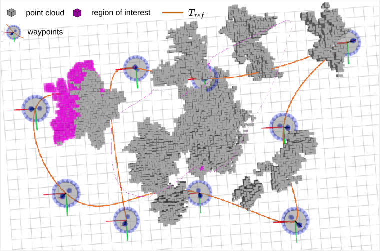
Motion Planning for Autonomous Vehicles

Curve Fitting

Geesara Kulathunga



April 5, 2023

Curve Fitting

Contents

N DEGREE POLYNOMIAL FITTING

The least-squares method can be used to fit *nth* order fitting

- Consider a set $(x_i, y_i),\ , i = 0, \ldots, m-1$

The least-squares method can be used to fit *nth* order fitting

- Consider a set $(x_i, y_i),\ , i = 0, \ldots, m-1$
- Define *nth* order fitting curve

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

N DEGREE POLYNOMIAL FITTING

The least-squares method can be used to fit *nth* order fitting

- Consider a set $(x_i, y_i), , i = 0, ..., m-1$
- Define *nth* order fitting curve

$$a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

- Convert to mean square error

$$J(x, y) = \Sigma_{i=0}^{m-1} (a_0 + a_1 x_i + a_2 x_i^2 + ... + a_n x_i^n - y_i)^2$$

N DEGREE POLYNOMIAL FITTING

The least-squares method can be used to fit *nth* order fitting

- Consider a set $(x_i, y_i),\ , i = 0, \ldots, m-1$
- Define *nth* order fitting curve

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

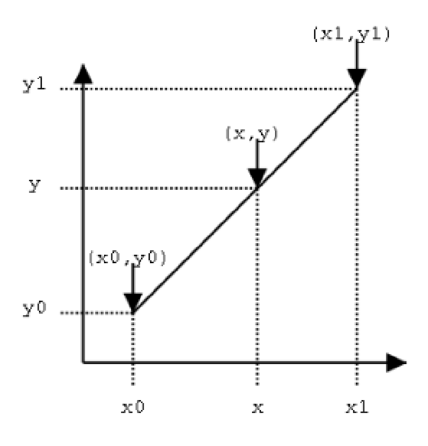
- Convert to mean square error

$$J(x, y) = \Sigma_{i=0}^{m-1} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_n x_i^n - y_i)^2$$

- Solve it

$$A^\top A \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_n \end{bmatrix} = A^\top \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_m \end{bmatrix}, A = \begin{bmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \ldots & x_m^n \end{bmatrix}$$

Linear interpolation



$$f(x) = f(x_0) + (x - x_0)\frac{f(x_0) - f(x_1)}{x_0 - x_1} \tag{1}$$

A **solution** of the **Euler-Lagrange** equation is called an **extremal** (minimum or maximum) of the **functional**. If Lagrangian $L(x, \dot{x})$ depends only on first-order derivatives, a second-order equation of motion can be found where only two boundary conditions are required, e.g., the **position** of the vehicle at an initial and final time. Such a **condition fixes the endpoint**. However, if Lagrangian $L(x, \dot{x}, \ddot{x})$ depends on second-order derivatives, a fourth-order equation of motion can be found. Hence, it requires four boundary conditions and **fixing the velocity** (as well as the **position**) at the **initial** and **final** time. Euler-Lagrange equations for a Lagrangian $L(x, \dot{x}, \ddot{x}, \dots)$ are given by

$$\frac{\partial L}{\partial x} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial L}{\partial x^{(n)}}\right) = 0. \tag{2}$$

Consider the AGV moves between the two positions within the time interval T.

$$J = \int_0^T L dt \tag{3}$$

where T is the motion duration and the performance index is L, the minimum jerk trajectory for unconstrained point-to-point movement, is

$$L = \left(\frac{\partial^3 x}{\partial t^3}\right)^2 + \left(\frac{\partial^3 y}{\partial t^3}\right)^2 \tag{4}$$

where, x and y indicates the position components. The objective is to deduce the local path minimizing the cost function J.

Minimum jerk trajectory (MJT) generation

Jerk is **the time derivation** of **acceleration**. Jerk is the way to define comfortness mathematically (or suppressing vibration effects or sudden acceleration change). Additionally, the first and second derivatives are continuous, so **continuous velocity and curvature are satisfied**.

- To solve this, Euler–Lagrange equation can be utilized.

$$\begin{aligned}
\frac{\partial L}{\partial x} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial L}{\partial x^{(n)}}\right) = 0, \\
\frac{\partial L}{\partial y} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial L}{\partial y^{(n)}}\right) = 0.
\end{aligned} \tag{5}$$

[1].https://courses.shadmehrlab.org/Shortcourse/minimumjerk.pdf

Minimum jerk trajectory (MJT) generation

- To solve this, Euler–Lagrange equation can be utilized.

$$\begin{aligned}\frac{\partial L}{\partial x}-\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right)+\cdots+(-1)^n\frac{d^n}{dt^n}\left(\frac{\partial L}{\partial x^{(n)}}\right)&=0,\\ \frac{\partial L}{\partial y}-\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right)+\cdots+(-1)^n\frac{d^n}{dt^n}\left(\frac{\partial L}{\partial y^{(n)}}\right)&=0.\end{aligned} \tag{5}$$

- Since jerk has to be minimized, $\frac{d}{dt}\left(\frac{\partial \dddot{x}^2}{\partial \dddot{x}}\right)=0$ and $\frac{d}{dt}\left(\frac{\partial \dddot{y}^2}{\partial \dddot{y}}\right)=0$ must be satisfied[1]. Hence,

$$\frac{d^6x}{dt^6}=0, \quad \frac{d^6y}{dt^6}=0 \tag{6}$$

[1].`https://courses.shadmehrlab.org/Shortcourse/minimumjerk.pdf`

Minimum jerk trajectory (MJT) generation

- To solve this, Euler–Lagrange equation can be utilized.

$$
\begin{aligned}
\frac{\partial L}{\partial x} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial L}{\partial x^{(n)}}\right) = 0, \\
\frac{\partial L}{\partial y} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial L}{\partial y^{(n)}}\right) = 0.
\end{aligned} \tag{5}
$$

- Since jerk has to be minimized, $\frac{d}{dt}\left(\frac{\partial \dddot{x}^2}{\partial \dddot{x}}\right) = 0$ and $\frac{d}{dt}\left(\frac{\partial \dddot{y}^2}{\partial \dddot{y}}\right) = 0$ must be satisfied[1]. Hence,

$$
\frac{d^6 x}{dt^6} = 0, \quad \frac{d^6 y}{dt^6} = 0 \tag{6}
$$

- Therefore, $x(t)$ and $y(t)$ must having the 5th order polynomial as follows:

$$
\begin{aligned}
x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\
y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5
\end{aligned} \tag{7}
$$

[1].https://courses.shadmehrlab.org/Shortcourse/minimumjerk.pdf

Quintic Polynomial

A polynomial of degree five defines a quintic function.

$$
\begin{aligned}
x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\
y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5
\end{aligned} \tag{8}
$$

Consider the initial condition $x_0, y_0, \dot{x}_0, \dot{y}_0, \ddot{x}_0, \ddot{y}_0$ at $t = 0$ and final condition $x_f, y_f, \dot{x}_f, \dot{y}_f, \ddot{x}_f, \ddot{y}_f$ at $t = T$ are given.

Quintic polynomial curve fitting **decouples** the along **x** and **y** directions, however, **position, velocity acceleration and jerk** are solved by **coupling**.

Hence,

$$a_0 = x_0, \quad a_1 = \dot{x}_0, \quad a_2 = \ddot{x}_0/2$$

$$A = \begin{bmatrix} t^3 & t^4 & t^5 \\ 3t^2 & 4t^3 & 5t^4 \\ 6t & 12t^2 & 20t^3 \end{bmatrix}, \quad b = \begin{bmatrix} x_f - a_0 - a_1 - a_2 t^2 \\ \dot{x}_f - a_1 - 2a_2 t \\ \ddot{x}_f - 2a_2 \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} = A^{-1} b$$

similar way $b_0, \ldots, b_5$ can be calculated. The higher-order derivatives can be estimated as follows:

$$\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \dot{x}(t) &= a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \\ \ddot{x}(t) &= 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \end{aligned} \tag{10}$$

- Given a set of points: $(x_0,y_0),(x_1,y_1),...,(x_n,y_n) \in \mathbb{R}^2$, to define a Lagrange polynomial, it is required to define a set of cardinal functions: $l_1,l_2,..,l_n \in \mathbb{P}^n$ such that

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{11}$$

for $\forall\, i \in [0,...,n]$. Term δ_{ij} is called Kronecker's delta.

- Given a set of points: $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)\in\mathbb{R}^2$, to define a Lagrange polynomial, it is required to define a set of cardinal functions: $l_1,l_2,..,l_n\in\mathbb{P}^n$ such that

$$l_i(x_j)=\delta_{ij}=\begin{cases}1, & i=j\\ 0, & i\neq j\end{cases} \tag{11}$$

  for $\forall\ i\in[0,\ldots,n]$. Term δ_{ij} is called Kronecker's delta.
- Term $\mathbb{P}^n$, denoted polynomial of nth order.

$$\begin{aligned} l_i(x) &= \prod_{j=0,j\neq i}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)\\ &= \frac{x-x_0}{x_i-x_0}\cdot\frac{x-x_1}{x_i-x_1}\cdots\frac{x-x_n}{x_i-x_n}\end{aligned} \tag{12}$$

  Both conditions: $l_i(x_i)=1$ and $l_i(x_k)=0$, $i\neq k$ can be verified.

Lagrange polynomials

- Therefore, the Lagrange form of a polynomial interpolation can be defined as

$$P_n(x) = \Sigma_{i=0}^{n} l_i(x) \cdot y_i \tag{13}$$

- Therefore, the Lagrange form of a polynomial interpolation can be defined as

$$P_n(x) = \Sigma_{i=0}^{n} l_i(x) \cdot y_i \tag{13}$$

- With that interpolation property is expressed as

$$P_n(x_j) = \Sigma_{i=0}^{n} l_i(x_j) \cdot y_i = y_j \tag{14}$$

- Formulate a second-order Lagrange polynomial that goes through these points: $x, y = (0, 1), (2/3, 0.5), (1, 0)$.

Lagrange polynomials

- Formulate a second-order Lagrange polynomial that goes through these points: $x, y = (0, 1), (2/3, 0.5), (1, 0)$.

- Define cardinal functions $l_0(x), l_1(x), l_2(x)$, afterwards Lagrange polynomial can be determined as

$$p_2(x) = l_0(x)y_0 + l_1(x)y_1 + l_2(x)y_2 \tag{15}$$

- Formulate a second-order Lagrange polynomial that goes through these points: $x, y = (0,1), (2/3, 0.5), (1,0)$.

- Define cardinal functions $l_0(x), l_1(x), l_2(x)$, afterwards Lagrange polynomial can be determined as

$$p_2(x) = l_0(x)y_0 + l_1(x)y_1 + l_2(x)y_2 \tag{15}$$

- The main **disadvantage** of the **Lagrange polynomial** is that **adding or removing a new point, it has to recompute all the** $l_i's$

Lagrange first order interpolation and second order interpolation

- Lagrange first-order interpolation

$$
\begin{aligned}
f(x) &= f(x_0) + (x - x_0)\frac{f(x_0) - f(x_1)}{x_0 - x_1} \\
&= \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)
\end{aligned}
\tag{16}
$$

Lagrange first order interpolation and second order interpolation

- Lagrange first-order interpolation

$$
\begin{aligned}
f(x) &= f(x_0) + (x - x_0)\frac{f(x_0) - f(x_1)}{x_0 - x_1} \\
&= \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)
\end{aligned}
\tag{16}
$$

- Lagrange second-order interpolation

$$
\begin{aligned}
f(x) &= f(x_0) + (x - x_0)\frac{f(x_0) - f(x_1)}{x_0 - x_1} + (x - x_0)(x - x_1)\frac{f(x_1) - f(x_2)}{x_0 - x_1} \\
&= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \\
&+ \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2)
\end{aligned}
\tag{17}
$$

$$f(x) = f(x_0)\delta_0(x) + f(x_1)\delta_1(x) + \ldots + f(x_n)\delta_n(x) \tag{18}$$

where $\delta_i(x)$ can be determined as

$$\delta_i(x) = \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j}\right) \tag{19}$$

Some example: `https://polympc.readthedocs.io/en/latest/ocp.html`

Given

$$
\begin{aligned}
\min_{f:[0,1]\to\mathbb{R}} \quad & \int_0^1 \left[f^{(2)}(t)\right]^2 dt \\
\text{s.t.} \quad & f(0)=a,\ f^{(1)}(0)=c \\
& f(1)=b,\ f^{(1)}(1)=d
\end{aligned}
\tag{20}
$$

- The objective

$$J(f) = \int_0^1 \left[f^{(2)}(t) \right]^2 dt$$

Variation of calculus to spline fitting

- The objective

$$J(f) = \int_0^1 \left[f^{(2)}(t)\right]^2 dt$$

- After considering the perturbation

$$J(f,p) = \int_0^1 \left[f^{(2)}(t) + hp^{(2)}(t)\right]^2 dt$$

- The objective

$$J(f) = \int_0^1 \left[f^{(2)}(t)\right]^2 dt$$

- After considering the perturbation

$$J(f,p) = \int_0^1 \left[f^{(2)}(t) + hp^{(2)}(t)\right]^2 dt$$

- Gateaux derivative

$$\begin{aligned}
dJ(f,p) &= \frac{d}{dh}\left(\int_0^1 \left[f^{(2)}(t) + hp^{(2)}(t)\right]^2 dt\right)_{h=0} \\
&= \int_0^1 2\left[f^{(2)}(t) + hp^{(2)}(t)\right] p^{(2)}(t)|_{h=0} dt \\
&= \int_0^1 2f^{(2)}(t)p^{(2)}(t)dt
\end{aligned}$$

Taking integration by parts

$$\begin{aligned} dJ(f,p) =& [2f^{(2)}(t)p^{(1)}(t)]_0^1 - [2f^{(3)}(t)p(t)]_0^1 + \int_0^1 2f^{(4)}(t)p(t)dt \\ =& \int_0^1 2f^{(4)}(t)p(t)dt \end{aligned}$$

If $f(t)$ is optimal for the considered constraint problem, then $dJ(f,p)=0$ as long as , $f(t)+hp(t)$ is feasible for small $h \Rightarrow p(0)=p(1)=p^{(1)}(0)=p^{(1)}(1)=0$. The function $p(t)$ can have infinitely many forms. Therefore, to obtain $dJ(f,p)=0$, $f^{(4)}(t)=0$. Hence, $f(t)=a^0+a^1t+a^2t^2+a^3t^3$.

Spline: piece-wise interpolation

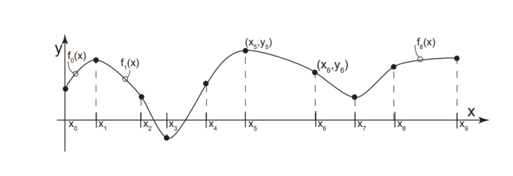
Only consider sub-interval without considering the **whole polynomial** as formulated in Lagrange nth order interpolation. Let $S(t)$ be interpolated function through a given set of points $(t_i, y_i)_{i=0}^{n}$. The ordered set $t_0 < t_1 < \ldots < t_n$ is called knots vector. Hence, $S(t)$ contains a set of piece-wise polynomials

$$S(t) = \begin{cases} S_0(t), & t_0 \le t \le t_1 \\ S_1(t), & t_1 \le t \le t_2 \\ \quad\vdots & \\ S_{n-1}(t) & t_{n-1} \le t \le t_n \end{cases} \tag{21}$$

$S(t)$ is a polynomial of degree k, if and only if $S(t)$ is k-1 times continuous differentiable

$$S_{i-1}(t_i) = S_i(t_i), S'_{i-1}(t_i) = S'_i(t_i), \ldots S_{i-1}^{(k-1)}(t_i) = S_i^{(k-1)}(t_i), \tag{22}$$

When n equals **1 linear** Spline, equals **2 quadratic** Spline, and equals **3 cubic** spline

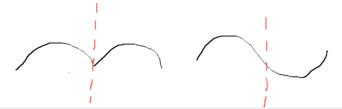


In general, $f(x_i) = a_i + b_i x + c_i x^2 + d_i x^3$, is the function which depicts the curve in between i^{th} and $i+1^{th}$ **control points**[1]. Hence, each curve represents by a cubic polynomial, with four coefficients for each. **How many parameters are to be solved?**

[1]. `https://people.cs.clemson.edu/~dhouse/courses/405/notes/splines.pdf`

Each segment pass through its control points
$f_i(x) = y_i,\ f_i(x_{i+1}) = y_{i+1}$

Consecutive segments should have the **same slop** and **same curvature** where they **join together** $f_i'(x_{i+1}) == f_{i+1}'(x_{i+1})$, $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$



How many parameters are to be solved?

Linear Spline

Piece-wise linear interpolation, i.e., straight-line. The constraints are

$$\begin{aligned} S_0(t_0) &= y_0 \\ S_{i-1}(t_i) = S_i(t_i) = y_i, \quad i = 1, 2, \ldots, n-1, \quad \Rightarrow S_i(t) &= y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i}(t - t_i) \\ S_{n-1}(t_n) &= y_n \end{aligned} \tag{23}$$

Given ordered set $(t_i, y_i)_{i=0}^{n}$, cubic spline can be defined as

$$S(t) = S_i(t) \quad for \quad t_i \le t \le t_{i+1} \tag{24}$$

where $S_i(t) = d_i(t-t_i)^3 + c_i(t-t_i)^2 + b_i(t-t_i) + a_i, i = 0, 1, ..., n-1$. Thus, the total number of unknown 4n. However, the following constraints must be satisfied $S(t)$ is a polynomial of degree k=3, if and only if $S(t)$ is k-1 times continuous differentiable

$$\begin{aligned} S_i(t_i) = y_i, S_i(t_{i+1}) = y_{i+1}, i = 0, 1, \ldots, n-1 \quad &\Rightarrow 2 \cdot n \text{ equations} \\ S_i'(t_{i+1}) = S_{i+1}'(t_{i+1}), i = 0, 1, \ldots, n-2 \quad &\Rightarrow n-1 \text{ equations} \\ S_i^{(2)}(t_{i+1}) = S_{i+1}^{(2)}(t_{i+1}), i = 0, 1, \ldots, n-2 \quad &\Rightarrow n-1 \text{ equations} \\ S_0^{(2)}(t_0) = 0,\ S_{n-1}^{(2)}(t_n) = 0 \quad &\Rightarrow 2 \text{ equations} \end{aligned} \tag{25}$$

- Consider $z_i = S^{(2)}(t_i), \quad i = 1, 2, \ldots, n-1, \quad z_0 = z_n = 0$. Since $S^{(2)}$ are linear functions, $S^{(2)}$ can be formulated in the Lagrange form

$$\begin{aligned} S_i^{(2)}(t) &= \frac{z_{i+1}}{t_{i+1} - t_i}(t - t_i) - \frac{z_i}{t_{i+1} - t_i}(t - t_{i+1}) \\ &= \frac{z_{i+1}}{h_i}(t - t_i) - \frac{z_i}{h_i}(t - t_{i+1}), \end{aligned} \tag{26}$$

where term $h_i = t_{i+1} - t_i$.

- Consider $z_i = S^{(2)}(t_i), \quad i = 1, 2, \ldots, n-1, \quad z_0 = z_n = 0$. Since $S^{(2)}$ are linear functions, $S^{(2)}$ can be formulated in the Lagrange form

$$\begin{aligned} S_i^{(2)}(t) &= \frac{z_{i+1}}{t_{i+1} - t_i}(t - t_i) - \frac{z_i}{t_{i+1} - t_i}(t - t_{i+1}) \\ &= \frac{z_{i+1}}{h_i}(t - t_i) - \frac{z_i}{h_i}(t - t_{i+1}), \end{aligned} \tag{26}$$

  where term $h_i = t_{i+1} - t_i$.

- After integration, terms $S_i'(t)$ and $S_i(t)$ can be derived as follows:

$$\begin{aligned} S_i'(t) &= \frac{z_{i+1}}{2h_i}(t - t_i)^2 - \frac{z_i}{2h_i}(t - t_{i+1})^2 + C_i - D_i \\ S_i(t) &= \frac{z_{i+1}}{6h_i}(t - t_i)^3 - \frac{z_i}{6h_i}(t - t_{i+1})^3 + C_i(t - t_i) - D_i(t - t_{i+1}) \end{aligned} \tag{27}$$

- Considering interpolating properties

$$S_i(t_i) = y_i, \Rightarrow y_i = -\frac{z_i}{6h_i}(-h_i)^3 - D_i(-h_i) \Rightarrow D_i = \frac{y_i}{h_i} - \frac{h_i}{6}z_i$$

$$S_i(t+1) = y_{i+1}, \Rightarrow y_{i+1} = \frac{z_{i+1}}{6h_i}(-h_i)^3 + C_i(-h_i) \Rightarrow C_i = \frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}$$

$$\Rightarrow y_{i+1} = a_{i+1} = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 \tag{28}$$

- Considering interpolating properties

$$S_i(t_i) = y_i, \Rightarrow y_i = -\frac{z_i}{6h_i}(-h_i)^3 - D_i(-h_i) \Rightarrow D_i = \frac{y_i}{h_i} - \frac{h_i}{6} z_i$$

$$S_i(t+1) = y_{i+1}, \Rightarrow y_{i+1} = \frac{z_{i+1}}{6h_i}(-h_i)^3 + C_i(-h_i) \Rightarrow C_i = \frac{y_{i+1}}{h_i} - \frac{h_i}{6} z_{i+1}$$

$$\Rightarrow y_{i+1} = a_{i+1} = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 \tag{28}$$

- Since D and C are known,

$$S_i(t) = \frac{z_{i+1}}{6h_i}(t - t_i)^3 - \frac{z_i}{6h_i}(t - t_{i+1})^3 + \left(\frac{y_{i+1}}{h_i} - \frac{h_i}{6} z_{i+1}\right)(t - t_i) - \left(\frac{y_i}{h_i} - \frac{h_i}{6} z_i\right)(t - t_{i+1}) \tag{29}$$

$$S_i'(t) = \frac{z_{i+1}}{2h_i}(t - t_i)^2 - \frac{z_i}{2h_i}(t - t_{i+1})^2 + \frac{y_{i+1} - y_i}{h_i} - \frac{z_{i+1} - z_i}{6} h_i$$

- Continuity of $S^{'}(t)$ requires $S'_{i-1}(t_i) = S'_i(t_i), i = 1, \ldots, n-1$,

$$\begin{aligned}
S'_i(t_i) &= -\frac{z_i}{2h_i}(-h_i)^2 + \underbrace{\frac{y_{i+1} - y_i}{h_i}}_{e_i} - \frac{z_{i+1} - z_i}{6} h_i \\
&= -\frac{1}{6} h_i z_{i+1} - \frac{1}{3} h_i z_i + e_i \\
S'_{i-1}(t_i) &= \frac{1}{6} h_{i-1} z_{i-1} + \frac{1}{3} h_{i-1} z_i + e_{i-1}
\end{aligned} \tag{30}$$

- Continuity of $S^{'}(t)$ requires $S'_{i-1}(t_i) = S'_i(t_i), i = 1, \ldots, n-1,$

$$\begin{aligned} S'_i(t_i) &= -\frac{z_i}{2h_i}(-h_i)^2 + \underbrace{\frac{y_{i+1} - y_i}{h_i}}_{e_i} - \frac{z_{i+1} - z_i}{6} h_i \\ &= -\frac{1}{6} h_i z_{i+1} - \frac{1}{3} h_i z_i + e_i \\ S'_{i-1}(t_i) &= \frac{1}{6} h_{i-1} z_{i-1} + \frac{1}{3} h_{i-1} z_i + e_{i-1} \end{aligned} \tag{30}$$

- Also, $S'_i(t_{i+1}) = S'_{i+1}(t_{i+1})$ and $S_i^{(2)}(t_{i+1}) = S_{i+1}^{(2)}(t_{i+1}), i = 0, \ldots, n-2,$

$$\begin{aligned} \Rightarrow b_{i+1} &= b_i + 2c_i h_i + 3d_i h_i^2 \\ \Rightarrow 2c_{i+1} &= 2c_i + 6d_i h_i \end{aligned} \tag{31}$$

- After setting them equal to each other,

$$\begin{cases} h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = 6(e_i - e_{i-1}), \quad i = 1, 2, \ldots, n-1 \\ z_0 = z_n = 0 \end{cases} \tag{32}$$

- After setting them equal to each other,

$$\begin{cases} h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = 6(e_i - e_{i-1}), & i = 1, 2, \ldots, n-1 \\ z_0 = z_n = 0 \end{cases} \tag{32}$$

- However, $z_i = S^{(2)}(t_i) = 2c_i$

$$\begin{cases} h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3(e_i - e_{i-1}), & i = 1, 2, \ldots, n-1 \\ z_0 = z_n = 0 \end{cases} \tag{33}$$

Cubic Spline

- Here both h_i both e_i are known, only the $\{c_i\}_{i=0}^{n}$ are unknown which can be solved by solving the following system of equations, where $\mathbf{A}$ is a $(n+1)\times(n+1)$ matrix and $\mathbf{Az}=\mathbf{b}$, in which $\mathbf{A}$

$$\mathbf{A}=\begin{pmatrix} 2(h_0+h_1) & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}$$

Cubic Spline

- Here both h_i both e_i are known, only the $\{c_i\}_{i=0}^{n}$ are unknown which can be solved by solving the following system of equations, where $\mathbf{A}$ is a $(n+1)\times(n+1)$ matrix and $\mathbf{A}\mathbf{z}=\mathbf{b}$, in which $\mathbf{A}$

$$\mathbf{A}=\begin{pmatrix} 2(h_0+h_1) & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}$$

- However, after incorporating the boundary condition, i.e., $z_0=S^{(2)}(t_i)=2c_0+6d_0(t_0-t_0)=0 \Rightarrow c_0=0,\ c_n=0.$

$$\mathbf{A}=\begin{pmatrix} 1 & 0 & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & 0 & 0 & 1 \end{pmatrix}$$

- In general, $\mathbf{A}$ is tri-diagonal, symmetric, and diagonal dominant. i.e., $2|h_{i-1}+h_i| > |h_i| + |h_{i-1}|$, which implies unique solution.

$$\mathbf{b} = \begin{pmatrix} 0 \\ 3(e_1 - e_0) \\ 3(e_2 - e_1) \\ \vdots \\ 3(e_{n-2} - e_{n-3}) \\ 3(e_{n-1} - e_{n-2}) \\ 0 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{pmatrix}$$

where $e_{i+1} - e_i = \frac{1}{h_{i+1}}(a_{i+2} - a_{i+1}) - \frac{1}{h_i}(a_{i+1} - a_i), \forall\ i = 0, \ldots, n.$

- In general, $\mathbf{A}$ is tri-diagonal, symmetric, and diagonal dominant. i.e., $2|h_{i-1}+h_i| > |h_i| + |h_{i-1}|$, which implies unique solution.

$$\mathbf{b} = \begin{pmatrix} 0 \\ 3(e_1 - e_0) \\ 3(e_2 - e_1) \\ \vdots \\ 3(e_{n-2} - e_{n-3}) \\ 3(e_{n-1} - e_{n-2}) \\ 0 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{pmatrix}$$

where $e_{i+1} - e_i = \frac{1}{h_{i+1}}(a_{i+2} - a_{i+1}) - \frac{1}{h_i}(a_{i+1} - a_i), \forall\, i = 0, \ldots, n.$

- Solving for d_i in eq.(31),

$$\begin{aligned} y_{i+1} =& a_{i+1} = a_i + b_i h_i + \frac{h_i^2}{3}(2c_i + c_{i+1}) \\ & \Rightarrow b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(c_{i+1} + 2c_i), \end{aligned} \tag{34}$$

Example 02

A fit cubic spline that passes these points: $(0,1),(1,e),(2,e^2),(3,e^3)$

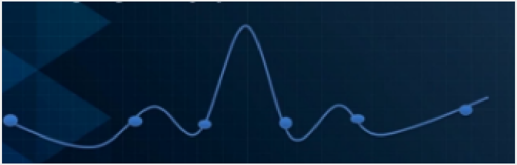
Example 02

A fit cubic spline that passes these points: $(0,1),(1,e),(2,e^2),(3,e^3)$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 3(e^2-2e+1) \\ 3(e^3-2e^2+e) \\ 0 \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \tag{35}$$

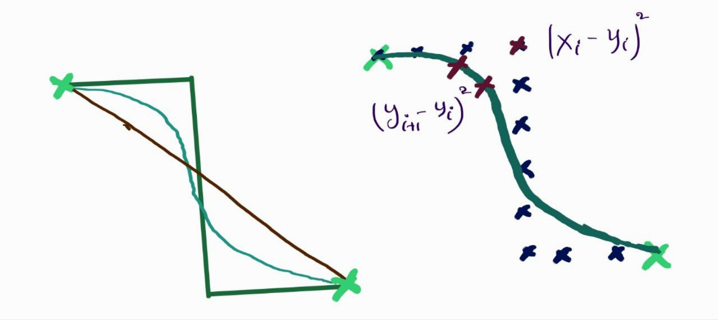
Cubic splines are continuous and smooth at the **connecting points**.

Other types of Curve fitting



- B-Spline: can generate control commands without smoothing
- Bezier
- Minimum-span
- Dubins curve: can not generate control commands without smoothing

- Suppose the planner gives a set of planning points $[x_1, x_2, ..., x_n]$
- Smoothed a set of points to be found $[y_1, y_2, ..., y_n]$



- By minimizing the following cost function

$$cost = \lambda_1 |x_i - y_i| + \lambda_2 |y_i - y_{i+1}|$$

  where λ_1 and λ_2 are regularization parameters. When λ_1 is larger than λ_2, the smooth point is closer to the original point, and vice versa, the smoother the path
- The gradient descent to find the minimum value to the defined threshold value of the cost function

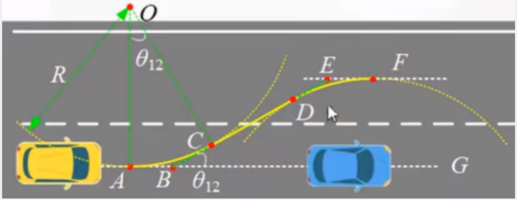
$$y_i = x_i,\ i = [0, ..., n]$$

traverse except for the start and end point and update y_i

$$y_i = y_i + \lambda_1 (x_i - y_i) + \lambda_2 (y_{i-1} - 2 \cdot y_i + y_{i+1})$$

Curve fitting: double arc trajectory interpolation

These types of trajectory fitting fit for turning and parking



`TODO:reference`

- A trajectory

$$p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i = [1, t, t^2, .., t^d] \cdot p \quad (36)$$

where $p = [p_0, p_1, \ldots, p_d]$.

[1]. Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors[C]//Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011: 2520-2525.
[2]. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments, Charles Richter, Adam Bry, and Nicholas Roy

- A trajectory

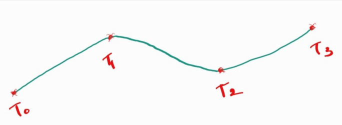
$$p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i = [1, t, t^2, .., t^d] \cdot p \quad (36)$$

where $p = [p_0, p_1, \ldots, p_d]$.

- Hence, at time t, position, velocity, acceleration, jerk, snap, etc, are calculated as

$$\begin{aligned} v(t) &= p^{(1)}(t) = [0, 1, 2t, 3t^2, 4t^3, \ldots, dt^{d-1}] \cdot p \\ a(t) &= p^{(2)}(t) = [0, 0, 2, 6t, 12t^2, \ldots, d(d-1)t^{d-2}] \cdot p \\ jerk(t) &= p^{(3)}(t) = [0, 0, 0, 6, 24t, \ldots, \frac{d!}{(d-3)!} t^{d-3}] \cdot p \\ snap(t) &= p^{(4)}(t) = [0, 0, 0, 0, 24, \ldots, \frac{d!}{(d-4)!} t^{d-4}] \cdot p \end{aligned} \quad (37)$$

[1]. Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors[C]//Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011: 2520-2525.
[2]. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments, Charles Richter, Adam Bry, and Nicholas Roy



Let $P_i(t)$ be the dth order polynomial in the ith segment that describes as follows:

$$P_i(t) = p_{i,0}t^0 + p_{i,1}t^1 + p_{i,2}t^2 + \ldots + p_{i,d}t^d = \begin{bmatrix} p_{i,0} & p_{i,1} & \ldots & p_{i,d} \end{bmatrix} \begin{bmatrix} t^0 \\ t^1 \\ \vdots \\ t^d \end{bmatrix}. \tag{38}$$

$P_i(t)$ provides a flat output for a given time index t for x, y, z, and yaw angle **independently** that is four-rotor drones and two-wheeled differential wheel robots (inaccurate simplification), their trajectories are independent on each axis.

THE OBJECTIVE FUNCTION WHEN $d = 7$

- $p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i$

THE OBJECTIVE FUNCTION WHEN $d = 7$

- $p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i$
- $p^{(4)}(t) = \Sigma_{i \geq 4} \frac{i!}{(i-4)!} t^{i-4} \cdot p_i$

The objective function when $d = 7$

- $p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i$
- $p^{(4)}(t) = \Sigma_{i \geq 4} \frac{i!}{(i-4)!} t^{i-4} \cdot p_i$
- $\left(p^{(4)}(t)\right)^2 = \Sigma_{i \geq 4} \Sigma_{l \geq 4} \frac{i!}{(i-4)!} t^{i-4} \frac{l!}{(l-4)!} t^{l+i-8} \cdot p_i \cdot p_l$

The objective function when $d = 7$

- $p(t) = p_0 + p_1 t + p_2 t^2 + \ldots + p_d t^d = \Sigma_{i=0}^{d} p_i t^i$
- $p^{(4)}(t) = \Sigma_{i \geq 4} \frac{i!}{(i-4)!} t^{i-4} \cdot p_i$
- $\left(p^{(4)}(t)\right)^2 = \Sigma_{i \geq 4} \Sigma_{l \geq 4} \frac{i!}{(i-4)!} t^{i-4} \frac{l!}{(l-4)!} t^{l+i-8} \cdot p_i \cdot p_l$
- $J = \int_{T_{j-1}}^{T_j} \left(p^{(4)}(t)\right)^2 dt$

The objective function when $d=7$

- $p(t) = p_0 + p_1 t + p_2 t^2 + ... + p_d t^d = \Sigma_{i=0}^{d} p_i t^i$
- $p^{(4)}(t) = \Sigma_{i\geq 4} \frac{i!}{(i-4)!} t^{i-4} \cdot p_i$
- $\left(p^{(4)}(t)\right)^2 = \Sigma_{i\geq 4} \Sigma_{l\geq 4} \frac{i!}{(i-4)!} t^{i-4} \frac{l!}{(l-4)!} t^{l+i-8} \cdot p_i \cdot p_l$
- $J = \int_{T_{j-1}}^{T_j} \left(p^{(4)}(t)\right)^2 dt$
- $J = \Sigma_{i\geq 4} \Sigma_{l\geq 4} \{ \frac{i!}{(i-4)!} t^{i-4} \frac{l!}{(l-4)!} (T_j^{l+i-7} - T_{j-1}^{l+i-7}) \cdot \frac{p_i \cdot p_l}{i+l-7} \}$

The objective function when $d = 7$

$$J_i = \int_{T_{j-1}}^{T_j} \left(p^{(4)}(t)\right)^2 dt$$

$$= \int_{T_{i-1}}^{T_i} \begin{bmatrix} p_d \\ p_{d-1} \\ \vdots \\ p_0 \end{bmatrix}^\top \begin{bmatrix} \frac{d!}{(d-4)!)} t^{d-4} \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{d!}{(d-4)!)} t^{d-4} \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}^\top \begin{bmatrix} p_d \\ p_{d-1} \\ \vdots \\ p_0 \end{bmatrix} = \begin{bmatrix} p_d \\ p_{d-1} \\ \vdots \\ p_0 \end{bmatrix}^\top$$

$$\begin{bmatrix} \frac{d!}{(d-4)!} \frac{d!}{(d-4)!} \frac{(T_i - T_{i-1})^{d+d-7}}{d+d-7} & \frac{d!}{(d-4)!} \frac{d!}{((d-5)!)d} \frac{(T_i - T_{i-1})^{d+d-1-7}}{d+d-1-7} & \dots \\ \vdots & \vdots & \vdots \\ \dots & \frac{j!}{(j-4)!} \frac{l!}{(l-4)!} \frac{(T_i - T_{i-1})^{j+l-7}}{j+l-7} & \dots \\ \vdots & \vdots & \vdots \\ 0 & 0 & \dots \end{bmatrix} \begin{bmatrix} p_d \\ p_{d-1} \\ \vdots \\ p_0 \end{bmatrix}$$

$$= P_i^\top Q_i P_i$$

The objective function when $d = 5$

In case, when using the 5th (d= $3 \times 2 - 1$) order polynomial, requires minimizing the jerk, i.e., $J = \int_0^T \left(\frac{d^3P(t)}{dt^3}\right)^2$, whose 6 unknown parameters are to be found. Then, the corresponding Q matrix is as follows:

$$Q = \begin{bmatrix} \frac{5!}{(5-3)!}\frac{5!}{(5-3)!}\frac{(T-0)^{5+5-5}}{5+5-5} = 720T^5 & 360T^4 & 120T^3 & 0 & 0 & 0 \\ 360T^4 & 192T^3 & 72T^2 & 0 & 0 & 0 \\ 120T^3 & 72T^2 & 36T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The objective function

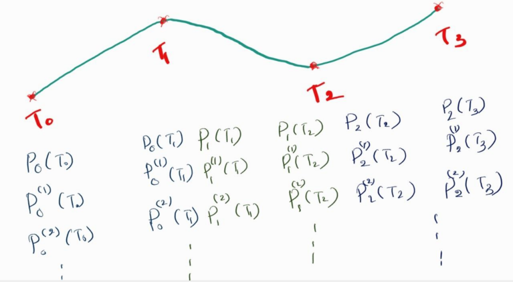
$$J = J_0 + J_1 + \ldots + J_M = \Sigma_{m=0}^{M} P_m Q_m P_m$$
$$= \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}^\top \begin{bmatrix} Q_1 & \ldots & 0 \\ 0 & \ddots & \vdots \\ 0 & \ldots & Q_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}$$

By optimising **J the minimum value** can be obtained.

$$J = J_0 + J_1 + \ldots + J_M = \Sigma_{m=0}^{M} P_m Q_m P_m$$
$$= \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}^\top \begin{bmatrix} Q_1 & \ldots & 0 \\ 0 & \ddots & \vdots \\ 0 & \ldots & Q_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}$$

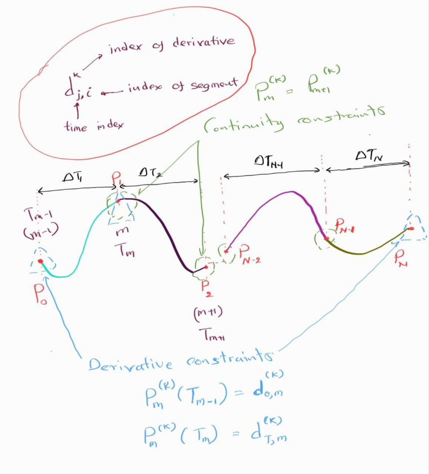
By optimising **J the minimum value** can be obtained.

However, the result of the possible optimization is that the **speed or acceleration between the intermediate points** is not **continuous**. Hence, some additional constraints are needed to limit the continuity.



ADDITIONAL CONSTRAINTS: CONTINUITY AND DIFFERENTIAL

- **Each segment** is a **polynomial**, i.e., it can be a **different order** of the polynomial, in general, the **order** is **fixed**
- **Time duration** is **known** in advance, and it can be varied or fixed between segments
- **Order of a polynomial** $d = 2l - 1$; l order of the optimization goal, e.g., snap minimization l=4



Additional constraints: continuity constraints

- Let $P_m(t)$ be the dth order polynomial in the mth segment that describes as follows:

$$P_m(t) = p_{m,0}t^0 + p_{m,1}t^1 + p_{m,2}t^2 + ... + p_{m,d}t^d = \Sigma_{j=0}^{d} p_{m,j}t^j \quad (39)$$

Additional constraints: continuity constraints

- Let $P_m(t)$ be the dth order polynomial in the mth segment that describes as follows:

$$P_m(t) = p_{m,0}t^0 + p_{m,1}t^1 + p_{m,2}t^2 + ... + p_{m,d}t^d = \Sigma_{j=0}^{d} p_{m,j}t^j \quad (39)$$

- Hence, to preserve the **continuity**, higher-order derivatives must satisfy, $k = 1, ..., l$(the highest order of derivative), for each of the **intermediate segments**

$$P_m^{(k)}(T_m) = P_{m+1}^{(k)}(T_m) \quad (40)$$

$$\begin{aligned} \Rightarrow \Sigma_{j \geq k} \frac{j!}{(j-k)!} T_m^{j-k} p_{m,j} - \Sigma_{r \geq k} \frac{r!}{(r-k)!} T_m^{r-k} p_{m+1,r} = 0 \\ \Rightarrow \begin{bmatrix} A_m & -A_{m+1} \end{bmatrix} \begin{bmatrix} P_m \\ P_{m+1} \end{bmatrix} = 0 \end{aligned} \quad (41)$$

Additional constraints : differential constraints

Differential constraints are applied **start** and **end** points of each **segment**.

$$\begin{bmatrix} T_{m-1}^{d} & T_{m-1}^{d-1} & \dots & T_{m-1}^{0} \\ T_{m}^{d} & T_{m}^{d-1} & \dots & T_{m}^{0} \\ dT_{m-1}^{d-1} & (d-1)T_{m-1}^{d-2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \dots & \frac{i!}{(i-k)!}T_{m-1}^{i-k} & \dots & \dots \\ \dots & \frac{i!}{(i-k)!}T_{m}^{i-k} & \dots & \dots \end{bmatrix} \begin{bmatrix} p_d \\ p_{d-1} \\ \vdots \\ p_0 \end{bmatrix} = \begin{bmatrix} \rho_{T_{m-1}} \\ \rho_{T_m} \\ v_{T_{m-1}} \\ v_{T_m} \\ a_{T_{m-1}} \\ a_{T_m} \\ \vdots \end{bmatrix} \tag{42}$$

$$\Rightarrow A_m P_m = d_m$$

Additional constraints

After **combining** the **continuity constraints** and the **differential constraints**

$$\begin{bmatrix} A_0 & 0 & 0 & 0 & \dots & 0 \\ A_0 & -A_1 & 0 & 0 & \dots & 0 \\ 0 & A_1 & 0 & 0 & \dots & 0 \\ 0 & A_1 & -A_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & A_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} = \begin{bmatrix} d_0 \\ 0 \\ d_1 \\ 0 \\ \vdots \\ d_M \end{bmatrix} \tag{43}$$

As a constrained quadratic problem

$$
\begin{aligned}
\min \quad & \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}^\top \begin{bmatrix} Q_1 & \dots & 0 \\ 0 & \ddots & \vdots \\ 0 & \dots & Q_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix} = P^\top Q P \\
\text{s.t.} \quad & \begin{bmatrix} A_0 & 0 & 0 & 0 & \dots & 0 \\ A_0 & -A_1 & 0 & 0 & \dots & 0 \\ 0 & A_1 & 0 & 0 & \dots & 0 \\ 0 & A_1 & -A_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & A_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} = \begin{bmatrix} d_0 \\ 0 \\ d_1 \\ 0 \\ \vdots \\ d_M \end{bmatrix}
\end{aligned}
\tag{44}
$$

Since **constraints** are **equality** constraints, such constraints can be **converted** into the **cost function** through some mathematical methods, then it becomes an **unconstrained optimization** problem, and then it can be **solved in a closed manner**. The benefits of closed solving are **faster solving** speed and **higher numerical stability**.

Let $P_m(t)$ be the dth order polynomial in the mth segment that describes as follows:

$$P_m(t) = p_{m,0}t^0 + p_{m,1}t^1 + p_{m,2}t^2 + \ldots + p_{m,d}t^d = \Sigma_{j=0}^{d} p_{m,j}t^j \tag{45}$$

For **minimum-snap trajectory** is required to have **a derived seventh-order polynomial**

$$P_m(t) = p_{m,0}t^0 + \ldots + p_{m,7}t^7 = \Sigma_{j=0}^{7} p_{m,j}t^j = \begin{bmatrix} p_{m,0} & p_{m,1} & \ldots & p_{m,7} \end{bmatrix} \begin{bmatrix} t^0 \\ t^1 \\ \vdots \\ t^7 \end{bmatrix}$$

Each **polynomial** has **8 unknown** coefficients. Hence **position**, **speed**, **acceleration**, and **jerk** constraints (including both **differential** and **continuity** constraints) for the endpoints of the trajectory are required.

$$
\begin{bmatrix} P_m(t) \\ P_m^{(1)(t)} \\ P_m^{(2)(t)} \end{bmatrix} = \begin{bmatrix} p_{m,5}t^5 & p_{m,4}t^4 & p_{m,3}t^3 & p_{m,2}t^2 & p_{m,1}t^1 & p_{m,0}t^0 \\ 5p_{m,5}t^4 & 4p_{m,4}t^3 & 3p_{m,3}t^2 & 2p_{m,2}t & p_{m,1} & 0 \\ 20p_{m,5}t^3 & 12p_{m,4}t^2 & 6p_{m,3}t^2 & 2p_{m,2} & 0 & 0 \end{bmatrix}
$$

$$
= \begin{bmatrix} t^5 & t^4 & t^3 & t^2 & t^1 & t^0 \\ 5t^4 & 4t^3 & 3t^2 & 2t & 1 & 0 \\ 20t^3 & 12t^2 & 6t^2 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{m,5} \\ p_{m,4} \\ p_{m,3} \\ p_{m,2} \\ p_{m,1} \\ p_{m,0} \end{bmatrix}
$$

Minimum-jerk trajectory generation: differential constraints

Consider **the relative time, i.e., start from zero and end at T, for each segment,**

$$P_m^{(k)}(T_m) = d_{T_m,j}^{(k)} \Rightarrow \Sigma_{j\geq k} \frac{j!}{(j-k)!} T_m^{j-k} p_{m,j}$$

and then

$$\begin{bmatrix} P(0) \\ P^{(1)}(0) \\ P^{(2)}(0) \\ P(T) \\ P^{(1)}(T) \\ P^{(2)}(T) \end{bmatrix} = \begin{bmatrix} 0^5 & 0^4 & 0^3 & 0^2 & 0^1 & 1 \\ 5\cdot 0^4 & 4\cdot 0^3 & 3\cdot 0^2 & 2\cdot 0 & 1 & 0 \\ 20\cdot 0^3 & 12\cdot 0^2 & 6\cdot 0^2 & 2 & 0 & 0 \\ T^5 & T^4 & T^3 & T^2 & T^1 & T^0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 20T^3 & 12T^2 & 6T^2 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix}$$

$$\Rightarrow d_m = A_m P_m$$

$$\Rightarrow P_m = A_m^{-1} d_m$$

Minimum-jerk trajectory generation: continuity constraints

$$A_m = \begin{bmatrix} 0^d & 0^{d-1} & \dots & 0^2 & 0^1 & 1 \\ d\cdot 0^{d-1} & (d-1)\cdot 0^{d-2} & \dots & 2\cdot 0^1 & 1 & 0 \\ d\cdot(d-1)\cdot 0^{d-2} & (d-1)\cdot(d-2)\cdot 0^{d-3} & \dots & 2\cdot 1\cdot 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ T^d & T^{d-1} & \dots & T^2 & T^1 & T^0 \\ d\cdot T^{d-1} & (d-1)\cdot T^{d-2} & \dots & 2\cdot T^1 & T^0 & 0 \\ d\cdot(d-1)\cdot T^{d-2} & (d-1)\cdot(d-2)\cdot T^{d-3} & \dots & 2\cdot 1\cdot T^0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$\min \quad \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix}^\top \begin{bmatrix} Q_1 & \dots & 0 \\ 0 & \ddots & \vdots \\ 0 & \dots & Q_M \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_M \end{bmatrix} = P^\top Q P \tag{46}$$

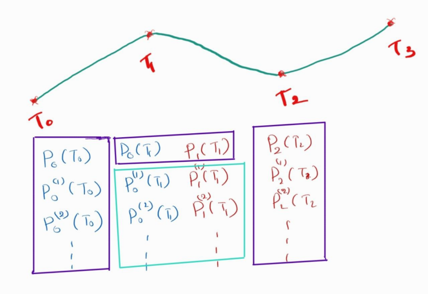
$$\min \quad \underbrace{\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_M \end{bmatrix}^\top}_{\mathbf{d}} \begin{bmatrix} A_0 & 0 & 0 & 0 & \dots & 0 \\ 0 & A_1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & A_M \end{bmatrix}^{-\top} \begin{bmatrix} Q_1 & \dots & 0 \\ 0 & \ddots & \vdots \\ 0 & \dots & Q_M \end{bmatrix} \tag{47}$$

$$\begin{bmatrix} A_0 & 0 & 0 & 0 & \dots & 0 \\ 0 & A_1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & A_M \end{bmatrix}^{-1} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_M \end{bmatrix}$$

$$\min \quad \mathbf{d}^\top A^{-\top} Q A^{-1} \mathbf{d} \tag{48}$$

The **differential constraints** have been brought into the **cost** function

Assume trajectory is defined from T_0 to T_2



The values of the variables d_{mf} in the purple box are all set in advance, and the variables d_{mp} in the light blue box are the optimal values assigned by the cost function during optimization, which are the variables that need to be optimized.

d_{mf} fixed derivatives at the start, the goal state and intermediate positions

d_{mp} free derivatives at all intermediate connections

Considering a trajectory with two segments

$$d_m = \begin{bmatrix} d^{(0)}_{0,0} \\ d^{(1)}_{0,0} \\ d^{(2)}_{0,0} \\ d^{(0)}_{T,0} \\ d^{(1)}_{T,0} \\ d^{(2)}_{T,0} \\ d^{(0)}_{0,1} \\ d^{(1)}_{0,1} \\ d^{(2)}_{0,1} \\ d^{(0)}_{T,1} \\ d^{(1)}_{T,1} \\ d^{(2)}_{T,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} d^{(0)}_{0,0} \\ d^{(1)}_{0,0} \\ d^{(2)}_{0,0} \\ d^{(0)}_{T,0} \\ d^{(0)}_{T,1} \\ d^{(1)}_{T,1} \\ d^{(2)}_{T,1} \\ d^{(1)}_{T,0} \\ d^{(2)}_{T,0} \end{bmatrix} = C^\top \begin{bmatrix} d_{mf} \\ d_{mp} \end{bmatrix}$$

Similar way, **the C matrix** can be generated for higher-order cases.
Due to the **nature of C**, the **continuity constraints** can be added into the **cost** function, because after using the C matrix, **equal variables are omitted** whose **values are equal**, consider $d_{mf} := d_F$ and $d_{mp} := d_P$

$$J = \min \quad \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top C A^{-\top} Q A^{-1} C^\top \begin{bmatrix} d_F \\ d_P \end{bmatrix} = \min \quad \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top R \begin{bmatrix} d_F \\ d_P \end{bmatrix} \tag{49}$$

$$\begin{aligned} J = \min \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top R \begin{bmatrix} d_F \\ d_P \end{bmatrix} &= \min \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \begin{bmatrix} d_F \\ d_P \end{bmatrix} \\ = d_F^\top R_{FF} d_F + d_F^\top R_{FP} d_P &+ d_P^\top R_{PF} d_F + d_P^\top R_{PP} d_P \end{aligned} \tag{50}$$

$$J = \min \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top C A^{-\top} Q A^{-1} C^\top \begin{bmatrix} d_F \\ d_P \end{bmatrix} = \min \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top R \begin{bmatrix} d_F \\ d_P \end{bmatrix} \quad (51)$$

The matrix Q is a symmetric matrix and J is a scalar, and $C A^{-\top} Q A^{-1} C^\top = \left(C A^{-\top} Q A^{-1} C^\top \right)^\top$. Hence, R must be a symmetric matrix. That is, $R_{PF} = R_{FP}^\top$. Therefore,

$$\begin{aligned} J =& \min \begin{bmatrix} d_F \\ d_P \end{bmatrix}^\top \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \begin{bmatrix} d_F \\ d_P \end{bmatrix} \\ =& d_F^\top R_{FF} d_F + d_F^\top R_{FP} d_P + d_P^\top R_{PF} d_F + d_P^\top R_{PP} d_P \\ =& d_F^\top R_{FF} d_F + 2 d_F^\top R_{FP} d_P + d_P^\top R_{PP} d_P \end{aligned} \quad (52)$$

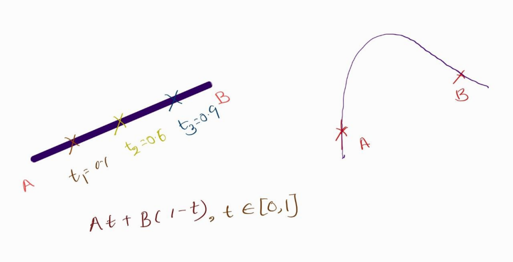
The optimal value for d_P is determined as

$$\begin{aligned} J &= d_F^\top R_{FF} d_F + 2 d_F^\top R_{FP} d_P + d_P^\top R_{PP} d_P \\ \frac{\partial J}{\partial d_P} &= 2 d_F^\top R_{FP} + 2 R_{PP} d_P = 0 \\ &\Rightarrow d_P^* = -R_{PP}^{-1} R_{FP}^\top d_F \end{aligned} \tag{53}$$

With that, the polynomial can be determined as

$$P = A^{-1} C^\top \begin{bmatrix} d_F \\ d_P^* \end{bmatrix}$$

B-splines and **Bezier** curves can be used for nonlinear curve fitting



Track interpolation and smoothing, i.e., trajectory interpolation, Path to trajectory control output, i.e., calculate position, speed, and acceleration with respect to time

The fitted curve **must pass** through the **start** and the **endpoint**, and the curve must not pass through the **intermediate control point**.

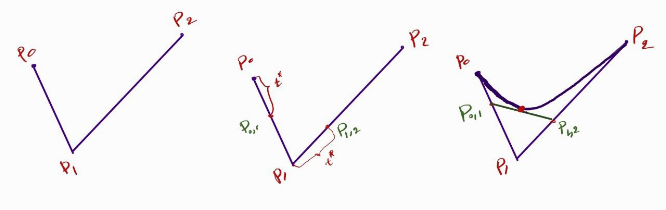
- First-order curve: it's a linear interpolation based on t

$$B_1(t) = P_0 + (P_1 - P_0)t \Rightarrow B_1(t) = (1-t)P_0 + tP_1,\ t \in [0,1]$$

Bezier curve fitting

- Second-order curve
  - Select 3 non-collinear points in the plane and connect them sequentially by straight lines
  - Select point $P_{0,1}$ in the first line segment P_0P_1, and find the corresponding point $P_{1,2}$ from the second line segment so that $P_0P_{0,1}$: P_0P_1 = $P_1P_{1,2}$:P_1P_2

$$P_{0,1} = (1-t)P_0 + tP_1, \quad P_{1,2} = (1-t)P_1 + tP_2$$



- Second-order curve

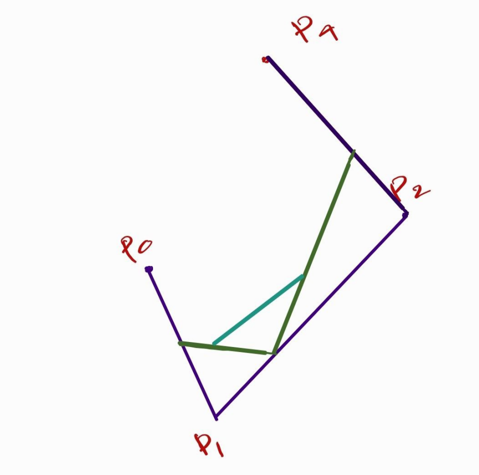
$$B_2(t) = (1-t)B_1(t)_{0,1} + tB_1(t)_{1,2} = (1-t)P_{0,1} + tP_{1,2}$$

- Second-order curve

$$B_2(t) = (1-t)B_1(t)_{0,1} + tB_1(t)_{1,2} = (1-t)P_{0,1} + tP_{1,2}$$

- $$\begin{aligned} B_2(t) &= (1-t)((1-t)P_0 + tP_1) + t((1-t)P_1 + tP_2) \\ &= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, t \in [0,1] \end{aligned}$$

- Third-order curve



Bezier curve formula

$$\mathbf{B}(t) = \Sigma_{i=0}^{n} P_i B_{i,n}(t), \quad t \in [0,1]$$

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i},\ i = 0, \ldots, n$$

Derivatives of Bezier curve

The control points are independent, hence, the derivation is directly deriving t

$$\frac{d}{dt} B_{i,n}(t) = n(B_{i-1,n-1}(t) - B_{i,n-1}(t))$$

$$\frac{d}{dt}\mathbf{B}(t) = \Sigma_{i=0}^{n-1} \Big(n(P_{i+1} - P_i) \Big) B_{i,n-1}(t)$$

If $\bar{P}_0 = n(P_1 - P_0), \bar{P}_1 = n(P_2 - P_1), \ldots, \bar{P}_{n-1} = n(P_n - P_{n-1})$

$$\frac{d}{dt}\mathbf{B}(t) = \Sigma_{i=0}^{n-1} \bar{P}_i B_{i,n-1}(t)$$

still a Bezier curve

Bezier curve fitting

- Curve shape is determined by its control points, if there are n control points, Bezier is constructed in order of n-1
- Local changes are not allowed, whole curve will change
- The sum of the coefficients is one, i.e., $(t+(1-t))^n = 1^n$
- Recursion property

$$B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0, 1, .., n$$

- Convex Hull property the Bezier curve will always be in the smallest **convex polygon** that contains **all the control points**
- The **first** and **last control points** are **exactly** the **start and end points** of the curve

- Let U be the set of non-decreasing numbers, $u_0 \leq u_1 \leq ... \leq u_m$ is called **knots**, the set U is called **knot vector**, and the half-open interval $[u_i, u_{i+1})$ is the ith **knot interval** (knot span). When u_i is equal, a **uniform B-spline** is formed.
- For $n+1$ number of control points P_i, $i = 0, ..., n$ connecting by m number of knots points, m = n+p+1, where p+1 order B-spline curve can e defined as

$$P(u) = \Sigma_{i=0}^{n} P_i N_{i,p}(u)$$

B-SPLINE CURVE FITTING

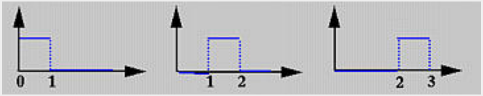
- Cox-de Boor recursive formula [1]:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

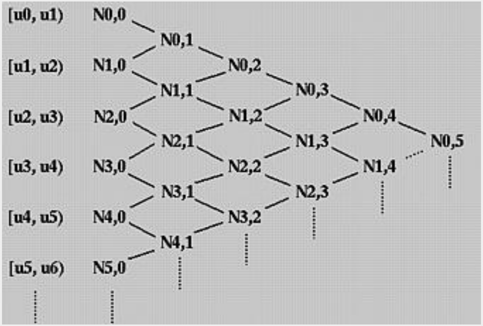
  where parameter p, the degree of the basis function.

- If p=0, all the basis functions become step functions, as indicated in the first expression. For example, consider $u_{=}0, u_1 = 1, u_2 = 2, u_3 = 3$, and p equals zero. That is, $N_{0,0}(u) = 1$ in $[0,1)$ and rest is zero, $N_{1,0}(u) = 1$ in $[1,2)$ and rest is zero, etc.



[1]https://en.wikipedia.org/wiki/De_Boor%27s_algorithm

- When p is greater than zero,



To calculate $N_{i,1}(u)$,$N_{i,0}(u)$,$N_{i+1,0}(u)$ are required. Hence, $N_{0,1}(u)$,$N_{1,1}(u)$,$N_{2,1}(u)$ can be calculated afterwards. Once, in general, $N_{i,1}(u)$ are computed, $N_{i,2}(u)$ can be computed and so on.

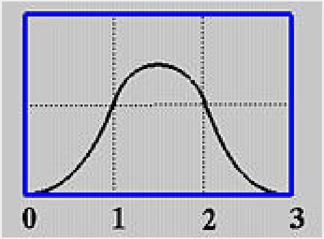
- Consider $U = \{0, 1, 2, 3\}$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

$$N_{0,1}(u) = uN_{0,0}(u) + (2 - u)N_{1,0}(u)$$

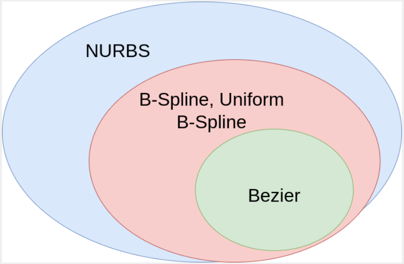
$$\vdots$$

$$N_{0,2}(u) = \frac{1}{2}(3 - u)^2$$



B-spline curve fitting: Properties

- **Bezier** curves **do not support local modification** and editing; however, **B-spline does support local editing**, i.e., $N_{i,p}$ is non-zero polynomial on $[u_i, u_{i+p+1}]$
- Relationship between B-Spine, NURBS, and Bezier

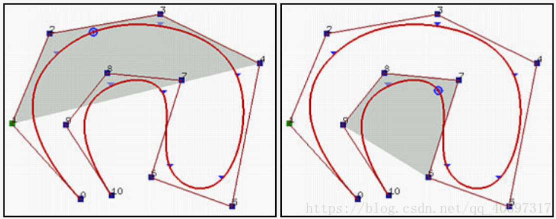


- $N_{i,p}(u)$ is a polynomial of degree p on u
- Non-negativity for all i, p and u $N_{i,p}(u)$ is non-negative

- Differentiability

$$\frac{d}{du}N_{i,p}(u) = (p-1)\left[\frac{N_{i,p-1}(u)}{u_{i+p}-u_i} - \frac{N_{i+1,p-1}(u)}{u_{i+p+1}-u_{i+1}}\right]$$

- Convex-hull property if u is in the interval u_i, u_{i+1}, then the curve that containing control points $P_{i-p}, P_{i-p+1}, ..., P_i$ form a convex hull



- Will not pass through its control points

- Generating the basis function table, knot vector with size n+p+1, where n number of control points, p is the order using **Clamped method**[1]
  Set the front and rear order +1 notes 4 to zero and one For example, if the control points are given by

$$0, \frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{4}{9}, \frac{5}{9}, \frac{6}{9}, \frac{7}{9}, \frac{8}{9}, 1.$$

  Then clamped list becomes : $0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1.$
- Or divide uniformly, e.g., 3rd-degree spline function, where knot vector can be defined as $t0 = -3\delta t, t1 = -2\delta t, t2 = -\delta t$ and so on

[1] Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight, Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu and Shaojie Shen, IEEE Robotics and Automation Letters (RA-L), 2019.

- Given time index t, calculate the corresponding position coordinate value or higher order values **Cox-DeBoor formula** [1]
- Let K be the number of control points, there are K-1 segments of trajectories, hence the domain of the 3-degree B-Spline curve is u_3, u_{3+k-1}, and there are a total of K+5 knot vectors, i.e., M=K-1+3+3, so there should be M-3 or K+2 control points

$$p(s(t)) = s(t)^\top M_{p_b+1} q_m$$

$$s(t) = [1\; s(t)\; s^2(t)\; \dots\; s^{p_b}(t)]^\top$$

$$q_m = [Q_{m-p_b}\; Q_{m-p_b+1}\; Q_{m-p_b+2}\; \dots Q_m]^\top$$

For a small curve defined on t_m, t_{m+1} on the B-Spline curve, p_{m-p_b}, p_m determined by these four control points, where $s(t) = \frac{t-t_m}{\delta t}$

[1] `https://en.wikipedia.org/wiki/De_Boor%27s_algorithm`

Matrix M_{p_b} is constant matrix

$$M^1 = [1], M^2 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, M^3 = \frac{1}{2!}\begin{bmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix}$$

$$M^4 = \frac{1}{3!}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}, M^5 = \frac{1}{4!}\begin{bmatrix} 1 & 11 & 11 & 1 & 0 \\ -4 & -12 & 12 & 4 & 0 \\ 6 & -6 & -6 & 6 & 0 \\ -4 & 12 & -12 & 4 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix}$$

K. Qin, “General matrix representations for b-splines,” The Visual Computer, vol. 16, no. 3, pp. 177–186, 2000

B-spline curve fitting: Implementation

- Calculate position constraints

$$x_i = \frac{1}{6}(Q_{i+1} - Q_i), i = 0, \ldots, K-1$$

- Calculate velocity and acceleration constraints It's required to calculate the derivative with respect to time. Since $s(t)$ is a function of t and has a constant term $\frac{1}{\delta t}$,first and second differentials are multiplied by $\frac{1}{\delta t}$ and $\frac{1}{\delta t^2}$, respectively.
- For K number of potion constraints, two velocity constraints, and two acceleration constraints, in total K+4 constraints, are needed for K+2 control points. Hence, the process of calculating B-spline is solving $Ax = b$, through matrix operation.

K. Qin, "General matrix representations for b-splines," The Visual Computer, vol. 16, no. 3, pp. 177–186, 2000

- Estimate velocity and acceleration

$$V_i = \frac{p_b(Q_{i+1} - Q_i)}{t_{i+p_b+1} - t_{i+1}},\ A_i = \frac{(p_b - 1)(V_{i+1} - V_i)}{t_{i+p_b+1} - t_{i+2}}$$

- Checking feasibility of velocity and acceleration

$$V'_i = \frac{1}{\mu_v}\frac{p_b(Q_{i+1} - Q_i)}{t_{i+p_b+1} - t_{i+1}} = \frac{1}{\mu_v}V_i$$

$$A'_i = \frac{1}{\mu_a}\frac{(p_b - 1)(\frac{1}{\mu_a}V'_{i+1} - \frac{1}{\mu_a}V'_i)}{t_{i+p_b+1} - t_{i+2}} = \frac{1}{\mu_a^2}\frac{(p_b - 1)(V_{i+1} - V_i)}{t_{i+p_b+1} - t_{i+2}}$$