-by-component.

We compare performance of SFPCA (with our selected tuning parameters) to classical PCA with different signal to noise ratios (SNR). For sparse and smooth data, SFPCA substantially outperforms PCA (classical PCA gives estimates that are not sparse, and much too non-smooth). As expected, when signal to noise ratio increases, SFPCA performance improves. In addition, as SNR increases, cross-validated proportion of variance explained more accurately estimates the true proportion of variance explained.

From simulation studies, we find that we need enough signal to estimate factors using SFPCA properly. Moreover, we need reasonable candidate values of the regularization parameters.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thank my advisor, Noah Simon, for the support and guidance during my time at University of Washington. I would like to thank you for teaching me being independent, encouraging my thesis work and for training me as a statistician. I am always appreciative of your advice on research and my career.

I would also like to thank my committee member, Andrew Zhou, for serving as my committee member. I also want to thank you for suggestions and comments on my thesis work.

DEDICATION

This thesis work is dedicated to my husband, Minkyu, who has been a supporter throughout graduate studies. I could not have done this without you. Thank you for your unwavering love, trust, encouragement and wisdom. I also dedicate this work to my parents, Woonwoo and Jungsil, who guided me to my dream. Without their support I may not have found myself at University of Washington. I will always appreciate unconditional love and guidance from them. I also like to thank my siblings, Jaeyoon and Keewook, for their emotional support. Finally, I like to give special thanks to my lovely dog, Theo.

Chapter 1

INTRODUCTION

With the development of technology, we are now able to collect many features on each subject — the number of features on each subject can easily exceed the number of subjects (this is particularly common for brain imaging data). With such high dimensional data, we often cannot use classical statistical techniques as their assumptions may not be satisfied.

Principal component analysis (PCA) is a commonly used technique to reduce the dimension of a data set. With PCA, we can also examine the major patterns in the data set by observation and feature. However, Johnstone & Lu (2009) show that PCA is hard to interpret and can be inconsistent with such high dimensional data. Moreover, PCA cannot take advantage of known structure in the data (eg. smoothness and sparsity).

It has been shown that functional principal component analysis (FPCA) and sparse principal component analysis (SPCA) can effectively estimate principal components in the high dimensional setting. Rice & Silverman (1991) introduced FPCA to encourage smoothness in the PCs. Zou et al. (2006) introduced SPCA, which encourages sparsity of features used in the principal components.

Sparse and smooth PCA may be of interest for high dimensional data, where the principal components are believed to be sparse and smooth. Specifically for fMRI data, where data are collected across time and in different regions of the brain, encouraging temporal smoothness of principal components may be appropriate. In addition, it may be of interest to explore specific regions that are associated with PCs, and discover how these regions differ by disease status. Allen (2013) introduced an optimization problem, sparse and functional principal component analysis (SFPCA), which encourages both row and/or column principal components are sparse and smooth.

We apply SFPCA to brain imaging data $\mathbf{X}_{n\times p}$ with n regions and p time points, focusing on the setting where the row principal components are sparse and the column loadings are smooth. SFPCA involves three regularization parameters: sparsity parameter, smoothness parameter, and number of components. The main goal of this thesis is to develop a data-driven method to select these parameters. The method is based on cross-validation; however, cross-validation with an unsupervised problem is not trivial. We leverage the time structure of brain imaging data in estimating held-out time-points in the test set. We also define the cross-validated proportion of variance explained for our problem and use it to select appropriate number of components (and regularization parameters for those components). We search for the regularization parameters sequentially, component-by-component.

In Chapter 2 we discuss various versions of principal component analysis and the optimization problem from Allen (2013). In addition, we discuss our problem, how we use cross-validation to select regularization parameters and the algorithm. In Chapter 3 we describe the data generation and settings for the simulation studies. In chapter 4, we provide the results of the simulations and plots showing the proportion of variance explained by each factors for different values of signal to noise ratio. In chapter 5, we show results from our algorithm with fMRI data and we conclude with a discussion in chapter 6.

Chapter 2

BACKGROUND

2.1 Principal component analysis

Principal component analysis (PCA) is commonly used dimension reduction technique, which preserves as much of the variation of the data set as possible (Jolliffe (2002)). PCA gives a new set of variables, principal components (PCs), which are linear combinations of all of the original variables (Berrendero et al. (2011)). The PCs are uncorrelated and selected to maximize the preserved variance.

Let $\mathbf{X}$ be a $n \times p$ data matrix, where n is the number of observations and p is the number of variables. We assume that $\mathbf{X}$ is centered. The PCs can be computed by eigenvalue decomposition (EVD) of $\mathbf{X}^T\mathbf{X}$. Jolliffe (2002) shows steps to find PCs by EVD of $\mathbf{X}^T\mathbf{X}$. Suppose that $\mathbf{x}$=$(x_1, x_2, \ldots, x_p)$ is a p-vector of features of $\mathbf{X}$. The first PC, $\boldsymbol{\alpha}_1^T\mathbf{x}$, can be found by

$$\begin{aligned} &\text{maximize} \quad \boldsymbol{\alpha}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\alpha}_1 \\ &\text{subject to} \quad \boldsymbol{\alpha}_1^T\boldsymbol{\alpha}_1 = 1. \end{aligned} \tag{2.1}$$

Problem (2.1) is equivalent to

$$\text{maximize} \quad \boldsymbol{\alpha}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\alpha}_1 - \lambda(\boldsymbol{\alpha}_1^T\boldsymbol{\alpha}_1 - 1) \tag{2.2}$$

where λ is a Lagrange multiplier. After differentiating by $\boldsymbol{\alpha}_1$, we now solve $(\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I})\boldsymbol{\alpha}_1 = 0$. We want to maximize $\boldsymbol{\alpha}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\alpha}_1 - \lambda(\boldsymbol{\alpha}_1^T\boldsymbol{\alpha}_1 - 1)$ and $\lambda\mathbf{I} = \mathbf{X}^T\mathbf{X}$. Thus, λ is the largest eigenvalue of $\mathbf{X}^T\mathbf{X}$ and $\boldsymbol{\alpha}_1$ is the corresponding eigenvector. We can find the 2^{nd} PC by solving (2.1) with $\boldsymbol{\alpha}_2$ instead and with a constraint that $\text{cov}(\boldsymbol{\alpha}_2^T\mathbf{x}, \boldsymbol{\alpha}_1^T\mathbf{x}) = 0$. We can again use Lagrange multipliers and differentiate with respect to $\boldsymbol{\alpha}_2$, we find that $\boldsymbol{\alpha}_2$ is the

eigenvector corresponding to the 2^{nd} largest eigenvalue of $\mathbf{X}^T\mathbf{X}$.

The principal components also can be found by singular value decomposition (SVD) of $\mathbf{X}$. Let $\mathbf{X}= \mathbf{UDV}^T$, where $\mathbf{U}$ is a matrix that columns are the left singular vectors of $\mathbf{X}$, $\mathbf{D}$ is a diagonal matrix with the singular values of $\mathbf{X}$, and $\mathbf{V}$ is a matrix that columns are the right singular vectors of $\mathbf{X}$. PCs are $\mathbf{UD}$ and the k^{th} PC's loadings are $\mathbf{v}_k$.

2.2 Sparse Principal component analysis

PCA is a useful tool to reduce the dimension of the data set. Also we can observe patterns in the data by plotting the first few columns of $\mathbf{U}$ and $\mathbf{V}$. However with high dimensional data, it is difficult to interpret the PCs — each PC includes all p features. Zou et al. (2006) introduced Sparse PCA (SPCA), where loadings for the principal components are sparse. With SPCA, the PCs are linear combinations of selected features instead of all p features. Zou et al. (2006) discussed how PCA problem can be formulated as a regression problem. Zou et al. (2006) used the *lasso* to solve the SPCA problem.

2.3 Functional Principal component analysis

When we have data that are curves, classical PCA is not a useful tool to examine the major patterns of the data set. Rice & Silverman (1991) introduced Functional PCA (FPCA). FPCA encourages smoothness of the spatial loadings — in particularRice & Silverman (1991) uses a second difference roughness penalty by adding a penalty term $\mathbf{v}^T\mathbf{F}^T\mathbf{F}\mathbf{v}$, where $\mathbf{F}^T\mathbf{F}$ is a smoothing matrix based on the second-differencing operator $\boldsymbol{F}$.

2.4 The idea of Allen et al

Our thesis is based on an optimization problem of Allen (2013). Allen (2013) propose to combine SPCA and FPCA to allow for both sparsity and smoothness of PCs/loadings: The SFPCA proposal by Allen (2013) generalizes the different approaches of SPCA and FPCA.

The optimization problem proposed by Allen (2013) is

$$\begin{aligned} \underset{\mathbf{u},\mathbf{v}}{\text{maximize}} \quad & \mathbf{u}^T\mathbf{X}\mathbf{v} - \lambda_{\mathbf{u}}P_{\mathbf{u}}(\mathbf{u}) - \lambda_{\mathbf{v}}P_{\mathbf{v}}(\mathbf{v}) \\ \text{subject to} \quad & \mathbf{u}^T(\mathbf{I} + \alpha_{\mathbf{u}}\boldsymbol{\Omega}_{\boldsymbol{u}})\mathbf{u} \leq 1 \\ & \mathbf{v}^T(\mathbf{I} + \alpha_{\mathbf{v}}\boldsymbol{\Omega}_{\boldsymbol{v}})\mathbf{v} \leq 1. \end{aligned} \tag{2.3}$$

where $\lambda_{\mathbf{u}}, \lambda_{\mathbf{v}} \geq 0$ are sparsity penalty parameters for $\mathbf{u}$ and $\mathbf{v}$, $P_{\mathbf{u}}(), P_{\mathbf{v}}()$ are sparsity penalties, $\alpha_u, \alpha_v \geq 0$ are smoothing penalty parameters for $\mathbf{u}$ and $\mathbf{v}$ with the symmetrized second differences matrices $\boldsymbol{\Omega}_{\boldsymbol{u}} = \mathbf{F}_u^T\mathbf{F}_u$ and $\boldsymbol{\Omega}_{\boldsymbol{v}} = \mathbf{F}_v^T\mathbf{F}_v$. (2.3) is a bi-concave problem in $\mathbf{u}$ and $\mathbf{v}$: With fixed $\mathbf{v}$, (2.3) is concave in $\mathbf{u}$ and for fixed $\mathbf{u}$, (2.3) is concave in $\mathbf{v}$.

Allen (2013) adopted an alternating algorithm introduced by Huang et al. (2009) for choosing penalty parameters. Huang et al. (2009) suggested to update for $\mathbf{u}$ with fixed $\mathbf{v}$ and find the optimal penalty parameter for $\mathbf{u}$. Similarly, for fixed $\mathbf{u}$, update $\mathbf{v}$ and find the optimal penalty parameter for $\mathbf{v}$ (Huang et al. 2009). With an alternating algorithm, Allen (2013) used BIC method to select the appropriate regularization parameters.

2.5 Application of Sparse Functional PCA

SFPCA is appropriate for brain imaging where data is collected in multiple brain regions over time. For example, a functional MRI (fMRI) samples blood-oxygen-level dependent (BOLD) signal every few seconds. Since the signal is sampled so frequently, and BOLD doesn't change *that* quickly it is sensible to model it as smooth in time.

Moreover, it may be of interest to find brain regions that drive the patterns in the data and study how these regions differ by disease groups (Lillemark et al. 2014) or related to certain tasks (Ye et al. 2016). For example, diagnosis of Alzheimer's disease in early stage is clinically important. However, current diagnostics of Alzheimer's disease have poor sensitivity and specificity (Huang et al. 2011). Thus, investigating brain regions that may be related to Alzheimer's disease can assist development of early diagnosis tools for Alzheimer's disease.

With classical PCA, the PCs we find are not smooth. With fMRI data, non-smooth PC factors might be inappropriate because of the structure of fMRI data. Moreover, classical PCA will find the factors that are linear combinations of all brain regions. Those patterns will likely include many irrelevant brain regions.

2.6 Our problem

As stated in the section 2.5, classical PCA finds PCs that are not sparse or smooth. For certain studies, it may be of interest to find the PC factors that are sparse and smooth. With SFPCA, we can find PC factors that are sparse and smooth. SFPCA requires choosing multiple regularization parameters. Our goal is to develop a data driven method to select the regularization parameters in SFPCA with brain imaging data.

We apply SFPCA on the brain imaging data $\mathbf{X}_{n\times p}$ with n brain regions and p time points. In contrast to Allen (2013)'s work, we encourage sparsity on the region (row) factors only and smoothness on the time (column) factors only.

We estimate $\hat{\mathbf{X}} = \sum_{k=1}^{R} \hat{d}_k \hat{\mathbf{u}}_k \hat{\mathbf{v}}_k{}^T$, where $\hat{d}_k, \hat{\mathbf{u}}_k, \hat{\mathbf{v}}_k$ minimize $\| \mathbf{X} - \hat{\mathbf{X}} \|_F^2$ subject to smooth penalty on $\mathbf{v}_k$ and sparse penalty of $\mathbf{u}_k$. We use the roughening matrix $\mathbf{\Omega} = \mathbf{F}^T\mathbf{F}$ with a second-differencing operator $\boldsymbol{F}$ for smoothness of $\mathbf{v}_k$. We use l_1 penalty to encourage sparsity of $\mathbf{u}_k$. We solve the following optimization problem

$$\begin{aligned} \underset{\mathbf{u},\mathbf{v}}{\text{maximize}} \quad & \mathbf{u}^T\mathbf{X}\mathbf{v} - \lambda|\mathbf{u}| \\ \text{subject to} \quad & \mathbf{u}^T\mathbf{u} \leq 1 \\ & \mathbf{v}^T(\mathbf{I} + \alpha\mathbf{\Omega})\mathbf{v} \leq 1. \end{aligned} \tag{2.4}$$

(2.4) is a bi-concave problem with three parameters to select. The main goal of this thesis is to develop a method to select these various parameters.

2.6.1 The regularization parameters

SFPCA is an unsupervised problem with multiple regularization parameters. Our problem involves three regularization parameters:

- Smoothing parameter: $\boldsymbol{\alpha}$
- Sparsity parameter: $\boldsymbol{\lambda}$
- Number of components: R

We need to develop a data-driven method to choose those regularization parameters.

2.6.2 Cross-validation with unsupervised problem

Cross-validation (CV) is one of the most popular methods for selecting the regularization parameters. It chooses the parameters with the minimum cross-validated prediction error. When we use CV, we divide the data into test data and train data then fit model with train data and evaluate using test data (this is done multiple times, over different folds, and averaged).

For a supervised problem, we fit the model using $\mathbf{Y}_{\text{train}}$ and $\mathbf{X}_{\text{train}}$, where $\mathbf{Y}$ is the outcome and $\mathbf{X}$ is the predictor. Then we evaluate the model on the test data. The regularization parameters are chosen by minimizing test prediction error calculated with $\mathbf{Y}_{\text{test}}$ and $\mathbf{X}_{\text{test}}$. However, in unsupervised problems, there is no specified "outcome" $\mathbf{Y}$ to predict. Therefore, it is not that trivial to use CV with unsupervised learning problems.

2.6.3 Previous work of CV with unsupervised problem

Perry (2009) introduced how to use CV for unsupervised problems, specifically PCA problems. Perry (2009) gave two approaches: a Wold-style approach and a Gabriel-style approach.

Wold-style

Wold-style removes randomly chosen entries of the data matrix $\mathbf{X}$ and treats estimation of those entries as a missing data problem. Let $\mathbf{S}$ be a set of indices of left-in elements of $\mathbf{X}$ and $\mathbf{S}^c$ be a set of indices of left-out elements of $\mathbf{X}$. Therefore, $\mathbf{X_S}$ indicates a set of left-in elements, a training set, and $\mathbf{X_{S^c}}$ is a test set with left-out elements of $\mathbf{X}$. The problem of Perry (2009) can be written as

$$minimize \sum_{i,j \in \mathbf{S}} (\mathbf{X}_{ij} - \boldsymbol{\mu}_{ij})^2 \text{ where } \boldsymbol{\mu} = \hat{\mathbf{d}}\hat{\mathbf{u}}\hat{\mathbf{v}}^T$$

Perry (2009) suggested using an expectation-maximization (EM) algorithm to estimate PCs on $\mathbf{X_S}$.

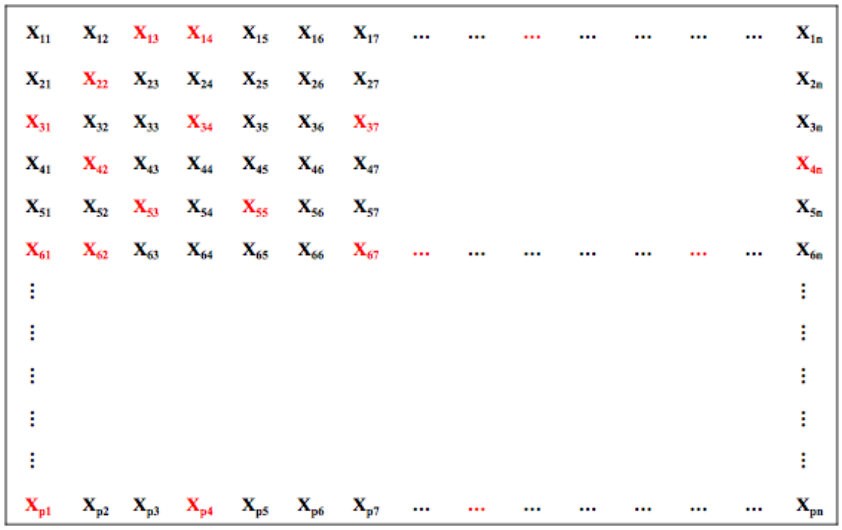
Gabriel-style

The Gabriel-style CV leaves out random blocks of data by taking a subset of columns of $\mathbf{X}$ as the outcome columns and the rest as the predictor columns. Similarly, by taking a subset of the rows of $\mathbf{X}$ as test rows and the rest as the training rows, we can partition the data matrix $\mathbf{X}$ as $\mathbf{outcome_{S^c}}$, $\mathbf{outcome_S}$, $\mathbf{predictor_{S^c}}$, and $\mathbf{predictor_S}$.

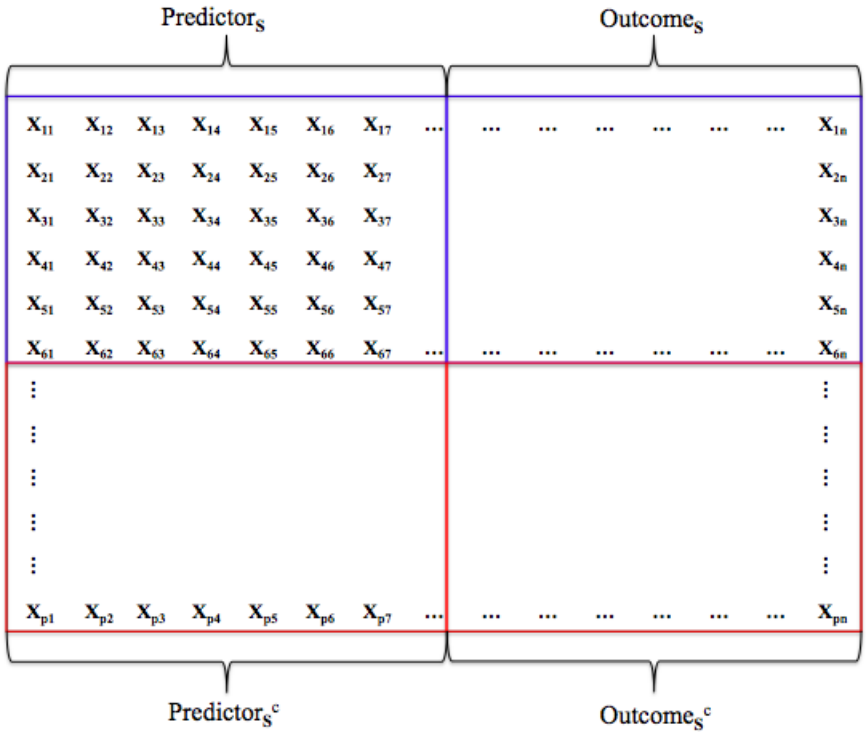
First we fit an SVD to $\mathbf{predictor_S}$ and get the principal component scores. Then using the principal component scores, we model $\mathbf{outcome_S}$ as a linear function of the component scores. Similarly, we can get the principal components score for $\mathbf{predictor_{S^c}}$. Using this score and the linear combination, we estimate $\mathbf{outcome_{S^c}}$ as a linear combination of the scores to get predictions. With $\mathbf{outcome_{S^c}}$ and predicted value, we calculate prediction error.

- Wold-style: leaves out random elements of $\mathbf{X}$.

- Gabriel-style: leaves out random blocks of data to partition the data into four subsets



(a)



(b)

Figure 2.1: (a) An example of Wold-style left-out entries for cross-validation with unsupervised problems. Elements written in red indicate left-out elements of $\mathbf{X}$. (b) An example of Gabriel-style left-out entries for cross-validation with unsupervised problems. Blue boxes indicate leave in sets while red boxes indicate leave out sets.

Figure 2.1 shows an example of Wold-style and Gabriel-style. Figure 2.1(a) illustrates Wold-style, with elements of $\mathbf{X}$ written in red which indicate randomly chosen left-out elements of $\mathbf{X}$. Figure 2.1(b) explains Gabriel-style, with two different colors of boxes: two blue boxes indicate left-in sets $\mathbf{X_S}$ while two red boxes indicate left-out sets $\mathbf{X_S^c}$. Gabriel-style divide the data set into train and test set as well as into predictor and outcome set. Note that we do not need to partition the data matrix into four sets as shown in Figure 2.1(b). The example shown in Figure 2.1(b) is a simple illustration of Gabriel-style.

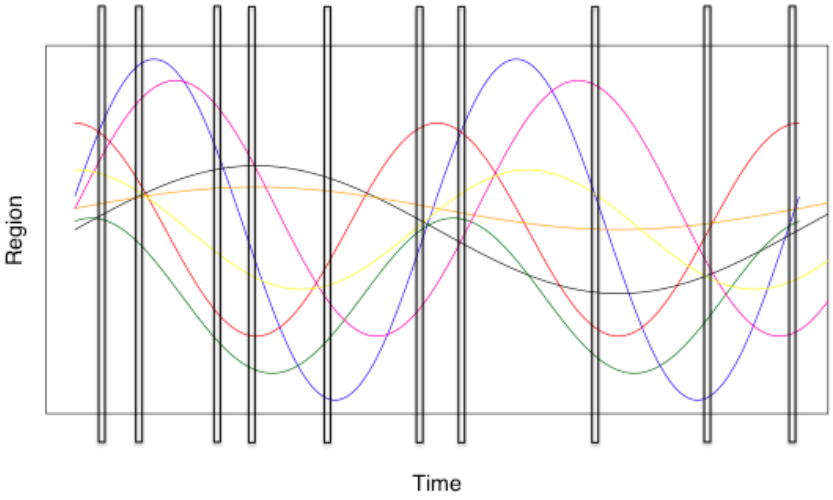
2.6.4 How we do CV

We do cross-validation similar to the Wold-style method with the time structure of the data set. The difference is that we remove time slices of $\mathbf{X}$ instead of random elements of $\mathbf{X}$. We cannot remove randomly selected columns with general data set, where we do not have a time structure within the data set. Note that the fMRI data set $\mathbf{X}$, consists BOLD signal of n brain regions and p time points. We have temporal structure and smoothness in columns in $\mathbf{X}$. We apply the temporal structure and smoothness of the data when we do CV to choose the regularization parameters. We remove randomly selected time columns and partition $\mathbf{X}$ into a training and a test set.

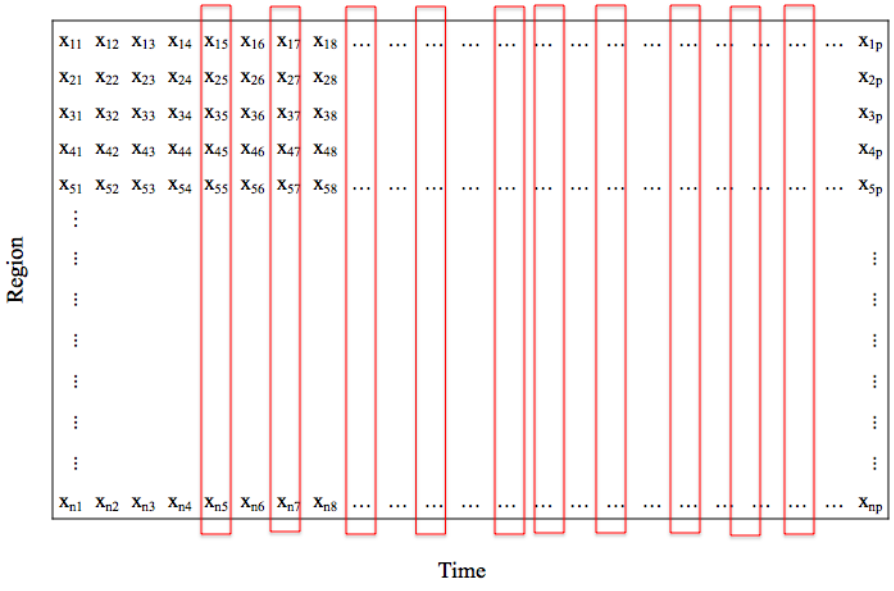
Figure 2.2(a) shows data $\mathbf{X}$ by time and brain regions. Ten boxes represent the randomly selected ten columns to be removed as a test set. Figure 2.2(b) explain how we remove these ten columns from the data matrix $\mathbf{X}$. Let $\mathbf{S}^c$ be a set of indices of left-out columns of $\mathbf{X}$. Then $\mathbf{X_S}$ indicates a set of left-in columns, a training set, and $\mathbf{X_{S^c}}$ is a test set with left-out columns of $\mathbf{X}$. $\mathbf{v_S}$ indicates corresponding elements of $\mathbf{v}$. Then we can re-write our problem (2.4) as

$$\begin{aligned} \underset{\mathbf{u},\mathbf{v}}{\text{maximize}} \quad & \mathbf{u}^T\mathbf{X_S}\mathbf{v_S} - \lambda|\mathbf{u}| \\ \text{subject to} \quad & \mathbf{u}^T\mathbf{u} \leq 1 \\ & \mathbf{v}^T(\mathbf{I} + \alpha\mathbf{\Omega})\mathbf{v} \leq 1. \end{aligned} \tag{2.5}$$

We use training set $\mathbf{X_S}$ and the vector with corresponding elements $\mathbf{v_S}$ in the objective



(a)



(b)

Figure 2.2: (a) Plot of the data in time and brain region. Ten black boxes are randomly selected left-out columns that we treat as a test set.(b) Example of removing randomly chosen ten columns from the data set.

function. However, we use all elements of $\mathbf{v}$ in the constraint to account for the smoothness of $\mathbf{v}$. With this constraint, we estimate left-out elements of $\mathbf{v}$ with given smooth structure of $\mathbf{v}$.

We solve the optimization problem (2.5) with k-folds. We find the regularization parameters with minimum $\| \mathbf{X}^c_{\mathbf{S}} - \hat{\mathbf{X}}^c_{\mathbf{S}} \|^2_F$. More details are given in sections 2.6.6 and 2.6.7.

2.6.5 Bi-concave problem

As mentioned in section 2.4, the optimization problem (2.3) is a bi-concave problem in $\mathbf{u}$ and $\mathbf{v}$. Our problem (2.4) is also a bi-concave problem. We solve problem (2.4) with general index set $\mathbf{A}$. With fixed $\mathbf{u}$, we solve

$$\begin{aligned} \underset{\mathbf{v}}{\text{maximize}} \quad & (\mathbf{u}^T \mathbf{X}_{\mathbf{A}}) \mathbf{v}_{\mathbf{A}} \\ \text{subject to} \quad & \mathbf{v}^T (\mathbf{I} + \alpha \mathbf{\Omega}) \mathbf{v} \leq 1. \end{aligned} \tag{2.6}$$

where $\mathbf{A}$ is an index set. $\mathbf{A}$ can be either $\mathbf{I}$ or $\mathbf{S}$. We use $\mathbf{S}$, when we do cross-validation to search for the parameters. We use $\mathbf{I}$ when we find $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$ and $\hat{\mathbf{d}}$ with chosen regularization parameters. This problem has a concave objective with quadratic convex inequality constraints in $\mathbf{v}$. Thus, this problem is a concave problem in $\mathbf{v}$ for fixed $\mathbf{u}$.

With fixed $\mathbf{v}$, we solve

$$\begin{aligned} \underset{\mathbf{u}}{\text{maximize}} \quad & \mathbf{u}^T (\mathbf{X}_{\mathbf{A}} \mathbf{v}_{\mathbf{A}}) - \lambda |\mathbf{u}| \\ \text{subject to} \quad & \mathbf{u}^T \mathbf{u} \leq 1 \end{aligned} \tag{2.7}$$

Problem (2.7) has a concave objective with quadratic convex inequality constraints in $\mathbf{u}$. Thus, this problem is a concave problem in $\mathbf{u}$ for fixed $\mathbf{v}$.

Algorithm 1 show steps to find $\hat{\mathbf{v}}$ with fixed $\mathbf{u}$ and $\hat{\mathbf{u}}$ with fixed $\mathbf{v}$. The stopping criterions used in Algorithm 1 is $\| \hat{\mathbf{v}}_{new} - \hat{\mathbf{v}}_{old} \|_1 < \epsilon$ and $\| \hat{\mathbf{u}}_{new} - \hat{\mathbf{u}}_{old} \|_1 < \epsilon$. We use CVXPY package with *SCS* solver to find $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$. We find $\hat{\mathbf{u}}$ with fixed $\mathbf{v}$. With estimated $\hat{\mathbf{u}}$, we find $\hat{\mathbf{v}}$.

To solve (2.4), we iteratively alternate between steps 1 and 2 of Algorithm 1, until overall convergence of both.

Algorithm 1 Problem (2.6) with fixed $\mathbf{u}$ and problem (2.7) with fixed $\mathbf{v}$

given $\mathbf{X}_{n,p}$, $\alpha > 0, \lambda > 0$ and $\mathbf{A}$
Initialize $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ to the first left/right singular vector of $\mathbf{X}$.
repeat until stopping criteria are satisfied

1. $\hat{\mathbf{u}} \leftarrow \underset{\mathbf{u}}{\text{argmax}} \ \mathbf{u}^T(\mathbf{X_A}\hat{\mathbf{v}}_\mathbf{A}) - \lambda|\mathbf{u}|$
   subject to $\mathbf{u}^T\mathbf{u} \leq 1$

2. $\hat{\mathbf{v}} \leftarrow \underset{\mathbf{v}}{\text{argmax}} \ (\hat{\mathbf{u}}^T\mathbf{X_A})\mathbf{v_A}$
   subject to $\mathbf{v}^T(\mathbf{I} + \alpha\boldsymbol{\Omega})\mathbf{v} \leq 1$

2.6.6 Searching for the parameters

For given sets of candidate R, $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ values, we evaluate how close $\hat{\mathbf{X}}_{\mathbf{S}^c}(\boldsymbol{\alpha},\ \boldsymbol{\lambda},\ R)$ is to $\mathbf{X}_{\mathbf{S}^c}$. We attempt to choose the candidate values that minimize this distance (though for computation reasons we may not get the exact optimal set). After finding parameters for a given component, we use Hotelling's deflation (Saad 1988) to "deflate" our $\mathbf{X}$ matrix before repeating our strategy to find the next component. This is explained in more detail in Section 2.6.7.

We estimate $\hat{\mathbf{X}}_{\mathbf{S}^c}(\boldsymbol{\alpha},\ \boldsymbol{\lambda},\ R) = \sum_{i=1}^{R} \hat{d}_i \hat{\mathbf{u}}_{i,\boldsymbol{\lambda}} \hat{\mathbf{v}}_{i,\boldsymbol{\alpha}}^T$, where $\hat{\mathbf{u}}_{i,\boldsymbol{\lambda}}$ and $\hat{\mathbf{v}}_{i,\boldsymbol{\alpha}}^T$ are the solutions to (2.6) and (2.7) and $\hat{d}_i = \hat{\mathbf{u}}_{i,\boldsymbol{\lambda}}^T \mathbf{X} \hat{\mathbf{v}}_{i,\boldsymbol{\alpha}}$. We attempt to find $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ that minimize $\| \mathbf{X}_{\mathbf{S}^c} - \hat{\mathbf{X}}_{\mathbf{S}^c} \|_F^2$.

Instead of searching for all combinations of candidates of $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$, which can be computationally intensive, we use a search strategy (which we detail in 2.6.7) to find one parameter at a time for each component. We are not jointly maximizing the optimization problem (2.5) with all combinations of candidates of the parameters.

For the number of components R, we use cross-validated proportion of variance explained (CVPVE) as a stopping criteria. CVPVE is explained in 2.6.8.

2.6.7 The sequential search

We search for the optimal values of $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$, and R using a sequential search. For each rank, we begin with a suitably "deflated" design matrix; we find the rank-1 approximation of our

deflated $\mathbf{X}$; and then further deflate by removing our rank-1 approximation.

For the rank-r approximation, we use Hotelling's deflation (Saad 1988) to search for the values of α and λ for rank-r approximation after we select the values of α and λ for rank-*1* approximation to rank-(r-*1*) approximation. We define $\tilde{\mathbf{X}}^r_{\mathbf{S}^c} = \mathbf{X}_{\mathbf{S}^c} - \sum_{i=1}^{r} \hat{\mathbf{X}}^i_{\mathbf{S}^c}$ and $\tilde{\mathbf{X}}^0_{\mathbf{S}^c} = \mathbf{X}_{\mathbf{S}^c}$, where $\hat{\mathbf{X}}^r_{\alpha,\lambda} = \hat{\mathrm{d}}_r \hat{\mathbf{u}}_r \hat{\mathbf{v}}_r^T$ is the rank-r approximation of $\mathbf{X}$ from SFPCA for given α and λ. Then we calculate $\| \mathbf{X}_{\mathbf{S}^c} - \sum_{i=1}^{r-1} \hat{\mathbf{X}}^i_{\mathbf{S}^c} - \hat{\mathbf{X}}^r_{\mathbf{S}^c} \|^2_{F,k} = \| \tilde{\mathbf{X}}^r_{\mathbf{S}^c} \|^2_{F,k}$ for each fold and given values of α and λ. Then we select the values of α and λ with minimum average of $\| \tilde{\mathbf{X}}^r_{\mathbf{S}^c} \|^2_{F,k}$ over k-fold.

2.6.8 Cross-validated proportion of variance explained

For each new component, we would like to estimate the additional proportion of variance explained by that component. In order to do this we need to evaluate the cumulative proportion of variance explained by all components up to that point. We discuss how to do this by cross-validation.

As said earlier, we divide $\mathbf{X}$ into $\mathbf{X}_{\mathbf{S}}$ and $\mathbf{X}^c_{\mathbf{S}}$. For each fold k, we calculate the cumulative version proportion of variance explained using the test set $\mathbf{X}^c_{\mathbf{S}}$ and average over k to get the cross-validated proportion of variance explained (CVPVE).

The proportion of variance explained can be defined as a ratio of variance explained to the total variance remaining. The variance explained by the rth component can by found by subtracting the residual variance from the total variance remaining. Similar to a regression problem, the residual is the difference between the predicted data matrix and the actual data matrix $\mathbf{X}$, what we refer to as the deflated $\mathbf{X}$ here.

The left-out total variance remaining for the *1*st component is $\| \mathbf{X}_{\mathbf{S}^c} \|^2_F$. For the rth component, we have to remove the previous r-1 components: $\tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c} \equiv \mathbf{X}_{\mathbf{S}^c} - \sum_{i=1}^{r-1} \hat{\mathbf{X}}^i_{\mathbf{S}^c}$. The variance remaining for the rth component is then $\| \tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c} \|^2_F$.

We can decompose the proportion of variance explained by the rth component as

(prop var remaining after r-1 component) $\times$ (prop of remaining var explained by component r) (2.8)

The proportion of remaining variance explained by component r can be approximated by $\frac{\| \tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c} \|^2_F - \| \tilde{\mathbf{X}}^{r}_{\mathbf{S}^c} \|^2_F}{\| \tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c} \|^2_F}$. Thus is we let $f(r)$ denote the CV proportion of variance explained (CVPVE) by the rth component (with $f(0) \equiv 0$), then we can calculate it by:

$$f(r) = \sum_{i=1}^{k} (1 - \sum_{j=0}^{r-1} f(j)) \times \frac{\| \tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c_i} \|^2_F - \| \tilde{\mathbf{X}}^{r}_{\mathbf{S}^c_i} \|^2_F}{\| \tilde{\mathbf{X}}^{r-1}_{\mathbf{S}^c_i} \|^2_F}$$

This is just formalizing (2.8), as $(1 - \sum_{j=0}^{r-1} f(j)) =$ (prop var remaining after r-1 component). We will refer to this quantity as the "cross-validated proportion of remaining variance" (CVPRV). We select the number of components by iterating our algorithm until CVPVE or CVPRV is smaller than 0.05.

2.7 Implementation

2.7.1 Selecting regularization parameters

A Two-dimensional grid search for λ and α would be very computationally intensive. We search for the smoothing parameter α first. With the selected smoothing parameter, we search for the sparsity parameter λ.

We use smoothness of time columns of brain imaging data for estimating the appropriate penalty parameters. We remove t randomly chosen time columns from the data matrix. To correctly account for the smoothness of time structure within the brain imaging data, we do not remove the first and the last five columns of the data matrix. We only remove the columns that are within 6^{th} and (p-6)th column. Let $\mathbf{S}$ be a set of indices of left-in columns and $\mathbf{S}^c$ be a set of indices of left-out columns of $\mathbf{X}$. Then $\mathbf{X}_{\mathbf{S}}$ is a training set and $\mathbf{X}_{\mathbf{S}^c}$ is a test set in our study. Similarly, $\mathbf{v}_{\mathbf{S}}$ indicates corresponding elements of $\mathbf{v}$.

α, the smoothing parameter

We search for the optimal value for α by solving the optimization problem (2.6) with fixed $\mathbf{u}$ and $\mathbf{X_S}$, a training set. We use $\mathbf{u} = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$, a unit norm vector here. This specific choice of $\mathbf{u}$ means that we are estimating smoothness by finding the optimal way to uniformly smooth each brain region.

The optimization problem for $\mathbf{v}$ with all candidates of α given can be written as:

$$\begin{aligned} \underset{\mathbf{v}}{\text{maximize}} \quad & \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}^T \mathbf{X_S v_S} \\ \text{subject to} \quad & \mathbf{v}^T(\mathbf{I} + \alpha\mathbf{\Omega})\mathbf{v} \leq 1. \end{aligned} \tag{2.9}$$

This is a concave problem. For all candidate α-values, we calculate $\text{SSE}_{k,\alpha} = \| \mathbf{X_{S^c}} \text{-} \hat{\mathbf{X}}_{\mathbf{S^c}} \|_F^2$ where $\hat{\mathbf{X}}_{\mathbf{S^c}} = (\frac{1}{\sqrt{n}} \hat{d} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \hat{\mathbf{v}}^T)_{\mathbf{S^c}}$ and $\hat{d} = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}^T \mathbf{X}\hat{\mathbf{v}}$. We repeat this k times and calculate the average of $\text{SSE}_{k,\alpha}$ over k. We select α with the minimum average of $\text{SSE}_{k,\alpha}$.

λ, the sparsity parameter

We use the value of α chosen in the previous section. For several given candidate λ-values we solve (2.5) to get estimates $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ which we can use to calculate CV-error. We choose the λ-value with the smallest cross-validated SSE. To solve (2.5) we apply the algorithm discussed in section 2.6.5 with $A = \mathbf{S}$.

With $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$, we calculate $\text{SSE}_{k,\lambda} = \left\| \mathbf{X_{S^c}} - \hat{\mathbf{X}}_{\mathbf{S^c}} \right\|_F^2$ for every candidates of λ, where $\hat{\mathbf{X}}_{\mathbf{S^c}} = (\hat{\mathbf{d}}\hat{\mathbf{u}}\hat{\mathbf{v}}^T)_{\mathbf{S^c}}$ and $\hat{d} = \hat{\mathbf{u}}^T\mathbf{X}\hat{\mathbf{v}}$. We repeat solving (2.9) and (2.10) k times and for each of the k-folds and λ. We choose λ with minimum average of $\text{SSE}_{k,\lambda}$. CVPVE is calculated by

the average of k cumulative version of PVE. With the chosen regularization parameters, we finally solve (2.4) to obtain $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ and the CVPVE for the chosen regularization parameters is reported.

2.7.2 The algorithm

Algorithm 2 SFPCA Algorithm

I. Cross-validation for α

given $\mathbf{X}_{n,p}$, $\mathbf{a} \succeq 0$: a vector of potential values of α, $\mathbf{\Omega} \succeq 0$, k, t

1. Initialize $\mathbf{v}$ to the first right singular vector of $\mathbf{X}$.
2. Repeat k times:
   (a) Randomly select t columns between 6 and p-6 to remove. Let $\mathbf{S}$ be a set of indices of leave in columns.
   (b) For all $\alpha \in \mathbf{a}$ solve (2.9) and calculate $\text{SSE}_{k,\alpha} = \parallel \mathbf{X}_{\mathbf{S}^c} - \hat{\mathbf{X}}_{\mathbf{S}^c} \parallel_F^2$
   where $\hat{\mathbf{X}}_{\mathbf{S}^c} = (\frac{1}{\sqrt{n}} \hat{d} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \hat{\mathbf{v}}^T)_{\mathbf{S}^c}$
3. For all $\alpha \in \mathbf{a}$, calculate $\overline{\text{SSE}}_\alpha = \frac{1}{k} \sum_{i=1}^k \text{SSE}_{i,\alpha}$.
4. Choose α with minimum $\overline{\text{SSE}}_\alpha$.
5. Return $\hat{\alpha}$ and $\mathbf{S}$.

II. Cross-validation for λ

given $\mathbf{X}_{n,p}$, $\mathbf{\Lambda} \succeq 0$: a vector of the potential values of λ, k, $\hat{\alpha}$ and $\mathbf{S}$.
Initialize $\mathbf{u}$ and $\mathbf{v}$ to the first left/right singular vector of $\mathbf{X}$.

2. Repeat k times:
   (b) For all $\lambda \in \mathbf{\Lambda}$ solve (2.7). Return $\hat{\mathbf{u}}$.
   (c) For given $\hat{\alpha}$ and $\hat{\mathbf{u}}$, estimate $\hat{\mathbf{v}}$ by solving (2.6)
   (d) Calculate $\text{SSE}_{k,\lambda} = \parallel \mathbf{X}_{\mathbf{S}^c} - \hat{\mathbf{X}}_{\mathbf{S}^c} \parallel_F^2$ where $\hat{\mathbf{X}}_{\mathbf{S}^c} = (\hat{d}\hat{\mathbf{u}}\hat{\mathbf{v}}^T)_{\mathbf{S}^c}$ and $\hat{d} = \hat{\mathbf{u}}^T \mathbf{X} \hat{\mathbf{v}}$.
   (e) Calculate $\text{PVE}_{k,\lambda} = \frac{\text{SSE}_{k,\lambda}}{\|\mathbf{X}_{\mathbf{S}^c}\|_F^2}$.
3. For all $\lambda \in \mathbf{\Lambda}$, calculate $\overline{\text{SSE}}_\lambda = \frac{1}{k} \sum_{i=1}^k \text{SSE}_{i,\lambda}$ and $\text{CVPVE}_\lambda = \frac{1}{k} \sum_{i=1}^k \text{PVE}_{i,\lambda}$.
4. Choose $\hat{\lambda}$ with minimum $\overline{\text{SSE}}_\lambda$.
5. Return $\hat{\lambda}$,$\hat{\alpha}$, and $\text{CVPVE}_{\hat{\lambda}}$.

III. Estimate $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$

given $\mathbf{X}_{n,p}$, $\hat{\lambda}$, $\hat{\alpha}$, $\mathbf{\Omega} \succeq 0$

1. Initialize $\mathbf{u}$ and $\mathbf{v}$ to the first left/right singular vector of $\mathbf{X}$.
2. Repeat until convergence:
   (a) For given $\hat{\lambda}$, estimate $\hat{\mathbf{u}}$ by solving: $\underset{\mathbf{u}}{\text{maximize}}\ \mathbf{u}^T \mathbf{X} \mathbf{v} - \hat{\lambda}\ |\mathbf{u}|$.
   subject to $\mathbf{u}^T \mathbf{u} \leq 1$
   (b) For given $\hat{\alpha}$, estimate $\hat{\mathbf{v}}$ by solving: $\underset{\mathbf{v}}{\text{maximize}}\ \hat{\mathbf{u}}^T \mathbf{X} \mathbf{v}$
   subject to $\mathbf{v}^T (\mathbf{I} + \hat{\alpha} \mathbf{\Omega}) \mathbf{v} \leq 1$
3. Return $\text{CVPVE}_{\hat{\lambda}}, \hat{\mathbf{u}}, \hat{\mathbf{v}}$, and $\hat{d} = \hat{\mathbf{u}}^T \mathbf{X} \hat{\mathbf{v}}$.

Chapter 3

SIMULATION STUDY

3.1 Data generation

The data matrix $\mathbf{X}_{n\times p}$ is generated by combinations of three vectors: $\mathbf{d}$, $\mathbf{u}$, and $\mathbf{v}$, and a noise matrix. We generate $\mathbf{X}=\sum_{i=1}^{I}\mathbf{d}_i\mathbf{u}_i\mathbf{v}_i^T+\epsilon$. For our simulation, we use n=100, p=200 and I=3.

$\mathbf{v}_i$ is a vector of length 200 and generated as a smooth signal from a sinusoidal curve with fixed number of cycles. $\mathbf{v}_1$ is a sinusoidal curve with 2 cycles, $\mathbf{v}_1$ is a sinusoidal curve with 4 cycles, and $\mathbf{v}_1$ is a sinusoidal curve with 6 cycles.

$\mathbf{u}_i$ is a vector of length 100 and generated from the left singular vector of a random Gaussian matrix with identity covariance structure. $\mathbf{u}_1$ is the first left singular vector, $\mathbf{u}_2$ is the second left singular vector, and $\mathbf{u}_3$ is the third left singular vector. We control sparsity level of $\mathbf{u}_i$ by randomly replacing m elements of $\mathbf{u}_i$ with 0. More specifically we replace 80, 85, and 90 elements of $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ with 0.

We control the signal to noise ratio (SNR) by setting different variance for the noise term. Thus, with the desired SNR, we generate $\mathbf{X}=\sum_{i=1}^{3}\mathbf{d}_i\mathbf{u}_i\mathbf{v}_i^T+\epsilon$, where $\epsilon\sim N(0,\sigma^2)$, $\sigma=\sqrt{\frac{var(\sum_{i=1}^{3}\mathbf{d}_i\mathbf{u}_i\mathbf{v}_i^T)}{SNR}}$.

3.2 Simulation settings

To simulate the data set, we choose specific values for the desired SNR. We use SNR of 0.1, 0.5, 1, 1.5, 2 and 2.5 for our simulation studies. The candidate values of α that we use are 10, 20, 50, 70, 100, 150, 200, and 300 and the candidate values of λ that we use are 0.1, 0.5, 0.7, 1, 1.5, 2, 2.5, and 3.

As discussed in section 2.6.5, we use following stopping criterion $\| \hat{\mathbf{v}}^{new} - \hat{\mathbf{v}}^{old} \|_1 < 0.0001$ and $\| \hat{\mathbf{u}}^{new} - \hat{\mathbf{u}}^{old} \|_1 < 0.0001$ for simulation studies. We do 50 simulations for each setting.

3.3 Performance measures

We compute bias^2 and mean squared error (MSE) for three row and three column factors per each simulation settings. We compare the bias^2 and MSE from SFPCA and SVD under same simulation settings. For each elements of $\hat{\mathbf{u}}$ and simulation settings, we calculate the

$$\mathbf{bias}^2_{\hat{\mathbf{u}}} = (E[\hat{\mathbf{u}}] - \mathbf{u})^2 = ((\frac{1}{50}\sum_{i=1}^{50} \hat{\mathbf{u}}^{(i)}) - \mathbf{u})^2$$

$$\mathbf{MSE}_{\hat{\mathbf{u}}} = \mathbf{bias}^2_{\hat{\mathbf{u}}} + \frac{1}{49}\sum_{i=1}^{50}(\hat{\mathbf{u}}^{(i)} - E[\hat{\mathbf{u}}])^2$$

where $\hat{\mathbf{u}}^{(i)}$ is estimated $\mathbf{u}$ from i^{th} simulation.

Similarly, for each elements $\hat{\mathbf{v}}$ and simulation settings, we calculate

$$\mathbf{bias}^2_{\hat{\mathbf{v}}} = (E[\hat{\mathbf{v}}] - \mathbf{v})^2 = ((\frac{1}{50}\sum_{i=1}^{50} \hat{\mathbf{v}}^{(i)}) - \mathbf{v})^2$$

$$\mathbf{MSE}_{\hat{\mathbf{v}}} = \mathbf{bias}^2_{\hat{\mathbf{v}}} + \frac{1}{49}\sum_{i=1}^{50}(\hat{\mathbf{v}}^{(i)} - E[\hat{\mathbf{v}}])^2$$

where $\hat{\mathbf{v}}^{(i)}$ is estimated $\mathbf{v}$ from i^{th} simulation.

We show $\text{bias}^2_{\hat{\mathbf{u}}} = \frac{1}{100}\sum_{i=1}^{100} \mathbf{bias}^2_{\hat{\mathbf{u}}}$, $\text{bias}^2_{\hat{\mathbf{v}}} = \frac{1}{200}\sum_{i=1}^{200} \mathbf{bias}^2_{\hat{\mathbf{v}}}$, $\text{MSE}^2_{\hat{\mathbf{u}}} = \frac{1}{100}\sum_{i=1}^{100} \mathbf{MSE}^2_{\hat{\mathbf{u}}}$ and $\text{MSE}^2_{\hat{\mathbf{v}}} = \frac{1}{200}\sum_{i=1}^{200} \mathbf{MSE}^2_{\hat{\mathbf{v}}}$, the average of $\mathbf{bias}^2_{\hat{\mathbf{u}}}, \mathbf{bias}^2_{\hat{\mathbf{v}}}, \mathbf{MSE}^2_{\hat{\mathbf{u}}}$ and $\mathbf{MSE}^2_{\hat{\mathbf{v}}}$ over 100 elements of $\hat{\mathbf{u}}$ and 200 elements of $\hat{\mathbf{v}}$ in Table A.1.

The true positive rate (TP) and true negative rate (TN) are computed for three row factors. We compare the TP and TN from SFPCA and SVD under same simulation settings. We calculate the proportion of correctly estimated zero loadings over 50 simulations for elements of $\hat{\mathbf{u}}$ and show the true positive rate, the average of these proportion over 100

elements of $\hat{\mathbf{u}}$, in Table A.2. We compute the proportion of correctly estimated non-zero loadings over 50 simulations for elements of $\hat{\mathbf{u}}$ and show true negative rate, the average of these proportion over 100 elements of $\hat{\mathbf{u}}$, in Table A.2. We round the loadings from SVD upto 4 decimal points when we calculate TP and TN, otherwise TP and TN will be 0 and 1 since SVD does not estimate factors with zero loadings.

Chapter 4

RESULTS

We generate the dataset based on different settings of signal to noise ratio (SNR). Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 each consist of six sub-figures: $\mathbf{u}_1$ (1st left singular vector), $\mathbf{u}_2$ (2nd left singular vector), $\mathbf{u}_3$ (3rd left singular vector), $\mathbf{v}_1$ (1st right singular vector), $\mathbf{v}_2$ (2nd right singular vector) and $\mathbf{v}_3$ (3rd right singular vector) and those estimated. All sub-figures in Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 include two different colors of lines. The black lines (—) are the true factors that we generate. The orange lines (—) are the estimated left singular vectors from SFPCA with different values of SNR. The blue lines (—) are the estimated right singular vectors from SFPCA with different values of SNR.

Figure 4.1 illustrates that we fail to estimate third components with such a small SNR. Among 50 simulations, 40 (80%) second singular vectors and 50 (100%) third singular vectors were estimated as vectors with zeros. We only estimated 20% of second singular vectors and 0% of third singular vectors.

By comparing Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6, we can observe the effect of SNR on SFPCA. For SNR=(0.1, 0.5, 1, 1.5, 2, 2.5), the percentages of estimated second singular vectors are (20%, 100%, 100%, 100%, 100%, 100%) and the percentages of estimated third singular vectors are (0%, 22%, 38%, 78%, 84%, 90%). As we increase SNR, the percentages of properly estimated second and third singular vectors increase as well.

Figure 4.9 shows the proportion of variance explained (PVE) of true factors and cross-validated proportion of variance explained (CVPVE) of the estimated factors with different values of SNR. Figure 4.9 consists of six sub-figures with six different values of SNR. All sub-figures include two different colors of lines. The black lines (—) are the PVE by true factors that we generate. The blue lines (—) are the CVPVE by estimated factors from

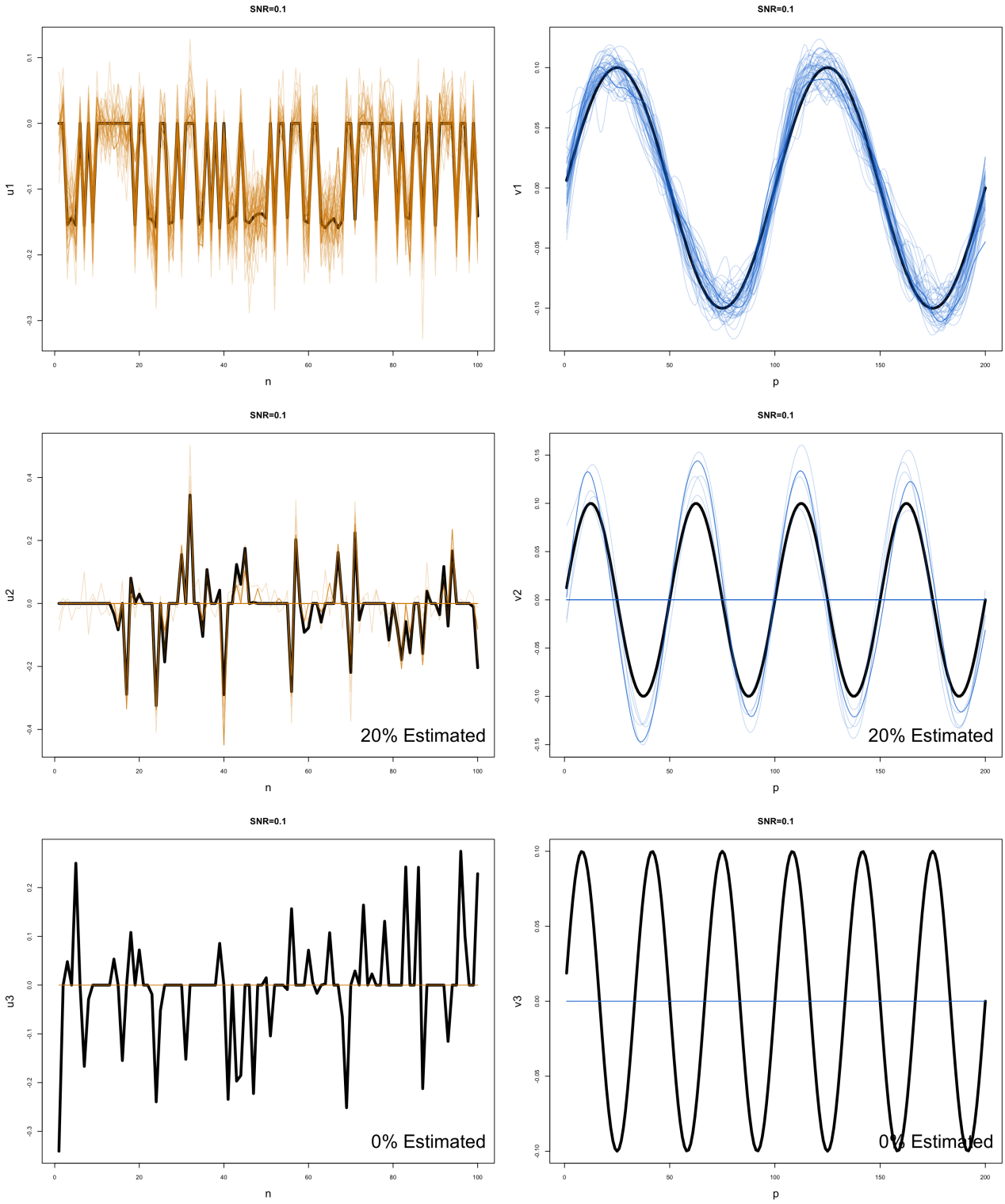


Figure 4.1: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=0.1. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 20% of second singular vectors and 0% of third singular vectors were estimated.

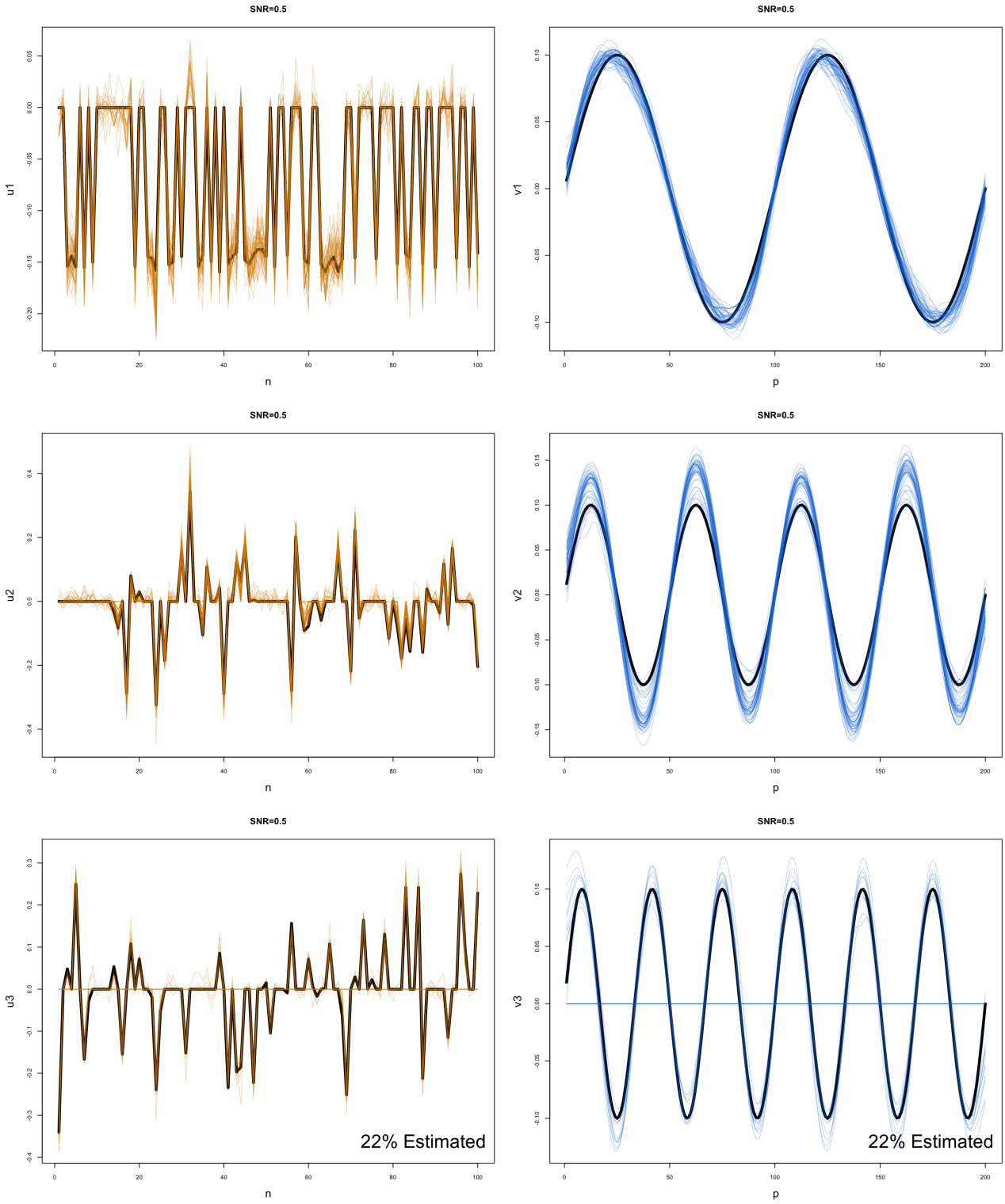


Figure 4.2: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=0.5. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 22% of third singular vectors were estimated.

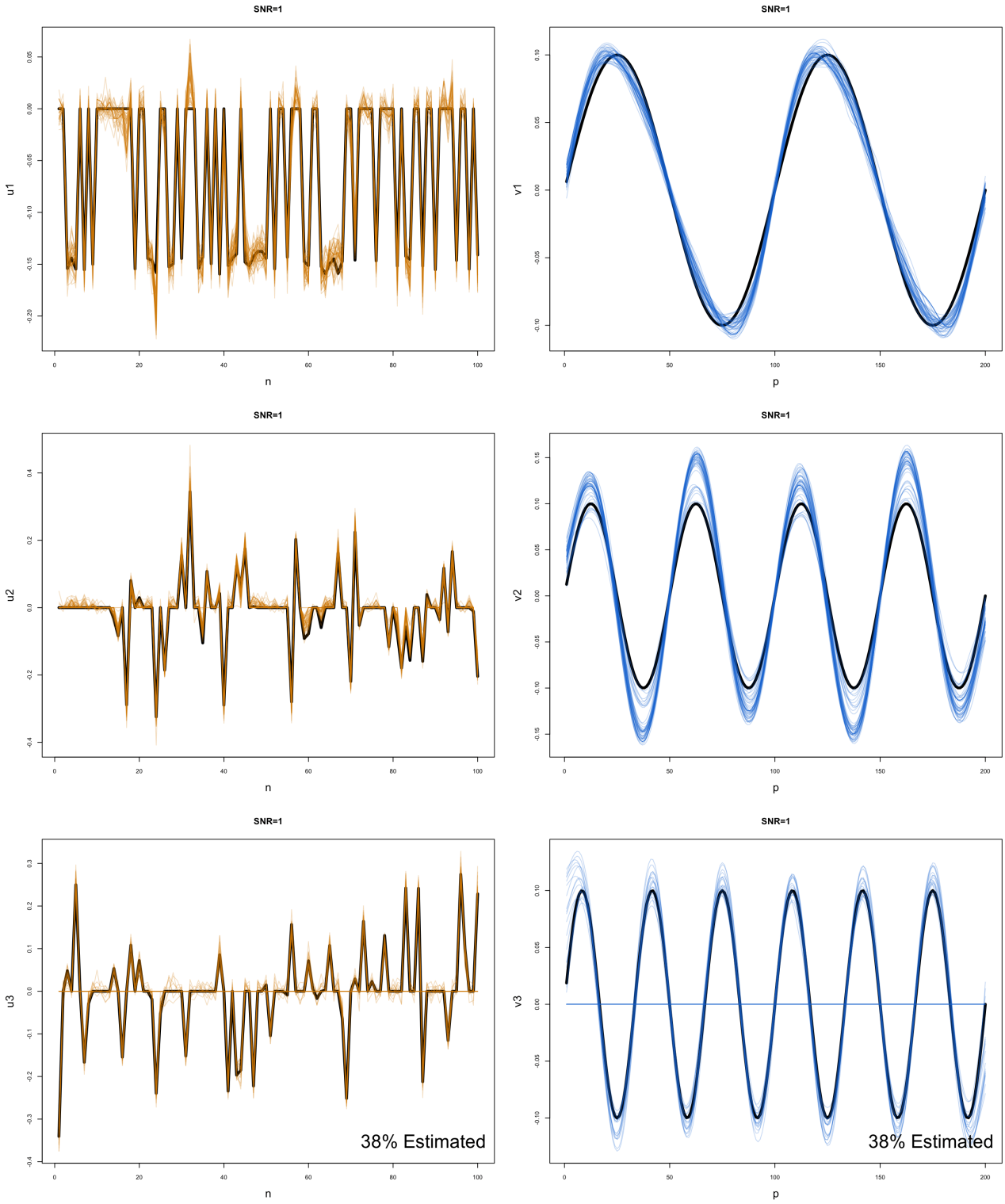


Figure 4.3: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=1. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 38% of third singular vectors were estimated.

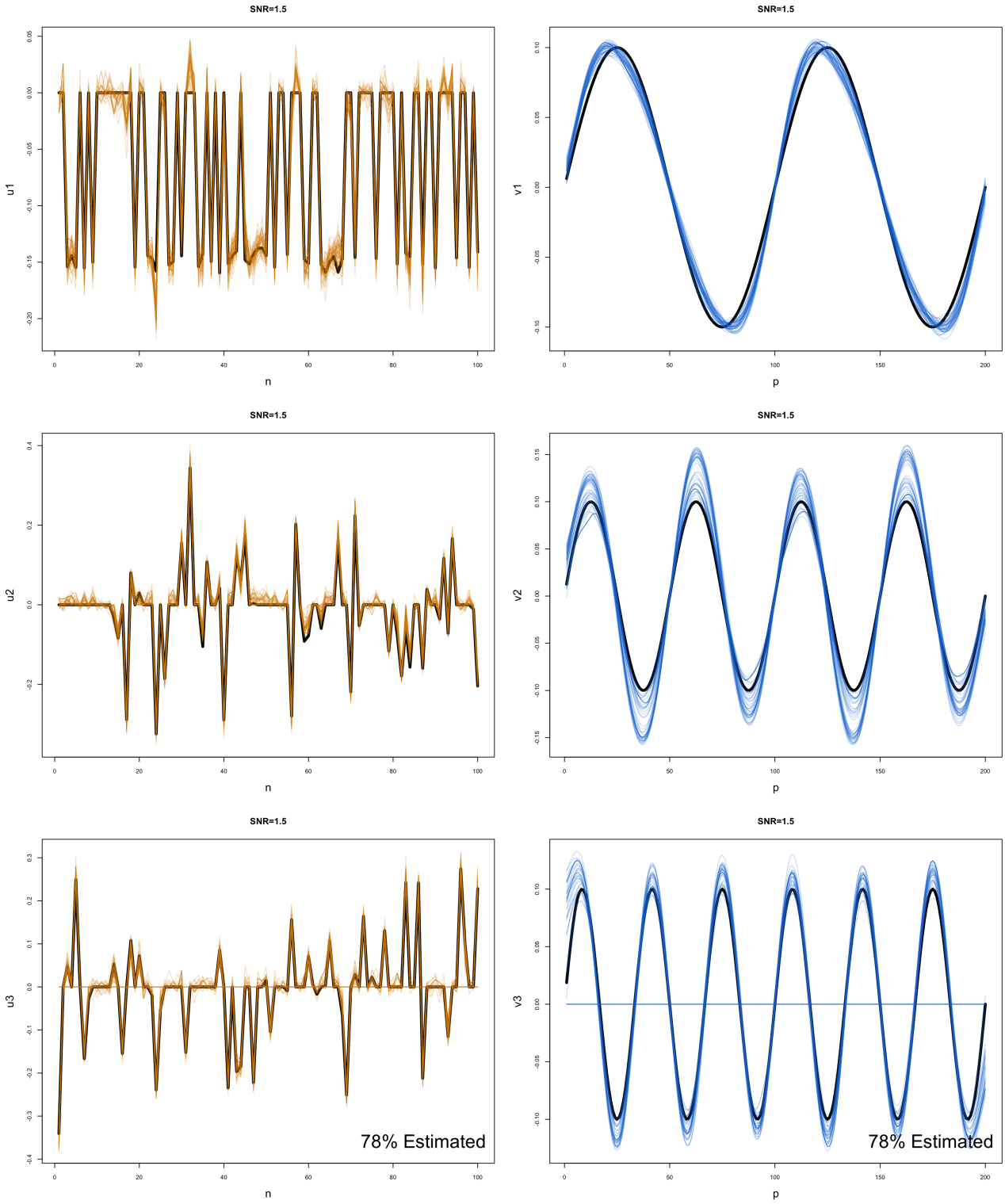


Figure 4.4: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=1.5. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 78% of third singular vectors were estimated.

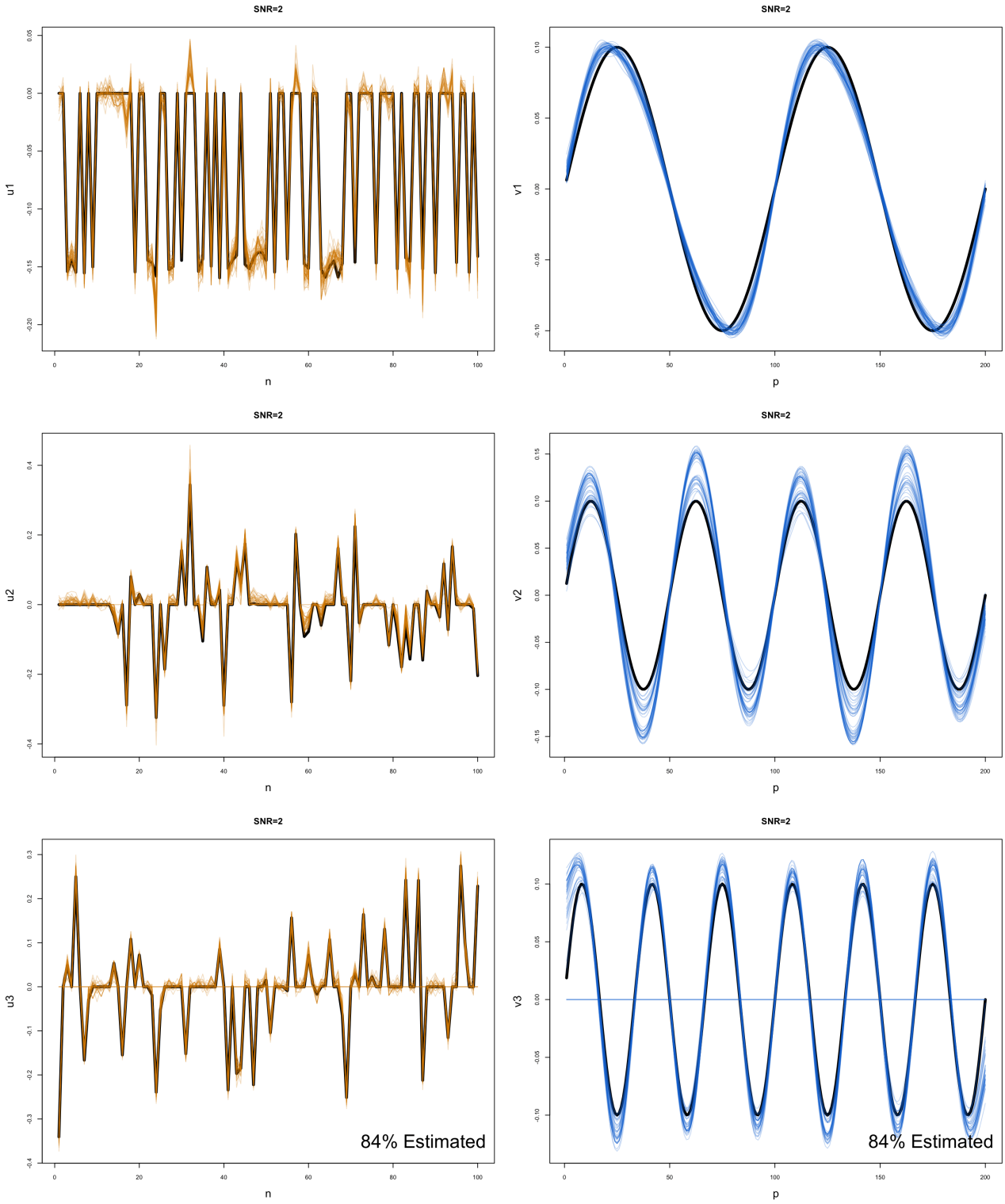


Figure 4.5: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=2. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 84% of third singular vectors were estimated.

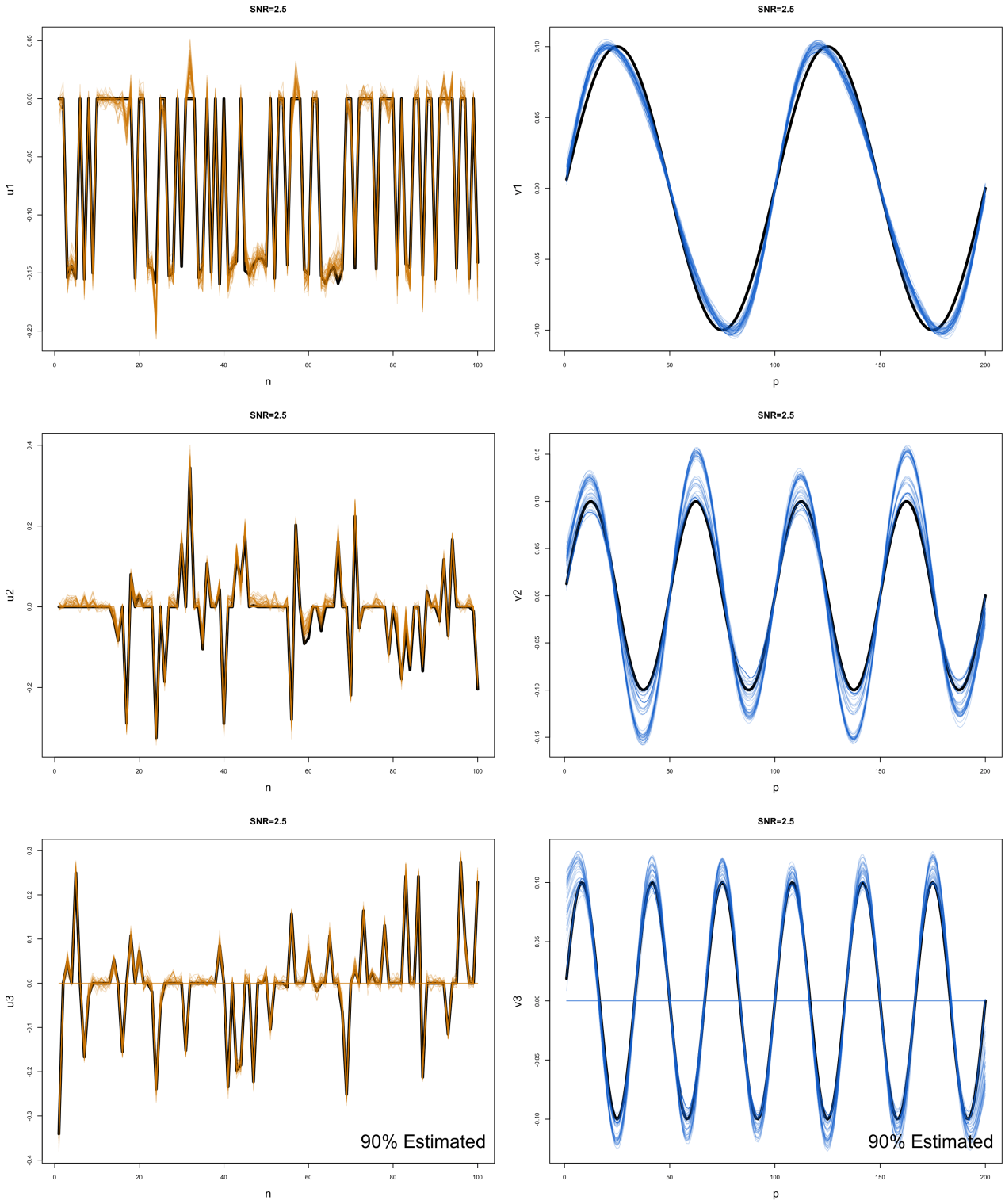


Figure 4.6: We compare true factors (—) with the estimated factors, left singular vectors (—) and right singular vectors (—) when SNR=2.5. In each panel, 50 estimated singular vectors with the optimal sparsity and smoothing parameter values are shown. Among 50 estimated singular vectors, 90% of third singular vectors were estimated.

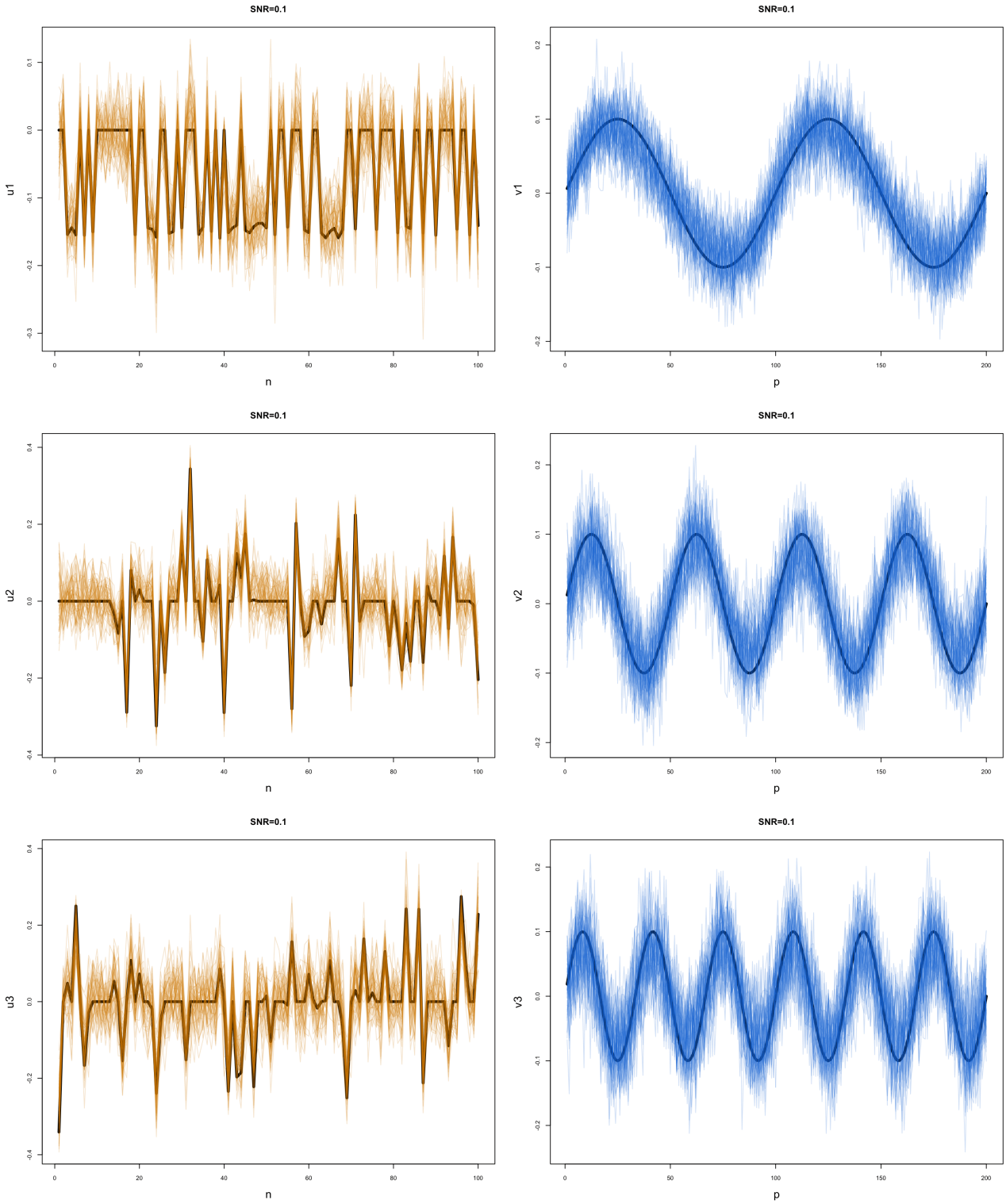


Figure 4.7: We compare true factors (—) with the estimated factors from SVD, left singular vectors (—) and right singular vectors (—) when SNR=0.1. In each panel, 50 estimated singular vectors are shown.

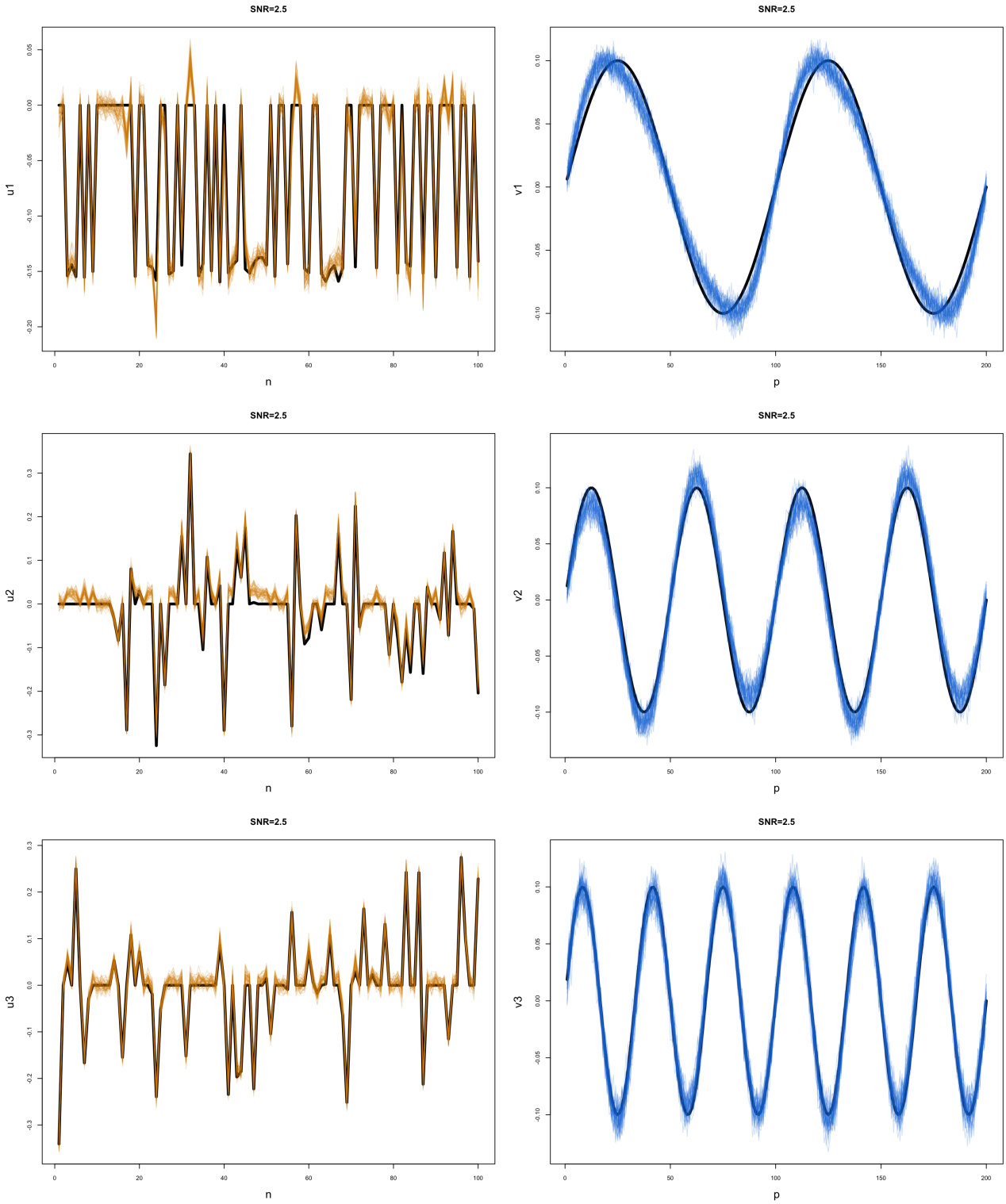


Figure 4.8: We compare true factors (—) with the estimated factors from SVD, left singular vectors (—) and right singular vectors (—) when SNR=2.5. In each panel, 50 estimated singular vectors are shown.

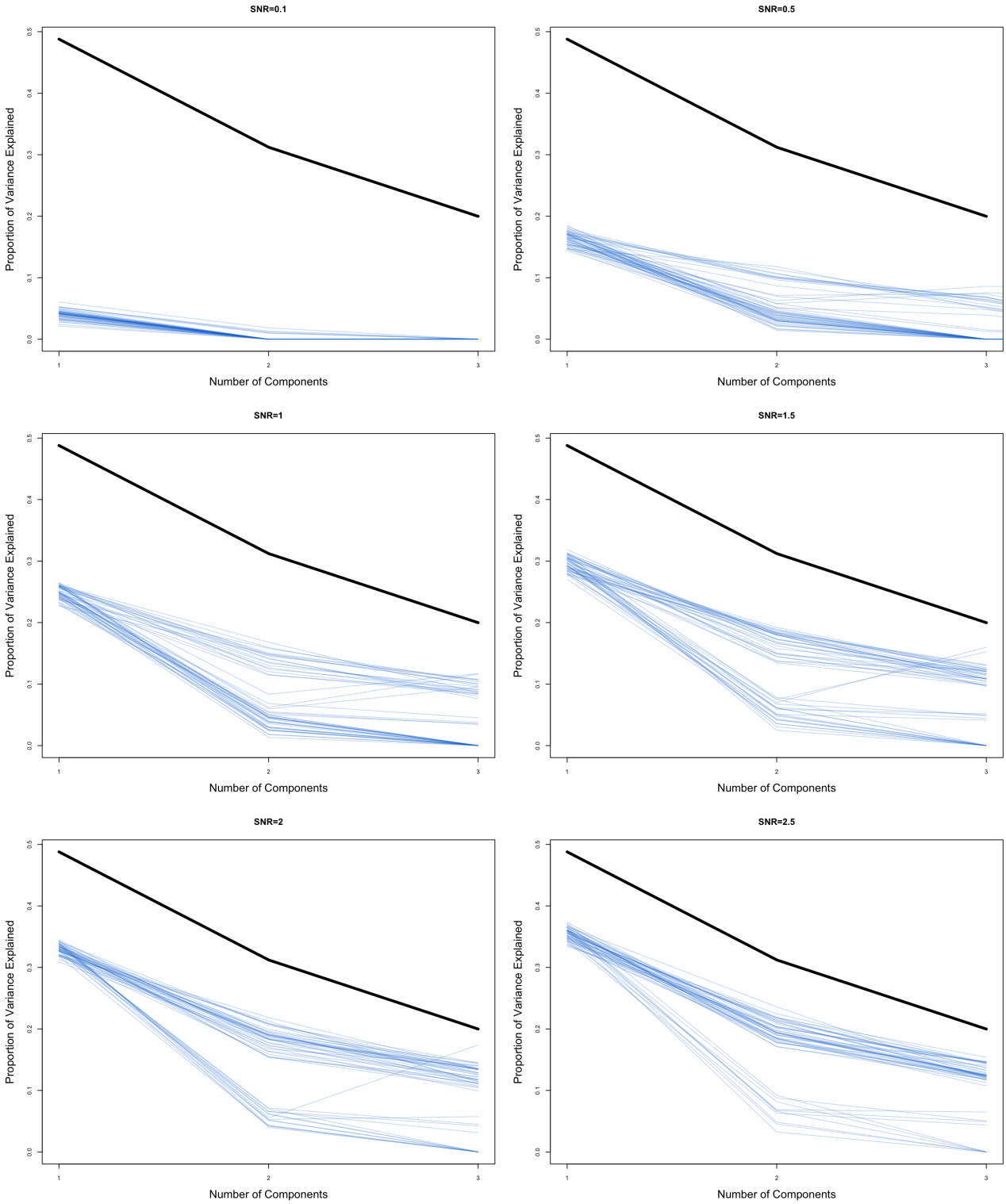


Figure 4.9: We compare proportion of variance explained of true factors (—) with cross-validated proportion of variance explained (CVPVE) of the estimated factors (—) when SNR=0.1, 0.5, 1, 1.5, 2 and 2.5. In each panel, 50 CVPVE of estimated singular vectors with chosen sparsity and smoothing parameter values are shown.

SFPCA with chosen smoothing and sparsity parameters and different values of SNR.

By comparing sub-figures of figure 4.9, we can see that the gap between the PVE of true factors and CVPVE of estimated factors from SFPCA decreases as SNR increases. Note that the PVE of true factors in figure 4.9 does not account for the noise term. The PVE of true factors accounting for the noise term will be lower than what has been plotted in figure 4.9, thus the difference will be smaller than what is plotted in figure 4.9.

Figures 4.7 and 4.8 show estimated factors from SVD when SNR is 0.1 and 2.5. By comparing Figure 4.1 and 4.7, we can see that SFPCA estimates the factors that are in shape of the true factors. The estimated factors from SVD is not sparse and too wiggly. However, SFPCA failed to estimate second and third factors while SVD estimated all three factors. Even when SNR increases to 2.5, estimated factors from SFPCA is close to the true factors.

Table A.1 shows bias^2 and MSE of estimated factors from SFPCA and SVD under each simulation settings. In general, we find that SFPCA has lower bias^2 and MSE. SVD has lower bias^2 and MSE when we estimate zero vectors for the third row and column factors with low SNR. As expected, bias^2 and MSE of SFPCA decrease as we increase SNR value.

Table A.2 displays the true positive rate (TP) and true negative rate (TN) of estimated row factors from SFPCA and SVD under each simulation settings. In all cases, SFPCA has higher TP than SVD since SVD does not estimate factors with zero loadings. True negative rate (TN) is higher with SVD in general since SVD estimates non zero loadings. We rounded loadings of SVD upto four decimal points. Otherwise, TP will be zero and TN will be one, which makes no sense to put it in Table A.2.

As we increase SNR, we get better TP and TN from SFPCA. Even we rounded loadings of SVD upto four decimal points, TP for SVD is lot smaller than TP of SFPCA. One can argue that SVD might estimate the true factors since TN is one in almost cases. However, it is because SVD estimate factors with non zero loadings.

Chapter 5

DATA EXAMPLE

Data used in this thesis were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (`adni.loni.usc.edu`). The ADNI was launched in 2003 as a public-private partnership, led by Principal Investigator Michael W. Weiner, MD. The primary goal of ADNI has been to test whether serial magnetic resonance imaging (MRI), positron emission tomography (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of mild cognitive impairment (MCI) and early Alzheimer's disease (AD). For up-to-date information, see `www.adni-info.org`.

We use preprocessed data by Madhyastha et al. (2013). Madhyastha et al. (2013) preprocessed raw fMRI ADNI data which corrects for motion and summarized the data by 264 region of interest (ROI) defined by Power et al. (2011). The sample consists of 21 participants with Alzheimer's disease and 21 participants in a matched control group by age and gender (Madhyastha et al. 2013). Madhyastha et al. (2013) summarized the 264 ROIs into seven sub-networks according to Yeo et al. (2011). The seven sub-networks are visual, somatomotor, dorsal attention, ventral attention, limbic, frontoparietal and default mode network (DMN).

Figures 5.1, 5.2, 5.3 and 5.4 show results from SVD and SFPCA of preprocessed ADNI dataset. We only show randomly chosen four participants: two participants with Alzheimer's disease and two participants in matched control group. Each figure consists of six sub-figures: estimated $\mathbf{u}_1$ (1st left singular vector ___), estimated $\mathbf{v}_1$ (1st right singular vector ___) and proportion of variance explained for SVD or cross-validated proportion of variance explained for SFPCA. As expected, factors estimated from SVD are not smooth and sparse. The first right singular vector estimated from SFPCA is smooth, which show the smooth major pattern

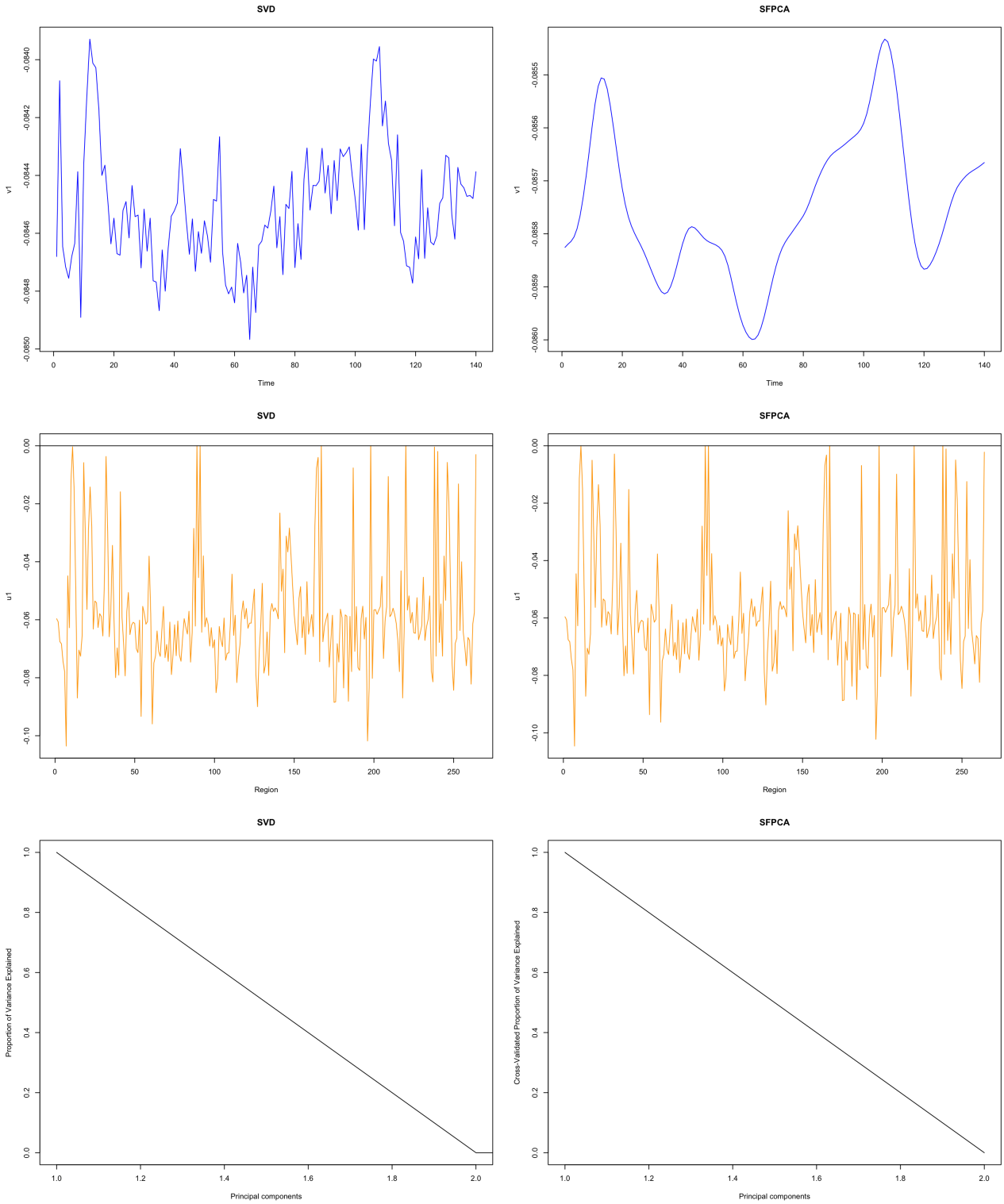


Figure 5.1: We compare the estimated right (—) and left (—) singular vectors from SVD and SFPCA using a ADNI data of participant with Alzheimer's disease.

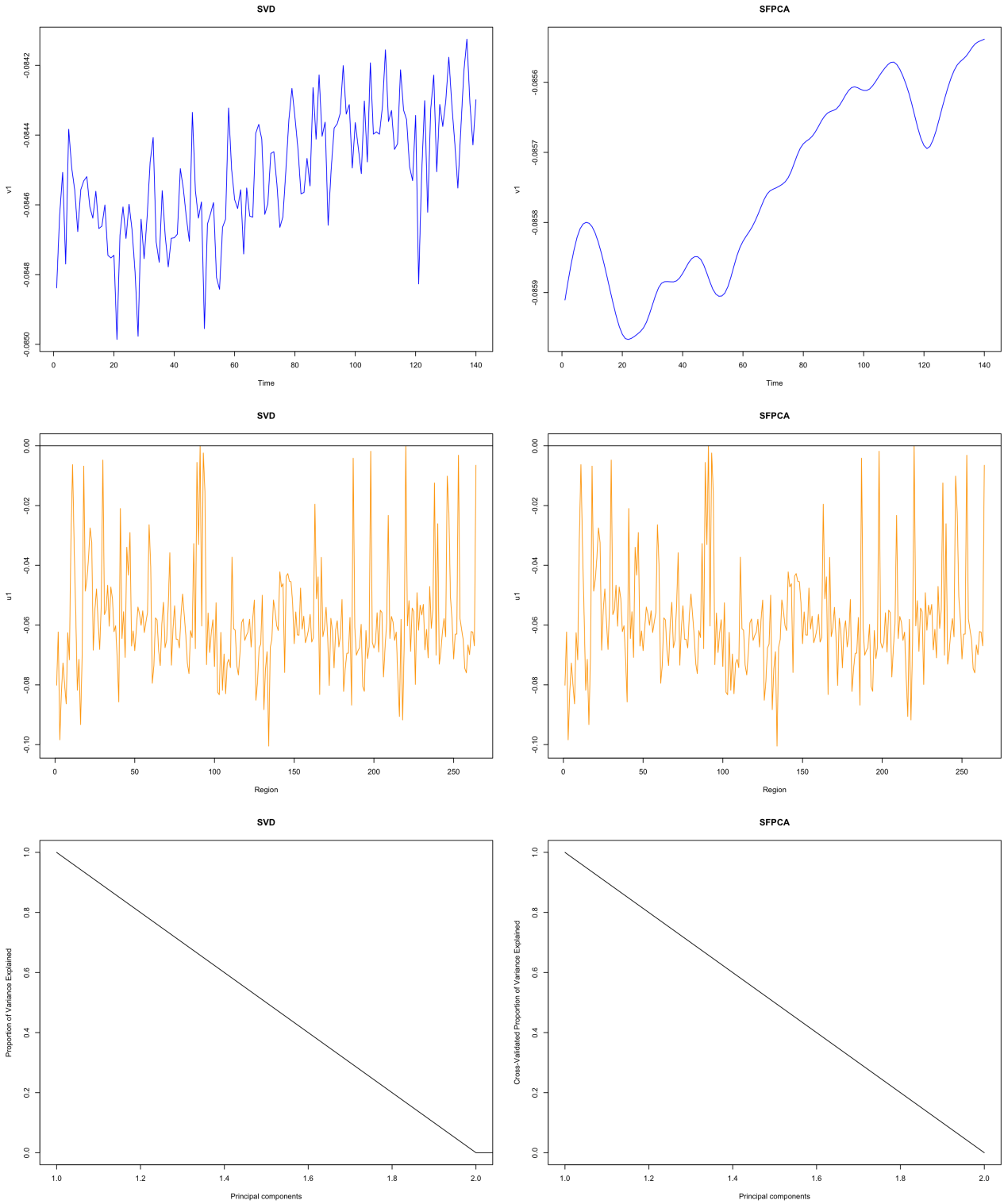


Figure 5.2: We compare the estimated right (—) and left (—) singular vectors from SVD and SFPCA using using a ADNI data of participant with Alzheimer's disease.

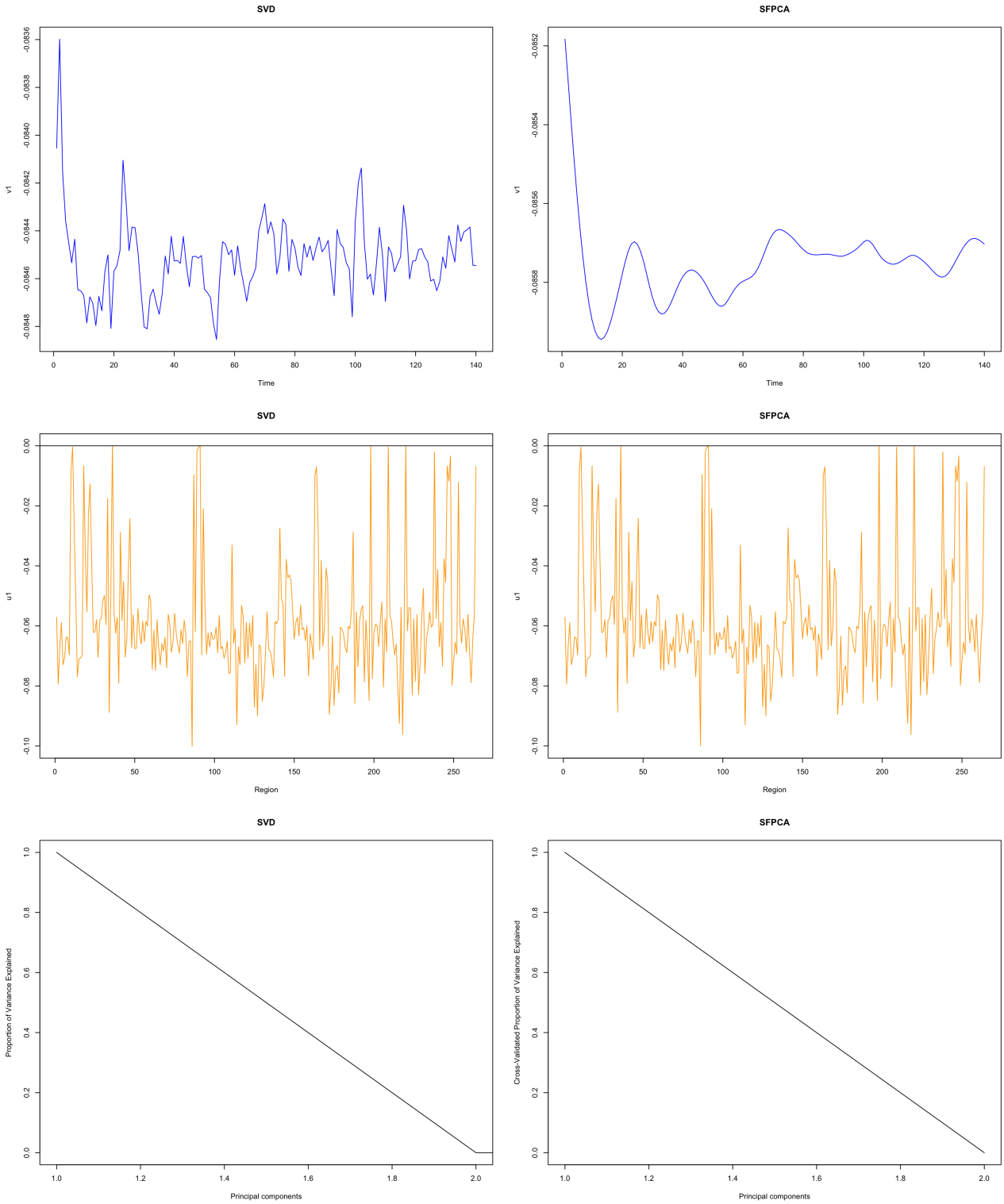


Figure 5.3: We compare the estimated right (—) and left (—) singular vectors from SVD and SFPCA using a ADNI data of participant in control group.

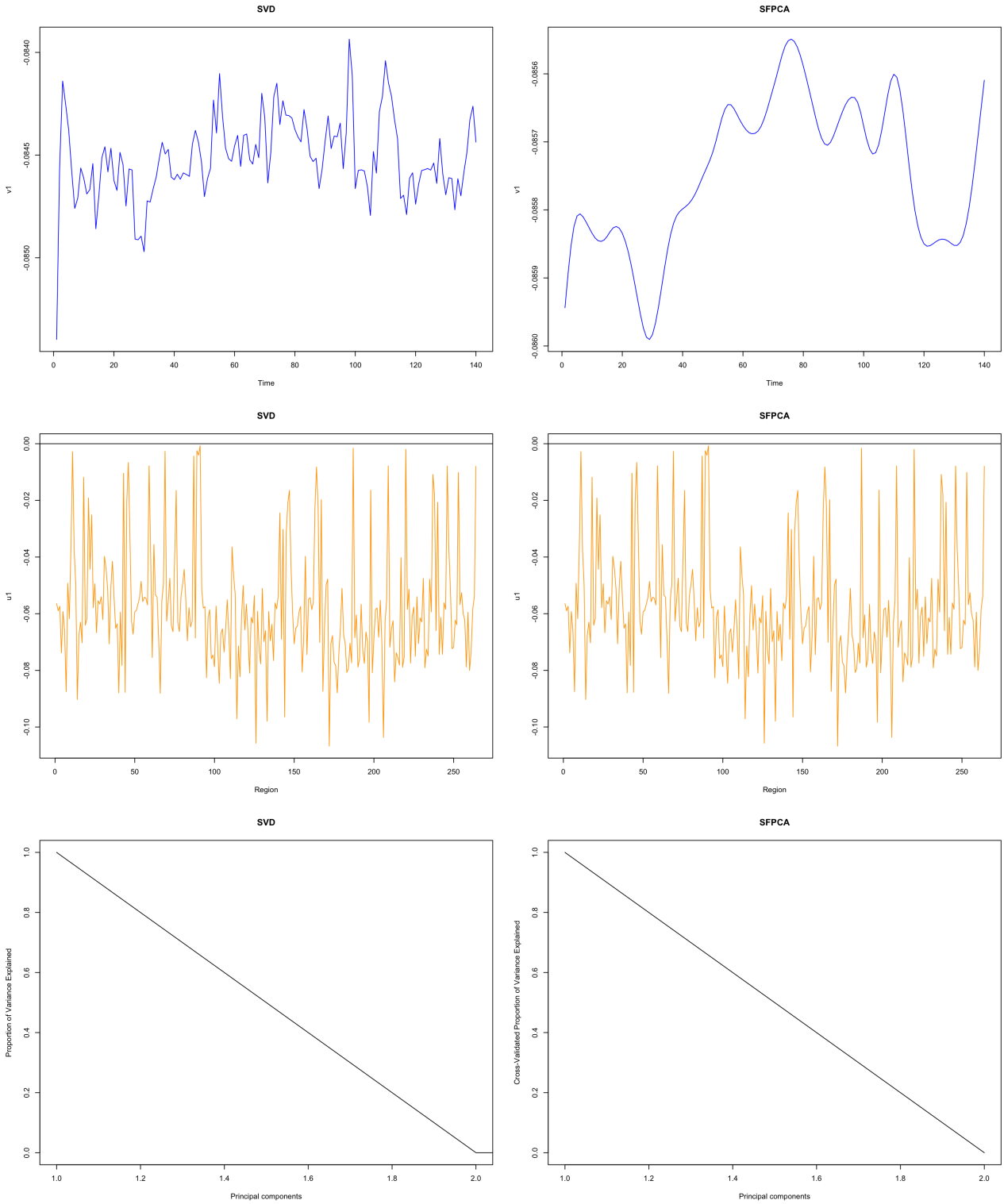


Figure 5.4: We compare the estimated right (___) and left (___) singular vectors from SVD and SFPCA using a ADNI data of participant in control group.

of data in time space. The first left singular vector estimated from SFPCA is slightly sparse, which show the major pattern of data in region space. Even the estimated left singular vector is slightly sparse, it actually has few loadings that are exactly zero while loadings for SVD are non-zero for all regions. We see that the estimated first factors explain more than 99% of total variance of the data for both SVD and SFPCA.

We count the number of subjects with zero loadings for row factor within each ROI. We compare the number of subjects for each ROI between AD group and matched control group. The difference of number of participants with zero loadings for each ROI between AD and control group was statistically significant ($p = 0.048$ from paired t-test), with higher number of participants with zero loadings in control group. The total number of zero loadings among 21 participants with AD across 264 brain ROIs is 49 while the total number of zero loadings among 21 participants in control group is 60. We compare the difference of number of participants with zero loadings for each ROIs separately but the difference was not significant with Bonferroni correction for multiple testing.

We also compare the estimated first right and left singular vectors from SVD and SFPCA using a row centered ADNI data. We centered the data by mean of each ROIs.

Figures 5.5, 5.6, 5.7 and 5.8 show results from SVD and SFPCA of row centered ADNI dataset. We only show randomly chosen four participants: two participants with Alzheimer's disease and two participants in matched control group. Each figure consists of six sub-figures: estimated u1 (1st left singular vector ___), estimated v1 (1st right singular vector ___) and proportion of variance explained for SVD or cross-validated proportion of variance explained for SFPCA. As expected, factors estimated from SVD are not smooth and sparse. The first right singular vector estimated from SFPCA is smooth, which show the smooth major pattern of data in time space. The first left singular vector estimated from SFPCA is sparse, which show the major pattern of data in region space. We see that the proportion of variance explained by first factors from SVD is higher than SFPCA in general.

We count the number of subjects with zero loadings for row factor within each ROI. We compare the number of subjects for each ROI between AD group and matched control group.

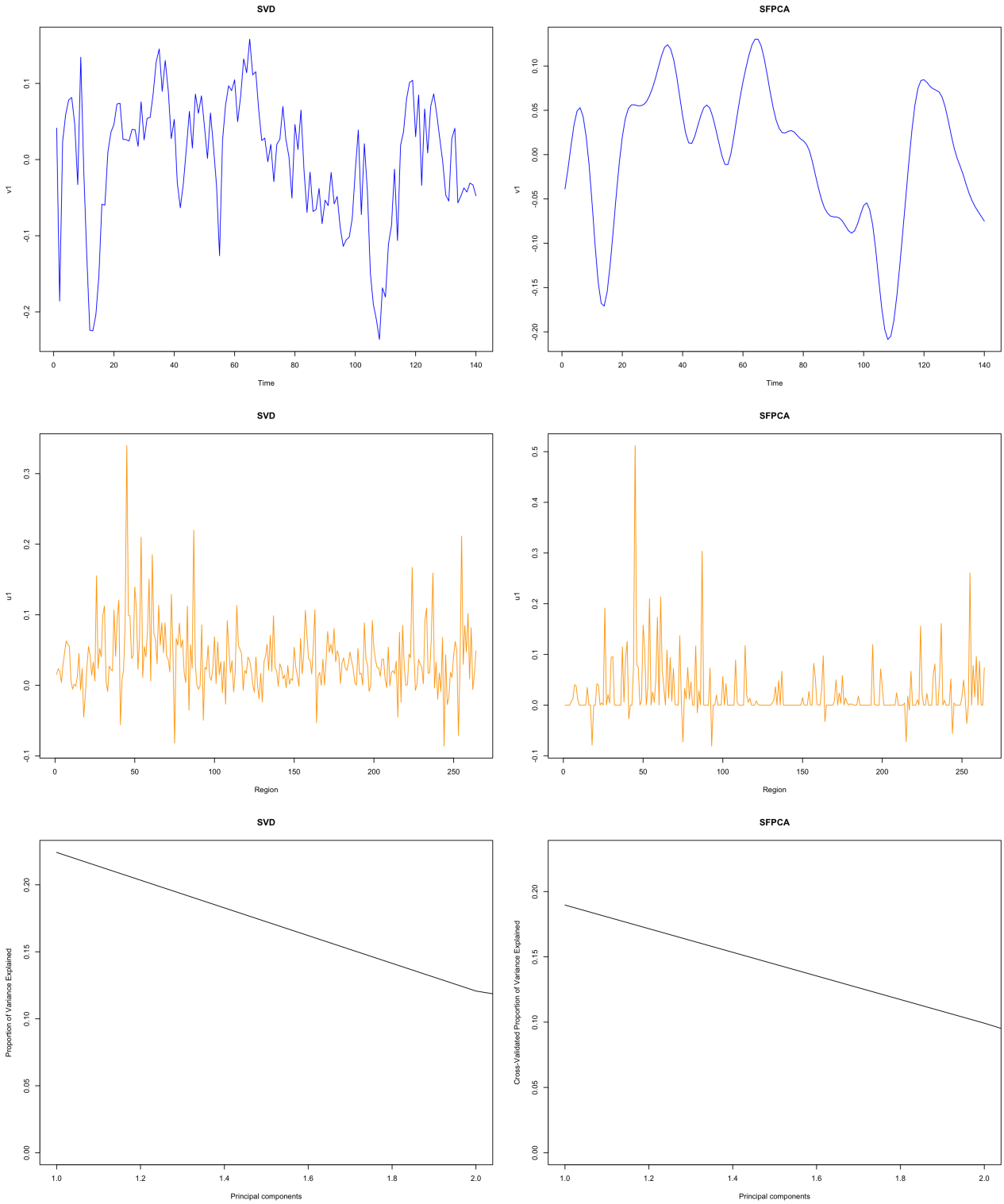


Figure 5.5: We compare the estimated first right (___) and first left (___) singular vectors from SVD and SFPCA using a row centered ADNI data of participant with Alzheimer's disease.

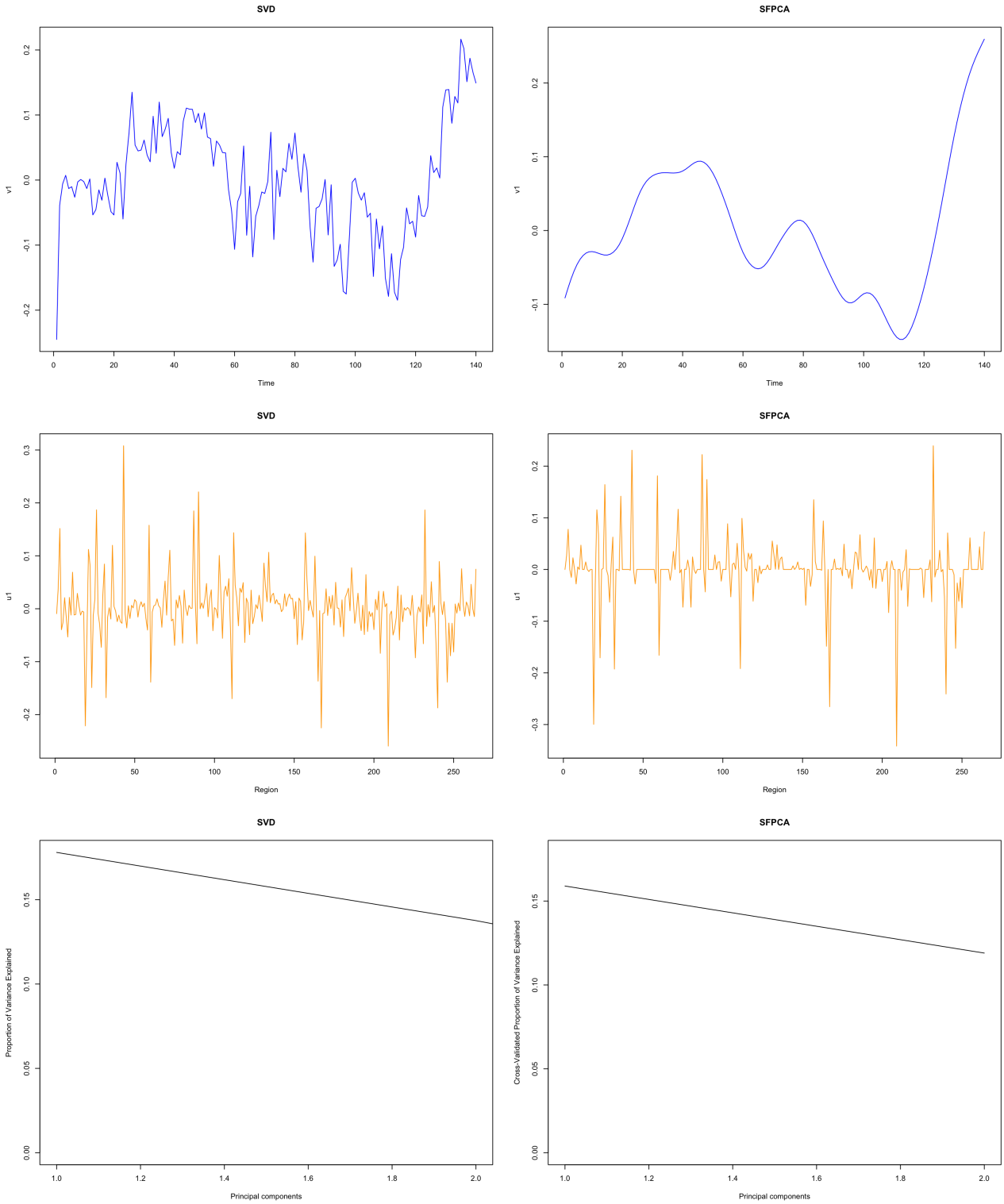


Figure 5.6: We compare the estimated first right (—) and first left (—) singular vectors from SVD and SFPCA using using a row centered ADNI data of participant with Alzheimer's disease.

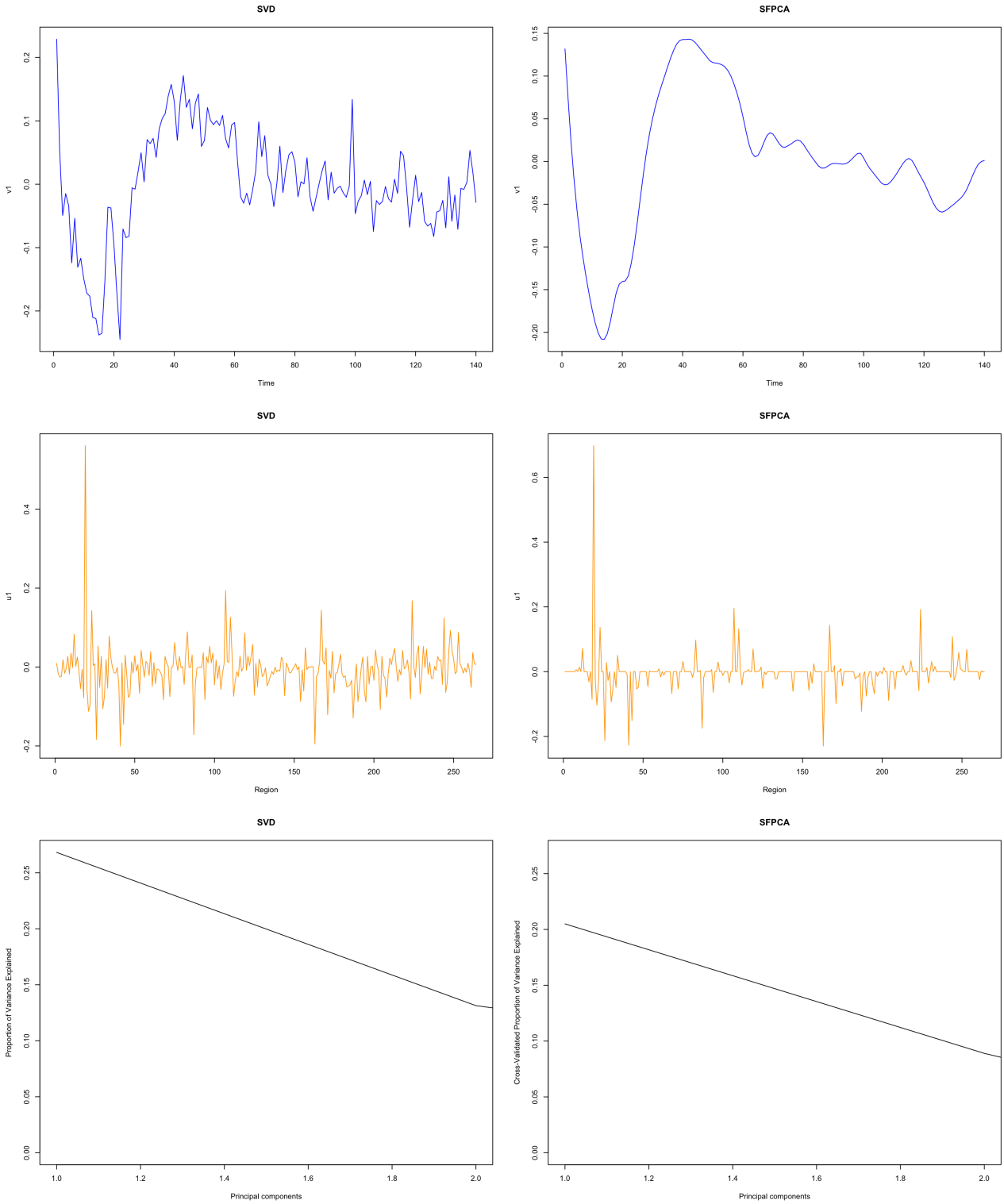


Figure 5.7: We compare the estimated first right (___) and first left (___) singular vectors from SVD and SFPCA using a row centered ADNI data of participant in control group.

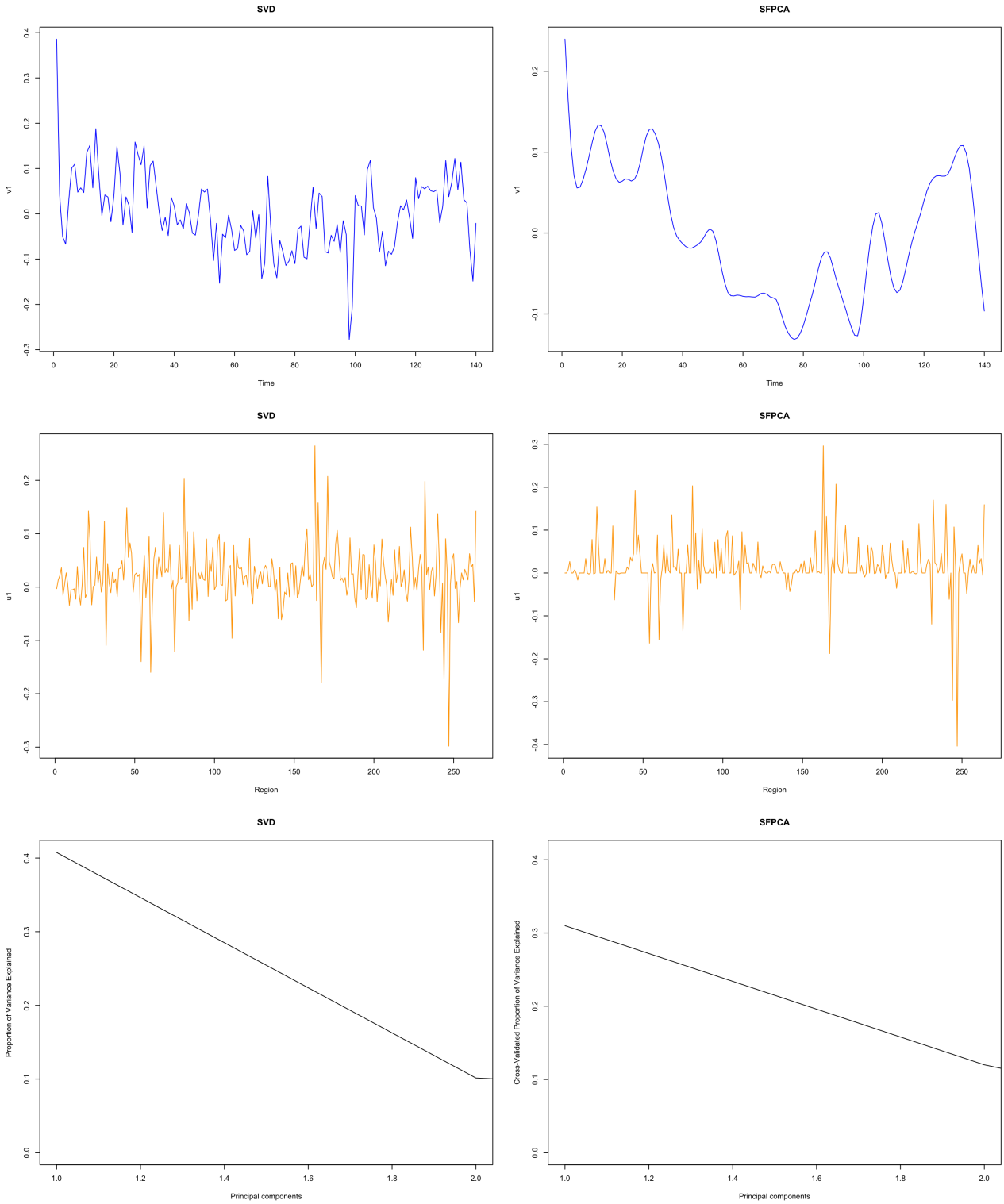


Figure 5.8: We compare the estimated first right (___) and first left (___) singular vectors from SVD and SFPCA using a row centered ADNI data of participant in control group.

The difference of number of participants with zero loadings for each ROI between AD and control group was statistically significant ($p = 0.01$ from paired t-test), with higher number of participants with zero loadings in control group. We compare the difference of number of participants with zero loadings for each ROIs separately. We use Bonferroni correction for multiple testing. The difference for each ROIs between AD and control group was not significant.

Chapter 6

DISCUSSION

In this thesis, we investigated a regularized PCA method with sparse row factors and smooth column factors. We introduced a method of cross-validating unsupervised problem with data that has a time structure.

In our simulations, we observe that SFPCA estimates the true factors accurately when signal to noise ratio (SNR) increases. As SNR increases, the cross-validated proportion of variance explained (CVPVE) is improved. We compare our method to SVD when the true column factors are smooth and row factors are sparse. SFPCA estimates the true factors more accurately than SVD. When SNR is small, SFPCA did not estimate second and third component while SVD estimated second and third component. However, the estimated factors from SVD with small SNR was not sparse and too wiggly. Even when we have more signal, SFPCA estimated factors that are close to the true factors. This indicates that using SFPCA with appropriate SNR is more effective when the true factors are sparse and smooth.

There are several limitations in our problem. In section 2.7.1, we explain steps to find regularization parameters. When we search for the smoothing parameter, we use a fixed vector for $\mathbf{u}$, $\mathbf{u} \leftarrow \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$. We use this specific vector to find the smoothing parameter under the working assumption of uniform smoothness across all brain regions. However in practice, putting equal weight across brain regions might not be the appropriate choice.

In the simulation studies, we require candidate values for sparsity and smoothing parameters. SFPCA can be sensitive to the candidates of the regularization parameters since the selected values must be from the candidate list. With poor candidate values, estimated factors from SFPCA will likely be meaningless.

As discussed in section 2.7.1, we search for smoothing parameter α and sparsity parameter λ sequentially. We are not jointly searching for all combinations of candidates of those parameters. Thus, we are not jointly maximizing the objective function in the optimization problem (2.5). Instead we search for the smoothing parameter that maximizes the objective in (2.6) and then search for the sparsity parameter that maximizes the objective in (2.7).

In summary, SFPCA estimates the smooth column factors and sparse row factors. Especially when we have more signal, SFPCA estimates all factors properly. However, reasonable candidates of the smoothing and sparsity parameters are needed for good performance of SFPCA.

BIBLIOGRAPHY

Allen, G. I. (2013), 'Sparse and functional principal components analysis', *arXiv:1309.2895 [stat.ML]* .
URL: *http://arxiv.org/abs/1309.2895*

Berrendero, J. R., Justel, A. & Svarc, M. (2011), 'Principal components for multivariate functional data', *Computational Statistics & Data Analysis* **55**(9), 2619–2634.

Huang, J. Z., Shen, H. & Buja, A. (2009), 'The analysis of two-way functional data using two-way regularized singular value decompositions', *Journal of the American Statistical Association* **104**(488).

Huang, S., Li, J., Ye, J., Wu, T., Chen, K., Fleisher, A. & Reiman, E. (2011), Identifying alzheimer's disease-related brain regions from multi-modality neuroimaging data using sparse composite linear discrimination analysis, *in* 'Advances in neural information processing systems', pp. 1431–1439.

Johnstone, I. M. & Lu, A. Y. (2009), 'On consistency and sparsity for principal components analysis in high dimensions', *Journal of the American Statistical Association* **104**(486).

Jolliffe, I. (2002), *Principal component analysis*, Wiley Online Library.

Lillemark, L., Sørensen, L., Pai, A., Dam, E. B. & Nielsen, M. (2014), 'Brain regions relative proximity as marker for alzheimers disease based on structural mri', *BMC medical imaging* **14**(1), 1.

Madhyastha, T., Cao, Y., Sujitnapisatham, S., Presnyakov, G., Chaovalitwongse, W. A., Grabowski, T., Initiative, A. D. N. et al. (2013), 'Link clustering to explore brain dynamics using resting state functional mri', *Journal of radiology and radiation therapy* **1**(2), 1012.

Perry, P. O. (2009), 'Cross-validation for unsupervised learning', *arXiv preprint arXiv:0909.3052* .

Power, J. D., Cohen, A. L., Nelson, S. M., Wig, G. S., Barnes, K. A., Church, J. A., Vogel, A. C., Laumann, T. O., Miezin, F. M., Schlaggar, B. L. et al. (2011), 'Functional network organization of the human brain', *Neuron* **72**(4), 665–678.

Rice, J. A. & Silverman, B. W. (1991), 'Estimating the mean and covariance structure nonparametrically when the data are curves', *Journal of the Royal Statistical Society. Series B (Methodological)* pp. 233–243.

Saad, Y. (1988), 'Projection and deflation method for partial pole assignment in linear state feedback', *Automatic Control, IEEE Transactions on* **33**(3), 290–297.

Ye, J., Li, Y., Lazar, N. A., Schaeffer, D. J. & McDowell, J. E. (2016), 'Finding common task-related regions in fmri data from multiple subjects by periodogram clustering and clustering ensemble', *Statistics in medicine* .

Yeo, B. T., Krienen, F. M., Sepulcre, J., Sabuncu, M. R., Lashkari, D., Hollinshead, M., Roffman, J. L., Smoller, J. W., Zöllei, L., Polimeni, J. R. et al. (2011), 'The organization of the human cerebral cortex estimated by intrinsic functional connectivity', *Journal of neurophysiology* **106**(3), 1125–1165.

Zou, H., Hastie, T. & Tibshirani, R. (2006), 'Sparse principal component analysis', *Journal of computational and graphical statistics* **15**(2), 265–286.

Appendix A

TABLE

Table A.1: Comparing Bias^2 and MSE of sparse functional principal component analysis (SFPCA) and singular value decomposition (SVD).

			SFPCA		SVD	
			$Bias^2$	MSE	$Bias^2$	MSE
SNR	0.1	u1	0.00009	0.00100	0.00014	0.00137
		u2	0.00671	0.00837	0.00040	0.00255
		u3	0.01000	0.01000	0.00046	0.00379
		v1	0.00006	0.00019	0.00014	0.00112
		v2	0.00279	0.00416	0.00024	0.00183
		v3	0.00500	0.00500	0.00038	0.00266
	0.5	u1	0.00008	0.00020	0.00013	0.00036
		u2	0.00027	0.00057	0.00031	0.00072
		u3	0.00615	0.00791	0.00010	0.00069
		v1	0.00004	0.00007	0.00010	0.00032
		v2	0.00055	0.00070	0.00020	0.00046
		v3	0.00295	0.00397	0.00025	0.00056
	1	u1	0.00008	0.00017	0.00015	0.00027
		u2	0.00023	0.00039	0.00035	0.00054
		u3	0.00390	0.00533	0.00009	0.00037
		v1	0.00008	0.00010	0.00012	0.00023
		v2	0.00038	0.00039	0.00020	0.00040
		v3	0.00178	0.00190	0.00018	0.00029
	1.5	u1	0.00007	0.00013	0.00013	0.00021
		u2	0.00016	0.00028	0.00032	0.00045
		u3	0.00024	0.00028	0.00008	0.00026
		v1	0.00007	0.00008	0.00011	0.00018
		v2	0.00022	0.00055	0.00010	0.00022
		v3	0.00044	0.00071	0.00002	0.00019
	2	u1	0.00006	0.00011	0.00013	0.00019
		u2	0.00017	0.00030	0.00032	0.00041
		u3	0.00036	0.00093	0.00008	0.00021
		v1	0.00007	0.00008	0.00011	0.00016
		v2	0.00020	0.00063	0.00011	0.00019
		v3	0.00010	0.00021	0.00002	0.00015
	2.5	u1	0.00007	0.00010	0.00013	0.00018
		u2	0.00015	0.00024	0.00031	0.00039
		u3	0.00020	0.00051	0.00007	0.00018
		v1	0.00007	0.00008	0.00011	0.00015
		v2	0.00040	0.00058	0.00010	0.00017
		v3	0.00007	0.00020	0.00002	0.00012

Table A.2: Comparing true positive and true negative of SFPCA and SVD.

			SFPCA		SVD	
			True Positive	True Negative	True Positive	True Negative
SNR	0.1	u1	0.4360	1.0000	0.0007	1.0000
		u2	0.9655	0.1379	0.0007	1.0000
		u3	1.0000	0.0000	0.0000	0.9995
	0.5	u1	0.6753	1.0000	0.0015	1.0000
		u2	0.7721	0.8569	0.0013	1.0000
		u3	0.9453	0.1825	0.0010	1.0000
	1	u1	0.6807	1.0000	0.0022	1.0000
		u2	0.7984	0.9128	0.0013	1.0000
		u3	0.8373	0.3480	0.0030	1.0000
	1.5	u1	0.6843	1.0000	0.0055	1.0000
		u2	0.7038	0.9554	0.0010	1.0000
		u3	0.6520	0.7110	0.0017	1.0000
	2	u1	0.7058	1.0000	0.0029	1.0000
		u2	0.7505	0.9410	0.0023	1.0000
		u3	0.6574	0.7640	0.0027	1.0000
	2.5	u1	0.7424	1.0000	0.0047	1.0000
		u2	0.7536	0.9605	0.0023	1.0000
		u3	0.7034	0.8095	0.0020	1.0000

Appendix B

PYTHON CODE

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx

################################################
##### updateV:
##With randomly removed time coulmns, get opt v
################################################
def updateV(Xint, left, out, center, row, alphav, N, eps):
    (n, p) =Xint.shape
    omegav=getmat(n, p)[1]
    uint=np.divide(np.ones(n), np.sqrt(n))
    uint=uint.reshape(n, 1)

    #center the data
    if center==1:
        X=rowcenterdat(Xint, row)

    else:
        X=Xint

    vvar=cvx.Variable(p)
    j=1
```

```
    u0, d0, v0=np.linalg.svd(Xint)
    vint=v0[0,0:]
    vint=vint.reshape(p, 1)
    Itmp=np.identity(p)
    for i in range(0, out.size):
        Itmp[out[i], out[i]]=0
    while j<=N:
        obj=cvx.Maximize(np.dot(np.dot(uint.T, X), Itmp)*vvar)
        constraints= [cvx.quad_form(vvar, (np.identity(p)+alphav*omegav))<=1]
        prob=cvx.Problem(obj, constraints)
        prob.solve(solver=cvx.SCS)

        if np.linalg.norm(vint-vvar.value, 1) < eps:
            break
        j=j+1
        vint=vvar.value
    out=out.astype(int)
    d=np.array(np.dot(uint.T, X)*vint)
    return(X[:, out], (d*uint*vint.T)[:, out])


##################################################
##getAlpha: Select alphav
##################################################
def getAlpha(Xint, center, row, nmiss, alphav, K, N, eps):
    nalph=alphav.shape[0]
    resultmat=np.zeros((nalph, K+2))
```

```
    for k in range(1, K+1):
        (out, left, Xhat)=timermv(Xint, nmiss)
        for u in range(1, nalph+1):
            (X, updatX)=updateV(Xint, left, out, center, row, alphav[u-1], N, eps)
            resultmat[u-1, 0]=alphav[u-1]
            resultmat[u-1, k]=np.square(np.linalg.norm(X-updatX))

    for v in range(1, nalph+1):
        resultmat[v-1, K+1]=np.mean(resultmat[v-1, range(1,K+1)])

    ind=np.argmin(resultmat[:, K+1])
    alphaout=resultmat[ind, 0]

    return(alphaout, resultmat)



##################################################
##updateUV: With randomly removed time coulmns, get opt v
and u ####################################
################################################
def updateUV(Xint, left, out, center, row, alphav, lambdau, N, eps):
    (n, p) =Xint.shape
    (omegau, omegav)=getmat(n, p)
    out=out.astype(int)
    left=left.astype(int)

    #center the data
    if center==1:
```

```
    X=rowcenterdat(Xint, row)
else:
    X=Xint
uvar=cvx.Variable(n)
vvar=cvx.Variable(p)
j=1

u0, d0, v0=np.linalg.svd(Xint)
uint=u0[0:,0]
uint=uint.reshape(n, 1)
vint=v0[0,0:]
vint=vint.reshape(p, 1)

Itmp=np.identity(p)
for i in range(0, out.size):
    Itmp[out[i], out[i]]=0

while j<=N:
    obj=cvx.Maximize(uvar.T*np.dot(np.dot(X, Itmp), vint)-lambdau*cvx.norm(uvar, 1))
    constraints=[cvx.norm(uvar, 2)<=1]
    prob=cvx.Problem(obj, constraints)
    prob.solve(solver=cvx.SCS)

    if np.linalg.norm(uint-uvar.value, 1) < eps:
        break
    uint=uvar.value

    pobj=cvx.Maximize(np.dot(np.dot(uint.T, X), Itmp)*vvar)
    vconstraints=[cvx.quad_form(vvar, np.identity(p)+alphav*omegav)<=1]
```

```
        vprob=cvx.Problem(pobj, vconstraints)
        vprob.solve(solver=cvx.SCS)
        if np.linalg.norm(vint-vvar.value, 1) < eps:
            break
        j=j+1
        vint=vvar.value

        d=np.array(np.dot(uint.T, X)*vint)
    return(X[:, out], np.multiply(d, uint*vint.T)[:, out])

##############################################
##### getLambda: Select lambdau
##############################################
def getLambda(Xint, center, row, nmiss, K, lambdau, alphaopt, N, eps):
    nlam=lambdau.size
    resultmat=np.zeros((nlam, K+2))
    pve=np.zeros((nlam, K+2))

    for k in range(1, K+1):
        (out, left, Xhat)=timermv(Xint, nmiss)
        for u in range(1, nlam+1):
            (Xout, updatX)=updateUV(Xint, left, out,
             center, row, alphaopt, lambdau[u-1], N, eps)
            resultmat[u-1, 0]=lambdau[u-1]
            resultmat[u-1, k]=np.square(np.linalg.norm(Xout-updatX))
            pve[u-1, 0]=lambdau[u-1]
            pve[u-1, k]=np.square(np.linalg.norm(Xout-updatX))/np.square(np.linalg.norm(Xout))
    for v in range(0, nlam):
        resultmat[v, K+1]=np.mean(resultmat[v, 1:K])
```

```
        pve[v, K+1]=np.mean(pve[v, 1:K])

    indmin=np.argmin(resultmat[:, K+1])
    lambdaout=resultmat[indmin, 0]

    return(lambdaout, pve[indmin, K+1])

################################################
##### fspcaprop: Calculate prop. of variance explained
################################################
def fspcaprop(Xint, center, row, Uest, Dest, Vest):
    if center==1:
        Xint=rowcenterdat(Xint, row)

    updatX=Dest*Uest*Vest.T
    PVE=np.square(np.linalg.norm(Xint-updatX, 2))/np.square(np.linalg.norm(Xint, 2))

    return(PVE)

################################################
##### getuv: Function for functional sparse pca
################################################
def getuv(Xint, center, row, alphav, lambdau, N, eps):
    if center==1:
        Xint=rowcenterdat(Xint, row)
    (n, p)=Xint.shape
    (omegau, omegav)=getmat(n, p)
    j=1
```

```
uvar=cvx.Variable(n)
vvar=cvx.Variable(p)

u0, d0, v0=np.linalg.svd(Xint)
uint=u0[0:,0]
uint=uint.reshape(n, 1)
vint=v0[0,0:]
vint=vint.reshape(p, 1)

while j<=N:
    obj=cvx.Maximize(uvar.T*Xint*vint-lambdau*cvx.norm(uvar, 1))
    const=[cvx.norm(uvar, 2) <= 1]
    prob=cvx.Problem(obj, const)
    prob.solve(solver=cvx.SCS)
    if np.linalg.norm(uint-uvar.value, 1)<eps:
        break
    uint=uvar.value

    vobj=cvx.Maximize(uint.T*Xint*vvar)
    vconst=[cvx.quad_form(vvar, np.identity(p)+alphav*omegav)<=1]
    vprob=cvx.Problem(vobj, vconst)
    vprob.solve(solver=cvx.SCS)
    if np.linalg.norm(vint-vvar.value, 1) < eps:
        break
    vint=vvar.value
    j=j+1
a=np.dot(uint.T, Xint)
d=np.dot(a, vint)
d=np.array(d)
```

```
    uint=np.array(uint)
    vint=np.array(vint)

    return(uint, d, vint, alphav, lambdau)


################################################
##### fspca: Estimate u (sparse) and v (smooth)
################################################
##### X: the data matrix
################################################
##### center=="TRUE": center the data matrix
################################################
##### row=="TRUE": center by row (else; center by column)
################################################
##### nmiss: number of columns that we remove
################################################
##### K : fold
################################################
##### alphav: smoothing penalty for  v
################################################
##### lambdau: sparse penalty for u
################################################
##### N: max number of iteration
################################################
def fspca(X, center, row, nmiss, K, alphav, lambdau, N, eps):
    i=1
    (n, p)=X.shape
```

```
U=np.zeros((n, min(n, p)))
V=np.zeros((p, min(n, p)))
D=np.zeros(min(n, p))
PV=np.zeros(min(n, p))
CPRV=np.zeros(min(n, p)+1)
CPRV[0]=1
CPVE=np.zeros(min(n, p))
VARout=np.zeros(min(n, p))
alp=np.zeros(min(n, p))
lam=np.zeros(min(n, p))

while i <= min(n, p):
    if i==1:
        Xint=X
    else:
        Uprev=U[:, i-2].reshape(n, 1)
        Vprev=V[:, i-2].reshape(p, 1)
        Xint=Xnew-D[i-2]*np.dot(Uprev, Vprev.T)

    (alpout, errors)=getAlpha(Xint, center, row, nmiss, alphav, K, N, eps)
    (lamout, cpv)=getLambda(Xint, center, row, nmiss, K, lambdau, alpout, N, eps)
    (Uest, Dest, Vest, alpout, lamout)=getuv(Xint, center, row, alpout, lamout, N, eps)
    pv=fspcaprop(Xint, center, row, Uest, Dest, Vest)
    U[:, i-1]=Uest[:,0]
    V[:, i-1]=Vest[:,0]
    D[i-1]=Dest
    alp[i-1]=alpout
    lam[i-1]=lamout
    VARout[i-1]=cpv
```

```
        if i==1:
            PV[i-1]=pv
            CPVE[i-1]=1-cpv
            CPRV[i]=cpv
            check=CPVE[i-1]
            check1=CPRV[i]
        else:
            PV[i-1]=(1-np.sum(PV[0:i-1]))*pv
            CPVE[i-1]=(1-np.sum(CPVE[0:i-1]))*(1-cpv)
            CPRV[i]=(1-np.sum(CPVE[0:i]))
            check=CPVE[i-1]
            check1=CPRV[i]

        if check<0.05 or check1<0.05:
            break
        i=i+1
        Xnew=Xint
    return(U[:, range(0, i+1)], D[0:i], V[:, range(0, i+1)], PV[0:i], CPRV[0:i+1], CPVE[0:i+1],
    VARout[0:i+1],i, alp[0:i], lam[0:i])

###### Run SFPCA with simulated signals
def runSFPCA(n, p, snr, d1, d2, d3, u1, u2, u3, v1, v2, v3, center, row, nmiss, K, alphav,
lambdau, N, eps, Ufilename, Dfilename, Vfilename, CPRVfilename,
CPVEfilename, Xfilename, efilename, alpfilename, lamfilename):
    (X, d1, d2, d3, u1, u2, u3, v1, v2, v3, e)=Xgen(d1, d2, d3, u1, u2, u3, v1, v2, v3,
    snr, n, p)
    savefile(X, Xfilename)
    savefile(e, efilename)
    (Uout, Dout, Vout, PV, CPRV, CPVE, VARout, i, alpout, lamout)=fspca(X, center, row, nmiss,
```

```
    K, alphav, lambdau, N, eps)
    mu=d1*u1*v1.T+d2*u2*v2.T+d3*u2*v3.T
    X1=d1*u1*v1.T
    X2=d2*u2*v2.T
    X3=d3*u2*v3.T

    savefile(Uout, Ufilename)
    savefile(Dout, Dfilename)
    savefile(Vout, Vfilename)
    savefile(CPRV, CPRVfilename)
    savefile(CPVE, CPVEfilename)
    savefile(alpout, alpfilename)
    savefile(lamout, lamfilename)
    return(i, X, Uout, Vout, Dout, CPRV, CPVE, alpout, lamout)

###### Run SFPCA with a real dataset
def dataFPCA(X, center, row, nmiss, K, alphav, lambdau, N, eps, Ufilename, Dfilename,
Vfilename, CPRVfilename, CPVEfilename, alpfilename, lamfilename):
    (Uout, Dout, Vout, PV, CPRV, CPVE, VARout, i, alpout, lamout)=fspca(X, center, row, nmiss,
    K, alphav, lambdau, N, eps)
    savefile(Uout, Ufilename)
    savefile(Dout, Dfilename)
    savefile(Vout, Vfilename)
    savefile(CPRV, CPRVfilename)
    savefile(CPVE, CPVEfilename)
    savefile(alpout, alpfilename)
    savefile(lamout, lamfilename)
    return(i, Uout, Vout, Dout, CPRV, CPVE)
```