

# Estimation of Non-Linear Covariate Effects with p-Splines

Term Paper {pSplineLocationScale}

Alexander Litwinov  
alexander.litwinov@stud.uni-goettingen.de

Fabian Lukassen  
fabian.lukassen@stud.uni-goettingen.de

Wendi Qu  
wendi.qu@stud.uni-goettingen.de

September 20, 2023

Term Paper for Module M.WIWI-QMW.0011  
(Advanced Statistical Programming with R)

Person Responsible for the Module:  
Prof. Dr. Elisabeth Bergherr  
Supervisor: Quentin Seifert  
Georg-August-Universität Göttingen

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Theoretical Background</b>	<b>2</b>
2.1	Splines . . . . .	2
2.1.1	Polynomial Splines . . . . .	3
2.1.2	Polynomial Splines and the Truncated Power Series . . . . .	4
2.1.3	b-Splines . . . . .	5
2.1.4	p-Splines . . . . .	6
2.2	The RS-Algorithm . . . . .	8
2.3	Location-Scale Model . . . . .	9
<b>3</b>	<b>Usage and implementation</b>	<b>10</b>
3.1	Splines . . . . .	10
3.2	RS-Algorithm . . . . .	11
3.3	User functions . . . . .	13
<b>4</b>	<b>Simulation Study</b>	<b>14</b>
4.1	Study design . . . . .	14
4.2	Results . . . . .	15
4.2.1	Bias . . . . .	16
4.3	Empirical Standard Deviation . . . . .	16
4.4	Mean Squared Error . . . . .	16
<b>5</b>	<b>Conclusion</b>	<b>17</b>

# 1 Introduction

The field of nonparametric regression offers a versatile and powerful approach to modeling the relationships between continuous variables and a dependent variable. Traditional linear models, while valuable in many contexts, often struggle to capture the complexities inherent in real-world applications. Whether due to theoretical limitations or the presence of uncertainty regarding the specific effects of covariates on the response variable, nonparametric regression steps in to provide a flexible and dynamic alternative. In this paper, we will look into nonparametric regression in more detail, with a specific focus on one of its essential tools: p-Splines.

This Paper introduces an R-Package `{pSplineLocationScale}` which is designed for computing p-Splines within the framework of Additive Models for Location and Scale, which also provides an optimization algorithm to determine the ideal model fit.

We start by giving a theoretical introduction to splines, RS algorithm and additive Models for Location and Scale. In the following we will explain the implementation and function of our package `{pSplineLocationScale}`, which will be tested in the next section using a simulation study. Lastly, we will evaluate the package `{pSplineLocationScale}` and this term paper in conclusion

## 2 Theoretical Background

We start by giving an introduction to splines in general, after which we look specifically at b- and p-Splines. Finally, the Additive Model for Location and Scale will be discussed in the context of this paper.

### 2.1 Splines

To begin with the topic of splines we introduce a method that enables the flexible modeling of the relationship between a continuous covariate and a dependent variable. These methods, known as scatter plot smoothers, aim to find a smooth function that represents the effect of the covariate. The data is presented as pairs of observations  $(y_i, z_i), i = 1, \dots, n$ , where  $y_i$  is the response variable and  $z_i$  is the corresponding value of the covariate. The basic idea is to explain the response variable by a deterministic function of the covariate, along with an error term:

$$y_i = f(z_i) + \varepsilon_i \tag{1}$$

By making various assumptions about the function, different modeling options are available. Constraints are often imposed on the smoothness of the function to simplify the estimation process. Similar to the classical linear model, assumptions are made about the error term, assuming it is independent, identically distributed, and has a mean of zero and constant variance.

$$E(\varepsilon_i) = 0 \quad \text{and} \quad \text{Var}(\varepsilon_i) = \sigma^2, \quad i = 1, \dots, n \quad (2)$$

The objective is to model the expected value of the response variable through the function  $f$ . In certain cases, it is assumed that the errors follow a normal distribution, especially when constructing confidence intervals for the function  $f$ .

Unlike linear regression models, these models typically involve a large number of parameters without individual interpretations. Despite this, the term "nonparametric regression" is still used. In fact, the term "nonparametric" is not consistently appropriate. While it accurately characterizes kernel smoothers and running statistics, spline smoothers possess parameters, even though their quantity may be substantial. It may be more suitable to refer to these methods as "overparametric" techniques or "anonymous" models, as the parameters lack scientific interpretation. Nevertheless, we will also use the term "nonparametric" to be consistent with the literature.

### 2.1.1 Polynomial Splines

In the initial approach for nonparametric regression, we consider polynomial splines or regression splines, which are closely related to polynomial regression models

$$f(z_i) = \gamma_0 + \gamma_1 z_i + \dots + \gamma_l z_i^l. \quad (3)$$

These models represent the relationship between the covariate  $z$  and the response variable  $y$  as a polynomial of degree  $l$ . We can determine the regression coefficients and the complete function  $f$  using ordinary least squares, similar to linear models. However, a purely polynomial model may not adequately estimate nonlinear functions. Higher degree polynomials can capture some features, but they result in a wiggly and unsatisfactory estimation in other regions. To enhance the flexibility of the polynomial model, we partition the covariate domain into intervals and estimate separate polynomials in each interval. This allows us to capture local patterns in the data. However, the piecewise polynomial model has a disadvantage: the separate polynomials on each interval do not lead to a smooth overall function, and the function values may differ at the interval boundaries. To address this, we introduce smoothness restrictions at the interval boundaries, leading to the concept

of polynomial splines. Polynomial splines define local polynomials on intervals but also require the resulting function to be continuously differentiable at the interval boundaries, ensuring smoothness. By specifying the knots, which determine the partition of the covariate domain, and the degree of the spline, we obtain a piecewise polynomial function that satisfies the desired smoothness restrictions. The choice of the spline degree determines the global smoothness, while the number and location of knots influence the diversity of available functions. Using more knots increases the number of piecewise polynomials constituting the spline. In the subsequent sections, we will delve into the influence of spline degree and knot configuration in more detail. Utilizing polynomial splines in nonparametric regression, we need a representation of the set of polynomial splines for a given degree and knot configuration. This can be achieved through different but equivalent approaches, one is the truncated power series and the other is the b-Splines approach, which will be discussed in the following sections (Fahrmeir et al., 2022).

### 2.1.2 Polynomial Splines and the Truncated Power Series

The model discussed in this subsection involves a global polynomial of degree  $l$ , with the intercept denoted as  $y_1$  instead of  $y_0$ . The coefficient of the highest polynomial term varies at each knot. This specification allows for the use of local polynomials within the intervals defined by the knots while maintaining overall smoothness. The truncated function  $(z - k_j)_+^l$  ensures a smooth change in slope, preserving the properties of a polynomial spline. Each polynomial spline of degree  $l$ , with knots  $\kappa_1 < \dots < \kappa_m$ , can be expressed as a linear combination of  $d = m + l - 1$  basis functions:

$$\begin{aligned} B_1(z) &= 1, B_2(z) = z, \dots, B_{l+1}(z) = z^l, \\ B_{l+2}(z) &= (z - \kappa_2)^l, \dots, B_d(z) = (z - \kappa_{m-1})^l + \end{aligned} \tag{4}$$

Approximating the nonparametric regression problem can be achieved by employing polynomial splines.

$$y_i = f(z_i) + \epsilon_i = \sum_{j=1}^d \gamma_j B_j(z_i) + \epsilon_i$$

$B_j$  is referred to as basis functions because they can uniquely represent all polynomial splines. To differentiate between different bases for the function space of splines, this specific basis is known as the truncated power series basis (TP basis).

$$Z = \begin{pmatrix} B_1(z_1) & \dots & B_d(z_1) \\ \vdots & \ddots & \vdots \\ B_1(z_n) & \dots & B_d(z_n) \end{pmatrix} = \begin{pmatrix} 1 & z_1 & \dots & z_l \\ 1 & (z_1 - \kappa_2)^l & \dots & (z_1 - \kappa_{m-1})^l \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z_n & \dots & z_l \end{pmatrix} \quad (5)$$

The parameters are calculated using the least squares estimate, which is equivalent to the estimation in a linear model:

$$y = Z\gamma + \epsilon \quad (6)$$

$$\hat{\gamma} = (Z'Z)^{-1}Z'y. \quad (7)$$

As already mentioned, the individual estimated parameters, the individual estimated parameters  $\hat{\gamma}_j$  do not offer substantial interpretive value. Rather, our primary concern is focused on examining the shape and form of the estimated function, which is a result of the estimated coefficients.

$$f(z) = z' \hat{\gamma} \quad (8)$$

Cubic splines are commonly chosen as a default option due to their ability to generate a smooth function that is twice continuously differentiable (Fahrmeir et al., 2022). There are various methods for determining the distribution of knots, in this paper, we will utilize equidistant knots as our chosen approach. To determine the number of knots we have to introduce a penalty term. But before exploring the introduction of a penalty term, it is necessary to introduce b-Splines as a flexible and effective method for data modeling.

### 2.1.3 b-Splines

b-Spline basis functions are formed by smoothly fusing piecewise polynomials at the knots to meet specific smoothness requirements. Specifically, each b-Spline basis function comprises  $(l+1)$  polynomial pieces of degree  $l$ , joined together in an  $(l-1)$ -times continuously differentiable manner. The function  $f(z)$  can be represented as a linear combination of  $d = m + l - 1$  basis functions. The basis functions are defined recursively as:

$$B_j^l(z) = \frac{z - \kappa_{j-l}}{\kappa_j - \kappa_{j-l}} B_{j-1}^{l-1}(z) + \frac{\kappa_{j+1} - z}{\kappa_{j+1} - \kappa_{j+1-l}} B_j^{l-1}(z),$$

where  $I$  is an indicator function defined as:

$$I(\kappa_j \leq z < \kappa_{j+1}) = \begin{cases} 1, & \kappa_j \leq z < \kappa_{j+1}, \\ 0, & \text{otherwise} \end{cases} \quad j = 1, \dots, d-1 \quad (9)$$

In order to employ the recursive definition of b-Splines for calculating the basis functions, we require  $2l$  outer knots, in addition to the inner knots  $\kappa_1, \dots, \kappa_m$ . This yields an extended knots sequence  $\kappa_{1-l}, \kappa_{1-l+1}, \dots, \kappa_{m+l-1}, \kappa_{m+l}$ . (Fahrmeir et al., 2022).

As can be seen in Figure 1, the smoothness of the spline depends on two parameters, the degree and especially the number of knots. A too-small number of knots might lead to a biased model, whereas too many knots result in an overfitted model. To find out which number of knots is optimal, p-Splines are introduced in the following.

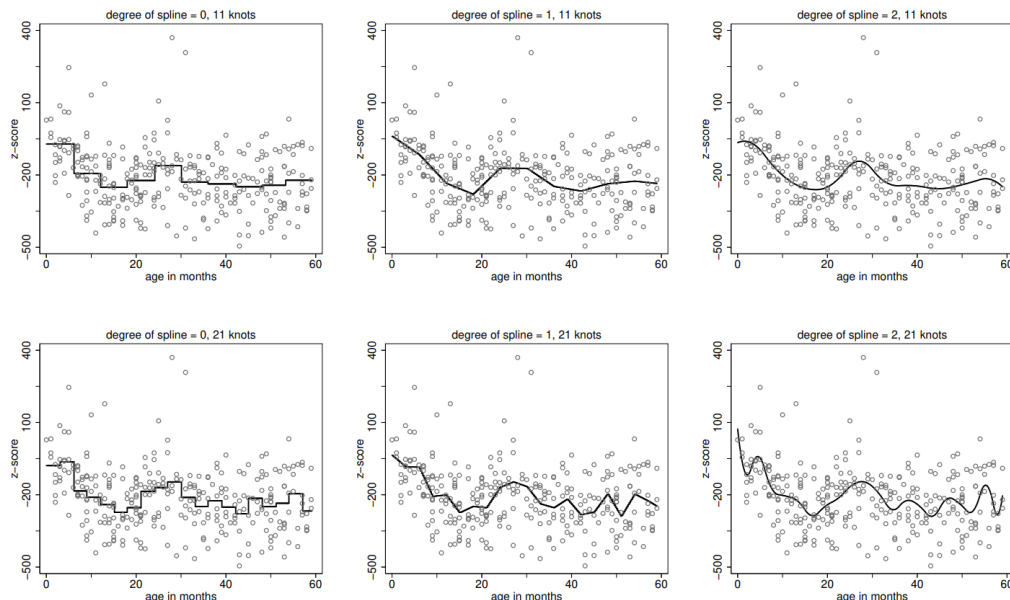


Figure 1: Polynomials with different degree and number of knots (Fahrmeier et al., 2019)

### 2.1.4 p-Splines

Penalized Splines (p-Splines) aim to approximate the function  $f(z)$  using a polynomial spline with many knots (around 20-40), providing flexibility to approximate complex functions. To avoid overfitting, a penalty term  $\lambda$  is added, minimizing a

penalized least squares (PLS) criterion instead of the regular least squares criterion. This approach allows for a more robust and controlled fitting of the function to the data (Eilers and Marx, 1996).

A convenient way to approximate the derivative of b-Splines is by using the first differences of the coefficient vector. To ensure smoothness and prevent large first derivative values, specialized penalties are introduced based on these differences. Additionally, higher-order differences can be used to achieve smoothness in higher-order derivatives, resulting in the penalized residual sum of squares.

$$PLS(\lambda) = \sum_{i=1}^n (y_i - \sum_{j=1}^d \gamma_j B_j(z_i))^2 + \lambda \sum_{j=r+1}^d (\Delta^r \gamma_j)^2, \quad (10)$$

where  $\Delta^r$  denotes the  $r$ th order differences (O'Sullivan, 1986).

When the smoothing parameter ( $\lambda$ ) is set to a large value ( $\lambda \rightarrow \infty$ ), the fit approaches a polynomial of degree  $r-1$  when using  $r$ th-order differences, given that the degree of the spline is large enough. As in Figure ?? illustrated indicates that higher values of  $\lambda$  lead to a smoother and more simplified approximation of the function, resembling a polynomial of the specified degree, and vice versa (Fahrmeir et al., 2022).

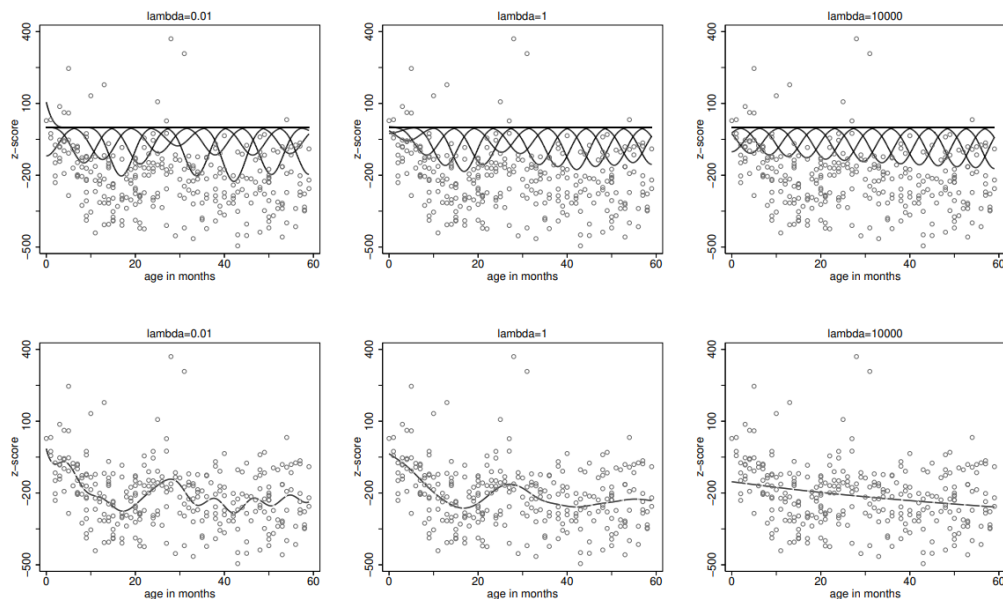


Figure 2: (Fahrmeir et al., 2022)



## 2.2 The RS-Algorithm

The RS-algorithm by Stasinopoulos is a fundamental method in the context of this Package {pSplineLocationScale} in order to estimate the parameters  $\mu$  and  $\sigma$ . This algorithm consists of three nested components: the outer iteration, the inner iteration (also known as the local scoring or GLIM algorithm), and the modified backfitting algorithm. These components work collaboratively to iteratively estimate distribution parameters until convergence is achieved (Stasinopoulos et al., 2017)..

The outer iteration, the algorithm fits models for various distribution parameters. It proceeds as follows: first, it fits a model for  $\mu$  (the location parameter) using the latest estimates of  $\hat{\sigma}$ ,  $\hat{\beta}$ , and  $\hat{\gamma}$ . Then, it fits a model for  $\sigma$  (the scale parameter) given the latest estimates of  $\hat{\mu}$ ,  $\hat{\beta}$ , and  $\hat{\gamma}$ .

After fitting these models, the algorithm calculates the global deviance, which is equal to minus twice the current fitted log-likelihood. If the global deviance has converged, the algorithm stops; otherwise, it repeats the process. Importantly, the algorithm only requires initial values for the distribution parameter vectors ( $\mu$ ,  $\sigma$ ,  $\beta$ ,  $\gamma$ ) and has been found to be stable and efficient even with simple starting values. The inner iteration focuses on fitting a single distribution parameter  $\theta_k$  (e.g.,  $\mu$ ,  $\sigma$ ,  $\nu$ ,  $\tau$ ) using a local scoring algorithm similar to that employed in generalized linear models (GLMs). The core equation for this iteration is:

$$z_k = \eta_k + w_k^{-1} \circ u_k \quad (11)$$

In this equation,  $z_k$  represents the modified response variable,  $\eta_k$  stands for the predictor vector of the  $k$ th parameter vector  $\theta_k$ , and  $w_k^{-1} \circ u_k$  signifies the Hadamard element-wise product of two vectors.

The score function  $u_k$  is defined as:

$$\frac{\partial \ell}{\partial \eta_k} = \left( \frac{\partial \ell}{\partial \theta_k} \right) \circ \left( \frac{d\theta_k}{d\eta_k} \right) \quad (12)$$

Here,  $\ell$  denotes the log-likelihood, and  $\frac{\partial \ell}{\partial \eta_k}$  represents the score function's first derivative with respect to the predictor.

The weights  $w_k$  utilized in the iteration are calculated based on three different methods to define  $f_k$ , which depend on the available information for the specific distribution. These methods encompass Fisher's scoring algorithm, the standard Newton-Raphson scoring algorithm, and a quasi-Newton scoring algorithm.

The inner iteration adjusts the predictor  $\eta_k$  using step parameters and halving steps to prevent over-jumping and ensure the convergence of the global deviance. In summary, the RS algorithm combines these nested iterations to iteratively estimate distribution parameters until convergence is achieved (Stasinopoulos et al., 2017). A flowchart of the RS-Algorithm can be found in Figure 3.

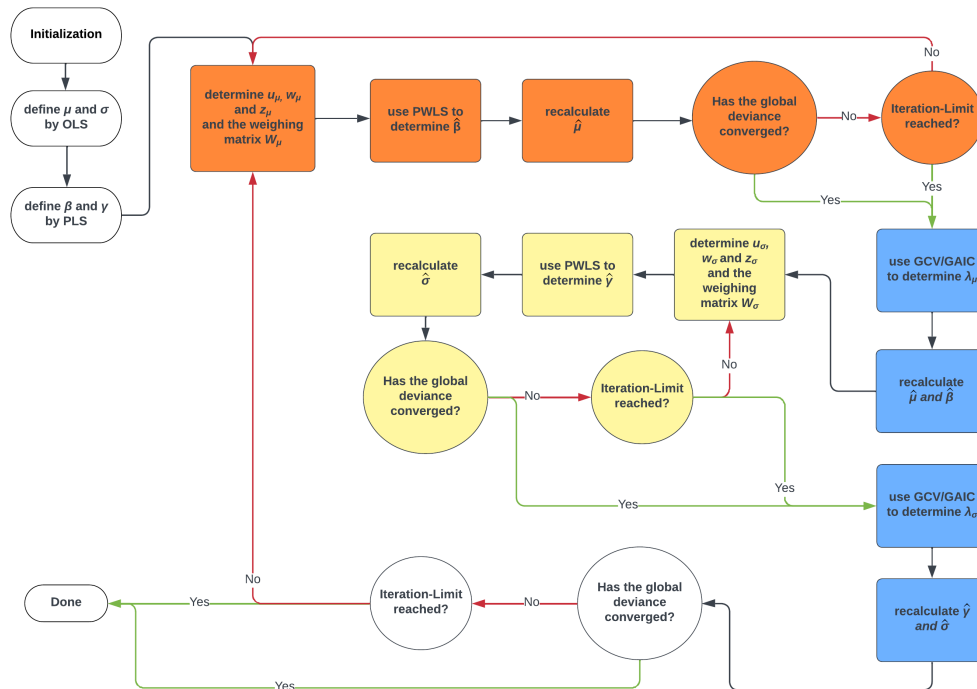


Figure 3: The Flowchart of the RS-Algorithm

## 2.3 Location-Scale Model

In this paper, after discussing the general idea of penalized b-Splines, we now explore the application of splines in the context of the Gaussian location-scale regression model. Unlike the usual linear regression model with a normally distributed response variable  $y_i \sim N(\mu, \sigma^2)$ , the location-scale regression model allows the mean (= the location) and the variance (= the scale) to vary across the data domain, leading to the following model (Stasinopoulos et al., 2017):

$$y_i \sim N(\mu_i, \sigma_i^2) \quad (13)$$

This modeling approach provides flexibility in capturing the complex relationships between the response variable and covariates. By incorporating splines for both the mean and scale across the data domain, the resulting Gaussian additive model for location and scale (GAMLSS) can handle various distributions.

## 3 Usage and implementation

In this section, we will provide an overview of the implementation of the R-package `{pSplineLocationScale}`. Firstly, we start with the code architecture, examining the basic algorithm for constructing the b- and p-Splines in the first part. In the second part, we will cover the RS-Algorithm. Finally, the third part will merge the first two components while introducing package-specific features, including functions designed for result summarization and visualization.

### 3.1 Splines

The code comprises a set of functions designed to facilitate the creation of b-Spline basis functions and perform penalized least squares regression using b-Splines. We will break down the key functions and their roles within the package.

The `create_knots_for_cox_de_boor()` function initiates the b-Spline construction by computing knots. It determines both inner and outer knots, where the number of inner knots can be specified by the user, and the number of outer knots is twice the polynomial degree ( $2l$ ). The function takes input parameters, including `knot_count`, `lower_bound`, `upper_bound`, and `poly_degree` (= the degree of the polynomial). The resulting knots sequence is stored in the variable `knots_sequence`.

The `indicator()` function is a utility function that returns an indicator value (1 or 0) based on the position relative to lower and upper bounds. It evaluates whether the position is greater than or equal to the lower bound and less than the upper bound, returning 1 for true and 0 for false. The function accepts the position, lower bound, and upper bound as input.

The `cox_de_boor()` function implements the Cox de Boor algorithm for calculating b-Spline basis functions. It computes the value of the b-Spline basis function at a given position within a specified range defined by lower and upper bounds. The function takes the position, knots sequence, lower index, and upper index as input.

and returns the value of the basis function.

The `create_basis_function_vector()` function computes the b-Spline basis function vector for a given set of knots and a vector of  $x$  values. It utilizes the `cox_de_boor()` function to calculate individual basis functions for each  $x$  value. The function accepts the `knots_sequence`, `vector_x`, `poly_degree`, and `basis_function_knot_position` as input, returning the basis function vector.

The `create_basis_function_matrix()` function combines b-Spline basis function vectors into a matrix, resulting in the design matrix  $Z$  for the b-Splines. Each column of the matrix corresponds to a different basis function. The function accepts the following parameters: knot count, lower and upper bounds, polynomial degree, and the input vector of  $x$  values. It returns the b-Spline basis function matrix.

The functions `create_diff_matrix()` and `create_penalty_matrix_K()` are responsible for generating the difference matrix used in penalized b-Splines and calculating the associated penalty matrix  $K$ . The order of the difference matrix is determined by the user, along with other parameters such as polynomial degree and knot count. The penalty matrix  $K$  is computed as the product of the transpose of the difference matrix and the difference matrix itself.

The `PLS_b_Splines()` function performs penalized least squares regression using b-Splines. It computes the b-Spline design matrix  $Z$ , the penalty matrix  $K$ , and the coefficients  $\gamma$ . The hyperparameter  $\lambda$  controls the amount of penalization. The function takes input parameters, including knot count, lower and upper bounds, polynomial degree, vectors of  $x$  values and response  $y$  values, and the hyperparameter  $\lambda$ . It returns a list containing the  $Z$  matrix and the coefficients  $\gamma$ .

This code facilitates the construction of b-Spline basis functions and their application in penalized least squares regression, providing essential tools for working with spline-based models.

## 3.2 RS-Algorithm

As already mentioned, the RS-Algorithm is the core component of the package, serving as the primary mechanism for estimating a location-scale model using p-Splines. This algorithm revolves around the `outer()` function, which, in turn, invokes two inner functions: `inner_mu()` and `inner_sigma()`. It also relies on the `init()` func-

tion for initialization and the `global_deviance()` function to monitor convergence.

The `init()` function initializes the starting values for the RS algorithm. Given input data vectors  $x$  and  $y$ , it employs a simple linear model to estimate the initial values of  $\hat{\mu}$  and  $\hat{\sigma}$ . The estimated mean and variance of the initial linear model serve as the initial estimates. This function returns a list containing  $\hat{\mu}$  and  $\hat{\sigma}$ .

The `global_deviance()` function calculates twice the negative log-likelihood of the model, serving as a convergence criterion within the RS algorithm. It takes the response vector  $y$ , the current estimate of  $\hat{\mu}$ , and the current estimate of  $\hat{\sigma}$  as inputs and returns the computed deviance.

The `outer()` function is the heart of the RS algorithm, performing the iterative updates of  $\hat{\mu}$  and  $\hat{\sigma}$  until convergence is reached or a specified maximum number of iterations (`maxit`) is reached. It accepts various input parameters related to data and algorithm settings, including the initial values of  $\hat{\mu}$  and  $\hat{\sigma}$ . This function calls `inner_mu()` and `inner_sigma()` to update  $\hat{\mu}$  and  $\hat{\sigma}$ , respectively. It also handles the optimization of lambda values (`lambda_mu` and `lambda_sigma`) based on the specified method (`opt_method`). The algorithm terminates when convergence is achieved, and it returns a list containing the final estimates of  $\hat{\mu}$ ,  $\hat{\sigma}$ ,  $\hat{\beta}$ , and  $\hat{\gamma}$ .

Both `inner_mu()` and `inner_sigma()` functions follow similar workflows. They calculate variables such as `u_mu`, `w_mu`, `z_mu`, and `W_mu` (or equivalent variables for  $\sigma$ ) for subsequent computations. The b-Spline coefficients ( $\hat{\beta}$  or  $\hat{\gamma}$ ) are updated using Penalized Weighted Least Squares (PWLS). Afterward,  $\hat{\mu}$  or  $\hat{\sigma}$  is recalculated based on the updated coefficients. Convergence is checked using the `global_deviance()` function within a limited number of iterations. Finally, the optimal lambda value is determined through either the Generalized Cross Validation (GCV) or Generalized Akaike Information Criterion (GAIC) method.

The `compute_lambda_gcv()` function is responsible for optimizing lambda values ( $\lambda_{\mu}$  or  $\lambda_{\sigma}$ ) using either GCV or GAIC methods. It conducts a grid search over a range of lambda values, calculating a corresponding score for each. The optimal lambda value is chosen as the one that yields the best score.

In summary, the RS-Algorithm is designed to estimate a location-scale model using b-Splines with penalization. It iteratively updates the mean and variance parameters while optimizing the penalization parameters (`lambda_mu` and `lambda_sigma`) to

achieve convergence. The algorithm provides flexibility in choosing optimization methods and accuracy settings (Riebl).

### 3.3 User functions

To effectively utilize this package, users primarily interact with the `lmls_bspline` function, which serves as the central entry point for running the Algorithm. When using this function, users are required to provide essential input arguments, which are `vector_x` and `response_vector_y`. While this function offers default values for the other arguments, it also allows users to customize these settings to suit their specific needs.

For b-Spline modeling, the default configurations include employing 40 knots (`knot_count = 40`), utilizing b-Spline polynomials with a degree of three (`poly_degree = 3`), and constructing a penalty matrix that involves difference matrices with an order of two (`order_difference_matrix_r = 2`). These default settings are chosen to be broadly applicable in practical scenarios involving b-Splines.

The algorithm's convergence criteria are preset with a maximum iteration limit of 50 (`max_iterations = 50`). Additionally, initial values for the smoothing parameters are predefined: `lambda_mu = 3` and `lambda_sigma = 500`. By default, the optimization method chosen for the `opt_method` argument is the Generalized Cross Validation (GCV). The `lambda_acc` parameter, which influences the algorithm's accuracy, is set at 50 by default. Users, however, retain the ability to fine-tune these parameters to better align with the characteristics of their specific dataset.

Before executing the function, robust input validation is performed. This validation checks critical aspects, such as ensuring that `vector_x` and `response_vector_y` are numeric and have the same length. If these criteria are not met, the function generates an error message. Furthermore, if a user specifies a knot count that exceeds the number of observations in the dataset, a warning is issued.

Once input validation is successfully completed, the function proceeds with the initialization of `mu_hat` and `sigma_hat` through the `init()` function. Subsequently, the `PLS_b_Splines()` function is invoked to calculate b-Splines, initializing values for `beta_hat` and `gamma_hat`. Finally, the RS-algorithm's `outer()` function is employed to estimate the location-scale model. The function returns the output of the `outer()` function as an `lmls_bspline` object.

To enhance clarity and interpretability, supplementary functions for plotting and summarizing results have been thoughtfully implemented. For example, the `plot_lmls_bspline` function is designed to compute a  $(1 - \alpha)$  confidence interval. It requires an `lmls_bspline` object as input and offers users the flexibility to customize the alpha parameter (set to 0.05 by default). Notably, this function includes safeguards to ensure that the provided input is indeed an `lmls_bspline` object and generates an error message if this condition is not met. It then proceeds to calculate the confidence interval and visualize the estimation using the `ggplot2` package.

Similarly, the `summary_lmls_bspline` function plays a vital role in providing insights into the estimated model. It conducts preliminary checks on input arguments and offers supplementary information, including the function call, the chosen optimization method, the estimated optimal values for `lambda_mu` and `lambda_sigma`, and, in cases of successful convergence, it reports the number of iterations required until convergence and the global deviance at that point. In instances where convergence is not achieved, it explicitly communicates this fact. Finally, the function provides the conclusive results for the vectors `mu_hat`, `sigma_hat`, `beta_hat`, and `gamma_hat`.

## 4 Simulation Study

Within this section, we describe the structure of our simulation study and present the results. The code and data relating to the simulation study can be found in the Appendix.

### 4.1 Study design

The design of our simulation study follows the **ADEMP**-design by Morris et al. (2019), which stands for **A**ims, **D**ata-generating mechanisms, **E**stimands and other targets, **M**ethods, and **P**erformance measures.

**Aims:** Our aim is to assess the efficiency of p-Spline estimation in our package for additive models for location and scale, with a particular focus on understanding bias and efficiency variations across different sample sizes.

**Data-generating mechanism:** For generating the simulation data, we utilize a

polynomial in the following form:

$$y = \sin(ax) + \cos(bx) + cx + \epsilon$$

where:

$$\epsilon \sim \mathcal{N}(0, (d \exp(e \sin(fx) + gx) + h)^2)$$

We select ten different sets of parameters for  $a, b, c, d, e, f, g$ , and  $h$  to create ten distinct polynomials. Each polynomial exhibits unique characteristics regarding the structure of  $\mu$  and  $\sigma$ .

Additionally, we use each polynomial to generate data for various sample sizes,  $n = \{50, 100, 200, 300, 500\}$ , resulting in a grid of 50 distinct datasets. These datasets are each generated  $X$  times in independent simulation runs.

**Estimands:** In accordance with our aims, we have  $\hat{\mu}$  and  $\hat{\sigma}$  as our estimands.

**Methods:** To assess the effectiveness of our software package, we determine  $\hat{\mu}$  and  $\hat{\sigma}$  by utilizing our package and then compare these estimations with the true values of  $\mu$  and  $\sigma$ . To achieve this, we employ the `lmls_bspline()` function across ten different polynomials while varying the sample size across five different levels,

**Performance Measure:** We evaluate our package's performance using bias, empirical standard deviation (ESD), and mean squared error (MSE). The definitions and estimates of these performance measures, adapted from Morris et al. (2019), see in Appendix A.

## 4.2 Results

This section presents the results of the simulation study. We start with the results of the bias, after which we move on to the empirical standard deviation and finally consider the mean squared error.

Before we get to the presentation of the results, it must be mentioned in advance that our data is flawed. As can be seen in the following tables and figures, there are data points that are exceptionally high. Unfortunately, we did not come to a solution of the problem. For this reason, the results are only presented in the following, but not discussed. However, in the last part, the conclusion, we will come back to the interpretation of the simulation study.



### 4.2.1 Bias

The results of the  $\mu$  and  $\sigma$  bias can be found in Figure 3.

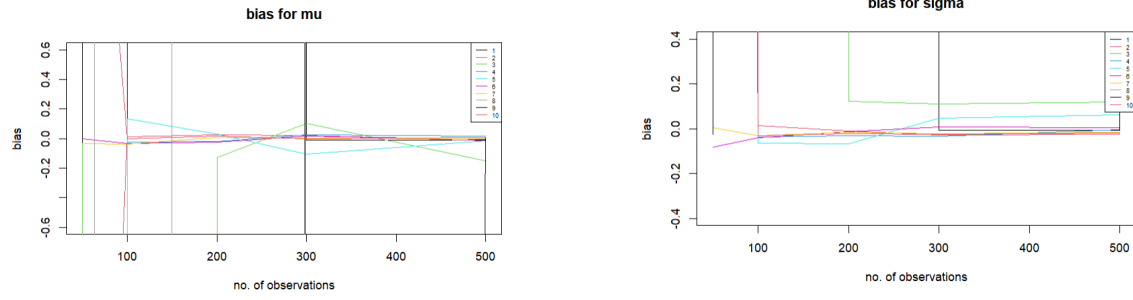


Figure 4: Bias for  $\mu$  and  $\sigma$

### 4.3 Empirical Standard Deviation

The results of the empirical standard deviation of  $\mu$  and  $\sigma$  are in Figure 4.

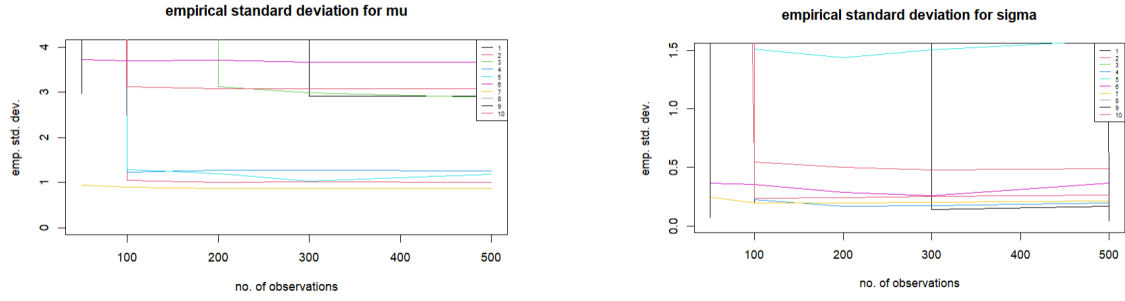


Figure 5: The mean squared error of  $\mu$  and  $\sigma$

### 4.4 Mean Squared Error

The results of the mean squared error of  $\mu$  and  $\sigma$  can be found in Figure 5.

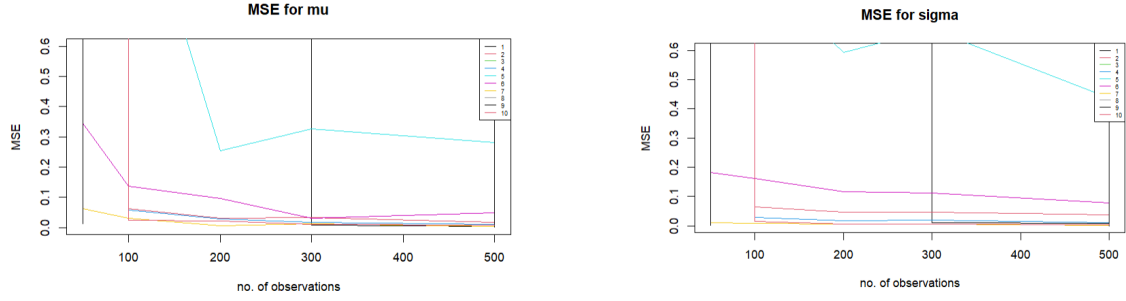


Figure 6: Bias for  $\mu$  and  $\sigma$

## 5 Conclusion

In this report, we presented our R implementation for the estimation of non-linear covariate effects with p-Splines. We presented a method for generating b- and p-Splines with an optimization algorithm. Furthermore, this was implemented in our R-package `{pSplineLocationScale}`. In our simulation study we could have shown that our package and p-splines are in general a very good way to model nonlinear data, however, this was denied to us due to our flawed simulation study.

Finally, we want to come back to the simulation study. As already mentioned, our data are erroneous due to a problem we cannot identify and thus cannot be interpreted in a meaningful way. The appendix contains a table with the results of the bias for  $\mu$  and  $\sigma$  as an example. The values have not been interpreted or explained, as this is not possible (see appendix). Nevertheless, we want to mention the efficacy of the splines and not least of the RS algorithm, even if we could not prove it.

## References

- P. H. Eilers and B. D. Marx. Flexible smoothing with b-splines and penalties. *Statistical science*, 11(2):89–121, 1996.
- L. Fahrmeir, T. Kneib, S. Lang, and B. D. Marx. Regression models. In *Regression: Models, methods and applications*, pages 23–84. Springer, 2022.
- T. P. Morris, I. R. White, and M. J. Crowther. Using simulation studies to evaluate statistical methods. *Statistics in medicine*, 38(11):2074–2102, 2019.
- F. O’Sullivan. A statistical perspective on ill-posed inverse problems. *Statistical science*, pages 502–518, 1986.
- H. Riebl. Location-scale regression and the lmls package. *Age [weeks]*, 20(30):40.
- M. D. Stasinopoulos, R. A. Rigby, G. Z. Heller, V. Voudouris, and F. De Bastiani. *Flexible regression and smoothing: using GAMLSS in R*. CRC Press, 2017.

## Appendix

Performance Measure	Definition	Estimate	Monte Carlo SE of Estimate
Bias	$E[\hat{\theta}] - \theta$	$\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} \hat{\theta}_l - \theta$	$\sqrt{\frac{1}{n_{\text{sim}}(n_{\text{sim}}-1)} \sum_{l=1}^{n_{\text{sim}}} (\hat{\theta}_l - \bar{\theta})^2}$
EmpSE	$\sqrt{\text{Var}(\hat{\theta})}$	$\sqrt{\frac{1}{n_{\text{sim}}-1} \sum_{l=1}^{n_{\text{sim}}} (\hat{\theta}_l - \bar{\theta})^2}$	$\frac{\widehat{\text{EmpSE}}}{\sqrt{2(n_{\text{sim}}-1)}}$
Relative % increase in precision (B vs A) <sup>a</sup>	$100 \left( \frac{\text{Var}(\hat{\theta}_A)}{\text{Var}(\hat{\theta}_B)} - 1 \right)$	$100 \left( \left( \frac{\widehat{\text{EmpSE}}_A}{\widehat{\text{EmpSE}}_B} \right)^2 - 1 \right)$	$200 \left( \frac{\widehat{\text{EmpSE}}_A}{\widehat{\text{EmpSE}}_B} \right)^2 \sqrt{\frac{1 - \text{Corr}(\hat{\theta}_A, \hat{\theta}_B)^2}{n_{\text{sim}} - 1}}$
MSE	$E[(\hat{\theta} - \theta)^2]$	$\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} (\hat{\theta}_l - \theta)^2$	$\sqrt{\frac{\sum_{l=1}^{n_{\text{sim}}} [(\hat{\theta}_l - \theta)^2 - \widehat{\text{MSE}}]^2}{n_{\text{sim}}(n_{\text{sim}}-1)}}$
Average ModSE <sup>a</sup>	$\sqrt{E[\widehat{\text{Var}}(\hat{\theta})]}$	$\sqrt{\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} \widehat{\text{Var}}(\hat{\theta}_l)}$	$\sqrt{\frac{\widehat{\text{Var}}[\widehat{\text{Var}}(\hat{\theta})]}{4n_{\text{sim}} \times \widehat{\text{ModSE}}}}$ <sup>b</sup>
Relative % error in ModSE <sup>a</sup>	$100 \left( \frac{\widehat{\text{ModSE}}}{\widehat{\text{EmpSE}}} - 1 \right)$	$100 \left( \frac{\widehat{\text{ModSE}}}{\widehat{\text{EmpSE}}} - 1 \right)$	$100 \left( \frac{\widehat{\text{ModSE}}}{\widehat{\text{EmpSE}}} \right) \sqrt{\frac{\widehat{\text{Var}}[\widehat{\text{Var}}(\hat{\theta})]}{4n_{\text{sim}} \times \widehat{\text{ModSE}}^4} + \frac{1}{2(n-1)}}$ <sup>b</sup>
Coverage	$\Pr(\hat{\theta}_{\text{low}} \leq \theta \leq \hat{\theta}_{\text{upp}})$	$\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} 1(\hat{\theta}_{\text{low},l} \leq \theta \leq \hat{\theta}_{\text{upp},l})$	$\sqrt{\frac{\widehat{\text{Cover}} \times (1 - \widehat{\text{Cover}})}{n_{\text{sim}}}}$
Bias-eliminated coverage	$\Pr(\hat{\theta}_{\text{low}} \leq \bar{\theta} \leq \hat{\theta}_{\text{upp}})$	$\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} 1(\hat{\theta}_{\text{low},l} \leq \bar{\theta} \leq \hat{\theta}_{\text{upp},l})$	$\sqrt{\frac{\text{B-E Cover} \times (1 - \text{B-E Cover})}{n_{\text{sim}}}}$
Rejection % (power or type I error)	$\Pr(p_l \leq \alpha)$	$\frac{1}{n_{\text{sim}}} \sum_{l=1}^{n_{\text{sim}}} 1(p_l \leq \alpha)$	$\sqrt{\frac{\widehat{\text{Power}} \times (1 - \widehat{\text{Power}})}{n_{\text{sim}}}}$

<sup>a</sup>Monte Carlo SEs are approximate for *Relative % increase in precision*, *Average ModSE*, and *Relative % error in ModSE*.

<sup>b</sup> $\widehat{\text{Var}}[\widehat{\text{Var}}(\hat{\theta})] = \frac{1}{n_{\text{sim}}-1} \sum_{l=1}^{n_{\text{sim}}} (\widehat{\text{Var}}(\hat{\theta}_l) - \frac{1}{n_{\text{sim}}} \sum_{j=1}^{n_{\text{sim}}} \widehat{\text{Var}}(\hat{\theta}_j))^2$ .

Figure 7: Performance measures: definitions, estimates, and Monte Carlo standard errors (Morris et al., 2019)

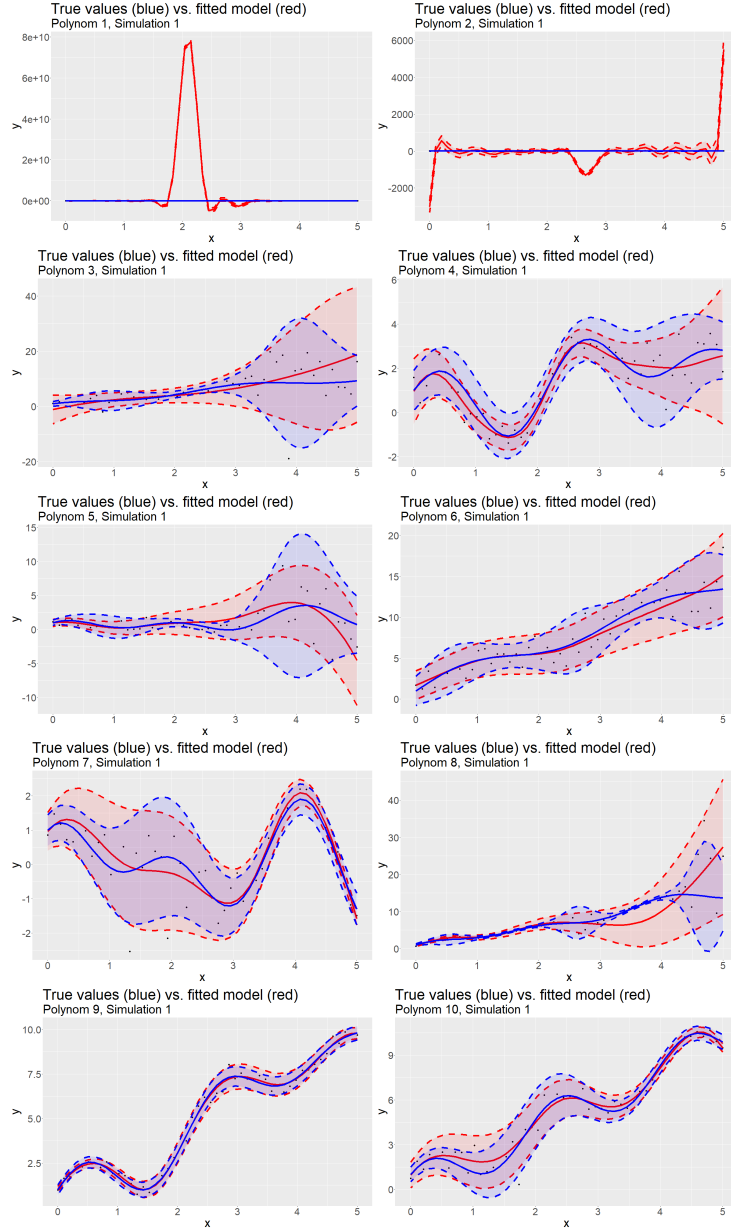
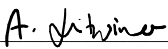


Figure 8: Comparison between true and fitted model for the ten generated polynoms used in the simulation study, exemplary for sample size of  $n = 50$ . The blue area are the 95% tolerance intervals of the true model and the red area are the 95% confidence intervals of the fitted model.

Table 1: Your Data

Entry	$\mu$	$\sigma$
1	999669770.329	27291457.196
2	-7.021	19.154
3	0.187	0.549
4	-16293.340	9686.925
5	-176452.566	623621.810
6	-0.002	-0.082
7	-0.031	0.004
8	564721.876	656043.702
9	-0.027	-0.025
10	3.673	881.299
11	0.022	0.032
12	0.013	0.014
13	-5841.379	26122.680
14	-0.025	-0.037
15	0.132	-0.064
16	-0.032	-0.040
17	-0.038	-0.032
18	-1532126.052	11681292.976
19	97708.693	112873.975
20	-0.002	-0.031
21	113087.026	71004.508
22	0.026	-0.011
23	-0.126	0.122
24	-0.019	-0.032
25	0.030	-0.068
26	-0.022	-0.013
27	0.006	-0.015
28	1545424.137	214415.319
29	8574.713	10195.807
30	0.015	-0.022
31	-0.009	-0.007
32	0.021	-0.027
33	0.104	0.110
34	0.026	-0.035
35	-0.103	0.048
36	0.016 <sup>21</sup>	0.009
37	0.004	-0.022
38	-37106.318	17302.662
39	-129.231	236.215
40	0.000	-0.024
41	-0.012	-0.007
42	-0.016	-0.026
43	0.152	0.110

We hereby affirm that we have written this paper independently without outside help and have used only the sources and aids indicated by us. We have marked passages taken verbatim or in spirit from other works, indicating the sources. We have observed the guidelines for ensuring good scientific practice at the University of Göttingen. We are aware that in case of violation of these principles the examination will be graded as failed.

\_\_\_\_\_

Alexander Litwinov

Bordeaux, September 20, 2023

\_\_\_\_\_

Fabian Lukassen

Kapstadt, September 20, 2023

\_\_\_\_\_

Wendi Qu

Marina, September 20, 2023