FUNCTIONAL MULTIDIMENSIONAL SCALING

by

Liting Li

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Mathematics

at

The University of Wisconsin - Milwaukee

May 2022

ABSTRACT

FUNCTIONAL MULTIDIMENSIONAL SCALING

by

Liting Li

The University of Wisconsin - Milwaukee, 2022
Under the Supervision of Professor Daniel Gervini

Multidimensional scaling is an important component in analyzing proximity (similarity or dissimilarity) between objects and plays a key role in creating low-dimensional visualizations of objects. Regardless of the progress in this area, traditional solutions of multidimensional scaling problems are inapplicable to the proximity which change in time. In this dissertation, we focus on dissimilarity instead of similarity. Motivated by the studies of functional data analysis, we extend the current multidimensional scaling techniques and propose a functional method to obtain lower-dimensional smooth representations in terms of time-varying dissimilarities. This method incorporates the smoothness approach of functional data analysis by using cubic B-spline basis functions. The model is also designed to arrive at optimal representations such that dissimilarities evaluated by estimated representations are almost the same as original dissimilarities of objects in a low dimension which is easier for people to recognize. We verify the feasibility of the model by running simulations, as well as using the closing prices of the S&P 500 stocks as a real case to analyze their dissimilarities. This case study reconstructs the 500 stocks with this functional multidimensional scaling method and provides us a good visualization on a 2D map for the 500 stocks so that we can see how their dissimilarities change smoothly in

each month of the year 2018. Following the analysis of all of the 500 stocks, the cluster analysis of the first 15 stocks is displayed based on some conditions so that it helps us see how the stocks move from month to month and offers a new tool to cluster the stocks in the future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The development of analyzing relative positions of objects has been prevalent in cluster analysis and dimensionality reduction with complicated data. Various methods have a prominent place in these two areas, including factor analysis, linear discriminant analysis, t-distributed stochastic neighbor embedding, etc. Multidimensional scaling (MDS) is also one of the critical methods which contains a family of statistical methods to provide a visual representation of proximities (similarities, dissimilarities, or distances) among a set of objects. For simplicity, we only focus on dissimilarities in this dissertation. By using MDS methods, we can see that objects which are more similar stay closer to each other, while objects which are less similar stay further from each other. The main goal of MDS is to represent objects in a low dimensional space without changing their original dissimilarities. Constructing a low dimensional space for objects is a major difference between MDS and many other dimensional reduction methods (for example, Principal Components Analysis). MDS starts with dissimilarities instead of a feature representation. In other words, unlike some other dimensionality reduction methods, MDS does not have a space at the beginning, but it rearranges objects and induces a space from dissimilarities. Thus, in addition to representing relationships between objects, MDS is also used to create low-dimensional space (usually, one, two, or three dimensions) for them. High-dimensional data causes substantial problems such as data sparsity, intractable computation and even out of human's visual perception, therefore, dimensionality reduction comes into play. Dimensionality reduction in MDS helps preserve dissimilarities between objects as much as possible when it rearranges objects in a low-dimensional space. Extensive research has shown

that MDS plays an important role in various areas, including product differentiation in marketing, credit card fraud detection, psychology analysis, classification of different types of pollution, etc.

The idea of MDS originated from the work made by Young and Householder (1938), but Torgerson (1952, 1958) was the first one to bring out a practical application of classical metric scaling based on quantitative data in psychophysics and sensory analysis. Since then, a considerable amount of literature has been published on the techniques in MDS. Gower (1966) established the formulation of the classical metric scaling technique and gave it the name "Principal Coordinates Analysis" which is now known as classical multidimensional scaling (classical MDS). On the other hand, Shepard (1962) and Kruskal (1964) developed classical nonmetric multidimensional scaling for qualitative data. Shepard provided an iterative approach, while Kruskal delivered analytic solutions. Numerous scholars have driven the further development of MDS based on these fundamentals. Previous studies of MDS, for instance, the algorithm designed by Guttman (1968) includes unidimensional scaling which represents objects in only one dimension. Sammon (1969) suggested a special case of metric scaling, which is called Sammon nonlinear mapping, to study pattern recognition. In order to deal with various forms of data, Takane (1977) developed alternating least squares scaling (ALSCAL). Critchley's intermediate method was provided by Critchley (1978) which transforms dissimilarities with a continuous parametric function with a combination of the classical MDS and metric least squares scaling. More recently, there are further aspects of MDS having been developed. For example, Schneider (1992) combined metric MDS and nonmetric MDS in a continuum parameter with the logistic function transforming dissimilarities. Chen and Chen (2000) carried out interactive

diagnostic plots for MDS in a computer program via the concept of color-linkage. Many of the statistical theories and methods also contribute to MDS. For example, maximum likelihood MDS methods were originally developed by Ramsay (1977, 1982), who is also recognized as one of the crucial founders of functional data analysis. After that, MDS becomes a growing area in Statistics. Hoffman and Buhmann (1994) applied MDS to pairwise clustering process in the maximum entropy framework. Williams (2000) described the relationship between the kernel PCA algorithm and metric MDS. Oh and Raftery (2001) obtained a Bayesian solution for MDS configuration (BMDS) with a Markov chain Monte Carlo algorithm which demonstrates that BMDS provides a much better fit to the data than classical MDS and ALSCAL, and discussed the determination of the dimension of configuration as well.

As essential background, the case of airline-distances analyzed by Izenman (2008) is one of the typical examples to illustrate the application of MDS in the real life. This application respectively recreates 2D and 3D MDS maps of 18 cities based on their airline distances which are shown in Table 1.1.

Table 1.1 Airline distances (km) between 18 cities

	Beijing	Cape Town	Hong Kong	Honolulu	London	Melbourne
Cape Town	12947					
Hong Kong	1972	11867				
Honolulu	8171	18562	8945			
London	8160	9635	9646	11653		
Melbourne	9093	10338	7392	8862	16902	
Mexico	12478	13703	14155	6098	8947	13557
Montreal	10490	12744	12462	7915	5240	16730
Moscow	5809	10101	7158	11342	2506	14418
New Delhi	3788	9284	3770	11930	6724	10192
New York	11012	12551	12984	7996	5586	16671
Paris	8236	9307	9650	11988	341	16793
Rio de Janeiro	17325	6075	17710	13343	9254	13227
Rome	8144	8417	9300	12936	1434	15987
San Francisco	9524	16487	11121	3857	8640	12644
Singapore	4465	9671	2575	10824	10860	6050
Stockholm	6725	10334	8243	11059	1436	15593
Tokyo	2104	14737	2893	6208	9585	8159

	Mexico	Montreal	Moscow	New Delhi	New York	Paris
Montreal	3728					
Moscow	10740	7077				
New Delhi	14679	11286	4349			
New York	3362	533	7530	11779		
Paris	9213	5522	2492	6601	5851	
Rio	7669	8175	11529	14080	7729	9146
Rome	10260	6601	2378	5929	6907	1108
S.F.	3038	4092	9469	12380	4140	8975
Singapore	16623	14816	8426	4142	15349	10743
Stockholm	9603	5900	1231	5579	6336	1546
Tokyo	11319	10409	7502	5857	10870	9738

	Rio	Rome	S.F.	Singapore	Stockholm
Rome	9181				
S.F.	10647	10071			
Singapore	15740	10030	13598		
Stockholm	10682	1977	8644	9646	
Tokyo	18557	9881	8284	5317	8193

The 2D and 3D maps of the MDS reconstruction based on the airline distances in Table 1.1 are given in Figure 1.1 and 1.2 as follows, which are displayed in Izenman (2008).

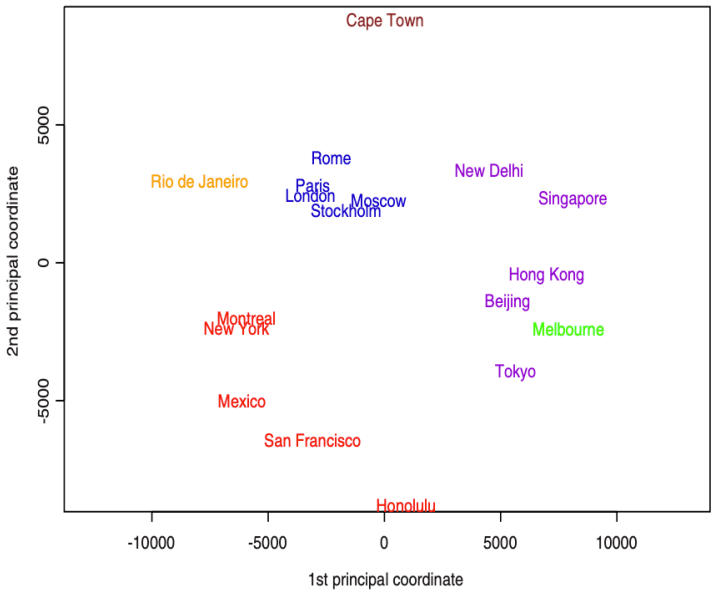

Figure 1.1 2D map of 18 cities by using classical MDS

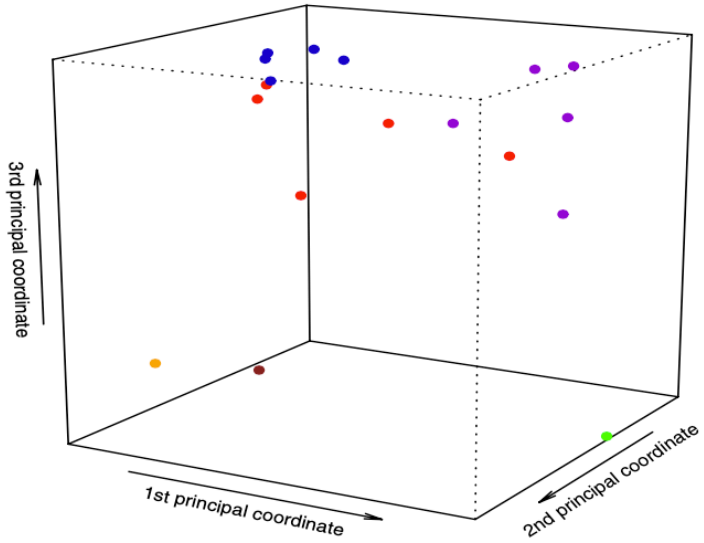

Figure 1.2 3D map of 18 cities by using classical MDS

Both graphs are what we expect to see based on Table 1.1, for example, the airline distance between London and Paris is 341 (km), thus, these two points are closer with each other than the other points in the 2D and 3D graphs. Also, if we look at the airline distances between Cape Town and the other cities, the values are very large, which are also shown by the corresponding positions in the 2D and 3D graphs. The corresponding point of Cape Town stays far away from the other points.

The dissimilarities in the example of airline distances do not change in time, however, dissimilarities are not always static in many real situations. For example, the dissimilarities of closing prices on the S&P 500 stocks can be different every month since the closing prices change everyday. To date, there is some research investigating the feasibility of temporal MDS methods. For instance, Ambrosi and Hansohm (1987) designed a dynamic MDS method regarding dissimilarities measured in consecutive time periods. As one of applications of time-varying MDS in the real world, Machado, Duarte and Duarte (2010) conducted MDS analysis in 15 stock markets over the world with their time-varying correlations. Most recent research also includes the work studied by Jackle, Fischer, Schreck and Keim (2016), which is a temporal MDS visualization technique that creates one-dimensional MDS plots for multivariate time series data in network security. While temporal MDS is a growing field, far too little attention has been paid to smoothness of MDS representations if dissimilarities of objects are considered to change in time.

Considering smooth MDS representations are worth to be analyzed, we take advantage of the techniques in Functional Data Analysis (FDA). The term “Functional Data Analysis” was given by Ramsay (1982), but this area originated in the work by Grenander (1950) and Rao

(1958) in fact. FDA refers to an important field of Statistics that studies samples of functions which are usually determined by smoothing raw data. It enables us to analyze data over a continuum, such as time, age, height, weight, wavelength, spatial location, etc. Thus, objects in FDA are functions instead of numbers. Recent core studies on FDA are made by Ramsay and Silverman (2002, 2006). More details about FDA will be discussed in Chapter 2.

The main purpose of this dissertation is to produce a functional multidimensional scaling model which is a novel MDS method along with time-varying dissimilarities between objects. The methodological approach taken in this study is a method based on functional data analysis. By employing the B-spline basis, we address smoothness of MDS solutions in a low-dimensional configuration.

The dissertation is composed of seven chapters, including this chapter. Chapter 2 contains a review of FDA and applies FDA to smooth the closing prices of a specific stock during the year 2018. Chapter 3 introduces the definition of proximity and detailed mathematical background in classical MDS. Also, we select the closing prices of the S&P 500 stocks from the first two weeks in the year 2018 to have a better understanding of the application of classical MDS. In Chapter 4, we propose the functional multidimensional scaling (FMDS) method for time-varying data. Under the Gaussian assumption for the process that generates the time-varying dissimilarities, the optimal smooth and low-dimensional representations can be estimated. To assess the performance, a simulation study is presented in Chapter 5. Chapter 6 applies the FMDS method to a case study in the stocks of the S&P 500 Index. At the end, the conclusion of the dissertation is stated in Chapter 7.

2 Functional Data Analysis

Functional data consists of independent and identically distributed curves (functions) over a continuous interval. According to the description made by Ramsay and Silverman (2002), we observe a set of n objects consisting the raw data (y_{ij}, t_j) for $i = 1,...,n$ and $j = 1,...,m$, where y_{ij} represents the value of the i^{th} object at the time point t_j, assuming the continuous interval mentioned previously is a time grid. In practice, the observed data is usually contaminated by random noise, resulting from random fluctuations around a smooth trajectory. Ramsay and Silverman (2002) also pointed out that smoothness is the significant assumption in functional data analysis (FDA) so that the consecutive points y_j and y_{j+1} are related to some extent and do not differ a lot from each other. There are two main types of functional data, dense functional data and sparse functional data. Dense functional data is recorded at the same dense time grid of ordered times $\{t_1, \ldots, t_m\}$ and the time gird is equally spaced, i.e., $t_j - t_{j-1} = t_{j+1} - t_j$. When n goes to infinity, $t_{j+1} - t_j$ tends to zero. As for sparse functional data, the time grid varies from objects to objects. In this dissertation, we only focus on the dense functional data and start from spline functions.

2.1 Spline Functions

Suppose that there is a grid (Atkinson, 1988)

$$a = t_0 < t_1 < \ldots < t_S = b.$$

$f(t)$ is a spline function of order $m \geq 1$ if it satisfies the following two properties:

(Property 1) $f(t)$ is a polynomial of degree $< m$ on each subinterval $[t_{i-1}, t_i]$, for $i = 1,...,S$.

(Property 2) The d^{th} derivative $f^{(d)}(t)$ is continuous on $[a, b]$, for $0 \leq d \leq m - 2$.

2.1.1 Order of a Polynomial and Knots

The order of a polynomial (Ramsay and Silverman, 2005) is the number of constants. It is one more than its degree as well. Order of 2, 3, and 4 are the most common situations in polynomials. Ramsay and Silverman (2005) explained it by using $g(t) = sin(t)$ as an example which is displayed in Figure 2.1. The dashed line in Figure 2.1 stands for the graph of $g(t) = sin(t)$, while the three solid lines indicate a piecewise linear spline function (order = 2), a piecewise quadratic spline function (order = 3), and a piecewise cubic spline function (order = 4), respectively. As we can see, a spline function is a useful tool to smooth data.

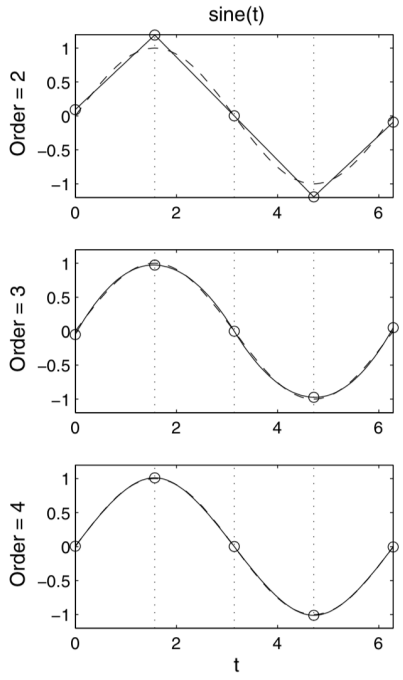

Figure 2.1 The spline functions have order 2, 3, and 4 (from top to bottom)

A spline divides into S subintervals separated by values t_l, $l = 1,...,S-1$ which are called knots. It also implies that the number of knots $L = S - 1$. Each of the above graphs shows that a spline function $f(t)$ divides into $S = 4$ subintervals by three knots t_1 (between 0 and 2), t_2 (between 2 and 4) and t_3 (between 4 and 6).

2.1.2 Degrees of Freedom

The relation between the total number of degrees q of freedom in a spline function, the order of the polynomials m and the number of knots L that was described by Ramsay and Silverman (2005) indicates

$$q = m + L.$$

For instance, the total number of degrees of freedom in the first spline function of Figure 1 is $q = 2 + 3 = 5$.

2.1.3 B-Spline and Basis System of Spline Functions

Ramsay and Silverman (2005) pointed out that a spline function $f(t)$ can be written as

$$f(t) = \sum_{k=1}^{m+L} c_k \phi_k(t),$$

where $\phi_k(t)$'s are basis functions chosen from different basis systems, and c_k's are coefficients which can be obtained by the least squares method in the section 2.2. B-spline basis system developed by de Boor (2001) is one of the most powerful basis systems. It is defined as follows.

Suppose that there exists a knot sequence $\{t_s\}$ (de Boor, 2001), the B-spline of order 1 for this knot sequence is defined as

$$B_{s1}(t) = \begin{cases} 1, & \text{if } t_s \le t < t_{s+1}, \\ 0, & \text{otherwise}. \end{cases}$$

with a constraint

$$\sum_s B_{s1}(t) = 1, \text{ for all } t.$$

When $t_s = t_{s+1}$, we have $B_{s1} = 0$. Based on the first-order B-splines, we obtain m^{th}-order ($m > 1$) B-splines by recurrence

$$B_{sm} = w_{sm}B_{s,m-1} + (1 - w_{s+1,m})B_{s+1,m-1},$$

where $w_{sm}(t) = \begin{cases} \frac{t - t_s}{t_{m+s-1} - t_s}, & \text{if } t_s \neq t_{m+s-1}, \\ 0, & \text{otherwise}. \end{cases}$

The cubic B-spline function (with order $m = 4$) is commonly used as the basis functions $\phi_k(t)$.

2.2 Smoothing Functional Data

2.2.1 Representation of Functional Data

At the beginning of Chapter 2, we briefly discuss functional data that is a sample of functions. For the sake of simplicity in the following review, we only consider one observation of functional data instead of a sample of functions. Suppose that there is an observation of the sample functions containing the raw data $\{(y_j, t_j)\}$, for $j = 1,...,m$, where y_j represents the value observed at the time point t_j belonged to a compact interval τ. In practice, the measurement process gives rise to random noise, thus, Ramsay and Silverman (2005) expressed each of the actual y_j data as

$$y_j = x(t_j) + \epsilon_j,$$

where $x(t)$ is the smooth function in the stochastic process with the mean function $\mu(t) = \mathbb{E}(x(t))$ and covariance function $c(s,t) = Cov[(x(s), x(t)]$, if $\mathbb{E}(\int_{t\in\tau} x^2(t)dt) < \infty$, and ϵ_j is the independent and identically distributed (i.i.d) random noise with zero mean and constant variance σ^2. In general, the underlying function $x(t)$ combines with the random noise ϵ_j to smooth the raw observed data y_j.

As mentioned in the section 2.1.3, a spline function can be written as linear combinations of basis functions because no practical limitations appear in the basis function approach. As a useful tool to smooth data, a spline function can be applied to $x(t)$. Accordingly, $x(t)$ can be modeled by

$$x(t) = \sum_{k=1}^{q} c_k \phi_k(t) = \mathbf{c}^\top \boldsymbol{\phi}(t),$$

where $\phi_k(t)$'s are the spline basis functions and $\mathbf{c}$ is a vector of length q consisting the coefficients c_k's. As we know from the section 2.1.2, the order of the polynomials m and the number of knots L on the splines determine the value of q.

2.2.2 Fitting Functional Data by Least Squares

In order to estimate the smooth function $x(t)$, we need to determine the coefficient vector of $\mathbf{c}$ by minimizing the least squares

$$SMSSE(\mathbf{y} \mid \mathbf{c}) = \sum_{j=1}^{m} [y_j - x(t_j)]^2$$

$$= \sum_{j=1}^{m} [y_j - \sum_{k=1}^{q} c_k \phi_k(t_j)]^2 .$$

This can be done as follows.

Let $\mathbf{\Phi}$ be the $m \times q$ matrix containing the values of $\phi_k(t_j)$, for $j = 1,...,m$, then

$$\begin{aligned} SMSSE(\mathbf{y} \mid \mathbf{c}) &= (\mathbf{y} - x(\mathbf{t}))^\top (\mathbf{y} - x(\mathbf{t})) \\ &= (\mathbf{y} - \mathbf{\Phi c})^\top (\mathbf{y} - \mathbf{\Phi c}) \\ &= \mathbf{y}^\top \mathbf{y} - \mathbf{c}^\top \mathbf{\Phi}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{\Phi c} + \mathbf{c}^\top \mathbf{\Phi}^\top \mathbf{\Phi c}, \end{aligned}$$

where $\mathbf{y} = [y_1, \ldots, y_m]^\top$. By taking the partial derivative of $SMSSE(\mathbf{y} \mid \mathbf{c})$ with respect to $\mathbf{c}$, we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{c}} (\mathbf{y}^\top \mathbf{y} - \mathbf{c}^\top \mathbf{\Phi}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{\Phi c} + \mathbf{c}^\top \mathbf{\Phi}^\top \mathbf{\Phi c}) &= 0 - \mathbf{\Phi}^\top \mathbf{y} - \mathbf{\Phi}^\top \mathbf{y} + 2\mathbf{\Phi}^\top \mathbf{\Phi c} \\ &= 2\mathbf{\Phi}^\top \mathbf{\Phi c} - 2\mathbf{\Phi}^\top \mathbf{y}. \end{aligned}$$

Let the above partial derivative be zero, we can estimate $\hat{\mathbf{c}}$ by

$$\hat{\mathbf{c}} = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{y},$$

then the vector of fitted values is obtained by

$$\hat{\mathbf{y}} = \hat{x}(\mathbf{t}) = \mathbf{\Phi}\hat{\mathbf{c}} = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{y} .$$

2.2.3 Choice of the Number of Knots

Since $\mathbf{\Phi}$ is an $m \times q$ matrix containing the values $\phi_k(t_i)$'s of basis spline function, the type of basis spline function and the number of basis functions q decide $\mathbf{\Phi}$. In this dissertation, we choose the cubic B-spline basis function because it is commonly used for spline smoothing. As for the value of q, it depends on the number of knots L on the splines because we know $q = m + L$ from the section 2.1.2. Since we use the cubic B-spline basis function, we have

$m = 4$. On the other hand, there are various methods to decide L, including the bias/variance trade-off criterion, stepwise variable selection, variable-pruning methods, etc. Although there is no gold standard methods for selecting the number of knots L, we will focus on the bias/variance trade-off criterion in this dissertation. The optimal value of L can be chosen by minimizing the mean-squared error

$$MSE[\hat{x}(\mathbf{t})] = Bias^2[\hat{x}(\mathbf{t})] + Var[\hat{x}(\mathbf{t})],$$

where $Bias^2[\hat{x}(\mathbf{t})] = x(\mathbf{t}) - E[\hat{x}(\mathbf{t})]$ and $Var[\hat{x}(\mathbf{t})] = E[\{\hat{x}(\mathbf{t}) - E[\hat{x}(\mathbf{t})]\}^2]$. The larger the value of L, the smaller $Bias^2[\hat{x}(\mathbf{t})]$. The smaller the value of L, the smaller $Var[\hat{x}(\mathbf{t})]$. The equation implies that minimizing MSE is a compromise between minimizing bias and variance. Figure 2.2 illustrates what bias and variance mean in a model.

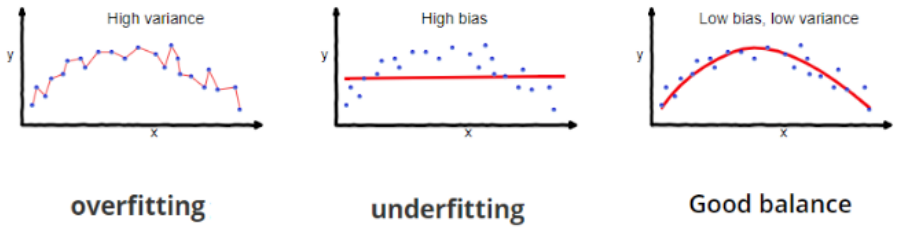

Figure 2.2 The impact of bias and variance on model

The bias of an estimator is the difference between this estimator's expected value and the true value of the parameter being estimated. A model is underfitting when it has high bias, because it does not generalize all data. In contrast, a model is overfitting when it has high variance. The variance of an estimator is the variability of model prediction for a value which tells us spread of our data. A model with high variance pays a lot of attention to training data but does not generalize on the data which it has not seen before. In fact, it is impossible to minimize

the bias and variance simultaneously, thereby, it is allowed to have a little bias if the variance can be reduced a lot. A real case for illustrating the relation between the number of knots on the splines and the mean-squared error of the estimators is shown as follows.

Consider the closing prices of Agilent Technologies in the year 2018. There were 251 trading days. Figure 2.3 is the plot of $MSE(L)$ for the cubic B-spline basis (order = 4) with L knots.

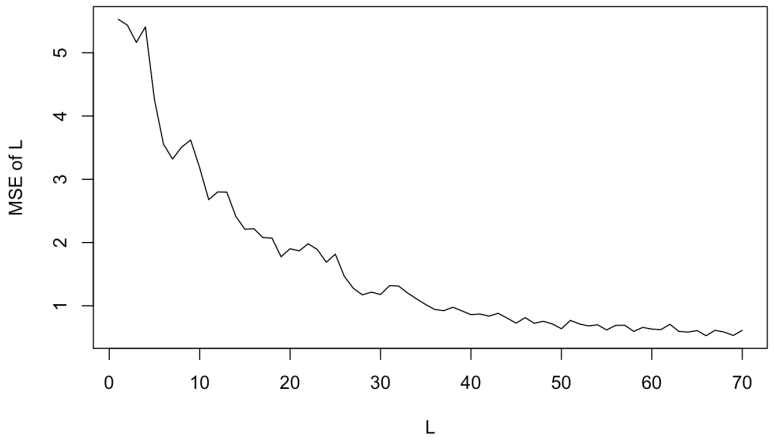

Figure 2.3 L v.s. $MSE(L)$

$MSE(L)$ reaches the minimum value at $L = 66$. Figure 2.4 shows the corresponding stock price $\hat{x}(\mathbf{t})$ with 66 knots over the year 2018.

The model is overfitting because $L = 66$ is too large yielding the large variance. As a result, the estimated curve, which is plotted by a blue curve, is wiggly and not smooth in Figure 2.4. Since the large value of L implies an overfitting model, it is necessary to introduce the regularization approach.

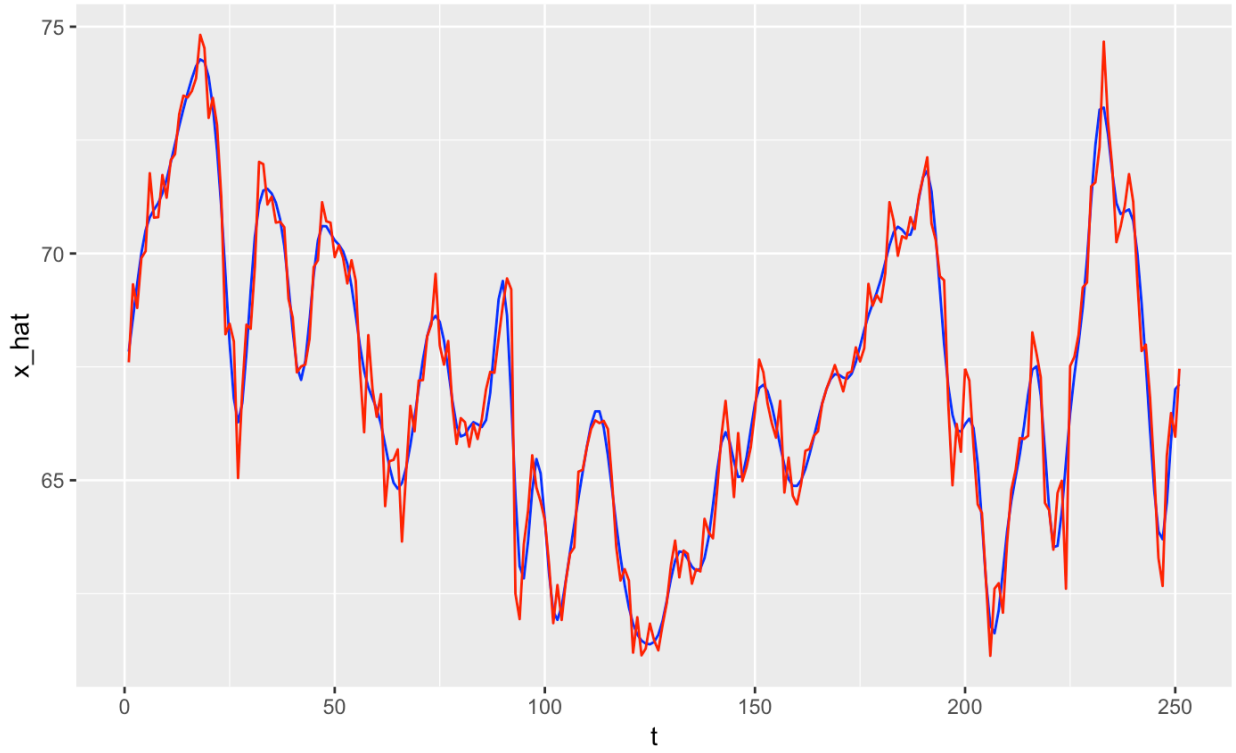

Figure 2.4 The trading day t v.s. the estimated closing price $\hat{x}(t)$

2.2.4 Regularization and Roughness Penalty Approach

Regularization is a technique which avoids the risk of overfitting by using the roughness penalty, resulting in improving smoothness on the estimated curve which can work well on unseen (test) data. Ramsay and Silverman (2005) modified $SMSSE(\mathbf{y}\,|\,\mathbf{c})$ and defined the penalized residual sum of squares as

$$PENSSE_{\lambda}(\mathbf{y}\,|\,\mathbf{c}) = [\mathbf{y} - x(\mathbf{t})]^{\top}[\mathbf{y} - x(\mathbf{t})] + \lambda \times PEN_2(x)$$

$$= (\mathbf{y} - \mathbf{\Phi c})^{\top}(\mathbf{y} - \mathbf{\Phi c}) + \lambda \times PEN_2(x)\,,$$

where λ is a smoothing parameter which controls the trade-off between the data and smoothness, and $PEN_2(x)$ is a roughness penalty that controls the smoothness of $x(\mathbf{t})$. $PEN_2(x)$ is the integrated squared second derivative,

$$PEN_2(x) = \int [D^2 x(s)]^2 \, ds$$

$$= \int [D^2 \mathbf{c}^\top \boldsymbol{\phi}(s)]^2 \, ds$$

$$= \int \mathbf{c}^\top \, D^2\boldsymbol{\phi}(s) \, D^2\boldsymbol{\phi}^\top(s) \, \mathbf{c} \, ds$$

$$= \mathbf{c}^\top \, [\int D^2 \, \boldsymbol{\phi}(s) \, D^2\boldsymbol{\phi}^\top(s) \, ds] \, \mathbf{c}$$

$$= \mathbf{c}^\top \mathbf{R} \mathbf{c} \, ,$$

where $\mathbf{R} = \int D^2\boldsymbol{\phi}(s) \, D^2\boldsymbol{\phi}'(s) \, ds$, an $(n \times n)$ matrix. Rewriting $PENSSE_\lambda$ with the matrix $\mathbf{R}$, we obtain

$$PENSSE_\lambda(\mathbf{y} \,|\, \mathbf{c}) = (\mathbf{y} - \boldsymbol{\Phi}\mathbf{c})^\top (\mathbf{y} - \boldsymbol{\Phi}\mathbf{c}) + \lambda \mathbf{c}^\top \mathbf{R} \mathbf{c} \, .$$

Now the goal is to minimize $PENSSE_\lambda(\mathbf{y} \,|\, \mathbf{c})$. Taking the first derivative with respect to $\mathbf{c}$ and letting the derivative be zero, we have

$$-2\boldsymbol{\Phi}^\top \mathbf{y} + 2\boldsymbol{\Phi}^\top \boldsymbol{\Phi} \mathbf{c} + 2\lambda \mathbf{R} \mathbf{c} = 0 \, .$$

Thus, the estimated coefficient vector is

$$\hat{\mathbf{c}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \mathbf{R})^{-1} \boldsymbol{\Phi}^\top \mathbf{y} \, .$$

The corresponding vector of fitted values is

$$\hat{\mathbf{y}} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \mathbf{R})^{-1} \boldsymbol{\Phi}^\top \mathbf{y} \, .$$

The optimal smoothing parameter λ can be measured by various methods, for instance, the Generalized Cross-Validation (GCV) method that was developed by Craven and Wahba (1979) is popular for determining the smoothing parameter λ.

3 Multidimensional Scaling

Given pairwise proximities (similarities or dissimilarities) of a set of objects, the main use of multidimensional scaling (MDS) in this dissertation is to create a low-dimensional map for the objects that preserves the given proximities. It is worth mentioning that MDS is different from some other dimensionality reduction treatments, for instance, Principal Components Analysis (PCA). PCA summarizes original variables of a set of observations in the minimum number of components so that each component explains the most variance. On the other hand, MDS starts with proximities among objects and creates a low-dimensional space that preserves the same proximities. MDS methods have been evolved in two distinct branches, depending on the type of data. One is metric multidimensional scaling (metric MDS) which deals with quantitative proximities (e.g., interval scale or ratio), while the other is nonmetric multidimensional scaling (nonmetric MDS) if proximities are qualitative (e.g., ordinal). In this chapter, we focus on metric MDS, almost exclusively on classical multidimensional scaling (classical MDS). As a basic concept about proximity, the following section 3.1 is introduced.

3.1 Proximity and Dissimilarity Matrix

Proximity measures the closeness of two objects. It can be similarity or dissimilarity, but we only consider dissimilarity in this dissertation for convenience.

Given a set of n objects (Izenman, 2008), let d_{ij} represents the dissimilarity between the i^{th} and the j^{th} object, satisfying

(a) $d_{ij} \geqslant 0$, for all i, j

(b) $d_{ii} = 0$, for all i

(c) $d_{ij} = d_{ji}$, for all i, j

When dissimilarities are metric distances, namely, quantitative values, there is one more condition required, that is,

(d) $d_{ij} \leqslant d_{is} + d_{sj}$, for all i, j, s.

Dissimilarity can be measured as any kinds of distance metric, including the Euclidean distance, Manhattan distance, Minkowski distance, etc.. In this dissertation, we use the Euclidean distance as dissimilarity and assume that n objects $\mathbf{y_1}, \ldots, \mathbf{y_n} \in \mathbb{R}^r$, then the dissimilarity between $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{ir}]^\top$ and $\mathbf{y}_j = [y_{j1}, y_{j2}, \ldots, y_{jr}]^\top$ is defined as

$$d_{ij} = \|\mathbf{y_i} - \mathbf{y_j}\| = \{\sum_{k=1}^{r} (y_{ik} - y_{jk})^2\}^{1/2},$$

and the squared Euclidean interpoint distance is given by

$$d_{ij}^2 = \|\mathbf{y}_i - \mathbf{y}_j\|^2 = (\mathbf{y}_i - \mathbf{y}_j)^\top (\mathbf{y}_i - \mathbf{y}_j)$$

The dissimilarities $\{d_{ij}\}$ construct an $(n \times n)$ dissimilarity matrix $\Delta = (d_{ij})$. Δ can be displayed as a lower-triangular or upper-triangular matrix because the diagonal entries are all zeros and it is a symmetric matrix.

If we have similarity s_{ij}, we often transform it into dissimilarity. Some possible transformations are listed in Cox (2001)

$$d_{ij} = 1 - s_{ij},$$

$$d_{ij} = c - s_{ij} \text{ (for some constant } c),$$

$$d_{ij} = \{2(1 - s_{ij})\}^{\frac{1}{2}}.$$

3.2 Metric Multidimensional Scaling

Usually, the original data points $\mathbf{y_1}, \ldots, \mathbf{y_n} \in \mathbb{R}^r$ are not given, instead, the dissimilarity matrix $\Delta = (d_{ij})$ of these n objects are provided. The aim of MDS is to find the optimal p -dimensional representation, $\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^p$ $(p < r)$, such that the dissimilarity $d_{ij}^{(x)}$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ satisfies

$$d_{ij}^{(x)} \approx f(d_{ij}),$$

It means that the dissimilarity of the optimal p-dimensional representation, $\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^p$ preserves the dissimilarities of the original data points $\mathbf{y_1}, \ldots, \mathbf{y_n} \in \mathbb{R}^r$. In the above equation, $f(d_{ij})$ is a parametric monotonic function of d_{ij}, such as $f(d_{ij}) = \alpha + \beta d_{ij}$, where α and β are unknown positive coefficients. Classical multidimensional scaling (classical MDS) is one of the main methods in metric MDS and will be discussed as follows.

3.2.1 Classical Multidimensional Scaling

Classical MDS was first given by Young and Householder (1938). In classical MDS, let $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$, we assume a centered configuration, i.e.

$$\sum_{i=1}^{n} x_{ik} = 0, \text{ for all } k = 1,2,...,p. \tag{3.1}$$

In classical MDS, there are two conditions required. Firstly, the dissimilarities are measured by Euclidean distances. Secondly, the parametric function $f(d_{ij})$ is taken to be the identity function, namely, $f(d_{ij}) = d_{ij}$. Thus, classical MDS is considered as an example of metric LS scaling. It is described by Ramsay and Silverman (2005) as follows.

Suppose that there is an $(n \times n)$ dissimilarity matrix $\Delta = (d_{ij})$ of n objects $\mathbf{y_1}, \ldots, \mathbf{y_n} \in \mathbb{R}^r$. Now the goal is to find a configuration of points $\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^p$ $(p < r)$ such that their Euclidean distances $\{d_{ij}^{(x)}\}$ satisfy

$$d_{ij}^{(x)} = d_{ij},$$

$$\text{i.e., } \|\mathbf{x}_i - \mathbf{x}_j\| = d_{ij}. \tag{3.2}$$

However, such a solution is not unique. It can be proved by considering an orthogonal matrix $\mathbf{\Gamma}$ and an arbitrary vector $\mathbf{c}$ which form a transformation of $\mathbf{x}_i$ and $\mathbf{x}_j$: $\mathbf{x}_i \to \mathbf{\Gamma}\mathbf{x}_i + \mathbf{c}$ and $\mathbf{x}_j \to \mathbf{\Gamma}\mathbf{x}_j + \mathbf{c}$. Thus, we have

$$\begin{aligned} \|(\mathbf{\Gamma}\mathbf{x}_i + \mathbf{c}) - (\mathbf{\Gamma}\mathbf{x}_j + \mathbf{c})\|^2 &= \|\mathbf{\Gamma}(\mathbf{x}_i - \mathbf{x}_j)\|^2 \\ &= (\mathbf{x}_i - \mathbf{x}_j)^\top \, \mathbf{\Gamma}^\top \, \mathbf{\Gamma}(\mathbf{x}_i - \mathbf{x}_j) \\ &= \|\mathbf{x}_i - \mathbf{x}_j\|^2 \end{aligned}$$

By taking the squares of (3.2), we get

$$\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^\top \mathbf{x}_j = d_{ij}^2.$$

Let $b_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ and $d_{i0}^2 = \|\mathbf{x}_i\|^2$, then

$$b_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i0}^2 - d_{j0}^2). \tag{3.3}$$

The constraints $\sum_{i=1}^{n} x_{ik} = 0$, for all $k = 1,2,...,p$, lead to

$$\sum_{i=1}^{n} b_{ij} = \sum_{i=1}^{n}\sum_{k=1}^{p} x_{ik}x_{jk} = \sum_{k=1}^{p} x_{jk}\sum_{i=1}^{n} x_{ik} = 0, \tag{3.4}$$

$$\sum_{j=1}^{n} b_{ij} = \sum_{j=1}^{n}\sum_{k=1}^{p} x_{ik}x_{jk} = \sum_{k=1}^{p} x_{ik}\sum_{j=1}^{n} x_{ik} = 0. \tag{3.5}$$

Summing $d_{ij}^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^\top\mathbf{x}_j$ over i and over j and combining the equations (3.4) and (3.5) yields the following equations:

$$n^{-1}\sum_{i} d_{ij}^2 = n^{-1}\sum_{i} d_{i0}^2 + d_{j0}^2, \tag{3.6}$$

$$n^{-1}\sum_{j} d_{ij}^2 = d_{i0}^2 + n^{-1}\sum_{j} d_{j0}^2, \tag{3.7}$$

$$n^{-2}\sum_{i}\sum_{j} d_{ij}^2 = 2n^{-1}\sum_{i} d_{i0}^2. \tag{3.8}$$

Substituting (3.6) - (3.8) into (3.3), we have

$$\begin{aligned} b_{ij} &= -\frac{1}{2}(d_{ij}^2 - d_{i0}^2 - d_{j0}^2) \\ &= -\frac{1}{2}d_{ij}^2 + \frac{1}{2}d_{i0}^2 + \frac{1}{2}d_{j0}^2 + \frac{1}{2n}(\sum_{j} d_{j0}^2 - \sum_{i} d_{i0}^2) \\ &= -\frac{1}{2}d_{ij}^2 + \frac{1}{2}(d_{i0}^2 + \frac{1}{n}\sum_{j} d_{j0}^2) + \frac{1}{2}(d_{j0}^2 + \frac{1}{n}\sum_{i} d_{i0}^2) - \frac{1}{n}\sum_{i} d_{i0}^2 \end{aligned}$$

$$= -\frac{1}{2}d_{ij}^2 + \frac{1}{2n}\sum_j d_{ij}^2 + \frac{1}{2n}\sum_i d_{ij}^2 - \frac{1}{2n^2}\sum_i\sum_j d_{ij}^2. \qquad (3.9)$$

Let $a_{ij} = -\frac{1}{2}d_{ij}^2$, $a_{i.} = \frac{1}{n}\sum_j a_{ij}$, $a_{.j} = \frac{1}{n}\sum_i a_{ij}$, and $a_{..} = \frac{1}{n^2}\sum_i\sum_j a_{ij}$, then (3.9) becomes

$$b_{ij} = a_{ij} - a_{i.} - a_{.j} + a_{..}$$

a_{ij}'s form an $n \times n$ matrix $\mathbf{A} = (a_{ij})$ and b_{ij}'s form $n \times n$ symmetric matrix $\mathbf{B} = (b_{ij})$, then

$$\mathbf{B} = \mathbf{HAH},$$

where $\mathbf{H} = \mathbf{I}_n - n^{-1}\mathbf{J}_n$ and $\mathbf{J}_n = \mathbf{1}_n\mathbf{1}_n^\top$ is an $n \times n$ matrix of ones. The matrix $\mathbf{B}$ is stated as a "double centered" version of $\mathbf{A}$. Let the eigenvalues of $\mathbf{B}$ be $\lambda_1, \ldots, \lambda_n$ and the eigenvectors be $\mathbf{v}_1, \ldots, \mathbf{v}_n$, then denote

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{pmatrix},$$

$$\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_n),$$

whose columns are the eigenvectors of $\mathbf{B}$. According to the spectral theorem,

$$\mathbf{B} = \mathbf{V}\Lambda\mathbf{V}^\top \qquad (3.10)$$

If $\mathbf{B}$ is positive semi-definite with rank $r(\mathbf{B}) = p < n$, then the largest p eigenvalues will be positive and the remaining $n - p$ eigenvalues will be zero. Suppose that $\lambda_1, \ldots, \lambda_p$ are the largest p eigenvalues and $\mathbf{v}_1, \ldots, \mathbf{v}_p$ are their corresponding eigenvectors, we construct two matrices

$$\mathbf{\Lambda}_1 = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda_p \end{pmatrix},$$

$$\mathbf{V}_1 = (\mathbf{v}_1, \dots, \mathbf{v}_p) .$$

We can reduce (3.10) to

$$\mathbf{B} = \mathbf{V}_1 \mathbf{\Lambda}_1 \mathbf{V}_1^\top = (\mathbf{V}_1 \mathbf{\Lambda}_1^{1/2})(\mathbf{\Lambda}_1^{1/2} \mathbf{V}_1^\top). \tag{3.11}$$

On the other hand, since $b_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$, then the matrix $\mathbf{B}$ can be expressed as

$$\mathbf{B} = \mathbf{X}\mathbf{X}^\top, \tag{3.12}$$

where $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top$. According to (3.11) and (3.12), we have

$$\mathbf{X} = \mathbf{V}_1 \mathbf{\Lambda}_1^{1/2} = (\sqrt{\lambda_1}\mathbf{v}_1, \dots, \sqrt{\lambda_p}\mathbf{v}_p) = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top,$$

where $\mathbf{x}_i$ is a $(p \times 1)$-vector for all $i = 1,...,n$. The vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ are the principal coordinates. They are the n points in p-dimensional space whose distances are $\|\mathbf{x}_i - \mathbf{x}_j\| = d_{ij}$. It means that $\mathbf{x}_1, \dots, \mathbf{x}_n$ are the solutions to the optimal representation in classical MDS.

3.3 Example for Classical Multidimensional Scaling

As mentioned in the section 2.1, dissimilarities do not have to be the Euclidean distances. In this example, we apply the algorithm of classical MDS to the daily closing prices of the S&P 500 stocks, but compute the dissimilarities with the correlation instead of the Euclidean distances. In a given week t, we observe the daily closing prices of stocks i and j, which are y_{ik} and y_{jk} , for

$i, j = 1,2,...,500$ and $k = 1,...,r$, where r is the number of trading days in the week. Thus, their correlation R_{ij} in the week t is given by

$$R_{ij} = \frac{\sum_{k=1}^{r}(y_{ik} - \bar{y}_i)(y_{jk} - \bar{y}_j)}{\sqrt{\sum_{k=1}^{r}(y_{ik} - \bar{y}_i)^2}\sqrt{\sum_{k=1}^{r}(y_{jk} - \bar{y}_j)^2}},$$

where $\bar{y}_i = \frac{1}{r}\sum_{k=1}^{r} y_{ik}$ and $\bar{y}_j = \frac{1}{r}\sum_{k=1}^{r} y_{jk}$.

Define the dissimilarities of the close prices as

$$d_{ij} = \frac{1 - R_{ij}}{2}.$$

Thus, the dissimilarity matrix $\mathbf{D}$ can be formed with these dissimilarities $\{d_{ij}\}$

$$\mathbf{D} = \begin{pmatrix} 0 & d_{12} & \dots & d_{1,500} \\ d_{21} & 0 & \dots & d_{2,500} \\ \vdots & \vdots & \dots & \vdots \\ d_{500,1} & d_{500,2} & \dots & 0 \end{pmatrix},$$

where $\begin{cases} d_{ij} = d_{ji} & \text{for } i \neq j, k = 1,...,m, \\ d_{ii} = 0, & \text{for all } i, k = 1,...,m. \end{cases}$

Consider the first trading week of the year 2018. Implementing the classical MDS method with the above dissimilarity matrix $\mathbf{D}$, we obtain the 2-dimensional ($p = 2$) and 3-dimensional ($p = 3$) plots of the stock closing prices in MDS reconstruction. Each plot has 500 points, where one for each stock.

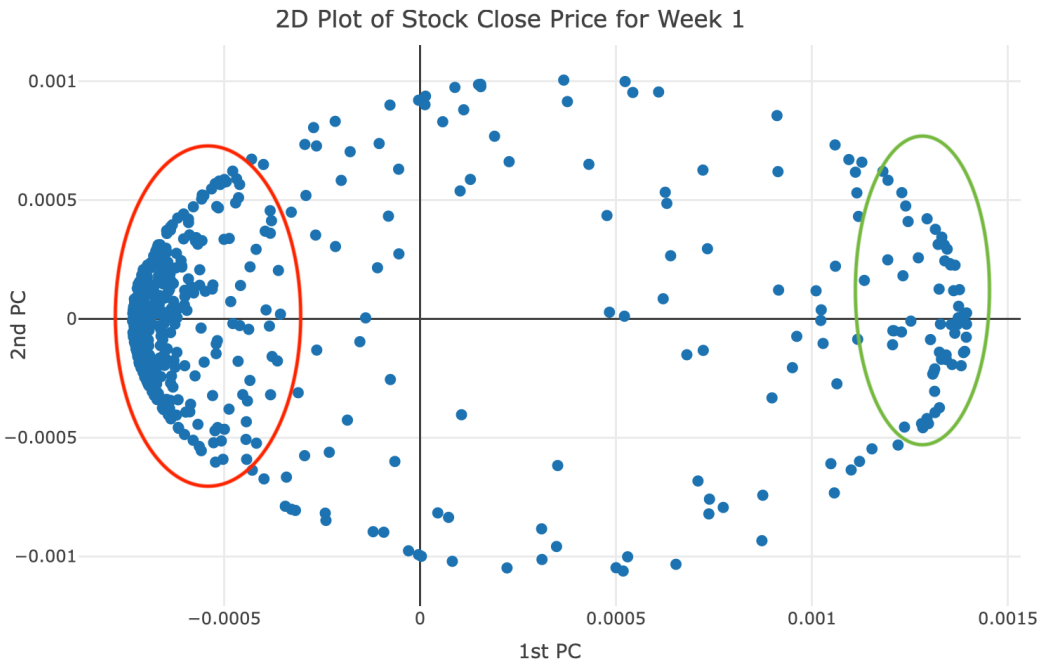

Figure 3.1 2D plot of the 500 stock closing prices in the 1^{st} week of Year 2018

Figure 3.1 displays the plot of the 2D MDS solution. Although some points are sparsity, there are two obvious clusters of the stocks which are circled by red and green. The red cluster has much more stocks than the green cluster. The points which are close together indicate the corresponding stocks having small dissimilarities (large correlation). In contrast, the points which are far apart from each other indicate the corresponding stocks having large dissimilarities (small correlation). But there are some stocks in the middle part not having obvious clusters which means they have large dissimilarities from others. We expect the same situation in the 3D MDS solution for this week.

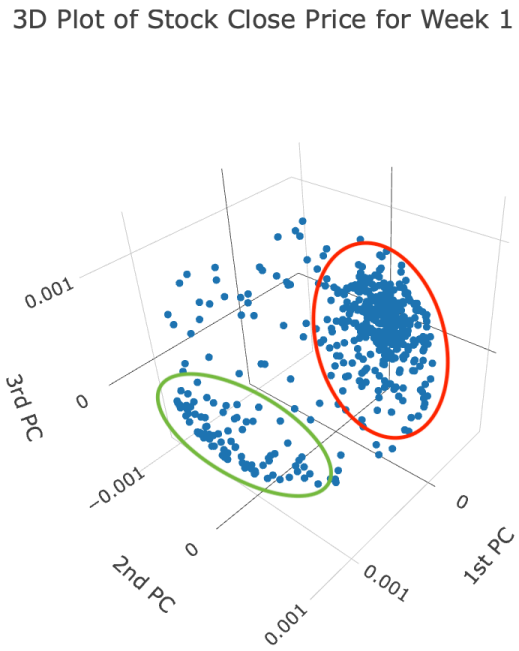

Figure 3.2 3D plot of the 500 stock closing prices in the 1^{st} week of Year 2018

Figure 3.2 is the corresponding 3D MDS reconstruction and shows the same situation as Figure 3.1. There are also two obvious clusters (red and green), where the red cluster has much more stocks than the green cluster. But the rest of the stocks do not have obvious clusters which means they have large dissimilarities from the others.

In these two plots, we can see that classical MDS provides a good visualization of the closing prices of the S&P 500 stocks in a fixed week. Furthermore, we want to see how the situation changes from the first trading week to the second trading week of the year 2018 . Thus, we create the 2-dimensional ($p = 2$) and 3-dimensional ($p = 3$) plots for the second trading week as shown in Figure 3.3 and 3.4, respectively.

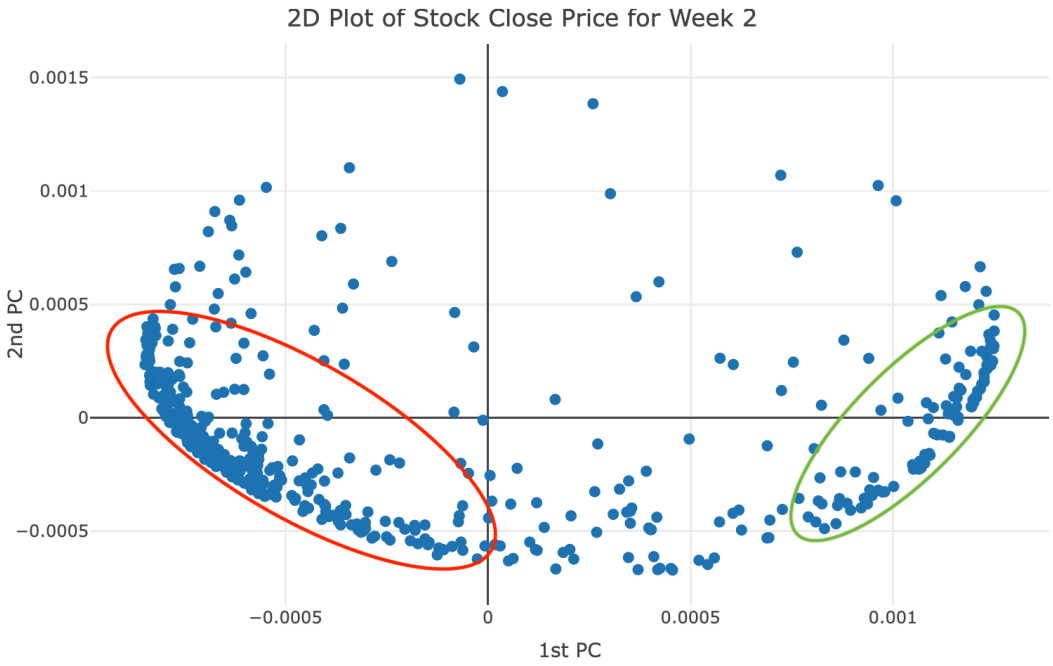

Figure 3.3 2D plot of the 500 stock closing prices in the 2^{nd} week of Year 2018

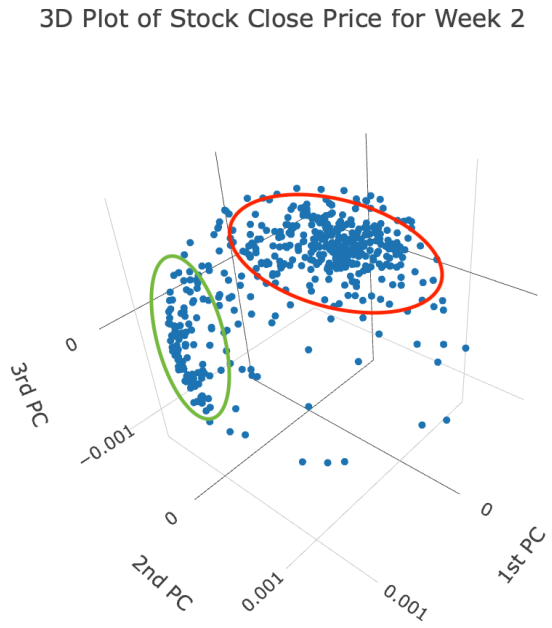

Figure 3.4 3D plot of the 500 stock closing prices in the 2^{nd} week of Year 2018

Although the pattern of the points seems different from the first week to the second week, the clusters actually do not change much from week to week. The two different vertical axes make the different visual effects somehow. In general, the plots are still similar. There are two obvious clusters of the stocks (circled by red and green) in which one cluster has much more stocks than the other cluster.

However, the above MDS solutions are only evaluated by the fixed and static time (week), and we do not obtain the MDS configuration changing in time. In other words, the MDS solutions are discrete, but not continuous during a time interval. In order to solve this issue and provide a smooth MDS representation over time, we propose the method in Chapter 4.

4 Functional Multidimensional Scaling

In this chapter, we propose a method regarding the time-varying multidimensional scaling. In some situations, dissimilarities are not fixed and static, but change in time. Assume that there are n objects $\mathbf{y}_1(t), \ldots, \mathbf{y}_n(t) \in \mathbb{R}^r$ at the time point t, and the dissimilarities between objects i and j are denoted by $d_{ij}(t)$. To be clarified, $d_{ij}(t)$ is not a function of t but the dissimilarity at the time point t. If we apply classical MDS to each discrete $d_{ij}(t)$, we can reconstruct the objects in a lower dimensional space, for example, they are $\mathbf{x}_1(t), \ldots, \mathbf{x}_n(t) \in \mathbb{R}^p$ $(p < r)$. However, $\mathbf{x}_i(t)$'s are not smooth functions of t, given the way they are constructed from eigenvectors, since there is no guarantee that the eigenvectors will change smoothly from time t_j to t_{j+1}, for each t in a continuous interval τ. Hence, classical MDS has failed to address the issues related to time-varying MDS if we seek for a smooth MDS solution.

4.1 Some Background on Time-Varying MDS

Up to now, no large-scale studies have been performed to investigate the MDS solutions with time-varying MDS. The work made by Ambrosi and Hansohm (1987) is one of the few literature about time-varying MDS. They brought out a dynamic MDS approach for producing an MDS reconstruction of n objects, given their dissimilarities measured at a consecutive time period. Their study is based on the loss function for nonmetric MDS at the time point t, where $t = 1,...,T$. Given d_{ij}^t as the dissimilarity between objects i and j, the distances $\hat{d}_{ij}^t$ among the

optimal dynamic MDS solutions $\mathbf{x}_i^t = [x_{i1}^t, \ldots, x_{ip}^t]^\top \in \mathbb{R}^p$ is obtained by minimizing the overall loss function with a penalty function U for the whole time period.

$$S_\epsilon = \frac{\sum_{t=1}^{T} \sum_{i<j} (d_{ij}^t - \hat{d}_{ij}^t)^2}{\sum_{t=1}^{T} \sum_{i<j} (d_{ij}^t - \bar{d}^t)^2} + \epsilon U,$$

where $\bar{d} = \dfrac{2}{n(n-1)} \sum_{i<j} \hat{d}_{ij}^t$ and ϵ is a constant satisfying $0 < \epsilon \ll 1$. The penalty function U can be defined by

$$U = \sum_{t=1}^{T-1} \sum_{i=1}^{n} \sum_{k=1}^{p} (x_{ik}^{t+1} - x_{ik}^t)^2.$$

However, this method leads to drastic change during the time period for each object. Cox and Cox (2001) carried out another dynamic MDS method for each time point separately by using the Procrustes analysis (Sibson, 1978) to obtain a lower-dimensional MDS configuration. As a result, the time trajectories of each object do not change as drastically as the solution found by Ambrosi and Hansohm, but the resulting MDS solutions were still not smooth as the time changes.

4.2 The Optimal Functional MDS Representations

In order to obtain smoothly time-varying representation $\mathbf{x}_i(t)$'s, we come up with a functional multidimensional scaling method (FMDS) such that

$$\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\| \approx d_{ij}(t), \text{ for } i, j = 1,...,n,$$

where $d_{ij}(t)$ is the dissimilarities between the given objects $\mathbf{y}_i(t)$ and $\mathbf{y}_j(t)$. Since B-spline basis functions contribute smoothness to functional data, we let $\mathbf{x}_i(t) = \mathbf{C}_i\boldsymbol{\beta}(t)$, where $\boldsymbol{\beta}(t)$ is the vector of q B-spline basis functions, and each $\mathbf{C}_i$ is the $(p \times q)$ coefficient matrix, for $i = 1,...,n$. As a review from the section 2.1.2, the value of q equals to the number of knots on B-splines plus the order of the B-spline functions. Hence, we can estimate $\mathbf{C}_1, \ldots, \mathbf{C}_n$ by minimizing the target function

$$F(\mathbf{C}_1, \ldots, \mathbf{C}_n) = \sum_{i<j}^{n} \sum_{k=1}^{m} \{d_{ij}^2(t_k) - \|\mathbf{x}_i(t_k) - \mathbf{x}_j(t_k)\|^2\}^2$$

$$= \sum_{i<j}^{n} \sum_{k=1}^{m} \{d_{ij}^2(t_k) - \|\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k)\|^2\}^2$$

$$= \sum_{i<j}^{n} \sum_{k=1}^{m} \{d_{ij}^2(t_k) - (\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))\}^2, \tag{4.1}$$

where m is the length of time period with a continuous interval τ.

The minimization can be achieved by letting the partial derivatives of $\mathbf{C}_h$ be zero for $h \neq j$, where $j = 1,...,n$, i.e., $\frac{\partial F}{\partial \mathbf{C}_h} = \mathbf{0}$ which is a $(p \times q)$ matrix. Thereby, we have the partial derivatives as follows

$$\frac{\partial F}{\partial \mathbf{C}_h} = \frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m} [d_{h1}^2(t_k) - (\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_1\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_1\boldsymbol{\beta}(t_k))]^2\Big)$$

$$+\frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m} [d_{h2}^2(t_k) - (\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_2\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_2\boldsymbol{\beta}(t_k))]^2\Big) + \cdots$$

$$+\frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m}[d^2_{h,h-1}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h-1}\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h-1}\boldsymbol{\beta}(t_k))]^2\Big)$$

$$+\frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m}[d^2_{h,h+1}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h+1}\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h+1}\boldsymbol{\beta}(t_k))]^2\Big)$$

$$+\frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m}[d^2_{h,h+2}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h+2}\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_{h+2}\boldsymbol{\beta}(t_k))]^2\Big)+\cdots$$

$$+\frac{\partial}{\partial \mathbf{C}_h}\Big(\sum_{k=1}^{m}[d^2_{hn}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_n\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_n\boldsymbol{\beta}(t_k))]^2\Big)$$

$$=\sum_{\substack{j=1\\ j\neq h}}^{n}\sum_{k=1}^{m}\frac{\partial}{\partial \mathbf{C}_h}[d^2_{hj}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))]^2)$$

$$=-4\sum_{\substack{j=1\\ j\neq h}}^{n}\sum_{k=1}^{m}[d^2_{hj}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))](\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))\boldsymbol{\beta}(t_k)^\top.$$

Accordingly, we set

$$\sum_{\substack{j=1\\ j\neq h}}^{n}\sum_{k=1}^{m}[d^2_{hj}(t_k)-(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))](\mathbf{C}_h\boldsymbol{\beta}(t_k)-\mathbf{C}_j\boldsymbol{\beta}(t_k))\boldsymbol{\beta}(t_k)^\top=0.$$

In the above equation, each $\mathbf{C}_h$ cannot be estimated in a closed form. To solve the parameters $\mathbf{C}_h$ out from the equation, we need to use an iterative procedure such as the Quasi-Newton method. Another difficulty in estimating the parameters $\mathbf{C}_h$ is that the computation can be time-consuming when the optimization problem is high-dimensional. For example, if

$n = 500$, $p = 2$ and $q = 10$, then the array $\mathbf{C} = (\mathbf{C}_1, \ldots, \mathbf{C}_{500})$ has $500 \times 2 \times 10 = 10000$ elements in total. Suppose that we take 1000 iterations, then there will be $10000 \times 1000 = 10000000$ computations to complete the algorithm. In order to reduce the computation burden, we come up with a method of stochastic gradient descent (SGD).

In practice, many target functions are composed of a sum of sub-functions, such as $F(\mathbf{C}_1, \ldots, \mathbf{C}_n)$ in (4.1). SGD methods complete optimizations by taking gradient steps with respect to the individual sub-functions of target functions. Adam SGD (Kingma and Ba, 2015) is a type of SGD methods which only requires first-order gradients of individual sub-functions with little memory requirement so that it is efficient for the high-dimensional optimization. In our target function (4.1), the individual sub-functions are the functions corresponding to the h^{th} and j^{th} objects as follows

$$f(\mathbf{C}_h, \mathbf{C}_j) = \sum_{k=1}^{m} \{d_{hj}^2(t_k) - \|\mathbf{x}_h(t_k) - \mathbf{x}_j(t_k)\|^2\}^2$$

$$= \sum_{k=1}^{m} \{d_{hj}^2(t_k) - (\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))\}^2,$$

which results in the first-order stochastic gradient functions with respect to $\mathbf{C}_h$ and $\mathbf{C}_j$

$$\frac{\partial f}{\partial \mathbf{C}_h} = -4\sum_{k=1}^{m} [d_{hj}^2(t_k) - (\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))][\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k)]\boldsymbol{\beta}(t_k),$$

$$\frac{\partial f}{\partial \mathbf{C}_j} = 4\sum_{k=1}^{m} [d_{hj}^2(t_k) - (\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))][\mathbf{C}_h\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k)]\boldsymbol{\beta}(t_k).$$

Assembled with Adam SGD, each iteration i updates each pair of $\mathbf{C}_h$ and $\mathbf{C}_j$ with random sample h selected from $\{1, \ldots, n-1\}$ and $j = h+1, h+2, \ldots, n$, respectively, that is,

$$\mathbf{C}_h^{(i+1)} = \mathbf{C}_h^{(i)} - \alpha \frac{\hat{\mathbf{m}}_h^{(i+1)}}{\sqrt{\hat{\mathbf{v}}_h^{(i+1)}} + \mathbf{e}_h},$$

$$\mathbf{C}_j^{(i+1)} = \mathbf{C}_j^{(i)} - \alpha \frac{\hat{\mathbf{m}}_j^{(i+1)}}{\sqrt{\hat{\mathbf{v}}_j^{(i+1)}} + \mathbf{e}_j},$$

where $\alpha = 0.001$, $\hat{\mathbf{m}}_h^{(i+1)} = \dfrac{\mathbf{m}_h^{(i+1)}}{1-\beta_1^{i+1}}$ is obtained by $\mathbf{m}_h^{(i+1)} = \beta_1 \mathbf{m}_h^{(i)} + (1-\beta_1)\dfrac{\partial f}{\partial \mathbf{C}_h}$ and a constant β_1, $\hat{\mathbf{m}}_j^{(i+1)} = \dfrac{\mathbf{m}_j^{(i+1)}}{1-\beta_1^{i+1}}$ is obtained by $\mathbf{m}_j^{(i+1)} = \beta_1 \mathbf{m}_j^{(i)} + (1-\beta_1)\dfrac{\partial f}{\partial \mathbf{C}_j}$ and the constant β_1, $\hat{\mathbf{v}}_h^{(i+1)} = \dfrac{\mathbf{v}_h^{(i+1)}}{1-\beta_2^{(i+1)}}$ is obtained by $\mathbf{v}_h^{(i+1)} = \beta_2 \mathbf{v}_h^{(i)} + (1-\beta_2)\dfrac{\partial f}{\partial \mathbf{C}_h} \circ \dfrac{\partial f}{\partial \mathbf{C}_h}$ and a constant β_2, $\hat{\mathbf{v}}_j^{(i+1)} = \dfrac{\mathbf{v}_j^{(i+1)}}{1-\beta_2^{(i+1)}}$ is obtained by $\mathbf{v}_j^{(i+1)} = \beta_2 \mathbf{v}_j^{(i)} + (1-\beta_2)\dfrac{\partial f}{\partial \mathbf{C}_j} \circ \dfrac{\partial f}{\partial \mathbf{C}_j}$ and the constant β_2, $\mathbf{e}_h$ and $\mathbf{e}_j$ are both $p \times q$ constant matrices. In Adam SGD, α is the step length, β_1 and β_2 are the decay rate parameters. Typically, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\mathbf{e}_h$ and $\mathbf{e}_j$ are both $p \times q$ matrices with all elements equal to 10^{-8}. Besides, the initial $\mathbf{m}_h^{(0)} = \mathbf{0}$ and $\mathbf{v}_h^{(0)} = \mathbf{0}$ are both $p \times q$ matrices, and the initial $\mathbf{m}_j^{(0)} = \mathbf{0}$ and $\mathbf{v}_j^{(0)} = \mathbf{0}$ are both $p \times q$ matrices as well. It is important to note that the random sample of h is without replacement from $\{1, \ldots, n-1\}$ in each for loop. In order to

speed up the convergence of the above algorithm, we initialize $\mathbf{C}_1, \ldots, \mathbf{C}_n$ by using classical MDS with the given dissimilarities d_{ij}'s.

The iteration stops when $\mathbf{C}_1, \ldots, \mathbf{C}_n$ tend to converge. In general, the algorithm is described in Table 4.1. The resulting parameters are denoted by $\hat{\mathbf{C}}_1, \ldots, \hat{\mathbf{C}}_n$ such that the optimal p -dimensional FMDS configurations are derived by $\hat{\mathbf{x}}_1(t) = \hat{\mathbf{C}}_1\boldsymbol{\beta}(t)$, ... , $\hat{\mathbf{x}}_n(t) = \hat{\mathbf{C}}_n\boldsymbol{\beta}(t)$.

Table 4.1 Modified Adam SGD

While $\|\mathbf{C}_h^{(i+1)} - \mathbf{C}_h^{(i)}\| \geq \epsilon$ and $\|\mathbf{C}_j^{(i+1)} - \mathbf{C}_j^{(i)}\| \geq \epsilon$ do

$$\mathbf{v}_h^{(i+1)} = \beta_2 \mathbf{v}_h^{(i)} + (1-\beta_2)\frac{\partial f}{\partial \mathbf{C}_h} \circ \frac{\partial f}{\partial \mathbf{C}_h}, \mathbf{v}_j^{(i+1)} = \beta_2 \mathbf{v}_j^{(i)} + (1-\beta_2)\frac{\partial f}{\partial \mathbf{C}_j} \circ \frac{\partial f}{\partial \mathbf{C}_j}$$

$$\mathbf{m}_h^{(i+1)} = \beta_1 \mathbf{m}_h^{(i)} + (1-\beta_1)\frac{\partial f}{\partial \mathbf{C}_h}, \mathbf{m}_j^{(i+1)} = \beta_1 \mathbf{m}_j^{(i)} + (1-\beta_1)\frac{\partial f}{\partial \mathbf{C}_j}$$

$$\hat{\mathbf{v}}_h^{(i+1)} = \frac{\mathbf{v}_h^{(i+1)}}{1-\beta_2^{(i+1)}}, \hat{\mathbf{v}}_j^{(i+1)} = \frac{\mathbf{v}_j^{(i+1)}}{1-\beta_2^{(i+1)}}$$

$$\hat{\mathbf{m}}_h^{(i+1)} = \frac{\mathbf{m}_h^{(i+1)}}{1-\beta_1^{i+1}}, \hat{\mathbf{m}}_j^{(i+1)} = \frac{\mathbf{m}_j^{(i+1)}}{1-\beta_1^{i+1}}$$

$$\mathbf{C}_h^{(i+1)} = \mathbf{C}_h^{(i)} - \alpha \frac{\hat{\mathbf{m}}_h^{(i+1)}}{\sqrt{\hat{\mathbf{v}}_h^{(i+1)}} + \mathbf{e}_h}, \mathbf{C}_j^{(i+1)} = \mathbf{C}_j^{(i)} - \alpha \frac{\hat{\mathbf{m}}_j^{(i+1)}}{\sqrt{\hat{\mathbf{v}}_j^{(i+1)}} + \mathbf{e}_j}$$

$i = i + 1$

5 Simulation Study

In this chapter, we conduct simulations to evaluate the performance of the proposed FMDS method by assessing the goodness of the approximation of the estimators as the sample size of time period increases. We use the root-mean-square error (RMSE) as the measure of the performance in the simulations and expect to see the RMSE reaches a smaller value as the sample size of time period increases.

5.1 Simulation Settings

In this section, we introduce the simulation settings to compare the RMSE of observed dissimilarities $d_{ij}(t)$ and estimated dissimilarities $\|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\|$. We take $n = 50$ and $p = 2$ as default in the following simulation.

Assume that we want to construct a p-dimensional FMDS representation for n functional objects $\mathbf{y}_1(t), \ldots, \mathbf{y}_n(t)$. Each $\mathbf{y}_i(t) = \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t)$, for $i = 1,...,n$, where $\boldsymbol{\beta}(t)$ is a vector of q B-spline basis functions. We can start from generating the $(p \times q)$ coefficient matrices $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_n^{(y)}$ from a Gaussian. Let $\mathbf{v}_i^{(y)}$ be $vec(\mathbf{C}_i^{(y)})$ and be independent and identically distributed by a multivariate normal distribution $MN(\mathbf{0}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a $(pq \times pq)$ identity matrix so that we can form the coefficient matrices $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_n^{(y)}$.

As for the basis functions $\boldsymbol{\beta}(t)$, we choose the cubic B-spline families with $L = 5$ and $L = 10$ equally spaced knots, respectively. Accordingly, $q = 5 + 4 = 9$ and $q = 10 + 4 = 14$

for the respective models. In each model, we consider four sample sizes for the time period m: 15, 50, 100 and 200. Thus, the simulation performs a total of 8 scenarios. Each scenario is replicated 300 times, which means we run 300 times of the random sample of $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_n^{(y)}$.

Given that the dissimilarities are evaluated by the Euclidean distances, the target function is determined by

$$F(\mathbf{C}_1, \ldots, \mathbf{C}_n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} \{ \|\mathbf{y}_i(t_k) - \mathbf{y}_j(t_k)\|^2 - \|\mathbf{x}_i(t_k) - \mathbf{x}_j(t_k)\|^2 \}^2$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} \{ \|\mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_k) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_k)\|^2 - \|\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_i\boldsymbol{\beta}(t_k)\|^2 \}^2.$$

We do not use a roughness penalty in the above target function because there are only five or ten knots on the B-splines. When the number of knots is not very large, the variance of the estimators is not large. Thus, we do not need a roughness penalty to avoid the overfitting issue. Assume that $\hat{\mathbf{C}}_i$'s are estimated by minimizing the above target function, and let $d_{ij}(t) = \|\mathbf{y}_i(t) - \mathbf{y}_j(t)\|$ be the observed dissimilarities and $\hat{d}_{ij}(t) = \|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\|$ be the estimated dissimilarities. Then for $m = 15, 50, 100, 200$, the mean squared error (MSE) of $d_{ij}(t_k)$ and $\hat{d}_{ij}(t_k)$ for each scenario in one replication can be evaluated as

$$MSE(m)_r = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} (d_{ij}(t_k) - \hat{d}_{ij}(t_k))^2}{mn(n-1)/2}, \tag{5.1}$$

for $r = 1,...,300$. After we run 300 replications, the RMSE of $d_{ij}(t_k)$ and $\hat{d}_{ij}(t_k)$ for each m is evaluated as

$$RMSE(m) = \sqrt{\frac{\sum_{r=1}^{300} MSE(m)_r}{300}}. \tag{5.2}$$

We expect to see that $RMSE(m)$ will decrease as the value of m increases.

On the other hand, we want to assess the goodness of the parameters $\mathbf{C}_i$'s, for $i = 1,...,n$. Since the estimated dissimilarities $\hat{d}_{ij}(t)$'s are rotation and translation invariant, the optimal solutions $\hat{\mathbf{x}}_i(t) = \hat{\mathbf{C}}_i\boldsymbol{\beta}(t)$ are rotation and translation equivalent. For any $p \times p$ orthogonal matrix $\boldsymbol{\Gamma}$, we have

$$\hat{d}_{ij}(t) = \|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\| = \|\boldsymbol{\Gamma}\hat{\mathbf{x}}_i(t) - \boldsymbol{\Gamma}\hat{\mathbf{x}}_j(t)\|,$$

hence, the $\{\boldsymbol{\Gamma}\hat{\mathbf{x}}_i(t)\}$'s provide an approximation to the dissimilarities $d_{ij}(t)$ as good as the $\hat{\mathbf{x}}_i(t)$'s produce, that is, all $\{\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t)\}$'s are equivalent as the optimal FMDS solutions $\hat{\mathbf{x}}_i(t) = \hat{\mathbf{C}}_i\boldsymbol{\beta}(t)$, for $i = 1,...,n$. Accordingly, our another purpose is to find the optimal orthogonal matrix $\hat{\boldsymbol{\Gamma}}$ in each replication such that

$$\mathcal{G}(\boldsymbol{\Gamma}) = \sum_{i=1}^{n}\int_{t=1}^{m} \|\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t)\|^2 \, dt$$

$$= \sum_{i=1}^{n}\int_{t=1}^{m} (\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t))^{\top}(\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t)) \, dt$$

arrives at the minimum. It can be done by implementing a curvilinear search method with the well-known Barzilai-Borwein (BB) step size which was proposed by Wen and Yin (2013). In order to facilitate this algorithm to find the optimization with the orthogonality constraint $\boldsymbol{\Gamma}^{\top}\boldsymbol{\Gamma} = \mathbf{I}$ on RStudio conveniently, we apply the trapezoidal rule to $\mathcal{G}(\boldsymbol{\Gamma})$ so that it becomes

$$\mathcal{G}(\boldsymbol{\Gamma}) = \sum_{i=1}^{n}\{\frac{0.5}{2}[g(1) + 2g(2) + 2g(3) + \ldots + 2g(2m-2) + g(2m-1)]\}$$

$$= \frac{1}{4}\sum_{i=1}^{n}[g(1) + 2g(2) + 2g(3) + \ldots + 2g(2m-2) + g(2m-1)],$$

where $g(k) = (\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_k))^\top(\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_k))$. The gradient function with respect to $\boldsymbol{\Gamma}$ is required in this method, and it is derived as

$$\frac{\partial \mathcal{G}}{\partial \boldsymbol{\Gamma}} = \frac{1}{4}\sum_{i=1}^{n}(2\mathbf{I})(\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_1) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_1))(\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_1))^\top$$

$$+\frac{1}{2}\sum_{i=1}^{n}\sum_{k=2}^{2m-2}(2\mathbf{I})(\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_k))(\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_k))^\top$$

$$+\frac{1}{4}\sum_{i=1}^{n}(2\mathbf{I})(\boldsymbol{\Gamma}\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_{2m-1}) - \mathbf{C}_i^{(y)}\boldsymbol{\beta}(t_{2m-1}))(\hat{\mathbf{C}}_i\boldsymbol{\beta}(t_{2m-1}))^\top.$$

Since the algorithm is an iterative method, the initial guess for the $p \times p$ orthogonal matrix $\boldsymbol{\Gamma}_0$ is needed and generated by the standard Gaussian. If $\boldsymbol{\Gamma}_0$ is not orthogonal, it will be processed by Gram-Schmidt. According to Lemma 1 in Wen and Yin (2013), let $\mathbf{G} = \frac{\partial \mathcal{G}}{\partial \boldsymbol{\Gamma}}$, we define $\nabla\mathcal{G} = \mathbf{G} - \boldsymbol{\Gamma}\mathbf{G}^\top\boldsymbol{\Gamma}$ and $\mathbf{A} = \mathbf{G}\boldsymbol{\Gamma}^\top - \boldsymbol{\Gamma}\mathbf{G}^\top$ which leads to $\nabla\mathcal{G} = \mathbf{A}\boldsymbol{\Gamma}$. Meanwhile, there are several parameters needed in the method, including $\rho_1, \delta, \eta, \epsilon \in (0, 1)$. In the k^{th} iteration, we set τ_k to either

$$\tau_k = \frac{tr((S_{k-1})^\top S_{k-1})}{|tr((S_{k-1})^\top Y_{k-1})|} \text{ or } \tau_k = \frac{|tr((S_{k-1})^\top Y_{k-1})|}{tr((Y_{k-1})^\top Y_{k-1})},$$

where $S_{k-1} = \boldsymbol{\Gamma}_k - \boldsymbol{\Gamma}_{k-1}$ and $\mathbf{Y}_{k-1} = \nabla\mathcal{G}(\boldsymbol{\Gamma}_k) - \nabla\mathcal{G}(\boldsymbol{\Gamma}_{k-1})$. The new orthogonal matrix $\boldsymbol{\Gamma}_{k+1}$ is generated iteratively by the steps in Table 5.1.

Table 5.1 A curvilinear search method with BB steps

while $\|\nabla \mathcal{G}(\mathbf{\Gamma}_k)\| > \epsilon$ do
while $\mathcal{G}(\mathbf{Y}_k(\tau_k)) \geq \mathbf{C}_k + \rho_1 \tau_k \mathcal{G}'(\mathbf{Y}_k(0))$ do
$\tau_k = \tau_k \delta$
$\mathbf{\Gamma}_{k+1} = \mathbf{Y}_k(\tau_k)$, $\mathbf{Q}_{k+1} = \eta \mathbf{Q}_k + 1$, and $\mathbf{C}_{k+1} = (\eta \mathbf{Q}_k \mathbf{C}_k + \mathcal{G}(\mathbf{\Gamma}_{k+1}))/\mathbf{Q}_{k+1}$
Set $\tau_k = max(min(\tau_{k+1,1}, 10^{20}), 10^{-20})$, $k = k + 1$

That is, the above iteration will stop until $\|\nabla \mathcal{G}(\mathbf{\Gamma}_k)\| \leq \epsilon$ and we will have the optimal orthogonal matrix $\hat{\mathbf{\Gamma}}$ which minimizes $\mathcal{G}(\mathbf{\Gamma})$. Based on the optimal equivalent solutions $\{\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i \boldsymbol{\beta}(t)\}$'s in each replication, for $i = 1,...,n$, we are also able to see how the parameter estimators $\mathbf{C}_1, \ldots, \mathbf{C}_n$ perform in the model by computing the RMSE of $\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i$ and $\mathbf{C}_i^{(y)}$ for the 300 replications with five and ten knots, respectively. It can be done as follows. First of all, for each scenario in the replication r, for $r = 1,...,300$, we evaluate the MSE of $\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i$ and $\mathbf{C}_i^{(y)}$ with

$$MSE(\hat{\mathbf{\Gamma}}, m)_r = \frac{\sum_{i=1}^{n} \sum_{j=1}^{pq} (\hat{a}_j^{(i)} - a_j^{(i)})^2}{npq}, \tag{5.3}$$

where $\hat{a}_j^{(i)}$'s are the elements of $vec(\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i)$, and $a_j^{(i)}$'s are the elements of $vec(\mathbf{C}_i^{(y)})$.

After we run 300 replications, we will have the RMSE of $\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i$and $\mathbf{C}_i^{(y)}$ for each m being evaluated as

$$RMSE(\hat{\mathbf{\Gamma}}, m) = \sqrt{\frac{\sum_{r=1}^{300} MSE(\hat{\mathbf{\Gamma}}, m)_r}{300}}. \tag{5.4}$$

Again, we expect to see that $RMSE(\hat{\mathbf{\Gamma}}, m)$ will decrease as the value of m increases.

5.2 Results

Based on the calculation of $RMSE(m)$ in (5.2), we have Table 5.2 which shows that the RMSE of $d_{ij}(t_k)$ and $\|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\|$ decreases as m increases, as expected, for both choices of the number of knots L on the cubic B-splines. Meanwhile, computing $RMSE(\hat{\mathbf{\Gamma}}, m)$ with the simulation settings in (5.4), the RMSE of $\mathbf{C}_i^{(y)}$ and $\hat{\mathbf{\Gamma}}\hat{\mathbf{C}}_i$ also decreases as m increases for both scenarios with five and ten knots. However, the situation with $L = 10$ knots attains estimators more accurate, especially for the estimator $\mathbf{C}_i$. It is attributed to the smaller bias as the number of knots is increasing. Also, comparing the RMSE values with $m = 100$ and $m = 200$, we can see that the RMSE values for both estimators $\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\|$ and $\mathbf{C}_i$ decrease faster from $m = 15$ to $m = 50$ in both scenarios with five and ten knots. The RMSE values tend to relatively stable when $m > 50$ but also continue to decrease as m increases, as expected.

Table 5.2 $RMSE(m)$ and $RMSE(\hat{\mathbf{\Gamma}}, m)$ with $m = 15, 50, 100, 200$ and $L = 5, 10$

		$L = 5$		
Estimator	$m = 15$	$m = 50$	$m = 100$	$m = 200$
$\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\|$	2.198	1.254	1.050	0.972
$\mathbf{C}_i$	0.399	0.306	0.279	0.266
		$L = 10$		
Estimator	$m = 15$	$m = 50$	$m = 100$	$m = 200$
$\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\|$	2.245	0.925	0.628	0.514
$\mathbf{C}_i$	0.622	0.257	0.215	0.211

6 Case Study

As we mentioned in Chapter 1, dissimilarities among objects are usually not fixed and static, but change in time, for example, the dissimilarities of closing prices of the stocks can vary from month to month since the closing prices change everyday. In this chapter, we apply the Functional Multidimensional Scaling (FMDS) method to the 500 stocks that make up the S&P 500 Index in the year 2018.

In the section 3.3, we made an example for the daily closing prices of the stocks in a given week. Now, we change the situation to months and suppose that there are r trading days in the month t, then we denote the daily closing prices for each stock in that month as $\mathbf{y}_i(t) = [y_{i1}(t), \ldots, y_{ir}(t)]^\top$, for $i = 1,...,500$. To be clarified, $\mathbf{y}_i(t)$ is not a function of t, but it only stands for the daily closing prices of the i^{th} stock during the month t. The daily closing prices data was downloaded from the website barchart.com. Accordingly, the correlation of stocks i and j on the month t is given by

$$R_{ij}(t) = \frac{\sum_{k=1}^{r}(y_{ik}(t) - \overline{y_i}(t))(y_{jk}(t) - \overline{y_j}(t))}{\sqrt{\sum_{k=1}^{r}(y_{ik}(t) - \overline{y_i}(t))^2}\sqrt{\sum_{k=1}^{r}(y_{jk}(t) - \overline{y_j}(t))^2}},$$

for $i, j = 1,...,500$, where $\overline{y_i}(t) = \frac{1}{r}\sum_{k=1}^{r} y_{ik}(t)$ and $\overline{y_j}(t) = \frac{1}{r}\sum_{k=1}^{r} y_{jk}(t)$. Instead of using the Euclidean distances, we define the dissimilarities of the closing prices as

$$d_{ij}(t) = \frac{1 - R_{ij}(t)}{2}.$$

Since $-1 \leq R_{ij}(t) \leq 1$, each $d_{ij}(t)$ satisfies $0 \leq d_{ij}(t) \leq 1$. If we do this for 12 months $t_1, \ldots, t_{12}$, we will obtain a series of dissimilarity matrices

$$\mathbf{D}(t_k) = \begin{pmatrix} 0 & d_{12}(t_k) & \ldots & d_{1,500}(t_k) \\ d_{21}(t_k) & 0 & \ldots & d_{2,500}(t_k) \\ \vdots & \vdots & \vdots & \vdots \\ d_{500,1}(t_k) & d_{500,2}(t_k) & \ldots & 0 \end{pmatrix},$$

where $\begin{cases} d_{ij}(t_k) = d_{ji}(t_k), & \text{for } i \neq j, k = 1,...,12, \\ d_{ii}(t_k) = 0, & \text{for all } i, k = 1,...,12. \end{cases}$

Since each $\mathbf{D}(t_k)$ is a symmetric matrix with zeroes in the diagonal, we can vectorize the upper triangular part of $\mathbf{D}(t_k)$ for each k without the diagonal for convenience so that we form a super dissimilarity matrix $\mathbf{D}$ with these vectorized upper triangular parts for the 12 months. As a result, it is constituted by $C_{500}^2 = 124750$ rows for the combinations of two stocks and 12 columns for the months, as follows.

$$\mathbf{D} = \begin{pmatrix} d_{12}(t_1) & d_{12}(t_2) & \ldots & d_{12}(t_{12}) \\ d_{13}(t_1) & d_{13}(t_2) & \ldots & d_{13}(t_{12}) \\ d_{23}(t_1) & d_{23}(t_2) & \ldots & d_{23}(t_{12}) \\ d_{14}(t_1) & d_{14}(t_2) & \ldots & d_{14}(t_{12}) \\ d_{24}(t_1) & d_{24}(t_2) & \ldots & d_{24}(t_{12}) \\ d_{34}(t_1) & d_{34}(t_2) & \ldots & d_{34}(t_{12}) \\ \vdots & \vdots & \vdots & \vdots \\ d_{499,500}(t_1) & d_{499,500}(t_2) & \ldots & d_{499,500}(t_{12}) \end{pmatrix}.$$

Now our goal is to come up with a method to obtain a smoothly time-varying MDS representation $\mathbf{x}_1(t), \ldots, \mathbf{x}_{500}(t) \in \mathbb{R}^p$ $(p < r)$ for the stocks of the S&P 500 Index with each t, such that

$$\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\| \approx d_{ij}(t),$$

for $i = 1,...,499$, $j = 1,...,500$ and $i < j$. Assume $\mathbf{x}_i(t) = \mathbf{C}_i \boldsymbol{\beta}(t)$, where $\boldsymbol{\beta}(t)$ is a vector of q B-spline basis functions, and each $\mathbf{C}_i$ is a $(p \times q)$ coefficient matrix. At the end of the case study, we display the FMDS representation $\mathbf{x}_1(t), \ldots, \mathbf{x}_{500}(t)$ in a 2-dimensional map in order to have a convenient visualization. That is, $p = 2$. Also, we select the first 15 stocks to analyze their clusters in January, February, March and April.

Considering that $\mathbf{D} = [d_{ij}(t_k)]$ is a (124750×12) super dissimilarity matrix and each $\mathbf{x}_i(t)$ is 2-dimensional, we choose a smaller value of q so that we can avoid the time-consuming optimization. On the other hand, it is also unacceptable if $q = m = 12$, because it leads to zero bias of the estimators but very high variance of the estimators. Hence, as a comprise between the bias and variance trade-off, we use $q = 10$, i.e. $L = 6$ on the B-splines. Accordingly, each $\mathbf{C}_i$ is a (2×10) coefficient matrix and $\boldsymbol{\beta}(t)$ is a vector of 10 B-spline basis functions with 6 equally spaced knots in the interval $[1, 12]$.

Since there are only 6 knots used in the basis functions, it is not necessary to add a roughness penalty to the least-square criterion. According to (4.1) in the section 4.2, we can estimate the $\mathbf{C}_i$'s by minimizing

$$F(\mathbf{C}_1, \ldots, \mathbf{C}_{500}) = \sum_{i<j}^{500} \sum_{k=1}^{12} \{d_{ij}^2(t_k) - \|\mathbf{x}_i(t_k) - \mathbf{x}_j(t_k)\|^2\}^2$$

$$= \sum_{i<j}^{500} \sum_{k=1}^{12} \{d_{ij}^2(t_k) - \|\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k)\|^2\}^2$$

$$= \sum_{i=1}^{499} \sum_{j=i+1}^{500} \sum_{k=1}^{12} \{d_{ij}^2(t_k) - (\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))^\top(\mathbf{C}_i\boldsymbol{\beta}(t_k) - \mathbf{C}_j\boldsymbol{\beta}(t_k))\}^2. \quad (6.1)$$

In this case, we use the Adam SGD method mentioned in the section 4.2 to solve the above optimization problem. As a result, the closing price of each stock can be modeled as a smooth function of time t, which is expressed as

$$\hat{\mathbf{x}}_i(t) = \hat{\mathbf{C}}_i\boldsymbol{\beta}(t),$$

for $i = 1,...,500$. Figure 6.1 shows the map of 2D FMDS solutions $\hat{\mathbf{x}}_1(t), \ldots, \hat{\mathbf{x}}_{500}(t)$ for the stocks in the 12 months of the year 2018. Each graph is for one month. We can see that some stocks are clustered in January, February, March, October and December but many stocks are dispersed in the other months.

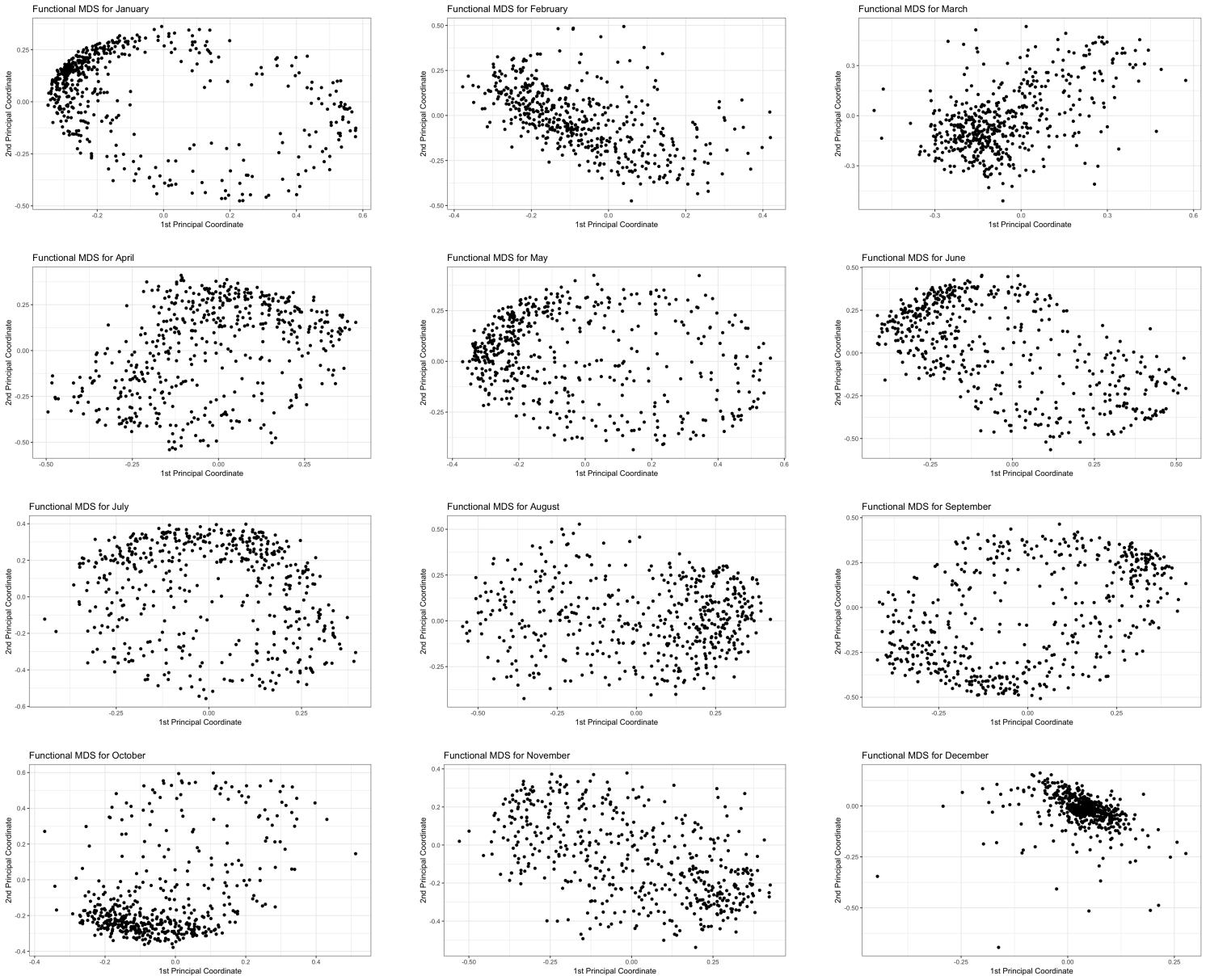

Figure 6.1 2D FMDS maps for the S&P 500 stocks in the 12 months of Year 2018

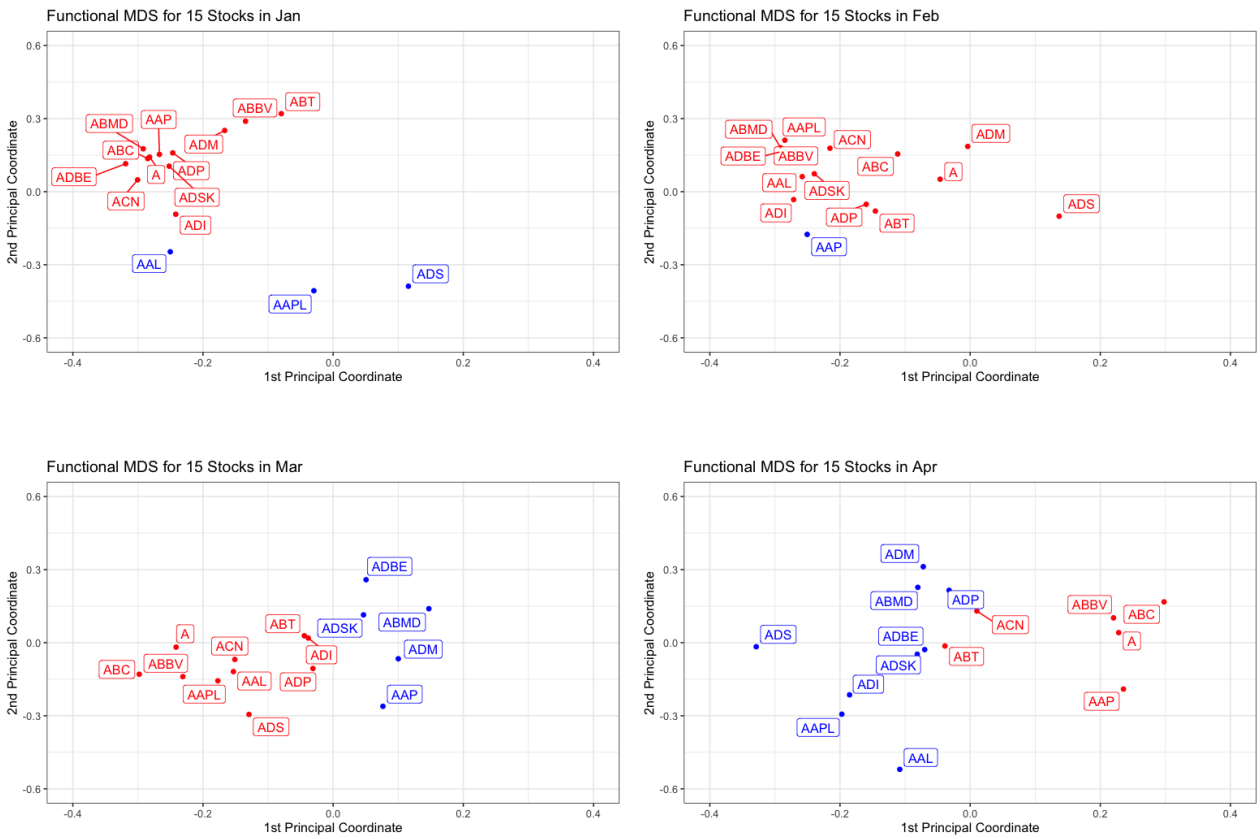

Figure 6.2 FMDS for 15 stocks in January, February, March, and April

Since the map of the 500 stocks analyzed in 12 months leads to a messy plot for each month, it is hard to have a clear visualization to observe how each stock change its position in each month. Specifically, we only select the first 15 stocks for 4 months as an example to see how the stocks move. Figure 6.2 shows the 2D FMDS maps for 15 stocks: A, AAL, AAP, AAPL, ABBV, ABC, ABMD, ABT, ACN, ADBE, ADI, ADM, ADP, ADS, and ADSK in January (top left), February (top right), March (bottom left), and April (bottom right), respectively. Similar to some methods in cluster analysis, the stocks can be grouped in clusters based on some conditions. For example, we randomly select a stock as the center point to separate the stocks in two clusters. Let's select the stock A (Agilent Technologies) as the center point. Since the stock A

is the 1^{st} stock in the S&P 500 Index, $\hat{d}_{1j}(t)$ means the estimated dissimilarities between the stock A and the j^{th} stock. The stocks in the red cluster have the dissimilarity (calculated as the Euclidean distance) $\hat{d}_{1j}(t) < 0.3$ from the stock A, whereas the stocks in the blue cluster have the dissimilarity $\hat{d}_{1j}(t) \geq 0.3$ from the stock A. In other words, the stock A is closer to the stocks in the red cluster than the stocks in the blue cluster. Meanwhile, we can see how the stocks move in each month, for instance, the stocks A and AAP (Advance Auto Parts) are close to each other in January, but they move apart from each other in February and March. In contrast, the stocks A and AAPL (Apple) belong to different clusters in January, but they move closer in February and March. On the other hand, the stock A, ABBV (AbbVie) and ABC (AmerisourceBergen) are very close to each other. The small dissimilarities among these three stocks demonstrate that they have an impact on each other during these four months. It makes sense because all of them belong to the sector in Health Care. Consequently, the maps made by the FMDS method show that the clusters have different stocks in each month.

Regarding the smoothness, Figure 6.3 shows two examples for the comparison between the observed dissimilarities $d_{ij}(t)$ (blue and dashed lines) and the estimated dissimilarities $\hat{d}_{ij}(t)$ (red and smooth curves). The left panel reflects the example of the pair stocks of A and AAP, and the right panel shows the example of the pair stocks of AAP and BA (Boeing). According to the red curves in the panels, we can see how the dissimilarities of A and AAP and the dissimilarities of AAP and BA change in month smoothly. Moreover, the red and smooth curves are very close to the corresponding blue and dashed lines. It means that the 2D smoothly time-varying

representation $\hat{\mathbf{x}}_i(t) = \hat{\mathbf{C}}_i\boldsymbol{\beta}(t)$ are modeled well in our FMDS method without changing the trend of the observed dissimilarities $d_{ij}(t)$'s.

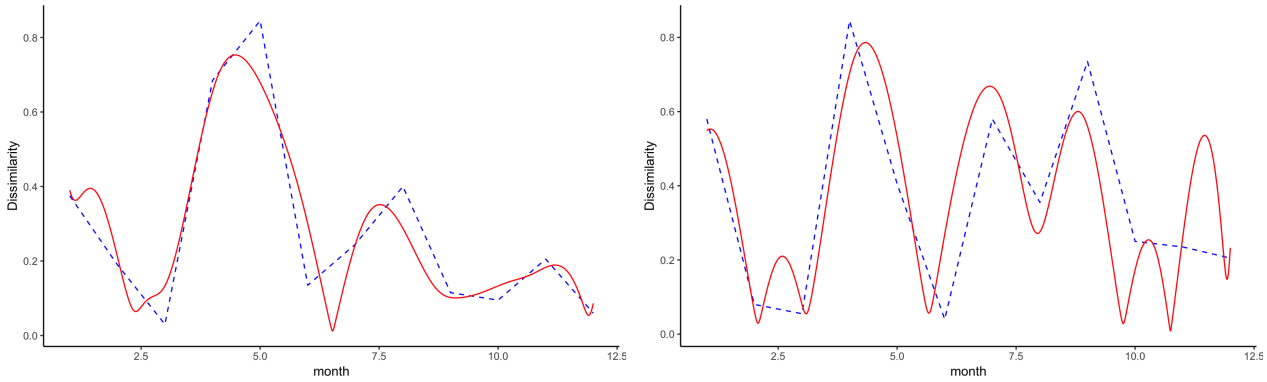

Figure 6.3 Dissimilarities change in month

Alternatively, we create the Shepard Diagrams in Figure 6.4 which show the observed dissimilarities (x-axis) versus the estimated dissimilarities (y-axis) for each month so that we can see how the estimated dissimilarities $\|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\|$ are close to the observed dissimilarities $d_{ij}(t)$. The graphs suggest that some of the observed dissimilarities are estimated well, while the other observed dissimilarities are underestimated or overestimated. In fact, we can expect this happens due to three main reasons for it. First of all, we are using the Adam SGD method which updates the pair of estimators $\mathbf{C}_i$ and $\mathbf{C}_j$ randomly in each iteration, so probably there are not enough updates for some pairs. Secondly, we are seeking for the local minimum of the target function (6.1) but not the global minimum. Also, the stopping threshold in the iterations of the Adam SGD method is one of the reason to cause different local minimums. The smaller stopping threshold we set, the smaller difference between the estimated and observed dissimilarities but also the heavier computation burden. Lastly, the estimated dissimilarities $\|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\|$ are

taken to be the Euclidean distance, while the observed dissimilarities $d_{ij}(t)$ are evaluated by the correlations among the closing prices of the stocks. It can be one of the reasons to make the difference occur.

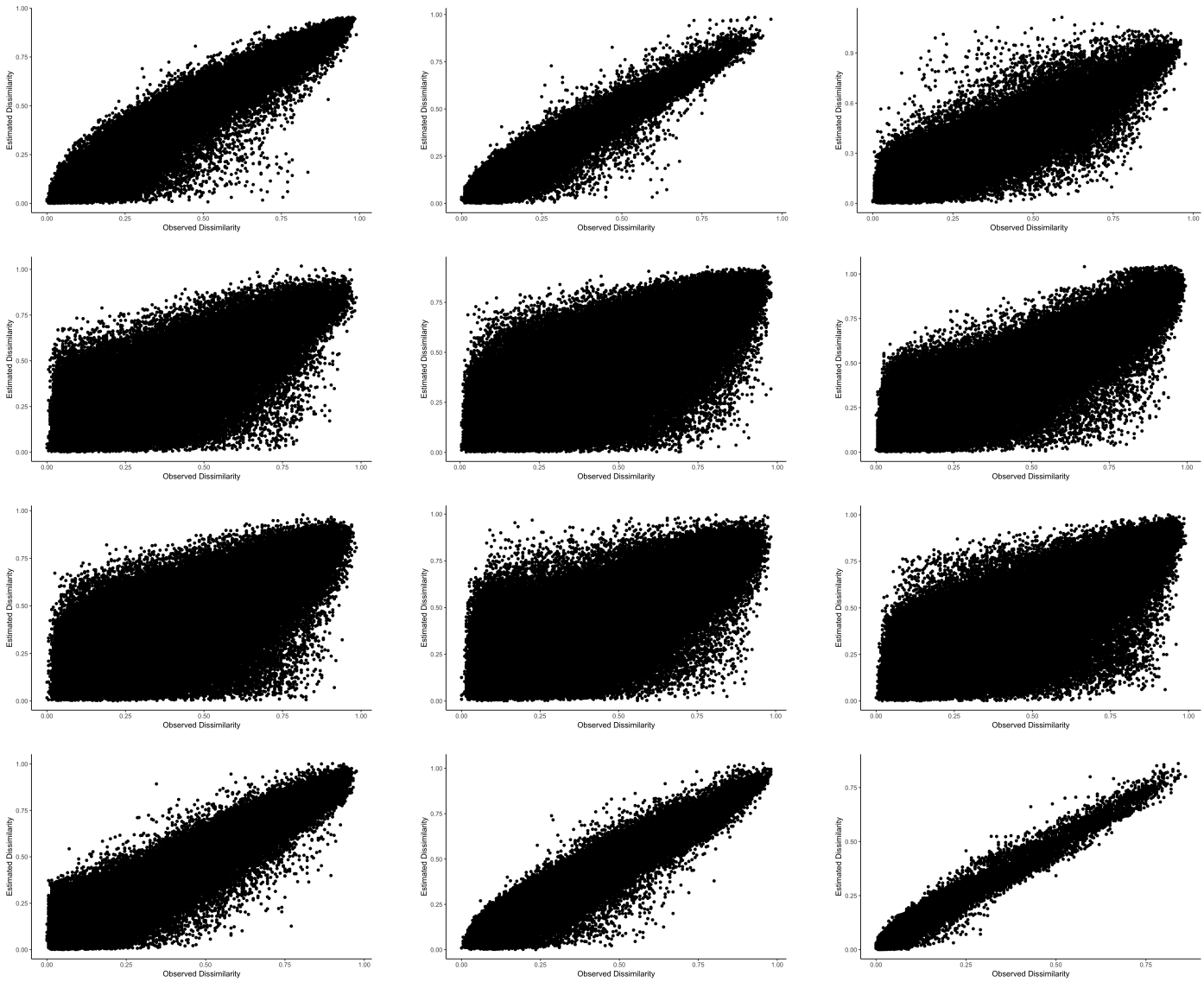

Figure 6.4 The Shepard Diagrams for each month

Figure 6.5 shows the residuals from the fit plotted against the sequence number, where residual = $\|\hat{\mathbf{x}}_i(t_k) - \hat{\mathbf{x}}_j(t_k)\| - d_{ij}(t_k)$. Although the residuals are relatively large for some stocks, the graph implies that the residuals are not large for most of the stocks. Hence, the difference between the observed and estimated dissimilarities tend to zero in general. It satisfies the situation in Figure 6.4. As a conclusion, there are 70.5% of the stocks having the absolute residual smaller than or equal to 0.1.

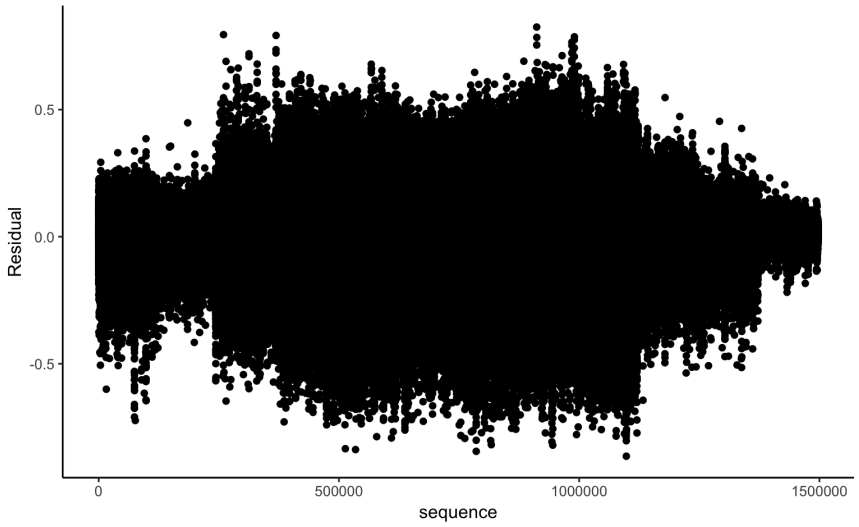

Figure 6.5 The residuals from the fit plotted against the sequence number

7 Conclusions

We have presented a spline model to obtain optimal MDS solutions in a low dimension for time-varying dissimilarities. The proposed functional multidimensional scaling (FMDS) method is preferred over the conventional MDS methods because it is a useful tool for smoothing MDS solutions with $\hat{\mathbf{x}}_i(t) = \hat{\mathbf{C}}_i\boldsymbol{\beta}(t)$ and visualizing the movement of objects over time. The results of the simulation study in Chapter 5 prove that our FMDS method performs well under the Gaussian assumptions. Also, our modified Adam SGD method applied to the optimization of the estimators $\mathbf{C}_i$'s in this study reduces the computation burden efficiently.

The insights gained from this dissertation make several main contributions to the current literatures. Firstly, the FMDS method takes advantage of the smoothness brought by the techniques in functional data analysis. Secondly, this method successfully reconstructs a low-dimensional space for objects so that the functional dissimilarities of objects are as close as their original dissimilarities. Thus, our method can subsequently be used to investigate the existence of time-varying clusters in data. For example, we can apply the FMDS method to stocks that tend to form groups by industrial sector or some other important characteristic, and see how the clusters change in time so that it can improve predictions of the relationships among the stock market. Last but not the least, the empirical findings in the study provide the evidence for the feasibility of displaying multidimensional scaling solutions temporally. Moreover, no previous study has investigated this area by employing the techniques in functional data analysis.

Bibliography

1. Young, G. and Householder, A.S. (1938) "*Discussion of a set of points in terms of their mutual distances.*". Psychometrika, 3, 19-22
2. Torgerson, W.S. (1952) "*Multidimensional scaling: 1. Theory and method.*". Psychometrika, 17, 401-419
3. Torgerson, W.S. (1958) "*Theory and Method of Scaling.*". New York: Wiley
4. Gower, J.C. (1966) "*Some distance properties of latent root arid vector methods in multivariate analysis.*". Biometrika, 53, 325-338
5. Shepard, R.N. (1962a) "*The analysis of proximities: multidimensional scaling with an unknown distance function I.*". Psychometrika, 27, 125-140
6. Shepard, R.N. (1962a) "*The analysis of proximities: multidimensional scaling with an unknown distance function II.*". Psychometrika, 27, 219-246
7. Kruskal, J.B. (1964a) "*Multidimensional scaling by optimizing goodness-of-fit to a nonmetric hypodissertation.*". Psychometrika, 29, 1-27
8. Kruskal, J.B. (1964b) "*Nonmetric multidimensional scaling: a numerical method.*". Psychometrika, 29, 115-129
9. Guttman, L. (1968) "*A general nonmetric technique for finding the smallest coordinate space for a configuration of points.*". Psychometrika, 33, 469-506
10. Sammon, J.W. (1969) "*A nonlinear mapping for data structure analysis.*". IEEE Transactions on Computers. 18 (5): 401, 402, 403-409

11. Takane, Y., Young, F.W. and de Leeuw, J. (1977) "*Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features.*". Psychometrika, 42, 7-67
12. Critchley, F. (1978) "*Multidimensional scaling: a short critique and a new method.*". In *COMPSTAT* 1978 (L.C.A. Corsten and J. Hermans, eds) pp. 297-303. Vienna: Physica-Verlag.
13. Schneider, R.B. (1992) "*A uniform approach to multidimensional scaling.*", Journal of Classification, 9, 257-273
14. Chen, C.H. and Chen, J.A. (2000) "*Interactive diagnostic plots for multidimensional scaling with applications in psychosis disorder data analysis.*". Unpublished manuscript
15. Ramsay, J.O. (1977) "*Maximum likelihood estimation in multidimensional scaling.*". Psychometrika, 42, 241-266
16. Ramsay, J.O. (1982) "*Some statistical approaches to multidimensional scaling data.*". J. R. Stat. Soc., A., 145, 285-312
17. Hoffman, T. and Buhmann, J. (1994) "*Multidimensional Scaling and Data Clustering.*". The MIT Press, Vol. 7, pages 459-466
18. Williams, C. K.I. (2000) "*On a Connection between Kernal PCA and Metric Multidimensional Scaling.*". NIPS 2000: Proceedings of the 13th International Conference on Neural Information Processing Systems, pages 654-660
19. Oh, M.S. and Raftery, A.E. (2001) "*Bayesian Multidimensional Scaling and Choice of Dimensional.*". Journal of the American Statistical Association, Vol. 96, No. 455, pp. 1031-1044

20. Grenander, U. (1950) "*Stochastic Processes and Statistical Inference.*". Arkiv Matematik 1: 195-277
21. Rao, C.R. (1958) "*Some Statistical Methods for Comparison of Growth Curves.*". Biometrics 14:1-17
22. Ramsay J.O. and Silverman, B.W. (2002) "*Applied Functional Data Analysis: Methods and Case Studies.*". Springer
23. Ramsay, J.O. and Silverman, B.W. (2005) "*Functional Data Analysis.*". Springer
24. Izenman, A.J. (2008) "*Modern Multivariate Statistical Techniques: Regression, Classification and Manifold Learning.*". Springer
25. Ambrosi, K. and Hansohm, J. (1987) "Ein Dynamischer Ansatz zur Repräsentation von Objekten.". In Operation Research Proceedings 1986, Berlin: Springer-Verlag
26. Jackle. D. and Fischer F. and Schreck T. And Keim D.A (2016) "Temporal MDS Plots for Aanlysis of Multivariate Data.". IEEE Transactions on Visualization and Computer Graphics 22 (1), 141-150
27. Machado, J.T. and Duarte F.B. and Duarte G.M. (2010) "Multidimensional Scaling Analysis of Stock Market Values.". Mathematical Problems in Engineering, Vol. 2012
28. Atkinson, K.E. (1988) "An Introduction to Numerical Analysis.". John Wiley & Sons
29. de Boor, C. (2001) "A Practical Guide to Splines, Revised Edition.". Springer, New York
30. Cox, M. A.A. and Cox, T.F. (2001) "Multidimensional Scaling.". Boca Raton: Chapman & Hall/CRC
31. Sibson, R. (1978) "Studies in the Robustness of Multidimensional Scaling: Procrustes Statistics.". J. R. Stat. Soc. B, 40, No.2, pp. 234-238

32. Wen, Z. and Yin, W. (2010) “A Feasible Method for Optimization with Orthogonality Constraints.”. Mathmatical Programming 142(1-2)
33. Kingma, D.P. and Ba, J.L. (2015) “ADAM: A Method for Stochastic Optimization”. ICLR 2015

Appendices

Appendix A: R Codes for Dissimilarity in Simulation Study with 5 Knots

1. This program 300replications.Rmd generates the (2×9) coefficient matrices $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$ from a Gaussian. $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$ are used to construct the observed data. The following Cy is a list containing 300 replications where each replication contains $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$.

```
# Input:
#     p: dimension of  FMDS solution (integer)
#     n: number of observations (integer)
#     L: number of knots for cubic B-spline basis (integer)
#     q: dimension of  cubic B-spline basis (integer)
#     mu: mean of each element in C_1^(y), ..., C_50^(y)
#    sigma: covariance matrix of the vectorized C_1^(y), ..., C_50^(y)
#    rep_num: number of replications
p <- 2
n <- 50
L <- 5
q <- L + 4
mu <- rep(0, p*q)
sigma <- diag(p*q)
rep_num <- 300
```

 # Generate the independent and identically vectorized $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$ *from the multivariate*

normal distribution with mean = mu and covariance matrix = sigma

```
set.seed(321)
C_y <- replicate(rep_num, mvrnorm(n = n, mu, sigma), simplify = FALSE)
C_y_mat <- array(0, dim = c(n, p*q, rep_num))
C_yT <- array(0, dim = c(p*q, n, rep_num))
py <- matrix(0, nrow = p*q*n, ncol = rep_num)
Cy <- vector(mode = 'list', length = rep_num)
for (i in 1:rep_num) {
  C_y_mat[,,i] <- matrix(unlist(C_y[i]), ncol = p*q, nrow = n)
  C_yT[,,i] <- t(C_y_mat[,,i])
  py[,i] <- as.vector(C_yT[,,i])
  Cy[[i]] <- array(py[,i], dim = c(p, q, n))
}
```

2. This program compact5knots.Rmd is a function including two parts. The first part uses the modified Adam SGD method to calculate the optimization of the objective function $F(\mathbf{C}_1, \ldots, \mathbf{C}_n)$ and obtains the following *adam_out* as the optimal $\hat{\mathbf{C}}_1, \ldots, \hat{\mathbf{C}}_n$. The second part calculates the MSE of $d_{ij}(t_k)$ and $\hat{d}_{ij}(t_k)$, that is, the equation (5.1) in the Section 5.1.

```
fmds_mse <- function(Cy_var, m, p, L, n, ep) {
  # B-Spline Basis
  q <- L + 4
```

```
 t <- seq(0, 1, length.out = m)
 b <- bs(t, knots = seq(min(t), max(t), length.out = L+2)[2:(L+1)], degree=3, intercept=T,
Boundary.knots = range(t))
 betabasis <- as.matrix(t(b))
 betavec <- as.vector(betabasis)
 B <- matrix(betavec, nrow = q)
 # Distance of y_i(t) and y_j(t)
 D_2y <- matrix(0, nrow = n*(n-1)/2, ncol = m)
 for (i in 1:(n-1)) {
  for (j in (i+1):n) {
    Cy_i<- Cy_var[,,i]
    Cy_j <- Cy_var[,j]
    Cy_ij <- Cy_i - Cy_j
    for (k in 1:m) {
      D_2y[(j-1)*(j-2)/2 + i, k] <- crossprod(Cy_ij %*% B[,k])
    }
   }
 }
 # Initial estimators
 C0 <- array(0,dim=c(p,q,n))
 Xraw <- array(0,dim=c(p,m,n))
 for (k in 1:m){
```

```
  d <- D_2y[,k]
  attr(d,"Size") <- n
  attr(d,"Diag") <- F
  attr(d,"Upper") <- F
  class(d) <- "dist"
  x <- cmdscale(d,k=p)
  for (i in 1:n){
    Xraw[,k,i] <- x[i,]
  }
}
for (i in 1:n){
  C0[,,i] <- t(solve(t(b)%*%b,t(b)%*%t(Xraw[,,i])))
}
# Stochastic Gradient Function
stochgr <- function(Chj, h ,j) {
        gr <- array(0, dim = c(p, q, 2))
        sum_months <- 0
        C_hj <- Chj[,,1]- Chj[,,2]
        for (k in 1:m) {
          sum_months <- sum_months - 4 * as.numeric(D_2y[(j-1)*(j-2)/2 + h, k] -
crossprod(C_hj %*% B[,k])) * C_hj %*% B[,k] %*% t(B[,k])
        }
```

```
        gr[,,1] <- sum_months          # derivative of C_h
        gr[,,2] <- -sum_months         # derivative of C_j
        gr
}
# Adam SGD method
adam_sgd <- function (C_var) {
  err <- 1
  iter <- 0
  C <- C_var
  beta1 <- 0.9
  beta2 <- 0.999
  v <- array(0, dim = c(p, q, n))
  M <- array(0, dim = c(p, q, n))
  e <- array(1e-8, dim = c(p, q, 2))
  step_len <- 0.001
   while (err >= ep) {
    newh <- sample(1:(n-1))            # Shuffle the first n-1 stocks
    for (i in 1:(n-1)) {
      for (r in seq(newh[i] + 1, n)) {
        if (err < ep) {break}
        vhj <- v[,,c(newh[i], r)]
        mhj <- M[,,c(newh[i], r)]
```

```
        C0hj <- C_var[,,c(newh[i], r)]
        mhj <- beta1 * mhj + (1 - beta1) * stochgr(C0hj, newh[i], r)
        mhj_hat <- mhj/(1 - beta1^(iter+1))
        vhj <- beta2 * vhj + (1 - beta2) * (stochgr(C0hj, newh[i], r))^2
        vhj_hat <- vhj/(1 - beta2^(iter+1))
        C[,,c(newh[i], r)] <- C0hj - step_len * mhj_hat / (sqrt(vhj_hat) + e)
        C_diff <- as.vector(reshape(C - C_var, p*q*n, 1))
        err <- sqrt(sum((C_diff)^2))
        v[,,c(newh[i], r)] <- vhj
        M[,,c(newh[i], r)] <- mhj
        C_var <- C
        iter <- iter + 1
      }
    }
  }
  C
}
adam_out <- adam_sgd(C0)
# Distance of x_i(t) and x_j(t)
D_2x <- matrix(0, nrow = n*(n-1)/2, ncol = m)
for (i in 1:(n-1)) {
  for (j in (i+1):n) {
```

```
    Cx_i<- adam_out[,,i]
    Cx_j <- adam_out[,,j]
    Cx_ij <- Cx_i - Cx_j
    for (k in 1:m) {
      D_2x[(j-1)*(j-2)/2 + i, k] <- crossprod(Cx_ij %*% B[,k])
    }
   }
  }
  # RMSE
  Dx_vec <- sqrt(as.vector(D_2x))
  Dy_vec <- sqrt(as.vector(D_2y))
  mse <- (rmse(Dy_vec, Dx_vec))^2
  return(list(optim_C = adam_out, mse = mse))
}
```

3. Repeat 300 times for the above function fmds_mse with $L = 5$ knots and $m = 15,50,100,200$, respectively to get the RMSE as the equation (5.2) in the Section 5.1.

```
total_rep <- 300
rep_seq <- 1:total_rep
# For m = 15
rep_fn1 <- function(rep_num) {
 fmds_mse(Cy = Cy[[rep_num]], m = 15, p = 2, L = 5, n = 50, ep = 0.00075)
```

```
}
set.seed(1505)
start_time1 <- Sys.time()
adam_res1 <- pblapply(rep_seq, rep_fn1)
end_time1 <- Sys.time()
end_time1 - start_time1
total_mse_1 <- 0
for (i in rep_seq) {
  total_mse_1 <- total_mse_1 + adam_res1[[i]]$mse
}
rmse_1 <- sqrt(total_mse_1/total_rep)
rmse_1
# For m = 50
rep_fn2 <- function(rep_num) {
  fmds_mse(Cy = Cy[[rep_num]], m = 50, p = 2, L = 5, n = 50, ep = 0.00075)
}
set.seed(5005)
start_time2 <- Sys.time()
adam_res2 <- pblapply(rep_seq, rep_fn2)
end_time2 <- Sys.time()
end_time2 - start_time2
total_mse_2 <- 0
```

```
for (i in rep_seq) {
  total_mse_2 <- total_mse_2 + adam_res2[[i]]$mse
}
rmse_2 <- sqrt(total_mse_2/total_rep)
rmse_2
# For m = 100
rep_fn3 <- function(rep_num) {
  fmds_mse(Cy = Cy[[rep_num]], m = 100, p = 2, L = 5, n = 50, ep = 0.00075)
}
set.seed(10005)
start_time3 <- Sys.time()
adam_res3 <- pblapply(rep_seq, rep_fn3)
end_time3 <- Sys.time()
end_time3 - start_time3
total_mse_3 <- 0
for (i in rep_seq) {
  total_mse_3 <- total_mse_3 + adam_res3[[i]]$mse
}
rmse_3 <- sqrt(total_mse_3/total_rep)
rmse_3
# For m = 200
rep_fn4 <- function(rep_num) {
```

```
  fmds_mse(Cy = Cy[[rep_num]], m = 200, p = 2, L = 5, n = 50, ep = 0.00075)
}
set.seed(20005)
start_time4 <- Sys.time()
adam_res4 <- pblapply(rep_seq, rep_fn4)
end_time4 <- Sys.time()
end_time4 - start_time4
total_mse_4 <- 0
for (i in rep_seq) {
  total_mse_4 <- total_mse_4 + adam_res4[[i]]$mse
}
rmse_4 <- sqrt(total_mse_4/total_rep)
rmse_4
```

Appendix B: R Codes for Dissimilarity in Simulation Study with 10 Knots

1. This program 300replications10knots.Rmd generates the (2×14) coefficient matrices $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$ from a Gaussian. $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$ are used to construct the observed data. The following Cy is a list containing 300 replication where each replication contains $\mathbf{C}_1^{(y)}, \ldots, \mathbf{C}_{50}^{(y)}$.

```
p <- 2

n <- 50

L <- 10

q <- L + 4

mu <- rep(0, p*q)

sigma <- diag(p*q)

rep_num <- 300

# Generate the independent and identically vectorized C_1^(y), ..., C_50^(y) from the multivariate

normal distribution with mean = mu and covariance matrix = sigma

set.seed(10)

C_y <- replicate(rep_num, mvrnorm(n = n, mu, sigma), simplify = FALSE)

C_y_mat <- array(0, dim = c(n, p*q, rep_num))

C_yT <- array(0, dim = c(p*q, n, rep_num))

py <- matrix(0, nrow = p*q*n, ncol = rep_num)
```

```
Cy <- vector(mode = 'list', length = rep_num)
for (i in 1:rep_num) {
  C_y_mat[,,i] <- matrix(unlist(C_y[i]), ncol = p*q, nrow = n)
  C_yT[,,i] <- t(C_y_mat[,,i])
  py[,i] <- as.vector(C_yT[,,i])
  Cy[[i]] <- array(py[,i], dim = c(p, q, n))
}
```

2. This program compact10knots.Rmd is a function including two parts. The first part uses the modified Adam SGD method to calculate the optimization of the objective function $F(\mathbf{C}_1, \ldots, \mathbf{C}_n)$ and obtains the following *adam_out* as the optimal $\hat{\mathbf{C}}_1, \ldots, \hat{\mathbf{C}}_n$. The second part calculates the MSE of $d_{ij}(t_k)$ and $\hat{d}_{ij}(t_k)$, that is, the equation (5.1) in the Section 5.1.

```
fmds_mse <- function(Cy_var, m, p, L, n, ep) {
  # B-Spline Basis
  q <- L + 4
  t <- seq(0, 1, length.out = m)
  b <- bs(t, knots = seq(min(t), max(t), length.out = L+2)[2:(L+1)], degree=3, intercept=T,
Boundary.knots = range(t))
  betabasis <- as.matrix(t(b))
  betavec <- as.vector(betabasis)
  B <- matrix(betavec, nrow = q)        # Cubic B-spline and 5 knots, so nrow = 5+4 = 9, q = 9
  # Distance of y_i(t) and y_j(t)
```

```
D_2y <- matrix(0, nrow = n*(n-1)/2, ncol = m)
for (i in 1:(n-1)) {
  for (j in (i+1):n) {
    Cy_i<- Cy_var[,,i]
    Cy_j <- Cy_var[,,j]
    Cy_ij <- Cy_i - Cy_j
    for (k in 1:m) {
      D_2y[(j-1)*(j-2)/2 + i, k] <- crossprod(Cy_ij %*% B[,k])
    }
   }
}
# Initial estimators
C0 <- array(0,dim=c(p,q,n))
Xraw <- array(0,dim=c(p,m,n))
for (k in 1:m){
  d <- D_2y[,k]
  attr(d,"Size") <- n
  attr(d,"Diag") <- F
  attr(d,"Upper") <- F
  class(d) <- "dist"
  x <- cmdscale(d,k=p)
  for (i in 1:n){
```

```
    Xraw[,k,i] <- x[i,]
  }
}
for (i in 1:n){
  C0[,,i] <- t(solve(t(b)%*%b,t(b)%*%t(Xraw[,,i])))
}
# Stochastic Gradient Function
stochgr <- function(Chj, h ,j) {
        gr <- array(0, dim = c(p, q, 2))
        sum_months <- 0
        C_hj <- Chj[,,1]- Chj[,,2]
        for (k in 1:m) {
          sum_months <- sum_months - 4 * as.numeric(D_2y[(j-1)*(j-2)/2 + h, k] -
crossprod(C_hj %*% B[,k])) * C_hj %*% B[,k] %*% t(B[,k])
        }
        gr[,,1] <- sum_months          # derivative of C_h
        gr[,,2] <- -sum_months         # derivative of C_j
        gr
}
# Adam SGD method
adam_sgd <- function (C_var) {
  err <- 1
```

```
iter <- 0
C <- C_var
beta1 <- 0.9
beta2 <- 0.999
v <- array(0, dim = c(p, q, n))
M <- array(0, dim = c(p, q, n))
e <- array(1e-8, dim = c(p, q, 2))
step_len <- 0.001
 while (err >= ep) {
  newh <- sample(1:(n-1))                # Shuffle the first n-1 stocks
  for (i in 1:(n-1)) {
    for (r in seq(newh[i] + 1, n)) {
      if (err < ep) {break}
      vhj <- v[,,c(newh[i], r)]
      mhj <- M[,,c(newh[i], r)]
      C0hj <- C_var[,,c(newh[i], r)]
      mhj <- beta1 * mhj + (1 - beta1) * stochgr(C0hj, newh[i], r)
      mhj_hat <- mhj/(1 - beta1^(iter+1))
      vhj <- beta2 * vhj + (1 - beta2) * (stochgr(C0hj, newh[i], r))^2
      vhj_hat <- vhj/(1 - beta2^(iter+1))
      C[,,c(newh[i], r)] <- C0hj - step_len * mhj_hat / (sqrt(vhj_hat) + e)
      C_diff <- as.vector(reshape(C - C_var, p*q*n, 1))
```

```
        err <- sqrt(sum((C_diff)^2))
        v[,,c(newh[i], r)] <- vhj
        M[,,c(newh[i], r)] <- mhj
        C_var <- C
        iter <- iter + 1
      }
    }
  }
  C
}
adam_out <- adam_sgd(C0)
# Distance of x_i(t) and x_j(t)
D_2x <- matrix(0, nrow = n*(n-1)/2, ncol = m)
for (i in 1:(n-1)) {
  for (j in (i+1):n) {
    Cx_i<- adam_out[,,i]
    Cx_j <- adam_out[,,j]
    Cx_ij <- Cx_i - Cx_j
    for (k in 1:m) {
      D_2x[(j-1)*(j-2)/2 + i, k] <- crossprod(Cx_ij %*% B[,k])
    }
  }
```

```
  }
  # RMSE
  Dx_vec <- sqrt(as.vector(D_2x))
  Dy_vec <- sqrt(as.vector(D_2y))
  mse <- (rmse(Dy_vec, Dx_vec))^2
  return(list(optim_C = adam_out, mse = mse))
}
```

3. Repeat 300 times for the above function fmds_mse with $L = 10$ knots and $m = 15,50,100,200$, respectively to get the RMSE as the equation (5.2) in the Section 5.1.

```
total_rep <- 300
rep_seq <- 1:total_rep
# For m = 15
rep_fn1 <- function(rep_num) {
  fmds_mse(Cy = Cy[[rep_num]], m = 15, p = 2, L = 10, n = 50, ep = 0.00075)
}
set.seed(1510)
start_time1 <- Sys.time()
adam_res1 <- pblapply(rep_seq, rep_fn1)
end_time1 <- Sys.time()
end_time1 - start_time1
total_mse_1 <- 0
```

```
for (i in rep_seq) {
  total_mse_1 <- total_mse_1 + adam_res1[[i]]$mse
}
rmse_1 <- sqrt(total_mse_1/total_rep)
rmse_1
# For m = 50
rep_fn2 <- function(rep_num) {
  fmds_mse(Cy = Cy[[rep_num]], m = 50, p = 2, L = 10, n = 50, ep = 0.00075)
}
set.seed(5010)
start_time2 <- Sys.time()
adam_res2 <- pblapply(rep_seq, rep_fn2)
end_time2 <- Sys.time()
end_time2 - start_time2
total_mse_2 <- 0
for (i in rep_seq) {
  total_mse_2 <- total_mse_2 + adam_res2[[i]]$mse
}
rmse_2 <- sqrt(total_mse_2/total_rep)
rmse_2
# For m = 100
rep_fn3 <- function(rep_num) {
```

```
 fmds_mse(Cy = Cy[[rep_num]], m = 100, p = 2, L = 10, n = 50, ep = 0.00075)
}
set.seed(10010)
start_time3 <- Sys.time()
adam_res3 <- pblapply(rep_seq, rep_fn3)
end_time3 <- Sys.time()
end_time3 - start_time3
total_mse_3 <- 0
for (i in rep_seq) {
  total_mse_3 <- total_mse_3 + adam_res3[[i]]$mse
}
rmse_3 <- sqrt(total_mse_3/total_rep)
rmse_3
# For m = 200
rep_fn4 <- function(rep_num) {
 fmds_mse(Cy = Cy[[rep_num]], m = 200, p = 2, L = 10, n = 50, ep = 0.00075)
}
set.seed(20010)
start_time4 <- Sys.time()
adam_res4 <- pblapply(rep_seq, rep_fn4)
end_time4 <- Sys.time()
end_time4 - start_time4
```

```
total_mse_4 <- 0

for (i in rep_seq) {

  total_mse_4 <- total_mse_4 + adam_res4[[i]]$mse

}

rmse_4 <- sqrt(total_mse_4/total_rep)

rmse_4
```

Appendix C: R Codes for Parameters C in Simulation Study with 5 Knots

```
# This is for L = 5, m = 15. For the simulations calculating with m = 50, 100, and 200, we just replace the following m with 50, 100, 200 and correspondingly replace adam_res1 with adam_res2, adam_res3, adam_res4.

    p <- 2          # 2-dimensional MDS

    n <- 50         # number of stocks

    m <- 15         # number of months

    L <- 5          # L equally spaced knots (internal break points)

    q <- L + 4      # dimensions of cubic B-spline

    t <- seq(1, m, by = 0.5)

    b <- bs(t, knots = seq(min(t), max(t), length.out = L+2)[2:(L+1)], degree=3, intercept=T, Boundary.knots = range(t))

    betabasis <- as.matrix(t(b))

    betavec <- as.vector(betabasis)

    Basis <- matrix(betavec, nrow = q)       # Cubic B-spline and 5 knots, so nrow = 5+4 = 9, q = 9

    total_rep <- 300

    r <- 1:total_rep

    set.seed(101)

    B <- gramSchmidt(matrix(rnorm(p*p), p, p))$Q
```

```
I <- diag(2)
fn <- function(B, r) {
 f1 <- 0
 fmid <- 0
 fm <- 0
 Cxr <- adam_res1[[r]]$optim_C
 for (i in 1:n) {
  f1 <- f1 + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,1])
  for (k in 2:(2*m-2)) {
   fmid <- fmid + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,k])
  }
  fm <- fm + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,2*m-1])
 }
 fn_val <- (f1 + 2 * fmid + fm)/4
 fn_val
}
gr <- function(B, r) {
 gn_1 <- matrix(0, 2, 2)
 gn_mid <- matrix(0, 2, 2)
 gn_m <- matrix(0, 2, 2)
 Cxr <- adam_res1[[r]]$optim_C
 for (i in 1:n) {
```

```
      gn_1 <- gn_1 + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,1] %*%
t(Cxr[,,i] %*% Basis[,1])
      for (k in 2:(2*m-2)) {
        gn_mid <- gn_mid + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,k]
%*% t(Cxr[,,i] %*% Basis[,k])
      }
      gn_m <- gn_m + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,2*m-1]
%*% t(Cxr[,,i] %*% Basis[,2*m-1])
    }
    gn_mat <- (gn_1 + 2 * gn_mid + gn_m)/4
    gn_mat
  }
  fit_singlefn <- function(r) {
    fit <- ortho_optim(B, fn, gr, r = r)
    P <- fit$B
    P
  }
  fit_300rep <- pblapply(r, fit_singlefn)
  mse_vec <- rep(0, total_rep)
  for (j in r) {
    for (i in 1:n) {
      gamma_array <- array(0, dim = c(p, q, n))
```

```
    true_array <- array(0, dim = c(p, q, n))
    gamma_array[,,i] <- fit_300rep[[j]] %*% adam_res1[[j]]$optim_C[,,i]
    true_array[,,i] <- Cy[[j]][,,i]
  }
  gamma_vec <- as.vector(gamma_array)
  true_vec <- as.vector(true_array)
  mse_vec[j] <- MSE(true_vec, gamma_vec)
}
gamma_rmse <- sqrt(mean(mse_vec))
gamma_rmse
```

Appendix D: R Codes for Parameter C in Simulation Study with 10 Knots

```
# This is for L = 10, m = 15. For the simulations calculating with m = 50, 100, and 200, we just replace the following m with 50, 100, 200 and correspondingly replace adam_res1 with adam_res2, adam_res3, adam_res4.
    p <- 2          # 2-dimensional MDS
    n <- 50         # number of stocks
    m <- 15         # number of months
    L <- 10         # L equally spaced knots (internal break points)
    q <- L + 4      # dimensions of cubic B-spline
    t <- seq(1, m, by = 0.5)
    b <- bs(t, knots = seq(min(t), max(t), length.out = L+2)[2:(L+1)], degree=3, intercept=T, Boundary.knots = range(t))
    betabasis <- as.matrix(t(b))
    betavec <- as.vector(betabasis)
    Basis <- matrix(betavec, nrow = q)      # Cubic B-spline and 5 knots, so nrow = 5+4 = 9, q = 9
    total_rep <- 300
    r <- 1:total_rep
    set.seed(101)
    B <- gramSchmidt(matrix(rnorm(p*p), p, p))$Q
```

```
I <- diag(2)
fn <- function(B, r) {
 f1 <- 0
 fmid <- 0
 fm <- 0
 Cxr <- adam_res1[[r]]$optim_C
 for (i in 1:n) {
  f1 <- f1 + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,1])
  for (k in 2:(2*m-2)) {
   fmid <- fmid + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,k])
  }
  fm <- fm + crossprod((B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,2*m-1])
 }
 fn_val <- (f1 + 2 * fmid + fm)/4
 fn_val
}
gr <- function(B, r) {
 gn_1 <- matrix(0, 2, 2)
 gn_mid <- matrix(0, 2, 2)
 gn_m <- matrix(0, 2, 2)
 Cxr <- adam_res1[[r]]$optim_C
 for (i in 1:n) {
```

```
      gn_1 <- gn_1 + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,1] %*%
t(Cxr[,,i] %*% Basis[,1])
      for (k in 2:(2*m-2)) {
        gn_mid <- gn_mid + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,k]
%*% t(Cxr[,,i] %*% Basis[,k])
      }
      gn_m <- gn_m + (I + t(I)) %*% (B %*% Cxr[,,i] - Cy[[r]][,,i]) %*% Basis[,2*m-1]
%*% t(Cxr[,,i] %*% Basis[,2*m-1])
    }
    gn_mat <- (gn_1 + 2 * gn_mid + gn_m)/4
    gn_mat
  }
  fit_singlefn <- function(r) {
    fit <- ortho_optim(B, fn, gr, r = r)
    P <- fit$B
    P
  }
  start_time <- Sys.time()
  fit_300rep <- pblapply(r, fit_singlefn)
  end_time <- Sys.time()
  end_time - start_time
  mse_vec <- rep(0, total_rep)
```

```
for (j in r) {
  for (i in 1:n) {
    gamma_array <- array(0, dim = c(p, q, n))
    true_array <- array(0, dim = c(p, q, n))
    gamma_array[,,i] <- fit_300rep[[j]] %*% adam_res1[[j]]$optim_C[,,i]
    true_array[,,i] <- Cy[[j]][,,i]
  }
  gamma_vec <- as.vector(gamma_array)
  true_vec <- as.vector(true_array)
  mse_vec[j] <- MSE(true_vec, gamma_vec)
}
gamma_rmse <- sqrt(mean(mse_vec))
gamma_rmse
```

Appendix E: R Codes for the Case Study of the S&P 500 Stocks

1. The matrix **D** in the following *UpperDissimilarity.RData* is the super dissimilarity matrix **D** in Chapter 6. The following result *adam_res* is the optimal $\hat{\mathbf{C}}_1, \ldots, \hat{\mathbf{C}}_{500}$.

```
load("UpperDissimilarity.RData")
Dsqr <- D^2
fmds <- function(m, p, L, n, ep) {
  # B-Spline Basis
  q <- L + 4
  t <- seq(0, 1, length.out = m)
   b <- bs(t, knots = seq(min(t), max(t), length.out = L+2)[2:(L+1)], degree=3, intercept=T,
Boundary.knots = range(t))
  betabasis <- as.matrix(t(b))
  betavec <- as.vector(betabasis)
  B <- matrix(betavec, nrow = q)
  # Initial estimators
  C0 <- array(0,dim=c(p,q,n))
  Xraw <- array(0,dim=c(p,m,n))
  for (k in 1:m){
   d <- D[,k]
   attr(d,"Size") <- n
```

```
  attr(d,"Diag") <- F
  attr(d,"Upper") <- F
  class(d) <- "dist"
  x <- cmdscale(d,k=p)
  for (i in 1:n){
    Xraw[,k,i] <- x[i,]
  }
}
for (i in 1:n){
  C0[,,i] <- t(solve(t(b)%*%b,t(b)%*%t(Xraw[,,i])))
}
 # Stochastic Gradient Function
 stochgr <- function(Chj, h ,j) {
            gr <- array(0, dim = c(p, q, 2))
            sum_months <- 0
            C_hj <- Chj[,,1]- Chj[,,2]
            for (k in 1:m) {
                        sum_months <- sum_months - 4 * as.numeric(Dsqr[(j-1)*(j-2)/2 + h, k] -
crossprod(C_hj %*% B[,k])) * C_hj %*% B[,k] %*% t(B[,k])
            }
            gr[,,1] <- sum_months              # derivative of C_h
            gr[,,2] <- -sum_months             # derivative of C_j
```

```
        gr
}
# Adam SGD method
adam_sgd <- function (C_var) {
  err <- 1
  iter <- 0
  C <- C_var
  beta1 <- 0.9
  beta2 <- 0.999
  v <- array(0, dim = c(p, q, n))
  M <- array(0, dim = c(p, q, n))
  e <- array(1e-8, dim = c(p, q, 2))
  step_len <- 0.001
   while (err >= ep) {
    newh <- sample(1:(n-1))              # Shuffle the first n-1 stocks
    for (i in 1:(n-1)) {
      for (r in seq(newh[i] + 1, n)) {
        if (err < ep) {break}
        vhj <- v[,,c(newh[i], r)]
        mhj <- M[,,c(newh[i], r)]
        C0hj <- C_var[,,c(newh[i], r)]
        mhj <- beta1 * mhj + (1 - beta1) * stochgr(C0hj, newh[i], r)
```

```
        mhj_hat <- mhj/(1 - beta1^(iter+1))
        vhj <- beta2 * vhj + (1 - beta2) * (stochgr(C0hj, newh[i], r))^2
        vhj_hat <- vhj/(1 - beta2^(iter+1))
        C[,,c(newh[i], r)] <- C0hj - step_len * mhj_hat / (sqrt(vhj_hat) + e)
        C_diff <- as.vector(reshape(C - C_var, p*q*n, 1))
        err <- sqrt(sum((C_diff)^2))
        v[,,c(newh[i], r)] <- vhj
        M[,,c(newh[i], r)] <- mhj
        C_var <- C
        iter <- iter + 1
      }
     }
    }
   C
 }
 adam_sgd(C0)
}
set.seed(10000)
adam_res <- fmds(m = 12, p = 2, L = 6, n = 500, ep = 0.00075)
adam_res
```