mathematics

MDPI

Article

Generalized de Boor–Cox Formulas and Pyramids for Multi-Degree Spline Basis Functions

Xu Ma and Wanqiang Shen *

School of Science, Jiangnan University, Wuxi 214122, China
* Correspondence: shenwanqiang@jiangnan.edu.cn

Abstract: The conventional B-splines possess the de Boor–Cox formula, which relates to a pyramid algorithm. However, for multi-degree splines, a de Boor–Cox-type evaluation algorithm only exists in some special cases. This paper considers any multi-degree spline with arbitrary degree and continuity, and provides two generalized de Boor–Cox-type relations. One uses several lower degree polynomials to build a combination to evaluate basis functions, whose form is similar to using the de Boor–Cox formula several times. The other is a linear combination of two functions out of the recursive definition, which keeps the combination coefficient polynomials of degree 1, so it is more similar to the de Boor–Cox formula and can be illustrated by several pyramids with different heights. In the process of calculating the recursions, a recursive representation using the Bernstein basis is used and numerically analyzed.

Keywords: B-spline; multi-degree spline; de Boor–Cox formula; pyramid algorithm; continuity

MSC: 65D07; 41A15

Citation: Ma, X.; Shen, W. Generalized de Boor–Cox Formulas and Pyramids for Multi-Degree Spline Basis Functions. *Mathematics* **2023**, *11*, 367. https://doi.org/10.3390/math11020367

Academic Editors: Juan Cao, Li Zhang and Hongwei Lin

Received: 22 November 2022
Revised: 27 December 2022
Accepted: 5 January 2023
Published: 10 January 2023

1. Introduction

B-splines are well-established tools in computer-aided geometric design. Their basic functions build the unique normalized B-basis in their spanned space of piecewise polynomials, which has optimal shape-preserving properties [1]. Multi-degree splines (MD-splines for short), also called changeable-degree splines, allow different degrees for different segments. In their spanned space of piecewise polynomials with varying degrees, their basic functions also construct the unique normalized B-basis [2]. Therefore, they can be viewed as a direct extension of B-splines.

For MD-splines, the use of splines with varying degrees were investigated by [3] as a tool for approximation. Some variable-degree polynomial functions in [4,5] were constructed for shape-preserving interpolation and more theories were proposed in [6–8], in [9] the concept of multi-degree splines was proposed and some MD-spline curves of degree 1, 2 and 3 were built. In [10], a two-degree-spline basis is provided for dividing the process of degree elevation of B-spline into corner-cutting form. In [2,11], two types of MD-splines with different continuities were defined, and [12] deals with an explicit expression for MD-splines. The MD-spline degree-elevation property is provided in [13]. The definition of MD-splines is improved in [14]; two extended partitions are made ingeniously to unify the two types of MD-splines. Such extended partitions for B-splines were proposed in [15]. The support of each basis function is easily determined by the indexes of two extended partitions, and, using this configuration, arbitrary continuity MD-splines can be created. Recent references offer several numerical algorithms due to the inefficiency of the integral definition for MD-splines in numerical calculation. They employ simpler bases to express MD-splines and to compute the representation matrix. In [16], the continuity constraints between spline pieces were collected in a matrix and then its null space was computed by recurrence. In [17], it was proven that the output of this algorithm is exactly the basis of MDB-splines and a Chebyshevian extension of the construction was presented in [18]. The

algorithm in [19] repeats the reverse knot insertion (RKI) process to evaluate the transition matrix. An improved version of the algorithm in [19] is outlined in [20]. it is a numerically stable method, because it avoids the derivative operations and minimizes the number of floating-point operations.

B-splines $F_{i,n}$ are recursively computed by applying the de Boor–Cox formula in [21,22], which is related to a pyramid algorithm in [23], namely,

$$F_{i,n}(t) = \frac{t - t_i}{t_{i+n} - t_i} F_{i,n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} F_{i+1,n-1}(t) \tag{1}$$

In the formula, a B-spline of degree n is represented by two B-splines of degree $n-1$, where the combination coefficients are both polynomials of degree 1. According to [24,25], the MD-splines build a de Boor–Cox-type formula if the continuity orders between neighboring pieces of different degrees are no more than 1. For general cases, the de Boor–Cox-type formula does not exist [14].

Our work is focused on more generalized de Boor–Cox-type relations for arbitrary-degree MD-splines with any continuity orders and provides two generalized forms. The first form uses the de Boor–Cox formula several times. A B-spline basis function of degree n is represented as a combination of $s+1$ B-splines of degree $n-s$ with polynomial coefficients $\sigma_{i,n-s}(t), \sigma_{i+1,n-s}(t), \ldots, \sigma_{i+s,n-s}(t)$ of degree s, i.e.

$$F_{i,n}(t) = \sigma_{i,n-s}(t)F_{i,n-s}(t) + \sigma_{i+1,n-s}(t)F_{i+1,n-s}(t) + \ldots + \sigma_{i+s,n-s}(t)F_{i+s,n-s}(t) \tag{2}$$

For MD-splines, each function $N_{i,n}, i = D-n+1, \ldots, \mathcal{K}$ at level n can be represented by functions at recursive level $n-s$, where these functions should be C^0 continuous or discontinuous at knots in their support intervals. On each interval, the sum of the degree of a function involved in the recurrence is similar to (2) and that of its coefficient polynomial is equal to the degree of $N_{i,n}$. We provide an algorithm to calculate these coefficient polynomials. In the meantime, functions at recursive level $n-1$ can also be represented by these functions at recursive level $n-s$, allowing the first form to be translated into a new form. In this new form, a function $N_{i,n}$ at recursive level n is represented by a combination of $N_{i,n-1}$, $N_{i+1,n-1}$ and functions at recursive level $n-s$, where the coefficient polynomials of $N_{i,n-1}$, and $N_{i+1,n-1}$ are bothpolynomials of degree 1. In the second form, a function $N_{i,n}$ at recursive level n is represented by two functions with combination coefficient polynomials of degree 1. The left coefficient polynomial is the result of subtracting the left endpoint of $N_{i,n}$ from t, and the right coefficient polynomial is the result of subtracting t from the right endpoint of $N_{i,n}$. Two representation functions are not in the integral recursion, but they can be constructed using knots derivative information of functions at recursive level $n-1, n-2, ..., 1$. We present a building method based on the Taylor expansion of polynomials which demonstrates the theoretical viability of this form. If we replace functions that are not applicable to de Boor–Cox-type formulas with their construction polynomials in the recursion of MD-spline basis, then any function can be evaluated using this form formula. Multiple pyramids of varying heights can be used to depict this recurrence. To obtain the two forms for MD-splines, we offer a recursive method to evaluate the representation matrix between the MD-spline basis and Bernstein basis, and design some numerical experiments for it. This method utilizes the integral property of Bernstein polynomials and possesses sufficient numerical accuracy and efficiency.

The remainder of this paper is organized as follows: Section 2 reviews the necessary definitions of MD-splines. Section 3 provides a recursive method by Bernstein representation and contains some numerical experiments. Section 4 reveals general recursive relations of the MD-spline basis and constructs de Boor–Cox-type formula. Finally, this paper is concluded with some conclusion.

2. MD-Spline Basis Review

In the following, we will recall some concepts of MD-splines given in [9,14]. MD-spline basis functions are piecewise polynomials defined on a partition of an interval, which is determined by a breakpoint sequence $\boldsymbol{T}$, a degree sequence $\boldsymbol{G}$, and a sequence $\boldsymbol{K}$ containing the orders of regularity.

Let $\boldsymbol{T} := \{t_i\}_{i=0}^{q+1}$, $a = t_0 < t_1 < \ldots < t_q < t_{q+1} = b$ be a strictly increasing partition of a given interval $\boldsymbol{I} := [a, b]$. If $d_i > 0, i = 0, \ldots, q$ is the degree of the MD-spline on $[t_i, t_{i+1})$, then let $\boldsymbol{G} := \{d_i\}_{i=0}^{q}$ be the degree sequence and $D = \max\{d_i\}$, $0 \leqslant i \leqslant q$. Finally, the entries of the order sequence $\boldsymbol{K} := \{k_i\}_{i=1}^{q}$, which describe the required continuity at breakpoints t_i, $i = 1, 2, \ldots, q$, are defined as follows:

$$k_i \leqslant \begin{cases} d_{i-1}, & if\ d_{i-1} = d_i, \\ \min\{d_i, d_{i-1}\}, & if\ d_{i-1} \neq d_i. \end{cases} \tag{3}$$

There are two extended partitions of $\boldsymbol{T}$ to determine the support intervals. The left extended partition is

$$\boldsymbol{S} := \{s_1, s_2, \ldots, s_{\mathcal{K}}\} := \{\underbrace{t_0, \ldots, t_0}_{d_0+1 \text{ times}}, \underbrace{t_1, \ldots, t_1}_{d_1-k_1 \text{ times}}, \ldots, \underbrace{t_q, \ldots, t_q}_{d_q-k_q \text{ times}}\}, \tag{4}$$

and the right one is

$$\boldsymbol{X} := \{x_1, x_2, \ldots, x_{\mathcal{K}}\} := \{\underbrace{t_1, \ldots, t_1}_{d_0-k_1 \text{ times}}, \ldots, \underbrace{t_q, \ldots, t_q}_{d_{q-1}-k_q \text{ times}}, \underbrace{t_{q+1}, \ldots, t_{q+1}}_{d_q+1 \text{ times}}\}, \tag{5}$$

where $\mathcal{K} := d_0 + \sum_{i=1}^{q}(d_i - k_i) + 1 = d_q + \sum_{i=1}^{q}(d_{i-1} - k_i) + 1$.

Figure 1 shows an example for illustrating the left and right extended partitions. Here, $\boldsymbol{T} := \{t_i\}_{i=0}^{5}$, $\boldsymbol{G} := \{3, 3, 4, 5, 5\}$, $\boldsymbol{K} := \{2, 2, 3, 4\}$, and $\mathcal{K} = 10$.

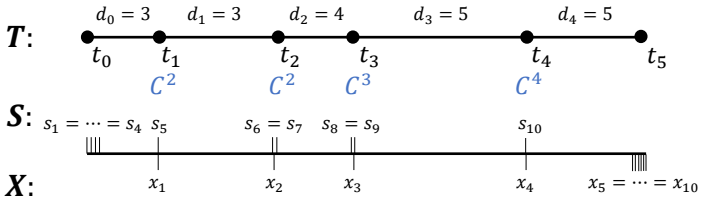

Figure 1. Original partition $\boldsymbol{T}$, left and right extended partitions $\boldsymbol{S}$ and $\boldsymbol{X}$, respectively.

The relation of indexes is noted by function p. In detail, $p(s_i)$ is the index of the corresponding breakpoint of s_i in $\boldsymbol{T}$ and $p(x_i)$ is the index of the corresponding breakpoint x_i in $\boldsymbol{T}$. That is, $t_{p(s_i)} = s_i$, and $t_{p(x_i)} = x_i$. For example, in Figure 1, since $s_6 = s_7 = t_2$, $p(s_6) = p(s_7) = 2$. Since $x_3 = t_3$, $p(x_3) = 3$.

Definition 1 (Multi-degree spline basis). *For $n = 0, 1, ..., D$, functions in $\{N_{i,n}(t)\}_{i=D-n+1}^{\mathcal{K}}$ are generated recursively over $\boldsymbol{T}$,$\boldsymbol{G}$ and $\boldsymbol{K}$. The final sequence $\{N_{i,D}(t)\}_{i=1}^{\mathcal{K}}$ is a MD-spline basis, given by*

$$N_{i,n}(t) = \begin{cases} 0, & d_j < D - n, \\ \begin{cases} 1 & t \in [t_j, t_{j+1}) \\ 0 & otherwise \end{cases}, & d_j = D - n, \\ \int_{-\infty}^{t}(\delta_{i,n-1}N_{i,n-1}(y) - \delta_{i+1,n-1}N_{i+1,n-1}(y))dy, & d_j > D - n. \end{cases} \tag{6}$$

for each nontrivial interval $[t_j, t_{j+1}) \subseteq [s_i, x_{i-D+n}) = [t_{p(s_i)}, t_{p(x_{i-D+n})})$, *where*

$$\delta_{i,n} := \left(\int_{-\infty}^{+\infty} N_{i,n}(y)dy \right)^{-1}$$

In addition, when $N_{i,n} = 0$, *we set*

$$\int_{-\infty}^{t} \delta_{i,n} N_{i,n}(y)dy = \begin{cases} 0 & t < s_i, \\ 1 & t \geq s_i. \end{cases}$$

MD-spline basis $\{N_{i,D}(t)\}_{i=1}^{\mathcal{K}}$ possesses some properties such as a B-spline basis:

- Local support: $N_{i,D}(t) = 0$ for $t \notin [s_i, x_i]$.
- Positivity: $N_{i,D}(t) > 0$ for $t \in (s_i, x_i)$.
- End point: $N_{i,D}(t)$ vanishes exactly $d_{p(s_i)} - max\{j \geq 0 | s_i = s_{i+j}\}$ times at s_i, and $d_{p(x_i)-1} - max\{j \geq 0 | x_{i-j} = x_i\}$ at x_i.
- Partition of unity: $\sum_i^{\mathcal{K}} N_{i,D}(t) \equiv 1$ for all $t \in [a, b]$.

Some properties of functions $N_{i,n}$, $i = D - n + 1, D - n + 2, \ldots, \mathcal{K}$ for any $0 \leqslant n \leqslant D$ are as follows:

- Support interval: the support interval of $N_{i,n}(t)$ is $[s_i, x_{i-D+n})$.
- Degree: on interval $[t_j, t_{j+1}) \subseteq [s_i, x_{i-D+n})$ with $s_i < x_{i-D+n}$, the degree of $N_{i,n}(t)$ equals to $d_j + n - D$ when $n \geq D - d_j$.
- Continuity: at knot t_j, $t_j \in (s_i, x_{i-D+n})$ with $s_i < x_{i-D+n}$, $N_{i,n}(t)$ is C^{k_j+n-D}-continuous when $n \geq D - k_j$.

In [24], C^k MD-splines are defined to describe some particular MD-spline bases, such as C^0, C^1 MD-splines.

Definition 2 (C^k multi-degree spline basis)**.** *An MD-spline basis is said to be a* C^k *MD-spline basis if its functions are, at most,* C^k *continuous at the joins between pieces of different degrees.*

In the example of Figure 1, as $d_0 = d_1$, $d_3 = d_4$, we do not need to consider the continuity at t_1 and t_4. Therefore, functions in this basis will be C^2-continuous at the join between $[t_0, t_2)$ and $[t_2, t_3)$, and be C^3-continuous at the join between $[t_2, t_3)$ and $[t_3, t_5)$. Therefore, the corresponding MD-spline basis will be a C^3 MD-spline basis.

3. Recursive Method

As Bernstein polynomials of degree n form a basis of the space $\mathcal{P}_n$ of polynomials of degree at most n, functions in $\{N_{i,n}(t)\}_{i=D-n+1}^{\mathcal{K}}$ can be represented by Bernstein polynomials with different degrees. Thus, the recursive process for defining an MD-spline basis can be performed from these Bernstein polynomials.

3.1. Bernstein Representation

Bernstein polynomials [26] of degree n on interval $[t_j, t_{j+1}]$ are defined as

$$b_{i,n}^j(t) := \frac{\binom{n}{i} (t - t_j)^i (t_{j+1} - t)^{n-i}}{(t_{j+1} - t_j)^n}, \; i = 0, 1, \ldots, n. \tag{7}$$

Let

$$\boldsymbol{B}_n^j(t) := \left(b_{0,n}^j(t), b_{1.n}^j(t), \ldots, b_{n,n}^j(t) \right)^T. \tag{8}$$

Then, each function $N_{i,n}$, $i = D-n+1, D-n+2, \ldots, \mathcal{K}$ and $n = 0, 1, \ldots, D$ can be linearly represented as

$$N_{i,n}(t) = a_{i,n}^{0,j} b_{0,n}^{j}(t) + a_{i,n}^{1,j} b_{1,n}^{j}(t) + \ldots + a_{i,n}^{n,j} b_{n,n}^{j}(t), \tag{9}$$

where $a_{i,n}^{0,j}, a_{i,n}^{1,j}, \ldots, a_{i,n}^{n,j}$ are some constants that are not all zero, and this equation can also be written as

$$N_{i,n}(t) = \boldsymbol{A}_{i,n}^{j} \boldsymbol{B}_{d_j+n-D}^{j}(t),\ t \in [t_j, t_{j+1}),\ j = p(s_i), p(s_i)+1, \ldots, p(x_{i+D-n}) - 1, \tag{10}$$

where $\boldsymbol{A}_{i,n}^{j} = \left(a_{i,n}^{0,j}, a_{i,n}^{1,j}, \ldots, a_{i,n}^{n,j}\right)$ is called the Bernstein representation vector of function $N_{i,n}$ on $[t_j, t_{j+1})$.

On the support interval of $N_{i,n}$, consistent with the previous settings, the number of elements in $\boldsymbol{T}$ is $q+2$. For each n, $\mathbb{B}_n$ stands for the whole set of Bernstein polynomials:

$$\mathbb{B}_n := \begin{pmatrix} \boldsymbol{B}_{d_1+n-D}^{1}(t) \\ \boldsymbol{B}_{d_2+n-D}^{2}(t) \\ \vdots \\ \boldsymbol{B}_{d_{l-1}+n-D}^{q+1}(t) \end{pmatrix}. \tag{11}$$

Let

$$\vec{\mathcal{A}}_{i,n} := \left(\boldsymbol{0}, \ldots, \boldsymbol{0}, \boldsymbol{A}_{i,n}^{p(s_i)}, \boldsymbol{A}_{i,n}^{p(s_i)+1}, \ldots, \boldsymbol{A}_{i,n}^{p(x_{i-D+n})-1}, \boldsymbol{0}, \ldots, \boldsymbol{0}\right). \tag{12}$$

Then

$$N_{i,n}(t) = \vec{\mathcal{A}}_{i,n} \mathbb{B}_n, \tag{13}$$

and the corresponding representation matrix is

$$\boldsymbol{N} = \boldsymbol{M} \mathbb{B}_n, \tag{14}$$

where

$$\begin{aligned} \boldsymbol{N} &:= \left(N_{D-n+1,n}(t), N_{D-n+2,n}(t), \ldots, N_{\mathcal{K},n}(t)\right)^T, \\ \boldsymbol{M} &:= \begin{pmatrix} \vec{\mathcal{A}}_{D-n+1,n} \\ \vdots \\ \vec{\mathcal{A}}_{\mathcal{K},n} \end{pmatrix}_{(\mathcal{K}-D+n) \times \sum_{j=1}^{q+1}(d_j+n-D+1)} \end{aligned} \tag{15}$$

Compared with the representation matrix in [19], matrix $\boldsymbol{M}$ represents functions in $\{N_{i,n}\}_{i=D-n+1}^{\mathcal{K}}$ at each recursive level n.

Definition 3 (C^k function)**.** *A function defined over $\boldsymbol{T}$, $\boldsymbol{G}$ and $\boldsymbol{K}$ is called a C^k function if it is at most C^k continuous at breakpoints within its support interval.*

C^k functions are not equivalent to functions in a C^k MD-spline basis, a C^k MD-spline basis only requires continuity at the join of different degrees. For example, a C^0 MD-spline basis may contain B-splines; however, B-splines are not C^0 continuous at their internal breakpoints. A C^0 MD-spline basis has some particular properties, such as the integrals of functions in it can be computed easily, which is used in [20]. C^0 functions are also particularly to be used in this section, as their Bernstein presentation vectors are easily expressed.

Example 1. *Consider a C^0 MD-spline basis with $\boldsymbol{T} := \{0,1,2,3,4,5,6,7\}$, $\boldsymbol{G} := \{3,3,3,4,4,4,4\}$ and $\boldsymbol{K} := \{2,2,0,3,3,3\}$. The first function $N_{1,4}$ and the last function $N_{13,4}$ are Bernstein polynomials. Function $N_{6,4}$ is C^0 continuous at $t = 3$, and it is plotted in red in Figure 2. Other*

functions are all B-spline functions, where $N_{i,4}, i = 2,3,4,5$ *are* C^2 *functions and* $N_{i,4}, i = 7,...,12$ *are* C^3 *functions. In addition, all functions* $N_{i,4}, i = 1,...,13$ *define a* C^0 *MD-spline basis. It holds*

$$N_{6,4}(t) = \Big((0,0,0,1),(1,0,0,0,0)\Big) \begin{pmatrix} b^2_{0,3}(t) \\ \vdots \\ b^2_{3,3}(t) \\ b^3_{0,4}(t) \\ \vdots \\ b^3_{4,4}(t) \end{pmatrix}. \tag{16}$$

$N_{6,4}$ is a C^0 function, and its Bernstein representation vector on each interval is composed of an entry equal to 1 and the rest are equal to 0, while the Bernstein representation vectors of B-spline functions include several nonzero entries.

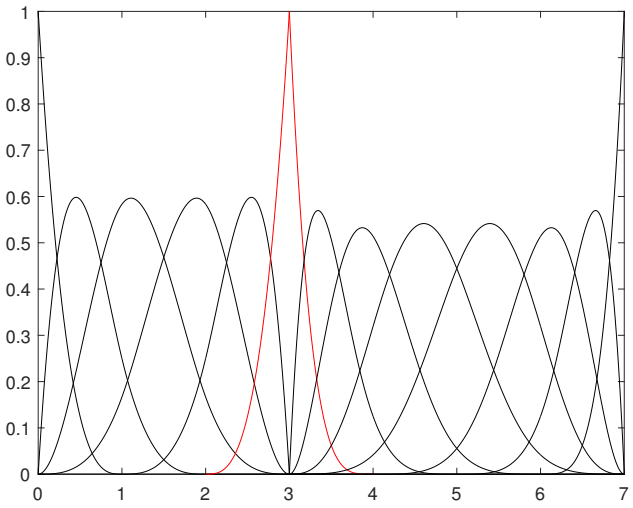

Figure 2. An example of C^0 multi-degree spline basis functions.

3.2. Recursive Process

The recursion of Bernstein representation vectors is determined by Equation (6) and the integral property of Bernstein polynomial [26]. Each vector $\boldsymbol{A}^j_{i,n}$ is affected not only by $\boldsymbol{A}^j_{r,n-1}$, $r = i, i+1$, but also by $\boldsymbol{A}^r_{i,n}$, $r = p(s_i), \ldots, j-1$ and $\boldsymbol{A}^r_{i,n}, r = j+1, \ldots, p(x_{i-D+n}) - 1$; hence, the recursive relationship should be evaluated on distinct intervals.

On interval $[t_j, t_{j+1}) \subseteq [t_{p(s_i)}, t_{p(x_{i+D-n})})$, consider the following $(d_j + n - D + 1) \times (d_j + n - D + 2)$ matrix

$$\boldsymbol{H} := \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{17}$$

The integral of Bernstein polynomials satisfy the equality

$$\begin{pmatrix} \int_{t_j}^t b^j_{0,d_j+n-D}(y)dy \\ \vdots \\ \int_{t_j}^t b^j_{d_j+n-D,d_j+n-D}(y)dy \end{pmatrix} = \frac{t_{j+1} - t_j}{d_j + n - D + 1} \boldsymbol{H} \boldsymbol{B}^j_{d_j+n-D+1}(t), \tag{18}$$

According to the integral formula, the integral of $N_{i,n}$ is determined as follows:

$$\int_{t_j}^{t} N_{i,n}(y)dy = \boldsymbol{A}_{i,n}^{j} \frac{t_{j+1} - t_j}{d_j + n - D + 1} \boldsymbol{H} \boldsymbol{B}_{d_j+n-D+1}^{j}(t). \tag{19}$$

Similarly, another part of the integral is

$$\int_{t}^{t_{j+1}} N_{i,n}(y)dy = \boldsymbol{A}_{i,n}^{j} \frac{t_{j+1} - t_j}{d_j + n - D + 1} \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \boldsymbol{B}_{d_j+n-D+1}^{j}(t). \tag{20}$$

The integral recursion can be substituted by the recursive process of Bernstein representation vectors according to Equation (19). The matrix $\boldsymbol{H}$ is just presented in (18) for simplicity of expression. A cumulative-sum operation can be used to describe the recursion of a representation vector in calculation.

Without loss of generality, assuming that $\boldsymbol{A}_{i,n}^{j} = (a_0, a_1, \ldots, a_{d_j+n-D})$ on interval $[t_j, t_{j+1})$, $j = p(s_i), \ldots, p(x_{i+n-D}) - 1$, it holds that

$$N_{i,n}(t) = (a_0, a_1, \ldots, a_{d_j+n-D}) \boldsymbol{B}_{d_j+n-D}^{j}(t). \tag{21}$$

We indicates the cumulative-sum operation by an operator Γ:

$$\Gamma(\boldsymbol{A}_{i,n}^{j}) := \frac{t_{j+1} - t_j}{d_j + n - D + 1} (0,\ a_0,\ a_0 + a_1,\ \ldots,\ a_0 + a_1 + \ldots + a_{d_j+n-D}). \tag{22}$$

Thus,

$$\int_{t_j}^{t} N_{i,n}(y)dy = \Gamma(\boldsymbol{A}_{i,n}^{j}) \boldsymbol{B}_{d_j+n-D+1}^{j}(t). \tag{23}$$

Then, we use an operator Λ to extract the last element of a vector. For example, $\Lambda(\boldsymbol{A}_{i,n}^{j}) = a_{d_j+n-D}$. Then,

$$\int_{t_j}^{t_{j+1}} N_{i,n}(y)dy = \Lambda\Big(\Gamma(\boldsymbol{A}_{i,n}^{j})\Big). \tag{24}$$

Consider $t \in [t_j, t_{j+1})$, $j = p(s_i) + 1, ..., p(x_{i+n-D}) - 1$. The integral starting from $t_{p(s_i)}$ is

$$\begin{aligned} \int_{t_{p(s_i)}}^{t} N_{i,n}(y)dy &= \sum_{r=p(s_i)}^{j-1} \left[\left(\Lambda\left(\Gamma(\boldsymbol{A}_{i,n}^{r})\right)\right)\right] + \Gamma(\boldsymbol{A}_{i,n}^{j}) \boldsymbol{B}_{d_j+n-D+1}^{j}(t), \\ &= \left[\sum_{r=p(s_i)}^{j-1} \left[\left(\Lambda\left(\Gamma(\boldsymbol{A}_{i,n}^{r})\right)\right)\right] \oplus \Gamma(\boldsymbol{A}_{i,n}^{j})\right] \boldsymbol{B}_{d_j+n-D+1}^{j}(t), \end{aligned} \tag{25}$$

where $\sum_{r=p(s_i)}^{j-1} \left[\left(\Lambda\left(\Gamma(\boldsymbol{A}_{i,n}^{r})\right)\right)\right]$ is a constant, and $\Gamma(\boldsymbol{A}_{i,n}^{j})$ is a vector. The symbol $\oplus$ means that $\sum_{r=p(s_i)}^{j-1} \left[\left(\Lambda\left(\Gamma(\boldsymbol{A}_{i,n}^{r})\right)\right)\right]$ is added to each element in $\Gamma(\boldsymbol{A}_{i,n}^{j})$.

Using the two operators above, $\delta_{i,n}$ can be calculated as follows:

$$\frac{1}{\delta_{i,n}} = \int_{-\infty}^{+\infty} N_{i,n}(y)dy = \sum_{r=p(s_i)}^{p(x_{i-D+n})-1} \left[\Lambda\left(\Gamma(\boldsymbol{A}_{i,n}^{r})\right)\right]. \tag{26}$$

Consistent with the conventional B-splines, functions in an MD-spline basis can be independently calculated using a recursive method. According to the Bernstein repre-

sentation, once the vector $\vec{\mathcal{A}}_{i,D}$ is calculated, function $N_{i,D}$ is determined. Consequently, Bernstein representation vectors can replace functions in the recursive process.

The support interval of $N_{i,D}$, $i = 1, \ldots, \mathcal{K}$ is $[s_i, x_i)$; therefore, there are $p(x_i) - p(s_i)$ intervals in it. Therefore, we establish an initial $(D+1) \times (p(x_i) - p(s_i))$ matrix $\boldsymbol{M}$; each entry is 0 and indicates a vector $\boldsymbol{A}^j_{k,0} = 0$, $k = i, \ldots, i+D$, $j = p(s_i), \ldots, p(x_i) - 1$.

Matrix $\boldsymbol{M}$ has $D+1$ rows, which means that there are $D+1$ corresponding functions on $[s_i, x_i)$ when $n = 0$. However, the indexes of these $D+1$ functions may exceed the limit range $[D+1, \mathcal{K}]$. There are two solutions for this. One does not generate rows whose indexes surpass the maximum allowed range. The other one disregards these rows because all of their elements equal 0. In order to make the algorithm more understandable, we employ the second solution . In the recursive process, we use these vectors $\boldsymbol{A}^j_{k,n}, n = 0, ..., D$ to explain the recursion of matrix $\boldsymbol{M}$ in the algorithm.

There are some details in this procedure.

- The operation (line 12) causes vectors $\boldsymbol{A}^j_{i,n}$ with the same index j have the same length, hence allowing these vectors can bulid the matrix $\boldsymbol{M}$.
- $\boldsymbol{M}_k$ (line 15) indicates the $k-th$ row of the matrix $\boldsymbol{M}$, and $(\delta_{k,n})^{-1} = \boldsymbol{A}^{p(x_{k+n-D})-1}_{k,n}$. Thus each row is normalized.
- There are only two cases of subtraction without considering the subtraction of zero rows (line 19 and 21).
- The subtraction is performed in range (lines 20, 22 and 23).

Algorithm 1 only calculates one function in MD-spline basis. If vectors in the initial matrix are changed to be $\boldsymbol{A}^j_{k,0}$, $k = 1, \ldots, \mathcal{K}+D$, $j = p(s_1), \ldots, p(x_{\mathcal{K}}) - 1$, all basis functions are produced.

The following is an example for Algorithm 1.

Example 2. *Consider a MD-spline basis with* $\boldsymbol{T} := \{0,1,4,7,10\}$, $\boldsymbol{G} := \{2,3,4,3\}$, *and* $\boldsymbol{K} := \{1,2,3\}$. *Two extended partition are* $\boldsymbol{S} := \{0,0,0,1,1,4,4\}$ *and* $\boldsymbol{X} := \{1,4,7,10,10,10,10\}$. *We are aming to calculate function* $N_{3,4}$. *These partitions are shown in Figure 3.*

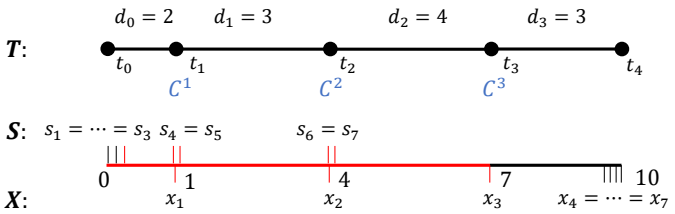

Figure 3. The setting of Example 2, and the support interval of $N_{3,4}$ is plotted in red .

The support interval of $N_{3,4}$ *is* $[s_3, x_3) = [0,7)$, *and* $p(s_3) = 0, p(x_3) = 3$. *Thus the order of the initial matrix* $\boldsymbol{M}$ *is* 5×3, *then*

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A}^0_{3,0} & \boldsymbol{A}^1_{3,0} & \boldsymbol{A}^2_{3,0} \\ \boldsymbol{A}^0_{4,0} & \boldsymbol{A}^1_{4,0} & \boldsymbol{A}^2_{4,0} \\ \boldsymbol{A}^0_{5,0} & \boldsymbol{A}^1_{5,0} & \boldsymbol{A}^2_{5,0} \\ \boldsymbol{A}^0_{6,0} & \boldsymbol{A}^1_{6,0} & \boldsymbol{A}^2_{6,0} \\ \boldsymbol{A}^0_{7,0} & \boldsymbol{A}^1_{7,0} & \boldsymbol{A}^2_{7,0} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{27}$$

At the beginning of each loop, value 1 is assigned to the vector that satisfies the condition in line 5 in Algorithm 1. Therefore, $\boldsymbol{A}^2_{7,0} = 1$.

Algorithm 1 Recursive process of Bernstein representation vectors.

Input: $\boldsymbol{T}$, $\boldsymbol{G}$, $\boldsymbol{K}$, i
Output: $\vec{\mathcal{A}}_{i,D}$

Initial $\boldsymbol{M} = \left(\boldsymbol{A}_{k,0}^{j}\right)$, $\boldsymbol{A}_{k,0}^{j} = 0, k = i, ..., i+D, j = p(s_i), ..., p(x_i) - 1$;
for n$\leftarrow$ 0 to $D-1$ **do**
  for k$\leftarrow$ i to $i+D-n$ **do**
    for j$\leftarrow$ $p(s_k)$ to $p(x_{k+n-D}) - 1$ **do**
      if $(d_j == D-n)$ **then** $\boldsymbol{A}_{k,n}^{j} \leftarrow 1$
      end if
    end for
    for j $\leftarrow$ $p(s_k)$ to $p(x_{k+n-D}) - 1$ **do**
      $\boldsymbol{A}_{k,n}^{j} \leftarrow \sum_{k=p(s_i)}^{j-1} \left(\Lambda\left(\Gamma(\boldsymbol{A}_{k,n}^{r})\right)\right) \oplus \Gamma(\boldsymbol{A}_{k,n}^{j})$
    end for
    for j $\leftarrow$ $p(s_i)$ to $p(x_i) - 1$ **do**
      if $(d_j >= 0)$ and $(\Lambda(\boldsymbol{A}_{k,n}^{j} == 0))$ **then** $\boldsymbol{A}_{k,n}^{j} \leftarrow \Gamma(\boldsymbol{A}_{k,n}^{j})$
      end if
    end for
    if $\Lambda(\boldsymbol{A}_{k,n}^{p(x_{k+n-D})-1}) \neq 0$ **then** $\boldsymbol{M}_k \leftarrow \left(\Lambda(\boldsymbol{A}_{k,n}^{p(x_{k+n-D})-1})\right)^{-1} \boldsymbol{M}_k$
    end if
  end for
  for k$\leftarrow$ i to $i+D-n-1$ **do**
    if $\Lambda(\boldsymbol{A}_{k,n}^{p(x_{k+n-D})-1}) == 0$ and $\Lambda(\boldsymbol{A}_{k+1,n}^{p(x_{k+1+n-D})-1}) \neq 0$ **then**
      $\boldsymbol{M}_k \leftarrow 1 - \boldsymbol{M}_{k+1}\ (p(s_k), p(x_{k+1+n-D}) - 1)$
    else if $\Lambda(\boldsymbol{A}_{k,n}^{p(x_{k+n-D})-1}) \neq 0$ and $\Lambda(\boldsymbol{A}_{k+1,n}^{p(x_{k+1+n-D})-1}) \neq 0$ **then**
      $\boldsymbol{M}_k \leftarrow \boldsymbol{M}_k - \boldsymbol{M}_{k+1}\ (p(s_k), p(x_{k+n-D}) - 1)$
      $\boldsymbol{M}_k \leftarrow 1 - \boldsymbol{M}_{k+1}\ (p(x_{k+n-D}), p(x_{k+1+n-D}) - 1)$
    end if
  end for
  Delete the last row of $\boldsymbol{M}$;
end for

Because there is no C^0 *function when* $n = 0, 1$. *In order to be concise, we start the calculation from* $n = 2$. *In this case, it is easy to get* $\boldsymbol{M}$ *:*

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A}_{3,2}^{0} & \boldsymbol{A}_{3,2}^{1} & \boldsymbol{A}_{3,2}^{2} \\ \boldsymbol{A}_{4,2}^{0} & \boldsymbol{A}_{4,2}^{1} & \boldsymbol{A}_{4,2}^{2} \\ \boldsymbol{A}_{5,2}^{0} & \boldsymbol{A}_{5,2}^{1} & \boldsymbol{A}_{5,2}^{2} \end{pmatrix} = \begin{pmatrix} 1 & (0,0) & (0,0,0) \\ 0 & (1,0) & (0,0,0) \\ 0 & (0,1) & (1,0,0) \end{pmatrix}. \tag{28}$$

In the third row of $\boldsymbol{M}$, *the results of cumulative sum and normalization operations are as follows:*

$$\Gamma(\boldsymbol{A}_{5,2}^{1}) = (0, 0, \frac{3}{2});\ \ \Lambda(\Gamma(\boldsymbol{A}_{5,2}^{1})) = \frac{3}{2}, \tag{29}$$

and

$$\Gamma(\boldsymbol{A}_{5,2}^{2}) = (0, 1, 1, 1);\ \ \Lambda(\Gamma(\boldsymbol{A}_{5,2}^{1})) \oplus \Gamma(\boldsymbol{A}_{5,2}^{2}) = (\frac{3}{2}, \frac{5}{2}, \frac{5}{2}, \frac{5}{2}). \tag{30}$$

Thus $\delta_{5,2} = \frac{2}{5}$. *Note that* $\Gamma(\boldsymbol{A}_{5,2}^{0}) = (0, 0)$. *After normalization, this row changes into*

$$\boldsymbol{M}_3 = \begin{pmatrix} (0,0) & (0,0,\frac{3}{5}) & (\frac{3}{5},1,1,1) \end{pmatrix}. \tag{31}$$

Similarly, the second row changes into

$$\boldsymbol{M}_2 = \begin{pmatrix} (0,0) & (0,1,1) & (0,0,0,0) \end{pmatrix}. \tag{32}$$

According to the substraction in Algorithm 1, $\boldsymbol{M}_2 \leftarrow \boldsymbol{M}_2 - \boldsymbol{M}_3$ *equals to* $(0,1,1) - (0,0,\frac{3}{5})$ *on interval* $[t_1, t_2)$ *and* $(1,1,1,1) - (\frac{3}{5},1,1,1)$ *on interval* $[t_2, t_3)$*. After substraction*

$$\boldsymbol{M}_2 = \begin{pmatrix}(0,0) & (0,1,\frac{2}{5}) & (\frac{2}{5},0,0,0)\end{pmatrix}. \tag{33}$$

$\boldsymbol{M}_1 \leftarrow \boldsymbol{M}_1 - \boldsymbol{M}_2$ *can be computed in the same manner. Then, remove the third row and* $\boldsymbol{M}$ *change into the subsequent one:*

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A}^0_{3,3} & \boldsymbol{A}^1_{3,3} & \boldsymbol{A}^2_{3,3} \\ \boldsymbol{A}^0_{4,3} & \boldsymbol{A}^1_{4,3} & \boldsymbol{A}^2_{4,3} \end{pmatrix} = \begin{pmatrix} (0,1) & (1,0,0) & (0,0,0,0) \\ (0,0) & (0,1,\frac{2}{5}) & (\frac{2}{5},0,0,0) \end{pmatrix}. \tag{34}$$

By repeating these operations, then the final vector will be

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A}^0_{3,4} & \boldsymbol{A}^1_{3,4} & \boldsymbol{A}^2_{3,4} \end{pmatrix} = \begin{pmatrix}(0,0,\frac{1}{3}) & (\frac{1}{3},1,\frac{7}{17},\frac{3}{17}) & (\frac{3}{17},0,0,0,0)\end{pmatrix}. \tag{35}$$

Therefore,

$$N_{3,4}(t) = \vec{\mathcal{A}}_{3,4}\mathbb{B}_4 = \begin{pmatrix} \boldsymbol{A}^0_{3,4} & \boldsymbol{A}^1_{3,4} & \boldsymbol{A}^2_{3,4} & \boldsymbol{0} \end{pmatrix}\mathbb{B}_4 = \boldsymbol{M}\begin{pmatrix} \boldsymbol{B}^0_2(t) \\ \boldsymbol{B}^1_3(t) \\ \boldsymbol{B}^2_4(t) \end{pmatrix}. \tag{36}$$

This example can aid comprehension of this section. Although only a portion of the calculation is displayed, the entire recursion is similar. The representation matrix may be calculated directly, and this method is quite intuitive and straightforward to comprehend.

3.3. Algorithm Analysis

The numerical stability of Algorithm 1 is influenced by the floating point operations. There are three types of operations in the algorithm.

1. Cumulative sum operation: the operation in Algorithm 1 (line 9) is composed of addition and multiplication. All elements in the vector are positive and increasing. If the length of an interval is excessively long, the number of bits of elements may vary significantly. This procedure frequently executes this operation, resulting in numerical inaccuracies.
2. Normalized operation: this operation restricts all elements in $\vec{\mathcal{A}}_{i,n}$ to the range $[0,1]$. Since the denominator is the largest number in this vector and all members are positive, this operation will not result in a significant amount of numerical inaccuracy.
3. Subtraction operation: Equation (19) prevents the scenario where two approximate floating point-numbers are subtracted.

The complexity of Algorithm 1 depends on the settings of the sequences $\boldsymbol{T}, \boldsymbol{G}, \boldsymbol{K}$, so, like reference [20], we only count the number of operations.

At each recursive level in Algorithm 1, there are no more than $\mathcal{K} - D + n$ non-zero rows of each $\boldsymbol{M}$. Therefore, the total number of non-zero rows will not exceed $\mathcal{K}D$ in the whole recursive process.

- Cumulative sum operation: the number of elements in $\boldsymbol{G}$ is $q+1$; thus, the number of this operation will not exceed $\mathcal{K}D(q+1)$.
- Normalized operation: each non-zero row will be normalized, so the maximum number of this operation is $\mathcal{K}D(q+1)$.
- Subtraction operation: similarly, there are $\mathcal{K}D$ subtraction operations, at most.

Thus, the number of operations will not exceed $\mathcal{K}D(2q+3)$, which is used to estimate the numerical complexity of Algorithm 1.

Due to the recursive nature of Algorithm 1, the cyclical calculation of floating-point numbers may lead to error accumulation. Based on the preceding research, the cumulative sum operation is the primary source of mistake. This error can be reduced by a bidirectional algorithm, and Equation (6) is reformed as follows:

$$N_{i,n}(t) = \begin{cases} \delta_{i,n-1}\int_{-\infty}^{t} N_{i,n-1}(s)ds, & t \in [s_i, s_{i+1}), \\ \delta_{i,n-1}\int_{-\infty}^{t} N_{i,n-1}(s)ds - \delta_{i+1,n-1}\int_{-\infty}^{t} N_{i+1,n-1}(s)ds, & t \in [s_{i+1}, \xi), \\ \delta_{i+1,n-1}\int_{t}^{+\infty} N_{i+1,n-1}(s)ds - \delta_{i,n-1}\int_{t}^{+\infty} N_{i,n-1}(s)ds, & t \in [\xi, x_{i+n-D-1}), \\ \delta_{i+1,n-1}\int_{t}^{+\infty} N_{i+1,n-1}(s)ds, & t \in [x_{i+n-D-1}, x_{i+n-D}), \end{cases} \tag{37}$$

where $\xi := s_{i+1} + \lfloor (x_{i+n-D-1} - s_{i+1})/2 \rfloor$. The symbol $\lfloor\rfloor$ means rounding down operation.

Then, Algorithm 1 is modified, and the subsequent Algorithm 2 is more numerically stable in calculation.

To represent the reverse integral, we define two reverse operators. Consistent with the prior setting $\boldsymbol{A}^j_{i,n} = (a_0, a_1, \ldots, a_{d_j+n-D})$, then $\overline{\Gamma}(\boldsymbol{A}^j_{i,n}) = (a_0 + \ldots + a_{d_j+n-D}, a_1 + \ldots + a_{d_j+n-D}, \ldots, a_{d_j+n-D}, 0)$ and $\overline{\Lambda}(\boldsymbol{A}^j_{i,n}) = a_0$. Algorithm 2 is as follows:

Algorithm 2 Revised recursive method.

Same setting as Algorithm 1;
for n← 0 to $D-1$ **do**
  for k← i to $i+D-n$ **do**
    for j← $p(s_k)$ to $p(x_{k+n-D})-1$ **do**
      if $(d_j == D-n)$ **then** $\boldsymbol{A}^j_{k,n} \leftarrow 1$
      end if
    end for
  end for
  $\boldsymbol{M}^L \leftarrow \boldsymbol{M}, \boldsymbol{M}^R \leftarrow \boldsymbol{M}$;
  for j ← $p(s_k)$ to $p(x_{k+n-D})-1$ **do**
    $\boldsymbol{M}^L_{k,j} \leftarrow \sum_{r=p(s_i)}^{j-1}\left(\Lambda\left(\Gamma(\boldsymbol{A}^r_{k,n})\right)\right) + \Gamma(\boldsymbol{A}^j_{k,n})$;
    $\boldsymbol{M}^R_{k,j} \leftarrow \sum_{r=j+1}^{p(x_{k-D+n})-1}\left(\overline{\Lambda}\left(\overline{\Gamma}(\boldsymbol{A}^r_{k,n})\right)\right) + \overline{\Gamma}(\boldsymbol{A}^j_{k,n})$;
  end for
  Extend zero vectors;
  if $\Lambda(\boldsymbol{M}^L_{k,p(x_{k+n-D})-1}) \neq 0$ **then** $\boldsymbol{M}^L_k \leftarrow \left(\Lambda(\boldsymbol{M}^L_{k,p(x_{k+n-D})-1})\right)^{-1}\boldsymbol{M}^L_k$
  end if
  if $\overline{\Lambda}(\boldsymbol{M}^R_{k,p(s_k)}) \neq 0$ **then** $\boldsymbol{M}^R_k \leftarrow \left(\overline{\Lambda}(\boldsymbol{M}^R_{k,p(s_k)})\right)^{-1}\boldsymbol{M}^R_k$
  end if
  for k← i to $i+D-n-1$ **do**
    for j← $p(s_k)$ to $p(s_{k+1})$ **do**
      $\boldsymbol{M}_{k,j} \leftarrow \boldsymbol{M}^L_{k,j}$;
    end for
    for j← $p(s_{k+1})$ to $p(s_{k+1}) + \lfloor (p(x_{k+n-D}) - 1 - p(s_{k+1}))/2 \rfloor$ **do**
      $\boldsymbol{M}_{k,j} \leftarrow \boldsymbol{M}^L_{k,j} - \boldsymbol{M}^L_{k+1,j}$;
    end for
    for j← $p(s_{k+1}) + \lfloor (p(x_{k+n-D}) - 1 - p(s_{k+1}))/2 \rfloor + 1$ to $p(x_{k+n-D})-1$ **do**
      $\boldsymbol{M}_{k,j} \leftarrow \boldsymbol{M}^R_{k+1,j} - \boldsymbol{M}^R_{k,j}$;
    end for
    for j← $p(x_{k+n-D})-1$ to $p(x_{k+n-D+1})-1$ **do**
      $\boldsymbol{M}_{k,j} = \boldsymbol{M}^R_{k+1,j}$
    end for
  end for
  Delete the last row of $\boldsymbol{M}$;
end for

3.4. Numerical Experiment

In [20], three typical methods for calculating an MD-spline basis are compared, which are named RKI/Greville, RKI/Derivative and H-Operator. Due to the higher order deriva-

tive operations, RKI/Derivative and H-Operator are not numerically stable, while RKI/Greville avoids these operations and it is numerically stable.

Then, we will design some experiments in Table 1 to test Algorithms 1 and 2. Calculation results are divided into 'Exact' and 'Numerical'. The 'Exact' results rely on the symbolic computation in MATLAB, while the 'Numerical' results are determined by the standard precision in MATLAB (rounding unit $e \approx 10^{-16}$).

Table 1. Sequences $\boldsymbol{T}, \boldsymbol{G}, \boldsymbol{K}$ for Tests.

	$\boldsymbol{T}$	$\boldsymbol{G}$	$\boldsymbol{K}$
Test 1	$\{0,1,3,7,9,10\}$	$\{6,5,5,4,5\}$	$\{5,4,3,4\}$
Test 2	$\{-10^6, -10^6+1, 0, 10^6-1, 10^6\}$	$\{5,3,3,5\}$	$\{3,2,3\}$
Test 3	$\{3^j\}, j=0,...,12$	$\{j\}, j=10,...,21$	$\{j\}, j=8,...,18$
Test 4	$\{2\times j\}, j=0,...,40$	$\{j\}, j=1,...,40$	$\{j\}, j=0,...,38$

Example 3. *Values in Table 2 are calculated from Algorithm 1, where 'Exact' in the second column indicates that the Bernstein representation matrix* $\boldsymbol{M}$ *is calculated through symbolic computation. Exact MD-spline basis and Bernstein basis are also calculated by symbolic computation. The absence of relative error in the third column demonstrates that Algorithm 1 is correct.*

Table 2. Algorithm 1 numerical results of Example 3 in Test 1.

t	'Exact' $N_5(t)$	Relative Error	'Numerical' $N_5(t)$	Relative Error
2	$1.673419034377224 \times 10^{-1}$	0	$1.673419034377223 \times 10^{-1}$	3.1616×10^{-16}
3	$3.951056203260072 \times 10^{-1}$	0	$3.951056203260071 \times 10^{-1}$	3.0341×10^{-16}
5	$3.797960943382186 \times 10^{-1}$	0	$3.797960943382184 \times 10^{-1}$	5.0188×10^{-16}
7	$5.419134913010994 \times 10^{-2}$	0	$5.419134913010981 \times 10^{-2}$	2.3148×10^{-15}

Example 4. *This experiment demonstrates that Algorithm 1 requires revision. The addition of two floating integers with significantly dissimilar digits will result in significant numerical errors. In Table 3, the value produced by Algorithm 1 on the third row has a large relative error and does not symmetric with the first value, whereas the problem is avoided by Algorithm 2.*

Table 3. Comparison of Algorithm 1 and Algorithm 2 numerical results in Test 2.

t	Alg 2 $N_5(t)$	Relative Error	Alg 1 $N_5(t)$	Relative Error
−999,999	$4.500002750000809 \times 10^{-13}$	2.2439×10^{-16}	$4.500002750000810 \times 10^{-13}$	4.4877×10^{-16}
0	$5.000000833333360 \times 10^{-1}$	2.2204×10^{-16}	$5.000000833333362 \times 10^{-1}$	2.2204×10^{-16}
999,999	$4.500002750000809 \times 10^{-13}$	2.2439×10^{-16}	$4.499733918805759 \times 10^{-13}$	5.9740×10^{-5}

Example 5. *Table 4 illustrates the values and relative errors of* $N_{17,21}$ *for Test 3. The RKI/Greville method is unquestionably stable. Algorithm 2 has adequate numerical precision in this instance, where the highest degree exceeds 20. This algorithm is typically stable in experiments below this degree.*

Table 4. Comparison of Algorithm 2 and RKI/Greville method numerical results in Test 3.

t	Alg 2 $N_{17}(t)$	Relative Error	RKI/Greville $N_{17}(t)$	Relative Error
27	$2.015443122101803 \times 10^{-15}$	3.5939×10^{-15}	$2.015443122101811 \times 10^{-15}$	1.9570×10^{-16}
243	$6.858212462569680 \times 10^{-4}$	3.1905×10^{-15}	$6.858212462569703 \times 10^{-4}$	1.5809×10^{-16}
729	$1.743353153410859 \times 10^{-1}$	7.3071×10^{-16}	$1.743353153410862 \times 10^{-1}$	6.3683×10^{-16}
2187	$3.858286138702032 \times 10^{-1}$	3.6662×10^{-16}	$3.858286138702034 \times 10^{-1}$	8.6325×10^{-16}
6561	$2.453976654577640 \times 10^{-3}$	3.9579×10^{-16}	$2.453976654577639 \times 10^{-3}$	1.7672×10^{-16}

Example 6. *This experiment shows that the efficiency of different methods, as measured by MATLAB execution time. Results in Table 4 are calculated using the same point values for four tests. All tests are performed on an environment consisting of MATLAB 2020a and the AMD Ryzen 5 4600U.*

Calculating the values of basis functions at the internal knots yields the running time. Note that Algorithm 2 can calculate a single function in the MD-spline basis; its experiment is divided into two separate instances. The information displayed in the second column of Table 4 represents the duration of a single basis function. In order to ensure that values of this function are not zero at most knots, we select the middle basis function. A second scenario involves calculating the values of all basis functions at knots.

Based on the data in Table 5, it is evident that the efficacy of Algorithm 2 is assured. When the degree of basis is low, as in Test 1 and Test 2, there is little difference between the computation times of single functions and whole functions. However, as the degree increases, so does the efficiency gap.

In some extreme cases, such as Test 4, the degrees of the basis functions are extremely high, making it inefficient to calculate the values of all functions. As the RKI/Greville method is calculated on separate pieces, it needs longer running time.

Table 5. Running time in different tests.

Test	Alg 2 (s)	Alg 2 Global (s)	RKI/Greville (s)
Test 1	0.091	0.106	0.117
Test 2	0.088	0.098	0.101
Test 3	0.105	0.254	0.301
Test 4	0.205	1.294	3.070

Compared to the RKI/Greville method, Algorithm 2 is less accurate numerically but more efficient in its calculations. Using a B-spline basis instead of a Bernstein basis as representation functions can improve the numerical precision due to the smaller representation matrix reducing the number of floating-point operations.

Recursive Bernstein representation is not only meaningful for calculation, but it also provides a tool for investigating the recursive relations of MD-spline.

4. Generalized de Boor–Cox-Type Recursive Relations

In this section, we will discuss two generalized de Boor–Cox-type relations of MD-splines.

4.1. The First Form of Generalized de Boor–Cox-Type Relation

A de Boor–Cox-type formula for C^1 MD-spline bases exists because C^0 functions are formed by two or three Bernstein polynomial segments, and vectors corresponding to C^0 functions on each interval are composed of an entry equal to 1 and the rest equal to several elements at 0. Therefore, C^1 functions can always be represented by two C^0 functions. However, this type of relation does not exist between C^k, $k > 1$, functions and C^{k-1} functions.

Theorem 1. *There is no de Boor–Cox-type recursive relation for C^k MD-spline basis for $k > 1$.*

Proof of Theorem 1. For a C^k, $k > 1$, function $N_{i,D}$, assume that the following equation holds:

$$N_{i,D}(t) = f_1(t)N_{i,D-1}(t) + f_2(t)N_{i+1,D-1}(t) \tag{38}$$

where $f_1(t)$ and $f_2(t)$ are two polynomials of degree 1. Without loss of generality, on a single interval $[t_j, t_{j+1})$, let $\boldsymbol{A}^j_{i,D-1} = (a_{11}, a_{12}, \ldots, a_{1s})$ and $\boldsymbol{A}^j_{i+1,D-1} = (a_{21}, a_{22}, \ldots, a_{2s})$. Both vectors $\boldsymbol{A}^j_{i,D-1}$ and $\boldsymbol{A}^j_{i+1,D-1}$ contain more than one non-zero element. Using $\boldsymbol{A}^j_{i,D}(h)$ to indicate the h-th element of $\boldsymbol{A}^j_{i,D}$, it should be satisfied that

$$\begin{aligned}\boldsymbol{A}_{i,D}^{j}(h) =& c + \frac{t-t_j}{t_{j+1}-t_j}\sum_{p=1}^{h}(\delta_{i,D-1}a_{1p} - \delta_{i+1,D-1}a_{2p}) + \frac{t_{j+1}-t}{t_{j+1}-t_j}\sum_{p=1}^{h-1}(\delta_{i,D-1}a_{1p} - \delta_{i+1,D-1}a_{2p}) \\ =& C_h + \frac{t-t_j}{t_{j+1}-t_j}(\delta_{i,D-1}a_{1h} - \delta_{i+1,D-1}a_{2h}) \\ =& f_1(t)a_{1h} + f_2(t)a_{2h},\end{aligned} \tag{39}$$

where c and C_h are two constants. Then, functions $f_1(t)$ and $f_2(t)$ will be

$$f_1(t) = \frac{t-t_j}{t_{j+1}-t_j}\delta_{i,D-1} + \alpha;\ \ f_2(t) = -\frac{t-t_j}{t_{j+1}-t_j}\delta_{i+1,D-1} + \beta;\ \ \alpha a_{1h} + \beta a_{2h} = C_h, \tag{40}$$

where α and β are two unknown real numbers. As h is arbitrary, we have

$$\alpha a_{11} + \beta a_{21} = C_1;\ \ldots\ \alpha a_{1h} + \beta a_{2h} = C_h;\ \ldots\ \alpha a_{1s} + \beta a_{2s} = C_s. \tag{41}$$

Since there are more than one non-zero element in $\boldsymbol{A}_{i,D-1}^{j}$ and $\boldsymbol{A}_{i+1,D-1}^{j}$, system (41) is overdetermined and has no solution in most cases. Thus, Equation (38) does not hold. □

Any C^k, $k > 1$, MD-spline basis function can be represented as several C^0 functions.

Theorem 2. *For C^k, $k > 1$ MD-spline basis, assume that functions $\{N_{i,s}\}_{i=D-s+1}^{\mathcal{K}}$ are all C^0 functions or Bernstein polynomials. Then, function $N_{i,n}$, $n > s$, can be represented as*

$$N_{i,n}(t) = f_{i,s}(t)N_{i,s}(t) + f_{i+1,s}(t)N_{i+1,s}(t) + \ldots + f_{i+n-s,s}(t)N_{i+n-s,s}(t). \tag{42}$$

Proof. Proof of Theorem 2 Since each piece of C^0 functions equals to Bernstein polynomial, coefficient polynomials $\{f_{j,s}\}_{j=i}^{i+n-s}$ can be determined by Bernstein representation vectors. Since $\{\boldsymbol{B}_{d_j+n-D}^{j}\}_{j=p(s_i)}^{p(x_{i+D-n})-1}$ can be expressed by $\{\boldsymbol{B}_{d_j+s-D}^{j}\}_{j=p(s_i)}^{p(x_{i+D-n})-1}$ as

$$\boldsymbol{B}_{d_j+n-D}^{j} = \boldsymbol{L}_{s,n}^{j}\boldsymbol{B}_{d_j+s-D}^{j}, \tag{43}$$

where

$$\boldsymbol{L}_{s,n}^{j} = \begin{pmatrix} b_{0,n-s}^{j}(t) & & & \\ b_{1,n-s}^{j}(t) & b_{0,n-s}^{j}(t) & & \\ \vdots & \vdots & \ddots & \\ b_{n-s,n-s}^{j}(t) & b_{n-s,n-s-1}^{j}(t) & \ddots & b_{0,n-s}^{j}(t) \\ & b_{n-s,n-s}^{j}(t) & \ddots & \vdots \\ & & & b_{n-s-1,n-s}^{j}(t) \\ & & & b_{n-s,n-s}^{j}(t) \end{pmatrix}, \tag{44}$$

$N_{i,n}$ can also be expressed by $\{\boldsymbol{B}_{d_j+s-D}^{j}\}_{j=p(s_i)}^{p(x_{i+D-n})-1}$. Therefore, Equation (42) always holds in the inner interval $[t_{p(s_i)+1}, t_{p(x_{i-D+n})-1})$, and we only need to focus on the first interval $[t_{p(s_i)}, t_{p(s_i)+1})$ and the last interval $[t_{p(x_{i-D+n})-1}, t_{p(x_{i-D+n})})$.

On the first one $[t_{p(s_i)}, t_{p(s_i)+1})$, assume that $\boldsymbol{A}_{i,s}^{p(s_i)}(j) = 1$; then, elements from the first to $(j+s-1)$th in $\boldsymbol{A}_{i,n}^{p(s_i)}$ are 0, and $\boldsymbol{A}_{i,n}^{p(s_i)}(j+s)$ is nonzero according to the integral property of Bernstein polynomials. Thus, elements from the first to the $(j-1)$th in $\boldsymbol{A}_{i,n}^{p(s_i)}\boldsymbol{L}_{s,n}^{p(s_i)}$ are 0, which means that $\boldsymbol{A}_{i,n}^{p(s_i)}\boldsymbol{L}_{s,n}^{p(s_i)} = \sum_{j=i}^{i+n-s} f_{j,s}(t)\boldsymbol{A}_{j,s}^{p(s_i)}$.

Similarly, on the last one $[t_{p(x_{i-D+n})-1}, t_{p(x_{i-D+n})})$, this relationship can be determined, then $N_{i,n}$ is expressed as Equation (42). □

Based on the recursive property of Bernstein polynomials, Theorem 2 reveals a relationship between functions in the recursive process, which is the first form of generalized de Boor–Cox-type relation. The coefficient polynomials $\{f_{j,s}\}_{j=i}^{i+n-s}$ are piecewise, but not unique. Moreover, their domains must fall inside the support intervals of the corresponding functions.

The recursive method of Bernstein representation vectors provides a tool for calculating the coefficient polynomials. Since C^0 functions and Bernstein polynomials have unique Bernstein representation vectors, all $\{f_{j,s}\})_{j=i}^{i+n-s}$ in Equation (42) will be computed from $\boldsymbol{A}_{i,n}^{j}\boldsymbol{L}_{s,n}^{j}, j = p(s_i), ..., p(x_{i+n-D}) - 1$.

Note that $\vec{\mathcal{A}}_{j,s}$, $j = i, \ldots, i+n-s$ has a single non-zero element 1 and has the same length as $\vec{\mathcal{A}}_{i,n}$ (line 6) on each interval. Therefore, coefficient $f_{j,s}$ is the corresponding element in $\vec{\mathcal{A}}_{i,n}$, which can be obtained by the element-wise division operation in Algorithm 3 (line 6).

Algorithm 3 Coefficient polynomials calculation.

Input: $\{\vec{\mathcal{A}}_{j,s}\}_{j=i}^{i+n-s}$
Output: $\{f_{j,s}\}_{j=i}^{i+n-s}$

1: Compute $\vec{\mathcal{A}}_{i,n}$ by Algorithm 2
2: **for** k$\leftarrow$ $p(s_i)$ to $p(x_{i+D}) - 1$ **do**
3: $\quad \boldsymbol{A}_{i,n}^{k} \leftarrow \boldsymbol{A}_{i,n}^{k}\boldsymbol{L}_{s,n}^{k}$;
4: **end for**
5: **for** j$\leftarrow$ i to $i+n-s$ **do**
6: $\quad \vec{\mathcal{A}}_{j,s} \leftarrow \vec{\mathcal{A}}_{j,s}./\vec{\mathcal{A}}_{i,n}$;
7: $\quad$ **for** k$\leftarrow$ $p(s_j)$ to $p(x_{i+s-D}) - 1$ **do**
8: $\quad\quad f_{j,s}^{k}(t) \leftarrow \left(\sum_{q=1}^{d_k+s-D+1} \boldsymbol{A}_{j,s}^{k}(q)\right)^{-1}$;
9: $\quad$ **end for**
10: **end for**

Typically, functions $N_{i,s}$, $N_{i+1,s}$, $\ldots$, $N_{i+n-s,s}$ are situated at a lower recursive level. Some C^0 functions occur at subsequent recursive levels. For instance, $N_{i,s+1}$ is a C^0 function and the representation functions of $N_{i,n}$ can be $N_{i,s+1}$, $N_{i+1,s}$, $\ldots$, $N_{i+n-s,s}$, which are situated at various recursive levels.

Example 7. *Assume that* $\boldsymbol{T} := \{0,1,2,3,5,8\}$, $\boldsymbol{G} := \{4,5,4,3,5\}$, $\boldsymbol{K} := \{3,2,3,2\}$. *Then,* $\{N_{i,2}\}_{i=5}^{8}$ *are all* C^0 *functions or Bernstein polynomials. Function* $N_{5,5}(t)$ *can be represented as follows:*

$$N_{5,5}(t) = f_{5,2}(t)N_{5,2}(t) + f_{6,2}(t)N_{6,2}(t) + f_{7,2}(t)N_{7,2}(t) + f_{8,2}(t)N_{8,2}(t), \tag{45}$$

considering the following diagonally chunked matrix:

$$\boldsymbol{L} = \begin{pmatrix} \boldsymbol{L}_{2,5}^{0} & & \\ & \boldsymbol{L}_{2,5}^{1} & \\ & & \boldsymbol{L}_{2,5}^{2} \end{pmatrix}, \tag{46}$$

where $\boldsymbol{L}_{2,5}^{0}$, $\boldsymbol{L}_{2,5}^{1}$ *and* $\boldsymbol{L}_{2,5}^{2}$ *can be obtained by Equation (44). Then, coefficient polynomials are determined by*

$$\boldsymbol{A}_{5,5}\boldsymbol{L} = \left((0, f_{5,2}^{0}(t)),\ (f_{5,2}^{1}(t), f_{6,2}(t), f_{7,2}(t)),\ (f_{8,2}(t), 0)\right). \tag{47}$$

Next, a distinct set of functions is employed to represent $N_{5,5}$. Note that $N_{7,3}$ is a C^0 function, and functions $f_{7,2}$, $f_{8,2}$ in Equation (45) can be replaced by $N_{7,3}$. Over three intervals, the coefficient polynomials must be computed successively. On interval $[2,3)$, it holds that

$$\boldsymbol{A}^2_{5,5}\boldsymbol{L}^2_{3,5} = \left(f^2_{7,3}(t), 0, 0\right). \tag{48}$$

On interval $[1,2)$, it holds that

$$\boldsymbol{A}^1_{5,5}\boldsymbol{L}^1_{3,5} = \left(g_1(t), g_2(t), f_{6,3}(t), f^1_{7,3}(t)\right), \tag{49}$$

for the same functions g_1 and g_2. Then,

$$(g_1(t), g_2(t), 0, 0)\boldsymbol{L}^2_{2,3} = \left(f^1_{5,2}(t), f_{6,2}(t), 0\right). \tag{50}$$

On interval $[0,1)$, it holds that

$$\boldsymbol{A}^0_{5,5}\boldsymbol{L}^0_{2,5} = \left(0, f^0_{5,2}(t)\right). \tag{51}$$

Thus,

$$N_{5,5}(t) = f_{5,2}(t)N_{5,2}(t) + f_{6,2}(t)N_{6,2}(t) + f_{6,3}(t)N_{6,3}(t) + f_{7,3}(t)N_{7,3}(t), \tag{52}$$

where functions $N_{5,2}$, $N_{6,2}$, $N_{6,3}$, ad $N_{7,3}$ are not at the same level, and the coefficient polynomials $f_{5,2}$ and $f_{6,2}$ are not equivalent to them in Equation (45).

Here, two distinct representations of this function are shown, and Figure 4 depicts the coefficient polynomials. These relationships are prevalent throughout the recursive procedure.

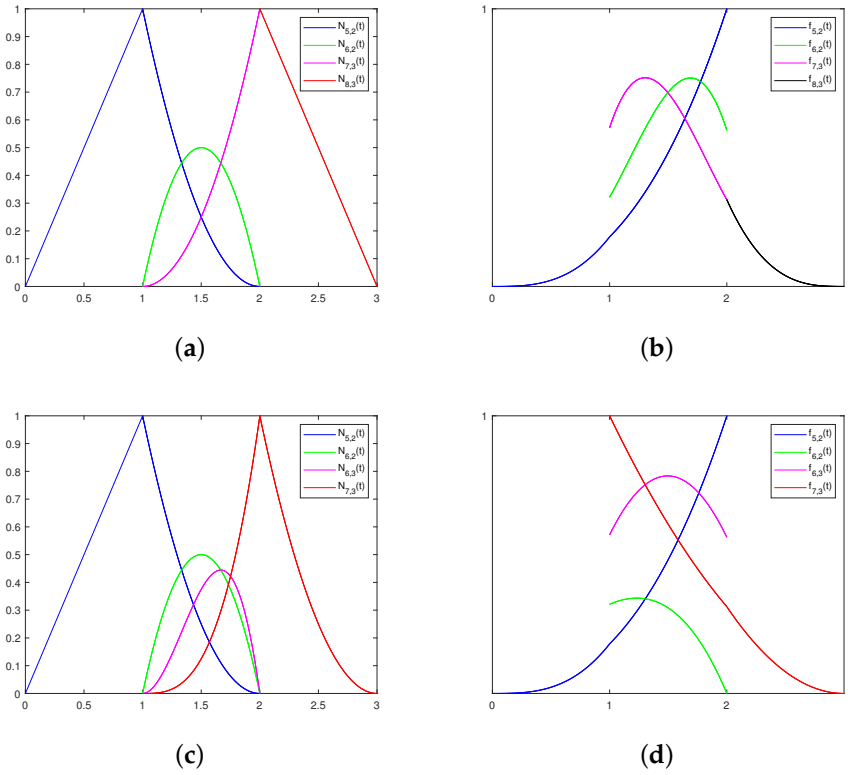

Figure 4. (**a**) Representation functions $N_{5,2}$, $N_{6,2}$, $N_{7,2}$, $N_{8,2}$. (**b**) Corresponding coefficient polynomials $f_{5,2}$, $f_{6,2}$, $f_{7,2}$, $f_{8,2}$. (**c**) Representation functions $N_{5,2}$, $N_{6,2}$, $N_{6,3}$, $N_{7,3}$. (**d**) Corresponding coefficient polynomials $f_{5,2}$, $f_{6,2}$, $f_{6,3}$, $f_{7,3}$.

Based on the above discussion, functions at the recursive levels n and $n-1$ can all be represented by several C^0 functions or Bernstein polynomials. Consequently, $N_{i,n}$ can be represented as follows:

$$N_{i,n}(t) = f_{i,n-1}(t)N_{i,n-1}(t) + f_{i+1,n-1}(t)N_{i+1,n-1}(t) + \sum_{j=i}^{i+n-s} f_{j,s}(t)N_{j,s}(t), \tag{53}$$

where functions $\{N_{j,s}\}_{j=i}^{i+n-s}$ refer to C^0 functions or Bernstein polynomials; $f_{i,n-1}$, $f_{i+1,n-1}$ are polynomials of degree 1. This equation is a general relationship for functions in the MD-spline recursive process.

This subsection summarizes several broad relationships that exist in the recursion; nonetheless, there is no law for generating coefficient polynomials. Consequently, the first generalized de Boor–Cox-type formula lacks practical relevance.

4.2. The Second Form of Generalized de Boor–Cox-Type Relation

Since Equation (38) cannot be found in the recursion of MD-spline basis, we must resort to functions outside of the recursion process in order to obtain this type of formula. If we design two suitable functions that can substitute two functions in the linear combination (42), we have a generalized de Boor–Cox-type formula.

For each $N_{i,n}$, $i = D-n+1, \ldots, \mathcal{K}$, $n = 0, \ldots, D$, we have the following result.

Theorem 3. *It holds that*

$$N_{i,n}(t) = (t - t_{p(s_i)})\mathcal{L}_{i,n}(t) + (t_{p(x_{i+n-D})} - t)\mathcal{R}_{i,n}(t), \tag{54}$$

where $\mathcal{L}_{i,n}$ and $\mathcal{R}_{i,n}$ are both piecewise polynomials.

Proof of Theorem 3. On interval $[t_j, t_{j+1}) \subset [s_i, x_{i+D-n})$, the Taylor expansion of function $N_{i,n}$ is given by all its derivatives. For convenience, let $f(t) = N_{i,n}(t), t \in [t_j, t_{j+1})$. Then

$$\begin{aligned}
f(t) =& f(t_j) + \sum_{r=1}^{d_j+n-D} \frac{1}{r!}(t-t_j)^r f^{(r)}(t_j) \\
=& f(t_{j+1}) + \sum_{r=1}^{d_j+n-D} \frac{1}{r!}(t_{j+1}-t)^r f^{(r)}(t_{j+1}) \\
=& f(t_j) + \sum_{r=1}^{d_j+n-D} \frac{1}{r!}(t - t_{p(s_i)} - t_j + t_{p(s_i)})^r f^{(r)}(t_j) \\
=& f(t_j) + \sum_{r=1}^{d_j+n-D} \frac{1}{r!}\left(\sum_{k=0}^{r}\binom{r}{k}(t-t_{p(s_i)})^k(t_{p(s_i)}-t_j)^{r-k}\right) f^{(r)}(t_j) \\
=& \sum_{r=0}^{d_j+n-D} (t_{p(s_i)} - t_j)^r f^{(r)}(t_j) \\
&+ (t - t_{p(s_i)}) \sum_{r=1}^{d_j+n-D} \frac{1}{r!}\left(\sum_{k=0}^{r-1}\binom{r}{k}(t-t_{p(s_i)})^k(t_{p(s_i)}-t_j)^{r-1-k}\right) f^{(r)}(t_j) \\
=& \sum_{r=0}^{d_j+n-D} (t_{j+1} - t_{p(x_{i+n-D})})^r f^{(r)}(t_{j+1}) \\
&+ (t_{p(x_{i+n-D})} - t) \sum_{r=1}^{d_j+n-D} \frac{1}{r!}\left(\sum_{k=0}^{r-1}\binom{r}{k}(t_{p(x_{i+n-D})}-t)^k(t_{j+1}-t_{p(x_{i+n-D})})^{r-1-k}\right) f^{(r)}(t_{j+1}).
\end{aligned} \tag{55}$$

Let

$$\begin{aligned}\mathcal{L}_{i,n}(t) =& \frac{1}{2}\sum_{r=1}^{d_j+n-D}\frac{1}{r!}\left(\sum_{k=0}^{r-1}\binom{r}{k}(t-t_{p(s_i)})^k(t_{p(s_i)}-t_j)^{r-1-k}\right)f^{(r)}(t_j)\\ &+\frac{1}{2(t_{p(x_{i+n-D})}-t_{p(s_i)})}\sum_{r=0}^{d_j+n-D}\left((t_{p(s_i)}-t_j)^r f^{(r)}(t_j)+(t_{j+1}-t_{p(x_{i+n-D})})^r f^{(r)}(t_{j+1})\right),\end{aligned} \tag{56}$$

and

$$\begin{aligned}\mathcal{R}_{i,n}(t) =& \frac{1}{2}\sum_{r=1}^{d_j+n-D}\frac{1}{r!}\left(\sum_{k=0}^{r-1}\binom{r}{k}(t_{p(x_{i+n-D})}-t)^k(t_{j+1}-t_{p(x_{i+n-D})})^{r-1-k}\right)f^{(r)}(t_{j+1})\\ &+\frac{1}{2(t_{p(x_{i+n-D})}-t_{p(s_i)})}\sum_{r=0}^{d_j+n-D}\left((t_{p(s_i)}-t_j)^r f^{(r)}(t_j)+(t_{j+1}-t_{p(x_{i+n-D})})^r f^{(r)}(t_{j+1})\right).\end{aligned} \tag{57}$$

Then, Equation (54) holds. □

Functions $\mathcal{L}_{i,n}$ and $\mathcal{R}_{i,n}$, here, are constructed by $N_{i,n}^{(r)}(t_j)$ and $N_{i,n}^{(r)}(t_{j+1}), r = 0, \ldots, d_j + n - D$. Due to the integral definition of the MD-spline basis, the values of derivatives at t_j, t_{j+1} can be calculated by the functions in previous recursive levels. The derivative equation is as follows:

$$N_{i,n}^{'}(t) = \delta_{i,n-1}N_{i,n-1}(t) - \delta_{i+1,n-1}N_{i+1,n-1}(t). \tag{58}$$

In addition, the values of $N_{i,n}(t_j)$ and $N_{i,n}(t_{j+1})$ can be calculated by the integral of $N_{i,n-1}$ and $N_{i+1,n-1}$.

Consequently, this form of the formula is conceptually similar to the de Boor–Cox formula. Despite the fact that new functions $\mathcal{L}_{i,n}$ and $\mathcal{R}_{i,n}$ must be generated, they are all constructed using information from previous functions. Only the intricacy of construction needs to be considered. Taylor expansions are only used for theoretical exposition and cannot be implemented in practice. The coefficient polynomials need only be guaranteed to be polynomials of degree 1; they need not be identical to them in Equation (54). It is even feasible to employ piecewise coefficients to lessen the complexity of construction.

The generalized de Boor–Cox-type formula will employ a generalized pyramid structure, which will be contrasted to the pyramid for B-spline in the accompanying illustration.

Example 8. *For B-spline basis* $\{F_{i,n}(t)\}_{i=1}^{\mathcal{K}}$*, all the basis functions possess similar pyramids; hence, often only one pyramid is depicted. Several pyramids may be utilized to describe the de Boor–Cox recursion for B-splines if the entire basis is used, as shown in Figure 5. The knot sequence is* $\boldsymbol{T} := \{0, 1, 2, 3, 4\}$ *and the degree is 4. All pyramids have the same height beginning with the degree 0 functions, i.e., the functions at level 0 in the de Boor–Cox-type recursion.*

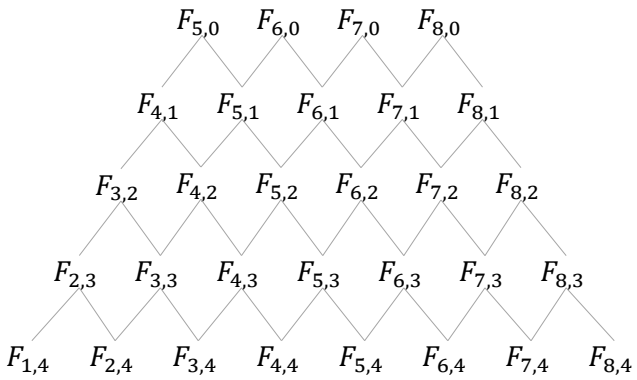

Figure 5. Pyramids for B-spline basis.

For a MD-spline basis with $\boldsymbol{T} := \{0,1,2,3,4\}$, $\boldsymbol{G} := \{3,2,4,3\}$, *and* $\boldsymbol{K} := \{1,1,2\}$, *the recursive process in Theorem 3 is shown in Figure 6. For a MD-spline basis, pyramids might have varying heights.*

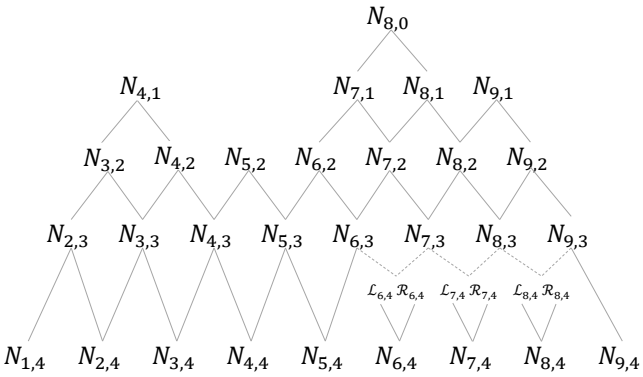

Figure 6. Pyramids for MD-spline basis.

Compared with the B-spline basis, the recursive process for the MD-spline basis is determined not only by the degrees but also by the continuous orders.

Although Theorem 3 provides an illustration of the theoretical plausibility of this design, the reality of the building itself is quite complicated. The Bernstein representation approach enables the construction of a structure, but because it is still sophisticated, it is not shown here. If a suitable construction approximation method exists, this theory will be practical.

5. Conclusions

In the paper, we present an evaluation approach for MD-splines. Unlike prior algorithms for evaluating MD-spline bases, this one is non-global, and its intermediate outcomes correspond to functions in the recursive process. Due to the absence of complicated operations, it is efficient in most cases. In addition, its numerical precision in normal degrees is comparable to that of the RKI/Greville approach. In addition to being useful for numerical evaluation, it offers a method for calculating coefficient polynomials in generalized de Boor–Cox-type relations.

Then, we provide two forms of the generalized de Boor–Cox-type formula for MD-splines. Both forms attempt to approximate the de Boor–Cox formula. The first form summarizes the majority of relationships that may appear in the recursion. The second form is a constructed structure. It is theoretically feasible, but requires a computationally cheap building technique.

It is important to note that the present value of the generalized de Boor–Cox-type formula is theoretical, but the preceding numerical algorithm has practical applicability. The second form of the generalized formula has potential because it simply has to address the construction problem. Future work might be devoted to the optimization of the numerical algorithm and the search for better construction techniques.

Author Contributions: Investigation, X.M. and W.S.; methodology, X.M. and W.S.; software, X.M.; writing—original draft preparation, X.M. and W.S.; writing—review and editing, X.M. and W.S.; funding acquisition, W.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (No. 61772013).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Carnicer, J.; Peña, J.M. Totally positive bases for shape preserving curve design and optimality of B-splines. *Comput. Aided. Geom. Design* **1994**, *11*, 633–654. [CrossRef]
2. Shen, W.; Wang, G. Changeable degree spline basis functions. *J. Comput. Appl. Math.* **2010**, *234*, 2516–2529. [CrossRef]
3. Nürnberger, G.; Schumaker, L.L.; Sommer, M.; Strauss, H. Generalized Chebyshevian splines. *SIAM J. Math. Anal.* **1984**, *15*, 790–804. [CrossRef]
4. Costantini, P.; Goodman, T.N.T.; Manni, C. Constructing C^3 shape-preserving interpolating space curves. *Adv. Comput. Math.* **2001**, *14*, 103–127. [CrossRef]
5. Costantini, P.; Manni, C. Shape-preserving C^3 Interpolation: The Curve Case. *Adv. Comput. Math.* **2003**, *18*, 41–63. [CrossRef]
6. Kaklis, P.D.; Pandelis, D.G. Convexity-preserving polynomial splines of non-uniform degree. *IMA J. Numer. Anal.* **1990**, *10*, 455–468. [CrossRef]
7. Costantini, P. Variable degree polynomial splines. In *Curves and Surfaces with Applications in CAGD*; Vanderbilt University Press: Nashville, TN, USA, 1997; pp. 85–94.
8. Costantini, P. Curve and surface construction using variable degree polynomial splines. *Comput. Aided. Geom. Design* **2000**, *17*, 419–446. [CrossRef]
9. Sederberg, T.W.; Zheng, J.; Song, X. Knot intervals and multi-degree splines. *Comput. Aided. Geom. Design* **2003**, *20*, 455–468. [CrossRef]
10. Wang, G.Z.; Deng, C.Y. On the degree elevation of B-spline curves and corner cutting. *J Comput. Aided. Geom. Design* **2007**, *24*, 90–98. [CrossRef]
11. Shen, W.; Wang, G. A basis of multi-degree splines. *Comput. Aided. Geom. Design* **2010**, *27*, 23–35. [CrossRef]
12. Shen, W.; Wang, G.; Yin, P. Explicit representations of changeable degree spline basis functions. *J. Comput. Appl. Math.* **2013**, *238*, 39–50. [CrossRef]
13. Shen, W.; Yin, P.; Tang, C. Degree elevation of changeable degree spline. *J. Comput. Appl. Math.* **2016**, *300*, 56–67. [CrossRef]
14. Beccari, C.V.; Casciola, G.; Morigi, S. On multi-degree splines. *Comput. Aided. Geom. Design* **2017**, *58*, 8–23. [CrossRef]
15. Buchwald, B.; Mühlbach, G. Construction of B-splines for generalized spline spaces generated from local ECT-systems. *J. Comput. Appl. Math.* **2003**, *159*, 249–267. [CrossRef]
16. Speleers, H. Algorithm 999: Computation of multidegree B-splines. *ACM. Trans. Math. Softw.* **2019**, *45*, 1–15. [CrossRef]
17. Toshniwal, D.; Speleers, H.; Hiemstra, R.R.; Manni, C.; Hughes, T.J. Multi-degree B-splines: Algorithmic computation and properties. *Comput. Aided. Geom. Design* **2020**, *76*, 101792. [CrossRef]
18. Hiemstra, R.R.; Hughes, T.J.; Manni, C.; Speleers, H.; Toshniwal, D. A Tchebycheffian extension of multi-degree B-splines: Algorithmic computation and properties. *SIAM. J. Numer. Anal.* **2020**, *2*, 1138–1163. [CrossRef]
19. Beccari, C.V.; Casciola, G. Matrix representations for multi-degree B-splines. *J. Comput. Appl. Math.* **2021**, *381*, 113007. [CrossRef]
20. Beccari, C.V.; Casciola, G. Stable numerical evaluation of multi-degree B-splines. *J. Comput. Appl. Math.* **2022**, *400*, 113743. [CrossRef]
21. De Boor, C. On calculating with B-splines. *J. Abbr.* **1972**, *6*, 50–62. [CrossRef]
22. Cox, M. The numerical evaluation of B-splines. *J. Inst. Math. Appl.* **1972**, *10*, 134–149. [CrossRef]
23. Goldman, R. Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling. In *The Morgan Kaufmann Series in Computer Graphics*; Morgan Kaufmann Publishers, Academic Press: San Diego, CA, USA, 2002; pp. 347–443.
24. Beccari, C.V.; Casciola, G.A. Cox-de Boor-type recurrence relation for C^1 multi-degree splines. *Comput. Aided. Geom. Design* **2019**, *75*, 101784. [CrossRef]
25. Li, X.; Huang, Z.J.; Liu, Z. A geometric approach for multi-degree spline. *J. Comput. Sci. Tech.* **2012**, *27*, 841–850. [CrossRef]
26. Farouki, R.T. The Bernstein polynomial basis: A centennial retrospective. *Comput. Aided. Geom. Design* **2012**, *29*, 379–419. [CrossRef]