May 2019

Outlier-Resistant Models for Doubly Stochastic Point Processes

Leo Stephan Elsaesser
University of Wisconsin-Milwaukee

OUTLIER-RESISTANT MODELS FOR DOUBLY STOCHASTIC POINT PROCESSES

by

Leo Elsaesser

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

in MATHEMATICS

at

The University of Wisconsin-Milwaukee

May 2019

ABSTRACT

OUTLIER-RESISTANT MODELS FOR DOUBLY STOCHASTIC POINT PROCESSES

by

Leo Elsaesser

The University of Wisconsin-Milwaukee, 2019
Under the Supervision of Professor Daniel Gervini

This thesis proposes an outlier-resistant multiplicative component model for doubly stochastic point processes. The model is based on a principal component decomposition of the log-intensity functions, using heavy-tailed t-distributions for the component scores. As an example of application, the temporal distribution of bike check-out times in the Divvy bike sharing system of Chicago is analyzed using the t-model.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

Point processes are a well studied area in probability and statistics and are the basis for models in many applications. While mostly single-realization cases are considered, situations where several replications of the underlying point process are available are relatively rarely explored in the literature [Gervini, 2017]. In this thesis, we focus on point processes which have a low intensity that gives rise to only few events but has many replications in turn. Wu et al. [2013] proposes a model to estimate realization-specific intensity functions with a functional principal component analysis (FPCA) approach, in which one estimates a covariance function by borrowing strength across all replications. For our applications, however, some realizations contain such a small number of events, that even this method is inadequate. Therefore, Gervini [2017] proposes an alternative FPCA model, which has, among others, the advantage be able to deal with that challenge. In this thesis, we will study this model and propose an adjustment to better deal with outliers among the replications.

This thesis is organized as follows. In Chapter 2, we will introduce fundamental ideas of functional data analysis, which are necessary to understand the model and its characteristics. We will then present the model and our adjustments in Chapter 3. In Chapter 4, we will consider an application to compare the performances of the two models.

Throughout this thesis, we will refer to a realization of the whole process, which is the set of all events given on the underlying interval, by 'realization' or 'replication'. We will indicate vectors and matrices by bold symbols. However, the symbol for one realization, which is a set, will be non-bold to distinguish it from a vector of several replications.

2 Functional data analysis

In this Chapter, we want to present some topics of functional data analysis to give a mathematical basis for the model introduced in Chapter 3. In Section 2.1, we will focus on smoothing data in terms of basis expansions followed by an introduction to principal component analysis for functional data in Section 2.2.

2.1 Smoothing functional data

Functional data analysis mainly studies discrete data measurements which we assume to be realizations of a smooth, latent function f. Examples could be height measurements of children or weather indicators over the year [see Ramsay and Silverman, 2005, Chap. 2]. In practice, we can see functional data as m discretely observed pairs (t_j, x_j), where x_j is a snapshot of the function at time t_j, maybe blurred by measurement error [Ramsay and Silverman, 2005]:

$$x_j = f(t_j) + \epsilon_j \quad j = 1, \dots m$$

When we find that underlying function, we can further analyze the data and highlight characteristics or study patterns among the data. We usually assume that the underlying function is smooth, so that we are able to calculate derivatives. For finding and investigating the function f, it would be nice to use techniques of the framework of linear models. Therefore, we want to express f in terms of a basis function system $\boldsymbol{\beta} = \{\beta_1, \dots, \beta_q\}$, where the basis functions β_l are mathematically independent and can approximate f arbitrarily well by a linear combination of a sufficient number of these functions. We have

$$f(t) = \sum_{l=1}^{q} c_l \beta_l(t) \quad \text{or} \quad f = \mathbf{c}^T \boldsymbol{\beta}$$

where $\mathbf{c}$ is a vector of coefficients/parameters c_l, which we can find with usual regression techniques. Let's define the $m \times q$ matrix $\mathbf{B}$ as containing the values $\beta_l(t_j)$. Minimizing

the least-squares criterion

$$(\mathbf{x} - \mathbf{Bc})^T(\mathbf{x} - \mathbf{Bc}) \tag{1}$$

gives us the known least-square estimator [Montgomery et al., 2012, p. 73]

$$\hat{\mathbf{c}} = \left(\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{x}.$$

The difference with multivariate data analysis is the basis system $\boldsymbol{\beta}$, that we need to choose. Its choice is important, since the function f inherits attributes of the basis, for example

$$Df(t) = \mathbf{c}^T D\boldsymbol{\beta}.$$

We present here the B-spline basis system, which we are using in our model and because it is the most common choice for approximating (non-periodic) functional data [Ramsay and Silverman, 2005, p. 46]. Splines are piece-wise polynomial curves, which pass through a given set of points and have a certain number of continuous derivatives [cf. Hastie et al., 2008, chap 5]. The B-spline basis system is a numerically convenient basis for the spline space. Cubic splines are cubic polynomials on the regions between a certain number of knots which are connected in a way, such that the spline has continuous second derivatives on the whole interval (see Figure 1). Since linear combination of splines are again splines of the same order and the same sequence of knots, the basis system are splines for themselves [Ramsay and Silverman, 2005, p. 49]. Now, the question is how many basis function do we need to construct a specific cubic spline f with K knots. We have $K + 1$ regions with respectively a cubic function f_k,

$$f_k(x) = a_{0k} + a_{1k}x + a_{2k}x^2 + a_{3k}x^3, \quad k = 1, ..., K + 1$$

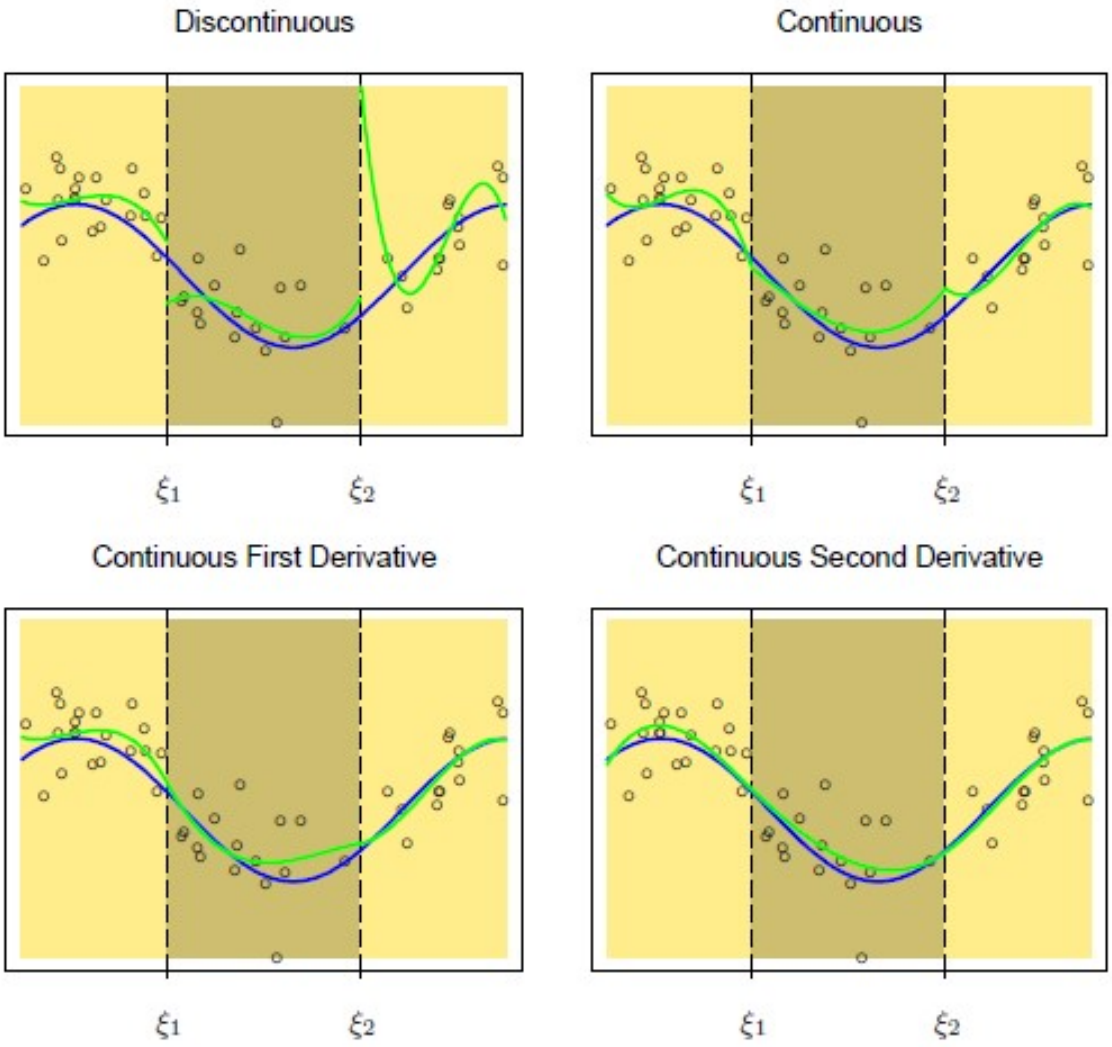


Figure 1: [Hastie et al., 2008, Fig. 5.2]; Illustration of piece-wise cubic polynomials (green), with increasing orders of continuity, which fit some artificial data. The blue curve represents the true function, from which the data were generated with Gaussian noise. The broken vertical lines indicate the positions of two knots ξ_1 and ξ_1

so this gives us $4(K+1)$ parameters. On the other hand, we have the K knots ϵ_k with respectively 3 constraints:

$$f_k(\epsilon_k) = f_{k+1}(\epsilon_k),\ f_k^{'}(\epsilon_k) = f_{k+1}^{'}(\epsilon_k),\ f_k^{''}(\epsilon_k) = f_{k+1}^{''}(\epsilon_k), \quad k = 1, .., K$$

So we need $q = 4(K+1) - 3K = K+4$ parameter c_q, which means that we need $K+4$ basis functions. There are many ways to construct such a basis system but the B-spline basis is popular because it allows efficient computations even for large number of knots K [Hastie et al., 2008, p. 144]. An explanation of how it is constructed is given in Hastie et al., 2008, Appendix Chap. 5.

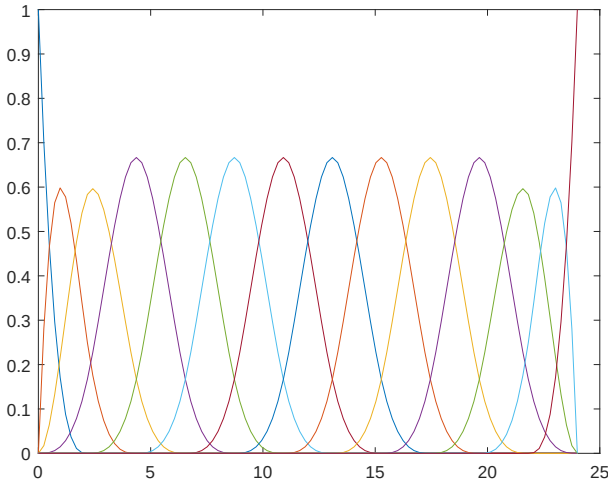


Figure 2: B-spline basis functions for fitting the data in Chapter 4.

Figure 2 shows an example of 14 B-splines basis functions, which were created with 10 equally spaced knots. Notice that the functions are only positive on a small part of the interval. When choosing a B-spline basis system, we can decide on the position as well as on the number of knots. Techniques for improving knot placement are discussed in de Boor [2001].

Choosing an amount of knots means to choose q, the number of basis functions and dimension of the basis expansion. When we choose q large, we get a better fit to data

and the bias

$$Bias[\hat{f}(t)] = f(t) - \mathbb{E}[\hat{f}(t)]$$

is small. For $q = m$, we even achieve interpolation but in that case we are also fitting noise and we wish to ignore that. For smaller q, we get a smoother function and the variance of the estimate

$$Var[\hat{f}(t)] = \mathbb{E}[\Big\{ f(t) - \mathbb{E}[\hat{f}(t)] \Big\}^2]$$

goes down. Then, we can e.g. compute sharper confidence intervals [cf. Montgomery et al., 2012, ch. 3.4]. This dilemma is called bias-variance trade-off and often solved by minimizing the mean-squared error

$$MSE[\hat{f}(t)] = \mathbb{E}[\{f(t) - f(t)]\}^2] = \Big(Bias[\hat{f}(t)]\Big)^2 + Var[\hat{f}(t)].$$

The smoothness of function f, we get with adjusting q, is often gained by unnecessarily sacrificing bias [Ramsay and Silverman, 2005, p. 84]. Therefore, smoothness is often achieved by using a roughness penalty for a given amount of basis functions. Ramsay and Silverman [2005] measures the roughness of a function by the integrated squared second derivative

$$Pen(f) = \int (D^2 f)^2 = \int (D^2 \mathbf{c}^T \boldsymbol{\beta})^2 = \int \mathbf{c}^T D^2 \boldsymbol{\beta} D^2 \boldsymbol{\beta}^T \mathbf{c} = \mathbf{c}^T \mathbf{R} \mathbf{c}$$

with $\mathbf{R} = \int D^2 \boldsymbol{\beta} D^2 \boldsymbol{\beta}^T$. Since the second derivative exists for cubic splines, we can add $Pen(f)$ as a penalty term to the least square criterion (1) multiplied by a smoothing parameter v.

$$(\mathbf{y} - \mathbf{B}\mathbf{c})^T (\mathbf{y} - \mathbf{B}\mathbf{c}) + v \mathbf{c}^T \mathbf{R} \mathbf{c} \tag{2}$$

By minimizing (2), we obtain the estimated coefficient vector

$$\hat{\mathbf{c}} = \left(\mathbf{B}^T\mathbf{B} + v\mathbf{R}\right)^{-1}\mathbf{B}^T\mathbf{y}.$$

[Ramsay and Silverman, 2005, p. 87]. This gives a direct optimization of smoothness in the function instead of implicitly in term of the number q of basis functions. When we choose v large, this means that the smoothness of f is more stressed than fitting the data. For $v \to \infty$, f would get linear.

A typical strategy for selecting v is cross-validation. In general, this means to split the data into training and validation samples, to fit the model to the training samples and check the result with validation samples. A common technique is to optimize (2) for a specific v with only leaving out one observation and to calculate the resulting error sum of squares for this observation. When we do this for all data points and sum up the results, we get a criterion for each v, which we want to minimize [Ramsay and Silverman, 2005, p. 96]. There are also other, faster ways to choose v with cross-validation, like the GCV-method, which is based on the degrees of freedom for a spline smooth [see Ramsay and Silverman, 2005, Chap. 5.4.3].

2.2 Principal component analysis for functional data

Let's assume, we got n functions f_i after smoothing n replications and we want to explore the features, which are characterizing most of these functions. With principal component analysis (PCA), which is a very common concept in multivariate statistics [e.g. see Izenman, 2008, Chap. 7], we can do this in a very informative and effective way.

Here, we will apply methods, known from multivariate statistics, to our functional analysis. Let's also assume, that the mean of our functions $f_\mu := \frac{1}{n}\sum_{i=1}^{n} f_i$ is zero. We can achieve this, by subtracting f_μ from each function. Ramsay and Silverman [2005] presents three ways to motivate PCA:

The first is to introduce a weighting function ϕ, with which we can calculate score values

$$u_i = \int \phi f_i, \quad i = 1, \ldots, n. \tag{3}$$

We want to choose a ϕ, that highlights strongly represented types of variation in the functions f_i. Therefore, we maximize

$$\sum_{i=1}^{n} u_i^2, \quad \text{s.t.} \quad \int \phi^2 = 1 \tag{4}$$

and get our first component ϕ_1. Since the mean of our functions is zero, it holds that

$$n\text{Var}(\mathbf{u}) = \sum_{i=1}^{n} u_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} u_i\right)^2 = \sum_{i=1}^{n} u_i^2 - \frac{1}{n}\left(\int \phi \sum_{i=1}^{n} f_i\right)^2 = \sum_{i=1}^{n} u_i^2$$

and thus, maximizing (4) is equal to maximize the variance of $\mathbf{u}$. To get more components, we can repeat this with the restriction that new components are orthogonal to the existing ones, i.e. $\int \phi_k \phi_m = 0$. With each component, we get new modes of variation, but $\text{Var}(\mathbf{u})$ will decline with each step and we will stop at a maximum index p.

A second characterization of PCA is in terms of an eigenanalysis of the covariance function

$$v(s,t) = \frac{1}{n}\sum_{i=1}^{n} f_i(s) f_i(t). \tag{5}$$

Let's assume $f_i \in \mathcal{L}^2$ and we can define the linear covariance operator V as

$$V : \mathcal{L}^2 \to \mathcal{L}^2, \phi \mapsto V\phi = \int v(\cdot, t)\phi(t)dt.$$

Let's also assume that there exits an orthonormal basis of eigenfunctions $\{\phi_k\}$ of $\mathcal{L}^2$ which fulfills the eigenequations

$$V\phi_k = \rho_k \phi_k \tag{6}$$

By (3) we could write f_i as

$$f_i = \sum_{k=1}^{\infty} u_{ik}\phi_k$$

where u_{ik} is the respective score of ϕ_k. Since

$$V\phi_k(s) = \int \frac{1}{n}\sum_{i=1}^{n} f_i(s)f_i(t)\phi_k(t)dt = \frac{1}{n}\sum_{i=1}^{n} f_i(s)\int f_i(t)\phi_k(t)dt = \frac{1}{n}\sum_{i=1}^{n} f_i(s)u_{ik}, \quad (7)$$

we have

$$\rho_k = \rho_k \int \phi_k\phi_k = \int \phi_k V\phi_k = \int \phi_k \frac{1}{n}\sum_{i=1}^{n} f_i u_{ik} = \frac{1}{n}\sum_{i=1}^{n} u_{ik}\int \phi_k f_i = \frac{1}{n}\sum_{i=1}^{n} u_{ik}^2.$$

So, (4) turns into finding the eigenvector ϕ_k with the largest eigenvalue ρ_k.

We can solve (6) by discretizing our functions. Therefore, we turn our functions f_i into vectors, by evaluating them on a grid of N equally spaced values on the interval of interest. V is then a $N \times N$ covariance matrix. Now, we can solve (6), e.g. by singular-value decomposition, and then, approximate the eigenfunctions ϕ_k with interpolation of the solutions.

To visualize the results, we can plot f_μ and functions obtained by adding/subtracting a multiple of the components β_l to/from f_μ (e.g. see Figure 5 of Chapter 4).

Another but also equivalent way to motivate principal components is to approximate all f_i as closely as possibly with a set of p orthonormal basis functions ϕ_k, i.e. find

$$\underset{\phi_k}{\operatorname{argmin}} \sum_{i=1}^{n} \int \left(f_i - \sum_{k=1}^{p} u_{ik}\phi_k \right)^2. \quad (8)$$

Seen in that way, PCA is a dimension-reduction technique, which projects the f_is into an optimal p-dimensional subspace. Since

$$\int \phi_l f_i = \int \phi_l \left(\sum_{k=1}^{p} u_{ik}\phi_k + \left(f_i - \sum_{k=1}^{p} u_{ik}\phi_k \right) \right) = \int u_{ikl}\phi_l\phi_l = u_{il}$$

we can refer to the u_{ik} of (8) as the component scores of (3).

In our model, we will assume that the f_is are realizations of a random process f with $\mu = \mathbb{E}f$ and $v(s,t) = cov(f(s), f(t))$. Then by (6) we would get the so-called Karhunen–Loève representation [cf. Wu et al., 2013, p. 4]

$$f = \mu + \sum_{k=1}^{\infty} U_k \phi_k, \tag{9}$$

where the component scores U_k are uncorrelated random variables with $\mathbb{E}U_k = 0$ and $\mathbb{E}U_k^2 = \rho_k$.

3 Multiplicative component models for replicated point processes

In this Chapter, we give an introduction to the multiplicative component model for replicated point processes of Gervini [2017], and propose an adjustment to better model data-sets with outliers.

3.1 The model

A point process X is a random countable set in a space $\mathcal{S}$. We call it locally finite if $\mathcal{P}(\#(X \cap B) < \infty) = 1$ for all bounded $B \subset \mathcal{S}$ and define $X_B := X \cap B$ [Gervini, 2017, p. 2]. Realizations of such processes are often modeled as (inhomogeneous) Poisson point processes:

Definition 1

[cf. Gervini, 2017, p. 2] Let $\lambda : \mathcal{S} \to [0, \infty)$ be a locally integrable function on a space $\mathcal{S}$. Then, a point process X on $\mathcal{S}$ is called Poisson point process (PPP) with intensity function λ, if for any bounded $B \subset \mathcal{S}$ it holds that

(i) $N(B) := \#(X \cap B)$ has Poisson distribution with rate $\int_B \lambda$.

(ii) Given $N(B) = m$, the m points in X_B are i.i.d. with density $\tilde{\lambda} = \lambda / \int_B \lambda$

Thus, for a PPP X, the density of X_B at $x_B = \{t_1, \ldots, t_m\}$ with $t_1, \ldots, t_m \in B$ and $m \in \mathbb{N}$ is given by

$$\begin{aligned} &f(x_B) = f(x_B \mid m) f(m) \quad \text{with} \\ &f(m) = \exp\left(-\int_B \lambda(t)dt\right) \frac{\left(\int_B \lambda(t)dt\right)^m}{m!} \\ &f(x_B \mid m) = \prod_{j=1}^{m} \frac{\lambda(t_j)}{\int_B \lambda(t)dt}. \end{aligned} \tag{10}$$

When we get several replications of a point process X, it would be very restrictive to model them with a unique underlying intensity function λ. Therefore, we assume that

there are subject-specific λ's for every realization, which occur as latent random effects [Gervini, 2017, p. 3]. We get the doubly stochastic process (X, Λ), where Λ is a latent intensity process, which characterizes the distribution of X.

In the following, we assume that all realizations of X are observed on the same region $B \subset \mathcal{S}$ and neglect the index B.

Our aim is to find a model for Λ, which gives us reasonable intensity functions $\lambda_1, \ldots, \lambda_n$ for n independent realizations $x_1, \ldots, x_n$ of X and highlights their variation. Since the intensity function of a PPP takes only non-negative values, we model Λ as the exponential of an unconstrained function $\log \Lambda$. Let's assume that we can write $\log \Lambda$ as in (9). When we also assume that $\log \Lambda - \mathbb{E}[\log \Lambda]$ is Gaussian, we have that the component scores U_k are also Gaussian and stochastically independent. This leads us to the following multiplicative component model [Gervini, 2017, p. 3]

$$\Lambda = \lambda_0 \prod_{k=1}^{p} \xi_k^{U_k} \quad \text{or} \quad \log \Lambda = \mu + \sum_{k=1}^{p} U_k \phi_k \tag{11}$$

where $\lambda_0 = \exp \mu$ is the baseline intensity and $\xi_k = \log \phi_k$, $k = 1, \ldots p$ are multiplicative components. $\mu \in L^2(B)$ and $\phi_1, \ldots, \phi_p$ are orthonormal functions in $L^2(B)$. The U_ks are independent $N(0, \sigma_k^2)$ random variables. For a realization x of X, we have a latent realization $\mathbf{u} := (u_1, \ldots, u_p)$ of $\mathbf{U} := (U_1, \ldots, U_p)$ which gives us the respective latent intensity λ.

3.2 Fitting the model

After defining our general model, we want to fit the model for n independent realizations $x_1, \ldots, x_n$. We model $\mu = \mathbf{c}_0^T \boldsymbol{\beta}$ and $\left\{\phi_k = \mathbf{c}_k^T \boldsymbol{\beta}\right\}_{k=1,\ldots,p}$ in term of a basis $\boldsymbol{\beta} := (\beta_1, \ldots, \beta_q)^T$. For temporal processes, where $\mathcal{S} = \mathbb{R}$, we can use a B-spline basis (see Chapter 2.1). If we choose a specific number of components p, we get the following vector of parameters

$$\boldsymbol{\theta} = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_p, \sigma_1^2, \ldots, \sigma_p^2)^T. \tag{12}$$

We estimate $\boldsymbol{\theta}$ by penalized maximum likelihood:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \rho_n(\boldsymbol{\theta})$$

where

$$\rho_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \log f_{\boldsymbol{\theta}}(x_i) - \nu_1 P(\mu) - \nu_2 \sum_{k=1}^{p} P(\phi_k) \tag{13}$$

We have two different smoothing parameters ν_1 and ν_2 for μ and the ϕ_ks, because the ϕ_ks are orthonormal functions with unit norm and μ has no unit norm.
We will proceed as follows: We update $\boldsymbol{\theta}$ in the order $c_0, c_1, \sigma_1^2, \ldots, c_p, \sigma_p^2$ with a Newton-Raphson algorithm to maximize $\rho_n(\boldsymbol{\theta})$. For a given $\boldsymbol{\theta}$, we can also approximate $f_{\boldsymbol{\theta}}(x_i)$ and $\hat{\mathbf{u}}_i = \mathbb{E}_{\boldsymbol{\theta}}[\mathbf{U} \mid x_i]$. The latter gives us estimates for the latent variables $\lambda_1, \ldots, \lambda_n$ (see Chapter 3.5). Then, we choose smoothing parameters by cross-validation.

3.3 Modeling Λ with t-distributed component scores

For our model in (11), we were assuming that $\log\Lambda - \mathbb{E}[\log\Lambda]$ is Gaussian and got that the component scores are independent $N(0, \sigma_k^2)$ random variables. This makes the ML-Estimation easy, however, for some applications it is not appropriate to assume that the underlying distribution is light-tailed.
In this thesis, we model the U_ks in term of p independent t-distributed random variables. This gives us the following density for $\mathbf{U}$:

$$f(\mathbf{u}) = \prod_{k=1}^{p} \frac{c}{\sigma_k}\left(1 + \frac{1}{v}\left(\frac{u_k}{\sigma_k}\right)^2\right)^{-\frac{v+1}{2}} \quad \text{with} \quad c = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma\left(\frac{v}{2}\right)\sqrt{\pi v}}$$

$v \in \mathbb{N}$ is the number of degrees of freedom and the $\sigma_k \in \mathbb{R}_+$ are scaling parameters for each U_k. Even though, the σ_k^2s are not the variances of the component scores (it's $Var(U_k) = \sigma_k^2 \frac{v}{v-2}$ for $v > 2$), they will take over the role of the variances of the normal distributed component scores in $\boldsymbol{\theta}$. For our calculations, we manually choose v (mostly $v = 1$) and write $f_{\boldsymbol{\theta}}(\mathbf{u})$ instead of $f(\mathbf{u};\ v, \sigma_1, \ldots, \sigma_p)$. Since we are still using ML-

Estimation, we also need to calculate

$$\log f_{\boldsymbol{\theta}}(\mathbf{u}) = \sum_{k=1}^{p} \log c - \frac{1}{2}\log \sigma_k^2 - \frac{v+1}{2}\log\left(1 + \frac{1}{v}\left(\frac{u_k}{\sigma_k}\right)^2\right). \tag{14}$$

3.4 Updating σ_k^2

We want to maximize $\rho_n(\boldsymbol{\theta})$ and we are doing this for one parameter at a time. In our t-model, we have the same calculations for the c_ks (except for the Laplace approximation, see Chapter 3.5) but the ones for the σ_k^2s are changing.
For the normal distributed U_ks, we were able to explicitly solve

$$\frac{\partial}{\partial \sigma_k^2}\rho_n(\boldsymbol{\theta}) = 0$$

and got the formula

$$\left(\sigma_k^2\right)^{new} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\boldsymbol{\theta}^{old}}\left[U_k^2 \mid x_i\right]. \tag{15}$$

$\mathbb{E}_{\boldsymbol{\theta}}\left[U_k \mid x_i\right]$ can be found with the Laplace approximation (see Chapter 3.5). For the t-distributed U_ks, we get

$$\begin{aligned} 0 \overset{!}{=} \frac{\partial}{\partial \sigma_k^2}\rho_n(\boldsymbol{\theta}) &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \sigma_k^2}\log f_{\boldsymbol{\theta}}(x_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\boldsymbol{\theta}}\left[\frac{\partial}{\partial \sigma_k^2}\log f_{\boldsymbol{\theta}}(\mathbf{U}) \mid x_i\right] \end{aligned}$$

For the second transformation see Gervini [2017, p. 4]. By differentiating (14), we get

$$\begin{aligned} \frac{\partial}{\partial \sigma_k^2}\log f_{\boldsymbol{\theta}}(\mathbf{u}) &= -\frac{1}{2\sigma_k^2} - \frac{v+1}{2}\frac{1}{1+\frac{1}{v}(\frac{u_k}{\sigma_k})^2}(-1)\frac{u_k^2}{v\sigma_k^4} \\ &= -\frac{1}{2\sigma_k^2} + \frac{(v+1)u_k^2}{v\sigma_k^2 + u_k^2}\frac{1}{2\sigma_k^2} \\ &= \frac{1}{2\sigma_k^2}\left(\frac{(v+1)u_k^2}{v\sigma_k^2 + u_k^2} - 1\right) \end{aligned}$$

which gives us

$$0 \stackrel{!}{=} \frac{\partial}{\partial \sigma_k^2} \rho_n(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2\sigma_k^2} \left(\mathbb{E}_{\boldsymbol{\theta}} \left[\frac{(v+1)U_k^2}{v\sigma_k^2 + U_k^2} \mid x_i \right] - 1 \right)$$

Here, σ_k^2 is inside the expected value, so, we cannot find a explicit solution like in (15). Therefore, we approximate

$$\mathbb{E}_{\boldsymbol{\theta}} \left[\frac{(v+1)U_k^2}{v\sigma_k^2 + U_k^2} \mid x_i \right] \approx \frac{(v+1)\mathbb{E}_{\boldsymbol{\theta}} \left[U_k^2 \mid x_i \right]}{v\sigma_k^2 + \mathbb{E}_{\boldsymbol{\theta}} \left[U_k^2 \mid x_i \right]}$$

to keep the calculations simple. For fixed k, we define $u_i^2 := \mathbb{E}_{\boldsymbol{\theta}}[U_k^2 \mid x_i]$ and we get

$$\begin{aligned} 0 &\stackrel{!}{=} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2\sigma_k^2} \left(\frac{(v+1)u_i^2}{v\sigma_k^2 + u_i^2} - 1 \right) \\ n &\stackrel{!}{=} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{v\sigma_k^2 + u_i^2} \\ \sigma_k^2 &\stackrel{!}{=} \frac{1}{n} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{v + u_i^2/\sigma_k^2} \end{aligned}$$

This is a recursive formula for σ_k^2 and we hope to get close to a fix point after some iterations. Let's define the function

$$g(x) := \frac{1}{n} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{v + u_i^2/x}$$

Obviously, g is strictly monotonously increasing in x and $g : (0, \infty) \to (0, \frac{v+1}{v} \frac{1}{n} \sum_{i=1}^{n} u_i^2)$. So, g must have at least one fix point (ref: Brouwer fix-point theorem). We can estimate

$$g(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{v + u_i^2/x} = x \frac{1}{n} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{vx + u_i^2} > x \frac{1}{n} \sum_{i=1}^{n} \frac{(v+1)u_i^2}{vx + u_{\max}^2} > x$$

for

$$\begin{aligned}
\frac{(v+1)\frac{1}{n}\sum_{i=1}^{n} u_i^2}{vx + u_{\max}^2} &> 1 \\
(v+1)\frac{1}{n}\sum_{i=1}^{n} u_i^2 &> vx + u_{\max}^2 \\
\frac{v+1}{v}\frac{1}{n}\sum_{i=1}^{n} u_i^2 - \frac{u_{\max}^2}{v} &> x
\end{aligned}$$

Thus, there is a fix point of g in the interval

$$\left(\frac{v+1}{v}\frac{1}{n}\sum_{i=1}^{n} u_i^2 - \frac{u_{\max}^2}{v}, \frac{v+1}{v}\frac{1}{n}\sum_{i=1}^{n} u_i^2 \right).$$

3.5 The Laplace approximation

When we model the intensity functions of a PPP X as a stochastic process, there exist no closed form for the density $f_{\boldsymbol{\theta}}(x)$ [Gervini, 2017, p. 7]. However, we would like to get a value for $f_{\boldsymbol{\theta}}(x)$ for a given $\boldsymbol{\theta}$ to compute the score $\rho_n(\boldsymbol{\theta})$ (see (13)). Therefore, we compute it by Laplace approximation of the integral over $\mathbf{U}$ [cf. Gervini, 2017, Chap. 1.5]. By doing so, we also get values for the first two moments of $\mathbf{U} \mid x$. We need these to update our parameters (see Chapter 3.4). We can write

$$f_{\boldsymbol{\theta}}(x) = \iint f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) f_{\boldsymbol{\theta}}(\mathbf{u}) d\mathbf{u} = \iint \exp g_{(x,\boldsymbol{\theta})}(\mathbf{u}) d\mathbf{u}$$

with

$$g_{(x,\boldsymbol{\theta})}(\mathbf{u}) = \log f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) + \log f_{\boldsymbol{\theta}}(\mathbf{u}) \tag{16}$$

For $\hat{\mathbf{u}} = \operatorname{argmax} g_{(x,\boldsymbol{\theta})}(\mathbf{u})$, we have $\nabla g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}}) = 0$. Thus, we get the Taylor approximation

$$g_{(x,\boldsymbol{\theta})}(\mathbf{u}) \approx g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}}) + .5\,(\mathbf{u} - \hat{\mathbf{u}})^T H g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\,(\mathbf{u} - \hat{\mathbf{u}})$$

and we can approximate

$$\begin{aligned} f_{\boldsymbol{\theta}}(x) &= \iint \exp g_{(x,\boldsymbol{\theta})}(\mathbf{u}) d\mathbf{u} \\ &\approx \iint \exp\left(g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}}) + .5\,(\mathbf{u}-\hat{\mathbf{u}})^T\, Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\,(\mathbf{u}-\hat{\mathbf{u}})\right) d\mathbf{u} \\ &= \exp\left(g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right) \iint \exp\left(-.5\,(\mathbf{u}-\hat{\mathbf{u}})^T\, \mathbf{S}^{-1}\,(\mathbf{u}-\hat{\mathbf{u}})\right) d\mathbf{u} \\ &= \exp\left(g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right) (2\pi)^{p/2} \det(\mathbf{S})^{1/2} \end{aligned} \tag{17}$$

with $\mathbf{S} = \left\{-Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right\}^{-1}$. In the last step, we used integration over the density of a multivariate normal distribution $N_p(\hat{\mathbf{u}}, \mathbf{S})$. Note, that $\mathbf{S}$ should be positive-definite [Chuong, 2008]. So, to approximate $f_{\boldsymbol{\theta}}(x)$, we need to find $\hat{\mathbf{u}}$ and $\mathbf{S}$.

We find $\hat{\mathbf{u}} = \operatorname{argmax} g_{(x,\boldsymbol{\theta})}(\mathbf{u})$ by a few steps of Newton-Raphson:

$$\hat{\mathbf{u}}^{new} = \hat{\mathbf{u}}^{old} - \left(Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}}^{old})\right)^{-1} \nabla g_{(x,\boldsymbol{\theta})}\left(\hat{\mathbf{u}}^{old}\right)$$

We need to calculate $\nabla g_{(x,\boldsymbol{\theta})}$ and $Hg_{(x,\boldsymbol{\theta})}$, which we get by finding $\nabla_{\mathbf{u}} \log f_{\boldsymbol{\theta}}(x \mid \mathbf{u})$, $\nabla \log f_{\boldsymbol{\theta}}(\mathbf{u})$, $H_{\mathbf{u}} \log f_{\boldsymbol{\theta}}(x \mid \mathbf{u})$, and $\nabla \log f_{\boldsymbol{\theta}}(\mathbf{u})$ (see (16)).

$f_{\boldsymbol{\theta}}(x \mid \mathbf{u})$ is just the density of x for a given $\lambda_{\mathbf{u}}$. We have its formula in (10). Thus,

$$\log f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) = -\int_B \lambda_{\mathbf{u}}(t) dt - \log m! + \sum_{j=1}^{m} \log \lambda_{\mathbf{u}}(t_j)$$

If we write $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_p)^T$, we have by (11) that

$$\log \lambda_{\mathbf{u}} = \mu + \mathbf{u}^T \boldsymbol{\phi} \quad \text{and} \quad \lambda_{\mathbf{u}} = \exp\left(\mu + \mathbf{u}^T \boldsymbol{\phi}\right)$$

and so

$$\begin{aligned}
\nabla_{\mathbf{u}} \log \lambda_{\mathbf{u}} &= \boldsymbol{\phi} \\
H_{\mathbf{u}} \log \lambda_{\mathbf{u}} &\equiv 0 \\
\nabla_{\mathbf{u}} \lambda_{\mathbf{u}} &= \lambda_{\mathbf{u}} \boldsymbol{\phi} \\
H_{\mathbf{u}} \lambda_{\mathbf{u}} &= \lambda_{\mathbf{u}} \boldsymbol{\phi} \boldsymbol{\phi}^T.
\end{aligned}$$

With this, we can calculate

$$\nabla_{\mathbf{u}} \log f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) = -\int_B \lambda_{\mathbf{u}}(t) \boldsymbol{\phi}(t) dt + \sum_{j=1}^{m} \boldsymbol{\phi}(t_j)$$

and

$$H_{\mathbf{u}} \log f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) = -\int_B \lambda_{\mathbf{u}}(t) \boldsymbol{\phi}(t) \boldsymbol{\phi}(t)^T dt. \tag{18}$$

$\log f_{\boldsymbol{\theta}}(\mathbf{u})$ is the marginal density of $\mathbf{U}$ (or Λ). With normal distributed U_ks, we got that

$$\frac{\partial}{\partial u_k} \log f_{\boldsymbol{\theta}}(\mathbf{u}) = -\frac{u_k}{\sigma_k^2} \quad \text{and} \quad \frac{\partial^2}{\partial^2 u_k} \log f_{\boldsymbol{\theta}}(\mathbf{u}) = -\frac{1}{\sigma_k^2} \quad k = 1, \ldots, p. \tag{19}$$

When we model the component scores by independent t-distributed random variables, we derive from (14) that

$$\begin{aligned}
\frac{\partial}{\partial u_k} \log f_{\boldsymbol{\theta}}(\mathbf{u}) &= -\frac{v+1}{2} \frac{1}{1 + \frac{1}{v}(\frac{u_k}{\sigma_k})^2} \frac{2u_k}{v\sigma_k^2} \\
&= -\frac{(v+1)u_k}{v\sigma_k^2 + u_k^2} \qquad k = 1, \ldots, p.
\end{aligned}$$

For $H \log f_{\boldsymbol{\theta}}(\mathbf{u})$, we derive

$$\begin{aligned}
\frac{\partial^2}{\partial^2 u_k} \log f_{\boldsymbol{\theta}}(\mathbf{u}) &= -(v+1) \frac{(v\sigma_k^2 + u_k^2) - u_k 2u_k}{(v\sigma_k^2 + u_k^2)^2} \\
&= -\frac{(v+1)\,(v\sigma_k^2 - u_k^2)}{(v\sigma_k^2 + u_k^2)^2} \qquad k = 1, \ldots, p
\end{aligned} \tag{20}$$

and

$$\frac{\partial^2}{\partial u_{k_1} \partial u_{k_2}} \log f_{\boldsymbol{\theta}}(\mathbf{u}) = 0 \qquad k_1, k_2 = 1, \ldots, p; \quad k_1 \neq k_2.$$

Now, we can also calculate $\mathbf{S}$ with $Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})$.
Moreover, we get by (17), that one could write

$$f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) f_{\boldsymbol{\theta}}(\mathbf{u}) = \exp g_{(x,\boldsymbol{\theta})}(\mathbf{u}) \approx \exp\left(g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right) (2\pi)^{p/2} \det(\mathbf{S})^{1/2} \varphi_{(\hat{\mathbf{u}},\mathbf{S})}(\mathbf{u}) \tag{21}$$

where $\varphi_{(\hat{\mathbf{u}},\mathbf{S})}$ denotes the density of a $N_p(\hat{\mathbf{u}}, \mathbf{S})$ random variable. By Bayes formula, we derive that

$$\begin{aligned}
f_{\boldsymbol{\theta}}(x) f_{\boldsymbol{\theta}}(\mathbf{u} \mid x) &= f_{\boldsymbol{\theta}}(x \mid \mathbf{u}) f_{\boldsymbol{\theta}}(\mathbf{u}) \\
f_{\boldsymbol{\theta}}(x) f_{\boldsymbol{\theta}}(\mathbf{u} \mid x) &\approx \exp\left(g_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right) (2\pi)^{p/2} \det(\mathbf{S})^{1/2} \varphi_{(\hat{\mathbf{u}},\mathbf{S})}(\mathbf{u}) \\
f_{\boldsymbol{\theta}}(x) f_{\boldsymbol{\theta}}(\mathbf{u} \mid x) &\approx f_{\boldsymbol{\theta}}(x) \varphi_{(\hat{\mathbf{u}},\mathbf{S})}(\mathbf{u}) \\
f_{\boldsymbol{\theta}}(\mathbf{u} \mid x) &\approx \varphi_{(\hat{\mathbf{u}},\mathbf{S})}(\mathbf{u})
\end{aligned}$$

and thus,

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{U} \mid x) \approx N_p(\hat{\mathbf{u}}, \mathbf{S}). \tag{22}$$

From (22), we can approximate the moments

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\theta}}[\mathbf{U} \mid x] &\approx \hat{\mathbf{u}} \\
\mathbb{E}_{\boldsymbol{\theta}}\left[\mathbf{U}\mathbf{U}^T \mid x\right] &\approx \mathbf{S} + \hat{\mathbf{u}}\hat{\mathbf{u}}^T
\end{aligned} \tag{23}$$

which we use to update the parameters (see Chapter 3.4) and to get estimates for the latent variables $\lambda_1, \ldots, \lambda_n$ (see Chapter 3.2).

3.6 Computational problems

When we fitted the t-model to our data, we got computational errors for some combinations of the parameters v and p. It seemed that for small values of v one could only compute a small number of components p. We found the following reason:
When we approximate $f_{\boldsymbol{\theta}}(x)$, we need to find the determinant of $\mathbf{S} = \left\{-Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})\right\}^{-1}$ (see (17)). In the Gaussian model, where Λ is assumed to be Gaussian, $\mathbf{S}$ is calculated with the Cholesky decomposition of $-Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})$. To find a Cholesky decomposition, a matrix has to be positive finite [Higham, 2011]. In the Gaussian model, $-Hg_{(x,\boldsymbol{\theta})}$ is positive definite for all $\mathbf{u}$, as one can see by (18) and (19). In our t-model, however, $\frac{\partial^2}{\partial^2 u_k} \log f_{\boldsymbol{\theta}}(\mathbf{u})$ gets positive whenever $v\sigma_k^2 < u_k^2$ (see (20)). So, $-Hg_{(x,\boldsymbol{\theta})}(\hat{\mathbf{u}})$ may not be positive definite if v is small. Since the t-distribution converges to the normal distribution for $v \to \infty$, we have to decide whether we want to compute many components or getting results which differ much from the Gaussian model by keeping v small.

4 Application: Chicago Bike-sharing System

In this Chapter, we analyze the check-out times of bike trips that took place between April 1 and November 31 of 2016 in Chicago's Divvy system and compare the results of the Gaussian model and the t-model. In this time period, 458 bike stations were active. For a single station, we consider the daily check-out times as a temporal point process, for which we have 244 realizations, and we model it as the replicated Poisson point process X of Chapter 3. It makes sense not to assume a fixed intensity function λ, since the bike demand strongly varies under the influence of external factors. For example, the weather will have an impact on the demand and there will be differences between weekdays and the weekend.

To illustrate the differences of the Gaussian and the t-model, we present the results for station 166, which has median annual count of check-outs [Gervini and Khanal, 2018, p. 15]. As spline basis for our functional parameters μ and $\{\phi_k\}$, we use cubic splines on (0,24) with ten equally spaced knots. According to Gervini and Khanal [2018], $p = 6$ components are sufficient to capture the most important modes of variation in the data. To compute $p = 2$ components, our t-model works only for $v = 3$ or more degrees of freedom of the t-distribution (see Chapter 3.6). For the given data, however, a t-distribution of three degrees of freedom for the component scores gives already very similar results to the normal distribution. For that reason and since two components are sufficient to show the differences between the models, we choose $v = 1$ and $p = 2$.

The estimated baseline intensity function $\hat{\lambda_0}$ is equal for both models and shown in Figure 3. We see one main peak in morning between 7 am and 8 am, where the maximum intensity is 2.5 check-outs per hour. There is one very small peak around midday and one last, small peak in the afternoon. In the night, the intensity is very low. The integral of $\hat{\lambda_0}$ over [0, 24] is 17.5, close to the mean daily count of 17.6.

One can see in Figure 4 that the components of the t-model tend to be closer to one in most areas but highlight areas with much variation even more. However, the first and the second component are very similar for the two models.

As one can see in Figure 5, the first component reduces the main peak we have in the

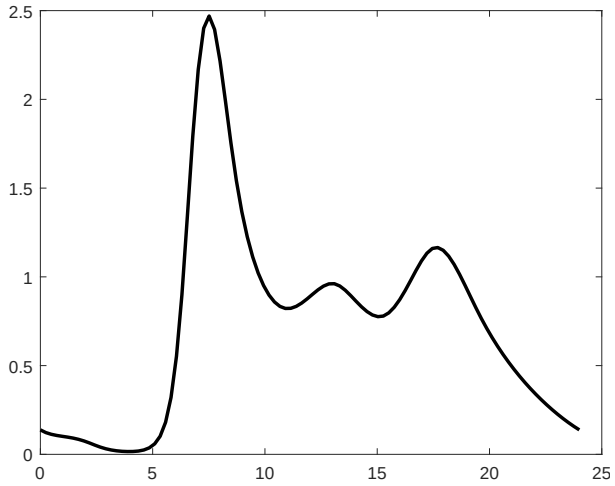
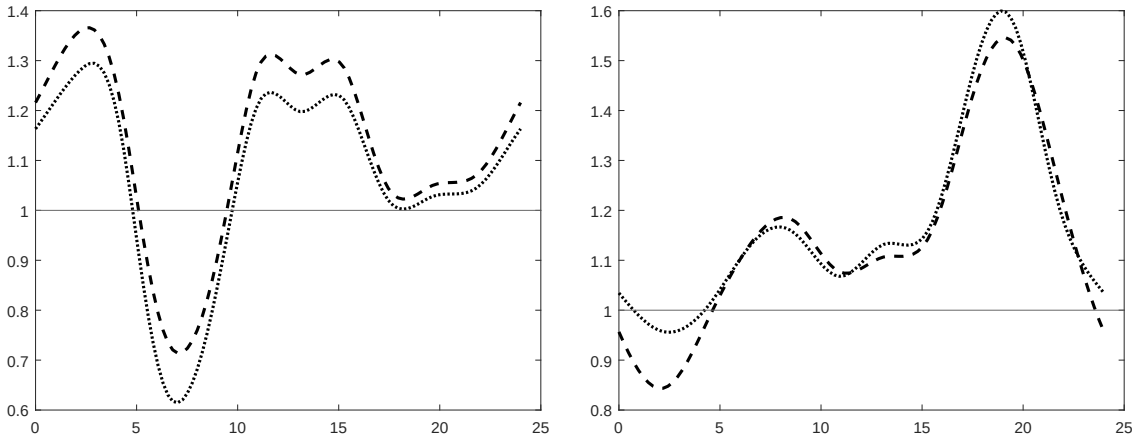
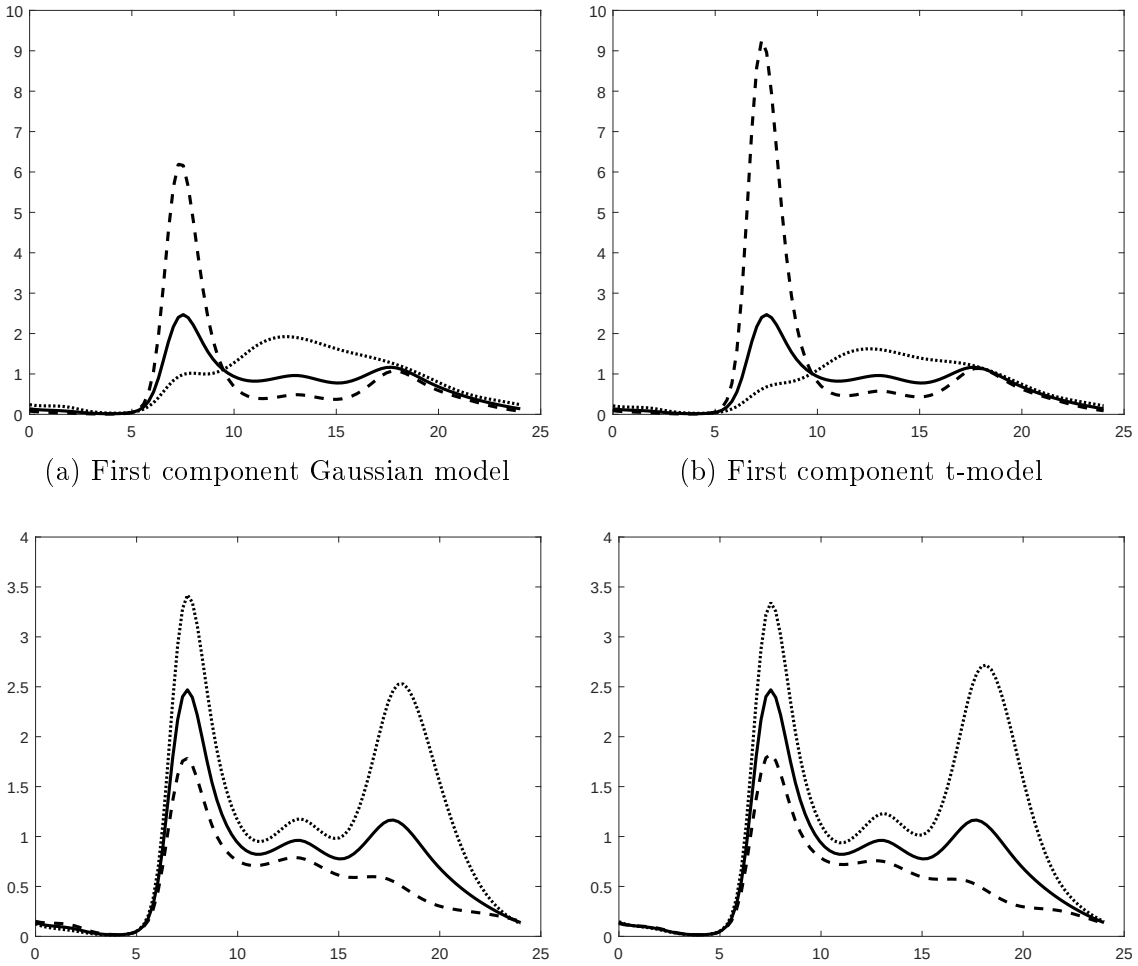


Figure 3: Baseline intensity function of daily bike demand for Divvy station 166.



(a) First multiplicative components (b) Second multiplicative components

Figure 4: Multiplicative components $\{\hat{\xi}_k\}_{k=1,2}$ estimated with the Gaussian model (dashed line) and the t-model (dotted line) of daily bike demand for Divvy station 166.



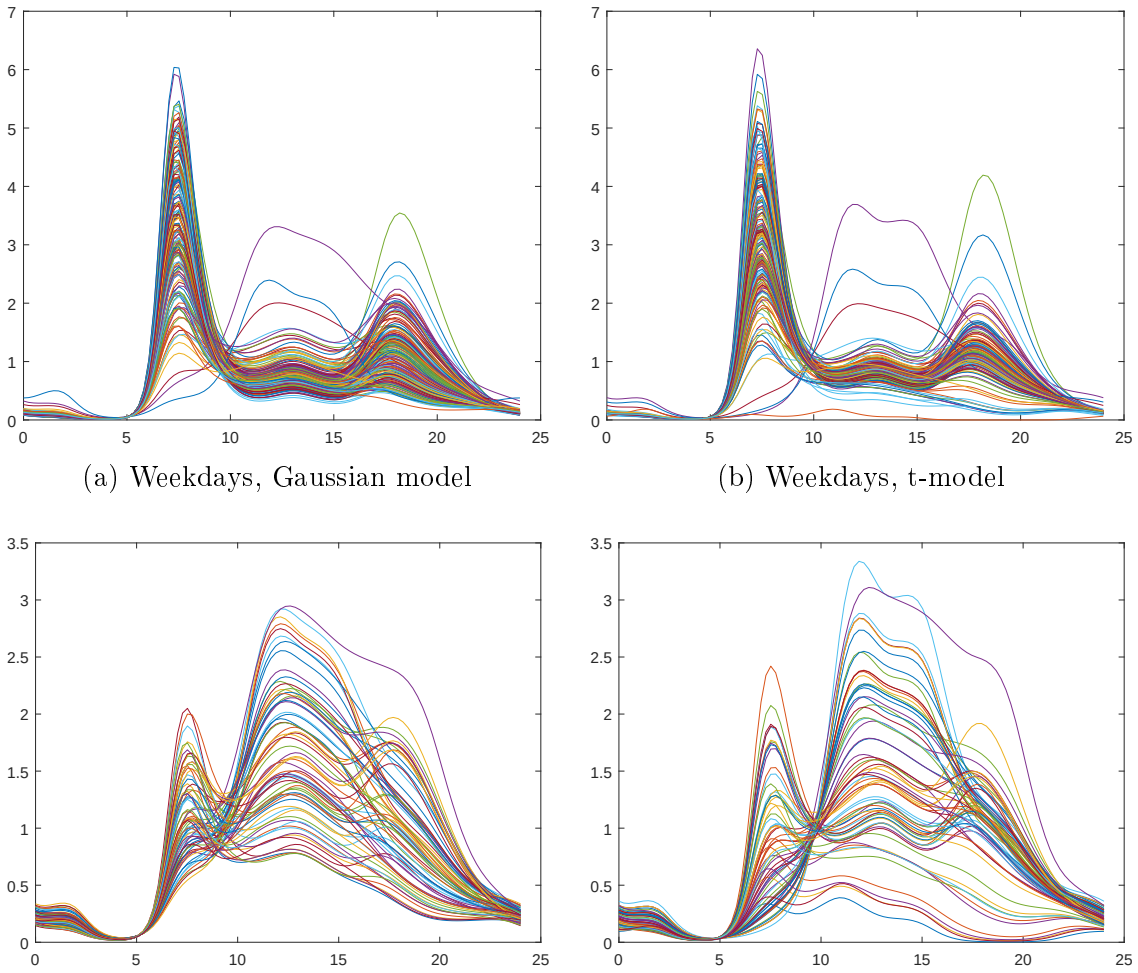
(a) First component Gaussian model

(b) First component t-model

(c) Second component Gaussian model

(d) Second component t-model

Figure 5: Illustration of the multiplicative components: Baseline (solid line) and baseline multiplied by a positive (dotted line) and a negative (dashed line) exponent of the component. For the first component we used ± 2 and for the second ± 1.



(a) Weekdays, Gaussian model

(b) Weekdays, t-model

(c) Weekends, Gaussian model

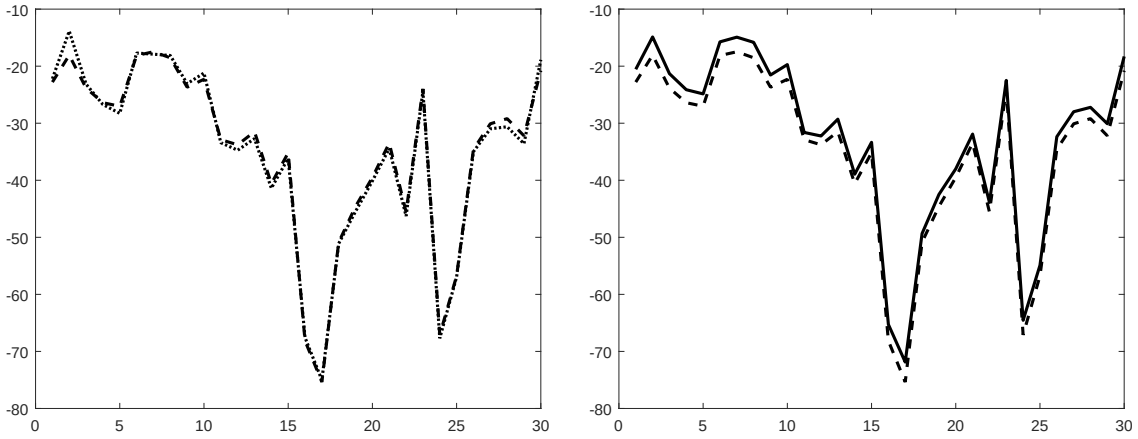
(d) Weekends, t-model

Figure 6: Daily intensity functions of bike demand seperated between weekdays and weekends.

morning and increases the peak around midday for positive component scores u_{i1}. The first component of the t-model is more focused on the main peak. In the illustration of the second component one can barely see differences.

It is also interesting to compare the daily intensity functions of the two models. In Figure 6 they are separated between weekdays and weekends. Their intensity functions look different and it seems as if there was more variability for the intensity functions of the weekends. For weekdays, the two models don't differ a lot. For weekend intensity functions, the t-model seems to get more variability than the Gaussian one. This could mean that the t-model can better fit atypical realizations of the point process.

To compare the performance of the two models, one could also think about comparing the calculated density values $f_{\boldsymbol{\theta}}(x_i)$ of each day. As one can see in Figure 7a, there are



(a) (b)

Figure 7: (a) shows $\log f(x_i)$ for the first 30 days of the time interval computed with the Gaussian model (dashed line) and with the t-model (dotted line).
(b) shows $\log f(x_i)$ (dashed line) and $\log f(x_i \mid \mathbf{u}_i)$ (solid line) computed with the Gaussian model.

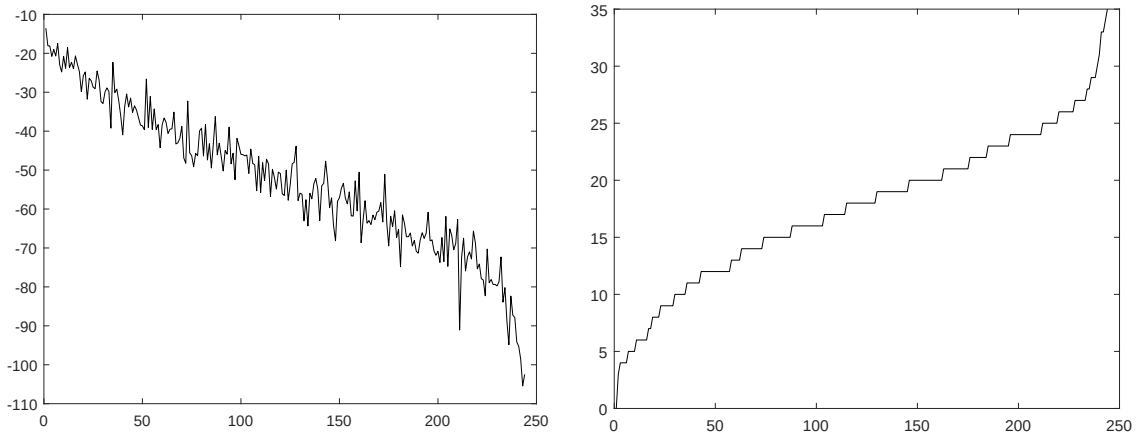
almost no differences between the models. It also seems that the values of the density function do not really depend on the realizations of Λ. Figure 7b compares the values of the density for not knowing the daily intensity functions and conditioned on the estimated $\hat{\lambda}_i$. Even there, the differences aren't large.

In Figure 8, one can see that the values of the log density mainly depend on the number of check-outs on the specific day. As one would expect, the density is smaller when there were more check-outs. This gives the idea of comparing the results of days with the same amount of check-outs.

This is done in Figure 9a and 9b. Where 9b shows a typical weekday check-out pattern, 9a's is irregular, maybe more typical for weekends. In 9b, the two density functions are almost the same, in 9a the t-model seems to give the better fit. This fits to the thought we had for Figure 6 that the t-models can fit outlier better. We can also see this for the extreme scenarios 9c and 9d. Let's give the values calculated with the Gaussian model index 1 and the ones calculated with the t-model index 2. We have the following results:

$i =$	38	77	67	238
$\log f_1(x_i)$	-55.9	-56.2	-102	-13.5
$\log f_2(x_i)$	-55.5	-56.6	-102	-6.29
$\int_{[024]} \lambda_i^1$	20.5	17.7	32.0	10.1
$\int_{[024]} \lambda_i^2$	21.8	18.9	34.5	1.4

We can see that the values of the density function are still very close. Only for November



(a) (b)

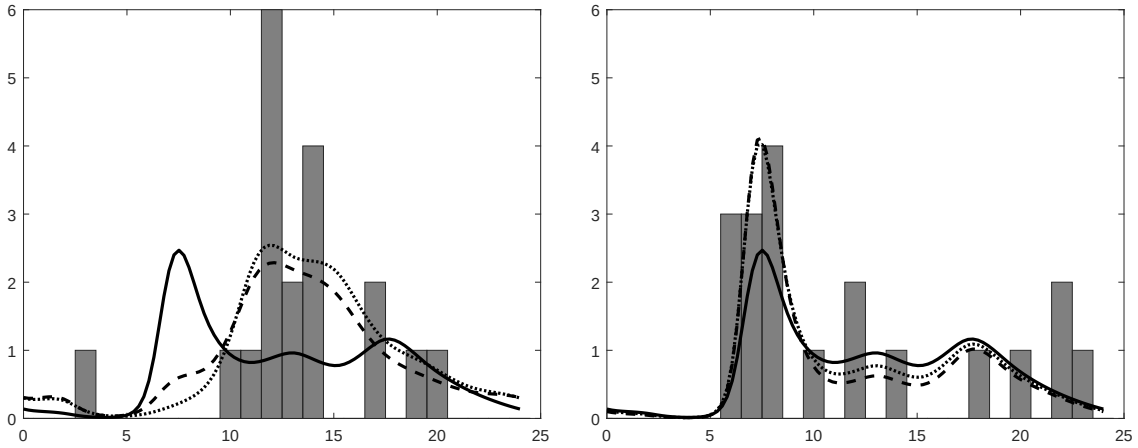
Figure 8: Here, the 244 days of our time interval are ordered by the total amount of daily check-outs. (a) shows $\log f(x_i)$ computed with the Gaussian model. (b) shows the total amount of check-outs for each day.

24 there is a significant difference. It is also interesting to compare the integral of the intensities over [0 24]. These values should be close to the number of check-outs and also here the t-model seems to work a bit better.

In Figure 10 one can see that the t-model gets smaller scores (absolute value) for more typical data but more extreme scores for outlier. This fits to the image we got so far that the t-model better fits outliers.

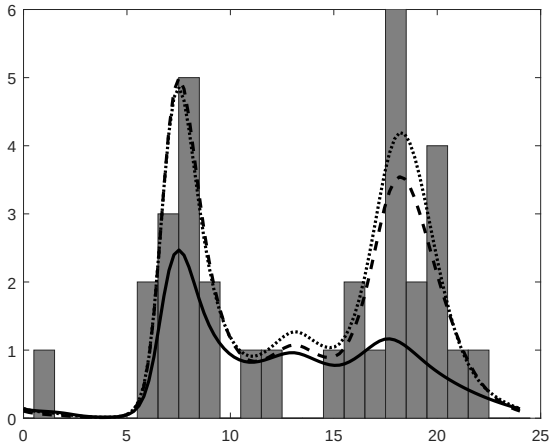
In a good model, one would also expect that the distribution of the calculated scores behave like the estimated distribution of U_ks. In Figure 11, we can see that the computed density functions of the t-model fit better to the histogram of the respective scores values than the ones of the Gaussian model.

It also seems that the empirical distributions of the component scores are skewed.

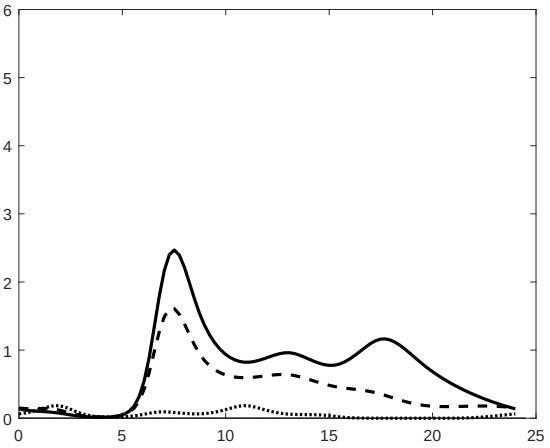


(a) Sunday, May 8 2016; $i = 38$

(b) Thursday, June 16 2016; $i = 77$

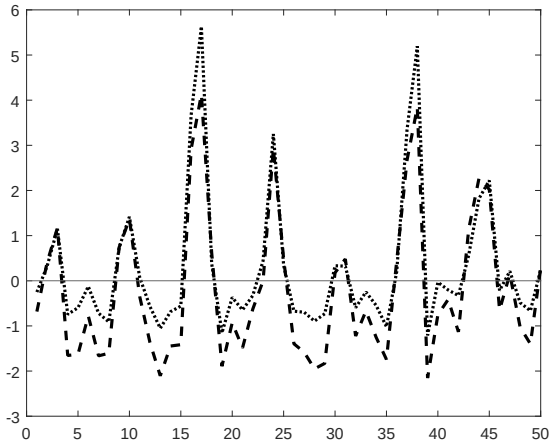


(c) Monday, June 6 2016; $i = 67$

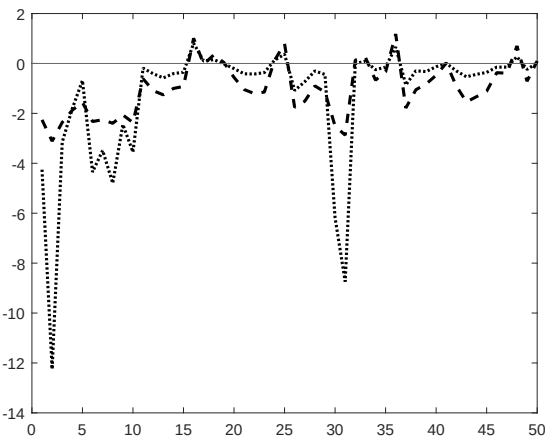


(d) Thursday, November 24 2016; $i = 238$

Figure 9: Histogram of the check-outs of a specific day vs. the baseline intensity (solid line) and the intensity computed with Gaussian model (dashed line) and with the t-model (dotted line).
(a) and (b) had both 19 check-outs. (c) shows the day with most check-outs (35) and (d) shows the days with least check-outs (0)

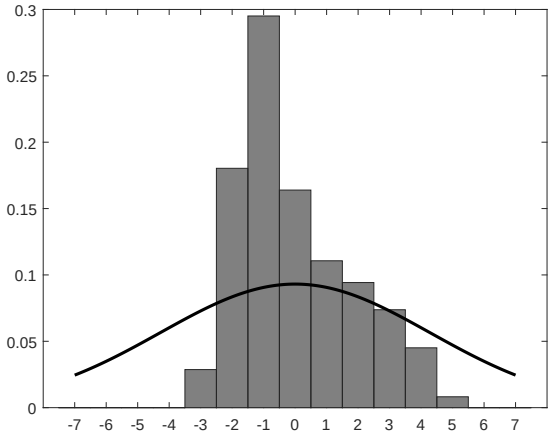


(a) Scores of the first component

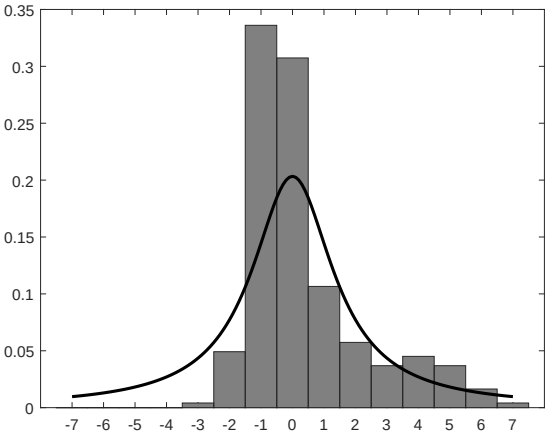


(b) Scores of the second component

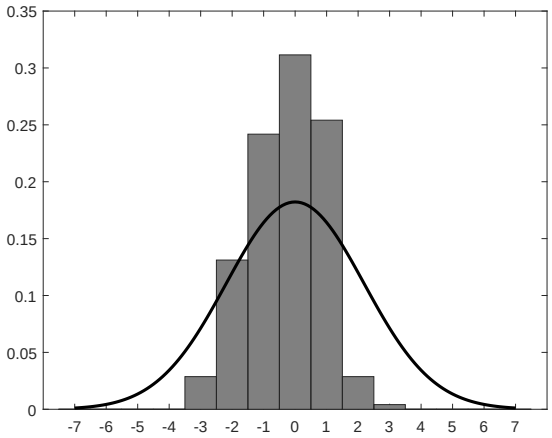
Figure 10: Component scores of the Gaussian model (dashed line) and the t-model (dotted line) for the first 50 days of our time interval



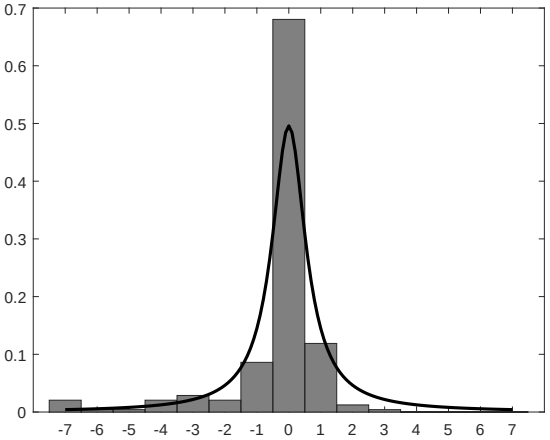
(a) First component, Gaussian model



(b) First component, t-model



(c) Second component, Gaussian model



(d) Second component, t-model

Figure 11: Histogram of the component scores vs. the computed density of U_k.

5 Conclusion

The aim of adjusting the model was to make it outlier-resistant. We have seen in the analysis of the data of the Chicago's Divvy bike sharing system that the t-model doesn't differ much from the Gaussian one, especially for typical course of check-out times. However, the t-model seems to fit these outliers better and also seems to represent the intensity functions of check-outs better in general. So, it would be interesting to fit the two model to other replicated point processes, which have more outliers than the Chicago bike data, and to see whether one gets greater differences. To the computational problems (see Chapter 3.6), the t-model is not really of use for modeling the bike data because for the degrees of freedom needed to compute more components, the two model are too similar.

6 MATLAB code

In this section, we present the MATLAB code, which was used to do the calculations of this thesis. The first part includes the general program and the following parts the functions used.

The main program:

```
data=importdata('station_166.txt');
% Create input cell array x for the functions
n=data(length(data),1);
days=data(:,1);
times=data(:,2);
x = cell(n,1);
for i = 1:n
x{i,1}=times(days==i);
end
% Creat indexes for weekdays and weekends
ind_weekdays = zeros(174,1);
ind_weekends = zeros(70,1);
we1 = 1;
we2 = 1;
for i = 1:n
if ismember(mod(i+3,7),[0,1,2,3,4])
ind_weekdays(we1,1) = i;
we1 = we1+1;
elseif ismember(mod(i+3,7),[5,6])
ind_weekends(we2,1) = i;
we2 = we2+1;
else
disp('Error')
end
end

% Find smoothing parameters
basis = struct('rng',[0 24],'or',4,'nk',10);
p=2;
[sm1,sm2,other] = cv_mcatpp_cyc(x,basis,p);

% Fit Gaussian model
itmax=50;
[c0,C,s2,u,logf] = mcatpp_cyc(x,basis,p,sm1,sm2,itmax);

% Fit t-model
```

```
v=1;
[c0t,Ct,s2t,ut,logft] = mcatpp_cyc_student(x,basis,p,sm1,sm2,itmax,v);

% Plot Baseline and components
t = linspace(basis.rng(1),basis.rng(2),100);
knots = linspace(basis.rng(1),basis.rng(2),basis.nk+2);
B = bspl(t,basis.or,knots,0);
plot(t,exp(B*c0),'-k','linewidth',2);
% Compare components normal and t
plot(t,exp(B*C(:,1)),'--k',t,exp(B*Ct(:,1)),':k','linewidth',2);
yline(1);

plot(t,exp(B*C(:,2)),'--k',t,exp(B*Ct(:,2)),':k','linewidth',2);
yline(1);
% Plot interpretation of components

vart1 = var(ut(:,1));
varn1 = var(u(:,1));
vart2 = var(ut(:,2));
varn2 = var(u(:,2));

splot = [2 1];
plot_component(1,B,c0,splot,C,t,'');
ylim([0 10]);
plot_component(1,B,c0t,splot,Ct,t,'st: ');
ylim([0 10]);
plot_component(2,B,c0,splot,C,t,'');
ylim([0 4]);
plot_component(2,B,c0t,splot,Ct,t,'student: ');
ylim([0 4]);

plot(t,exp(B*(c0+C*u(ind_weekdays,:)')));
plot(t,exp(B*(c0+Ct*ut(ind_weekdays,:)')));
plot(t,exp(B*(c0+C*u(ind_weekends,:)')));
ylim([0 3.5]);
plot(t,exp(B*(c0+Ct*ut(ind_weekends,:)')));
ylim([0 3.5]);

% Plot values of log density
zoom = 1:30;

plot(zoom,logf(zoom,1),'--k',zoom,logft(zoom,1),':k','linewidth',2);
% Create values of density given u / lambda
m=sum(exp(B*c0)/100*24);
d=zeros(i,1);
for k=1:i
d(k)=den_eva(x{k},u(k,:),t,B,c0,C);
end
```

```
dt=zeros(i,1);
for k=1:i
dt(k)=den_eva(x{k},ut(k,:),t,B,c0t,Ct);
end
dB=zeros(i,1);
uB=zeros(i,2);
for k=1:i
dB(k)=den_eva(x{k},uB(k,:),t,B,c0t,Ct);
end
% Plot denstiy and density given u in one graph
plot(zoom,logf(zoom,1),'--k',zoom,log(d(zoom,1)),'-k','linewidth',2);

% Find x sorted by the amount of points and the respective indixes
x_lengths = zeros(n,1);
for ind = 1:n
x_lengths(ind,1) = length(x{ind,1});
end
[x_sort,x_ind] = sort(x_lengths);
% Plot values of log density sorted by amount of points
plot(1:244,logf(x_ind,1),'k');
plot(1:244,x_sort,'k');

% Plot daily component scores (normal vs t)
zoom2 = 1:50;

plot(zoom2,u(zoom2,1),'--k',zoom2,ut(zoom2,1),':k','linewidth',2);
yline(0);
plot(zoom2,u(zoom2,2),'--k',zoom2,ut(zoom2,2),':k','linewidth',2);
yline(0);

% Plot component scores vs distribution
norm1 = makedist('Normal','mu',0,'sigma',s2(1));
plot_comparepdf(u(:,1),norm1,"Normal Dist: U-values of 1st component"...
,['N(0,',num2str(s2(1),2),')']);
t1 = makedist('tLocationScale','mu',0,'sigma',s2t(1),'nu',1);
plot_comparepdf(ut(:,1),t1,"t-Dist: U-values of 1st component"...
,['t-dist, \sigma=',num2str(s2t(1),2)]);
norm2 = makedist('Normal','mu',0,'sigma',s2(2));
plot_comparepdf(u(:,2),norm2,"Normal Dist: U-values of 2nd component"...
,['N(0,',num2str(s2(2),2),')']);
t2 = makedist('tLocationScale','mu',0,'sigma',s2t(2),'nu',1);
plot_comparepdf(ut(:,2),t2,"t-Dist: U-values of 2nd component"...
,['t-dist, \sigma=',num2str(s2t(2),2)]);

% Plot Basis fcts
plot(t,B);

% Plot density vs points where #points is 19
```

```
anzges = x_ind(x_sort==19)' ;
plotday(anzges(1),u,ut,logf,logft,d,dt,B,c0,C,Ct,x,t);
%38 - its Sunday may 8 -2016
plotday(anzges(2),u,ut,logf,logft,d,dt,B,c0,C,Ct,x,t);
%77 - its a Thursday june 16 -2016

anzges = x_ind(x_sort==35)' ;
plotday(anzges(1),u,ut,logf,logft,d,dt,B,c0,C,Ct,x,t);

anzges = x_ind(x_sort==0)' ;
plotday(anzges(1),u,ut,logf,logft,d,dt,B,c0,C,Ct,x,t);
```

These functions were used to create plots:

```
function plot_comparepdf(u,pd,titl,distName)
low = min(u,-7);
upp = max(u,7);
binWidth = 1;
binCtrs = low:binWidth:upp; %Bin centers, same thing, it depends on your data
counts = hist(u,binCtrs);
prob = counts / (244 * binWidth);
H = bar(binCtrs,prob,'hist');
set(H,'facecolor',[0.5 0.5 0.5],'DisplayName','assigned u-values');
% get the N(0,1) pdf on a finer grid
hold on;
gri = low:.1:upp;
mypdf = pdf(pd,gri); %requires Statistics toolbox
plot(gri,mypdf,'k','linewidth',2,'DisplayName',distName);
%     title(titl);
%     legend;
hold off;
end

function plot_component(k,B,c0,s2,C,t,S)
lmb0 = exp(B*c0);
lmbplus = exp(B*c0+2*sqrt(s2(k))*B*C(:,k));
lmbmin = exp(B*c0-2*sqrt(s2(k))*B*C(:,k));
plot(t,lmb0,'-k','linewidth',2);
hold on;
plot(t,lmbmin,'--k','linewidth',2);
plot(t,lmbplus,':k','linewidth',2);
hold off;
%     title([S,'Interpretation of component ',int2str(k)]);
end

function [den]=den_eva(xi,ui,t,B,c0,C)
lam = exp(B*c0+ui(1)*B*C(:,1)+ui(2)*B*C(:,2));
int = sum(lam/100*24);
m = length(xi);
den = exp(-int)/factorial(m);
for j = 1:m
[ ~, ix ] = min( abs( t-xi(j) ) );
den=den*lam(ix);
end
end

function plotday(anz,u,ut,logf,logft,d,dt,B,c0,C,Ct,x,t)
u(anz,:)
ut(anz,:)
logf(anz)
logft(anz)
```

```
log(d(anz))
log(dt(anz))
lam_B = exp(B*c0);
sum(lam_B/100*24)
lam_n = exp(B*(c0+C*u(anz,:)'));
sum(lam_n/100*24)
lam_t = exp(B*(c0+Ct*ut(anz,:)'));
sum(lam_t/100*24)
binCtrs = 0:1:24;
counts = hist(x{anz,1},binCtrs);
H = bar(binCtrs,counts,'hist');
set(H,'facecolor',[0.5 0.5 0.5],'DisplayName','assigned u-values');
xlim([0 25])
ylim([0 6])
hold on;
plot(t,exp(B*(c0)),'-k','linewidth',2); %blue
plot(t,exp(B*(c0+C*u(anz,:)')),'--k','linewidth',2); %red
plot(t,exp(B*(c0+Ct*ut(anz,:)')),':k','linewidth',2);
hold off;
end
```

These functions were used to fit the Gaussian model:

```
function [optsm1,optsm2,other] = cv_mcatpp_cyc(x,basis,p)
% [optsm1,optsm2,other] = cv_mcatpp_cyc(x,basis,p)
%
% Cross-validation search of smoothing parameters for temporal MCA
%   with cyclic border condition
%  (Five-fold cross-validation is used)
%
% INPUT:
%   x: Observed time points (n x 1 cell).
%      Each x{i} is a vector containing the data from replication i.
%   basis: B-spline basis parameters. Struct with the following fields:
%       rng: Time range (1 x 2 vector).
%       or: Spline order (integer; 4 is cubic splines).
%       nk: Number of knots (integer). Knots will be equally spaced.
%   p: Number of model components (integer>=0).
%
% OUTPUT:
%   optsm1: Optimal smoothing parameter for the mean (scalar>=0)
%   optsm2: Optimal smoothing parameter for components (scalar>=0)
%           (Returns [] if p=0).
%   other: Additional output. Struct with the following fields:
%       optOF1: Optimal value of the objective function at optsm1.
%       optOF2: Optimal value of the objective function at optsm2.
%                (Returns [] if p=0).
%       smgrid: Grid of smoothing parameters used.
%       OF1grid: Objective function at smgrid for mean-only models.
%       OF2grid: Objective function at smgrid for p-component models.
%                 (Returns [] if p=0).
%
% Programs called: MCATPP_CYC, PRED_MCATPP
%
% Version: June 2018

% Cross-validation
itmax = 10;
smgrid = 10.^(-7:.5:-1)';
Ng = length(smgrid);
% Find optimal sm for mean
disp('Finding optimal sm1')
OF1grid = -Inf(Ng,1);
optOF1 = -Inf;
optsm1 = -Inf;
for i = 1:Ng
disp(['Cross-validating for grid point ' num2str(i) ' of ' num2str(Ng)])
logf = cv_5f(x,basis,0,smgrid(i),0,itmax);
OF1grid(i) = mean(logf);
if OF1grid(i)>optOF1
```

```
optOF1 = OF1grid(i);
optsm1 = smgrid(i);
end
end
% Find optimal sm for components
disp(' ')
disp('Finding optimal sm2')
if p==0
optOF2 = [];
optsm2 = [];
OF2grid = [];
else
OF2grid = -Inf(Ng,1);
optOF2 = -Inf;
for i = 1:Ng
disp(['Cross-validating for grid point ' num2str(i) ' of ' num2str(Ng)])
logf = cv_5f(x,basis,p,optsm1,smgrid(i),itmax);
OF2grid(i) = mean(logf);
if OF2grid(i)>optOF2
optOF2 = OF2grid(i);
optsm2 = smgrid(i);
end
end
end
% Output
other = struct('optOF1',optOF1,'optOF2',optOF2,'smgrid',smgrid,...
'OF1grid',OF1grid,'OF2grid',OF2grid);
end

%%%% -------- AUXILIARY FUNCTIONS

function logf = cv_5f(x,basis,p,sm1,sm2,itmax)
% Five-fold cross-validation for MCATPP
% Computes cross-validated log-densities
% Programs called: MCATPP_CYC, PRED_MCATPP
n = length(x);
logf = -Inf(n,1);
B = round(n/5);
for i = 1:5
itest = ((i-1)*B+1):(i*B);
if i==5
itest = ((i-1)*B+1):n;
end
itrain = setdiff(1:n,itest);
try
[T,c0,C,s2] = evalc('mcatpp_cyc(x(itrain),basis,p,sm1,sm2,itmax)');
catch ME1
c0 = [];
```

```
C = [];
s2 = [];
end
if ~isempty(c0)
[T,logf(itest)] = evalc('pred_mcatpp(x(itest),basis,c0,C,s2)');
end
end
end
```

```
function [c0,C,s2,u,logf] = mcatpp_cyc(x,basis,p,sm1,sm2,itmax)
% [c0,C,s2,u,logf] = mcatpp_cyc(x,basis,p,sm1,sm2,itmax)
%
% Multiplicative Component Analysis for Temporal Point Processes
%   (PCA of log-intensities) with cyclic border condition
%
% INPUT:
%   x: Observed time points (n x 1 cell).
%      Each x{i} is a vector containing the data from replication i.
%   basis: B-spline basis parameters. Struct with the following fields:
%       rng: Time range (1 x 2 vector).
%       or: Spline order (integer; 4 is cubic splines).
%       nk: Number of knots (integer). Knots will be equally spaced.
%   p: Number of model components (integer>=0).
%   sm1: Smoothing parameter for the mean (scalar>=0).
%   sm2: Smoothing parameter for the components (scalar>=0).
%        (All components have norm 1 but the mean does not, so different
%         sm's may be needed to attain the same degree of smoothness).
%   itmax: Maximum number of iterations (integer).
%
% OUTPUT:
%   c0: Mean basis coefficients (q x 1).
%   C: Component basis coefficients (q x p).
%   s2: Component variances (p x 1).
%   u: Individual component scores (n x p).
%   logf: Individual log-densities (n x 1).
%
% External calls: BSPL
%
% Version: June 2018

% Input check
c0 = [];
C = [];
s2 = [];
u = [];
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
n = length(x);

% Data filtering (elimination of data outside RNG)
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

% Initialization
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
Bab0 = bspl([a b],basis.or,knt,0);
Bab1 = bspl([a b],basis.or,knt,1);
```

```
Cyc = [Bab0(1,:)-Bab0(2,:); Bab1(1,:)-Bab1(2,:)];
Pcyc = null(Cyc);


% -----> Initial mean-only model
c0 = (Pcyc*Pcyc')*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)])
err = 1;
iter = 0;
while err>1e-3 && iter<itmax
iter = iter + 1;
c00 = c0;
OF0 = OF;
[gc,Hc] = derivc0(sumB,c0,dt,B0);
gpll = gc-2*sm1*J2*c0;
Hpll = Hc-2*sm1*J2;
direction = Pcyc*((Pcyc'*Hpll*Pcyc)\(Pcyc'*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
% Stopping criterion
if OF<=OF0 || ~all(isfinite(c0))
disp('No further improvement in obj. func. is possible')
c0 = c00;
OF = OF0;
end
err = norm(c0-c00)/norm(c00);
disp(['Iteration: ' num2str(iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(err)])
end

% -----> Sequential PC estimation
C = zeros(q,p);
s2 = zeros(p,1);
u = zeros(n,p);
u2 = zeros(n,p);
for ic = 1:p
if ic==1
P = Pcyc;
```

```
else
P = null([Cyc; C(:,1:ic-1)'*J0]);
end
% Initial estimators
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);
s2(ic) = var(u(:,ic));
else
s2(ic) = s2(ic-1)/2;
end
[u(:,1:ic),u2(:,1:ic),logf] = ...
compeff(sumB,c0,C(:,1:ic),s2(1:ic),dt,B0,m,u(:,1:ic));
OF = mean(logf)-sm1*c0'*J2*c0-sm2*sum(diag(C(:,1:ic)'*J2*C(:,1:ic)));
disp(['---> Computing component ' num2str(ic)])
disp(['Iteration: 0, Pen. loglik: ' num2str(OF)])
err = 1;
iter = 0;
% Iterations
while err>1e-3 && iter<itmax
iter = iter + 1;
% Update C
c00 = C(:,ic);
u00 = u;
u200 = u2;
OF0 = OF;
[gc,Hc] = derivc(sumB,c0,C(:,1:ic),dt,B0,u(:,1:ic),u2(:,1:ic));
gpll = gc-2*sm2*J2*C(:,ic);
Hpll = Hc-2*sm2*J2;
direction = P*((P'*Hpll*P)\(P'*gpll));
OF = -Inf;
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
C(:,ic) = c00-step*direction;
C(:,ic) = C(:,ic)/sqrt(C(:,ic)'*J0*C(:,ic));
[u(:,1:ic),u2(:,1:ic),logf] = ...
compeff(sumB,c0,C(:,1:ic),s2(1:ic),dt,B0,m,u00(:,1:ic));
OF = mean(logf) - sm1*c0'*J2*c0 ...
-sm2*sum(diag(C(:,1:ic)'*J2*C(:,1:ic)));
k = k+1;
end
if OF<=OF0 || ~all(isfinite(C(:,ic)))
disp('No further improvement in obj. func. is possible')
C(:,ic) = c00;
u = u00;
```

```
u2 = u200;
end
% Update s2
s2 = mean(u2,1)';
% Stopping criterion
err = norm(C(:,ic)-c00)/norm(c00);
disp(['Iteration: ' num2str(iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(err)])
end
end
end

%%%%%%------ AUXILIARY FUNCTIONS

function logf = complogf0(sumB,c0,dt,B0,m)
%Computes log-densities for mean-only model
logf = -sum(exp(B0*c0))*dt + sumB*c0 - gammaln(m+1);
end

function [gc0,Hc0] = derivc0(sumB,c0,dt,B0)
% Derivatives of loglik/n w.r.t c0
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 [u,u2,logf] = compeff(sumB,c0,C,s2,dt,B0,m,u_ini)
%Computes random effects and log-pdf using Laplace approximation
[n,p] = size(u_ini);
Phi = B0*C;
logf = zeros(n,1);
u = u_ini;
u2 = u_ini.^2;
for i = 1:n
% Compute log(f(x))
uL = u_ini(i,:);
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:5
uL = uL - D_gi/H_gi;
lmbi = exp(B0*c0+B0*C*uL');
D_gi = -lmbi'*Phi*dt + sumB(i,:)*C - uL./s2';
H_gi = -Phi'*((lmbi*ones(1,p)).*Phi)*dt - diag(1./s2);
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C*uL') - gammaln(m(i)+1) ...
-sum(uL.^2./(2*s2')) - .5*sum(log(2*pi*s2));
logf(i) = gi + (p/2)*log(2*pi) - .5*logdet(-H_gi);
S = (-H_gi)\eye(p);
```

```
u(i,:) = uL;
u2(i,:) = diag(S)' + uL.^2;
end
end

function [gc,Hc] = derivc(sumB,c0,C,dt,B0,u,u2)
% Derivatives of loglik/n w.r.t C(:,end)
% Uses ad-hoc approx of second derivatives and plug-in scores for integrals
[n,p] = size(u);
q = length(c0);
ng = size(B0,1);
lmb = exp(B0*c0*ones(1,n)+B0*C*u');
ulmb = (ones(ng,1)*u(:,p)').*lmb;
u2lmb = (ones(ng,1)*u2(:,p)').*lmb;
gc = (-B0'*mean(ulmb,2))*dt + (sumB'*u(:,p)/n);
Hc = -(B0'*((mean(u2lmb,2)*ones(1,q)).*B0))*dt;
end

function y = logdet(A)
% log(det(A)) for symmetric non-neg A
R = chol(A);
y = 2*sum(log(diag(R)));
end
```

These functions were used to fit the t-model:

```
function [c0,C,s2,u,logf] = mcatpp_cyc_student(x,basis,p,sm1,sm2,itmax,v)
% [c0,C,s2,u,logf] = mcatpp_cyc(x,basis,p,sm1,sm2,itmax)
%
% Multiplicative Component Analysis for Temporal Point Processes
%   (PCA of log-intensities) with cyclic border condition
%
% INPUT:
%   x: Observed time points (n x 1 cell).
%      Each x{i} is a vector containing the data from replication i.
%   basis: B-spline basis parameters. Struct with the following fields:
%       rng: Time range (1 x 2 vector).
%       or: Spline order (integer; 4 is cubic splines).
%       nk: Number of knots (integer). Knots will be equally spaced.
%   p: Number of model components (integer>=0).
%   sm1: Smoothing parameter for the mean (scalar>=0).
%   sm2: Smoothing parameter for the components (scalar>=0).
%        (All components have norm 1 but the mean does not, so different
%         sm's may be needed to attain the same degree of smoothness).
%   itmax: Maximum number of iterations (integer).
%
% OUTPUT:
%   c0: Mean basis coefficients (q x 1).
%   C: Component basis coefficients (q x p).
%   s2: Component variances (p x 1).
%   u: Individual component scores (n x p).
%   logf: Individual log-densities (n x 1).
%
% External calls: BSPL
%
% Version: June 2018

% Input check
c0 = [];
C = [];
s2 = [];
u = [];
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
```

```
n = length(x);

% Data filtering (elimination of data outside RNG)
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

% Initialization
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
Bab0 = bspl([a b],basis.or,knt,0);
Bab1 = bspl([a b],basis.or,knt,1);
Cyc = [Bab0(1,:)-Bab0(2,:); Bab1(1,:)-Bab1(2,:)];
Pcyc = null(Cyc);


% -----> Initial mean-only model
c0 = (Pcyc*Pcyc')*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)])
err = 1;
iter = 0;
while err>1e-3 && iter<itmax
iter = iter + 1;
c00 = c0;
OF0 = OF;
[gc,Hc] = derivc0(sumB,c0,dt,B0);
gpll = gc-2*sm1*J2*c0;
Hpll = Hc-2*sm1*J2;
direction = Pcyc*((Pcyc'*Hpll*Pcyc)\(Pcyc'*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
% Stopping criterion
if OF<=OF0 || ~all(isfinite(c0))
disp('No further improvement in obj. func. is possible')
c0 = c00;
OF = OF0;
end
err = norm(c0-c00)/norm(c00);
disp(['Iteration: ' num2str(iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(err)])
end

% -----> Sequential PC estimation
C = zeros(q,p);
s2 = zeros(p,1);
u = zeros(n,p);
u2 = zeros(n,p);
for ic = 1:p
if ic==1
P = Pcyc;
else
P = null([Cyc; 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);
s2(ic) = var(u(:,ic));
else
s2(ic) = s2(ic-1)/2;
end
[u(:,1:ic),u2(:,1:ic),logf] = ...
compeff(sumB,c0,C(:,1:ic),s2(1:ic),dt,B0,m,u(:,1:ic),v);
OF = mean(logf)-sm1*c0'*J2*c0-sm2*sum(diag(C(:,1:ic)'*J2*C(:,1:ic)));
disp(['---> Computing component ' num2str(ic)])
disp(['Iteration: 0, Pen. loglik: ' num2str(OF)])
err = 1;
iter = 0;
while err>1e-3 && iter<itmax
iter = iter + 1;
c00 = C(:,ic);
u00 = u;
u200 = u2;
OF0 = OF;
[gc,Hc] = derivc(sumB,c0,C(:,1:ic),dt,B0,u(:,1:ic),u2(:,1:ic));
gpll = gc-2*sm2*J2*C(:,ic);
Hpll = Hc-2*sm2*J2;
direction = P*((P'*Hpll*P)\(P'*gpll));
OF = -Inf;
k = 0;
while OF<=OF0 && k<6
step = 0.7^k;
C(:,ic) = c00-step*direction;
C(:,ic) = C(:,ic)/sqrt(C(:,ic)'*J0*C(:,ic));
[u(:,1:ic),u2(:,1:ic),logf] = ...
compeff(sumB,c0,C(:,1:ic),s2(1:ic),dt,B0,m,u00(:,1:ic),v);
OF = mean(logf) - sm1*c0'*J2*c0 ...
-sm2*sum(diag(C(:,1:ic)'*J2*C(:,1:ic)));
k = k+1;
end
if OF<=OF0 || ~all(isfinite(C(:,ic)))
disp('No further improvement in obj. func. is possible')
C(:,ic) = c00;
u = u00;
u2 = u200;
end
u2m = mean(u2,1)';
s2(ic)=u2m(ic);
for k=1:10
s2(ic)=fixs2(s2(ic),u2(:,ic),v);
end
err = norm(C(:,ic)-c00)/norm(c00);
disp(['Iteration: ' num2str(iter) ', Pen. loglik: ' ...
num2str(OF) ', Error: ' num2str(err)])
```

```
end
end
end

%%%%%%------ AUXILIARY FUNCTIONS

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 [u,u2,logf] = compeff(sumB,c0,C,s2,dt,B0,m,u_ini,v)
[n,p] = size(u_ini);
Phi = B0*C;
logf = zeros(n,1);
u = u_ini;
u2 = u_ini.^2;
for i = 1:n
uL = u_ini(i,:);
uL2 = u_ini(i,:);
D_gi = zeros(1,p);
H_gi = eye(p);
for steps = 1:5
uL = uL - D_gi/H_gi;

uL2 = uL.^2;

lmbi = exp(B0*c0+B0*C*uL');

D_gi = -lmbi'*Phi*dt + sumB(i,:)*C - (v+1).*uL./(v.*s2'+uL2);

H_gi = -Phi'*((lmbi*ones(1,p)).*Phi)*dt - diag((v+1)...
.*(v.*s2'-uL2)./(v.*s2'+uL2).^2);
end
gi = -sum(lmbi)*dt + sumB(i,:)*(c0+C*uL') - gammaln(m(i)+1) ...
+ p*gammaln((v+1)/2) - p*gammaln(v/2) - p/2*log(pi*v) ...
- .5*sum(log(s2)) - (v+1)/2*sum(log(1.+(1/v).*uL2./s2'));

logf(i) = gi + (p/2)*log(2*pi) - .5*logdet(-H_gi);
S = (-H_gi)\eye(p);
u(i,:) = uL;
u2(i,:) = diag(S)' + uL.^2;
end
```

```
end

function [gc,Hc] = derivc(sumB,c0,C,dt,B0,u,u2)
[n,p] = size(u);
q = length(c0);
ng = size(B0,1);
lmb = exp(B0*c0*ones(1,n)+B0*C*u');
ulmb = (ones(ng,1)*u(:,p)').*lmb;
u2lmb = (ones(ng,1)*u2(:,p)').*lmb;
gc = (-B0'*mean(ulmb,2))*dt + (sumB'*u(:,p)/n);
Hc = -(B0'*((mean(u2lmb,2)*ones(1,q)).*B0))*dt;
end

function y = logdet(A)
R = chol(A);
y = 2*sum(log(diag(R)));
end

function [s2new]=fixs2(s2alt,u,v)
s2new = mean((1+v).*u./(v+u./s2alt));
end
```

References

Daniel Gervini. Multiplicative component models for replicated point processes. ArXiv 1705.09693, 2017.

Shuang Wu, Hans-Georg Mueller, and Zhen Zhang. Functional data analysis for point processes with rare events. *Statistica Sinica*, 23:1–23, 2013.

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

Douglas C. Montgomery, Elizabeth A. Peck, and G Geoffrey Vining. *Introduction to Linear Regression Analysis*. John Wiley and Sons, Incorporated, fifth edition, 2012.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, second edition, 2008.

C. de Boor. *A Practical Guide to Splines*. Springer, revised edition, 2001.

Alan Julian Izenman. *Modern Multivariate Statistical Techniques*. Springer, 2008.

B. Do Chuong. The multivariate gaussian distribution. 2008. URL http://cs229.stanford.edu/section/gaussians.

N.J. Higham. Cholesky factorization. *Encyclopedia of Mathematics*, 2011. ISBN 1402006098.

Daniel Gervini and Manoj Khanal. Exploring patterns of demand in bike sharing systems via replicated point process models. ArXiv 1802.04755, 2018.