Journal of Data Science 19 (3), 498–517
July 2021
DOI: 10.6339/21-JDS1020
Computing in Data Science

Shape-Restricted Regression Splines with R Package *splines2*

Wenjie Wang[1,*] and Jun Yan[2]
[1]*Eli Lilly and Company, Indianapolis, Indiana, USA*
[2]*Department of Statistics, University of Connecticut, Storrs, Connecticut, USA*

Abstract

Splines are important tools for the flexible modeling of curves and surfaces in regression analyses. Functions for constructing spline basis functions are available in R through the base package *splines*. When the curves to be modeled have known characteristics in monotonicity or curvature, more efficient statistical inferences are possible with shape-restricted splines. Such splines, however, are not available in the R package *splines*. The package *splines2* provides easy-to-use shape-restricted spline basis functions, along with their derivatives and integrals which are important tools in many inference scenarios. It also provides additional splines and features that are not available in the *splines* package, such as periodic splines and generalized Bernstein polynomials. The usages of the functions are illustrated with shape-restricted regression, recurrent event data analysis, and extreme-value copulas.

Keywords *Cox–de Boor algorithm; derivatives; integrals; monotone regression; periodic splines*

1 Introduction

Splines are piecewise polynomials that are continuously differentiable up to a certain degree, connected at a sequence of breakpoints called knots. A spline function can be represented by a linear combination of spline basis functions. A variety of spline-based regression methods such as smoothing splines, penalized splines (Eilers and Marx, 1996), and shape-restricted splines (e.g., Meyer, 2008; Wang and Ghosh, 2012) have been widely used for nonparametric regression problems. Non-uniform B-splines are often used as the basis functions in spline-based methods. Besides B-splines, I-splines (Ramsay, 1988) as integrals of M-splines (Curry and Schoenberg, 1966) are used for monotone regressions; C-splines (Meyer, 2008) as integrals of I-splines are used for shape-restricted regressions. In some applications, such as time-to-event analysis, the derivatives or integrals of the basis functions are needed in inferences. A user-friendly and computationally efficient implementation to construct the desired spline basis functions, along with their derivatives or integrals, is necessary for routine uses of spline-based methods.

There are several existing implementations providing various spline basis functions and their derivatives/integrals in R. We focus on those R packages that are available on the Comprehensive R Archive Network (CRAN). The base R package *splines* (R Core Team, 2020) provides functions to construct B-splines, their derivatives, and natural cubic splines. Packages *ibs* (Chen, 2018) and *pbs* (Wang, 2013) provide the integral of B-splines and periodic B-splines, respectively. The *orthogonalsplinebasis* package (Redd, 2015) provides functions for orthogonal B-spline basis functions and their derivatives and integrals. See also Perperoglou et al. (2019) for a recent review of other packages that contain spline-based regression routines.

*Corresponding Author. Email: wang@wwenjie.org.

Received July 20, 2021; Accepted July 20, 2021

The package *splines2* is intended to be a comprehensive, efficient supplement to the package *splines*. Since its first appearance on CRAN in September 2016, *splines2* has been actively developed and maintained over the years. The package contains several attractive features. First, it provides spline basis functions that are not available in *splines*. Such splines include M-splines, I-splines, and C-splines for shape-restricted regression; generalized Bernstein polynomials; and periodic splines. Second, it generalizes and refines the implementation of the B-splines and natural cubic spline basis functions in *splines*. It allows basis functions of degree zero, which seems to be trivial but is particularly useful for certain applications (Shea and Torgovitsky, 2020; Mogstad et al., 2018). The implementation of natural cubic splines utilizes a closed-form null space, which is computationally more efficient than using the QR decomposition as done in *splines*. The basis functions generated this way are nonnegative within the given boundary, which makes it trivial to impose nonnegativity on the resulting spline function for some applications. The third distinct feature of *splines2* is that derivatives and integrals of all the basis functions (except the integrals of C-splines) are provided, which can be necessary for some applications such as time-to-event modeling; the package *splines* only contains derivatives but not integrals of B-splines. Lastly, the under-the-hood C++ implementation based on the packages *Rcpp* (Eddelbuettel, 2013) and *RcppArmadillo* (Eddelbuettel and Sanderson, 2014) are made accessible from C++. See Section 3 for details. The C++ engine is also why *splines2* is computationally more efficient than a few existing packages in handling common tasks such as integration.

The rest of the article is organized as follows. In Section 2, we provide a practical review to those spline basis functions implemented in *splines2* with a focus on M-splines, I-splines, C-splines, and periodic splines. The usage of the main functions in the package is summarized and illustrated in Section 3. In Section 4, we demonstrate the applications of the package to shape-restricted regression, time-to-event data analysis, and extreme-value copulas, respectively. Section 5 concludes with a discussion. An introduction to generalized Bernstein polynomials, B-splines, and natural cubic splines, as well as code for micro-benchmarks, are available in the Supplementary Materials.

2 Spline Basis Functions

2.1 Knot Sequences

It is necessary to define knot sequences before introducing spline basis functions. Suppose m ($m \geqslant 0$) distinct internal knots, $\xi_1, \ldots, \xi_m$, are placed within the boundary knots L and U satisfying $L < \xi_1 < \cdots < \xi_m < U$. Define $t_1 = \cdots = t_d = L$, $t_{d+j} = \xi_j$, $j \in \{1, \ldots, m\}$, and $U = t_{d+m+1} = \cdots = t_{d+p}$, where k and $d = k+1$ represent the polynomial *degree* and the *order* of a basis function, respectively, and $p = d+m$ represent the degrees of freedom of a basis function. Such a nondecreasing sequence $t_1, \ldots, t_{d+p}$ is called the *simple knot sequence* by Ramsay (1988) or the *clamped knot vector* by Piegl and Tiller (1997), where each boundary knot is repeated d times (to allow basis functions to be discontinuous at the boundary) while each interior knot appears only once. Given internal knots, $\xi_1, \ldots, \xi_m$, and boundary knots, L and U, we denote the simple knot sequence intended for splines of degree k by $\boldsymbol{s}_k$. Splines of degree $k \in \{1, 2, \ldots\}$ from the simple knot sequence $\boldsymbol{s}_k$ are continuously differentiable at internal knots up to $k-1$ times, which is the most commonly used definition of splines.

More generally, a knot is called an *r-fold knot* (Prautzsch et al., 2002, Chapter 5) or to have *multiplicity* r if the knot appears r times in the knot sequence, where $r \leqslant k$ is a positive integer. Multiplicities induce discontinuities. For example, B-splines are continuous up to the $(k-r)$th

derivative at the internal knot ξ_j that has multiplicity r. Further, we can relax $t_1 = \cdots = t_d = L$ and $t_{d+m+1} = \cdots = t_{d+p} = U$, respectively, to $t_1 \leqslant \cdots \leqslant t_d = L$ and $U = t_{d+m+1} \leqslant \cdots \leqslant t_{d+p}$. Such a generalized knot sequence, $t_1 \leqslant \cdots \leqslant t_d = L < t_{d+1} \leqslant \cdots \leqslant t_{d+m} < U = t_{d+m+1} \leqslant \cdots \leqslant t_{d+p}$, is referred to as the *extended partition* in Schumaker (2007), where m represents the total multiplicities of internal knots instead of the number of distinct internal knots. The degrees of freedom of the resulting spline basis functions equal the length of the knot sequence minus the order.

2.2 M-Splines, I-Splines, and C-Splines

M-splines are also called *Curry–Schoenberg B-splines* in De Boor (1978). Given a simple knot sequence $\boldsymbol{s}_k$, the ith M-spline basis function (Curry and Schoenberg, 1966) of degree k denoted by $M_{i,k}(x \mid \boldsymbol{s}_k)$ can be considered as the normalized B-spline basis function satisfying

$$M_{i,k}(x \mid \boldsymbol{s}_k) = (k+1)B_{i,k}(x \mid \boldsymbol{s}_k)/(t_{i+k+1} - t_i), \tag{1}$$

so that $\int_{t_1}^{t_{d+p}} M_{i,k}(x \mid \boldsymbol{s}_k) = 1$, where $B_{i,k}(x \mid \boldsymbol{s}_k)$ is the corresponding B-spline basis function, $i \in \{1, \ldots, p\}$. See the Supplementary Materials for a brief introduction to B-splines.

Similar to the Cox–de Boor recursive formula for B-splines, the M-spline basis function $M_{i,k}(x \mid \boldsymbol{s}_k)$ can be defined recursively by

$$M_{i,k}(x \mid \boldsymbol{s}_k) = \frac{k+1}{k(t_{i+k+1} - t_i)}\left[(x - t_i)M_{i,k-1}(x \mid \boldsymbol{s}_k) + (t_{i+k+1} - x)M_{i+1,k-1}(x \mid \boldsymbol{s}_k)\right].$$

Furthermore, the first derivative of $M_{i,k}(x \mid \boldsymbol{s}_k)$ is as follows:

$$\frac{\mathrm{d}}{\mathrm{d}x}M_{i,k}(x \mid \boldsymbol{s}_k) = \frac{k+1}{t_{i+k+1} - t_i}\left(M_{i,k-1}(x \mid \boldsymbol{s}_k) - M_{i+1,k-1}(x \mid \boldsymbol{s}_k)\right). \tag{2}$$

The M-splines integrated from t_1 to x were referred to as *I-splines* in Ramsay (1988). To be more specific, the ith I-spline basis function of degree k denoted by $I_{i,k}(x \mid \boldsymbol{s}_k)$ for $i \in \{1, \ldots, p\}$ is defined to be $\int_{t_1}^{x} M_{i,k}(t \mid \boldsymbol{s}_k)\,\mathrm{d}t$. We have

$$I_{i,k}(x \mid \boldsymbol{s}_k) = \sum_{l=i+1}^{p+1} B_{l,k+1}(x \mid \boldsymbol{s}_{k+1}) = \sum_{l=i+1}^{p+1}\left(\frac{t_{l+k+2} - t_l}{k+2}\right)M_{l,k+1}(x \mid \boldsymbol{s}_{k+1}), \tag{3}$$

where t_{l+k+2} and t_{l+1} are defined for the knot sequence $\boldsymbol{s}_{k+1}$.

M-splines are nonnegative for $L \leqslant x \leqslant U$. Hence, I-splines are monotonically nondecreasing from L to U by definition. Ramsay (1988) proposed using I-splines for monotone regression. A monotonically nondecreasing (nonincreasing) function can be fitted by a linear combination of I-splines and an additional intercept term, where the monotonicity is ensured by constraining the coefficients of I-splines to be nonnegative (nonpositive).

For regressions with curvature restrictions, Meyer (2008) introduced a class of convex splines known as C-splines. By integrating both sides of (3) for I-splines of degree $k+1$, we obtain the formula for the integral of the ith I-spline basis function denoted by $\tilde{C}_{i,k}(x \mid \boldsymbol{s}_k)$ as follows:

$$\tilde{C}_{i,k}(x \mid \boldsymbol{s}_k) = \sum_{l=i+1}^{p+1}\int_{t_1}^{x} B_{l,k+1}(t \mid \boldsymbol{s}_{k+1})dt = \sum_{l=i+1}^{p+1}\left(\frac{t_{l+k+2} - t_l}{k+2}\right)I_{l,k+1}(x \mid \boldsymbol{s}_{k+1}).$$

We define the ith C-spline basis function denoted by $C_{i,k}(x \mid \boldsymbol{s}_k)$ to be the scaled $\tilde{C}_{i,k}(x \mid \boldsymbol{s}_k)$,

$$C_{i,k}(x \mid \boldsymbol{s}_k) = \tilde{C}_{i,k}(x \mid \boldsymbol{s}_k)/\tilde{C}_{i,k}(U \mid \boldsymbol{s}_k), \tag{4}$$

so that $C_{i,k}(U \mid \boldsymbol{s}_k) = 1$ for $i \in \{1, \ldots, p\}$. A convex or concave function can be approximated by a linear combination of C-splines, an identity function $f(x) = x$, and an intercept term, where the coefficients of C-splines are constrained to be nonnegative or nonpositive to guarantee convexity or concavity, respectively.

A set of M-splines and the corresponding I-splines and C-splines are visualized in the first row of Figure 1.

2.3 Periodic Splines

We define a spline function $s(x)$ of degree k to be *periodic* with a cyclic interval from L to U if it satisfies that $s(x) = s(x + U - L)$ and $\mathrm{d}^\kappa s(x)\big/\mathrm{d}x^\kappa = \mathrm{d}^\kappa s(x + U - L)\big/\mathrm{d}x^\kappa$, $x \in \mathbb{R}$, where $\kappa \in \{1, \ldots, k-1\}$. In applications to computer-aided design (CAD), periodic basis functions are used to form closed curves or surfaces (e.g., Farin and Hansford, 2000).

Given that $m \geqslant k-1$, Piegl and Tiller (1997, Chapter 12) gave two sufficient conditions for a B-spline function of degree k to be periodic with a cyclic interval from L to U as follows:

1. The underlying B-spline basis functions are based on an extended knot sequence $\boldsymbol{s}_k$ satisfying $t_{k-i} = t_{k-i+1} - (t_{m+d+1-i} - t_{m+d-i})$ and $t_{m+d+2+i} = t_{m+d+1+i} - (t_{k+2+i} - t_{k+1+i})$ for $i \in \{0, \ldots, k-1\}$, where $t_d = L$ and $t_{d+m+1} = U$;
2. The first k coefficients and the last k coefficients must coincide.

The first condition essentially provides a simple procedure to construct an extended knot vector from the given m internal knots $t_{d+1}, \ldots, t_{d+m}$, and a cyclic interval from L to U. For the B-splines $\{B_1, \ldots, B_{m+d}\}$ resulting from the extended knot sequence satisfying the first condition, we incorporate the second condition by summing up the first k and the last k basis functions, respectively, to a new set of k basis functions $\{B_1 + B_{m+2}, \ldots, B_k + B_{m+d}\}$, which along with the remaining basis functions $\{B_{k+1}, \ldots, B_{m+1}\}$ (if $m \geqslant k$) form a set of periodic B-spline basis functions. A periodic spline function can then be simply represented by a linear combination of these basis functions.

Such a procedure can also be applied directly to construct periodic M-spline basis functions. The constructed periodic M-splines inherit the normalization property of regular M-splines and have unit integrals from L to U. It is thus easier to compute the integrals of the basis functions from L to any $x \geqslant U$. For instance, a set of periodic M-splines and the corresponding first derivatives and integrals are visualized in the second row of Figure 1.

3 Usage

Package *splines2* provides implementation of those spline basis functions reviewed in Section 2 and the Supplementary Materials. The function interfaces are designed to be similar and consistent to the function `bs()` of *splines* so that no unnecessary learning curve is required for those users familiar with this package. The common arguments are `x`, `df`, `knots`, `degree`, `intercept`, and `Boundary.knots`, all of which have the same meaning and equivalent default values as `bs()` except that the default value of the argument `intercept` is `TRUE` for `iSpline()` and `cSpline()`. The `x` represents the predictor variable and allows `NA`'s. Either `df` or `knots` can be specified to adjust the degrees of freedom of the basis functions given their polynomial degree specified by

Table 1: Summary of the function interfaces compared to the arguments of `splines::bs()` from *splines*.

Function	Additional Arguments	Inapplicable Arguments
`bSpline()`	`derivs`, `integral`, `...`	
`ibs()`	`...`	
`dbs()`	`derivs`, `...`	
`bernsteinPoly()`	`derivs`, `integral`, `...`	`knots`
`naturalSpline()`	`derivs`, `integral`, `...`	`degree`
`mSpline()`	`derivs`, `integral`, `periodic`, `...`	
`iSpline()`	`derivs`, `...`	
`cSpline()`	`derivs`, `scale`, `...`	

`degree`. The function `bSpline()` from *splines2* extends the `bs()` function by allowing `degree = 0` for piecewise constant basis functions. If `intercept = TRUE`, the complete set of basis functions will be returned. Otherwise, the first basis function will be excluded from the returned matrix.

Table 1 gives a summary of the argument list of the main functions compared to the `bs()` function. The argument `derivs` takes a nonnegative integer representing the order of derivatives of the spline basis functions, while the argument `integral` takes a boolean value and the integrals of the basis functions will be returned if `integral = TRUE` is specified. For C-splines, the argument `scale` takes a boolean value indicating if the scaling of the integral of the I-spline basis functions in (4) is needed.

The package also provides several S3 methods for the basis matrix returned by any of the main functions. More specifically, we implemented a series of `predict()` methods to evaluate the basis functions at any (new) `x` specified by its second argument `newx` following the method `predict.bs()` of *splines*. In addition, the package provides `deriv()` methods for the derivatives of each basis function. The order of the derivatives can be specified by its second argument `derivs`. The `knots()` methods retrieve the placed internal or boundary knots, which can be useful when the knots are determined automatically based on the input `x`.

We created various example spline basis functions in the following code chunk by using the package *splines2*. We fixed the degree of the basis functions to be 3 and placed the boundary knots at 0 and 2. For all the spline basis functions, we placed the internal knots at 0.3, 0.7, 1.2, and 1.5. The resulting basis functions are visualized by distinct colors and line types in Figure 1, where the placement of the internal knots is indicated by gray dashed vertical lines.

```
library(splines2)
x <- seq.int(0, 2, 0.01)           # set x
int_knots <- c(0.3, 0.7, 1.2, 1.5) # internal knots
ms_mat <- mSpline(x, knots = int_knots, intercept = TRUE) # M-splines
is_mat <- iSpline(x, knots = int_knots, intercept = TRUE) # I-splines
cs_mat <- cSpline(x, knots = int_knots, intercept = TRUE) # C-splines
## periodic splines, their derivatives, and integrals
pm_mat  <- mSpline(x, knots = int_knots, intercept = TRUE, periodic = TRUE)
dpm_mat <- mSpline(x, knots = int_knots, intercept = TRUE, periodic = TRUE,
                   derivs = 1) # or equivalently `deriv(pm_mat)`
```

```
ipm_mat <- mSpline(x, knots = int_knots, intercept = TRUE, periodic = TRUE,
                   integral = TRUE)
bs_mat  <- bSpline(x, knots = int_knots, intercept = TRUE) # B-splines
dbs_mat <- deriv(bs_mat)                                   # B-spline Derivatives
ibs_mat <- ibs(x, knots = int_knots, intercept = TRUE) # B-spline Integrals
## natural cubic splines, their derivatives, and integrals
ns_mat  <- naturalSpline(x, knots = int_knots, intercept = TRUE)
dns_mat <- deriv(ns_mat)
ins_mat <- naturalSpline(x, knots = int_knots, intercept = TRUE,
                         integral = TRUE)
```

To be more specific, we first illustrated M-splines, I-splines, and C-splines, respectively, with the functions `mSpline()`, `iSpline()`, and `cSpline()`; see the first row of Figure 1 for the basis functions. Second, we created the periodic M-splines introduced in Section 2.3 and their derivatives and integrals, respectively, by using the function `mSpline()`; see the second row of Figure 1 for the corresponding basis functions. Third, we obtained B-splines, natural cubic splines, and their first derivatives and integrals, respectively. We refer readers to the Supplementary Materials for the introduction to B-splines and natural cubic splines implemented in the package. The resulting basis functions are visualized in the last two rows of Figure 1.

In addition to the interface in R, the package *splines2* includes a C++ header-only library, which allows the construction of spline basis matrices directly in C++ with the help of the packages *Rcpp* and *RcppArmadillo*. The *Rcpp* interface is intended for package developers who would like to use the package at C++ level. An introduction is available in one of the package vignettes (accessible by `vignette("splines2-wi-rcpp")`).

4 Applications

4.1 Shape-Restricted Regression

Shape constraints in terms of monotonicity or curvature arise naturally in many applications. Examples are growth curves in population health, dose-response models in phase I clinical trials, and utility functions in economics. Consider a general nonparametric regression model $E(Y \mid X) = f(X)$, where X and Y are $\mathbb{R}$-valued predictor and response variables, respectively. The observed data are independent pairs of (x_i, y_i), $i \in \{1, \ldots, n\}$. The true function $f(\cdot)$ is assumed to be continuous with certain known shape constraints. We illustrate how to apply I-splines and C-splines to shape-restricted regression following Ramsay (1988) and Meyer (2008), respectively, to estimate $f(\cdot)$.

A non-decreasing curve can be fitted by a linear combination of I-splines with an additional intercept where the coefficients of the I-spline basis functions are constrained to be nonnegative (Ramsay, 1988). The coefficients can be estimated by constrained least squares,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left[y_i - (\alpha_0 + \boldsymbol{\beta}^\top \boldsymbol{I}(x_i)) \right]^2, \text{ s.t. } \boldsymbol{\beta} \geqslant \mathbf{0}, \tag{5}$$

where the y_i's are the realizations of the Y_i's, $\boldsymbol{\theta} = (\alpha_0, \boldsymbol{\beta}^\top)^\top$, α_0 is the intercept, and $\boldsymbol{I}(x_i)$ is a vector that consists of I-splines evaluated at x_i. The minimization problem can be solved by

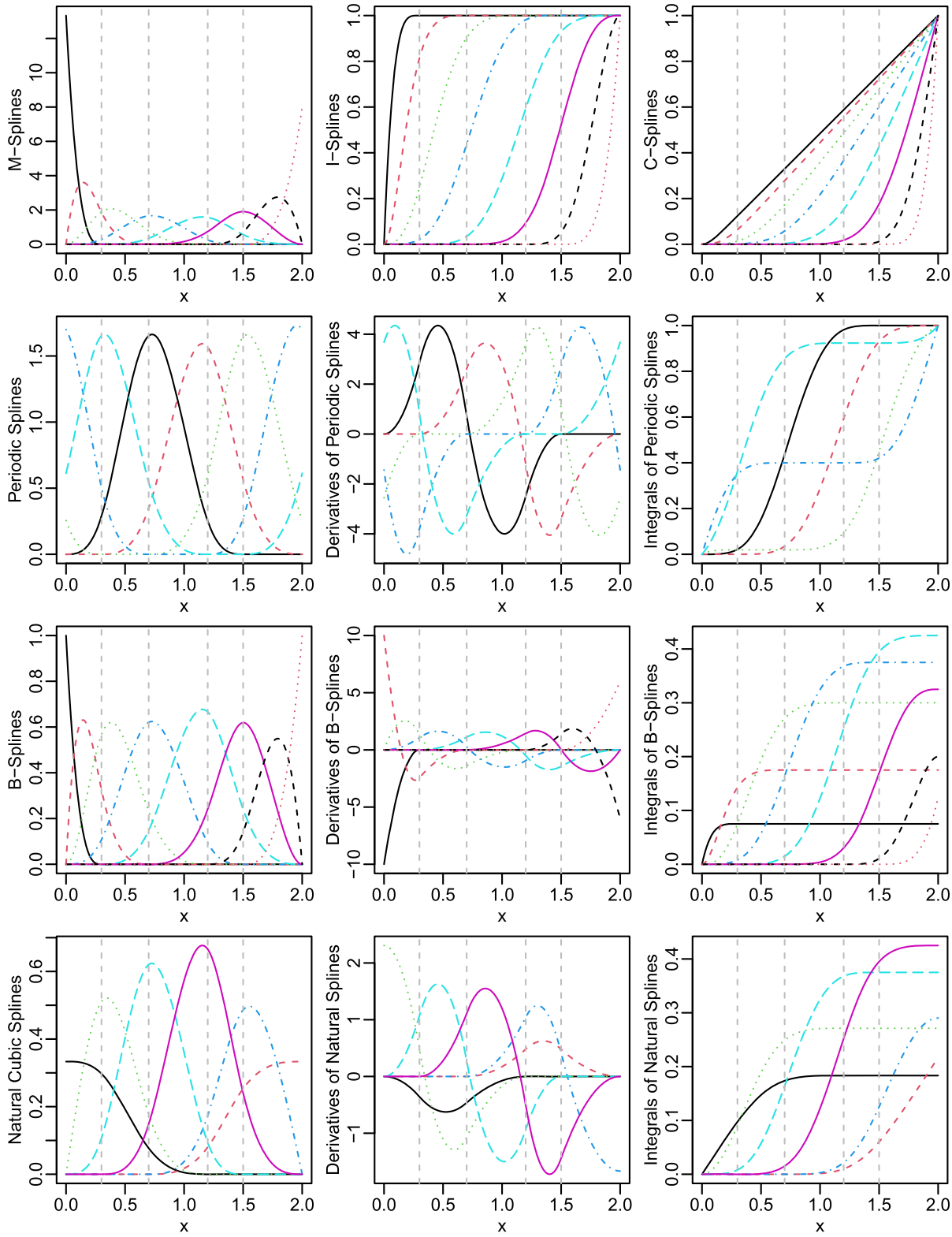

Figure 1: Various example spline basis functions, their derivatives and integrals that are available in the R package *splines2*.

nonnegative least squares methods, or more generally, quadratic programming methods. The latter solve the minimization problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\boldsymbol{d}^\top \boldsymbol{\theta} + \frac{1}{2}\boldsymbol{\theta}^\top \boldsymbol{D}\boldsymbol{\theta}, \text{ s.t. } \boldsymbol{A}^\top \boldsymbol{\theta} \geqslant \boldsymbol{b}_0, \tag{6}$$

where $\boldsymbol{d}$ and $\boldsymbol{b}_0$ are given column vectors, and $\boldsymbol{D}$ and $\boldsymbol{A}$ are known matrices. Define $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$, $\boldsymbol{X} = \begin{bmatrix}\mathbf{1} & \boldsymbol{I}(\boldsymbol{X})\end{bmatrix}$, and $\boldsymbol{I}(\boldsymbol{X}) = \begin{bmatrix}\boldsymbol{I}(x_1) & \cdots & \boldsymbol{I}(x_n)\end{bmatrix}^\top$, where $\mathbf{1}$, $\boldsymbol{I}(\boldsymbol{X})$, and $\{\boldsymbol{I}(x_i)\}$ are treated as submatrices. Problem (5) is solved with $\boldsymbol{d} = \boldsymbol{y}^\top \boldsymbol{X}$, $\boldsymbol{b}_0 = \mathbf{0}$, $\boldsymbol{D} = \boldsymbol{X}^\top \boldsymbol{X}$, and $\boldsymbol{A}^\top = \begin{bmatrix}\mathbf{0} & \mathcal{I}_p\end{bmatrix}$, where $\mathcal{I}_p$ is an identity matrix of size $p \times p$, and p is the degrees of freedom of the I-spline basis functions.

We applied `iSpline()` and implemented the estimation procedure with the help of the *quadprog* package (Turlach and Weingessel, 2019) as follows:

```
library(splines2)
library(quadprog)
## Standardize x and y
standardize <- function(x, y) {
    sx <- scale(x); sy <- scale(y)
    x_bar <- attr(sx, "scaled:center"); x_sd <- attr(sx, "scaled:scale")
    y_bar <- attr(sy, "scaled:center"); y_sd <- attr(sy, "scaled:scale")
    list(sx = sx, x_bar = x_bar, x_sd = x_sd,
         sy = sy, y_bar = y_bar, y_sd = y_sd)
}
## Solve by quadratic programming and scale coefficients back
qp_solve <- function(standardized, A_mat, x_mat, ...) {
    d_vec <- with(standardized, crossprod(sx, sy)) # (y^T X)^T
    D_mat <- crossprod(standardized$sx)            # X^T X
    sbeta <- quadprog::solve.QP(D_mat, d_vec, A_mat, ...)$solution
    ## scale the coefficients back
    beta0 <- with(standardized,
                  -as.numeric(crossprod(x_bar * y_sd / x_sd, sbeta)) + y_bar)
    beta1 <- with(standardized, y_sd / x_sd * sbeta)
    list(x_mat = x_mat,
         coefficients = c("(Intercept)" = beta0, beta1),
         fitted = as.numeric(x_mat %*% beta1) + beta0)
}
## Monotone regression by I-splines
mono_fit <- function(x, y, monotone = c("increasing", "decreasing"), ...) {
    monotone <- match.arg(monotone)
    x_mat <- iSpline(x, ..., intercept = TRUE)     # create I-splines
    standardized <- standardize(x_mat, y)          # standardization
    A_mat <- diag(ncol(standardized$sx))           # set matrix A
    if (monotone == "decreasing") A_mat <- -A_mat  # flip the sign of matrix A
    qp_solve(standardized, A_mat, x_mat)
}
```

The standardizations of $\boldsymbol{y}$ and each column of $\boldsymbol{X}$ by the function `standardize()` increase the numerical stability and simplify the procedure by eliminating the need to include an additional

intercept term. Notice that the linear constraints on the coefficients of the unstandardized basis functions can be equivalently applied to the coefficients of the standardized basis functions without modifying $\boldsymbol{A}$. The function `qp_solve()` returns the estimates of $\boldsymbol{\theta}$ by solving the quadratic programming problem. It is straightforward to fit a nonincreasing curve with I-splines by flipping the sign of $\boldsymbol{A}$ in the function `mono_fit()`.

We applied `mono_fit()` to the age-income data used in Ruppert et al. (2003) and Meyer (2008). The dataset consists of 205 Canadian workers and is available in the *SemiPar* package (Wand, 2018). A nondecreasing relationship between the age of workers and the expected logarithm of the income was considered. In the following code chunk, we fitted three nondecreasing curves by I-splines and specified the degrees of freedom of the basis functions to be 6, 10, and 14, respectively.

```
data(age.income, package = "SemiPar")
dfs <- c(6, 10, 14)
mfits <- lapply(dfs, function(d) {
    with(age.income, mono_fit(age, log.income, df = d))
})
```

A convex or concave curve can be fitted by a linear combination of C-spline basis functions, an identity function, and an intercept term, where the coefficients of the C-splines are constrained to be nonnegative or nonpositive, respectively (Meyer, 2008). For convex regression, we estimate the coefficients by constrained least squares to minimize the squared error loss,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left[y_i - (\alpha_0 + \alpha_1 x_i + \boldsymbol{\beta}^\top \boldsymbol{C}(x_i)) \right]^2, \text{ s.t. } \boldsymbol{\beta} \geqslant \mathbf{0}, \tag{7}$$

where the y_i's are the realizations of the Y_i's, $\boldsymbol{\theta} = (\alpha_0, \alpha_1, \boldsymbol{\beta}^\top)^\top$, α_0 is the intercept, α_1 is the coefficient of the linear term, and $\boldsymbol{C}(x_i)$ is a vector that consists of C-splines evaluated at x_i. Define $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ and $\boldsymbol{X} = \begin{bmatrix} \mathbf{1} & \boldsymbol{x} & \boldsymbol{C}(\boldsymbol{X}) \end{bmatrix}$, where $\boldsymbol{C}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{C}(x_1) & \cdots & \boldsymbol{C}(x_n) \end{bmatrix}^\top$. Problem (7), again, is a quadratic programming problem with $\boldsymbol{d} = \boldsymbol{y}^\top \boldsymbol{X}$, $\boldsymbol{b_0} = \mathbf{0}$, $\boldsymbol{D} = \boldsymbol{X}^\top \boldsymbol{X}$, and $\boldsymbol{A}^\top = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathcal{I}_p \end{bmatrix}$, where $\mathcal{I}_p$ is an identity matrix of size $p \times p$ and p is the degrees of freedom of the C-spline basis functions. The sign of $\boldsymbol{A}$ is flipped for concave regression.

A combination of the curvature constraint and the monotonicity constraint can be achieved by introducing an additional constraint in terms of the first derivative at one of the boundary knots. For instance, to fit a nondecreasing convex (or nonincreasing concave) curve, it suffices to additionally impose the constraint that the first derivative of the curve is nonnegative (or nonpositive) at the left boundary. Similarly, to fit a nondecreasing concave (or nonincreasing convex) curve, we can additionally impose the constraint that the first derivative of the curve is nonnegative (or nonpositive) at the right boundary. Given the first derivatives of the C-splines, it is straightforward to add the derivative constraint at the boundary to the original quadratic programming problem for the curvature constraints only. We applied `cSpline()` and implemented the estimation procedure named `curv_fit()` in the following code chunk:

```
## curvature-restricted regression
curv_fit <- function(x, y, monotone = c("none", "increasing", "decreasing"),
                     curvature = c("convex", "concave"), ...) {
    monotone <- match.arg(monotone); curvature <- match.arg(curvature)
```

```
    min_x <- min(x); range_x <- max(x) - min_x
    x_mat <- cSpline(x, ..., intercept = TRUE) # create C-splines
    if (monotone != "none") {
        right_side <- (monotone == "decreasing" && curvature == "convex") ||
            (monotone == "increasing" && curvature == "concave")
        side_knot <- knots(x_mat, "boundary")[right_side + 1]
        dx_mat <- cbind(1 / range_x, deriv(predict(x_mat, side_knot)))
    }
    x_mat <- cbind("(Linear)" = (x - min_x) / range_x, x_mat)
    standardized <- standardize(x_mat, y)
    A_mat <- rbind(0, diag(ncol(x_mat) - 1L))
    if (curvature == "concave") A_mat <- - A_mat
    if (monotone == "decreasing") dx_mat <- - dx_mat
    if (monotone != "none") {
        for (j in seq_len(ncol(dx_mat)))
            dx_mat[, j] <- dx_mat[, j] / standardized$x_sd[j]
        A_mat <- cbind(A_mat, as.numeric(dx_mat))
    }
    qp_solve(standardized, A_mat, x_mat)
}
```

Similarly as before, we standardized each C-spline basis function and the identity term by `standardize()`. The linear constraints for the desired curvature (and monotonicity) can be applied to the standardized basis functions and the standardized linear term without any changes. The coefficient estimates are obtained by solving the quadratic programming problem and scaling in the helper function `qp_solve()`. Now suppose there is a concave relationship between age and the expected logarithm of the income. We can apply the function `curv_fit()` to the age-income data as follows:

```
cfits <- lapply(dfs, function(d) {
    with(age.income, curv_fit(age, log.income, "none", "concave", df = d))
})
```

The fitted nondecreasing and concave curves are visualized, respectively, in the left and right panel of Figure 2, from which we may find that despite having different degrees of freedom, the fitted curves stay closely to each other in each panel. It implies that with shape constraints, the fitted curves are robust with respect to the choice of the degrees of freedom and the placement of the internal knots. We also fitted a natural cubic smoothing spline function (via `stats::smooth.spline()`) to the age-income data for comparison. The tuning parameter of the smoothing spline was selected by generalized cross-validation. The fitted smoothing spline is visualized in both panels of Figure 2, which is neither nondecreasing nor convex without shape constraints.

To demonstrate simultaneous monotone and curvature constraints, as a second example, we applied `mono_fit()` and `curv_fit()` to the *onions* dataset that is also available in the *SemiPar* package. The degrees of freedom of the I-splines and C-splines considered were similarly 6, 10, and 14.

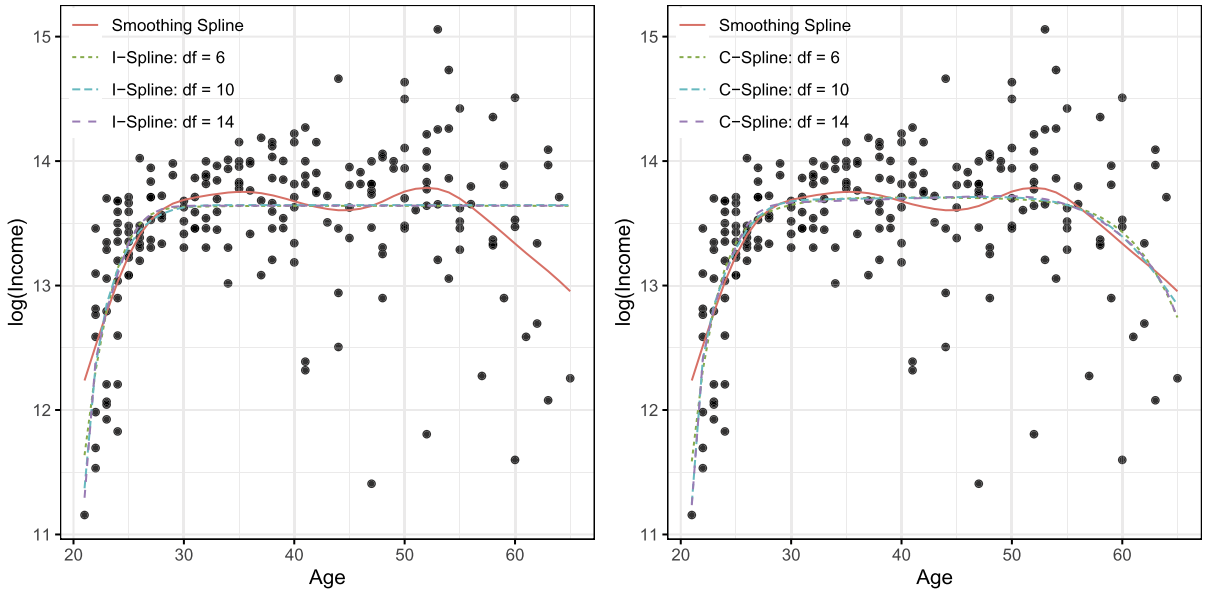

Figure 2: The fitted nondecreasing functions by I-splines (in the left panel) and the fitted concave functions by C-splines (in the right panel) together with the fitted natural cubic smoothing spline for the age-income data.

```
data(onions, package = "SemiPar")
mfits <- lapply(dfs, function(d) {
    with(onions, mono_fit(dens, log(yield), "decreasing", df = d))
})
cfits <- lapply(dfs, function(d) {
    with(onions, curv_fit(dens, log(yield), "decreasing", "convex", df = d))
})
```

It is assumed that the logarithm of the onion yield (grams per plant) decreases as the areal density of plants (plants per square meter) increases. We additionally assumed a convex relationship between log(yield) and density, which means that the decreasing rate of log(yield) did not increase as the density increased. The fitted curves with monotonicity constraint are only visualized in the left panel of Figure 3 and the fitted curves with additional convex constraints are visualized in the right panel of Figure 3. With additional curvature constraints, the convex curves fitted by C-splines of different degrees of freedom stay more closely to each other than the nonincreasing curves fitted by I-splines. Similarly, we fitted a natural cubic smoothing spline and visualized it in both panels of Figure 3 for comparison, from which we found that the fitted convex curves were very close to the fitted smoothing spline function overall.

4.2 Time-to-Event Data Analysis

Splines are natural choices to model the baseline hazard function $h_0(t)$, the cumulative hazard function $H_0(t)$, or their logarithms, respectively, for time-to-event data. From $\int_0^t h_0(s)\,\mathrm{d}s = H_0(t)$, either the derivatives or the integrals of the spline basis functions are usually needed to construct the likelihood function. Similar to the shape-restricted regression, constraints can be imposed to

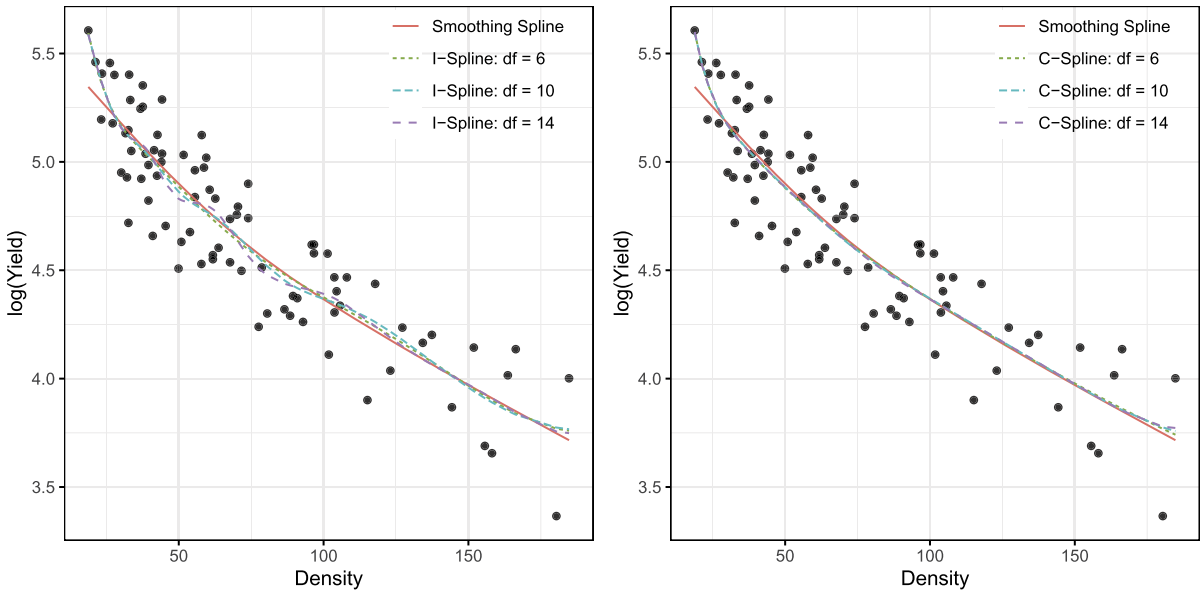

Figure 3: The fitted nonincreasing functions by I-splines (in the left panel) and the fitted convex functions by C-splines (in the right panel) with the natural cubic fitted smoothing spline for the onions data.

ensure that the estimated baseline cumulative hazard function is nondecreasing over time and the corresponding baseline hazard estimates are nonnegative. A simple example was given by Rosenberg (1995), where the baseline hazard function is modeled by B-splines as

$$h_0(t) = \sum_{i=1}^{p} \beta_i B_{i,k}(t), \quad H_0(t) = \int_0^t h_0(s)ds = \sum_{i=1}^{p} \beta_i \int_0^t B_{i,k}(s)ds,$$

where $B_{i,k}(t)$ represents the ith B-spline basis of degree k, and p denotes the degrees of freedom of the basis functions.

Spline-based methods have been proposed for modeling recurrent event data as well. For instance, Fu et al. (2016) proposed a gamma frailty model with a piecewise constant rate function as follows:

$$\rho(t \mid x_i, r_i) = r_i \rho_0(t) \exp\{\boldsymbol{x}_i^\top \boldsymbol{\beta}\},$$

where the r_i's are i.i.d. frailty effects from a gamma distribution with unit mean, $\rho_0(\cdot)$ is the baseline rate function estimated by piecewise constant basis functions, the $\boldsymbol{x}_i$'s represent covariates, and $\boldsymbol{\beta}$ is the covariate coefficient vector of interest. Wang et al. (2020) extended the proposed model by modeling the baseline rate function by M-splines in the R package *reda*. The model estimation is based on the maximization of the marginal likelihood function, which involves integrating the rate function. For some applications such as seasonal illness or warranty claims of snowmobiles, the rate function can be periodic, in which case, the periodic splines introduced in Section 2.3 become of interest.

For illustration, we generated cyclic recurrent events from 500 processes with a periodic baseline rate function as follows:

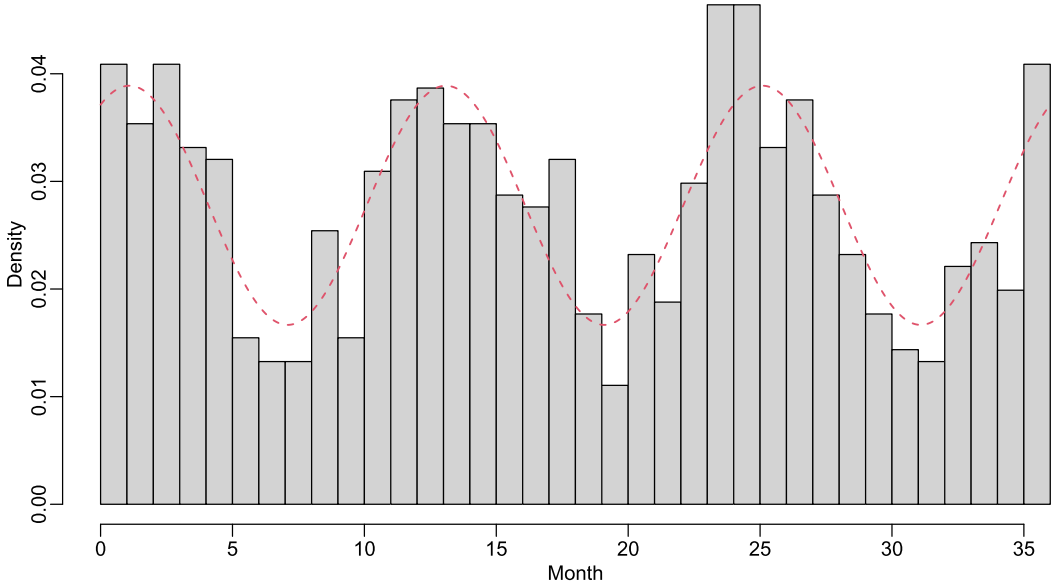

Figure 4: Histogram of the simulated cyclic recurrent event times. The periodic baseline rate function is indicated by the red dashed line after scaling for visual comparison.

```
## simulate recurrent event data from a periodic rate function
library(reda)
set.seed(123)         # set random number seed
n <- 500              # number of event process
beta <- 0.2           # covariate coefficient
tau <- 36             # end of follow-up
right <- 12           # end of cyclic interval
foo <- function(x)    # the example periodic function
    sin(x * pi / 6 + 1) / 50 + 0.05
dat <- simEventData(n, z = matrix(rnorm(n)), zCoef = beta,
                    endTime = tau, rho = foo, frailty = rgamma,
                    arguments = list(frailty = list(shape = 2, scale = 0.5)))
```

The generation utilized the function `simEventData()` of the *reda* package. For simplicity, we considered one covariate generated from the standard normal distribution and a frailty term from the gamma distribution with shape parameter 2 and scale parameter 0.5. To resemble the seasonality trend of some practical examples, we set the time unit to be one month and let the cyclic period last 12 months. We considered the cyclic function $f(x) = \sin(\pi x/6 + 1)/50 + 0.05$. The end of follow-up was fixed to be 36 months to include 3 cyclic intervals. The histogram of the simulated event times is given in Figure 4, from which we may find a clear periodic pattern of the event density. To visually compare the histogram and underlying baseline rate function, we scaled the latter so that its integral attained one at the end of the follow-up. The scaled baseline rate function represented by the red dashed line looks similar to the histogram of the event times in Figure 4 regarding the seasonality trend, which suggests that the `reda::simEventData()` works as expected.

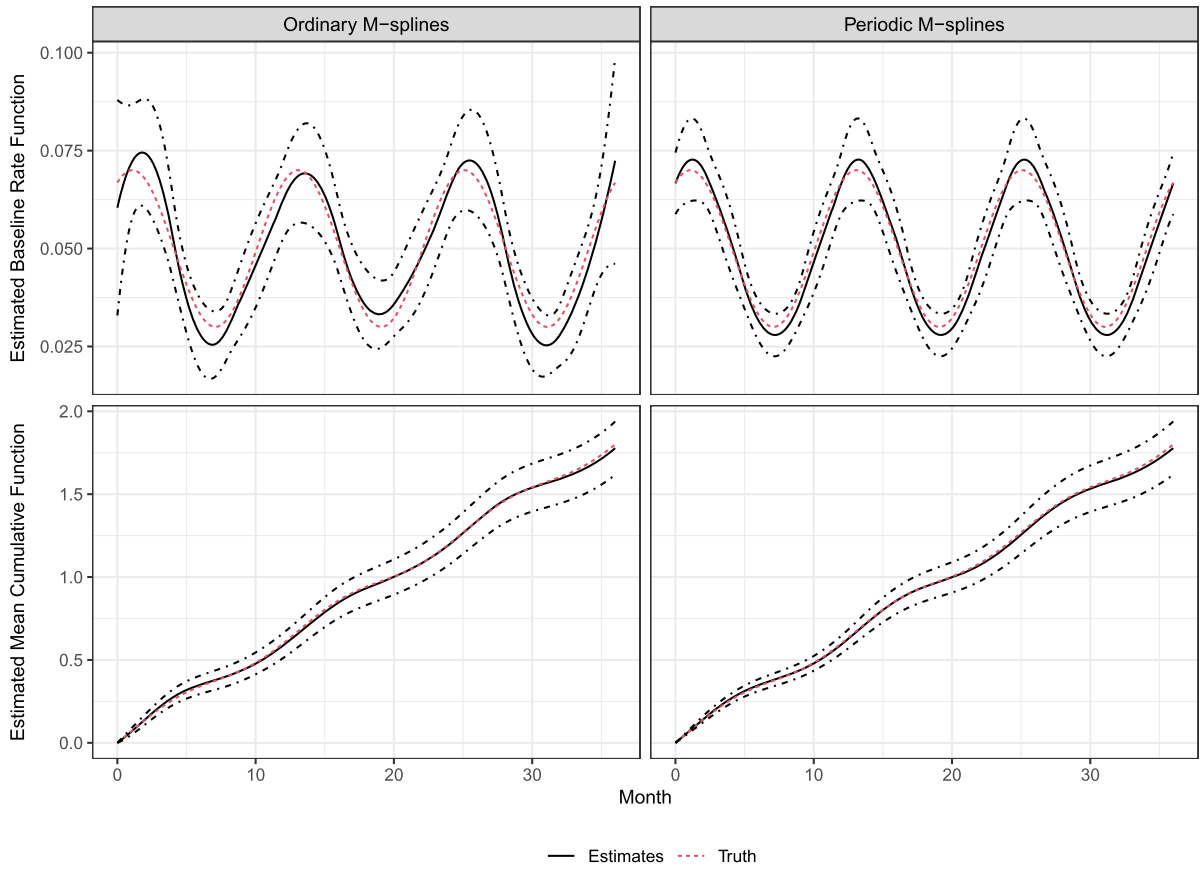

Figure 5: Estimated baseline rate functions and mean cumulative functions by using ordinary M-splines and periodic splines, respectively. The black solid lines are the estimates; the dot-dashed lines are the pointwise 95% confidence intervals; the red dashed lines are the true curves.

The `splines2::mSpline()` returns the integrals of the periodic basis functions when both `periodic` and `integral` are specified to be `TRUE`, which makes it straightforward to obtain the maximum likelihood estimates as if the piecewise constant basis functions were used to model the baseline rate function. We assumed that the length of each cyclic interval was known to be 12 months. For the simulated recurrent event data, we applied the ordinary M-splines and periodic M-splines to fit the baseline rate function, respectively, and obtained the model estimation results with the help of the function `rateReg()` of the *reda* package, which builds on the periodic splines from the *splines2* package. We considered quadratic basis functions. For ordinary M-splines, we placed eight internal knots at 4, 8, ..., 32, and placed boundary knots at 0 and 36 (the end of follow-up). For periodic M-splines, the boundary knots define the cyclic interval and thus were placed at 0 and 12. Given the boundary knots, we placed three internal knots at the 25%, 50%, and 75% quantiles of the observed event times relative to the beginning of the cyclic interval that they belong to, which turned out to be approximately 2.2, 4.8, and 9.2 for the simulated dataset.

The estimated baseline rate functions and their integrals, the estimated baseline mean cumulative functions (MCF) are visualized, respectively, in the upper panel and the lower panel of Figure 5, where the point estimates are represented by the black solid lines, the pointwise

95% confidence interval estimates are indicated by the dash-dotted lines, while the underlying true baseline rate functions and mean cumulative function are indicated by the red dashed lines, respectively.

As we expected, the estimated baseline rate function (by using periodic M-splines) followed the seasonality trend of the true baseline rate function closer over time than using ordinary M-splines. Despite the bias around the nadir and the peak of the true baseline rate function, the pointwise confidence interval (CI) estimates of both approaches contained the true baseline rate function at each time point. However, the width of the CI estimates by using periodic M-splines was smaller than when using ordinary M-splines, especially around the boundary. The comparison of the estimated models was similar in terms of the baseline MCF. The estimated mean cumulative functions were both close to the true curve over time. However, using periodic M-splines resulted in smaller bias and narrower pointwise CI estimates.

4.3 Extreme-Value Copulas

Consider two random variables X and Y with continuous cumulative distribution functions F and G, respectively. Let $H(x, y) = P(X \leqslant x, Y \leqslant y)$ denote their joint distribution function. By Sklar's Theorem, there exists a two-dimensional copula C such that $H(x, y) = C(F(x), G(y))$ and $C(u, v) = P(F(X) \leqslant u, G(Y) \leqslant v)$. Following Pickands (1981), a copula C is a bivariate extreme-value copula if and only if there exists a convex function $A : [0, 1] \mapsto [1/2, 1]$ such that for any $u, v \in (0, 1)$,

$$C(u, v) = \exp\left\{\log(uv)A\left(\frac{\log(v)}{\log(uv)}\right)\right\}.$$

The convex function $A(t)$ is called the Pickands dependence function associated with C and satisfies

$$\max(t, 1-t) \leqslant A(t) \leqslant 1, t \in [0, 1], \tag{8}$$

and $A(0) = A(1) = 1$. Each $A(t)$ uniquely defines an extreme-value copula.

Estimation of $A(t)$ can be done by with a nonparametric regression. For a random bivariate sample $\{(x_i, y_i)\}$ from (X, Y), let F_n and G_n denote the empirical cumulative distribution functions of X and Y, respectively. Define $u_i = nF_n(x_i)/(n+1)$ and $v_i = nG_n(y_i)/(n+1)$, where the asymptotically negligible scaling factor $n/(n+1)$ was introduced to avoid problems with density evaluation at the boundaries of $[0, 1]^2$. Let C_n denote the empirical copula and define

$$t_i = \frac{\log(v_i)}{\log(u_i v_i)}, \quad z_i = \frac{\log\{C_n(u_i, v_i)\})}{\log(u_i v_i)}.$$

Then, $A(t)$ can be estimated by nonparametrically regressing the z_i's on the t_i's. Cormier et al. (2014) fit a regression model with the quantile smoothing splines (Koenker et al., 1994; Ng and Maechler, 2007) subject to the convexity, boundary, and endpoint constraints of $A(t)$ with the R package *cobs* (Ng and Maechler, 2020). Nonetheless, the boundary constraints in (8) were imposed by imposing that $\hat{A}(j/N) \geqslant \max(j/N, 1 - j/N)$, where $j \in \{1, \ldots, N-1\}$ for some large N such as $N = 100$. In fact, given that $A(t)$ is convex and passes through 1 when $t = 0$ and $t = 1$, an elegant and efficient way to impose the constraints is to require

$$A'(0) \geqslant -1, \quad A'(1) \leqslant 1, \tag{9}$$

where $A'(t)$ is the derivative of $A(t)$.

We estimate $A(t)$ in a shape-restricted regression problem with the additional derivative and endpoint conditions at the boundaries, which guarantees the curvature and boundary constraints. Let $\boldsymbol{C}(t_i)$ denote a vector that consists of C-splines evaluated at t_i and let $\mathrm{d}\boldsymbol{C}(t)/\mathrm{d}t$ denote the elementwise first derivative of $\boldsymbol{C}(t)$. Following Problem (7) in Section 4.1, we applied C-splines to estimate $A(t)$ and reformulated the estimation problem to obtain the following quadratic programming problem,

$$-\boldsymbol{z}^\top \boldsymbol{X}\boldsymbol{\theta} + \frac{1}{2}\boldsymbol{\theta}\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta}, \text{ s.t. } \boldsymbol{A}_1^\top \boldsymbol{\theta} = \boldsymbol{1},\ \boldsymbol{A}_2^\top \boldsymbol{\theta} \geqslant -\boldsymbol{1},\ \boldsymbol{A}_3^\top \boldsymbol{\theta} \geqslant \boldsymbol{0},$$

where $\boldsymbol{z} = (z_1, \ldots, z_n)^\top$, $\boldsymbol{t} = (t_1, \ldots, t_n)^\top$, $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{1} & \boldsymbol{t} & \boldsymbol{C}(\boldsymbol{t})\end{bmatrix}$, $\boldsymbol{C}(\boldsymbol{t}) = \begin{bmatrix}\boldsymbol{C}(t_1) & \cdots & \boldsymbol{C}(t_n)\end{bmatrix}^\top$,

$$\boldsymbol{A}_1^\top = \begin{bmatrix}1 & 0 & \boldsymbol{C}(0)^\top \\ 1 & 1 & \boldsymbol{C}(1)^\top\end{bmatrix}, \quad \boldsymbol{A}_2^\top = \begin{bmatrix}0 & 1 & \mathrm{d}\boldsymbol{C}(t)/\mathrm{d}t\,\big|_{t=0} \\ 0 & -1 & -\,\mathrm{d}\boldsymbol{C}(t)/\mathrm{d}t\,\big|_{t=1}\end{bmatrix},$$

and $\boldsymbol{A}_3^\top = \begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} & \mathcal{I}_p\end{bmatrix}$. The constraints imposed by $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, and $\boldsymbol{A}_3$ correspond to the endpoint constraints, the boundary conditions in (9), and the convexity restriction, respectively.

Given the formulated quadratic programming problem, we wrote a straightforward implementation of the estimation procedure named `fitA()` as follows:

```
fitA <- function(x, y, ...) {
    ## create C-spline basis functions
    x_mat <- cSpline(x, ..., Boundary.knots = c(0, 1), intercept = TRUE)
    dx_mat <- cbind(0, 1, deriv(predict(x_mat, c(0, 1))))    # first derivatives
    x_mat <- cbind("(Intercept)" = 1, "(Linear)" = x, x_mat) # matrix X
    A1 <- cbind(c(1, rep(0, ncol(x_mat) - 1L)), 1)           # matrix A_1
    A2 <- cbind(dx_mat[1, ], dx_mat[2, ])                    # matrix A_2
    A3 <- rbind(0, 0, diag(ncol(x_mat) - 2))                 # matrix A_3
    A_mat <- cbind(A1, A2, A3)                               # all constraints
    b0 <- c(1, 1, - 1, - 1, rep(0, ncol(A_mat) - 4))         # RHS of >=
    d_vec <- crossprod(x_mat, y)                             # (z^T X)^T
    D_mat <- crossprod(x_mat)                                # X^T X
    res <- quadprog::solve.QP(D_mat, d_vec, A_mat, bvec = b0,
                              meq = 2)      # first 2 restrictions are equalities
    list(xt = x, coefficients = res$solution,
         fitted = as.numeric(x_mat %*% res$solution))
}
```

For example, we constructed a non-exchangeable extreme-value copula by using Khoudraji's device (Khoudraji, 1995) $\mathrm{kho}_{\boldsymbol{s}}(C_1, C_2)$ with shape vector $\boldsymbol{s} \in [0,1]^2$ defined by

$$\mathrm{kho}_{\boldsymbol{s}}(C_1, C_2)(\boldsymbol{u}) = C_1(\boldsymbol{u}^{\boldsymbol{1}-\boldsymbol{s}})C_2(\boldsymbol{u}^{\boldsymbol{s}}),$$

where C_1 and C_2 are two copulas, and $\boldsymbol{u}^{\boldsymbol{v}} = (u_1^{v_1}, u_2^{v_2})$ follows the elementwise definition for all $\boldsymbol{u}, \boldsymbol{v} \in [0,1]^2$. We took C_1 and C_2 to be the independence copula Π and the Gumbel–Hougaard copula with parameter $\alpha = 4$, respectively, which resulted in a non-exchangeable version of the Gumbel-Hougaard copula by choosing s_2 close to 1.

To demonstrate the usage of `fitA()` and to compare the fitted with the true Pickands dependence function $A_\theta^{\mathrm{KGH}}(t)$, we wrote two helper functions named `kc()` and `getZ()` to set

some default values ($(\alpha, s_2) = (4, 0.95)$) when generating data from the bivariate Khoudraji–Gumbel–Hougaard (KGH) family. We obtained the pairs of $\{(t_i, z_i)\}$ with the package *copula* (Hofert et al., 2020). The Pickands dependence function denoted by $A_\theta^{\mathrm{KGH}}(t)$ can be obtained by applying the function `A()` of the package *copula* following Section 3.4.2 of Hofert et al. (2018).

```
library(copula)
kc <- function(s1, s2 = 0.95, alpha = 4) {
    khoudrajiCopula(copula2 = gumbelCopula(alpha), shapes = c(s1, s2))
}
## Obtain the pairs of {(t_i, z_i)}
getZ <- function(dat) {
    pseudo <- pobs(dat)
    loguv <- rowSums(log(pseudo))
    cvalue <- C.n(pseudo, pseudo)
    data.frame(ti = log(pseudo[, 2]) / loguv, zi = log(cvalue) / loguv)
}
```

In the following code chunk, we considered two different settings where $s_1 = 0.4$ or $s_1 = 0.8$ was set, and simulated bivariate samples of size 500 in each setting. For each simulated dataset, we calculated the pairs $\{(t_i, z_i)\}$ and fitted convex regression splines from C-spline basis functions of degrees of freedom 8, 12, and 16, respectively.

```
set.seed(1)
n <- 500
s1_vec <- c(0.4, 0.8)
dfs <- c(8, 12, 16)
plot_dat <- do.call(rbind, lapply(s1_vec, function(s1) {
    kg <- kc(s1)
    dat <- getZ(rCopula(n, kg))
    dat$est <- A(kg, dat$ti)
    dat$model <- "Truth"
    dat2 <- do.call(rbind, lapply(dfs, function(df) {
        Ahat <- with(dat, fitA(ti, zi, df = df))$fitted
        data.frame(ti = dat$ti, zi = dat$zi, est = Ahat,
                   model = paste("C-splines: df =", df))
    }))
    out <- rbind(dat, dat2)
    out$s1 <- s1
    out
}))
```

The simulated pairs and the fitted curves are visualized in Figure 6, which also includes the corresponding true Pickands dependence functions $A_\theta^{\mathrm{KGH}}(t)$ for comparison. The fitted curves of different degrees of freedom are similar to one another, all reasonably close to $A_\theta^{\mathrm{KGH}}(t)$ whether $s_1 = 0.4$ or $s_1 = 0.8$ for the simulated dataset. In addition, the plots visually confirm that the fitted curves satisfy both the boundary constraints and curvature constraints as they should by design.

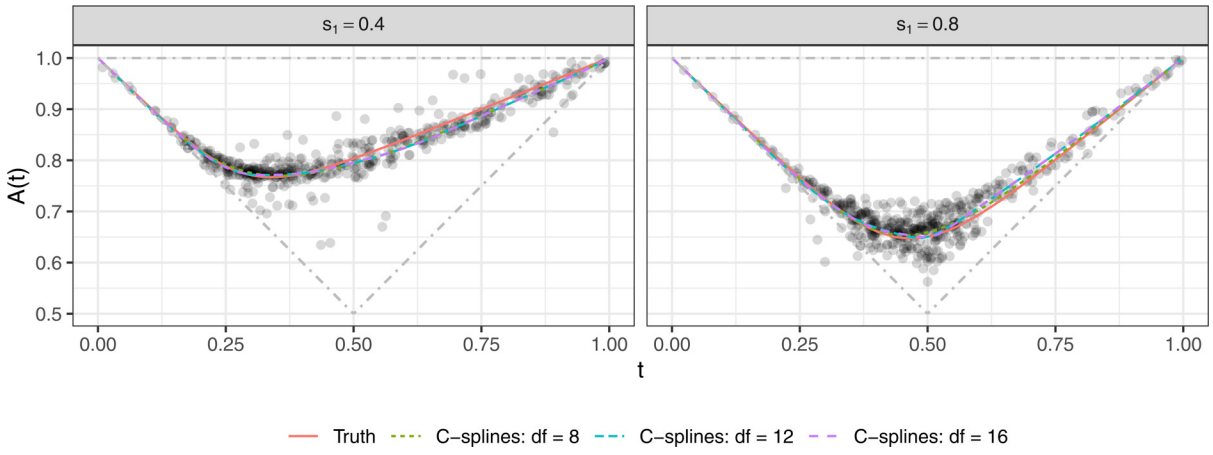

Figure 6: The Pickands dependence function of the bivariate Khoudraji–Gumbel–Hougaard copula for distinct values of shape parameter s_1 and their estimates by C-splines of different degrees of freedom. The dash-dotted lines in gray represent the boundary constraints.

5 Discussion

The shape-restricted splines are the main part of the functionalities provided by the package *splines2*. Additional features that are not reviewed here include generalized Bernstein polynomials, B-splines, natural cubic splines, and an *Rcpp* interface; see package vignettes for details. The R interface of the main functions in *splines2* follows that of the function `bs()` in the package *splines* for consistency. The derivatives and integrals of the spline basis functions are handy in applications when they are needed. The *Rcpp* interface allows direct access to the C++ implementation for R package developers. Flexible and powerful as the *splines2* package is, it is the users' responsibility in practice to be aware of the applicability and limitations of the spline basis functions. For example, for monotone regression in general, an intercept term needs to be considered in addition to the I-spline basis functions. For a general regression problem without shape constraints, C-splines should not be used without a good reason.

The efficiency of the *splines2* package was compared with other packages for the same tasks. We conducted micro-benchmarks to investigate the relative performance of our implementation with the help of the package *microbenchmark* (Mersmann, 2019). The tasks were to evaluate the B-splines (and their derivatives and integrals), Bernstein polynomials, natural cubic splines, and periodic splines at a sequence of equally spaced points in $[0, 1]$ with step size 0.001. The boundary knots were set to be 0 and 1, respectively, and the internal knots were set to be $\{0.1, \ldots, 0.9\}$ when they are applicable. The micro-benchmark results from 1000 replicates are visualized in Figure 7. While the micro-benchmarks are by no means comprehensive, they suggest the high efficiency of the implementation in *splines2*. For instance, the package *ibs* implements the integral of the B-splines by directly following the recursive formula, while the implementation in the package *splines2* takes advantage of the local support property of B-splines. Consequently, the function `ibs::ibs()` is much slower than the function `splines2::ibs()`, even though both are based on C++ behind the scene.

We restricted our attention to implementations in R. There are of course similar implementations in other programming languages. For example, the modules *scipy.signal* and *scipy.interpolate* of *Scipy* of Python, and the *Curve Fitting Toolbox* of MATLAB provide B-splines func-

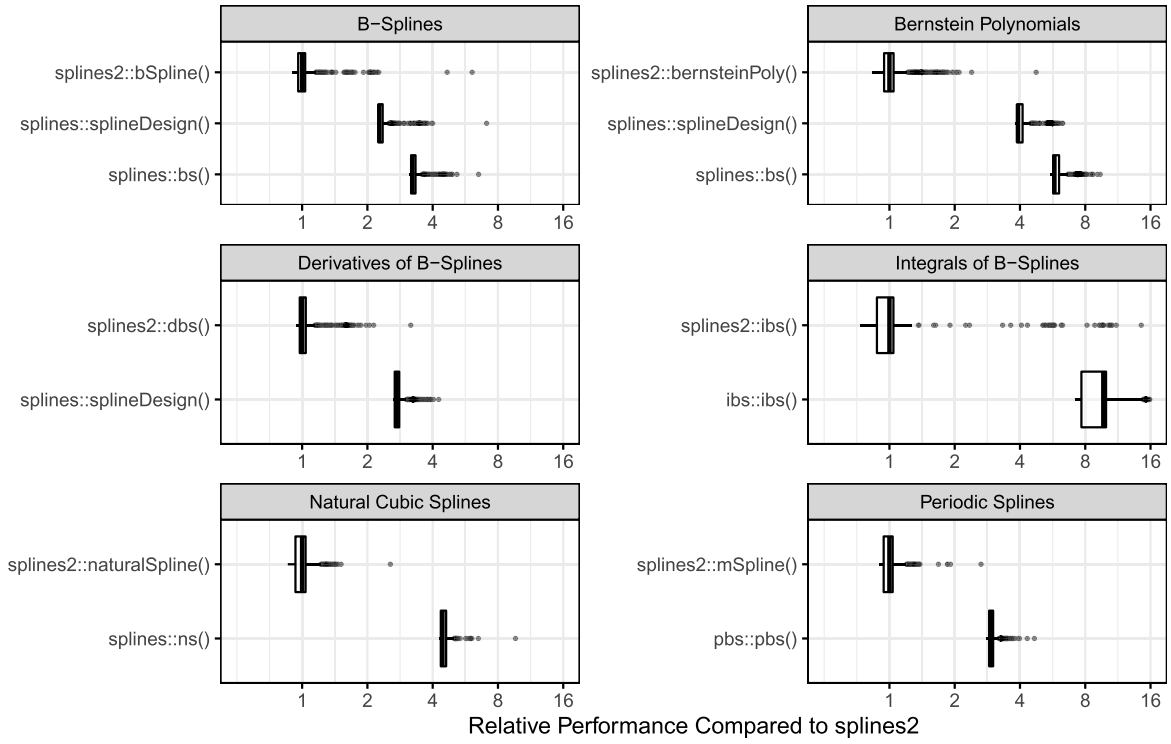

Figure 7: Example micro-benchmark results for reference.

tionality. Detailed introduction and comparison to those implementations are beyond the scope of this article.

Supplementary Material

We review the generalized Bernstein polynomials, B-splines, and Natural cubic splines implemented in the package *splines2* but not covered in the main text. We also provide the R code that produced the micro-benchmark results.

References

Chen F (2018). *ibs: Integral of B-Spline Functions*. R package version 1.4.

Cormier E, Genest C, Nešlehová JG (2014). Using B-splines for nonparametric inference on bivariate extreme-value copulas. *Extremes*, 17(4): 633–659.

Curry HB, Schoenberg IJ (1966). On Pólya frequency functions IV: The fundamental spline functions and their limits. *Journal d'Analyse Mathématique*, 17(1): 71–107.

De Boor C (1978). *A Practical Guide to Splines*, volume 27. Springer-Verlag, New York.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer.

Eddelbuettel D, Sanderson C (2014). RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71: 1054–1063.

Eilers PHC, Marx BD (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2): 89–102.

Farin G, Hansford D (2000). *The Essentials of CAGD*. CRC Press.

Fu H, Luo J, Qu Y (2016). Hypoglycemic events analysis via recurrent time-to-event (HEART) models. *Journal of Biopharmaceutical Statistics*, 26(2): 280–298.

Hofert M, Kojadinovic I, Maechler M, Yan J (2020). *copula: Multivariate Dependence with Copulas*. R package version 1.0-1.

Hofert M, Kojadinovic I, Mächler M, Yan J (2018). *Elements of Copula Modeling with R*. Springer.

Khoudraji A (1995). *Contributions à l'étude des copules et à la modélisation de valeurs extrêmes bivariées*, Ph.D. thesis, Université Laval, Québec, Canada.

Koenker R, Ng P, Portnoy S (1994). Quantile smoothing splines. *Biometrika*, 81(4): 673–680.

Mersmann O (2019). *microbenchmark: Accurate Timing Functions*. R package version 1.4-7.

Meyer MC (2008). Inference using shape-restricted regression splines. *The Annals of Applied Statistics*, 2(3): 1013–1033.

Mogstad M, Santos A, Torgovitsky A (2018). Using instrumental variables for inference about policy relevant treatment parameters. *Econometrica*, 86(5): 1589–1619.

Ng P, Maechler M (2007). A fast and efficient implementation of qualitatively constrained quantile smoothing splines. *Statistical Modelling*, 7(4): 315–328.

Ng PT, Maechler M (2020). *COBS – Constrained B-Splines (Sparse Matrix Based)*. R package version 1.3-4.

Perperoglou A, Sauerbrei W, Abrahamowicz M, Schmid M (2019). A review of spline function procedures in R. *BMC Medical Research Methodology*, 19(1): 46.

Pickands J (1981). Multivariate extreme value distribution (with discussion). In: *Proceedings 43th, Session of International Statistical Institution*, volume 2, 859–878, 894–902. Buenos Aires.

Piegl L, Tiller W (1997). *The NURBS Book*. Springer Science & Business Media, 2 edition.

Prautzsch H, Boehm W, Paluszny M (2002). *Bézier and B-Spline Techniques*. Springer Science & Business Media.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ramsay JO (1988). Monotone regression splines in action. *Statistical Science*, 3(4): 425–441.

Redd A (2015). *orthogonalsplinebasis: Orthogonal B-Spline Basis Functions*. R package version 0.1.6.

Rosenberg PS (1995). Hazard function estimation using B-splines. *Biometrics*, 51(3): 874–887.

Ruppert D, Wand MP, Carroll RJ (2003). *Semiparametric Regression. 12*. Cambridge University Press.

Schumaker L (2007). *Spline Functions: Basic Theory*. Cambridge University Press. 3 edition.

Shea J, Torgovitsky A (2020). *ivmte: An R Package for Implementing Marginal Treatment Effect Methods*. University of Chicago. Becker Friedman Institute for Economics. Working Paper (2020-01).

Turlach BA, Weingessel A (2019). *quadprog: Functions to Solve Quadratic Programming Problems*. R package version 1.5-8.

Wand M (2018). *SemiPar: Semiparametic Regression*. R package version 1.0-4.2.

Wang J, Ghosh SK (2012). Shape restricted nonparametric regression with Bernstein polynomials. *Computational Statistics & Data Analysis*, 56(9): 2729–2741.

Wang S (2013). *pbs: Periodic B-Splines*. R package version 1.1.

Wang W, Fu H, Yan J (2020). *reda: Recurrent Event Data Analysis*. R package version 0.5.2.