Lecture Note: Splines

Marc Toussaint

Learning & Intelligent Systems Lab, TU Berlin

August 27, 2024

A spline is a piece-wise polynomial path $x : [0, T] \to \mathbb{R}^n$. Let's first clearly distinguish the use of words *knot*, *waypoint*, and *control point*:

- A **knot** t_i is a point in *time*, $t_i \in \mathbb{R}$, we assume $t_i \in [0, T]$. For a spline, we have a non-decreasing sequence of knots $t_0, .., t_m$ (we assume $t_0 = 0$ and $t_m = T$) which partition the time interval $[0, T]$ into pieces $[t_i, t_{i+1}]$ so that the path is polynomial in each piece. Note that we may often have double or triple knots, meaning that several consecutive knots $t_i = t_{i+1}$ are equal, especially at the beginning and end.
- A **waypoint** x_i is a point on the path, typically at a knot, $x_i = x(t_i)$. So a path really passes through a waypoint. At waypoints, we often also care about velocities v_i and accelerations α_i, where $v_i = \dot{x}(t_i)$, $a_i = \ddot{x}(t_i)$, .
- A **control point** z_j is (usually) not a point *on* the path, but it indirectly defines the path as an linear combination of several control points. B-splines, defined below, make this explicit.

In robotics, there are two main conventions to define and parameterize splines: Hermite splines and B-splines. Hermite splines are defined by the knot sequence and explicitly prescribing waypoints x_i and (for cubic) velocities v_i at each knot (for quintic also acceperations a_i). In contrast, B-splines are specified by the knot sequence and K control points z_j. As in B-splines we do not need to provide velocities as part of the specification, they are sometimes easier to use in practice. However, the resulting path does not go (exactly) through the provided control points – but below we explain how a by simple matrix inversion (done just once) we can choose control points to ensure B-splines to pass through given waypoints.

Cubic splines are a common choice in robotics, as they have a still continuous (piece-wise linear) acceleration profile, and therefore limited jerk (3rd time derivative).

In the following we first discuss a single cubic spline-piece as a means of control, then Hermite splines, then B-splines.

Single cubic piece for timing-optimal control to a target

The following discusses a single cubic spline piece and how to use it for timing-optimal control to a target. Although very simple, the method is a powerful alternative to typical PD-control to a target. It also lays foundations on how timing-optimality can be realized with Hermite splines.

Consider a cubic polynomial $x(t) = at^3 + bt^2 + ct + d$. Given four boundary conditions $x(0) = x_0, \dot{x}(0) = v_0, x(\tau) = x_\tau, \dot{x}(\tau) = v_\tau$, the four coefficients are

$$d = x_0 \ , \tag{1}$$

$$c = \dot{x}_0 \ , \tag{2}$$

$$b = \frac{1}{\tau^2}\Big[3(x_\tau - x_0) - \tau(\dot{x}_\tau + 2\dot{x}_0)\Big] \ , \tag{3}$$

$$a = \frac{1}{\tau^3}\Big[-2(x_0 - x_\tau) + \tau(\dot{x}_\tau + \dot{x}_0)\Big] \ . \tag{4}$$

This cubic spline is in fact the solution to an optimization problem, namely it is the path that minimizes accelerations between these boundary conditions and it can therefore be viewed as the solution to optimal control with acceleration costs:

$$\min_x \int_0^\tau \ddot{x}(t)^2 \, dt \qquad \text{s.t.} \ \begin{pmatrix} x(0) \\ \dot{x}(0) \end{pmatrix} = \begin{pmatrix} x_0 \\ v_0 \end{pmatrix}, \ \begin{pmatrix} x(\tau) \\ \dot{x}(\tau) \end{pmatrix} = \begin{pmatrix} x_1 \\ v_1 \end{pmatrix} . \tag{5}$$

The minimal costs can analytically be given as

$$\int_0^T \ddot{x}(t)^2 dt = 4\tau b^2 + 12\tau^2 ab + 12\tau^3 a^2 \tag{6}$$

$$= \frac{12}{\tau^3}\,[(x_1 - x_0) - \frac{\tau}{2}(v_0 + v_1)]^2 + \frac{1}{\tau}(v_1 - v_0)^2 \tag{7}$$

$$= \frac{12}{\tau^3} D^\top D + \frac{1}{\tau} V^\top V \ , \quad D := (x_1 - x_0) - \frac{\tau}{2}(v_0 + v_1), \ V := v_1 - v_0, \tag{8}$$

$$= \tilde{D}^\top \tilde{D} + \tilde{V}^\top \tilde{V} \ , \quad \tilde{D} := \sqrt{12}\ \tau^{-\frac{3}{2}}\ D, \ \tilde{V} := \tau^{-\frac{1}{2}}\ V \ , \tag{9}$$

where we used some help of computer algebra to get this right.

Eq. (7) explicitly gives the optimal cost in terms of boundary conditions (x_0, v_0, x_1, v_1) and time τ. Eq. (9) rewrites this as sum-of-squares, which help using quasi-Newton methods. This is a very powerful formulation to optimize boundary conditions and τ. The following is a simple application that realizes reactive control.

Single-piece optimal timing control

Consider the system is in state $(x, \dot{x})$ and you want to control it to a reference point $(x_{\text{ref}}, \dot{x}_{\text{ref}} = 0)$. An obvious approach would be to use a PD law $\ddot{x}_{\text{des}} = k_p(x_{\text{ref}} - x) + k_d(\dot{x}_{\text{ref}} - \dot{x})$ and translate $\ddot{x}_{\text{des}}$ to controls using inverse dynamics. By choosing k_p and k_d appropriately one can generate any desired damped/oscillatory-exponential approach behavior.

However, while PD laws are fundamental for low-level optimal control under noise (e.g. as result of the Riccati equation), they are actually not great for generating more macroscopic approach behavior: They are "only" exponentially converging, never really reaching the target in a definite time, never providing a clear expected time-to-target. And accelerations seem intuitively too large when far from the set point, and too small when close. (Which is why many heuristics were proposed, such as capped PD laws.)

Instead of imposing a desired PD behavior, we can impose a desired cubic spline behavior, which leads to succinct convergence in a finite expected time-to-target, as well as moderate gains when far. The approach is simply to choose an optimal τ (time-to-target) that minimizes

$$\min_{\tau, x} \ \alpha\tau + \int_0^\tau \ddot{x}(t)^2 \, dt \tag{10}$$

under our boundary conditions, assuming a cubic spline $x(t), t \in [0, \tau]$. Using (7), we know the optimal x and optimal control costs for given τ. When $\delta = x_{\text{ref}} - x$ and v are co-linear (i.e., the system moves towards the target), computer algebra can tell us the optimal τ:

$$\tau^* = \frac{1}{\alpha}\Big[\sqrt{6|\delta|\alpha + v^2} - |v|\Big] \ . \tag{11}$$

If the system has a lateral movement, the analytical solution provided by computer algebra is overly complex, but a numerical solution to the least-squares form (9) is very very efficient. However, in

practise, using (11) with scalar $v \leftarrow (\delta^\top v)/|\delta|$ for easy timing control of convergence to a target is highly effective and versatile.

To make this a reactive control scheme, in each control cycle

$$\tau^*$$

is reevaluated and the corresponding cubic spline reference send to low-level control. If there are no perturbations, the estimated τ^* will be the true time-to-target. See the SecMPD paper for details and comparision to a PD approach behavior.

Hermite Cubic Splines

A Hermite cubic spline is specified by the series of non-decreasing time knots, $t_0, .., t_m \in [0, T]$, $t_0 = 0, t_m = T$, and the waypoints x_i *and velocities* v_i at each time knot. There are no double knots, so the interval $[0, T]$ is split in m cubic pieces, where the ith piece is determined by the boundary conditions $(x_{i\text{-}1}, v_{i\text{-}1}, x_i, v_i)$ and $\tau_i = t_i - t_{i\text{-}1}$.

Specifying the timings (i.e., knots) and velocities of all waypoints is often not easy for a user. Therefore the question is whether a series of given weypoints can easily be augmented with optimal knots and waypoint velocities.

Continuity of states, velocities and accelerations at the knots is an issue. If we require the path to be $\mathcal{C}^0$ (continuous in states $x(t)$ only), each piece needs to respect one constraint with the previous, and for m pieces we have $1 + 3m$ degrees of freedom (assuming $x(t) \in \mathbb{R}^1$ for simplicity). If we require $\mathcal{C}^1$ (continuity in velocities), m pieces have $2 + 2m$ parameters; for $\mathcal{C}^2$ (continuity in accelerations) we only have $3 + m$ parameters; and for $\mathcal{C}^3$ (continuity in jerk) we have 4 parameters no matter how many pieces – which shows that using a single piece is the only option to get continuous jerk.

One typically assumes $\mathcal{C}^2$ (continuous accelerations), and optimizing both, the knots and waypoint velocities, under our optimal control objective is rather efficient and effective under these constraints. Note that the optimal control cost over the full spline is just the sum of single piece costs (9). This represents costs as a least-squares of differentiable features, where D can be interpreted as distance to be covered by accelerations, and V as necessary total acceleration, and the Jacobians of $\tilde{D}$ and $\tilde{V}$ w.r.t. all boundary conditions and τ_i are trivial. Exploiting this least-squares formulation we can use the Gauss-Newton approximate Hessian.

As a concequence, it is fairly efficient to solve for $\tau_{1:m}$, $v_{1:m\text{-}1}$ given $v_0, v_m, x_{0:m}$ under continuous acceleration constraints subject to total time and control costs.

As a final note, in Hermite quintic splines we need positions x_i, velocities v_i and accelerations a_i at each knot, which describe the quintic polynomial pieces between knots.

B-Splines

In B-splines, the path $x : [0, T] \to \mathbb{R}^n$ is expressed as a linear combination of control points $z_0, .., z_K \in \mathbb{R}^n$,

$$x(t) = \sum_{i=0}^{K} B_{i,p}(t)\ z_i\ , \tag{12}$$

where $B_{i,p} : \mathbb{R} \to \mathbb{R}$ maps the time t to the weighting of the ith control point – it blends in and out the ith control point. For any t it holds that $\sum_{i=0}^{K} B_{i,p}(t) = 1$, i.e., all the weights $B_{i,p}(t)$ sum to one (as with a probability distribution over i), and the path point $x(t)$ is therefore always in the convex hull of control points.

Concerning terminology, actually the functions $B_{i,p}(t)$ themselves are called **B-splines**, not the resulting path $x(t)$. (But in everyday robotics language, one often calls the path a B-spline.) As the linear (scalar) product in (12) is trivial, the maths (and complexity of code) is all about the B-splines $B_{i,p}(t)$.

The B-spline functions $B_{i,p}(t)$ are fully specified by a non-decreasing series of time knots $t_0, .., t_m \in [0, T]$ and the integer degree $p \in \{0, 1, ..\}$. Namely, the recursive definition is

$$B_{i,0}(t) = [t_i \leq t < t_{i+1}] \ , \quad \text{for } 0 \leq i \leq m-1 \ , \tag{13}$$

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} \ B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} \ B_{i+1,p-1}(t) \ , \quad \text{for } 0 \leq i \leq m-p-1 \ . \tag{14}$$

The zero-degree B-spline functions $B_{i,0}$ are binary indicators of $t_i \leq t < t_{i+1}$, and i ranges from $i = 0, .., m-1$. The 1st-degree B-spline functions $B_{i,1}$ have support in $t_i \leq t < t_{i+2}$ and i only ranges from $i = 0, .., m-2$, so that the normalization $\sum_{i=0}^{m-2} B_{i,1}(t) = 1$ holds (the recursion would also not be clearly defined for $i > m-2$). In general, degree p B-spline functions $B_{i,p}$ have support in $t_i \leq t < t_{i+p+1}$ and i ranges from $i = 0, .., m+p-1$, which is why we need $K+1$ control points $z_{0:K}$ with

$$K = m + p - 1 \ , \tag{15}$$

which ensures the normalization property $\sum_{i=0}^{K} B_{i,p}(t) = 1$ for every degree. The time derivative

$$\dot{B}_{i,p}(t) = \frac{1}{t_{i+p} - t_i}\Big[B_{i,p\text{-}1}(t) + (t - t_i) \ \dot{B}_{i,p\text{-}1}(t)\Big] + \frac{1}{t_{i+p+1} - t_{i+1}}\Big[- B_{i+1,p\text{-}1}(t) + (t_{i+p+1} - t) \ \dot{B}_{i+1,p\text{-}1}(t)\Big] \tag{16}$$

shows that the derivative of a degree-p B-spline is a linear combination of degree-$p-1$ splines. Therefore, a degree-p spline is always $p-1$-order differentiable. For instance, a quadratic B-spline has continuous velocities; a cubic B-spline has continuous accelerations.

Illustration

Fig. 1 provides an illustration of B-spline functions for degrees $p = 0, .., 4$. Above each plot of functions, a rough illustration of a resulting path is provided, where bullets indicate control points. Note that this illustration implies a localization of control points in time, namely roughly where the coresponding weighting function (B-spline function) is highest. But control points are formally not localized in time, they are just being linearly combined, $x(t) = \sum_{i=0}^{K} B_{i,p}(t) \ z_i$, with different weighting in time. However, intuitively we can see that for odd degrees, the "localization in time" of control points roughly aligns with knots, while for even degrees the localization is between knots. Further, the illustrations assume multi-knots at the start and end (namely p+1-fold knots), which ensures that the spline starts with z_0 and ends with z_K. Duplicated control points $z_{0:p//2}$ and $z_{K-p//2:K}$ (illustrated with gray bars) are needed to ensure also zero vel/acc/jerk at start and end (where $p//2$ means integer division). However, by choosing the initial/final $p//2+1$ control points non-equal we can impose arbitrary initial/final velocity (and acceleration for $p \geq 4$), as detailed below.

Retrieving the piece-wise polynomials: Convertion to Hermite spline

B-splines might seem rather different from Hermite splines. However, it still holds that the resulting path $x(t)$ as well as all the B-spline functions $B_{i,p}$ are piece-wise polynomials, with pieces of degree p between knots. So the semantics of the knot times $t_0, .., t_m$ is the separation of polynomial pieces – as is generally the case for splines – and not the temporal localization of control points. Inspecting the definition (13) of the B-spline functions one may tell that they are polynomial between knots, but one cannot easily read of the polynomial coefficients.

A simple and efficient way to convert a B-spline to Hermite cubic spline is to evaluate $(x(t), \dot{x}(t))$ at each knot and use equations (1)-(4) to get the four polynomial coefficient for each piece. Is the B-spline of degree less than cubic, the respective coefficient will be zero. This clarifies that each cubic B-spline can directly be converted to cubic Hermite.

The conversion from Hermite to B-spline is equally possible *iff* the Hermite spline is smooth (i.e., $p-1$-order continuous) at all knots. Focussing on cubic splines, $p-1$-order continuous means that accelerations

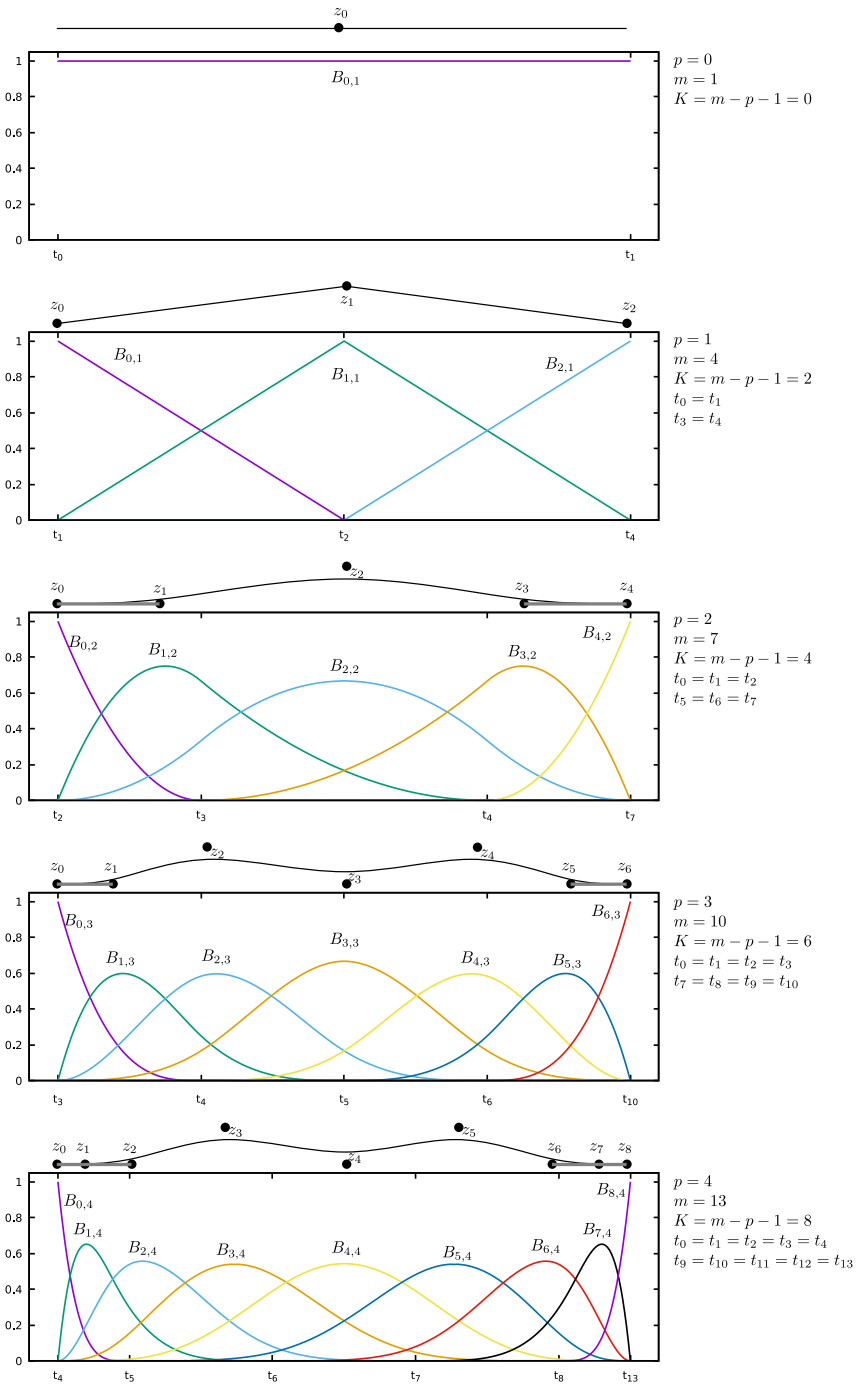

Figure 1: See section 0.3.1 for explanations.

are continuous; each cubic piece has 4 parameters, but 3 are bound by continuity with the next piece. If we have m pieces we have $3 + m$ parameters in total. For a cubic B-spline with multi-knots we need $m' = m + 6$ knots (3 duplicates at start and end), which also gives $m' - p = m + 3$ parameters. Below we discuss concretely how to retrieve a B-spline that explicitly goes through given waypoints, which can be used to do this convertion from a smooth cubic Hermite to cubic B-spline.[1]

In conclusion, we can convert back-and-forth between Hermite and B-spline *iff* we talk about $p-1$-order continuous splines. In this view, neither of them is better than the other – they are just alternative parameterizations of smooth splines.

B-spline Matrix for Time Discretized Paths

Splines describe a continuous path $x(t)$, but often we want to evaluate this path only at a finite number of time slices $t \in \{\hat{t}_1, .., \hat{t}_S\} \subset [0, T]$. E.g., this could be a grid of $S = 100$ time slices over which we want to optimize using KOMO, and for which we have to compute collision features. Let $x \in \mathbb{R}^{S\times n}$ be the time discretized path, and $z \in \mathbb{R}^{(K+1)\times n}$ be the stack of control points. Then the B-spline representation becomes

$$x = B_p z \ , \quad \text{with } B_p \in \mathbb{R}^{S\times(K+1)},\ B_{p,si} = B_{i,p}(\hat{t}_s) \ , \tag{17}$$

where B_p is the B-spline matrix of degree p for this particular time grid $\{\hat{t}_1, .., \hat{t}_S\}$.

So whenever we have a problem (e.g., NLP) defined over the fine resolution samples x, the B-spline matrix provides a linear re-parameterization and it is trivial to pull gradients (and Hessians) back to define a problem over z. In our code, KOMO defines NLPs over discretized trajectories x – it is trivial to wrap this with a linear B-spline parameterization to then imply a much lower-dimensional NLP over the control points z.

Ensuring B-splines pass through waypoints

As we emphasized, the control point parameterization is not necessarily intuitive for a user, as the resulting path does not transition through control points and control points are not "time-localized" at the knot times. If a user provides a series of waypoints at desired times $\hat{t}_s$, how can we construct a B-spline to ensure transitioning through these waypoints at the desired times?

The answer is again the matrix equation. Consider the cubic spline case and that the start and end points and times are fixed. Therefore $z_{0:1}$ and $z_{K\text{-}1:K}$, as well as knots $t_{0:3}$ and $t_{m-3:m}$ are fixed. The user wants waypoints $x_1, .., x_S$ at times $\hat{t}_1, .., \hat{t}_S$ *between* start and end.

We can distribute S knots $t_{4:3+S}$ uniformly between start and end knots (or also at $\hat{t}_1, .., \hat{t}_S$), from which it follows we have $m = S + 7$, and $K = m - p - 1 = S + 3$, which are $K + 1 = S + 4$ control points in total, of which 4 are already fixed. So the S middle control points are still free, and matrix inversion gives them from the desired waypoints,

$$z_{2:S+1} = B^{\text{-}1} x_{1:S} \ , \quad \text{with } B \in \mathbb{R}^{S\times S},\ B_{si} = B_{i+1,3}(\hat{t}_s),\ s, i = 1, .., S \ . \tag{18}$$

Ensuring boundary velocities

Consider an online control situation where the system is in state $(x, \dot{x})$ and we want to steer it through future waypoints. In the B-spline representation we have to construct a spline that starts with current state as starting boundary.

For degrees 2 and 3 this is simple to achieve: In both cases we usually have $z_0 = z_1$ and $z_{K\text{-}1} = z_K$ to ensure zero start and end velocities. Modifying z_1 directly leads to the start velocity $\dot{x}(0) = \dot{B}_{0,p}(0)z_0 + \dot{B}_{1,p}(0)z_1$.

[1] The method uses a matrix inversion. But the matrix is band-diagonal and the complexity of the inversion is linear in m. I'm not aware of a simpler way to do the conversion.

But because of the normalization we have $\dot{B}_{0,p}(0) = -\dot{B}_{1,p}(0)$, and therefore

$$\dot{x}(0) = \dot{B}_{0,p}(0)(z_0 - z_1) \tag{19}$$

$$z_1 = z_0 - \frac{\dot{x}(0)}{\dot{B}_{0,p}(0)} \ . \tag{20}$$

Gradients

The gradients of a B-spline represented path w.r.t. control points are trivial. But the gradients w.r.t. the knots are less trivial. Here the basic equations:

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t) \tag{21}$$

$$=: v\ B_{i,p-1} + w\ B_{i+1,p-1} \tag{22}$$

$$\partial_{t_i} B_{i,p} = \Big[\frac{-1}{t_{i+p} - t_i} + \frac{t - t_i}{(t_{i+p} - t_i)^2}\Big]\ B_{i,p-1} + v\ \partial_{t_i} B_{i,p-1} + w\ \partial_{t_i} B_{i+1,p-1} \tag{23}$$

$$= \Big[\frac{-1}{t - t_i} + \frac{1}{t_{i+p} - t_i}\Big]\ v\ B_{i,p-1} + v\ \partial_{t_i} B_{i,p-1} + w\ \partial_{t_i} B_{i+1,p-1} \tag{24}$$

$$\partial_{t_{i+1}} B_{i,p} = \Big[\frac{1}{t_{i+p+1} - t_{i+1}}\Big]\ w\ B_{i+1,p-1} + v\ \partial_{t_{i+1}}\ B_{i,p-1} + w\ \partial_{t_{i+1}}\ B_{i+1,p-1} \tag{25}$$

$$\partial_{t_{i+p}} B_{i,p} = \Big[-\frac{1}{t_{i+p} - t_i}\Big]\ v\ B_{i,p-1} + v\ \partial_{t_{i+p}}\ B_{i,p-1} + w\ \partial_{t_{i+p}}\ B_{i+1,p-1} \tag{26}$$

$$\partial_{t_{i+p+1}} B_{i,p} = \Big[\frac{1}{t_{i+p+1} - t} - \frac{1}{t_{i+p+1} - t_{i+1}}\Big]\ w\ B_{i+1,p-1} + v\ \partial_{t_{i+p+1}}\ B_{i,p-1} + w\ \partial_{t_{i+p+1}}\ B_{i+1,p-1} \tag{27}$$