Supplementary Materials to "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.*

1 Polynomial/Spline Basis Functions

In this section, we introduce the polynomial or spline basis functions that are implemented in the package *splines2* but not covered in the main text.

1.1 Generalized Bernstein Polynomials

Bernstein polynomials of degree k correspond to the polynomial terms from the binomial expansion of $1 = [u + (1-u)]^k$. The ith Bernstein polynomial basis denoted by $\tilde{G}_{i,k}(u)$ are defined for $0 \le u \le 1$ as

$$\tilde{G}_{i,k}(u) = \binom{k}{i} u^i (1-u)^{k-i}, \ i \in \{0, \ldots, k\}. \tag{1}$$

For a given boundary $[L, U]$ and $L \le x \le U$, the generalized Bernstein polynomials are defined by replacing u with $(x-L)/(U-L)$ in (1),

$$G_{i,k}(x) = \frac{1}{(U-L)^k} \binom{k}{i} (x-L)^i (U-x)^{k-i}, \ i \in \{0, \ldots, k\}, \tag{2}$$

which can also be defined through a recursive manner (see **?**, Chapter 2) as follows:

$$G_{i,k}(x) = \frac{x-L}{U-L} G_{i-1,k-1}(x) + \frac{U-x}{U-L} G_{i,k-1}(x), \ i \in \{0, \ldots, k\},$$

where $G_{0,0}(x) = 1$ and $G_{-1,k-1}(x) = G_{k,k-1} = 0$ for $k \in \{1, 2, \ldots\}$. In fact, the generalized Bernstein polynomials of degree k are equivalent to the B-splines of a same degree with no internal knots. See Section 1.2 for a brief introduction to B-splines. Therefore, Bernstein polynomials and B-splines share several common properties. For example, we have $G_{i,k}(x) \ge 0$, $\sum_{i=0}^{k} G_{i,k}(x) = 1$, $G_{i,k}(x-L) = G_{k-i,k}(U-x)$, and $G_{i,k}(L) = G_{k-i,k}(U) = \mathbb{1}(i=0)$, $\forall i \in \{0, \ldots, k\}$.

The first derivatives of the generalized Bernstein polynomials can be derived directly from (2) or given recursively by

$$\frac{\mathrm{d}}{\mathrm{d}x} G_{i,k}(x) = \frac{k}{U-L} \left(G_{i-1,k-1}(x) - G_{i,k-1}(x) \right),$$

which makes it easier to obtain the derivatives of higher order. Similarly, the integrals of the generalized Bernstein polynomials can be defined recursively by

$$\int_L^x G_{i,k}(t) dt = \frac{U-L}{k+1} \sum_{l=i+1}^{k+1} G_{l,k+1}(x).$$

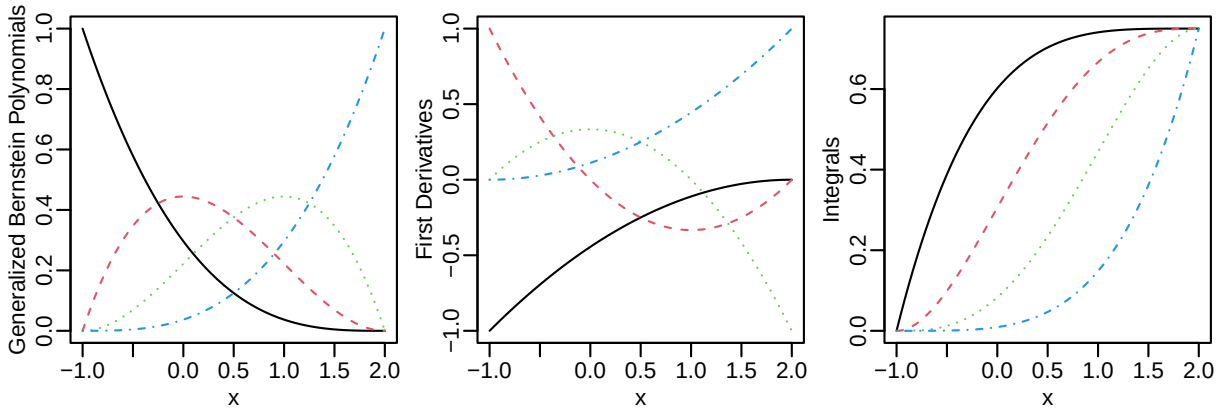

Figure S1: Generalized Cubic Bernstein Polynomial basis functions and their first derivatives and integrals.

Figure S1 visualizes the generalized Bernstein polynomial basis functions of degree 3, their first derivatives, and integrals, respectively, which can be obtained by the function `bernsteinPoly()` of the package *splines2* as follows:

```
x <- seq.int(- 1, 2, 0.01) # set x from -1 to 2
## generalized cubic Bernstein polynomials, their derivatives, and integrals
bp_mat <- bernsteinPoly(x, intercept = TRUE)                    # basis functions
dbp_mat <- bernsteinPoly(x, intercept = TRUE, derivs = 1)       # 1st derivatives
ibp_mat <- bernsteinPoly(x, intercept = TRUE, integral = TRUE) # integrals
```

The cubic basis functions were produced by default. For the first derivatives, we specified the argument `derivs = 1` of the function `bernsteinPoly()`. Alternatively, we can utilize the function `deriv()` to obtain the first derivatives with ease (e.g., `deriv(bp_mat)`). The integrals were returned when we specified `integral = TRUE` (and `derivs = 0` by default).

1.2 B-Splines

B-splines or basic splines are widely used tools in numerical analysis and have close connections to other spline basis functions introduced in this article and the main text. For a given $k \in \{1, 2, \ldots\}$, the ith B-spline basis of degree k (or order $d = k+1$) denoted by $B_{i,k}(x)$ based on the simple knot sequence $\boldsymbol{s}_k$ can be defined by the following Cox–de Boor recursive formula (**??**):

$$B_{i,k}(x \mid \boldsymbol{s}_k) = \left(\frac{x - t_i}{t_{i+k} - t_i}\right) B_{i,k-1}(x \mid \boldsymbol{s}_k) + \left(\frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}}\right) B_{i+1,k-1}(x \mid \boldsymbol{s}_k), \tag{3}$$

with

$$B_{l,0}(x \mid \boldsymbol{s}_k) = \begin{cases} 1, & t_l \le x < t_{l+1} \\ 0, & \text{otherwise} \end{cases}, \; l \in \{1, \ldots, d + p - 1\}, \tag{4}$$

where $L \le x < U$, $p = m + d$ represents the degrees of freedom, and $i \in \{1, \ldots, p\}$. The recursive definition implicitly follows the convention that $B_{i,k-1}(x \mid \boldsymbol{s}_k) = B_{i,k-1}(x \mid \boldsymbol{s}_k)/(t_{i+k} - t_i) = 0$ if

$t_{i+k} = t_i$. In practice, we may let $B_{i,k}(U \mid \boldsymbol{s}_k)$ take the limit from the left so that each basis is defined for all $x \in \{x \mid L \leq x \leq U\}$.

Notice that the simple knot sequence $\boldsymbol{s}_k$ that depends on the boundary knots, the internal knots, and the degree k is fixed when we apply the recursive formula (3). For instance, the simple knot sequence $\boldsymbol{s}_2$ for the quadratic spline basis functions satisfies $t_1 = t_2 = t_3 = L < t_4 < \cdots < t_{m+3} < U = t_{m+4} = t_{m+5} = t_{m+6}$, where $t_{j+3} = \xi_j$, $j \in \{1, \ldots, m\}$. From (4), we have $B_{1,0}(x \mid \boldsymbol{s}_2) = B_{2,0}(x \mid \boldsymbol{s}_2) = 0$, $B_{3,0}(x \mid \boldsymbol{s}_2) = \mathbf{1}(t_3 \leq x < t_4)$, ..., $B_{m+3,0}(x \mid \boldsymbol{s}_2) = \mathbf{1}(t_{m+3} \leq x < t_{m+4})$, and $B_{m+4,0}(x \mid \boldsymbol{s}_2) = B_{m+5,0}(x \mid \boldsymbol{s}_2) = 0$. Then we may obtain $B_{i,1}(x \mid \boldsymbol{s}_2)$ for $i \in \{1, \ldots, p+1\}$ by setting $k = 1$ in (3). At last, the desired quadratic basis $B_{i,1}(x \mid \boldsymbol{s}_2)$ for $i \in \{1, \ldots, p\}$ can be produced by (3) again with $k = 2$.

The B-splines of degree k are nonnegative over $[L, R]$ subject to $\sum_{i=1}^{p} B_{i,k}(x \mid \boldsymbol{s}_k) = 1$. In addition, the B-splines have *local support* meaning that $B_{i,k}(x \mid \boldsymbol{s}_k)$ is positive for $x \in (t_i, t_{i+k+1})$ but zero for x outside of $[t_i, t_{i+k+1}]$, which is an important property we may utilize when implementing (3). Although the B-splines are defined recursively, they are still polynomials in essence. Therefore, one would expect closed-form expressions of their derivatives and integrals. **?**, Chapter 10 gave the derivatives and integrals of B-splines in closed-form formulas, which were more recently reviewed and discussed by **?**.

The first derivative of ith B-spline basis of degree k, for $k \in \{1, 2, \ldots\}$, can be derived by induction using (3) and is given by

$$\frac{\mathrm{d}}{\mathrm{d}x} B_{i,k}(x \mid \boldsymbol{s}_k) = \left(\frac{k}{t_{i+k} - t_i}\right) B_{i,k-1}(x \mid \boldsymbol{s}_k) - \left(\frac{k}{t_{i+k+1} - t_{i+1}}\right) B_{i+1,k-1}(x \mid \boldsymbol{s}_k), \tag{5}$$

where $i \in \{1, \ldots, p\}$. We may derive the second derivatives by taking the derivatives again on both sides of (5) and applying (5) for B-splines of degree $k-1$. Such a procedure can be repeated for the derivatives of higher order.

To derive the integrals of B-splines, let us consider the first derivative of the following spline function $s(x) = \sum_{i=1}^{p+1} \beta_i B_{i,k+1}(x \mid \boldsymbol{s}_{k+1})$,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x} \sum_{i=1}^{p+1} \beta_i B_{i,k+1}(x \mid \boldsymbol{s}_{k+1}) &= \sum_{i=1}^{p+1} \left[\frac{\beta_i (k+1)}{t_{i+k+1} - t_i}\right] B_{i,k}(x \mid \boldsymbol{s}_{k+1}) - \left[\frac{\beta_i (k+1)}{t_{i+k+2} - t_{i+1}}\right] B_{i+1,k}(x \mid \boldsymbol{s}_{k+1}) \\
&= \sum_{i=2}^{p+1} (k+1) \left(\frac{\beta_i - \beta_{i-1}}{t_{i+k+1} - t_i}\right) B_{i,k}(x \mid \boldsymbol{s}_{k+1}),
\end{aligned} \tag{6}$$

where t_l is defined for $\boldsymbol{s}_{k+1}$, $l \in \{1, \ldots, d+p+2\}$, and the second equation holds as we have $B_{1,k}(x \mid \boldsymbol{s}_{k+1}) = B_{p+2,k}(x \mid \boldsymbol{s}_{k+1}) = 0$. Let $\beta_l = \mathbf{1}(l \geq j)$ for a given $j \in \{2, \ldots, p+1\}$. Then

$$\frac{\mathrm{d}}{\mathrm{d}x} \sum_{l=j}^{p+1} B_{l,k+1}(x \mid \boldsymbol{s}_{k+1}) = \left(\frac{k+1}{t_{j+k+1} - t_j}\right) B_{j,k}(x \mid \boldsymbol{s}_{k+1}). \tag{7}$$

By integrating the both side of (7), replacing $B_{j,k}(x \mid \boldsymbol{s}_{k+1})$ equivalently with $B_{j-1,k}(x \mid \boldsymbol{s}_k)$, and replacing j with $i+1$, we obtain the integral of $B_{i,k}(x \mid \boldsymbol{s}_k)$ from t_1 to x as follows:

$$\int_{t_1}^{x} B_{i,k}(t \mid \boldsymbol{s}_k) dt = \left(\frac{t_{i+k+2} - t_{i+1}}{k+1}\right) \sum_{l=i+1}^{p+1} B_{l,k+1}(x \mid \boldsymbol{s}_{k+1}). \tag{8}$$

where t_{i+k+2} and t_{i+1} are defined for $\boldsymbol{s}_{k+1}$, $i \in \{1, \ldots, p\}$.

1.3 Natural Cubic Splines

A cubic spline function $s(x) = \boldsymbol{\beta}^\top \boldsymbol{B}_3(x)$ is called a natural cubic spline function if it is subject to the additional constraints that $\mathrm{d}^2 s(x)/\mathrm{d}x^2 \,\big|_{x=L} = \mathrm{d}^2 s(x)/\mathrm{d}x^2 \,\big|_{x=U} = 0$, where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^\top$, $\boldsymbol{B}_3(x) = (B_{1,3}(x), \ldots, B_{p,3}(x))^\top$ is the cubic spline vector, and $p = m + 4$. In addition, a natural cubic spline function is linear beyond the boundary, which suggests that the first (second) derivatives remain constants (zeros) outside the boundary. By definition, one can construct a natural cubic spline function from cubic splines, such as cubic B-splines, and impose the curvature constraints at and beyond the boundary. Alternatively, we show how to derive a set of cubic spline basis functions that satisfy those curvature conditions.

Let $\boldsymbol{B}_3(x)$ represent the cubic B-spline basis vector for given boundary and m distinct interior knots. Consider a set of cubic splines $\{N_i(x) \mid i \in \{1, \ldots, l\}\}$ satisfying that $\boldsymbol{N}(x) = \boldsymbol{H}^\top \boldsymbol{B}_3(x)$, where $\boldsymbol{N}(x) = (N_1(x), \ldots, N_l(x))^\top$ and $\boldsymbol{H}$ is a $p \times l$ full column rank matrix. Suppose $\tilde{s}(x) = \tilde{\boldsymbol{\beta}}^\top \boldsymbol{N}(x) = \tilde{\boldsymbol{\beta}}^\top \boldsymbol{H}^\top \boldsymbol{B}_3(x)$ is a natural cubic spline function, where $\tilde{\boldsymbol{\beta}} = (\tilde{\beta}_1, \ldots, \tilde{\beta}_l)^\top$. Let $\mathrm{d}^2 \boldsymbol{B}_3(x)/\mathrm{d}x^2 = (\mathrm{d}^2 B_{1,3}(x)/\mathrm{d}x^2, \ldots, \mathrm{d}^2 B_{p,3}(x)/\mathrm{d}x^2)^\top$ denote the elementwise second derivatives of $\boldsymbol{B}_3(x)$. Then the curvature conditions at the boundary are equivalent to $\boldsymbol{C}^\top \boldsymbol{H} \tilde{\boldsymbol{\beta}} = \boldsymbol{0}$, where $\boldsymbol{C} = (\mathrm{d}^2 \boldsymbol{B}_3(x)/\mathrm{d}x^2 \,\big|_{x=L}, \mathrm{d}^2 \boldsymbol{B}_3(x)/\mathrm{d}x^2 \,\big|_{x=U})$. It can be verified from (5) that $\boldsymbol{C}$ is of full column rank. We define $\boldsymbol{N}(x) = \boldsymbol{H}^\top \boldsymbol{B}_3(x)$ forms a set of natural cubic splines if $\boldsymbol{C}^\top \boldsymbol{H} \tilde{\boldsymbol{\beta}} = \boldsymbol{0}$ holds $\forall \boldsymbol{\beta} \in \mathbb{R}^p$. Therefore, it is desired to find a matrix $\boldsymbol{H}$ such that $\boldsymbol{C}^\top \boldsymbol{H} = \boldsymbol{0}$, i.e., the columns of $\boldsymbol{H}$ belong to the null space of $\boldsymbol{C}^\top$.

For a $p \times q$ full column rank rectangular matrix $\boldsymbol{C}$, where $p > q = 2$, one may obtain a set of orthogonal basis functions for the null space of $\boldsymbol{C}^\top$ from the QR decomposition of $\boldsymbol{C}$. More specifically, suppose the QR decomposition of $\boldsymbol{C}$ gives

$$\boldsymbol{C} = \boldsymbol{Q}\boldsymbol{R} = [\boldsymbol{Q}_1 \; \boldsymbol{Q}_2] \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{Q}_1 \boldsymbol{R}_1,$$

where $\boldsymbol{Q}$ is a $p \times p$ orthogonal matrix, $\boldsymbol{Q}_1 \in \mathbb{R}^{p \times q}$ and $\boldsymbol{Q}_2 \in \mathbb{R}^{p \times (p-q)}$ subject to $\boldsymbol{Q}_1^\top \boldsymbol{Q}_2 = \boldsymbol{0}$, $\boldsymbol{R} \in \mathbb{R}^{p \times q}$, and $\boldsymbol{R}_1 \in \mathbb{R}^{q \times q}$. Then the columns of $\boldsymbol{Q}_2$ can serve as the orthogonal basis functions for the null space of $\boldsymbol{C}^\top$ as $\boldsymbol{C}^\top \boldsymbol{Q}_2 = \boldsymbol{0}$. Let $\boldsymbol{H} = \boldsymbol{Q}_2$ and we obtain a set of natural cubic spline bases as $\boldsymbol{N}(x) = \boldsymbol{Q}_2^\top \boldsymbol{B}_3(x)$. It is the procedure that the function `ns()` of the *splines* package follows.

The approach based on the QR decomposition works for any cubic splines. However, it does not take advantage of the local support property of B-splines. We can explicitly write down $\boldsymbol{C}$ for cubic B-splines from (5). The first and second columns of $\boldsymbol{C}$ denoted by $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are, respectively,

$$\boldsymbol{C}_1 = \left(C_{1,1}, C_{2,1}, C_{3,1}, \boldsymbol{0}_{p-3}^\top\right)^\top, \; \boldsymbol{C}_2 = \left(\boldsymbol{0}_{p-3}^\top, C_{p-2,2}, C_{p-1,2}, C_{p,2}\right)^\top,$$

where $C_{1,1} = 6[(t_5 - t_2)(t_5 - t_3)]^{-1}$, $C_{3,1} = 6[(t_6 - t_3)(t_5 - t_3)]^{-1}$, $C_{2,1} = -C_{1,1} - C_{3,1}$, $C_{p-2,2} = 6[(t_{m+6} - t_{m+3})(t_{m+6} - t_{m+4})]^{-1}$, $C_{p,2} = 6[(t_{m+7} - t_{m+4})(t_{m+6} - t_{m+4})]^{-1}$, $C_{p-1,2} = -C_{p-2,2} - C_{p,2}$, and $\boldsymbol{0}_{p-3}$ is a zero vector of length $(p-3)$. Given the explicit expression of $\boldsymbol{C}$ for cubic B-splines and number of internal knots, we are able to obtain specific choices of $\boldsymbol{H}$ such that $\boldsymbol{C}^\top \boldsymbol{H} = \boldsymbol{0}$ without using the QR decomposition. When no internal knot or one internal knot is placed, we may choose, respectively,

$$\boldsymbol{H}^\top = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}, \tag{9}$$

and

$$\boldsymbol{H}^\top = \begin{bmatrix} -C_{2,1}/C_{1,1} & 1 & 0 & 0 & 0 \\ 0 & -C_{3,1}/C_{2,1} & 1 & -C_{p-2,2}/C_{p-1,2} & 0 \\ 0 & 0 & 0 & 1 & C_{p-1,2}/C_{p,2} \end{bmatrix}. \tag{10}$$

While two or more internal knots are placed, we can let

$$\boldsymbol{H}^\top = \begin{bmatrix} 1 & 1 & 1 & \mathbf{0}_{m-2}^\top & 0 & 0 & 0 \\ 0 & 1 & -C_{2,1}/C_{3,1} & \mathbf{0}_{m-2}^\top & 0 & 0 & 0 \\ \mathbf{0}_{m-2} & \mathbf{0}_{m-2} & \mathbf{0}_{m-2} & \boldsymbol{I}_{m-2} & \mathbf{0}_{m-2} & \mathbf{0}_{m-2} & \mathbf{0}_{m-2} \\ 0 & 0 & 0 & \mathbf{0}_{m-2}^\top & -C_{p-1,2}/C_{p-2,2} & 1 & 0 \\ 0 & 0 & 0 & \mathbf{0}_{m-2}^\top & 1 & 1 & 1 \end{bmatrix}, \tag{11}$$

where $\boldsymbol{I}_{m-2}$ is a $(m-2) \times (m-2)$ identity matrix and $\mathbf{0}_{m-2}$ is a zero vector of length $(m-2)$. Notice that the chosen $\boldsymbol{H}$ consists of nonnegative elements. The natural cubic splines given by $\boldsymbol{N}(x) = \boldsymbol{H}^\top \boldsymbol{B}_3(x)$ are thus nonnegative within the boundary. Additionally, the derivatives and integrals of the resulting natural cubic splines can be obtained from the derived $\boldsymbol{H}^\top$. For example, the element-wise first derivatives of $\boldsymbol{N}(x)$ are given by $\boldsymbol{H}^\top \, \mathrm{d}\boldsymbol{B}_3(x)/\mathrm{d}x$. The function `naturalSpline()` in the *splines2* package follows this approach to produce natural cubic splines, their corresponding derivatives, and integrals.

2 Micro-Benchmarks

The R code that produced the micro-benchmark results in Section 5 of the main text is as follows.

```
library(microbenchmark)
library(splines)
library(splines2)

set.seed(123)
x <- seq.int(0, 1, 1e-3) # set x
degree <- 3              # cubic basis functions
ord <- degree + 1        # set order
internal_knots <- seq.int(0.1, 0.9, 0.1)      # set internal knots
boundary_knots <- range(x)                    # set boundary knots
## knot sequence without internal knots
all_knots0 <- rep(boundary_knots, each = ord)
## knot sequence with internal knots
all_knots <- sort(c(internal_knots, rep(boundary_knots, ord)))
coef_sp <- rnorm(length(all_knots) - ord) # set coef for comparing ibs::ibs()
derivs <- 2                               # comparing second derivatives

## B-splines
bs_benchmark <- microbenchmark(
    "splines::bs()" = bs(x, knots = internal_knots, degree = degree,
                         intercept = TRUE),
    "splines::splineDesign()" = splineDesign(x, knots = all_knots, ord = ord),
```

```
    "splines2::bSpline()" = bSpline(x, knots = internal_knots, degree = degree,
                                    intercept = TRUE),
    times = 1e3
)

## Bernstein Polynomial (B-splines without internal knots)
bp_benchmark <- microbenchmark(
    "splines::bs()" = bs(x, degree = degree, intercept = TRUE),
    "splines::splineDesign()" = splineDesign(x, knots = all_knots0, ord = ord),
    "splines2::bernsteinPoly()" = bernsteinPoly(x, degree = degree,
                                                intercept = TRUE),
    times = 1e3
)

## Second Derivatives of B-splines
dbs_benchmark <- microbenchmark(
    "splines::splineDesign()" = splineDesign(x, knots = all_knots,
                                             ord = ord, derivs = derivs),
    "splines2::dbs()" = dbs(x, derivs = derivs, knots = internal_knots,
                            degree = degree, intercept = TRUE),
    times = 1e3
)

## Integrals of B-splines
ibs_benchmark <- microbenchmark(
    "ibs::ibs()" = ibs::ibs(x, knots = all_knots, ord = ord, coef = coef_sp),
    "splines2::ibs()" = as.numeric(
        splines2::ibs(x, knots = internal_knots, degree = degree,
                      intercept = TRUE) %*% coef_sp
    ),
    times = 1e3
)

## Natural Cubic Splines
ns_benchmark <- microbenchmark(
    "splines::ns()" = ns(x, knots = internal_knots, intercept = TRUE),
    "splines2::naturalSpline()" = naturalSpline(
        x, knots = internal_knots, intercept = TRUE
    ),
    times = 1e3
)

## Periodic Splines
pbs_benchmark <- microbenchmark(
    "pbs::pbs()" = pbs::pbs(x, knots = internal_knots, degree = degree,
                            intercept = TRUE, periodic = TRUE),
```

```
    "splines2::mSpline()" = mSpline(
        x, knots = internal_knots, degree = degree,
        intercept = TRUE, periodic = TRUE
    ),
    times = 1e3
)
```

The information of the R session where the micro-benchmarks were performed is as follows:

```
xfun::session_info(package = c("splines", "splines2",
                               "ibs", "pbs", "microbenchmark"))
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Arch Linux
##
## Locale:
##   LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##   LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##   LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##   LC_PAPER=en_US.UTF-8       LC_NAME=C
##   LC_ADDRESS=C               LC_TELEPHONE=C
##   LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## Package version:
##   graphics_4.1.0           grDevices_4.1.0          ibs_1.4
##   methods_4.1.0            microbenchmark_1.4-7     pbs_1.1
##   Rcpp_1.0.6               RcppArmadillo_0.10.5.0.0 splines_4.1.0
##   splines2_0.4.3           stats_4.1.0              utils_4.1.0
```

We visualized the relative performance given in Figure 7 of the main text as follows:

```
## add labels
bs_benchmark$basis <- "B-Splines"
bp_benchmark$basis <- "Bernstein Polynomials"
dbs_benchmark$basis <- "Derivatives of B-Splines"
ibs_benchmark$basis <- "Integrals of B-Splines"
ns_benchmark$basis <- "Natural Cubic Splines"
pbs_benchmark$basis <- "Periodic Splines"

## relative benchmark by median of splines2
rel_med <- function(dat) {
    idx <- grepl("splines2", dat$expr, fixed = TRUE)
    med_x <- median(dat[idx, "time"])
    dat$time <- dat$time / med_x
    dat
```

```
}

## create data for the plot
gg_dat <- rbind(
    rel_med(bs_benchmark),
    rel_med(bp_benchmark),
    rel_med(dbs_benchmark),
    rel_med(ibs_benchmark),
    rel_med(ns_benchmark),
    rel_med(pbs_benchmark)
)
gg_dat <- as.data.frame(gg_dat)

## visualization with the help of ggplot2
library(ggplot2)
ggplot(data = gg_dat, aes(x = expr, y = time)) +
    geom_boxplot(outlier.size = 0.5, alpha = 0.5) +
    coord_flip() +
    facet_wrap(vars(basis), scales = "free", ncol = 2) +
    scale_y_continuous(trans = "log", breaks = c(1, 2, 4, 8, 16),
                       limits = c(0.5, 16)) +
    theme_bw() +
    ylab("Relative Performance Compared to splines2") +
    xlab("")
```

References

Bhatti M, Bracken P (2006). The calculation of integrals involving B-splines by means of recursion relations. *Applied Mathematics and Computation*, 172(1): 91–100.

Cox MG (1972). The numerical evaluation of B-splines. *IMA Journal of Applied Mathematics*, 10(2): 134–149.

De Boor C (1972). On calculating with B-splines. *Journal of Approximation Theory*, 6(1): 50–62.

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

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