AN ADDITIVE BIVARIATE HIERARCHICAL MODEL FOR FUNCTIONAL DATA AND RELATED COMPUTATIONS

A Dissertation

by

ANDREW MIDDLETON REDD

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2010

Major Subject: Statistics

An Additive Bivariate Hierarchical Model

for Functional Data and Related Computations

AN ADDITIVE BIVARIATE HIERARCHICAL MODEL FOR FUNCTIONAL DATA AND RELATED COMPUTATIONS

A Dissertation

by

ANDREW MIDDLETON REDD

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Raymond J. Carroll
Committee Members,	Michael Longnecker
	Rosemary Walzem
	Lan Zhou
Head of Department,	Simon J. Sheather

August 2010

Major Subject: Statistics

ABSTRACT

An Additive Bivariate Hierarchical Model
for Functional Data and Related Computations. (August 2010)
Andrew Middleton Redd, B.S., Weber State University;
M.S., Texas A&M University
Chair of Advisory Committee: Dr. Raymond J. Carroll

The work presented in this dissertation centers on the theme of regression and computation methodology. Functional data is an important class of longitudinal data, and principal component analysis is an important approach to regression with this type of data. Here we present an additive hierarchical bivariate functional data model employing principal components to identify random effects. This additive model extends the univariate functional principal component model. These models are implemented in the pfda package for R. To fit the curves from this class of models orthogonalized spline basis are used to reduce the dimensionality of the fit, but retain flexibility. Methods for handing spline basis functions in a purely analytical manner, including the orthogonalizing process and computing of penalty matrices used to fit the principal component models are presented. The methods are implemented in the R package orthogonalsplinebasis.

The projects discussed involve complicated coding for the implementations in R. To facilitate this I created the NppToR utility to add R functionality to the popular windows code editor Notepad++. A brief overview of the use of the utility is also included.

To Tiffany, Corbin and Carter

ACKNOWLEDGMENTS

The author's research was supported by a grant from the National Cancer Institute (CA57030). I would like to thank Dr. Raymond Carroll for great advice and guidance in this research.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Finding information from data is the soul of statistics. When one wants to find the structure that underlies data we often use regression. Regression methodology for functional data is what we consider here. I present an additive bivariate functional data model that is based on the functional principal component model. This complex model gives rise to many computational issues. Here we present the model and the solution to implementing the algorithm.

Functional data analysis is an important class of longitudinal data that occurs in many fields. Functional data is often complex which creates interesting statistical and computational challenges. Functional data arises from the assumption that longitudinal measurements on a subject reflect points sampled from an underlying function. Examples abound from anywhere that things are expected to change smoothly, such as weight gain or loss over time, or blood concentration levels of medication after being administered. In these examples we know them to vary continuously but they can only be sampled at discrete time points.

In functional principal components we examine the structure that constitutes the curves we have sampled. Each sample is assumed to have a continuous curve for which we have a finite set of points that represent our information about the curve. We desire to extract an estimate of the actual curve, as well as understand the structure of the curve.

In the functional principal component framework, each subject curve is assumed to be composed from a global mean and a weighted sum of principal component

---

This dissertation follows the style of Biometrika.

curves, where each subject has its own set of weights. The principal component functions are common across the entire data set. In this way, by examining the curves across an entire dataset we are able to dissect the curves into their constituent components, and reveal the structure of the data.

In real life, functions are not always functions of only one variable. In this paper we extend the base model into functions of multiple variables. By considering two variables that both act in the principal component framework we are able to handle more complex data and reveal more structure. This model is complex, but so is the data that it is used to analyze. The details of this model are given in Chapter II.

The functional principal component models are computationally expensive to fit. To make the principal component models usable it is important to have an efficient high performance algorithm. Chapter III discusses the details of the R package pfda. This high performance package implements in C code the EM algorithms that fit the functional principal component models. In addition to the additive model presented in Chapter II, the original model for sparse irregular functional data presented by James et al. (2000) and the paired model (Zhou et al., 2008), also based on The James et al. model, are implemented. By implementing the algorithms in high efficiency C code, we are able to see improvements of over 200 times over the execution time for a pure R implementation. What was a ten hour procedure becomes a two minute procedure. This execution time is reasonable, and makes the models practical for researchers to use.

To model the curves in any of the principal component models B-splines are used to reduce the dimensionality of the fit while retaining flexibility to fit the data. B-splines use basis functions that add up to create a spline curve. Any curve that can be fit with any spline curve can also be fit with a B-spline curve of the same degree. Instead of using the standard B-spline basis functions we use an orthogonalized set

of basis functions to make the model identifiable. Chapter IV discusses a method for handling the orthogonalization of the B-spline basis functions as well as computing penalty matrices. This methods allows for computing the transformations without numerical calculus, and reduces the calculus steps to linear algebra operations. This greatly improves speed and accuracy.

Programming these many implementations and complicated algorithms is difficult, having the proper tools greatly improves code readability and programmer efficiency. While programming the packages presented here I created another program to assist me. The NppToR utility adds R language structures and interoperability to the popular Windows code editor Notepad++. Notepad++ is very popular in the programmer community, and adding support for R interaction turns it into a powerful R code editor as well.

CHAPTER II

AN ADDITIVE BIVARIATE HIERARCHICAL FUNCTIONAL DATA MODEL

A. Introduction

In this chapter we discuss the base principal component model and one extension to that base model. Functional data analysis deals with modeling data over time, or equivalently another variable. In functional principal components we attempt to decompose a function into components that describe the function.

Functional data is a special category of analysis for longitudinal data, where the data is assumed to be sampled from underlying functions. In functional data analysis the interest is not only at the discrete sampling points but also in the time or space between the points, as well as the relationship between the functions and other variables. James et al. (2000) introduce a functional principal component model to analyze such data. The original model by James et al. (2000) was extended by Zhou et al. (2008) to jointly model two response curves. Zhou also made several contributions to the algorithms used to fit the model in the univariate case.

This chapter discusses a model that extends the James et al. model to consider functions of two variables. The data required for fitting bivariate curves in practical applications will often out pace the capabilities of the researcher to gather the required data to perform such estimations. To overcome this limitation, which can be thought of as a case of dimensionality, we propose a model where the bivariate function is fit with additive functions from each of the predictor variables.

This model is for longitudinal data, where each subject has repeated observations over the course of the study. Covariates are allowed as subject-specific variables. The response Y is a function of two domain variables, T and X, which vary across

observations. The effects of the covariates Z are assumed to have an additive effect, leading to the general bivariate model: for the i$^{\text{th}}$ individual,

$$Y_i(t,x) \;=\; h_i(t,x) + Z_i^{\mathrm{T}}\theta_Z + \epsilon_i(t) \tag{2.1}$$

where $\epsilon_i(t)$ is random noise with mean zero and variance σ^2.

The bivariate function $h_i(\cdot)$ is difficult to estimate completely nonparametrically. To simplify we assume that the function is additive, so that

$$h_i(t,x) \;=\; \{\mu_T(t) + h_{Ti}(t)\} + \{\mu_X(x) + h_{Xi}(x)\}\,, \tag{2.2}$$

$$\tag{2.3}$$

where $\mu_T(t)$ and $\mu_X(x)$ are the mean fixed effects for the variables T and X, respectively. The random effect curves, $h_{Ti}(t)$ and $h_{Xi}(x)$, are the individual subject-specific deviations from the mean. Each of these functions are assumed to be estimable by a set of principal components.

$$h_{Ti}(t) \;=\; \sum_{j=1}^{K_T} f_j(t)\alpha_i = f(t)\alpha_i; \tag{2.4}$$

$$h_{Xi}(x) \;=\; \sum_{j=1}^{K_X} g_j(t)\beta_j = g(x)\beta_i, \tag{2.5}$$

where $f(t)$ and $g(x)$ denote the sets of principal component functions for the variables T and X, and (α_i, β_i) denote principal component scores, i.e. subject-specific random effects. There are K_T principal componentes for T, and K_X principal components for X.

Section B gives the background and foundations of the functional principal component model as a single function of one variable. Section C gives details of the bivariate hierarchical functional principal component model. Section D gives more details on selecting the parameters that control the fit of the model: the number of

principal components, and the penalty parameters. Section E describes issues that arise in computing the solution. An example is given in Section F, using data from a longitudinal study for AIDS patients on antiviral medication (Lederman et al., 1998; Liang et al., 2003).

B. The Univariate Model

Here we first describe the univariate model, using T and Z. In the univariate functional data framework each observational unit or subject has its own curve that we are interested in modeling. The function is the sum of a mean function of time, $\mu(t)$, plus an individual deviation from the mean, $h_i(t)$, plus additional additive effects from other variables that influence the model, denoted by Z_i, and given as

$$Y_i(t) = \mu(t) + h_i(t) + Z_i^{\mathrm{T}}\theta_Z + \epsilon_i(t),$$

where $\epsilon_i(t)$ denotes random noise. The individual deviations from the mean are restricted to the weighted sum of the principal component functions $f(t) = \{f_1(t), \ldots, f_K(t)\}$ where the weights are the principal component scores $\alpha_i = (\alpha_{i,1}, \ldots, \alpha_{i,K})$, so that $h_i(t) = f(t)^{\mathrm{T}}\alpha_i = \sum_{j=1}^{K} \alpha_{i,j} f_j(t)$. The principal component scores α_i are also assumed independent and identically distributed and to follow a normal distribution with each of the components independent of the others. The univariate principal component model then can be expressed as a mixed effects model. For the first K principal components, in functional form, the model is

$$\begin{aligned} Y_i(t) &= \mu(t) + f(t)\alpha_i + \epsilon_i(t); \\ \alpha_i &\sim \text{Normal}(0, D_\alpha); \\ D_\alpha &= \text{Diagonal}(\sigma_1, \ldots, \sigma_K), \end{aligned} \tag{2.6}$$

with $\sigma_1 > \cdots > \sigma_K$. The ordering of the variances of the scores reflect that the first principal component explains the most variance, the second the second most and so on. For the model to be identifiable the principal components must be orthogonal, and the scores independent. The orthogonality means that

$$\int_{t_{\min}}^{t_{\max}} f_i(t) f_j(t) \mathrm{d}t = \delta_{ij},$$

for $i = 1, \ldots, K$ and $j = 1, \ldots, K$, where δ_{ij} is the Kronecker delta,

$$\delta_{ij} \quad = \quad \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

We use a B-spline representation for the principal components. Let $b(t) = \{b_1(t), \ldots, b_P(t)\}$ be the vector of B-spline basis functions evaluated at t. Let $f(t) = \{f_1(t), \ldots, f_K(t)\}^{\mathrm{T}}$. Each of the functions in the model is expressed in terms of the basis. Let $\mu(t) = b(t)\theta_\mu; f(t) = b(t)\Theta_f$, where θ_μ is a fixed but unknown vector of spline coefficients, and Θ_f is a $P \times K$ matrix of spline coefficients transforming the P basis function $b(t)$ into the K principal component functions $f(t)$.

To enforce the orthogonality constraints on the principal components the B-spline basis functions are restricted to be orthogonal, that is

$$\int_{t_{\min}}^{t_{\max}} b_i(t) b_j(t) \mathrm{d}t = \delta_{ij}.$$

For the principal components to be orthogonal using the orthogonal spline basis functions, the coefficient matrix Θ_f must be orthogonal, i.e. $\Theta_f^{\mathrm{T}} \Theta_f = I$.

In practice, $Y_i(t)$ is observed at the finite set of times $(t_{i1}, \ldots, t_{im_i})$. Let B_i be the B-spline basis matrix resulting from evaluating the basis functions at the time

points $t_i = (t_{i1}, \ldots, t_{im_i})$.

$$B_i = \begin{pmatrix} b_1(t_{i,1}) & \cdots & b_P(t_{i,1}) \\ \vdots, & \ddots, & \vdots \\ b_1(t_{i,m_i}) & \cdots & b_P(t_{i,m_i}) \end{pmatrix}.$$

The model expressed for the finite data and in terms of splines is

$$\begin{aligned} Y_i &= B_i\theta_\mu + B_i\Theta_f + \epsilon_i; \\ \epsilon_i &\sim \text{Normal}(\mathbf{0}, \sigma^2 I_{m_i}), \end{aligned}$$

where $Y_i = \{Y_{i1}, \ldots, Y_{im_i}\}^{\mathrm{T}}$.

James et al. (2000) control the smoothness of the curve estimates by limiting the number of fixed knots for the splines. This can be problematic in many situations, such as small data sets where the number of usable knots is limited. Zhou et al. (2008) propose using a moderate number of knots and employing the method of penalized likelihood. The roughness penalties control the flexibility of the fitted functions $\mu(t)$ and $f_1(t), \ldots, f_K(t)$. The penalties proposed derive from the squared second derivative of the curve to be estimated. This method penalizes directly the smoothness of the curve. The penalty term of the univariate model takes the form

$$\lambda_\mu \theta_\mu^{\mathrm{T}} \int b^{(2)}(t) b^{(2)}(t)^{\mathrm{T}} \mathrm{d}t \; \theta_\mu + \lambda_f \sum_{j=1}^{K} \theta_{fj}^{\mathrm{T}} \int b^{(2)}(t) b^{(2)}(t)^{\mathrm{T}} \mathrm{d}t \; \theta_{fj},$$

where $b^{(j)}(t) = \{b_1^{(j)}(t), \ldots, b_P^{(j)}(t)\}$ is the set of j^{th} derivatives of the B-spline basis functions $b(t)$.

C. The Bivariate Model

The bivariate model is a generalization of the univariate model given in Section B. Equations (2.1) – (2.4) give the basic structure of the additive bivariate hierarchical functional data model. The complete model is

$$\begin{aligned} Y_i(t) &= \left\{\mu_T(t) + f(t)^{\mathrm{T}}\alpha_i\right\} + \left\{\mu_X(x) + g(t)^{\mathrm{T}}\beta_i\right\} + Z_i\theta_Z + \epsilon_i(t); \quad (2.7) \\ \alpha_i &\sim \text{Normal}(0, D_\alpha); \quad (2.8) \\ \beta_i &\sim \text{Normal}(0, D_\alpha); \\ \text{cov}(\alpha, \beta) &= C. \end{aligned}$$

The principal component functions are required to be orthogonal, i.e. $\int f_i(t)f_j(t)\mathrm{d}t = \delta_{ij}$ and $\int g_i(x)g_j(x)dt = \delta_{ij}$ where δ_{ij} is the Kronecker delta. Like the univariate model, D_α and D_β are both diagonal with decreasing elements. In contrast to the univariate model, the model now has two domains for the splines, X and T, each of which requires a set of B-spline basis functions to estimate the splines. Each set of basis functions are required to be orthogonal. Let $B_{T,i}$ be the matrix resulting from evaluating the orthogonalized basis functions for T, $b_T(t)$, at the points $t_i = (t_{i,1}, \ldots, t_{i,m_i})$, and $B_{X,i}$ be the analogous matrix for evaluating the orthogonalized basis functions for X, $b_X(x)$, at $x_i = (x_{i,1}, \ldots, x_{i,m_i})$. The resulting model for finite data is

$$\begin{aligned} Y_i &= B_{T,i}\theta_T + B_{X,i}\theta_X + B_{T,i}\Theta_f\alpha_i + B_{T,i}\Theta_g\beta_i + Z_i + \epsilon_i; \\ \epsilon_i &\sim \text{Normal}(\mathbf{0}, \sigma^2 I_{n_i}); \\ \Theta_f^{\mathrm{T}}\Theta_f &= I; \\ \Theta_g^{\mathrm{T}}\Theta_g &= I, \end{aligned}$$

where Z_i represents the $n_i \times K_z$ matrix of Z variables, which are added to the model in a purely parametric form.

The smoothness of the curves is controlled by four penalty terms, $\lambda_T, \lambda_X, \lambda_f$, and λ_g, one each for the four groups of curves, two for the means, one for the principal components of T, and one for the principal components of X. The penalties, which are added to the log likelihood, take the form

$$\lambda_T \theta_T^{\mathrm{T}} K_t \theta_T + \lambda_X \theta_X^{\mathrm{T}} K_x \theta_X + \lambda_f \sum_{j=1}^{K_T} \theta_{fj}^{\mathrm{T}} K_t \theta_{fj} + \lambda_g \sum_{j=1}^{K_X} \theta_{gj}^{\mathrm{T}} K_x \theta_{gj}$$

where

$$\begin{aligned} K_t &= = \int_{t_{\min}}^{t_{\max}} b_T^{(2)}(t) b_T^{(2)}(t)^{\mathrm{T}} \mathrm{d}t; \\ K_x &= = \int_{x_{\min}}^{x_{\max}} b_X^{(2)}(t) b_X^{(2)}(t)^{\mathrm{T}} \mathrm{d}x. \end{aligned}$$

The bivariate hierarchical functional principal component model is fit using an EM algorithm, with the principal component scores as missing data. The details of the EM algorithm are given in the Appendix. The algorithm must be fit given the parameters of the model: the number of principal components, and the penalty parameters. Methods for selecting these parameters are discussed in Section D.

D. Parameter Selection

The additive model presented here has several parameters that have to be specified or estimated: (a) the number of principal components, and (b) the penalty parameters.

The penalty parameters can be handled in a few different ways; specified by the researcher, or found through an optimization algorithm with a criteria such as Akaike's Information Criteria (AIC). The EM fitting algorithm when the penalties

are given is computationally intensive. In addition the space for finding the penalty parameters is four dimensional, two for each predictor variable, one each for the mean and principal component functions. Optimizing over a four dimensional space, while it may be the ideal, is not always pragmatic for exploratory data analysis with many data sets due to very high computational requirements. We have also witnessed that in some datasets the AIC can suffer from local minima affecting the results, which can be mitigated by changing the starting value of the optimization algorithm.

A quick approach for exploratory data analysis is for the researcher to specify the penalty parameters. Unfortunately, the raw penalty parameters are uninterpretable, and so have little guidance as to what is a reasonable penalty. By transforming the penalties into degrees of freedom, such as in Daniels et al. (2000), the penalties can be specified in terms familiar to researchers. While specifying the penalty parameters can give useful and reasonable results, the results will not be optimal, unless they happen to coincide with the results of optimizing with a criteria.

The number of principal components could likewise be specified, but ideally should be chosen by a criteria like AIC. The search space for finding the number of principal components is only two dimensional and discrete. The optimal penalties are dependent on the number of principal components. For analysis that requires finding both the number of principal components and the penalties, a search for the optimal penalties must be made at each different amount of principal components.

E. Computational Issues

1. Knot Selection

Following Ruppert et al. (2003) we select a nominal number of knots at the quantiles of the distribution. We choose 11 by default. Ruppert et al. (2003) points out that

we should pick enough knots to capture the structure in the data, but not so many that it will unduly increase the computational burden.

2. Identifiability

For the model to be identifiable the principal component curves must be orthogonal, $\int f_i(t)f_j(t)\mathrm{d}t = \delta_{ij}$, otherwise two different curves would be estimating the same portion of the model. Using orthogonal basis functions, this requirement is equivalent to requiring an orthogonal coefficient matrix. This is the case with both variable T and X. Chapter IV gives how to easily handle the orthogonalization of these basis functions in a convenient and completely analytical way.

An identifiability issue that is unique to the additive principal component model is the presence of two intercepts. Either mean function from T or X could estimate the intercept, meaning that the intercept is only identifiable as the sum of the two. To make the model identifiable one of the intercepts must be fixed. To fix one of the intercepts one of the basis functions is removed, either from the basis functions for T or from those for X, we use X but the choice is arbitrary.

F. Example: Aids Data

Zhou et al. (2008) discuss a dataset that considers AIDS patients over time. The study was by the AIDS Clinical Trials Group, ATCG 315 (Lederman et al., 1998; Liang et al., 2003). This study follows HIV-1 infected patients who are being treated by antiviral medication. The study tracks the patients at regular intervals from initial treatment on up to 196 days. The variables of interest in this study are the viral load, measured as plasma HIV RNA copies, and immunological response, measured by the CD4 glycoprotein markers.

Zhou et al. (2008) consider the viral load, and immunological response as two responses and jointly model the responses in the principal component framework. In this paper we consider that immunological response is not only correlated with viral load but consider the possibility that it is actually influenced by it. This bivariate model considers time and viral load as the two domains.

This model serves as as an example of the method and so all recommendations are followed here. The penalties as well as the number of principal components are chosen by AIC. The model is expressed the same as previously with T denoting the time or day of the trial and X denoting the RNA segment measurement of viral load. The parameters are interpreted as θ_T is the coefficient vector for the mean CD4 response over time and θ_X is the mean response for the viral load. The principal components Θ_f and Θ_g are corresponding to the day and viral load respectively. For the model under consideration there is no other variable effects considered, so Z does not appear in the model, and that step in the estimation is skipped.

Figure 1 shows the curves that are estimated from the model. For the two principal components found for viral load, the variance of the principal component scores is approximately one quarter of the variance for the first principal component. Figure 2 shows a sample of reconstructed curves from the model.

The additive model can be complicated and confusing if researchers are not careful. Since the model considers both variables together rather than separately it helps to view the model in a 3 dimensional plot. Figure 3 show the additive model for the aids data. Another point to consider and examine when interpreting mean curves is that principal components can play a larger role than the mean functions. it is important to examine the distribution of the principal component scores, because while they are assumed to be mean zero, they are not constrained to be mean zero. It is entirely possible that when viewing results that don't entirely make sense, such as

mean curves that have the wrong slope, the principal components may be absorbing that effect.

By examining the mean and principal component functions we conclude that the time effect is nominal, reducing to essentially an intercept for the model. The majority of the variability is explained by the viral load. The principal components of time have a little effect on adjusting the intercept for each subject. The mean curve for the viral load shows that as viral load increases the immune response decreases, the basic workings of the HIV virus. The principal components show how the mean curves are altered for each subject in the study by adjusting the curvature of the response.

G. Summary

The bivariate additive model presented here goes beyond what has been done previously. By investigating the effects of multiple variables in the principal component framework we are able to correct for the correlation in the effects that may not be apparent in separate models. The additive bivariate principal component model gives deeper understanding into the structure of the data, than was available previously.

This analysis agrees with the results obtained by Zhou et al. (2008) on the general points. We restrict our analysis to a subset of the range that Zhou et al. considers because of the uncertainty in the later portions of the study, which Zhou et al. recognize. An interesting point that was not made by Zhou et al. is that the after accounting for the effect of viral load on the immunological response the time since baseline was small. This makes clinical sense, since the purpose of the study is to investigate antiviral medications, the assumption is that the medications will inhibit or suppress the viral load and allow the immunological response to recover.

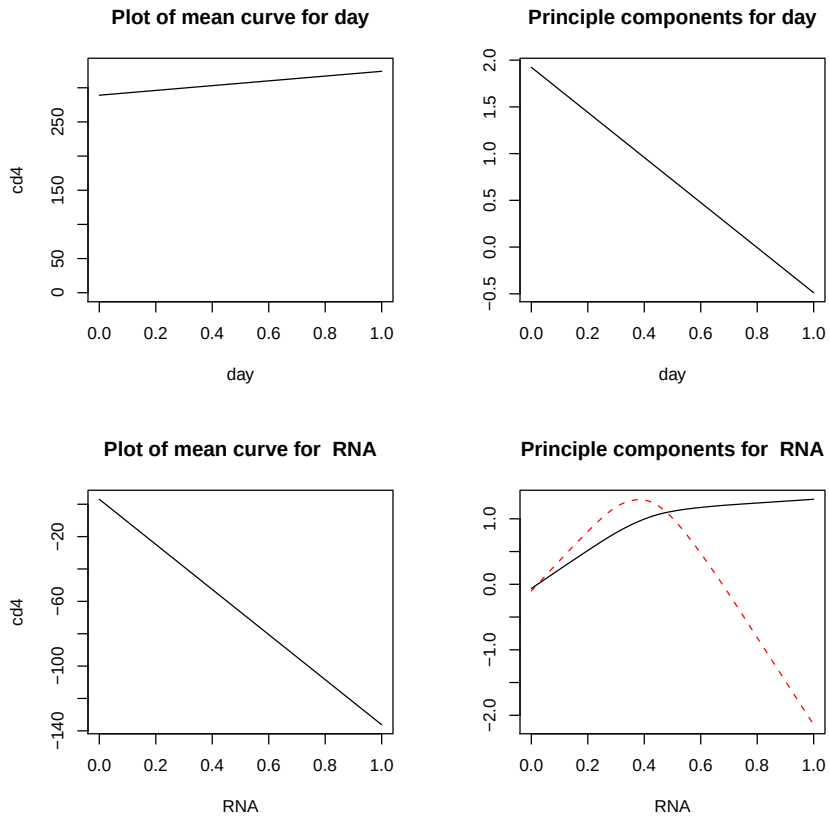

Fig. 1. The estimates of the mean and principal component curves from the AIDS example model.

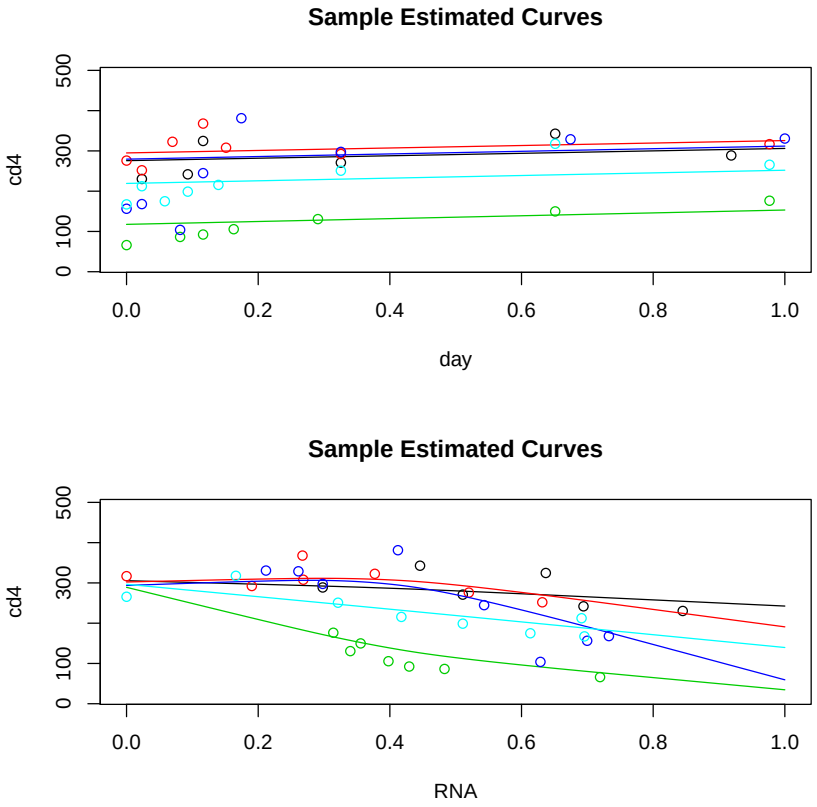

Fig. 2. A sample of curves estimated from the AIDS model, shown for both day and RNA. Each color represents a different subject.

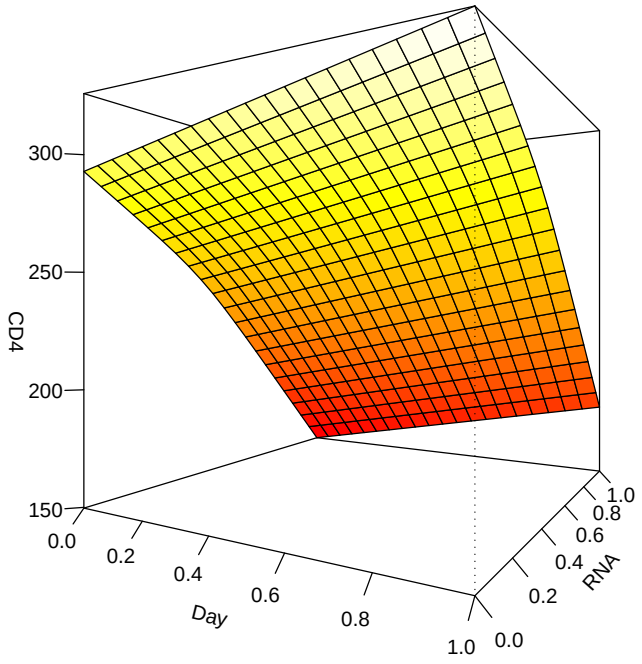

Fig. 3. The additive model seen in a 3D perspective plot.

CHAPTER III

THE PFDA PACKAGE: PRINCIPAL COMPONENT ANALYSIS FOR FUNCTIONAL DATA

A. Introduction

Functional principal components is a useful way to look at complex longitudinal data. One disadvantage is that the models are complex and difficult to fit. Not only is the EM fitting algorithm computationally expensive, but there are many model parameters that must also be estimated. An efficient implementation is essential to making these models usable. I have implemented the fitting algorithms in the R package `pfda` .

The structure of this chapter is as follows. Section (B) gives a brief introduction to the models that the package can fit. Section C gives an introduction to the package and its primary function `pfda()`. Sections (D)–(G) discusses the arguments to `pfda()` in detail including allowable values and the defaults. The programming strategy, and problems that were overcome, are discussed in Section H.

B. The Models

There are five functional principal component models that are implemented in the `pfda` package. The models include univariate, paired and additive models. The responses for the paired and univariate model can also include binary response variables. Chapter II gives the details of the models, but for the sake of completeness in this chapter we will review in general terms the functional principal component models.

The univariate model in the functional form is

$$Y_i(t) = \mu(t) + f(t)\alpha_i + \epsilon_i(t), \tag{3.1}$$

where $\epsilon_i(t)$ denotes random noise. The model denotes an overall mean by $\mu(t)$ and the subject specific deviation from the mean by $f(t)\alpha(i)$, which is formed by the weighted sum of basis functions $f(t) = \{f_1(t), \ldots, f_P(t)\}$. The model was introduced by James et al. (2000). The principal component scores, α_i, are assumed to be independent and identically distributed. They follow a multivariate normal distribution, mean zero, and independent but decreasing variances. The errors are independent normal random variables with common variance.

The paired model of Zhou et al. (2008) considers two response variables that are both modeled with the principal component framework shown in Equation (3.1).

$$\begin{aligned} Y_i(t) &= \mu(t) + f(t)\alpha_i + \epsilon_i(t); \\ Z_i(t) &= \nu(t) + g(t)\beta_i + \zeta_i(t). \end{aligned}$$

The difference between the paired model and two univariate models are that the principal component scores (α_i, β_i) are assumed to be correlated. This functional correlation is a primary interest of the model.

The additive model considers a single variable that has two predictor variables. The details of the univariate and additive models are given in Chapter II covering the additive model. The model, in the functional form, is

$$Y_i(t) = \{\mu_T(t) + f(t)\alpha_i\} + \{\mu_X(x) + g(x)\beta_i\} + \epsilon_i(t).$$

The mean functions for each of the T and X are represented by $\mu_T(t)$ and $\mu_X(x)$. The two sets of principal component functions are $f(t)$ and $g(x)$.

The model in Equation 3.1 is used directly for a model with a continuous re-

sponse, but can also be applied to models with a binary response. The binary response W, is assumed to be an indicator for the continuous latent variable, Y. By including a step for the stochastic approximation of Y, the model can be applied to binary responses. The paired model, can also handle a binary response in the same manner, but only one of the responses is allowed to be a binary variable. The additive model does not support binary response variables.

C. The Package

The principal component models for functional data analysis, presented in Section (B), are implemented in the R package `pfda` . The models and fitting algorithms can be accessed through the gateway function `pfda()`. Figure (4) shows the typical arguments for the pfda function. Section (D) discusses the structure of the formulas for the different models in detail. The `driver` argument, which can manually specify the model to fit, is also considered there. The `data` argument is a data frame or environment where the variable names used in the formulas can be resolved to data.

The `knots` argument indicates the location of the knots used for the B-splines used in the curves. For additive models the knots argument must be a list of length 2. Ambiguity can be erased by naming the elements of the list corresponding to the names use in the formula, but is not necessary and the elements of the list are assumed to follow the same order as the variables appear in the formula. The number of knots can also be controlled through the control structure that is discussed in Section (G). The default is to use eleven knots at the quantiles of the data.

The penalties are fit through either the `penalties` or `df` arguments, but not both. The `penalties` argument gives the raw penalties, which are uninterpretable. The `df` argument gives an approximate number degrees of freedom for the curve.

```
pfda(formula,    # Specifies variables and model
     data,       # Data frame of environment
     knots,      # Knot placement
     penalties,  # Penalty parameters
     df,         # Penalties specified in degrees of freedom
     k,          # Number of principal components
     control,    # List of control parameters
     driver)     # Clarifies model
```

Fig. 4. The pfda function for fitting functional principal components.

The reason that the degrees of freedom are approximate is because the true degrees of freedom are complex and depend on parameters only known after the model is fit. Penalties are discussed in detail in Section (F).

The number of principal components is specified by `k`. For the paired and the additive models, there are two sets of principal components and `k` should be a vector of length 2. The number of principal components are discussed in Section (E).

The model also has several several extra parameters that control the EM algorithm, for example, the convergence tolerance. The `control` argument is a list of these control values. The `pfdaControl()` function generates the list of appropriate defaults, and should always be used. The controls for fitting the algorithm are discussed in Section (G).

D. Formulas

The `formula` argument of `pfda()` determines both the variables used and the model that is fit. The model to fit is identified by the structure of the formula used. I implement two new formula operators that only work with the `pfda` package;

- `%&%` bind together variables, on the left side of the formula indicates the paired model, on the right an additive variable;
- `%|%` is always on the right side of the formula and indicates the domain variable(s) on the left and subject identifier on the right of this operator.

For models that support extra, non principal component, variables, univariate and additive models, the `+` operator indicates these. The models that can support a binary response can be identified automatically if the variables passed into pfda are already class logical or factor.

The `driver` argument to `pfda()` can specify the exact model to be fit. This argument does not usually need to be specified since the model to be fit is inferred from the structure of the formula.

- `Y ~ T %|% ID` is the formula structure for a univariate model;
- `Y %&% Z ~ T %|% ID` gives a paired model;
- `Y ~ T %&% X %|% ID` fits an additive model over the variables T and X.

The options for the driver function are:

- "single.continuous"
- "single.binary"
- "dual.continuous"

- "dual.mixed"
- "additive"

The only times that a user may need to specify the driver is to use one of the binary cases, "single.binary" or "dual.mixed".

E. Number of Principal Components

The number of principal components is an important parameter in all of the models, and is controlled with the `k` argument. For the models with two sets of principal components, any of the paired or the additive models, the number of principal components needs to be a vector or length 2.

When the number of principal components are not given, they are chosen by AIC. Figure (5) shows the algorithm in pseudo code for determining the number of principal components. This incremental method is the only possible solution, since fitting too many principal components in this framework is unstable. For modeling the paired responses, ignoring the correlation turns the model into two univariate cases, and the number of principal components can be determined by fitting the models ignoring correlation, then refitting for the joint model, as recommended in Zhou et al. (2008).

The additive model is complex. Fitting the number of principal components to the variables separately then combining like in the paired models can easily lead to over fitting the model. The additive model requires a generalization of the algorithm in Figure (5). This step-wise procedure considers both directions before taking a step, but will not add more than one principal component for either T or X at one time.

One problem with finding the optimal number of principal components is that the optimal penalty parameters are dependent on the number of principal components.

0 let $k = 1$

1 Fit model with k principal components. Let AIC_k be the AIC for the model.

2 Fit model with $k + 1$ principal components. Let AIC_{k+1} be the AIC for this model.

3 If $AIC_{k+1} < AIC_k$, set $k = k + 1$ and go to 1. Otherwise stop with the optimal model that has k principal components.

Fig. 5. The algorithm for finding the optimal number of principal components for the univariate model.

Selection of the penalty parameters is addressed in the next section.

F. Penalty Parameter Specification and Optimization

Penalty parameters control the smoothness of the curves estimated from the models. Since the penalty parameters are so important to fitting the model the penalty system was made flexible. Users can specify penalty parameters directly through the `penalties` argument to `pfda()`. As has been stated, these have complicated units and so are uninterpretable. The second option is to specify the number of degrees of freedom through the `df` argument. This inverts an approximation of the degrees of freedom to obtain penalty parameters for fitting the EM algorithm. This is useful for exploratory data analysis because researchers often, from their subject expertise, have a good idea of how smooth a curve will need to be to fit the data. Both arguments are restricted to positive values, and degrees of freedom must have reasonable values; for example, a line cannot have less than one degree of freedom here, and usually has at least 2. The `penalties` and `df` arguments should not be specified simultaneously,

$$\begin{array}{cc} & \begin{array}{cc} Mean & PC \end{array} \\ \begin{array}{c} X \\ T \end{array} & \begin{pmatrix} \lambda_T & \lambda_f \\ \lambda_X & \lambda_g \end{pmatrix} \end{array}$$

Fig. 6. Illustration of format of penalty matrix.

but if they are the `penalties` argument has priority.

The `penalties` and `df` arguments are given as a vector or matrix of parameters. For the single models this is simply a vector with the penalties for mean and principal components, respectively. With those models with four penalties, paired and additive, the penalties are given for the mean functions first then the penalties for the principal components. For the additive model this is $(\lambda_T, \lambda_X, \lambda_f, \lambda_g)$. This arrangement reflects that the penalties form a two by two matrix with the rows being the variables, the first column for the mean, and the second for the principal components, illustrated in Figure 6.

One way that the penalty system is flexible is that the penalties can be partially specified, through either the `penalties` or `df` arguments. If an element of either of these is specified as `NA` the parameter will be chosen by optimizing a criteria. The criteria to be used in optimizing is specified through the `control$penalty.method` object. Valid options are `"CV"` for cross validation and `"AIC"` for Akaike's Information Criteria. Cross validation is not recommended but is an option for those who may prefer it. The number of folds for cross validation is specified with `control$nfolds`. The number of folds, which is specified as a natural number greater than two, specifies the number of equal proportions to with hold and estimate from the remaining data.

AIC is the recommended methods for optimizing the penalty parameters and is chosen by default. The optimization is performed through the R function optim,

which is the recommendation when optimizing over multiple dimensions; a common goal when optimizing the penalty parameters. AIC can suffer from local minima when optimizing. This can be overcome by specifying the `optim.start` parameter of the `pfdaControl()` function. This must be specified in the terms of raw parameters. Also to help with the optimization the default starting point is a low 2.1 degrees of freedom to favor smooth lines. The default optimization method is the Nelder-Mead method, and can be changed with specifying `optim.method` in `pfdaControl()`, which might be desired if optimizing only one penalty parameters.

G. Fitting Controls

The values needed for controlling the fit such as specifying a minimum variance or the tolerance to determine convergence are specified in the `control` argument to the `pfda()` function. Several values have been discussed in the previous sections. The value for the `control` argument is created by the `pfdaControl()`. Widely appropriate defaults are selected for values not specified, so it is unlikely that the user will have to specify many of these parameters, but for the sake of completeness they are discussed here. Only those named values specified by `pfdaControl()` are discussed here even though there are some that can be specified, such as `optim.start`, that are not created if they are not explicitly provided. Those are either discussed elsewhere and appropriate defaults cannot be determined in advance, or they are considered to be for internal use only.

- `penalty.method` was discussed in Section (F) and specifies the optimizing criteria. Only ‘‘`CV`" and ‘‘`AIC`") are supported.

- `minimum.variance` specifies the minimum variance allowed and defaults to 1^{-4}. Also used for small numbers that are used to stabilize some computations.

- `convergence.tolerance` defaults to 0.01 and is the criteria for the convergence of the EM algorithm. Convergence criteria is computed as the sum of the absolute relative difference of all fixed components.

- `max.iterations` determines the maximum number of iterations allowed before the algorithm is determined to have failed to converge. The default is ten thousand, which was determined on reasonable datasets with extreme parameters that showed slow convergence. It is typical that, on reasonable dataset with close to optimal parameters, the number of iterations stay below fifty.

- `nfolds` Used only with cross validation to determine the number of cross validated sets.

The next three arguments are control parameters for the stochastic approximation involved with the univariate and paired models that use binary responses;

- `binary.k0` and `binary.kr` control the number of samples to draw during the burn-in period and afterwards, and

- `binary.burnin` specifies the length of the burn-in period.

The last argument `nknots` controls the default number of knots that are chosen for creating the B-spline basis functions. The knots are chosen by the quantile function, and only unique knots are taken. These knots then are passed to `OrthogonalSplineBasis` from the `orthogonalsplinebasis` package to create the basis functions used.

H. Approach to Implementation

1. Introduction

The models in the `pfda` package are complex and computationally intensive. Naive approaches to program the algorithms for the functional principal components fail to

reach performance requirements.

When prototyping these algorithm I quickly found that R code was far too slow for implementing the algorithms. The R language is interpreted and convenient for programming. Unfortunately, convenience for programming is bought with speed and efficiency. Luckily R allows for compiled extensions that give great control over speed and efficiency.

I found that the primary culprit for the naive program is the memory management system. R has a garbage collection system , which is a programming language feature that allocates memory as variables are created but does not release memory until it gets to a point where there are too many unused variables. The algorithms involve several large matrices that lead to large intermediate values that are allocated in each function call and then left for the garbage collector. The problem with this in the algorithms like those used in the this package are that it adds too much overhead to the base computations.

The compiled code allows for very fine control over memory management. By moving the EM algorithm to C code I was simultaneously able to optimize the memory and use directly the most appropriate multiplication functions from the BLAS and LAPACK high performance libraries that R is built on.

The final solution has the EM algorithm programed in C using the BLAS and LAPACK routines, and is wrapped around R to make the interface easy and accessible. The R portion of the code also handles the optimization for the penalties and the number of principal components.

The final solution greatly improves the speed over the pure R approach. In terms of quantifying the improvement from the naive implementation to the final hybrid approach we have seen anywhere from 200 times faster with the hybrid approach, so that the computation time is measured in minutes and seconds rather than hours and

days.

2. C implementation

It makes programming sense to have all the algorithms together in one package. All the models have similarities leading to the EM algorithms being very similar and in particular having steps that are duplicated or special cases of steps for other models. Each model requires its own function for The C code. By programming the steps once and reusing them I can ensure code quality. Figure 7 shows an example of computing the residuals. This function is used in every step.

The memory in the C code is optimized by allocating once the total amount of memory needed and using it as a stack for all the functions. This adds a burden of computing the maximum requirements beforehand, but greatly speeds up the algorithm. Figure 8 shows the core function for the univariate algorithm. It also shows the `pfdaAlloc_d()` function that handles allocating space for temporary variables from the memory pool `dp`, structured as a stack. Each function that uses temporary variables has a `dp` argument that holds a pointer to the current position in the allocated stack.

3. R implementation

In the R code the primary function is the `pfda()` function. This acts as a gateway function to the models. Each model has a function, called a driver function, that handles optimizing the parameters appropriate to the model and passing the execution on to the compiled portion of the code.

The `pfda` package takes full advantage of the programming structures in R. Similar to the C code with the steps or the EM algorithm being similar, the optimization steps are also very similar. The similarities are gathered together in blocks of code

```
void single_c_resid(
  double       * const Ry,
  double const * const y,
  double const * const Z,
  int    const * const nobs,
  int    const * const M,
  int    const * const N,
  int    const * const kz,
  int    const * const k,
  double const * const B,
  int    const * const p,
  double const * const tz,
  double const * const tm,
  double const * const tf,
  double const * const alpha,
  int const * const dl, double*dp)
{
  if(Ry!=y)dcopy_(M,y,&one,Ry,&one);
  if(Z && kz && tz && *kz)dgemv_(&NoTrans, M, kz, &mOne,
    Z, M, tz, &one, &dOne, Ry, &one);
  pfda_computeResid( Ry, Ry, nobs, M, N, k, B, p, tm, tf,
    alpha, dl, dp);
}
```

Fig. 7. The C function for computing residuals.

```
void single_c_core(
  double * const y, // ... other input variables
  double * dp, int * ip){
  double * btb = pfdaAlloc_d(*p**p**N,&dp);
  pfda_computebtb(btb,N,B,M,p,nobs,dl);
  pfda_s_i(tm, tf, alpha, Da, aa, sigma, y, nobs, M, N, k,
    B, btb, p, minV, dl, dp, ip);
  int I=0;
  double    sigma_old=0, convergenceCriteria=0;
  double * tmOld  = pfdaAlloc_d(*p     , &dp);
  //Allocate other variable for computing convergence
  while(I < *maxI){
    // setup for convergence DELETED
    single_c_E( alpha, aa, Saa, y, Z, B, tz, tm, tf, Da,
      sigma, nobs, N, M, kz, k, p, dl, dp, ip);
    single_c_unpenalized( tz, y, Z, B, tm, tf, alpha, nobs,
      N, M, kz, k, p, dl, dp, ip);
    // other steps and compute convergence criteria DELETED
    I++;if(convergenceCriteria < *tol)break;
  }
  // finishing code DELETED
}
```

Fig. 8. The core function for computing the EM algorithm for the univariate model.

called expressions. These blocks of code are combined together to form the driver functions, inserted where appropriate with `eval`. Although this approach adds complexity to the execution structure it improves maintainability and makes performance and behavior uniform.

Figure 9 shows how the R code optimizes the penalties. This is a special helper function that has to be declared to be internal to the driver function, giving it special privileges of accessing variables from the environment of the driver function call. It also employs a special function `RecallWith` that allows all the parameters to be the same except those specified in the arguments of `RecallWith`. This is one code block can, by careful programming, handle the optimization of the penalty parameters for all the cases, both those with two and those with four penalty parameters.

I. Conclusion

The solution to functional principal components presented here and in the `pfda` package, is efficient in many ways. It is accessible and easy to learn for the user. It is also has high performance. It is fast and memory efficient. The models are complex, but by choosing reasonable defaults for the parameters it can and optimizes those it cannot, the functions presented here keep complexity to a minimum.

The pfda package can be downloaded from the Comprehensive R Archive Network, CRAN. The development versions can be downloaded from The `pfda` package homepage at R-forge, `http://pfda.r-forge.r-project.org`.

```
.F.optimize.penalties<-function(){
  pix<-which(is.na(penalties))
  if(control$penalty.method=='CV'){
    # Cross validation code removed
  } else if(control$penalty.method=='AIC') {
    message("optimizing penalties using AIC")
    aicf<-function(pen){
      p<-penalties
      p[pix]<-exp(pen)
      if(any(is.infinite(p)))return(Inf)
      m<-try(RecallWith(penalties=p,fname=fname),silent=TRUE)
      if(class(m)[1]=="try-error") NA else AIC(m)
    }
    if(is.null(control$optim.start))
      control$optim.start<-
        rep(l.from.df(2.1,Bt,Kt),length(pix))
    optimpar<-optim(log(control$optim.start),aicf,
      method=control$optim.method)
    penalties[pix] <- exp(optimpar$par)
    RecallWith(penalties=penalties,fname=fname)
  }
}
```

Fig. 9. The code block for optimizing penalties.

CHAPTER IV

A COMMENT ON THE ORTHOGONALIZATION OF B-SPLINES BASIS FUNCTIONS AND THEIR DERIVATIVES

A. Introduction

The intent of this chapter is to show how to accurately, simply and quickly construct orthogonal B-spline basis functions, their derivatives, of any order, and their integrals. This has important impact on a recent paper by Zhou, et al. (2008), as we now describe.

Zhou, et al. consider a functional data analysis problem based upon a principal components approach. For the i^{th} individual, they propose the model

$$\begin{aligned} Y_i(t) &= \mu(t) + \sum_{j=1}^{k} f_j(t)\alpha_{ij} + \epsilon_i(t) \\ &= \mu(t) + f(t)^{\mathrm{T}}\alpha_i + \epsilon_i(t), \end{aligned}$$

where $\mu(t)$ is the overall mean, f_j is the j^{th} principal component function, $f = (f_1, \ldots, f_k)^{\mathrm{T}}$, and $\epsilon_i(t)$ is the random error. The principal components are subject to the orthogonality constraint $\int f_j f_l = \delta_{jl}$, the Kronecker delta.

Suppose that $b(t) = \{b_1(t), \ldots, b_q(t)\}^{\mathrm{T}}$ is an orthogonal spline basis with dimension q. Let θ_μ and Θ_f be, respectively, a q-dimensional vector and a q by k matrix of spline coefficients. Zhou, et al. represent the mean and principal component functions by a linear combination of the basis functions, $\mu(t) = b(t)^{\mathrm{T}}\theta_\mu$ and $f(t)^{\mathrm{T}} = b(t)^{\mathrm{T}}\Theta_f$. This yields the reduced rank principal components model,

$$Y_i(t) = b(t)^{\mathrm{T}}\theta_\mu + b(t)^{\mathrm{T}}\Theta_f\alpha_i + \epsilon_i(t),$$

where $\epsilon_i(t) \sim (0, \sigma_\epsilon^2)$, $\alpha_i \sim (0, D_\alpha)$, $D_\alpha =$ diagonal matrix, subject to

$$\Theta_f^{\mathrm{T}}\Theta_f = I, \quad \int b(t)b(t)^{\mathrm{T}}\mathrm{d}t = I. \tag{4.1}$$

The equations in (4.1) imply that

$$\int f(t)f(t)^{\mathrm{T}}\mathrm{d}t = \Theta_f^{\mathrm{T}} \int b(t)b(t)^{\mathrm{T}}\mathrm{d}t\,\Theta_f = I,$$

which are the usual orthogonality constraints on the principal component curves, and show the necessity of using an orthogonalized set of basis functions. The parameters of the model to be estimated are θ_μ, Θ_f, $D_\alpha = \mathrm{cov}(\alpha_i)$, and σ_ϵ^2.

If $\mathcal{L}_i(\Psi)$ is the log likelihood function of the parameters Ψ of the model, they then propose a penalized log likelihood of the form

$$\sum_{i=1}^{n} \mathcal{L}_i(\Psi) + \lambda_\mu \theta_\mu \int b''(t)b''(t)^{\mathrm{T}}\theta_\mu \mathrm{d}t + \lambda_f \sum_{j=1}^{d} \theta_{fj}^{\mathrm{T}} \int b''(t)b''(t)^{\mathrm{T}}\mathrm{d}t\,\theta_{fj}. \tag{4.2}$$

Zhou, et al. (2008) propose using numerical integration to construct approximately orthogonal basis functions and approximate second derivatives.

The purpose of this chapter is to show that there exist results in the literature (Qin, 2000) that enable much faster, simple and exact computation of the orthogonal basis functions and direct calculation of their second derivatives.

An outline of this chapter is as follows. In Section B, we give a matrix representation for B-splines , while Section C shows how to orthogonalize them. Section D shows how to compute the derivatives of the orthogonalized basis functions simply and efficiently, while Section E shows how to compute the integral and hence the penalties in (4.2) exactly and efficiently.

B. Qin's Matrix Representation

In this chapter we assume that there are $n+1$ knots $\{t_0, \ldots, t_n\}$ and that we are using piecewise polynomials of degree $k-1$. Therefore, there are $p = n-(k-1)$ basis functions $b_k(t) = (b_{1,k}(t), \ldots, b_{p,k}(t))^{\mathrm{T}}$. Let $V = (v_1, \ldots, v_p)^{\mathrm{T}}$ represent the control points for the spline curve, which could represent $\theta_m u, \theta_n u, \theta_f$, or θ_g in Zhou, et al.

Qin's representation is formed specifically for a given knot interval

$$t \in [t_i, t_{i+1}), t_i < t_{i+1}.$$

In this given interval there are k non-zero B-spline basis functions. We will represent these by $B_{i,k}(t)$. Qin's representation then gives that

$$B_{i,k}(t) = U(t,i)^{\mathrm{T}} M_k(i)$$

where

$$\begin{aligned} U(t,i) &= (1, u, \ldots, u^{k-1})^{\mathrm{T}} \text{ and} \\ u &= (t - t_i)/(t_{i+1} - t_i). \end{aligned}$$

The segment of the curve on the interval $t \in [t_i, t_{i+1})$ can then be represented by

$$c_i(t) = B_{i,k}(t) I_k(i) V = U(t,i)^{\mathrm{T}} M_k(i) I_k(i) V$$

where $I_k(i) = [\mathbf{0}_{i-k}\ I\ \mathbf{0}_{p-i}]$ is a matrix that serves the purpose of selecting the basis functions that are nonzero in the specified interval and $\mathbf{0}_j$ is a $k \times j$ matrix of zeros.

The $M_k(i)$ matrices are formed by the recursion relationship

$$
\begin{aligned}
& M_k(i) \\
& = \begin{pmatrix} M_{k-1}(i) \\ \mathbf{0} \end{pmatrix} \\
& \quad \times \left\{ \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} + \left[\begin{pmatrix} \mathbf{0} & \mathbf{I} \end{pmatrix} - \begin{pmatrix} I & \mathbf{0} \end{pmatrix} \right] \begin{pmatrix} d_{0,i-k+2} \\ d_{0,i-k+3} \\ \vdots \\ d_{0,i} \end{pmatrix} \right\} \\
& \quad + \begin{pmatrix} \mathbf{0} \\ M_{k-1}(i) \end{pmatrix} \\
& \quad \times \left\{ \left[\begin{pmatrix} \mathbf{0} & I \end{pmatrix} - \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \right] \begin{pmatrix} d_{1,i-k+2} \\ d_{1,i-k+3} \\ \vdots \\ d_{1,i} \end{pmatrix} \right\}
\end{aligned}
$$

and $M_0(i) = [1]$, where

$$d_{0,j} = \frac{t_i - t_j}{t_{j+k-1} - t_j}, \text{ and } d_{1,j} = \frac{t_{i+1} - t_i}{t_{j+k-1} - t_j},$$

with the convention that $0/0 = 0$.

From this specification we can define the basis functions and subsequently the entire curve. The curve is specified as the sum of the individual curve segments each

on its respective interval. The curve is then represented as

$$\begin{aligned} C_k(t) &= \sum_{i=k-1}^{n-k} c_i(t)\mathtt{1}_{[t_i,t_{i+1})}(t) \\ &= \sum_{i=k-1}^{n-k} \mathtt{1}_{[t_i,t_{i+1})}(t)U(t,i)^{\mathrm{T}}M_k(i)I_k(i)V, \end{aligned}$$

where $\mathtt{1}$ denotes the indicator function. The basis functions given in the matrix specification are

$$b_k(t) = \sum_{i=k-1}^{n-k} \mathtt{1}_{[t_i,t_{i+1})}I_k(i)^{\mathrm{T}}M_k(i)^{\mathrm{T}}U(t,i).$$

C. Orthogonalization

To orthogonalize the basis functions we search for a linear transformation, Λ, such that $\tilde{b}(t) = \Lambda b(t)$ forms an orthogonal set of functions. Let

$$\Sigma = \int_{t_{k-1}}^{t_{n-(k-1)}} b(t)b(t)^{\mathrm{T}}\mathrm{d}t$$

be the positive definite matrix formed by the integral of the outer product of the vector of B-spline basis functions with itself over the defined domain. Then, if $\Lambda = \Sigma^{-1/2}$ is the inverse of the square root of the matrix, $\tilde{b}(t)$ forms an orthogonal set of functions.

$$\begin{aligned} &\int_{t_{k-1}}^{t_{n-(k-1)}} \tilde{b}(t)\tilde{b}(t)^{\mathrm{T}}\mathrm{d}t \\ &= \int_{t_{k-1}}^{t_{n-(k-1)}} \Lambda b(t)b(t)^{\mathrm{T}}\Lambda^{\mathrm{T}}\mathrm{d}t \\ &= \Sigma^{-1/2}\int_{t_{k-1}}^{t_{n-(k-1)}} b(t)b(t)^{\mathrm{T}}\mathrm{d}t\Sigma^{-1/2} \\ &= I. \end{aligned}$$

Thanks to the matrix representation given above, the computation of Σ becomes simple and can be computed without any complicated integration, as follows:

$$\begin{aligned}\Sigma &= \int_{t_{k-1}}^{t_{n-(k-1)}} b(t)b(t)^{\mathrm{T}}\mathrm{d}t \\ &= \int_{t_{k-1}}^{t_{n-(k-1)}} \sum_{i=0}^{n}\sum_{j=0}^{n} \mathbb{1}_{[t_i,t_{i+1})}\mathbb{1}_{[t_j,t_j+1)} \\ &\qquad \times I_k(i)^{\mathrm{T}}M_k(i)^{\mathrm{T}}U(t,i)U(t,j)^{\mathrm{T}}M_k(j)I_k(j)\mathrm{d}t.\end{aligned}$$

The integral can be brought into the summations and note that the product of indicator functions results in a Kronecker delta that removes the necessity of the double summation.

$$\mathbb{1}_{[t_i,t_{i+1})}\mathbb{1}_{[t_j,t_j+1)} = \delta_{i,j} \equiv \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

This reduces the expression to

$$\begin{aligned}\Sigma &= \sum_{i=0}^{n}\int_{t_{k-1}}^{t_{n-(k-1)}} \mathbb{1}_{[t_i,t_{i+1})}I_k(i)^{\mathrm{T}}M_k(i)^{\mathrm{T}}U(t,i) \\ &\qquad \times U(t,i)^{\mathrm{T}}M_k(i)I_k(i)\mathrm{d}t \\ &= \sum_{i=0}^{n} I_k(i)^{\mathrm{T}}M_k(i)^{\mathrm{T}} \\ &\qquad \times \int_{t_i}^{t_{i+1}} U(t,i)U(t,i)^{\mathrm{T}}\mathrm{d}tM_k(i)I_k(i). \qquad (4.3)\end{aligned}$$

This result is due to the fact that only the $U(t,i)$ depends on t. To further simplify we focus on the integral at the center of the last expression. Note that in $U(t,i)$ $u = (t-t_i)/(t_{i+1}-t_i)$, using this as a substitution and letting $U = (1,u,\ldots,u^k-1)^{\mathrm{T}}$, we

have the simplification

$$\int_{t_i}^{t_{i+1}} U(t,i)U(t,i)^{\mathrm{T}}\mathrm{d}t = (t_{i+1}-t_i)\int_0^1 UU^{\mathrm{T}}du. \tag{4.4}$$

The expression in (4.4) can be evaluated simply and results in a Hankel matrix of the form

$$\int_0^1 UU^{\mathrm{T}}du = \Delta = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{k} \\ \frac{1}{2} & \frac{1}{3} & & \iddots & \vdots \\ \frac{1}{3} & & \iddots & & \vdots \\ \vdots & \iddots & & & \vdots \\ \frac{1}{k} & \cdots & \cdots & \cdots & \frac{1}{2k-1} \end{pmatrix}. \tag{4.5}$$

Combining (4.3) and (4.5) we get the simple representation

$$\Sigma = \sum_{i=0}^{n} I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} \Delta M_k(i) I_k(i),$$

which can be easily computed. Then Λ can be computed by Cholesky or eigenvalue decomposition. Unless there is an extraordinary number of knots and basis functions, the computation of the square root matrix will not present any difficulties.

D. Derivatives

Qin notes that with this matrix representation taking derivatives also becomes a rather simple process. The derivative of the spline function is the derivative of the $U(t)$ vector times the same matrix of coefficients times the additional weight of the width of the interval. Shown as follows:

$$C'(t) = b'(t)^{\mathrm{T}} V \tag{4.6}$$

and

$$b'(t) = \sum_{i=k-1}^{n-k} \mathtt{1}_{[t_i,t_{i+1})} I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} U'(t,i) \tag{4.7}$$

where the derivative is taken with respect to t. By the same substitution for u that was used in section C with integrals, the derivative can be reduced to a general form for all intervals. This form can also be expressed in terms of a matrix operation. To express this, notation will need to be developed. Let A be a $k \times k$ nilpotent matrix with ones in the subdiagonal immediately below the primary diagonal. Let B be a diagonal matrix with the elements $1, \ldots, k$ on the main diagonal.

$$A = \begin{pmatrix} \mathbf{0}^{\mathrm{T}} & 0 \\ \mathbf{I} & \mathbf{0} \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & \cdots & k \end{pmatrix}$$

Then we define the derivative matrix as the product of these two matrices $D = AB$. Thus we can express the derivative of U as $U' = DU$. Moreover, the process can be iterated to obtain the ℓth derivative given by

$$U^{(l)}(t,i) = (t_{i+1} - t_i)^{-l} D^l U^{(l)}(t,i). \tag{4.8}$$

Combining the equations (4.6–4.8) yields a general representation of the derivatives of splines and the derivative of the basis functions.

$$C^{(l)}(t) = b^{(l)}(t)^{\mathrm{T}} V; \tag{4.9}$$

$$b^{(l)}(t) = \sum_{i=k-1}^{n-k} \mathbb{1}_{[t_i,t_{i+1})}(t_{i+1}-t_i)^{-l} \times I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^l U(t,i). \tag{4.10}$$

E. Application to Zhou, et al.

Here we show how to compute the integrals in (4.2). This can be reduced to simple matrix algebra when applying Qin's matrix representation and the formulas developed in (4.5),(4.9),and(4.10), since

$$\begin{aligned}
&\int_{-\infty}^{\infty} b''(t) b''(t)^{\mathrm{T}} \mathrm{d}t \\
&= \int_{-\infty}^{\infty} \sum_{i=k-1}^{n-k} \sum_{j=k-1}^{n-k} \frac{\mathbb{1}_{[t_i,t_{i+1})} \mathbb{1}_{[t_j,t_{j+1})}}{(t_{i+1}-t_i)^4} \\
&\qquad \times \left[I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^2 U(t,i) \right. \\
&\qquad \left. \times U(t,j)^{\mathrm{T}} (D^2)^{\mathrm{T}} M_k(j) I_k(j) \right] \mathrm{d}t \\
&= \int_{-\infty}^{\infty} \sum_{i=k-1}^{n-k} \frac{\mathbb{1}_{[t_i,t_{i+1})}}{(t_{i+1}-t_i)^4} \\
&\qquad \times I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^2 U(t,i) \\
&\qquad \times U(t,i)^{\mathrm{T}} (D^2)^{\mathrm{T}} M_k(i) I_k(i) \mathrm{d}t
\end{aligned}$$

$$= \sum_{i=k-1}^{n-k} \int_{t_i}^{t_{i+1}} (t_{i+1} - t_i)^{-4}$$

$$\times I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^2 U(t,i)$$

$$\times U(t,i)^{\mathrm{T}} (D^2)^{\mathrm{T}} M_k(i) I_k(i) \mathrm{d}t$$

$$= \sum_{i=k-1}^{n-k} (t_{i+1} - t_i)^{-3} I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^2$$

$$\times \left(\int_0^1 U U^{\mathrm{T}} \mathrm{d}t \right) (D^2)^{\mathrm{T}} M_k(i) I_k(i)$$

$$= \sum_{i=k-1}^{n-k} (t_{i+1} - t_i)^{-3} I_k(i)^{\mathrm{T}} M_k(i)^{\mathrm{T}} D^2$$

$$\times \Delta (D^2)^{\mathrm{T}} M_k(i) I_k(i).$$

F. Example

We have written an R package entitled `orthogonalsplinebasis` that uses the matrix representation presented here to represent and manipulate the basis functions directly. The package is available from The Comprehensive R Archive Network (R Development Core Team 2009). We will use this package to present an example of the matrix representation. For this example consider a spline basis on the domain $(0, 5)$ with one internal knot equally spaced between the endpoints of the interval. This makes the complete set of knots (0, 0, 0, 0, 2.5, 5, 5, 5, 5) for the standard third degree splines. The figures for this example are collected in the appendix. Figure 10 shows the original basis functions, before any manipulations such as orthogonalization. The

basis functions after orthogonalization are shown in Figure 11. The derivatives for the three non-zero derivatives of the orthogonal basis function are shown in Figure 12. The definite integral of the spline basis functions is defined as

$$B(t) \;=\; \int_{-\infty}^{t} b(x)dx,$$

where $b(t)$ is a set of basis functions, and the integration is performed element-wise over the vector of functions. The $-\infty$ is for convenience, since we assume that the basis functions are zero outside of the interval $(t_{k-1},\ t_{n-(k-1)})$. The integral is also computed explicitly and shown for both the original basis functions and the orthogonalized basis functions in Figure 13.

The R package also supports basic spline fitting with a penalized least squares methods. The penalty term, $\mathcal{K}$, is a multiplier, λ, times the inner product of the second derivative of the spline curve, as in Zhou, et al. (2008).

$$\mathcal{K} \;=\; \lambda\mu^{\mathrm{T}}\psi\mu, \text{ and} \tag{4.11}$$

$$\psi \;=\; \begin{pmatrix} 84.0 & -154.0 & 178.8 & -137.3 & 74.9 \\ -154.0 & 290.3 & -351.9 & 269.2 & -134.1 \\ 178.8 & -351.9 & 484.6 & -471.9 & 301.3 \\ -137.3 & 269.2 & -471.9 & 788.8 & -749.0 \\ 74.9 & -134.1 & 301.3 & -749.0 & 833.3 \end{pmatrix}. \tag{4.12}$$

Here we are using the orthogonalized basis functions to estimate the curve. This penalty effectively penalizes smoothness of the curve, a higher penalty leads to a smoother curve. Figure 14 shows a generated example with fits for different values of the penalty parameter. The data was generated from an exponential curve with

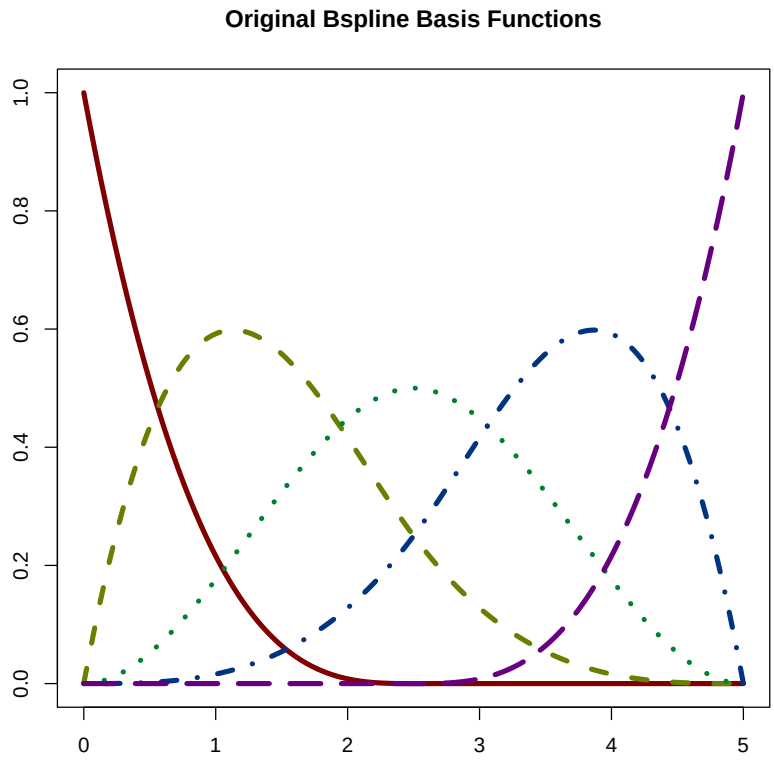

Fig. 10. The basis functions for a third degree spline curve with equally spaced knots on (0,5).

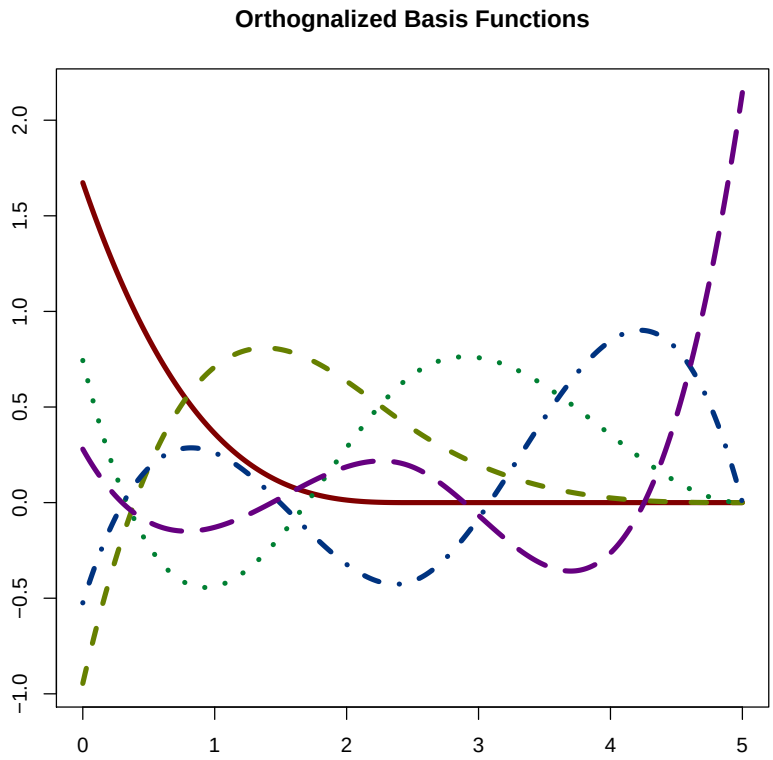

Fig. 11. The orthogonalized basis functions.

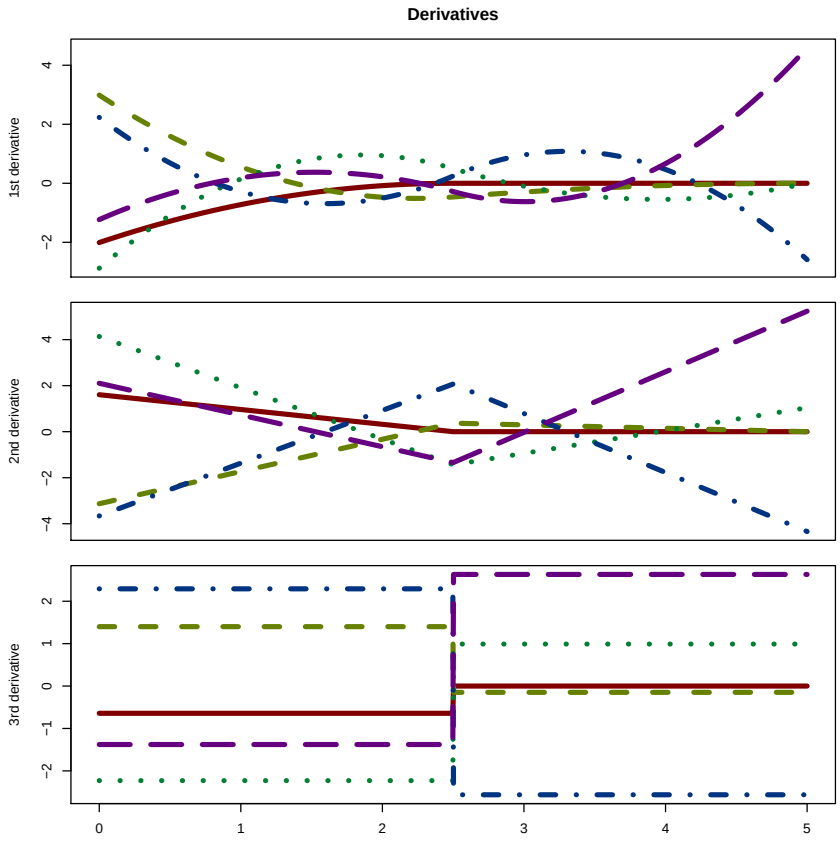

Fig. 12. The derivatives for the basis functions for the original basis functions.

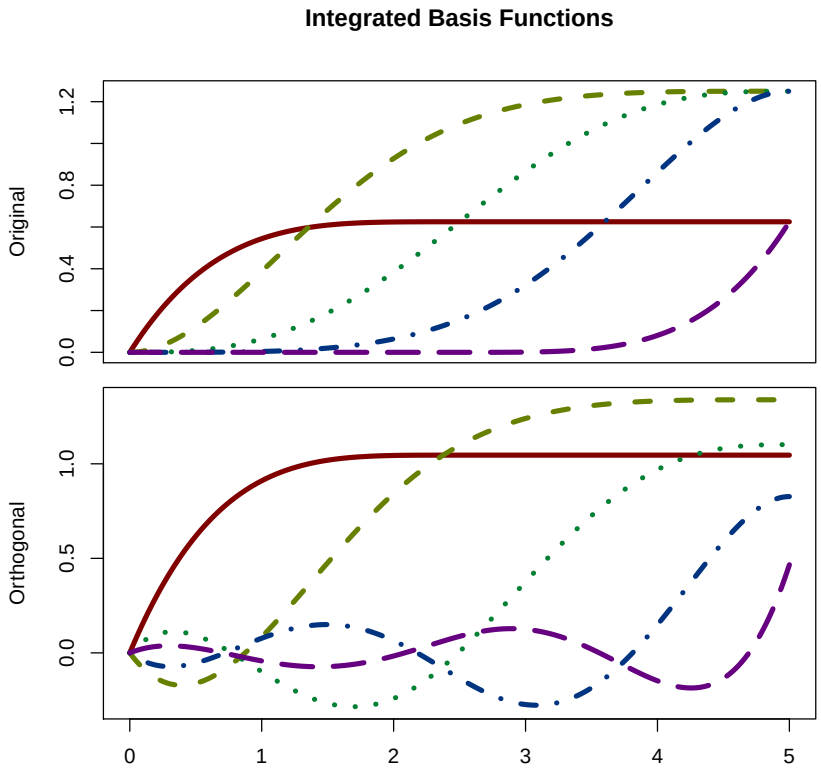

Fig. 13. The integrated basis functions for both the original and the orthogonalized basis functions.

normal errors and a standard deviation of five,

$$\begin{aligned} y &= \exp(x) + \epsilon \\ \epsilon &= \text{Normal}(0, \sigma = 5) \end{aligned}$$

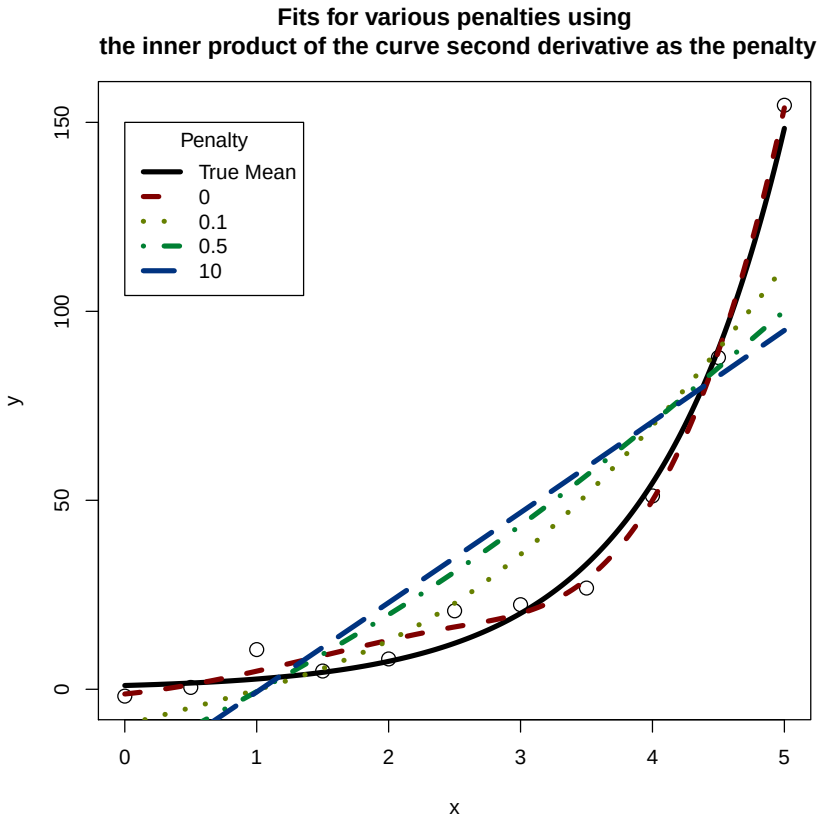

Fig. 14. The estimated curves for fits made with different values of the penalty parameter.

G. Discussion

By using Qin's representation (Qin 2000) many important formulas can be derived for the properties of the basis functions. This representation not only helps with

orthogonalization but also with derivatives and integration. Applying these formulas to the Zhou, et al. (2008) paper will help theoretically as well as computationally, as explicit formulas are available for any terms that involve the basis functions.

CHAPTER V

NPPTOR: R INTERACTION FOR NOTEPAD++

R (Ihaka & Gentleman, 1996) is a powerful programming language and analysis environment. Although interactions with R can be limited to single line commands entered into an interactive R session, this is certainly not the norm. To harness the full power of R, and it is far more common to write functions and blocks of code that span several lines and are intended to be evaluated together. To make these blocks and the scripts they form, an editor is needed. Since R scripts are just text, any text editor will get the job done, but the more features it has, and the more elegant it is, the more fun programming will be.

On Linux, users are forced to use an external text editor, such as Vim or Emacs. These have been modified to have advantages such as syntax highlighting and code folding. Both Vim and Emacs have methods for interacting with an active R session. R for Windows on the other hand comes with a built-in editor which is devoid of these features, but has the advantage of providing an easy way to evaluate code in R. The presence of this built-in editor discourages users from using the alternatives available. Some alternatives on the Windows system are Tinn-R (Grosjean, 2008) and Eclipse (Eclipse Foundation, 2010) with the StatET plugin (The StatET Team, 2010). This chapter introduces the NppToR utility that adds R interactions and syntax highlighting for Notepad++ (Notepad++ Team, 2010).

NppToR has been under development for over a year and has gained a serious user base without ever being promoted outside of the R mailing lists. This chapter gives an introduction and makes the case for the benefits of NppToR and Notepad++. Instructions are given as to how a user would use and configure NppToR to make it work best for them. NppToR can be downloaded from SourceForge (Sourceforge.net,

2010b).

A. R Interactions for Notepad++

Notepad++ is often cited as the most popular text editor for developers on Windows (see Pash, 2008; Gube, 2009). Notepad++ is fast and powerful, supports macros, auto-completion, syntax highlighting, code folding, and many other features, in a tabbed multi-document view. Just like Vim and Emacs are the most popular Linux editors, it makes sense to leverage Notepad++ as an editor for R on Windows.

Notepad++ supports a wide array of languages, and with the release of version 5.6 supports R as a built-in language. Prior to version 5.6 the only support for R was in the form of the user defined language system. One of the functions of NppToR is to generate the files for use with the user defined language system.

The primary purpose of NppToR is to add the interaction capabilities between R and Notepad++ that are present in the Windows R GUI and the other alternatives. The ideology behind NppToR is to enable this connection without painful altering of either R or Notepad++, and that configuring should be kept to a minimum. Because of this motivation NppToR is neither an R package nor a Notepad++ plugin. It is a small separate utility that enables evaluating code written in Notepad++ in R. One advantage of the approach of NppToR is it allows for connecting to a running R session rather than running an R session specifically for the interaction which is necessary with some other approaches.

B. Using NppToR

NppToR is a small utility that sits in the system tray and enables R interaction for Notepad++. Using NppToR occurs almost exclusively through the use of keyboard

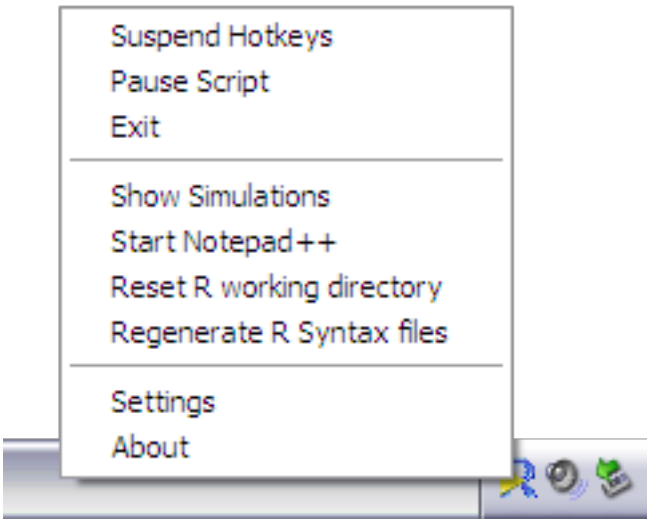

Fig. 15. NppToR sits as a utility in the system tray.

shortcuts. The shortcuts are completely configurable through the configuration dialogs accessed through the menu system from the system tray. Figure 15 shows the NppToR icon in the Windows system tray along with the menu accessed through a right click of the mouse.

NppToR has several ways to get your code into R for evaluation. The most commonly used is evaluating code line by line or by selection. This is done by default with the F8 key, which can be changed in the settings. Both pass selection and pass line are activated by the same shortcut, evaluating a selection if text is selected in Notepad++, otherwise evaluating the line of code the cursor is on and moving the cursor to the beginning of the next line to step rapidly through lines of code. Also available are passing an entire script and passing to the point in the file marked by the cursor, with `<control>+F8`, and `<shift>+F8` respectively. With any of these options NppToR seeks out a current running R console and pastes the code into the R GUI window. If NppToR cannot find a suitable R GUI Window it launches one, and pastes the code into that window.

If in addition to NppToR, R has the `rcom` package installed and loaded, NppToR can take advantage of this and enable the silent transfer option, with `<alt>+F8` as the default shortcut. The purpose of the silent transfer is to silently evaluate code that you do not want shown in the R GUI window. This is useful for such things as updating the definitions of large functions.

NppToR does require specific settings on the R GUI window to work correctly. Specifically the R GUI is required to run in the single document interface mode rather than the default multiple document interface mode. The multiple document interface is not needed when running NppToR, since Notepad++ will be used as the script editor rather than the built-in editor. To make this as easy on the user as possible NppToR makes use of a special 'RConsole' file that is used with every spawned R process which forces the single document interface.

NppToR spawned R processes are started in the same location as the script to make file referencing easier. It is typical that on Windows file names in R scripts are full path names. With NppToR this is much easier since file names can be made relative and the directory does not have to be changed every time an R session is started. For example if the script reads in a data file 'data.txt' and the data file is in the same directory as the script the file reference can simply be made as

Example 1 `data<-read.table("data.txt")`

rather than

Example 2 `data<-read.table("C:/path/to/data.txt")`

There is also a menu item on the system tray menu to change an R session working directory to the directory of the current script in Notepad++.

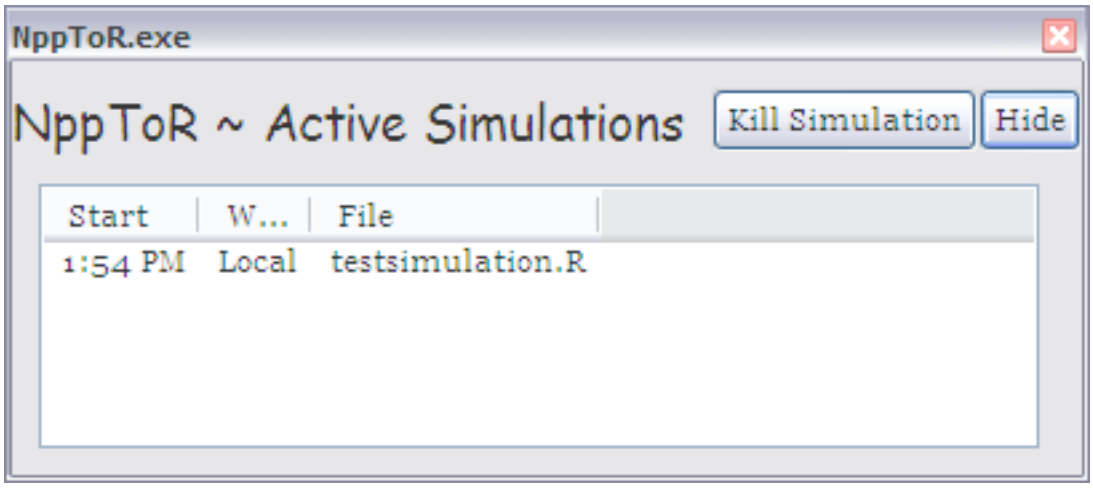

Fig. 16. NppToR monitors active simulations allowing for an easy way to kill off simulations that have been running too long.

The last method for R interaction is with batch processing. When doing simulations these often run for hours, if not days, and are typically run in R's batch mode. NppToR maps the shortcut `<control>+<alt>+F8` to running a script in batch mode. In addition it keeps track of running simulations through a special window, shown in Figure 16. This window easily allows for terminating simulations early. It also keeps track of how long they have been running. Double clicking on the filename will bring up a caption telling how long the simulation has been running for, to help the user determine if the script has gotten out of hand or not. The window can be hidden then called back through the system tray menu.

R has command like `edit` and `fix` to launch editors and scripts from inside R itself. The idea behind `fix` is that you can alter the definition of functions and objects in the workspace, with an editor. R monitors the editor and when closed updates the definition of the object. On windows this editor is the R built-in editor by default, but we would prefer to use Notepad++ for all the reasons already mentioned. Unfortunately, because of the multi-document interface of Notepad++ the mechanisms

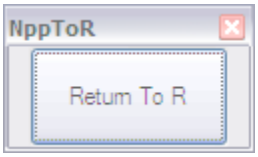

Fig. 17. The button that shows up when using Notepad++ to `fix` an R object.

for monitoring the external program don't work as expected. AutoHotKey, which the majority of NppToR is written in, provides a nice work around. Settings in the 'Rprofile' file that NppToR uses when spawning R processes, located in the NppToR install directory, specify the settings and work around to use Notepad++ as the R editor. Copying the contents of the file to the global R profile will make the setting of using Notepad++ as the editor universal, regardless of whether an R process was spawned by NppToR or not. When fixing or editing an item in R, The file is brought up in Notepad++ and an overlayed button, shown in Figure 17 is placed on the screen. When the user is finished editing the object clicking the button will return the object to R to update the definition. If the user wishes not to update the object just click the red close button instead. With two way interaction between Notepad++ and R, Notepad++ is a complete replacement for the R built-in editor.

C. Configuring NppToR

Changes to NppToR settings are made with the settings dialog, shown in Figure 18, accessible through the system tray menu. The program should work from install without any configuring on the part of the user but there are several circumstances that the user will want to change the settings, from the simplest of changing the keyboard shortcuts to fine tuning the performance.

The shortcuts for the various methods of passing code are fully and easily configurable. The keys are changed by typing in the code for the key. There is a key for

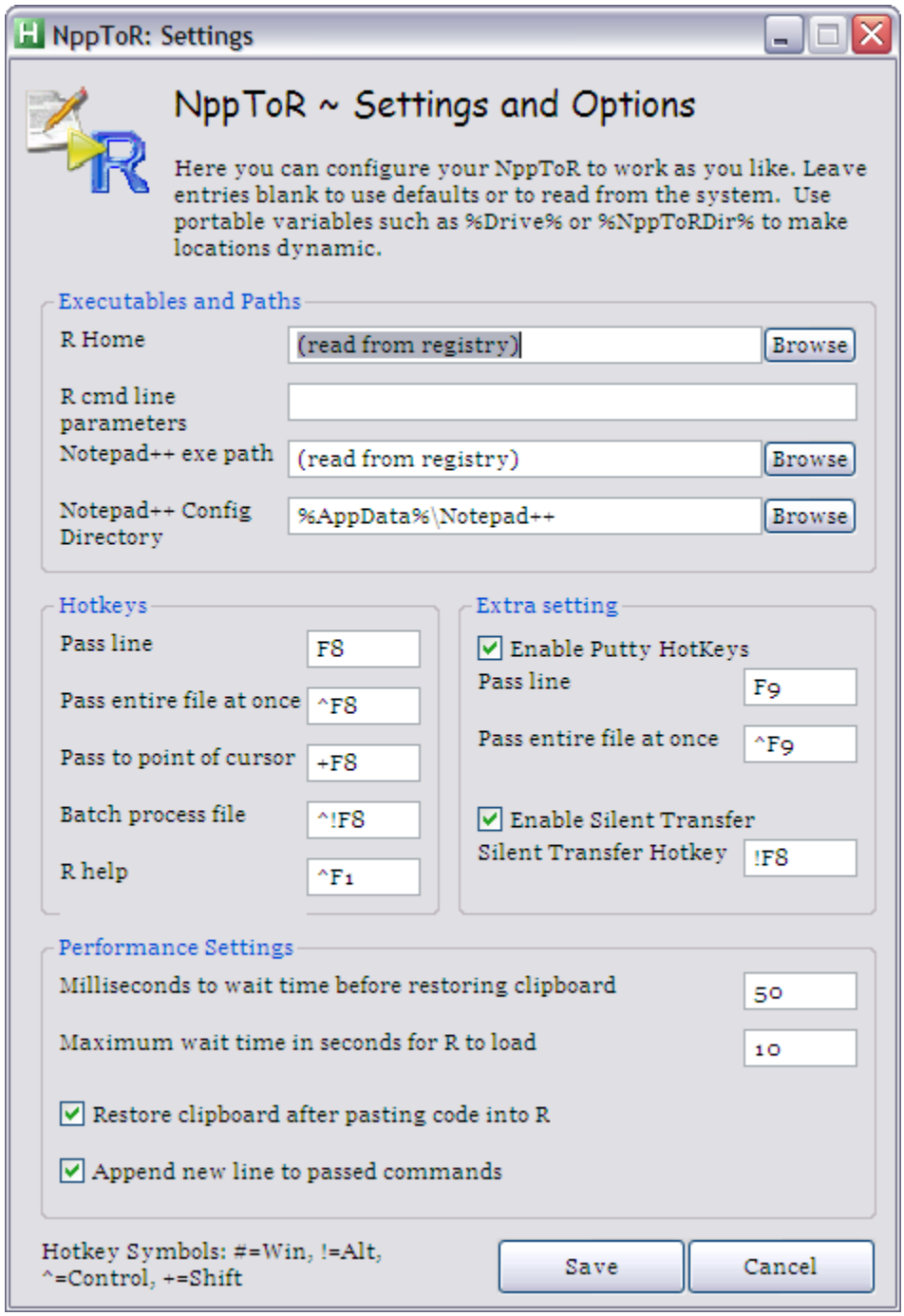

Fig. 18. NppToR offers all configurable settings in one central configuration dialog.

the codes on the bottom of the settings dialog. Simply prefix the shortcut keys with symbols for one or more modifier keys, for example the code for the default batch processing command is `^!F8` which reads as `<control>+<alt>+F8`, meaning the user holds down both the control and alt keys when pressing the F8 function key to send a script to R as a simulation batch file.

The default keyboard shortcuts are specifically chosen to not interfere with the main use of Notepad++. The keyboard shortcuts that are chosen for NppToR will override any function mapped to that key in Notepad++. Since NppToR is developed with the idea of not modifying the interfaced programs, Notepad++ has no awareness of the keyboard shortcuts employed by NppToR. This leaves the burden on the user that if they want to change the shortcuts and that those shortcuts then interfere with a Notepad++ function, that they must remap the Notepad++ keyboard shortcuts, which are also fully customizable.

The executable paths are the next most likely things that the user may want to configure. By default, the settings here are read from the registry for both the R directory and the Notepad++ directory, where both are stored in a standard installation of either. In non-standard installations, the directories may not be found in the registry and can be specified here. This is particularly the case if using portable versions of the programs. A portable program is one that leaves no traces of being run on the computer. Portable versions of both Notepad++ (Notepad++ Portable, 2009) and R (Redd, 2010) exist. NppToR supports the built-in Windows environment variables, denoted in percent signs , and adds two additional portable variables the `%drive%` and the `%npptordir%` pointing to the drive letter and the launch directory of NppToR respectively.

The command line parameters will be passed onto the R GUI. Run `Rgui --help` to see the options. For example, `--no-save -q` will turn off the message asking if

you want to save on exit and suppress the startup message.

The performance settings should usually not have to be tweaked unless NppToR is running on particularly slow computer. NppToR uses the clipboard to transfer code into R. If there happens to be anything on the clipboard NppToR saves the clipboard and then restores it after the code was pasted into R, if the 'Restore clipboard after pasting code into R' option is set, which it is by default. The 'Millisecond to wait before restoring clipboard' options gives the window of time that R has to paste the code from when NppToR issues the paste command until it wipes the clipboard of the code and restores the previous content. If this setting is too low the wrong thing will be passed into R, and if it is too high performance is inhibited in issuing multiple commands in rapid sequence, as often happens when evaluating through code. The default setting tries to reach a reasonable medium that should work for most modern computers, but may occasionally need to be adjusted.

The Maximum time to wait for R to load is a simple time limit to determine if a R process was successfully spawned before giving an error. In a properly configured system this setting should matter very little, and should only need to be increased if it takes longer than 10 seconds for R to load.

The 'Append new line to passed commands' only takes effect when passing selections, and not when passing line by line. The effect is that, when checked, any passed code is automatically evaluated. Unchecking this option allows for passing bits to the same line of code in R to construct a statement before evaluating it. The default is for this option to be turned on and is the behavior that is usually seen in other editors.

The PuTTy hotkey settings control the similar settings for interacting with a remote R session running through PuTTy which, will be discussed in the next section.

D. Interacting with R on a Server

NppToR not only allows for interaction with the R GUI on the current machine but also allows interaction with an active remote session that is run through the SSH client PuTTy. Often times users find that a server setup with R may run much faster than a setup on the users Windows box. If a user can use PuTTy to open an active R session on the remote machine NppToR can be used to facilitate passing code into PuTTy, and thus into the R session on the remote machine. Be warned that NppToR will not spawn remote processes, and that only passing by line or the entire file at one is supported at this time.

E. Generating Syntax Files

Even though the current version of Notepad++ has native rules for highlighting and code folding for R as a programming language, the rules generated by NppToR may be preferred. The native Notepad++ highlighting omits several reserved keywords, whereas the NppToR generated files includes these keywords as well as all the keywords from the base and recommended packages. NppToR includes a syntax generator utility, shown in Figure 19, that can integrate the keywords from a users library for the user installed packages.

Figure 20 shows a simple comparison and illustrates some omissions of the built-in syntax highlighter. The reserved keyword `TRUE` is omitted and the period is recognized as a word boundary, and while it does hold some special meaning in the class system it does not denote separate variable names and should not break highlighting as is shown here.

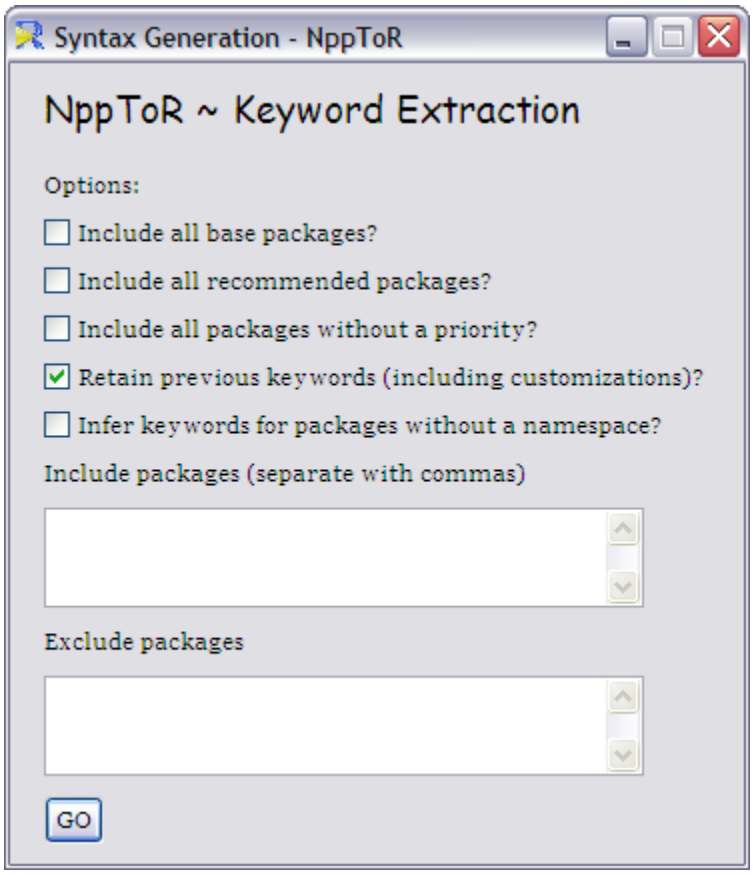

Fig. 19. NppToR dynamically generates syntax files based on the contents of the user's library.

```
1 id.by.time<-if(TRUE){
1 id.by.time<-if(TRUE){
```

Fig. 20. A comparison of the built-in syntax highlighting (top) to NppToR syntax highlighting (bottom).

F. Installing NppToR

A prerequisite of NppToR is to have installed Notepad++, which can be downloaded from the SourceForge download page (Sourceforge.net, 2010a). NppToR can be downloaded from its SourceForge page (Sourceforge.net, 2010b). NppToR has both an installable version and a version that can be ran without any install. The installed version copies the executables to the users application data folder. This makes the install user specific but removes the requirement that the install have administrator privileges. One major advantage of the install is that is adds a shortcut to the users, startup folder so that NppToR is always on. Since NppToR is a very small utility, it loads quickly and consumes nearly no resources when not in use. It will not interfere with any programs other than Notepad++. The non-install version is useful for portable setups, which was discussed in the section on settings. It is also an option for people who want to test NppToR.

G. Comparison with Alternatives

NppToR provides sharp contrast to the alternatives on Windows. In this comparison, I will exclude the options of Vim and Emacs for Windows, since they are natively Linux/Unix programs and have too foreign a feel for the majority of purely Windows users to be comfortable with. Eclipse is a large integrated development environment that organized large projects very well. This approach is the antithesis of NppToR. Tinn-R is similar to what is achieved with Notepad++ and NppToR, a text editor that can send commands to an R session. This approach though requires installation for both the program and in R packages. It is also a stand alone project that forked from the now defunct Tinn project, and so is left as an editor solely for R. Notepad++, being one of the most popular text editors on Windows as well as being open source

software like R, benefits from constant updates and a large pool of contributors and users to offer support, test bugs and provide extensions.

H. Summary

NppToR mirrors the advantages of editors on other operating systems for Windows, and does so in a native Windows approach. The power and strength of NppToR comes in its simplicity. It leverages an already powerful editor to become a superior editor for R. It provides all the functionality that a developer would want for R, and enhances the experience in using both Notepad++ and R. NppToR provides several avenues for interaction with R. Notepad++ also has the added advantage of supporting multiple languages for those who develop packages that may have code in C or FORTRAN and documentation in TEX. Notepad++ is a native Windows program and prescribes to conventional Windows standards, minimizing the learning curve. As users become proficient with the vast array of keyboard commands and the macro system the true power of Notepad++ is revealed.

CHAPTER VI

SUMMARY

In this dissertation we have discussed principal components in the functional framework. The additive bivariate hierarchical model, uses these principal components. In these models we consider data that is not only irregular but sparse, which enables these models to be applied to a wide array of data sets, not only those with regular and dense points. As we saw with the example of the AIDS data, these methods can be applied to a wide variety of problems from the health and medical fields.

The ability to apply the models on such structures do not come free, assumptions must be made that enable sharing of information across samples. Splines are used to reduce the dimensionality of the problem to allow it to be identifiable in the sparse data. From the papers in respected journals (James et al., 2000; Zhou et al., 2008), we see that the handling of the orthogonalization of the B-spline basis functions is not obvious. For this the `orthogonalsplinebasis` package for R is presented that handles all the necessary calculations and greatly simplifies the process of forming the matrices necessary for fitting the models.

Another cost of the flexibility of the models is the computational cost. The `pfda` package, give a high efficiency implementation that does the computations in reasonable time. Naive and pure R implementations fail, but a hybrid approach with both R and C code reaches the requirements for a usable package.

All the programming involved had the byproduct of improving programming in R on windows, in the form of NppToR. NppToR turns Notepad++ into the best R programming environment on windows. It adds R interactions, and creates the most dynamic syntax highlighting of any windows editor.

REFERENCES

Daniels, M. J., Dominici, F., Samet, J. M. & Zeger, S. L. (2000). Estimating particulate matter-mortality dose-response curves and threshold levels: an analysis of daily time-series for the 20 largest us cities. *American Journal of Epidemiology* **152**, 397–406.

Eclipse Foundation (2010). Eclipse.org home. `http://www.eclipse.org/`.

Grosjean, P. (2008). Tinn-r. `http://www.sciviews.org/Tinn-R/`.

Gube, J. (2009). The 15 most popular text editors for developers. `http://sixrevisions.com/web-development/the-15-most-popular-text-editors-for-developers/`.

Ihaka, R. & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**, 299–314.

James, G. M., Hastie, T. J. & Sugar, C. A. (2000). Principal component models for sparse functional data. *Biometrika* **87**, 587–602.

Lederman, M. M., Connick, E., Landay, A., Kuritzkes, D. R., Spritzler, J. & et al. (1998). Immunologic responses associated with 12 weeks of combination antiretroviral therapy consisting of zidovudine, lamivudine and ritonavir: results of aids clinical trials group protocol 315. *Journal of Infectious Diseases* **178**, 70–79.

Liang, H., Wu, H. & Carroll, R. J. (2003). The relationship between virologic responses in aids clinical research using mixed-effects varying-coefficient models with measurement error. *Biostatistics* **178**, 70–79.

Notepad++ Portable (2009). Notepad++ portable edition. http://sourceforge.net/projects/notepadpluspe/.

Notepad++ Team (2010). About Notepad++. http://notepad-plus.sourceforge.net/.

Pash, A. (2008). Best of the best: The hive five winners. http://lifehacker.com/5052582/best-of-the-best-the-hive-five-winners.

Redd, A. (2010). R portable. http://sourceforge.net/projects/rportable/.

Ruppert, D., Wand, M. P. & Carroll, R. J. (2003). *Semiparametric Regression*. Cambridge, UK: Cambridge University Press.

Sourceforge.net (2010a). Notepad++. http://sourceforge.net/projects/notepad-plus/.

Sourceforge.net (2010b). Npptor. http://sourceforge.net/projects/npptor/.

The StatET Team (2010). Eclipse plug-in for R: "StatET". http://www.walware.de/goto/statet.

Zhou, L., Huang, J. Z. & Carroll, R. J. (2008). Joint modelling of paired sparse functional data using principal components. *Biometrika* **95**, 601–619.

APPENDIX A

DETAILS OF THE EM ALGORITHM FOR THE ADDITIVE PRINCIPAL COMPONENT MODEL

Likelihood

In the forming of the likelihood we assume that the full data consists of the response, y_i, and the random effect principal component scores α_i and β_i for $i = 1, \ldots, n$. We assume that the random effects are missing and employ an EM algorithm to maximize the likelihood. Since the method uses splines to estimate the curves, a penalty is added to the log likelihood that effectively penalizes the smoothness of the curves. This penalty is formed in terms of the second derivative of the curve.

Let $\Omega = (\theta_Z, \theta_T, \theta_X, \Theta_f, \Theta_g, D_\alpha, D_\beta, \Lambda, \sigma)$ be the set of parameters for the model. The log likelihood for the full data decomposes into

$$\begin{aligned} \log\{\mathcal{L}(y, \alpha, \beta|\Omega)\} &= \sum_{i=1}^{n} \left[\log\left\{\mathcal{L}\left(Y_i|\alpha_i, \beta_i, \Omega\right)\right\} + \log\left\{\mathcal{L}\left(\alpha, \beta|\Omega\right)\right\}\right] \\ \log\left\{\mathcal{L}\left(Y_i|\alpha_i, \beta_i, \Omega\right)\right\} &= -(m_i/2)\log(\sigma^2) - (2\sigma^2)^{-1} \times \\ &\qquad \left\|Y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{X,i}\theta_X - B_{T,i}\Theta_f\alpha_i - B_{X,i}\Theta_g\beta_i\right\|^2. \end{aligned}$$

Where $B_{T,i}$ and $B_{X,i}$ are the matrices resulting from evaluating the points t_i and x_i for the B-spline basis functions $b_T(t)$ and $b_X(x)$ respectively.

The relationship between α and β are assumed to be jointly normal with correlation, but the components of α to be independent of other components of α and the same for components of β. This is more easily structured as a regression model between β and α. The choice of which variable to regress on the other is arbitrary

but changes the interpretation of Λ.

$$\begin{aligned} \beta_i|\alpha_i &= \Lambda\alpha_i + \eta_i, \\ \eta_i &= \text{Normal}(0, \Sigma_e ta), \\ \Sigma_\eta &= D_\beta - \Lambda D_\alpha \Lambda^{\mathrm{T}}, \end{aligned}$$

So that the log likelihoods come out to be

$$\begin{aligned} \log\left\{\mathcal{L}\left(\alpha, \beta|\Omega\right)\right\} &= \log\left\{\mathcal{L}\left(\beta|\alpha, \Omega\right)\right\} + \log\left\{\mathcal{L}\left(\alpha|\Omega\right)\right\} \\ \log\left\{\mathcal{L}\left(\beta|\alpha, \Omega\right)\right\} &= -(1/2)\log|\Sigma_\eta| - (1/2)\beta_i^{\mathrm{T}}\Sigma_\eta^{-1}\beta_i, \\ \log\left\{\mathcal{L}\left(\alpha|\Omega\right)\right\} &= -(1/2)\log|D_\alpha| - (1/2)\alpha_i^{\mathrm{T}}D_\alpha^{-1}\alpha_i, \end{aligned}$$

where D_α and D_β denote diagonal matrices with elements of descending order.

Penalized Likelihood

The likelihood above is not maximized to fit the model, as this may be over parameterized in some cases. To control the smoothness of the nonparametric curves involved penalties are added to the likelihood. The penalties that are included derive from directly penalizing the smoothness of the curve. For this case the measure of smoothness is defined as the integral of the squared second derivative of the curve. This penalty

$$K_t = \int_{t_{(1)}}^{t_{(N)}} b^{(2)}(t) b^{(2)}(t)^{\mathrm{T}} \mathrm{d}t \tag{A.1}$$

is the penalty matrix that is derived from the outer product of the second derivative of the basis functions for the T variable, for use with the T domain curves. Thanks to the

use of splines the integrated second derivative of the mean curve for T, $\mu_T(t) = b(t)^{\mathrm{T}}\theta_T$ is

$$\int_{t_{(1)}}^{t_{(N)}} \mu_T(t)^{\mathrm{T}}\mu_t(t)\mathrm{d}t \;=\; \theta_T^{\mathrm{T}} K_t \theta_T.$$

The formulas for the integrals for the other curves involved are formed in the same manner. This yields the form the penalties take. The penalized log likelihood is

$$\log\{\mathcal{L}(Y,\alpha,\beta)\} + \lambda_T\theta_T^{\mathrm{T}}K_t\theta_T + \lambda_X\theta_X^{\mathrm{T}}K_x\theta_X + \lambda_f \mathtt{1}^{\mathrm{T}}\Theta_f^{\mathrm{T}}K_t\Theta_f\mathtt{1} + \lambda_g\mathtt{1}^{\mathrm{T}}\Theta_g^{\mathrm{T}}K_x\Theta_g\mathtt{1}$$

where $\mathtt{1}$ denotes a vector of ones of appropriate length.

Marginal Likelihood

Since the principal component scores α and β are missing, the marginal log likelihood of Y is important to understand.

$$\begin{aligned}
\log\{\mathcal{L}(y|\Omega)\} &= \sum_{i=1}^{n} \log\{\mathcal{L}(y_i|\Omega)\} \\
\Sigma_i &= B_{T,i}\Theta_f D_\alpha \Theta_f^{\mathrm{T}} B_{T,i}^{\mathrm{T}} + B_{X,i}\Theta_g D_\beta \Theta_g^{\mathrm{T}} B_{X,i}^{\mathrm{T}} + \\
&\quad B_{T,i}\Theta_f D_\alpha \Lambda^{\mathrm{T}} \Theta_g^{\mathrm{T}} B_{X,i}^{\mathrm{T}} + B_{X,i}\Theta_g \Lambda D_\alpha \Theta_f^{\mathrm{T}} B_{T,i}^{\mathrm{T}} + \sigma^2 I_{m_i} \\
r_i &= y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{X,i}\theta_X \\
\log\{\mathcal{L}(y_i|\Omega)\} &= -(1/2)\log|\Sigma_i| - (1/2) r_i^{\mathrm{T}}\Sigma_i^{-1} r_i
\end{aligned}$$

AIC

The Akaike's information criteria is used for determining both the number of principal components, and the optimal penalty parameters. It is computed by adding a complexity penalty to the marginal likelihood of Y given the parameters of the model

Ω_{model}. The AIC is approximated by

$$\begin{aligned}
AIC_{\text{model}} &= -2\log\{\mathcal{L}(y|\Omega)\} + 2 * (\text{df}_\text{T} + \text{df}_\text{X} + \text{K}_\alpha\text{df}_\text{f} + \text{K}_\beta\text{df}_\text{g}) \\
\text{df}_\text{T} &= \text{trace}\left\{ \left(\sum_{i=1}^{n} B_{T,i}^\text{T} B_{T,i} + \lambda_T K_t \right)^{-1} \left(\sum_{i=1}^{n} B_{T,i}^\text{T} B_{T,i} \right) \right\} \\
\text{df}_\text{X} &= \text{trace}\left\{ \left(\sum_{i=1}^{n} B_{X,i}^\text{T} B_{X,i} + \lambda_X K_x \right)^{-1} \left(\sum_{i=1}^{n} B_{X,i}^\text{T} B_{X,i} \right) \right\} \\
\text{df}_\text{f} &= \text{trace}\left\{ \left(\sum_{i=1}^{n} B_{T,i}^\text{T} B_{T,i} + \lambda_f K_t \right)^{-1} \left(\sum_{i=1}^{n} B_{T,i}^\text{T} B_{T,i} \right) \right\} \\
\text{df}_\text{g} &= \text{trace}\left\{ \left(\sum_{i=1}^{n} B_{X,i}^\text{T} B_{X,i} + \lambda_g K_t \right)^{-1} \left(\sum_{i=1}^{n} B_{X,i}^\text{T} B_{X,i} \right) \right\}
\end{aligned}$$

As is discussed in Section D, this formula only approximates the AIC, due to inaccuracies inherent in the estimation of the degrees of freedom, particularly for the degrees of freedom associated with Θ_f and Θ_g.

E-Step of the EM algorithm

The E step of the EM algorithm consists of updating the distribution of the missing principal component scores, α and β, given the data, y and the current estimates of the parameters Ω_{curr}. The distribution is assumed to be jointly Normal, and the

updated distribution can be given as

$$\begin{aligned}
\begin{pmatrix}\alpha_i\\ \beta_i\end{pmatrix} |\Omega_{\text{curr}}, y_i &\sim \text{Normal}\left\{\begin{pmatrix}\mu_i\\ \nu_i\end{pmatrix}, \begin{pmatrix}\Sigma_{\alpha\alpha,i} & \Sigma_{\alpha\beta,i}\\ \Sigma^{\text{T}}_{\alpha\beta,i} & \sigma_{\beta\beta,i}\end{pmatrix}\right\},\\
\begin{pmatrix}\mu_i\\ \nu_i\end{pmatrix} &= \begin{pmatrix}\Sigma_{\alpha\alpha,i} & \Sigma_{\alpha\beta,i}\\ \Sigma^{\text{T}}_{\alpha\beta,i} & \sigma_{\beta\beta,i}\end{pmatrix}\begin{pmatrix}\Theta_f^{\text{T}} B_{T,i}^{\text{T}} R_i^{\text{T}}/\sigma^2\\ \Theta_g^{\text{T}} B_{X,i}^{\text{T}} R_i^{\text{T}}/\sigma^2\end{pmatrix},\\
R_i &= y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{X,i}\theta_X,\\
\begin{pmatrix}\Sigma_{\alpha\alpha,i} & \Sigma_{\alpha\beta,i}\\ \Sigma^{\text{T}}_{\alpha\beta,i} & \sigma_{\beta\beta,i}\end{pmatrix} &= \begin{pmatrix}\Delta_\alpha + \Theta_f B_{T,i}^{\text{T}} B_{T,i}\Theta_f/\sigma^2 & \chi + \Theta_f^{\text{T}} B_{T,i}^{\text{T}} B_{X,i}\Theta_g/\sigma^2\\ \chi^{\text{T}} + \Theta_g^{\text{T}} B_{X,i} B_{T,i}\Theta_f/\sigma^2 & \Delta_\beta + \Theta_g^{\text{T}} B_{X,i}^{\text{T}} B_{X,i}\Theta_g\end{pmatrix},\\
\begin{pmatrix}\Delta_\alpha & \chi\\ \chi^{\text{T}} & \Delta_\beta\end{pmatrix} &= \begin{pmatrix}D_\alpha & C\\ C^{\text{T}} & D_\beta\end{pmatrix}^{-1}
\end{aligned}$$

From here the needed moments used in the M-step are straight forward to compute.

M-Step of the EM algorithm

The algorithm for fitting the additive model presented in this paper, takes the form of an EM algorithm. The missing portion of the data is the principal component scores, α and β, which are assumed to have a joint Gaussian distribution. Each of the marginal distributions for α and β are assumed to be uncorrelated Gaussian distributions.

The steps of the maximization portion of the algorithm are given in the following paragraphs, and reflect the order in the implementation. While estimating the components of the model all other parameters are held constant at their current estimates.

1. Additive unpenalized variables are the simplest to estimate. The estimates

are obtained by standard regression on the residuals.

$$\widehat{\theta_Z} = \left(Z^{\mathrm{T}}Z\right)^{-1}\left\{\sum_{i=1}^{n} Z_i\left(y_i - B_{T,i}\theta_T - B_{X,i}\theta_X - B_{T,i}\Theta_f\alpha_i - B_{X,i}\Theta_g\beta_i\right)\right\}$$

2. Second, estimate the mean curves for both variables. For the T variable, in terms of the specified penalty parameter λ_T is

$$\begin{aligned}\widehat{\theta_T} = & \left\{\sum_{i=1}^{n}\left(B_{T,i}^{\mathrm{T}}B_{T,i} + \sigma^2\lambda_T\mathcal{K}_T\right)\right\}^{-1} \\ & \sum_{i=1}^{n} B_{T,i}^{\mathrm{T}}\left(y_i - Z_i\theta_Z - B_{X,i}\theta_X - B_{T,i}\Theta_f E(\alpha_i|y_i,\Omega_{\mathrm{curr}}) - \right. \\ & \left. B_{X,i}\Theta_g E(\beta_i|y_i,\Omega_{\mathrm{curr}})\right),\end{aligned}$$

where $\mathcal{K}_T$is the penalty matrix defined in equation A.1 The equation is analogous for the estimating equation of θ_X;

$$\begin{aligned}\widehat{\theta_X} = & \left\{\sum_{i=1}^{n}\left(B_{X,i}^{\mathrm{T}}B_{X,i} + \sigma^2\lambda_X\mathcal{K}_X\right)\right\}^{-1} \\ & \sum_{i=1}^{n} B_{X,i}^{\mathrm{T}}\left(y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{T,i}\Theta_f E(\alpha_i|y_i,\Omega_{\mathrm{curr}}) - \right. \\ & \left. B_{X,i}\Theta_g E(\beta_i|y_i,\Omega_{\mathrm{curr}})\right).\end{aligned}$$

3. The estimation of the principal component curves is more complicated than any of the other estimations, due to the relationship between the two sets of principal components and the correlation between the scores. To estimate the curves each curve must be estimated individually. Let $\Theta_f = (\theta_{f1},\ldots,\theta_{fK_T})$, and θ_{fj} is the j^{th} column of Θ_f and the coefficients for the j^{th} principal component curve. Also let $\Theta_{f\setminus j}$ be the matrix of Θ_f with the j^{th} column removed, and let $\alpha_{i,\setminus j}$ the vector α_i with the j^{th}

element removed.

$$\begin{aligned}
\widehat{\theta_{fj}} &= \left[\sum_{i=1}^{n} E(\alpha_{i,j}^2|\Omega_{\text{curr}}, y_i) B_{T,i}^{\mathrm{T}} B_{T,i} + \sigma^2 \lambda_f \mathcal{K}_T\right]^{-1} \\
&\quad \sum_{i=1}^{n} B_{T,i} \left\{R_i E(\alpha_{i,j}|\Omega_{\text{curr}}, y_i) - B_{X,i}\Theta_g E(\alpha_{i,j}, \beta_i|\Omega_{\text{curr}}, y_i) - \right. \\
&\quad \left. B_{T,i}\Theta_{f\setminus j} E(\alpha_{i,j}\alpha_{i,\setminus j}|\Omega_{\text{curr}}, y_i)\right\} \\
R_i &= y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{X,i}\theta_X
\end{aligned}$$

There is an analogous formula for updating the estimates of a column j of Θ_g.

$$\begin{aligned}
\widehat{\theta_{gj}} &= \left[\sum_{i=1}^{n} E(\beta_{i,j}^2|\Omega_{\text{curr}}, y_i) B_{X,i}^{\mathrm{T}} B_{X,i} + \sigma^2 \lambda_g \mathcal{K}_X\right]^{-1} \\
&\quad \sum_{i=1}^{n} B_{X,i} \left\{R_i E(\beta_{i,j}|\Omega_{\text{curr}}, y_i) - B_{T,i}\Theta_f E(\beta_{i,j}, \alpha_i|\Omega_{\text{curr}}, y_i) - \right. \\
&\quad \left. B_{X,i}\Theta_{g\setminus j} E(\beta_{i,j}\beta_{i,\setminus j}|\Omega_{\text{curr}}, y_i)\right\} \\
R_i &= y_i - Z_i\theta_Z - B_{T,i}\theta_T - B_{X,i}\theta_X
\end{aligned}$$

Each column is also orthogonalized, using a Gram-Schmidt process, to those columns that have already been updated. Failure to orthogonalize will result in biased results for the variances in the next step.

4. The last step is to estimate the variance components of the model, σ^2, D_α, and D_β. In estimating these variances the estimates of Θ_f and Θ_g are updated to force the orthogonality constraints on them. To estimate the overall variance, σ^2,

$$\widehat{\sigma}^2 = M^{-1} \sum_{i=1}^{n} \sum_{j=1}^{m_i} (y_{ij} - Z_{ij}\theta_Z - B_{T,ij}\theta_T - B_{X,ij}\theta_X - B_{T,ij}\Theta_f\alpha_i - B_{X,ij}\Theta_g\beta_i)^2$$

The variances for the principal components are estimated through an Eigenvalue

decomposition.

$$\begin{aligned}
\Theta_{f,\text{new}} D_\alpha \Theta_{f,\text{new}}^{\mathrm{T}} &= \Theta_{f,\text{old}} \left(\frac{1}{n} \sum_{i=1}^{n} E(\alpha_i \alpha_i^{\mathrm{T}} | \Omega_{\text{curr}}, y_i) \right) \Theta_{f,\text{old}}^{\mathrm{T}} \\
\Theta_{g,\text{new}} D_\beta \Theta_{g,\text{new}}^{\mathrm{T}} &= \Theta_{g,\text{old}} \left(\frac{1}{n} \sum_{i=1}^{n} E(\beta_i \beta_i^{\mathrm{T}} | \Omega_{\text{curr}}, y_i) \right) \Theta_{g,\text{old}}^{\mathrm{T}}
\end{aligned}$$

The regression coefficient between the principal components is estimated with

$$\widehat{\lambda} = \left\{ \sum_{i=1}^{n} E(\alpha_i \beta_i | \Omega_{\text{curr}}, y_i) \right\}^{\mathrm{T}} \left\{ \sum_{i=1}^{n} E(\alpha_i \alpha_i | \Omega_{\text{curr}}, y_i) \right\}^{-1}$$

VITA

Name: Andrew Middleton Redd

Address: Department of Statistics
Texas A&M University
3141 TAMU
College Station, TX 77843-3141

Email Address: aredd@stat.tamu.edu

Education: B.S., Applied Mathematics, Weber State University, 2005
M.S., Statistics, Texas A&M University, 2008
Ph.D., Statistics, Texas A&M University, 2010