DOUBLY STOCHASTIC MODEL WITH COVARIATES FOR REPLICATED POISSON POINT PROCESSES

by

Shenyan Pan

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Mathematics

at

The University of Wisconsin–Milwaukee

August 2025

ABSTRACT

DOUBLY STOCHASTIC MODEL WITH COVARIATES FOR REPLICATED POISSON POINT PROCESSES

by
Shenyan Pan

The University of Wisconsin–Milwaukee, 2025
Under the Supervision of Professor Daniel Gervini

Poisson point processes (PPPs) are powerful tools for modeling random point occurrences in multidimensional spaces, with applications across various fields. Although the traditional literature has focused on single realizations, replicated point processes are becoming increasingly common due to the growing availability of complex data. This dissertation develops a doubly stochastic model for replicated PPPs that incorporates covariates, extending latent component models to capture external effects. The proposed model expresses the log-intensity function as the sum of a mean function and latent component scores that vary with covariates. To ensure identifiability, component scores are constrained to be zero-mean and uncorrelated via centering and orthogonality. Parameter estimation is performed using penalized maximum likelihood, employing Newton–Raphson updates and the Laplace approximation for conditional distributions. Simulation studies assess the model's stability across various covariate structures (linear and nonlinear), baseline rates, and sample sizes. The results demonstrate decreasing error with increasing sample size, confirming the estimators' consistency. The model is applied to real data from the Divvy bicycle-sharing system in Chicago, analyzing daily usage at a representative station. The results reveal a nonlinear relationship between temperature and ridership, with peak usage occurring at moderate temperatures and declines observed under extreme heat or cold. This modeling framework improves the interpretability and predictive accuracy of PPPs with covariates, offering practical insights for applications such as fleet allocation in bicycle-sharing systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am deeply grateful to Professor Daniel Gervini, my major advisor, for his patient guidance, steadfast motivation, and invaluable feedback throughout my work. His mentorship has played a crucial role in both my academic growth and career development.

I also want to thank Professor Vytaras Brazauskas, Professor David Spade, Professor Lei Wang, and Professor Chao Zhu for their thoughtful comments and suggestions as members of my dissertation committee. Their contributions significantly enhanced my work and made my defense both a memorable and rewarding experience.

Special thanks go to Professor Jeb Willenbring and Professor Rebecca Bourn, the graduate program advisors, for their ongoing support, encouragement, and guidance throughout my doctoral program.

Lastly, I express my heartfelt appreciation to my family for their unwavering support. Their confidence in me has been a constant source of strength, especially during challenging times.

1 INTRODUCTION

Poisson point processes (PPPs) are versatile theoretical models for applications involving the random occurrence of points in multidimensional spaces, where both the number of points and their locations are treated as random variables. PPPs in time and space have broad applications across fields such as neuroscience, ecology, astronomy, and seismology. Examples are provided in classic textbooks such as Cox and Isham (1980), Diggle (2013), Møller and Waagepetersen (2003), Streit (2010), and Snyder and Miller (2012), as well as in the papers cited below.

However, the PPPs literature has primarily focused on single-realization cases, such as the distribution of trees in a single forest (Jalilian et al., 2013) or cells in a single tissue sample (Diggle et al., 2006). Situations with multiple replications of a process are increasingly common, driven by the availability of complex data, making replicated point processes more prevalent. This area remains relatively underexplored in the literature. Relevant works include Diggle et al. (1991), Baddeley et al. (1993), Diggle et al. (2000), Bell and Grunwald (2004), Landau et al. (2004), Wager et al. (2004), and Pawlas (2011). These studies, however, focus on estimators for summary statistics rather than intensity functions, which can provide more detailed insights.

When multiple replications are available, intensity functions can be estimated by borrowing strength across replications. For instance, Wu et al. (2013) propose estimators for the mean and principal components of independent and identically distributed realizations of a temporal doubly stochastic process, using kernel estimators of covariance functions. Gervini (2016) introduces an additive independent component model that unifies temporal and spatial cases and extends easily to regression and multivariate settings. Gervini and Baur (2020) further extend this method to marked point processes, offering a robust framework for modeling complex spatio-temporal data.

Motivated by these studies, this dissertation develops a doubly stochastic model with

covariates for replicated Poisson point processes. The subsequent sections present a mathematical formulation that incorporates covariates to capture properties of the observed data. The model accounts for both a simple linear trend and a more general nonlinear trend in latent scores. We derive the penalized log-likelihood to balance model fit and complexity, and apply the Newton–Raphson method for efficient and stable estimation of the model parameters. Later sections validate the model using simulated data and apply it to real-world bicycle-sharing system data, demonstrating its ability to predict usage patterns and inform operational strategies. This work aims to provide a rigorous, data-driven framework to enhance the efficiency and sustainability of bicycle-sharing systems, improving their reliability and accessibility for urban users.

2 MODELING BICYCLE-SHARING SYSTEM

2.1 Introduction to Bicycle-sharing Systems

As cities worldwide seek sustainable transport options, bicycle-sharing systems have gained widespread adoption, driven by their low environmental impact and operational flexibility. These systems provide bicycles for short-term use, typically through a network of automated docking stations where users can borrow a bicycle from one station and return it to another within the same network. Available either for a fee or free of charge, bicycle-sharing systems cater to a wide range of users, supporting daily commutes, recreational activities, and connections to public transit systems. In recent years, such systems have gained widespread popularity in major urban centers, driven by increasing demand for sustainable transportation options and supportive city planning initiatives that prioritize green mobility (Shaheen et al., 2010).

The smooth operation of a bicycle-sharing system depends on maintaining consistent availability of both bicycles and docking spaces. When a user arrives at a station to find no bicycles available or reaches a destination with no empty docks, they must seek out alternative stations, which can lead to inconvenience and frustration. This often discourages users from relying on the system, reducing its overall effectiveness. A key challenge in these systems is the natural imbalance in bicycle flows: the movement of bicycles from one station to another is rarely matched by an equal return flow. For example, during morning commutes, users may ride from residential neighborhoods to business districts, resulting in a surplus of bicycles at downtown stations and a shortage elsewhere (Nair and Miller-Hooks, 2011). These spatial imbalances create operational challenges that must be addressed to ensure reliable service.

To manage these imbalances, bicycle-sharing systems employ both short-term and long-term strategies. In the short term, operators often use trucks or vans to manually redis-

tribute bicycles, typically during off-peak hours, to prepare the network for the next day's demand. This process involves moving bicycles from stations with excess supply to those with shortages, ensuring that users can access bicycles when and where they need them. Over the long term, effective planning of station locations and capacities is crucial. By strategically placing new docking stations in areas with high demand or connectivity to other transport modes, operators can minimize imbalances and enhance system efficiency. Both short-term redistribution and long-term planning rely on a deep understanding of bicycle usage patterns, which vary across time and space due to factors such as time of day, day of the week, weather conditions, and urban land use patterns. This discussion is supported by previous studies; see Bahadori et al. (2021), Lei et al. (2023), Seo et al. (2022)

As previous literature has shown, understanding the spatio-temporal patterns of bicycle demand is essential for optimizing fleet management and ensuring the success of bicycle-sharing systems. Demand for bicycles fluctuates significantly throughout the day and week. For instance, stations near office districts may experience high checkout rates during morning rush hours as commuters travel to work, while the same stations may see high return rates in the evening. Conversely, stations in recreational or residential areas may experience peak demand during weekends or evenings. External factors, such as temperature, precipitation, or proximity to transit hubs, further influence usage patterns. By analyzing these patterns, operators can predict when and where bicycles are needed most, enabling proactive redistribution and informed infrastructure planning. This data-driven approach not only improves user satisfaction but also enhances the integration of bicycle-sharing systems into broader urban transportation networks.

2.2 Divvy: Chicago's Bicycle-sharing System

The Divvy bicycle-sharing system in Chicago exemplifies the growing trend of urban bicycle-sharing programs, offering a network of automated docking stations where users

can borrow bicycles for short-term trips and return them to any station within the system. The Chicago Data Portal (https://data.cityofchicago.org) provides public access to comprehensive trip data, recording every bicycle checkout and return within the network. This dissertation analyzes trips from April 1 to November 30, 2016, a period chosen to capture peak bicycle usage, as demand significantly declines during Chicago's winter months due to cold weather and reduced outdoor activity. Over these 244 days, the Divvy system recorded 3,068,211 bicycle trips across 458 active stations, highlighting the system's extensive activity and the variability in demand across different locations in the city.

Each bicycle trip is represented as an observation (t, s) in a spatio-temporal process, where t denotes the trip's starting time and s indicates the origin station. The 244 days from April 1 to November 30 serve as $n = 244$ replications of this process, providing a robust dataset for statistical analysis of usage patterns. Bicycle demand can be modeled as a multivariate temporal point process, with each of the 458 stations corresponding to a dimension of the process (Gervini and Khanal, 2018). Demand exhibits significant spatial variation, driven by station location and surrounding urban context. For example, station 386 on the South Side recorded only 29 trips annually, reflecting low usage in a less dense area, while station 35 near Navy Pier, a popular tourist destination, recorded 85,314 trips, indicating high demand in a recreational hub. For stations with high daily trip counts, such as those in downtown or near tourist attractions, the temporal distribution of checkouts can be estimated using kernel smoothing or other density estimation methods, offering insights into daily usage patterns (Silverman, 1986).

Analysis of the Divvy data reveals pronounced spatio-temporal patterns in bicycle demand, reflecting the influence of both temporal and contextual factors. For example, studies of Station 166—located at the corner of Ashland and Wrightwood Avenues—identified a strong weekly periodicity, with demand peaking on Sundays and dipping on Wednesdays or Thursdays. A seasonal trend is also evident, with demand peaking during the

summer months, when warmer weather encourages outdoor activity, and declining toward late fall as temperatures drop (Gervini and Khanal, 2018). Additionally, distinct differences emerge between weekday and weekend usage: weekday trips often occur early in the morning with downtown destinations, suggesting commuter use, while weekend trips are more frequent in the afternoon and remain within local neighborhoods, indicating recreational use (Gervini, 2022). These patterns highlight the need for models that account for both temporal periodicity and spatial heterogeneity to better inform operational decisions.

This dissertation proposes a multivariate temporal point process model to characterize bicycle demand, treating checkout times at each station as random events. The 244 daily replications enable estimation of the intensity function, which varies across time, driven by covariates such as external factors like temperature. The model can also be applied to return times, providing a comprehensive framework for understanding both borrowing and returning behaviors. By incorporating covariates in latent scores, the model demonstrates its ability to predict usage patterns and support operational strategies, such as truck-based redistribution and strategic station placement, to enhance the Divvy system's efficiency and sustainability for Chicago's urban users.

2.3 Poisson Point Processes

A point process X is a random countable set in a space $\mathscr{S}$, where $\mathscr{S}$ is typically $\mathbb{R}$ for temporal processes or $\mathbb{R}^2$ for spatial processes (Møller and Waagepetersen, 2003, Chapter2; Streit, 2010, Chapter 2) . The process X is called locally finite if $\#(X \cap B) < \infty$ with probability one for any bounded set $B \subseteq \mathscr{S}$. In this case, we define the count function $N(B) = \#(X \cap B)$, which characterizes the process X. Let X be locally finite and define $X_B = (X \cap B)$. Given a locally integrable function $\lambda : \mathscr{S} \to [0, \infty]$ such that, for any bounded $B \subseteq \mathscr{S}$, the integral $\int_B \lambda$ is finite, we say that X is a Poisson process with intensity function λ, denoted by $X \sim \mathscr{P}(\lambda)$, if:

(i) $N(B)$ follows a Poisson distribution with rate $\int_B \lambda$, and

(ii) conditional on $N(B) = m$, the m points in $X \cap B$ are independent and identically distributed (i.i.d.) with density $\widetilde{\lambda} = \frac{\lambda}{\int_B \lambda}$.

For $X \sim \mathscr{P}(\lambda)$, the density function of X_B at $x_B = \{t_1, \ldots, t_m\}$ is:

$$\begin{aligned} f(x_B) &= f(m) f(t_1, ..., t_m | m) \\ &= \exp\left\{ - \int_B \lambda(t) dt \right\} \frac{\left\{ \int_B \lambda(t) dt \right\}^m}{m!} \times \prod_{j=1}^{m} \widetilde{\lambda}(t_j) \\ &= \exp\left\{ - \int_B \lambda(t) dt \right\} \frac{1}{m!} \prod_{j=1}^{m} \lambda(t_j) \end{aligned} \tag{2.1}$$

where the probability density function (pdf) for a set-valued point process X, as understood in the sense of Proposition 3.1 of Møller and Waagepetersen (2003), is defined with respect to the family of locally finite subsets of $\mathscr{S}$. Specifically, let $\mathscr{N} = \{A \subseteq \mathscr{S} : \#(A \cap B) < \infty \text{ for all bounded } B \subseteq \mathscr{S}\}$ denote the collection of all locally finite subsets of $\mathscr{S}$. For any subset $F \subseteq \mathscr{N}$, the probability measure of the point process X is defined as follows:

$$\begin{aligned} P(X_B \in F) &= \sum_{m=0}^{\infty} \int_B \cdots \int_B \mathbb{I}(\{t_1, \ldots, t_m\} \in F) f(\{t_1, \ldots, t_m\}) dt_1 \ldots dt_m \\ &= \sum_{m=0}^{\infty} \frac{\exp\{ - \int_B \lambda(t) dt \}}{m!} \int_B \cdots \int_B \mathbb{I}(\{t_1, \ldots, t_m\} \in F) \{ \prod_{j=1}^{m} \lambda(t_j) \} dt_1 \ldots dt_m \end{aligned} \tag{2.2}$$

2.4 Basic Latent Model

Individual realizations of point processes are often modeled as Poisson processes with fixed intensities λ. However, for replicated processes, a single intensity λ rarely provides an adequate fit across all replications. Instead, it is more appropriate to assume that the intensities vary from replication to replication. A convenient approach is to treat λ as a

random parameter, or latent random effect, leading to a doubly stochastic process (Møller and Waagepetersen, 2003, Ch.5; Streit, 2010, Ch.8). A doubly stochastic process is a pair (X, Λ), where $X \mid \Lambda = \lambda$ is a Poisson process with intensity λ, and Λ is a random function that takes values in the space $\mathscr{F}$ of non-negative, locally integrable functions on $\mathscr{S}$. The replications can be viewed as n independent and identically distributed (i.i.d.) realizations $(X_1, \Lambda_1), \ldots, (X_n, \Lambda_n)$ of the doubly stochastic process (X, Λ), where X_i is observable, but Λ_i is a latent process.

The latent process Λ is not explicitly measured, but it captures the distribution of the point process X. Gervini and Khanal (2018), and Gervini (2017), propose a model for Λ based on the Karhunen-Loève expansion for stochastic processes (Ash et al., 2014). The model is of the form:

$$\log \Lambda(t) = \mu(t) + \sum_{k=1}^{p} U_k \phi_k(t) \tag{2.3}$$

where $\mu \in L^2(B)$, $\phi_1, \ldots, \phi_p$ are orthonormal functions, and $U_1, \ldots, U_p$ are independent $\mathcal{N}(0, \sigma_k^2)$ random variables. We refer to the ϕ_ks as components and the U_ks as component scores. Model (2.3) translates into a multiplicative model for $\Lambda(t)$:

$$\Lambda(t) = \lambda_0(t) \prod_{k=1}^{p} (\exp \phi_k(t))^{U_k} \tag{2.4}$$

with $\lambda_0 = \exp \mu$ as the baseline intensity function. For estimation, the mean function μ and each component ϕ_k are modeled using spline functions for temporal processes, or radial Gaussian kernel functions for spatial processes.

3 THEORETICAL MODELS

3.1 The Covariates Model

In Model (2.3), the functional parameters μ and ϕ_k require estimation. We adopt a semiparametric approach, modeling them using basis functions $\beta_1, \ldots, \beta_q$, such as B-splines for temporal processes or radial Gaussian kernels for spatial processes. Specifically, we express $\mu(t) = c_0^{\mathbf{T}}\boldsymbol{\beta}(t)$ and $\phi_k(t) = c_k^{\mathbf{T}}\boldsymbol{\beta}(t)$, where $\boldsymbol{\beta}$ is the vector of basis functions β_k. Model (2.3) can then be expressed as:

$$\log \Lambda(t) = (c_o + \mathbf{CU})^{\mathbf{T}}\boldsymbol{\beta}(t) \tag{3.1}$$

where $\mathbf{C} = [c_1, \ldots, c_p]$ and $\mathbf{U} = (U_1, \ldots, U_p)^{\mathbf{T}}$. The orthonormality of the ϕ_ks is expressed as $\mathbf{C}^{\mathbf{T}}\mathbf{J}_0\mathbf{C} = \mathbf{I}$, where $\mathbf{J}_0 = \int \boldsymbol{\beta}\boldsymbol{\beta}^{\mathbf{T}}$. Gervini (2017) and Gervini and Khanal (2018) study different realizations of the component score $\mathbf{U}$ defined earlier, but they assume that the U_ks are independent random variables.

This dissertation extends Model (2.3) to accommodate covariates. We introduce an observation Z, such as average daily temperature in the bicycle-sharing system, alongside X. Then, our model has n replications of $(X_1, \Lambda_1, Z_1), \ldots, (X_n, \Lambda_n, Z_n)$, and we express the component scores $\mathbf{U}$ in terms of covariate Z for each component:

$$U_{ik} = g_k(Z_i) + \varepsilon_{ik}, i = 1, \ldots, n, k = 1, \ldots, p \tag{3.2}$$

for unknown functions g_ks and random errors ε_{ik}s.

We consider Z as univariate and the g_ks as either linear or nonlinear. The simple linear form is modeled as $g_k(z) = d_k z$, where d_k is a scalar. Alternatively, the nonlinear form is modeled with spline functions: $g_k(z) = d_k^{\mathbf{T}}\boldsymbol{\gamma}(z)$, where $\boldsymbol{\gamma}(z)$ is a vector of spline ba-

sis functions and d_k is a vector of coefficient. We assume the ε_ks follow $N(0, \sigma_k^2)$ and are independent of Z. The parameters c_0, c_k, and d_k, along with the σ_k^2s, are estimated by penalized maximum likelihood, conditionally on z, since no assumptions are made about the covariate's distribution.

3.2 Model Identifiability and Bias Correction

The proposed model (2.3) includes a baseline mean function $\mu(t)$ and component score U_k that introduce variability across observations. Identifiability in this context ensures that the model parameters—particularly $\mu(t)$—can be uniquely estimated. Specifically, we require that $\mathbb{E}[\log \Lambda(t)] = \mu(t)$, so that $\mu(t)$ corresponds to the expected log-intensity function. This condition is essential to avoid ambiguity in parameter estimation and to ensure that the estimator $\widehat{\mu}(t)$ accurately reflects the true mean. To achieve this, it is necessary to impose identifiability constraints on the model parameters. We assume that $\mathbb{E}(u_k) = 0$, ensuring that the component score does not systematically influence the mean intensity. The following subsections describe the theoretical requirements, practical adjustments, and implications of these identifiability conditions in data analysis.

3.2.1 Ensuring Zero-Mean Component Scores

The component score u_k is modeled as:

$$u_k = g_k(z) + \varepsilon_k, \tag{3.3}$$

where $g_k(z)$ is a function of the covariate z (e.g., temperature), capturing the systematic effect of z, and $\varepsilon_k \sim \mathcal{N}(0, \sigma_k^2)$ is a noise term introducing random effect, with $\mathbb{E}(\varepsilon_k) = 0$.

The expectation of u_k is:

$$\mathbb{E}(u_k) = \mathbb{E}[g_k(z)] + \mathbb{E}(\varepsilon_k) = \mathbb{E}[g_k(z)]. \tag{3.4}$$

To satisfy $\mathbb{E}(u_k) = 0$, we must ensure:

$$\mathbb{E}[g_k(z)] = 0. \tag{3.5}$$

This condition is pivotal, as any non-zero expectation of $g_k(z)$ introduces a systematic bias in $\mathbb{E}[\log \Lambda(t)]$, causing $\hat{\mu}(t)$ to deviate from the true $\mu(t)$. The noise term ε_k poses no issue, as its zero mean is guaranteed by the normal distribution. However, the function $g_k(z)$ requires careful design to satisfy the zero-expectation condition, since its expectation depends on the distribution of z and the form of g_k. The following subsections explore how to achieve $\mathbb{E}[g_k(z)] = 0$ for different forms of $g_k(z)$, with theoretical considerations and practical adjustments for both the simple linear model and more general cases.

Simple Linear Model

The simple linear model, defined as:

$$g(z) = a + b \cdot z, \tag{3.6}$$

represents a straightforward relationship between the covariate z and the random effect, where a is the intercept and b is the slope coefficient. To ensure $\mathbb{E}(g(z)) = 0$, we compute:

$$\mathbb{E}[g(z)] = \mathbb{E}[a + b \cdot z] = a + b\mathbb{E}(z). \tag{3.7}$$

This expectation is zero only if:

$$a + b\mathbb{E}(z) = 0. \tag{3.8}$$

This condition is satisfied by setting the intercept $a = 0$ and ensuring the covariate z is centered, i.e., $\mathbb{E}(z) = 0$. In the context of this dissertation, where z represents daily temperature, using the raw temperature T as z is problematic, as $\mathbb{E}(T) \neq 0$. Instead, we define the centered covariate:

$$z = T - \mathbb{E}(T). \tag{3.9}$$

For a sample of daily temperatures $T_1, T_2, \ldots, T_n$, we estimate $\mathbb{E}(T)$ using the sample mean:

$$\hat{\mu}_T = \frac{1}{n} \sum_{i=1}^{n} T_i, \tag{3.10}$$

and compute:

$$z_i = T_i - \hat{\mu}_T. \tag{3.11}$$

By setting $a = 0$, the model becomes $g(z) = b \cdot z$, and:

$$\mathbb{E}[g(z)] = b\mathbb{E}(z) = b \cdot 0 = 0, \tag{3.12}$$

ensuring $\mathbb{E}(u_k) = 0$. This centering process eliminates the systematic shift in the random effects, addressing the possible bias in $\hat{\mu}(t)$. Centering z aligns the covariate's distribution around zero, preventing the intercept from introducing a constant offset that would otherwise affect the model. This adjustment is straightforward to implement, requiring only the subtraction of the sample mean from each temperature observation.

General Model

The general model, defined as:

$$g_k(z) = d_k^{\mathbf{T}} \gamma(z), \tag{3.13}$$

where d_k is the coefficient vector, and $\gamma(z)$ is the vector of spline basis functions. The expectation is:

$$\mathbb{E}[g_k(z)] = \mathbb{E}[d_k^{\mathbf{T}} \gamma(z)] = d_k^{\mathbf{T}} \mathbb{E}[\gamma(z)]. \tag{3.14}$$

To satisfy $\mathbb{E}(g_k(z)) = 0$, we require:

$$d_k^{\mathbf{T}} \mathbb{E}[\gamma(z)] = 0, \tag{3.15}$$

which holds if:

$$\mathbb{E}[\gamma(z)] = 0. \tag{3.16}$$

This condition ensures that the basis functions, when averaged over the distribution of z, do not introduce a systematic bias. To achieve this, we define a transformed covariate:

$$\mathbf{W} = \gamma(z) - \mathbb{E}[\gamma(z)], \tag{3.17}$$

so that:

$$\mathbb{E}[\mathbf{W}] = \mathbb{E}[\gamma(z) - \mathbb{E}[\gamma(z)]] = 0. \tag{3.18}$$

The model then becomes:

$$g_k(z) = d_k^{\mathbf{T}} \mathbf{W}, \tag{3.19}$$

guaranteeing:

$$\mathbb{E}[g_k(z)] = d_k^{\mathbf{T}} \mathbb{E}[\mathbf{W}] = 0. \tag{3.20}$$

For a sample $z_1, z_2, \ldots, z_n$, we estimate $\mathbb{E}[\gamma(z)]$ using the sample mean:

$$\overline{\gamma(z)} = \frac{1}{n} \sum_{i=1}^{n} \gamma(z_i), \tag{3.21}$$

and compute:

$$\mathbf{W}_i = \gamma(z_i) - \overline{\gamma(z)}. \tag{3.22}$$

This transformation centers the feature vector $\gamma(z)$, analogous to centering z in the linear model, but applies to a vector of functions, accommodating more complex covariate relationships. This approach is theoretically significant, as it generalizes the centering principle to flexible model specifications, ensuring identifiability across a wide range of applications. Our simulations employ the general form but focus on the quadratic function only. It is included here to provide a comprehensive theoretical framework to facilitate future extensions of the model to more complex covariate effects.

3.2.2 Uncorrelated Component Scores

We enforce the uncorrelatedness of component scores to guarantee the identifiability of the component $\phi_k(t)$:

$$\mathbb{E}(u_k u_l) = 0 \quad \text{for} \quad k \neq l. \tag{3.23}$$

Given $u_k = g_k(z) + \varepsilon_k$ and $u_l = g_l(z) + \varepsilon_l$, with ε_k independent of each other and of z, we have $\mathbb{E}(\varepsilon_k \varepsilon_l) = 0$ for $k \neq l$. Thus, the correlation condition becomes:

$$\mathbb{E}(u_k u_l) = \mathbb{E}[(g_k(z) + \varepsilon_k)(g_l(z) + \varepsilon_l)] = \mathbb{E}[g_k(z) g_l(z)] + \mathbb{E}(\varepsilon_k \varepsilon_l) = \mathbb{E}[g_k(z) g_l(z)]. \tag{3.24}$$

To satisfy $\mathbb{E}(u_k u_l) = 0$, we require:

$$\mathbb{E}[g_k(z)g_l(z)] = 0 \quad \text{for} \quad k \neq l. \tag{3.25}$$

For the general model $g_k(z) = d_k^{\mathbf{T}}\mathbf{W}$, where $\mathbf{W} = \gamma(z) - \mathbb{E}[\gamma(z)]$, we compute:

$$\mathbb{E}[g_k(z)g_l(z)] = \mathbb{E}[d_k^{\mathbf{T}}\mathbf{W}(d_l^{\mathbf{T}}\mathbf{W})] = \mathbb{E}[d_k^{\mathbf{T}}\mathbf{W}\mathbf{W}^{\mathbf{T}}d_l] = d_k^{\mathbf{T}}\mathbb{E}[\mathbf{W}\mathbf{W}^{\mathbf{T}}]d_l. \tag{3.26}$$

This expectation is zero if:

$$d_k^{\mathbf{T}}\mathbb{E}[\mathbf{W}\mathbf{W}^{\mathbf{T}}]d_l = 0 \quad \text{for} \quad k \neq l, \tag{3.27}$$

Let:

$$\Xi = \mathbb{E}[\mathbf{W}\mathbf{W}^{\mathbf{T}}] \tag{3.28}$$

and the coefficients d_ks must satisfy the condition $d_k^{\mathbf{T}}\Xi d_l = 0$ for $k \neq l$. This orthogonality condition ensures that the coefficient vectors d_k and d_l are orthogonal with respect to the covariance structure of $\mathbf{W}$, preventing overlap between the component scores u_k and u_l. This is crucial for distinguishing the contributions of each $\phi_k(t)$ in the model, as correlated components would confound the estimation, leading to unreliable inferences.

In the simulations conducted, we assumed that the covariate z influences only a single component score u_k, and the algorithm is designed to update exclusively that component's parameters. Specifically, we pick a $g_k(z)$ function for u_k, assuming that other components u_l (for $l \neq k$) are not directly affected by z. To maintain uncorrelated random effects, the algorithm initializes the coefficient vectors with orthogonal values, ensuring that $\mathbb{E}[u_k u_l] \approx 0$ for $k \neq l$. This approach leverages the independence of noise terms ε_k and the single-component update to simplify the correlation structure, mitigating potential issues with non-orthogonality. By focusing updates on a single component, the

correlation between component scores remains negligible, aligning with the theoretical requirement for identifiability of $\phi_k(t)$. This simulation design, while not implementing the full orthogonality conditions, provides a practical framework for testing the model's performance under controlled conditions.

3.3 Parameter Estimation

Let $\boldsymbol{\theta}$ denote the collection of model parameters c_0, c_k, d_k, and σ_k^2. We drop the subscript B in X_B under the assumption that region B remains constant. The conditional density of X_B at x is:

$$f(x|z;\boldsymbol{\theta}) = \int f(x|\varepsilon, z) f(\varepsilon) d\varepsilon \tag{3.29}$$

where, for $x = \{t_1, \ldots, t_m\}$,

$$f(x|\varepsilon, z) = \frac{\exp\{-\int_B \lambda_{\varepsilon,z}(t)dt\}}{m!} \prod_{j=1}^{m} \lambda_{\varepsilon,z}(t_j) \tag{3.30}$$

with $\lambda_{\varepsilon,z}(t) = \exp\left\{\mu(t) + \mathbf{u}^{\mathbf{T}}\boldsymbol{\phi}(t)\right\}$, and $\mathbf{u} = g(z) + \varepsilon$ in nonlinear case. Thus, we have: $g(z) = d^{\mathbf{T}}\gamma(z)$, where $\gamma(z)$ is a vector of spline basis functions, and:

$$\begin{aligned} \lambda_{\varepsilon,z}(t) &= \exp\left\{\mu(t) + (g(z) + \varepsilon)^{\mathbf{T}}\boldsymbol{\phi}(t)\right\} \\ &= \exp\left\{\mu(t) + \sum_{k=1}^{p}(g_k(z) + \varepsilon_k)\phi_k(t)\right\} \\ &= \exp\left\{c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}\left(d_k^{\mathbf{T}}\gamma_k(z) + \varepsilon_k\right)c_k^{\mathbf{T}}\beta(t)\right\} \end{aligned} \tag{3.31}$$

$$\log\lambda(t) = \mu(t) + \sum_{k=1}^{p}\left(g_k(z) + \varepsilon_k\right)^{\mathbf{T}}\phi_k(t) = c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}\left(d_k^{\mathbf{T}}\gamma_k(z) + \varepsilon_k\right)c_k^{\mathbf{T}}\beta(t) \tag{3.32}$$

and

$$\begin{aligned}\log f(x \mid \varepsilon, z) &= -\int_B \lambda_{\varepsilon,z}(t)dt + \sum_{j=1}^{m} \log \lambda_{\varepsilon,z}(t_j) - \log m! \\ &= -\int_B \exp\left\{ c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p} \left(d_k^{\mathbf{T}}\gamma_k(z) + \varepsilon_k\right) c_k^{\mathbf{T}}\beta(t) \right\} + \\ &\{c_0^{\mathbf{T}} + \sum_{k=1}^{p}(d_k^{\mathbf{T}}\gamma_k(z) + \varepsilon_k)c_k^{\mathbf{T}}\} \sum_{j=1}^{m} \beta(t_j) - \log m!\end{aligned} \tag{3.33}$$

The simple linear case can be handled in a similar manner; more details are provided in Appendix A.1. The expression (3.33) depends on the t_js only through

$$b := \sum_{j=1}^{m} \beta(t_j) \tag{3.34}$$

For $f(\varepsilon)$ we have

$$f(\varepsilon) = \prod_{k=1}^{p} \frac{1}{(2\pi\sigma_k^2)^{1/2}} \exp(-\frac{\varepsilon_k^2}{2\sigma_k^2}) \tag{3.35}$$

So

$$\log f(\varepsilon) = \sum_{k=1}^{p} \left(-\frac{1}{2} \log 2\pi\sigma_k^2 - \frac{\varepsilon_k^2}{2\sigma_k^2} \right) \tag{3.36}$$

The parameters c_0, c_k, and d_k, along with the variances σ_k^2s of the ε_k, are estimated by penalized maximum likelihood, using the standard roughness penalty for spline smoothing. The penalized maximum likelihood estimator $\hat{\boldsymbol{\theta}}$ is then based on n independent realizations $x_1, \ldots, x_n$:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \ell_n(\boldsymbol{\theta}) \tag{3.37}$$

Where

$$\ell_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log f(x_i|z_i) - \xi_1 P(\mu) - \xi_2 \sum_{k=1}^{p} P(\phi_k) - \xi_3 \sum_{k=1}^{p} P(g_k) \tag{3.38}$$

ξ_1, ξ_2 and ξ_3 are smoothing parameters for μ, ϕ_ks, and g_ks. Since the ϕ_ks have unit norm but μ and the g_ks do not, it may be necessary to use ξ_1, ξ_2 and ξ_3 on different scales to achieve the same degree of smoothness.

3.4 Roughness Penalty Functions

The roughness penalties $P(\mu)$, $P(\phi_k)$, and $P(g_k)$ in (3.38), as described in Ramsay and Silverman (2005), take the form $P(f) = \int_B \|\mathbf{H}f(t)\|_F^2$. Here, $\mathbf{H}$ denotes the Hessian matrix of f, and $\|\cdot\|_F^2$ denotes the squared Frobenius norm of a matrix.

For spatial processes, the roughness penalty is typically expressed as:

$$P(f) = \int \left\{ \left(\frac{\partial^2 f}{\partial t_1^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial t_1 \partial t_2}\right)^2 + \left(\frac{\partial^2 f}{\partial t_2^2}\right)^2 \right\}, \tag{3.39}$$

while for temporal processes, it simplifies to $P(f) = \int (f'')^2$, where f'' denotes the second derivative of f with respect to time.

Since the dimension q of the functional basis $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ may be large, roughness penalties are necessary to ensure that the estimated functions μ, ϕ_k, and g_k are smooth.

When $f(t) = \mathbf{c}^{\mathbf{T}}\boldsymbol{\beta}(t)$, the penalty $P(f)$ becomes a quadratic form in $\mathbf{c}$: $P(f) = \mathbf{c}^{\mathbf{T}}\boldsymbol{\Omega}\mathbf{c}$. Consequently, we have $P(\mu) = c_0^{\mathbf{T}}\boldsymbol{\Omega}c_0$ and $P(\phi_k) = c_k^{\mathbf{T}}\boldsymbol{\Omega}c_k$.

For the temporal case, the matrix $\boldsymbol{\Omega}$ is given by:

$$\boldsymbol{\Omega} = \int_B \boldsymbol{\beta}''(t)\boldsymbol{\beta}''(t)^{\mathbf{T}} dt, \tag{3.40}$$

where the second derivative is applied component-wise to $\boldsymbol{\beta}(t)$.

For the spatial case, $\boldsymbol{\Omega}$ is decomposed as:

$$\boldsymbol{\Omega} = \boldsymbol{\Omega}_{11} + 2\boldsymbol{\Omega}_{12} + \boldsymbol{\Omega}_{22}, \tag{3.41}$$

with $\boldsymbol{\Omega}_{ij} = \int \boldsymbol{\beta}^{ij}(t)\boldsymbol{\beta}^{ij}(t)^{\mathbf{T}}dt$, where $\boldsymbol{\beta}^{ij}(t)$ denotes the second derivative of $\boldsymbol{\beta}(t)$ with respect to t_i and t_j.

Similarly, the penalty for g_k is given by $P(g_k) = d_k^{\mathbf{T}}\boldsymbol{\Delta}d_k$. For temporal processes, $\boldsymbol{\Delta}$ is defined as:

$$\boldsymbol{\Delta} = \int \boldsymbol{\gamma}''(z)\boldsymbol{\gamma}''(z)^{\mathbf{T}}dz, \tag{3.42}$$

and for spatial processes, it is decomposed analogously as:

$$\boldsymbol{\Delta} = \boldsymbol{\Delta}_{11} + 2\boldsymbol{\Delta}_{12} + \boldsymbol{\Delta}_{22}. \tag{3.43}$$

4 MAXIMIZING THE PENALIZED LIKELIHOOD

4.1 Derivative of Coefficients

The derivatives of ℓ_n are derived as follows. First, note that if $\mathbf{c}$ generically represent either c_0 or a c_k, then (omitting $\boldsymbol{\theta}$ from the notation of f; note that z and ε are assumed to be independent)

$$\begin{aligned}
\nabla_{\mathbf{c}} \log f(x \mid z) &= \frac{1}{f(x \mid z)} \int \{\nabla_{\mathbf{c}} f(x \mid \varepsilon, z)\} f(\varepsilon) d\varepsilon \\
&= \frac{1}{f(x \mid z)} \int \{\nabla_{\mathbf{c}} \log f(x \mid \varepsilon, z)\} f(x \mid \varepsilon, z) f(\varepsilon) d\varepsilon \\
&= \int \{\nabla_{\mathbf{c}} \log f(x \mid \varepsilon, z)\} f(\varepsilon \mid x) d\varepsilon \\
&= \mathbb{E}\{\nabla_{\mathbf{c}} \log f(x|\varepsilon, z)|x\}
\end{aligned} \tag{4.1}$$

From (3.33), we have:

$$\nabla_{\mathbf{c}} \log f(x|\varepsilon, z) = -\nabla_{\mathbf{c}} \int_B \lambda_{\varepsilon,z}(t) dt + \sum_{j=1}^{m} \nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t_j) \tag{4.2}$$

and

$$\begin{aligned}
\nabla_{\mathbf{c}} \int_B \lambda_{\varepsilon,z}(t) dt &= \int_B \nabla_{\mathbf{c}} \lambda_{\varepsilon,z}(t) dt \\
&= \int_B \{\nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t)\} \lambda_{\varepsilon,z}(t) dt
\end{aligned} \tag{4.3}$$

we have

$$\nabla_{\mathbf{c}} \log f(x \mid \varepsilon, z) = -\int_B \{\nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t)\} \lambda_{\varepsilon,z}(t) dt + \sum_{j=1}^{m} \nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t_j) \tag{4.4}$$

Then

$$\nabla_{\mathbf{c}} \log f(x \mid z) = -\int_B \mathbb{E}\left[\{\nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t)\}\, \lambda_{\varepsilon,z}(t) \mid x\right] dt + \sum_{j=1}^{m} \mathbb{E}\left[\nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}\left(t_j\right) \mid x\right] \tag{4.5}$$

Applying the same approach, we derive the result for the d_k:

$$\nabla_{\mathbf{d}} \log f(x \mid z) = -\int_B \mathbb{E}\left[\{\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t)\}\, \lambda_{\varepsilon,z}(t) \mid x\right] dt + \sum_{j=1}^{m} \mathbb{E}\left[\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}\left(t_j\right) \mid x\right] \tag{4.6}$$

See Appendix A.2 for details.

For the σ_k^2 terms, notice that only $f(x \mid \varepsilon)$ is differentiated:

$$\begin{aligned}
\frac{\partial}{\partial \sigma_k^2} \log f(x \mid z) =& \frac{1}{f(x \mid z)} \int f(x \mid \varepsilon, z) \{\frac{\partial}{\partial \sigma_k^2} f(\varepsilon)\} d\varepsilon \\
=& \frac{1}{f(x \mid z)} \int f(x \mid \varepsilon, z) \{\frac{\partial}{\partial \sigma_k^2} \log f(\varepsilon)\} f(\varepsilon) d\varepsilon \\
=& \int \{\frac{\partial}{\partial \sigma_k^2} \log f(\varepsilon)\} f(\varepsilon \mid x) d\varepsilon \\
=& \mathbb{E}\{\frac{\partial}{\partial \sigma_k^2} \log f(\varepsilon) \mid x\}
\end{aligned} \tag{4.7}$$

Since

$$\frac{\partial}{\partial \sigma_k^2} \log f(\varepsilon) = -\frac{1}{2\sigma_k^2} + \frac{\varepsilon_k^2}{2(\sigma_k^2)^2} \tag{4.8}$$

we get

$$\frac{\partial}{\partial \sigma_k^2} \log f(x \mid z) = -\frac{1}{2\sigma_k^2} + \frac{\mathbb{E}(\varepsilon_k^2 \mid x)}{2(\sigma_k^2)^2} \tag{4.9}$$

4.2 Newton–Raphson Algorithm

We apply a Newton–Raphson algorithm with coordinate-wise updates to maximize ℓ_n; that is, we will update the parameters $c_0, c_1, \ldots, c_p$, $d_1, \ldots, d_p$, and $\sigma_1^2, \ldots, \sigma_p^2$ one at a time and in that order. For instance, we update σ_k^2 as follows:
From (3.38) and (4.9),

$$\frac{\partial}{\partial \sigma_k^2}\ell_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \sigma_k^2}\log f(x_i \mid z_i) = -\frac{1}{2\sigma_k^2} + \frac{1}{2(\sigma_k^2)^2}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(\varepsilon_k^2 \mid x_i), \quad \text{for } i = 1, \ldots, n \tag{4.10}$$

We then obtain the explicit update by setting the above equation equal to zero:

$$(\sigma_k^2)^{new} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(\varepsilon_k^2 \mid x_i) \tag{4.11}$$

4.2.1 Mean-Only Model

$\lambda_{\varepsilon,z}(t) = \exp\left\{\mu(t) + (g(z)+\varepsilon)^{\mathbf{T}}\boldsymbol{\phi}(t)\right\}$ in 3.31, under the mean-only model, implies that $\log\lambda_0(t) = \mu(t) = c_0^{\mathbf{T}}\beta(t)$, which is constant across replications, and

$$\nabla_{\mathbf{c}_0}\log\lambda_0(t) = \boldsymbol{\beta}(t) \tag{4.12}$$

To update c_0 under the mean-only model, we proceed as follows, using 3.38:

$$\nabla_{\mathbf{c}_0}\ell_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{c}_0}\log f\left(x_i \mid z_i\right) - \xi_1 2\boldsymbol{\Omega}\mathbf{c}_0 \tag{4.13}$$

then from 4.5,

$$\begin{aligned}
\nabla_{\mathbf{c}_0}\ell_n(\boldsymbol{\theta}) &= -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\{\nabla_{\mathbf{c_0}}\log\lambda_0(t)\}\,\lambda_0(t)\mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\nabla_{\mathbf{c_0}}\log\lambda_0\left(t_{ij}\right)\mid x_i\right] - \xi_1 2\boldsymbol{\Omega}\mathbf{c}_0 \\
&= -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\boldsymbol{\beta}(t)\lambda_0(t)\mid x_i\right]dt + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\boldsymbol{\beta}(t_{ij})\mid x_i\right] - \xi_1 2\boldsymbol{\Omega}\mathbf{c}_0 \\
&= -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\lambda_0(t)dt + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\boldsymbol{\beta}\left(t_{ij}\right) - \xi_1 2\boldsymbol{\Omega}\mathbf{c}_0 \\
&= -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\lambda_0(t)dt + \frac{1}{n}\sum_{i=1}^{n}\mathbf{b}_i - \xi_1 2\boldsymbol{\Omega}\mathbf{c}_0
\end{aligned} \tag{4.14}$$

with

$$b_i = \sum_{j=1}^{m_i}\beta(t_{ij}) \tag{4.15}$$

For the second derivatives, we have:

$$\mathrm{H}_{\mathrm{c}_0}\ell_n(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\boldsymbol{\beta}(t)^{\mathbf{T}}\lambda_0(t)dt - \xi_1 2\boldsymbol{\Omega} \tag{4.16}$$

Since c_0 is unconstrained, we apply a standard Newton update:

$$\mathbf{c}_0^{\text{new}} = \mathbf{c}_0^{\text{old}} - \{\mathrm{H}_{\mathbf{c}_0}\ell_n(\boldsymbol{\theta}^{\text{old}})\}^{-1}\nabla_{\mathbf{c}_0}\ell_n(\boldsymbol{\theta}^{\text{old}}) \tag{4.17}$$

4.2.2 Estimation of Covariate Model Coefficients

For the coefficients related to the component score $\mathbf{u}$, we discuss both a simple linear relationship and a more general nonlinear relationship. For the general nonlinear model,

defined as: $g_k(z) = d_k^{\mathbf{T}}\gamma_k(z)$, the parameters c_k and d_k can be updated using the following procedure.

Updating c_k: From (3.32), (3.38), and (4.5) above, we have:

$$\nabla_{\mathbf{c}_k}\ell_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{c}_k}\log f\left(x_i \mid z_i\right) - \xi_2 2\Omega\mathbf{c}_k \tag{4.18}$$

Since

$$\nabla_{\mathbf{c}_k}\log\lambda_{\varepsilon,z}(t) = (d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\boldsymbol{\beta}(t) \tag{4.19}$$

we have

$$\begin{aligned}
\nabla_{\mathbf{c}_k}\ell_n(\boldsymbol{\theta}) =& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\{\nabla_{\mathbf{c}_k}\log\lambda_{\varepsilon,z}(t)\}\,\lambda_{\varepsilon,z}(t) \mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\nabla_{\mathbf{c}_k}\log\lambda_{\varepsilon,z}\left(t_{ij}\right) \mid x_i\right] - \xi_2 2\boldsymbol{\Omega}\mathbf{c}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\left\{(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\boldsymbol{\beta}(t)\right\}\lambda_{\varepsilon,z}(t) \mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\boldsymbol{\beta}(t_{ij}) \mid x_i\right] - \xi_2 2\boldsymbol{\Omega}\mathbf{c}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\mathbb{E}\left[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\lambda_{\varepsilon,z}(t) \mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k) \mid x_i\right]\sum_{j=1}^{m_i}\boldsymbol{\beta}(t_{ij}) - \xi_2 2\Omega\mathbf{c}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\mathbb{E}\left[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\lambda_{\varepsilon,z}(t) \mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k) \mid x_i\right]\mathbf{b}_i - \xi_2 2\Omega\mathbf{c}_k
\end{aligned} \tag{4.20}$$

For the second derivatives, it is best to use the approximation (which ignores the fact that

$\mathbb{E}$ depends on θ):

$$\begin{aligned}\nabla_{\mathbf{c}}\mathbb{E}\{\lambda_{\varepsilon,z}(t) \mid x_i\} &\approx \mathbb{E}\{\nabla_{\mathbf{c}}\lambda_{\varepsilon,z}(t) \mid x_i\} \\ &= \mathbb{E}[\{\nabla_{\mathbf{c}} \log \lambda_{\varepsilon,z}(t)\}\lambda_{\varepsilon,z}(t) \mid x_i]\end{aligned} \tag{4.21}$$

So, for the particular case of c_k we have:

$$\begin{aligned}\nabla_{\mathbf{c}_k}\mathbb{E}\{(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\lambda_{\varepsilon,z}(t) \mid x_i\} &\approx \mathbb{E}\{(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\nabla_{\mathbf{c}_k}\lambda_{\varepsilon,z}(t) \mid x_i\} \\ &= \mathbb{E}[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)\{\nabla_{\mathbf{c}_k} \log \lambda_{\varepsilon,z}(t)\}\lambda_{\varepsilon,z}(t) \mid x_i] \\ &= \mathbb{E}[(d_k^{\mathbf{T}}\gamma_k(z) + \varepsilon_k)^2\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i]\end{aligned} \tag{4.22}$$

Thus,

$$\mathrm{H}_{\mathrm{c}_k}\ell_n(\boldsymbol{\theta}) \approx -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\boldsymbol{\beta}(t)^{\mathbf{T}}\mathbb{E}[(d_k^{\mathbf{T}}\gamma_k(z_i) + \varepsilon_k)^2\lambda_{\varepsilon,z}(t) \mid x_i]dt - \xi_2 2\Omega \tag{4.23}$$

Since the components are aggregated sequentially, we update only the last component at each stage. Given that $c_1, \ldots, c_{k-1}$ have already been updated, let Γ be a $q \times \{q-(k-1)\}$ orthonormal basis of $\left\{\mathbf{J}_0 c_1^{new}, \ldots, \mathbf{J}_0 c_{k-1}^{new}\right\}^{\perp}$, where $\mathbf{J}_0 = \int_B \boldsymbol{\beta}(t)\boldsymbol{\beta}(t)^{\mathbf{T}}dt$:

$$\begin{aligned}\overline{\mathbf{c}}_k^{\text{new}} &= \Gamma\Gamma^{\mathbf{T}}\mathbf{c}_k^{\text{old}} - \Gamma\left\{\Gamma^{\mathbf{T}}\mathrm{H}_{\mathrm{c}_k}\ell_n\left(\boldsymbol{\theta}^{\text{old}}\right)\Gamma\right\}^{-1}\Gamma^{\mathbf{T}}\nabla_{\mathrm{c}_k}\ell_n\left(\boldsymbol{\theta}^{\text{old}}\right) \\ \mathbf{c}_k^{\text{new}} &= \overline{\mathbf{c}}_k^{\text{new}} \Big/ \left\{\left(\overline{\mathbf{c}}_k^{\text{new}}\right)^{\mathbf{T}}\mathbf{J}_0\overline{\mathbf{c}}_k^{\text{new}}\right\}^{1/2}\end{aligned} \tag{4.24}$$

Update d_k: To update d_k, notice that from (3.32), we have:

$$\nabla_{\mathbf{d}_k} \log \lambda_{\varepsilon,z}(t) = \gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t) \tag{4.25}$$

Then, from (3.38), we obtain:

$$\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{d}_k}\log f\left(x_i \mid z_i\right) - \xi_3 2\Delta\mathbf{d}_k \tag{4.26}$$

$$\begin{aligned}
\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}) =& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\{\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}(t)\}\,\lambda_{\varepsilon,z}(t) \mid x_i\right]dt \\
&+ \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}\left(t_{ij}\right) \mid x_i\right] - \xi_3 2\Delta\mathbf{d}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i\right]dt + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t_{ij}) \mid x_i\right] \\
&- \xi_3 2\Delta\mathbf{d}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i\right]dt + \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}} \mid x_i\right]\sum_{j=1}^{m_i}\boldsymbol{\beta}(t_{ij}) \\
&- \xi_3 2\Delta\mathbf{d}_k \\
=& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i\right]dt + \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\gamma_k(z_i)c_k^{\mathbf{T}} \mid x_i\right]\mathbf{b}_i - \xi_3 2\Delta\mathbf{d}_k
\end{aligned} \tag{4.27}$$

For the second derivative, we proceed as before:

$$\begin{aligned}
\nabla_{\mathbf{d}_k}\mathbb{E}(\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i) &\approx \mathbb{E}\{\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\nabla_{\mathbf{d}_k}\lambda_{\varepsilon,z}(t) \mid x_i\} \\
&= \mathbb{E}[\gamma_k(z_i)c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\{\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}(t)\}\lambda_{\varepsilon,z}(t) \mid x_i] \\
&= \mathbb{E}[\gamma_k(z_i)\left(c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\right)^2\lambda_{\varepsilon,z}(t)\gamma_k^{\mathbf{T}}(z_i) \mid x_i]
\end{aligned} \tag{4.28}$$

$$\mathrm{H}_{\mathrm{d}_k}\ell_n(\boldsymbol{\theta}) \approx -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}[\gamma_k(z_i)\left(c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\right)^2\lambda_{\varepsilon,z}(t)\gamma_k^{\mathbf{T}}(z_i) \mid x_i]dt - \xi_2 2\Delta \tag{4.29}$$

Since d_k is a single vector, we apply a standard Newton update:

$$\mathbf{d}_k^{\text{new}} = \mathbf{d}_k^{\text{old}} - \{\mathrm{H}_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}^{\text{old}})\}^{-1}\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}^{\text{old}}) \tag{4.30}$$

The simple linear model can be handled in a similar manner; more details are provided in Appendix A.3.

4.3 Laplace Approximation

The conditional densities $f(x \mid z)$ are computed by Laplace approximation. We have

$$\begin{aligned} f(x \mid z) &= \int f(x|\varepsilon, z) f(\varepsilon) d\varepsilon \\ &= \int \exp g(\varepsilon) d\varepsilon \end{aligned} \tag{4.31}$$

with

$$g(\varepsilon) = \log f(x \mid \varepsilon, z) + \log f(\varepsilon) \tag{4.32}$$

If $\hat{\varepsilon} = \arg\max g(\varepsilon)$, then $g(\varepsilon) \approx g(\hat{\varepsilon}) + \frac{1}{2}(\varepsilon - \hat{\varepsilon})^{\mathbf{T}} \mathbf{H} g(\hat{\varepsilon})(\varepsilon - \hat{\varepsilon})$ and

$$f(x \mid z) \approx \exp\{g(\hat{\varepsilon})\}(2\pi)^{\frac{p}{2}} \det(\mathbf{S})^{\frac{1}{2}} \tag{4.33}$$

with

$$\mathbf{S} = \{-\mathbf{H} g(\hat{\varepsilon})\}^{-1} \tag{4.34}$$

In effect, we are approximating:

$$f(x|\varepsilon, z) f(\varepsilon) \approx \exp\{g(\hat{\varepsilon})\}(2\pi)^{\frac{p}{2}} \det(\mathbf{S})^{\frac{1}{2}} \varphi_{(\hat{\varepsilon}, \mathbf{S})}(\varepsilon) \tag{4.35}$$

where $\varphi_{(\hat{\varepsilon},\mathbf{S})}(\varepsilon)$ denotes the pdf of a $N_p(\hat{\varepsilon}, \mathbf{S})$, so $\varepsilon \mid x \approx N_p(\hat{\varepsilon}, \mathbf{S})$. Then we can also approximate the moments:

$$\begin{aligned} \mathbb{E}(\varepsilon \mid x) &\approx \hat{\varepsilon} \\ \mathbb{E}\left(\varepsilon\varepsilon^{\mathbf{T}} \mid x\right) &\approx \mathbf{S} + \hat{\varepsilon}\hat{\varepsilon}^{\mathbf{T}} \end{aligned} \tag{4.36}$$

We find $\hat{\varepsilon}$ by a few steps of Newton-Raphson method for each x_i, since

$$g(\varepsilon) = -\int \lambda_{\varepsilon,z}(t)dt + \sum_{j=1}^{m} \log \lambda_{\varepsilon,z}\left(t_j\right) - \log m! + \sum_{k=1}^{p}\left(-\frac{1}{2}\log 2\pi\sigma_k^2 - \frac{\varepsilon_k^2}{2\sigma_k^2}\right) \tag{4.37}$$

the derivatives with respect to ε are, with $\mathbf{\Sigma} = \operatorname{diag}(\boldsymbol{\sigma}^2)$,

$$\nabla g(\varepsilon) = -\int \lambda_{\varepsilon,z}(t)\boldsymbol{\phi}(t)dt + \sum_{j=1}^{m} \boldsymbol{\phi}\left(t_j\right) - \mathbf{\Sigma}^{-1}\varepsilon \tag{4.38}$$

and

$$\mathbf{H}g(\varepsilon) = -\int \lambda_{\varepsilon,z}(t)\boldsymbol{\phi}(t)\boldsymbol{\phi}(t)^{\mathbf{T}}dt - \mathbf{\Sigma}^{-1} \tag{4.39}$$

5 SIMULATION

5.1 Design of Simulation Study

We implemented the algorithms in MATLAB to compute the proposed estimators. To assess their finite-sample performance, we conduct a series of simulation studies. Specifically, we simulate data from model (2.3) in a one-dimensional setting, as the estimation procedure is applied separately to each dimension. The simulations evaluate the consistency of the estimators by examining how their accuracy improves as the sample size increases.

We simulate data from model (2.3) for $p = 1$. Two types of simulations are conducted to evaluate the estimators under different conditions. In the first setting, the component score $\mathbf{u}$ follows a linear trend; in the second, $\mathbf{u}$ follows a quadratic trend. In both cases, $\mathbf{u}$ is modeled as the sum of a deterministic function g and a random error term ε, consistent with the structure assumed in the model. These trends are designed to reflect patterns that may arise in the real data application. We also examine the effect of the expected number of observations per replication, specifically the baseline rate $\int_a^b \lambda_0(t)dt$, which is determined by $\mu(t)$. For the temperature, we use the 2016 temperature range, which can be partitioned into the desired sample sizes.

We consider the model with $\mu(t) = \sin(\pi t) + c$ and $\phi(t) = \sqrt{2}\sin(\pi t) + c$ for $t \in [0, 1]$. Since $\int_0^1 \exp\sin(\pi t)dt \approx 1.98$, we set $c = \log 5$ and $c = \log 15$, corresponding to approximate baseline rates of 10 and 30, respectively. For simulation, we use sample sizes of 91, 183, and 366, corresponding to one quarter, half a year, and a full year of daily data, respectively.

For the estimation of the functional parameters, B-splines are widely used because they provide smooth, flexible approximations with strong numerical stability. Their recursive definition, introduced by de Boor (1972), enables efficient and stable computation of basis functions, even at higher degrees. This structure supports adaptability to complex data,

making B-splines especially valuable for smoothing tasks. We adopt a cubic B-spline basis with 10 equally spaced knots in $(0, 1)$, which has dimension $q = 14$. This is large enough for the smooth functions we are estimating. The smoothness parameters used in the simulation are chosen based on preliminary tests. Specifically, for μ, value of 1×10^{-3} is used; for ϕ, a value of 1×10^{-4} is used; and for g, a value of 1×10^{-5} is selected. These choices reflect the differing magnitudes and characteristics of the respective functions, ensuring appropriate smoothing for each component.

We examine three estimators: μ, ϕ, and g. For μ and g, we define the bias as $\|\mathbb{E}(\widehat{\mu} - \mu)\|$ and $\|\mathbb{E}(\widehat{g} - g)\|$, respectively; the standard deviation as $\left[\mathbb{E}\left\|\widehat{\mu} - \mathrm{E}(\widehat{\mu})\right\|^2\right]^{\frac{1}{2}}$ and $\left[\mathbb{E}\left\|\widehat{g} - \mathrm{E}(\widehat{g})\right\|^2\right]^{\frac{1}{2}}$; and the RMSE as $\left\{\mathbb{E}\left\|\widehat{\mu} - \mu\right\|^2\right\}^{\frac{1}{2}}$ and $\left\{\mathbb{E}\left\|\widehat{g} - g\right\|^2\right\}^{\frac{1}{2}}$, where $\|\cdot\|$ denotes the usual $L^2[0, 1]$ norm. For ϕ, we could not use these quantities directly due to sign indeterminacy (i.e., it is not possible to determine whether $\widehat{\phi}$ estimates ϕ or $-\phi$). Instead, we consider the bivariate estimators $\widehat{\phi(s)}\widehat{\phi(t)}$ for $\phi(s)\phi(t)$, which are sign-invariant. We then define the bias, standard deviation, and root mean squared error as before, except that $\|\cdot\|$ is taken to be the bivariate L^2 norm on $[0, 1] \times [0, 1]$. Expectations of these 3 functions are approximated using Monte Carlo simulation with 200 independent replications for each simulation scenario. We anticipate that, when the sample size is sufficiently large, the component score estimators exhibit consistency across a range of settings.

5.2 Initialization

This dissertation, as an extension of previous work Gervini (2017) and Gervini and Khanal (2018), preserves several features of the earlier model—especially in the mean-only formulation. For parameter initialization, we follow the approach proposed in earlier research, with appropriate modifications to accommodate the specifics of our setting.

The initialization of the mean function begins by setting $c_0 = c\mathbf{1}$, where each element of the vector takes the same constant value c. Since B-spline basis functions satisfy the partition of unity property, $\beta(t)^{\mathrm{T}}\mathbf{1} = 1$, this specification implies that the mean function

is constant over time, i.e., $\mu(t) \equiv c$, and consequently, the baseline intensity function is $\lambda_0(t) \equiv e^c$. Therefore, we have $\int_a^b \lambda_0(t)\,dt = e^c(b-a)$, and under the mean-only model, the m_is are independent and identically distributed as $\mathscr{P}\left(\int_a^b \lambda_0(t)\,dt\right)$. A natural estimator for $e^c(b-a)$ is the empirical average count $\bar{m}$, leading to the initialization $\hat{c} = \log\left(\frac{\bar{m}}{b-a}\right)$.

We then initialize the first component with $c_1 = \alpha\mathbf{1}$, which corresponds to a constant function $\phi_1(t) \equiv \alpha$. Enforcing the unit norm condition yields $\|\phi_1\|^2 = \int_a^b \alpha^2\,dt = \alpha^2(b-a) = 1$, implying $\alpha = \frac{1}{(b-a)^{\frac{1}{2}}}$. For the one-component initial model, we have $\lambda_i(t) = \lambda_0(t)e^{u_{i1}\alpha} = \lambda_0(t)e^{(g_1(z_i)+\varepsilon_{i1})\alpha}$, and hence $\int \lambda_i(t)\,dt = e^{(g_1(z_i)+\varepsilon_{i1})\alpha}\int \lambda_0$. Given that m_i approximates $\int \lambda_i$, we estimate ε_{i1} as $\hat{\varepsilon}_{i1} = \frac{\log\left(\frac{m_i}{\int \lambda_0}\right)}{\alpha} - g_1(z_i)$, up to a scale factor. The initial variance σ_1^2 is then estimated as the sample variance of the $\hat{\varepsilon}_{i1}$ values.

For each subsequent component, we first define the initial variance recursively as $\hat{\sigma}_k^2 = \frac{\hat{\sigma}_{k-1}^2}{2}$. Then, let $\bar{c}_k$ be the orthogonal projection of $\mathbf{1}$ onto the space $\{\mathbf{J}_0 c_1, \ldots, \mathbf{J}_0 c_{k-1}\}^{\perp}$; that is, if Γ denotes a $q \times (q-k+1)$ orthonormal basis of $\{\mathbf{J}_0 c_1, \ldots, \mathbf{J}_0 c_{k-1}\}^{\perp}$, then the projection is given by $\bar{c}_k = \Gamma\Gamma^{\mathrm{T}}\mathbf{1}$. We then define $\hat{c}_k = \frac{\bar{\mathbf{c}}_k}{\left(\bar{\mathbf{c}}_k^{\mathrm{T}}\mathbf{J}_0\bar{\mathbf{c}}_k\right)^{\frac{1}{2}}}$.

5.3 Analysis of Simulation Results: Linear Model

For the simple linear model, we assume $u = g(z) + \epsilon$, where $g(z) = d \cdot z$ captures the linear trend, z denotes the mean-centered temperature data, and ϵ is a random error term. Since the component function is constrained to have unit norm, we cannot scale u arbitrarily large, as doing so would compromise numerical stability. To balance interpretability and robustness, the slope d is set to 0.3, and the error term ϵ is assigned a moderate standard deviation of 0.03.

The simulation results are presented in Table 1 and Table 2 below. For these estimators, all three performance metrics—bias, standard deviation, and RMSE—steadily decrease with increasing sample size at both baseline rates, indicating improved accuracy and stability of estimation. Notably, the bias for $\widehat{\mu}$ remains relatively stable across different sample

sizes and baseline rates, while its standard deviation and RMSE exhibit the expected decline. In contrast, the estimators $\widehat{g}$ and $\widehat{\phi}$ show clear convergence toward the true functions g and ϕ, with consistent improvements observed at both baseline rates. This demonstrates the model's robustness under different baseline rates.

Table 1: Simulation Results for Baseline Rate = 10, Linear u

Parameter	$N = 91$	$N = 183$	$N = 366$
μ-Bias	0.0768	0.0747	0.0725
μ-STD	0.0520	0.0376	0.0268
μ-RMSE	0.0997	0.0865	0.0791
Φ_2 (Bivariate)-Bias	0.0842	0.0643	0.0525
Φ_2 (Bivariate)-STD	0.2021	0.1499	0.1107
Φ_2 (Bivariate)-RMSE	0.2186	0.1629	0.1223
g-Bias	5.6317×10^{-4}	3.7867×10^{-4}	2.2905×10^{-4}
g-STD	0.0311	0.0212	0.0157
g-RMSE	0.0338	0.0248	0.0192

Note: Bias, standard deviation (STD), and root mean square error (RMSE) of parameter estimators μ, Φ_2, and g.

Table 2: Simulation Results for Baseline Rate = 30, Linear u

Parameter	$N = 91$	$N = 183$	$N = 366$
μ-Bias	0.0756	0.0730	0.0720
μ-STD	0.0356	0.0247	0.0184
μ-RMSE	0.0856	0.0781	0.0750
Φ_2 (Bivariate)-Bias	0.0608	0.0505	0.0430
Φ_2 (Bivariate)-STD	0.1457	0.1039	0.0709
Φ_2 (Bivariate)-RMSE	0.1577	0.1154	0.0828
g-Bias	6.8061×10^{-4}	2.9093×10^{-4}	2.3724×10^{-4}
g-STD	0.0177	0.0122	0.0082
g-RMSE	0.0241	0.0157	0.0141

Note: Bias, standard deviation (STD), and root mean square error (RMSE) of parameter estimators μ, Φ_2, and g.

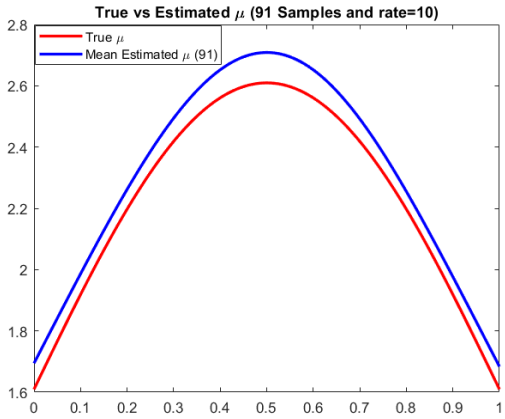

Figure 1: Fitted μ (Linear model); Sample: 91, Rate: 10

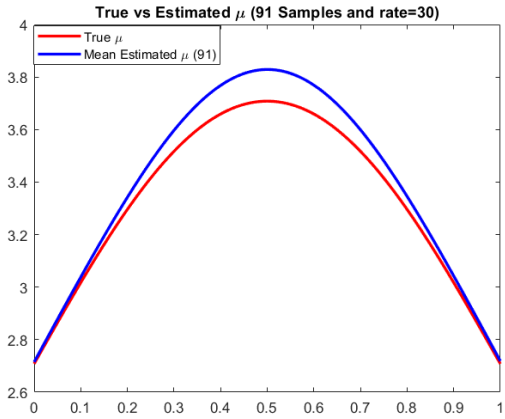

Figure 2: Fitted μ (Linear model); Sample: 91, Rate: 30

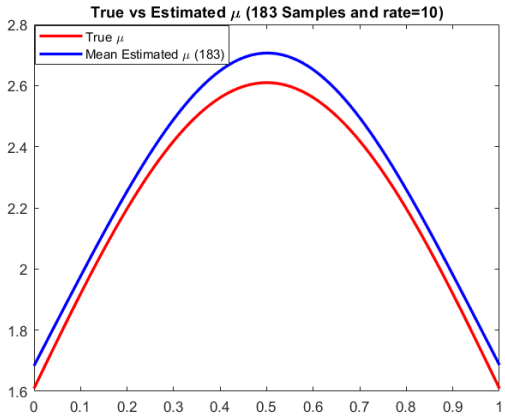

Figure 3: Fitted μ (Linear model); Sample: 183, Rate: 10

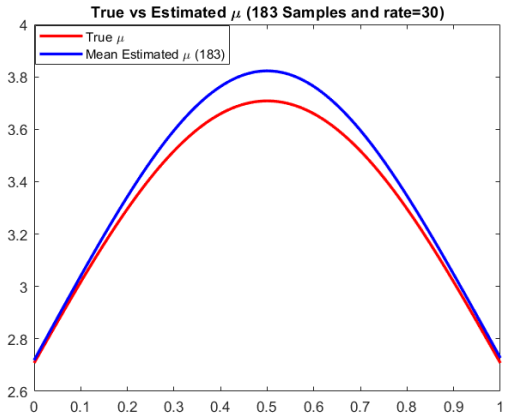

Figure 4: Fitted μ (Linear model); Sample: 183, Rate: 30

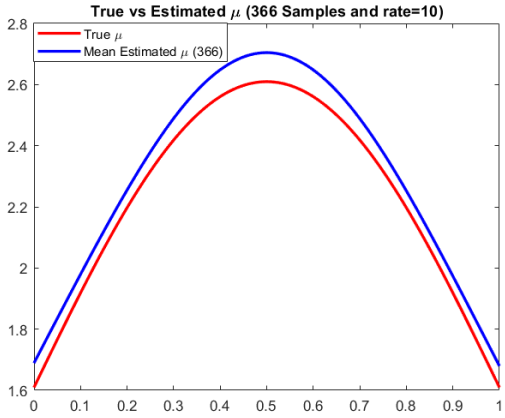

Figure 5: Fitted μ (Linear model); Sample: 366, Rate: 10

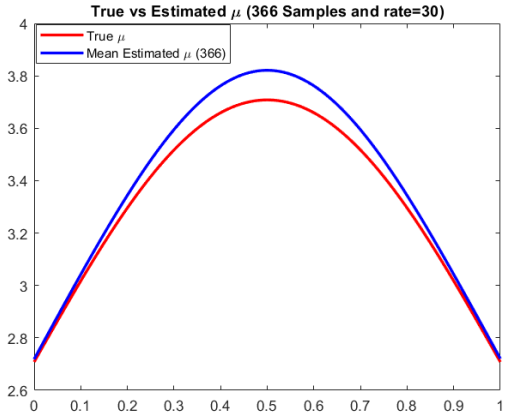

Figure 6: Fitted μ (Linear model); Sample: 366, Rate: 30

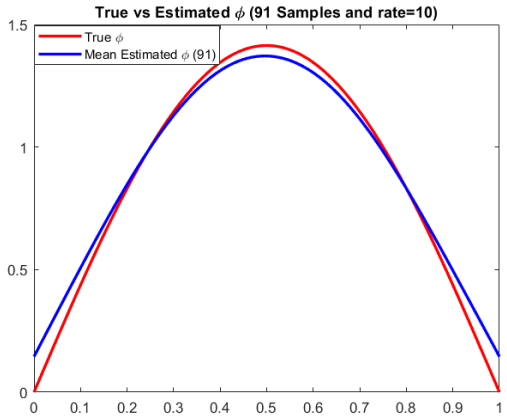

Figure 7: Fitted ϕ (Linear model); Sample: 91, Rate: 10

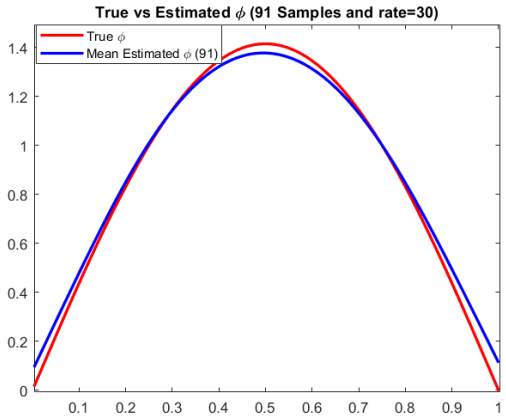

Figure 8: Fitted ϕ (Linear model); Sample: 91, Rate: 30

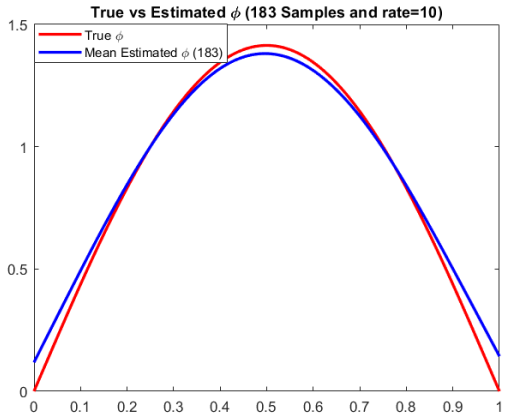

Figure 9: Fitted ϕ (Linear model); Sample: 183, Rate: 10

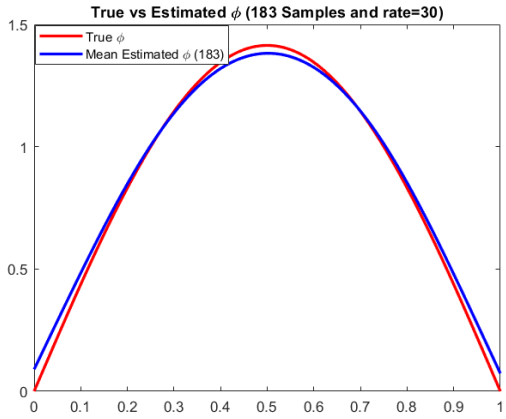

Figure 10: Fitted ϕ (Linear model); Sample: 183, Rate: 30

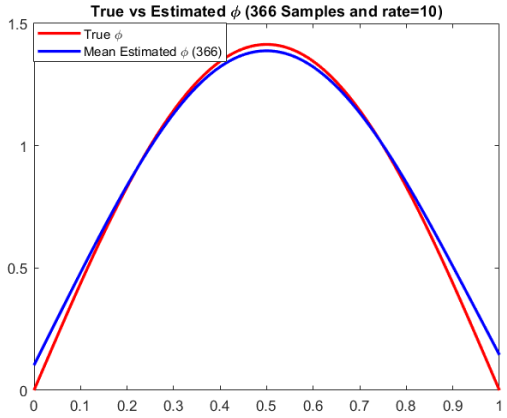

Figure 11: Fitted ϕ (Linear model); Sample: 366, Rate: 10

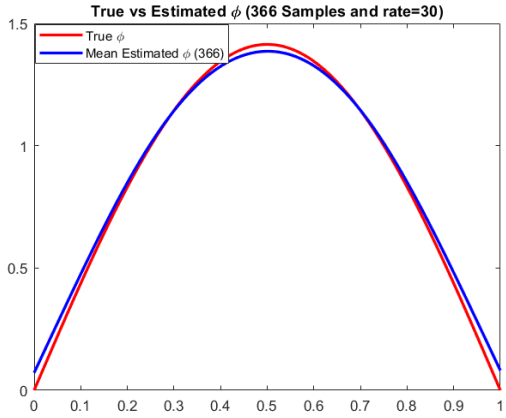

Figure 12: Fitted ϕ (Linear model); Sample: 366, Rate: 30

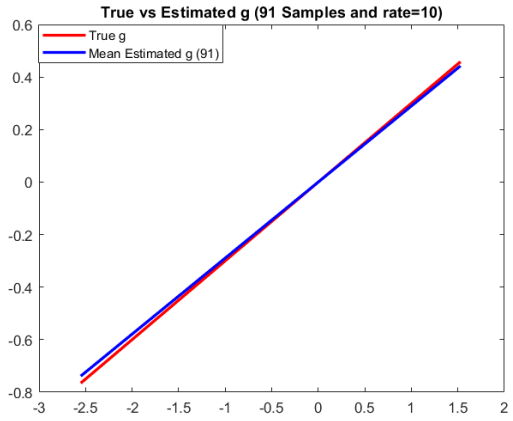

Figure 13: Fitted g (Linear model); Sample: 91, Rate: 10

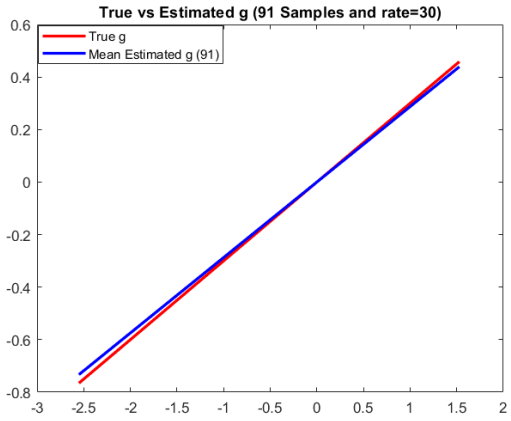

Figure 14: Fitted g (Linear model); Sample: 91, Rate: 30

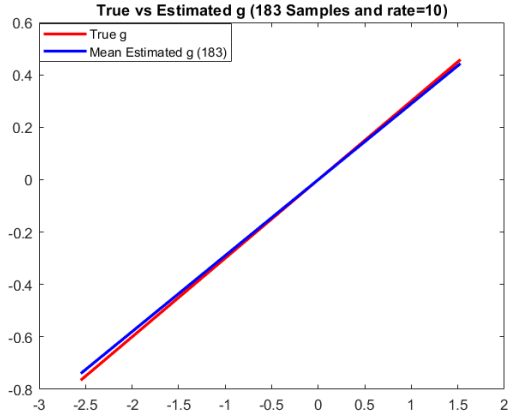

Figure 15: Fitted g (Linear model); Sample: 183, Rate: 10

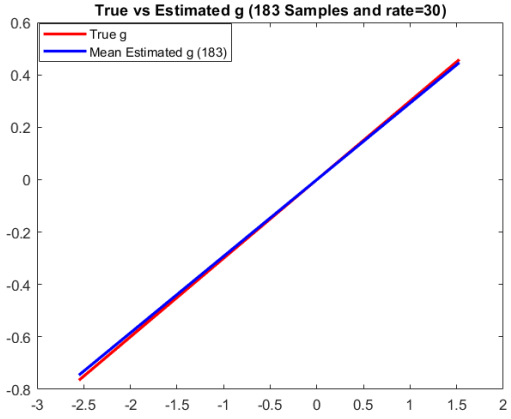

Figure 16: Fitted g (Linear model); Sample: 183, Rate: 30

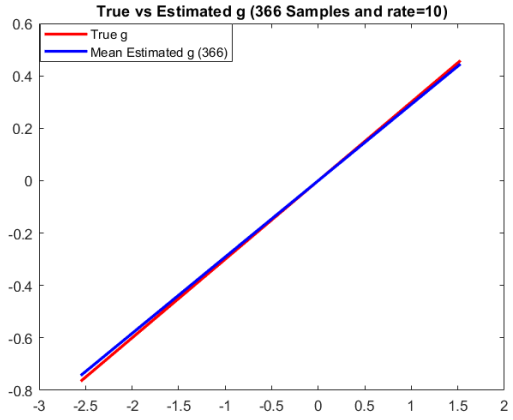

Figure 17: Fitted g (Linear model); Sample: 366, Rate: 10

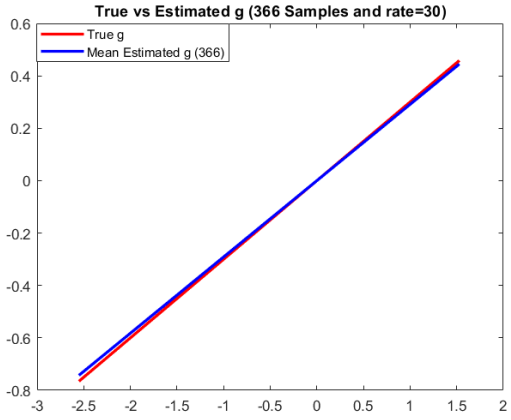

Figure 18: Fitted g (Linear model); Sample: 366, Rate: 30

The above figures display the curve fitting results for Function μ, Function ϕ, and Function g, where the estimated curves represent the mean of 200 replications obtained from independently simulated datasets under the linear model. The estimated ϕ and g functions exhibit consistently good fits across different sample sizes and baseline rates, indicating the robustness of the estimation procedure under varying conditions. In comparison, the estimated μ functions tend to show slightly larger deviations from the true curves. However, when the estimated mean function μ is rescaled to 97% of its original magnitude, the overall curve fit improves substantially, as illustrated in the figures below.

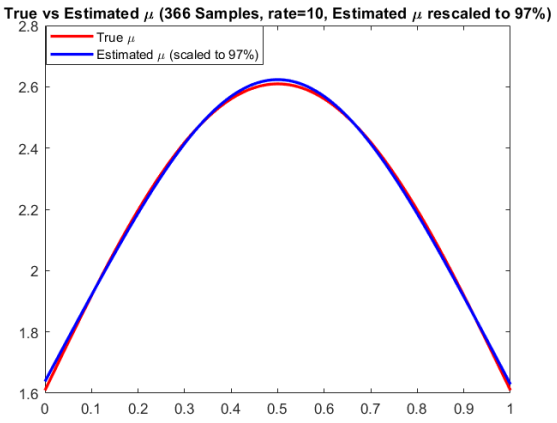

Figure 19: Fitted μ (Linear model), Rescaled; Sample: 366, Rate: 10

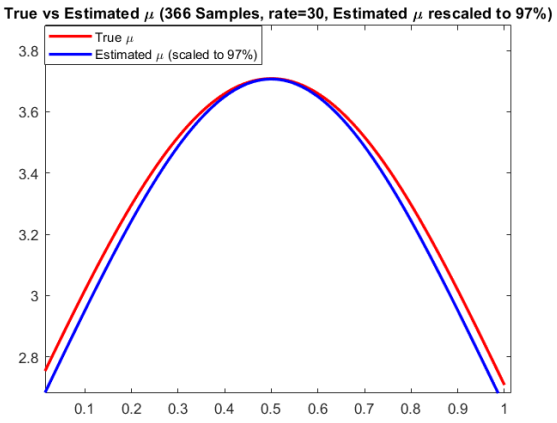

Figure 20: Fitted μ (Linear model), Rescaled; Sample: 366, Rate: 30

This minor discrepancy—on the order of 3%—is well within an acceptable error margin and may be attributed to inherent numerical approximations used in the estimation algorithm, which inevitably introduce minor computational inaccuracies as well as bias from penalization used to enforce smoothness. Despite these factors, the general shape and trend of μ are still well captured, supporting the reliability of the approach.

5.4 Analysis of Simulation Results: Quadratic Model

For the quadratic model, we assume $u = g(z) + \epsilon$, where $g(z) = d^{\mathrm{T}}\gamma(z)$ captures the quadratic trend, z denotes temperature, $\gamma(z)$ represents the mean-centered quadratic B-

spline basis functions with no interior knots on z, and ϵ is a random error term. The function $g(z)$ was selected in a manner similar to the simple linear model, with a standard deviation of 0.3, and the random error term was assigned a standard deviation of 0.03.

The simulated results are displayed in Table 3 and Table 4. We observe performance patterns that are highly consistent with those in the linear case. Most of the three metrics—bias, standard deviation, and RMSE—decrease with increasing sample sizes at both baseline rates, reaffirming that estimation accuracy improves as more data becomes available. An exception occurs at the intermediate sample size of 183 at baseline rate 30, where the bias of $\widehat{\mu}$ slightly increases before decreasing again at 366. This fluctuation is minor, and given the overall improvement trend, we still consider the estimator to be consistent. As in the linear model, the bias of $\widehat{\mu}$ remains relatively stable across sample sizes, while its standard deviation and RMSE show notable reductions. The estimators $\widehat{g}$ and $\widehat{\phi}$ again show strong convergence toward the true functions, even under the more complex quadratic structure.

Table 3: Simulation Results for Baseline Rate = 10, Quadratic u

Parameter	$N = 91$	$N = 183$	$N = 366$
μ-Bias	0.0599	0.0586	0.0580
μ-STD	0.0519	0.0364	0.0258
μ-RMSE	0.0876	0.0731	0.0657
Φ_2 (Bivariate)-Bias	0.0804	0.0648	0.0512
Φ_2 (Bivariate)-STD	0.2188	0.1616	0.1154
Φ_2 (Bivariate)-RMSE	0.2328	0.1739	0.1261
g-Bias	0.0104	0.0087	0.0082
g-STD	0.0344	0.0251	0.0173
g-RMSE	0.0358	0.0265	0.0191

Note: Bias, standard deviation (STD), and root mean square error (RMSE) of parameter estimators μ, Φ_2, and g.

Table 4: Simulation Results for Baseline Rate = 30, Quadratic u

Parameter	$N = 91$	$N = 183$	$N = 366$
μ-Bias	0.0574	0.0599*	0.0562
μ-STD	0.0356	0.0249	0.0181
μ-RMSE	0.0699	0.0665	0.0598
Φ_2 (Bivariate)-Bias	0.0468	0.0446	0.0399
Φ_2 (Bivariate)-STD	0.1544	0.1035	0.0801
Φ_2 (Bivariate)-RMSE	0.1611	0.1125	0.0893
g-Bias	0.0065	0.0063	0.0019
g-STD	0.0200	0.0157	0.0106
g-RMSE	0.0209	0.0169	0.0108

Note: Bias, standard deviation (STD), and root mean square error (RMSE) of parameter estimators μ, Φ_2, and g.

The fitted curves for Function μ, Function ϕ, and Function g shown below represent the mean of 200 replications obtained from independently simulated datasets under the quadratic model.

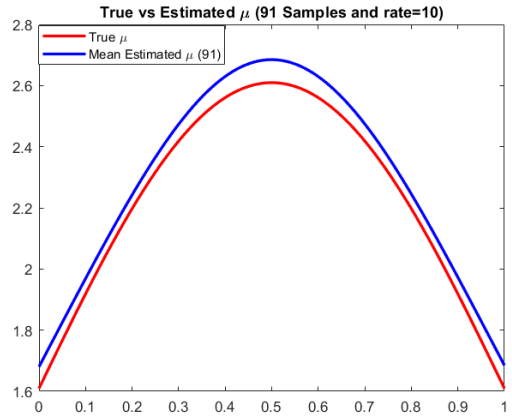

Figure 21: Fitted μ (Quadratic model); Sample: 91, Rate: 10

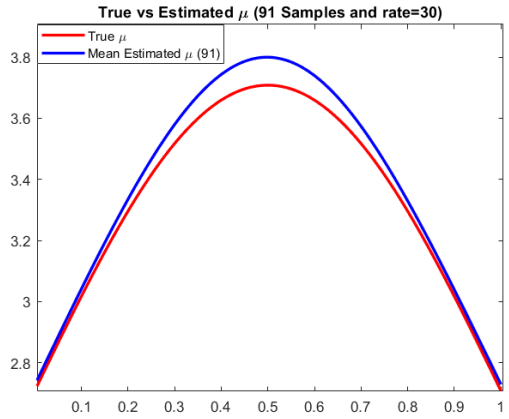

Figure 22: Fitted μ (Quadratic model); Sample: 91, Rate: 30

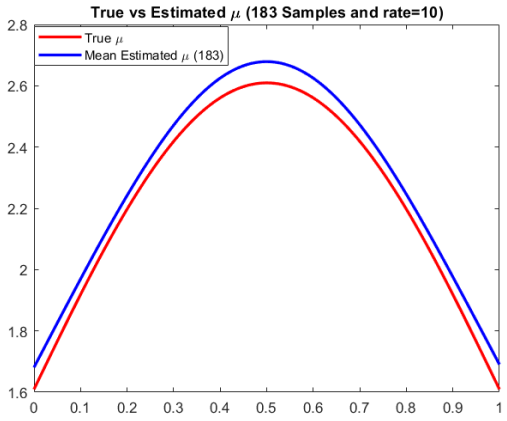

Figure 23: Fitted μ (Quadratic model); Sample: 183, Rate: 10

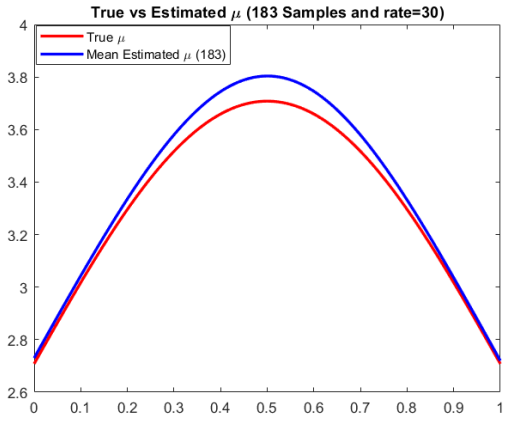

Figure 24: Fitted μ (Quadratic model); Sample: 183, Rate: 30

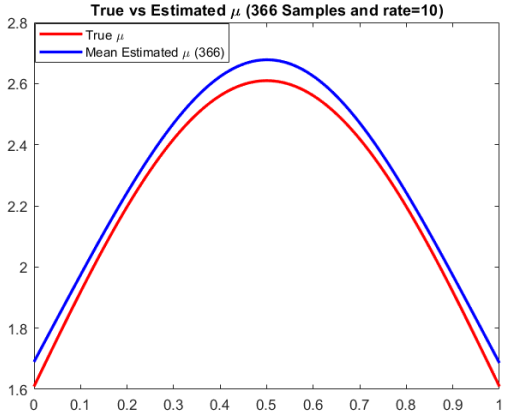

Figure 25: Fitted μ (Quadratic model); Sample: 366, Rate: 10

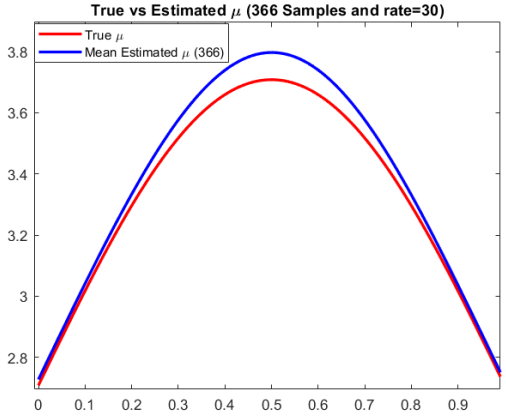

Figure 26: Fitted μ (Quadratic model); Sample: 366, Rate: 30

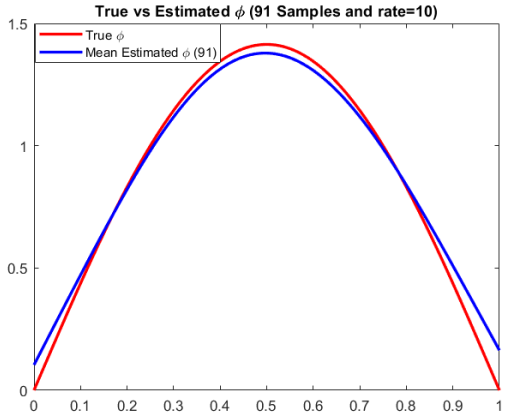

Figure 27: Fitted ϕ (Quadratic model); Sample: 91, Rate: 10

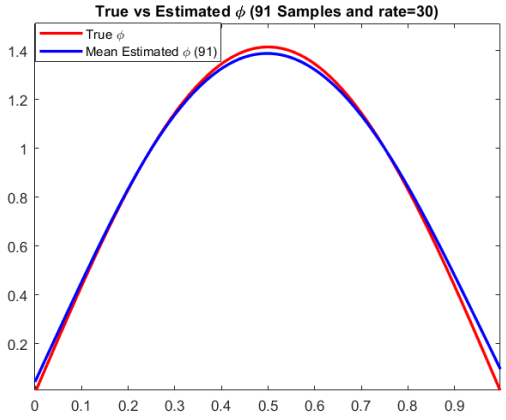

Figure 28: Fitted ϕ (Quadratic model); Sample: 91, Rate: 30

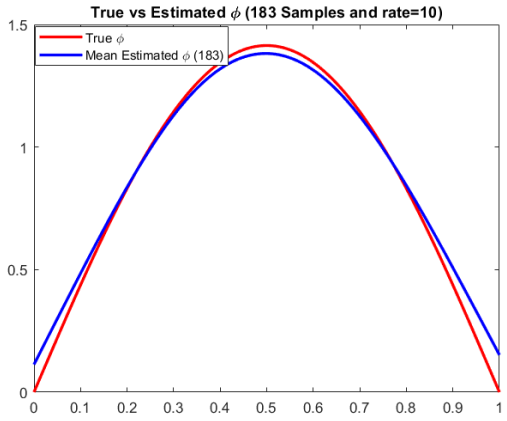

Figure 29: Fitted ϕ (Quadratic model); Sample: 183, Rate: 10

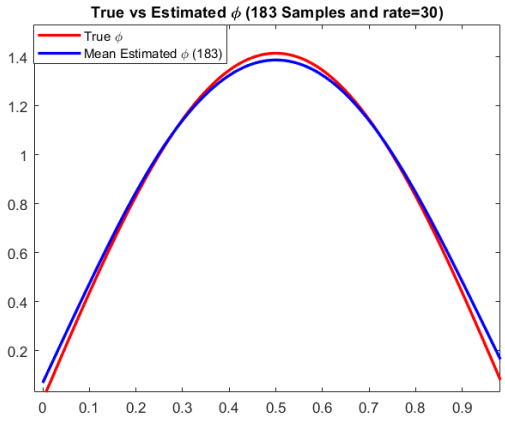

Figure 30: Fitted ϕ (Quadratic model); Sample: 183, Rate: 30

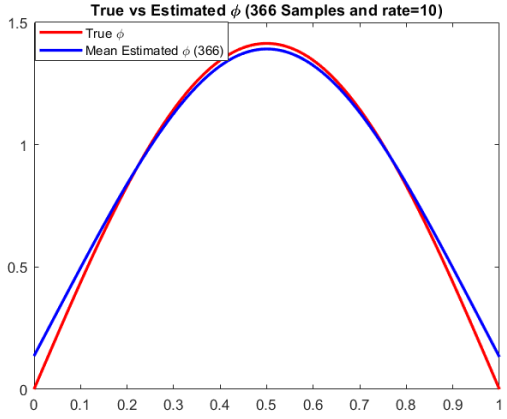

Figure 31: Fitted ϕ (Quadratic model); Sample: 366, Rate: 10

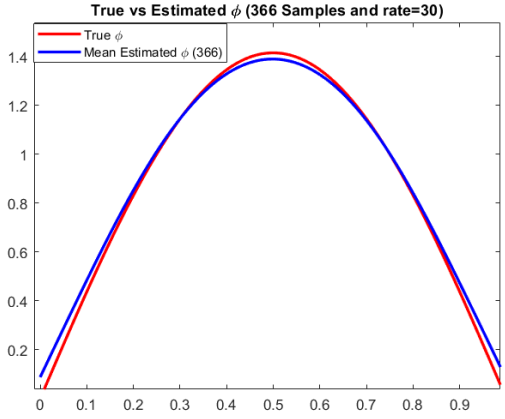

Figure 32: Fitted ϕ (Quadratic model); Sample: 366, Rate: 30

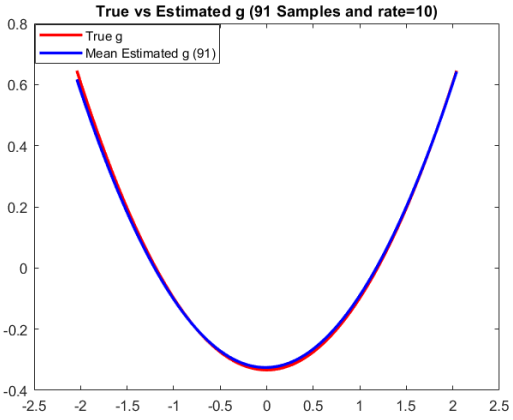

Figure 33: Fitted g (Quadratic model); Sample: 91, Rate: 10

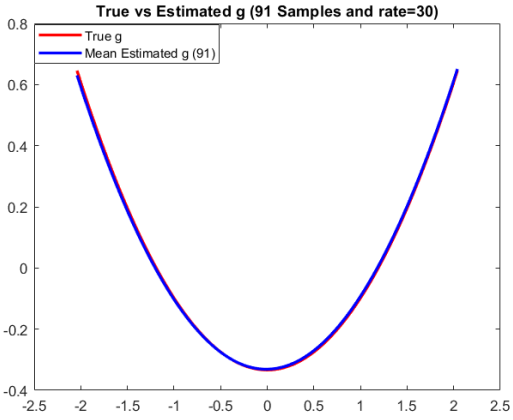

Figure 34: Fitted g (Quadratic model); Sample: 91, Rate: 30

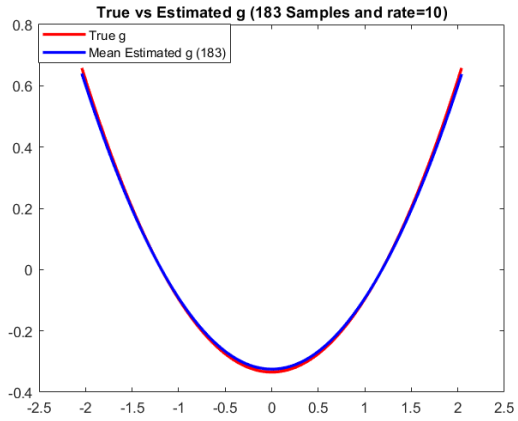

Figure 35: Fitted g (Quadratic model); Sample: 183, Rate: 10

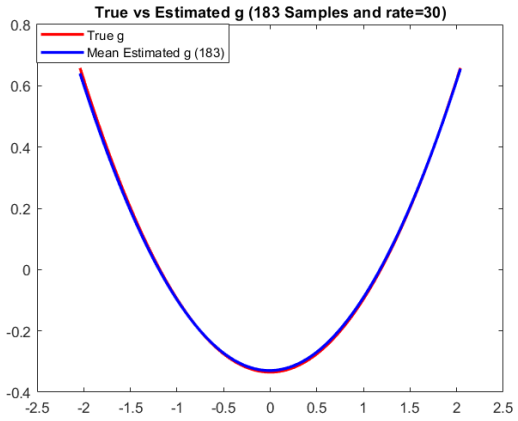

Figure 36: Fitted g (Quadratic model); Sample: 183, Rate: 30

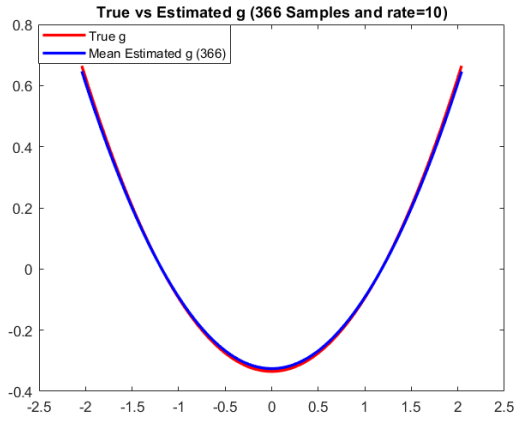

Figure 37: Fitted g (Quadratic model); Sample: 366, Rate: 10

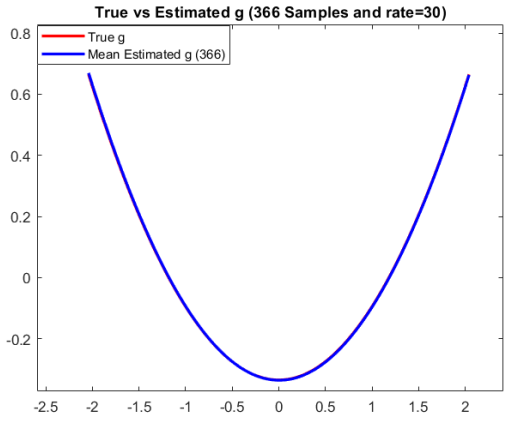

Figure 38: Fitted g (Quadratic model); Sample: 366, Rate: 30

These curves illustrate the estimation performance clearly. Both ϕ and g are consistently well estimated across different sample sizes and baseline rates, reflecting the robustness of the estimation method under a nonlinear model. Although μ shows slightly larger deviations from the true function, rescaling its magnitude to 98% significantly improves the visual fit, as shown in the rescaled Function μ below. As with the linear case, this 2% minor discrepancy likely arises from inherent numerical approximations in the estimation procedure or penalization bias. Overall, the general trend and shape of all three functions are well recovered, confirming the reliability of the approach even in the presence of nonlinear covariate effects.

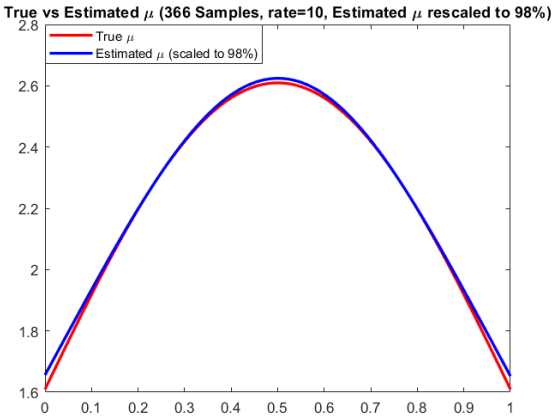

Figure 39: Fitted μ (Quadratic model), Rescaled; Sample: 366, Rate: 10

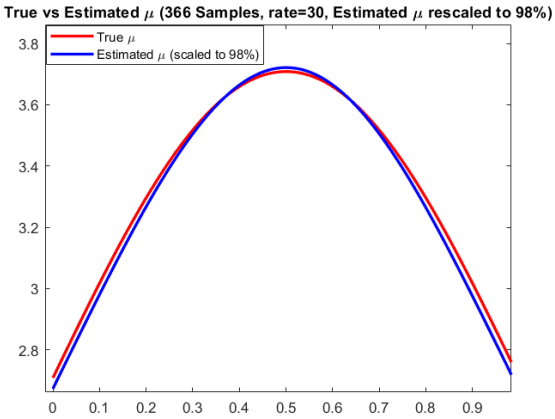

Figure 40: Fitted μ (Quadratic model), Rescaled; Sample: 366, Rate: 30

We conducted simulation studies under both linear and nonlinear covariate models to evaluate the performance of the proposed model. Across a range of sample sizes and baseline rates, the estimators for μ, ϕ, and g consistently exhibited decreasing bias, standard deviation, and RMSE as sample size increased, confirming improved accuracy with more data. In both the linear and quadratic cases, the estimators $\widehat{\phi}$ and $\widehat{g}$ demonstrated strong convergence to their true functions. While $\widehat{\mu}$ showed slightly larger deviations, the magnitude of these errors remained small. The overall trends of all functions were well recovered—supporting the stability and robustness of the method under varying covariate structures.

6 EMPIRICAL APPLICATION

6.1 Description of Divvy Data

Previous research has shown that the Divvy dataset exhibits pronounced temporal patterns in bicycle demand, shaped by both seasonal trends and weekly usage cycles. Building on these findings, we now introduce the covariate-based modeling framework to this dataset in order to extract deeper insights and quantify the effects of external factors such as temperature on temporal demand variation.

In the simulation study, we considered baseline rates of 10 and 30 and demonstrated the model's stability under both linear and nonlinear trends. The real data used in the application exhibits a similar scenario, so the model's stability in this context is supported by the simulation results. We selected a specific time period (April 1 to November 30, 2016) and Station 166, located at the corner of Ashland and Wrightwood Avenues. This combination was chosen because the average daily number of bicycle trips at this station during this time falls within a moderate range, consistent with the simulation's assumed rate levels. Such consistency helps ensure that the stability observed in simulations—with comparable sample sizes—is maintained when applied to real data. This time frame offers $n = 244$ daily replications of the temporal process, providing a robust basis for statistical analysis. The temperature data is the daily average temperature collected from https://www.weather.gov/wrh/Climate?wfo=lot.

6.2 Evaluation of the Mean-Only Model

We begin the empirical results with the mean-only model. The following figures display the estimated μ and ϕ functions obtained from the algorithm for the linear and quadratic models. In the simulation studies, the μ and ϕ functions exhibit relatively simple trends, characterized by smooth sinusoidal patterns. In contrast, when applied to real data from

the Divvy bicycle-sharing system, the estimated μ and ϕ functions display more complex fluctuations in both models, reflecting intricate daily bike usage patterns. These variations highlight the model's ability to capture complexities in the real data, providing a more detailed depiction of ridership behavior compared to the idealized trends observed in simulations.

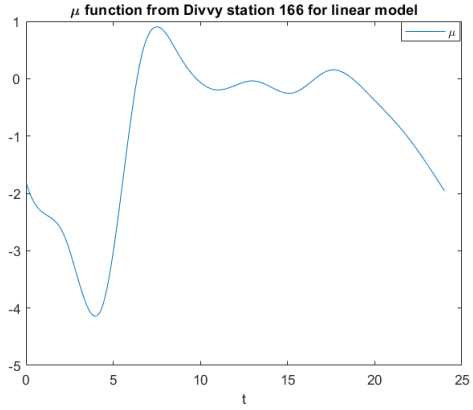

Figure 41: μ Function from Divvy Station 166 for Linear model

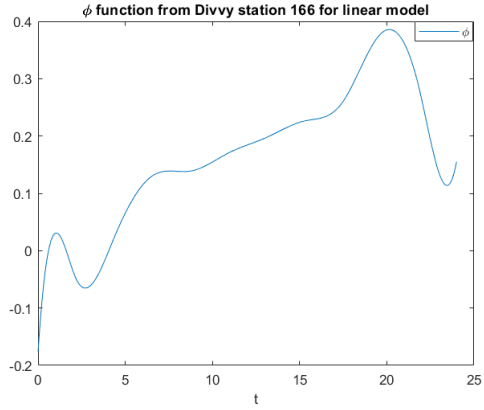

Figure 42: ϕ Function from Divvy Station 166 for Linear model

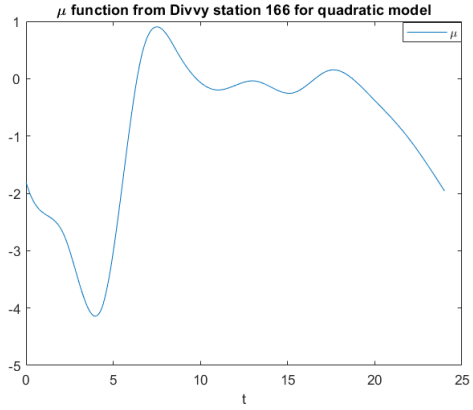

Figure 43: μ Function from Divvy Station 166 for Quadratic model

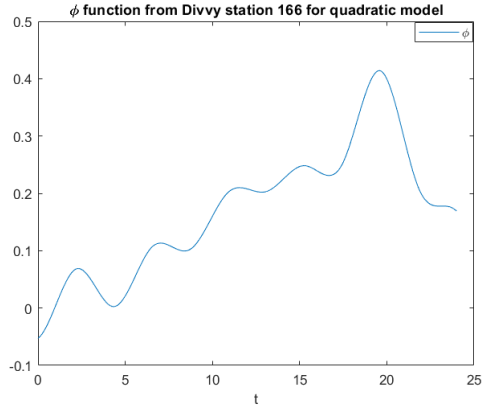

Figure 44: ϕ Function from Divvy Station 166 for Quadratic model

The model in Equation 2.4 can then be understood by plotting the baseline intensity function plus and minus a multiple of the ϕ function, as shown below:

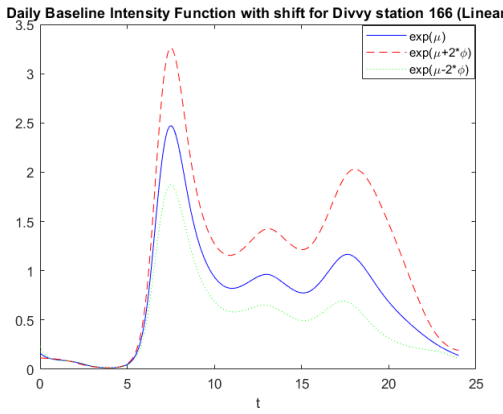

Figure 45: Daily Baseline Intensity Function with Shift for Divvy station 166 (Linear)

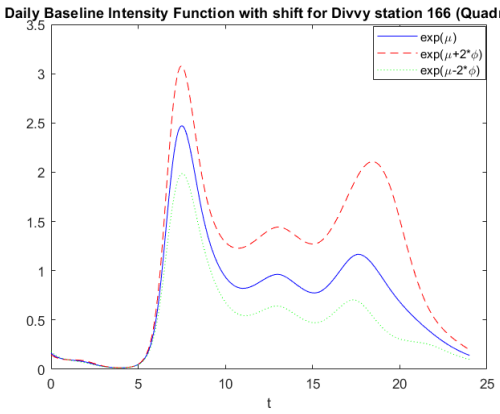

Figure 46: Daily Baseline Intensity Function with Shift for Divvy Station 166 (Quadratic)

The linear and the quadratic cases exhibit many similarities in the mean-only model. The solid line represents the baseline intensity function of daily bicycle usage at Station 166 in the Divvy bicycle-sharing system. We can observe three peaks: the first and largest around 7:30–8:00 a.m., the second around 12:30–1:00 p.m., and the last around 5:30–6:00 p.m. The dashed and dotted lines represent the baseline multiplied by the exponent of the component in Equation 2.4, evaluated at positive and negative component scores, respectively. A negative component score $\mathbf{u}$ corresponds to a flattening of the baseline function, while a positive score corresponds to a sharpening of it. The $\mathbf{u}$ effects vary across the time periods, and we will discuss the aggregate effect in the next section.

6.3 Assessing Covariate Effects on Bicycle Usage

We investigate the effect of temperature on bicycle usage by comparing two models based on observed data from the Divvy bicycle-sharing system. Both models share the following general form:

$$u = g(z) + \epsilon,$$

where u denotes the latent score related to bicycle usage, z represents temperature, and ϵ is a random error term.

- **Linear model**: $g(z) = d \cdot z$, where z is the mean-centered temperature variable, capturing a simple linear dependence on temperature.

- **Quadratic model**: $g(z) = d^{\mathbf{T}}\gamma(z)$, where $\gamma(z)$ is a mean-centered quadratic B-spline basis with no interior knots, enabling a global quadratic relationship.

The figures below display results from real Divvy bicycle-sharing data, illustrating the estimated temperature effect on bicycle usage under two models: linear and quadratic. From these figures, the quadratic model appears to better capture the relationship between temperature and bicycle usage.

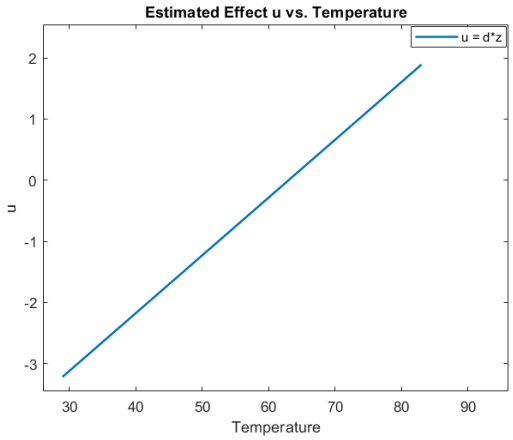

Figure 47: Temperature vs. u (Linear model)

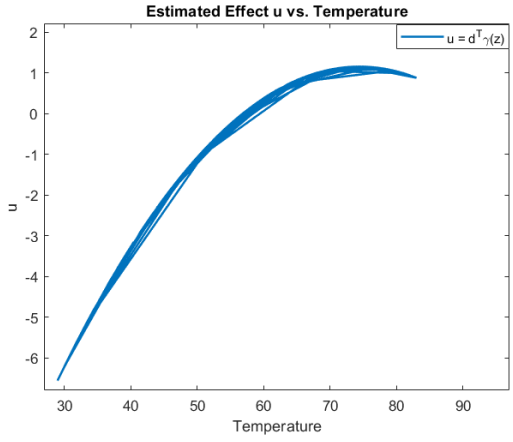

Figure 48: Temperature vs. u (Quadratic model, Line plot)

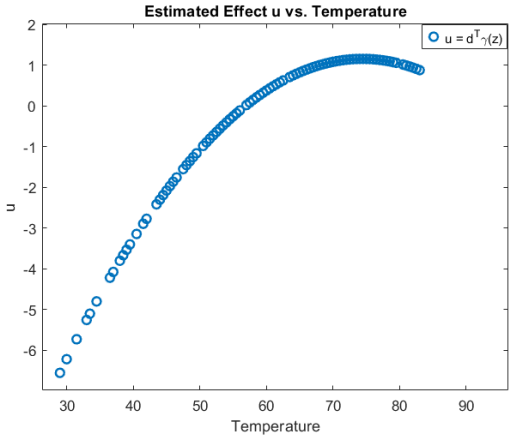

Figure 49: Temperature vs. u (Quadratic model, Scatter plot)

The linear model reflects a simple upward trend—suggesting that bicycle usage increases as temperature rises. However, it fails to account for more complex behavioral responses to extremely hot weather. In contrast, the quadratic model reveals a more realistic and detailed pattern: bicycle usage peaks at moderate temperatures (approximately 75°F), when outdoor conditions are most favorable, and declines at both low and high temperatures. Although the **u** values decrease during periods of extreme heat, they remain positive, indicating that summer tends to elevate the baseline function, whereas winter, with negative **u** values, tends to lower it. This nonlinear relationship is clearly depicted in the figures and aligns with common behavioral intuition—people are more likely to ride bicycles when the weather is neither too cold nor too hot.

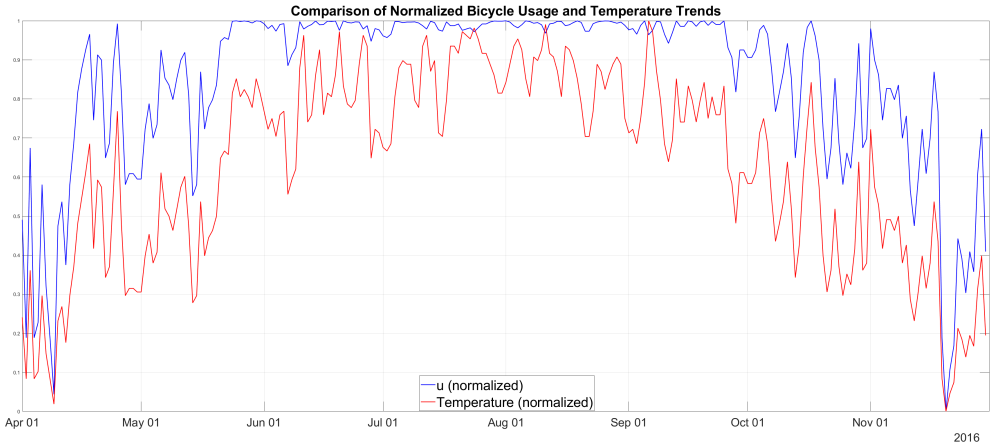

Figure 50: Bicycle Usage and Temperature: Normalized Trends

The figure above shows the normalized trends of **u**, which represent bicycle usage, along with the corresponding temperatures over the time period. Both exhibit quadratic trends with considerable fluctuations over time. In general, bicycle usage is greater on warm days than on cold days. During spring (April through late May) and during fall and winter (after late September), bicycle usage exhibits a positive relationship with temperature; increases in temperature are accompanied by increases in usage, and decreases in temperature are accompanied by decreases in usage. In summer (mid-June through early September), a noteworthy pattern emerges: when temperatures reach very high lev-

els, many of the temperature peaks coincide with reductions in bicycle usage, consistent with intuitive expectations regarding the deterrent effect of extreme heat.

These findings suggest that the nonlinear model more accurately reflects real-world bicycle usage patterns and provides meaningful insights into how temperature influences demand. From an operational standpoint, this has practical implications for fleet management: bicycles can be strategically redistributed to match demand peaks during moderate weather and scaled down during periods of extreme temperatures, improving overall efficiency and service quality.

7 CONCLUSION AND RESEARCH OUTLOOK

In this dissertation, we developed a doubly stochastic model with covariates for replicated Poisson point processes and applied it to real-world data from the Divvy bicycle-sharing system. We incorporated temperature as a covariate in the component score function, which in turn reflects bicycle usage. By comparing linear and nonlinear forms of this function, we investigated the pattern by which temperature influences bicycle usage. Empirical findings show that while a linear model captures a general upward trend in bicycle usage with rising temperature, it fails to reflect behavioral changes at temperature extremes. The nonlinear model—implemented using a quadratic B-spline basis—better captures the demand sensitivity to both cold and hot conditions. Specifically, bicycle usage peaks in a moderate temperature range, and drops at both low and high extremes, aligning well with practical expectations about rider comfort and behavior.

These results highlight the value of incorporating nonlinear covariate structures into temporal point process models, especially when external factors influence behavior in complex ways. The ability to estimate smooth nonlinear functions offers interpretability with improved flexibility, making the method well-suited for real-world applications.

Overall, this dissertation provides a flexible, data-driven modeling framework that can be extended and refined to accommodate a wide variety of data patterns. However, this dissertation focuses on the effect of a single covariate—daily average temperature. Temperature, as a covariate, is relatively straightforward to interpret and verify intuitively, making it a natural starting point for exploring covariate effects. Future work may extend this approach in several important directions. First, incorporating multiple covariates (e.g., wind speed or humidity) may help build a more comprehensive understanding of the data. Second, introducing categorical or event-based covariates, such as "rainy day" versus "sunny day," which require grouping or interaction terms, may facilitate a more effective comparison of the data. Third, some other covariates may introduce more complex

effects such as periodic patterns, so the model may need to be adapted with more flexible structures to better capture these patterns.

This dissertation focuses on data from a single station which limits generalizability across the whole bicycle-sharing system, so applying the model to other stations may reveal additional spatial patterns beyond the temporal features explored here. Investigating spatial heterogeneity across multiple locations could support a spatio-temporal extension of the model.

REFERENCES

R.B. Ash, M.F. Gardner, Z.W. Birnbaum, and E. Lukacs. *Topics in Stochastic Processes: Probability and Mathematical Statistics: A Series of Monographs and Textbooks*. Probability and mathematical statistics. Elsevier Science, 2014. ISBN 9781483191430. URL https://books.google.com/books?id=z4biBQAAQBAJ.

B.A. Baddeley, R.A. Moyeed, C.V. Howard, and A. Boyde. Analysis of three-dimensional point pattern with replication. *Applied Statistics*, 42(4):641–668, 1993.

M.S. Bahadori, A.B. Gonçalves, and F. Moura. A systematic review of station location techniques for bicycle-sharing systems planning and operation. *ISPRS International Journal of Geo-Information*, 10(8), 2021. ISSN 2220-9964. doi: 10.3390/ijgi10080554. URL https://www.mdpi.com/2220-9964/10/8/554.

M.L. Bell and G.K. Grunwald. Mixed models for the analysis of replicated spatial point patterns. *Biostatistics*, 5(4):633–648, 2004.

D.R. Cox and V. Isham. *Point Processes*. Chapman and Hall/CRC, 1980. ISBN 0412219107.

C. de Boor. On calculating with b-splines. *Journal of Approximation Theory*, 6(1):50–62, 1972. ISSN 0021-9045. doi: https://doi.org/10.1016/0021-9045(72)90080-9. URL https://www.sciencedirect.com/science/article/pii/0021904572900809.

P. J. Diggle, S. J. Eglen, and J. B. Troy. *Modelling the Bivariate Spatial Distribution of Amacrine Cells*, pages 215–233. Case Studies in Spatial Point Process Modeling. Springer-Verlag, 2006.

P.J. Diggle. *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*. Chapman and Hall/CRC, 2013. ISBN 9781032477473.

P.J. Diggle, N. Lange, and F.M. Bene. Analysis of variance for replicated spatial point patterns in clinical neuroanatomy. *Journal of the American Statistical Association*, 86(415): 618–625, 1991.

P.J. Diggle, J. Mateu, and H. E. Clough. A comparison between parametric and non-parametric approaches to the analysis of replicated spatial point pattern. *Advances in Applied Probability*, 32(2):331–343, 2000.

D. Gervini. Independent component models for replicated point processes. *Spatial Statistics*, 18:474–488, 2016. ISSN 2211-6753.
URL:https://www.sciencedirect.com/science/article/pii/S2211675316300963.

D. Gervini. Multiplicative component models for replicated point processes, 2017. URL https://arxiv.org/abs/1705.09693.

D. Gervini. Doubly stochastic models for spatio-temporal covariation of replicated point processes. *Canadian Journal of Statistics*, 50(1):287–303, 2022. doi: https://doi.org/10.1002/cjs.11638. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cjs.11638.

D. Gervini and T. J. Baur. Joint models for grid point and response processes in longitudinal and functional data. *Statistica Sinica*, 30(4):1905–1924, 2020. ISSN 10170405, 19968507. URL https://www.jstor.org/stable/26969400.

D. Gervini and M. Khanal. Exploring patterns of demand in bike sharing systems via replicated point process models. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 68(3):585–602, 2018. ISSN 0035-9254. doi: 10.1111/rssc.12322. URL https://doi.org/10.1111/rssc.12322.

A. Jalilian, Y. Guan, and R. Waagepetersen. Decomposition of variance for spatial cox pro-

cesses. *Scandinavian Journal of Statistics*, 40(1):119–137, 2013. ISSN 03036898, 14679469. URL http://www.jstor.org/stable/23357256.

S. Landau, S. Rabe-Hesketh, and Ian P. Everall. Nonparametric one-way analysis of variance of replicated bivariate spatial point patterns. *Biometrical Journal*, 46(1):19–34, 2004. doi: https://doi.org/10.1002/bimj.200310010. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.200310010.

Y. Lei, J. Zhang, and Z. Ren. A study on bicycle-sharing dispatching station site selection and planning based on multivariate data. *Sustainability*, 15(17), 2023. ISSN 2071-1050. doi: 10.3390/su151713112. URL https://www.mdpi.com/2071-1050/15/17/13112.

J. Møller and R.P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press, 2003. ISBN 9780203496930. URL https://books.google.com/books?id=dBNOHvElXZ4C.

R. Nair and E. Miller-Hooks. Fleet management for vehicle sharing operations. *Transportation Science*, 45(4):524–540, 2011. doi: 10.1287/trsc.1100.0347. URL https://doi.org/10.1287/trsc.1100.0347.

Z. Pawlas. Estimation of summary characteristics from replicated spatial point processes. *Kybernetika*, 47(6):880–892, 2011. URL http://eudml.org/doc/197263.

J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer, 2005.

Y.H. Seo, D.K. Kim, S. Kang, Y.J. Byon, and S.Y. Kho. Rebalancing docked bicycle sharing system with approximate dynamic programming and reinforcement learning. *Journal of Advanced Transportation*, 2022(1):2780711, 2022. doi: https://doi.org/10.1155/2022/2780711. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/2780711.

S.A. Shaheen, S. Guzman, and H. Zhang. Bikesharing in europe, the americas, and asia: Past, present, and future. *Transportation Research Record*, 2143(1):159–167, 2010. doi: 10.3141/2143-20. URL https://doi.org/10.3141/2143-20.

B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. 1986. ISBN 9780412246203. URL https://books.google.com/books?id=e-xsrjsL7WkC.

D.L. Snyder and M.I. Miller. *Random Point Processes in Time and Space*. Springer Texts in Electrical Engineering. Springer New York, 2012. ISBN 9781461231660. URL https://books.google.com/books?id=c_3UBwAAQBAJ.

R.L. Streit. *Poisson Point Processes: Imaging, Tracking, and Sensing*. Springer US, 2010. ISBN 9781441969231. URL https://books.google.com/books?id=KAWmFYUJ5zsC.

C. G. Wager, B. A. Coull, and N. Lange. Modelling spatial intensity for replicated inhomogeneous point patterns in brain imaging. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(2):429–446, 2004. doi: https://doi.org/10.1046/j.1369-7412.2003.05285.x. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1046/j.1369-7412.2003.05285.x.

S. Wu, H.G. Müller, and Z. Zhang. Functional data analysis for point processes with rare events. *Statistica Sinica*, 23(1):1–23, 2013. ISSN 10170405, 19968507. URL http://www.jstor.org/stable/24310512.

APPENDIX

A.1 λ in Simple Linear Model

For $\lambda_{\varepsilon,z}(t) = \exp\left\{\mu(t) + \mathbf{u}^{\mathbf{T}}\boldsymbol{\phi}(t)\right\}$, with $\mathbf{u} = g(z) + \varepsilon$ in the simple linear case. We have: $u = dz + \varepsilon$

$$\begin{aligned}\lambda_{\varepsilon,z}(t) &= \exp\left\{\mu(t) + (g(z)+\varepsilon)^{\mathbf{T}}\boldsymbol{\phi}(t)\right\} \\ &= \exp\left\{\mu(t) + \sum_{k=1}^{p}(g_k(z)+\varepsilon_k)\phi_k(t)\right\} \\ &= \exp\left\{c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}\left(d_k z + \varepsilon_k\right)c_k^{\mathbf{T}}\beta(t)\right\}\end{aligned} \tag{A.1}$$

$$\log\lambda(t) = \mu(t) + \sum_{k=1}^{p}\left(g_k(z)+\varepsilon_k\right)^{\mathbf{T}}\phi_k(t) = c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}\left(d_k z + \varepsilon_k\right)c_k^{\mathbf{T}}\beta(t) \tag{A.2}$$

and

$$\begin{aligned}\log f(x \mid \varepsilon, z) &= -\int_B \lambda_{\varepsilon,z}(t)dt + \sum_{j=1}^{m}\log\lambda_{\varepsilon,z}\left(t_j\right) - \log m! \\ &= -\int_B \exp\left\{c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}\left(d_k z + \varepsilon_k\right)c_k^{\mathbf{T}}\beta(t)\right\} + \\ &\quad \{c_0^{\mathbf{T}} + \sum_{k=1}^{p}(d_k z + \varepsilon_k)c_k^{\mathbf{T}}\}\sum_{j=1}^{m}\beta(t_j) - \log m!\end{aligned} \tag{A.3}$$

A.2 Derivative of Coefficient d

$$\begin{aligned}\nabla_{\mathbf{d}} \log f(x \mid z) &= \frac{1}{f(x \mid z)} \int \{\nabla_{\mathbf{d}} f(x \mid \varepsilon, z)\} f(\varepsilon) d\varepsilon \\ &= \frac{1}{f(x \mid z)} \int \{\nabla_{\mathbf{d}} \log f(x \mid \varepsilon, z)\} f(x \mid \varepsilon, z) f(\varepsilon) d\varepsilon \\ &= \int \{\nabla_{\mathbf{d}} \log f(x \mid \varepsilon, z)\} f(\varepsilon \mid x) d\varepsilon \\ &= \mathbb{E}\{\nabla_{\mathbf{d}} \log f(x|\varepsilon, z)|x\}\end{aligned} \tag{A.4}$$

Since

$$\nabla_{\mathbf{d}} \log f(x|\varepsilon, z) = -\nabla_{\mathbf{d}} \int_B \lambda_{\varepsilon,z}(t) dt + \sum_{j=1}^{m} \nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t_j) \tag{A.5}$$

and

$$\begin{aligned}\nabla_{\mathbf{d}} \int_B \lambda_{\varepsilon,z}(t) dt &= \int_B \nabla_{\mathbf{d}} \lambda_{\varepsilon,z}(t) dt \\ &= \int_B \{\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t)\} \lambda_{\varepsilon,z}(t) dt\end{aligned} \tag{A.6}$$

we have

$$\nabla_{\mathbf{d}} \log f(x \mid \varepsilon, z) = -\int_B \{\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t)\} \lambda_{\varepsilon,z}(t) dt + \sum_{j=1}^{m} \nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t_j) \tag{A.7}$$

Then

$$\nabla_{\mathbf{d}} \log f(x \mid z) = -\int_B \mathbb{E}\left[\{\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t)\} \lambda_{\varepsilon,z}(t) \mid x\right] dt + \sum_{j=1}^{m} \mathbb{E}\left[\nabla_{\mathbf{d}} \log \lambda_{\varepsilon,z}(t_j) \mid x\right] \tag{A.8}$$

A.3 Coefficient Update Procedure under Simple Linear Model

The simple linear model is defined as: $u_k = g_k(z) + \varepsilon_k = d_k z + \varepsilon_k$

Update c_k:

$$\log \lambda(t) = \mu(t) + \sum_{k=1}^{p} \left(d_k z + \varepsilon_k\right)^{\mathbf{T}} \phi_k(t) = c_0^{\mathbf{T}} \beta(t) + \sum_{k=1}^{p} \left(d_k z + \varepsilon_k\right) c_k^{\mathbf{T}} \beta(t) \tag{A.9}$$

$$\nabla_{\mathbf{c}_k} \log \lambda_{\varepsilon,z}(t) = (d_k z + \varepsilon_k)\boldsymbol{\beta}(t), \quad \text{for } i = 1, \ldots, n. \tag{A.10}$$

$$\nabla_{\mathbf{c}_k} \ell_n(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{c}_k} \log f\left(x_i \mid z_i\right) - \xi_2 2\Omega \mathbf{c}_k \tag{A.11}$$

$$\begin{aligned}
\nabla_{\mathbf{c}_k} \ell_n(\boldsymbol{\theta}) =& -\frac{1}{n} \sum_{i=1}^{n} \int_B \mathbb{E}\left[\left\{\nabla_{\mathbf{c}_k} \log \lambda_{\varepsilon,z}(t)\right\} \lambda_{\varepsilon,z}(t) \mid x_i\right] dt \\
&+ \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \mathbb{E}\left[\nabla_{\mathbf{c}_k} \log \lambda_{\varepsilon,z}\left(t_{ij}\right) \mid x_i\right] - \xi_2 2\boldsymbol{\Omega}\mathbf{c}_k \\
=& -\frac{1}{n} \sum_{i=1}^{n} \int_B \mathbb{E}\left[\left\{(d_k z_i + \varepsilon_k)\boldsymbol{\beta}(t)\right\} \lambda_{\varepsilon,z}(t) \mid x_i\right] dt \\
&+ \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \mathbb{E}\left[(d_k z_i + \varepsilon_k)\boldsymbol{\beta}(t_{ij}) \mid x_i\right] - \xi_2 2\boldsymbol{\Omega}\mathbf{c}_k \\
=& -\frac{1}{n} \sum_{i=1}^{n} \int_B \boldsymbol{\beta}(t)\mathbb{E}\left[(d_k z_i + \varepsilon_k)\lambda_{\varepsilon,z}(t) \mid x_i\right] dt \\
&+ \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[(d_k z_i + \varepsilon_k) \mid x_i\right] \sum_{j=1}^{m_i} \boldsymbol{\beta}(t_{ij}) - \xi_2 2\Omega \mathbf{c}_k \\
=& -\frac{1}{n} \sum_{i=1}^{n} \int_B \boldsymbol{\beta}(t)\mathbb{E}\left[(d_k z_i + \varepsilon_k)\lambda_{\varepsilon,z}(t) \mid x_i\right] dt \\
&+ \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[(d_k z_i + \varepsilon_k) \mid x_i\right] \mathbf{b}_i - \xi_2 2\Omega \mathbf{c}_k
\end{aligned} \tag{A.12}$$

$$\begin{aligned}\nabla_{\mathbf{c}_k}\mathbb{E}\{(d_k z_i+\varepsilon_k)\lambda_{\varepsilon,z}(t)\mid x_i\} &\approx \mathbb{E}\{(d_k z_i+\varepsilon_k)\nabla_{\mathbf{c}_k}\lambda_{\varepsilon,z}(t)\mid x_i\}\\ &= \mathbb{E}[(d_k z_i+\varepsilon_k)\{\nabla_{\mathbf{c}_k}\log\lambda_{\varepsilon,z}(t)\}\lambda_{\varepsilon,z}(t)\mid x_i]\\ &= \mathbb{E}[(d_k z_i+\varepsilon_k)^2\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t)\mid x_i]\end{aligned} \tag{A.13}$$

$$\mathrm{H}_{\mathrm{c}_k}\ell_n(\boldsymbol{\theta}) \approx -\frac{1}{n}\sum_{i=1}^{n}\int_B \boldsymbol{\beta}(t)\boldsymbol{\beta}(t)^{\mathbf{T}}\mathbb{E}[(d_k z_i+\varepsilon_k)^2\lambda_{\varepsilon,z}(t)\mid x_i]dt - \xi_2 2\Omega \tag{A.14}$$

Since the components are aggregated sequentially, we update only the last component at each stage. Given that $c_1, \ldots, c_{k-1}$ have already been updated, let Γ be a $q \times \{q-(k-1)\}$ orthonormal basis of $\left\{\mathbf{J}_0 c_1^{new}, \ldots, \mathbf{J}_0 c_{k-1}^{new}\right\}^{\perp}$

$$\begin{aligned}\overline{\mathbf{c}}_k^{\text{new}} &= \Gamma\Gamma^{\mathbf{T}}\mathbf{c}_k^{\text{old}} - \Gamma\left\{\Gamma^{\mathbf{T}}\mathrm{H}_{\mathrm{c}_k}\ell_n\left(\boldsymbol{\theta}^{\text{old}}\right)\Gamma\right\}^{-1}\Gamma^{\mathbf{T}}\nabla_{\mathrm{c}_k}\ell_n\left(\boldsymbol{\theta}^{\text{old}}\right)\\ \mathbf{c}_k^{\text{new}} &= \overline{\mathbf{c}}_k^{\text{new}} \Big/ \left\{\left(\overline{\mathbf{c}}_k^{\text{new}}\right)^{\mathbf{T}}\mathbf{J}_0\overline{\mathbf{c}}_k^{\text{new}}\right\}^{1/2}\end{aligned} \tag{A.15}$$

Update d_k:

$$\log\lambda(t) = \mu(t) + \sum_{k=1}^{p}(d_k z+\varepsilon_k)^{\mathbf{T}}\phi_k(t) = c_0^{\mathbf{T}}\beta(t) + \sum_{k=1}^{p}(d_k z+\varepsilon_k)\, c_k^{\mathbf{T}}\beta(t) \tag{A.16}$$

$$\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}(t) = z_i c_k^{\mathbf{T}}\boldsymbol{\beta}(t) \tag{A.17}$$

$$\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{d}_k}\log f\left(x_i \mid z_i\right) - \xi_3 2\Delta\mathbf{d}_k \tag{A.18}$$

$$\begin{aligned}\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}) =& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[\left\{\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}(t)\right\}\lambda_{\varepsilon,z}(t)\mid x_i\right]dt\\ &+\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}\left(t_{ij}\right)\mid x_i\right] - \xi_3 2\Delta\mathbf{d}_k\\ =& -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[z_i c_k^{\mathbf{T}}\boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t)\mid x_i\right]dt + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_i}\mathbb{E}\left[z_i c_k^{\mathbf{T}}\boldsymbol{\beta}(t_{ij})\mid x_i\right]\end{aligned}$$

$$
\begin{aligned}
&- \xi_3 2\Delta \mathbf{d}_k \\
&= -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[z_i c_k^{\mathbf{T}} \boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i\right] dt + \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[z_i c_k^{\mathbf{T}} \mid x_i\right]\sum_{j=1}^{m_i}\boldsymbol{\beta}(t_{ij}) \\
&- \xi_3 2\Delta \mathbf{d}_k \\
&= -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}\left[z_i c_k^{\mathbf{T}} \boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i\right] dt + \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[z_i c_k^{\mathbf{T}} \mid x_i\right]\mathbf{b}_i - \xi_3 2\Delta \mathbf{d}_k
\end{aligned} \tag{A.19}
$$

$$
\begin{aligned}
\nabla_{\mathbf{d}_k}\mathbb{E}(z_i c_k^{\mathbf{T}} \boldsymbol{\beta}(t)\lambda_{\varepsilon,z}(t) \mid x_i) &\approx \mathbb{E}\{z_i c_k^{\mathbf{T}} \boldsymbol{\beta}(t)\nabla_{\mathbf{d}_k}\lambda_{\varepsilon,z}(t) \mid x_i\} \\
&= \mathbb{E}[z_i c_k^{\mathbf{T}} \boldsymbol{\beta}(t)\{\nabla_{\mathbf{d}_k}\log\lambda_{\varepsilon,z}(t)\}\lambda_{\varepsilon,z}(t) \mid x_i] \\
&= \mathbb{E}[(z_i)^2(\mathbf{c}_k^{\top}\boldsymbol{\beta}(t))^2\lambda_{\varepsilon,z}(t) \mid x_i]
\end{aligned} \tag{A.20}
$$

$$
\mathrm{H}_{\mathrm{d}_k}\ell_n(\boldsymbol{\theta}) \approx -\frac{1}{n}\sum_{i=1}^{n}\int_B \mathbb{E}[(z_i)^2(\mathbf{c}_k^{\top}\boldsymbol{\beta}(t))^2\lambda_{\varepsilon,z}(t) \mid x_i]dt - \xi_2 2\Delta \tag{A.21}
$$

Since d_k is a single value, we apply a standard Newton update:

$$
\mathbf{d}_k^{\text{new}} = \mathbf{d}_k^{\text{old}} - \{\mathrm{H}_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}^{\text{old}})\}^{-1}\nabla_{\mathbf{d}_k}\ell_n(\boldsymbol{\theta}^{\text{old}}) \tag{A.22}
$$

A.4 MATLAB Code for B-Spline Function

```
%save as bspl.m
function y = bspl(x,k,t,r)
if nargin<4
error('Not enough input arguments')
end
if size(t,1)>1
t = t';
end
m = length(x);
n = length(t);
y = zeros(m,n+k-2);
if r==0
tt = [t(1)*ones(1,k-1), t, t(n)*ones(1,k-1)];
n = length(tt);
b = zeros(1,k);
dr = zeros(1,k-1);
dl = zeros(1,k-1);
for l = 1:m
b(1) = 1;
i = find(tt<=x(l),1,'last');
if i==n, i = n-k; end
for j = 1:k-1
dr(j) = tt(i+j)-x(l);
dl(j) = x(l)-tt(i+1-j);
saved = 0;
for o = 1:j
term = b(o)/(dr(o)+dl(j+1-o));
b(o) = saved + dr(o)*term;
saved = dl(j+1-o)*term;
end
b(j+1) = saved;
end
y(l,i-k+1:i) = b;
end
else
tt = [repmat(t(1),1,k-2), t, repmat(t(n),1,k-2)];
B = bspl(x,k-1,t,r-1);
msp = ((k-1)./(ones(m,1)*(tt(k:n+2*(k-2))-tt(1:n+k-3)))).*B;
y(:,1) =  - msp(:,1);
y(:,2:n+k-3) = msp(:,1:n+k-4) - msp(:,2:n+k-3);
y(:,n+k-2) =  msp(:,n+k-3);
end
```

A.5 MATLAB Code for Simulating Linear Data

```
%save as simu_data_1.m
function [X_all, M_all, U_all, T_all, Mu_all, Phi_all,xFit] = simu_data_1(c, n, m, temperature,scale,v)
X_all = cell(1, m);
M_all = cell(1, m);
U_all = cell(1, m);
T_all = cell(1, m);
Mu_all = cell(1, m);
Phi_all = cell(1, m);
[ulin,~,xFit]= linear(n, scale, temperature,v);
for i = 1:m
[X, M, U,  T, Mu, Phi] = gen1(c, n,ulin);
X_all{i} = X;
M_all{i} = M;
U_all{i} = U;
T_all{i} = T;
Mu_all{i} = Mu;
Phi_all{i} = Phi;
end
end
function [x, m, u, t, mu, phi] = gen1(c, n,ulin)
t = linspace(0, 1, 300);
mu = sin(pi*t) + c;
phi = sqrt(2)*sin(pi*t);
u=ulin;
lmb = exp(ones(n, 1) * mu + u' * phi);
Ilmb = sum(lmb, 2) * (t(2) - t(1));
Mlmb = max(lmb, [], 2);
m = poissrnd(Ilmb);
x = cell(n, 1);
for i = 1:n
x{i} = zeros(m(i), 1);
if m(i) > 0
k = 0;
while k < m(i)
tt = rand(1);
y = Mlmb(i) * rand(1);
lmb_tt = interp1(t, lmb(i, :), tt);
if y <= lmb_tt
k = k + 1;
x{i}(k) = tt;
end
end
end
end
end
function [ulin, y,xFit] = linear(n, scale, temperature,v)
z = (temperature(:,1) - mean(temperature(:,1) )) / std(temperature(:,1));
xFit=linspace(min(z), max(z), n);
y = scale * xFit;
ulin = y-mean(y) +v * randn(1, n);
end
```

A.6 MATLAB Code for Simulating Quadratic Data

```
%save as simu_data_g.m
function [X_all, y_true,xCentered,M_all, U_all, T_all, Mu_all, Phi_all,coefficients] = simu_data_g(c, n, m, temp,a, v,std_target);
X_all = cell(1, m);
M_all = cell(1, m);
U_all = cell(1, m);
T_all = cell(1, m);
Mu_all = cell(1, m);
Phi_all = cell(1, m);
[uquad, y_true,xCentered,coefficients] = quad(n, temp,a, v, std_target);
for i = 1:m
[X, M, U,  T, Mu, Phi] = gen1(c, n,uquad);
X_all{i} = X;
M_all{i} = M;
U_all{i} = U;
T_all{i} = T;
Mu_all{i} = Mu;
Phi_all{i} = Phi;
end
end
function [x, m, u, t, mu, phi] = gen1(c, n,uquad)
t = linspace(0, 1, 300);
mu = sin(pi*t) + c;
phi = sqrt(2)*sin(pi*t);
u = uquad;
lmb = exp(ones(n, 1) * mu + u' * phi);
Ilmb = sum(lmb, 2) * (t(2) - t(1));
Mlmb = max(lmb, [], 2);
m = poissrnd(Ilmb);
x = cell(n, 1);
for i = 1:n
x{i} = zeros(m(i), 1);
if m(i) > 0
k = 0;
while k < m(i)
tt = rand(1);
y = Mlmb(i) * rand(1);
lmb_tt = interp1(t, lmb(i, :), tt);
if y <= lmb_tt
k = k + 1;
x{i}(k) = tt;
end
end
end
end
end

function [uquad, y_true, xCentered, coefficients] = quad(n, temp, a, v, std_target)
z = (temp(:,1) - mean(temp(:,1))) / std(temp(:,1));
x_min = min(z);
x_max = max(z);
x_center = (x_min + x_max) / 2;
xFit = linspace(x_min, x_max, n);
```

```
xCentered = xFit - x_center;
b = 0;
c = 0;
y_raw = a * xCentered.^2 + b * xCentered + c;
y_centered = y_raw - mean(y_raw);
current_std = std(y_centered);
y_true = y_centered * (std_target / current_std);
uquad = y_true + v * randn(1, n);
scale_factor = std_target / current_std;
coefficients = [a, b, c] * scale_factor;
end
```

A.7 MATLAB Code for Computing Simulated Linear Data

```
%need the scrpit: bspl.m
%save as linsimu.m
function [c0,C_j,D,s2,e,logf] = linsimu(x,z,d,basis,p,j,itmax,sm1,sm2,sm3)

n = length(x);
m = zeros(n,1);
a = basis.rng(1);
b = basis.rng(2);
for i = 1:n
x{i}(x{i}<a) = [];
x{i}(x{i}>b) = [];
m(i) = length(x{i});
end

q = basis.or + basis.nk;
t = linspace(a,b,300);
dt = t(2)-t(1);
knt = linspace(a,b,basis.nk+2);
B0 = bspl(t,basis.or,knt,0);
J0 = (B0'*B0)*dt;
if basis.or>2
B2 = bspl(t,basis.or,knt,2);
J2 = (B2'*B2)*dt;
else
J2 = zeros(q,q);
end
sumB = zeros(n,q);
for i = 1:n
B_i = bspl(x{i},basis.or,knt,0);
sumB(i,:) = sum(B_i,1);
end

z=(z-mean(z))';
c0 = log(mean(m)/(b-a))*ones(q,1);
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
disp('--->␣Computing␣mean')
disp(['Iteration:␣0,␣Pen.␣loglik:␣' num2str(OF)])
errC0 = 1;
total_iter = 0;
while errC0 >1e-3 && total_iter<itmax
total_iter = total_iter + 1;
c00 = c0;
OF0 = OF;
[gc0,Hc0] = derivc0(sumB,c0,dt,B0);
gpll = gc0-2*sm1*J2*c0;
Hpll = Hc0-2*sm1*J2;
direction = Hpll\gpll;
OF = -Inf;
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
```

```
c0 = c00-step*direction;
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
k = k+1;
end

if OF<=OF0 || ~all(isfinite(c0))
disp('No further improvement in obj. func. is possible')
c0 = c00;
OF = OF0;
end
errC0  = norm(c0-c00)/norm(c00);
disp(['Iteration: ' num2str(total_iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(errC0)])
end

C = zeros(q, p);
u = zeros(n, p);
D = d;
fprintf('Initial D: %.6f\n', D);

for ic = 1:p
if ic == 1
P = eye(q);
else
P = null(C(:,1:ic-1)' * J0);
end

C(:,ic) = (P * P') * ones(q,1);
C(:,ic) = C(:,ic) / sqrt(C(:,ic)' * J0 * C(:,ic));

if ic == 1
Ilmb0 = sum(exp(B0 * c0)) * dt;
u(:,ic) = sqrt(b - a) * log(max(m,1) / Ilmb0);
else
u(:,ic) = u(:,ic-1) / 2;
end
end

e=(u(:,j)-D*z)/10;
s2 = var(e);

errC = 1;
errD = 1;
max_C_iter = 10;
max_D_iter = 10;
min_C_iter = 5;
min_D_iter = 5;
C_j = C(:, j);

[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;

while (errC > 1e-4 || errD > 1e-4 || d_iter < min_D_iter) && total_iter < itmax
d_iter = 0;
while (errD > 1e-4 || d_iter < min_D_iter) && d_iter < max_D_iter && total_iter < itmax
```

```
total_iter = total_iter + 1;
d_iter = d_iter + 1;
d00 = D; e00 = e; e200 = e2;
OF0 = OF; k = 0;

[gd, Hd] = derivD(sumB, c0, C_j, D, dt, B0, z, e00);
gdp = gd - 2 * sm3 * D;
Hdp = Hd - 2 * sm3;
directionD = Hdp \ gdp;

if ~all(isfinite(directionD))
warning('Non-finite␣directionD␣encountered');
end

OF = -Inf;
eps_obj = 1e-8;
while (OF < OF0 - eps_obj) && k < 10
step = 0.7^k;
D = d00 - step * directionD;
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e00, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;
k = k + 1;
end

if OF <= OF0 || ~all(isfinite(D))
D = d00; e = e00; e2 = e200;
end

s2 = mean(e2, 1);
errD = norm(D - d00) / norm(d00);
fprintf('Iteration:␣%d,␣D␣Iteration:␣%d,OF:␣%.6f,␣Error␣D:␣%.6f\n', total_iter, d_iter, OF, errD);
end

c_iter = 0;
while (errC > 1e-4|| c_iter < min_C_iter) && c_iter < max_C_iter && total_iter < itmax
total_iter = total_iter + 1;
c_iter = c_iter + 1;
c00 = C_j; e00 = e; e200 = e2;
[gc, Hc] = derivC(sumB, c0, C_j, D, dt, B0, z, e00);
gcp = gc - 2 * sm2 * J2 * C_j;
Hcp = Hc - 2 * sm2 * J2;
directionC = P * ((P' * Hcp * P) \ (P' * gcp));

if ~all(isfinite(directionC(:)))
warning('Non-finite␣directionC␣encountered');
end

OF0 = OF; l = 0;
while OF <= OF0 && l < 10
step = 0.4^l;
C_j = c00 - step * directionC;
C_j = C_j / sqrt(C_j' * J0 * C_j);
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;
l = l + 1;
```

```
end

if OF <= OF0 || ~all(isfinite(C_j))
C_j = c00; e = e00; e2 = e200;
end

s2 = mean(e2, 1);
errC = norm(C_j - c00) / norm(c00);
fprintf('Iteration: %d, C Iteration: %d,OF: %.6f, Error C: %.6f\n', total_iter, c_iter, OF, errC);
end
end

if total_iter >= itmax
disp('Maximum number of iterations reached before convergence.');
end
fprintf('Final D estimate: %.6f\n', D);
end

function logf = complogf0(sumB,c0,dt,B0,m)
logf = -sum(exp(B0*c0))*dt + sumB*c0 - gammaln(m+1);
end

function [gc0,Hc0] = derivc0(sumB,c0,dt,B0)
q = size(B0,2);
Hc0 = -(B0'*((exp(B0*c0)*ones(1,q)).*B0))*dt;
gc0 = -B0'*exp(B0*c0)*dt + mean(sumB,1)';
end

function [e,e2,logf] =  compeff(sumB,c0,C_j,D,s2,dt,B0,m,e_ini,z)
[n,p] = size(e_ini);
Phi = B0*C_j;
logf = zeros(n,1);
e = e_ini;
e2 = e_ini.^2;
for i = 1:n
eL = e_ini(i);
uL = D*z(i) + eL;
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:5
eL = eL - D_gi/H_gi;
lmbi = exp(B0*c0+B0*C_j*uL);
D_gi = -lmbi'*Phi*dt + sumB(i,:)*C_j- eL/s2;
H_gi = -Phi' * (lmbi .* Phi) * dt - 1/s2;
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C_j*uL) - gammaln(m(i)+1) ...
-sum(eL.^2./(2*s2)) - .5*sum(log(2*pi*s2));
logf(i) = gi + (p/2)*log(2*pi) - 0.5*log(-H_gi);
S = -1/H_gi;
e(i) = eL;
e2(i) = S + eL^2;
end
end

function [gc,Hc] = derivC(sumB,c0,C_j,D,dt,B0,z,e)
n = length(e);
```

```
q = length(c0);
ng = size(B0,1);
u_j=D*z+e;
u_j2=u_j.^2;
lmbC = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
ulmbC = (ones(ng,1)*u_j').*lmbC;
u2lmbC = (ones(ng,1)*u_j2').*lmbC;
gc = (-B0'*mean(ulmbC,2))*dt + (sumB'*u_j/n);
Hc = -(B0'*((mean(u2lmbC,2)*ones(1,q)).*B0))*dt;
if ~all(isfinite(gc(:))) || ~all(isfinite(Hc(:)))
warning('Non-finite␣gradient␣or␣Hessian␣encountered.');
end
end

function [gd,Hd] = derivD(sumB,c0,C_j,D,dt,B0,z,e)
n = length(e);
lmbD = exp(B0*c0*ones(1,n)+B0*C_j*(D*z+e)');
gd = -(z'* ((B0*C_j)'*lmbD)') /n* dt + (z'* (sumB*C_j)/n);
Hd = -(z.^2'* (((B0*C_j).^2)' * lmbD)') * dt/n;
if ~all(isfinite(gd)) || ~all(isfinite(Hd))
warning('Non-finite␣gradient␣or␣Hessian␣encountered.');
end
end
```

A.8 MATLAB Code for Computing Simulated Quadratic Data

```
%need the scrpit: bspl.m
%save as quadsimu.m
function [c0,C_j,D,s2,e,logf,r1,z] = quadsimu(x,z,basis,p,j,itmax,sm1,sm2,sm3)
c0 = [];
D = [];
s2 = [];
e = [];
logf = [];

if ~iscell(x)
disp('Error: X must be cell array')
return
else
[mx,nx] = size(x);
if (mx>1 && nx>1)
disp('Error: X must be a one-dimensional cell array')
return
end
end

if ~isvector(z)
disp('Error: Z must be vector array')
return
else
[mt, nt] = size(z);
if (mt > 1 && nt > 1)
disp('Error: z must be a one-dimensional array')
return
else
if length(x) ~= length(z)
disp('Error: the dimension of X and Z must be the same ')
return
end
end
end

n = length(x);
m = zeros(n,1);
a = basis.rng(1);
b = basis.rng(2);
for i = 1:n
x{i}(x{i}<a) = [];
x{i}(x{i}>b) = [];
m(i) = length(x{i});
end
if any(m==0)
disp('Warning: Some x{i}s have no data within basis range')
end
if any(m>=200)
disp('Warning: Some x{i}s have more than 200 observations')
disp('This may cause Inf values in the likelihood function')
disp('This method is intended for relatively small x{i}s')
```

```
disp('For large x{i}s you can just use kernel smoothing')
end

q = basis.or + basis.nk;
t = linspace(a,b,300);
dt = t(2)-t(1);
knt = linspace(a,b,basis.nk+2);
B0 = bspl(t,basis.or,knt,0);
J0 = (B0'*B0)*dt;
if basis.or>2
B2 = bspl(t,basis.or,knt,2);
J2 = (B2'*B2)*dt;
else
J2 = zeros(q,q);
end
sumB = zeros(n,q);
for i = 1:n
B_i = bspl(x{i},basis.or,knt,0);
sumB(i,:) = sum(B_i,1);
end

minz=min(z);
maxz=max(z);
kntz = linspace(minz,maxz,2);
r0 = bspl(z',3,kntz,0);
r1 = r0 - mean(r0);
zz=linspace(minz,maxz,n);
dz=zz(2)-zz(1);
if basis.or>2
r2 = bspl(z,3,kntz,2);
r3 = r2-mean(r2);
R3 = (r3'*r3)*dz;
else
R3 = zeros(3,3);
end

c0 = log(mean(m)/(b-a))*ones(q,1);
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
disp('--->Computing mean')
disp(['Iteration: 0, Pen. loglik: ' num2str(OF)])
errC0 = 1;
iter = 0;
while errC0 >1e-3 && iter<itmax
iter = iter + 1;
c00 = c0;
OF0 = OF;
[gc0,Hc0] = derivc0(sumB,c0,dt,B0);
gpll = gc0-2*sm1*J2*c0;
Hpll = Hc0-2*sm1*J2;
direction = Hpll\gpll;
OF = -Inf;
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
c0 = c00-step*direction;
```

```
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
k = k+1;
end

if OF<=OF0 || ~all(isfinite(c0))
disp('No␣further␣improvement␣in␣obj.␣func.␣is␣possible')
c0 = c00;
OF = OF0;
end
errC0  = norm(c0-c00)/norm(c00);
disp(['Iteration:␣' num2str(iter) ',␣Pen.␣loglik:␣' ...
num2str(OF) ',␣Error:␣' num2str(errC0)])
end

C = zeros(q, p);
D = [0.1,0.1,0.1]' ;
u = zeros(n, p);

for ic = 1:p
if ic==1
P = eye(q);
else
P = null(C(:,1:ic-1)'*J0);
end

C(:,ic) = (P*P')*ones(q,1);
C(:,ic) = C(:,ic)/sqrt(C(:,ic)'*J0*C(:,ic));
if ic==1
Ilmb0 = sum(exp(B0*c0))*dt;
u(:,ic) = sqrt(b-a)*log(max(m,1)/Ilmb0);
else
u(:,ic) = u(:,ic-1)/2;
end
end

e = (u(:,j) - r1 * D)/20;
s2=var(e);
errC = 1;
errD = 1;
total_iter = 0;
max_C_iter = 5;
max_D_iter = 5;
C_j = C(:, j);

[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, r1);

OF = mean(logf)-sm1*c0'*J2*c0-sm2*C_j'*J2*C_j-sm3*D'*R3*D;
fprintf('log:%f\n', OF)
while (errC > 1e-4 || errD > 1e-4) && total_iter < itmax
d_iter = 0;
while errD > 1e-4 && d_iter < max_D_iter && total_iter < itmax
total_iter = total_iter + 1;
d_iter = d_iter + 1;
d00 = D; e00 = e; e200 = e2;
OF0 = OF;
```

```
k = 0;

[gd, Hd] = derivD(sumB, c0, C_j, dt, D, B0, r1, e00);
gdp = gd - 2 * sm3 * R3 * D;
Hdp = Hd - 2 * sm3 * R3 + 1e-4 * eye(size(Hd));
directionD = Hdp \ gdp;

OF = -Inf;
while OF <= OF0 && k < 10
step = 0.7^k;
D = d00 - step * directionD;
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e00, r1);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D' * R3 * D;
k = k + 1;
end

OF_drop = OF0 - OF;
if (OF_drop > 1e-4) && (norm(D - d00)/norm(d00) < 5e-4)
D = d00; e = e00; e2 = e200;
OF = OF0;
fprintf('D update rejected: OF dropped too much\n');
else
errD = norm(D - d00) / norm(d00);
end

s2 = mean(e2, 1)';
fprintf('D Iteration: %d, OF: %.6f, Error D: %.6f\n', d_iter, OF, errD)
end

c_iter = 0;
while errC > 1e-4 && c_iter < max_C_iter && total_iter < itmax
total_iter = total_iter + 1;
c_iter = c_iter + 1;

c00 = C_j; e00 = e; e200 = e2;
[gc, Hc] = derivC(sumB, c0, C_j, D, dt, B0, r1, e00);
gcp = gc - 2 * sm2 * J2 * C_j;
Hcp = Hc - 2 * sm2 * J2;
directionC = P * ((P' * Hcp * P) \ (P' * gcp));

OF0 = OF; l = 0;
while OF <= OF0 && l < 10
step = 0.7^l;
C_j = c00 - step * directionC;
C_j = C_j / sqrt(C_j' * J0 * C_j);
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, r1);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D' * R3 * D;
l = l + 1;
end

OF_drop = OF0 - OF;
if (OF_drop > 1e-4) && (norm(C_j - c00)/norm(c00) < 5e-4)
C_j = c00; e = e00; e2 = e200;
OF = OF0;
fprintf('C update rejected: OF dropped too much\n');
else
```

```
errC = norm(C_j - c00) / norm(c00);
end

s2 = mean(e2, 1)';
fprintf('C Iteration: %d, OF: %.6f, Error C: %.6f\n', c_iter, OF, errC)
end
end

if total_iter >= itmax
disp('Maximum number of iterations reached before convergence.');
end
end

function logf = complogf0(sumB,c0,dt,B0,m)
logf = -sum(exp(B0*c0))*dt + sumB*c0 - gammaln(m+1);
end

function [gc0,Hc0] = derivc0(sumB,c0,dt,B0)
q = size(B0,2);
Hc0 = -(B0'*((exp(B0*c0)*ones(1,q)).*B0))*dt;
gc0 = -B0'*exp(B0*c0)*dt + mean(sumB,1)';
end

function [e,e2,logf] =  compeff(sumB,c0,C_j,D,s2,dt,B0,m,e_ini,r1)
[n,p] = size(e_ini);
Phi = B0*C_j;
logf = zeros(n,1);
e = e_ini;
e2 = e_ini.^2;
for i = 1:n
eL = e_ini(i,:);
uL = r1(i,:) * D + eL;
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:10
eL = eL - D_gi/H_gi;
lmbi = exp(B0*c0+B0*C_j*uL');
D_gi = -lmbi'*Phi*dt + sumB(i,:)*C_j- eL./s2';
H_gi = -Phi' * (lmbi .* Phi) * dt - 1/s2;
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C_j*uL') - gammaln(m(i)+1) ...
-sum(eL.^2./(2*s2')) - .5*sum(log(2*pi*s2));
logf(i) = gi + (p/2)*log(2*pi) - 0.5*log(-H_gi);
S = -1/H_gi;
e(i,:) = eL;
e2(i,:) = S + eL^2;
end
end

function [gc,Hc] = derivC(sumB,c0,C_j,D,dt,B0,r1,e)
n = length(e);
q = length(c0);
ng = size(B0,1);
u_j=r1*D+e;
u_j2=u_j.^2;
lmbC = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
```

```
ulmbC = (ones(ng,1)*u_j').*lmbC;
u2lmbC = (ones(ng,1)*u_j2').*lmbC;
gc = (-B0'*mean(ulmbC,2))*dt + (sumB'*u_j/n);
Hc = -(B0'*((mean(u2lmbC,2)*ones(1,q)).*B0))*dt;
end

function [gd,Hd] = derivD(sumB,c0,C_j,dt,D,B0,r1,e)
n = length(e);
u_j=r1*D+e;
lmbD = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
gd = -((r1'*((B0*C_j)'*lmbD)')/n)*dt + (r1'*sumB*C_j)/n;
Hd = -(r1'*diag(mean(((B0*C_j).^2)'*lmbD,2))*r1)/n*dt;
end
```

A.9 MATLAB Code for Linear Model Error Estimation and Curve Fitting Results at Baseline Rate 10

```
%need the "temp2016_simu" variable in Rawdata.mat
%need the scrpit: bspl.m
%need linsimu.m
c1=log(5);
basis_simu = struct('rng', [0 1], 'or', 4, 'nk', 10);
t = linspace(0, 1, 300);
knt = linspace(0,1,basis_simu.nk+2);
B0 = bspl(t,basis_simu.or,knt,0);

[x_c1_91] = simu_data_l(c1, 91, 200, temp2016_simu,0.3,0.03);
xx_c1_91 = cell(91, 200);
for i = 1:200
for j = 1:91
xx_c1_91{j, i} = x_c1_91{i}{j};
end
end
numCols_91 = size(xx_c1_91, 2);
c0_c1_91 = cell(1, numCols_91);
C_c1_91 = cell(1, numCols_91);
D_c1_91 = zeros(1, numCols_91);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_91=linspace(min(z), max(z), 91);
for i = 1:numCols_91
x_col = xx_c1_91(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_91,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_91{i} = c0;
C_c1_91{i} = C_j;
D_c1_91(i) = D;
end

[x_c1_183] = simu_data_l(c1, 183, 200, temp2016_simu,0.3,0.03);
xx_c1_183 = cell(183, 200);
for i = 1:200
for j = 1:183
xx_c1_183{j, i} = x_c1_183{i}{j};
end
end
numCols_183 = size(xx_c1_183, 2);
c0_c1_183 = cell(1, numCols_183);
C_c1_183 = cell(1, numCols_183);
D_c1_183 = zeros(1, numCols_183);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_183=linspace(min(z), max(z), 183);
for i = 1:numCols_183
x_col = xx_c1_183(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_183,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_183{i} = c0;
C_c1_183{i} = C_j;
D_c1_183(i) = D;
end
```

```
[x_c1_366] = simu_data_l(c1, 366, 200, temp2016_simu,0.3,0.03);
xx_c1_366 = cell(366, 200);
for i = 1:200
for j = 1:366
xx_c1_366{j, i} = x_c1_366{i}{j};
end
end
numCols_366 = size(xx_c1_366, 2);
c0_c1_366 = cell(1, numCols_366);
C_c1_366 = cell(1, numCols_366);
D_c1_366 = zeros(1, numCols_366);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_366=linspace(min(z), max(z), 366);
for i = 1:numCols_366
x_col = xx_c1_366(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_366,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_366{i} = c0;
C_c1_366{i} = C_j;
D_c1_366(i) = D;
end

mu_true_c1 = sin(pi*t) + c1;
phi_true_c1 = sqrt(2) * sin(pi*t);

mu_simu_c1_91 = zeros(300, 200);
for i = 1:200
mu_simu_c1_91(:, i) = B0 * c0_c1_91{i};
end
mu_mean_c1_91=mean(mu_simu_c1_91, 2);

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c1_91, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (91)', 'Location', 'Best');
title('True vs Estimated \mu (91 Samples and rate=10)');

mu_simu_c1_183 = zeros(300, 200);
for i = 1:200
mu_simu_c1_183(:, i) = B0 * c0_c1_183{i};
end
mu_mean_c1_183=mean(mu_simu_c1_183, 2);

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c1_183, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (183)', 'Location', 'Best');
title('True vs Estimated \mu (183 Samples and rate=10)');

mu_simu_c1_366 = zeros(300, 200);
for i = 1:200
mu_simu_c1_366(:, i) = B0 * c0_c1_366{i};
end
mu_mean_c1_366=mean(mu_simu_c1_366, 2);
```

```
figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c1_366, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (366)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples and rate=10)');

phi_simu_c1_91 = zeros(300, 200);
for i = 1:200
phi_simu_c1_91(:, i) = B0 * C_c1_91{i};
end
phi_mean_c1_91=mean(phi_simu_c1_91 , 2);

figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c1_91, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (91)', 'Location', 'Best');
title('True vs Estimated \phi (91 Samples and rate=10)');

phi_simu_c1_183 = zeros(300, 200);
for i = 1:200
phi_simu_c1_183(:, i) = B0 * C_c1_183{i};
end
phi_mean_c1_183=mean(phi_simu_c1_183 , 2);

figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c1_183, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (183)', 'Location', 'Best');
title('True vs Estimated \phi (183 Samples and rate=10)');

phi_simu_c1_366 = zeros(300, 200);
for i = 1:200
phi_simu_c1_366(:, i) = B0 * C_c1_366{i};
end
phi_mean_c1_366=mean(phi_simu_c1_366 , 2);

figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c1_366, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (366)', 'Location', 'Best');
title('True vs Estimated \phi (366 Samples and rate=10)');

bias_mu_c1_91 = norm(mu_mean_c1_91 - mu_true_c1')/sqrt(numel(mu_mean_c1_91));
bias_mu_c1_183 = norm(mu_mean_c1_183 - mu_true_c1')/sqrt(numel(mu_mean_c1_183));
bias_mu_c1_366 = norm(mu_mean_c1_366 - mu_true_c1')/sqrt(numel(mu_mean_c1_366));

deviation_mu_c1_91 = mu_simu_c1_91 - mu_mean_c1_91;
std_mu_c1_91 = sqrt(mean(var(deviation_mu_c1_91, 0, 1)));

deviation_mu_c1_183 = mu_simu_c1_183 - mu_mean_c1_183;
std_mu_c1_183 = sqrt(mean(var(deviation_mu_c1_183, 0, 1)));
```

```
deviation_mu_c1_366 = mu_simu_c1_366 - mu_mean_c1_366;
std_mu_c1_366 = sqrt(mean(var(deviation_mu_c1_366, 0, 1)));

error_mu_c1_91=mu_simu_c1_91 - mu_true_c1';
rmse_mu_c1_91 =  norm(error_mu_c1_91, 'fro') / sqrt(numel(error_mu_c1_91));

error_mu_c1_183=mu_simu_c1_183 - mu_true_c1';
rmse_mu_c1_183 =  norm(error_mu_c1_183, 'fro') / sqrt(numel(error_mu_c1_183));

error_mu_c1_366=mu_simu_c1_366 - mu_true_c1';
rmse_mu_c1_366 =  norm(error_mu_c1_366, 'fro') / sqrt(numel(error_mu_c1_366));

shifted_phi_true_c1 = circshift(phi_true_c1, 1, 1);
bi_phi_true_c1 = phi_true_c1.* shifted_phi_true_c1;

shifted_phi_c1_91 = circshift(phi_simu_c1_91, 1, 1);
bi_phi_c1_91 = phi_simu_c1_91.* shifted_phi_c1_91;
mean_bi_phi_c1_91=mean(bi_phi_c1_91, 2);
bias_bi_phi_c1_91 = norm(mean_bi_phi_c1_91 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_c1_91));

shifted_phi_c1_183 = circshift(phi_simu_c1_183, 1, 1);
bi_phi_c1_183 = phi_simu_c1_183.* shifted_phi_c1_183;
mean_bi_phi_c1_183=mean(bi_phi_c1_183, 2);
bias_bi_phi_c1_183 = norm(mean_bi_phi_c1_183 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_c1_183));

shifted_phi_c1_366 = circshift(phi_simu_c1_366, 1, 1);
bi_phi_c1_366 = phi_simu_c1_366.* shifted_phi_c1_366;
mean_bi_phi_c1_366=mean(bi_phi_c1_366, 2);
bias_bi_phi_c1_366 = norm(mean_bi_phi_c1_366 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_c1_366));

deviation_phi_c1_91 = bi_phi_c1_91 - mean_bi_phi_c1_91;
std_phi_c1_91 = sqrt(mean(var(deviation_phi_c1_91, 0, 1)));

deviation_phi_c1_183 = bi_phi_c1_183 - mean_bi_phi_c1_183;
std_phi_c1_183 = sqrt(mean(var(deviation_phi_c1_183, 0, 1)));

deviation_phi_c1_366 = bi_phi_c1_366 - mean_bi_phi_c1_366;
std_phi_c1_366 = sqrt(mean(var(deviation_phi_c1_366, 0, 1)));

error_phi_c1_91=bi_phi_c1_91 - bi_phi_true_c1';
rmse_phi_c1_91 =  norm(error_phi_c1_91, 'fro') / sqrt(numel(error_phi_c1_91));

error_phi_c1_183=bi_phi_c1_183 - bi_phi_true_c1';
rmse_phi_c1_183 =  norm(error_phi_c1_183, 'fro') / sqrt(numel(error_phi_c1_183));

error_phi_c1_366=bi_phi_c1_366 - bi_phi_true_c1';
rmse_phi_c1_366 =  norm(error_phi_c1_366, 'fro') / sqrt(numel(error_phi_c1_366));

g_true_c1_91=0.3*xFit_91;
g_mean_c1_91=mean(D_c1_91)*xFit_91;
bias_g_c1_91 = mean(g_mean_c1_91 - g_true_c1_91)/ sqrt(numel(g_mean_c1_91));

g_true_c1_183=0.3*xFit_183;
g_mean_c1_183=mean(D_c1_183)*xFit_183;
bias_g_c1_183 = mean(g_mean_c1_183 - g_true_c1_183)/ sqrt(numel(g_mean_c1_183));
```

```
g_true_c1_366=0.3*xFit_366;
g_mean_c1_366=mean(D_c1_366)*xFit_366;
bias_g_c1_366 = mean(g_mean_c1_366 - g_true_c1_366)/ sqrt(numel(g_mean_c1_366));

g_simu_c1_91=D_c1_91'*xFit_91;
deviation_g_c1_91 = g_simu_c1_91 - g_mean_c1_91;
std_g_c1_91 = sqrt(mean(var(deviation_g_c1_91, 0, 1)));

g_simu_c1_183=D_c1_183'*xFit_183;
deviation_g_c1_183 = g_simu_c1_183 - g_mean_c1_183;
std_g_c1_183 = sqrt(mean(var(deviation_g_c1_183, 0, 1)));

g_simu_c1_366=D_c1_366'*xFit_366;
deviation_g_c1_366 = g_simu_c1_366 - g_mean_c1_366;
std_g_c1_366 = sqrt(mean(var(deviation_g_c1_366, 0, 1)));

error_g_c1_91=g_simu_c1_91 - g_true_c1_91;
rmse_g_c1_91 =  norm(error_g_c1_91, 'fro') / sqrt(numel(error_g_c1_91));

error_g_c1_183=g_simu_c1_183 - g_true_c1_183;
rmse_g_c1_183 =  norm(error_g_c1_183, 'fro') / sqrt(numel(error_g_c1_183));

error_g_c1_366=g_simu_c1_366 - g_true_c1_366;
rmse_g_c1_366 =  norm(error_g_c1_366, 'fro') / sqrt(numel(error_g_c1_366));

figure;
plot(xFit_91, g_true_c1_91, 'r', 'LineWidth', 2);
hold on;
plot(xFit_91, g_mean_c1_91, 'b', 'LineWidth', 2);
legend('True␣g', 'Mean␣Estimated␣g␣(91)', 'Location', 'Best');
title('True␣vs␣Estimated␣g␣(91␣Samples␣and␣rate=10)');

figure;
plot(xFit_183, g_true_c1_183, 'r', 'LineWidth', 2);
hold on;
plot(xFit_183, g_mean_c1_183, 'b', 'LineWidth', 2);
legend('True␣g', 'Mean␣Estimated␣g␣(183)', 'Location', 'Best');
title('True␣vs␣Estimated␣g␣(183␣Samples␣and␣rate=10)');

figure;
plot(xFit_366, g_true_c1_366, 'r', 'LineWidth', 2);
hold on;
plot(xFit_366, g_mean_c1_366, 'b', 'LineWidth', 2);
legend('True␣g', 'Mean␣Estimated␣g␣(366)', 'Location', 'Best');
title('True␣vs␣Estimated␣g␣(366␣Samples␣and␣rate=10)');

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c1_366 * 0.97, 'b', 'LineWidth', 2);
legend('True␣\mu', 'Estimated␣\mu␣(scaled␣to␣97%)', 'Location', 'Best');
title('True␣vs␣Estimated␣\mu␣(366␣Samples,␣rate=10,␣Estimated␣\mu␣rescaled␣to␣97%)');
```

A.10 MATLAB Code for Linear Model Error Estimation and Curve Fitting Results at Baseline Rate 30

```
%need the "temp2016_simu" variable in Rawdata.mat
%need the scrpit: bspl.m
%need linsimu.m
c2=log(15);
basis_simu = struct('rng', [0 1], 'or', 4, 'nk', 10);
t = linspace(0, 1, 300);
knt = linspace(0,1,basis_simu.nk+2);
B0 = bspl(t,basis_simu.or,knt,0);

[x_c2_91] = simu_data_l(c2, 91, 200, temp2016_simu,0.3,0.03);
xx_c2_91 = cell(91, 200);
for i = 1:200
for j = 1:91
xx_c2_91{j, i} = x_c2_91{i}{j};
end
end
numCols_91 = size(xx_c2_91, 2);
c0_c2_91 = cell(1, numCols_91);
C_c2_91 = cell(1, numCols_91);
D_c2_91 = zeros(1, numCols_91);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_91=linspace(min(z), max(z), 91);
for i = 1:numCols_91
x_col = xx_c2_91(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_91,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_91{i} = c0;
C_c2_91{i} = C_j;
D_c2_91(i) = D;
end

[x_c2_183] = simu_data_l(c2, 183, 200, temp2016_simu,0.3,0.03);
xx_c2_183 = cell(183, 200);
for i = 1:200
for j = 1:183
xx_c2_183{j, i} = x_c2_183{i}{j};
end
end
numCols_183 = size(xx_c2_183, 2);
c0_c2_183 = cell(1, numCols_183);
C_c2_183 = cell(1, numCols_183);
D_c2_183 = zeros(1, numCols_183);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_183=linspace(min(z), max(z), 183);
for i = 1:numCols_183
x_col = xx_c2_183(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_183,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_183{i} = c0;
C_c2_183{i} = C_j;
D_c2_183(i) = D;
end
```

```
[x_c2_366] = simu_data_l(c2, 366, 200, temp2016_simu,0.3,0.03);
xx_c2_366 = cell(366, 200);
for i = 1:200
for j = 1:366
xx_c2_366{j, i} = x_c2_366{i}{j};
end
end
numCols_366 = size(xx_c2_366, 2);
c0_c2_366 = cell(1, numCols_366);
C_c2_366 = cell(1, numCols_366);
D_c2_366 = zeros(1, numCols_366);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
xFit_366=linspace(min(z), max(z), 366);
for i = 1:numCols_366
x_col = xx_c2_366(:, i);
[c0,C_j,D,s2,e,logf] = linsimu(x_col,xFit_366,-0.1,basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_366{i} = c0;
C_c2_366{i} = C_j;
D_c2_366(i) = D;
end

mu_true_c2 = sin(pi*t) + c2;
phi_true_c2 = sqrt(2) * sin(pi*t);

mu_simu_c2_91 = zeros(300, 200);
for i = 1:200
mu_simu_c2_91(:, i) = B0 * c0_c2_91{i};
end
mu_mean_c2_91=mean(mu_simu_c2_91, 2);

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c2_91, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (91)', 'Location', 'Best');
title('True vs Estimated \mu (91 Samples and rate=30)');

mu_simu_c2_183 = zeros(300, 200);
for i = 1:200
mu_simu_c2_183(:, i) = B0 * c0_c2_183{i};
end
mu_mean_c2_183=mean(mu_simu_c2_183, 2);

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c2_183, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (183)', 'Location', 'Best');
title('True vs Estimated \mu (183 Samples and rate=30)');

mu_simu_c2_366 = zeros(300, 200);
for i = 1:200
mu_simu_c2_366(:, i) = B0 * c0_c2_366{i};
end
mu_mean_c2_366=mean(mu_simu_c2_366, 2);
```

```
figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c2_366, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (366)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples and rate=30)');

phi_simu_c2_91 = zeros(300, 200);
for i = 1:200
phi_simu_c2_91(:, i) = B0 * C_c2_91{i};
end
phi_mean_c2_91=mean(phi_simu_c2_91 , 2);

figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c2_91, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (183)', 'Location', 'Best');
title('True vs Estimated \phi (91 Samples and rate=30)');

phi_simu_c2_183 = zeros(300, 200);
for i = 1:200
phi_simu_c2_183(:, i) = B0 * C_c2_183{i};
end
phi_mean_c2_183=mean(phi_simu_c2_183 , 2);

figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c2_183, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (183)', 'Location', 'Best');
title('True vs Estimated \phi (183 Samples and rate=30)');

phi_simu_c2_366 = zeros(300, 200);
for i = 1:200
phi_simu_c2_366(:, i) = B0 * C_c2_366{i};
end
phi_mean_c2_366=mean(phi_simu_c2_366 , 2);

figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_mean_c2_366, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (366)', 'Location', 'Best');
title('True vs Estimated \phi (366 Samples and rate=30)');

bias_mu_c2_91 = norm(mu_mean_c2_91 - mu_true_c2')/sqrt(numel(mu_mean_c2_91));
bias_mu_c2_183 = norm(mu_mean_c2_183 - mu_true_c2')/sqrt(numel(mu_mean_c2_183));
bias_mu_c2_366 = norm(mu_mean_c2_366 - mu_true_c2')/sqrt(numel(mu_mean_c2_366));

deviation_mu_c2_91 = mu_simu_c2_91 - mu_mean_c2_91;
std_mu_c2_91 = sqrt(mean(var(deviation_mu_c2_91, 0, 1)));

deviation_mu_c2_183 = mu_simu_c2_183 - mu_mean_c2_183;
std_mu_c2_183 = sqrt(mean(var(deviation_mu_c2_183, 0, 1)));
```

```
deviation_mu_c2_366 = mu_simu_c2_366 - mu_mean_c2_366;
std_mu_c2_366 = sqrt(mean(var(deviation_mu_c2_366, 0, 1)));

error_mu_c2_91=mu_simu_c2_91 - mu_true_c2';
rmse_mu_c2_91 =  norm(error_mu_c2_91, 'fro') / sqrt(numel(error_mu_c2_91));

error_mu_c2_183=mu_simu_c2_183 - mu_true_c2';
rmse_mu_c2_183 =  norm(error_mu_c2_183, 'fro') / sqrt(numel(error_mu_c2_183));

error_mu_c2_366=mu_simu_c2_366 - mu_true_c2';
rmse_mu_c2_366 =  norm(error_mu_c2_366, 'fro') / sqrt(numel(error_mu_c2_366));

shifted_phi_true_c2 = circshift(phi_true_c2, 1, 1);
bi_phi_true_c2 = phi_true_c2.* shifted_phi_true_c2;

shifted_phi_c2_91 = circshift(phi_simu_c2_91, 1, 1);
bi_phi_c2_91 = phi_simu_c2_91.* shifted_phi_c2_91;
mean_bi_phi_c2_91=mean(bi_phi_c2_91, 2);
bias_bi_phi_c2_91 = norm(mean_bi_phi_c2_91 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_c2_91));

shifted_phi_c2_183 = circshift(phi_simu_c2_183, 1, 1);
bi_phi_c2_183 = phi_simu_c2_183.* shifted_phi_c2_183;
mean_bi_phi_c2_183=mean(bi_phi_c2_183, 2);
bias_bi_phi_c2_183 = norm(mean_bi_phi_c2_183 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_c2_183));

shifted_phi_c2_366 = circshift(phi_simu_c2_366, 1, 1);
bi_phi_c2_366 = phi_simu_c2_366.* shifted_phi_c2_366;
mean_bi_phi_c2_366=mean(bi_phi_c2_366, 2);
bias_bi_phi_c2_366 = norm(mean_bi_phi_c2_366 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_c2_366));

deviation_phi_c2_91 = bi_phi_c2_91 - mean_bi_phi_c2_91;
std_phi_c2_91 = sqrt(mean(var(deviation_phi_c2_91, 0, 1)));

deviation_phi_c2_183 = bi_phi_c2_183 - mean_bi_phi_c2_183;
std_phi_c2_183 = sqrt(mean(var(deviation_phi_c2_183, 0, 1)));

deviation_phi_c2_366 = bi_phi_c2_366 - mean_bi_phi_c2_366;
std_phi_c2_366 = sqrt(mean(var(deviation_phi_c2_366, 0, 1)));

error_phi_c2_91=bi_phi_c2_91 - bi_phi_true_c2';
rmse_phi_c2_91 =  norm(error_phi_c2_91, 'fro') / sqrt(numel(error_phi_c2_91));

error_phi_c2_183=bi_phi_c2_183 - bi_phi_true_c2';
rmse_phi_c2_183 =  norm(error_phi_c2_183, 'fro') / sqrt(numel(error_phi_c2_183));

error_phi_c2_366=bi_phi_c2_366 - bi_phi_true_c2';
rmse_phi_c2_366 =  norm(error_phi_c2_366, 'fro') / sqrt(numel(error_phi_c2_366));

g_true_c2_91=0.3*xFit_91;
g_mean_c2_91=mean(D_c2_91)*xFit_91;
bias_g_c2_91 = mean(g_mean_c2_91 - g_true_c2_91)/ sqrt(numel(g_mean_c2_91));

g_true_c2_183=0.3*xFit_183;
g_mean_c2_183=mean(D_c2_183)*xFit_183;
bias_g_c2_183 = mean(g_mean_c2_183 - g_true_c2_183)/ sqrt(numel(g_mean_c2_183));
```

```
g_true_c2_366=0.3*xFit_366;
g_mean_c2_366=mean(D_c2_366)*xFit_366;
bias_g_c2_366 = mean(g_mean_c2_366 - g_true_c2_366)/ sqrt(numel(g_mean_c2_366));

g_simu_c2_91=D_c2_91'*xFit_91;
deviation_g_c2_91 = g_simu_c2_91 - g_mean_c2_91;
std_g_c2_91 = sqrt(mean(var(deviation_g_c2_91, 0, 1)));

g_simu_c2_183=D_c2_183'*xFit_183;
deviation_g_c2_183 = g_simu_c2_183 - g_mean_c2_183;
std_g_c2_183 = sqrt(mean(var(deviation_g_c2_183, 0, 1)));

g_simu_c2_366=D_c2_366'*xFit_366;
deviation_g_c2_366 = g_simu_c2_366 - g_mean_c2_366;
std_g_c2_366 = sqrt(mean(var(deviation_g_c2_366, 0, 1)));

error_g_c2_91=g_simu_c2_91 - g_true_c2_91;
rmse_g_c2_91 =  norm(error_g_c2_91, 'fro') / sqrt(numel(error_g_c2_91));

error_g_c2_183=g_simu_c2_183 - g_true_c2_183;
rmse_g_c2_183 =  norm(error_g_c2_183, 'fro') / sqrt(numel(error_g_c2_183));

error_g_c2_366=g_simu_c2_366 - g_true_c2_366;
rmse_g_c2_366 =  norm(error_g_c2_366, 'fro') / sqrt(numel(error_g_c2_366));

figure;
plot(xFit_91, g_true_c2_91, 'r', 'LineWidth', 2);
hold on;
plot(xFit_91, g_mean_c2_91, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (91)', 'Location', 'Best');
title('True vs Estimated g (91 Samples and rate=30)');

figure;
plot(xFit_183, g_true_c2_183, 'r', 'LineWidth', 2);
hold on;
plot(xFit_183, g_mean_c2_183, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (183)', 'Location', 'Best');
title('True vs Estimated g (183 Samples and rate=30)');

figure;
plot(xFit_366, g_true_c2_366, 'r', 'LineWidth', 2);
hold on;
plot(xFit_366, g_mean_c2_366, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (366)', 'Location', 'Best');
title('True vs Estimated g (366 Samples and rate=30)');

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_mean_c2_366 * 0.97, 'b', 'LineWidth', 2);
legend('True \mu', 'Estimated \mu (scaled to 97%)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples, rate=30, Estimated \mu rescaled to 97%)');
```

A.11 MATLAB Code for Quadratic Model Error Estimation and Curve Fitting Results at Baseline Rate 10

```
%need the "temp2016_simu" variable in Rawdata.mat
%need the scrpit: bspl.m
%need quadsimu.m
c1=log(5);
basis_simu = struct('rng', [0 1], 'or', 4, 'nk', 10);
t = linspace(0, 1, 300);
knt = linspace(0,1,basis_simu.nk+2);
B0 = bspl(t,basis_simu.or,knt,0);

[y_c1_91,y_true_c1_91,xCentered_c1_91] = simu_data_g(c1, 91, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c1_91 = cell(91, 200);
for i = 1:200
for j = 1:91
yy_c1_91{j, i} = y_c1_91{i}{j};
end
end
numCols_91 = size(yy_c1_91, 2);
c0_c1_91 = cell(1, numCols_91);
C_c1_91 = cell(1, numCols_91);
D_c1_91 = cell(1, numCols_91);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_91=linspace(min(z), max(z), 91);
for i = 1:numCols_91
y_col = yy_c1_91(:, i);
[c0,C_j,D,s2,e,logf,r1_c1_91] = quadsimu(y_col,yFit_91',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_91{i} = c0;
C_c1_91{i} = C_j;
D_c1_91{i} = D;
end

[y_c1_183,y_true_c1_183,xCentered_c1_183] = simu_data_g(c1, 183, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c1_183 = cell(183, 200);
for i = 1:200
for j = 1:183
yy_c1_183{j, i} = y_c1_183{i}{j};
end
end
numCols_183 = size(yy_c1_183, 2);
c0_c1_183 = cell(1, numCols_183);
C_c1_183 = cell(1, numCols_183);
D_c1_183 = cell(1, numCols_183);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_183=linspace(min(z), max(z), 183);
for i = 1:numCols_183
y_col = yy_c1_183(:, i);
[c0,C_j,D,s2,e,logf,r1_c1_183] = quadsimu(y_col,yFit_183',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_183{i} = c0;
C_c1_183{i} = C_j;
D_c1_183{i} = D;
end
```

```
[y_c1_366,y_true_c1_366,xCentered_c1_366] = simu_data_g(c1, 366, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c1_366 = cell(366, 200);
for i = 1:200
for j = 1:366
yy_c1_366{j, i} = y_c1_366{i}{j};
end
end
numCols_366 = size(yy_c1_366, 2);
c0_c1_366 = cell(1, numCols_366);
C_c1_366 = cell(1, numCols_366);
D_c1_366 = cell(1, numCols_366);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_366=linspace(min(z), max(z), 366);
for i = 1:numCols_366
y_col = yy_c1_366(:, i);
[c0,C_j,D,s2,e,logf,r1_c1_366] = quadsimu(y_col,yFit_366',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c1_366{i} = c0;
C_c1_366{i} = C_j;
D_c1_366{i} = D;
end

mu_true_c1 = sin(pi*t) + c1;
phi_true_c1 = sqrt(2) * sin(pi*t);

mu_q_simu_c1_91 = zeros(300, 200);
for i = 1:200
mu_q_simu_c1_91(:, i) = B0 * c0_c1_91{i};
end
mu_q_mean_c1_91=mean(mu_q_simu_c1_91, 2);

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c1_91, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (91)', 'Location', 'Best');
title('True vs Estimated \mu (91 Samples and rate=10)');

mu_q_simu_c1_183 = zeros(300, 200);
for i = 1:200
mu_q_simu_c1_183(:, i) = B0 * c0_c1_183{i};
end
mu_q_mean_c1_183=mean(mu_q_simu_c1_183, 2);

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c1_183, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (183)', 'Location', 'Best');
title('True vs Estimated \mu (183 Samples and rate=10)');

mu_q_simu_c1_366 = zeros(300, 200);
for i = 1:200
mu_q_simu_c1_366(:, i) = B0 * c0_c1_366{i};
end
mu_q_mean_c1_366=mean(mu_q_simu_c1_366, 2);
```

```
figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c1_366, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (366)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples and rate=10)');

phi_q_simu_c1_91 = zeros(300, 200);
for i = 1:200
phi_q_simu_c1_91(:, i) = B0 * C_c1_91{i};
end
phi_q_mean_c1_91=mean(phi_q_simu_c1_91 , 2);

figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c1_91, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (91)', 'Location', 'Best');
title('True vs Estimated \phi (91 Samples and rate=10)');

phi_q_simu_c1_183 = zeros(300, 200);
for i = 1:200
phi_q_simu_c1_183(:, i) = B0 * C_c1_183{i};
end
phi_q_mean_c1_183=mean(phi_q_simu_c1_183 , 2);

figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c1_183, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (183)', 'Location', 'Best');
title('True vs Estimated \phi (183 Samples and rate=10)');

phi_q_simu_c1_366 = zeros(300, 200);
for i = 1:200
phi_q_simu_c1_366(:, i) = B0 * C_c1_366{i};
end
phi_q_mean_c1_366=mean(phi_q_simu_c1_366 , 2);
figure;
plot(t, phi_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c1_366, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (366)', 'Location', 'Best');
title('True vs Estimated \phi (366 Samples and rate=10)');

%mu
bias_mu_q_c1_91 = norm(mu_q_mean_c1_91 - mu_true_c1')/ sqrt(numel(mu_q_mean_c1_91));
bias_mu_q_c1_183 = norm(mu_q_mean_c1_183 - mu_true_c1')/ sqrt(numel(mu_q_mean_c1_183));
bias_mu_q_c1_366 = norm(mu_q_mean_c1_366 - mu_true_c1')/ sqrt(numel(mu_q_mean_c1_366));

deviation_mu_q_c1_91 = mu_q_simu_c1_91 - mu_q_mean_c1_91;
std_mu_q_c1_91 = sqrt(mean(var(deviation_mu_q_c1_91, 0, 1)));
deviation_mu_q_c1_183 = mu_q_simu_c1_183 - mu_q_mean_c1_183;
std_mu_q_c1_183 = sqrt(mean(var(deviation_mu_q_c1_183, 0, 1)));
deviation_mu_q_c1_366 = mu_q_simu_c1_366 - mu_q_mean_c1_366;
```

```
std_mu_q_c1_366 = sqrt(mean(var(deviation_mu_q_c1_366, 0, 1)));

error_mu_q_c1_91=mu_q_simu_c1_91 - mu_true_c1';
rmse_mu_q_c1_91 = norm(error_mu_q_c1_91, 'fro') / sqrt(numel(error_mu_q_c1_91));
error_mu_q_c1_183=mu_q_simu_c1_183 - mu_true_c1';
rmse_mu_q_c1_183 = norm(error_mu_q_c1_183, 'fro') / sqrt(numel(error_mu_q_c1_183));
error_mu_q_c1_366=mu_q_simu_c1_366 - mu_true_c1';
rmse_mu_q_c1_366 = norm(error_mu_q_c1_366, 'fro') / sqrt(numel(error_mu_q_c1_366));

%phi
shifted_phi_true_c1 = circshift(phi_true_c1, 1, 1);
bi_phi_true_c1 = phi_true_c1.* shifted_phi_true_c1;

shifted_phi_q_c1_91 = circshift(phi_q_simu_c1_91, 1, 1);
bi_phi_q_c1_91 = phi_q_simu_c1_91.* shifted_phi_q_c1_91;
mean_bi_phi_q_c1_91=mean(bi_phi_q_c1_91, 2);
bias_bi_phi_q_c1_91 = norm(mean_bi_phi_q_c1_91 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_q_c1_91));
shifted_phi_q_c1_183 = circshift(phi_q_simu_c1_183, 1, 1);
bi_phi_q_c1_183 = phi_q_simu_c1_183.* shifted_phi_q_c1_183;
mean_bi_phi_q_c1_183=mean(bi_phi_q_c1_183, 2);
bias_bi_phi_q_c1_183 = norm(mean_bi_phi_q_c1_183 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_q_c1_183));
shifted_phi_q_c1_366 = circshift(phi_q_simu_c1_366, 1, 1);
bi_phi_q_c1_366 = phi_q_simu_c1_366.* shifted_phi_q_c1_366;
mean_bi_phi_q_c1_366=mean(bi_phi_q_c1_366, 2);
bias_bi_phi_q_c1_366 = norm(mean_bi_phi_q_c1_366 - bi_phi_true_c1')/ sqrt(numel(mean_bi_phi_q_c1_366));

deviation_phi_q_c1_91 = bi_phi_q_c1_91 - mean_bi_phi_q_c1_91;
std_phi_q_c1_91 = sqrt(mean(var(deviation_phi_q_c1_91, 0, 1)));
deviation_phi_q_c1_183 = bi_phi_q_c1_183 - mean_bi_phi_q_c1_183;
std_phi_q_c1_183 = sqrt(mean(var(deviation_phi_q_c1_183, 0, 1)));
deviation_phi_q_c1_366 = bi_phi_q_c1_366 - mean_bi_phi_q_c1_366;
std_phi_q_c1_366 = sqrt(mean(var(deviation_phi_q_c1_366, 0, 1)));

error_phi_q_c1_91=bi_phi_q_c1_91 - bi_phi_true_c1';
rmse_phi_q_c1_91 = norm(error_phi_q_c1_91, 'fro') / sqrt(numel(error_phi_q_c1_91));
error_phi_q_c1_183=bi_phi_q_c1_183 - bi_phi_true_c1';
rmse_phi_q_c1_183 = norm(error_phi_q_c1_183, 'fro') / sqrt(numel(error_phi_q_c1_183));
error_phi_q_c1_366=bi_phi_q_c1_366 - bi_phi_true_c1';
rmse_phi_q_c1_366 = norm(error_phi_q_c1_366, 'fro') / sqrt(numel(error_phi_q_c1_366));

%g
g_q_true_c1_91=y_true_c1_91;
mean_D_q_c1_91 = mean(cell2mat(D_c1_91), 2);
g_q_mean_c1_91=r1_c1_91*mean_D_q_c1_91;
bias_g_q_c1_91 = norm(g_q_mean_c1_91 - g_q_true_c1_91')/ sqrt(numel(g_q_mean_c1_91));
g_q_true_c1_183=y_true_c1_183;
mean_D_q_c1_183 = mean(cell2mat(D_c1_183), 2);
g_q_mean_c1_183=r1_c1_183*mean_D_q_c1_183;
bias_g_q_c1_183 = norm(g_q_mean_c1_183 - g_q_true_c1_183')/ sqrt(numel(g_q_mean_c1_183));
g_q_true_c1_366=y_true_c1_366;
mean_D_q_c1_366 = mean(cell2mat(D_c1_366), 2);
g_q_mean_c1_366=r1_c1_366*mean_D_q_c1_366;
bias_g_q_c1_366 = norm(g_q_mean_c1_366 - g_q_true_c1_366')/ sqrt(numel(g_q_mean_c1_366));

g_q_simu_c1_91 = r1_c1_91*cell2mat(D_c1_91);
deviation_g_q_c1_91 = g_q_simu_c1_91 - g_q_mean_c1_91;
```

```
std_g_q_c1_91 = sqrt(mean(var(deviation_g_q_c1_91, 0, 1)));
g_q_simu_c1_183 = r1_c1_183*cell2mat(D_c1_183);
deviation_g_q_c1_183 = g_q_simu_c1_183 - g_q_mean_c1_183;
std_g_q_c1_183 = sqrt(mean(var(deviation_g_q_c1_183, 0, 1)));
g_q_simu_c1_366 = r1_c1_366*cell2mat(D_c1_366);
deviation_g_q_c1_366 = g_q_simu_c1_366 - g_q_mean_c1_366;
std_g_q_c1_366 = sqrt(mean(var(deviation_g_q_c1_366, 0, 1)));

error_g_q_c1_91=g_q_simu_c1_91 - g_q_true_c1_91';
rmse_g_q_c1_91 = norm(error_g_q_c1_91, 'fro') / sqrt(numel(error_g_q_c1_91));
error_g_q_c1_183=g_q_simu_c1_183 - g_q_true_c1_183';
rmse_g_q_c1_183 = norm(error_g_q_c1_183, 'fro') / sqrt(numel(error_g_q_c1_183));
error_g_q_c1_366=g_q_simu_c1_366 - g_q_true_c1_366';
rmse_g_q_c1_366 = norm(error_g_q_c1_366, 'fro') / sqrt(numel(error_g_q_c1_366));

figure;
plot(xCentered_c1_91, g_q_true_c1_91, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c1_91, g_q_mean_c1_91, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (91)', 'Location', 'Best');
title('True vs Estimated g (91 Samples and rate=10)');

figure;
plot(xCentered_c1_183, g_q_true_c1_183, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c1_183, g_q_mean_c1_183, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (183)', 'Location', 'Best');
title('True vs Estimated g (183 Samples and rate=10)');

figure;
plot(xCentered_c1_366, g_q_true_c1_366, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c1_366, g_q_mean_c1_366, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (366)', 'Location', 'Best');
title('True vs Estimated g (366 Samples and rate=10)');

figure;
plot(t, mu_true_c1, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c1_366 * 0.98, 'b', 'LineWidth', 2);
legend('True \mu', 'Estimated \mu (scaled to 98%)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples, rate=10, Estimated \mu rescaled to 98%)');
```

A.12 MATLAB Code for Quadratic Model Error Estimation and Curve Fitting Results at Baseline Rate 30

```
%need the "temp2016_simu" variable in Rawdata.mat
%need the scrpit: bspl.m
%need quadsimu.m
c2=log(15);
basis_simu = struct('rng', [0 1], 'or', 4, 'nk', 10);
t = linspace(0, 1, 300);
knt = linspace(0,1,basis_simu.nk+2);
B0 = bspl(t,basis_simu.or,knt,0);

[y_c2_91,y_true_c2_91,xCentered_c2_91] = simu_data_g(c2, 91, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c2_91 = cell(91, 200);
for i = 1:200
for j = 1:91
yy_c2_91{j, i} = y_c2_91{i}{j};
end
end
numCols_91 = size(yy_c2_91, 2);
c0_c2_91 = cell(1, numCols_91);
C_c2_91 = cell(1, numCols_91);
D_c2_91 = cell(1, numCols_91);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_91=linspace(min(z), max(z), 91);
for i = 1:numCols_91
y_col = yy_c2_91(:, i);
[c0,C_j,D,s2,e,logf,r1_c2_91] = quadsimu(y_col,yFit_91',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_91{i} = c0;
C_c2_91{i} = C_j;
D_c2_91{i} = D;
end

[y_c2_183,y_true_c2_183,xCentered_c2_183] = simu_data_g(c2, 183, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c2_183 = cell(183, 200);
for i = 1:200
for j = 1:183
yy_c2_183{j, i} = y_c2_183{i}{j};
end
end
numCols_183 = size(yy_c2_183, 2);
c0_c2_183 = cell(1, numCols_183);
C_c2_183 = cell(1, numCols_183);
D_c2_183 = cell(1, numCols_183);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_183=linspace(min(z), max(z), 183);
for i = 1:numCols_183
y_col = yy_c2_183(:, i);
[c0,C_j,D,s2,e,logf,r1_c2_183] = quadsimu(y_col,yFit_183',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_183{i} = c0;
C_c2_183{i} = C_j;
D_c2_183{i} = D;
end
```

```
[y_c2_366,y_true_c2_366,xCentered_c2_366] = simu_data_g(c2, 366, 200, temp2016_simu,0.1, 0.03,0.3);
yy_c2_366 = cell(366, 200);
for i = 1:200
for j = 1:366
yy_c2_366{j, i} = y_c2_366{i}{j};
end
end
numCols_366 = size(yy_c2_366, 2);
c0_c2_366 = cell(1, numCols_366);
C_c2_366 = cell(1, numCols_366);
D_c2_366 = cell(1, numCols_366);
z = (temp2016_simu - mean(temp2016_simu)) / std(temp2016_simu);
yFit_366=linspace(min(z), max(z), 366);
for i = 1:numCols_366
y_col = yy_c2_366(:, i);
[c0,C_j,D,s2,e,logf,r1_c2_366] = quadsimu(y_col,yFit_366',basis_simu,1,1,100,0.001,0.0001,0.00001);
c0_c2_366{i} = c0;
C_c2_366{i} = C_j;
D_c2_366{i} = D;
end

mu_true_c2 = sin(pi*t) + c2;
phi_true_c2 = sqrt(2) * sin(pi*t);

mu_q_simu_c2_91 = zeros(300, 200);
for i = 1:200
mu_q_simu_c2_91(:, i) = B0 * c0_c2_91{i};
end
mu_q_mean_c2_91=mean(mu_q_simu_c2_91, 2);

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c2_91, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (91)', 'Location', 'Best');
title('True vs Estimated \mu (91 Samples and rate=30)');

mu_q_simu_c2_183 = zeros(300, 200);
for i = 1:200
mu_q_simu_c2_183(:, i) = B0 * c0_c2_183{i};
end
mu_q_mean_c2_183=mean(mu_q_simu_c2_183, 2);

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c2_183, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (183)', 'Location', 'Best');
title('True vs Estimated \mu (183 Samples and rate=30)');

mu_q_simu_c2_366 = zeros(300, 200);
for i = 1:200
mu_q_simu_c2_366(:, i) = B0 * c0_c2_366{i};
end
mu_q_mean_c2_366=mean(mu_q_simu_c2_366, 2);
```

```
figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c2_366, 'b', 'LineWidth', 2);
legend('True \mu', 'Mean Estimated \mu (366)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples and rate=30)');

phi_q_simu_c2_91 = zeros(300, 200);
for i = 1:200
phi_q_simu_c2_91(:, i) = B0 * C_c2_91{i};
end
phi_q_mean_c2_91=mean(phi_q_simu_c2_91 , 2);

figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c2_91, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (91)', 'Location', 'Best');
title('True vs Estimated \phi (91 Samples and rate=30)');

phi_q_simu_c2_183 = zeros(300, 200);
for i = 1:200
phi_q_simu_c2_183(:, i) = B0 * C_c2_183{i};
end
phi_q_mean_c2_183=mean(phi_q_simu_c2_183 , 2);

figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c2_183, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (183)', 'Location', 'Best');
title('True vs Estimated \phi (183 Samples and rate=30)');

phi_q_simu_c2_366 = zeros(300, 200);
for i = 1:200
phi_q_simu_c2_366(:, i) = B0 * C_c2_366{i};
end
phi_q_mean_c2_366=mean(phi_q_simu_c2_366 , 2);
figure;
plot(t, phi_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, phi_q_mean_c2_366, 'b', 'LineWidth', 2);
legend('True \phi', 'Mean Estimated \phi (366)', 'Location', 'Best');
title('True vs Estimated \phi (366 Samples and rate=30)');

%mu
bias_mu_q_c2_91 = norm(mu_q_mean_c2_91 - mu_true_c2')/ sqrt(numel(mu_q_mean_c2_91));
bias_mu_q_c2_183 = norm(mu_q_mean_c2_183 - mu_true_c2')/ sqrt(numel(mu_q_mean_c2_183));
bias_mu_q_c2_366 = norm(mu_q_mean_c2_366 - mu_true_c2')/ sqrt(numel(mu_q_mean_c2_366));

deviation_mu_q_c2_91 = mu_q_simu_c2_91 - mu_q_mean_c2_91;
std_mu_q_c2_91 = sqrt(mean(var(deviation_mu_q_c2_91, 0, 1)));
deviation_mu_q_c2_183 = mu_q_simu_c2_183 - mu_q_mean_c2_183;
std_mu_q_c2_183 = sqrt(mean(var(deviation_mu_q_c2_183, 0, 1)));
deviation_mu_q_c2_366 = mu_q_simu_c2_366 - mu_q_mean_c2_366;
```

```
std_mu_q_c2_366 = sqrt(mean(var(deviation_mu_q_c2_366, 0, 1)));

error_mu_q_c2_91=mu_q_simu_c2_91 - mu_true_c2';
rmse_mu_q_c2_91 = norm(error_mu_q_c2_91, 'fro') / sqrt(numel(error_mu_q_c2_91));
error_mu_q_c2_183=mu_q_simu_c2_183 - mu_true_c2';
rmse_mu_q_c2_183 = norm(error_mu_q_c2_183, 'fro') / sqrt(numel(error_mu_q_c2_183));
error_mu_q_c2_366=mu_q_simu_c2_366 - mu_true_c2';
rmse_mu_q_c2_366 = norm(error_mu_q_c2_366, 'fro') / sqrt(numel(error_mu_q_c2_366));

%phi
shifted_phi_true_c2 = circshift(phi_true_c2, 1, 1);
bi_phi_true_c2 = phi_true_c2.* shifted_phi_true_c2;

shifted_phi_q_c2_91 = circshift(phi_q_simu_c2_91, 1, 1);
bi_phi_q_c2_91 = phi_q_simu_c2_91.* shifted_phi_q_c2_91;
mean_bi_phi_q_c2_91=mean(bi_phi_q_c2_91, 2);
bias_bi_phi_q_c2_91 = norm(mean_bi_phi_q_c2_91 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_q_c2_91));
shifted_phi_q_c2_183 = circshift(phi_q_simu_c2_183, 1, 1);
bi_phi_q_c2_183 = phi_q_simu_c2_183.* shifted_phi_q_c2_183;
mean_bi_phi_q_c2_183=mean(bi_phi_q_c2_183, 2);
bias_bi_phi_q_c2_183 = norm(mean_bi_phi_q_c2_183 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_q_c2_183));
shifted_phi_q_c2_366 = circshift(phi_q_simu_c2_366, 1, 1);
bi_phi_q_c2_366 = phi_q_simu_c2_366.* shifted_phi_q_c2_366;
mean_bi_phi_q_c2_366=mean(bi_phi_q_c2_366, 2);
bias_bi_phi_q_c2_366 = norm(mean_bi_phi_q_c2_366 - bi_phi_true_c2')/ sqrt(numel(mean_bi_phi_q_c2_366));

deviation_phi_q_c2_91 = bi_phi_q_c2_91 - mean_bi_phi_q_c2_91;
std_phi_q_c2_91 = sqrt(mean(var(deviation_phi_q_c2_91, 0, 1)));
deviation_phi_q_c2_183 = bi_phi_q_c2_183 - mean_bi_phi_q_c2_183;
std_phi_q_c2_183 = sqrt(mean(var(deviation_phi_q_c2_183, 0, 1)));
deviation_phi_q_c2_366 = bi_phi_q_c2_366 - mean_bi_phi_q_c2_366;
std_phi_q_c2_366 = sqrt(mean(var(deviation_phi_q_c2_366, 0, 1)));

error_phi_q_c2_91=bi_phi_q_c2_91 - bi_phi_true_c2';
rmse_phi_q_c2_91 = norm(error_phi_q_c2_91, 'fro') / sqrt(numel(error_phi_q_c2_91));
error_phi_q_c2_183=bi_phi_q_c2_183 - bi_phi_true_c2';
rmse_phi_q_c2_183 = norm(error_phi_q_c2_183, 'fro') / sqrt(numel(error_phi_q_c2_183));
error_phi_q_c2_366=bi_phi_q_c2_366 - bi_phi_true_c2';
rmse_phi_q_c2_366 = norm(error_phi_q_c2_366, 'fro') / sqrt(numel(error_phi_q_c2_366));

%g
g_q_true_c2_91=y_true_c2_91;
mean_D_q_c2_91 = mean(cell2mat(D_c2_91), 2);
g_q_mean_c2_91=r1_c2_91*mean_D_q_c2_91;
bias_g_q_c2_91 = norm(g_q_mean_c2_91 - g_q_true_c2_91')/ sqrt(numel(g_q_mean_c2_91));
g_q_true_c2_183=y_true_c2_183;
mean_D_q_c2_183 = mean(cell2mat(D_c2_183), 2);
g_q_mean_c2_183=r1_c2_183*mean_D_q_c2_183;
bias_g_q_c2_183 = norm(g_q_mean_c2_183 - g_q_true_c2_183')/ sqrt(numel(g_q_mean_c2_183));
g_q_true_c2_366=y_true_c2_366;
mean_D_q_c2_366 = mean(cell2mat(D_c2_366), 2);
g_q_mean_c2_366=r1_c2_366*mean_D_q_c2_366;
bias_g_q_c2_366 = norm(g_q_mean_c2_366 - g_q_true_c2_366')/ sqrt(numel(g_q_mean_c2_366));

g_q_simu_c2_91 = r1_c2_91*cell2mat(D_c2_91);
deviation_g_q_c2_91 = g_q_simu_c2_91 - g_q_mean_c2_91;
```

```
std_g_q_c2_91 = sqrt(mean(var(deviation_g_q_c2_91, 0, 1)));
g_q_simu_c2_183 = r1_c2_183*cell2mat(D_c2_183);
deviation_g_q_c2_183 = g_q_simu_c2_183 - g_q_mean_c2_183;
std_g_q_c2_183 = sqrt(mean(var(deviation_g_q_c2_183, 0, 1)));
g_q_simu_c2_366 = r1_c2_366*cell2mat(D_c2_366);
deviation_g_q_c2_366 = g_q_simu_c2_366 - g_q_mean_c2_366;
std_g_q_c2_366 = sqrt(mean(var(deviation_g_q_c2_366, 0, 1)));

error_g_q_c2_91=g_q_simu_c2_91 - g_q_true_c2_91';
rmse_g_q_c2_91 = norm(error_g_q_c2_91, 'fro') / sqrt(numel(error_g_q_c2_91));
error_g_q_c2_183=g_q_simu_c2_183 - g_q_true_c2_183';
rmse_g_q_c2_183 = norm(error_g_q_c2_183, 'fro') / sqrt(numel(error_g_q_c2_183));
error_g_q_c2_366=g_q_simu_c2_366 - g_q_true_c2_366';
rmse_g_q_c2_366 = norm(error_g_q_c2_366, 'fro') / sqrt(numel(error_g_q_c2_366));

figure;
plot(xCentered_c2_91, g_q_true_c2_91, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c2_91, g_q_mean_c2_91, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (91)', 'Location', 'Best');
title('True vs Estimated g (91 Samples and rate=30)');

figure;
plot(xCentered_c2_183, g_q_true_c2_183, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c2_183, g_q_mean_c2_183, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (183)', 'Location', 'Best');
title('True vs Estimated g (183 Samples and rate=30)');

figure;
plot(xCentered_c2_366, g_q_true_c2_366, 'r', 'LineWidth', 2);
hold on;
plot(xCentered_c2_366, g_q_mean_c2_366, 'b', 'LineWidth', 2);
legend('True g', 'Mean Estimated g (366)', 'Location', 'Best');
title('True vs Estimated g (366 Samples and rate=30)');

figure;
plot(t, mu_true_c2, 'r', 'LineWidth', 2);
hold on;
plot(t, mu_q_mean_c2_366 * 0.98, 'b', 'LineWidth', 2);
legend('True \mu', 'Estimated \mu (scaled to 98%)', 'Location', 'Best');
title('True vs Estimated \mu (366 Samples, rate=30, Estimated \mu rescaled to 98%)');
```

A.13 MATLAB Code for Computing Real Data

```
%need the scrpit: bspl.m
%save as quadreal.m
function [c0,C_j,D,s2,e,logf,r1,z,B0] = quadreal(x,z,basis,p,j,itmax,sm1,sm2,sm3)
c0 = [];
D = [];
s2 = [];
e = [];
logf = [];

if ~iscell(x)
disp('Error: X must be cell array')
return
else
[mx,nx] = size(x);
if (mx>1 && nx>1)
disp('Error: X must be a one-dimensional cell array')
return
end
end

if ~isvector(z)
disp('Error: Z must be vector array')
return
else
[mt, nt] = size(z);
if (mt > 1 && nt > 1)
disp('Error: z must be a one-dimensional array')
return
else
if length(x) ~= length(z)
disp('Error: the dimension of X and Z must be the same ')
return
end
end
end

n = length(x);
m = zeros(n,1);
a = basis.rng(1);
b = basis.rng(2);
for i = 1:n
x{i}(x{i}<a) = [];
x{i}(x{i}>b) = [];
m(i) = length(x{i});
end
if any(m==0)
disp('Warning: Some x{i}s have no data within basis range')
end
if any(m>=200)
disp('Warning: Some x{i}s have more than 200 observations')
disp('This may cause Inf values in the likelihood function')
disp('This method is intended for relatively small x{i}s')
```

```
disp('For large x{i}s you can just use kernel smoothing')
end

q = basis.or + basis.nk;
t = linspace(a,b,300);
dt = t(2)-t(1);
knt = linspace(a,b,basis.nk+2);
B0 = bspl(t,basis.or,knt,0);
J0 = (B0'*B0)*dt;
if basis.or>2
B2 = bspl(t,basis.or,knt,2);
J2 = (B2'*B2)*dt;
else
J2 = zeros(q,q);
end
sumB = zeros(n,q);
for i = 1:n
B_i = bspl(x{i},basis.or,knt,0);
sumB(i,:) = sum(B_i,1);
end

z = (z - mean(z)) / std(z);
minz=min(z);
maxz=max(z);
kntz = linspace(minz,maxz,2);
%kntz = linspace(minz,maxz,7);%For 5 knots
r0 = bspl(z',3,kntz,0);
%r0 = bspl(z',4,kntz,0); %For 5 knots
r1 = r0 - mean(r0);
zz=linspace(minz,maxz,n);
dz=zz(2)-zz(1);
if basis.or>2
r2 = bspl(z,3,kntz,2);
%r2 = bspl(z,4,kntz,2);%For 5 knots
r3 = r2-mean(r2);
R3 = (r3'*r3)*dz;
else
R3 = zeros(3,3);
end

c0 = log(mean(m)/(b-a))*ones(q,1);
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
disp('---> Computing mean')
disp(['Iteration: 0, Pen. loglik: ' num2str(OF)])
errC0 = 1;
total_iter= 0;
while errC0 >1e-3 && total_iter<itmax
total_iter = total_iter + 1;
c00 = c0;
OF0 = OF;
[gc0,Hc0] = derivc0(sumB,c0,dt,B0);
gpll = gc0-2*sm1*J2*c0;
Hpll = Hc0-2*sm1*J2;
direction = Hpll\gpll;
OF = -Inf;
```

```
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
c0 = c00-step*direction;
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
k = k+1;
end

if OF<=OF0 || ~all(isfinite(c0))
disp('No␣further␣improvement␣in␣obj.␣func.␣is␣possible')
c0 = c00;
OF = OF0;
end
errC0  = norm(c0-c00)/norm(c00);
disp(['Iteration:␣' num2str(total_iter) ',␣Pen.␣loglik:␣' ...
num2str(OF) ',␣Error:␣' num2str(errC0)])
end

C = zeros(q, p);
D = [0.1,0.1,0.1]' ;
%D = 0.1*ones(9,1);  %For 5 knots
u = zeros(n, p);

for ic = 1:p
if ic==1
P = eye(q);
else
P = null(C(:,1:ic-1)'*J0);
end

C(:,ic) = (P*P')*ones(q,1);
C(:,ic) = C(:,ic)/sqrt(C(:,ic)'*J0*C(:,ic));
if ic==1
Ilmb0 = sum(exp(B0*c0))*dt;
u(:,ic) = sqrt(b-a)*log(max(m,1)/Ilmb0);
else
u(:,ic) = u(:,ic-1)/2;
end
end

e = (u(:,j) - r1 * D)/20;
s2=var(e);
errC = 1;
errD = 1;
max_C_iter = 5;
max_D_iter = 5;
C_j = C(:, j);

[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, r1);

OF = mean(logf)-sm1*c0'*J2*c0-sm2*C_j'*J2*C_j-sm3*D'*R3*D;
while (errC > 1e-4 || errD > 1e-4) && total_iter < itmax
d_iter = 0;
while errD > 1e-4 && d_iter < max_D_iter && total_iter < itmax
total_iter = total_iter + 1;
```

```
d_iter = d_iter + 1;
d00 = D; e00 = e; e200 = e2;
OF0 = OF;
k = 0;

[gd, Hd] = derivD(sumB, c0, C_j, dt, D, B0, r1, e00);
gdp = gd - 2 * sm3 * R3 * D;
Hdp = Hd - 2 * sm3 * R3 + 1e-4 * eye(size(Hd));
directionD = Hdp \ gdp;

OF = -Inf;
while OF <= OF0 && k < 10
step = 0.7^k;
D = d00 - step * directionD;
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e00, r1);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D' * R3 * D;
k = k + 1;
end

OF_drop = OF0 - OF;
if (OF_drop > 1e-4) && (norm(D - d00)/norm(d00) < 5e-4)
D = d00; e = e00; e2 = e200;
OF = OF0;
fprintf('D update rejected: OF dropped too much\n');
else
errD = norm(D - d00) / norm(d00);
end

s2 = mean(e2, 1)';
fprintf('Iteration: %d, D Iteration: %d,Pen. loglik: %.6f, Error D: %.6f\n', total_iter, d_iter, OF, errD)
end

c_iter = 0;
while errC > 1e-4 && c_iter < max_C_iter && total_iter < itmax
total_iter = total_iter + 1;
c_iter = c_iter + 1;

c00 = C_j; e00 = e; e200 = e2;
[gc, Hc] = derivC(sumB, c0, C_j, D, dt, B0, r1, e00);
gcp = gc - 2 * sm2 * J2 * C_j;
Hcp = Hc - 2 * sm2 * J2;
directionC = P * ((P' * Hcp * P) \ (P' * gcp));

OF0 = OF; l = 0;
while OF <= OF0 && l < 10
step = 0.7^l;
C_j = c00 - step * directionC;
C_j = C_j / sqrt(C_j' * J0 * C_j);
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, r1);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D' * R3 * D;
l = l + 1;
end

OF_drop = OF0 - OF;
if (OF_drop > 1e-4) && (norm(C_j - c00)/norm(c00) < 5e-4)
C_j = c00; e = e00; e2 = e200;
```

```
OF = OF0;
fprintf('C update rejected: OF dropped too much\n');
else
errC = norm(C_j - c00) / norm(c00);
end

s2 = mean(e2, 1)';
fprintf('Iteration: %d, C Iteration: %d,Pen. loglik: %.6f, Error C: %.6f\n', total_iter, c_iter, OF, errC)
end
end

if total_iter >= itmax
disp('Maximum number of iterations reached before convergence.');
end
end

function logf = complogf0(sumB,c0,dt,B0,m)
logf = -sum(exp(B0*c0))*dt + sumB*c0 - gammaln(m+1);
end

function [gc0,Hc0] = derivc0(sumB,c0,dt,B0)
q = size(B0,2);
Hc0 = -(B0'*((exp(B0*c0)*ones(1,q)).*B0))*dt;
gc0 = -B0'*exp(B0*c0)*dt + mean(sumB,1)';
end

function [e,e2,logf] =  compeff(sumB,c0,C_j,D,s2,dt,B0,m,e_ini,r1)
[n,p] = size(e_ini);
Phi = B0*C_j;
logf = zeros(n,1);
e = e_ini;
e2 = e_ini.^2;
for i = 1:n
eL = e_ini(i,:);
uL = r1(i,:) * D + eL;
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:10
eL = eL - D_gi/H_gi;
lmbi = exp(B0*c0+B0*C_j*uL');
D_gi = -lmbi'*Phi*dt + sumB(i,:)*C_j- eL./s2';
H_gi = -Phi' * (lmbi .* Phi) * dt - 1/s2;
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C_j*uL') - gammaln(m(i)+1) ...
-sum(eL.^2./(2*s2')) - .5*sum(log(2*pi*s2));
logf(i) = gi + (p/2)*log(2*pi) - 0.5*log(-H_gi);
S = -1/H_gi;
e(i,:) = eL;
e2(i,:) = S + eL^2;
end
end

function [gc,Hc] = derivC(sumB,c0,C_j,D,dt,B0,r1,e)
n = length(e);
q = length(c0);
ng = size(B0,1);
```

```
u_j=r1*D+e;
u_j2=u_j.^2;
lmbC = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
ulmbC = (ones(ng,1)*u_j').*lmbC;
u2lmbC = (ones(ng,1)*u_j2').*lmbC;
gc = (-B0'*mean(ulmbC,2))*dt + (sumB'*u_j/n);
Hc = -(B0'*((mean(u2lmbC,2)*ones(1,q)).*B0))*dt;
end

function [gd,Hd] = derivD(sumB,c0,C_j,dt,D,B0,r1,e)
n = length(e);
u_j=r1*D+e;
lmbD = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
gd = -((r1'*((B0*C_j)'*lmbD)')/n)*dt + (r1'*sumB*C_j)/n;
Hd = -(r1'*diag(mean(((B0*C_j).^2)'*lmbD,2))*r1)/n*dt;
end

%need the scrpit: bspl.m
%save as linreal.m
function [c0,C_j,D,s2,e,logf,z,B0] = linreal(x,z,d,basis,p,j,itmax,sm1,sm2,sm3)
c0 = [];
D = [];
s2 = [];
e = [];
logf = [];

if ~iscell(x)
disp('Error: X must be cell array')
return
else
[mx,nx] = size(x);
if (mx>1 && nx>1)
disp('Error: X must be a one-dimensional cell array')
return
end
end

if ~isvector(z)
disp('Error: Z must be vector array')
return
else
[mt, nt] = size(z);
if (mt > 1 && nt > 1)
disp('Error: z must be a one-dimensional array')
return
else
if length(x) ~= length(z)
disp('Error: the dimension of X and Z must be the same')
return
end
end
end
n = length(x);
m = zeros(n,1);
a = basis.rng(1);
b = basis.rng(2);
```

```
for i = 1:n
x{i}(x{i}<a) = [];
x{i}(x{i}>b) = [];
m(i) = length(x{i});
end

q = basis.or + basis.nk;
t = linspace(a,b,300);
dt = t(2)-t(1);
knt = linspace(a,b,basis.nk+2);
B0 = bspl(t,basis.or,knt,0);
J0 = (B0'*B0)*dt;
if basis.or>2
B2 = bspl(t,basis.or,knt,2);
J2 = (B2'*B2)*dt;
else
J2 = zeros(q,q);
end
sumB = zeros(n,q);
for i = 1:n
B_i = bspl(x{i},basis.or,knt,0);
sumB(i,:) = sum(B_i,1);
end

%z=(z-mean(z))' in simulation
z=(z-mean(z))'/std(z);
c0 = log(mean(m)/(b-a))*ones(q,1);
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
disp('--->␣Computing␣mean')
disp(['Iteration:␣0,␣Pen.␣loglik:␣' num2str(OF)])
errC0 = 1;
total_iter = 0;
while errC0 >1e-3 && total_iter<itmax
total_iter = total_iter + 1;
c00 = c0;
OF0 = OF;
[gc0,Hc0] = derivc0(sumB,c0,dt,B0);
gpll = gc0-2*sm1*J2*c0;
Hpll = Hc0-2*sm1*J2;
direction = Hpll\gpll;
OF = -Inf;
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
c0 = c00-step*direction;
logf = complogf0(sumB,c0,dt,B0,m);
OF = mean(logf)-sm1*c0'*J2*c0;
k = k+1;
end

if OF<=OF0 || ~all(isfinite(c0))
disp('No␣further␣improvement␣in␣obj.␣func.␣is␣possible')
c0 = c00;
OF = OF0;
end
```

```
errC0  = norm(c0-c00)/norm(c00);
disp(['Iteration: ' num2str(total_iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(errC0)])
end

C = zeros(q, p);
u = zeros(n, p);
D = d;

for ic = 1:p
if ic == 1
P = eye(q);
else
P = null(C(:,1:ic-1)' * J0);
end

C(:,ic) = (P * P') * ones(q,1);
C(:,ic) = C(:,ic) / sqrt(C(:,ic)' * J0 * C(:,ic));

if ic == 1
Ilmb0 = sum(exp(B0 * c0)) * dt;
u(:,ic) = sqrt(b - a) * log(max(m,1) / Ilmb0);
else
u(:,ic) = u(:,ic-1) / 2;
end
end

e=(u(:,j)-D*z)/20;
s2 = var(e);

errC = 1;
errD = 1;
max_C_iter = 10;
max_D_iter = 10;
min_C_iter = 5;
min_D_iter = 5;

C_j = C(:, j);

[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;

while (errC > 1e-4 || errD > 1e-4 || d_iter < min_D_iter) && total_iter < itmax
d_iter = 0;
while (errD > 1e-4 || d_iter < min_D_iter) && d_iter < max_D_iter && total_iter < itmax

total_iter = total_iter + 1;
d_iter = d_iter + 1;
d00 = D; e00 = e; e200 = e2;
OF0 = OF; k = 0;

[gd, Hd] = derivD(sumB, c0, C_j, D, dt, B0, z, e00);
gdp = gd - 2 * sm3 * D;
Hdp = Hd - 2 * sm3;
directionD = Hdp \ gdp;
```

```
if ~all(isfinite(directionD))
warning('Non-finite directionD encountered');
end

OF = -Inf;
eps_obj = 1e-8;
while (OF < OF0 - eps_obj) && k < 10
step = 0.7^k;
D = d00 - step * directionD;
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e00, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;
k = k + 1;
end

if OF <= OF0 || ~all(isfinite(D))
D = d00; e = e00; e2 = e200;
end

s2 = mean(e2, 1);
errD = norm(D - d00) / norm(d00);
fprintf('Iteration: %d, D Iteration: %d,Pen. loglik: %.6f, Error D: %.6f\n', total_iter, d_iter, OF, errD);
end

c_iter = 0;
while (errC > 1e-4|| c_iter < min_C_iter) && c_iter < max_C_iter && total_iter < itmax
total_iter = total_iter + 1;
c_iter = c_iter + 1;
c00 = C_j; e00 = e; e200 = e2;
[gc, Hc] = derivC(sumB, c0, C_j, D, dt, B0, z, e00);
gcp = gc - 2 * sm2 * J2 * C_j;
Hcp = Hc - 2 * sm2 * J2;
directionC = P * ((P' * Hcp * P) \ (P' * gcp));

if ~all(isfinite(directionC(:)))
warning('Non-finite directionC encountered');
end

OF0 = OF; l = 0;
while OF <= OF0 && l < 10
step = 0.4^l;
C_j = c00 - step * directionC;
C_j = C_j / sqrt(C_j' * J0 * C_j);
[e, e2, logf] = compeff(sumB, c0, C_j, D, s2, dt, B0, m, e, z);
OF = mean(logf) - sm1 * c0' * J2 * c0 - sm2 * C_j' * J2 * C_j - sm3 * D^2;
l = l + 1;
end

if OF <= OF0 || ~all(isfinite(C_j))
C_j = c00; e = e00; e2 = e200;
end

s2 = mean(e2, 1);
errC = norm(C_j - c00) / norm(c00);
fprintf('Iteration: %d, C Iteration: %d,Pen. loglik: %.6f, Error C: %.6f\n', total_iter, c_iter, OF, errC);
end
end
```

```
if total_iter >= itmax
disp('Maximum number of iterations reached before convergence.');
end
%fprintf('Final D estimate: %.6f\n', D);
end

function logf = complogf0(sumB,c0,dt,B0,m)
logf = -sum(exp(B0*c0))*dt + sumB*c0 - gammaln(m+1);
end

function [gc0,Hc0] = derivc0(sumB,c0,dt,B0)
q = size(B0,2);
Hc0 = -(B0'*((exp(B0*c0)*ones(1,q)).*B0))*dt;
gc0 = -B0'*exp(B0*c0)*dt + mean(sumB,1)';
end

function [e,e2,logf] =  compeff(sumB,c0,C_j,D,s2,dt,B0,m,e_ini,z)
[n,p] = size(e_ini);
Phi = B0*C_j;
logf = zeros(n,1);
e = e_ini;
e2 = e_ini.^2;
for i = 1:n
eL = e_ini(i);
uL = D*z(i) + eL;
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:5
eL = eL - D_gi/H_gi;
lmbi = exp(B0*c0+B0*C_j*uL);
D_gi = -lmbi'*Phi*dt + sumB(i,:)*C_j- eL/s2;
H_gi = -Phi' * (lmbi .* Phi) * dt - 1/s2;
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C_j*uL) - gammaln(m(i)+1) ...
-sum(eL.^2./(2*s2)) - .5*sum(log(2*pi*s2));
logf(i) = gi + (p/2)*log(2*pi) - 0.5*log(-H_gi);
S = -1/H_gi;
e(i) = eL;
e2(i) = S + eL^2;
end
end

function [gc,Hc] = derivC(sumB,c0,C_j,D,dt,B0,z,e)
n = length(e);
q = length(c0);
ng = size(B0,1);
u_j=D*z+e;
u_j2=u_j.^2;
lmbC = exp(B0*c0*ones(1,n)+B0*C_j*u_j');
ulmbC = (ones(ng,1)*u_j').*lmbC;
u2lmbC = (ones(ng,1)*u_j2').*lmbC;
gc = (-B0'*mean(ulmbC,2))*dt + (sumB'*u_j/n);
Hc = -(B0'*((mean(u2lmbC,2)*ones(1,q)).*B0))*dt;
if ~all(isfinite(gc(:))) || ~all(isfinite(Hc(:)))
warning('Non-finite gradient or Hessian encountered.');
```

```
    end
    end

    function [gd,Hd] = derivD(sumB,c0,C_j,D,dt,B0,z,e)
    n = length(e);
    lmbD = exp(B0*c0*ones(1,n)+B0*C_j*(D*z+e)');
    gd = -(z'* ((B0*C_j)'*lmbD)') /n* dt + (z'* (sumB*C_j)/n);
    Hd = -(z.^2'* (((B0*C_j).^2)' * lmbD)') * dt/n;
    if ~all(isfinite(gd)) || ~all(isfinite(Hd))
    warning('Non-finite gradient or Hessian encountered.');
    end
    end

% Code to implement the above scripts and generate figures
% (save the following code in a new script separate from the scrpits above)
% Requires variables "x_real" and "temp2016_real" from Rawdata.mat
% Requires the script: bspl.m
% Modify the code in quadreal to handle the case with 5 interior knots
basis_real = struct('rng', [0 24], 'or', 4, 'nk', 10);
t = linspace(0, 24, 300);
knt = linspace(0,24,basis_real.nk+2);

%code for the quadratic models
[c0,C_j,D,s2,e,logf,r1,z,B0] = quadreal(x_real,temp2016_real(:,1)',basis_real,1,1,100,0.001,0.0001,0.00001);
plot(t,B0*c0)
legend('\mu');
xlabel('t');
title('\mu function from Divvy station 166 for quadratic model');

plot(t,B0*C_j)
legend('\phi');
xlabel('t');
title('\phi function from Divvy station 166 for quadratic model');

lmbplus = exp(B0*c0+2*B0*C_j);
lmbmin = exp(B0*c0-2*B0*C_j);

plot(t, exp(B0*c0), 'b-', t, lmbplus, 'r--', t, lmbmin, 'g:');
legend('exp(\mu)', 'exp(\mu+2*\phi)', 'exp(\mu-2*\phi)');
xlabel('t');
title('Daily Baseline Intensity Function with shift for Divvy station 166 (Quadratic)');

plot(temp2016_real(:,1), r1 * D, 'LineWidth', 1.5);
xlabel('Temperature');
ylabel('u');
title('Estimated Effect u vs. Temperature');
legend('u = d^T\gamma(z)', 'Location', 'best');

scatter(temp2016_real(:,1), r1 * D, 'LineWidth', 1.5);
xlabel('Temperature');
ylabel('u');
title('Estimated Effect u vs. Temperature');
legend('u = d^T\gamma(z)', 'Location', 'best');
ax = gca;
ax.Box = 'on';
ax.Layer = 'top';
```

```
u=r1*D;
u_norm = (u - min(u)) / (max(u) - min(u));
T_norm = (temp2016_real(:,1) - min(temp2016_real(:,1))) / ...
(max(temp2016_real(:,1)) - min(temp2016_real(:,1)));
n = length(u_norm);
startDate = datetime(2016,4,1);
xDates = startDate + caldays(0:n-1);
figure;
plot(xDates, u_norm, '-b', 'LineWidth', 1.5); hold on;
plot(xDates, T_norm, '-r', 'LineWidth', 1.5);
legend('u (normalized)', 'Temperature (normalized)','FontSize', 28);
%xlabel('Date');
title('Comparison of Normalized Bicycle Usage and Temperature Trends', 'FontSize', 28);
grid on;
xticks(startDate:calmonths(1):xDates(end));
xtickformat('MMM dd');
ax = gca;
ax.XAxis.FontSize = 24;

%code for the linear models
[c0,C_j,D,s2,e,logf,z,B0] = linreal(x_real,temp2016_real(:,1)',0.1,basis_real,1,1,100,0.001,0.0001,0.00001);
plot(t,B0*c0)
legend('\mu');
xlabel('t');
title('\mu function from Divvy station 166 for linear model');

plot(t,B0*C_j)
legend('\phi');
xlabel('t');
title('\phi function from Divvy station 166 for linear model');

lmbplus = exp(B0*c0+2*B0*C_j);
lmbmin = exp(B0*c0-2*B0*C_j);

plot(t, exp(B0*c0), 'b-', t, lmbplus, 'r--', t, lmbmin, 'g:');
legend('exp(\mu)', 'exp(\mu+2*\phi)', 'exp(\mu-2*\phi)');
xlabel('t');
title('Daily Baseline Intensity Function with shift for Divvy station 166 (Linear)');

z=(temp2016_real(:,1)-mean(temp2016_real(:,1)))'/std(temp2016_real(:,1));
plot(temp2016_real(:,1), D*z, 'LineWidth', 1.5);
xlabel('Temperature');
ylabel('u');
title('Estimated Effect u vs. Temperature');
legend('u = d*z ','Location', 'best');
```