Instytut Informatyki
Wydział Matematyki i Informatyki
Uniwersytet Wrocławski

FILIP CHUDY

NOWE ALGORYTMY DLA WIELOMIANÓW BERNSTEINA, ICH BAZ DUALNYCH I FUNKCJI B-SKLEJANYCH

Praca doktorska

Promotor: dr hab. Paweł Woźny

Wrocław 2022

Institute of Computer Science
Department of Mathematics and Computer Science
University of Wrocław

FILIP CHUDY

NEW ALGORITHMS FOR BERNSTEIN POLYNOMIALS, THEIR DUAL BASES, AND B-SPLINE FUNCTIONS

Ph.D. Thesis

Supervisor: dr hab. Paweł Woźny

Wrocław 2022

Acknowledgments

I would like to thank dr hab. Paweł Woźny for his kind and thoughtful guidance, as well as valuable suggestions and comments which helped improve this thesis.

Wrocław, 2022

Streszczenie

Praca przedstawia nowe algorytmy dla krzywych Béziera, krzywych B-sklejanych i dualnych wielomianów Bernsteina. Zaproponowane metody pozwalają na przyspieszenie obliczeń wykonywanych m.in. w grafice komputerowej i analizie numerycznej.

Nowy algorytm szybkiego wyznaczania wartości krzywej Béziera łączy zalety znanych wcześniej metod rozwiązywania tego problemu: liniową złożoność schematu Hornera oraz interpretację geometryczną, własność otoczki wypukłej i operowanie na kombinacjach wypukłych właściwe algorytmowi de Casteljau. Zaproponowaną metodę można stosować nie tylko do wielomianowych i wymiernych krzywych Béziera, ale też do tzw. *wymiernych obiektów parametrycznych*. Ważnym ich przykładem są wymierne tensorowe i trójkątne powierzchnie Béziera. Zastosowanie zaprezentowanego w pracy podejścia pozwala osiągnąć optymalną złożoność, tj. proporcjonalną do liczby punktów kontrolnych definiujących te obiekty.

Opisano też nowy algorytm wyznaczania współczynników funkcji B-sklejanych w bazie Bernsteina-Béziera, oparty na nowej zależności różniczkowo-rekurencyjnej spełnianej przez te funkcje. Przy pewnych założeniach o węzłach definiujących te funkcje, algorytm jest optymalny. Przedstawiony został również szkic podobnego algorytmu dla reprezentacji funkcji B-sklejanych w bazie potęgowej. Jeżeli znane są współczynniki Bernsteina-Béziera funkcji B-sklejanych, można zastosować nowy algorytm ewaluacji krzywej Béziera, by przyspieszyć procedurę wyznaczenia punktu na krzywej B-sklejanej. Przy obliczaniu w wielu punktach wartości wielu opartych na tych samych węzłach krzywych B-sklejanych uzyskuje się algorytm mający niższą złożoność niż w metodzie wykorzystującej algorytm de Boora-Coxa. Stosując podobne podejście można wyznaczyć każdą funkcję B-sklejaną w czasie liniowym względem jej stopnia.

W pracy podano też wiele nowych związków różniczkowych, różniczkowo-rekurencyjnych i rekurencyjnych spełnianych przez dualne wielomiany Bernsteina tego samego stopnia. Znajomość takich związków rekurencyjnych pozwala, na przykład, na znalezienie wartości wszystkich $n+1$ dualnych wielomianów Bernsteina tego samego stopnia n w optymalnym czasie $O(n)$. Rekurencji tych można też użyć do ewaluacji dowolnej kombinacji liniowej dualnych wielomianów Bernsteina stopnia n, np. poprzez użycie algorytmu typu Clenshawa. Taką procedurę można wykonać w czasie liniowym względem stopnia wielomianu zapisanego w bazie dualnych wielomianów Bernsteina, czyli równym złożoności obliczeniowej schematu Hornera. W pracy przedstawiono kilka związków rekurencyjnych łączących kolejne dualne wielomiany Bernsteina tego samego stopnia: jednorodne rzędu czwartego, trzeciego i drugiego oraz niejednorodny rzędu pierwszego. Związek niejednorodny rzędu pierwszego został przetestowany pod względem efektywności numerycznej. Przeprowadzone eksperymenty pokazują, że algorytm wykorzystujący ten związek daje dobre wyniki nawet dla tak wysokich stopni dualnych wielomianów Bernsteina jak 3000 czy 5000.

Związki rekurencyjne między dualnymi wielomianami Bernsteina tego samego stopnia można też zastosować do równoległego obniżania stopnia krzywej Béziera z ograniczeniami. Dzięki nim można istotnie ułatwić korzystanie z obliczeń równoległych, choć przy zachowaniu dotychczasowej złożoności, poprzez przekształcenie związku rekurencyjnego wykorzystywanego w podejściu opartym na bazach dualnych.

Abstract

The thesis presents new algorithms for Bézier curves, B-spline curves, and dual Bernstein polynomials. The proposed methods allow to accelerate the computations performed, e.g., in computer graphics and numerical analysis.

A new algorithm for fast evaluation of a Bézier curve combines the qualities of previously known methods for solving this problem, i.e., the linear complexity of the Horner's scheme and the geometric interpretation, the convex hull property, and operating only on convex combinations which are the advantages of the de Casteljau algorithm. The new method can be used not only for polynomial and rational Bézier curves but also for so-called *rational parametric objects*. Their prominent examples are rational rectangular and rational triangular Bézier surfaces. The algorithm has optimal complexity, i.e., proportional to the number of control points which define these objects.

Additionally, a new algorithm for computing the Bernstein-Bézier coefficients of B-spline functions which is based on a new differential-recurrence relation satisfied by B-spline functions has been described. Under some assumptions about the knots which define these functions, the algorithm is optimal. A sketch of a similar algorithm for the power basis coefficients of B-spline functions has been presented as well. If the Bernstein-Bézier coefficients of B-spline functions are known, one can evaluate one of the functions in linear time with respect to its coefficients. The new algorithm for evaluating a Bézier curve can also be used to accelerate the evaluation of many B-spline curves at multiple points, thus getting a method with lower complexity than the one based on the de Boor-Cox algorithm.

In this thesis, many new diferential, differential-recurrence, and recurrence relations satisfied by dual Bernstein polynomials of the same degree have been given. Such recurrence relations allow, e.g., to find the values of all $n + 1$ dual Bernstein polynomials of degree n in the optimal $O(n)$ time. Moreover, these relations can be used to evaluate any linear combination of dual Bernstein polynomials of degree n, e.g., by applying the Clenshaw-type algorithm. Such procedure can be performed in linear time with respect to the degree of the dual Bernstein basis used, i.e., it has the same complexity as the Horner's scheme. The new recurrence relations for dual Bernstein polynomials of the same degree are: homogeneous of orders four, three, and two, as well as non-homogeneous of order one. The numerical efficiency of the non-homogeneous relation of order one has been tested. According to performed experiments, an algorithm based on this relation works well even for high degrees of dual Bernstein polynomials, like 3000 or 5000.

The recurrence relations which connect dual Bernstein polynomials of the same degree can find their application in parallel constrained degree reduction of a Bézier curve. They can simplify the recurrence relation previously used in the approach which applies dual bases, thus allowing to use simpler parallel computations, although the total complexity of this method remains the same.

Contents

Chapter 1

Introduction

In this thesis, algorithms for evaluating various parametric objects — in particular, Bézier curves and surfaces as well as B-spline curves — are accelerated, which can result in much faster rendering of CAGD objects. Similarly, the evaluation of dual Bernstein polynomials — which find many applications in, e.g., computer graphics, approximation theory, numerical analysis — has been accelerated. This allows to use dual basis techniques for Bernstein polynomials with much higher efficiency, thus reducing the computational bottleneck of dual projections.

Chapter 1 introduces the concepts and notions which are necessary for the remaining chapters.

Chapter 2 presents and expands the results given previously in [96]. More precisely, a geometric algorithm for evaluating (polynomial or rational) Bézier curves has been introduced. For a d-dimensional curve of degree n, its computational complexity is $O(nd)$, which is optimal. To the best of the author's knowledge, this is the first known algorithm which evaluates a Bézier curve and is both geometric and has $O(nd)$ complexity. Notice that the de Casteljau algorithm is geometric but has $O(n^2d)$ complexity, while the Horner's scheme has the desired $O(nd)$ complexity while not being geometric. The algorithm can be generalized to other rational parametric objects. In particular, in the case of rectangular and triangular Bézier surfaces, the complexity is proportional to the number of control points (i.e., the method is optimal), which is a significant improvement over the corresponding de Casteljau algorithms.

In Chapter 3, a new differential-recurrence relation for the B-spline functions of the same degree is shown. From this relation, a recursive method of computing the coefficients of B-spline functions of degree m in Bernstein-Bézier form is derived. Its complexity is proportional to the number of coefficients in the case of coincident boundary knots. This means that, asymptotically, the algorithm is optimal. In other cases, the complexity is increased by at most $O(m^3)$. When the Bernstein-Bézier coefficients of B-spline basis functions are known, one can, e.g., compute any of them in $O(m)$ time or convert a piece of a d-dimensional B-spline curve of degree m over one knot span to a Bézier curve in $O(m^2)$ time and then evaluate it in $O(md)$ time using the geometric algorithm given in Chapter 2. Since one only needs to convert each knot span once, it scales well when evaluating many B-spline curves at multiple points, e.g., in order to render it. Such approach has lower computational complexity than using the de Boor-Cox algorithm. The problem of finding the coefficients of the B-spline functions in the power basis can be solved similarly.

Chapter 4 is based on [18] and presents new differential and differential-recurrence relations

satisfied by dual Bernstein polynomials. They find their application in proving new recurrence relations for dual Bernstein polynomials in the following chapter.

Chapter 5 presents, in greater detail, the recurrence relations for dual Bernstein polynomials which were previously shown in [18, 19]. These results allow to solve several problems posed when using dual Bernstein basis. In particular, such a result allows to compute a linear combination of dual Bernstein polynomials of degree n in $O(n)$ time, reducing the complexity by an order of magnitude. One of the recurrence relations is analyzed in greater detail and it is shown that its numerical performance is good even for dual Bernstein bases of high degree ($n \approx 3000, 5000$). This makes dual Bernstein bases a more potent tool in a multitude of their applications.

In Chapter 6, the recurrence relation for dual Bernstein polynomials given in the previous chapter finds another application, as it can be used to simplify the computation of coefficients for k, l-constrained degree reduction of Bézier curves given in [97]. While the new approach does not improve the computational complexity of the degree reduction, it allows more parallel computations.

As of the time of writing, the results presented in Chapters 3 and 6 have not yet been published. Most of the relations, equations and algorithms which are given in this thesis have been checked using the computer algebra system Maple™ (see [71]).

1.1 Remarks on points and vectors

In computer-aided geometric design, to preserve the geometric interpretations of some methods, the algorithms often operate not in the vector space $\mathbb{R}^d$ but in the point space $\mathbb{E}^d$. The approach presented here can be examined in greater detail, e.g., in [78, Chapter 1], [36, Chapter 2]. It will be of use in Chapter 2. To avoid confusion, points in $\mathbb{E}^d$ will be denoted using the upper-case letters $\mathsf{A}, \mathsf{B}, \ldots$, while vectors in $\mathbb{R}^d$ will be denoted using the lower-case letters $\mathsf{a}, \mathsf{b}, \ldots$.

Fact 1.1. *The following types of operations are well-defined in $\mathbb{E}^d$ and $\mathbb{R}^d$:*

point - point = vector,	
point + vector = point	*(translation),*
vector + vector = vector	*(vector composition),*
scalar × vector = vector	*(vector scaling).*

The basic operation types given in Fact 1.1 can be combined to create more sophisticated ones.

Example 1.2. *Let $\mathsf{A}, \mathsf{B} \in \mathbb{E}^d$. The operation*

$$\frac{2}{5}\mathsf{A} + \frac{3}{5}\mathsf{B}$$

gives a point in $\mathbb{E}^d$. It can be interpreted as an equivalent form

$$\mathsf{A} + \frac{3}{5}\mathsf{B} - \frac{3}{5}\mathsf{A} = \mathsf{A} + \frac{3}{5}(\mathsf{B} - \mathsf{A}),$$

which consists of operations explicitly shown in Fact 1.1. The result is the point A translated by the vector $\frac{3}{5}(\mathsf{B} - \mathsf{A})$, i.e., a vector $\mathsf{B} - \mathsf{A}$ scaled by $\frac{3}{5}$.

A similar restructuring can be done for any weighted sum of points under the condition that the sum of all weights equals 1.

Fact 1.3. *Let $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and $\sum_{k=0}^{n} \alpha_k = 1$. Let $A_0, A_1, \ldots, A_n \in \mathbb{E}^d$. The weighted sum*

$$\sum_{k=0}^{n} \alpha_k A_k$$

is a barycentric combination *of points $A_0, A_1, \ldots, A_n$ and is a point in $\mathbb{E}^d$. It can be represented, for example, as*

$$A_0 + \sum_{k=1}^{n} \alpha_k (A_k - A_0).$$

Remark 1.4. *In the approach presented here, of all linear combinations of points, only the barycentric ones are interpreted as points. This allows the operations on points to be invariant with respect to translation. More precisely, the barycentric combination of $A_0, A_1, \ldots, A_n \in \mathbb{E}^d$, i.e.,*

$$\sum_{k=0}^{n} \alpha_k A_k,$$

satisfies

$$\sum_{k=0}^{n} \alpha_k \left(A_k + b\right) = \sum_{k=0}^{n} \alpha_k A_k + \left(\sum_{k=0}^{n} \alpha_k\right) b = \sum_{k=0}^{n} \alpha_k A_k + b$$

for any vector b in $\mathbb{R}^d$, thus

$$\sum_{k=0}^{n} \alpha_k \left(A_k + b\right) - b = \sum_{k=0}^{n} \alpha_k A_k$$

Just as in the case of points, the basic operations given in Fact 1.1 can be combined to create more sophisticated expressions for vectors.

Example 1.5. *Let $A, B, C \in \mathbb{E}^d$. The operation*

$$\frac{2}{5}A + \frac{3}{5}B - C$$

gives a vector in $\mathbb{R}^d$. It can be interpreted as an equivalent form

$$\frac{2}{5}(A - C) + \frac{3}{5}(B - C),$$

which consists of operations explicitly shown in Fact 1.1.

One can check that this example can be extended to any weighted sum of points if the sum of all weights equals 0.

Fact 1.6. *Let $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and $\sum_{k=0}^{n} \alpha_k = 0$. Let $A_0, A_1, \ldots, A_n \in \mathbb{E}^d$. The weighted sum*

$$\sum_{k=0}^{n} \alpha_k A_k$$

is a vector in $\mathbb{R}^d$.

Definition 1.7. *A barycentric combination of* $A_0, A_1, \ldots, A_n \in \mathbb{E}^d$ *with weights* $\omega_0, \omega_1, \ldots, \omega_n$,

$$\sum_{k=0}^{n} \omega_k A_k,$$

is a convex combination *if* $\omega_0, \omega_1, \ldots, \omega_n \geq 0$.

Definition 1.8. *Let* $C \subseteq \mathbb{E}^d$. *A* convex hull *of* C, *denoted as*

$$conv\ C,$$

is the smallest convex set such that $C \subseteq conv\ C$. *Equivalently,*

$$conv\ C = \{A \in \mathbb{E}^d : A \text{ is a convex combination of } C\}.$$

The divide and conquer algorithm for finding a convex hull of n points in $\mathbb{E}^d$, if $d \in \{2, 3\}$ is given in [79]. Its complexity is $O(n \log n)$. Some additional classical algorithms can be found in [5, 48]. The lower bound for finding a convex hull of n points in two or three dimensions is, however, $O(n \log h)$, where h is the number of points which form the convex hull. Two algorithms which have this optimal complexity can be found in [55] and [16], with the latter being simpler.

For a higher dimension count, the complexity is $O(n^{\lfloor d/2 \rfloor})$. See [26, p. 256–257].

1.2 Parametric curves

Parametric curves are useful in computer-aided geometric design. They serve as a memory-efficient way of representing certain geometric objects.

Definition 1.9. *Let* $[a, b] \subseteq \mathbb{R}$. *A* parametric curve *is a continuous function* $\mathsf{F} : [a, b] \to \mathbb{E}^d$.

When F is known, one can evaluate the parametric curve for any $t \in [a, b]$.

Example 1.10. *A parametric curve* $\mathsf{F} : [0, 20\pi] \to \mathbb{E}^2$ *is given by*

$$\mathsf{F}(t) := (t \sin 2t, \cos 3t)\,.$$

Figure 1.1 illustrates this case.

Example 1.11. *A spiral can be obtained using a parametric curve* $\mathsf{F} : [0, T] \to \mathbb{E}^2$ *given by*

$$\mathsf{F}(t) := (t \cos t, t \sin t)\,.$$

Figure 1.2 illustrates this case for $T = 30\pi$.

Definition 1.12. *Let the parametric curves*

$$\mathsf{F} : [a, b] \to \mathbb{E}^d$$

and

$$\mathsf{G} : [b, c] \to \mathbb{E}^d$$

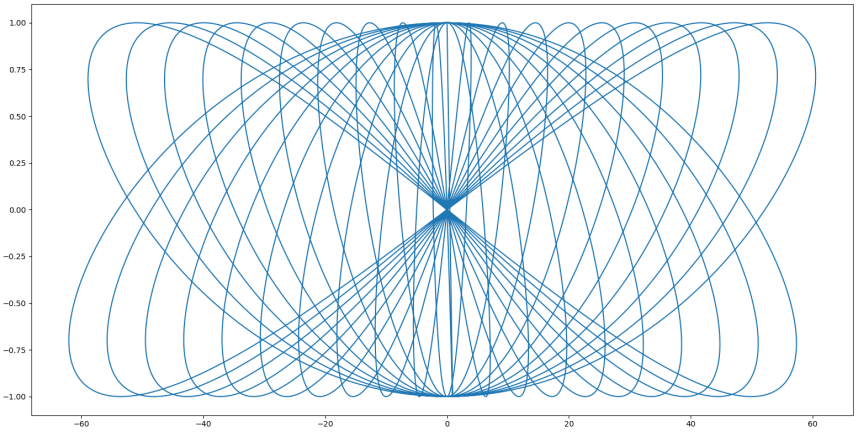

Figure 1.1: An illustration of Example 1.10.

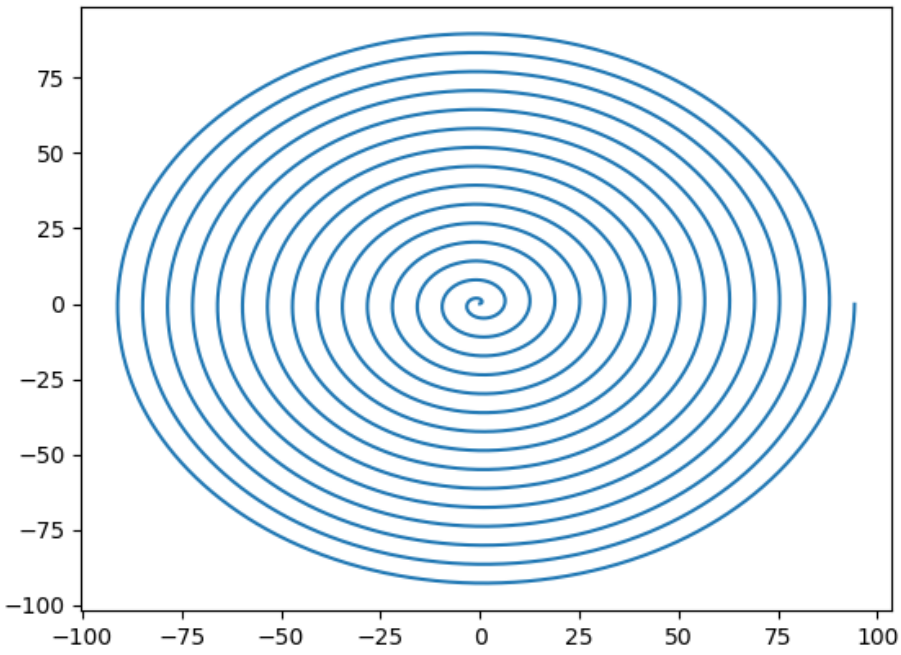

Figure 1.2: An illustration of Example 1.11.

satisfy

$$F^{(k)}(b) = G^{(k)}(b) \qquad (k = 0, 1, \ldots, n).$$

A composite curve

$$H : [a, c] \to \mathbb{E}^d$$

given by the formula

$$H(t) := \begin{cases} F(t) & (t \in [a,b)), \\ G(t) & (t \in [b,c]) \end{cases}$$

contains a joining of the curves F and G of the C^n class.

Example 1.13. *A composite curve $F\colon [0,3] \to \mathbb{E}^2$ is given by*

$$F(t) := \begin{cases} \left(-\frac{16089}{4}t^3 + \frac{26437}{4}t^2 - 1882t + 315, -\frac{12027}{4}t^3 + \frac{14255}{4}t^2 - 453t - 1215\right) & (t \in [0,1]), \\ \left(1129t^3 - 5514t^2 + 8288t - 2883, -1849t^3 + 8250t^2 - 10812t + 3300\right) & (t \in [1,2]), \\ \left(-1394t^3 + 9624t^2 - 21988t + 17301, 1745t^3 - 13314t^2 + 32316t - 25452\right) & (t \in [2,3]). \end{cases}$$

Figure 1.3 illustrates this case.

Example 1.14. *A segment with ends at points $A, B \in \mathbb{E}^2$ can be expressed as a parametric curve $F\colon [0,1] \to \mathbb{E}^2$ given by*

$$F(t) := (1-t)A + tB.$$

In Example 1.14, the formula for a parametric curve has been expressed using a convex combination of points in $\mathbb{E}^2$. This is a common approach when defining the parametric curves.

Definition 1.15. *Let $a, b \in \mathbb{R}$ and $a \leq b$. For $W_0, W_1, \ldots, W_n \in \mathbb{E}^d$ and continuous functions $b_0, b_1, b_2, \ldots, b_n : [a,b] \to \mathbb{R}$ such that $\sum_{k=0}^{n} b_k(t) \equiv 1$ for all $t \in [a,b]$, the function*

$$F(t) := \sum_{k=0}^{n} b_k(t) W_k \qquad (t \in [a,b])$$

is a parametric curve. The points $W_0, W_1, \ldots, W_n$ are the control points *of the curve F and $b_0, b_1, \ldots, b_n$ are its* basis functions*.*

Usually, the basis functions are selected so that $b_0(a) = b_n(b) = 1$ and $b_k(t) \geq 0$ for all $k = 0, 1, \ldots, n$ and $t \in [a,b]$. The former implies that $\mathsf{F}(a) = \mathsf{W}_0$ and $\mathsf{F}(b) = \mathsf{W}_n$, while the latter guarantees that

$$\mathsf{F}(t) \in \operatorname{conv}\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n\} \qquad (t \in [a,b]).$$

Example 1.16. *Let $A, B, C \in \mathbb{E}^d$. The function*

$$F(t) := (1-t)A + t\sin^2(2\pi t)B + t\cos^2(2\pi t)C \qquad (t \in [0,1])$$

is a parametric curve with control points A, B, C and their corresponding basis functions $1 - t, t\sin^2(2\pi t), t\cos^2(2\pi t)$.

Figure 1.4 illustrates this case for

$$A = (3,4)\,, B = (-2,7)\,, C = (0,-3)\,.$$

Note that for $t \in [0,1]$, all points on a curve stay within the convex hull of A, B, C.

In particular, the basis functions of a curve can be defined using polynomials or rational functions. A well-known example is the family of polynomial or rational Bézier curves and B-spline curves, which will be presented in more detail in Sections 1.6 and 1.9, respectively.

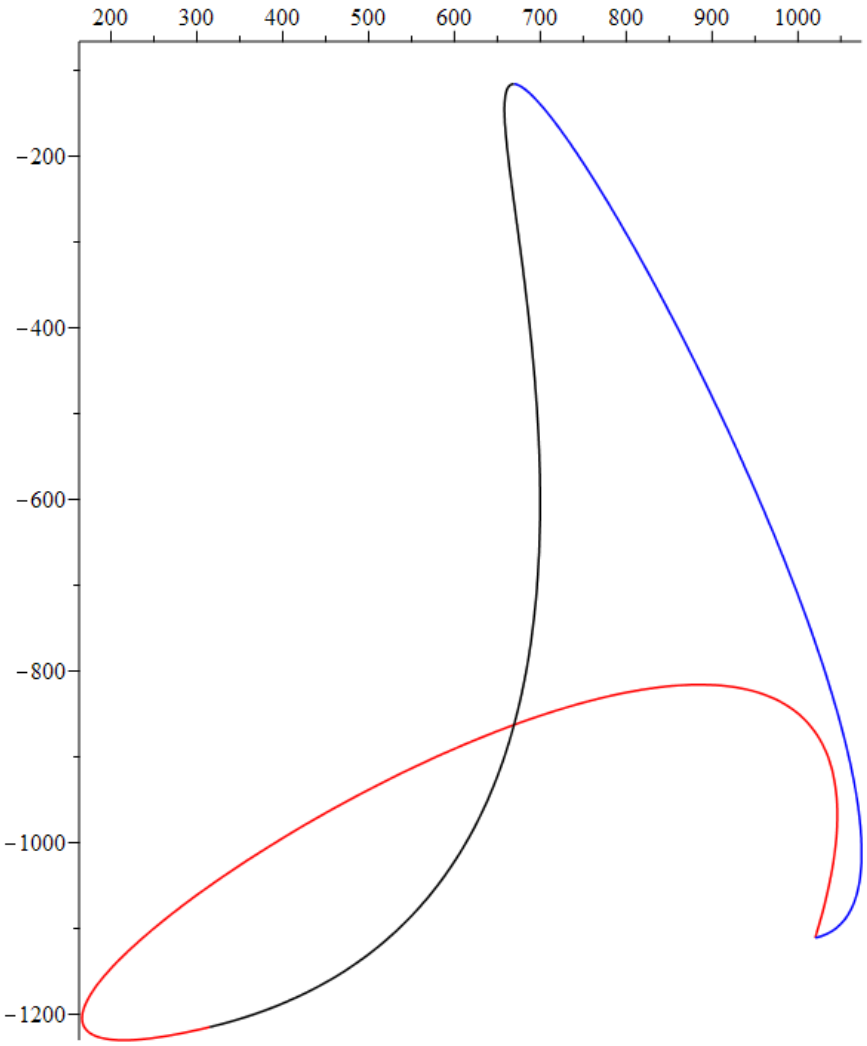

Figure 1.3: An illustration of Example 1.13. Note the different types of joinings: C^0 at $[1020, -1111]^T$, C^1 at $[315, -1215]^T$, and C^2 at $[669, -116]^T$.

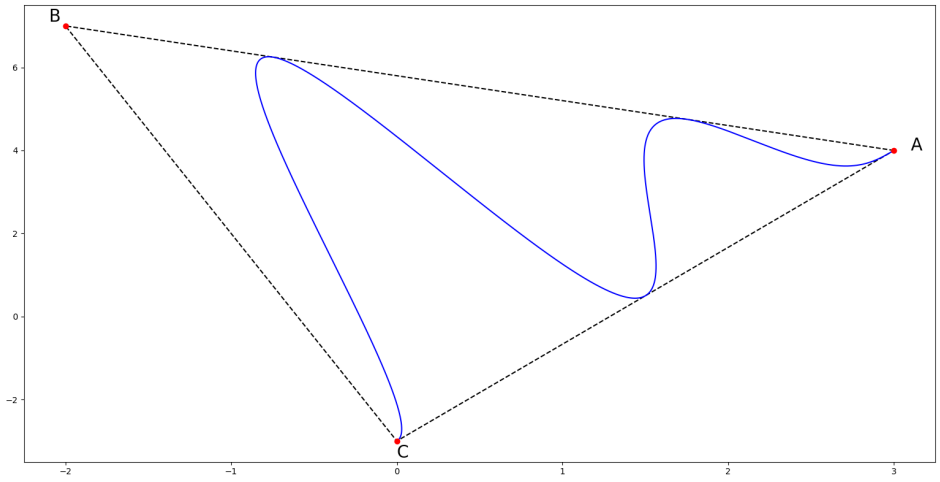

Figure 1.4: An illustration of Example 1.16.

1.3 Hypergeometric functions

Many special functions or orthogonal polynomials which find their applications in approximation theory or numerical analysis can be expressed as a function from the hypergeometric functions family, i.e., in their *hypergeometric form*.

Hypergeometric functions are themselves considered special functions and they satisfy a multitude of identities, some of which will be used to prove new results in the subsequent chapters. Some of the functions which appear throughout the thesis can be expressed in their *hypergeometric form*, which will be used to prove some of their properties.

First, let us introduce the Pochhammer symbol, which is frequently used when dealing with hypergeometric functions.

Definition 1.17 ([3, Eq. (1.1.2)]). *The* Pochhammer symbol $(x)_n$ *is defined for any* $x \in \mathbb{C}$ *and* $n \in \mathbb{N}$ *in a following way:*

$$(x)_0 := 1, \qquad (x)_n := \prod_{k=0}^{n-1}(x+k) \qquad (k \in \mathbb{N} \setminus \{0\}).$$

The Pochhammer symbol is a generalization of a factorial. One can easily see that

$$k! = (1)_k \qquad (k \in \mathbb{N}).$$

The value of $(x)_n$ can be expressed, if $x, x+n \notin \mathbb{Z} \setminus \mathbb{N}$, as a ratio of two gamma functions (cf. [25]):

$$(x)_n = \frac{\Gamma(x+n)}{\Gamma(x)}.$$

This, in particular, means that for $k \in \mathbb{N} \setminus \{0\}$,

$$(k)_n = \frac{(k+n-1)!}{(k-1)!}.$$

Using gamma functions, the binomial coefficient can be generalized so that it has two real arguments:

$$\binom{x}{y} = \frac{\Gamma(x+1)}{\Gamma(y+1)\Gamma(x-y+1)}.$$

In particular, if $n \in \mathbb{N}$, the relation simplifies to

$$\binom{x}{n} = \frac{(x-n+1)_n}{n!}. \tag{1.1}$$

The Pochhammer symbol is useful in expressing hypergeometric functions in a concise fashion.

Definition 1.18 ([3, §2.1]). *A* generalized hypergeometric function ${}_pF_q$ *is defined by*

$${}_pF_q\left(\begin{matrix} a_1, a_2, \ldots, a_p \\ b_1, b_2, \ldots, b_q \end{matrix} \middle| x\right) := \sum_{k=0}^{\infty} \frac{(a_1)_k (a_2)_k \ldots (a_p)_k}{(b_1)_k (b_2)_k \ldots (b_q)_k} \cdot \frac{x^k}{k!},$$

where $p, q \in \mathbb{N}$, $a_i \in \mathbb{C}$ $(i = 1, 2, \ldots, p)$, $b_j \in \mathbb{C}$ $(j = 1, 2, \ldots, q)$, $x \in \mathbb{C}$.

Example 1.19 ([3, §2])**.** *Some classical functions can be expressed in the hypergeometric form. For example:*

$$
\begin{aligned}
e^x &= \sum_{k=0}^{\infty} \frac{x^k}{k!} = {}_0F_0\left(\begin{array}{c} - \\ - \end{array} \middle| x \right), \\
\ln(1+x) &= \sum_{k=0}^{\infty} \frac{(-1)^k x^{k+1}}{k+1} = x\,{}_2F_1\left(\begin{array}{c} 1,1 \\ 2 \end{array} \middle| -x \right) \qquad (|x|<1), \\
\sin x &= \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!} = x\,{}_0F_1\left(\begin{array}{c} - \\ 3/2 \end{array} \middle| -x^2/4 \right), \\
\cos x &= \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} = {}_0F_1\left(\begin{array}{c} - \\ 1/2 \end{array} \middle| -x^2/4 \right).
\end{aligned}
$$

If any of the upper parameters is a non-positive integer then the series is finite and is a polynomial in x.

Example 1.20.

$$
{}_2F_1\left(\begin{array}{c} -2, a_2 \\ b_1 \end{array} \middle| x \right) = \sum_{k=0}^{2} \frac{(-2)_k (a_2)_k}{(b_1)_k k!} x^k = 1 + \frac{-2a_2}{b_1} x + \frac{(-2)(-1)a_2(a_2+1)}{(b_1)(b_1+1)2!} x^2.
$$

The hypergeometric functions used in this thesis will be, in fact, polynomials. The hypergeometric representation of polynomials is useful due to the fact that there is a sizable amount of known hypergeometric identities. A long, but by no means exhaustive, list can be found in [3]. A particular hypergeometric identity which will find its application in proving some properties of dual Bernstein polynomials (cf. §1.5.2) in Chapter 6 is the Chu-Vandermonde identity.

Theorem 1.21 ([3, Corollary 2.2.3])**.** *For $n \in \mathbb{N}$, and $a, b, c \in \mathbb{R}$ such that $c - a + n > 0$, the following Chu-Vandermonde identity holds:*

$$
{}_2F_1\left(\begin{array}{c} -n, a \\ c \end{array} \middle| 1 \right) = \frac{(c-a)_n}{(c)_n}. \tag{1.2}
$$

1.3.1 The Zeilberger's algorithm

There is a wide array of algorithms which allow to discover and prove new identities, with hypergeometric identities among them. The best-known algorithms are presented, e.g., in [57, 75].

In this thesis, the Zeilberger's algorithm will be used to prove one identity in Chapter 4. The main idea of the algorithm is to construct a recurrence relation for $f(n)$ such that

$$
f(n) := \sum_{k \in \mathbb{Z}} F(n,k),
$$

where $F(n,k)$ is a given hypergeometric term, i.e. $F(n+1,k)/F(n,k)$ and $F(n,k+1)/F(n,k)$ are both rational functions of n and k.

Let us set $J \geq 1$. The Zeilberger's algorithm finds polynomials $a_0, a_1, \ldots, a_J$ in n which are also independent of k and a function $G(n,k)$ such that $\frac{G(n,k)}{F(n,k)}$ is a rational function of n, k, so that a relation

$$\sum_{j=0}^{J} a_j(n) F(n+j,k) = G(n,k+1) - G(n,k)$$

is satisfied, or proves that this relation cannot be satisfied for the chosen J.

After applying the sum over k to both sides, one gets

$$\sum_{j=0}^{J} a_j(n) f(n+j) = \sum_{k\in\mathbb{Z}} G(n,k+1) - \sum_{k\in\mathbb{Z}} G(n,k) = 0.$$

This gives a recurrence relation for f which can be used to efficiently compute $f(n)$. In some cases, such as $J = 1$ or $a_j(n)$ being constant (with respect to n) for low J, solving the recurrence relation is simple.

A big advantage of the Zeilberger's algorithm is that the process can be done almost automatically and efficiently realized in symbolic computing environments, e.g., Maple™ (cf. [71]). For more details regarding the Zeilberger's algorithm, see [75, §6], [57, §7], [100, 101].

Example 1.22. *The Chu-Vandermonde identity can be proved using the Zeilberger's algorithm in the following way. Let*

$$f(n) := {}_2F_1\left(\begin{matrix} -n, a \\ c \end{matrix} \,\middle|\, 1 \right) = \sum_{k=0}^{\infty} \frac{(-n)_k (a)_k}{(c)_k k!} \equiv \sum_{k=0}^{\infty} F(n,k).$$

Since $(-n)_k = 0$ and, in consequence, $F(n,k) = 0$ for $k > n$, k only takes natural values from 0 to n in the sums, giving

$$f(n) := \sum_{k=0}^{n} F(n,k).$$

Let us set $J := 1$. Then, one seeks such $a_0(n), a_1(n), G(n,k)$ that the relation

$$a_0(n) F(n,k) + a_1(n) F(n+1,k) = G(n,k+1) - G(n,k)$$

holds. Applying the Zeilberger's algorithm gives

$$\begin{aligned} &a_0(n) := c + n - a, \qquad a_1(n) := -(c+n), \\ &G(n,k) := (c+k-1) \cdot k \cdot \frac{(c+n-a)F(n,k) - (c+n)F(n+1,k)}{(c+n-a)(n+1-k) - (c+n)(n+1)}. \end{aligned}$$

Certainly, $G(n,0) = G(n,n+2) = 0$. Applying the sum over $k = 0, 1, \ldots, n+1$, would give

$$a_0(n) \sum_{k=0}^{n+1} F(n,k) + a_1(n) \sum_{k=0}^{n+1} F(n+1,k) = G(n,n+2) - G(n,0) = 0,$$

thus

$$(c+n-a) f(n) - (c+n) f(n+1) = 0.$$

This leaves the first-order homogeneous recurrence relation

$$f(n+1) = \frac{c+n-a}{c+n} f(n),$$

which, for $f(0) = 1$*, has the explicit solution*

$$f(n) = \frac{(c-a)_n}{(c)_n}.$$

See Theorem 1.21.

1.4 Orthogonal and dual bases

Orthogonality is an extremely useful concept in least-squares approximation. If a basis of space S is known and if the basis is orthogonal, one can easily find an optimal least-square approximation in S of objects from the space T such that $S \subseteq T$. In this thesis, the space S will always be a space of polynomials of degree at most n (denoted as Π_n). In §1.4.1, the concepts of orthogonal bases and orthogonal projection will be introduced, along with two families of orthogonal polynomials — Jacobi and Hahn polynomials — which will find their applications in this thesis.

The concept of orthogonal bases can be generalized to get dual bases. This allows to use a primary basis of S together with its associated dual basis.

1.4.1 Orthogonal bases and orthogonal polynomials

Definition 1.23. *Let* $B = \{b_0, b_1, \ldots, b_n\}$ *be a basis of* S*. Let* $\langle \cdot, \cdot \rangle : S \times S \to \mathbb{R}$ *be a scalar product in* S*.* B *is an* orthogonal basis *of* S *if, for* $j \neq k$*,* $\langle b_j, b_k \rangle = 0$ *and* $\langle b_j, b_j \rangle > 0$*. If, additionally,* $\langle b_j, b_j \rangle = 1$*, the basis is also called* orthonormal*.*

Theorem 1.24 ([22, Theorem 4.5.13])**.** *Let* $\{b_0, b_1, \ldots, b_n\}$ *be an orthogonal basis of* S*. Let* $\langle \cdot, \cdot \rangle : S \times S \to \mathbb{R}$ *be a scalar product in* S*. Let* $\| \cdot \|$ *be a norm associated with the scalar product* $\langle \cdot, \cdot \rangle$*, i.e.,* $\|f\|^2 := \langle f, f \rangle$*.*

Then

$$\min_{w \in S} \|f - w\| = \left\| f - \sum_{k=0}^{n} \frac{\langle f, b_k \rangle}{\langle b_k, b_k \rangle} b_k \right\|.$$

In particular, if $f \in S$*,*

$$f = \sum_{k=0}^{n} \frac{\langle f, b_k \rangle}{\langle b_k, b_k \rangle} b_k.$$

Often, due to their efficiency and applications in approximation, orthogonal polynomial bases are used.

Theorem 1.25 ([22, Theorem 4.5.19])**.** *For every scalar product* $\langle \cdot, \cdot \rangle$*, there is a family of orthogonal polynomials* p_k *(*$k = 0, 1, \ldots$*) such that* p_k *has exactly degree* k *and is orthogonal to polynomials of degree less than* k*. The family is uniquely determined apart from the fact that the leading coefficients can be given arbitrary positive values.*

The polynomials from the family p_k ($k = 0, 1, \ldots$) satisfy the recurrence relation of the form

$$p_{k+1}(x) = \left(x - \frac{\langle xp_k, p_k \rangle}{\langle p_k, p_k \rangle} \right) p_k(x) - \frac{\langle p_k, p_k \rangle}{\langle p_{k-1}, p_{k-1} \rangle} p_{k-1}(x) \qquad (k \geq 1),$$

with initial values $p_0(x) = 1$ and $p_1(x) = x - \dfrac{\langle x, 1 \rangle}{\langle 1, 1 \rangle}$.

Jacobi polynomials

The Jacobi polynomials are orthogonal with respect to the scalar product

$$\int_{-1}^{1} (1-x)^\alpha (1+x)^\beta f(x) g(x) \mathrm{d}x, \tag{1.3}$$

where the parameters $\alpha, \beta \in \mathbb{R}$ satisfy $\alpha, \beta > -1$. The Jacobi polynomials, certainly, satisfy Theorem 1.25. For the reader's convenience, an explicit recurrence relation is given.

Theorem 1.26. *Let $P_n^{(\alpha,\beta)}$ be the nth Jacobi polynomial with parameters α, β. The following recurrence relation holds:*

$$\phi_2(n) P_n^{(\alpha,\beta)}(x) = \phi_1(n) P_{n-1}^{(\alpha,\beta)}(x) - \phi_0(n) P_{n-2}^{(\alpha,\beta)}(x) \qquad (n \geq 2), \tag{1.4}$$

where

$$\begin{aligned} \phi_0(n) &:= 2(n+\alpha-1)(n+\beta-1)(2n+\alpha+\beta), \\ \phi_1(n) &:= (2n+\alpha+\beta-1), [(2n+\alpha+\beta)(2n+\alpha+\beta-2)x + \alpha^2 - \beta^2], \\ \phi_2(n) &:= 2n(n+\alpha+\beta)(2n+\alpha+\beta-2). \end{aligned}$$

Additionally, a hypergeometric representation (see §1.3) of Jacobi polynomials is known:

$$P_n^{(\alpha,\beta)}(x) = \frac{(\alpha+1)_n}{n!} \, {}_2F_1\left(\begin{matrix} -n, n+\alpha+\beta+1 \\ \alpha+1 \end{matrix} \,\middle|\, \frac{1-x}{2} \right) \qquad (n = 0, 1, \ldots) \tag{1.5}$$

(cf. [3, Definition 2.5.1]).

Some parameter choices for α, β give well-known families of orthogonal polynomials. The recurrence relations for them can be derived from Eq. (1.4). When $\alpha = \beta = 0$, i.e., the considered scalar product simplifies to

$$\int_{-1}^{1} f(x) g(x) \mathrm{d}x,$$

then

$$P_n^{(0,0)}(x) \equiv L_n(x),$$

where L_n is the nth Legendre polynomial (cf. [58, Table 18.3.1]). A hypergeometric form of Legendre polynomials follows from Eq. (1.5):

$$L_n(x) = {}_2F_1\left(\begin{matrix} -n, n+1 \\ 1 \end{matrix} \,\middle|\, \frac{1-x}{2} \right) \qquad (n = 0, 1, \ldots).$$

Another well-established subtype of Jacobi polynomials are Chebyshev polynomials T_n (cf. [58, Table 18.3.1]):

$$T_n \equiv 2^{2n}\binom{2n}{n}^{-1} P_n^{(-1/2,-1/2)}.$$

Similarly to the Legendre polynomials, a hypergeometric form of Chebyshev polynomials is known:

$$T_n(x) = {}_2F_1\left(\begin{array}{c} -n, n \\ 1/2 \end{array} \middle| \frac{1-x}{2}\right) \qquad (n = 0, 1, \ldots).$$

Shifted Jacobi polynomials

In this thesis, shifted Jacobi polynomials $R_n^{(\alpha,\beta)}$ will be of particular interest. They can be obtained by using the relation

$$R_n^{(\alpha,\beta)}(x) = P_n^{(\alpha,\beta)}(2x-1) \qquad (n = 0, 1, \ldots).$$

Certainly, then, the shifted Jacobi polynomials are orthogonal with respect to the scalar product

$$\langle f, g\rangle_{\alpha,\beta} = \int_0^1 (1-x)^\alpha x^\beta f(x)g(x)\mathrm{d}x \tag{1.6}$$

(cf. (1.3)). More precisely,

$$\left\langle R_k^{(\alpha,\beta)}, R_l^{(\alpha,\beta)}\right\rangle_{\alpha,\beta} = \delta_{kl}h_k \qquad (k, l \in \mathbb{N}), \tag{1.7}$$

where δ_{kl} is the *Kronecker delta* ($\delta_{kl} = 0$ for $k \neq l$ and $\delta_{kk} = 1$) and

$$h_k := K\,\frac{(\alpha+1)_k(\beta+1)_k}{k!(2k/\sigma+1)(\sigma)_k} \qquad (k = 0, 1, \ldots)$$

with

$$\sigma := \alpha + \beta + 1 \tag{1.8}$$

and

$$K \equiv K_{\alpha,\beta} := \Gamma(\alpha+1)\Gamma(\beta+1)/\Gamma(\sigma+1). \tag{1.9}$$

A version of Theorem 1.26 is given for shifted Jacobi polynomials.

Theorem 1.27 ([56, §1.8])**.** *Shifted Jacobi polynomials satisfy the second-order recurrence relation of the form*

$$\xi_0(n)R_n^{(\alpha,\beta)}(x) + \xi_1(n)R_{n+1}^{(\alpha,\beta)}(x) + \xi_2(n)R_{n+2}^{(\alpha,\beta)}(x) = 0 \qquad (n = 0, 1, \ldots), \tag{1.10}$$

where

$$\xi_0(n) := -2(n+\alpha+1)(n+\beta+1)(2n+\sigma+3), \tag{1.11}$$
$$\xi_1(n) := (2n+\sigma+2)\{(2n+\sigma+1)(2n+\sigma+3)(2x-1) + \alpha^2 - \beta^2\}, \tag{1.12}$$
$$\xi_2(n) := -2(n+2)(n+\sigma+1)(2n+\sigma+1) \tag{1.13}$$

(cf. (1.8)*).*

Remark 1.28. *The recurrence relation* (1.10) *can be used, for example, in fast and accurate methods for evaluating the values* $R_n^{(\alpha,\beta)}(x)$ *for a given* x, α, β *and all* $0 \le n \le N$*, where* N *is a fixed natural number, with* $O(N)$ *computational complexity. For more details about performing computations with recurrence relations properly, see [93].*

The representation of shifted Jacobi polynomials in the $(1-x)^j$ basis is given in the hypergeometric form as

$$R_n^{(\alpha,\beta)}(x) = \frac{(\alpha+1)_n}{n!}\,{}_2F_1\left(\begin{matrix}-n,\, n+\alpha+\beta+1 \\ \alpha+1\end{matrix}\,\middle|\, 1-x\right) \qquad (n=0,1,\ldots) \tag{1.14}$$

(cf., e.g., [56, §1.8]).

Theorem 1.29 ([3, p. 117])**.** *Shifted Jacobi polynomials satisfy the symmetry relation*

$$R_n^{(\alpha,\beta)}(x) = (-1)^n R_n^{(\beta,\alpha)}(1-x) \tag{1.15}$$

for $n \in \mathbb{N}$ *and* $\alpha, \beta > -1$*.*

Theorem 1.30 ([3, Eq. (6.3.8)])**.** *Shifted Jacobi polynomials satisfy the relation:*

$$\left(R_n^{(\alpha,\beta)}(x)\right)' = (n+\alpha+\beta+1) R_{n-1}^{(\alpha+1,\beta+1)}(x). \tag{1.16}$$

for $n \in \mathbb{N}$ *and* $\alpha, \beta > -1$*.*

Theorem 1.31 ([3, Eq. (6.4.20) and (6.4.23)])**.** *Shifted Jacobi polynomials satisfy the relations:*

$$(1-x)R_n^{(\alpha+1,\beta)}(x) = -\frac{n+1}{2n+\sigma+1}R_{n+1}^{(\alpha,\beta)}(x) + \frac{n+\alpha+1}{2n+\sigma+1}R_n^{(\alpha,\beta)}(x), \tag{1.17}$$

$$xR_n^{(\alpha,\beta+1)}(x) = \frac{n+1}{2n+\sigma+1}R_{n+1}^{(\alpha,\beta)}(x) + \frac{n+\beta+1}{2n+\sigma+1}R_n^{(\alpha,\beta)}(x), \tag{1.18}$$

for $n \in \mathbb{N}$*,* $\alpha, \beta > -1$ *and* σ *given by* (1.8)*.*

Theorem 1.32 ([56, Eq. (1.8.5)])**.** *Shifted Jacobi polynomials satisfy the second-order differential equation with polynomial coefficients of the form:*

$$\boldsymbol{L}^{(\alpha,\beta)} R_k^{(\alpha,\beta)}(x) = \lambda_k^{(\alpha,\beta)} R_k^{(\alpha,\beta)}(x) \qquad (k=0,1,\ldots), \tag{1.19}$$

where

$$\boldsymbol{L}^{(\alpha,\beta)} := x(x-1)\boldsymbol{D}^2 + \tfrac{1}{2}\left(\alpha-\beta+(\sigma+1)(2x-1)\right)\boldsymbol{D}, \qquad \lambda_k^{(\alpha,\beta)} := k(k+\sigma),$$

and $\boldsymbol{D} := \dfrac{d}{dx}$ *is a differentiation operator with respect to the variable* x*.*

For more properties and applications of Jacobi polynomials, see, e.g., [3, 56].

Hahn polynomials

Another family of orthogonal polynomials which will be used in this thesis are Hahn polynomials.

Definition 1.33 ([56, §1.5])**.** *The kth Hahn polynomial with parameters $\alpha, \beta > -1$ and $N \geq k$ is given in the hypergeometric form*

$$Q_k(x;\alpha,\beta;N) := {}_3F_2\left(\begin{matrix} -k,\ k+\alpha+\beta+1,\ -x \\ \alpha+1,\ -N \end{matrix} \middle| 1 \right) \qquad (k=0,1,\ldots,N;\ N\in\mathbb{N}) \quad (1.20)$$

(see §1.3).

Theorem 1.34 ([97, Eq. (2.4)], [56, §1.5])**.** *Hahn polynomials are orthogonal with respect to the scalar product*

$$\langle f,g\rangle_H := \sum_{x=0}^{N} \binom{\alpha+x}{x}\binom{\beta+N-x}{N-x} f(x)g(x) \qquad (\alpha,\beta>-1)$$

(cf. (1.1)*), i.e.,*

$$\langle Q_k, Q_\ell\rangle_H = \delta_{k\ell} h_k^{(\alpha,\beta,N)} \qquad (0\leq k,\ell\leq N)$$

for some positive $h_k^{(\alpha,\beta,N)}$.

Theorem 1.35 ([32, p. 9],[53, Eq. (1.15)])**.** *Hahn polynomials satisfy a symmetry relation:*

$$(\alpha+1)_k Q_k(x;\alpha,\beta,N) = (-1)^k(\beta+1)_k Q_k(N-x;\beta,\alpha,N). \quad (1.21)$$

A difference equation is known for Hahn polynomials.

Theorem 1.36 ([56, Eq. (1.5.5)], [56, §1.5], [97, Eq. (A.17)])**.** *Hahn polynomials satisfy the second-order difference equation with polynomial coefficients of the form*

$$\mathcal{L}_x^{(\alpha,\beta,N)} Q_k(x;\alpha,\beta;N) = \lambda_k^{(\alpha,\beta)} Q_k(x;\alpha,\beta;N) \qquad (k=0,1,\ldots), \quad (1.22)$$

where

$$\mathcal{L}_x^{(\alpha,\beta,N)} f(x) := a(x)f(x+1) - c(x)f(x) + b(x)f(x-1), \quad (1.23)$$

and

$$a(x) := (x-N)(x+\alpha+1), \qquad b(x) := x(x-\beta-N-1), \qquad c(x) := a(x)+b(x).$$

1.4.2 Dual bases

While the orthogonal projection is certainly useful as a tool for least-square approximation, it has significant drawbacks. Often, when solving an approximation problem, the solution is needed in a specific (and usually not orthogonal) basis. In order to use the orthogonal projection, then, one needs to find the solution's representation in the orthogonal basis and then change the basis to the desired one. Changing the basis is, however, computationally costly and can be numerically unstable.

Dual bases allow to perform the approximation directly in the desired basis, which reduces the numerical risks and can speed up the computations. Figure 1.5 presents the difference between using dual bases and the orthogonal projection.

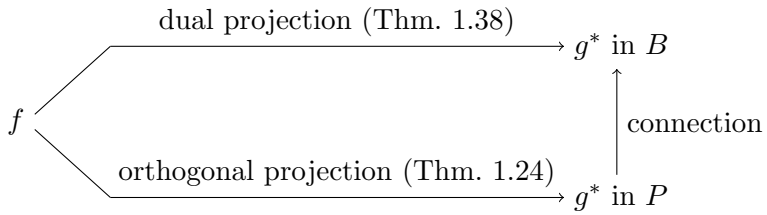

Figure 1.5: A diagram illustrating two approaches to solving the following problem: for given functions f, a space S and inner product $\langle\cdot,\cdot\rangle$ (with P being the orthogonal basis of S with respect to $\langle\cdot,\cdot\rangle$), find the representation of $g^* \in S$ in the basis B (where lin $B = S$) such that $\|f - g^*\| = \min_{g\in S} \|f - g\|$, for $\|\cdot\| := \sqrt{\langle\cdot,\cdot\rangle}$.

Definition 1.37 ([94, Eq. (1.1)])**.** *Let $B = \{b_0, b_1, \ldots, b_n\}$ be a basis of S. Let $\langle\cdot,\cdot\rangle : S \times S \to \mathbb{R}$ be a scalar product in S. Then $D = \{d_0, d_1, \ldots, d_n\}$ is* dual *to B with respect to $\langle\cdot,\cdot\rangle$ if:*

1. *lin B = lin $D = S$,*
2. *$\langle b_k, d_j\rangle = \delta_{kj}$,*

where δ_{kj} is the Kronecker delta ($\delta_{kk} = 1$, if $k \neq j$ then $\delta_{kj} = 0$).

Dual bases are a generalization of orthogonal bases — note that an orthonormal basis is dual to itself. One can also generalize the orthogonal projection (see Theorem 1.24) to find the representation of a function in the primary base.

Theorem 1.38 ([94, §4])**.** *Let $\{b_0, b_1, \ldots, b_n\}$ be a basis of S. Let $\{d_0, d_1, \ldots, d_n\}$ be its dual basis with respect to $\langle\cdot,\cdot\rangle$. Let $\|\cdot\|_2 := \sqrt{\langle\cdot,\cdot\rangle}$. Then*

$$\min_{w\in S} \|f - w\|_2 = \left\| f - \sum_{k=0}^{n} \langle f, d_k\rangle b_k \right\|_2 .$$

In particular, when $f \in S$,

$$f = \sum_{k=0}^{n} \langle f, d_k\rangle b_k.$$

Dual bases have an interesting connection with the orthonormal basis which spans the same space. It is a useful tool for discovering the representation of dual bases.

Theorem 1.39 ([60, Lemma 2.1])**.** *Let $B = \{b_0, b_1, \ldots, b_n\}$, $D = \{d_0, d_1, \ldots, d_n\}$ be two bases of S such that D is dual to B with respect to $\langle\cdot,\cdot\rangle$. Let $P = \{p_0, p_1, \ldots, p_n\}$ be the orthonormal basis of S wrt. the same scalar product. Let c_{ij} ($i, j = 0, 1, \ldots, n$) be defined so that*

$$p_i = \sum_{j=0}^{n} c_{ij} b_j \qquad (0 \leq i \leq n).$$

The elements of D have the following representation in the basis P:

$$d_j = \sum_{i=0}^{n} c_{ij} p_i \qquad (0 \leq j \leq n).$$

Corollary 1.40. *From Theorem 1.39, it follows that*

$$d_j = \sum_{k=0}^{n} \Big(\sum_{i=0}^{n} c_{ij} c_{ik} \Big) b_k.$$

Some more general properties of dual bases can be found in [44, 94, 95]. Additional properties for dual polynomial bases are proposed in [42]. In this thesis, the main dual basis that will be considered is the dual Bernstein polynomial basis. It will be introduced, along with Bernstein polynomials, in §1.5. A second important dual basis which will be used in this thesis is the dual discrete Bernstein polynomial basis (cf. §1.5.4).

Other dual bases are also considered in the literature, such as, for example, dual B-spline functions ([95]) and functionals ([49]), dual Wang-Bézier and dual Bézier-Said-Wang type generalized Ball polynomials ([4, 102, 103, 105]), dual NS-power bases ([104]), bivariate dual Bernstein polynomials ([64, 98]), dual tensor product Bernstein polynomials ([62]).

Constructing a dual basis from the definition

One approach to construct a dual basis (see, e.g., [94, §2]) applies Definition 1.37. Let $B = \{b_0, b_1, \ldots, b_n\}$, $D = \{d_0, d_1, \ldots, d_n\}$ be two bases of S such that D is dual to B with respect to $\langle \cdot, \cdot \rangle$. Let a_{ij} $(i, j = 0, 1, \ldots, n)$ be the coefficients of the representation of d_i in the basis B, i.e.,

$$d_i = \sum_{j=0}^{n} a_{ij} b_j \qquad (i = 0, 1, \ldots, n).$$

The conditions in Definition 1.37 give $(n+1)^2$ equations

$$\langle d_i, b_k \rangle = \sum_{j=0}^{n} a_{ij} = \langle b_j, b_k \rangle = \delta_{jk} \qquad (i, k = 0, 1, \ldots, n),$$

which can be expressed in the matrix form

$$G \times A = I \qquad (G := [g_{ij}], A := [a_{ij}]; \quad G, A \in \mathbb{R}^{(n+1)\times(n+1)}),$$

where $g_{ij} := \langle b_i, b_j \rangle$ and I is the identity matrix. The matrix G is the so-called Gram matrix and, clearly, A is its inverse.

While this method is very simple, its high complexity and numerical risks of inverting the Gram matrix are its significant drawbacks. One thus needs other methods for dual basis construction.

Constructing a dual basis using recurrence relations

Some methods to construct the dual basis have been given by Woźny in [94, 95]. The method described here is the one which was proposed in [95] (with amendments given in [44]).

Let $B_n := \{b_0, b_1, \ldots, b_n\}$ be a basis of the space S_n. Let a known basis

$$D_n := \{d_0^{(n)}, d_1^{(n)}, \ldots, d_n^{(n)}\}$$

be dual to B_n with respect to $\langle \cdot, \cdot \rangle$. Now, let $B_{n+1} := B_n \cup \{b_{n+1}\}$ such that B_{n+1} is the basis of the space S_{n+1}. Let $D_{n+1} := \left\{ d_0^{(n+1)}, d_1^{(n+1)}, \ldots, d_{n+1}^{(n+1)} \right\}$ be dual to B_{n+1} with respect to the same scalar product. In order to construct D_{n+1} from the elements of B_{n+1} and D_n, a recurrence relation can be used.

Theorem 1.41 ([95, Theorem 2.3]). *The elements of the dual bases D_n and D_{n+1} satisfy the relations*

$$d_j^{(n+1)} = d_j^{(n)} - w_j^{(n+1)} d_{n+1}^{(n+1)} \qquad (i = 0, 1, \ldots, n),$$

where $w_j^{(n+1)} := \langle d_j^{(n)}, b_{n+1} \rangle$.

In order to find $d_{n+1}^{(n+1)}$, one can express it in the basis $D_n \cup \{b_{n+1}\}$, which certainly spans S_{n+1} because D_n and B_n span exactly the same space:

$$d_{n+1}^{(n+1)} = \sum_{k=0}^{n} c_k^{(n+1)} d_k^{(n)} + c_{n+1}^{(n+1)} b_{n+1}.$$

After applying the orthogonality conditions from Definition 1.37, one gets $n+1$ equations of the form

$$\begin{cases} \langle d_{n+1}^{(n+1)}, b_i \rangle = c_i^{(n+1)} + c_{n+1}^{(n+1)} \langle b_{n+1}, b_i \rangle = 0 & (i = 0, 1, \ldots, n), \\ \langle d_{n+1}^{(n+1)}, b_{n+1} \rangle = \sum_{k=0}^{n} c_k^{(n+1)} \langle d_k^{(n)}, b_{n+1} \rangle + c_{n+1}^{(n+1)} \langle b_{n+1}, b_{n+1} \rangle = 1. \end{cases}$$

These equations have a simple solution:

$$\begin{cases} c_i^{(n+1)} = -c_{n+1}^{(n+1)} \langle b_{n+1}, b_i \rangle & (i = 0, 1, \ldots, n), \\ c_{n+1}^{(n+1)} = \left(\langle b_{n+1}, b_{n+1} \rangle - \sum_{k=0}^{n} \langle b_{n+1}, b_k \rangle \langle d_k^{(n)}, b_{n+1} \rangle \right)^{-1}. \end{cases}$$

In some particular applications, such as in degree reduction of Bézier curves with box constraints (see [44]), sometimes it is desirable to reduce the size of the basis, i.e., when the basis D_{n+1} (dual to B_{n+1}) is known, one needs to find D_n (dual to B_n).

Theorem 1.42 ([44, Theorem 4.3]). *The following relations between the elements of D_n and D_{n+1} hold:*

$$d_i^{(n)} = d_i^{(n+1)} - \frac{\left\langle d_i^{(n+1)}, d_{n+1}^{(n+1)} \right\rangle}{\left\langle d_{n+1}^{(n+1)}, d_{n+1}^{(n+1)} \right\rangle} d_{n+1}^{(n+1)} \qquad (i = 0, 1, \ldots, n).$$

1.5 Bernstein polynomials, dual Bernstein polynomials and their properties

1.5.1 Bernstein polynomials

The family of Bernstein polynomials was used by S. N. Bernstein in his constructive proof of the Weierstrass approximation theorem, which states that any continuous function can be approximated over a closed and bounded interval with arbitrary precision using polynomials. For more details, see [24, §10.3]. They came into prominence, however, half a century later when they became a basis for a particular family of parametric curves (cf. Section 1.2) — Bézier curves. Section 1.6 describes these objects in more detail.

Definition 1.43. *For a set $n \in \mathbb{N}$ and $i \in \{0, 1, \ldots, n\}$, B_i^n is the* ith Bernstein polynomial of degree n. *B_i^n is given by the formula*

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}. \tag{1.24}$$

An example family of Bernstein polynomials of degree 5 is shown in Figure 1.6. The widespread use of Bernstein polynomials is due to their simplicity and useful properties.

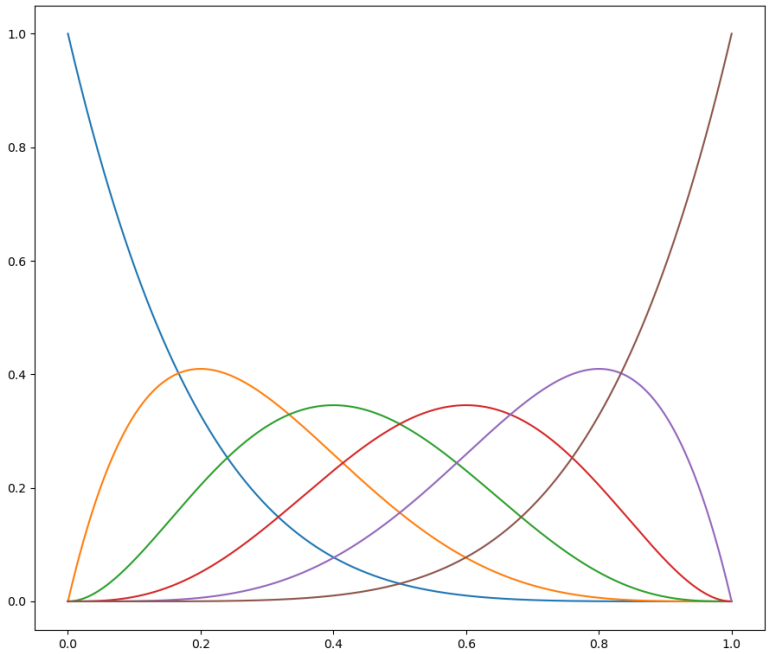

Figure 1.6: The family of Bernstein polynomials of degree 5 in the interval $[0, 1]$.

Remark 1.44. *From Definition 1.43, several properties follow.*

1. *B_k^n has a root with multiplicity k at 0 (for $k = 1, 2, \ldots, n$) and a root with multiplicity $n - k$ at 1 (for $k = 0, 1, \ldots, n-1$). For $t \in (0, 1)$, $B_k^n(t) > 0$ for all $k = 0, 1, \ldots, n$.*
2. *B_k^n has exactly one local maximum in $[0, 1]$ at $\dfrac{k}{n}$.*
3. *Bernstein polynomials satisfy the symmetry relation $B_k^n(t) = B_{n-k}^n(1-t)$.*

Theorem 1.45 ([36, Eq. (6.20) and (6.21)])**.** *The polynomial B_i^n can be expressed as a linear combination of t^k monomials ($k = i, i+1, \ldots, n$):*

$$B_i^n(t) = \sum_{k=i}^{n} (-1)^{i+k} \binom{n}{k} \binom{k}{i} t^k. \tag{1.25}$$

The monomial t^k can be expressed as a linear combination of $B_k^n, B_{k+1}^n, \ldots, B_n^n$:

$$t^k = \binom{n}{k}^{-1} \sum_{i=k}^{n} \binom{i}{k} B_i^n(t). \tag{1.26}$$

Corollary 1.46. *From equations* (1.25) *and* (1.26)*, it follows that Bernstein polynomials of degree n form a basis of Π_n.*

Bernstein polynomials possess the partition of unity property, which is useful in their applications in computer graphics (cf. Section 1.1) and provides greater numerical stability than using a basis without that property (see, e.g., [39]).

Remark 1.47 ([77, P1.1, P1.2])**.** *For any $n \in \mathbb{N}$ and $t \in \mathbb{R}$,*

$$\sum_{i=0}^{n} B_i^n(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = (t+1-t)^n \equiv 1. \tag{1.27}$$

Additionally, if $t \in [0,1]$, then $B_i^n(t) \geq 0$.

In practice, this property means that the values of Bernstein polynomials for a certain t can be used as weights in a barycentric combination (cf. Fact 1.3). If $t \in [0,1]$, this combination is not only barycentric but also convex (cf. Definition 1.7).

Remark 1.48. *In the sequel, the convention is applied that $B_i^n \equiv 0$ if $i < 0$ or $i > n$.*

Theorem 1.49 ([36, Eq. (5.2) and (6.26)])**.** *For any $t \in \mathbb{R}$, Bernstein polynomials satisfy the recurrence relations connecting the polynomials of two consecutive degrees:*

$$B_i^n(t) = tB_{i-1}^{n-1}(t) + (1-t)B_i^{n-1}(t) \qquad (i = 0, 1, \ldots, n), \tag{1.28}$$

$$B_i^n(t) = \frac{n-i+1}{n+1} B_i^{n+1}(t) + \frac{i+1}{n+1} B_{i+1}^{n+1}(t) \qquad (i = 0, 1, \ldots, n). \tag{1.29}$$

The relation (1.29) is known as *the degree elevation formula*.

Theorem 1.50 ([36, Eq. (6.22)])**.** *For $n \in \mathbb{N}$, $k = 0, 1, \ldots, n$ and $c, t \in \mathbb{R}$, Bernstein polynomials satisfy the relation*

$$B_k^n(ct) = \sum_{j=0}^{n} B_k^j(c) B_j^n(t). \tag{1.30}$$

The equation (1.30) will be applied in §1.7.2 in the problem of Bézier curve subdivision.

In this thesis, differential-recurrence relations for Bernstein polynomials will find their application.

Theorem 1.51 ([77, P1.7])**.** *For $n \in \mathbb{N}$ and $k = 0, 1, \ldots, n$, the following differential-recurrence relation for Bernstein polynomials holds:*

$$\Big(B_k^n(t)\Big)' = n\Big(B_{k-1}^{n-1}(t) - B_k^{n-1}(t)\Big). \tag{1.31}$$

Theorem 1.51 can be combined with Definition 1.43 and Eq. (1.29) to get additional relations.

Theorem 1.52. *Let $n \in \mathbb{N}$, $k = 0, 1, \ldots, n$ and $t \in \mathbb{R}$. Bernstein polynomials satisfy the following differential-recurrence relations:*

$$\Big(B_k^n(t)\Big)' = (n-k+1)B_{k-1}^n(t) + (2k-n)B_k^n(t) - (k+1)B_{k+1}^n(t), \tag{1.32}$$

$$t\Big(B_k^n(t)\Big)' = kB_k^n(t) - (k+1)B_{k+1}^n(t). \tag{1.33}$$

Proof. Applying Eq. (1.29) to elevate the degree of the right-hand side of Eq. (1.31) and grouping the elements on the right-hand side proves Eq. (1.32). Eq. (1.33) follows from applying the identity

$$ntB_k^{n-1}(t) = n\binom{n-1}{k}t^{k+1}(1-t)^{n-(k+1)} = (k+1)B_{k+1}^n(t)$$

to Eq. (1.29) twice. □

Bernstein polynomials have a nice connection with two families of orthogonal polynomials, namely shifted Jacobi polynomials and Hahn polynomials (see §1.4.1).

Theorem 1.53 ([60, Eq. (5.4)])**.** *Bernstein polynomials have the following shifted Jacobi form:*

$$B_k^n(x) = \binom{n}{k}(\alpha+1)_{n-k}(\beta+1)_k \sum_{i=0}^{n} \frac{(2i+\sigma)(-n)_i}{(\alpha+1)_i(i+\sigma)_{n+1}} Q_i(k;\beta,\alpha,n)R_i^{(\alpha,\beta)}(x) \tag{1.34}$$

for $k = 0, 1, \ldots, n$.

For more properties, history and applications of Bernstein polynomials, see [38].

1.5.2 Dual Bernstein polynomials

For many years, Bernstein basis polynomials have been used in computer-aided geometric design, approximation theory, numerical analysis and computational mathematics. See, e.g., books [11, 36] and the article [38], as well as the papers cited therein. These applications of Bernstein polynomials can be further expanded if dual Bernstein polynomials are known and can be evaluated efficiently.

Dual Bernstein polynomials associated with the Legendre inner product were introduced by Ciesielski in 1987 [20]. Their properties and generalizations were studied, e.g., by Jüttler [52], Rababah and Al-Natour [80, 81], as well as by Lewanowicz and Woźny [60, 61, 97]. A more general version of dual Bernstein polynomials was introduced in [60]. These polynomials are associated with the shifted Jacobi inner product (cf. (1.6)), which is a generalization of the previously considered Legendre inner product (see p. 13).

Dual Bernstein polynomials appear in the formulas for the coefficients of dual projections into the Bernstein-Bézier basis. For that reason, dual Bernstein polynomials have recently been extensively studied and found many theoretical (see [20, 52, 60, 61, 80]) and practical applications. For example, these dual polynomials are very useful in: curve intersection using Bézier clipping ([7, 66, 90]); degree reduction and merging of Bézier curves ([46, 47, 97, 99]); polynomial approximation of rational Bézier curves ([63]); numerical solving of boundary value problems ([45]) or even fractional partial differential equations ([50, 51]). Skillful use of these

polynomials often results in less costly algorithms of solving many computational problems. More properties of dual Bernstein polynomials and algorithms for their fast evaluation will be given in Chapters 4 and 5.

The inner product with respect to which dual Bernstein polynomials are constructed is the shifted Jacobi inner product (cf. (1.6)), with the parameters $\alpha, \beta > -1$.

Definition 1.54 ([60, §5]). *Let the inner product $\langle \cdot, \cdot \rangle_{\alpha,\beta}$ be given by Eq.* (1.6). Dual Bernstein polynomials of degree n,

$$D_0^n(x;\alpha,\beta),\ D_1^n(x;\alpha,\beta),\ \ldots,\ D_n^n(x;\alpha,\beta) \in \Pi_n,$$

are defined so that the following conditions hold:

$$\left\langle B_i^n, D_j^n(\cdot;\alpha,\beta) \right\rangle_{\alpha,\beta} = \delta_{ij} \qquad (i,j = 0,1,\ldots,n). \tag{1.35}$$

In the case $\alpha = \beta = 0$, these polynomials were introduced earlier by Ciesielski in [20]. One can prove that dual Bernstein polynomials form a basis of the Π_n space.

Remark 1.55. *In the sequel, the convention is adopted that $D_i^n(x;\alpha,\beta) := 0$ for $i < 0$ or $i > n$.*

Definition 1.54 does not give explicit expressions for dual Bernstein polynomials. The expressions for them can be derived, for example, by using Theorems 1.39 and 1.53.

Theorem 1.56 ([60, Theorem 5.2]). *For $i = 0,1,\ldots,n$, one can express dual Bernstein polynomials $D_i^n(x;\alpha,\beta)$ using Hahn polynomials and shifted Jacobi polynomials (cf. Definition 1.33 and Eq.* (1.14)*) as follows:*

$$D_i^n(x;\alpha,\beta) = K^{-1} \sum_{k=0}^{n} (-1)^k \frac{(2k/\sigma+1)(\sigma)_k}{(\alpha+1)_k} Q_k(i;\beta,\alpha;n) R_k^{(\alpha,\beta)}(x), \tag{1.36}$$

with $\sigma := \alpha+\beta+1$ and $K := \Gamma(\alpha+1)\Gamma(\beta+1)/\Gamma(\sigma+1)$.

Remark 1.57 ([60, Corollary 5.3]). *Dual Bernstein polynomials satisfy a symmetry property. From Eq.* (1.36) *and the symmetry properties of Jacobi (see Eq.* (1.15)*) and Hahn (see Eq.* (1.21)*) polynomials, it follows that*

$$D_i^n(x;\alpha,\beta) = D_{n-i}^n(1-x;\beta,\alpha) \qquad (i = 0,1,\ldots,n). \tag{1.37}$$

Theorem 1.58 ([60, Corollary 5.4]). *The polynomial $D_i^n(x;\alpha,\beta)$ can be expressed as a short linear combination of $\min(i, n-i)+1$ shifted Jacobi polynomials with shifted parameters:*

$$\begin{aligned} D_i^n(x;\alpha,\beta) &= \frac{(-1)^{n-i}(\sigma+1)_n}{K\,(\alpha+1)_{n-i}(\beta+1)_i} \sum_{k=0}^{i} \frac{(-i)_k}{(-n)_k} R_{n-k}^{(\alpha,\beta+k+1)}(x), \\ D_{n-i}^n(x;\alpha,\beta) &= \frac{(-1)^{i}(\sigma+1)_n}{K\,(\alpha+1)_{i}(\beta+1)_{n-i}} \sum_{k=0}^{i} (-1)^k \frac{(-i)_k}{(-n)_k} R_{n-k}^{(\alpha+k+1,\beta)}(x), \end{aligned} \tag{1.38}$$

where $i = 0,1,\ldots,n$.

Additionally, [61, §2] gives a Bernstein-Bézier representation of dual Bernstein polynomials and a recurrence relation for its coefficients. Some symmetries between the coefficients of such representation are given in [61, Remark 3.5] and [67, Proposition 3].

When $i = 0$ or $i = n$, the sums in Theorem 1.58 simplify to:

$$D_0^n(x;\alpha,\beta) = \frac{(-1)^n(\sigma+1)_n}{K\,(\alpha+1)_n} R_n^{(\alpha,\beta+1)}(x), \tag{1.39}$$

$$D_n^n(x;\alpha,\beta) = \frac{(\sigma+1)_n}{K\,(\beta+1)_n} R_n^{(\alpha+1,\beta)}(x). \tag{1.40}$$

Bernstein polynomials are usually considered in the interval $[0,1]$ and the formulas for their values at $x = 0$ or $x = 1$ are significantly simpler than in the general case. That is the case with dual Bernstein polynomials as well.

Remark 1.59. *Using [18, Eq. (3.1)], Eq.* (1.2) *and symmetry* (1.37)*, one can check that*

$$D_i^n(1;\alpha,\beta) = (-1)^{n-i}\frac{(\sigma+1)_n(n-i+\alpha+2)_i}{K\,n!(\beta+1)_i} \qquad (0 \le i \le n), \tag{1.41}$$

$$D_i^n(0;\alpha,\beta) = (-1)^{i}\frac{(\sigma+1)_n(i+\beta+2)_{n-i}}{K\,n!(\alpha+1)_{n-i}} \qquad (0 \le i \le n). \tag{1.42}$$

Theorem 1.60. *A representation of dual Bernstein polynomials in the basis $(1-x)^j$, for $j = 0, 1, \ldots, n$, is given by the following expression:*

$$D_i^n(x;\alpha,\beta) = A_{ni}^{(\alpha,\beta)}\frac{(\alpha+1)_n}{(n+1)!}\sum_{j=0}^{n} B_{nj}^{(\alpha,\beta)}\,{}_3F_2\left(\begin{matrix} j-n,\,-i,\,1 \\ -n,\,-n-\alpha \end{matrix}\,\middle|\, 1\right)\cdot(1-x)^j,$$

where

$$A_{ni}^{(\alpha,\beta)} := \frac{(-1)^{n-i}(n+1)(\sigma+1)_n}{K\,(\alpha+1)_{n-i}(\beta+1)_i}, \qquad B_{nj}^{(\alpha,\beta)} := \frac{(-n)_j(n+\sigma+1)_j}{j!(\alpha+1)_j}. \tag{1.43}$$

Proof. The hypothesis follows from applying the representation of shifted Jacobi polynomials in the $(1-x)^j$ basis (given in Eq. (1.14)) to Eq. (1.38) and doing some algebra. □

Recurrence relations can be an efficient way of computing the value of some polynomial families. In [60], a relation connecting dual Bernstein polynomials of two subsequent degrees was given.

Theorem 1.61 ([60, Theorem 5.1])**.** *The following recurrence relation, which connects dual Bernstein polynomials of degrees $n+1$ and n, as well as the shifted Jacobi polynomial of degree $n+1$, holds:*

$$D_i^{n+1}(x;\alpha,\beta) = \left(1-\frac{i}{n+1}\right) D_i^n(x;\alpha,\beta) + \frac{i}{n+1}\,D_{i-1}^n(x;\alpha,\beta) + C_{ni}^{(\alpha,\beta)}R_{n+1}^{(\alpha,\beta)}(x), \tag{1.44}$$

where $0 \le i \le n+1$, and

$$C_{ni}^{(\alpha,\beta)} := (-1)^{n-i+1}\frac{(2n+\sigma+2)(\sigma+1)_n}{K(\alpha+1)_{n-i+1}(\beta+1)_i}. \tag{1.45}$$

The case $\alpha = \beta = 0$ of this relation was found earlier by Ciesielski in [20]. The recurrence relation allows to compute $D_i^n(x;\alpha,\beta)$ for fixed i using the *triangular* recurrence scheme. The necessary sequence of shifted Jacobi polynomials can be computed in $O(n)$ time using, for example, the recurrence relation (1.10), thus giving the total $O(n^2)$ complexity. Using the same recursive approach, one can compute all dual Bernstein polynomials of degree n in the same $O(n^2)$ time.

Recurrence relations connecting the dual Bernstein polynomials of the same degree were presented in [18] and [96] and are expanded upon in Chapter 5. Applying this approach reduces the complexity of finding all dual Bernstein polynomials

$$D_0^n(x;\alpha,\beta), D_1^n(x;\alpha,\beta), \ldots, D_n^n(x;\alpha,\beta)$$

to $O(n)$ time.

1.5.3 Dual constrained Bernstein polynomials

For $k+l \le n$, let $\Pi_n^{(k,l)} := \{p \in \Pi_n : \ p^{(i)}(0) = 0, \text{ and } p^{(j)}(1) = 0 \text{ for } 0 \le i \le k-1, 0 \le j \le l-1\}$. Certainly, the Bernstein polynomials $B_k^n, B_{k+1}^n, \ldots, B_{n-l}^n$ form a *constrained Bernstein basis* of $\Pi_n^{(k,l)}$ (and thus $\dim \Pi_n^{(k,l)} = n-k-l+1$). The basis dual to it, i.e., the *dual constrained Bernstein basis*,

$$D_k^{(n,k,l)}(x;\alpha,\beta), D_{k+1}^{(n,k,l)}(x;\alpha,\beta), \ldots, D_{n-l}^{(n,k,l)}(x;\alpha,\beta) \in \Pi_n^{(k,l)},$$

consists of the so-called *dual constrained Bernstein polynomials*, i.e.,

$$\langle D_i^{(n,k,l)}, B_j^n \rangle_{\alpha,\beta} = \delta_{ij} \qquad (k \le i,j \le n-l)$$

(cf. (1.6) and (1.35)). Obviously,

$$D_k^{(n,0,0)}(x;\alpha,\beta) = D_k^n(x;\alpha,\beta).$$

For $k = l$, i.e., in the space $\Pi_n^{(k,k)}$, the representation of dual constrained Bernstein polynomials in the Bernstein basis is given for $\alpha = \beta = 0$ in [52], and, for any $\alpha, \beta > -1$, in [81]. In the general case, i.e., $k+l \le n$ and $\alpha, \beta > -1$, the representation of dual constrained Bernstein polynomials in the Bernstein basis is given in [61, Theorem 3.1], with a recurrence relation satisfied by its coefficients. Some symmetries between the coefficients of such representation are given in [61, Remark 3.5] and [67, Proposition 3].

Dual constrained Bernstein polynomials can be expressed using the (unconstrained) dual Bernstein polynomials of lower degree.

Theorem 1.62 ([97, Theorem 3.1])**.** *For $i = k, k+1, \ldots, n-l$, the following formula holds:*

$$D_i^{(n,k,l)}(x;\alpha,\beta) = \binom{n-k-l}{i-k} \binom{n}{i}^{-1} x^k (1-x)^l D_{i-k}^{n-k-l}(x;\alpha+2l,\beta+2k).$$

Dual constrained Bernstein polynomials have their applications in reducing the degree of Bézier curves with constraints (see [97] and §1.7.4).

1.5.4 Discrete Bernstein and dual Bernstein polynomials

Discrete Bernstein polynomials of degree $n \in \mathbb{N}$ are defined as

$$b_i^n(x;N) := \frac{1}{(-N)_n}\binom{N}{i}(-x)_i(x-N)_{n-i} \qquad (0 \le i \le n \le N; N \in \mathbb{N})$$

(cf. [86, 87], [97, §A.3]). A family of polynomials

$$d_0^n(x;\alpha,\beta,N), d_1^n(x;\alpha,\beta,N), \ldots, d_n^n(x;\alpha,\beta,N) \in \Pi_n,$$

which is dual to the discrete Bernstein polynomials with respect to the Hahn scalar product $\langle\cdot,\cdot\rangle_H$ (cf. Theorem 1.34), i.e.,

$$\langle d_i^n(\cdot;\alpha,\beta,N), B_j^n\rangle_H = \delta_{ij} \qquad (k \le i,j \le n-l)$$

(cf. (1.35)), will be considered. These polynomials are known as *discrete dual Bernstein polynomials*.

Discrete dual Bernstein polynomials satisfy certain properties, of which some find application in reducing the degree of Bézier curves with constraints (see [97] and §1.7.4).

Remark 1.63 ([97, Corollary A.4])**.** *Discrete dual Bernstein polynomials satisfy the following symmetry property:*

$$d_i^n(x;\alpha,\beta,N) = d_{n-i}^n(N-x,\beta,\alpha,N) \qquad (0 \le i \le n \le N). \tag{1.46}$$

Theorem 1.64 ([97, Theorem A.5])**.** *Discrete dual Bernstein polynomials have the following form as a short linear combination of Hahn polynomials with shifted parameters:*

$$d_i^n(x;\alpha,\beta,N) = A_n^N \frac{(-n-\beta)_i}{(\alpha+1)_i}\sum_{k=0}^{i}\frac{(-i)_k(-n-\sigma-N)_k}{(-n-\beta)_k(1-N)_k}Q_{n-k}(N-x;\beta,\alpha+k+1,N-k-1), \tag{1.47}$$

where $i = 0,1,\ldots,n$ $(n \le N)$, and

$$A_n^N := \frac{N!(1-N)_n}{n!(n+\sigma+1)_N}.$$

Theorem 1.65 ([97, Theorem A.6])**.** *Discrete dual Bernstein polynomials*

$$d_i^n(x) \equiv d_i^n(x;\alpha,\beta,N)$$

satisfy the following difference-recurrence relation:

$$a_N(x)d_i^n(x+1) + \Big[c_n(i) - c_N(x)\Big]d_i^n(x) + b_N(x)d_i^n(x-1) - a_n(i)d_{i+1}^n(x) - b_n(i)d_{i-1}^n(x) = 0$$

for $0 \le i \le n \le N$, where $d_{-1}^n(x) = d_{n+1}^n(x) := 0$, and

$$a_n(x) := (x-n)(x+\alpha+1), \qquad b_n(x) := x(x-\beta-n-1), \qquad c_n(x) := a_n(x) + b_n(x).$$

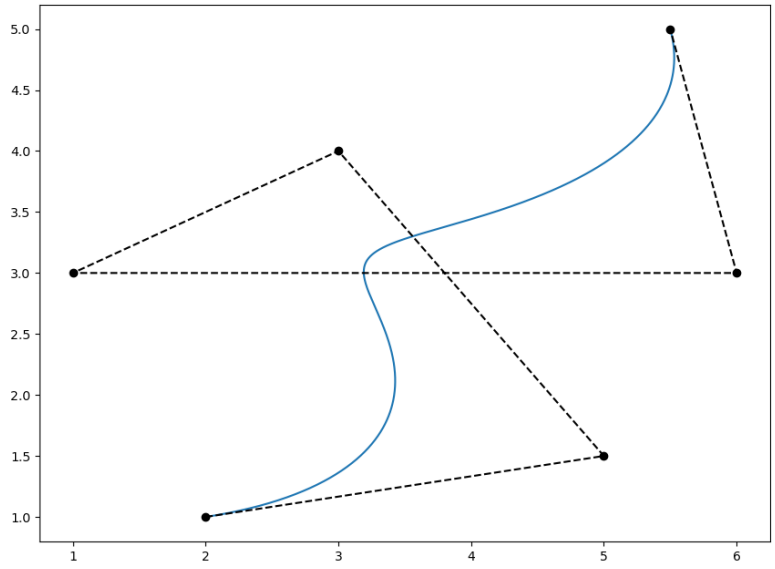

Figure 1.7: A Bézier curve of degree 5 with control points $\mathsf{W}_0 = (2,1)$, $\mathsf{W}_1 = (5,1.5)$, $\mathsf{W}_2 = (3,4)$, $\mathsf{W}_3 = (1,3)$, $\mathsf{W}_4 = (6,3)$, $\mathsf{W}_5 = (5.5,5)$. The dashed line connecting the control points is called the *control polygon* of the curve.

1.6 Bézier curves

The intention behind inventing Bézier curves was to make computer-aided techniques for automobile design possible and intuitive. Pierre Bézier and Paul de Casteljau's work resulted in settling on a polynomial curve with control points and used Bernstein polynomials as a basis. Such approach gives a family of curves which have very neat properties, allowing the designers to easily control their shape and behavior. For more information about the history of Bézier curves, see, e.g., [9, 12–14, 29–31], as well as [36, §1] and [38, §4].

Polynomial Bézier curves are a particular family of parametric curves which is defined as a convex combination of control points (cf. §1.2). The points are weighted using Bernstein polynomials.

Definition 1.66. *A* (polynomial) Bézier curve $\mathsf{P}_n : [0,1] \to \mathbb{E}^d$ of degree n *with* control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n \in \mathbb{E}^d$ *is defined by the formula*

$$\mathsf{P}_n(t) := \sum_{k=0}^{n} B_k^n(t)\mathsf{W}_k, \tag{1.48}$$

where B_k^n is the kth Bernstein polynomial of degree n (see Definition 1.43).

An example polynomial Bézier curve is shown in Figure 1.7.

Rational Bézier curves are a generalized version of Bézier curves, with each control point additionally having an assigned weight.

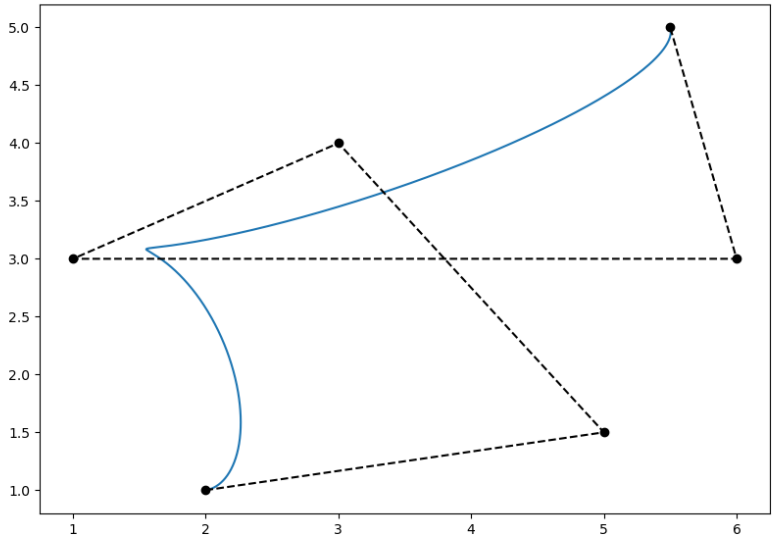

Figure 1.8: A rational Bézier curve of degree 5 with the same control points as in Figure 1.7, with weights $\omega_0 = \omega_2 = \omega_4 = \omega_5 = 1$ and $\omega_1 = 0.2$, $\omega_3 = 8$. Note that, compared to Figure 1.7, the curve stays much closer to the point W_3, while the opposite can be observed for the point W_1. The dashed line is the control polygon of the curve (cf. Figure 1.7).

Definition 1.67. *A* rational Bézier curve $\mathsf{R}_n : [0,1] \to \mathbb{E}^d$ of degree n *with* control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n \in \mathbb{E}^d$ *and their corresponding* weights $\omega_0, \omega_1, \ldots, \omega_n > 0$ *is defined by the formula*

$$\mathsf{R}_n(t) := \frac{\sum_{k=0}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)}, \tag{1.49}$$

where B_k^n is the kth Bernstein polynomial of degree n.

Figure 1.8 shows an example rational Bézier curve.

Remark 1.68. *When $\omega_0 = \omega_1 = \ldots = \omega_n$, the weights can be eliminated from Eq.* (1.49) *and, after using Eq.* (1.27) *in the denominator, a polynomial Bézier curve is obtained.*

From the properties of Bernstein polynomials, some properties of Bézier curves follow.

Theorem 1.69. *A rational Bézier curve R_n of degree n with control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n \in \mathbb{E}^d$ and their corresponding weights $\omega_0, \omega_1, \ldots, \omega_n > 0$ satisfies the following properties.*

1. *$\mathsf{R}_n(0) = \mathsf{W}_0$ and $\mathsf{R}_n(1) = \mathsf{W}_n$.*

2. *From the symmetry property of Bernstein polynomials, it follows that*

$$\frac{\sum_{k=0}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)} = \frac{\sum_{k=0}^{n} \omega_{n-k} B_k^n(1-t) \mathsf{W}_{n-k}}{\sum_{k=0}^{n} \omega_{n-k} B_k^n(1-t)}.$$

3. *One can define a rational Bézier curve over $t \in [a, b]$ using the linear parameter transformation:*

$$\mathsf{R}_n(t) = \frac{\sum_{k=0}^{n} \omega_k B_k^n(u) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(u)} \qquad \left(u := \frac{t-a}{b-a}\right).$$

Remark 1.70. *Both polynomial and rational Bézier curves have the convex hull property. Let R_n be a rational Bézier curve with control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$ and positive weights $\omega_0, \omega_1, \ldots, \omega_n$. Let C be the convex hull of points $\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n\}$. For any $t \in [0, 1]$, $\mathsf{R}_n(t) \in C$.*

For more properties of Bézier curves, see, e.g., [36, §4-6, §13].

1.7 Algorithms for Bézier curves

Now, some well-known results related to the most important algorithms for Bézier curves are described briefly. See, e.g., [22, 36, 54, 77, 78].

1.7.1 Evaluating a point on the curve

One can evaluate a point on a (polynomial or rational) Bézier curve using the de Casteljau algorithm. It is a classic result, covered extensively in literature (see, e.g., [9], [36, §4.2] for the polynomial de Casteljau algorithm and [36, §13.2] for the rational de Casteljau algorithm). The algorithms presented here will be improved in Chapter 2.

The algorithm is based on Eq. (1.28). When applied to Eq. (1.48), one gets

$$\mathsf{P}_n(t) = \sum_{k=0}^{n} [t B_{k-1}^{n-1}(t) + (1-t) B_k^{n-1}(t)] \mathsf{W}_k = \sum_{k=0}^{n-1} B_k^{n-1}(t) [t \mathsf{W}_{k+1} + (1-t) \mathsf{W}_k],$$

which is a formula for a polynomial Bézier curve of lower degree with new control points (dependent on the value of the parameter $t \in [0, 1]$) given as convex combinations of control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$. The process can be repeated until the degree of the curve reaches zero and only one control point remains. Figure 1.9 illustrates the computation of the de Casteljau algorithm. An implementation of the polynomial de Casteljau algorithm is given in Algorithm 1.1.

The case of rational Bézier curves is similar and Eq. (1.28) is used here as well. This time, however, one has to take the weights $\omega_0, \omega_1, \ldots, \omega_n$ into account. Applying Eq. (1.28)

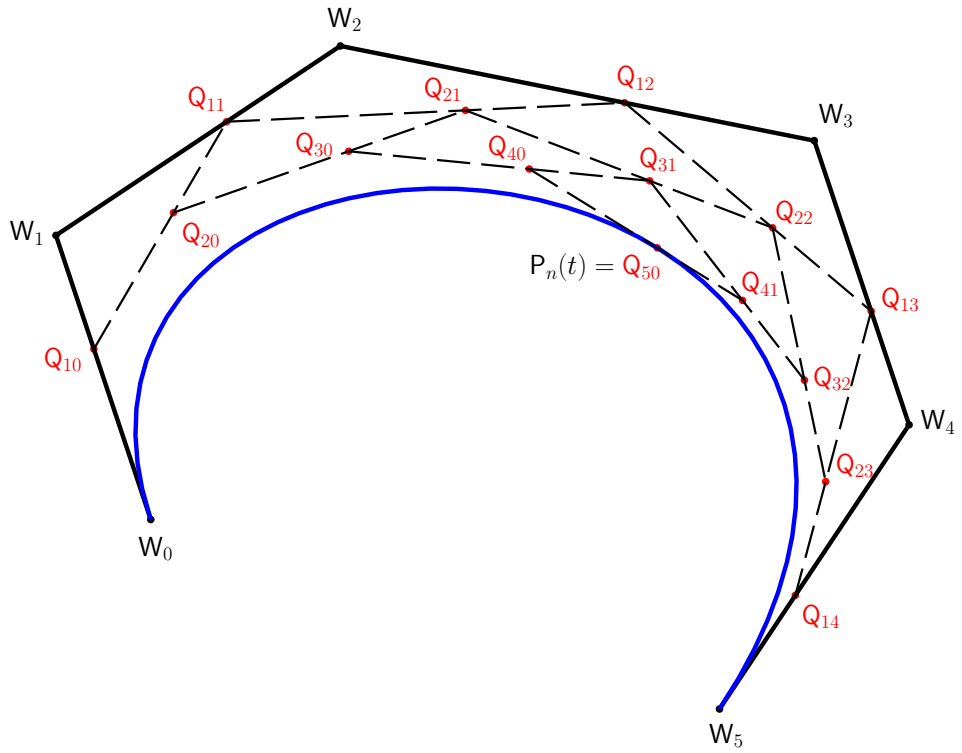

Figure 1.9: Computation of a point on a planar polynomial Bézier curve of degree $n = 5$ using the de Casteljau algorithm, with the notation as in (1.50).

Algorithm 1.1 De Casteljau algorithm

```
1:  procedure BEVAL(n, t, W)
2:      t_1 ← 1 − t
3:      for i ← 0, n do
4:          Q_i ← W_i
5:      end for
6:      for k ← 1, n do
7:          for i ← 0, n − k do
8:              Q_i ← t_1 · Q_i + t · Q_{i+1}
9:          end for
10:     end for
11:     return Q_0
12: end procedure
```

to Eq. (1.49) gives

$$\mathsf{R}_n(t) = \frac{\displaystyle\sum_{k=0}^{n} \omega_k [t B_{k-1}^{n-1}(t) + (1-t) B_k^{n-1}(t)] \mathsf{W}_k}{\displaystyle\sum_{k=0}^{n} \omega_k [t B_{k-1}^{n-1}(t) + (1-t) B_k^{n-1}(t)]} = \frac{\displaystyle\sum_{k=0}^{n-1} B_k^{n-1}(t) [\omega_{k+1} t \mathsf{W}_{k+1} + \omega_k (1-t) \mathsf{W}_k]}{\displaystyle\sum_{k=0}^{n-1} B_k^{n-1}(t) [\omega_{k+1} t + \omega_k (1-t)]}.$$

Let the new weights $\omega'_0, \omega'_1, \ldots, \omega'_{n-1}$ be defined as

$$\omega'_k := \omega_{k+1}t + \omega_k(1-t) \qquad (k = 0, 1, \ldots, n-1).$$

Note that the new weights (dependent on the parameter $t \in [0,1]$) are given as convex combinations of non-negative weights, therefore all of them are non-negative as well. Thus, we have

$$\mathsf{R}_n(t) = \frac{\displaystyle\sum_{k=0}^{n-1} \omega'_k B_k^{n-1}(t) \left[\frac{\omega_{k+1}t}{\omega'_k} \mathsf{W}_{k+1} + \left(1 - \frac{\omega_{k+1}t}{\omega'_k}\right) \mathsf{W}_k \right]}{\displaystyle\sum_{k=0}^{n-1} B_k^{n-1}(t)\omega'_k}.$$

Just as in the case of polynomial Bézier curves, a rational Bézier curve has been expressed as a rational Bézier curve of lower degree, with new weights ω'_k and control points. The new control points, also dependent on the parameter $t \in [0,1]$, given by

$$W'_k := \frac{\omega_{k+1}t}{\omega'_k} \mathsf{W}_{k+1} + \left(1 - \frac{\omega_{k+1}t}{\omega'_k}\right) \mathsf{W}_k \qquad (k = 0, 1, \ldots, n-1),$$

are convex combinations of the original control points. An implementation of this method is given in Algorithm 1.2.

Algorithm 1.2 Rational de Casteljau algorithm

```
1:  procedure RatBEval(n, t, ω, W)
2:      t_1 ← 1 − t
3:      for i ← 0, n do
4:          w_i ← ω_i
5:          Q_i ← W_i
6:      end for
7:      for k ← 1, n do
8:          for i ← 0, n − k do
9:              u ← t_1 · w_i
10:             v ← t · w_{i+1}
11:             w_i ← u + v
12:             u ← u/w_i
13:             v ← 1 − u
14:             Q_i ← u · Q_i + v · Q_{i+1}
15:         end for
16:     end for
17:     return Q_0
18: end procedure
```

In some applications, such as curve subdivision, it is useful to store all the intermediate points computed in the (polynomial or rational) de Casteljau algorithm. In the polynomial case, one can express the algorithm using the following recurrence scheme:

$$\begin{cases} \mathsf{Q}_{0k} := \mathsf{W}_k & (k = 0, 1, \ldots, n), \\ \mathsf{Q}_{jk} := (1-t)\mathsf{Q}_{j-1,k} + t\mathsf{Q}_{j-1,k+1} & (j = 1, 2, \ldots, n;\ k = 0, 1, \ldots, n-j), \\ \mathsf{P}_n(t) = \mathsf{Q}_{n0}. \end{cases} \tag{1.50}$$

One can find an explicit expression for all the points.

Theorem 1.71. *The points* Q_{jk} $(j = 0, 1, \ldots, n; k = 0, 1, \ldots, n - j)$ *which are defined in Eq.* (1.50) *can be expressed explicitly as*

$$\mathsf{Q}_{jk} = \sum_{i=0}^{j} B_i^j(t)\,\mathsf{W}_{i+k}.$$

Proof. Let us fix a natural number n. For $j = 0$, the theorem holds, as $\mathsf{Q}_{0k} = B_0^0(t) \cdot \mathsf{W}_k$. Now, assuming that the theorem holds for all $\mathsf{Q}_{j-1,k}$ $(k = 0, 1, \ldots, n - j + 1)$, one needs to check that it also holds for all Q_{jk}. Applying the induction assumption twice in Eq. (1.50) gives

$$\mathsf{Q}_{jk} = (1-t)\sum_{i=0}^{j-1} B_i^{j-1}(t)\mathsf{W}_{i+k} + t\sum_{i=0}^{j-1} B_i^{j-1}(t)\mathsf{W}_{i+k+1} = \sum_{i=0}^{j}\left[(1-t)B_i^{j-1}(t) + tB_{i-1}^{j-1}(t)\right]\mathsf{W}_{i+k}$$

which, after applying Eq. (1.28), gives

$$\mathsf{Q}_{jk} = \sum_{i=0}^{j} B_i^j(t)\mathsf{W}_{i+k}.$$

□

Analogous properties hold for rational Bézier curves and the rational de Casteljau algorithm. For rational Bézier curves, the following recurrence scheme can be used to compute the required weights and points:

$$\begin{cases} w_{0k} := \omega_k \qquad \mathsf{Q}_{0k} := \mathsf{W}_k & (k = 0, 1, \ldots, n), \\ w_{jk} := (1-t)w_{j-1,k} + tw_{j-1,k+1} & (j = 1, 2, \ldots, n;\ k = 0, 1, \ldots, n-j), \\ \mathsf{Q}_{jk} := \left(1 - \dfrac{w_{j-1,k+1}t}{w_{jk}}\right)\mathsf{Q}_{j-1,k} + \dfrac{w_{j-1,k+1}t}{w_{jk}}\mathsf{Q}_{j-1,k+1} & \\ \qquad\qquad (j = 1, 2, \ldots, n;\ k = 0, 1, \ldots, n-j), & \\ \mathsf{R}_n(t) = \mathsf{Q}_{n0}. & \end{cases} \tag{1.51}$$

Same as before, one can find an explicit expression for all the points and weights.

Theorem 1.72. *The points* Q_{jk} *and their corresponding weights* w_{jk} $(j = 0, 1, \ldots, n; k = 0, 1, \ldots, n - j)$*, which are defined in Eq.* (1.51)*, can be expressed explicitly as*

$$w_{jk} = \sum_{i=0}^{j} \omega_{i+k} B_i^j(t), \qquad \mathsf{Q}_{jk} = \frac{\displaystyle\sum_{i=0}^{j} \omega_{i+k} B_i^j(t)\,\mathsf{W}_{i+k}}{\displaystyle\sum_{i=0}^{j} \omega_{i+k} B_i^j(t)}.$$

Proof. Let us fix $n \in \mathbb{N}$. For $j = 0$, the theorem holds, as $w_{0k} = \omega_k$ and $\mathsf{Q}_{0k} = \mathsf{W}_k$. Now, assuming that the theorem holds for all $w_{j-1,k}, \mathsf{Q}_{j-1,k}$ $(k = 0, 1, \ldots, n-j+1)$, one needs to check that it also holds for all w_{jk}, Q_{jk}. Applying the induction assumption in Eq. (1.51) gives

$$\begin{cases} w_{jk} := \displaystyle\sum_{i=0}^{j} \omega_{i+k} \left[(1-t) B_i^{j-1}(t) + t B_{i-1}^{j-1}(t) \right] = \sum_{i=0}^{j} \omega_{i+k} B_i^j(t), \\ \mathsf{Q}_{jk} := \displaystyle\sum_{i=0}^{j} \frac{\omega_{i+k} \left[(1-t) B_i^{j-1}(t) + t B_{i-1}^{j-1}(t) \right] \mathsf{W}_{i+k}}{w_{jk}} = \sum_{i=0}^{j} \frac{\omega_{i+k} B_i^j(t) \mathsf{W}_{i+k}}{w_{jk}}. \end{cases}$$

□

1.7.2 Curve subdivision

Curve subdivision is a very common problem to consider. Given a Bézier curve P_n of degree n with control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$ and $u \in [0,1]$, one needs to find two Bézier curves $\mathsf{P}_n^L, \mathsf{P}_n^R$ of degree n such that

$$\begin{cases} \mathsf{P}_n(u \cdot t) = \mathsf{P}_n^L(t) & (t \in [0,1]), \\ \mathsf{P}_n(u + (1-u) \cdot t) = \mathsf{P}_n^R(t) & (t \in [0,1]). \end{cases} \tag{1.52}$$

It is sufficient to find the control points of $\mathsf{P}_n^L, \mathsf{P}_n^R$.

In this problem, the intermediate points computed in the de Casteljau algorithm find their application.

Theorem 1.73 ([36, Eq. (5.29)])**.** *Let P_n be a Bézier curve of degree n with control points $W_0, W_1, \ldots, W_n$. Let $u \in [0,1]$. Then the fragment $P_n^L(t) \equiv P_n([0,u])$ (cf. (1.52)) is a Bézier curve of degree n with control points $Q_{00}, Q_{10}, \ldots, Q_{n0}$, where the points Q_{j0} $(j = 0, 1, \ldots, n)$ are defined as in Eq. (1.50).*

Corollary 1.74. *One can find a Bézier curve P_n^R of degree n such that $P_n^R \equiv P_n([u,1])$ (cf. (1.52)). Its control points are $Q_{n0}, Q_{n-1,1}, \ldots, Q_{1,n-1}, Q_{0n}$.*

Proof. By reversing the order of the control points of P_n, the problem can be reduced to the case covered by Theorem 1.73. The Bézier curve S_n obtained in this way satisfies a condition

$$\mathsf{S}_n(t) = \mathsf{P}_n^R(1-t) \qquad (0 \le t \le 1).$$

Therefore, to get the control points of P_n^R, one has to reverse the order of the control points of S_n, thus giving $\mathsf{Q}_{n0}, \mathsf{Q}_{n-1,1}, \ldots, \mathsf{Q}_{1,n-1}, \mathsf{Q}_{0n}$. □

An analogous reasoning gives the subdivision of a rational Bézier curve.

Theorem 1.75 ([36, §13.2])**.** *Let R_n be a rational Bézier curve of degree n. Let its control points and weights be $W_0, W_1, \ldots, W_n$ and $\omega_0, \omega_1, \ldots, \omega_n$, respectively. Let $u \in [0,1]$. Then the fragment $R_n([0,u])$ is a rational Bézier curve of degree n with control points $V_{00}, V_{10}, \ldots, V_{n0}$ and weights $v_{00}, v_{10}, \ldots, v_{n0}$, where the points V_{j0} and weights v_{j0} $(j = 0, 1, \ldots, n)$ are defined as in Eq. (1.51).*

Proof. One seeks a rational Bézier curve R_n^L of degree n which satisfies the relation

$$\mathsf{R}_n^L(t) = \mathsf{R}_n(t \cdot u).$$

The right-hand side, by Definition 1.67, can be expressed as

$$\mathsf{R}_n(t \cdot u) = \frac{\sum_{i=0}^{n} \omega_i B_i^n(t \cdot u) \mathsf{W}_i}{\sum_{i=0}^{n} \omega_i B_i^n(t \cdot u)},$$

which, after applying Eq. (1.30), gives

$$\mathsf{R}_n(t \cdot u) = \frac{\sum_{i=0}^{n} \omega_i \sum_{j=0}^{n} B_i^j(u) B_j^n(t) \mathsf{W}_i}{\sum_{i=0}^{n} \omega_i \sum_{j=0}^{n} B_i^j(u) B_j^n(t)}.$$

After changing the order of the summation and applying Remark 1.48 in the inner sum, one gets

$$\mathsf{R}_n(ut) = \frac{\sum_{j=0}^{n} B_j^n(t) \left(\sum_{i=0}^{j} \omega_i B_i^j(u) \mathsf{W}_i \right)}{\sum_{j=0}^{n} \left(\sum_{i=0}^{j} \omega_i B_i^j(u) \right) B_j^n(t)},$$

which, by Theorem 1.72, is

$$\mathsf{R}_n(ut) = \frac{\sum_{j=0}^{n} v_{j0} B_j^n(t) \frac{\sum_{i=0}^{j} \omega_i B_i^j(u) \mathsf{W}_i}{v_{j0}}}{\sum_{j=0}^{n} v_{j0} B_j^n(t)} = \frac{\sum_{j=0}^{n} v_{j0} B_j^n(t) \mathsf{V}_{j0}}{\sum_{j=0}^{n} v_{j0} B_j^n(t)}.$$

The control points of R_n^L thus are $\mathsf{V}_{00}, \mathsf{V}_{10}, \ldots, \mathsf{V}_{n0}$, with weights $v_{00}, v_{10}, \ldots, v_{n0}$. □

Corollary 1.76. *One can find a rational Bézier curve R_n^R of degree n such that $\mathsf{R}_n^R([0,1]) = \mathsf{R}_n([u,1])$ $(0 \le u \le 1)$. The curve R_n^R satisfies the condition*

$$\mathsf{R}_n^R(t) = \mathsf{P}_n((1-u)t + u) \qquad (0 \le t \le 1).$$

Its control points are

$$\mathsf{V}_{n0}, \mathsf{V}_{n-1,1}, \ldots, \mathsf{V}_{1,n-1}, \mathsf{V}_{0n}$$

and their corresponding weights are

$$v_{n0}, v_{n-1,1}, \ldots, v_{1,n-1}, v_{0n}.$$

The proof of Corollary 1.76 is analogous to the proof of Corollary 1.74.

1.7.3 Degree elevation

A Bézier curve of degree n can be expressed as a Bézier curve of degree $n+1$. This effect can be achieved using *degree elevation*. The method described in this section is also classical and can be found in more detail in, e.g., [78, §3.11], [36, §6.1]. More precisely, when given the control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$ of a polynomial Bézier curve P_n, one has to find the control points $\mathsf{V}_0, \mathsf{V}_1, \ldots, \mathsf{V}_{n+1}$ of a curve P_{n+1} so that

$$\mathsf{P}_n(t) = \mathsf{P}_{n+1}(t) \qquad (0 \le t \le 1)$$

or, explicitly,

$$\sum_{k=0}^{n} B_k^n(t)\mathsf{W}_k = \sum_{k=0}^{n+1} B_k^{n+1}(t)\mathsf{V}_k \qquad (0 \le t \le 1).$$

This can be achieved by applying Eq. (1.29) to the left-hand side:

$$\sum_{k=0}^{n} B_k^n(t)\mathsf{W}_k = B_0^{n+1}(t)\mathsf{W}_0 + \sum_{k=1}^{n} B_k^{n+1}(t)\left[\frac{n-k+1}{n+1}\mathsf{W}_k + \frac{k}{n+1}\mathsf{W}_{k-1}\right] + B_{n+1}^{n+1}(t)\mathsf{W}_n.$$

This means that the control points of P_{n+1} can be expressed as the convex combinations of $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$:

$$\mathsf{V}_k := \frac{n-k+1}{n+1}\mathsf{W}_k + \frac{k}{n+1}\mathsf{W}_{k-1} \qquad (k = 0, 1, \ldots, n+1). \tag{1.53}$$

Note that, for $k = 0$ and $k = n+1$, even though there are undefined points W_{-1} and W_{n+1}, their corresponding coefficient is zero and they can be omitted.

An analogous procedure can be done with rational Bézier curves. One needs to find the control points $\mathsf{V}_0, \mathsf{V}_1, \ldots, \mathsf{V}_{n+1}$ and their corresponding weights $v_0, v_1, \ldots, v_{n+1}$ so that

$$\frac{\displaystyle\sum_{k=0}^{n} \omega_k B_k^n(t)\mathsf{W}_k}{\displaystyle\sum_{k=0}^{n} \omega_k B_k^n(t)} = \frac{\displaystyle\sum_{k=0}^{n+1} v_k B_k^{n+1}(t)\mathsf{V}_k}{\displaystyle\sum_{k=0}^{n+1} v_k B_k^{n+1}(t)} \qquad (0 \le t \le 1).$$

One can apply Eq. (1.29) to the left-hand side to get

$$\frac{\displaystyle\sum_{k=0}^{n} \omega_k B_k^n(t)\mathsf{W}_k}{\displaystyle\sum_{k=0}^{n} \omega_k B_k^n(t)} = \frac{\displaystyle\sum_{k=0}^{n+1} B_k^{n+1}(t)\left[\omega_k \frac{n-k+1}{n+1}\mathsf{W}_k + \omega_{k-1}\frac{k}{n+1}\mathsf{W}_{k-1}\right]}{\displaystyle\sum_{k=0}^{n+1}\left[\omega_k \frac{n-k+1}{n+1} + \omega_{k-1}\frac{k}{n+1}\right]B_k^{n+1}(t)},$$

where the convention as in the case of polynomial Bézier curves is used, i.e., $\omega_{-1}, \omega_{n+1} := 0$. One can express v_k as

$$v_k := \frac{n-k+1}{n+1}\omega_k + \frac{k}{n+1}\omega_{k-1} \qquad (k = 0, 1, \ldots, n+1),$$

which immediately gives an expression for V_k:

$$\mathsf{V}_k := \frac{(n-k+1)\omega_k \mathsf{W}_k + k\omega_{k-1}\mathsf{W}_{k-1}}{(n-k+1)\omega_k + k\omega_{k-1}} \qquad (k = 0, 1, \ldots, n+1).$$

The presented methods only elevate the curve's degree by one. In order to elevate the curve's degree by more than one, one can use this technique multiple times or, alternatively, use the explicit formula for degree elevation by any positive r:

$$B_k^n(t) = \sum_{j=k}^{k+r} \frac{\binom{n}{k}\binom{r}{j-k}}{\binom{n+r}{j}} B_j^{n+r}(t)$$

(cf. [40, Eq. (26)]).

Both in the polynomial and the rational case, for a curve with control points in $\mathbb{E}^d$, the computational complexity of elevating its degree from n to $n+1$ is $O(nd)$. If one needs to elevate the curve's degree by r, the computational complexity is $O((n+r)rd)$.

1.7.4 Degree reduction

In some cases, it may be necessary to reduce the degree of a Bézier curve. The problem is then to find a curve of lower degree which is a good enough approximation of the higher-degree curve. There are practical reasons to reduce the degree of a curve such as data compression and data exchange between CAD systems. As it is evident from the problem of degree elevation, a curve of low degree can have its degree elevated by an arbitrary number. The same effect can be achieved unintentionally while designing the curve.

Two methods for degree reduction will be presented here. The first is based on the degree elevation algorithm and is useful due to its simplicity while the second applies dual bases to find the optimal curve of lower degree in terms of least-square approximation.

One method of degree reduction is based on Eq. (1.53) which was used in the process of degree elevation. It can be found, e.g., in [73, 76], [54, §1.9.2] or [77, §5.6]. When reducing the degree, however, one has to solve the opposite problem: given the control points $\mathsf{V}_0, \mathsf{V}_1, \ldots, \mathsf{V}_{n+1}$ of a curve, one needs to find the control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$ of a curve of degree n. The Eq. (1.53) is thus re-expressed as

$$\mathsf{W}_k := \frac{n+1}{n-k+1}\mathsf{V}_k - \frac{k}{n-k+1}\mathsf{W}_{k-1} \qquad (k = 0, 1, \ldots, n) \tag{1.54}$$

or

$$\mathsf{W}_{k-1} := \frac{n+1}{k}\mathsf{V}_k - \frac{n-k+1}{k}\mathsf{W}_k \qquad (k = 1, \ldots, n+1). \tag{1.55}$$

Analogous relations can be derived for rational Bézier curves.

The idea behind this approach is to select a natural k (such that $k \leq n$ — usually $k \approx n/2$) and compute the points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_k$ using Eq. (1.54) and the points $\mathsf{W}_n, \mathsf{W}_{n-1}, \ldots, \mathsf{W}_k$ using Eq. (1.55). Note that the point W_k is computed twice — if the curve's *real degree* is n or less, both results are identical. In numerical practice, the values of W_k may differ slightly due to error accumulation. If two values of W_k are different, an average is used.

This method of degree reduction is widely used because of its simplicity. However, a significant drawback of this approach is that the lower-degree curve is not optimal in terms of approximation and thus its shape may significantly differ from the optimal curve of the

same degree. The effect is especially visible around $t = 0.5$. For such t, the curve's shape is influenced the most by the control points which have been computed in the last iterations of equations (1.54) and (1.55). This method can be used to reduce the degree of a curve by more than one. The approach used is similar to the one used for degree elevation.

Alternatively, one can use dual projection to reduce the degree of a curve (see, e.g., [97, §1] and the papers cited therein). Given a Bézier curve P_n of degree n with control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n \in \mathbb{E}^d$, i.e.,

$$\mathsf{P}_n(t) := \sum_{i=0}^{n} \mathsf{W}_i B_i^n(t) \qquad (0 \le t \le 1), \tag{1.56}$$

one can find the Bézier curve P_m of degree $m < n$, i.e.,

$$\mathsf{P}_m(t) := \sum_{i=0}^{m} \mathsf{V}_i B_i^m(t) \qquad (0 \le t \le 1), \tag{1.57}$$

where $\mathsf{V}_0, \mathsf{V}_1, \ldots, \mathsf{V}_m \in \mathbb{E}^d$, such that:

(a)
$$\begin{aligned} \mathsf{P}_n^{(i)}(0) &= \mathsf{P}_m^{(i)}(0) \qquad (i = 0, 1, \ldots, k-1), \\ \mathsf{P}_n^{(j)}(1) &= \mathsf{P}_m^{(j)}(1) \qquad (j = 0, 1, \ldots, l-1), \end{aligned} \tag{1.58}$$

where $k + l \le m$,

(b) the value
$$\int_0^1 (1-t)^\alpha t^\beta \|\mathsf{P}_n(t) - \mathsf{P}_m(t)\|_2^2 \mathrm{d}t$$
is minimized.

Here, $\|v\|_2$ denotes the length of vector $v \in \mathbb{R}^d$, i.e.,

$$\|v\|_2 \equiv \|[v_1, v_2, \ldots, v_d]^T\|_2 := \sqrt{\sum_{k=1}^{d} v_k^2}. \tag{1.59}$$

The quantities k, l serve as constraints which preserve the shape of the curve at its ends. One needs to find the solution in the $\Pi_n^{(k,l)}$ space (cf. §1.5.3), which is spanned by both the constrained Bernstein basis and its dual counterpart. For an unconstrained version of the problem, one can take $k = l = 0$.

The problem has been considered in multiple papers, both in constrained and unconstrained versions, e.g., in [1, 2, 10, 17, 34, 37, 59, 68, 69, 82, 91, 92, 97, 106]. Out of these works, the method given in [97] has the lowest $O(mn)$ complexity which has not yet been further improved. In chapter 6, this method is modified so that a significant part of the computations can be done in parallel, while maintaining the total complexity.

In this section, the results given by Woźny and Lewanowicz in [97] will be presented. Note, in particular, the coefficients Φ and Ψ, which will be extensively used in the degree reduction algorithm and in Chapter 6.

Theorem 1.77 ([97, Theorem 3.2]). *For $i = k, k+1, \ldots, m-l$ and $0 \le k+l \le m$, the following formula holds:*

$$D_i^{(m,k,l)}(x;\alpha,\beta) = \sum_{j=k}^{n-l} \Phi_{ij} D_j^{(n,k,l)}(x;\alpha,\beta)$$

(cf. §1.5.3), where the coefficients $\Phi_{ij} \equiv \Phi_{ij}^{(n,m,k,l)}(\alpha,\beta)$ are given in terms of discrete dual Bernstein polynomials (cf. §1.5.4), namely, for $j = k, k+1, \ldots, n-l$,

$$\Phi_{ij} = \binom{m-k-l}{i-k}\binom{n}{j}\binom{m}{i}^{-1} \frac{(\alpha+2l+1)_{n-l-j}(\beta+2k+1)_{j-k}}{(n-k-l)!}\Psi_{ij}, \tag{1.60}$$

with

$$\Psi_{ij} := d_{i-k}^{m-k-l}(j-k;\beta+2k,\alpha+2l,n-k-l). \tag{1.61}$$

Remark 1.78 ([97, Remark 3.3]). *Obviously, for $k \le i \le m-l$ and $l \le j \le n-l$,*

$$\Phi_{ij} = \left\langle B_j^n, D_i^{(m,k,l)} \right\rangle_{\alpha,\beta}, \tag{1.62}$$

where the inner product $\langle \cdot,\cdot \rangle_{\alpha,\beta}$ is given by

$$\langle f, g\rangle_{\alpha,\beta} := \int_0^1 (1-x)^\alpha x^\beta f(x) g(x)\, dx \qquad (\alpha,\beta > -1)$$

(cf. [97, Eq. (1.3)]) and (1.6).

Theorem 1.79 ([97, Theorem 4.1]). *Given the coefficients $W_0, W_1, \ldots, W_n$ of* (1.56)*, the coefficients $V_0, V_1, \ldots, V_m$ of* (1.57) *such that*

$$\|P_n - R_m\|_2^2 \equiv \langle P_n - R_m, P_n - R_m\rangle_{\alpha,\beta}$$

is minimized subject to the constraints (1.58) *are given by*

$$V_i = \binom{n}{i}\binom{m}{i}^{-1} \sum_{h=0}^{i-1} (-1)^{i+h}\binom{i}{h} V_h \quad (i = 0,1,\ldots,k-1), \tag{1.63}$$

$$V_{m-i} = (-1)^i \binom{n}{i}\binom{m}{i}^{-1} \sum_{h=0}^{i} (-1)^{i+h}\binom{i}{h} W_{n-i+h} - \sum_{h=1}^{i} (-1)^h \binom{i}{h} V_{m-i+h} \tag{1.64}$$
$$(i = 0,1,\ldots,l-1),$$

$$V_i = \binom{m-k-l}{i-k}\binom{m}{i}^{-1} (n-k-l)!^{-1} \sum_{j=k}^{n-l} v_j \Psi_{ij} \quad (i = k, k+1, \ldots, m-l), \tag{1.65}$$

where

$$v_j := (\alpha+2l+1)_{n-l-j}(\beta+2k+1)_{j-k}\left[\binom{n}{j} - \Big(\sum_{h=0}^{k-1} + \sum_{h=m-l+1}^{m}\Big)\binom{n-m}{j-h}\binom{m}{h} V_h\right]$$
$$(j = k, k+1, \ldots, n-l), \tag{1.66}$$

and Ψ_{ij} is defined as in Theorem 1.77.

$$\begin{array}{ccccccc} & 0 & 0 & \cdots & 0 & & \\ 0 & \Psi_{kk} & \Psi_{k,k+1} & \cdots & \Psi_{k,n-l} & 0 \\ 0 & \Psi_{k+1,k} & \Psi_{k+1,k+1} & \cdots & \Psi_{k+1,n-l} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \Psi_{m-l,k} & \Psi_{m-l,k+1} & \cdots & \Psi_{m-l,n-l} & 0 \\ & 0 & 0 & \cdots & 0 & \end{array}$$

Figure 1.10: The Ψ table.

The elements Ψ_{ij} $(k \le i \le m-l, k \le j \le n-l)$, with using the convention that $\Psi_{ij} = 0$ for other choices of i, j (cf. (1.61)), can be arranged into a Ψ table (cf. Figure 1.10). In order to efficiently use Theorem 1.79, one needs to compute the elements of the Ψ table fast, i.e., in the $O(nm)$ time. To do so, the following recurrence relations are useful.

Theorem 1.80 ([97, Theorem 5.1])**.** *The quantities*

$$\Psi_{kk}, \Psi_{k,k+1}, \dots, \Psi_{k,n-l}$$

satisfy the recurrence relation

$$\begin{cases} \Psi_{kk} := (k+l-n)C \displaystyle\sum_{h=0}^{m-k-l} (-1)^h \binom{m-k-l}{h} \frac{(m+k+l+\sigma+1)_h}{(k+l+h-n)(\alpha+2l+1)_h}, \\ \Psi_{k,k+1} := (-1)^{m-k-l} C \dfrac{(\beta+2k+2)_{m-k-l}}{(\alpha+2l+1)_{m-k-l}}, \\ \Psi_{k,j+1} = E(j)\Psi_{kj} + F(j)\Psi_{k,j-1} \qquad (k+1 \le j \le n-l-1), \end{cases} \tag{1.67}$$

where

$$\begin{aligned} C &:= \frac{(n-k-l)!(1-n)_m(m+\sigma+1)_{k+l}}{(m-k-l)!(1-n)_{k+l}(m+\sigma+1)_n}, \\ E(j) &:= 1 - F(j) - \frac{(m-k-l)(m+k+l+\sigma+1)}{(n-l-j)(j+k+\beta+1)}, \\ F(j) &:= \frac{(k-j+1)(n+l-j+\alpha+1)}{(n-l-j)(j+k+\beta+1)}, \end{aligned}$$

and $\sigma := \alpha + \beta + 1$.

Theorem 1.81 ([97, Theorem 5.2])**.** *The quantities* Ψ_{ij} *satisfy the following recurrence relation:*

$$\Psi_{i+1,j} = B(m,i)^{-1}\Big(A(n,j)\Psi_{i,j-1} + [C(m,i) - C(n,j)]\Psi_{ij} + B(n,j)\Psi_{i,j+1} - A(m,i)\Psi_{i-1,j}\Big), \tag{1.68}$$

where

$$\begin{cases} A(r,s) := (k-s)(r+l-s+\alpha+1), \\ B(r,s) := (s+l-r)(k+s+\beta+1), \\ C(r,s) := A(r,s) + B(r,s). \end{cases} \tag{1.69}$$

Algorithm 1.3 gives the full procedure for reducing a Bézier curve's degree with constraints in $O(nm)$ time.

Remark 1.82 ([97, Remark 5.3])**.** *If $\alpha = \beta$ and $k = l$, the Ψ table satisfies the following symmetry property:*

$$\Psi_{ij} = \Psi_{m-i,n-j} \qquad (k \leq i \leq m-k, k \leq j \leq n-k).$$

It is thus sufficient to compute half of the Ψ table using Theorems 1.80 and 1.81.

Algorithm 1.3 Degree reduction of a Bézier curve using the Ψ table organized using Theorems 1.80 and 1.81

procedure BezierDegRedPsi($m, n, k, l, \alpha, \beta, \mathsf{W}_0, \ldots, \mathsf{W}_n$)
  $\Psi \leftarrow$ `Matrix(`m, n`)`
  for $j \leftarrow 0, k-1$ **do**
    $\mathsf{V}_j \leftarrow$ `Eq.` (1.63)
  end for
  for $j \leftarrow m, m-l+1$ **do**
    $\mathsf{V}_j \leftarrow$ `Eq.` (1.64)
  end for
  for $j \leftarrow k, n-l$ **do**
    $v_j \leftarrow$ `Eq.` (1.66)
  end for
  for $j \leftarrow k, n-l$ **do**
    $\Psi_{kj} \leftarrow$ `recurrence relation` (1.67)
  end for
  for $i \leftarrow k, m-l-1$ **do**
    for $j \leftarrow k, n-l$ **do**
      $\Psi_{i+1,j} \leftarrow$ `recurrence relation` (1.68)
    end for
  end for
  for $j \leftarrow k, m-l$ **do**
    $\mathsf{V}_j \leftarrow$ `Eq.` (1.65)
  end for
  return V
end procedure

1.7.5 Approximating any parametric curve with a Bézier curve

By applying Theorem 1.38, one can use dual projection to approximate any parametric curve with a Bézier curve. Let $\mathsf{F} : [0,1] \to \mathbb{E}^d$ be a parametric curve such that

$$\mathsf{F}(t) := (\mathsf{F}_1(t), \mathsf{F}_2(t), \ldots, \mathsf{F}_d(t)) \qquad (\mathsf{F}_i : [0,1] \to \mathbb{R};\ i = 1, 2, \ldots, d)$$

and $n \in \mathbb{N}$. Let $\mathsf{P}_n^* : [0,1] \to \mathbb{E}^d$ be an optimal polynomial parametric curve of degree n that minimizes the error

$$\int_0^1 (1-t)^\alpha t^\beta \|\mathsf{F}(t) - \mathsf{P}_n^*(t)\|_2^2 \mathrm{d}t$$

(cf. (1.59)). The control points $\mathsf{I}_0, \mathsf{I}_1, \ldots, \mathsf{I}_n \in \mathbb{E}^d$ of P_n^*,

$$\mathsf{I}_k \equiv (\mathsf{I}_{k1}, \mathsf{I}_{k2}, \ldots, \mathsf{I}_{kd}) \qquad (0 \le k \le n),$$

are given, using the dual projection, by scalar products:

$$\mathsf{I}_{kj} \equiv \langle \mathsf{F}_j, D_k^n \rangle_{\alpha,\beta} = \int_0^1 (1-t)^\alpha t^\beta \mathsf{F}_j(t) D_k^n(t;\alpha,\beta) \mathrm{d}t \qquad (0 \le k \le n; 1 \le j \le d). \tag{1.70}$$

Thus, we have

$$\mathsf{P}_n^*(t) = \sum_{k=0}^{n} \mathsf{I}_k B_k^n(t).$$

Certainly, if F is a polynomial curve of degree n then this method gives an exact Bernstein-Bézier representation of F.

If the formula for F does not allow to compute the integrals symbolically or no such formula is available, a quadrature approach (see, e.g., [22, §5]) can be used to find the approximate values of the integrals (1.70) for all $k = 0, 1, \ldots, n$ and $j = 1, 2, \ldots, d$. This approach creates the need for a method which allows to find the value of $D_k^n(\cdot;\alpha,\beta)$ for many nodes $t_0, t_1, \ldots, t_M \in [0,1]$, for all $0 \le k \le n$. An efficient algorithm with $O(Mn)$ complexity, for doing that will be introduced in Chapter 5.

1.8 Bézier surfaces

Bézier surfaces are an extension of the ideas used to develop Bézier curves. Instead of using the univariate Bernstein polynomials as the basis functions, bivariate versions can be used. The two most prominent Bézier surface types are rectangular (tensor product) and triangular Bézier surfaces (sometimes also called Bézier patches).

According to Farin [36, p. 245, p. 309], both rectangular and triangular Bézier surfaces were explored by de Casteljau at Citroën [29, 30]. He considered triangular surfaces as a more *natural* extension than rectangular surfaces. For a more detailed overview of triangular Bézier patches, see the surveys [6, 28, 35], as well as the papers cited therein.

Here, the Bézier surfaces will be considered only in their rational variant. To get the polynomial Bézier surfaces, it is sufficient to assume that all weights are equal, which allows to eliminate them from the equations.

Many fundamental operations which can be performed on a (polynomial or rational) Bézier curve can also be done for (polynomial or rational) Bézier surfaces, e.g., creating composite surfaces, subdivision, degree elevation or reduction. The algorithms for these operations can be found in, e.g., the articles [61, 98] and the papers cited therein, as well as the books [36, 54, 78]. Just as in the case of one-dimensional Bézier curves, algorithms analogous to the de Casteljau algorithm can be used to evaluate a point on a Bézier surface. In Section 2.4, more efficient algorithms for such evaluation will be given.

1.8.1 Rational rectangular Bézier surfaces

Definition 1.83 ([78, §9.2])**.** *For $m, n, d \in \mathbb{N}$, weights $\omega_{ij} > 0$ and control points $\mathsf{W}_{ij} \in \mathbb{E}^d$ $(0 \le i \le m,\ 0 \le j \le n)$,* a rational rectangular Bézier surface *is a function $\mathsf{S}_{mn} : [0,1]^2 \to \mathbb{E}^d$*

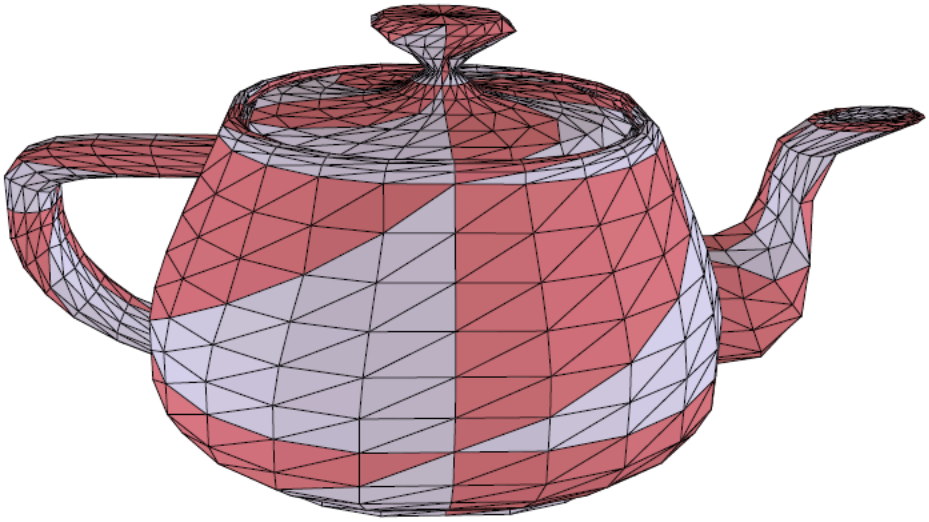

Figure 1.11: The Utah Teapot modeled using 64 composite triangular Bézier patches. Image taken from [85, Fig. 14].

given by the formula

$$\mathsf{S}_{mn}(s,t) := \frac{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{n} \omega_{ij}\mathsf{W}_{ij}B_i^m(s)B_j^n(t)}{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{n} \omega_{ij}B_i^m(s)B_j^n(t)}.$$

Obviously,

$$B_i^m(s)B_j^n(t) \geq 0 \qquad (s,t \in [0,1])$$

and

$$\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(s)B_j^n(t) = \sum_{i=0}^{m} B_i^m(s) \sum_{j=0}^{n} B_j^n(t) = 1.$$

This means that $\mathsf{S}_{mn}(s,t)$ is a convex combination of its control points for all $s,t \in [0,1]$. Thus, the entire surface is within the convex hull (cf. Definition 1.8) of the control points.

To compute a point on a rational rectangular Bézier surface, one can use the rational rectangular de Casteljau algorithm. Just as the rational de Casteljau algorithm (cf. §1.7.1), it is based on Eq. (1.28). An implementation is given in Algorithm 1.4 (cf. Algorithm 1.2).

The computational complexity of Algorithm 1.4 is $O((n^2+m)md)$. When $m \geq n$, fewer arithmetic operations have to be performed. Thus, if $n > m$, one can consider evaluating the rational rectangular Bézier surface with control points

$$\mathsf{V}_{ij} := \mathsf{W}_{ji} \qquad (0 \leq i \leq n;\ 0 \leq j \leq m)$$

at point (t,s). It can easily be checked that the resulting point is exactly $\mathsf{S}_{mn}(s,t)$.

Algorithm 1.4 Rational rectangular de Casteljau algorithm

procedure RATRECTBEVAL($m, n, s, t, \omega, \mathsf{W}$)
  $t_1 \leftarrow 1 - t$
  for $i \leftarrow 0, m$ **do**
    for $j \leftarrow 0, n$ **do**
      $w_{ij} \leftarrow \omega_{ij}$
      $\mathsf{Q}_{ij} \leftarrow \mathsf{W}_{ij}$
    end for
  end for
  for $k \leftarrow n-1, 0$ **do**
    for $j = 0, k$ **do**
      for $i = 0, m$ **do**
        $u \leftarrow t \cdot w_{i,j+1} + t_1 \cdot w_{ij}$
        $\mathsf{Q}_{ij} \leftarrow \dfrac{t \cdot w_{i,j+1}}{u} \cdot \mathsf{Q}_{i,j+1} + \dfrac{t_1 \cdot w_{ij}}{u} \cdot \mathsf{Q}_{ij}$
        $w_{ij} \leftarrow u$
      end for
    end for
  end for
  return `RatBEval`$(m, s, w_{\cdot 0}, \mathsf{Q}_{\cdot 0})$
end procedure

1.8.2 Rational triangular Bézier surfaces

A second type of rational Bézier surfaces are rational triangular Bézier surfaces. Unlike rectangular surfaces, which had the domain of $[0,1]^2$, the triangular surfaces' domain is $\{(s,t) : \ s,t \geq 0, s+t \leq 1\}$. Because of that, it operates using a different family of basis functions.

Definition 1.84 ([78, §10.1])**.** *The* (i,j)th triangular Bernstein polynomial of degree n *is given by the following formula:*

$$B_{ij}^n(s,t) := \frac{n!}{i!j!(n-i-j)!} s^i t^j (1-s-t)^{n-i-j} \qquad (0 \leq i+j \leq n).$$

The triangular Bernstein polynomials satisfy a recurrence relation which connects triangular Bernstein polynomials of subsequent degrees.

Theorem 1.85 ([36, Eq. (17.8)])**.** *The following relation holds for* $i, j \geq 0$, $i + j \leq n$*:*

$$B_{ij}^n(s,t) = sB_{i-1,j}^{n-1}(s,t) + tB_{i,j-1}^{n-1}(s,t) + (1-s-t)B_{ij}^{n-1}(s,t). \tag{1.71}$$

A convention analogous to the one for univariate Bernstein polynomials (cf. Remark 1.48) is adopted, i.e., if $i < 0 \vee j < 0 \vee i + j > n$*, then* $B_{ij}^n(s,t) \equiv 0$*.*

Remark 1.86 ([78, §10.1])**.** *From Definition 1.84, it is clear that* $B_{ij}^n(s,t) \geq 0$ *for* $(s,t) \in \{(s,t) : \ s,t \geq 0; \ s+t \leq 1\}$*.*

Theorem 1.87 ([78, §10.1]). *The triangular Bernstein polynomials satisfy the partition of unity property, i.e.,*

$$\sum_{i=0}^{n}\sum_{j=0}^{n-i} B_{ij}^{n}(s,t) \equiv 1$$

for $n \in \mathbb{N}$ and $s, t \in \mathbb{R}$.

Remark 1.86 and Theorem 1.87 imply that the triangular Bernstein polynomials are a sound choice for a basis which can be used to construct a rational triangular parametric surface.

Definition 1.88 ([78, §10.2]). *For $n, d \in \mathbb{N}$, weights $v_{ij} > 0$ and control points $\mathsf{V}_{ij} \in \mathbb{E}^d$ $(0 \le i \le n,\ 0 \le j \le n-i)$,* a rational triangular Bézier surface *is a function $\mathsf{T}_n : \{(s,t) : s,t \ge 0,\ s+t \le 1\} \to \mathbb{E}^d$ given by the formula*

$$\mathsf{T}_n(s,t) := \frac{\displaystyle\sum_{i=0}^{n}\sum_{j=0}^{n-i} v_{ij}\mathsf{V}_{ij}B_{ij}^{n}(s,t)}{\displaystyle\sum_{i=0}^{n}\sum_{j=0}^{n-i} v_{ij}B_{ij}^{n}(s,t)}.$$

For (s,t) in the domain of T_n, the point $\mathsf{T}_n(s,t)$ is a convex combination of the surface's control points. Thus, the entire surface T_n is within the convex hull (cf. Definition 1.8) of the control points.

A point on a rational triangular Bézier surface can be computed using the rational triangular de Casteljau algorithm. The idea behind the algorithm is similar to the one used in the case of the rational de Casteljau algorithm for curves. In this case, however, the reduction of the degree of the basis functions is achieved using Eq. (1.71) (see, e.g., [78, §10.4]). An implementation of the rational triangular de Casteljau algorithm is given in Algorithm 1.5. Its complexity is $O(n^3 d)$.

Algorithm 1.5 Rational triangular de Casteljau algorithm

procedure RatTriBEval(n, s, t, v, V)
  $st1 \leftarrow 1 - s - t$
  for $i \leftarrow 0, n$ **do**
    for $j \leftarrow 0, n - i$ **do**
      $w_{ij} \leftarrow v_{ij}$
      $\mathsf{Q}_{ij} \leftarrow \mathsf{V}_{ij}$
    end for
  end for
  for $k \leftarrow 1, n$ **do**
    for $i = 0, n - k$ **do**
      for $j = 0, n - k - i$ **do**
        $u \leftarrow s \cdot w_{i+1,j} + t \cdot w_{i,j+1} + st1 \cdot w_{ij}$
        $\mathsf{Q}_{ij} \leftarrow \frac{s \cdot w_{i+1,j}}{u} \cdot \mathsf{Q}_{i+1,j} + \frac{t \cdot w_{i,j+1}}{u} \cdot \mathsf{Q}_{i,j+1} + \frac{st1 \cdot w_{ij}}{u} \cdot \mathsf{Q}_{ij}$
        $w_{ij} \leftarrow u$
      end for
    end for
  end for
  return Q_{00}
end procedure

1.9 B-splines

Bézier-type objects are, by far, not the only parametric objects (curves or surfaces) used in computer-aided design and modeling. Despite their elegance and some desirable properties, Bernstein polynomials have a significant drawback. For any $n, i \in \mathbb{N}$ such that $i \leq n$, the value of a Bernstein polynomial $B_i^n(t)$ (cf. Definition 1.43) is non-zero for all $t \in (0, 1)$. In practice, when operating on a Bézier curve (cf. Definition 1.66), any change in one control point's position modifies the curve over its whole length.

To address this issue, B-spline functions can be used. They are constructed in a way which eliminates this drawback. When used as a basis for parametric curves (known as B-spline curves), they cause any change to a control point to only have a local effect on a curve. An example basis is presented in Figure 1.12, while Figure 1.13 gives an example of such a B-spline curve.

Splines are commonly used in a wide variety of applications, e.g., in computer-aided geometric design, approximation theory and numerical analysis. See, e.g., [33, 36, 41, 54, 77, 78]. The way in which the splines and B-splines are introduced in Sections §1.9.1-1.9.2 relies heavily on [33, §1.1-1.2].

1.9.1 Spline functions

Definition 1.89 ([33, Definition 1.1])**.** *A function s, defined on a finite interval $[a, b]$, is called a* spline function of degree *m, having* as knots *the strictly increasing sequence t_j ($j = 0, 1, \ldots, n$) such that $t_0 = a$ and $t_n = b$, if the following two conditions are satisfied.*

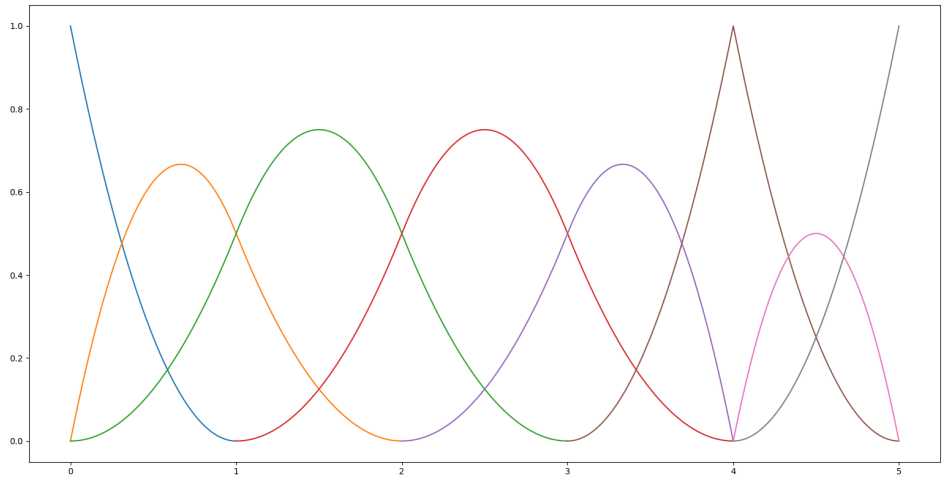

Figure 1.12: An example of a polynomial basis over the interval $[0, 5]$, consisting of B-spline basis functions of degree 2. The knot sequence is $\{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$. Note that at any point, there are at most three non-zero basis functions. Plot recreated from [72, Fig. 2].

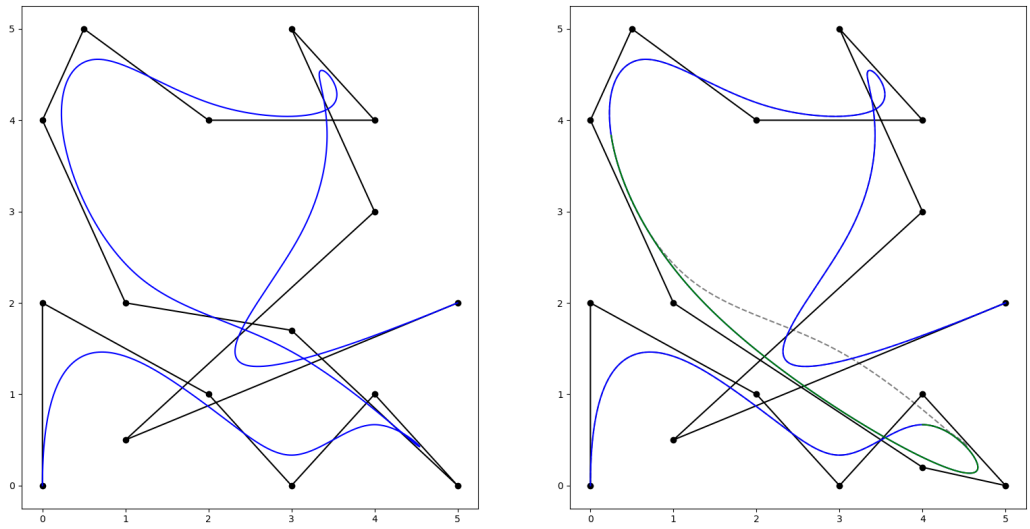

Figure 1.13: Changing the coordinates of one control point of a B-spline curve results only in a local change to the curve. To the left, a cubic B-spline curve is presented, along with its control points. To the right, a new curve with its control points is given, differing only in one control point's coordinates. The pieces of the curve that remain unchanged are drawn in blue. The modified part of the curve is drawn in green, while the grey dash line depicts the curve before moving the control point.

1. *On each knot interval* $[t_j, t_{j+1}]$, s *is given by a polynomial of degree* m *at most:*

$$s_{|[t_j,t_{j+1}]} \in \Pi_m \qquad (j = 0, 1, \ldots, n-1).$$

2. *The function* s *and its derivatives up to order* $m-1$ *are all continuous on* $[a, b]$*:*

$$s \in C^{m-1}[a, b].$$

Remark 1.90. *Every polynomial on* $[a, b]$ *of degree* $\leq m$ *is also a spline function of degree* m *on* $[a, b]$*, for any sequence of knots. In general, a spline of degree* m *will, however, be given by a different polynomial of degree* $\leq m$ *in each of the knot intervals.*

In practice, it is sometimes desirable for some knots to be coincident, as will be apparent in §1.9.2.

Remark 1.91 (cf. [33, p. 4])**.** *Definition 1.89 states explicitly that the knots must be strictly increasing. In what follows, it will sometimes be extended to consider splines with coincident knots. In such case, the continuity condition would have to be relaxed in the sense that, for* $\ell \leq m$*, if* $t_{i-1} < t_i = \ldots = t_{i+\ell} < t_{i+\ell+1}$ *(in other words, if* t_i *has multiplicity* $\ell + 1$*),* s *will only be required to have continuous derivatives up to order* $m - 1 - \ell$ *at the point* t_i*.*

The set of all splines of the same degree and with the same knots forms a vector space.

Remark 1.92 ([33, Eq. (1.7)])**.** *Let the space of splines of degree* m *having the knots*

$$\Omega_n := \{t_0, t_1, \ldots, t_n\}$$

be denoted by $\mathcal{S}_m(\Omega_n)$*. Since the spline is a polynomial of degree* $\leq m$ *over each of* n *knot intervals and there are* m *continuity conditions at knots* $t_1, t_2, \ldots, t_{n-1}$*, the dimension of* $\mathcal{S}_m(\Omega_n)$ *is:*

$$\dim(\mathcal{S}_m(\Omega_n)) = (m+1)n - (n-1)m = n + m.$$

Since the spline $s \in \mathcal{S}_m(\Omega_n)$ is piecewise polynomial, one can express it as a polynomial over one of the knot intervals, e.g., in the following form:

$$s(x) := p_{mj}(x) = \sum_{i=0}^{m} a_{i,j}(x - t_j)^i \qquad (t_j \leq x \leq t_{j+1},\ j = 0, 1, \ldots, n-1).$$

While, in some cases, it may be useful to use separate basis functions for each of the knot intervals, one can instead use a common basis for all knot intervals. One of such bases uses the so called *truncated power functions*:

$$(x - c)_+^m := \begin{cases} (x-c)^m & (x \geq c), \\ 0 & (x < c). \end{cases}$$

Remark 1.93 ([33, Eq. (1.9)])**.** *It can be proved that every spline* $s \in \mathcal{S}_m(\Omega_n)$ *has a unique representation of the form*

$$s(x) = \sum_{i=0}^{m} b_i x^i + \sum_{i=1}^{n-1} c_i (x - t_i)_+^m.$$

The basis used in Remark 1.93 has a drawback of being ill-conditioned (cf. [33, p. 5]). Instead, a numerically stable and useful, for example in CAGD, B-spline basis can be used.

1.9.2 B-spline functions

In order to introduce a B-spline function, the *generalized divided differences* will be used.

Definition 1.94 ([22, §4.2.1]). *The* generalized divided difference *of a univariate function f at the knots $x_i, x_{i+1}, \ldots, x_k$ (which may be coincident), denoted by $[x_i, x_{i+1}, \ldots, x_k]f$, is defined in the following recursive way:*

$$[x_i, x_{i+1}, \ldots, x_{i+\ell}]f := \begin{cases} \dfrac{[x_{i+1}, \ldots, x_{i+\ell}]f - [x_i, \ldots, x_{i+\ell-1}]f}{x_{i+\ell} - x_i} & (x_i \neq x_{i+\ell}), \\ \dfrac{f^{(\ell)}(x_i)}{\ell!} & (x_i = \ldots = x_{i+\ell}). \end{cases}$$

In particular, $[x_i]f = \dfrac{f^{(0)}(x_i)}{0!} = f(x_i)$.

Definition 1.95 ([78, §5.11]). *The* B-spline function N_{mi} of degree $m \in \mathbb{N}$ *with knots* $t_i \leq t_{i+1} \leq \ldots \leq t_{i+m+1}$ *is defined as*

$$N_{mi}(u) := (t_{i+m+1} - t_i)[t_i, t_{i+1}, \ldots, t_{i+m+1}](t - u)_+^m,$$

where the generalized divided difference acts on the variable t.

Certainly, $N_{mi} \in \mathcal{S}_m(\{t_i, t_{i+1}, \ldots, t_{i+m+1}\})$.

A knot t_j can be both in the knot interval $[t_{j-1}, t_j]$ and $[t_j, t_{j+1}]$. To eliminate the ambiguity, half-open *knot spans* are going to be used.

Definition 1.96 ([77, §2.2]). *The half-open interval, $[t_i, t_{i+1})$, is called the* ith knot span*; it can have zero length, since knots need not be distinct.*

For a given sequence of knots $t_0, t_1, \ldots, t_n$, $n - m$ linearly independent B-spline functions of degree m (N_{mi} for $i = 0, 1, \ldots, n - 1 - m$) can be constructed. To form a basis of $\mathcal{S}_m(\Omega_n)$, $2m$ additional functions are required. They can be obtained by introducing $2m$ *boundary knots* $t_{-m}, t_{-m+1}, \ldots, t_{-1}, t_{n+1}, t_{n+2}, \ldots, t_{n+m}$, such that

$$t_{-m} \leq t_{-m+1} \leq \ldots \leq t_{-1} \leq t_0, \quad t_n \leq t_{n+1} \leq \ldots \leq t_{n+m}.$$

The boundary knots can be chosen arbitrarily. This allows to construct additional B-splines using Definition 1.95, giving $n + m$ B-spline functions:

$$N_{mi} \in \mathcal{S}_m(\Omega_n) \qquad (i = -m, -m + 1, \ldots, n - 1).$$

Theorem 1.97 ([78, §5.7]). *The B-spline functions of degree m with a given knot sequence that do not vanish over an arbitrary knot span $[t_i, t_{i+1})$ are linearly independent over $[t_i, t_{i+1})$.*

A dimension count shows that the B-spline functions $N_{m,-m}, N_{m,-m+1}, \ldots, N_{m,n-1}$ with the knots $t_{-m}, t_{-m+1}, \ldots, t_{n+m}$ form a basis of $\mathcal{S}_m(\Omega_n)$.

Each spline $s \in \mathcal{S}_m(\Omega_n)$ thus has a unique representation in the B-spline form:

$$s(u) = \sum_{i=-m}^{n-1} c_i N_{mi}(u). \tag{1.72}$$

A popular choice for the boundary knots is to make them coincident with t_0 and t_n, i.e.,

$$t_{-m} = t_{-m+1} = \ldots = t_{-1} = t_0 = a, \quad b = t_n = t_{n+1} = \ldots = t_{n+m}. \tag{1.73}$$

In this case,

$$s(a) = c_{-m}, \quad s(b) = c_{n-1}.$$

If $n = 1$ and the boundary knots are coincident, it can be proved that the B-spline basis reduces to the Bernstein-Bézier basis, i.e.,

$$N_{mi}(u) = B^m_{i+m}\Big(\frac{u-a}{b-a}\Big).$$

See Section 1.5.

The continuity conditions for splines hold as well for B-splines.

Theorem 1.98 ([77, Property 2.5])**.** *All derivatives of N_{mi} exist in the interior of a knot span (where it is a polynomial). At a knot N_{mi} is $m-k$ times continuously differentiable, where k is the multiplicity of the knot. Hence, increasing m increases continuity, and increasing knot multiplicity decreases continuity.*

The B-spline functions, like the family of Bernstein polynomials of an arbitrary degree, have properties which make them a good choice for a parameterization of a family of curves.

Theorem 1.99 ([77, Properties 2.3, 2.4, 2.6])**.** *B-spline functions satisfy the following properties.*

1. *$N_{mi}(u) \geq 0$ for all m, i, u (non-negativity).*
2. *For an arbitrary knot span, $[t_j, t_{j+1})$,*

$$\sum_{i=j-m}^{j} N_{mi}(u) = 1$$

   for all $u \in [t_j, t_{j+1})$ (partition of unity).
3. *Except for the case $m = 0$, N_{mi} attains exactly one maximum value.*

1.9.3 Differential and recurrence relations for B-splines

Computing the B-spline functions or their derivatives using Definition 1.95, while possible, is costly. Instead, one can use the recurrence and differential-recurrence relations. One such relation will serve as a foundation for an algorithm which computes the value of a spline curve given in the form (1.72) at a given point.

Theorem 1.100 ([77, Property 2.2])**.** *In any given knot span, $[t_j, t_{j+1})$, at most $m+1$ of the N_{mi} are non-zero, namely the functions $N_{m,j-m}, N_{m,j-m+1}, \ldots, N_{mj}$. In other words, the B-spline function N_{mi} has support $[t_i, t_{m+i+1}]$.*

Remark 1.101. *In the sequel, a convention is adopted that $\dfrac{N_{mk}(u)}{t_{m+k+1} - t_k} := 0$ if $t_k = t_{m+k+1}$.*

The B-spline functions satisfy the following de Boor-Mansfield-Cox recursion formula (see, e.g., [27, §2], [21, Eq. (6.1)], [41, Eq. (7.8)]).

Theorem 1.102. *The B-spline functions satisfy the recurrence relation of the form*

$$N_{mi}(u) = (u - t_i)\frac{N_{m-1,i}(u)}{t_{m+i} - t_i} + (t_{m+i+1} - u)\frac{N_{m-1,i+1}(u)}{t_{m+i+1} - t_{i+1}} \tag{1.74}$$

(cf. Remark 1.101).

Additionally, it can be checked that

$$N_{0i}(u) = \begin{cases} 1 & (u \in [t_i, t_{i+1})), \\ 0 & \text{otherwise.} \end{cases} \tag{1.75}$$

This observation serves as a base for recursive computations with B-spline functions. This, along with Theorem 1.102, allows the computation of a B-spline function or their linear combination. An algorithm to do so will be given in §1.9.5.

Moreover, a differential-recurrence relation between N'_{mi} and $N_{m-1,i}, N_{m-1,i+1}$ is known.

Theorem 1.103 ([77, Eq. (2.7)])**.** *The derivative of a B-spline function can be expressed as*

$$N'_{mi}(u) = m \cdot \left(\frac{N_{m-1,i}(u)}{t_{m+i} - t_i} - \frac{N_{m-1,i+1}(u)}{t_{m+i+1} - t_{i+1}} \right) \tag{1.76}$$

(cf. Remark 1.101).

1.9.4 B-spline curves

From the graphical perspective, B-spline curves are of use and interest due to their properties. While they have good numerical properties, their advantage over Bézier curves (cf. Section 1.6) lies in their *locality* — a change to one of the curve's control points changes only a fragment of the curve which is influenced by the corresponding B-spline function's support. See Figures 1.12 and 1.13. Due to their properties such as non-negativity and partition of unity, B-spline functions are well-suited to be used as a basis of a parametric curve family, i.e., B-spline curves.

Definition 1.104. *A* B-spline curve of degree m *over the non-empty interval* $[a, b]$ *with knots*

$$t_{-m} \leq \ldots \leq t_0 = a \leq t_1 \leq \ldots \leq b = t_n \leq \ldots \leq t_{n+m}$$

and control points $\mathsf{W}_{-m}, \mathsf{W}_{-m+1}, \ldots, \mathsf{W}_{n-1} \in \mathbb{E}^d$ *is defined as*

$$\mathsf{S}(t) := \sum_{i=-m}^{n-1} N_{mi}(t)\mathsf{W}_i \qquad (t \in [a, b]).$$

Unlike the case of Bézier curves, a modification of a B-spline curve's control point has only *local* effect on the curve.

Remark 1.105 ([77, Property 3.6])**.** *From Theorem 1.100, it folows that moving the control point* W_i *changes* S *only in the interval* $[t_i, t_{i+m+1})$.

Theorem 1.106 ([77, Property 3.5])**.** *The B-spline curve S satisfies the convex hull property, i.e.,*

$$\mathsf{S}([t_0, t_n]) \subseteq conv\{\mathsf{W}_{-m}, \mathsf{W}_{-m+1}, \ldots, \mathsf{W}_{n-1}\}.$$

Additionally,

$$\mathsf{S}([t_i, t_{i+1})) \subseteq conv\{\mathsf{W}_{i-m}, \mathsf{W}_{i-m+1}, \ldots, \mathsf{W}_i\} \qquad (i = 0, 1, \ldots, n-1).$$

Remark 1.107 ([77, Property 3.1])**.** *Bézier curves are a particular subtype of B-spline curves. More precisely, when $n = 1$, $t_{-m} = t_{-m+1} = \ldots = t_0 = 0$ and $t_1 = t_2 = \ldots = t_{m+1} = 1$,*

$$N_{m,i-m}(t) = B_i^m(t) \qquad (t \in [0, 1],\ i = 0, 1, \ldots, m).$$

This means that, for control points $\mathsf{W}_{-m}, \mathsf{W}_{-m+1}, \ldots, \mathsf{W}_0$,

$$\mathsf{S}(t) = \sum_{i=-m}^{0} N_{mi}(t)\mathsf{W}_i = \sum_{i=0}^{m} B_i^m(t)\mathsf{W}_{i-m} \qquad (t \in [0, 1]).$$

1.9.5 The de Boor-Cox algorithm

Theorem 1.102 and Eq. (1.75) can be used to compute a point on a B-spline curve. This approach, applied to explicitly compute the values of B-spline functions, has been proposed by de Boor in [27, p. 55–57]. The algorithm given in [27, p. 57–59] (see also [36, Eq. (8.3)]), which directly computes a point on a B-spline curve is known as the de Boor-Cox algorithm and has $O(dm^2)$ computational complexity.

Let $u \in [t_j, t_{j+1})$. Then, by Definition 1.104 and Theorem 1.100,

$$\mathsf{S}(u) = \sum_{i=j-m}^{j} N_{mi}(u)\mathsf{W}_i.$$

If $m = 0$ then, certainly, $\mathsf{S}(u) = \mathsf{W}_j$. If $m > 0$, however, Theorem 1.102 can be applied to get

$$\mathsf{S}(u) = \sum_{i=j-m}^{j} (u - t_i)\frac{N_{m-1,i}(u)}{t_{m+i} - t_i}\mathsf{W}_i + \sum_{i=j-m+1}^{j+1} (t_{m+i} - u)\frac{N_{m-1,i}(u)}{t_{m+i} - t_i}\mathsf{W}_{i-1},$$

which can be simplified to

$$\mathsf{S}(u) = \sum_{i=j-m+1}^{j} N_{m-1,i}(u)\Big(\frac{u - t_i}{t_{m+i} - t_i}\mathsf{W}_i + \frac{t_{m+i} - u}{t_{m+i} - t_i}\mathsf{W}_{i-1}\Big).$$

Let us take a look at the quantities $\dfrac{u - t_i}{t_{m+i} - t_i}, \dfrac{t_{m+i} - u}{t_{m+i} - t_i}$. Certainly,

$$t_i \le t_j \le u < t_{j+1} \le t_{m+i},$$

thus $\dfrac{u - t_i}{t_{m+i} - t_i}, \dfrac{t_{m+i} - u}{t_{m+i} - t_i} \ge 0$. Additionally,

$$\frac{u - t_i}{t_{m+i} - t_i} + \frac{t_{m+i} - u}{t_{m+i} - t_i} = 1.$$

Thus

$$\frac{u-t_i}{t_{m+i}-t_i}\mathsf{W}_i + \frac{t_{m+i}-u}{t_{m+i}-t_i}\mathsf{W}_{i-1}$$

is a convex combination of W_{i-1} and W_i.

Corollary 1.108. *Let $u \in [t_j, t_{j+1})$, and*

$$\begin{cases} \mathsf{W}_i^{(0)} := \mathsf{W}_i & (i = j-m, j-m+1, \ldots, j), \\ \mathsf{W}_i^{(k)} := \dfrac{u-t_i}{t_{m+i+1-k}-t_i}\mathsf{W}_i^{(k-1)} + \dfrac{t_{m+i+1-k}-u}{t_{m+i+1-k}-t_i}\mathsf{W}_{i-1}^{(k-1)} & (1 \le k \le m;\ j-m+k \le i \le j). \end{cases}$$

Then $\mathsf{S}(u) = \mathsf{W}_j^{(m)}$.

The recurrence scheme given in Corollary 1.108 is the foundation of the de Boor-Cox algorithm. A more concrete implementation is presented in Algorithm 1.6, where $\mathtt{t} := t_{-m}, t_{-m+1}, \ldots, t_{n+m}$.

Algorithm 1.6 The de Boor-Cox algorithm

```
1:  procedure BSplineEval(m, u, t, W)
2:      j ← GetKnotSpan(u, t)
3:      for i ← j − m, j do
4:          W_i^(0) ← W_i
5:      end for
6:      for k ← 1, m do
7:          for i ← j − m + k, j do
8:              a ← (u − t_i) / (t_{m+i+1−k} − t_i)
9:              W_i^(k) ← a · W_i^(k−1) + (1 − a) · W_{i−1}^(k−1)
10:         end for
11:     end for
12:     return W_j^(m)
13: end procedure
```

Chapter 2

Fast evaluation of Bézier-type objects

For a given $t \in [0, 1]$, the corresponding point on a rational Bézier curve can be computed using the de Casteljau algorithm (see Section 1.7 and, in particular, Algorithms 1.1 and 1.2). The classic de Casteljau algorithm for evaluating a point on a rational or polynomial Bézier curve has a geometric interpretation, good numerical properties (cf. [70]), and computes only convex combinations of points in $\mathbb{E}^d$. Such properties are desirable from a numerical and geometric point of view.

However, the computational complexity of these algorithms is higher than the lower bound of that problem, which is achieved by algorithms based on the Horner's scheme. One of them computes the coordinates of the point on a polynomial Bézier curve in $\mathbb{E}^d$ is to use the algorithm proposed in [89] for evaluating a polynomial p given in the form

$$p(t) := \sum_{k=0}^{n} p_k t^k (1-t)^{n-k} \quad (p_k \in \mathbb{R})$$

d times (once for each dimension). This method has $O(dn)$ computational complexity and $O(1)$ memory complexity. It uses the concept of Horner's rule (see, e.g., [22, Eq. (1.2.2)], [78, §2.3]). This approach can be adapted for rational Bézier curves. For numerical reasons, using the Horner's rule is not recommended (see [23, 39, 40, 77]).

Other methods for evaluating a Bézier curve are also known. See, e.g., [8] or [74], where the case of Bézier surfaces was also studied (cf. Section 2.4), and the papers cited therein.

The faster algorithms do not, however, have good numerical properties or a geometric interpretation and cannot be expressed as a series of convex operations on points in $\mathbb{E}^d$. An algorithm which combines good numerical properties of the de Casteljau algorithm with the efficiency of the Horner's scheme would be widely useful. Given that in order to render a Bézier curve one has to compute multiple points, such an improvement would allow for a significant reduction of time required for such computations, even for curves of low degrees.

In this chapter, such algorithm will be presented for rational Bézier curves. Section 2.1 contains the reasoning behind the construction of a linear-time algorithm for evaluating Bézier curves, while Section 2.2 focuses on its efficient implementation. Note that, although the presented algorithm is designed with rational Bézier curves in mind, it can be easily applied to polynomial Bézier curves as well.

The approach used in the algorithm can be generalized and used for any *rational parametric object* which satisfies certain conditions. This generalization is expanded upon in Section 2.3.

This generalized approach can be, for instance, used in case of rectangular and triangular Bézier surfaces. An outline of the recommended approach to their computation is given in Section 2.4.

This chapter contains the results obtained by Woźny and Chudy in [96], along with some new observations, mainly in generalizing some formulas for rational Bézier curves and approaching the computations for Bézier surfaces. Although the idea used in the paper and, consequently, in this chapter, appears to be simple, to the best of the author's knowledge, it was not presented or used previously. Despite this simplicity, the observation has significant ramifications with respect to efficient computations of Bézier-type objects and other *rational parametric objects*.

The algorithm presented in [96] and in this chapter can be used, as seen in [83], to precompute some quantities to further accelerate the computations for Bézier curves of very high degree or for computing many points on a single curve — however, at the cost of losing the geometric interpretation.

Additionally, in [84], it is shown how to adapt the algorithm to evaluate algebraic-hyperbolic PH curves (EPH curves) for a fixed parameter t. The approach presented there is to convert an EPH curve into a Bézier curve such that they have the same value at t, and then to evaluate the newly found Bézier curve using the method presented in [96] and in this chapter.

2.1 New algorithm for evaluating Bézier curves

Let $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n$ be points in $\mathbb{E}^d$ $(n, d \in \mathbb{N})$. Recall that B_k^n, the kth *Bernstein polynomial* of degree n, is defined in Eq. (1.24) as

$$B_k^n(t) := \binom{n}{k} t^k (1-t)^{n-k} \quad (0 \leq k \leq n).$$

A *rational Bézier curve* in $\mathbb{E}^d$ with control points $\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_n \in \mathbb{E}^d$ and weights $\omega_0, \omega_1, \ldots, \omega_n \in \mathbb{R}_+$ is given by Eq. (1.49) as

$$\mathsf{R}_n(t) := \frac{\sum_{k=0}^{n} \omega_k \mathsf{W}_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \quad (t \in [0,1]).$$

One can separate the expression for a rational Bézier curve into a sum of two terms:

$$\mathsf{R}_n(t) = \frac{\sum_{k=0}^{i} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)} + \frac{\sum_{k=i+1}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)}.$$

The first term contains all the information from the first $i+1$ control points and their weights. The idea for the algorithm is to represent it as a point with an assigned weight:

$$\mathsf{R}_n(t) = \frac{\sum_{k=0}^{i} \omega_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \cdot \frac{\sum_{k=0}^{i} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{i} \omega_k B_k^n(t)} + \frac{\sum_{k=i+1}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)}.$$

Note that

$$\frac{\sum_{k=0}^{i} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{i} \omega_k B_k^n(t)}$$

is a convex combination of points in $\mathbb{E}^d$ and, therefore, is also a point in $\mathbb{E}^d$.

Let Q_i be defined as

$$\mathsf{Q}_i \equiv \mathsf{Q}_i(t) := \frac{\sum_{k=0}^{i} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{i} \omega_k B_k^n(t)}. \tag{2.1}$$

This allows to express $\mathsf{R}_n(t)$ as a convex combination of points Q_i and $\mathsf{W}_{i+1}, \mathsf{W}_{i+2}, \ldots, \mathsf{W}_n$:

$$\mathsf{R}_n(t) = \frac{\sum_{k=0}^{i} \omega_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \mathsf{Q}_i + \frac{\sum_{k=i+1}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)}.$$

This means that W_i needs used only once to compute Q_i. Note that, for $i = n$,

$$\mathsf{R}_n(t) = \frac{\sum_{k=0}^{n} \omega_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \mathsf{Q}_n + \frac{\sum_{k=n+1}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)} = \mathsf{Q}_n.$$

Now, compare the expressions for $\mathsf{R}_n(t)$ using Q_i and Q_{i-1}, respectively,

$$\frac{\sum_{k=0}^{i} \omega_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \mathsf{Q}_i + \frac{\sum_{k=i+1}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)} = \frac{\sum_{k=0}^{i-1} \omega_k B_k^n(t)}{\sum_{k=0}^{n} \omega_k B_k^n(t)} \mathsf{Q}_{i-1} + \frac{\sum_{k=i}^{n} \omega_k B_k^n(t) \mathsf{W}_k}{\sum_{k=0}^{n} \omega_k B_k^n(t)},$$

which can be simplified to

$$\mathsf{Q}_i = \frac{\sum_{k=0}^{i-1} \omega_k B_k^n(t)}{\sum_{k=0}^{i} \omega_k B_k^n(t)} \mathsf{Q}_{i-1} + \frac{\omega_i B_i^n(t)}{\sum_{k=0}^{i} \omega_k B_k^n(t)} \mathsf{W}_i.$$

Let $h_i(t)$ be defined as

$$h_i(t) := \frac{\omega_i B_i^n(t)}{\sum_{k=0}^{i} \omega_k B_k^n(t)} \qquad (i = 1, 2, \ldots, n), \tag{2.2}$$

with $h_0(t) := 1$. This allows to present the recurrence relation for Q_i in a concise form:

$$\mathsf{Q}_i = (1 - h_i(t))\mathsf{Q}_{i-1} + h_i(t)\mathsf{W}_i. \tag{2.3}$$

In particular, $\mathsf{Q}_0 = \mathsf{W}_0$.

One can use (2.2) to find a recurrence relation which connects $h_{i-1}(t)$ and $h_i(t)$ $(i = 1, 2, \ldots, n)$. Note that

$$\frac{\omega_i B_i^n(t)}{h_i(t)} = \sum_{k=0}^{i} \omega_k B_k^n(t) = \frac{\omega_{i-1} B_{i-1}^n(t)}{h_{i-1}(t)} + \omega_i B_i^n(t),$$

which, after some algebra, gives

$$h_i(t) = \frac{t(n-i+1)\omega_i h_{i-1}(t)}{(1-t)i\omega_{i-1} + t(n-i+1)\omega_i h_{i-1}(t)} \qquad (i = 1, 2, \ldots, n). \tag{2.4}$$

Equations (2.4) and (2.3) together with the values for Q_0 and h_0, form a recursive scheme of the form

$$\begin{cases} h_0 := 1, \quad \mathsf{Q}_0 := \mathsf{W}_0, \\ h_i := \dfrac{\omega_i h_{i-1} t(n-i+1)}{\omega_{i-1} i(1-t) + \omega_i h_{i-1} t(n-i+1)}, \\ \mathsf{Q}_i := (1 - h_i)\mathsf{Q}_{i-1} + h_i \mathsf{W}_i \end{cases} \tag{2.5}$$

for $i = 1, 2, \ldots, n$, with $h_i \equiv h_i(t)$.

Theorem 2.1. *For all $k = 0, 1, \ldots, n$, $t \in [0, 1]$, the quantities h_k and Q_k satisfy:*

1. $h_k \in [0, 1]$,
2. $\mathsf{Q}_k \in \mathbb{E}^d$,
3. $\mathsf{Q}_k \in C_k \equiv conv\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_k\}$ *(consequently, $conv\{\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_k\} \subseteq C_k$).*

Moreover, $\mathsf{R}_n(t) = \mathsf{Q}_n$.

Proof. Recall that, for $t \in (0,1)$, both $B_k^n(t)$ and ω_k are strictly positive for all $0 \le k \le n$. From (2.2), it follows that $h_k(t) \in [0,1]$ for $t \in (0,1)$. It remains to check that, for $t = 0$ and $t = 1$, the property still holds. Clearly, $h_0(t) \equiv 1$. In Eq. (2.2), substituting $t = 0$ gives

$$h_k(0) = \frac{\omega_k \binom{n}{k} 0^k 1^{n-k}}{\omega_0} = 0 \qquad (k = 1, 2, \ldots, n).$$

Using induction, it follows from Eq. (2.4) that

$$h_k(1) = 1 \qquad (k = 1, 2, \ldots, n).$$

Therefore, for $t \in [0,1]$, $h_k(t) \in [0,1]$.

From (2.1), it easily follows that $\mathsf{Q}_k \in \mathbb{E}^d$, $\mathsf{Q}_k \in \mathrm{conv}\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_k\}$, and $\mathsf{Q}_n = \mathsf{R}_n(t)$. □

In each step of the proposed method, the point Q_i (a convex combination of points Q_{i-1} and W_i) is computed. The last point Q_n is equal to the point $\mathsf{R}_n(t)$. The method can be thus used to compute a point on a rational Bézier curve in linear time. Moreover, the method has a geometric interpretation and computes only convex combinations of the control points. Efficient implementation of the method, along with its theoretical and practical costs, is presented in Section 2.2.

One can prove the following result which tells even more about the geometric properties of the new method.

Theorem 2.2. *Let the numbers h_k and the points Q_k $(0 \le k \le n)$ be computed by* (2.5) *for a given $t \in [0,1]$. The point $\mathsf{R}_n(u)$, where $u \in [0,1]$, is in the convex hull of the points $\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_n$ if $u \le t$. It means that*

$$\mathsf{R}_n([0,u]) \subseteq \mathit{conv}\{\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_n\} \quad (u \le t).$$

Proof. Observe that for $t = 0$, $\mathsf{Q}_0 = \mathsf{Q}_1 = \ldots = \mathsf{Q}_n = \mathsf{W}_0 = \mathsf{R}_n(0)$. For $t = 1$, $\mathsf{Q}_i = \mathsf{W}_i$ $(i = 0, 1, \ldots, n)$. In these cases, the theorem follows immediately.

It remains to consider the case $t \in (0,1)$. From Eq. (2.4), it follows that

$$\frac{1 - h_i}{h_i} = h_{i-1}^{-1} \frac{\omega_{i-1} B_{i-1}^n(t)}{\omega_i B_i^n(t)},$$

which, in turn, gives the relation

$$1 - h_i = \frac{h_i}{h_{i-1}} \frac{\omega_{i-1} B_{i-1}^n(t)}{\omega_i B_i^n(t)}. \tag{2.6}$$

This relation can be applied to Eq. (2.3) to obtain a subtraction-free version of the recursive scheme.

Eq. (2.3), combined with Eq. (2.6), provides an expression of W_i in terms of Q_i and Q_{i-1}:

$$\mathsf{W}_i = \frac{1}{h_i} \mathsf{Q}_i - \frac{1}{h_{i-1}} \frac{\omega_{i-1}(1-t)i}{\omega_i t(n-i+1)} \mathsf{Q}_{i-1} \qquad (i = 1, 2, \ldots, n).$$

Additionally, $\mathsf{W}_0 = \mathsf{Q}_0$. Now, one can substitute the points W_i in the expression for $\mathsf{R}_n(u)$ (cf. (1.49)):

$$\mathsf{R}_n(u) = \frac{\omega_0 B_0^n(u)\mathsf{Q}_0 + \sum_{i=1}^{n} \omega_i B_i^n(u)\Big(\frac{1}{h_i}\mathsf{Q}_i - \frac{1}{h_{i-1}}\frac{\omega_{i-1}(1-t)i}{\omega_i t(n-i+1)}\mathsf{Q}_{i-1}\Big)}{\sum_{k=0}^{n} \omega_k B_k^n(u)}.$$

After some algebra one gets

$$\mathsf{R}_n(u) = \frac{\frac{\omega_n}{h_n} B_n^n(u)\mathsf{Q}_n + \sum_{i=0}^{n-1} \Big(1 - \frac{u(1-t)}{(1-u)t}\Big)\frac{\omega_i}{h_i} B_i^n(u)\mathsf{Q}_i}{\sum_{k=0}^{n} \omega_k B_k^n(u)}. \tag{2.7}$$

If $u, t \in (0,1)$ and $u \le t$ then $1 - \dfrac{u(1-t)}{(1-u)t} \ge 0$, thus the right-hand side of Eq. (2.7) is a convex combination of points Q_i. □

This mimics a similar property of the de Casteljau algorithm and may come useful, e.g., in detecting curve intersections (see, e.g., [90]).

The recursive scheme presented in this section can also be used for polynomial Bézier curves. In this case, all weights ω_i are equal. This allows to simplify the recurrence relations:

$$\begin{cases} h_0 := 1, \quad \mathsf{Q}_0 := \mathsf{W}_0, \\ h_i := \dfrac{h_{i-1} t(n-i+1)}{i(1-t) + h_{i-1} t(n-i+1)}, \\ \mathsf{Q}_i := (1-h_i)\mathsf{Q}_{i-1} + h_i \mathsf{W}_i \end{cases} \tag{2.8}$$

for $i = 1, 2, \ldots, n$. Similarly, $\mathsf{Q}_n = \mathsf{P}_n(t)$ (cf. (1.48)). Figure 2.1 illustrates the new method in case of a planar polynomial Bézier curve of degree $n = 5$.

The proposed method can find additional application in curve subdivision (cf., e.g., Section 1.7.2 and [36, §5.4]). Let $u \in (0,1)$ be fixed. Using Theorem 1.73, the points

$$\mathsf{V}_i := \sum_{k=0}^{i} B_k^i(u)\mathsf{W}_k \quad (0 \le i \le n)$$

are the control points of the polynomial Bézier curve P_n^L that is the *left part* of the Bézier curve (1.48) with $t \in [0,u]$, i.e., $\mathsf{P}_n^L[0,1] \equiv P_n[0,u]$. One can check that

$$\begin{cases} \mathsf{V}_i = \sum_{k=0}^{i} h_k^{-1} \dfrac{n-i}{n-k} B_k^i(u)\mathsf{Q}_k \quad (0 \le i \le n-1), \\ \mathsf{V}_n = \mathsf{Q}_n. \end{cases}$$

where the numbers h_k and the points Q_k $(0 \le j \le n)$ are computed using (2.8) with $t := u$.

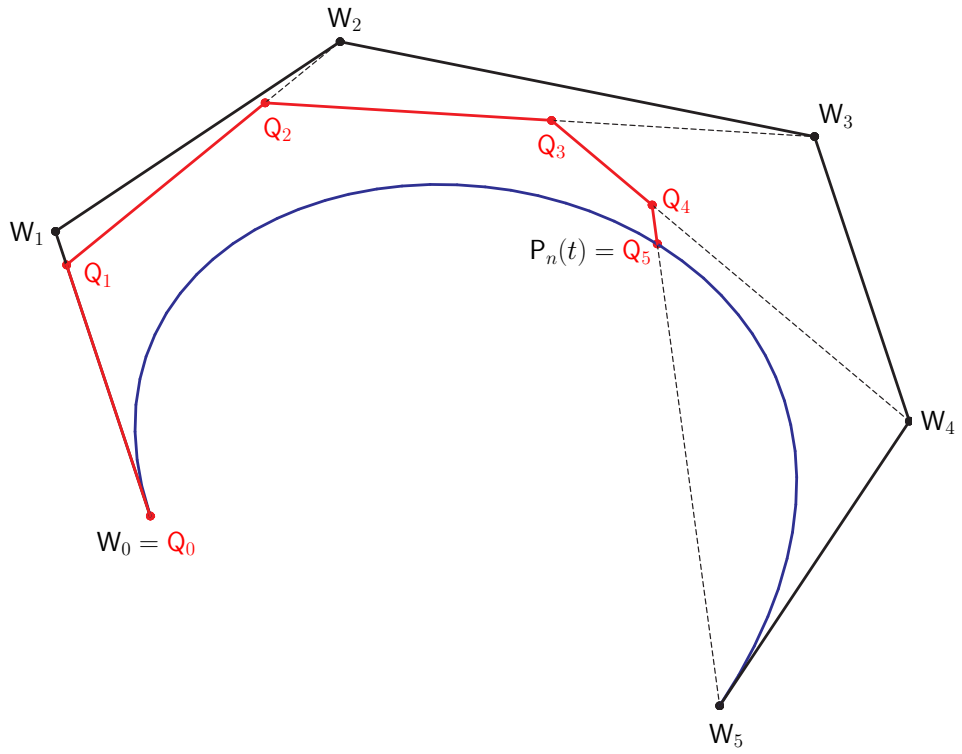

Figure 2.1: Computation of a point on a planar polynomial Bézier curve of degree $n = 5$ using the new method. The method computes much fewer intermediate points compared to the de Casteljau algorithm (cf. Figure 1.9 which illustrates using it for the same curve and t).

A similar result can be obtained for rational Bézier curves. Theorem 1.72 gives the control points V_i along with their corresponding weights v_i of the *left part* R_n^L of the rational Bézier curve R_n:

$$\mathsf{V}_i := \frac{\sum_{k=0}^{i} \omega_k B_k^i(u) \mathsf{W}_k}{\sum_{k=0}^{i} \omega_k B_k^i(u)}, \qquad v_i := \sum_{k=0}^{i} \omega_k B_k^i(u),$$

i.e., $\mathsf{R}_n^L[0,1] \equiv \mathsf{R}_n[0,u]$. Using the same approach as in the polynomial case, the points V_i can be expressed as

$$\begin{cases} \mathsf{V}_i = \dfrac{\sum_{k=0}^{i} h_k^{-1} \omega_k \dfrac{n-i}{n-k} B_k^i(u) \mathsf{Q}_k}{\sum_{k=0}^{i} \omega_k B_k^i(u)} & (0 \le i \le n-1), \\ \mathsf{V}_n = \mathsf{Q}_n, \end{cases}$$

where the numbers h_k and the points Q_k $(0 \le j \le n)$ are computed using (2.5) with $t := u$.

2.2 Implementation and cost

In this section, efficient and numerically safe implementations of the method given in Section 2.1 will be presented. The new algorithms allow to evaluate a single curve at a single point with $O(dn)$ computational complexity and $O(1)$ memory complexity.

Algorithm 2.1 First implementation

```
1: procedure NewRatBEval1(n, t, ω, W)
2:     h ← 1
3:     u ← 1 − t
4:     n_1 ← n + 1
5:     Q ← W_0
6:     for k ← 1, n do
7:         h ← h · t · (n_1 − k) · ω_k
8:         h ← h/(k · u · ω_{k−1} + h)
9:         h_1 ← 1 − h
10:        Q ← h_1 · Q + h · W_k
11:    end for
12:    return Q
13: end procedure
```

The implementation provided in Algorithm 2.1 is a straight-forward implementation of the recursive scheme presented in (2.5). It requires $(3d + 8)n + 1$ floating-point arithmetic operations (flops) to compute a point on a rational Bézier curve of degree n in $\mathbb{E}^d$.

Algorithm 2.2 Second implementation

```
1: procedure NewRatBEval2(n, t, ω, W)
2:     h ← 1
3:     u ← 1 − t
4:     n_1 ← n + 1
5:     Q ← W_0
6:     if t ≤ 0.5 then
7:         u ← t/u
8:         for k ← 1, n do
9:             h ← h · u · (n_1 − k) · ω_k
10:            h ← h/(k · ω_{k−1} + h)
11:            h_1 ← 1 − h
12:            Q ← h_1 · Q + h · W_k
13:        end for
14:    else
15:        u ← u/t
16:        for k ← 1, n do
17:            h ← h · (n_1 − k) · ω_k
18:            h ← h/(k · u · ω_{k−1} + h)
19:            h_1 ← 1 − h
20:            Q ← h_1 · Q + h · W_k
21:        end for
22:    end if
23:    return Q
24: end procedure
```

Algorithm 2.2 decreases the number of flops to $(3d + 7)n + 2$. The main idea is to

precompute $\frac{t}{1-t}$ or its inverse and use it in the subsequent computations. However, for numerical reasons (cf. lines 7 and 15 in Algorithm 2.2), it is necessary to use a conditional statement. More precisely, one has to check whether $t \in [0, 0.5]$ or $t \in (0.5, 1]$, which can be easily done (it is enough to check the exponent of a floating-point number t).

Note that in the case of polynomial Bézier curves (1.48), one only needs to set $\omega_k := 1$ $(0 \le k \le n)$ in the given algorithms, thus simplifying used formulas. Then the number of flops is equal to $(3d+6)n+1$ in Algorithm 2.1 and $(3d+5)n+2$ in Algorithm 2.2.

		new method (Alg. 2.2)	de Casteljau (Alg. 1.1, 1.2)
polynomial Bézier curve	in total	$(3d+5)n+2$	$\frac{3dn(n+1)}{2}+1$
	add/sub	$(d+2)n+1$	$\frac{dn(n+1)}{2}+1$
	mult	$2(d+1)n$	$dn(n+1)$
	div	$n+1$	0
rational Bézier curve	in total	$(3d+7)n+2$	$\frac{(3d+5)n(n+1)}{2}+1$
	add/sub	$(d+2)n+1$	$\frac{(d+2)n(n+1)}{2}+1$
	mult	$2(d+2)n$	$(d+1)n(n+1)$
	div	$n+1$	$\frac{n(n+1)}{2}$

Table 2.1: Numbers of flops.

The numbers of flops for both the polynomial and rational versions of the new algorithm and the de Casteljau algorithm, which both have a geometric interpretation and compute only convex combinations of control points, are given in Table 2.1.

Example 2.3. *Table 2.2 shows the comparison between the running times of the de Casteljau algorithm and Algorithm 2.2 both for Bézier curves and rational Bézier curves (in the case of Bézier curves, Algorithm 2.2 has been simplified), for $d \in \{2, 3\}$. The results have been obtained on a computer with `Intel Core i5-2540M CPU` at `2.60GHz` processor and `4GB RAM`, using `GNU C Compiler 7.4.0` (single precision).*

The following numerical experiments have been conducted. For a fixed n, 10000 *curves of degree n are generated. Their control points $\mathsf{W}_k \in [-1, 1]^d$ and — in the rational case — weights $\omega_k \in [0.01, 1]$ $(0 \le k \le n)$ have been generated using the `rand()` C function. Each curve is then evaluated at 501 points $t_i := i/500$ $(0 \le i \le 500)$. Each algorithm is tested using*

the same curves. Table 2.2 shows the total running time of all 501×10000 *evaluations.*

		Bézier curve		rational Bézier curve	
n	d	new method (cf. Alg. 2.2)	de Casteljau (cf. Alg. 1.1)	new method (cf. Alg. 2.2)	de Casteljau (cf. Alg. 1.2)
1	2	**0.287**	0.297	**0.366**	0.542
	3	**0.485**	0.535	**0.543**	0.709
2	2	**0.300**	0.327	**0.373**	0.600
	3	**0.486**	0.508	**0.551**	0.764
3	2	**0.344**	0.495	**0.409**	0.907
	3	**0.498**	0.697	**0.553**	1.150
4	2	**0.401**	0.714	**0.490**	1.356
	3	**0.522**	1.031	**0.605**	1.728
5	2	**0.479**	1.076	**0.572**	1.921
	3	**0.585**	1.435	**0.673**	2.442
6	2	**0.554**	1.313	**0.660**	2.568
	3	**0.671**	1.905	**0.764**	3.276
10	2	**0.876**	3.044	**1.052**	6.212
	3	**1.049**	4.470	**1.185**	7.918
15	2	**1.288**	6.152	**1.524**	13.024
	3	**1.514**	9.249	**1.723**	16.603
20	2	**1.697**	10.503	**2.000**	22.376
	3	**2.008**	15.789	**2.288**	28.659

Table 2.2: Running times comparison (in seconds) for Example 2.3. The source code in C which was used to perform the tests is available at `https://bit.ly/fch-phd-ch2`.

Observe that in the case of polynomial Bézier curves, the quantities h, which are computed in the new algorithms, do not depend on the control points. One can use this fact in the fast evaluation of M Bézier curves of the same degree n for the same value of the parameter t. Such a method requires $(3dM+5)n+2$ flops while the direct use of the de Casteljau algorithm means that all computations have to be repeated M times, i.e., the number of flops is equal to $3Mdn(n+1)/2+1$.

Example 2.4. *Table 2.3 shows the comparison between the running times of the de Casteljau algorithm for polynomial Bézier curves, a version of Algorithm 2.2 which has been simplified for polynomial Bézier curves (cf. Table 2.2), as well as a modification of Algorithm 2.2 which*

computes the quantities h once for each t and uses them in evaluation of each polynomial Bézier curve. The running times are computed for $d \in \{2, 3\}$. The results have been obtained on a computer with `Intel Core i5-2540M CPU` *at* `2.60GHz` *processor and* `4GB RAM`*, using* `GNU C Compiler 7.4.0` *(single precision).*

The following numerical experiments have been conducted. For a fixed n, 10000 *curves of degree n are used. The curves are identical with those used in Example 2.3. Each curve is evaluated at 501 points $t_i := i/500$ $(0 \leq i \leq 500)$. Each algorithm is tested using the same curves. Table 2.3 shows the total running time of all* 501×10000 *evaluations.*

n	d	new method with shared h_i (cf. Alg. 2.2)	new method (cf. Alg. 2.2)	de Casteljau (cf. Alg. 1.1)
1	2	0.288	**0.287**	0.297
	3	0.492	**0.485**	0.535
2	2	**0.290**	0.300	0.327
	3	0.487	**0.486**	0.508
3	2	**0.305**	0.344	0.495
	3	**0.477**	0.498	0.697
4	2	**0.319**	0.401	0.714
	3	**0.492**	0.522	1.031
5	2	**0.354**	0.479	1.076
	3	**0.511**	0.585	1.435
6	2	**0.395**	0.554	1.313
	3	**0.545**	0.671	1.905
10	2	**0.602**	0.876	3.044
	3	**0.786**	1.049	4.470
15	2	**0.856**	1.288	6.152
	3	**1.123**	1.514	9.249
20	2	**1.121**	1.697	10.503
	3	**1.470**	2.008	15.789

Table 2.3: Running times comparison (in seconds) for Example 2.4. The source code in C which was used to perform the tests is available at `https://bit.ly/fch-phd-ch2`.

Remark 2.5. *In rather rare cases ($h_k \approx 1$), the problem of cancellation of digits ([22, §2.3.4]) can occur while $1 - h_k$ is computed (cf. h_1 in Algorithms 2.1, 2.2). One can avoid this problem*

using the relation

$$1 - h_k = \frac{h_k}{h_{k-1}} \frac{\omega_{k-1} k(1-t)}{\omega_k t(n-k+1)} \quad (1 \le k \le n),$$

if computations with high accuracy are necessary.

2.3 Generalizations of the algorithm

The algorithms presented in previous sections of this chapter can be generalized for a broader family of objects. The nature of these objects is not limited to curves — as will be further shown in Section 2.4, it can also be applied, e.g., to surfaces.

Let $b_k : D \to \mathbb{R}$ $(k = 0, 1, \ldots, N;\ N \in \mathbb{N})$ be real-valued multivariate *basis functions* such that

$$b_k(\boldsymbol{t}) \ge 0, \quad \sum_{k=0}^{N} b_k(\boldsymbol{t}) \equiv 1 \tag{2.9}$$

for $\boldsymbol{t} \in C \subseteq D$. Let a *rational parametric object* $\mathsf{S}_N : C \to \mathbb{E}^d$ $(d \in \mathbb{N})$ be defined by

$$\mathsf{S}_N(\boldsymbol{t}) := \frac{\displaystyle\sum_{k=0}^{N} \omega_k \mathsf{W}_k b_k(\boldsymbol{t})}{\displaystyle\sum_{k=0}^{N} \omega_k b_k(\boldsymbol{t})} \tag{2.10}$$

with the *weights* $\omega_k > 0$, and *control points* $\mathsf{W}_k \in \mathbb{E}^d$ $(0 \le k \le N)$. If $\omega_0 = \omega_1 = \ldots = \omega_N$, then

$$\mathsf{S}_N(\boldsymbol{t}) = \sum_{k=0}^{N} \mathsf{W}_k b_k(\boldsymbol{t}).$$

It is clear that rational Bézier curves are an example of such objects, with $C = [0, 1]$, $D = \mathbb{R}$, $\boldsymbol{t} = t$ and $b_k(\boldsymbol{t}) = B_k^N(t)$.

In the sequel, it will be proven that for a given $\boldsymbol{t} \in C$, the point $\mathsf{S}_N(\boldsymbol{t}) \in \mathbb{E}^d$ can be computed by Algorithm 2.3. Note that Algorithm 2.3 omits possible numerical difficulties connected with a particular choice of functions b_k. The implementation of this algorithm for a concrete case should take said difficulties into account.

Remark 2.6. *Let us fix $\boldsymbol{t} \in C$. Suppose that there exists $1 \le k \le N$ such that $b_k(\boldsymbol{t}) = 0$. Then one has the division by 0 in the line 5 of Algorithm 2.3. Such special cases should be considered separately, e.g., by omitting them in Eq.* (2.10)*. Observe that it is always possible because for at least one $0 \le j \le N$, $b_j(\boldsymbol{t}) > 0$ (cf.* (2.9)*). For the sake of simplicity, in the sequel, it is assumed that these special cases do not occur, i.e., $b_k(\boldsymbol{t}) > 0$ for $0 \le k \le N$.*

The correctness of the algorithm, as well as some of its properties, is proven in the following theorem, which is a generalized version of Theorem 2.1.

Theorem 2.7. *The quantities h_k and Q_k $(0 \le k \le N)$ computed by Algorithm 2.3 have the following properties:*

1. $h_k \in (0, 1]$,

Algorithm 2.3 Computation of $\mathsf{S}_N(\boldsymbol{t})$

1: **procedure** GenAlg($N, \boldsymbol{t}, \omega, \mathsf{W}$)
2: $h_0 \leftarrow 1$
3: $\mathsf{Q}_0 \leftarrow \mathsf{W}_0$
4: **for** $k \leftarrow 1, N$ **do**
5: $h_k \leftarrow \left(1 + \dfrac{\omega_{k-1} \cdot b_{k-1}(\boldsymbol{t})}{h_{k-1} \cdot \omega_k \cdot b_k(\boldsymbol{t})}\right)^{-1}$
6: $\mathsf{Q}_k \leftarrow (1 - h_k) \cdot \mathsf{Q}_{k-1} + h_k \cdot \mathsf{W}_k$
7: **end for**
8: **return** Q_N
9: **end procedure**

2. $\mathsf{Q}_k \in \mathbb{E}^d$,

3. $\mathsf{Q}_k \in C_k \equiv conv\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_k\}$ *(consequently,* $conv\{\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_k\} \subseteq C_k$*).*

Moreover, $\mathsf{S}_N(\boldsymbol{t}) = \mathsf{Q}_N$.

Proof. Let us define $h_0 := 1$, $\mathsf{Q}_0 := \mathsf{W}_0$, and

$$h_k := \frac{\omega_k b_k(\boldsymbol{t})}{\sum_{j=0}^{k} \omega_j b_j(\boldsymbol{t})}, \quad \mathsf{Q}_k := \frac{\sum_{j=0}^{k} \omega_j \mathsf{W}_j b_j(\boldsymbol{t})}{\sum_{j=0}^{k} \omega_j b_j(\boldsymbol{t})}$$

($k = 1, 2, \ldots, N$). It is clear that $h_k \in (0, 1]$, $\mathsf{Q}_k \in \mathbb{E}^d$ for $0 \le k \le N$, and $\mathsf{S}_N(\boldsymbol{t}) = \mathsf{Q}_N$. Certainly,

$$\mathsf{Q}_k \in \mathrm{conv}\{\mathsf{W}_0, \mathsf{W}_1, \ldots, \mathsf{W}_k\} \quad (0 \le k \le N).$$

To complete the proof, it is enough to check that:

$$\begin{cases} (1 - h_k)\mathsf{Q}_{k-1} + h_k \mathsf{W}_k = \mathsf{Q}_k, \\ \omega_k b_k(\boldsymbol{t}) h_k^{-1} = \omega_{k-1} b_{k-1}(\boldsymbol{t}) h_{k-1}^{-1} + \omega_k b_k(\boldsymbol{t}) \end{cases}$$

for $1 \le k \le N$ (cf. lines 5, 6 in Algorithm 2.3). □

Algorithm 2.3 has a geometric interpretation, uses only convex combinations of control points of S_N and has linear complexity with respect to N — under the assumption that all quotients of two consecutive basis functions can be computed in the total time $O(N)$.

Remark 2.8. *Similarly to the case of Bézier curves, the following relation holds*

$$1 - h_k = \frac{h_k}{h_{k-1}} \frac{\omega_{k-1} b_{k-1}(\boldsymbol{t})}{\omega_k b_k(\boldsymbol{t})} \tag{2.11}$$

for $1 \le k \le N$. *Using this simple relation, one can propose a subtraction-free version of Algorithm 2.3. Such formulation can be important for numerical reasons (cf. the problem of cancellation of digits; see, e.g., [22, §2.3.4]).*

Relation (2.11) will be used to prove the following theorem which shows an important property of Algorithm 2.3.

Theorem 2.9. *Suppose that*

$$\frac{b_k(\boldsymbol{t})}{b_{k+1}(\boldsymbol{t})} \leq \frac{b_k(\boldsymbol{u})}{b_{k+1}(\boldsymbol{u})} \quad (0 \leq k \leq N-1).$$

Then the point $\mathsf{S}_N(\boldsymbol{u}) \in \mathbb{E}^d$ *is in the convex hull of the points* $\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_N$ *computed by Algorithm 2.3.*

Proof. The proof follows the same path as the proof of its particular case, namely Theorem 2.2. Let the numbers h_k and the points Q_k $(0 \leq k \leq N)$ be computed by Algorithm 2.3 for a fixed $\boldsymbol{t} \in C$.

Using relation (2.11) and the assumption that $h_k \neq 0$ $(1 \leq k \leq N)$ (cf. Remark 2.6), observe that

$$\mathsf{W}_k = h_k^{-1}\mathsf{Q}_k - h_{k-1}^{-1}\frac{\omega_{k-1}b_{k-1}(\boldsymbol{t})}{\omega_k b_k(\boldsymbol{t})}\mathsf{Q}_{k-1}$$

for $1 \leq k \leq N$. Thus, after simple algebra, one gets

$$\begin{aligned}\mathsf{S}_N(\boldsymbol{u}) = D_N(\boldsymbol{u})^{-1}\Bigg(&\frac{\omega_N}{h_N}b_N(\boldsymbol{u})\cdot\mathsf{Q}_N \\ &+\sum_{k=0}^{N-1}\frac{\omega_k}{h_k}b_k(\boldsymbol{u})\left(1-\frac{b_k(\boldsymbol{t})b_{k+1}(\boldsymbol{u})}{b_{k+1}(\boldsymbol{t})b_k(\boldsymbol{u})}\right)\cdot\mathsf{Q}_k\Bigg),\end{aligned}$$

where $D_N(\boldsymbol{u}) := \sum_{k=0}^{N}\omega_k b_k(\boldsymbol{u}) > 0$.

Now, from the assumptions, it easily follows that because the values h_k $(0 \leq k \leq N)$ are positive (cf. Theorem 2.7) the point $\mathsf{S}_N(\boldsymbol{u})$ is in the set $\operatorname{conv}\{\mathsf{Q}_0, \mathsf{Q}_1, \ldots, \mathsf{Q}_N\}$. □

2.4 Example: Bézier surfaces

The general algorithm presented in Section 2.3 is not limited to curves. It can be applied to many other (rational) parametric objects, provided that their control points and basis functions b_k can be ordered in such a way that satisfies the algorithm's principles.

In this section, applications of Algorithm 2.3 will be shown for two such families of parametric objects — rational rectangular Bézier surfaces and rational triangular Bézier surfaces (see Definitions 1.83 and 1.88, respectively). Both surface types are *rational parametric objects* (cf. (2.10)), thus one can apply Algorithm 2.3 to derive efficient methods which have geometric interpretations, compute only convex combinations of points and allow to evaluate Bézier surfaces in linear time with respect to the number of control points.

In both cases, it is necessary to arrange the set of control points, along with their corresponding weights and basis functions (cf. (2.9)), into a one-dimensional sequence. Since Algorithm 2.3 is agnostic of the chosen sequence (as long as satisfies necessary conditions), many possible choices may be valid. For both surface types, two sequence choices will be proposed in their corresponding subsections. Each approach will have a corresponding recursive scheme for computing the intermediate values in Algorithm 2.3. Some technical details are omitted for clarity.

2.4.1 Computations for rational rectangular Bézier surfaces

Recall that a d-dimensional rational rectangular Bézier surface of degrees $m, n \in \mathbb{N}$ with weights $\omega_{ij} > 0$ and control points $\mathsf{W}_{ij} \in \mathbb{E}^d$ $(0 \leq i \leq m,\ 0 \leq j \leq n)$ is given by the formula

$$\mathsf{S}_{mn}(s,t) := \frac{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{n} \omega_{ij}\mathsf{W}_{ij}B_i^m(s)B_j^n(t)}{\displaystyle\sum_{i=0}^{m}\sum_{j=0}^{n} \omega_{ij}B_i^m(s)B_j^n(t)}$$

(cf. Definition 1.83). From the properties of its basis functions $B_i^m(s) \cdot B_j^n(t)$, it is clear that a rational rectangular Bézier surface $\mathsf{S}_{mn}(s,t)$ is a rational parametric object. The desired computational complexity of Algorithm 2.3 is $O(nm \cdot d)$, proportional to the number of control points, compared to the de Casteljau algorithm's complexity of $O(m(n^2+m)d)$. See Algorithm 1.4.

Remark 2.10. *If $s \in \{0,1\} \vee t \in \{0,1\}$, then (s,t) lies on the* boundary *rational Bézier curve with boundary control points and weights. The method described in Section 2.1 can be used in this case.*

It is thus sufficient to consider the problem of computing a point on a rational rectangular Bézier surface only for $(s,t) \in (0,1)^2$. The basis functions for such s,t are strictly positive, which eliminates the risk of division by zero when computing their ratios in Algorithm 2.3.

Now, let us consider the one-dimensional order in which the basis functions $B_i^m(s) \cdot B_j^n(t)$, along with their corresponding weights and control points, have to be arranged in order to apply Algorithm 2.3 efficiently. In the case of rational rectangular Bézier surfaces, all ratios of basis functions are of the form

$$\frac{B_i^m(s)}{B_k^m(s)} \cdot \frac{B_j^n(t)}{B_\ell^n(t)}.$$

Note that for $k \in \mathbb{N}, k \leq m-i$,

$$\frac{B_{i+k}^m(s)}{B_i^m(s)} = \frac{(m-i-k+1)_k}{(i+1)_k} \cdot \Big(\frac{s}{1-s}\Big)^k.$$

It is reasonable to choose an order of basis functions which leads to simplified computations. Additionally, one has to take into account some aspects related to numerical problems.

In the sequel, two choices for the ordering of the basis functions are presented and analyzed. For the reader's convenience, the analogues of quantities h_k and points Q_k from Algorithm 2.3 have two indices instead to correspond to the surface's structure.

Row-by-row approach

One can interpret the set of control points of a rational rectangular Bézier surface as a rectangular grid having $m+1$ rows with $n+1$ points in each row.

In such approach, the sequence of control points is as follows:

- the sequence begins with W_{00},
- $\mathsf{W}_{i,j-1}$ is followed by W_{ij} $(0 \leq i \leq m,\ 1 \leq j \leq n)$,

- $\mathsf{W}_{i-1,n}$ is followed by W_{i0} $(1 \le i \le m)$.

The sequences of weights ω_{ij} and basis functions $B_i^m(s) \cdot B_j^n(t)$ $(0 \le i \le m,\, 0 \le j \le n)$ are set accordingly. Figure 2.2 illustrates this approach.

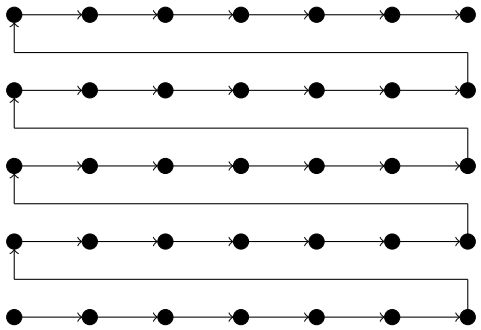

Figure 2.2: The sequence of control points used in the row-by-row approach for rational rectangular Bézier surfaces.

While executing Algorithm 2.3 for this ordering, ratios of the form

$$\frac{B_i^m(s)B_{j+1}^n(t)}{B_i^m(s)B_j^n(t)} = \frac{n-j}{j+1} \cdot \frac{t}{1-t}, \qquad \frac{B_{i+1}^m(s)B_0^n(t)}{B_i^m(s)B_n^n(t)} = \frac{(m-i)s}{(i+1)(1-s)} \cdot \left(\frac{1-t}{t}\right)^n$$

are used.

Let us fix $(s,t) \in (0,1)^2$ (cf. Remark 2.10). Now, based on Algorithm 2.3, the sequences of quantities h_{ij} and points $\mathsf{Q}_{ij} \in \mathbb{E}^d$ $(0 \le i \le m,\, 0 \le j \le n)$ are defined in the following recursive way using the order described above:

$$\begin{cases} h_{00} := 1, \\ \mathsf{Q}_{00} := \mathsf{W}_{00}, \\ h_{ij} := \left(1 + \dfrac{j\omega_{i,j-1}(1-t)}{n_j\omega_{ij}h_{i,j-1}t}\right)^{-1} & (i = 0,1,\ldots,m, j = 1,2,\ldots,n), \\ \mathsf{Q}_{ij} := (1-h_{ij})\mathsf{Q}_{i,j-1} + h_{ij}\mathsf{W}_{ij} & (i = 0,1,\ldots,m, j = 1,2,\ldots,n), \\ h_{i0} := \left(1 + \dfrac{i\omega_{i-1,n}(1-s)t_n}{m_i\omega_{i0}h_{i-1,n}s}\right)^{-1} & (i = 1,2,\ldots,m), \\ \mathsf{Q}_{i0} := (1-h_{i0})\mathsf{Q}_{i-1,n} + h_{i0}\mathsf{W}_{i0} & (i = 1,2,\ldots,m). \end{cases}$$

where $t_n := \dfrac{t^n}{(1-t)^n}$, $m_i := m-i+1$, $n_j := n-j+1$. Theorem 2.7 implies that $\mathsf{S}_{mn}(s,t) = \mathsf{Q}_{mn}$.

Algorithm 2.4 computes $\mathsf{S}_{mn}(s,t)$ for $(s,t) \in (0,1)^2$ and presents a more concrete implementation of the general method. It uses the same approach as Algorithm 2.1 when computing the values h_{ij}.

Let us take a look at the computational complexity of Algorithm 2.4. One has to perform $4+(m\cdot(n+1)+n)\cdot(3d+8)+T(t^n)+T((1-t)^n)$ flops over the course of the algorithm, where

Algorithm 2.4 Implementation of Alg. 2.3 for rational rectangular Bézier surface, row-by-row-approach

```
1: procedure RectBEvalRR(m, n, s, t, ω, W)
2:     st1 ← s · (1 − t)^n
3:     st2 ← (1 − s) · t^n
4:     u ← 1 − t
5:     n_1 ← n + 1
6:     m_1 ← m + 1
7:     h ← 1
8:     Q ← W_00
9:     for j ← 1, n do
10:        h ← (n_1 − j) · ω_0j · h · t
11:        h ← h/(h + j · ω_0,j−1 · u)
12:        Q ← (1 − h) · Q + h · W_0j
13:    end for
14:    for i ← 1, m do
15:        h ← (m_1 − i) · ω_i0 · h · st1
16:        h ← h/(h + i · ω_i−1,n · st2)
17:        Q ← (1 − h) · Q + h · W_i0
18:        for j ← 1, n do
19:            h ← (n_1 − j) · ω_ij · h · t
20:            h ← h/(h + j · ω_i,j−1 · u)
21:            Q ← (1 − h) · Q + h · W_ij
22:        end for
23:    end for
24:    return Q
25: end procedure
```

$T(t^n)$ and $T((1-t)^n)$ are the numbers of flops to compute t^n and $(1-t)^n$, respectively. For example, if exponentiation by squaring (see [43, §2.1]) is used, each of them is $O(\log\ n)$, with the exact number of operations dependent on the number of ones in the binary representation of n. This does not, however, change the total computational complexity of $O(mn \cdot d)$.

One can further reduce the number of arithmetic operations using the same technique which has been used in Algorithm 2.2. This, however, creates numerical concerns which have to be taken into account.

The idea is to pre-compute some elements of the basis function ratios. The ratio of two consecutive basis functions in the same row contains the expression $\frac{1-t}{t}$. One can store this expression (if $t \geq 0.5$) or its inverse (if $t < 0.5$) to use it throughout the execution of the algorithm. The ratios of two functions in different rows are not optimized and are computed in the same way as in Algorithm 2.4. This saves $(m+1)n - 1$ flops.

Folding approach

The folding approach is an alternative to the row-by-row approach which eliminates the need for computing t^n and $(1-t)^n$. Each step of the algorithm is computed in $O(d)$ time.

It differs from the row-by-row approach in the way of connecting rows. The folding approach avoids using the more complex

$$\frac{B_{i+1}^m(s)B_0^n(t)}{B_i^m(s)B_n^n(t)} = \frac{(m-i)s}{(i+1)(1-s)} \cdot \left(\frac{1-t}{t}\right)^n$$

ratio by transitioning between rows using *simpler* ratios:

$$\frac{B_{i+1}^m(s)B_0^n(t)}{B_i^m(s)B_0^n(t)} = \frac{B_{i+1}^m(s)B_n^n(t)}{B_i^m(s)B_n^n(t)} = \frac{(m-i)s}{(i+1)(1-s)}.$$

This means that each ratio used is the ratio of two consecutive Bernstein polynomials and thus can be computed in $O(1)$ time. In the folding approach, the sequence of control points is set as follows:

- the sequence begins with W_{00},
- $\mathsf{W}_{i,j-1}$ is followed by W_{ij} $(0 \leq i \leq m,\ 2 \mid i,\ 1 \leq j \leq n)$,
- $\mathsf{W}_{i,j+1}$ is followed by W_{ij} $(0 \leq i \leq m,\ 2 \nmid i,\ 0 \leq j \leq n-1)$,
- $\mathsf{W}_{i-1,n}$ is followed by W_{in} $(1 \leq i \leq m,\ 2 \nmid i)$,
- $\mathsf{W}_{i-1,0}$ is followed by W_{i0} $(1 \leq i \leq m,\ 2 \mid i)$,

with the sequences of weights ω_{ij} and basis functions $B_i^m(s)B_j^n(t)$ $(0 \leq i \leq m,\ 0 \leq j \leq n)$ set accordingly. Figure 2.3 illustrates this approach.

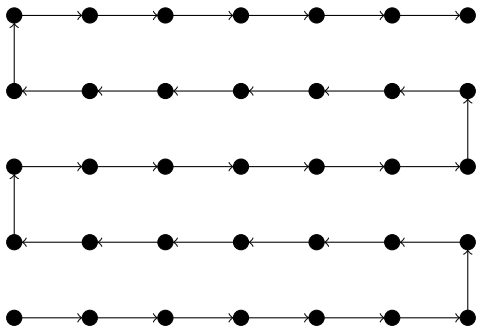

Figure 2.3: The sequence of control points used in the folding approach for rational rectangular Bézier surfaces.

Let us fix $(s,t) \in (0,1)^2$ (cf. Remark 2.10). Now, based on Algorithm 2.3, the sequences of quantities h_{ij} and points $\mathsf{Q}_{ij} \in \mathbb{E}^d$ $(0 \leq i \leq m,\ 0 \leq j \leq n)$ are defined in the following

recursive way using the order described above:

$$\begin{cases} h_{00} := 1, \\ \mathsf{Q}_{00} := \mathsf{W}_{00}, \\ h_{ij} := \left(1 + \dfrac{j\omega_{i,j-1}(1-t)}{n_j\omega_{ij}h_{i,j-1}t}\right)^{-1} & (i = 0,1,\ldots,m;\ 2 \mid i;\ j = 1,2,\ldots,n), \\ \mathsf{Q}_{ij} := (1-h_{ij})\mathsf{Q}_{i,j-1} + h_{ij}\mathsf{W}_{ij} & (i = 0,1,\ldots,m;\ 2 \mid i;\ j = 1,2,\ldots,n), \\ h_{in} := \left(1 + \dfrac{i\omega_{i-1,n}(1-s)}{m_i\omega_{in}h_{i-1,n}s}\right)^{-1} & (i = 1,2,\ldots,m;\ 2 \nmid i), \\ \mathsf{Q}_{in} := (1-h_{in})\mathsf{Q}_{i-1,n} + h_{in}\mathsf{W}_{in} & (i = 1,2,\ldots,m;\ 2 \nmid i), \\ h_{ij} := \left(1 + \dfrac{(n-j)\omega_{i,j+1}t}{(j+1)\omega_{ij}h_{i,j+1}(1-t)}\right)^{-1} & (i = 1,2,\ldots,m;\ 2 \nmid i;\ j = n-1,n-2,\ldots,1,0), \\ \mathsf{Q}_{ij} := (1-h_{ij})\mathsf{Q}_{i,j+1} + h_{ij}\mathsf{W}_{ij} & (i = 1,2,\ldots,m;\ 2 \nmid i;\ j = n-1,n-2,\ldots,1,0), \\ h_{i0} := \left(1 + \dfrac{i\omega_{i-1,0}(1-s)}{m_i\omega_{i0}h_{i-1,0}s}\right)^{-1} & (i = 1,2,\ldots,m;\ 2 \mid i), \\ \mathsf{Q}_{i0} := (1-h_{i0})\mathsf{Q}_{i-1,0} + h_{i0}\mathsf{W}_{i0} & (i = 1,2,\ldots,m;\ 2 \mid i), \end{cases}$$

where $m_i := m - i + 1$, $n_j := n - j + 1$.

Theorem 2.7 implies that $\mathsf{S}_{mn}(s,t) = \mathsf{Q}_{mn}$. Algorithm 2.5 computes $\mathsf{S}_{mn}(s,t)$ for $(s,t) \in (0,1)^2$ (cf. Remark 2.10) and presents a more concrete implementation of the general method. It uses the same approach as Algorithm 2.1 when computing the values h_{ij}.

Let us take a look at the computational complexity of Algorithm 2.5. In total, one has to perform $2 + (m \cdot (n+1) + n) \cdot (3d + 8)$ flops, which is lower by $2 + T(t^n) + T((1-t)^n)$ than when using Algorithm 2.4. They do, however, have the same asymptotic complexity of $O(mn \cdot d)$.

One can use a similar strategy to the one used in Algorithm 2.2 to further reduce the number of necessary arithmetic operations. The idea is to precompute two ratios and use them throughout the algorithm. The exact forms of the ratios depend on whether $s < 0.5$ and $t < 0.5$. If $t < 0.5$ then the first ratio is $\dfrac{t}{1-t}$; otherwise, it is $\dfrac{1-t}{t}$. Similarly, the second ratio is $\dfrac{s}{1-s}$ if $s < 0.5$, and $\dfrac{1-s}{s}$ otherwise.

In the folding approach no ratio of basis functions is dependent both on s and t. This means that the precomputed ratios avoid the numerical difficulties which were present in the row-by-row approach. This allows to reduce the number of flops by $(m+1)(n+1) - 3$.

Algorithm 2.5 Implementation of Alg. 2.3 for rational rectangular Bézier surfaces, folding approach

procedure RECTBEVALF($m, n, s, t, \omega, \mathsf{W}$)
  $u \leftarrow 1 - t$
  $r \leftarrow 1 - s$
  $n_1 \leftarrow n + 1$
  $m_1 \leftarrow m + 1$
  $h \leftarrow 1$
  $\mathsf{Q} \leftarrow \mathsf{W}_{00}$
  for $j \leftarrow 1, n$ **do**
    $h \leftarrow (n_1 - j) \cdot \omega_{0j} \cdot h \cdot t$
    $h \leftarrow h/(h + j \cdot \omega_{0,j-1} \cdot u)$
    $\mathsf{Q} \leftarrow (1 - h) \cdot \mathsf{Q} + h \cdot \mathsf{W}_{0j}$
  end for
  for $i \leftarrow 1, m$ **do**
    if $2 \mid i$ **then**
      $h \leftarrow (m_1 - i) \cdot \omega_{i0} \cdot h \cdot s$
      $h \leftarrow h/(h + i \cdot \omega_{i-1,0} \cdot r)$
      $\mathsf{Q} \leftarrow (1 - h) \cdot \mathsf{Q} + h \cdot \mathsf{W}_{i0}$
      for $j \leftarrow 1, n$ **do**
        $h \leftarrow (n_1 - j) \cdot \omega_{ij} \cdot h \cdot t$
        $h \leftarrow h/(h + j \cdot \omega_{i,j-1} \cdot u)$
        $\mathsf{Q} \leftarrow (1 - h) \cdot \mathsf{Q} + h \cdot \mathsf{W}_{ij}$
      end for
    else
      $h \leftarrow (m_1 - i) \cdot \omega_{in} \cdot h \cdot s$
      $h \leftarrow h/(h + i \cdot \omega_{i-1,n} \cdot r)$
      $\mathsf{Q} \leftarrow (1 - h) \cdot \mathsf{Q} + h \cdot \mathsf{W}_{in}$
      for $j \leftarrow n - 1, 0$ **do**
        $h \leftarrow (j + 1) \cdot \omega_{ij} \cdot h \cdot u$
        $h \leftarrow h/(h + (n - j) \cdot \omega_{i,j+1} \cdot t)$
        $\mathsf{Q} \leftarrow (1 - h) \cdot \mathsf{Q} + h \cdot \mathsf{W}_{ij}$
      end for
    end if
  end for
  return Q
end procedure

2.4.2 Computations for rational triangular Bézier surfaces

Recall that a d-dimensional rational triangular Bézier surface of degree $n \in \mathbb{N}$ with weights $v_{ij} > 0$ and control points $\mathsf{V}_{ij} \in \mathbb{E}^d$ $(0 \le i \le n,\ 0 \le j \le n-i)$ is given by the formula

$$\mathsf{T}_n(s,t) := \frac{\sum_{i=0}^{n}\sum_{j=0}^{n-i} v_{ij}\mathsf{V}_{ij}B_{ij}^n(s,t)}{\sum_{i=0}^{n}\sum_{j=0}^{n-i} v_{ij}B_{ij}^n(s,t)}$$

(cf. Definition 1.88). The basis functions in this case are *triangular Bernstein polynomials* (cf. Definition 1.84):

$$B_{ij}^n(s,t) := \frac{n!}{i!j!(n-i-j)!}s^i t^j (1-s-t)^{n-i-j} \qquad (0 \le i+j \le n; i,j \in \mathbb{N}).$$

For rational triangular Bézier surfaces of degree n, the desired computational complexity of Algorithm 2.3 is $O(n^2 \cdot d)$, proportionally to the number of control points. In contrast, the de Casteljau algorithm for rational triangular Bézier surfaces has the computational complexity of $O(n^3 \cdot d)$ (see §1.8.2).

Remark 2.11. *If $s = 0 \vee t = 1 \vee s + t = 1$, then (s,t) lies on the* boundary *rational Bézier curve with boundary control points and weights. The method described in Section 2.1 can be used in this case.*

It is thus sufficient to consider the problem of computing a point on a rational triangular Bézier surface only for $(s,t) \in \{(s,t) : s,t > 0,\ s+t < 1\}$. The basis functions for such s,t are strictly positive, which eliminates the risk of divisions by zero when computing their ratios in Algorithm 2.3.

The ratios of basis functions are of the form

$$\frac{B_{ij}^n(s,t)}{B_{k\ell}^n(s,t)} = \frac{k!\ell!(n-k-\ell)!}{i!j!(n-i-j)!}s^{i-k}t^{j-\ell}(1-s-t)^{k+\ell-i-j}.$$

It is reasonable to arrange the basis functions (with their corresponding control points and weights) in an order which is simple to compute.

Remark 2.12. *Computing said ratios may pose numerical difficulties, depending on the values of s,t. They can sometimes be reduced by re-formulating the problem using a simple transposition. Certainly,*

$$\mathsf{T}_n(s,t) = \mathsf{G}_n(t,s),$$

where G_n is a rational triangular Bézier surface with transposed weights $\tau_{ji} := v_{ij}$ and control points $\mathsf{K}_{ji} := \mathsf{V}_{ij}$ $(0 \le i+j \le n; i,j \in \mathbb{N})$.

Similarly to the case of rational rectangular Bézier surfaces, the control points V_{ij} of a rational triangular Bézier surface $\mathsf{T}_n(s,t)$ of degree n can be arranged into a triangular grid with $n+1$ rows, where the ith row has $n-i$ elements $(0 \le i \le n)$. In the sequel, the analogues of quantities h_k and points Q_k from Algorithm 2.3 have two indices instead, to correspond with the structure of the surfaces.

Row-by-row approach

In this case, two basis function ratio types will be used. The first one is

$$\frac{B^n_{i,j+1}(s,t)}{B^n_{ij}(s,t)} = \frac{n-i-j}{j+1} \cdot \frac{t}{1-s-t}.$$

The second allows to progress from one row to another:

$$\frac{B^n_{i0}(s,t)}{B^n_{i-1,n-i+1}(s,t)} = \frac{n-i+1}{i} \cdot \frac{s(1-s-t)^{n-i}}{t^{n-i+1}}.$$

In the case of rational rectangular Bézier surfaces, an analogous operation required the precomputation of t^n and $(1-t)^n$. For triangular Bézier surfaces, that is no longer the case. It is sufficient to perform the computations for decreasing i — that way, one can store the intermediate values of $\frac{s(1-s-t)^{n-i}}{t^{n-i+1}}$ and increment the exponents in $O(1)$ time for each transition between rows.

In this approach, the sequence of control points is as follows:

- the sequence begins with V_{n0},
- $\mathsf{V}_{i,j+1}$ is followed by V_{ij} $(0 \le i \le n-1,\ 0 \le j \le n-i-1)$,
- $\mathsf{V}_{i+1,0}$ is followed by $\mathsf{V}_{i,n-i}$ $(0 \le i \le n-1)$,

with the sequences of weights v_{ij} and basis functions $B^n_{ij}(s,t)$ $(i,j \ge 0, i+j \le n)$ set accordingly. Figure 2.4 illustrates this approach.

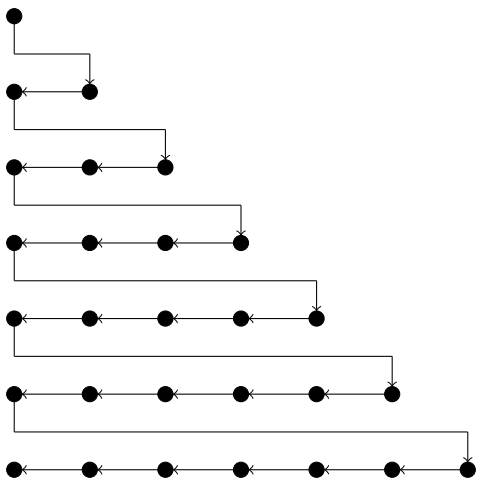

Figure 2.4: The sequence of control points used in the row-by-row approach for rational triangular Bézier surfaces.

Let us fix a point (s,t) inside the triangle $\{(s,t) : s,t > 0,\ s+t < 1\}$ (cf. Remark 2.11). Based on Algorithm 2.3, the sequences of quantities g_{ij} and points $\mathsf{U}_{ij} \in \mathbb{E}^d$ $(0 \le i+j \le n)$

are defined in the following recursive way using the order described above:

$$\begin{cases} g_{n0} := 1, \\ \mathsf{U}_{n0} := \mathsf{V}_{n0}, \\ g_{i,n-i} := \left(1 + \dfrac{v_{i+1,0} n_{i+1} s r^{n-i-1}}{v_{i,n-i} g_{i+1,0} (i+1) t^{n-i}}\right)^{-1} & (i = 0, 1, \ldots, n-1), \\ \mathsf{U}_{i,n-i} := (1 - g_{i,n-i})\mathsf{U}_{i,n-i} + g_{i,n-i}\mathsf{V}_{i,n-i} & (i = 0, 1, \ldots, n-1), \\ g_{ij} := \left(1 + \dfrac{v_{i,j+1} n_{i+j+1} t}{g_{i,j+1} v_{ij} (j+1) r}\right)^{-1} & (i = 0, 1, \ldots, n, j = n-i-1, n-i-2, \ldots, 1, 0), \\ \mathsf{U}_{ij} := (1 - g_{ij})\mathsf{U}_{i,j+1} + g_{ij}\mathsf{V}_{ij} & (i = 0, 1, \ldots, n, j = n-i-1, n-i-2, \ldots, 1, 0), \end{cases}$$

where $r := 1 - s - t$, $n_j := n - j + 1$. It follows from Theorem 2.7 that $\mathsf{T}_n(s,t) = \mathsf{U}_{00}$.

Algorithm 2.6 computes $\mathsf{T}_n(s,t)$ for $s, t > 0$, $s + t < 1$ presents a more concrete implementation of the general method. It uses the same approach to computing the values of h_{ij} as Algorithm 2.1.

Let us take a look at the computational complexity of Algorithm 2.6. In total, one has to perform $2n - 1 + \left(n + \dfrac{(n+1)n}{2}\right) \cdot (3d + 8)$ flops. The asymptotic computational complexity of Algorithm 2.6 is $O(n^2 \cdot d)$.

One can use a similar approach to the one used in Algorithm 2.2 to further reduce the number of necessary flops. The idea is to precompute some elements of the basis function ratios. The ratio of two consecutive basis functions in the same row contains the expression $\dfrac{t}{1-s-t}$. One can store this expression (if $s + t \leq 0.5$) or its inverse (if $t \geq 0.5$) to use it throughout the execution of the algorithm. Additionally, if $s \geq 0.5$, one can *transpose* the control points (cf. Remark 2.12) before the computation. If one of the conditions is met, this saves $\dfrac{(n+1)n}{2} - 1$ flops. In the remaining case, i.e., $s, t < 0.5$ and $s + t > 0.5$, the unmodified Algorithm 2.6 should be used.

The recommended approach in each case is illustrated in Figure 2.5.

Folding approach

The folding approach is an alternative to the row-by-row approach which eliminates the need for storing the intermediate values $t1, s1$ (see Algorithm 2.6). Each step of the algorithm is computed exactly in $O(d)$ time.

This approach shifts between rows using the simpler ratios

$$\frac{B^n_{i+1,0}(s,t)}{B^n_{i0}(s,t)} = \frac{(n-i)s}{(i+1)(1-s-t)}, \qquad \frac{B^n_{i+1,n-i+1}(s,t)}{B^n_{i,n-i}(s,t)} = \frac{(n-i)s}{(i+1)t}.$$

In the folding approach, the sequence of control points is set as follows:

- the sequence begins with V_{00},
- $\mathsf{V}_{i,j-1}$ is followed by V_{ij} $(0 \leq i \leq n,\ 2 \mid i,\ 1 \leq j \leq n - i)$,

Algorithm 2.6 Implementation of Alg. 2.3 for rational triangular Bézier surface, row-by-row-approach

```
1: procedure TriBEvalRR(m, n, s, t, v, V)
2:     r ← 1 − s − t
3:     t1 ← t
4:     s1 ← s
5:     g ← 1
6:     U ← V_{n0}
7:     for i ← n − 1, 1 do
8:         ni ← n − i
9:         g ← v_{i,n−i} · g · (i + 1) · t1
10:        g ← g/(g + v_{i+1,0} · ni · s1
11:        U ← (1 − g) · U + g · V_{i,n−i}
12:        for j ← n − i − 1, 0 do
13:            g ← g_{i,j+1} · v_{ij} · (j + 1) · r
14:            g ← g/(g + v_{i,j+1} · (ni − j) · t)
15:            U ← (1 − g) · U + g · V_{ij}
16:        end for
17:        t1 ← t1 · t
18:        s1 ← s1 · r
19:    end for
20:    g ← v_{0n} · g · t1
21:    g ← g/(g + v_{10} · n · s1)
22:    U ← (1 − g) · U + g · V_{0n}
23:    for j ← n − 1, 0 do
24:        g ← g_{0,j+1} · v_{0j} · (j + 1) · r
25:        g ← g/(g + v_{0,j+1} · (n − j) · t)
26:        U ← (1 − g) · U + g · V_{0j}
27:    end for
28:    return U
29: end procedure
```

- $\mathsf{V}_{i,j+1}$ is followed by V_{ij} $(0 \le i \le n,\ 2 \nmid i,\ 0 \le j \le n-i-1)$,
- $\mathsf{V}_{i-1,n-i+1}$ is followed by $\mathsf{V}_{i,n-i}$ $(1 \le i \le n,\ 2 \nmid i)$,
- $\mathsf{V}_{i-1,0}$ is followed by V_{i0} $(1 \le i \le n,\ 2 \mid i)$,

with the sequences of weights v_{ij} and basis functions $B_{ij}^n(s,t)$ $(0 \le i \le n,\ 0 \le j \le n-i)$ set accordingly. Figure 2.6 illustrates this approach.

Let us fix $(s,t) \in (0,1)^2$ (cf. Remark 2.11). Now, based on Algorithm 2.3, the sequences of quantities g_{ij} and points $\mathsf{U}_{ij} \in \mathbb{E}^d$ $(0 \le i \le n,\ 0 \le j \le n-i)$ are defined in the following recursive way using the order described above:

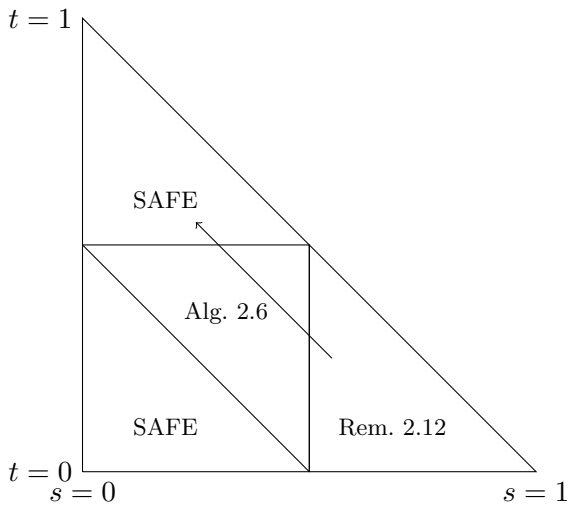

Figure 2.5: Numerical safety of the approach similar to Algorithm 2.2 for the row-by-row approach for rational triangular Bézier surfaces. The unsafe area could instead be computed using Algorithm 2.6.

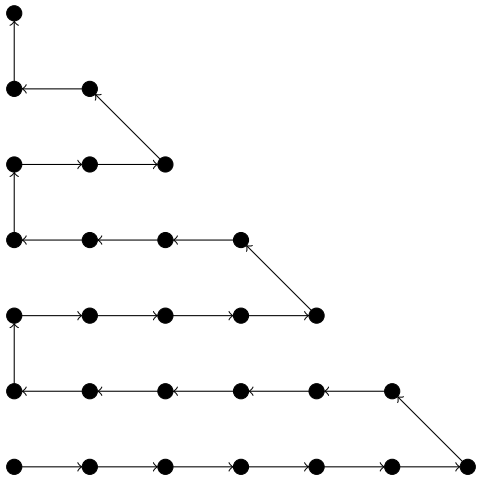

Figure 2.6: The sequence of control points used in the folding approach for rational triangular Bézier surfaces.

$$
\begin{cases}
g_{00} := 1, \\
\mathsf{U}_{00} := \mathsf{V}_{00}, \\
g_{ij} := \left(1 + \dfrac{v_{i,j-1} j r}{g_{i,j-1} v_{ij} n_{i+j} t}\right)^{-1} & (i = 0, 1, \ldots, n;\ 2 \mid i;\ j = 1, 2, \ldots, n-i), \\
\mathsf{U}_{ij} := (1 - g_{ij})\mathsf{U}_{i,j-1} + g_{ij}\mathsf{V}_{ij} & (i = 0, 1, \ldots, m;\ 2 \mid i;\ j = 1, 2, \ldots, n), \\
g_{i,n-i} := \left(1 + \dfrac{v_{i-1,n-i+1} i t}{g_{i-1,n-i+1} v_{i,n-i} n_i s}\right)^{-1} & (i = 1, 2, \ldots, n;\ 2 \nmid i), \\
\mathsf{U}_{i,n-i} := (1 - g_{i,n-i})\mathsf{U}_{i-1,n-i+1} + h_{i,n-i}\mathsf{V}_{i,n-i} & (i = 1, 2, \ldots, n;\ 2 \nmid i), \\
g_{ij} := \left(1 + \dfrac{v_{i,j+1} n_{i+j+1} t}{g_{i,j+1} v_{ij} (j+1) r}\right)^{-1} & (i = 1, 2, \ldots, n;\ 2 \nmid i;\ j = 0, 1, \ldots, n-i-1), \\
\mathsf{U}_{ij} := (1 - g_{ij})\mathsf{U}_{i,j+1} + g_{ij}\mathsf{V}_{ij} & (i = 1, 2, \ldots, n;\ 2 \nmid i;\ j = 0, 1, \ldots, n-i-1), \\
g_{i0} := \left(1 + \dfrac{v_{i-1,0} i r}{g_{i-1,0} v_{i0} n_i s}\right)^{-1} & (i = 1, 2, \ldots, m;\ 2 \mid i), \\
\mathsf{U}_{i0} := (1 - g_{i0})\mathsf{U}_{i-1,0} + g_{i0}\mathsf{V}_{i0} & (i = 1, 2, \ldots, m;\ 2 \mid i),
\end{cases}
$$

where $r := 1 - s - t$, $n_i := n - i + 1$.

Theorem 2.7 implies that $\mathsf{T}_n(s,t) = \mathsf{U}_{n0}$. Algorithm 2.7 presents a more concrete implementation of the general method, adjusted for numerical difficulties. The approach is similar to the one presented in Algorithm 2.1.

Let us take a look at the computational complexity of Algorithm 2.7. In total, one has to perform $2 + \left(\dfrac{n(n+1)}{2} + n\right) \cdot (3d + 8)$ flops, which is lower by $2n - 3$ than when using Algorithm 2.6. They do, however, have the same asymptotic complexity of $O(n^2 \cdot d)$.

Additional flops can be eliminated when using an approach similar to the one presented in Algorithm 2.2. Just as in the row-by-row approach, this can be achieved by precomputing $\dfrac{t}{1-s-t}$ (if $s + t \leq 0.5$) or $\dfrac{1-s-t}{t}$ (if $t \geq 0.5$). If $s \geq 0.5$, one can *transpose* the control points (cf. Remark 2.12) before the computation. If either of the conditions is satisfied, this saves $\dfrac{(n+1)n}{2} - 1$ flops. In the remaining case, i.e., $s, t < 0.5$ and $s + t > 0.5$, the non-modified Algorithm 2.6 should be used. These approaches are illustrated in Figure 2.7.

Algorithm 2.7 Implementation of Alg. 2.3 for rational triangular Bézier surfaces, folding approach

1: **procedure** TRIBEVALF($m, n, s, t, v, \mathsf{V}$)
2: $r \leftarrow 1 - s - t$
3: $n_1 \leftarrow n + 1$
4: $g \leftarrow 1$
5: $\mathsf{U} \leftarrow \mathsf{V}_{00}$
6: **for** $j \leftarrow 1, n$ **do**
7: $g \leftarrow g \cdot v_{0j} \cdot (n_1 - j) \cdot t$
8: $g \leftarrow g/(g + v_{0,j-1} \cdot j \cdot r)$
9: $\mathsf{U} \leftarrow (1 - g) \cdot \mathsf{U} + g \cdot V_{0j}$
10: **end for**
11: **for** $i \leftarrow 1, n$ **do**
12: $n1i \leftarrow n_1 - i$
13: **if** $2 \mid i$ **then**
14: $g \leftarrow g \cdot v_{i0} \cdot n1i \cdot s$
15: $g \leftarrow g/(g + v_{i-1,0} \cdot i \cdot r)$
16: $\mathsf{U} \leftarrow (1 - g) \cdot \mathsf{U} + g \cdot V_{i0}$
17: **for** $j \leftarrow 1, n - i$ **do**
18: $g \leftarrow g \cdot v_{ij} \cdot (n1i - j) \cdot t$
19: $g \leftarrow g/(g + v_{i,j-1} \cdot j \cdot r)$
20: $\mathsf{U} \leftarrow (1 - g) \cdot \mathsf{U} + g \cdot V_{ij}$
21: **end for**
22: **else**
23: $g \leftarrow g \cdot v_{i,n-i} \cdot n1i \cdot s$
24: $g \leftarrow g/(g + v_{i-1,n-i+1} \cdot i \cdot t)$
25: $\mathsf{U} \leftarrow (1 - g) \cdot \mathsf{U} + g \cdot V_{in}$
26: **for** $j \leftarrow n - i - 1, 0$ **do**
27: $j_1 \leftarrow j + 1$
28: $g \leftarrow g \cdot v_{ij} \cdot j_1 \cdot r$
29: $g \leftarrow g/(g + v_{i,j+1} \cdot (n_A - j_1) \cdot t)$
30: $\mathsf{U} \leftarrow (1 - g) \cdot \mathsf{U} + g \cdot V_{ij}$
31: **end for**
32: **end if**
33: **end for**
34: **return** U
35: **end procedure**

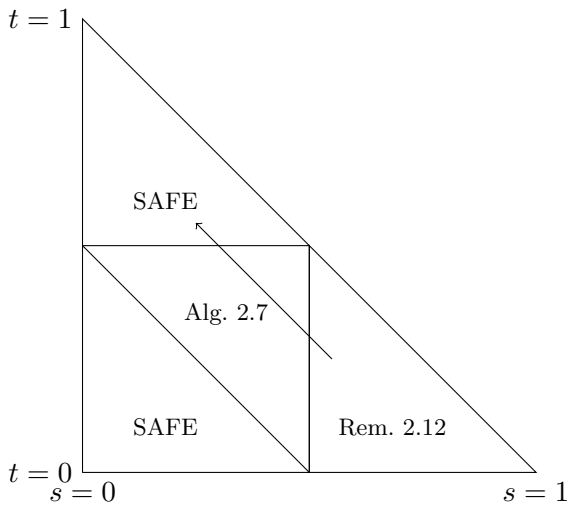

Figure 2.7: Numerical safety of the approach similar to Algorithm 2.2 for the folding approach for rational triangular Bézier surfaces. The unsafe area could instead be computed using Algorithm 2.7.

Chapter 3

New methods for B-spline functions

Let $m, n \in \mathbb{N}$. The knots

$$\underbrace{t_{-m} \leq \ldots \leq t_{-1} \leq t_0}_{\text{boundary knots}} \leq \underbrace{t_1 \leq \ldots \leq t_{n-1}}_{\text{inner knots}} \leq \underbrace{t_n \leq t_{n+1} \leq \ldots \leq t_{n+m}}_{\text{boundary knots}},$$

where $t_0 < t_n$ (i.e., the knots $\Omega_n := \{t_0, t_1, \ldots, t_n\}$ provide a partition of the interval $[t_0, t_n]$) serve as a support for a B-spline basis of degree m over $[t_0, t_n]$. The B-spline functions $N_{m,-m}, N_{m,-m+1}, \ldots, N_{m,n-1}$ form a basis for the set $\mathcal{S}_m(\Omega_n)$ of all splines of degree m over $[t_0, t_n]$. See Section 1.9.

The B-spline function N_{mi} with knots

$$t_i \leq t_{i+1} \leq \cdots \leq t_{m+i+1}$$

(cf. Definition 1.95), has support $[t_i, t_{m+i+1}]$, i.e., $N_{mi}(u)$ can be non-zero only for $u \in [t_i, t_{m+i+1}]$.

In the sequel, the convention given in Remark 1.101 is extended so that for any quantity Q, if $t_{m+i+1} = t_i$ then $\dfrac{Q}{t_{m+i+1} - t_i} := 0$.

Recall Theorem 1.102, which provides the recurrence relation

$$N_{mi}(u) = (u - t_i)\frac{N_{m-1,i}(u)}{t_{m+i} - t_i} + (t_{m+i+1} - u)\frac{N_{m-1,i+1}(u)}{t_{m+i+1} - t_{i+1}}.$$

Together with the initial value

$$N_{0i}(u) := \begin{cases} 1 & (u \in [t_i, t_{i+1})), \\ 0 & \text{otherwise}, \end{cases}$$

it is the theoretical basis for the de Boor-Cox algorithm (see §1.9.5 and Algorithm 1.6). From Definition 1.95, a differential-recurrence relation

$$N'_{mi}(u) = m \cdot \left(\frac{N_{m-1,i}(u)}{t_{m+i} - t_i} - \frac{N_{m-1,i+1}(u)}{t_{m+i+1} - t_{i+1}} \right)$$

given in Theorem 1.103 can be derived as well.

Both Theorems 1.102 and 1.103 connect the B-spline functions of different degrees. Their application thus requires using the *triangular* recurrence scheme.

A point on a B-spline curve of degree m with control points from a d-dimensional space $\mathbb{E}^d$ can be computed using the de Boor-Cox algorithm (see §1.9.5 and, in particular, Algorithm 1.6) which is based on the triangular recurrence scheme and has a geometric interpretation and good numerical properties. Similarly to the de Casteljau algorithms in the case of Bézier-type objects in $\mathbb{E}^d$, the de Boor-Cox algorithm has computational complexity of $O(m^2 d)$. However, if the coefficients of the (piecewise polynomial) B-spline functions are known, one can use a different approach to evaluate a B-spline curve

Let the adjusted Bernstein-Bézier basis form of the B-spline function N_{mi} over a single non-empty knot span $[t_j, t_{j+1}) \subset [t_0, t_n]$ $(j = i, i+1, \ldots, i+m)$ be

$$N_{mi}(u) = \sum_{k=0}^{m} b_k^{(i,j)} B_k^m\Big(\frac{u - t_j}{t_{j+1} - t_j}\Big) \qquad (u \in [t_j, t_{j+1})), \tag{3.1}$$

with $b_k^{(i,j)} \equiv b_{k,m}^{(i,j)}$. The coefficients $b_k^{(i,j)}$ for $j = 0, 1, \ldots, n-1$, $i = j-m, j-m+1, \ldots, j$, $k = 0, 1, \ldots, m$ need to be found.

Let the adjusted power basis form of the B-spline function N_{mi} over a single non-empty knot span $[t_j, t_{j+1}) \subset [t_0, t_n]$ $(j = i, i+1, \ldots, i+m)$ be

$$N_{mi}(u) = \sum_{k=0}^{m} a_k^{(i,j)} (u - t_j)^k \qquad (u \in [t_j, t_{j+1})), \tag{3.2}$$

with $a_k^{(i,j)} \equiv a_{k,m}^{(i,j)}$. In this case, the problem is to find all the coefficients $a_k^{(i,j)}$ for $j = 0, 1, \ldots, n-1$, $i = j-m, j-m+1, \ldots, j$, $k = 0, 1, \ldots, m$.

Note that a B-spline function is *piecewise* polynomial, i.e., in each non-empty knot span, the coefficients have to be computed separately. The exact approach to the problem heavily depends on the polynomial basis used in the computations. When the coefficients are already known, it is possible to compute a B-spline function in linear time with respect to its degree. One can also simplify, e.g., the evaluation of a point on a B-spline curve or perform some operations analytically.

Explicit expressions for the adjusted power basis coefficients of N_{mi} have been given in [65], and the result can be adapted for the adjusted Bernstein-Bézier form. The serious drawback of this approach, however, is high complexity, which greatly limits the use of this result in computational practice.

Let

$$s(t) := \sum_{i=-m}^{n-1} c_i N_{mi}(t). \tag{3.3}$$

An algorithm for finding the adjusted power basis coefficients of a spline s over a knot span $[t_j, t_{j+1})$ can be found in [33]. It uses Taylor series expansion to express the spline as

$$s(t) := \sum_{r=0}^{m} \frac{s^{(r)}(t_j)}{r!} (t - t_j)^r \qquad (t \in [t_j, t_{j+1}))$$

(cf. [33, Eq. (1.41)]). For $r = 0, 1, \ldots, m$, the derivatives

$$s^{(r)}(t_j) = \frac{m!}{(m-r)!} \cdot \sum_{i=j-m+r}^{j} c_i^r N_{m-r,i}(t_j) \qquad (t \in [t_j, t_{j+1}))$$

can be computed recursively as follows. Set

$$c_i^0 := c_i \qquad (i = j-m, \ldots, j)$$

(cf. (3.3)). Then

$$c_i^r := \begin{cases} \dfrac{c_i^{r-1} - c_{i-1}^{r-1}}{t_{m+i+1-r} - t_i} & (t_i < t_{m+i+1-r}), \\ 0 & \text{otherwise.} \end{cases}$$

for $r \geq 1$ and $j - m + r \leq i \leq j$ (cf. [33, Eq. (1.39) and (1.40)]). This allows to compute all the coefficients c_i^r in $O(m^2)$ time. All necessary values $N_{m-r,i}(t_j)$ can be computed using Theorem 1.102 in $O(m^2)$ time. The adjusted power basis coefficients of a B-spline over one knot span can thus be found in $O(m^2)$ time.

This approach can be used to find the adjusted power basis coefficients of one B-spline function. To find the coefficients of N_{mi} over $[t_j, t_{j+1})$, it is enough to set

$$c_k = \begin{cases} 1 & (k = i), \\ 0 & \text{otherwise} \end{cases}$$

(see (3.3)). The cost of finding the coefficients $a_k^{(i,j)}$ of N_{mi} is $O(m^2)$. In total, to find the adjusted power basis coefficients over $[t_j, t_{j+1})$ for all B-spline functions N_{mi} such that $j - m \leq i \leq j$, one has to do $O(m^3)$ operations. Let us assume that there are n_e non-empty knot spans $[t_j, t_{j+1})$ such that $j = 0, 1, \ldots, n-1$. To find the coefficients of all B-spline functions over all non-empty knot spans $[t_j, t_{j+1})$ for $j = 0, 1, \ldots, n-1$, one would need to perform $O(n_e m^3)$ operations.

With a similar approach, one can find the Bernstein-Bézier coefficients of N_{mi} over the knot span $[t_j, t_{j+1})$. One can check that

$$b_k^{(i,j)} = \frac{(m-k)!}{m!} N_{mi}^{(k)}(t_j) - \sum_{\ell=0}^{k-1} (-1)^{k-\ell} \binom{k}{\ell} b_\ell^{(i,j)} \qquad (k = 0, 1, \ldots, m)$$

(cf. [36, Eq. (5.25)] and [61, Theorem 4.1]). Just as in the case of the power basis, the Bernstein-Bézier coefficients of N_{mi} over $[t_j, t_{j+1})$ can be found in $O(m^2)$ time. In total, to find these coefficients of all B-spline functions over all non-empty knot spans $[t_j, t_{j+1})$ for $j = 0, 1, \ldots, n-1$, it is required to perform $O(n_e m^3)$ operations.

The approach given in [88] and [15] serves to convert a B-spline curve segment into a Bézier curve. It can be adapted to give an algorithm with $O(m^3)$ complexity for finding the adjusted Bernstein-Bézier coefficients $b_k^{(i,j)}$ of a single basis function N_{mi}. Doing so for each B-spline function in each non-empty knot span takes $O(n_e m^4)$ operations.

If there are recurrence relations for the coefficients of the B-spline functions over multiple knot spans, one can instead use them to efficiently find each of the coefficients. Over the course of this chapter, such computationally simple recurrence relations for the coefficients of the adjusted Bernstein-Bézier and power forms will be derived from a new differential-recurrence relation for the B-spline functions.

Remark 3.1. *In the sequel, an assumption will be used that no inner knot $t_1, t_2, \ldots, t_{n-1}$ has multiplicity greater than m. This guarantees the B-spline functions' continuity in (t_1, t_n).*

The assumption regarding the multiplicity of the inner knots is very common and intuitive, as it guarantees the continuity of a B-spline curve. It was used, e.g., in [33, 65] and [95, §3].

Let us suppose that there are n_e non-empty knot spans $[t_j, t_{j+1})$ such that $j = 0, 1, \ldots, n-1$. The main goal of this chapter is to give a recursive way of computing all $O(n_e m^2)$ coefficients $b_k^{(i,j)}$ (cf. (3.1)) or $a_k^{(i,j)}$ (cf. (3.2)) of the B-spline functions in $O(n_e m^2)$ time is given, assuming that all boundary knots are coincident.

Possible applications of this result can be as follows. Once the Bernstein-Bézier coefficients $b_k^{(i,j)}$ are known, each point on a B-spline curve S,

$$\mathsf{S}(u) := \sum_{i=-m}^{n-1} N_{mi}(u)\mathsf{W}_i \qquad (t_0 \le u \le t_n;\ \mathsf{W}_i \in \mathbb{E}^d),$$

can be computed in $O(m^2 + md)$ time using the algorithm proposed in Chapter 2. If there are N such points on M curves (each with the same knots), the total complexity is $O(n_e m^2 + M(m^2 + Nmd))$, compared to $O(MNm^2 d)$ when using the de Boor-Cox algorithm. Performed experiments confirm that the new method is faster than the de Boor-Cox algorihms even for low $M \approx 2, 3$. Using a similar approach, it is also possible to compute the value of $N_{mi}(u)$ in $O(m)$ time.

This chapter is organized as follows. Section 3.1 contains the differential-recurrence relation between the B-spline functions of the same degree. It will be the foundation for new recurrence relations which can be used to formulate an algorithm which computes the coefficients $a_k^{(i,j)}$ or $b_k^{(i,j)}$ of B-spline functions over each knot span. In Section 3.2, the algorithm for finding the coefficients in the adjusted Bernstein-Bézier form if $t_{-m} = t_0$, $t_n = t_{n+m}$ and all inner knots $t_1, t_2, \ldots, t_{n-1}$ have multiplicity 1 is given. The computational complexity of the method is $O(nm^2)$ which means that, asymptotically, it is optimal. Section 3.3 expands upon using the new algorithm to compute multiple points on multiple B-spline curves. The results of performed experiments are given there, showing that the new algorithm performs favorably compared to the de Boor-Cox algorithm and the method based on [27, p. 57–59]. The assumptions about knot multiplicity which were made in Section 3.2 are then relaxed in Section 3.4 to cover all cases (cf. Remark 3.1). In Section 3.5, the results given in Section 3.2 are adapted to the adjusted power basis.

3.1 New differential-recurrence relation for B-spline functions

Using the known recurrence relation (cf. Theorem 1.102) which connects B-spline functions of consecutive degrees, one can find a recurrence relation which is satisfied by their coefficients in the chosen basis.

Lemma 3.2. *For $u \in [t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$, let the representation of $N_{mi}(u)$ in the adjusted Bernstein-Bézier basis be*

$$N_{mi}(u) = \sum_{k=0}^{m} b_{k,m}^{(i,j)} B_k^m(t),$$

where $t := \dfrac{u - t_j}{t_{j+1} - t_j}$. *The coefficients* $b_{k,m}^{(i,j)}$ *satisfy the following recurrence relation:*

$$b_{k,m}^{(i,j)} = \frac{k}{m}\Big(\frac{t_{j+1} - t_i}{t_{m+i} - t_i} b_{k-1,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k-1,m-1}^{(i+1,j)}\Big) + \frac{m-k}{m}\Big(\frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)},\Big). \quad (3.4)$$

where $k = 0, 1, \ldots, m$ *and* $b_{-1,m-1}^{(i,j)} = b_{-1,m-1}^{(i+1,j)} = b_{m,m-1}^{(i,j)} = b_{m,m-1}^{(i+1,j)} := 0$.

Proof. Theorem 1.102 states that

$$N_{mi}(u) = (u - t_i)\frac{N_{m-1,i}(u)}{t_{m+i} - t_i} + (t_{m+i+1} - u)\frac{N_{m-1,i+1}(u)}{t_{m+i+1} - t_{i+1}}.$$

After applying the adjusted Bernstein-Bézier representations of $N_{m-1,i}(u)$ and $N_{m-1,i+1}(u)$, one gets

$$N_{mi}(u) = \frac{\sum_{k=0}^{m-1} b_{k,m-1}^{(i,j)}(u - t_i)B_k^{m-1}(t)}{t_{m+i} - t_i} + \frac{\sum_{k=0}^{m-1} b_{k,m-1}^{(i+1,j)}(t_{m+i+1} - u)B_k^{m-1}(t)}{t_{m+i+1} - t_{i+1}}.$$

Now, note that

$$(u - t_i) = (u - t_j) + (t_j - t_i) = (t_{j+1} - t_j) \cdot t + (t_j - t_i)$$

and

$$(t_{m+i+1} - u) = (t_{m+i+1} - t_{j+1}) + (t_{j+1} - u) = (t_{m+i+1} - t_{j+1}) + (t_{j+1} - t_j) \cdot (1 - t),$$

which gives

$$N_{mi}(u) = \sum_{k=0}^{m-1} \frac{(t_{j+1} - t_j)}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} \cdot tB_k^{m-1}(t) + \sum_{k=0}^{m-1} \Big(\frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)}\Big) B_k^{m-1}(t) + \sum_{k=0}^{m-1} \frac{t_{j+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)} \cdot (1 - t)B_k^{m-1}(t).$$

Now, from Eq. (1.29) and Definition 1.43, one can raise the degree of Bernstein polynomials to get

$$N_{mi}(u) = \sum_{k=0}^{m-1} \frac{m-k}{m}\Big(\frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)}\Big) B_k^m(t) + \sum_{k=0}^{m-1} \frac{k+1}{m}\Big(\frac{(t_{j+1} - t_i)}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)}\Big) B_{k+1}^m(t),$$

which, after some additional algebra, gives

$$N_{mi}(u) = \sum_{k=0}^{m} \Big[\frac{k}{m} \Big(\frac{(t_{j+1} - t_i)}{t_{m+i} - t_i} b_{k-1,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k-1,m-1}^{(i+1,j)} \Big) + \frac{m-k}{m} \Big(\frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)} \Big) \Big] B_k^m(t).$$

□

Lemma 3.2 gives a recurrence relation for the adjusted Bernstein-Bézier coefficients of B-spline basis functions of different degrees, defined using the same knot sequence. While this relation can be used to find the values of the coefficients, it is not optimal in terms of computational complexity as the recurrence scheme is analogous to the one used in the de Boor-Cox algorithm.

At the end of this section, a new differential-recurrence relation for the B-spline functions of the same degree m will be derived. It is shown that, by using this result, it is possible to efficiently find all the Bernstein-Bézier coefficients of the B-spline functions (see §3.2). When the coefficients are known, one can use the new algorithm given in Chapter 2 to evaluate a B-spline function of degree m in $O(m)$ time or a B-spline curve, which, when evaluating many curves at multiple points, has lower computational complexity than using the de Boor-Cox algorithm.

However, Lemma 3.2 can be used to prove some properties which indicate that the adjusted Bernstein-Bézier basis is numerically sound for B-spline functions.

Theorem 3.3. *For $u \in [t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$, the coefficients $b_{k,m}^{(i,j)}$ $(k = 0, 1, \ldots, m)$ of the adjusted Bernstein-Bézier representation of the B-spline function N_{mi} (cf. Eq. (3.1)) are non-negative.*

Proof. If $j < i$ or $j > i+m$, then all coefficients of $N_{mi}(u)$ are zero and thus are non-negative. The case $i \le j \le i + m$ can be proved using induction on m.

Base case ($m = 0$): the non-negativity of the coefficients follows directly from Eq. (1.75).

Induction step ($m - 1 \to m$): let us assume that the theorem is true for all B-spline functions of degree $m - 1$. Let us then take a look at (3.4):

$$b_{k,m}^{(i,j)} = \frac{k}{m} \Big(\frac{t_{j+1} - t_i}{t_{m+i} - t_i} b_{k-1,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k-1,m-1}^{(i+1,j)} \Big) + \frac{m-k}{m} \Big(\frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)} \Big).$$

It is clear that the fractions

$$\frac{t_{j+1} - t_i}{t_{m+i} - t_i}, \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}}, \frac{t_j - t_i}{t_{m+i} - t_i}, \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}}, \frac{m-k}{m}$$

are non-negative, as $t_k \le t_\ell$ if $k < \ell$. The remaining arithmetic operations are additions and multiplications of non-negative elements, which clearly gives a non-negative result. □

Theorem 3.4. *For $u \in [t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$, the following relation holds:*

$$\sum_{i=j-m}^{j} b_{k,m}^{(i,j)} = 1 \qquad (k = 0, 1, \ldots, m),$$

where $b_{k,m}^{(i,j)}$ are the adjusted Bernstein-Bézier coefficients of the B-spline function (cf. Eq. (3.1)*).*

Proof. Base case ($m = 0$): from Eq. (1.75), it follows that $b_{0,0}^{(j,j)} = 1$.

Induction step ($m - 1 \to m$): let us assume that

$$\sum_{i=j-m+1}^{j} b_{k,m-1}^{(i,j)} = 1 \qquad (k = 0, 1, \ldots, m-1)$$

for any B-spline basis function of degree $m - 1$.

Now, expressing $b_{k,m}^{(i,j)}$ using Lemma 3.2 gives

$$\sum_{i=j-m}^{j} b_{k,m}^{(i,j)} = \frac{k}{m}\Big(\sum_{i=j-m}^{j} \frac{t_{j+1} - t_i}{t_{m+i} - t_i} b_{k-1,m-1}^{(i,j)} + \sum_{i=j-m}^{j} \frac{t_{m+i+1} - t_{j+1}}{t_{m+i+1} - t_{i+1}} b_{k-1,m-1}^{(i+1,j)} \Big)$$
$$+ \frac{m-k}{m}\Big(\sum_{i=j-m}^{j} \frac{t_j - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)} + \sum_{i=j-m}^{j} \frac{t_{m+i+1} - t_j}{t_{m+i+1} - t_{i+1}} b_{k,m-1}^{(i+1,j)} \Big).$$

After eliminating the vanishing summands and some algebra, one gets

$$\sum_{i=j-m}^{j} b_{k,m}^{(i,j)} = \frac{k}{m} \sum_{i=j-m+1}^{j} \frac{t_{m+i} - t_i}{t_{m+i} - t_i} b_{k-1,m-1}^{(i,j)} + \frac{m-k}{m} \sum_{i=j-m+1}^{j} \frac{t_{m+i} - t_i}{t_{m+i} - t_i} b_{k,m-1}^{(i,j)}$$
$$= \frac{k}{m} \sum_{i=j-m+1}^{j} b_{k-1,m-1}^{(i,j)} + \frac{m-k}{m} \sum_{i=j-m+1}^{j} b_{k,m-1}^{(i,j)}.$$

Applying the induction assumption gives

$$\sum_{i=j-m}^{j} b_{k,m}^{(i,j)} = \frac{k}{m} \cdot 1 + \frac{m-k}{m} \cdot 1 = 1.$$

□

Using equations (1.74) and (1.76), one can derive new differential-recurrence relations for the B-spline functions of the same degree. For example, this result can be used to efficiently compute the coefficients of the N_{mi} functions (which are polynomial in each of the *knot spans*) in an adjusted Bernstein-Bézier or power basis.

Theorem 3.5. *Let*

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_{n-1} < t_n = t_{n+1} = \ldots = t_{n+m}$$

(cf. (1.73)*). The following relations hold:*

$$mN_{m,-m}(u) - (u - t_1) \cdot N'_{m,-m}(u) = 0, \tag{3.5}$$

$$N_{mi}(u) + \frac{t_i - u}{m} N'_{mi}(u) = \frac{t_{m+i+1} - t_i}{t_{m+i+2} - t_{i+1}} \left(N_{m,i+1}(u) + \frac{t_{m+i+2} - u}{m} N'_{m,i+1}(u) \right) \tag{3.6}$$
$$(i = -m, -m+1, \ldots, n-2),$$

$$mN_{m,n-1}(u) - (u - t_{n-1}) \cdot N'_{m,n-1}(u) = 0. \tag{3.7}$$

Proof. Equations (3.5) and (3.7) follow easily from equations (1.74) and (1.76). The relation (3.6) follows directly from taking the expression for $N_{m+1,i}(u)$ from Eq. (1.74) and differentiating it, then equating it with the expression for $N'_{m+1,i}(u)$ given in Eq. (1.76). □

Theorem 3.5 can be used to find a recurrence relation satisfied by the adjusted Bernstein-Bézier coefficients of B-spline functions of the same degree, as will be shown in §3.2.

3.2 Recurrence relations for B-spline functions' coefficients in adjusted Bernstein-Bézier basis

Assume that

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_{n-1} < t_n = t_{n+1} = \ldots = t_{n+m}.$$

For each knot span $[t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$, one needs to find the coefficients of N_{mi} $(i = j-m, j-m+1, \ldots, j)$ in the following adjusted Bernstein-Bézier basis form:

$$N_{mi}(u) = \sum_{k=0}^{m} b_k^{(i,j)} B_k^m(t) \qquad (t_j \le u < t_{j+1}),$$

where $b_k^{(i,j)} \equiv b_{k,m}^{(i,j)}$ and

$$t \equiv t^{(j)}(u) := \frac{u - t_j}{t_{j+1} - t_j}, \tag{3.8}$$

(cf. (3.1)). Additionally, then, $u = (t_{j+1} - t_j) \cdot t + t_j$.

Certainly, $N_{mi}(u) \equiv 0$ if $u < t_i$ or $u > t_{m+i+1}$, which means that for a given knot span $[t_j, t_{j+1})$, one only needs to find the coefficients of $N_{m,j-m}, N_{m,j-m+1}, \ldots, N_{mj}$, as all coefficients of other B-spline functions over this knot span are identical to zero. Thus, in each of n knot spans, there are $m+1$ non-vanishing B-spline functions, each with $m+1$ coefficients.

The following problem is considered.

Problem 3.6. *Let*

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_n = t_{n+1} = \ldots = t_{n+m}$$

(cf. (1.73)*). Find the adjusted Bernstein-Bézier basis coefficients* $b_k^{(i,j)}$ *(cf.* (3.1)*) of all functions* N_{mi} *over all non-trivial knot spans* $[t_j, t_{j+1}) \subset [t_0, t_n]$*, i.e., for* $j = 0, 1, \ldots, n-1$ *and* $i = j-m, j-m+1, \ldots, j$*.*

Solving Problem 3.6 requires computing $n(m+1)^2$ coefficients $b_k^{(i,j)}$. In this section, it will be shown how to do it in $O(nm^2)$ time — proportionally to the number of coefficients. Theorem 3.5 serves as a foundation of the presented approach. More precisely, the theorem will be used to construct recurrence relations for the coefficients $b_k^{(i,j)}$ which allow solving Problem 3.6 efficiently.

The results for particular cases will be presented in stages. In §3.2.1, an explicit expression for the coefficients of N_{mj} and $N_{m,j-m}$ over $[t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$ will be found. This will, in particular, cover the only non-trivial knot span for $N_{m,n-1}$. In §3.2.2, Eq. (3.6) will be applied to find the coefficients of N_{mi} for $j = n-1, n-2, \ldots, 0$ and $i = j-1, j-2, \ldots, j-m+1$.

3.2.1 Stage 1

For $j = 0, 1, \ldots, n-1$, one can use Eq. (1.74) for $i = j$, along with the fact that $N_{\ell,j+1} \equiv 0$ over $[t_j, t_{j+1})$ ($\ell = m-1, m-2, \ldots, 0$), to find that

$$N_{mj}(u) = \frac{(u-t_j)^m}{\prod_{k=1}^{m}(t_{j+k}-t_j)} N_{0j}(u) = \frac{(t_{j+1}-t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+k}-t_j)} B_m^m(t).$$

It means that

$$\begin{cases} b_k^{(j,j)} = 0 & (k = 0, 1, \ldots, m-1), \\ b_m^{(j,j)} = \dfrac{(t_{j+1}-t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+k}-t_j)}, \end{cases} \tag{3.9}$$

where $0 \le j \le n-1$.

Using the same approach for $N_{m,j-m}$ over $[t_j, t_{j+1})$ gives

$$N_{m,j-m}(u) = \frac{(t_{j+1}-t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+1}-t_{j+1-k})} B_0^m(t).$$

The coefficients $b_k^{(j-m,j)}$ $(k = 0, 1, \ldots, m)$ are thus given by the following formula:

$$\begin{cases} b_0^{(j-m,j)} = \dfrac{(t_{j+1}-t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+1}-t_{j+1-k})}, \\ b_k^{(j-m,j)} = 0 \qquad\qquad (k = 1, 2, \ldots, m), \end{cases} \tag{3.10}$$

where $0 \le j \le n-1$. The adjusted Bernstein-Bézier (cf. (3.1)) coefficients of N_{mj} and $N_{m,j-m}$ over the knot span $[t_j, t_{j+1})$ have been found for $j = 0, 1, \ldots, n-1$.

In the sequel, the following observation will be of use.

Remark 3.7. *Note that*

$$N_{m,n-1}(t_n) = \frac{(t_n - t_{n-1})^{m-1}}{\prod_{k=1}^{m-1}(t_{n+k}-t_{n-1})} B_m^m(1) = 1,$$

since $t_n = t_{n+1} = \ldots = t_{n+m}$. The B-spline functions have the partition of unity property and are non-negative (cf. Theorem 1.99), it is thus clear that

$$N_{mi}(t_n) = 0 \qquad (i = -m, -m+1, \ldots, n-2).$$

Similarly,

$$N_{m,-m}(t_0) = \frac{(t_1 - t_0)^m}{\prod_{k=1}^{m}(t_1 - t_{1-k})} B_0^m(0) = 1,$$

since $t_{-m} = t_{-m+1} = \ldots = t_0$. It follows that

$$N_{mi}(t_0) = 0 \qquad (i = -m+1, -m+2, \ldots, 0).$$

3.2.2 Stage 2

To compute the coefficients of all functions N_{mi} over knot spans $[t_j, t_{j+1})$ such that $j = n-1, n-2, \ldots, 0$ and $i = j-1, j-2, \ldots, j-m+1$, Eq. (3.6) will be used. The following identity will be useful when operating on Eq. (3.6):

$$\Big(N_{mi}(u)\Big)' = \frac{dN_{mi}(u)}{du} = \sum_{k=0}^{m} b_k^{(i,j)} \frac{dB_k^m(t)}{dt} \cdot \frac{dt}{du} = (t_{j+1} - t_j)^{-1} \sum_{k=0}^{m} b_k^{(i,j)} \Big(B_k^m(t)\Big)' \tag{3.11}$$

(cf. (3.8)).

Let

$$v_i \equiv v_{mi} := \frac{t_{m+i+1} - t_i}{t_{m+i+2} - t_{i+1}}. \tag{3.12}$$

Substituting the adjusted Bernstein-Bézier forms of N_{mi} and $N_{m,i+1}$ in the knot span $[t_j, t_{j+1})$ and applying Eq. (3.11) into Eq. (3.6) gives

$$\sum_{k=0}^{m} b_k^{(i,j)} B_k^m(t) + \Big(\frac{t_i - t_j}{m(t_{j+1} - t_j)} - \frac{t}{m}\Big) \left(\sum_{k=0}^{m} b_k^{(i,j)} B_k^m(t)\right)' =$$
$$= v_i \left(\sum_{k=0}^{m} b_k^{(i+1,j)} B_k^m(t) + \Big(\frac{t_{m+i+2} - t_j}{m(t_{j+1} - t_j)} - \frac{t}{m}\Big) \left(\sum_{k=0}^{m} b_k^{(i+1,j)} B_k^m(t)\right)'\right).$$

After using identities (1.32) and (1.33) and doing some algebra, one gets

$$\sum_{k=0}^{m} \Big(l_{ki} b_{k-1}^{(i,j)} + d_{ki} b_k^{(i,j)} + u_{ki} b_{k+1}^{(i,j)}\Big) B_k^m(t) =$$
$$= v_i \sum_{k=0}^{m} \Big(l_{k,m+i+2} b_{k-1}^{(i+1,j)} + d_{k,m+i+2} b_k^{(i+1,j)} + u_{k,m+i+2} b_{k+1}^{(i+1,j)}\Big) B_k^m(t),$$

where

$$l_{kr} := k(t_{j+1} - t_r), \quad d_{kr} := (m-k)(t_{j+1} - t_r) + k(t_r - t_j), \quad u_{kr} := (m-k)(t_r - t_j).$$

Matching the coefficients of Bernstein polynomials on both sides gives a set of $m+1$ equations of the form:

$$\begin{cases} (t_{j+1} - t_i) b_0^{(i,j)} + (t_i - t_j) b_1^{(i,j)} = v_i \Big((t_{j+1} - t_{m+i+2}) b_0^{(i+1,j)} + (t_{m+i+2} - t_j) b_1^{(i+1,j)}\Big), \\ \\ l_{ki} b_{k-1}^{(i,j)} + d_{ki} b_k^{(i,j)} + u_{ki} b_{k+1}^{(i,j)} = v_i \Big(l_{k,m+i+2} b_{k-1}^{(i+1,j)} + d_{k,m+i+2} b_k^{(i+1,j)} + u_{k,m+i+2} b_{k+1}^{(i+1,j)}\Big) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (k = 1, 2, \ldots, m-1), \\ \\ (t_{j+1} - t_i) b_{m-1}^{(i,j)} + (t_i - t_j) b_m^{(i,j)} = v_i \Big((t_{j+1} - t_{m+i+2}) b_{m-1}^{(i+1,j)} + (t_{m+i+2} - t_j) b_m^{(i+1,j)}\Big). \end{cases} \tag{3.13}$$

Theorem 3.8. *For $j = 0, 1, \ldots, n-1$ and $i = j-1, j-2, \ldots, j-m+1$, assuming that the coefficients $b_k^{(i+1,j)}$ $(k = 0, 1, \ldots, m)$ are known, the values $b_0^{(i,j)}, b_1^{(i,j)}, \ldots, b_m^{(i,j)}$ satisfy a first-order non-homogeneous recurrence relation*

$$(t_{j+1} - t_i)b_k^{(i,j)} + (t_i - t_j)b_{k+1}^{(i,j)} = A(m,i,j,k) \qquad (k = 0, 1, \ldots, m-1), \tag{3.14}$$

where

$$A(m,i,j,k) := v_i\Big((t_{j+1} - t_{m+i+2})b_k^{(i+1,j)} + (t_{m+i+2} - t_j)b_{k+1}^{(i+1,j)}\Big).$$

Proof. Base case ($k = 0$ and $k = m$): the relation holds and is presented in the first and the last equations of the system (3.13).

Induction step ($k \to k+1$): the $(k+2)$th equation in the system (3.13) is

$$\begin{aligned} l_{k+1,i}b_k^{(i,j)} + d_{k+1,i}b_{k+1}^{(i,j)} + u_{k+1,i}b_{k+2}^{(i,j)} = \\ = v_i\Big(l_{k+1,m+i+2}b_k^{(i+1,j)} + d_{k+1,m+i+2}b_{k+1}^{(i+1,j)} + u_{k+1,m+i+2}b_{k+2}^{(i+1,j)}\Big). \end{aligned}$$

Subtracting sidewise the induction assumption scaled by $\dfrac{l_{k+1,i}}{(t_{j+1} - t_i)} = k+1$ gives, after some algebra,

$$(t_{j+1} - t_i)b_{k+1}^{(i,j)} + (t_i - t_j)b_{k+2}^{(i,j)} = v_i\Big((t_{j+1} - t_{m+i+2})b_{k+1}^{(i+1,j)} + (t_{m+i+2} - t_j)b_{k+2}^{(i+1,j)}\Big),$$

which concludes the proof. □

From Theorem 3.8, it follows that there are m independent equations in the system (3.13), as one of them is redundant. One thus needs an initial value to find the values of all $b_0^{(i,j)}, b_1^{(i,j)}, \ldots, b_m^{(i,j)}$ using the recurrence relation (3.14).

If $j = n-1$, Remark 3.7 can be used to find that

$$N_{mi}(t_n) = b_m^{(i,n-1)} = 0 \qquad (i = n-2, n-3, \ldots, n-m).$$

In this case, the recurrence relation given in Theorem 3.8 simplifies to

$$(t_n - t_i)b_k^{(i,n-1)} = (t_{n-1} - t_i)b_{k+1}^{(i,n-1)} + v_i(t_n - t_{n-1})b_{k+1}^{(i+1,n-1)}.$$

It means that, for $i = n-2, n-3, \ldots, n-m$, the following recurrence relation holds:

$$\begin{cases} b_m^{(i,n-1)} = 0, \\ b_k^{(i,n-1)} = \dfrac{t_{n-1} - t_i}{t_n - t_i}b_{k+1}^{(i,n-1)} + \dfrac{t_n - t_{n-1}}{t_n - t_{i+1}}b_{k+1}^{(i+1,n-1)} \quad (k = m-1, m-2, \ldots, 0). \end{cases} \tag{3.15}$$

For $i = n-2, n-3, \ldots, n-m$, assuming that the coefficients $b_k^{(i+1,n-1)}$ are known ($k = 1, 2, \ldots, m$), the recurrence relation (3.15) has an explicit solution

$$\begin{cases} b_m^{(i,n-1)} = 0, \\ b_k^{(i,n-1)} = \dfrac{t_n - t_{n-1}}{t_n - t_{i+1}} \displaystyle\sum_{\ell=0}^{m-k-1} \Big(\frac{t_{n-1} - t_i}{t_n - t_i}\Big)^{\ell} b_{k+1+\ell}^{(i+1,n-1)} \quad (k = 0, 1, \ldots, m-1). \end{cases} \tag{3.16}$$

To find the initial value, if $j < n-1$ and $i = j-1, j-2, \ldots, j-m+1$, the right continuity condition will be used, i.e.,

$$N_{mi}(t_{j+1}^-) = N_{mi}(t_{j+1}^+).$$

More precisely,

$$N_{mi}(t_{j+1}^-) = \sum_{k=0}^{m} b_k^{(i,j)} B_k^m(1) = b_m^{(i,j)}$$

and

$$N_{mi}(t_{j+1}^+) = \sum_{k=0}^{m} b_k^{(i,j+1)} B_k^m(0) = b_0^{(i,j+1)},$$

which gives the relation

$$b_m^{(i,j)} = b_0^{(i,j+1)}.$$

This completes the recurrence scheme for $j = n-2, n-3, \ldots, 0$ and $i = j-1, j-2, \ldots, j-m+1$:

$$\begin{cases} b_m^{(i,j)} = b_0^{(i,j+1)}, \\ b_k^{(i,j)} = \dfrac{t_j - t_i}{t_{j+1} - t_i} b_{k+1}^{(i,j)} + \dfrac{v_i}{t_{j+1} - t_i}\Big((t_{j+1} - t_{m+i+2}) b_k^{(i+1,j)} + (t_{m+i+2} - t_j) b_{k+1}^{(i+1,j)}\Big) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (k = m-1, m-2, \ldots, 0). \end{cases} \tag{3.17}$$

From Eq. (3.17) follows an explicit formula for the coefficients $b_k^{(i,j)}$ $(k = 0, 1, \ldots, m)$, assuming that the coefficients $b_0^{(i,j+1)}$ and $b_k^{(i+1,j)}$ $(k = 0, 1, \ldots, m)$ are known:

$$b_k^{(i,j)} = \Big(\frac{t_j - t_i}{t_{j+1} - t_i}\Big)^{m-k} b_0^{(i,j+1)} + \sum_{\ell=0}^{m-k-1} \Big(\frac{t_j - t_i}{t_{j+1} - t_i}\Big)^{\ell} \frac{v_i}{t_{j+1} - t_i} q_{k+\ell}, \tag{3.18}$$

where

$$q_\ell := (t_{j+1} - t_{m+i+2}) b_\ell^{(i+1,j)} + (t_{m+i+2} - t_j) b_{\ell+1}^{(i+1,j)},$$

and $0 \le j \le n-2$, $j-m+1 \le i \le j-1$.

The coefficients of N_{mi} have been found for $j = 0, 1, \ldots, n-1$ and $i = j-1, j-2, \ldots, j-m+1$.

3.2.3 The theorem and the algorithm

The results presented in §3.2.1 and §3.2.2 can be combined to prove the following theorem.

Theorem 3.9. *Let*

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_{n-1} < t_n = t_{n+1} = \ldots = t_{n+m}.$$

The $n(m+1)^2$ adjusted Bernstein-Bézier coefficients $b_k^{(i,j)}$ of the B-spline functions N_{mi} over each knot span $[t_j, t_{j+1})$ (cf. (3.1)), for $j = 0, 1, \ldots, n-1$, $i = j-m, j-m+1, \ldots, j$ and $k = 0, 1, \ldots, m$, can be computed in the computational complexity $O(nm^2)$ in the following way:

1. *For $j = 0, 1, \ldots, n-1$ and $k = 0, 1, \ldots, m$, the coefficients $b_k^{(j,j)}$ and $b_k^{(j-m,j)}$ are given explicitly in equations (3.9) and (3.10), respectively.*

2. *For $j = n-1$, $i = n-2, n-3, \ldots, n-m$ and $k = m, m-1, \ldots, 0$, the coefficients $b_k^{(i,n-1)}$ $(k = 0, 1, \ldots, m)$ are computed by the recurrence relation* (3.15) *(for their explicit forms, see* (3.16)*).*

3. *For $j = n-2, n-3, \ldots, 0$, $i = j-1, j-2, \ldots, j-m+1$ and $k = m, m-1, \ldots, 0$, the coefficients $b_k^{(i,j)}$ are computed by the recurrence relation* (3.17) *(for their explicit forms, see* (3.18)*).*

Example 3.10. *Let us set $m := 3$, $n := 5$. Let the knots be*

t_{-3}	t_{-2}	t_{-1}	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
0	0	0	0	3	5	6	9	10	10	10	10

.

Figure 3.1 illustrates the approach to computing all necessary adjusted Bernstein-Bézier coefficients of B-spline functions, given in Theorem 3.9. Arrows denote recurrent dependence. Diagonally striped squares are computed using Eq. (3.9)*. Horizontally striped squares are computed using Eq.* (3.10)*. White squares are computed using either Eq.* (3.15) *(for $u \in [t_4, t_5)$) or* (3.17) *(for $u < t_4$).*

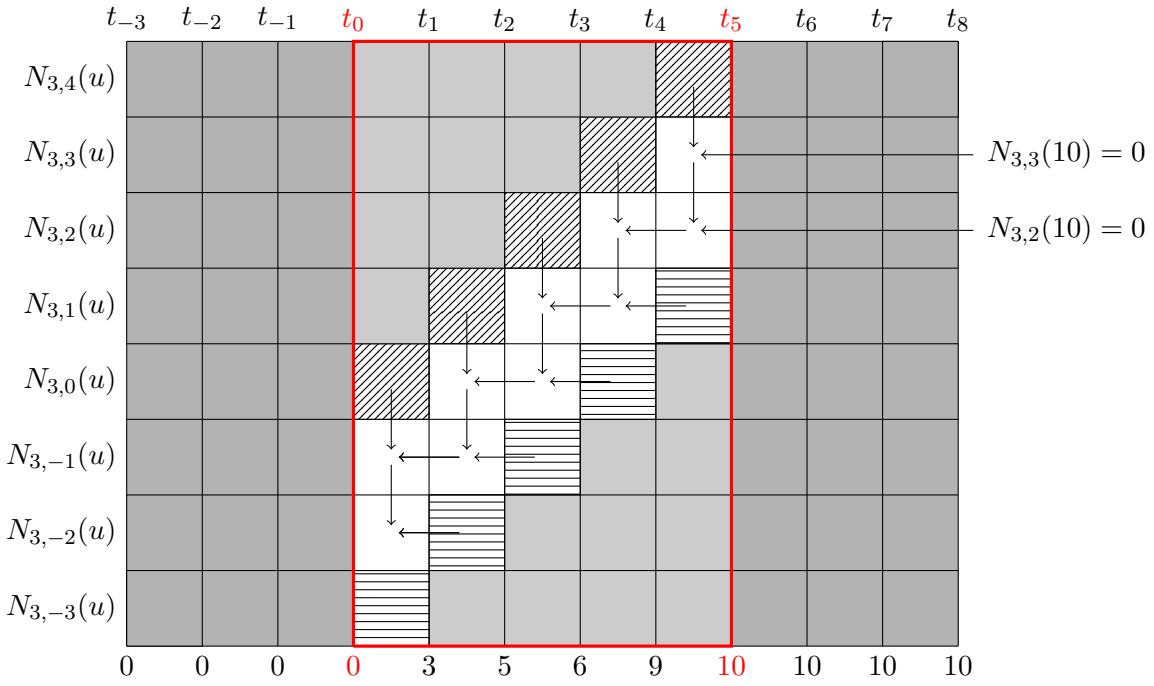

Figure 3.1: An illustration of Example 3.10.

Implementation

Algorithm 3.1 implements the approach proposed in Theorem 3.9. It returns a sparse array $B \equiv B[0..n-1, -m..n-1, 0..m]$, where

$$B[j, i, k] = b_k^{(i,j)} \qquad (0 \le j < n,\ -m \le i < n,\ 0 \le k \le m)$$

(cf. (3.1)).

For each of the n knot spans, one has to compute the coefficients of $m+1$ functions ($n(m+1)^2$ coefficients in total). Computing all coefficients of one B-spline function in a given knot span requires $O(m)$ operations. In total, then, the complexity of Algorithm 3.1 is $O(nm^2)$ — giving the optimal $O(1)$ time per coefficient.

Algorithm 3.1 Computing the coefficients of the adjusted Bernstein-Bézier form of the B-spline functions

procedure BSplineBBF($n, m, [t_{-m}, t_{-m+1}, \ldots, t_{n+m}]$)
  $B \leftarrow$ `SparseArray[0..n-1, -m..n-1, 0..m](fill=0)`
  for $j \leftarrow 0, n-1$ **do**
    $B[j, j, m] \leftarrow \dfrac{(t_{j+1} - t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+k} - t_j)}$
    $B[j, j-m, 0] \leftarrow \dfrac{(t_{j+1} - t_j)^{m-1}}{\prod_{k=2}^{m}(t_{j+1} - t_{j+1-k})}$
  end for
  for $i \leftarrow n-2, n-m$ **do**
    for $k \leftarrow m-1, 0$ **do**
      $B[n-1, i, k] \leftarrow \dfrac{t_{n-1} - t_i}{t_n - t_i} \cdot B[n-1, i, k+1] + \dfrac{t_n - t_{n-1}}{t_n - t_{i+1}} \cdot B[n-1, i+1, k+1]$
    end for
  end for
  for $j \leftarrow n-2, 0$ **do**
    for $i \leftarrow j-1, j-m+1$ **do**
      $v \leftarrow \dfrac{t_{m+i+1} - t_i}{t_{m+i+2} - t_{i+1}}$
      $B[j, i, m] \leftarrow B[j+1, i, 0]$
      for $k = m-1, 0$ **do**
        $B[j, i, k] \leftarrow \dfrac{t_j - t_i}{t_{j+1} - t_i} \cdot B[j, i, k+1] + \dfrac{v}{t_{j+1} - t_i} \cdot \Big((t_{j+1} - t_{m+i+2}) \cdot B[j, i+1, k] + (t_{m+i+2} - t_j) \cdot B[j, i+1, k+1]\Big)$
      end for
    end for
  end for
  return B
end procedure

3.3 Fast computation of multiple points on multiple B-spline curves

Let $u \in [t_j, t_{j+1})$ and $t := \dfrac{u - t_j}{t_{j+1} - t_j}$. By solving Problem 3.6, the Bernstein-Bézier coefficients of the B-spline functions are found. A point on a B-spline curve can thus be expressed as

$$\mathsf{S}(u) = \sum_{i=j-m}^{j} \Big(\sum_{k=0}^{m} b_k^{(i,j)} B_k^m(t) \Big) \mathsf{W}_i.$$

The inner sums

$$p_i(u) := \sum_{k=0}^{m} b_k^{(i,j)} B_k^m(t) \equiv N_{mi}(u) \qquad (i = j-m, j-m+1, \ldots, j) \tag{3.19}$$

can be treated as polynomial Bézier curves with control points

$$b_k^{(i,j)} \in \mathbb{E}^1$$

and thus can be computed using the geometric Algorithm 2.2 in total time $O(m^2)$ — more precisely, $O(m)$ per each of $m+1$ sums. It also means that — when the Bernstein-Bézier coefficients $b_k^{(i,j)}$ are already known — any B-spline function can be computed in $O(m)$ time.

Example 3.11. *A comparison of the new method of evaluating B-spline functions and using Theorem 1.102 has been done. The results have been obtained on a computer with* `Intel Core i5-6300U CPU` *at* `2.40GHz` *processor and* `4GB RAM`*, using* `GNU C Compiler 11.2.0` *(single precision).*

For each $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50\}$ *and* $m = 3, 4, \ldots, 15$*, a sequence of knots has been generated* 100 *times. The knot span lengths* $t_{j+1} - t_j \in [1/50, 1]$ *(*$j = 0, 1, \ldots, n-1$*;* $t_0 = 0$*) have been generated using the* `rand()` *C function. Then,* $50 \cdot n + 1$ *points such that* $t_{j\ell} := t_j + \ell/50 \times (t_{j+1} - t_j)$ *for* $j = 0, 1, \ldots, n-1$ *and* $\ell = 0, 1, \ldots, 49$*, with the remaining point being* $t_{n0} \equiv t_n$*, are generated. The boundary knots are coincident.*

At each point $t_{j\ell} \in [t_j, t_{j+1})$*, all* $m+1$ *B-spline functions* N_{mi} *(*$i = j-m, j-m+1, \ldots, j$*) which do not vanish at* $t_{j\ell}$ *are evaluated using both algorithms. Due to the size of the table, the resulting running times are available at* `https://bit.ly/fch-phd-phd-ch3-table-fun`*.*

The new method consistently performs faster than evaluating B-spline functions using Theorem 1.102. The new method reduced the running time for any dataset by 33-47%*, while the total running time was reduced by* 45%*. The source code in C which was used to perform the tests is available at* `https://bit.ly/fch-phd-ch3`*.*

Note that the sums p_i do not depend on the control points. Afterwards, computing a convex combination of $m+1$ points from $\mathbb{E}^d$, i.e.,

$$\mathsf{S}(u) = \sum_{i=j-m}^{j} p_i(u) \mathsf{W}_i,$$

requires $O(md)$ arithmetic operations. Observe that these values may also be computed using the geometric algorithm proposed in §2.3 (cf. Theorem 1.99). In total, then, assuming that the Bernstein-Bézier coefficients of the B-spline functions over each knot span $[t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$ are known, $O(m \cdot (m+d))$ arithmetic operations are required, for $u \in [t_j, t_{j+1})$, to compute a point $\mathsf{S}(u)$ on a B-spline curve.

When it is required to compute the values of S for many parameters $u_0, u_1, \ldots, u_N$, one would have to perform $O(nm^2)$ arithmetic operations to find the coefficients of the B-spline functions over each knot span and then do $O(m \cdot (m+d))$ operations for each of $N+1$ points that are to be computed. In total, the computational complexity of this approach is $O(nm^2 + Nm \cdot (m+d))$.

Due to the fact that the sums p_i (cf. (3.19)) do not depend on the control points, they can be used for computing a point on multiple B-spline curves, all of degree m, with the same knots.

Problem 3.12. *For M B-spline curves $\mathsf{S}_0, \mathsf{S}_1, \ldots, \mathsf{S}_{M-1}$ with the knots*

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_n = t_{n+1} = \ldots = t_{n+m}.$$

and the control points of S_k being

$$\mathsf{W}_{k,-m}, \mathsf{W}_{k,-m+1}, \ldots, \mathsf{W}_{k,n-1} \in \mathbb{E}^d \qquad (k = 0, 1, \ldots, M-1),$$

compute the value of each of the B-spline curve S_k at points

$$u_0, u_1, \ldots, u_{N-1}$$

such that $t_0 \leq u_k \leq t_n$ for all $k = 0, 1, \ldots, N-1$. More precisely, for $k = 0, 1, \ldots, M-1$ and $\ell = 0, 1, \ldots, N-1$, compute the points

$$\mathsf{S}_k(u_\ell).$$

One can efficiently solve Problem 3.12 in the following way. Using Algorithm 3.1 allows to compute all the adjusted Bernstein-Bézier coefficients of B-spline functions (cf. Problem 3.6) in $O(nm^2)$ time. Now, one needs to compute the values

$$p_i(u_\ell) \qquad (\ell = 0, 1, \ldots, N-1,\ u_\ell \in [t_j, t_{j+1}),\ i = j-m, j-m+1, \ldots, j)$$

(cf. (3.19)), which takes $O(Nm^2)$ time. Using these values, computing

$$\mathsf{S}_k(u_\ell) = \sum_{i=j-m}^{j} p_i(u_\ell)\mathsf{W}_{ki} \qquad (\ell = 0, 1, \ldots, N-1,\ k = 0, 1, \ldots, M-1,\ u_\ell \in [t_j, t_{j+1}))$$

takes $O(MNmd)$ time. In total, then, the complexity of this approach is $O(nm^2 + Nm^2 + NMmd)$, compared to the complexity of using the de Boor-Cox algorithm to solve Problem 3.12, i.e., $O(NMm^2d)$.

A comparison of running times is given in Example 3.13. The new algorithm is compared to executing the de Boor-Cox algorithm (cf. §1.9.5) and to an alternative way of computing the B-spline functions based on Theorem 1.102 (see [27, p. 55–57]) and then evaluating the point in the same way as in the new method.

Example 3.13. *Table 3.1 shows the comparison between the running times of the de Boor-Cox algorithm, an algorithm which computes the values of B-spline functions using Theorem 1.102 and then computes the points, and the new method described above and using Algorithm 3.1.*

The results have been obtained on a computer with `Intel Core i5-6300U CPU` *at* `2.40GHz` *processor and* `4GB RAM`*, using* `GNU C Compiler 11.2.0` *(single precision).*

The following numerical experiments have been conducted. For fixed $n = 20$ and $d = 2$, for each $M \in \{1, 5, 10, 20, 50, 100\}$ and $m \in \{3, 5, 7, 9, 11\}$, a sequence of knots and control points has been generated 100 times. The control points $\mathsf{W}_{ki} \in [-1, 1]^d$ ($i = -m, -m+1, \ldots, n-1$, $k = 0, 1, \ldots, M-1$) and the knot span lengths $t_{j+1} - t_j \in [1/50, 1]$ ($j = 0, 1, \ldots, n-1$; $t_0 = 0$) have been generated using the `rand()` *C function. Each algorithm is then tested using the same knots and control points. Each curve is evaluated at 1001 points which are $t_j + \ell/50 \times (t_{j+1} - t_j)$ for $j = 0, 1, \ldots, n-1$ and $\ell = 0, 1, \ldots, 49$, with the remaining point being t_n. The boundary knots are coincident. Table 3.1 shows the total running time of all $100 \times 1001 \times M$ curve evaluations for each method.*

M	m	de Boor-Cox	eval splines	new method
1	3	**0.032**	0.046	0.035
1	5	0.036	0.050	**0.033**
1	7	0.062	0.079	**0.050**
1	9	0.100	0.122	**0.075**
1	11	0.154	0.178	**0.103**
5	3	0.085	0.049	**0.041**
5	5	0.195	0.094	**0.073**
5	7	0.340	0.126	**0.096**
5	9	0.533	0.188	**0.132**
5	11	0.732	0.240	**0.167**
10	3	0.170	0.081	**0.072**
10	5	0.372	0.129	**0.109**
10	7	0.651	0.190	**0.152**
10	9	1.028	0.255	**0.199**
10	11	1.453	0.319	**0.244**
20	3	0.339	0.139	**0.127**
20	5	0.721	0.210	**0.185**
20	7	1.269	0.295	**0.250**
20	9	2.022	0.388	**0.323**
20	11	2.889	0.493	**0.401**
50	3	0.845	0.314	**0.301**
50	5	1.786	0.469	**0.427**
50	7	3.305	0.671	**0.610**
50	9	5.267	0.864	**0.756**
50	11	9.436	1.328	**1.196**
100	3	1.734	0.631	**0.618**
100	5	3.776	0.954	**0.901**
100	7	7.000	1.323	**1.225**
100	9	10.458	1.590	**1.470**
100	11	14.722	1.882	**1.712**

Table 3.1: Running times comparison (in seconds) for Example 3.13. The source code in C which was used to perform the tests is available at `https://bit.ly/fch-phd-ch3`.

Example 3.14. *Experiments similar to Example 3.13, with a wider choice of parameters, has been performed. The results have been obtained on the same computer, software, and*

precision.

For each $d \in \{1, 2, 3\}$, $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50\}$, $m = 3, 4, \ldots, 15$, *and* $M \in \{1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 50, 100\}$ *a sequence of knots and control points has been generated* 100 *times. The control points* $\mathbf{W}_{ki} \in [-1, 1]^d$ *(*$i = -m, -m+1, \ldots, n-1$, $k = 0, 1, \ldots, M-1$*) and the knot span lengths* $t_{j+1} - t_j \in [1/50, 1]$ *(*$j = 0, 1, \ldots, n-1$; $t_0 = 0$*) have been generated using the* `rand()` *C function. Each algorithm is then tested using the same knots and control points. Each curve is evaluated at* $50 \cdot n + 1$ *points which are* $t_j + \ell/50 \times (t_{j+1} - t_j)$ *for* $j = 0, 1, \ldots, n-1$ *and* $\ell = 0, 1, \ldots, 49$*, with the remaining point being* t_n*. The boundary knots are coincident. Due to the size of the table, the resulting running times are available at* `https://bit.ly/fch-phd-phd-ch3-table`*.*

The results show that the new method is significantly faster than the de Boor-Cox algorithm except for the case $M = 1$*. While the acceleration with respect to the approach which utilizes Theorem 1.102 is smaller, it is also consistent, getting lower running time in* 99.95% *test cases.*

Some statistics regarding the experiments are given in Table 3.2.

Algorithm	Total running time [s]	Relative to new method
de Boor-Cox	14769.25	6.81
eval splines	2600.92	1.20
new method	2167.92	—

Algorithm	New method win %	Max time rel. to new method	Min time rel. to new method
de Boor-Cox	97.01%	11.664	0.688
eval splines	99.95%	1.808	0.998

Table 3.2: Statistics for Example 3.14. The source code in C which was used to perform the tests is available at `https://bit.ly/fch-phd-ch3`.

3.4 Generalizations

The approach presented in Sections 3.2 and 3.5 can be generalized so that the inner knots may have their multiplicity higher than 1 (cf. Remark 3.1) or the boundary knots are of multiplicity lower than $m + 1$.

3.4.1 Inner knots of any multiplicity

When an inner knot has multiplicity over 1, some knot spans $[t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$ are empty. It is only necessary to find the B-spline functions' coefficients over the non-empty knot spans. If there are n_e such knot spans, one only needs to find $n_e(m+1)^2$ coefficients, and the algorithm will have $O(n_e m^2)$ complexity. To use the continuity condition, the following definition will be useful.

Definition 3.15. *The* left neighbor *of a given knot* t_k *is the knot* t_ℓ *if* ℓ *is the largest natural number such that* $t_\ell < t_k$*, i.e.,* $[t_\ell, t_{\ell+1})$ *is non-empty and* $t_{\ell+1} = t_k$*. The* right neighbor *of a*

given knot t_k is the knot t_r if r is the smallest natural number such that $t_k < t_r$, i.e., $[t_{r-1}, t_r)$ is non-empty and $t_k = t_{r-1}$.

Note that in the case considered in Section 3.2, the right neighbor of t_j ($j = 0, 1, \ldots, n-1$) is always t_{j+1}.

From Remark 3.1, it follows that each B-spline function is continuous in (t_0, t_n). The only modification then is in the continuity condition in Eq. (3.17). Let us consider a non-empty knot span $[t_j, t_{j+1})$ ($j = 0, 1, \ldots, n-2$). Let t_r be the right neighbor of t_{j+1}, i.e., $t_{r-1} = t_{j+1}$. In this case, the continuity property at t_{j+1} is

$$\sum_{k=0}^{m} b_k^{(i,j)} B_k^m\Big(\frac{t_{j+1} - t_j}{t_{j+1} - t_j}\Big) = \sum_{k=0}^{m} b_k^{(i,r-1)} B_k^m\Big(\frac{t_{j+1} - t_{r-1}}{t_r - t_{r-1}}\Big),$$

which simplifies to

$$b_m^{(i,j)} = b_0^{(i,r-1)}.$$

In such case, the recurrence relation (3.17) takes the form

$$\begin{cases} b_m^{(i,j)} = b_0^{(i,r-1)}, \\ b_k^{(i,j)} = \dfrac{t_j - t_i}{t_{j+1} - t_i} b_{k+1}^{(i,j)} + \dfrac{v_i}{t_{j+1} - t_i}\Big((t_{j+1} - t_{m+i+2}) b_k^{(i+1,j)} + (t_{m+i+2} - t_j) b_{k+1}^{(i+1,j)}\Big) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (k = m-1, m-2, \ldots, 0) \end{cases}$$

(cf. Eq. (3.12)), where t_r is the right neighbor of t_{j+1}, and $j = n-2, n-3, \ldots, 0$, $i = j-1, j-2, \ldots, j-m+1$. Example 3.16 presents this approach.

Example 3.16. *Let us set $m := 3$, $n := 5$. Let the knots be*

t_{-3}	t_{-2}	t_{-1}	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
0	0	0	0	3	3	5	9	10	10	10	10

The knot t_1 is of multiplicity 2. To compute the adjusted Bernstein-Bézier coefficients of the B-spline functions over $[t_0, t_1)$ a continuity condition with the knot span $[t_2, t_3)$ is used, as $t_1 = t_2$. Figure 3.2 illustrates this approach to computing all necessary coefficients, analogous to Example 3.10.

3.4.2 Boundary knots of multiplicity lower than $m + 1$

First, note that in Section 3.2, only the assumption that $t_n = t_{n+m}$ is used, therefore if that condition holds, Theorem 3.9 and Algorithm 3.1 still apply, regardless of the multiplicity of boundary knots $t_{-m}, t_{-m+1}, \ldots, t_0$.

If the boundary knot t_n has multiplicity lower than $m + 1$, the problem can be reduced so that it can be solved using Theorem 3.9. Its drawback, however, is higher complexity.

The idea is to *inflate* the multiplicity of t_{n+m} up to $m+1$. More precisely, let $t_{n+m-\ell-1} < t_{n+m-\ell} = t_{n+m}$, i.e., $t_{n+m-\ell}$ has multiplicity $\ell + 1$. Let the $m - \ell$ new knots $t_{n+m+1} = t_{n+m+2} = \ldots = t_{n+2m-\ell}$ be defined so that

$$t_{n+m} = t_{n+m+1}.$$

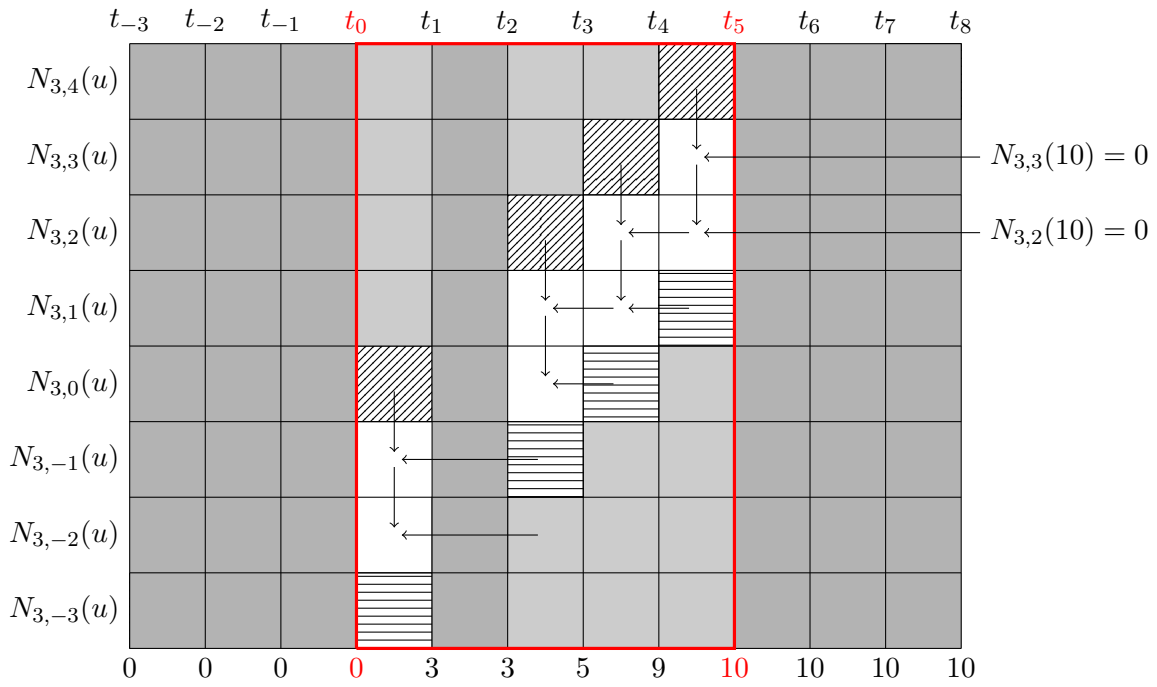

Figure 3.2: An illustration of Example 3.16.

This allows to execute Algorithm 3.1 with

$$n_1 := n + m - \ell, \quad m_1 := m$$

and

$$\underbrace{t_{-m} \leq \ldots \leq t_{-1} \leq t_0}_{\text{boundary knots}} \leq \underbrace{t_1 < \ldots < t_{n+m-1}}_{\text{inner knots}} \leq \underbrace{t_{n+m} = t_{n+m+1} = \ldots = t_{n+2m-\ell}}_{\text{boundary knots}}.$$

It remains then to return the coefficients of N_{mi} over $[t_j, t_{j+1})$ for $j = 0, 1, \ldots, n-1$ and $i = j-m, j-m+1, \ldots, j$. This approach requires the computation of $O((n+m-\ell)m^2)$ coefficients and is presented in Example 3.17.

Example 3.17. *Let us set $m := 3$, $n := 2$. Let the knots be*

t_{-3}	t_{-2}	t_{-1}	t_0	t_1	t_2	t_3	t_4	t_5
-3	-2	-1	0	1	2	3	4	5

.

After adding the knots $t_6 = t_7 = t_8$ such that $t_5 = t_8$ (thus increasing n by 3), the problem takes the form

t_{-3}	t_{-2}	t_{-1}	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
-3	-2	-1	0	1	2	3	4	5	5	5	5

.

Figure 3.3 illustrates the application of Algorithm 3.1 (cf. Example 3.10) computing all the adjusted Bernstein-Bézier coefficients of the inflated problem. The coefficients which are relevant to the solution of the primary problem are in the red frame.

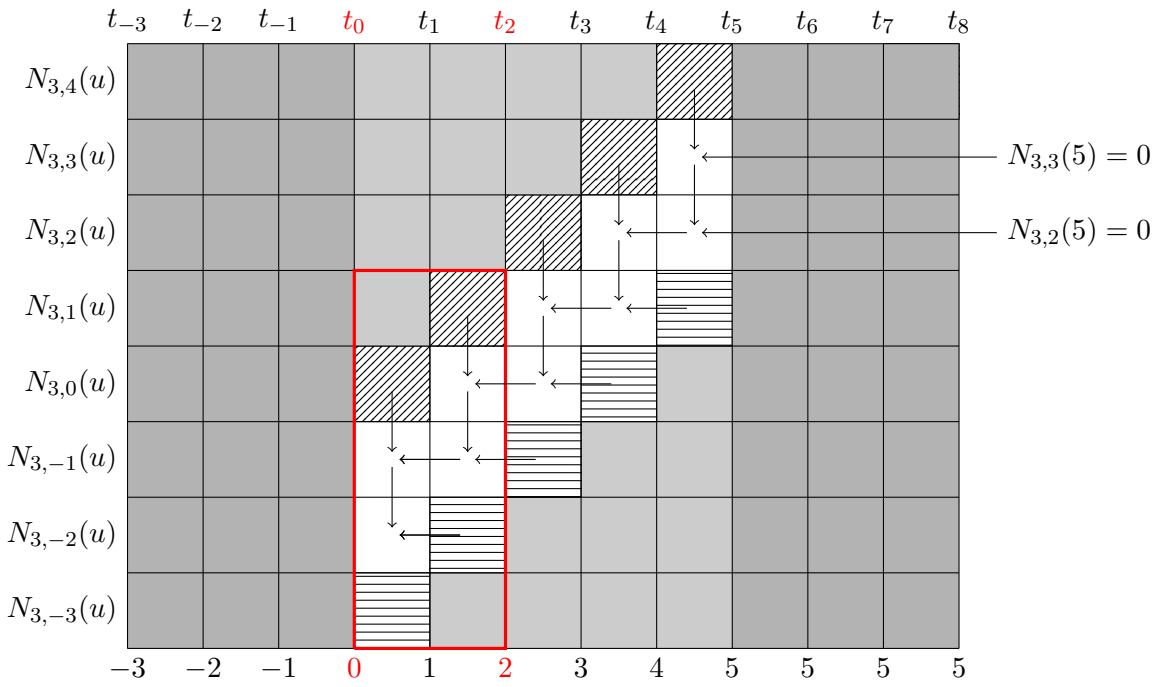

Figure 3.3: An illustration of Example 3.17.

3.5 B-spline functions' coefficients in adjusted power basis

The coefficients of the B-spline functions in the adjusted power basis can be computed using a very similar approach to the one presented in Section 3.2 for the Bernstein-Bézier basis. In this section, the approach will be outlined, with some details omitted when they are analogous to the Bernstein-Bézier case.

Just as in Section 3.2, let us assume that the boundary knots are coincident and all inner knots are of multiplicity 1, i.e.,

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_n = t_{n+1} = \ldots = t_{n+m}$$

(cf. (1.73)). Recall that N_{mi} over $[t_j, t_{j+1})$ in the adjusted power basis has the form (3.1).

Remark 3.18. *In the sequel, it will be assumed that* $u \in [t_j, t_{j+1})$.

For $j = 0, 1, \ldots, n-1$ and $i = j$, using Eq. (1.74) gives an explicit representation in the adjusted power basis:

$$N_{mi}(u) = \frac{u - t_i}{t_{m+i} - t_i} N_{m-1,i}(u) = \ldots = \prod_{j=1}^{m} \frac{u - t_i}{t_{j+i} - t_i} N_{0i}(u) = \frac{(u - t_i)^m}{\prod_{j=1}^{m}(t_{j+i} - t_i)}.$$

In the same way, an expression for N_{mi} over $[t_{m+i}, t_{m+i+1})$ could be found. However, the recurrence relation (1.74) would give it in the $(u - t_{m+i+1})^k$ basis and one would need $O(m^2)$ operations to convert it to the $(u - t_{m+i})^k$ basis. Due to that, Eq. (3.6) will be used to find the coefficients of $N_{m,j-m}$ over the knot span $[t_j, t_{j+1})$ $(j = 0, 1, \ldots, n-1)$.

For $j = n-1, n-2, \ldots, 0$ and $i = j-1, j-2, \ldots, j-m$, Eq. (3.6) is be used to get m equations:

$$(m-k)a_k^{(i,j)} + (k+1)(t_i - t_j)a_{k+1}^{(i,j)} = v_i\Big((m-k)a_k^{(i+1,j)} + (k+1)(t_{m+i+2} - t_j)a_{k+1}^{(i+1,j)}\Big) \qquad (k = 0, 1, \ldots, m-1),$$

where

$$v_i \equiv v_{mi} := \frac{t_{m+i+1} - t_i}{t_{m+i+2} - t_{i+1}}$$

(cf. Eq. (3.12)). In a particular case of $j = 0$, the relation simplifies to

$$a_k^{(i,0)} = v_i\Big(a_k^{(i+1,0)} + \frac{(k+1)(t_{m+i+2} - t_0)}{m-k}a_{k+1}^{(i+1,0)}\Big) \qquad (k = 0, 1, \ldots, m-1),$$

giving the expressions for all coefficients except $a_m^{(i,0)}$.

If $j = n-1$, one can use Remark 3.7 to complete the system of equations:

$$\begin{bmatrix} s_0 & s_1 & s_2 & & \cdots & s_m \\ l_0 & d_0 & 0 & & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ & & l_k & d_k & & \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & & \cdots & 0 & l_{m-1} & d_{m-1} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(i,n-1)} \\ a_1^{(i,n-1)} \\ \vdots \\ a_k^{(i,n-1)} \\ \vdots \\ a_m^{(i,n-1)} \end{bmatrix} = v_i \begin{bmatrix} 0 \\ l_0 a_0^{(i+1)} + r_0 a_1^{(i+1)} \\ \vdots \\ l_k a_k^{(i+1)} + r_k a_{k+1}^{(i+1)} \\ \vdots \\ l_{m-1} a_{m-1}^{(i+1)} + r_{m-1} a_m^{(i+1)} \end{bmatrix},$$

where $a_\ell^{(j)} \equiv a_\ell^{(j,n-1)}$ and

$$l_k := m-k, \quad d_k := (k+1)(t_i - t_{n-1}), \quad r_k := (k+1)(t_n - t_{n-1}), \quad s_k := (t_n - t_{n-1})^k.$$

One can use Gaussian elimination so that only the last element in the first row of the matrix is non-zero. For $k = 0, 1, \ldots, m-1$, let g_k be the factor by which the $(k+2)$th row (i.e., the row containing l_k and d_k) is multiplied before being subtracted from the first row. More precisely, the following recurrence relation with an initial value needs to be satisfied:

$$\begin{cases} g_0 = m^{-1}, \\ g_k = \dfrac{(t_n - t_{n-1})^k}{m-k} - g_{k-1}\dfrac{k(t_i - t_{n-1})}{m-k} \quad (k = 1, 2, \ldots, m-1). \end{cases} \tag{3.20}$$

It is clear that one can compute all $g_0, g_1, \ldots, g_{m-1}$ in $O(m)$ time. One can check that

$$g_k = \frac{(t_{n-1} - t_i)^k}{m\binom{m-1}{k}} \sum_{h=0}^{k} \binom{m}{h} \Big(-\frac{t_n - t_{n-1}}{t_i - t_{n-1}}\Big)^h \qquad (k = 0, 1, \ldots, m-1),$$

however, it will simplify the expressions and computation if the recursive form is used instead. After performing the elimination, the first row of the matrix gives an expression for $a_m^{(i,n-1)}$:

$$a_m^{(i,n-1)} = \frac{-v_i}{s_m - g_{m-1}d_{m-1}} \sum_{k=0}^{m-1} g_k(l_k a_k^{(i+1,n-1)} + r_k a_{k+1}^{(i+1,n-1)}).$$

This can be computed in $O(m)$ time. The remaining coefficients can be then found using the recurrence relations given in other rows of the matrix:

$$\begin{cases} a_m^{(i)} = \dfrac{-v_i}{s_m - g_{m-1}d_{m-1}} \displaystyle\sum_{k=0}^{m-1} g_k(l_k a_k^{(i+1)} + r_k a_{k+1}^{(i+1)}), \\ a_k^{(i)} = l_k^{-1}\Big(- d_k a_{k+1}^{(i)} + v_i(l_k a_k^{(i+1)} + r_k a_{k+1}^{(i+1)})\Big) \qquad (k = m-1, m-2, \ldots, 0), \end{cases}$$

where $a_\ell^{(j)} \equiv a_\ell^{(j,n-1)}$ and

$$l_k := m - k, \quad d_k := (k+1)(t_i - t_{n-1}), \quad r_k := (k+1)(t_n - t_{n-1}), \quad s_k := (t_n - t_{n-1})^k,$$

and the values g_k are given by (3.20).

For $j = 0, 1, \ldots, n-2$, finding the initial value for the recurrence scheme can be done by using the continuity condition at t_{j+1}. The knot t_{j+1} has multiplicity 1, therefore, from Theorem 1.98, it follows that the $(m-1)$th derivative of N_{mi} is continuous at t_{j+1}:

$$N_{mi}^{(m-1)}(t_{j+1}^-) = N_{mi}^{(m-1)}(t_{j+1}^+).$$

It is easy to check that

$$N_{mi}^{(m-1)}(t_{j+1}^-) = a_{m-1}^{(i,j)}(m-1)! + a_m^{(i,j)} m!(t_{j+1} - t_j)$$

and

$$N_{mi}^{(m-1)}(t_{j+1}^+) = a_{m-1}^{(i,j+1)}(m-1)!,$$

thus

$$a_{m-1}^{(i,j)} + a_m^{(i,j)} m(t_{j+1} - t_j) = a_{m-1}^{(i,j+1)}.$$

This, together with the previously found equation, i.e.,

$$a_{m-1}^{(i,j)} + m(t_i - t_j)a_m^{(i,j)} = v_i\Big(a_{m-1}^{(i+1,j)} + m(t_{m+i+2} - t_j)a_m^{(i+1,j)}\Big),$$

allows to find an expression for $a_m^{(i,j)}$ (assuming that $a_{m-1}^{(i+1,j)}$ and $a_m^{(i+1,j)}$ are known):

$$m(t_{j+1} - t_i)a_m^{(i,j)} = a_{m-1}^{(i,j+1)} - v_i\Big(a_{m-1}^{(i+1,j)} + m(t_{m+i+2} - t_j)a_m^{(i+1,j)}\Big),$$

which completes the recurrence scheme. This proves the following theorem.

Theorem 3.19. *Let*

$$t_{-m} = t_{-m+1} = \ldots = t_0 < t_1 < \ldots < t_{n-1} < t_n = t_{n+1} = \ldots = t_{n+m}$$

(cf. (1.73)*). The* $n(m+1)^2$ *coefficients* $a_k^{(i,j)}$ *of the B-spline functions* N_{mi} *(*$i = -m, -m+1, \ldots, n-1$*) over each knot span* $[t_j, t_{j+1})$ *(*$0 \le j \le n-1$*,* $i = j-m, j-m+1, \ldots, j$*,* $0 \le k \le m$*) in the adjusted power basis (cf. Eq.* (3.2)*) can be computed in* $O(nm^2)$ *time in the following way.*

1. *For* $j = 0, 1, \ldots, n-1$ *and* $i = j$*, we have*

$$\begin{cases} a_k^{(j,j)} = 0 \qquad (k = 0, 1, \ldots, m-1), \\ a_m^{(j,j)} = \Big(\prod_{\ell=1}^m (t_{j+\ell} - t_j)\Big)^{-1}. \end{cases}$$

2. *For $j = n-1$ and $i = n-2, n-3, \ldots, n-1-m$, the coefficients of N_{mi} are given by the recurrence relation*

$$\begin{cases} a_m^{(i,n-1)} = \dfrac{-v_i}{s_m - g_{m-1}d_{m-1}} \displaystyle\sum_{k=0}^{m-1} g_k (l_k a_k^{(i+1,n-1)} + r_k a_{k+1}^{(i+1,n-1)}), \\ a_k^{(i,n-1)} = l_k^{-1}\Big(- d_k a_{k+1}^{(i,n-1)} + v_i (l_k a_k^{(i+1,n-1)} + r_k a_{k+1}^{(i+1,n-1)})\Big) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (k = m-1, m-2, \ldots, 0), \end{cases} \tag{3.21}$$

where

$$l_k := m-k, \quad d_k := (k+1)(t_i - t_{n-1}), \quad r_k := (k+1)(t_n - t_{n-1}), \quad s_k := (t_n - t_{n-1})^k,$$

and the values g_k are given by (3.20).

3. *For $j = n-2, n-3, \ldots, 0$ and $i = j-1, j-2, \ldots, j-m$, the coefficients of N_{mi} over $[t_j, t_{j+1})$ are given by the recurrence relation*

$$\begin{cases} a_m^{(i,j)} = \dfrac{a_{m-1}^{(i,j+1)} - v_i\Big(a_{m-1}^{(i+1,j)} + m(t_{m+i+2} - t_j) a_m^{(i+1,j)}\Big)}{m(t_{j+1} - t_i)}, \\ a_k^{(i,j)} = \dfrac{(k+1)(t_j - t_i)}{m-k} a_{k+1}^{(i,j)} + v_i\Big(a_k^{(i+1,j)} + \dfrac{(k+1)(t_{m+i+2} - t_j)}{m-k} a_{k+1}^{(i+1,j)}\Big) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (k = 0, 1, \ldots, m-1). \end{cases}$$

Algorithm 3.2 implements the approach given in Theorem 3.19. It returns a sparse array $A \equiv A[0..n-1, -m..n-1, 0..m]$, where

$$A[j, i, k] = a_k^{(i,j)} \qquad (0 \le j < n,\ -m \le i < n,\ 0 \le k \le m)$$

(cf. (3.2)).

Similarly to the case of Bernstein-Bézier basis, the complexity of Algorithm 3.2 is $O(nm^2)$ — giving the optimal $O(1)$ time per coefficient.

Algorithm 3.2 Computing the coefficients of the adjusted power form of the B-spline functions

procedure BSplineP($n, m, [t_{-m}, t_{-m+1}, \ldots, t_{n+m}]$)
  $A \leftarrow$ `SparseArray[0..n-1, -m..n-1, 0..m](fill=0)`
  for $i \leftarrow 0, n-1$ **do**
    $A[i, i, m] \leftarrow \left(\prod_{\ell=1}^{m} (t_{i+\ell} - t_i) \right)^{-1}$
  end for
  for $i \leftarrow n-2, n-1-m$ **do**
    $A[n-1, i, 0..m] \leftarrow$ `Eq.` (3.21)
  end for
  for $j \leftarrow n-2, 0$ **do**
    for $i \leftarrow j-1, j-m$ **do**
      $v \leftarrow \dfrac{t_{m+i+1} - t_i}{t_{m+i+2} - t_{i+1}}$
      $w \leftarrow A[j, i+1, m-1] + m \cdot (t_{m+i+2} - t_j) \cdot A[j, i+1, m]$
      $A[j, i, m] \leftarrow \dfrac{A[j+1, i, m-1] - v \cdot w}{m \cdot (t_{j+1} - t_i)}$
      for $k = m-1, 0$ **do**
        $q \leftarrow \dfrac{k+1}{m-k}$
        $w \leftarrow A[j, i+1, k] + (t_{m+i+2} - t_j) \cdot q \cdot A[j, i+1, k+1]$
        $A[j, i, k] \leftarrow (t_j - t_i) \cdot q \cdot A[j, i, k+1] + v \cdot w$
      end for
    end for
  end for
  return A
end procedure

Chapter 4

New differential relations for dual Bernstein polynomials

Let $\boldsymbol{D}$ be the differentiation operator with respect to the variable x ($\boldsymbol{D} := \dfrac{\mathrm{d}}{\mathrm{d}x}$) and $\boldsymbol{I}$ be the identity operator ($\forall z, \boldsymbol{I}z = z$) (cf. §1.32). In the sequel, let $\sigma := \alpha + \beta + 1$ and $K := \dfrac{\Gamma(\sigma+1)}{\Gamma(\alpha+1)\Gamma(\beta+1)}$ (cf. (1.9) and (1.8)).

Recall that the ith dual Bernstein polynomial (see §1.5.2) of degree n with parameters α, β is given by

$$D_i^n(x;\alpha,\beta) = A_{ni}^{(\alpha,\beta)} \frac{(\alpha+1)_n}{(n+1)!} \sum_{j=0}^{n} B_{nj}^{(\alpha,\beta)} F(i,j) \cdot (1-x)^j,$$

where

$$A_{ni}^{(\alpha,\beta)} := \frac{(-1)^{n-i}(n+1)(\sigma+1)_n}{K\,(\alpha+1)_{n-i}(\beta+1)_i}, \qquad B_{nj}^{(\alpha,\beta)} := \frac{(-n)_j(n+\sigma+1)_j}{j!(\alpha+1)_j}$$

and $F(i,j)$ is a hypergeometric function defined as

$$F(i,j) := {}_3F_2\left(\begin{matrix} j-n,\ -i,\ 1 \\ -n,\ -n-\alpha \end{matrix} \,\middle|\, 1 \right) \qquad (i,j = 0,1,\ldots,n),$$

and 0 otherwise (cf. (1.43)). There are other representations of dual Bernstein polynomials, given, e.g., in equations (1.36) and (1.38). They will not, however, be used in this chapter.

In the chapter, results published in [18] are presented. The differential-recurrence relations given in Section 4.1 will be useful in deriving differential equations for dual Bernstein polynomials in Section 4.2. The differential-recurrence relations given in Section 4.1 will find further use in Chapter 5.

4.1 Differential-recurrence relations

Theorem 4.1. *For $i = 0,1,\ldots,n$, the following formula holds:*

$$\Big((1-x)\boldsymbol{D} - (n-i+\alpha+1)\boldsymbol{I}\Big) D_i^n(x;\alpha,\beta) = \frac{(i-n)(i+\beta+1)}{i+1} D_{i+1}^n(x;\alpha,\beta) - A_{ni}^{(\alpha,\beta)} \frac{n+\alpha+1}{i+1} R_n^{(\alpha,\beta+1)}(x) \quad (4.1)$$

(cf. (1.43)*).*

Proof. Using the representation of $D_i^n(x;\alpha,\beta)$ and $D_{i+1}^n(x;\alpha,\beta)$ in the $(1-x)^j$ basis, the fact that

$$A_{n,i+1}^{(\alpha,\beta)} = \frac{i-n-\alpha}{\beta+1+i}\cdot A_{ni}^{(\alpha,\beta)},$$

and the representation of shifted Jacobi polynomials in the $(1-x)^j$ basis, i.e.,

$$R_n^{(\alpha,\beta+1)}(x) = \frac{(\alpha+1)_n}{n!}\sum_{j=0}^{n} B_{nj}^{(\alpha,\beta)}(1-x)^j$$

(cf. (1.14)), it is possible to express the hypothesis in its equivalent form:

$$\sum_{j=0}^{n} G(n,i,j,\alpha)\cdot B_{nj}^{(\alpha,\beta)}(1-x)^j \equiv 0,$$

where

$$G(n,i,j,\alpha) := -F(i,j)\cdot(n-i+\alpha+1+j)(i+1)+(i-n)(n-i+\alpha)F(i+1,j)+(n+\alpha+1)(n+1).$$

To prove the theorem, it is sufficient to prove that, for given n,i,α and all $j=0,1,\ldots,n$, $G(n,i,j,\alpha)=0$. Indeed, using the Zeilberger's algorithm (see §1.3.1), this relation holds true. The proof is quite technical and is therefore omitted. □

From Theorem 4.1, a second differential-recurrence relation can be derived.

Theorem 4.2. *For $i=0,1,\ldots,n$, we have*

$$\Big(x\boldsymbol{D}+(i+\beta+1)\boldsymbol{I}\Big)D_i^n(x;\alpha,\beta) = \frac{i(n-i+\alpha+1)}{n-i+1}D_{i-1}^n(x;\alpha,\beta) + A_{ni}^{(\alpha,\beta)}\frac{n+\beta+1}{n-i+1}R_n^{(\alpha+1,\beta)}(x). \tag{4.2}$$

Proof. The theorem can be proven by applying symmetry relations (1.15) and (1.37) in Eq. (4.1):

$$\Big((1-x)\boldsymbol{D}-(n-i+\alpha+1)\boldsymbol{I}\Big)D_{n-i}^n(1-x;\beta,\alpha) = \frac{(i-n)(i+\beta+1)}{i+1}D_{n-i-1}^n(1-x;\beta,\alpha) - A_{ni}^{(\alpha,\beta)}\frac{n+\alpha+1}{i+1}(-1)^n R_n^{(\beta+1,\alpha)}(1-x).$$

The reassignment of variables

$$x := 1-x, \quad i := n-i, \quad \alpha := \beta, \quad \beta := \alpha$$

results in a new identity having the following form:

$$\Big(-x\boldsymbol{D}-(i+\beta+1)\boldsymbol{I}\Big)D_i^n(x;\alpha,\beta) = \frac{-i(n-i+\alpha+1)}{n-i+1}D_{i-1}^n(x;\alpha,\beta) - A_{n,n-i}^{(\beta,\alpha)}\frac{n+\beta+1}{n-i+1}(-1)^n R_n^{(\alpha+1,\beta)}(x).$$

After checking that

$$(-1)^n A_{n,n-i}^{(\beta,\alpha)} = \frac{(-1)^{n+i}(n+1)(\sigma+1)_n}{K\,(\beta+1)_i(\alpha+1)_{n-i}} = (-1)^{2i}\cdot A_{ni}^{(\alpha,\beta)} = A_{ni}^{(\alpha,\beta)},$$

the proof is concluded. □

The third differential-recurrence relation is of a different kind. It relates the first and second derivatives with three consecutive dual Bernstein polynomials.

Theorem 4.3. *The following relation holds:*

$$\begin{aligned}\Big(x(x-1)\boldsymbol{D}^2+\tfrac{1}{2}(\alpha-\beta+(\sigma+1)(2x-1))\boldsymbol{D}\Big)D_i^n(x;\alpha,\beta) \qquad (4.3)\\ =(i-n)(i+\beta+1)D_{i+1}^n(x;\alpha,\beta)+i(i-\alpha-n-1)D_{i-1}^n(x;\alpha,\beta)\\ -(i(i-\alpha-n-1)+(i-n)(i+\beta+1))D_i^n(x;\alpha,\beta),\end{aligned}$$

where $i=0,1,\ldots,n$.

Proof. Note that the operator on the left-hand-side of the equation is identical to the operator used in Eq. (1.19), namely

$$\boldsymbol{L}^{(\alpha,\beta)}:=x(x-1)\boldsymbol{D}^2+\tfrac{1}{2}\left(\alpha-\beta+(\sigma+1)(2x-1)\right)\boldsymbol{D}.$$

Using this observation and the representation (1.36) of dual Bernstein polynomials,

$$D_i^n(x;\alpha,\beta)=K^{-1}\sum_{k=0}^{n}(-1)^k\frac{(2k/\sigma+1)(\sigma)_k}{(\alpha+1)_k}Q_k(i;\beta,\alpha;n)R_k^{(\alpha,\beta)}(x),$$

one can express the left-hand side of the hypothesis as

$$\boldsymbol{L}^{(\alpha,\beta)}D_i^n(x;\alpha,\beta)=K^{-1}\sum_{k=0}^{n}(-1)^k\frac{(2k/\sigma+1)(\sigma)_k}{(\alpha+1)_k}Q_k(i;\beta,\alpha;n)\boldsymbol{L}^{(\alpha,\beta)}R_k^{(\alpha,\beta)}(x).$$

Now, one can apply Eq. (1.19) to each element of the sum, giving

$$\boldsymbol{L}^{(\alpha,\beta)}D_i^n(x;\alpha,\beta)=K^{-1}\sum_{k=0}^{n}(-1)^k\frac{(2k/\sigma+1)(\sigma)_k}{(\alpha+1)_k}R_k^{(\alpha,\beta)}(x)k(k+\sigma)Q_k(i;\beta,\alpha;n).$$

Recall (cf. Eq. (1.22)) that

$$k(k+\sigma)Q_k(i;\beta,\alpha;n)=\mathcal{L}_i^{(\beta,\alpha,n)}Q_k(i;\beta,\alpha;n),$$

where the operator $\mathcal{L}_i^{(\beta,\alpha,n)}$ is given by (1.23). Applying this relation to the left-hand side, in connection with Eq. (1.36), gives

$$\begin{aligned}\boldsymbol{L}^{(\alpha,\beta)}D_i^n(x;\alpha,\beta)&=\mathcal{L}_i^{(\beta,\alpha,n)}D_i^n(x;\alpha,\beta)\\ &\equiv(i-n)(i+\beta+1)D_{i+1}^n(x;\alpha,\beta)+i(i-\alpha-n-1)D_{i-1}^n(x;\alpha,\beta)\\ &\quad-(i(i-\alpha-n-1)+(i-n)(i+\beta+1))D_i^n(x;\alpha,\beta).\end{aligned}$$

□

4.2 Differential equations for dual Bernstein polynomials

The new differential-recurrence relations given in Section 4.1 can be used to derive differential equations for $D_i^n(x;\alpha,\beta)$.

Theorem 4.4. *Dual Bernstein polynomials satisfy the second-order non-homogeneous differential equation with polynomial coefficients of the form*

$$\boldsymbol{M}_{ni}^{(\alpha,\beta)} D_i^n(x;\alpha,\beta) = (n+\sigma+1) A_{ni}^{(\alpha,\beta)} R_n^{(\alpha+1,\beta+1)}(x), \tag{4.4}$$

(cf. (1.43)*), where*

$$\boldsymbol{M}_{ni}^{(\alpha,\beta)} := x(x-1)\boldsymbol{D}^2 + \Big((n+\sigma+3)x - i - \beta - 2\Big)\boldsymbol{D} + (n+\sigma+1)\boldsymbol{I}.$$

Proof. Using Theorem 4.1, one can find an expression for $D_{i+1}^n(x;\alpha,\beta)$ for $i = 0, 1, \ldots, n-1$:

$$D_{i+1}^n(x;\alpha,\beta) = \frac{i+1}{(i-n)(i+\beta+1)}\Big(\Big((1-x)\boldsymbol{D} - (n-i+\alpha+1)\boldsymbol{I}\Big) D_i^n(x;\alpha,\beta) + A_{ni}^{(\alpha,\beta)} \frac{n+\alpha+1}{i+1} R_n^{(\alpha,\beta+1)}(x)\Big). \tag{4.5}$$

Similarly, from Theorem 4.2 follows an expression for $D_{i-1}^n(x;\alpha,\beta)$ for $i = 1, 2, \ldots, n$:

$$D_{i-1}^n(x;\alpha,\beta) = \frac{(n-i+1)}{i(n-i+\alpha+1)}\Big(\Big(x\boldsymbol{D} + (i+\beta+1)\boldsymbol{I}\Big) D_i^n(x;\alpha,\beta) - A_{ni}^{(\alpha,\beta)} \frac{n+\beta+1}{n-i+1} R_n^{(\alpha+1,\beta)}(x)\Big). \tag{4.6}$$

Now, substituting equations (4.5) and (4.6) into the right-hand side of Eq. (4.3) and applying simple algebra gives

$$\Big(x(x-1)\boldsymbol{D}^2 + ((n+\sigma+3)x - (i+\beta+2))\boldsymbol{D} + (n+\sigma+1)\boldsymbol{I}\Big) D_i^n(x;\alpha,\beta) = A_{ni}^{(\alpha,\beta)}\Big((n+\alpha+1)R_n^{(\alpha,\beta+1)}(x) + (n+\beta+1)R_n^{(\alpha+1,\beta)}(x)\Big).$$

Note that the left-hand side is exactly $\boldsymbol{M}_{ni}^{(\alpha,\beta)} D_i^n(x;\alpha,\beta)$.

What remains is to check that

$$(n+\alpha+1)R_n^{(\alpha,\beta+1)}(x) + (n+\beta+1)R_n^{(\alpha+1,\beta)}(x) = (n+\sigma+1)R_n^{(\alpha+1,\beta+1)}(x).$$

This follows from the representation of shifted Jacobi polynomials (cf. Eq. (1.14)), which, when applied to the left-hand side, gives

$$\begin{aligned}
&(n+\alpha+1)R_n^{(\alpha,\beta+1)}(x) + (n+\beta+1)R_n^{(\alpha+1,\beta)}(x) \\
&\quad= \frac{(\alpha+2)_n}{n!}\sum_{k=0}^{n}\Big(\frac{(\alpha+1)(\alpha+2)_k}{(\alpha+1)_k} + n + \beta + 1\Big)\frac{(-n)_k(n+\alpha+\beta+2)_k}{(\alpha+2)_k k!}(1-x)^k \\
&\quad= (n+\sigma+1)\frac{(\alpha+2)_n}{n!}\Big(1 + \sum_{k=1}^{n}\frac{n+\alpha+\beta+2+k}{n+\alpha+\beta+2}\cdot\frac{(-n)_k(n+\alpha+\beta+2)_k}{(\alpha+2)_k k!}(1-x)^k\Big) \\
&= \Big(n+\sigma+1\Big)\frac{(\alpha+2)_n}{n!}\sum_{k=0}^{n}\frac{(-n)_k(n+\alpha+\beta+3)_k}{(\alpha+2)_k k!}(1-x)^k = \Big(n+\sigma+1\Big)R_n^{(\alpha+1,\beta+1)}(x).
\end{aligned}$$

□

The existence of a second-order non-homogeneous differential equation means that there exists a homogeneous third-order differential equation. Indeed, one can apply the first-order differential operator

$$\Big(R_n^{(\alpha+1,\beta+1)}(x)\boldsymbol{D} - (\boldsymbol{D}R_n^{(\alpha+1,\beta+1)}(x))\boldsymbol{I}\Big) \equiv \Big(R_n^{(\alpha+1,\beta+1)}(x)\boldsymbol{D} - (n+\alpha+\beta+3)R_{n-1}^{(\alpha+2,\beta+2)}(x)\boldsymbol{I}\Big)$$

(cf. (1.16)) to eliminate the right-hand side.

Corollary 4.5. *Dual Bernstein polynomials $D_i^n(x;\alpha,\beta)$ $(i=0,1,\ldots,n)$ satisfy the third-order differential equation with polynomial coefficients of the form*

$$\Big(R_n^{(\alpha+1,\beta+1)}(x)\boldsymbol{D} - (n+\alpha+\beta+3)R_{n-1}^{(\alpha+2,\beta+2)}(x)\boldsymbol{I}\Big)\boldsymbol{M}_{ni}^{(\alpha,\beta)}D_i^n(x;\alpha,\beta) = 0$$

(cf. Theorem 4.4).

The polynomials $R_n^{(\alpha+1,\beta+1)}(x), R_{n-1}^{(\alpha+2,\beta+2)}(x)$ are not computationally simple. For practical reasons, it may be useful to find a differential equation with simple coefficients – even at the cost of higher order.

Recall (cf. (1.19)) that

$$\Big(\boldsymbol{L}^{(\alpha,\beta)} - n(n+\sigma)\boldsymbol{I}\Big)R_n^{(\alpha,\beta)}(x) = 0.$$

Let $\boldsymbol{N}_n^{(\alpha,\beta)}$ be the second-order differential operator of the form

$$\boldsymbol{N}_n^{(\alpha,\beta)} := \boldsymbol{L}^{(\alpha,\beta)} - n(n+\sigma)\boldsymbol{I}.$$

Applying the operator $\boldsymbol{N}_n^{(\alpha+1,\beta+1)}$ to both sides of Eq. (4.4) allows to eliminate the non-homogeneity, which results in the following homogeneous differential equation for $D_i^n(x;\alpha,\beta)$.

Corollary 4.6. *Dual Bernstein polynomials $D_i^n(x;\alpha,\beta)$ $(i=0,1,\ldots,n)$ satisfy the fourth-order differential equation with polynomial coefficients of the form*

$$\boldsymbol{Q}_4 D_i^n(x;\alpha,\beta) \equiv \boldsymbol{N}_n^{(\alpha+1,\beta+1)}\boldsymbol{M}_{ni}^{(\alpha,\beta)}D_i^n(x;\alpha,\beta) = 0. \tag{4.7}$$

After evaluating the composition of two second-order differential operators, one can express the differential equation (4.7) in an equivalent form:

$$\sum_{j=0}^{4} w_j(x)\boldsymbol{D}^j D_i^n(x;\alpha,\beta) = 0,$$

where

$$\begin{aligned}
&w_4(x) := x^2(x-1)^2, \qquad w_3(x) := x(x-1)[(n+2\sigma+10)x - i - 2\beta - 6],\\
&w_2(x) := [(n+\sigma+3)(\sigma-n+7)+\sigma+3]x^2\\
&\qquad\qquad +[(n-1)^2+\alpha n - 2\beta - (\sigma+3)(i+2\beta+8) - 5]x + (\beta+2)(i+\beta+3),\\
&w_1(x) := -(n+\sigma+2)[(n^2+(n-2)(\sigma+3))x + (2-n)(i+\beta+2) - 2i],\\
&w_0(x) := -n(n+\sigma+1)_2.
\end{aligned}$$

Note that

$$\deg w_j = j.$$

The relations given in this chapter have been used in [18] to find a recurrence relation of order 4 for dual Bernstein polynomials of the same degree. This result, along with new recurrence relations of lower order, will be presented in the next chapter.

Chapter 5

New recurrence relations for dual Bernstein polynomials

In the sequel, let $\sigma := \alpha + \beta + 1$ and $K := \dfrac{\Gamma(\sigma+1)}{\Gamma(\alpha+1)\Gamma(\beta+1)}$ (cf. (1.9) and (1.8)).

5.1 Homogeneous fourth-order recurrence relation

Using the results given in Chapter 4, it is possible to construct a homogeneous recurrence relation connecting five consecutive (with respect to i) dual Bernstein polynomials of the same degree n.

Let $\mathcal{E}^m$ be the mth *shift operator* acting on the variable i in the following way:

$$\mathcal{E}^m z_i := z_{i+m} \qquad (m \in \mathbb{Z}). \tag{5.1}$$

To simplify the notation, let $\mathcal{I} \equiv \mathcal{E}^0$ and $\mathcal{E} \equiv \mathcal{E}^1$. The following theorem holds.

Theorem 5.1. *Dual Bernstein polynomials satisfy the second-order non-homogeneous recurrence relation of the form*

$$\mathcal{M}_i^{(\alpha,\beta,n)} D_i^n(x;\alpha,\beta) = G_{ni}^{(\alpha,\beta)}(x), \tag{5.2}$$

where $i = 0, 1, \ldots, n$, *and*

$$\begin{aligned}
\mathcal{M}_i^{(\alpha,\beta,n)} &:= (i)_2(n-i+\alpha+1)(x-1)\mathcal{E}^{-1} - (n-i)_2(i+\beta+1)x\mathcal{E} \\
&\quad +(i+1)(n-i+1)\Big((i+\beta+1)(1-x) + (n-i+\alpha+1)x\Big)\mathcal{I}, \\
G_{ni}^{(\alpha,\beta)}(x) &:= A_{ni}^{(\alpha,\beta)}\Big((i+1)(n+\beta+1)(1-x)R_n^{(\alpha+1,\beta)}(x) \\
&\quad +(n-i+1)(n+\alpha+1)xR_n^{(\alpha,\beta+1)}(x)\Big)
\end{aligned}$$

(cf. (1.43)*).*

Proof. The result follows directly from subtracting the relation (4.1), multiplied by x, from the relation (4.2), multiplied by $1-x$. □

Notice that the quantity $\left(A_{ni}^{(\alpha,\beta)}\right)^{-1} G_{ni}^{(\alpha,\beta)}(x)$ is a polynomial of the first degree in variable i. One can thus eliminate it using the identity $(\mathcal{E}-\mathcal{I})^2 \left(A_{ni}^{(\alpha,\beta)}\right)^{-1} G_{ni}^{(\alpha,\beta)}(x) = 0$. By applying the operator

$$\mathcal{N}_i^{(\alpha,\beta,n)} := \mathcal{E}^{-1}(\mathcal{E}-\mathcal{I})^2 \left(A_{ni}^{(\alpha,\beta)}\right)^{-1} \mathcal{I}$$

to both sides of Eq. (5.2), a fourth-order homogeneous recurrence relation for the dual Bernstein polynomials is obtained. Note that $\mathcal{E}^{-1}$ in the operator $\mathcal{N}_i^{(\alpha,\beta,n)}$ applies to all the terms and by itself does not change its order.

Corollary 5.2. *Dual Bernstein polynomials satisfy the fourth-order recurrence relation of the form*

$$\mathcal{N}_i^{(\alpha,\beta,n)} \mathcal{M}_i^{(\alpha,\beta,n)} D_i^n(x;\alpha,\beta) = 0 \qquad (0 \le i \le n). \tag{5.3}$$

The operator $\mathcal{N}_i^{(\alpha,\beta,n)} \mathcal{M}_i^{(\alpha,\beta,n)}$ is a composition of two second-order difference operators. The explicit form of the simplified recurrence relation (5.3) is as follows:

$$\sum_{j=-2}^{2} v_j(i) D_{i+j}^n(x;\alpha,\beta) = 0, \tag{5.4}$$

where

$$\begin{aligned}
v_{-2}(i) &:= (1-x)(i-1)_2(n-i+\alpha)_3,\\
v_{-1}(i) &:= -i(n-i+\alpha)_2\{(i+\beta)(n-3i)\\
&\qquad +[n(n-3i+\alpha-\beta+4)+i(4i-\alpha+3\beta-4)+2(\alpha+2)]x\},\\
v_0(i) &:= (i+\beta)(n-i+\alpha)[z(i)x+(i+1)(i+\beta+1)(3i-2n)],\\
v_1(i) &:= (i-n)(i+\beta)_2\{(i+2)(i+\beta+2)\\
&\qquad -[n(2n-5i+2\alpha)+i(4i-3\alpha+\beta+4)+2(\beta+2)]x\},\\
v_2(i) &:= x(i+\beta)(i+\beta+1)_2(n-i-1)_2,
\end{aligned}$$

and $z(i) := -6i^3+3(3n+\alpha-\beta)i^2-[n(5n-6\beta)+(4n+3)\sigma+3]i+n[(n+1)(n+\alpha+1)+2\beta+2]$.

Each of the coefficients v_k can be computed in $O(1)$ time. Numerical experiments concerning the stability and efficiency of the fourth-order recurrence relation can be found in [18]. Note that the method of evaluating dual Bernstein polynomials based on the recurrence relation (5.4) gives good results for low n ($n \approx 20, 30$). Since then, however, a new recurrence relations of lower order has been found and are presented in Section 5.2 and [96]. The relation has been thoroughly tested in [96] and in Section 5.4 with improved performance and good numerical stability compared to the results for the fourth-order recurrence relation. Additionally, the method of evaluating $D_i^n(x;\alpha,\beta)$ based on the recurrence relation given in Section 5.2 gives very good results even for high n ($n \approx 3000, 5000$).

5.2 Non-homogeneous first-order relation

Using the properties of dual Bernstein polynomials and the relation (1.10), one can find a simple first-order non-homogeneous recurrence relation for dual Bernstein polynomials.

Theorem 5.3. *For $i = 0, 1, \ldots, n$, the following relation holds:*

$$(x-1)(i+1)D_i^n(x;\alpha,\beta) + x(n-i)D_{i+1}^n(x;\alpha,\beta) = \frac{-C_{n,i+1}^{(\alpha,\beta)}}{2n+\sigma+2} T_{ni}^{(\alpha,\beta)}(x), \tag{5.5}$$

where the notation used is that of (1.45)*, and*

$$T_{ni}^{(\alpha,\beta)}(x) := (n-i)(n+\alpha+1)xR_n^{(\alpha,\beta+1)}(x) + (i+1)(n+\beta+1)(1-x)R_n^{(\alpha+1,\beta)}(x). \tag{5.6}$$

Proof. In the sequel, the following identity will be useful:

$$\frac{2n+\sigma+1}{n+1} T_{ni}^{(\alpha,\beta)}(x) = (n+\alpha+1)(n+\beta+1)R_n^{(\alpha,\beta)}(x) + \Big((n-i)(n+\alpha+1) - (i+1)(n+\beta+1)\Big)R_{n+1}^{(\alpha,\beta)}(x). \tag{5.7}$$

It can be verified using (1.17) and (1.18).

Let us use induction on n. First, observe that for any $i = n$, the relation (5.5) immediately follows from (1.40). So, in particular, it also holds for $n = 0$.

Now, suppose that (5.5) is true for some natural number n and all $0 \le i \le n$. One has to prove that

$$(x-1)(i+1)D_i^{n+1}(x;\alpha,\beta) + x(n-i+1)D_{i+1}^{n+1}(x;\alpha,\beta) + \frac{C_{n+1,i+1}^{(\alpha,\beta)}}{2n+\sigma+4} T_{n+1,i}^{(\alpha,\beta)}(x) \equiv 0,$$

where $0 \le i \le n+1$. It is already known that it holds for $i = n+1$. Assume that $0 \le i \le n$. Applying twice Eq. (1.44) to the left-hand side, using Eq. (5.7) and doing simple algebra, one can obtain its equivalent form

$$\begin{aligned}
&\frac{n-i+1}{n+1}\Big[(x-1)(i+1)D_i^n(x;\alpha,\beta) + x(n-i)D_{i+1}^n(x;\alpha,\beta)\Big] \\
&\qquad + \frac{i+1}{n+1}\Big[(x-1)iD_{i-1}^n(x;\alpha,\beta) + x(n-i+1)D_i^n(x;\alpha,\beta)\Big] \\
&+ \left((x-1)(i+1)C_{ni}^{(\alpha,\beta)} + x(n-i+1)C_{n,i+1}^{(\alpha,\beta)} + C_{n+1,i+1}^{(\alpha,\beta)} \frac{(n+\alpha+2)(n+\beta+2)}{(n+2)^{-1}(2n+\sigma+3)_2}\right) R_{n+1}^{(\alpha,\beta)}(x) \\
&\qquad + \frac{C_{n+1,i+1}^{(\alpha,\beta)}(n+2)}{(2n+\sigma+3)_2}\Big((n-i+1)(n+\alpha+2) - (i+1)(n+\beta+2)\Big)R_{n+2}^{(\alpha,\beta)}(x).
\end{aligned}$$

Applying twice the induction assumption to terms in square brackets and after some algebra, gives

$$G_{ni}^{(\alpha,\beta)}\left(\xi_0(n)R_n^{(\alpha,\beta)}(x) + \xi_1(n)R_{n+1}^{(\alpha,\beta)}(x) + \xi_2(n)R_{n+2}^{(\alpha,\beta)}(x)\right), \tag{5.8}$$

where the notation used is that of (1.11)–(1.13), and

$$G_{ni}^{(\alpha,\beta)} := -C_{ni}^{(\alpha,\beta)} \frac{(n-i+1)(n-i+\alpha+1) - (i+1)(i+\beta+1)}{2(n+1)(2n+\sigma+2)_2(n-i+\alpha+1)}.$$

Indeed, it follows from (1.10) that (5.8) is equal to zero. At the end, note the special case: in (5.8), if $i = n - i$ and $\alpha = \beta$, both the expression in brackets and $G_{ni}^{(\alpha,\beta)}$ are equal to zero. □

Note that shifted Jacobi polynomials appearing in Theorem 5.3 (cf. (5.6)) do not depend on i. Now, solving this first-order non-homogeneous recurrence relation and using (1.39) results — after some algebra — in a more explicit formula for dual Bernstein polynomials.

Corollary 5.4. *For $i = 0, 1, \ldots, n$, we have*

$$D_i^n(x;\alpha,\beta) = \binom{n}{i}^{-1} \frac{(-1)^{n-i}(\sigma+1)_n}{K(\alpha+1)_n(\beta+1)_n} \left(\frac{x-1}{x}\right)^i \times \left[R_n^{(\alpha,\beta+1)}(x) S_{ni}^{(\alpha+1,\beta)}\left(\frac{x}{x-1}\right) - R_n^{(\alpha+1,\beta)}(x) S_{n,i-1}^{(\alpha,\beta+1)}\left(\frac{x}{x-1}\right)\right],$$

where

$$S_{mk}^{(a,b)}(z) := (b+1)_m \sum_{j=0}^{k} \frac{(-m)_j(-m-a)_j}{j!(b+1)_j} z^j \qquad (0 \le k \le m)$$

(cf. (1.14)*).*

5.3 Applications of relations for dual Bernstein polynomials

Let us consider the following problem.

Problem 5.5. *Let us fix numbers: $n \in \mathbb{N}$, $x \in [0,1]$ and $\alpha, \beta > -1$. Consider the problem of computing the values*

$$D_i^n(x;\alpha,\beta)$$

for all $i = 0, 1, \ldots, n$.

An efficient solution of this problem gives us, e.g., the fast method of evaluating the polynomial

$$d(x) := \sum_{i=0}^{n} d_i D_i^n(x;\alpha,\beta), \tag{5.9}$$

where the coefficients $d_0, d_1, \ldots, d_n$ are given. Notice that such representation plays a crucial role in, for example, the algorithm for merging of Bézier curves which has been recently proposed in [99].

On the other hand, in many applications, such as least-square approximation in Bézier form (cf. [61, 63]) or numerical solving of boundary value problems (cf., e.g., report [45]) or fractional partial differential equations (see [50, 51] and papers cited therein), it is necessary to compute the collection of integrals of the form

$$I_k := \int_0^1 (1-x)^\alpha x^\beta f(x) D_k^n(x;\alpha,\beta)\,\mathrm{d}x$$

for all $k = 0, 1, \ldots, n$ and a given function f. The motivation for computing I_k is that the polynomial

$$p_n^*(x) := \sum_{k=0}^{n} I_k B_k^n(x)$$

minimizes the value of the least-square error

$$\int_0^1 (1-x)^\alpha x^\beta (f(x)-p_n(x))^2 \mathrm{d}x \qquad (p_n \in \Pi_n)$$

(cf. §1.7.5). The numerical approximations of the integrals $I_0, I_1, \ldots, I_n$ involving the dual Bernstein polynomials can be computed, for example, by quadrature rules (see, e.g., [22, §5]). It also requires the fast evaluation of polynomials $D_0^n(x;\alpha,\beta), D_1^n(x;\alpha,\beta), \ldots, D_n^n(x;\alpha,\beta)$ in many *nodes*.

The solutions of Problem 5.5 which use the representations (1.36), (1.38) or (1.43) of dual Bernstein polynomials, or the recurrence relation (1.44) satisfied by these polynomials, have too high computational complexity (notice that one additionally has to compute shifted Jacobi and/or Hahn polynomials, cf. (1.14) and (1.20)).

It is more efficient to use the recurrence relation (5.4) which is not explicitly related to shifted Jacobi and Hahn polynomials. This relation allows to solve the problem with the computational complexity $O(n)$. For details, see [93, §7 and §10.2]. Experiments have shown that the new method is much faster and gives good numerical results for low n ($n \approx 20, 30$). See [18, §6]. Moreover, one can use the relation (5.5) to achieve the same $O(n)$ computational complexity but with a relation of lower order, which can lead to greater numerical stability. As it will be shown in Section 5.4, that is indeed the case. Notice that previously known methods have $O(n^2)$ or even $O(n^3)$ computational complexity.

The Horner's rule (see, e.g., [22, Eq. (1.2.2)]) for evaluating the nth degree polynomial given in the power basis also has the computational complexity $O(n)$. Taking into account that the dual Bernstein basis is much more complicated than the power basis, the algorithms based on the recurrence relation for evaluating $D_i^n(x;\alpha,\beta)$ or a polynomial given in the form (5.9) are interesting.

5.4 Algorithms for evaluating dual Bernstein polynomials

Let us come back to Problem 5.5 of computing all $n+1$ dual Bernstein polynomials of degree n for fixed $n \in \mathbb{N}$, $\alpha, \beta > -1$ and $x \in [0,1]$. Recall that these polynomials are dual to Bernstein basis polynomials in the interval $[0,1]$ (see (1.6) and Definition 1.54). So, in the context of applications of polynomials $D_i^n(x;\alpha,\beta)$ presented in Section 5.3, the issue of their evaluation for $0 \le x \le 1$ is the most important.

If $x \in \{0,1\}$ then the value of the dual Bernstein polynomial can be easily obtained (cf. (1.41) or (1.42)). Now, suppose that $x \in (0,1)$. In this section algorithms for evaluating polynomials $D_i^n(x;\alpha,\beta)$ $(0 \le i \le n)$ using the first-order non-homogeneous recurrence relation (cf. Theorem 5.3) are proposed.

To obtain accurate methods, it is necessary to be mindful of numerical difficulties arising when recursive computations are performed. See [93]. This is the reason that, in the sequel, two ways of using relation (5.5), i.e., with a *forward* and a *backward* direction of computations, are considered.

For a fixed $0 \le i \le n$, let us define a *forward computation* of $D_i^n(x;\alpha,\beta)$ as a computation that, starting from $D_0^n(x;\alpha,\beta)$, computes $D_i^n(x;\alpha,\beta)$ using Theorem 5.3. Analogously, a *backward computation* of $D_i^n(x;\alpha,\beta)$ starts with $D_n^n(x;\alpha,\beta)$ and uses Theorem 5.3 as well. As mentioned before, some numerical difficulties may arise when performing these computations — especially for sufficiently large n and i. One can mitigate this issue by performing, for certain parameter $J \in \mathbb{N}$ $(0 \le J \le n)$, a forward computation of $D_i^n(x;\alpha,\beta)$ for

$i = 0, 1, \ldots, J$ and a backward computation of $D_i^n(x; \alpha, \beta)$ for $i = J+1, J+2, \ldots, n$. Note that, using Eq. (1.37), a backward computation can be expressed as a forward computation with changed parameters.

In order to determine the value of J, one can use the function

$$J \equiv J(n, x) = \text{round}\,(n \cdot p(x))\,, \tag{5.10}$$

where p is a cubic polynomial in x which satisfies the following interpolation conditions:

x	0.01	0.3	0.7	0.99
$p(x)$	0.1	0.4	0.6	0.9

and round(z) denotes the nearest integer to the real number z. It can be checked that

$$\begin{aligned} p(x) \quad = \quad & 1.58084223194525186\ldots \cdot x^3 - 2.37126334791787779\ldots \cdot x^2 \\ & +1.62239798468112882\ldots \cdot x + 0.08401156564574855\ldots. \end{aligned}$$

Such choice of J has been established experimentally and is used in all algorithms, as well as numerical tests (see §5.4.2).

5.4.1 Algorithms

For $n \in \mathbb{N}$ and $\alpha, \beta > -1$, an implementation of a forward computation of $D_i^n(x; \alpha, \beta)$ for $i = 0, 1, \ldots, j$ and a fixed $0 \le j \le n$ at one point $x \in (0, 1)$ is presented in Algorithm 5.1.

Algorithm 5.1 Computation of $j+1$ first dual Bernstein polynomials of degree n at point $x \in (0, 1)$

```
1:  procedure DualBer(n, α, β, x, j, K)
2:      α1 ← α + 1,  β1 ← β + 1
3:      n1 ← n + α1,  x1x ← (x − 1)/x
4:      C ← (−1)^{n+1} · K/n1 · ∏_{j=0}^{n−1} (1 + β1/(j + α1))
5:      R1 ← n1 · R_n^{(α,β1)}(x)
6:      R2 ← x1x · (n + β1) · R_n^{(α1,β)}(x)
7:      D[0] ← −C · R1
8:      for i ← 1, j do
9:          p ← i − n − 1
10:         q ← i/p
11:         C ← C · (p − α1)/(i + β)
12:         D[i] ← q · x1x · D[i − 1] − C · (R1 + q · R2)
13:     end for
14:     return D
15: end procedure
```

For fixed $n \in \mathbb{N}$ and $\alpha, \beta > -1$, Algorithm 5.2 computes the values of all $n+1$ dual Bernstein polynomials of degree n at one point $x \in (0, 1)$. It computes the value J (cf. (5.10)) and then performs two forward computations, utilizing Algorithm 5.1. This algorithm returns an array $D \equiv D[0..n]$, where

$$D[i] = D_i^n(x; \alpha, \beta) \qquad (0 \le i \le n).$$

Remark 5.6. *Shifted Jacobi polynomials* $R_n^{(\alpha,\beta+1)}$ *and* $R_n^{(\alpha+1,\beta)}$ *(cf. lines 5, 6 in Algorithm 5.1) can be evaluated using the recurrence relation* (1.10) *(cf. Remark 1.28) or even the explicit formula* (1.14)*. Thus, the computational complexity of Algorithm 5.2 is* $O(n)$*.*

Algorithm 5.2 Computation of all $n+1$ dual Bernstein polynomials of degree n at point $x \in (0,1)$

1: **procedure** AllDualBer(n, α, β, x)
2: $\alpha 1 \leftarrow \alpha + 1,\ \beta 1 \leftarrow \beta + 1$
3: $K \leftarrow \Gamma(\alpha 1 + \beta 1) / (\Gamma(\alpha 1) \cdot \Gamma(\beta 1))$
4: $J \leftarrow J(n, x)$
5: $D[0..J] \leftarrow$ DualBer($n, \alpha, \beta, x, J, K$)
6: $D[J+1..n] \leftarrow$ ReverseArray(DualBer($n, \beta, \alpha, 1-x, n-J-1, K$))
7: **return** D
8: **end procedure**

Note that the quantities q and C in Algorithm 5.1, as well as the quantity K in Algorithm 5.2, are independent of x. They can be, therefore, computed once for given $n \in \mathbb{N}$, $\alpha, \beta > -1$ and used across multiple instances of Problem 5.5 for different values of $x \in (0,1)$. This approach is realized in Algorithms 5.3 and 5.4. Note that they require $O(n)$ additional memory to store C and q. The execution of Algorithm 5.4 returns a two-dimensional array

Algorithm 5.3 Computation of $j+1$ first dual Bernstein polynomials of degree n at point $x \in (0,1)$ — with preprocessing

1: **procedure** DualBer-2($n, \alpha, \beta, x, j, q, C$)
2: $\alpha 1 \leftarrow \alpha + 1,\ \beta 1 \leftarrow \beta + 1$
3: $n1 \leftarrow n + \alpha 1,\ x1x \leftarrow (x-1)/x$
4: $R1 \leftarrow n1 \cdot R_n^{(\alpha,\beta 1)}(x)$
5: $R2 \leftarrow x1x \cdot (n+\beta 1) \cdot R_n^{(\alpha 1,\beta)}(x)$
6: $D[0] \leftarrow C[0] \cdot R1/n1$
7: **for** $i \leftarrow 1, j$ **do**
8: $D[i] \leftarrow q[i-1] \cdot x1x \cdot D[i-1] - C[i] \cdot (R1 + q[i-1] \cdot R2)$
9: **end for**
10: **return** D
11: **end procedure**

$D \equiv D[0..M, 0..n]$, where

$$D[m,i] = D_i^n(x_m; \alpha, \beta) \qquad (0 \le m \le M;\ 0 \le i \le n).$$

The computational complexity of this algorithm is $O(nM)$ (cf. Remark 5.6).

5.4.2 Numerical experiments

The presented algorithms have been tested for numerical stability. The computations have been performed in the computer algebra system Maple™14—using single (`Digits:=8`), double (`Digits:=18`) and quadruple (`Digits:=32`) precision — on a computer with `Intel(R)`

Algorithm 5.4 Computation of all $n+1$ dual Bernstein polynomials of degree n at multiple points $x_0, x_1, \ldots, x_M \in (0,1)$

procedure AllDualBer-2($n, \alpha, \beta, [x_0, x_1, \ldots, x_M]$)
  $\alpha 1 \leftarrow \alpha + 1,\ \beta 1 \leftarrow \beta + 1$
  $K \leftarrow \Gamma(\alpha 1 + \beta 1)/\left(\Gamma(\alpha 1) \cdot \Gamma(\beta 1)\right)$
  $q[0] \leftarrow -1/n$
  $C[0] \leftarrow (-1)^n \cdot K \cdot \prod_{j=0}^{n-1}(1 + \beta 1/(j + \alpha 1))$
  $C[1] \leftarrow C[0]/\beta 1$
  for $i \leftarrow 1, n-1$ **do**
    $p \leftarrow i - n$
    $q[i] \leftarrow (i+1)/p$
    $C[i+1] \leftarrow C[i] \cdot (p - \alpha 1)/(i + \beta 1)$
  end for
  for $m \leftarrow 0, M$ **do**
    $J \leftarrow J(n, x_m)$
    $D[m, 0..J] \leftarrow$ DualBer-2($n, \alpha, \beta, x_m, J, q, C$)
    $D[m, J+1..n] \leftarrow$ ReverseArray(DualBer-2($n, \beta, \alpha, 1 - x_m, n - J - 1, q, C$))
  end for
  return D
end procedure

`Core(TM) i5-2540M CPU @ 2.60GHz` processor and `4 GB` of `RAM`. The code is available at `https://bit.ly/fch-phd-ch5`.

The *accuracy* of approximation $\widetilde{v}$ of a nonzero number v is measured by computing the quantity

$$\texttt{acc}(\widetilde{v}, v) := -\log_{10}\left|1 - \frac{\widetilde{v}}{v}\right|. \tag{5.11}$$

Hence, $\texttt{acc}(\widetilde{v}, v)$ is the number of exact significant decimal digits (`acc` in short) in the approximation $\widetilde{v}$ of the number v.

For fixed $n \in \mathbb{N}$ and $\alpha, \beta > -1$, the experiments involved computing values of all $n+1$ dual Bernstein polynomials of degree n for $x \in \{0.01, 0.02, \ldots, 0.99\}$ using Algorithm 5.4, where `Maple™14` `GAMMA` and `JacobiP` procedures have been used to compute values of Γ function and shifted Jacobi polynomials, respectively. For each of $(n+1) \cdot 99$ obtained values, the number of exact significant decimal digits has been computed (cf. (5.11)), where results computed by the same algorithm but in a 512-digit arithmetic (`Digits:=512`) have been assumed to be accurate while comparing to these done for `Digits:=8,18,32`.

The experiments have been performed for dual Bernstein polynomials of degrees

$$n \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$$

and three α, β choices — *Legendre's* ($\alpha = \beta = 0$), *Chebyshev's* ($\alpha = \beta = -0.5$), and a *non-standard* choice ($\alpha = -0.33$, $\beta = 5.6$). A mean (Table 5.1), first percentile (Table 5.2) and minimal (Table 5.3) number of exact significant decimal digits have been computed.

The numerical results show that the proposed method for evaluating dual Bernstein polynomials works very well even for large degrees. Note that results given in Tables 5.1 and 5.2 are almost the same. It indicates that at least 99% of obtained values have greater or similar

		`Digits:=8`	`Digits:=18`	`Digits:=32`
$n = 10$	$\alpha = \beta = 0$	7.64	17.67	31.80
	$\alpha = \beta = -0.5$	6.97	17.03	31.04
	$\alpha = -0.33, \beta = 5.6$	7.14	17.39	31.27
$n = 20$	$\alpha = \beta = 0$	7.24	17.15	31.14
	$\alpha = \beta = -0.5$	6.72	16.82	30.95
	$\alpha = -0.33, \beta = 5.6$	6.65	17.47	31.17
$n = 50$	$\alpha = \beta = 0$	6.77	17.58	30.46
	$\alpha = \beta = -0.5$	6.58	17.47	30.55
	$\alpha = -0.33, \beta = 5.6$	7.00	17.43	30.94
$n = 100$	$\alpha = \beta = 0$	6.98	16.80	30.32
	$\alpha = \beta = -0.5$	6.46	17.06	31.00
	$\alpha = -0.33, \beta = 5.6$	6.79	17.30	31.16
$n = 200$	$\alpha = \beta = 0$	7.28	16.56	31.18
	$\alpha = \beta = -0.5$	6.41	16.12	30.65
	$\alpha = -0.33, \beta = 5.6$	6.21	17.02	31.00
$n = 500$	$\alpha = \beta = 0$	6.65	17.01	30.95
	$\alpha = \beta = -0.5$	6.08	16.36	30.70
	$\alpha = -0.33, \beta = 5.6$	6.13	16.80	30.91
$n = 1000$	$\alpha = \beta = 0$	6.51	16.31	30.23
	$\alpha = \beta = -0.5$	6.23	16.56	29.99
	$\alpha = -0.33, \beta = 5.6$	5.85	15.73	29.73
$n = 2000$	$\alpha = \beta = 0$	6.09	16.88	29.64
	$\alpha = \beta = -0.5$	6.87	16.43	29.56
	$\alpha = -0.33, \beta = 5.6$	6.21	15.81	30.43
$n = 5000$	$\alpha = \beta = 0$	5.57	15.36	30.12
	$\alpha = \beta = -0.5$	5.62	15.45	29.57
	$\alpha = -0.33, \beta = 5.6$	5.29	15.42	30.51

Table 5.1: Mean number of `acc` (cf. (5.11)) obtained by using Algorithm 5.4 for $x \in \{0.01, 0.02, \ldots, 0.99\}$.

number of exact significant decimal digits than these given in Table 5.1, thus making the presented algorithms useful (for example, in numerical evaluation of integrals (1.70), even for large n). Even though in *pessimistic cases* the algorithms lose a significant amount of precision (especially for `Digits:=8`; see Table 5.3), they happen rarely (compare with Table 5.2) and do not significantly affect the average number of correct decimal digits (cf. Table 5.1).

		Digits:=8	Digits:=18	Digits:=32
$n = 10$	$\alpha = \beta = 0$	6.34	16.36	30.47
	$\alpha = \beta = -0.5$	6.01	16.18	30.41
	$\alpha = -0.33, \beta = 5.6$	6.31	16.26	30.35
$n = 20$	$\alpha = \beta = 0$	6.12	16.23	30.25
	$\alpha = \beta = -0.5$	5.99	16.16	30.32
	$\alpha = -0.33, \beta = 5.6$	6.13	16.30	30.18
$n = 50$	$\alpha = \beta = 0$	6.31	16.44	30.16
	$\alpha = \beta = -0.5$	6.22	16.28	30.26
	$\alpha = -0.33, \beta = 5.6$	6.31	16.31	30.34
$n = 100$	$\alpha = \beta = 0$	6.32	16.38	30.15
	$\alpha = \beta = -0.5$	6.17	16.36	30.36
	$\alpha = -0.33, \beta = 5.6$	6.28	16.27	30.23
$n = 200$	$\alpha = \beta = 0$	6.11	16.13	30.21
	$\alpha = \beta = -0.5$	6.05	15.89	30.11
	$\alpha = -0.33, \beta = 5.6$	5.96	16.17	30.15
$n = 500$	$\alpha = \beta = 0$	6.17	16.18	30.15
	$\alpha = \beta = -0.5$	5.90	16.02	30.17
	$\alpha = -0.33, \beta = 5.6$	5.94	16.03	30.16
$n = 1000$	$\alpha = \beta = 0$	6.05	16.02	29.93
	$\alpha = \beta = -0.5$	5.87	16.11	29.82
	$\alpha = -0.33, \beta = 5.6$	5.74	15.61	29.63
$n = 2000$	$\alpha = \beta = 0$	5.84	16.05	29.54
	$\alpha = \beta = -0.5$	6.02	16.01	29.42
	$\alpha = -0.33, \beta = 5.6$	5.86	15.61	29.96
$n = 5000$	$\alpha = \beta = 0$	5.46	15.30	29.65
	$\alpha = \beta = -0.5$	5.49	15.36	29.46
	$\alpha = -0.33, \beta = 5.6$	5.24	15.35	29.82

Table 5.2: First percentile number of `acc` (cf. (5.11)) obtained by using Algorithm 5.4 for $x \in \{0.01, 0.02, \ldots, 0.99\}$.

		`Digits:=8`	`Digits:=18`	`Digits:=32`
	$\alpha = \beta = 0$	4.09	14.69	28.96
$n = 10$	$\alpha = \beta = -0.5$	4.97	15.36	29.06
	$\alpha = -0.33, \beta = 5.6$	4.93	14.96	29.09
	$\alpha = \beta = 0$	5.33	15.35	29.41
$n = 20$	$\alpha = \beta = -0.5$	4.86	14.87	29.56
	$\alpha = -0.33, \beta = 5.6$	5.39	15.12	28.91
	$\alpha = \beta = 0$	4.83	14.65	28.75
$n = 50$	$\alpha = \beta = -0.5$	4.47	14.32	28.48
	$\alpha = -0.33, \beta = 5.6$	4.65	14.13	29.16
	$\alpha = \beta = 0$	4.47	14.84	28.73
$n = 100$	$\alpha = \beta = -0.5$	4.36	14.37	28.48
	$\alpha = -0.33, \beta = 5.6$	2.98	12.84	27.35
	$\alpha = \beta = 0$	3.41	13.54	27.19
$n = 200$	$\alpha = \beta = -0.5$	3.62	13.42	27.73
	$\alpha = -0.33, \beta = 5.6$	4.17	14.31	28.04
	$\alpha = \beta = 0$	3.15	13.65	27.06
$n = 500$	$\alpha = \beta = -0.5$	2.01	12.28	26.47
	$\alpha = -0.33, \beta = 5.6$	3.37	13.37	27.26
	$\alpha = \beta = 0$	2.92	12.97	27.30
$n = 1000$	$\alpha = \beta = -0.5$	3.21	13.41	27.56
	$\alpha = -0.33, \beta = 5.6$	3.18	12.99	27.44
	$\alpha = \beta = 0$	2.16	12.24	26.03
$n = 2000$	$\alpha = \beta = -0.5$	1.46	11.85	25.40
	$\alpha = -0.33, \beta = 5.6$	2.11	11.93	25.97
	$\alpha = \beta = 0$	0	5.38	18.85
$n = 5000$	$\alpha = \beta = -0.5$	0	4.25	18.41
	$\alpha = -0.33, \beta = 5.6$	0	5.64	19.33

Table 5.3: Minimal number of `acc` (cf. (5.11)) obtained by using Algorithm 5.4 for $x \in \{0.01, 0.02, \ldots, 0.99\}$.

5.5 Homogeneous relations of degree two and three

Using Theorem 5.3, one can obtain homogeneous recurrence relations of order 2 and 3 for dual Bernstein polynomials by eliminating the non-homogeneity.

Corollary 5.7. *Dual Bernstein polynomials satisfy the second-order recurrence relation of the form*

$$u_0(i)D_i^n(x;\alpha,\beta) + u_1(i)D_{i+1}^n(x;\alpha,\beta) + u_2(i)D_{i+2}^n(x;\alpha,\beta) = 0 \qquad (0 \le i \le n-2),$$

where

$$\begin{aligned}
u_0(i) &:= (x-1)(i+1)(n-i+\alpha)T_{n,i+1}^{(\alpha,\beta)}(x),\\
u_1(i) &:= x(n-i)(n-i+\alpha)T_{n,i+1}^{(\alpha,\beta)}(x) + (x-1)(i+2)(i+\beta+2)T_{ni}^{(\alpha,\beta)}(x),\\
u_2(i) &:= x(n-i-1)(i+\beta+2)T_{ni}^{(\alpha,\beta)}(x),
\end{aligned}$$

and the notation used is that of (5.6).

Proof. First, observe that from Eq. (1.45) it follows that

$$(n-i+\alpha)C_{n,i+1}^{(\alpha,\beta)} + (i+\beta+2)C_{n,i+2}^{(\alpha,\beta)} = 0.$$

Now, let us add the relation (5.5) multiplied by $(n-i+\alpha)T_{n,i+1}^{(\alpha,\beta)}(x)$ to the same relation for $i := i+1$ and multiplied by $(i+\beta+2)T_{ni}^{(\alpha,\beta)}(x)$. As a result, one gets

$$\begin{aligned}
u_0(i)D_i^n(x;\alpha,\beta) + u_1(i)D_{i+1}^n(x;\alpha,\beta) + u_2(i)D_{i+2}^n(x;\alpha,\beta) &= \\
= \frac{-T_{ni}^{(\alpha,\beta)}(x)T_{n,i+1}^{(\alpha,\beta)}(x)}{(2n+\sigma+2)} &\left((n-i+\alpha)C_{n,i+1}^{(\alpha,\beta)} + (i+\beta+2)C_{n,i+2}^{(\alpha,\beta)}\right) = 0.
\end{aligned}$$

□

The coefficients u_j $(j = 0, 1, 2)$ are not simple because they depend on two shifted Jacobi polynomials of degree n in x. However, these polynomials are independent of i and can be efficiently computed with the recurrence relation (1.10) (cf. Remark 1.28) and re-used for all remaining i. Thus, Corollary 5.7 may be useful in numerical practice.

One can get a recurrence relation with simple coefficients by using the same method which was used in the proof of Theorem 5.2.

Corollary 5.8. *For $0 \le i \le n-3$, the polynomials $D_i^n(x;\alpha,\beta)$ satisfy the following third-order recurrence relation:*

$$w_0(i)D_i^n(x;\alpha,\beta) + w_1(i)D_{i+1}^n(x;\alpha,\beta) + w_2(i)D_{i+2}^n(x;\alpha,\beta) + w_3(i)D_{i+3}^n(x;\alpha,\beta) = 0. \quad (5.12)$$

Here

$$\begin{aligned}
w_0(i) &:= (x-1)(i+1)(n-i+\alpha-1)_2,\\
w_1(i) &:= (n-i+\alpha-1)[x(n-i)(n-i+\alpha) + 2(x-1)(i+2)(i+\beta+2)],\\
w_2(i) &:= (i+\beta+2)[(x-1)(i+3)(i+\beta+3) + 2x(n-i-1)(n-i+\alpha-1)],\\
w_3(i) &:= x(n-i-2)(i+\beta+2)_2.
\end{aligned}$$

Proof. From Theorem 5.3, it follows that

$$\frac{2n+\sigma+2}{-C_{n,i+1}^{(\alpha,\beta)}}\Big((x-1)(i+1)D_i^n(x;\alpha,\beta)+x(n-i)D_{i+1}^n(x;\alpha,\beta)\Big)=T_{ni}^{(\alpha,\beta)}(x). \tag{5.13}$$

Now, $T_{ni}^{(\alpha,\beta)}(x)$ is a polynomial of degree 1 in i. Therefore,

$$(\mathcal{E}-\mathcal{I})^2 T_{ni}^{(\alpha,\beta)}(x)=0,$$

where $\mathcal{E}$ acts on the variable i (cf. (5.1)). Applying the operator $(\mathcal{E}-\mathcal{I})^2$ to both sides of (5.13) gives

$$(\mathcal{E}-\mathcal{I})^2\frac{2n+\sigma+2}{-C_{n,i+1}^{(\alpha,\beta)}}\Big((x-1)(i+1)\mathcal{I}+x(n-i)\mathcal{E}\Big)D_i^n(x;\alpha,\beta)=0.$$

The left-hand side can be expanded as follows:

$$\begin{aligned}(x-1)\frac{i+1}{-C_{n,i+1}^{(\alpha,\beta)}}D_i^n(x;\alpha,\beta)+\left(x\frac{n-i}{-C_{n,i+1}^{(\alpha,\beta)}}-2(x-1)\frac{i+2}{-C_{n,i+2}^{(\alpha,\beta)}}\right)D_{i+1}^n(x;\alpha,\beta)\\ +\left((x-1)\frac{i+3}{-C_{n,i+3}^{(\alpha,\beta)}}-2x\frac{n-i-1}{-C_{n,i+2}^{(\alpha,\beta)}}\right)D_{i+2}^n(x;\alpha,\beta)+x\frac{n-i-2}{-C_{n,i+3}^{(\alpha,\beta)}}D_{i+3}^n(x;\alpha,\beta)=0.\end{aligned}$$

Observe that from Eq. (1.45), it follows that

$$\frac{C_{n,i+3}^{(\alpha,\beta)}}{C_{n,i+2}^{(\alpha,\beta)}}=\frac{-(n-i+\alpha-1)}{i+\beta+3}$$

and, in turn,

$$\frac{C_{n,i+3}^{(\alpha,\beta)}}{C_{n,i+1}^{(\alpha,\beta)}}=\frac{(n-i+\alpha-1)_2}{(i+\beta+2)_2}.$$

After multiplying each side by $-C_{n,i+3}^{(\alpha,\beta)}(i+\beta+2)_2$, one gets

$$w_0(i)D_i^n(x;\alpha,\beta)+w_1(i)D_{i+1}^n(x;\alpha,\beta)+w_2(i)D_{i+2}^n(x;\alpha,\beta)+w_3(i)D_{i+3}^n(x;\alpha,\beta)=0.$$

□

Compared to relation (5.4), Eq. (5.12) is simpler: i) it has lower order (third instead of fourth); ii) its coefficients w_j $(0\le j\le 3)$ are cubic (instead of quintic) polynomials in i.

Remark 5.9. *Using the new recurrence relations given in this Section, one can solve Problem 5.5 with $O(n)$ computational complexity (cf. Remark 1.28) — the same as in the case of the first-order non-homogeneous relation, presented in more detail in Section 5.4.*

Chapter 6

Fast parallel k, l-constrained Bézier curve degree reduction

The new recurrence relations given in Chapter 5 can find an application, e.g., in reducing the degree of a Bézier curve (cf. §1.7.4). In this chapter, a k, l-constrained version of this problem will be considered, i.e., for a Bézier curve P_n of degree n, one seeks a curve P_m of degree m $(m < n)$ which, for $k, l \in \mathbb{N}$ such that $k + l < m$, satisfies the conditions

$$\mathsf{P}_n^{(z)}(0) = \mathsf{P}_m^{(z)}(0) \qquad (z = 0, 1, \ldots, k-1),$$

$$\mathsf{P}_n^{(z)}(1) = \mathsf{P}_m^{(z)}(1) \qquad (z = 0, 1, \ldots, l-1),$$

and the curve P_m is optimal in the sense of the least-square approximation related to the shifted Jacobi scalar product $\langle \cdot, \cdot \rangle_{\alpha,\beta}$ (cf. (1.6)). It means that the value of the integral

$$\int_0^1 (1-x)^\alpha x^\beta \|\mathsf{P}_n(t) - \mathsf{P}_m(t)\|_2^2 dt \qquad (\alpha, \beta > -1)$$

(cf. (1.59)) is minimized. In the sequel, when no parameters of a scalar product are given, it is assumed that the parameters are α, β, i.e.,

$$\langle \cdot, \cdot \rangle \equiv \langle \cdot, \cdot \rangle_{\alpha,\beta}.$$

As shown in [97], finding the curve P_m can be done using the dual projection (cf. Theorem 1.38), which involves computing $O(nm)$ scalar products of Bernstein polynomials and their dual counterparts of different degrees. In §1.7.4, these products are scaled by a certain factor and arranged into the Ψ table (see Theorem 1.77 and Figure 1.10), where

$$\Psi = [\Psi_{ij}]$$

with

$$\Psi_{ij} := d_{i-k}^{m-k-l}(j-k; \beta+2k, \alpha+2l, n-k-l)$$

(cf. (1.61)) is defined in terms of dual discrete Bernstein polynomials (see §1.5.4). The proposed method of evaluating the elements of the Ψ table is using the recurrence relation given in Theorem 1.81. Due to the shape of the recurrence relation, the options to perform the computations in parallel are limited.

In Section 6.1, new recurrence relations are given, based on Theorems 1.81 and 5.3. More precisely, a first-order non-homogeneous relation connecting Ψ_{ij} with $\Psi_{i+1,j+1}$ is found. This, along with the relation given in Theorem 1.81, allows to derive two additional non-homogeneous second-order recurrence relations, connecting the quantities $\Psi_{i,j-1}$, Ψ_{ij}, $\Psi_{i,j+1}$ and $\Psi_{i-1,j}$, Ψ_{ij}, $\Psi_{i+1,j}$, respectively.

New recurrence schemes which use the new relations are given in Section 6.2. Aside from computing the necessary initial values, the new recurrence relations allow to compute each row, column or diagonal (depending on the relation used) of the Ψ table independently. Even though the asymptotical number of arithmetic operations required to compute the Ψ table is unchanged, the new schemes are more convenient when it comes to parallel computations.

6.1 New recurrence relations for Ψ_{ij}

6.1.1 Diagonal recurrence relation for Ψ_{ij}

Theorem 6.1. *Let $m, n, k, l \in \mathbb{N}$ be such that $m < n$ and $k + l \leq m$. The quantities Ψ_{ij}, defined by Eq. (1.61), ($i = k, k+1, \ldots, m-l-1$, $j = k, k+1, \ldots, n-l-1$) satisfy the following first-order non-homogeneous recurrence relation:*

$$(m-l-i)(\beta+k+1+j)\Psi_{i+1,j+1} - (i-k+1)(\alpha+l+n-j)\Psi_{ij} = L\rho_i \cdot \Big(U \cdot Q_{m-k-l} + V \cdot \eta_i \cdot Q_{m-k-l+1}\Big), \quad (6.1)$$

where

$$Q_h := Q_h(j-k; \beta+2k, \alpha+2l, n-k-l-1) \quad (6.2)$$

are the Hahn polynomials (see Definition 1.33),

$$\begin{cases} L \equiv L^{(\alpha,\beta)}_{mnkl} := \dfrac{(n-k-l)!(k+l+1-n)_{m-k-l}(\beta+2k+1)_{m-k-l+1}}{(m-k-l)!(2m+\sigma+1)(m+k+l+\sigma+1)_{n-k-l-1}}, \\ U \equiv U^{\alpha}_{mkl} := (m-k-l+1)(m+\alpha+l-k+1), \\ V \equiv V^{(\alpha,\beta)}_{mn} := \dfrac{m+1-n}{m+n+\sigma}, \\ \rho_i \equiv \rho_i^{(m,n,k,l,\alpha,\beta)} := \dfrac{(-1)^{m-l-i+1}}{(\alpha+2l+1)_{m-l-i}(\beta+2k+1)_{i-k+1}}, \\ \eta_i \equiv \eta_i^{(m,n,k,l,\alpha,\beta)} := (m-l-i)(m-k+\alpha+l+1) - (i-k+1)(m-l+\beta+k+1), \end{cases} \quad (6.3)$$

and $\sigma := \alpha + \beta + 1$ (cf. (1.8)).

Proof. Theorem 1.62 shows the following relation between constrained dual Bernstein polynomials and non-constrained dual Bernstein polynomials of a lower degree:

$$D_i^{(m,k,l)}(x; \alpha, \beta) = \frac{\binom{m-k-l}{i-k} x^k (1-x)^l}{\binom{m}{i}} D^{m-k-l}_{i-k}(x; \alpha+2l, \beta+2k).$$

Now, let us take Eq. (5.5) for $n := m-k-l$, $i := i-k$, $\alpha := \alpha+2l$, $\beta := \beta+2k$, i.e.,

$$(x-1)(i-k+1)D^{m-k-l}_{i-k}(x; \alpha+2l, \beta+2k) + x(m-l-i)D^{m-k-l}_{i-k+1}(x; \alpha+2l, \beta+2k) = \frac{-C^{(\alpha+2l,\beta+2k)}_{m-k-l,i-k+1}}{2m+\sigma+2} T^{(\alpha+2l,\beta+2k)}_{m-k-l,i-k}(x)$$

(cf. (5.6) and (1.45)). Substituting dual Bernstein polynomials with constrained dual Bernstein polynomials of higher degree gives

$$\frac{-\binom{m}{i}(i-k+1)}{\binom{m-k-l}{i-k}x^k(1-x)^{l-1}}D_i^{(m,k,l)}(x;\alpha,\beta)+\frac{\binom{m}{i+1}(m-l-i)}{\binom{m-k-l}{i+1-k}x^{k-1}(1-x)^l}D_{i+1}^{(m,k,l)}(x;\alpha,\beta)$$
$$=\frac{-C_{m-k-l,i-k+1}^{(\alpha+2l,\beta+2k)}}{2m+\sigma+2}T_{m-k-l,i-k}^{(\alpha+2l,\beta+2k)}(x).$$

Now, one can apply $\left\langle B_{k+j}^{n+k+l-1},\cdot\right\rangle$ to both sides to get

$$\frac{-\binom{m}{i}(i-k+1)}{\binom{m-k-l}{i-k}}\left\langle B_{k+j}^{n+k+l-1},\frac{D_i^{(m,k,l)}(x;\alpha,\beta)}{x^k(1-x)^{l-1}}\right\rangle$$
$$+\frac{\binom{m}{i+1}(m-l-i)}{\binom{m-k-l}{i+1-k}}\left\langle B_{k+j}^{n+k+l-1},\frac{D_{i+1}^{(m,k,l)}(x;\alpha,\beta)}{x^{k-1}(1-x)^l}\right\rangle$$
$$=\frac{-C_{m-k-l,i-k+1}^{(\alpha+2l,\beta+2k)}}{2m+\sigma+2}\left\langle B_{k+j}^{n+k+l-1},T_{m-k-l,i-k}^{(\alpha+2l,\beta+2k)}\right\rangle.$$

Certainly,

$$\frac{B_{k+j}^{n+k+l-1}(x)}{x^k(1-x)^{l-1}}=\binom{n+k+l-1}{k+j}\binom{n}{j}^{-1}B_j^n(x),$$
$$\frac{B_{k+j}^{n+k+l-1}(x)}{x^{k-1}(1-x)^l}=\binom{n+k+l-1}{k+j}\binom{n}{j+1}^{-1}B_{j+1}^n(x),$$

thus

$$\binom{n+k+l-1}{k+j}\left(\frac{-\binom{m}{i}(i-k+1)}{\binom{m-k-l}{i-k}\binom{n}{j}}\Phi_{ij}+\frac{\binom{m}{i+1}(m-l-i)}{\binom{m-k-l}{i+1-k}\binom{n}{j+1}}\Phi_{i+1,j+1}\right)$$
$$=\frac{-C_{m-k-l,i-k+1}^{(\alpha+2l,\beta+2k)}}{2m+\sigma+2}\left\langle B_{k+j}^{n+k+l-1},T_{m-k-l,i-k}^{(\alpha+2l,\beta+2k)}\right\rangle,$$

where $\Phi_{ij}:=\left\langle B_j^n,D_i^{(m,k,l)}(\cdot;\alpha,\beta)\right\rangle$ (cf. (1.62)). Using Eq. (1.60), one can express Φ_{ij} and $\Phi_{i+1,j+1}$ in terms of Ψ_{ij} and $\Psi_{i+1,j+1}$ to get

$$(m-l-i)(\beta+2k+1+j-k)\Psi_{i+1,j+1}-(i-k+1)(\alpha+l+n-j)\Psi_{ij}$$
$$=\frac{-(n-k-l)!C_{m-k-l,i-k+1}^{(\alpha+2l,\beta+2k)}}{(2m+\sigma+2)(\alpha+2l+1)_{n-l-j-1}(\beta+2k+1)_{j-k}}$$
$$\cdot\binom{n+k+l-1}{k+j}^{-1}\left\langle B_{k+j}^{n+k+l-1},T_{m-k-l,i-k}^{(\alpha+2l,\beta+2k)}\right\rangle. \quad (6.4)$$

The left-hand sides of (6.4) and (6.1) are identical. It remains to check that the right-hand sides of these equations are identical as well.

Recall (cf. (5.7)) that

$$T^{(\alpha+2l,\beta+2k)}_{m-k-l,i-k}(x) \;=\; \frac{m-l+\beta+k+1}{2m+\sigma+1}UR^{(\alpha+2l,\beta+2k)}_{m-k-l}(x) + \frac{m-k-l+1}{2m+\sigma+1}\eta_i R^{(\alpha+2l,\beta+2k)}_{m-k-l+1}(x)$$

(cf. (6.3)). Additionally, from (1.45), it follows that

$$C^{(\alpha+2l,\beta+2k)}_{m-k-l,i-k+1} = \frac{-(2m+\sigma+2)(\sigma+2k+2l+1)_{m-k-l}}{K_{\alpha+2l,\beta+2k}}\rho_i$$

(cf. (6.3) and (1.9)). These observations allow to express the right-hand side of (6.4) as

$$\begin{aligned}\binom{n+k+l-1}{k+j}^{-1} &\frac{(n-k-l)!(\sigma+2k+2l+1)_{m-k-l}}{(2m+\sigma+1)(\alpha+2l+1)_{n-l-j-1}(\beta+2k+1)_{j-k}K_{\alpha+2l,\beta+2k}}\rho_i\\ &\times\Big((m-l+\beta+k+1)U\Big\langle B^{n+k+l-1}_{k+j}, R^{(\alpha+2l,\beta+2k)}_{m-k-l}\Big\rangle\\ &\qquad + (m-k-l+1)\eta_i\Big\langle B^{n+k+l-1}_{k+j}, R^{(\alpha+2l,\beta+2k)}_{m-k-l+1}\Big\rangle\Big). \quad (6.5)\end{aligned}$$

Now, let us focus on the scalar products. Observe that

$$\begin{aligned}&\Big\langle B^{n+k+l-1}_{k+j}, R^{(\alpha+2l,\beta+2k)}_{h}\Big\rangle\\ &\qquad = \binom{n+k+l-1}{k+j}\binom{n-k-l-1}{j-k}^{-1}\Big\langle B^{n-k-l-1}_{j-k}, R^{(\alpha+2l,\beta+2k)}_{h}\Big\rangle_{\alpha+2l,\beta+2k}. \quad (6.6)\end{aligned}$$

Eq. (1.34) gives a shifted Jacobi form of a Bernstein polynomial:

$$\begin{aligned}B^{n-k-l-1}_{j-k}(x) = &\binom{n-k-l-1}{j-k}(\alpha+2l+1)_{n-l-1-j}(\beta+2k+1)_{j-k}\\ &\times\sum_{i=0}^{n-k-l-1}\frac{(2i+\sigma+2k+2l)(-(n-k-l-1))_i}{(\alpha+2l+1)_i(i+\sigma+2k+2l)_{n-k-l}}\cdot Q_i\cdot R^{(\alpha+2l,\beta+2k)}_i(x)\end{aligned}$$

(cf. (6.2)). The polynomials $R^{(\alpha+2l,\beta+2k)}_h$ are orthogonal with respect to $\Big\langle\cdot,\cdot\Big\rangle_{\alpha+2l,\beta+k}$, with

$$\begin{aligned}&\Big\langle R^{(\alpha+2l,\beta+2k)}_h, R^{(\alpha+2l,\beta+2k)}_h\Big\rangle_{\alpha+2l,\beta+2k}\\ &\qquad = K_{\alpha+2l,\beta+2k}\cdot\frac{(\alpha+2l+1)_h(\beta+2k+1)_h(\sigma+2k+2l)}{h!(2h+\sigma+2k+2l)(\sigma+2k+2l)_h} \quad (6.7)\end{aligned}$$

(cf. (1.7)). Combining the relations (6.6), (1.34) and (6.7) gives

$$\begin{aligned}\Big\langle B^{n+k+l-1}_{k+j}, R^{(\alpha+2l,\beta+2k)}_h\Big\rangle = &\binom{n+k+l-1}{k+j}(\alpha+2l+1)_{n-l-1-j}(\beta+2k+1)_{j-k}\\ &\times K_{\alpha+2l,\beta+2k}\frac{(-(n-k-l-1))_h(\beta+2k+1)_h(\sigma+2k+2l)}{(h+\sigma+2k+2l)_{n-k-l}h!(\sigma+2k+2l)_h}Q_h. \quad (6.8)\end{aligned}$$

After applying (6.8) twice to (6.5) and doing some algebra, one gets

$$L\rho_i\Big(UQ_{m-k-l}+\eta_i VQ_{m-k-l+1}\Big)$$

(cf. (6.3)). Thus the right-hand side of (6.4) is identical to the right-hand side of (6.1). □

From Eq. (6.1), one can get a recurrence relation for discrete dual Bernstein polynomials (see §1.5.4). It is a discrete equivalent of Theorem 5.3.

Corollary 6.2. *For $m < n$, $i = 0, \ldots, m-1$, $j = 0, 1, \ldots, n-1$ and $\alpha, \beta > -1$, the discrete dual Bernstein polynomials satisfy a first-order non-homogeneous recurrence relation of the form*

$$\begin{aligned}
&(m-i)(\beta+1+j)d_{i+1}^m(j+1;\beta,\alpha,n) - (i+1)(\alpha+n-j)d_i^m(j;\beta,\alpha,n) \\
&= \frac{(-1)^{m-i+1}n!(1-n)_m(\beta+i+2)_{m-i}}{m!(2m+\sigma+1)(m+\sigma+1)_{n-1}(\alpha+1)_{m-i}} \cdot \Big((m+1)(m+\alpha+1)\cdot Q_m(j;\beta,\alpha,n-1) \\
&\qquad + \frac{m+1-n}{m+n+\sigma}\cdot[(m-i)(m+\alpha+1) - (i+1)(m+\beta+1)]\cdot Q_{m+1}(j;\beta,\alpha,n-1)\Big).
\end{aligned}$$

Proof. The result immediately follows from applying Eq. (1.61) to (6.1). □

6.1.2 Horizontal and vertical recurrence relations for Ψ_{ij}

Let $m, n, k, l \in \mathbb{N}$ be such that $m < n$ and $k + l \le m$. Let L, U, V, ρ_i, η_i be defined as in (6.3). Let $A(r,s)$, $B(r,s)$, $C(r,s)$ be defined by Eq. (1.69).

Theorem 6.3. *For $k+1 \le i \le m-l-1$ and $k+1 \le j \le n-l-1$, the quantities $\Psi_{i,j-1}$, Ψ_{ij}, $\Psi_{i,j+1}$ satisfy a second-order non-homogeneous recurrence relation of the form:*

$$\begin{aligned}
&A(n,i,j)\Psi_{i,j-1} + [C(m,i) - C(n,j)]\Psi_{ij} + B(n,m,i,j)\Psi_{i,j+1} \\
&\qquad = \frac{(m+l-i+\alpha+1)L\rho_{i-1}}{(\alpha+l+n-j)}\cdot\Big(U\cdot Q_{m-k-l}(0) + V\cdot\eta_{i-1}\cdot Q_{m-k-l+1}(0)\Big) \\
&\qquad\qquad - \frac{(k+i+\beta+1)L\rho_i}{(\beta+k+j)}\cdot\Big(U\cdot Q_{m-k-l}(1) + V\cdot\eta_i\cdot Q_{m-k-l+1}(1)\Big),
\end{aligned}$$

where

$$\begin{aligned}
A(n,i,j) &:= A(n,j) + \frac{(k+i+\beta+1)(i-k+1)(\alpha+l+n-j+1)}{(\beta+k+j)}, \\
B(n,m,i,j) &:= B(n,j) + \frac{(m+l-i+\alpha+1)(m-l-i+1)(\beta+k+1+j)}{(\alpha+l+n-j)}, \\
Q_h(\delta) &:= Q_h(j-k-\delta;\beta+2k,\alpha+2l,n-k-l-1) \qquad (\delta\in\{0,1\}).
\end{aligned}$$

Proof. From Eq. (6.1), it follows that

$$\begin{aligned}
\Psi_{i+1,j} = \frac{1}{(m-l-i)(\beta+k+j)}\Big((i-k+1)(\alpha+l+n-j+1)\Psi_{i,j-1} \\
+ L\rho_i\cdot\Big(U\cdot Q_{m-k-l}(1) + V\cdot\eta_i\cdot Q_{m-k-l+1}(1)\Big)\Big)
\end{aligned}$$

and

$$\begin{aligned}
\Psi_{i-1,j} = \frac{1}{(i-k)(\alpha+l+n-j)}\Big((m-l-i+1)(\beta+k+1+j)\Psi_{i,j+1} \\
- L\rho_{i-1}\cdot\Big(U\cdot Q_{m-k-l}(0) + V\cdot\eta_{i-1}\cdot Q_{m-k-l+1}(0)\Big)\Big).
\end{aligned}$$

One can apply these equations to Eq. (1.68) to get

$$\begin{aligned}\Big(A(n,j)+\frac{(k+i+\beta+1)(i-k+1)(\alpha+l+n-j+1)}{(\beta+k+j)}\Big)\Psi_{i,j-1}+\Big[C(m,i)-C(n,j)\Big]\Psi_{ij}\\ +\Big(B(n,j)+\frac{(m+l-i+\alpha+1)(m-l-i+1)(\beta+k+1+j)}{(\alpha+l+n-j)}\Big)\Psi_{i,j+1}\\ =\frac{(m+l-i+\alpha+1)L\rho_{i-1}}{(\alpha+l+n-j)}\cdot\Big(U\cdot Q_{m-k-l}(0)+V\cdot\eta_{i-1}\cdot Q_{m-k-l+1}(0)\Big)\\ -\frac{(k+i+\beta+1)L\rho_i}{(\beta+k+j)}\cdot\Big(U\cdot Q_{m-k-l}(1)+V\cdot\eta_i\cdot Q_{m-k-l+1}(1)\Big).\end{aligned}$$

□

In the Ψ table, $\Psi_{k-1,j}\equiv\Psi_{m-l+1,j}\equiv 0$ (cf. Figure 1.10) for $j=k,k+1,\ldots,n-l$, thus simpler relations exist for the first and the last row. The relation for the first row of the Ψ table, i.e., for $\Psi_{k,j}$ ($j=l,l+1,\ldots,n-l$), is given by (1.67). A similar relation can be proved for the last (i.e., $i:=m-l$) row of the Ψ table.

Theorem 6.4. *For $k<m-l$, the quantities $\Psi_{m-l,j}$ ($j=k,k+1,\ldots,n-l-1$) satisfy a second-order homogeneous recurrence relation of the form*

$$\Psi_{m-l,j+1}=H(j)\Psi_{m-l,j}-S(j)\Psi_{m-l,j-1}\qquad(j=k+1,\ldots,n-l-2),\tag{6.9}$$

where

$$\begin{aligned}\Psi_{m-l,k}&=\frac{(n-k-l)!(k+l+1-n)_{m-k-l}}{(m-k-l)!(m+\sigma+k+l+1)_{n-k-l}},\\ \Psi_{m-l,k+1}&=H(k)\Psi_{m-l,k}.\end{aligned}\tag{6.10}$$

Here

$$\begin{cases}H(j)\equiv H(j;m,n,k,l,\alpha,\beta):=\Big(1+S(j)+\dfrac{(m-k-l)(m+k+l+\sigma+1)}{(j+l+1-n)(j+\beta+k+1)}\Big),\\ S(j)\equiv S(j;m,n,k,l,\alpha,\beta):=\dfrac{(j-k)(j-\alpha-l-1-n)}{(j+l+1-n)(j+\beta+k+1)}.\end{cases}$$

Proof. From Eq. (1.61), it follows that the elements in the last row can be expressed as a particular case of dual discrete Bernstein polynomials:

$$\Psi_{m-l,j}=d^{m-k-l}_{m-l-k}(j-k;\beta+2k,\alpha+2l,n-k-l),$$

which, after applying the symmetry relation (1.46), gives

$$\Psi_{m-l,j}=d^{m-k-l}_{m-l-k}(j-k;\beta+2k,\alpha+2l,n-k-l)=d^{m-k-l}_{0}(n-l-j;\alpha+2l,\beta+2k,n-k-l).$$

The dual discrete Bernstein polynomial on the right-hand side can be expressed as a linear combination of Hahn polynomials with shifted parameters (cf. (1.47)), which results in the following representation of $\Psi_{m-l,j}$:

$$\Psi_{m-l,j}=\frac{(n-k-l)!(k+l+1-n)_{m-k-l}}{(m-k-l)!(m+\sigma+k+l+1)_{n-k-l}}Q_{m-k-l}(j-k;\beta+2k,\alpha+2l+1,n-k-l-1).\tag{6.11}$$

Eq. (1.20) allows to substitute the Hahn polynomial with its hypergeometric form

$$ {}_3F_2\left(\begin{matrix} k+l-m, m+k+l+\sigma+1, k-j \\ \beta+2k+1, k+l+1-n \end{matrix} \middle| 1 \right). $$

For $j := k$ and $j := k+1$, the upper parameter $k-j$ of the hypergeometric function is 0 or -1, respectively. After applying Definition 1.18, the expressions for $\Psi_{m-l,k}$ and $\Psi_{m-l,k+1}$ immediately follow.

To find the recurrence relation to compute the remaining elements of the last row, one can use the difference equation for Hahn polynomials given in Theorem 1.36. Using the substitution

$$ k := m-k-l, \quad x := j-k, \quad \alpha := \beta+2k, \quad \beta := \alpha+2l+1 $$

results in

$$ \begin{aligned} &(j+l+1-n)(j+\beta+k+1)Q_{m-k-l}(j-k+1;\beta+2k,\alpha+2l+1,n-k-l-1) \\ &-\Big[(j+l+1-n)(j+\beta+k+1)+(j-k)(j-\alpha-l-1-n)\Big]Q_{m-k-l}(j-k;\beta+2k,\alpha+2l+1,n-k-l-1) \\ &\quad +(j-k)(j-\alpha-l-1-n)Q_{m-k-l}(j-k-1;\beta+2k,\alpha+2l+1,n-k-l-1) \\ &\quad =(m-k-l)(m+k+l+\sigma+1)Q_{m-k-l}(j-k;\beta+2k,\alpha+2l+1,n-k-l-1). \end{aligned} $$

Using (6.11), one can represent the Hahn polynomials with the quantities $\Psi_{m-l,j-1}$, $\Psi_{m-l,j}$, $\Psi_{m-l,j+1}$, and, after some algebra, get the recurrence relation

$$ \Psi_{m-l,j+1} = H(j)\Psi_{m-l,j} - S(j)\Psi_{m-l,j-1} \qquad (j = k+1,\ldots,n-l-1). $$

□

Note that $\Psi_{m-l,n-l}$ cannot be computed using Theorem 6.4 and it has to be computed using a different relation (e.g., Theorem 6.1). If $k = m-l$ then $\Psi_{m-l,j} \equiv \Psi_{kj}$ and thus it can be computed using Theorem 1.80.

Theorem 6.5. *For $k+1 \le i \le m-l-1$ and $k+1 \le j \le n-l-1$, the quantities $\Psi_{i-1,j}$, Ψ_{ij}, $\Psi_{i+1,j}$ satisfy the following second-order non-homogeneous recurrence relation:*

$$ \begin{aligned} &A(n,m,i,j)\Psi_{i-1,j} + [C(m,i)-C(n,j)]\Psi_{ij} + B(m,i,j)\Psi_{i+1,j} \\ &\qquad = \frac{(k-j)L\rho_i}{(i-k+1)} \cdot \Big(U \cdot Q_{m-k-l}(1) + V \cdot \eta_i \cdot Q_{m-k-l+1}(1)\Big) \\ &\qquad\quad - \frac{(j+l-n)L\rho_{i-1}}{(m-l-i+1)} \cdot \Big(U \cdot Q_{m-k-l}(0) + V \cdot \eta_{i-1} \cdot Q_{m-k-l+1}(0)\Big), \end{aligned} $$

where

$$ \begin{aligned} A(n,m,i,j) &:= -A(m,i) \cdot \left(1 + \frac{(j+l-n)(\alpha+l+n-j)}{(m-l-i+1)(m+l-i+\alpha+1}\right), \\ B(m,i,j) &:= -B(m,i) \cdot \left(1 + \frac{(k-j)(\beta+k+j)}{(k+i+\beta+1)(i-k+1)}\right), \\ Q_h(\delta) &:= Q_h(j-k-\delta;\beta+2k,\alpha+2l,n-k-l-1) \qquad (\delta \in \{0,1\}). \end{aligned} $$

Proof. From Eq. (6.1), it follows that:

$$\Psi_{i,j-1} = \frac{1}{(i-k+1)(\alpha+l+n-j+1)}\Big((m-l-i)(\beta+k+j)\Psi_{i+1,j} - L\rho_i \cdot \Big(U \cdot Q_{m-k-l}(1) + V \cdot \eta_i \cdot Q_{m-k-l+1}(1)\Big)\Big),$$

$$\Psi_{i,j+1} = \frac{1}{(m-l-i+1)(\beta+k+1+j)}\Big((i-k)(\alpha+l+n-j)\Psi_{i-1,j} + L\rho_{i-1} \cdot \Big(U \cdot Q_{m-k-l}(0) + V \cdot \eta_{i-1} \cdot Q_{m-k-l+1}(0)\Big)\Big).$$

Applying these relations to Eq. (1.68) and doing simple algebra concludes the proof. □

The same approach, when taking into account that some elements in Eq. (1.68) are equal to zero, gives the recurrence relations for the kth and $(n-l)$th columns of the Ψ table.

Theorem 6.6. *The elements Ψ_{ik} ($i = k+1, k+2, \ldots, m-l-1$) satisfy the second-order non-homogeneous recurrence relation of the form*

$$B(m,i)\Psi_{i+1,k} - \Big[C(m,i) - C(n,k)\Big]\Psi_{ik} + \Big(A(m,i) + \frac{(n-k-l)(i-k)(\alpha+l+n-k)}{m-l-i+1}\Big)\Psi_{i-1,k} = \frac{k+l-n}{m-l-i+1} L\rho_{i-1} \cdot \Big(U + V \cdot \eta_{i-1}\Big). \quad (6.12)$$

Proof. From Eq. (1.68) and the fact that $\Psi_{i,k-1} \equiv 0$, one gets

$$[C(m,i) - C(n,k)]\Psi_{ik} + B(n,k)\Psi_{i,k+1} - A(m,i)\Psi_{i-1,k} - B(m,i)\Psi_{i+1,k} = 0. \quad (6.13)$$

Adding (6.12) and (6.13) and using the fact that $B(n,k) = (k+l-n)(2k+\beta+1)$ (cf. (1.69)) gives, after some algebra,

$$-(i-k)(\alpha+l+n-k)\Psi_{i-1,k} + (m-l-i+1)(2k+\beta+1)\Psi_{i,k+1} = L\rho_{i-1} \cdot \Big(U + V \cdot \eta_{i-1}\Big). \quad (6.14)$$

It is sufficient to prove that (6.14) holds.

Now, let us take a look at $\Psi_{i,k+1}$. Eq. (6.1), combined with the fact that

$$Q_h(0; \alpha, \beta, N) \equiv 1$$

for any $h \in \mathbb{N}$ (cf. Definition 1.33), gives a recursive expression

$$(m-l-i+1)(\beta+2k+1)\Psi_{i,k+1} = (i-k)(\alpha+l+n-k)\Psi_{i-1,k} + L\rho_{i-1} \cdot \Big(U + V \cdot \eta_{i-1}\Big),$$

which concludes the proof. □

Remark 6.7. *The initial value*

$$\Psi_{m-l,k} = \frac{(n-k-l)!(k+l+1-n)_{m-k-l}}{(m-k-l)!(m+\sigma+k+l+1)_{n-k-l}}$$

is given by (6.10)*. By applying Theorem 6.1 to the initial values given in* (6.10) *and using some algebra, one gets an expression for* $\Psi_{m-l-1,k}$*:*

$$\Psi_{m-l-1,k} = -\frac{\Psi_{m-l,k}}{\alpha+2l+1} \cdot \Big(m+k-l+\beta+1+\frac{m+k+l+\sigma+1}{n-k-l-1}\Big).$$

Theorem 6.8. *The elements* $\Psi_{i,n-l}$ *(*$i = k+1, k+2, \ldots, m-l-1$*) satisfy the second-order non-homogeneous recurrence relation of the form*

$$\Big(\frac{(k-n+l)(m-l-i)(\beta+k+n-l)}{i-k+1} - B(m,i)\Big)\Psi_{i+1,n-l} + [C(m,i) - C(n,n-l)]\Psi_{i,n-l}$$
$$- A(m,i)\Psi_{i-1,n-l} = \frac{k-n+l}{i-k+1} \cdot L\rho_i \cdot \Big(U \cdot Q_{m-k-l}(n-k-l-1; \beta+2k, \alpha+2l, n-k-l-1)$$
$$+ V\eta_i \cdot Q_{m-k-l+1}(n-k-l-1; \beta+2k, \alpha+2l, n-k-l-1)\Big).$$

Proof. The proof is analogous to the proof of Theorem 6.6 and is completed by applying Theorem 6.1 to Eq. (1.68). □

6.2 Computing the Ψ table

The recurrence relations presented in the previous section can be used to efficiently find the values of Ψ_{ij} for $k \le i \le m-l, k \le j \le n-l$ (cf. §1.7.4). Although, when compared to the method based on Eq. (1.68), these approaches do not improve the computational complexity as a whole, they provide a much more potent ground for parallel computations.

6.2.1 Efficient computation of parameters in recurrence relations

Computing the coefficients

The coefficients L, U, V can be computed once and used throughout the computation of the whole table. The complexity of computing L is $O(n)$ (taking into account that $n > m$), while U and V can be computed in $O(1)$ time.

Now, one needs to compute ρ_i and η_i for $i = k, k+1, \ldots, m-l-1$. This can be done efficiently in $O(n)$ time using the following relations (they follow directly from Eq. (6.3)):

$$\begin{cases} \rho_k = \dfrac{(-1)^{m-k-l+1}}{(\alpha+2l+1)_{m-k-l}(\beta+2k+1)}, & \\ \rho_{i+1} = \dfrac{(-1)(\beta+k+i+2)}{(m+\alpha+l-i)} \cdot \rho_i & (i = k, k+1, \ldots, m-l-2), \\ \eta_k = (m-k-l)(m-k+\alpha+l) - (\beta+2k+1), & \\ \eta_{i+1} = \eta_i - (2m+\alpha+\beta+2) & (i = k, k+1, \ldots, m-l-2). \end{cases}$$

While the recursive approach to computing η_i does not reduce the $O(m-k-l)$ asymptotic computational complexity, this way of computation reduces the number of arithmetic operations required.

Computing the Hahn polynomials

Over the course of the computations, in Theorems 6.1, 6.3, and 6.5, it is necessary to compute the values of Hahn polynomials. More precisely, one needs to find the values of

$$Q_{m-k-l}(j;\beta+2k,\alpha+2l,n-k-l-1) \qquad (j=0,1,\ldots,n-k-l-1)$$

and

$$Q_{m-k-l+1}(j;\beta+2k,\alpha+2l,n-k-l-1) \qquad (j=0,1,\ldots,n-k-l-1).$$

Let $\delta \in \{0,1\}$. Using Eq. (1.20), one gets simple expressions for $j=0$ and $j=1$:

$$Q_{m-k-l+\delta}(0;\beta+2k,\alpha+2l,n-k-l-1) \equiv 1,$$

$$Q_{m-k-l+\delta}(1;\beta+2k,\alpha+2l,n-k-l-1) = 1-\frac{(m-k-l+\delta)(m+\delta+\alpha+\beta+k+l+1)}{(\beta+2k+1)(n-k-l-1)}.$$

The values of Hahn polynomials for larger j can be found using the difference equation (1.22), i.e.,

$$\begin{aligned}
&Q_{m-k-l+\delta}(x+1;\beta+2k,\alpha+2l,n-k-l-1)\\
&\quad=\Big(1+\frac{x(x-\alpha+k-l-n)+(m-k-l+\delta)(m+\delta+\sigma+k+l)}{(x-n+k+l+1)(x+\beta+2k+1)}\Big)\\
&\qquad\times Q_{m-k-l+\delta}(x;\beta+2k,\alpha+2l,n-k-l-1)\\
&\quad-\frac{x(x-\alpha+k-l-n)}{(x-n+k+l+1)(x+\beta+2k+1)}Q_{m-k-l+\delta}(x-1;\beta+2k,\alpha+2l,n-k-l-1).
\end{aligned}$$

6.2.2 Computations based on the diagonal recurrence relation

In §1.7.4, a simple way for computing the elements in the kth row is given, i.e., one can easily compute all elements Ψ_{kj} for $j=k,k+1,\ldots,n-l$ using Theorem 1.80. This gives an initial value which allows to use Theorem 6.1 to compute all Ψ_{ij} such that $j \geq i$. One still has to find the initial values for the case when $j<i$. To do that, one can compute a part of the last row. It is only necessary to compute the elements $\Psi_{m-l,j}$ for $j=k,k+1,\ldots,m-l-1$ using Theorem 6.4. The remainder of the last row, i.e., $\Psi_{m-l,m-l}$ and all subsequent elements, will be computed using Theorem 6.1, starting from the kth row. Note that each of $n+m-2k-2l+1$ diagonal recurrence sequences (one for each initial value) is independent and can be computed in parallel. Figure 6.1 demonstrates this approach. Its implementation is given in Algorithm 6.1.

6.2.3 Computations based on the vertical recurrence relation

Provided that at least two elements in each column are given, Theorems 6.5, 6.6 and 6.8 allow to compute all the elements in the Ψ table.

The elements $\Psi_{kk},\Psi_{k,k+1},\ldots,\Psi_{k,n-l}$ can be found using Theorem 1.80. These quantities can then be used to compute $\Psi_{k+1,k+1},\Psi_{k+1,k+2},\ldots,\Psi_{k+1,n-l}$ by applying Theorem 6.1. The values $\Psi_{m-l,k}$ and $\Psi_{m-l-1,k}$ are given by Remark 6.7.

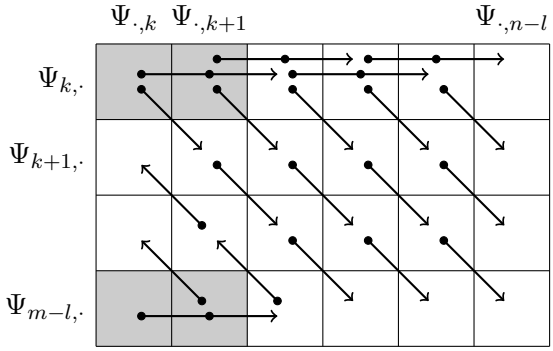

Figure 6.1: A recurrence scheme for the computation of the Ψ table organized around Theorem 6.1. Gray squares are computed using explicit formulas, while the arrows denote the recurrence relations used.

Algorithm 6.1 Computation of the Ψ table organized using Theorem 6.1

```
1: procedure PsiDiag(m, n, k, l, α, β)
2:     Ψ ← Matrix(m, n)
3:     for j ← k, n − l do
4:         Ψ_kj ← Theorem 1.80
5:     end for
6:     parallel for z ← 0, n − k − l do
7:         for i ← k, m − l do
8:             Ψ_{i,i+z} ← Theorem 6.1
9:         end for
10:    end parallel for
11:    for j ← k, m − l − 1 do
12:        Ψ_{m−l,j} ← Theorem 6.4
13:    end for
14:    parallel for j ← k + 1, m − l − 1 do
15:        for z ← 1, j − k do
16:            Ψ_{m−l−z,j−z} ← Theorem 6.1
17:        end for
18:    end parallel for
19:    return Ψ
20: end procedure
```

Each column now has two consecutive elements known, and one can apply the vertical recurrence relations given in Theorems 6.5, 6.6 and 6.8 to compute all the column. Note that at this point there is no interdependence between the columns and each can be evaluated in parallel. This approach is presented in Algorithm 6.2 and illustrated in Figure 6.2.

One can facilitate more parallel computations. Just as it is the case with computing the first two rows with the exception of $\Psi_{k+1,k}$, one can use the recurrence relations (6.9) and (6.1) to compute the last two rows except $\Psi_{m-l-1,n-l}$. By doing so, one can divide each column in two and compute these two parts in parallel, which nearly doubles the number of possible simultaneous computations of the elements of the Ψ table.

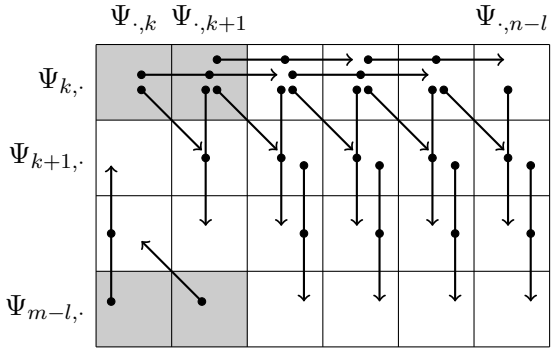

Figure 6.2: A recurrence scheme for the computation of the Ψ table organized around Theorem 6.5. Gray squares are computed using explicit formulas, while the arrows denote the recurrence relations used.

Algorithm 6.2 Computation of the Ψ table organized using Theorem 6.5

```
1: procedure PsiVer(m, n, k, l, α, β)
2:     Ψ ← Matrix(m, n)
3:     for j ← k, n − l − 1 do
4:         Ψ_kj ← Theorem 1.80
5:         Ψ_{k+1,j+1} ← Theorem 6.1
6:     end for
7:     Ψ_{k,n−l} ← Theorem 1.80
8:     Ψ_{m−l,k} ← Remark 6.7
9:     Ψ_{m−l−1,k} ← Remark 6.7
10:    for i ← m − l − 2, k + 1 do
11:        Ψ_ik ← Theorem 6.6
12:    end for
13:    parallel for j ← 1, n − k − l − 1 do
14:        for i ← 1, m − k − l − 1 do
15:            Ψ_{i+1,j} ← Theorem 6.5
16:        end for
17:    end parallel for
18:    for i ← 1, n − k − l − 1 do
19:        Ψ_{i+1,n−k−l} ← Theorem 6.8
20:    end for
21:    return Ψ
22: end procedure
```

Bibliography

[1] Y. J. Ahn. Using Jacobi polynomials for degree reduction of Bézier curves with C^k-constraints. *Computer Aided Geometric Design*, 20:423–434, 2003.

[2] Y. J. Ahn, B.-G. Lee, Y. Park, and J. Yoo. Constrained polynomial degree reduction in the L_2 norm equals best weighted euclidean approximation of Bézier coefficients. *Computer Aided Geometric Design*, 21:181–191, 2004.

[3] G. E. Andrews, R. Askey, and R. Roy. *Special functions encyclopedia of mathematics and its applications*, volume 71. Cambridge University Press, Cambridge, 1999.

[4] D. Bakhshesh and M. R. Samiee. The weighted dual functions for Wang-Said type generalized ball bases with and without boundary constraints. *International Journal of Computer and Electrical Engineering*, 4:573–577, 2012.

[5] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[6] R. E. Barnhill. Surfaces in computer aided geometric design: a survey with new results. *Computer Aided Geometric Design*, 2(1):1–17, 1985.

[7] M. Bartoň and B. Jüttler. Computing roots of polynomials by quadratic clipping. *Computer Aided Geometric Design*, 24:125–141, 2007.

[8] L.H. Bezerra. Efficient computation of Bézier curves from their Bernstein-Fourier representation. *Applied Mathematics and Computation*, 220:235–238, 2013.

[9] W. Boehm and A. Müller. On de Casteljau's algorithm. *Computer Aided Geometric Design*, 16:587–605, 1999.

[10] G. Brunnett, T. Schreiber, and J. Braun. The geometry of optimal degree reduction of Bézier curves. *Computer Aided Geometric Design*, 13:773–788, 1996.

[11] J. Bustamante. *Bernstein operators and their properties*. Birkhäuser, 2017.

[12] P. Bézier. Définition numérique des courbes et surfaces I (*in French*). *Automatisme*, XI:625–632, 1966.

[13] P. Bézier. Définition numérique des courbes et surfaces II (*in French*). *Automatisme*, XII:17–21, 1967.

[14] P. Bézier. Procédé de définition numérique des courbes et surfaces non mathématiques (*in French*). *Automatisme*, XIII:189–196, 1968.

[15] W. Böhm. Über die Konstruktion von B-Spline-Kurven (*in German*). *Computing*, 18: 161–166, 1977.

[16] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Communications of the ACM*, 16:361–368, 1996.

[17] G.-D. Chen and G.-J. Wang. Optimal degree reduction of Bézier curves with constraints of endpoint continuity. *Computer Aided Geometric Design*, 19:365–377, 2002.

[18] F. Chudy and P. Woźny. Differential-recurrence properties of dual Bernstein polynomials. *Applied Mathematics and Computation*, 338:537–543, 2018.

[19] F. Chudy and P. Woźny. Fast and accurate evaluation of dual Bernstein polynomials. *Numerical Algorithms*, 87:1001–1015, 2021.

[20] Z. Ciesielski. The basis of B-splines in the space of algebraic polynomials. *Ukrainian Mathematical Journal*, 38:311–315, 1987.

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

[22] G. Dahlquist and Å. Björck. *Numerical methods in scientific computing. Vol. I.* SIAM, 2008.

[23] M. Daniel and J. C. Daubisse. The numerical problem of using Bézier curves and surfaces in the power basis. *Computer Aided Geometric Design*, 6:121–128, 1989.

[24] K. R. Davidson and A. P. Donsig. *Real Analysis with Real Applications*. Prentice Hall, Inc., Upper Saddle River, 2002.

[25] P. J. Davis. Leonhard Euler's integral: A historical profile of the gamma function. *The American Mathematical Monthly*, 66(10):849–869, 1959.

[26] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Heidelberg, 3 edition, 2008.

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

[28] C. de Boor. B-form basics. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 131–148. SIAM, Philadelphia, 1987.

[29] P. de Casteljau. Outillage méthodes calcul (*in French*). Technical report, André Citroën Automobile SA, 1959.

[30] P. de Casteljau. Courbes et surfaces à pôles (*in French*). Technical report, André Citroën Automobile SA, 1959.

[31] P. de Casteljau. De Casteljau's autobiography: My time at Citroën. *Computer Aided Geometric Design*, 16:583–586, 1999.

[32] H. Dette. New bounds for Hahn and Krawtchouk polynomials. Technical Report 93-31C, Institut für Mathematische Stochastik, Technische Universität Dresden, 1993. `https://arxiv.org/abs/math/9406223`.

[33] P. Dierckx. *Curve and Surface Fitting with Splines*. Clarendon Press, 1993.

[34] M. Eck. Least squares degree reduction of Bézier curves. *Computer Aided Design*, 27: 845–851, 1995.

[35] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3 (2):83–127, 1986.

[36] G. Farin. *Curves and surfaces for Computer-Aided Geometric Design. A practical guide*. Academic Press, Boston, 5 edition, 2002.

[37] R. T. Farouki. Convergent inversion approximations for polynomials in Bernstein form. *Computer Aided Geometric Design*, 17:179–196, 2000.

[38] R. T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29:379–419, 2012.

[39] R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.

[40] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.

[41] R. Goldman. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling*. Morgan Kaufmann, 2002.

[42] R. N. Goldman. Dual polynomial bases. *Journal of Approximation Theory*, 79(3): 311–346, 1994.

[43] D. M. Gordon. A survey of fast exponentiation method. *Journal of Algorithms*, 27(1): 129–146, 1998.

[44] P. Gospodarczyk and P. Woźny. Efficient degree reduction of Bézier curves with box constraints using dual bases. Technical Report 2016-12-09, University of Wrocław, Institute of Computer Science, 2016. `https://arxiv.org/abs/1511.08264`.

[45] P. Gospodarczyk and P. Woźny. An iterative approximate method of solving boundary value problems using dual Bernstein polynomials. Technical Report 2018-03-01, University of Wrocław, Institute of Computer Science, 2018. `http://arxiv.org/abs/1709.02162`.

[46] P. Gospodarczyk, S. Lewanowicz, and P. Woźny. $G^{k,l}$-constrained multi-degree reduction of Bézier curves. *Numerical Algorithms*, 71:121–137, 2016.

[47] P. Gospodarczyk, S. Lewanowicz, and P. Woźny. Degree reduction of composite Bézier curves. *Applied Mathematics and Computation*, 293:40–48, 2017.

[48] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[49] Z. Guohui, L. Xiuping, and S. Zhixun. A dual functional to the univariate B-spline. *Journal of Computational and Applied Mathematics*, 195:292–299, 2006.

[50] M. Jani, E. Babolian, and S. Javadi. Bernstein modal basis: Application to the spectral Petrov-Galerkin method for fractional partial differential equations. *Mathematical Methods in the Applied Sciences*, 40:7663–7672, 2017.

[51] M. Jani, S. Javadi, E. Babolian, and D. Bhatta. Bernstein dual-Petrov-Galerkin method: application to 2D time fractional diffusion equation. *Computational and Applied Mathematics*, 37:2335–2353, 2018.

[52] B. Jüttler. The dual basis functions of the Bernstein polynomials. *Advances in Computational Mathematics*, 8:345–352, 1998.

[53] S. Karlin and J. L. McGregor. The Hahn polynomials, formulas and an application. *Scripta Mathematica*, 26:33–46, 1961.

[54] P. Kiciak. *Podstawy modelowania krzywych i powierzchni (in Polish)*. Wydawnictwo WNT, 2019.

[55] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 1(15):287–299, 1986.

[56] R. Koekoek and R. F. Swarttouw. The Askey-scheme of hypergeometric orthogonal polynomials and its q-analogue. Technical Report 98-17, Delft University of Technology, Faculty of Information Technology and Systems, Department of Technical Mathematics and Informatics, 1998.

[57] W. Koepf. *Hypergeometric Summation: An Algorithmic Approach to Summation and Special Function Identities*. Springer, 2014.

[58] T. H. Koornwinder, R. Wong, R. Koekoek, and R. F. Swarttouw. Orthohonal polynomials. In F. Olver, D. Lozier, R. Boisvert, and C. Clark, editors, *The NIST Handbook of Mathematical Functions*, chapter 18, pages 435–484. Cambridge University Press, New York, NY, 2010.

[59] B.-G. Lee, Y. Park, and J. Yoo. Application of Legendre-Bernstein basis transformations to degree elevation and degree reduction. *Computer Aided Geometric Design*, 19:709–718, 2002.

[60] S. Lewanowicz and P. Woźny. Dual generalized Bernstein basis. *Journal of Approximation Theory*, 138:129–150, 2006.

[61] S. Lewanowicz and P. Woźny. Bézier representation of the constrained dual Bernstein polynomials. *Applied Mathematics and Computation*, 218:4580–4586, 2011.

[62] S. Lewanowicz and P. Woźny. Multi-degree reduction of tensor product Bézier surfaces with general boundary constraints. *Applied Mathematics and Computation*, 217:4596–4611, 2011.

[63] S. Lewanowicz, P. Woźny, and P. Keller. Polynomial approximation of rational Bézier curves with constraints. *Numerical Algorithms*, 59:607–622, 2012.

[64] S. Lewanowicz, P. Keller, and P. Woźny. Constrained approximation of rational triangular Bézier surfaces by polynomial triangular Bézier surfaces. *Numerical Algorithms*, 75:93–111, 2017.

[65] L. Liu and G. Wang. Explicit matrix representation for NURBS curves and surfaces. *Computer Aided Geometric Design*, 19(6):409–419, 2002.

[66] L. Liu, L. Zhang, B. Lin, and G. Wang. Fast approach for computing roots of polynomials using cubic clipping. *Computer Aided Geometric Design*, 26:547–559, 2009.

[67] L. Lu. Gram matrix of Bernstein basis: Properties and applications. *Journal of Computational and Applied Mathematics*, 280:37–41, 2015.

[68] L. Lu and G. Wang. Optimal degree reduction of Bézier curves with G^2-continuity. *Computer Aided Geometric Design*, 23:673–683, 2006.

[69] L. Lu and G. Wang. Application of Chebyshev II-Bernstein basis transformations to degree reduction of Bézier curves. *Journal of Computational and Applied Mathematics*, 221:52–65, 2008.

[70] E. Mainar and J. M. Peña. Error analysis of corner cutting algorithms. *Numerical Algorithms*, 22:41–52, 1999.

[71] Maplesoft, a division of Waterloo Maple Inc.. Maple, 2010.

[72] H. Mohammadi and A. R. Setoodeh. FSDT-based isogeometric analysis for free vibration behavior of functionally graded skew folded plates. *Iranian Journal of Science and Technology, Transactions of Mechanical Engineering*, 44:841–863, 2020.

[73] Y. Park and U. J. Choi. Degree reduction of Bézier curves and its error analysis. *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, 36(4): 399–413, 1995.

[74] J. Peters. Evaluation and approximate evaluation of the multivariate Bernstein-Bézier form on a regularly partitioned simplex. *ACM Transactions on Mathematical Software (TOMS)*, 20(4):460–480, December 1994.

[75] M. Petkovšek, H. S. Wilf, and D. Zeilberger. $A = B$. A K Peters Ltd., Wellesley, MA, 1996.

[76] L. Piegl and W. Tiller. Algorithm for degree reduction of B-spline curves. *Computer-Aided Design*, 27:101–110, 1995.

[77] L. A Piegl and W. Tiller. *The NURBS Book*. Springer, 1996.

[78] H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and B-Spline Techniques*. Springer, 2002.

[79] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.

[80] A. Rababah and M. Al-Natour. The weighted dual functionals for the univariate Bernstein basis. *Applied Mathematics and Computation*, 186:1581–1590, 2007.

[81] A. Rababah and M. Al-Natour. Weighted dual functions for Bernstein basis satisfying boundary conditions. *Applied Mathematics and Computation*, 199:1581–1590, 2008.

[82] A. Rababah, B.-G. Lee, and J. Yoo. A simple matrix form for degree reduction of Bézier curves using Chebyshev-Bernstein basis transformations. *Applied Mathematics and Computation*, 181:310–318, 2006.

[83] A. Ramanantoanina and K. Hormann. New shape control tools for rational Bézier curve design. *Computer Aided Geometric Design*, 88:102003, 2021.

[84] L. Romani and A. Viscardi. Construction and evaluation of PH curves in exponential-polynomial spaces. `https://arxiv.org/abs/2111.12479`.

[85] S. H. M. Roth, P. Diezi, and M. H. Gross. Triangular Bézier clipping. *CS technical report*, 347, 2000.

[86] P. Sablonnière. Discrete Bézier curves and surfaces. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 497–515. Academic Press, New York, 1992.

[87] P. Sablonnière. Discrete Bernstein bases and Hahn polynomials. *Journal of Computational and Applied Mathematics*, 49:233–241, 1993.

[88] P. Sablonniére. Spline and Bézier polygons associated with a polynomial spline curve. *Computer-Aided Design*, 10:257–261, 1978.

[89] L.L. Schumaker and W. Volk. Efficient evaluation of multivariate polynomials. *Computer Aided Geometric Design*, 3:149–154, 1986.

[90] T. W. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer Aided Geometric Design*, 22:538–549, 1990.

[91] H. Sunwoo. Matrix representation for multi-degree reduction of Bézier curves. *Computer Aided Geometric Design*, 22:261–273, 2005.

[92] H. Sunwoo and N. Lee. A unified matrix representation for degree reduction of Bézier curves. *Computer Aided Geometric Design*, 21:151–164, 2004.

[93] J. Wimp. *Computation with recurrence relations*. Pitman Publishing, London, 1984.

[94] P. Woźny. Construction of dual bases. *Journal of Computational and Applied Mathematics*, 245:75–85, 2013.

[95] P. Woźny. Construction of dual B-spline functions. *Journal of Computational and Applied Mathematics*, 260:301–311, 2014.

[96] P. Woźny and F. Chudy. Linear-time geometric algorithm for evaluating Bézier curves. *Computer Aided-Design*, 118:102760, 2020.

[97] P. Woźny and S. Lewanowicz. Multi-degree reduction of Bézier curves with constraints, using dual Bernstein basis polynomials. *Computer Aided Geometric Design*, 26:566–579, 2009.

[98] P. Woźny and S. Lewanowicz. Constrained multi-degree reduction of triangular Bézier surfaces using dual Bernstein polynomials. *Journal of Computational and Applied Mathematics*, 235:785–804, 2010.

[99] P. Woźny, P. Gospodarczyk, and S. Lewanowicz. Efficient merging of multiple segments of Bézier curves. *Applied Mathematics and Computation*, 268:354–363, 2015.

[100] D. Zeilberger. A fast algorithm for proving terminating hypergeometric identities. *Discrete Mathematics*, 90:207–211, 1990.

[101] D. Zeilberger. The method of creative telescoping. *Journal of Symbolic Computation*, 11:195–204, 1991.

[102] L. Zhang, J. Tan, H. Wu, and Z. Liu. The weighted dual functions for Wang-Bézier type generalized ball bases and their applications. *Applied Mathematics and Computation*, 215:22–36, 2009.

[103] L. Zhang, H. Wu, and J. Tan. Dual bases for Wang-Bézier basis and their applications. *Applied Mathematics and Computation*, 214:218–227, 2009.

[104] L. Zhang, H. Wu, and J. Tan. Dual basis functions for the NS-power basis and their applications. *Applied Mathematics and Computation*, 207:434–441, 2009.

[105] L. Zhang, J. Tan, and Z. Dong. The dual bases for the Bézier-Said-Wang type generalized ball polynomial bases and their applications. *Applied Mathematics and Computation*, 217:3088–3101, 2010.

[106] R.-J. Zhang and G.-J. Wang. A note on the paper in CAGD (2004, 21 (2), 181-191). *Computer Aided Geometric Design*, 22:815–817, 2005.

Index