MAT4170

Exercises for Spline Methods

August Femtehjell

Spring 2025

Contents

1 Bernstein polynomials **2**
1.1 Partition of unity . . . 2
1.2 Recursion . . . 2
1.3 Linear independence . . . 3
1.4 Differentiation . . . 4
1.5 Integration . . . 4
1.6 Bernstein-Beziér polynomials . . . 5
1.7 The de Casteljau algorithm . . . 7
1.8 BB splines . . . 8
1.9 Spline interpolation . . . 9
1.9.1 Linear spline interpolation . . . 9
1.9.2 Cubic Hermite spline interpolation . . . 9

2 B-Splines **10**

Abstract

This document contains my summary for the course MAT4170 – Spline Methods, taught at the University of Oslo in the spring of 2025, in preperation for the final oral exam. This summary is based on the lecture notes for the course, draft dated 25th of March, 2025. The code for everything, as well as this document, can be found at my GitHub repository: `https://github.com/augustfe/MAT4170`.

1 Bernstein polynomials

A Bernstein polynomial denoted B_i^d is defined by

$$B_i^d(x) = \binom{d}{i} x^i (1-x)^{d-i}, \tag{1.1}$$

where d is the degree of the polynomial, and i is the index of the polynomial. These polynomials satisfy a number of interesting properties.

1.1 Partition of unity

Firstly, they are all non-negative on the interval $[0, 1]$. We can clearly see this as then both $x \geq 0$ and $1 - x \geq 0$ (and of course the binomial coefficient as well). In addition, for all $x \in \mathbb{R}$ we have that

$$\sum_{i=0}^{d} B_i^d(x) = 1, \tag{1.2}$$

and the polynomials therefore form a partition of unity. We can see this by noting

$$1 = 1^d = (x + (1-x))^d = \sum_{i=0}^{d} \binom{d}{i} x^i (1-x)^{d-i} = \sum_{i=0}^{d} B_i^d(x), \tag{1.3}$$

by the binomial theorem.

1.2 Recursion

In order to compute the value of a Bernstein polynomial efficiently, we note that

$$\binom{d}{i} = \binom{d-1}{i-1} + \binom{d-1}{i}, \tag{1.4}$$

most easily recalled by thinking about Pascals triangle. With this, we have that

$$\begin{aligned}
B_i^d &= \binom{d}{i} x^i (1-x)^{d-i} \\
&= \left(\binom{d-1}{i-1} + \binom{d-1}{i} \right) x^i (1-x)^{d-i} \\
&= x \binom{d-1}{i-1} x^{i-1} (1-x)^{(d-1)-(i-1)} + (1-x) \binom{d-1}{i} x^i (1-x)^{(d-1)-i} \\
&= x B_{i-1}^{d-1}(x) + (1-x) B_i^{d-1},
\end{aligned}$$

forming the basis for de Casteljau's algorithm for BB polynomials, which we will get back to later. Note here that we follow the convention of setting $B_i^d(x) = 0$ for $i < 0$. The algorithm here basically amounts to following this recursion in a triangular scheme, starting from B_0^0, and then computing towards the desired value(s).

1.3 Linear independence

The Bernstein polynomials are linearly independent. This means that we can write any polynomial of the form

$$p(x) = \sum_{i=0}^{d} a_i x^i \tag{1.5}$$

as

$$p(x) = \sum_{i=0}^{d} c_i B_i^d(x). \tag{1.6}$$

In order to show this, we show that x^j can be written as a linear combination of Bernstein polynomials. By the Binomial theorem, we have

$$\begin{aligned} x^j &= x^j(x + (1-x))^{d-j} \\ &= x^j \sum_{i=0}^{d-j} \binom{d-j}{i} x^i (1-x)^{d-(i+j)} \\ &= \sum_{i=j}^{d} \binom{d-j}{i-j} x^i (1-x)^{d-i}. \end{aligned}$$

Next, we solve for the correct binomial coefficient.

$$\begin{aligned} \alpha_i \binom{d}{i} &= \binom{d-j}{i-j} \\ \alpha_i \frac{d!}{i!(d-i)!} &= \frac{(d-j)!}{(i-j)!(d-i)!} \\ \alpha_i &= \frac{(d-j)!}{d!} \frac{i!}{(i-j)!} \end{aligned}$$

Inserting this back, we then have

$$x^j = \sum_{i=j}^{d} \frac{(d-j)!}{d!} \frac{i!}{(i-j)!} \binom{d}{i} x^i (1-x)^{d-i} = \frac{(d-j)!}{d!} \sum_{i=j}^{d} \frac{i!}{(i-j)!} B_i^d(x), \tag{1.7}$$

which is the desired linear combination.

1.4 Differentiation

In a similar fashion to the recursion shown, the derivative of a Bernstein polynomial takes a very simple form. We have that for $1 \leq i \leq d-1$

$$\begin{aligned}(B_i^d)'dx &= \frac{d}{dx}\left(\binom{d}{i}x^i(1-x)^{d-i}\right)\\ &= \binom{d}{i}\left(ix^{i-1}(1-x)^{d-i} - (d-i)x^i(1-x)^{d-i-1}\right)\\ &= d\left(\binom{d-1}{i-1}x^{i-1}(1-x)^{(d-1)-(i-1)} - \binom{d-1}{i}x^i(1-x)^{(d-1)-i}\right)\\ &= d\left(B_{i-1}^{d-1}(x) - B_i^{d-1}(x)\right).\end{aligned}$$

The formula also works out in the cases where $i = 0$ or $i = d$.

1.5 Integration

Integration of Bernstein polynomials has a very nice property, namely that

$$\int_0^1 B_i^d(x)\,dx = \frac{1}{d+1}. \tag{1.8}$$

In order to show this, we first note that as the Bernstein polynomials of a given degree form a partition of unity, we have

$$\sum_{i=0}^{d}\int_0^1 B_i^d(x)\,dx = 1. \tag{1.9}$$

Next, using the formula for differentiation and the fundamental theorem of calculus, we have for $0 < i < d+1$

$$\int_0^1 (B_i^{d+1}(x))\,dx = B_i^{d+1}(1) - B_i^{d+1}(0) = 0. \tag{1.10}$$

Using instead the differentiation formula we have

$$\begin{aligned}\int_0^1 (B_i^{d+1}(x))\,dx &= 0\\ (d+1)\int_0^1 B_{i-1}^d(x) - B_i^d(x)\,dx &= 0\\ \int_0^1 B_{i-1}^d(x)\,dx &= \int_0^1 B_i^d(x)\,dx,\end{aligned}$$

which holds for $i = 1, \dots, d$. This means that all Bernstein polynomials integrate to the same value. As we have $d+1$ of these, we conclude by the partition of unity that the identity holds.

1.6 Bernstein-Beziér polynomials

We call a polynomial $p \in \pi_d$ written on the form

$$p(x) = \sum_{i=0}^{d} c_i B_i^d(x) \tag{1.11}$$

a Bernstein-Bezier polynomial, with coefficients $c_i \in \mathbb{R}$. With the coefficients, we define the *control points* (ξ_i, c_i) with $\xi_i = \frac{i}{d}$. One interesting property of a BB-polynomial, is that it tends to mimic the shape of the polygon defined from the control points, called the control polygon.

Initially, we can easily see that the values match at the endpoints, as

$$p(\xi_0) = p(0) = \sum_{i=0}^{d} c_i B_i^d(0) = c_0 B_0^d(0) = c_0. \tag{1.12}$$

The same is found at the other end. In addition, the value of $p(x)$ is a convex combination of the coefficients as all B_i^d are non-negative. This means that

$$\min_{0 \le i \le d} c_i \le p(x) \le \max_{0 \le i \le d} c_i, \qquad 0 \le x \le 1. \tag{1.13}$$

Additionaly, we have that the first derivative matches at each endpoint.

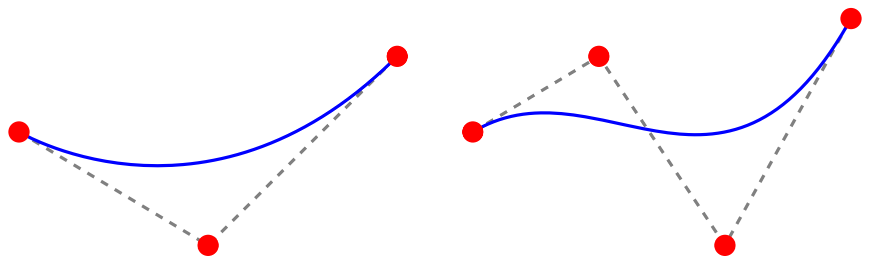

Figure 1: BB-polynomials of degree two and three. (Approximate, the quadratic one is wrong.)

The derivative of a BB polynomial is given by

$$
\begin{aligned}
p'(x) &= \sum_{i=0}^{d} c_i (B_i^d)'(x) \\
&= d \sum_{i=0}^{d} c_i (B_{i-1}^{d-1}(x) - B_i^{d-1}(x)) \\
&= d \sum_{i=0}^{d} c_i B_{i-1}^{d-1}(x) - d \sum_{i=0}^{d} c_i B_i^{d-1}(x) \\
&= d \sum_{i=-1}^{d-1} c_{i+1} B_i^{d-1}(x) - d \sum_{i=0}^{d-1} c_i B_i^{d-1}(x) \\
&= d \sum_{i=0}^{d-1} c_{i+1} B_i^{d-1}(x) - d \sum_{i=0}^{d-1} c_i B_i^{d-1}(x) \\
&= d \sum_{i=0}^{d-1} (c_{i+1} - c_i) B_i^{d-1}(x)
\end{aligned}
$$

such that

$$p'(0) = d(c_1 - c_0). \tag{1.14}$$

The slope between the first two control points is given by

$$\frac{y_1 - y_0}{x_1 - x_0} = \frac{c_1 - c_0}{\frac{1}{d} - \frac{0}{d}} = d(c_1 - c_0), \tag{1.15}$$

and we see that they match. Again, the same property holds at the other end. We write the derivative compactly as

$$p'(x) = d \sum_{i=0}^{d-1} \Delta c_i B_i^{d-1}(x), \tag{1.16}$$

with $\Delta c_i = c_{i+1} - c_i$.

This generalizes to higher order derivatives, with

$$p^{(r)}(x) = \frac{d!}{(d-r)!} \sum_{i=0}^{d-r} \Delta^r c_i B_i^{d-r}(x), \tag{1.17}$$

where $\Delta^r c_i$ is the r-th order forward difference, defined by

$$\Delta^r c_i = \Delta^{r-1} c_{i+1} - \Delta^{r-1} c_i. \tag{1.18}$$

Integrating a BB polynomial is now easy, with the result from Bernstein polynomials at hand. We have

$$\int_0^1 p(x)\,dx = \int_0^1 \sum_{i=0}^{d} c_i B_i^d(x)\,dx = \sum_{i=0}^{d} c_i \int_0^1 B_i^d(x)\,dx = \frac{\sum_{i=0}^{d} c_i}{d+1}, \tag{1.19}$$

which in other words is just the average of the coefficients.

1.7 The de Casteljau algorithm

The de Castljau algorithm gives an efficient way to evaluate a BB polynomial at a point. We could use the recursion formula found previously, however we can instead recurse on the coefficients. The algorithm is as follows:

1. Set $c_i^0 = c_i$ for $i = 0, \dots, d$.
2. For $r = 1, \dots, d$, compute $c_i^r = (1-x)c_i^{r-1} + xc_{i+1}^{r-1}$ for $i = 0, \dots, d-r$.
3. The final value c_0^r is the value $p(x)$.

To see why the final value is correct, note that by the recursion

$$\begin{aligned} p(x) &= \sum_{i=0}^{d} c_i B_i^d(x) \\ &= \sum_{i=0}^{d} c_i^0 (xB_{i-1}^{d-1}(x) + (1-x)B_i^{d-1}(x)) \\ &= \sum_{i=0}^{d} c_i^0 xB_{i-1}^{d-1}(x) + \sum_{i=0}^{d} c_i^0 (1-x)B_i^{d-1}(x) \\ &= \sum_{i=0}^{d-1} c_{i+1} xB_i^{d-1} + \sum_{i=0}^{d-1} c_i^0 (1-x)B_i^{d-1} \\ &= \sum_{i=0}^{d-1} c_i^1 B_i^{d-1}. \end{aligned}$$

Continuing with this recursion, we end up with

$$p(x) = \sum_{i=0}^{0} c_i^d B_i^0 = c_0^d, \tag{1.20}$$

as desired.

1.8 BB splines

In order to model complex functions with polynomials, we would typically have to have very high degree polynomials, which is not ideal. What we do instead is utilize piecewise polynomial functions, called splines. We will firstly do this with BB polynomial pieces, and in order to do that we need to amend a detail.

So far, we've only considered BB polynomials on the unit interval $[0, 1]$, but we now wish to do so on an arbitrary interval $[a, b]$. The conversion is rather simple, as by writing

$$x = (1 - \lambda)a + \lambda b, \tag{1.21}$$

we get the mapping

$$\lambda(x) = \frac{x - a}{b - a}, \tag{1.22}$$

such that $\lambda(x) \in [0, 1]$ for $x \in [a, b]$. The only real difference we have now is that a BB polynomial

$$p(x) = \sum_{i=0}^{d} c_i B_i^d(\lambda) \tag{1.23}$$

has control points (ξ_i, c_i), with $\xi_i = \frac{d-i}{d}a + \frac{i}{d}b$.

Everything else is practically indentical, although we have to be careful to note that we require the chain rule when taking derivatives, as we need to compute

$$\frac{dB_i^d(\lambda)}{dx} = \frac{dB_i^d(\lambda)}{d\lambda}\frac{d\lambda}{dx} = \frac{1}{b-a}\frac{dB_i^d(\lambda)}{d\lambda} \tag{1.24}$$

Suppose we want to construct a spline s defined by two polynomial pieces p and q on consecutive intervals $[a, b]$ and $[b, c]$, by

$$s(x) = \begin{cases} p(x), & a \leq x < b, \\ q(x), & b \leq x < c. \end{cases} \tag{1.25}$$

Note that we've defined the spline to be left continuous. We represent both p and q by BB polynomials, with

$$p(x) = \sum_{i=0}^{d} c_i B_i^d(\lambda) \quad \text{and} \quad q(x) = \sum_{i=0}^{d} e_i B_i^d(\mu), \tag{1.26}$$

where λ and μ are the local coordinates for each interval.

The question then is how continuous we want this spline to be at the $s(b)$. If we only want continuity, we have by the end point property that we need $c_d = e_0$, a relatively simple condition. More generally, for C^r continuity, we require that

$$p^{(k)}(b) = q^{(k)}(b) \qquad k = 0, 1, \ldots, r. \tag{1.27}$$

Writing out these expressions we have

$$\frac{d!}{(d-k)!}\frac{1}{(b-a)^k}\Delta^k c_{d-r} = \frac{d!}{(d-k)!}\frac{1}{(c-b)^k}\Delta^k e_0$$
$$\frac{1}{(b-a)^k}\Delta^k c_{d-r} = \frac{1}{(c-b)^k}\Delta^k e_0$$

as our condition. Say for instance that we have the c_i coefficients at hand, and wish to choose the e_i coefficients such that we have C^r continuity at the knot. We can then choose

$$e_i = \sum_{j=0}^{i} c_{d-i+j} B_j^i(\alpha) \qquad i = 0, \ldots, r, \tag{1.28}$$

where $\alpha = \frac{c-a}{b-a}$. I won't promise to remember the specifics of this...

1.9 Spline interpolation

Say we have some data, and we wish to fit a function to this. How should we proceed?

1.9.1 Linear spline interpolation

Linear spline interpolation is perhaps the simplest. Here, we have $x_1, x_2, \ldots, x_m \in \mathbb{R}$ and associated values $y_1, y_2, \ldots, y_m \in \mathbb{R}.$, and wish to find a linear spline $g : [x_1, x_m] \to \mathbb{R}$ such that

$$g(x_i) = y_i, \qquad i = 1, 2, \ldots, m. \tag{1.29}$$

We can do this simply by setting

$$g_i(x) = y_i \frac{x_{i+1} - x}{x_{i+1} - x} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}, \tag{1.30}$$

and having $g(x) = g_i(x)$ for $x_i \in [x_i, x_{i+1}]$. We can just as easily use BB polynomials, with

$$g_i(x) = y_i B_0^1(\lambda_i) + y_{i+1} B_1^1(\lambda_i). \tag{1.31}$$

1.9.2 Cubic Hermite spline interpolation

If we use higher degree polynomial pieces, we are able to get a smoother spline. We now let the g_i segments be cubic polynomials, and we now have slope values $s_1, s_2, \ldots, s_m$ in addition to the values previously, and we want to find g such that

$$g(x_i) = y_i \quad \text{and} \quad g'(x_i) = s_i, \qquad i = 1, 2, \ldots, m. \tag{1.32}$$

2 B-Splines