JRL302: Robotics Technologies

Lecture 4: Introduction to Splines

Mayank Sewlia
sewlia@iitd.ac.in

Q1. How is dynamic programming different from direct transcription?

DP solves the problem by backward recursion on the value function, giving a policy from every state! Whereas direct transcription turns the problem into one large NLP over states and controls and returns a single local trajectory from a given initial state.

Q2. Doesn't numerical error accumulate in transcription as well as shooting?

Both have per-step truncation error, but shooting compounds it along the whole integration (and amplifies sensitivity to early controls), while transcription enforces dynamics as local constraints, so the error stays local and the problem is better conditioned.

Q3. When should we use DP rather than direct optimisation?

DP is useful when the state dimension is low (about 5 or fewer) and you need a feedback policy, a global optimum, or have non-smooth or hybrid dynamics (or LQR); otherwise use direct methods for higher dimensions, a single initial state and smooth problems.

Q4. Can PMP handle bounded controls?

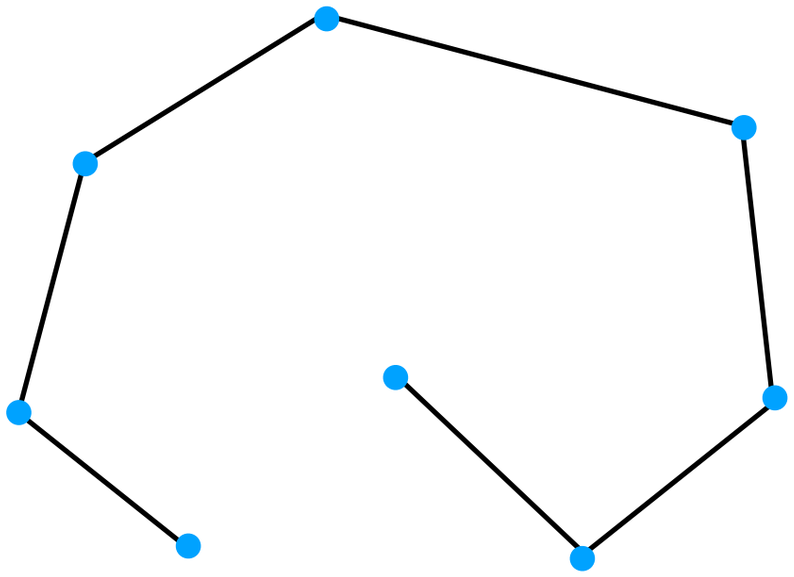
$$\dot{x} = \frac{\partial H}{\partial \lambda} \qquad \text{(State equation)}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x} \qquad \text{(Costate equation)}$$

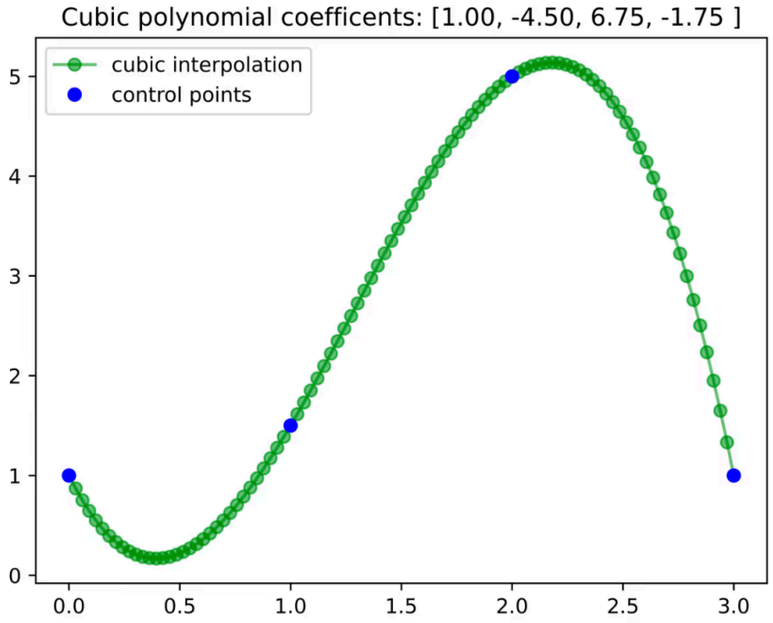
$$u^* \in \arg\min_{u \in \mathcal{U}} H(x, u, \lambda) \qquad \text{(Minimum condition)}$$

$$u^*(t) = \mathrm{clip}(-\lambda_v(t), -u_{\max}, u_{\max})\,.$$

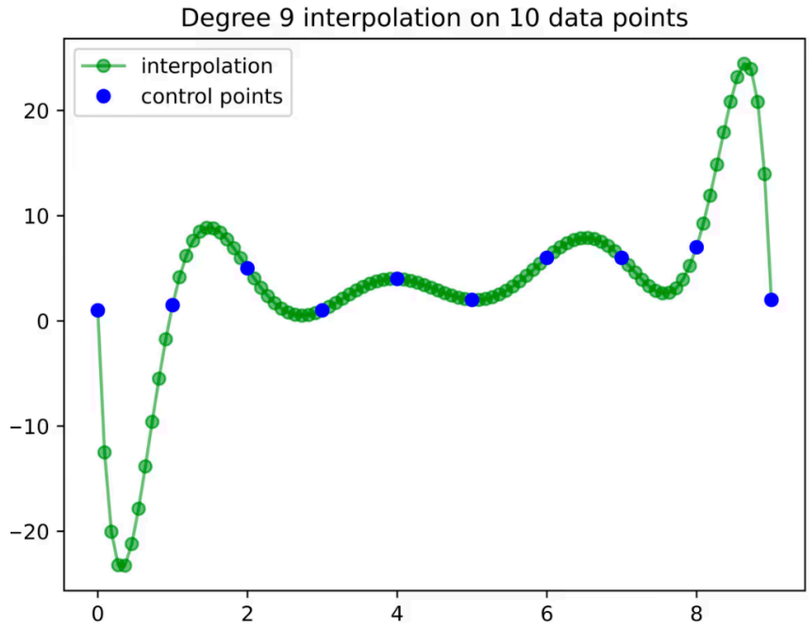
Why Splines?



Piecewise Linear Trajectory

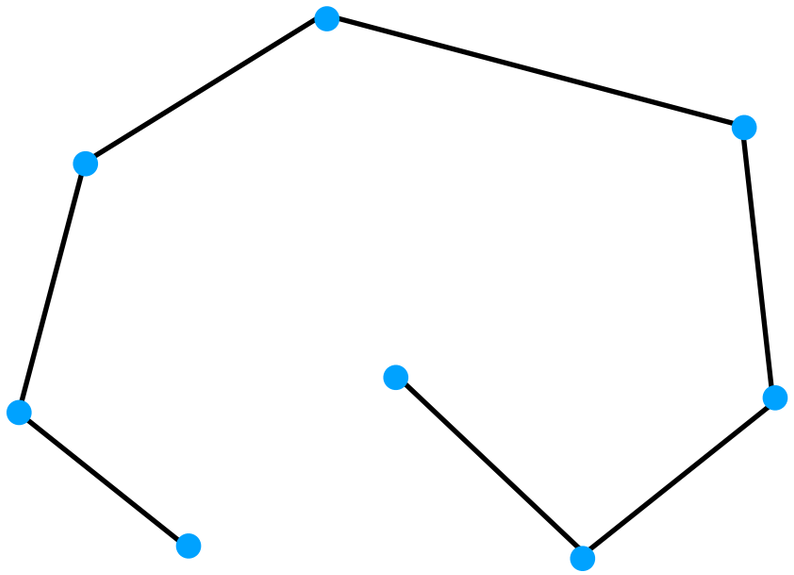


Smooth Spline Trajectory

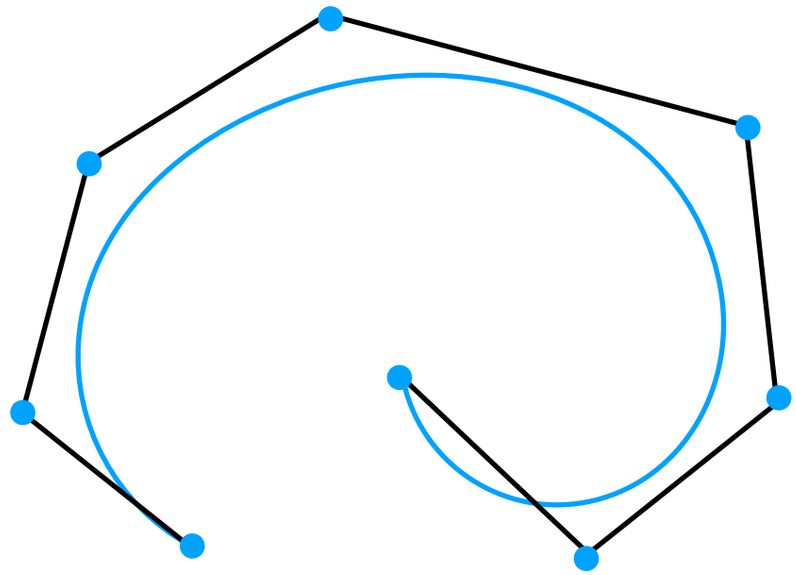


- Position is continuous but velocity discontinuous at corners
- High-degree polynomial can interpolate but introduces global coupling and oscillations
- Low degree polynomials offer local control and smoothness

Why Splines?



Piecewise Linear Trajectory



Smooth Spline Trajectory

- Position is continuous but velocity discontinuous at corners
- High-degree polynomial can interpolate but introduces global coupling and oscillations
- Low degree polynomials offer local control and smoothness

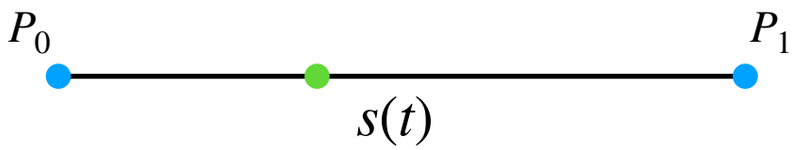
General representation of splines

$$s(t) = \sum_{i=0}^{n} P_i B_i(t)$$

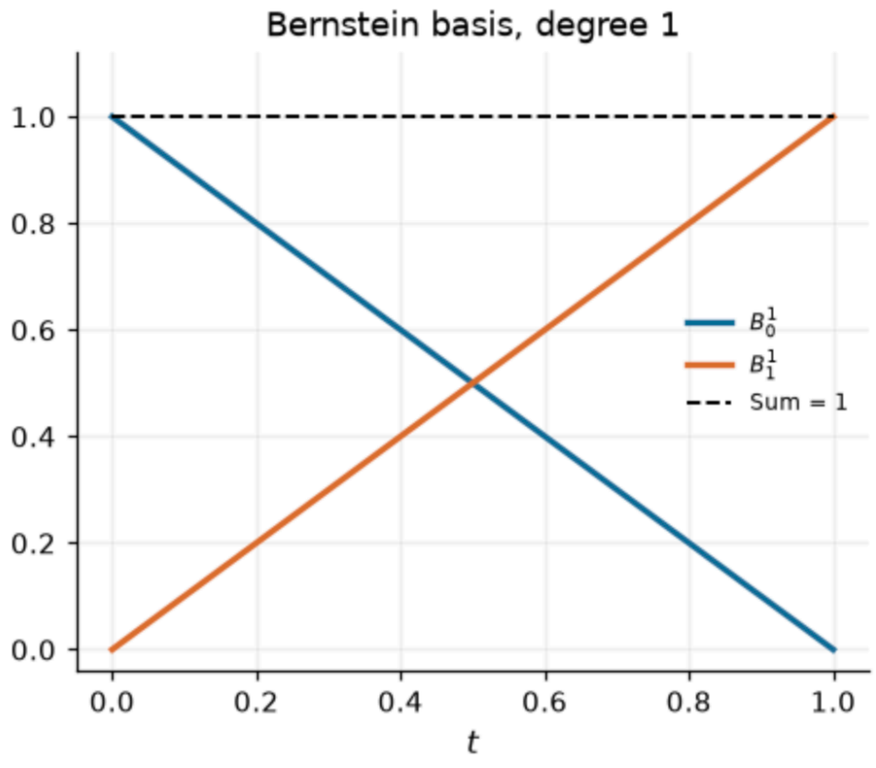
P_i Coefficients or control points

B_i Basis functions

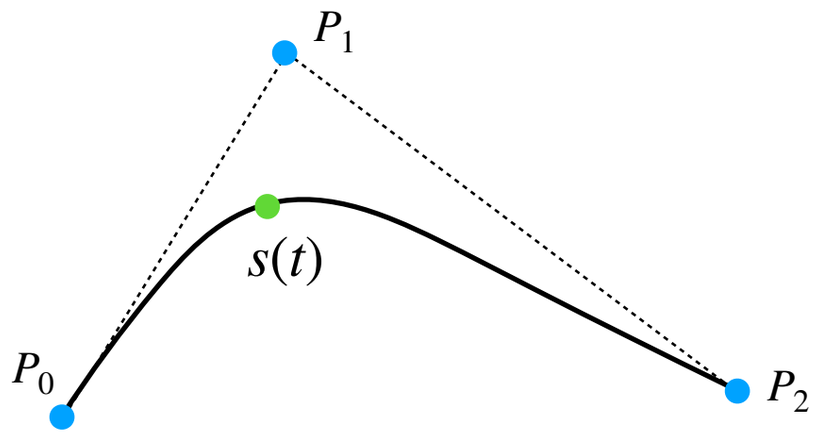
Bernstein Polynomials: Degree 1



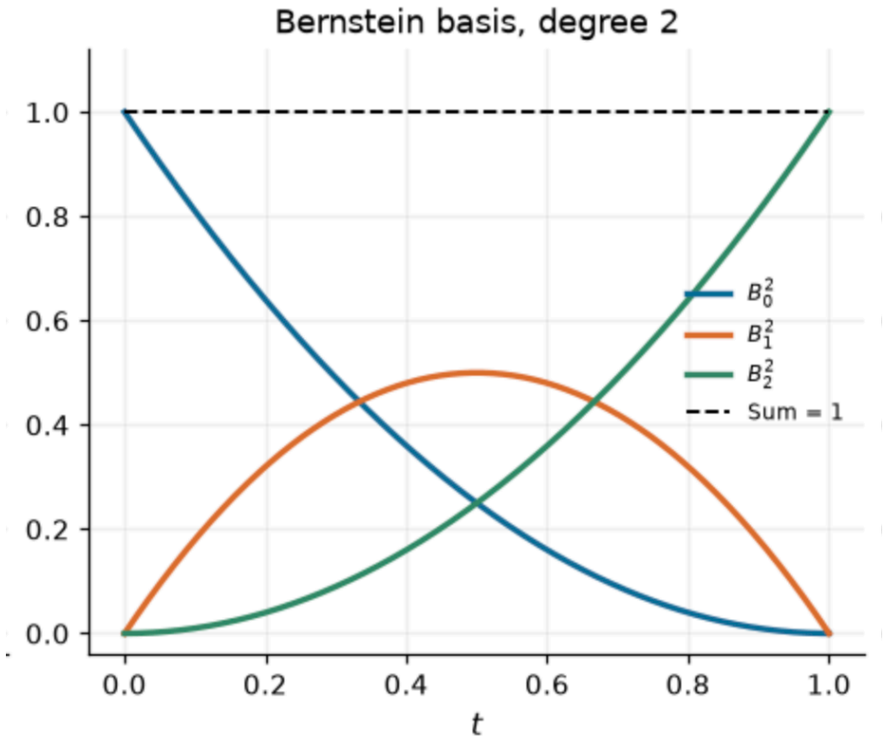
$$s(t) = (1-t)P_0 + tP_1$$



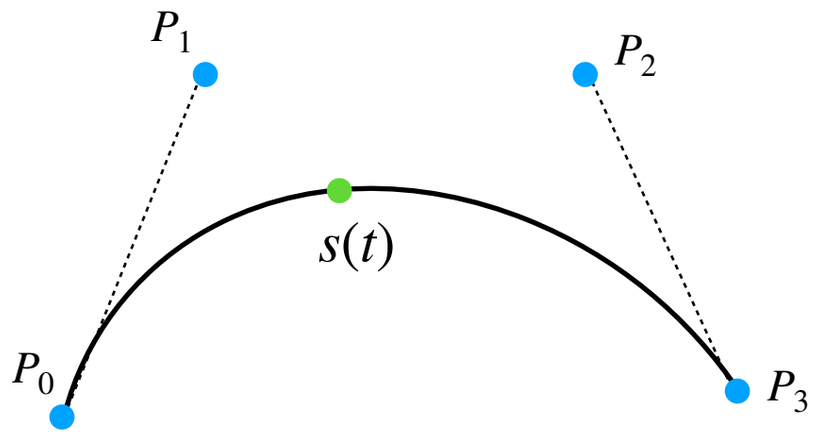
Bernstein Polynomials: Degree 2



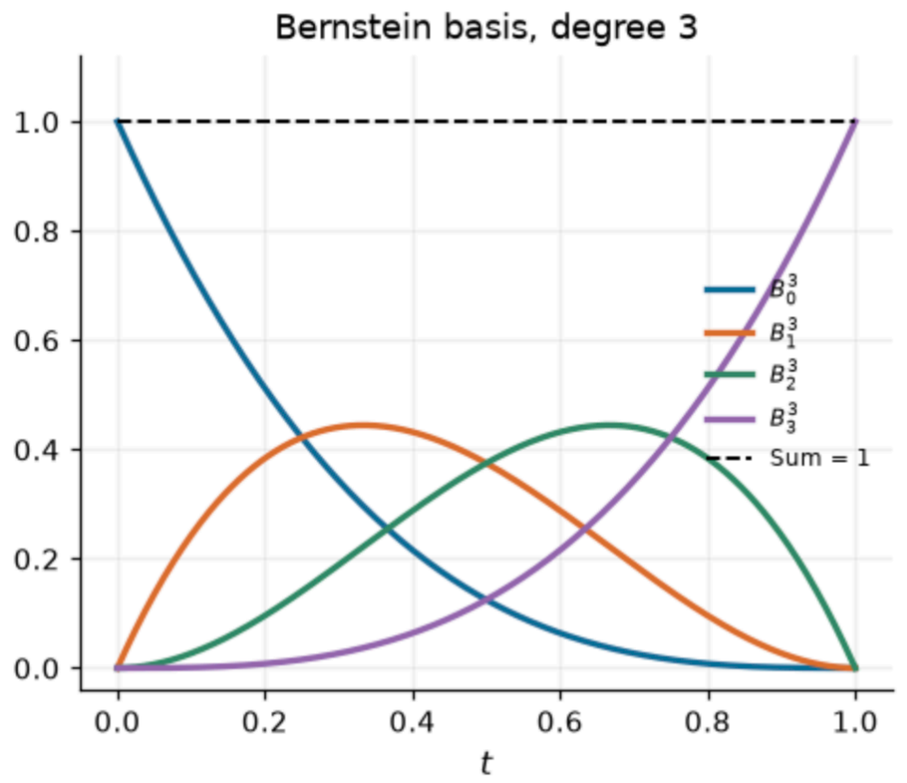
$$s(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$



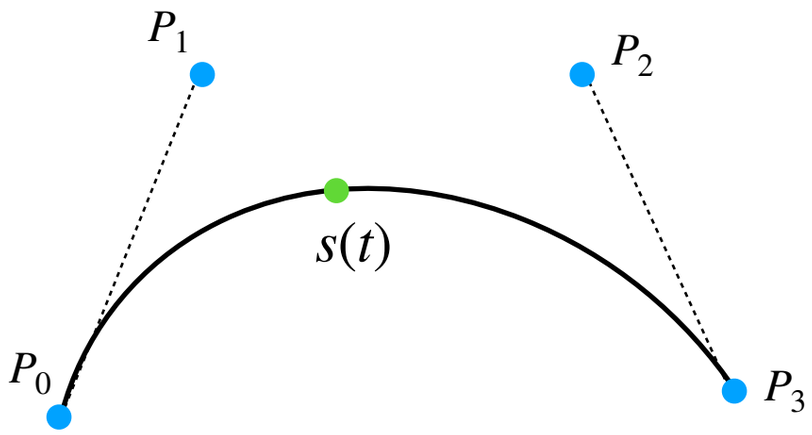
Bernstein Polynomials: Degree 3



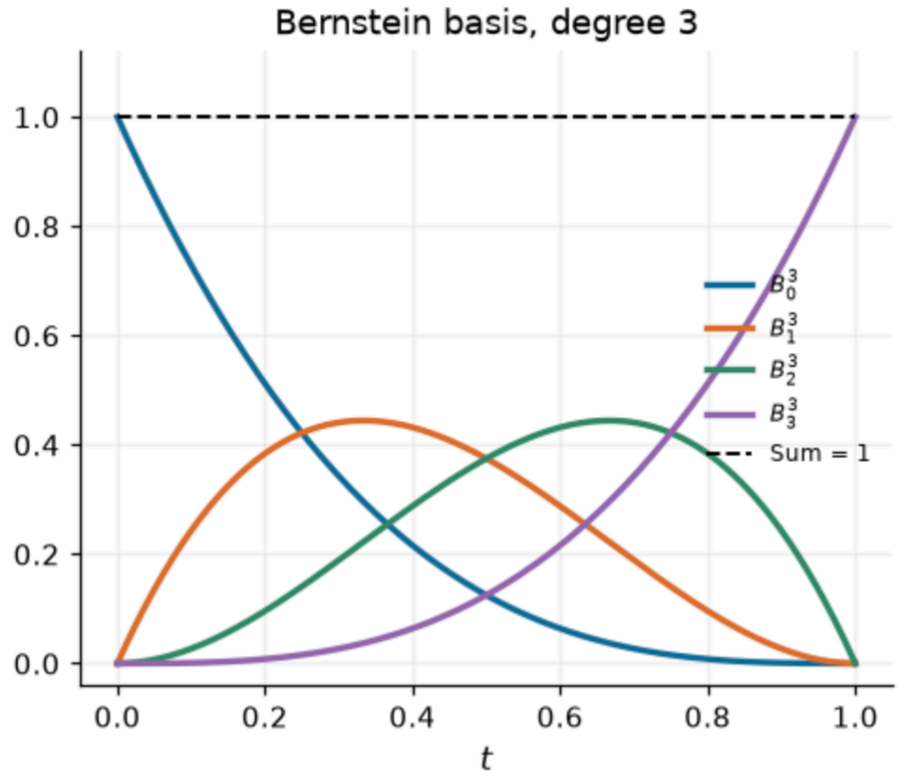
$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$



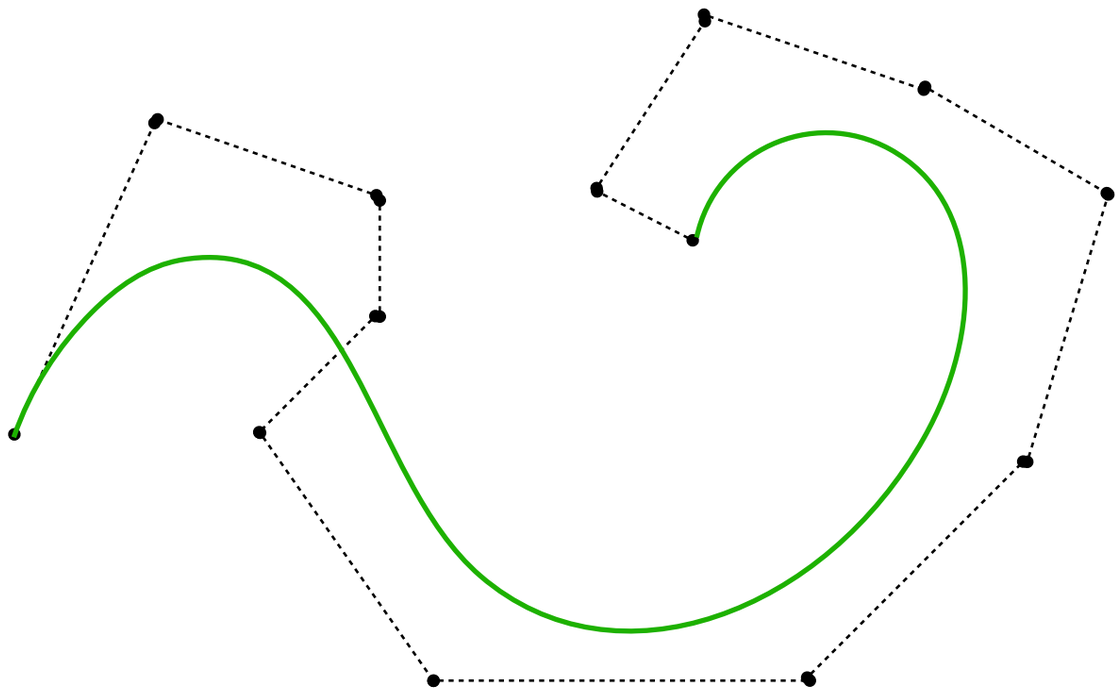
Bernstein Polynomials: Degree 3

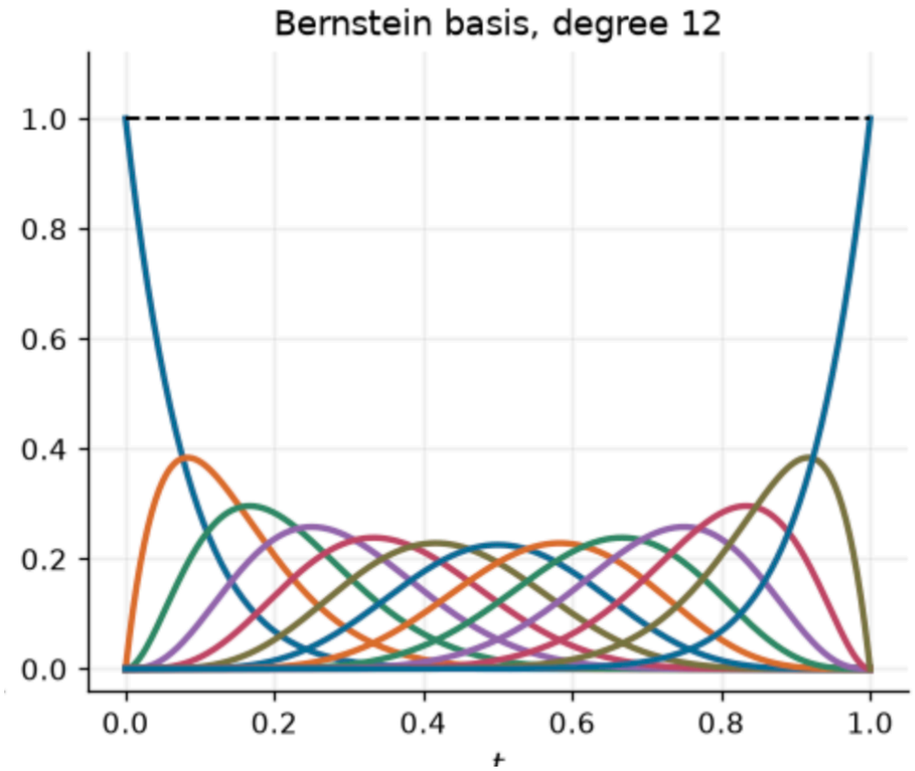


$$\mathbf{P}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}$$

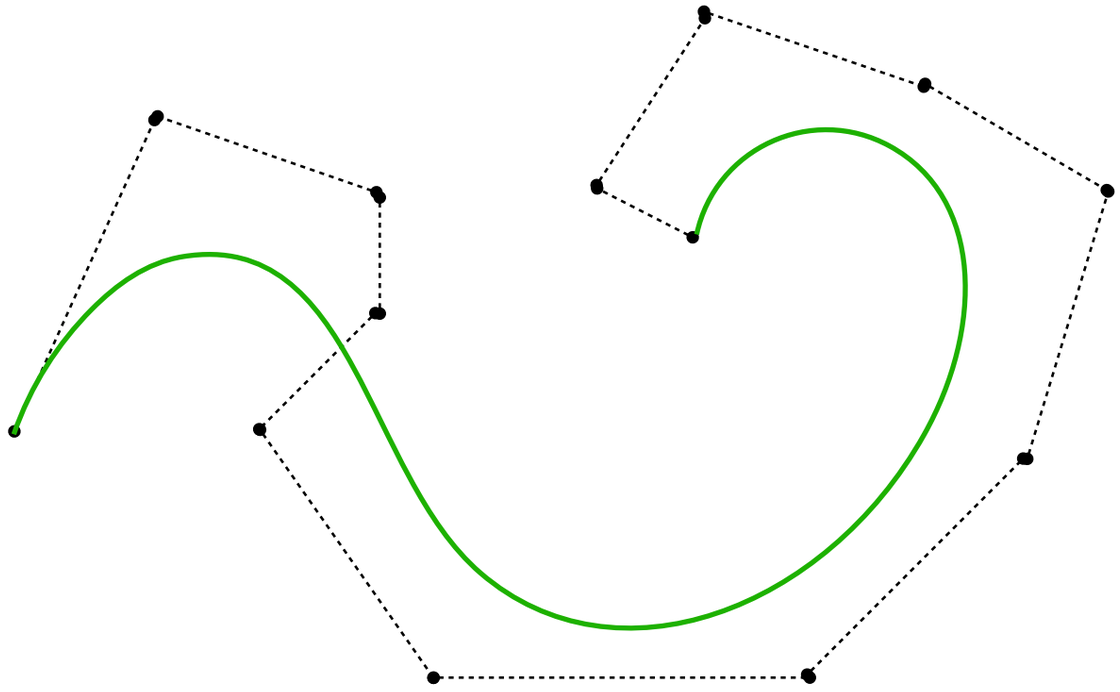


Bernstein Polynomials: Degree 12

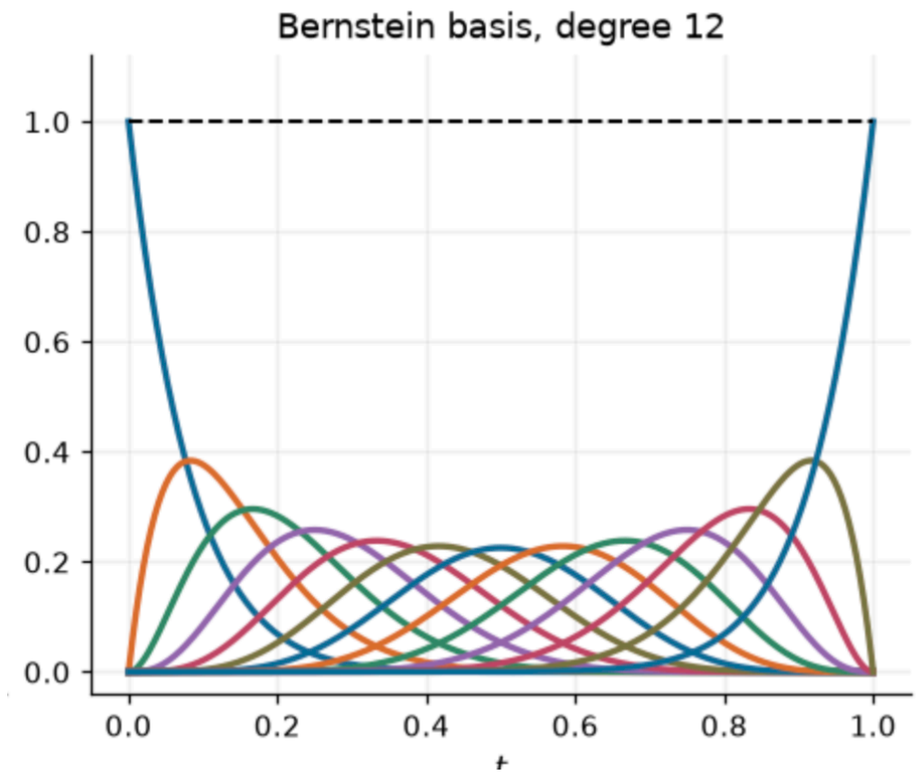




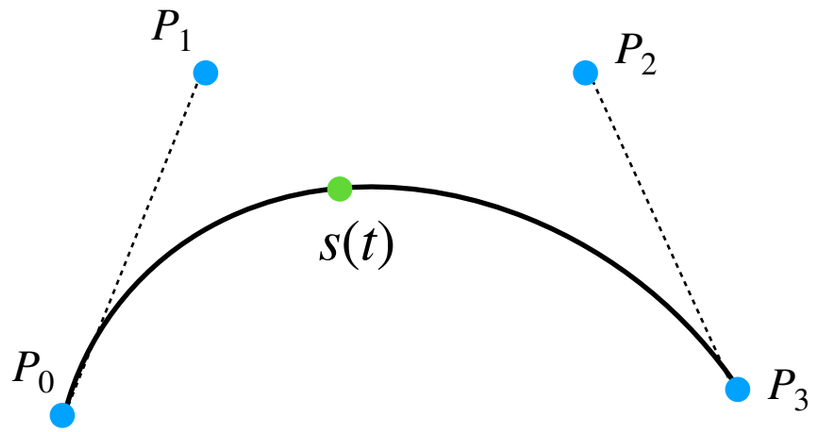
Bernstein Polynomials: Degree 12



- No local control
- Doesn't actually pass through most points
- Unstable/Expensive to calculate

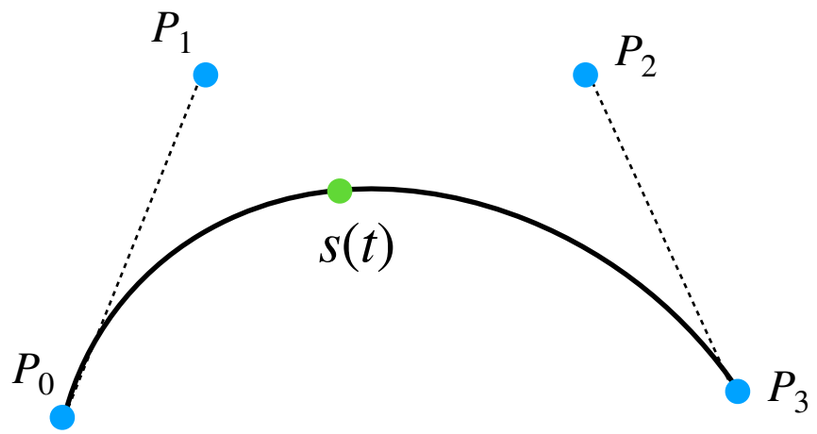


Bezier Splines: Chaining cubic Bernstein polynomials

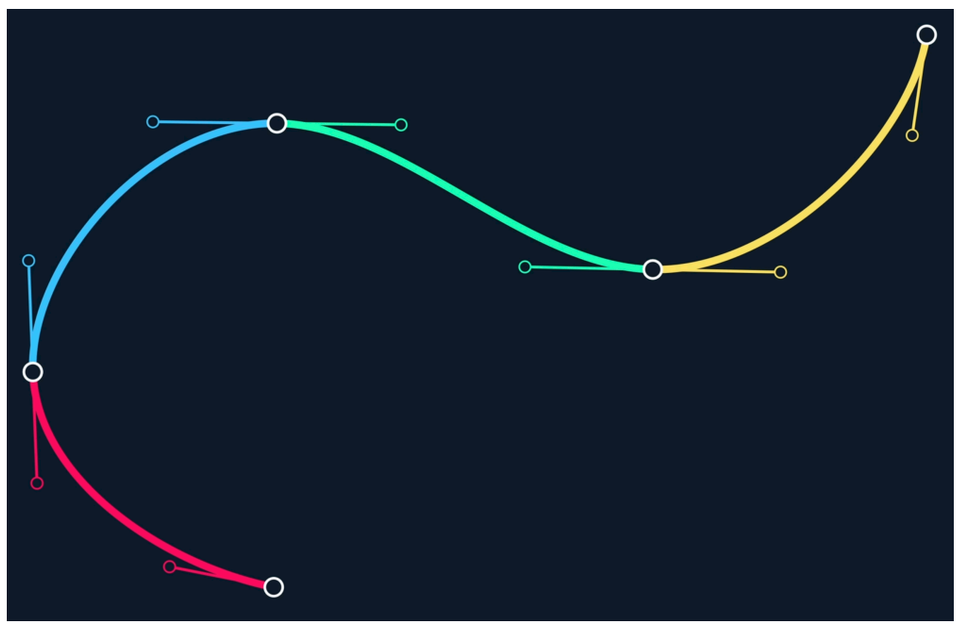


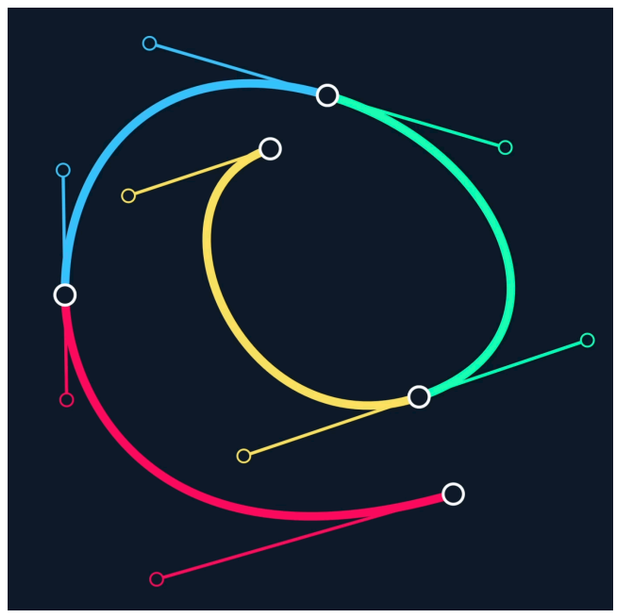
$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Bezier Splines: Chaining cubic Bernstein polynomials



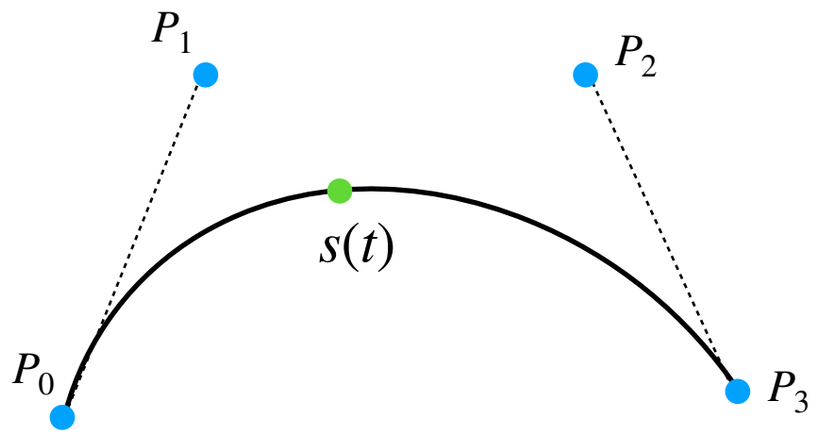
$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$



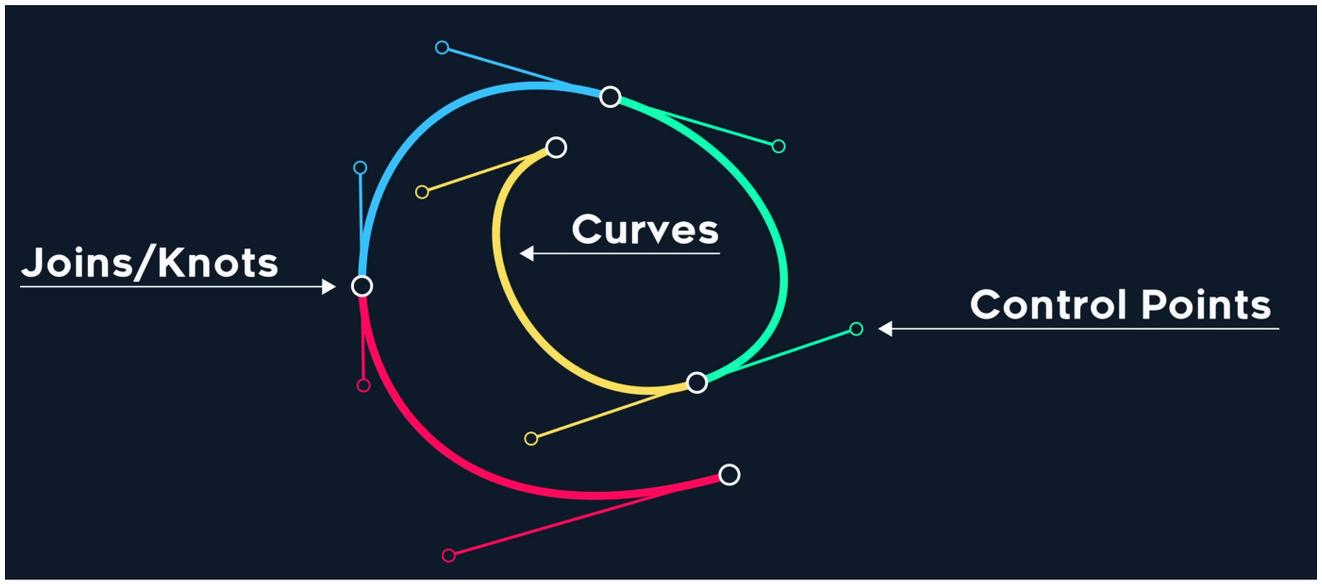




Bezier Splines: Chaining cubic Bernstein polynomials

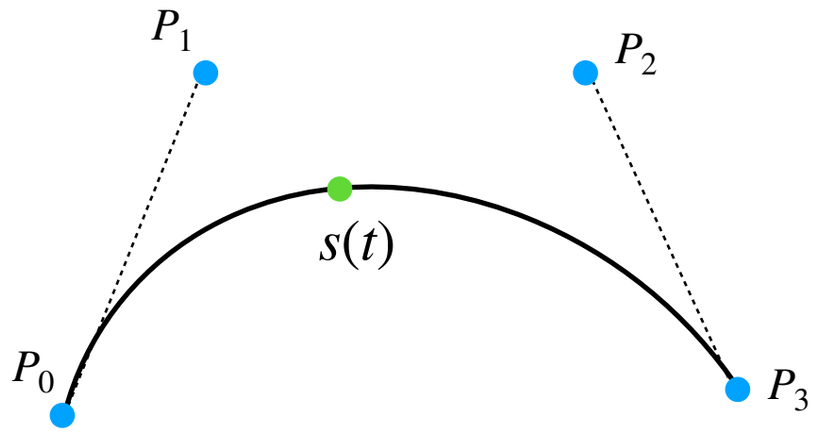


$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$



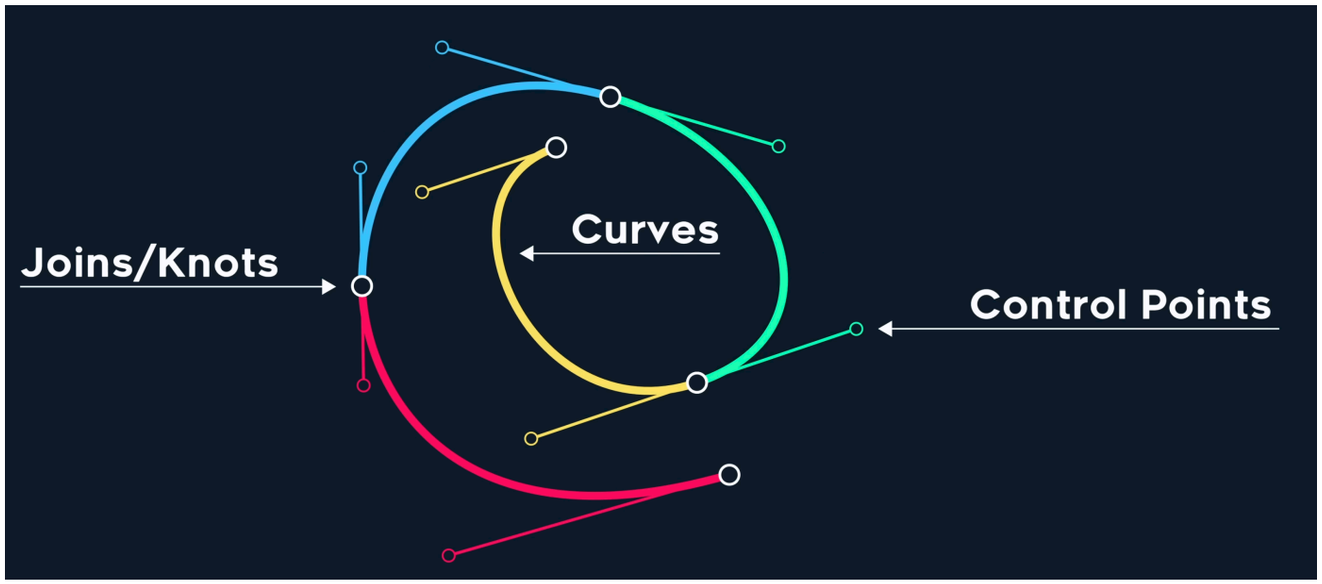


Bezier Splines: Chaining cubic Bernstein polynomials



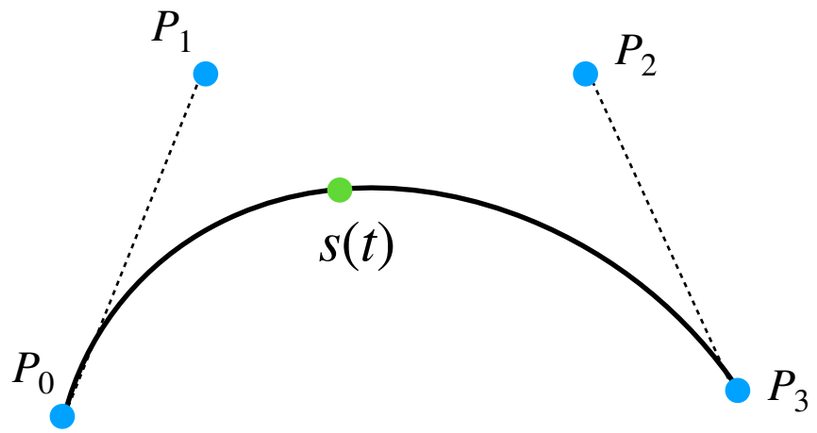
$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

- No continuity built-in
- C^1 continuity with collinear points
- Local control
- Cheaper computationally
- Interpolating every 3rd point



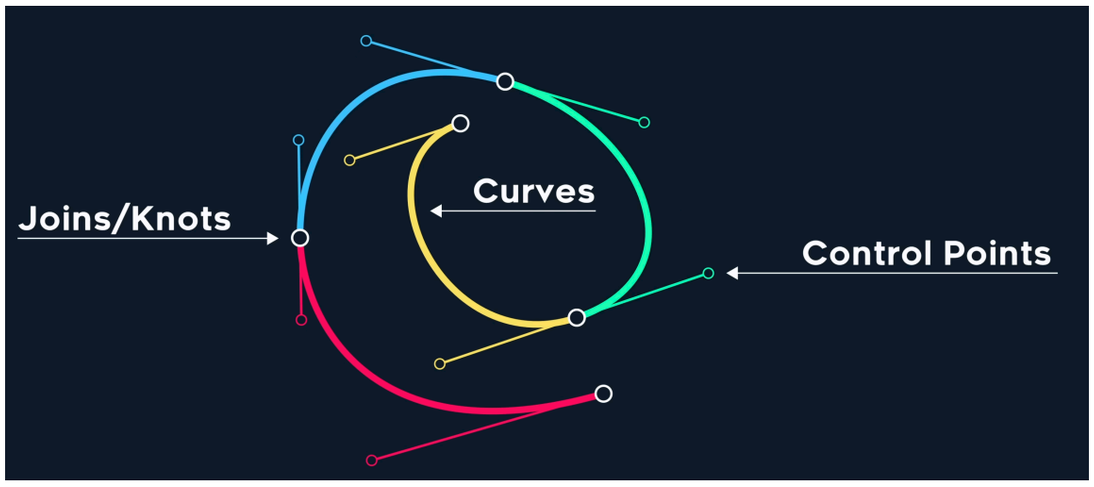


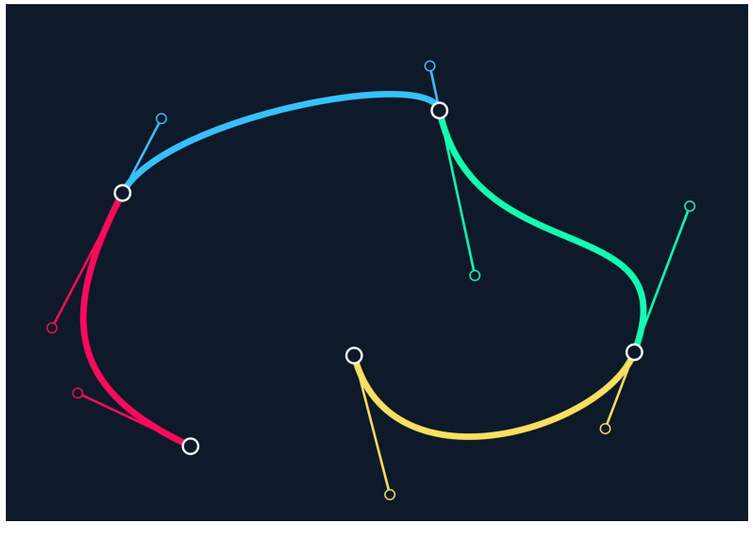
Bezier Splines: Chaining cubic Bernstein polynomials



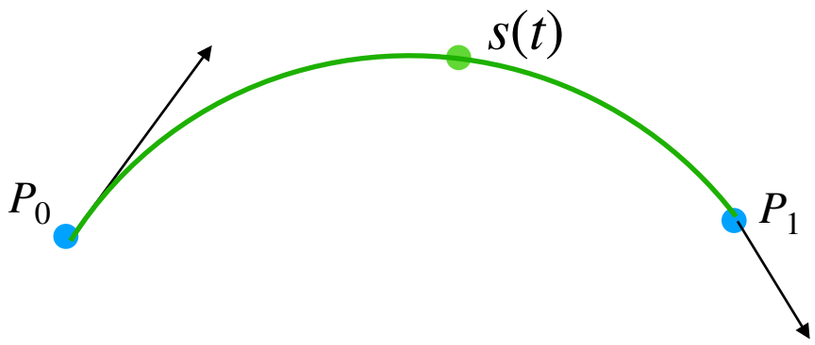
$$s(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

- No continuity built-in
- C^1 continuity with collinear points
- Local control
- Cheaper computationally
- Interpolating every 3rd point



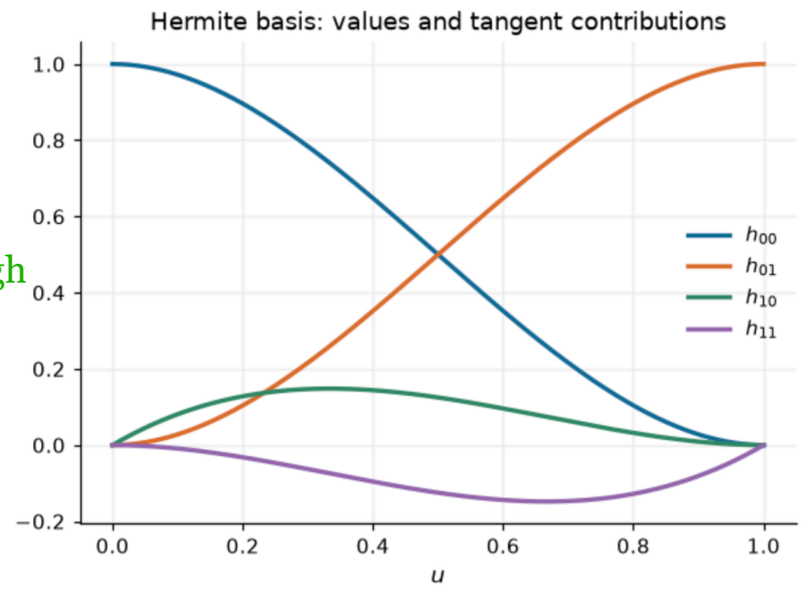


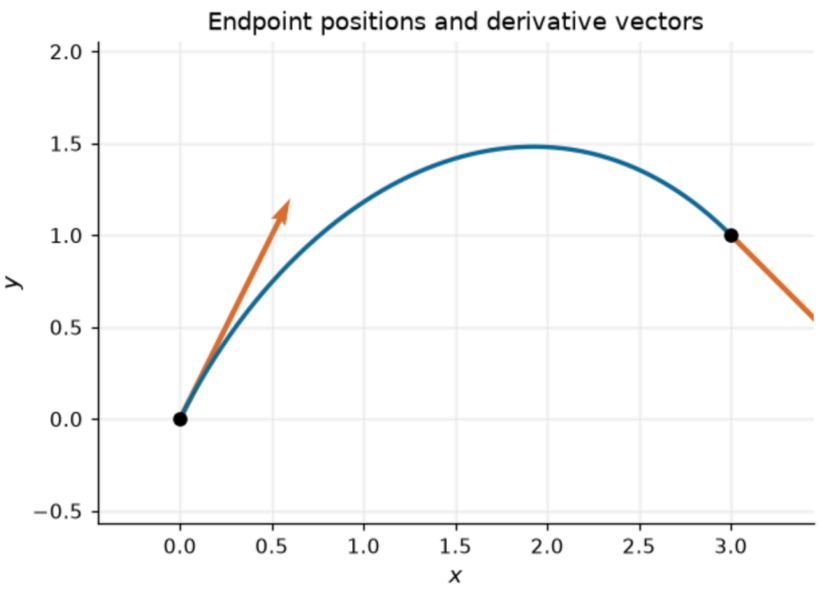
Hermite Curves



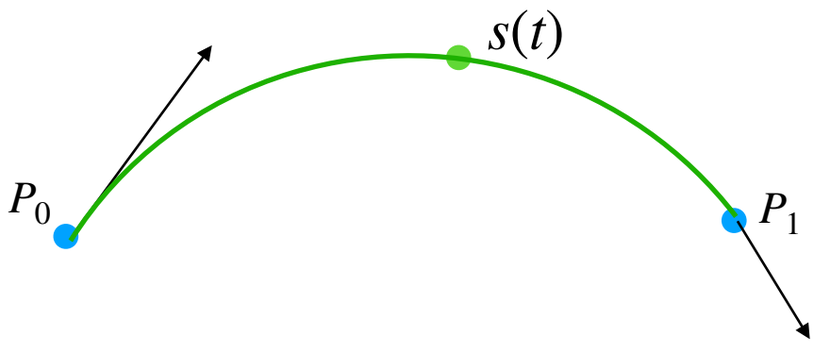
$$s(t) = h_{00}\mathbf{P}_0 + h_{01}\mathbf{P}_1 + h_{10}\mathbf{T}_0 + h_{11}\mathbf{T}_1$$

- C^1 continuous
- Explicitly passes through points





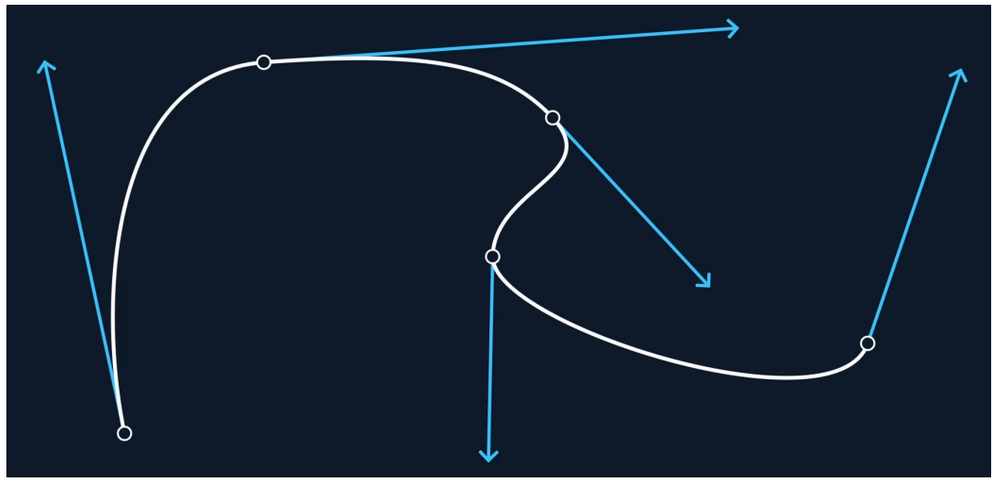
Hermite Curves

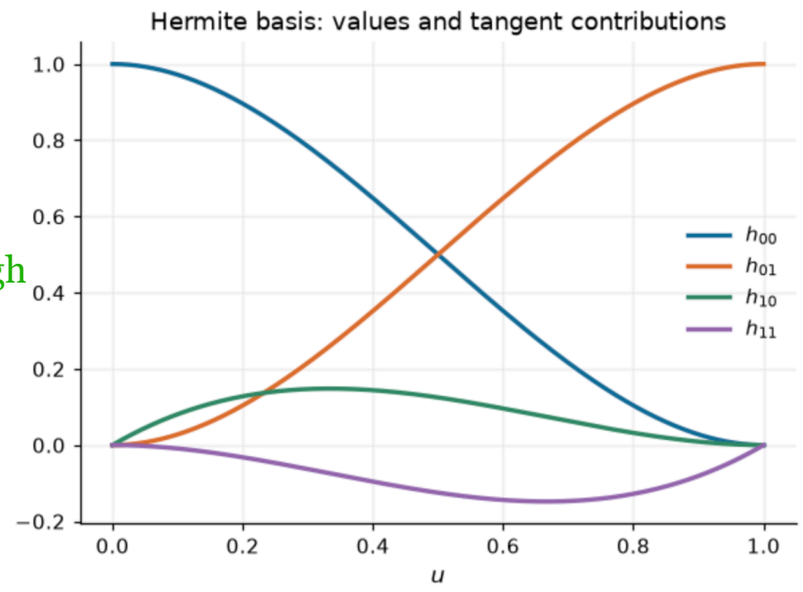


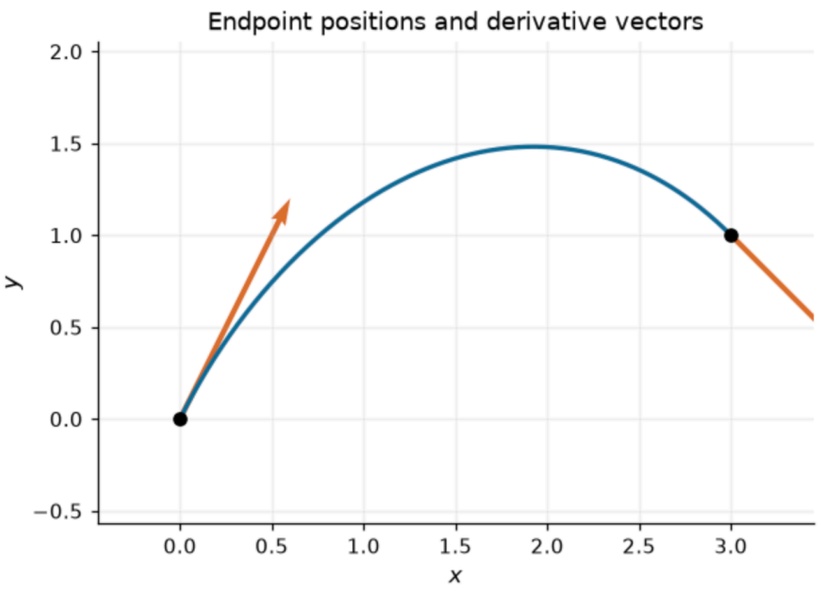
$$s(t) = h_{00}\mathbf{P}_0 + h_{01}\mathbf{P}_1 + h_{10}\mathbf{T}_0 + h_{11}\mathbf{T}_1$$

- C^1 continuous
- Explicitly passes through points

Hermite Splines





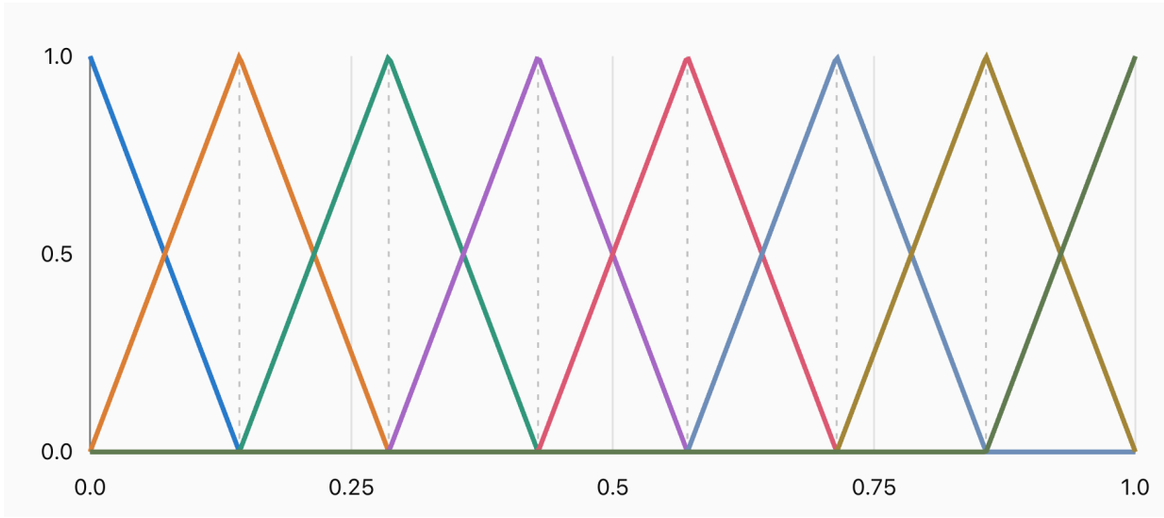


B-Splines: In search for C^2 continuous splines

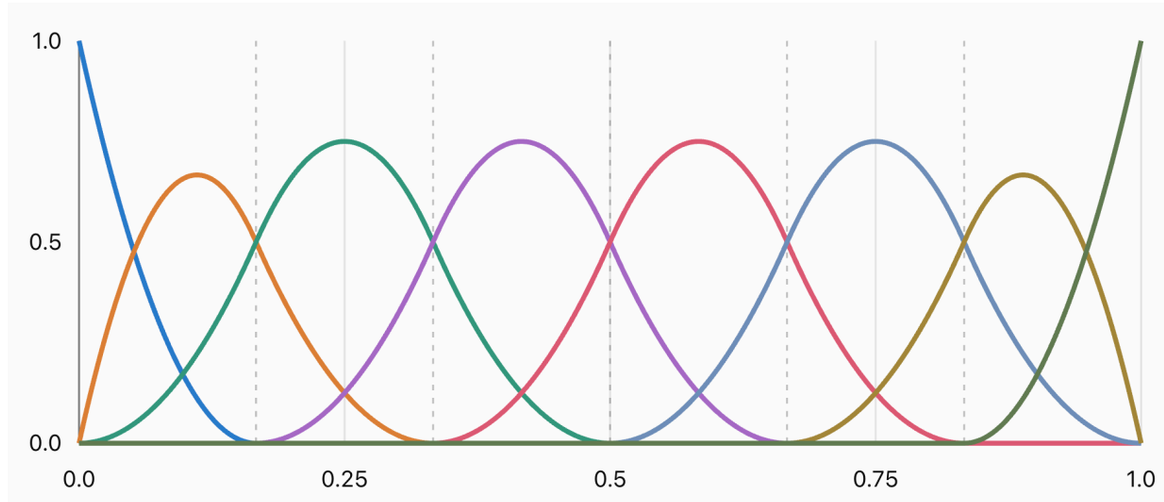
$$s(t) = = \sum_{i=0}^{n} N_{i,p}(t) P_i$$

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

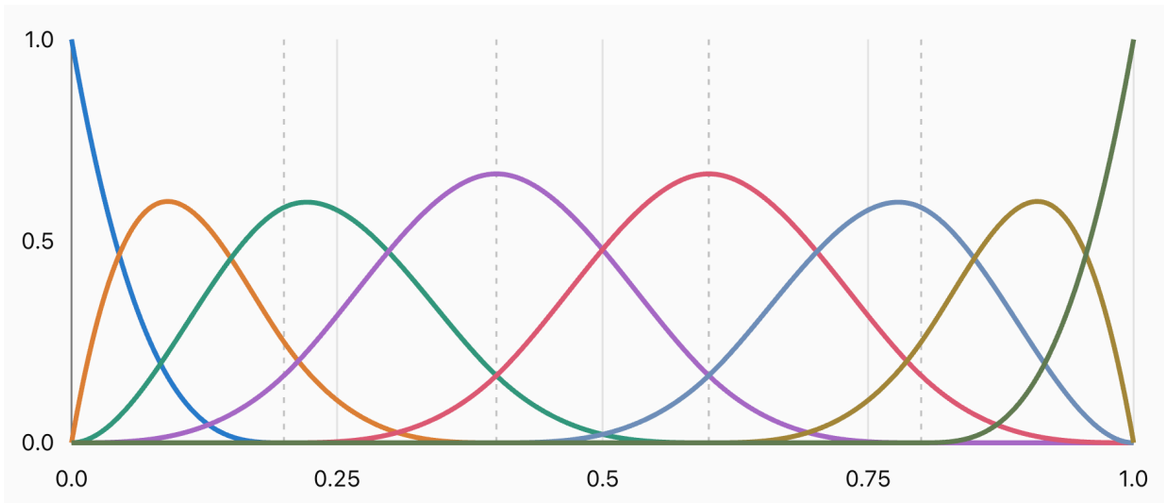
Linear



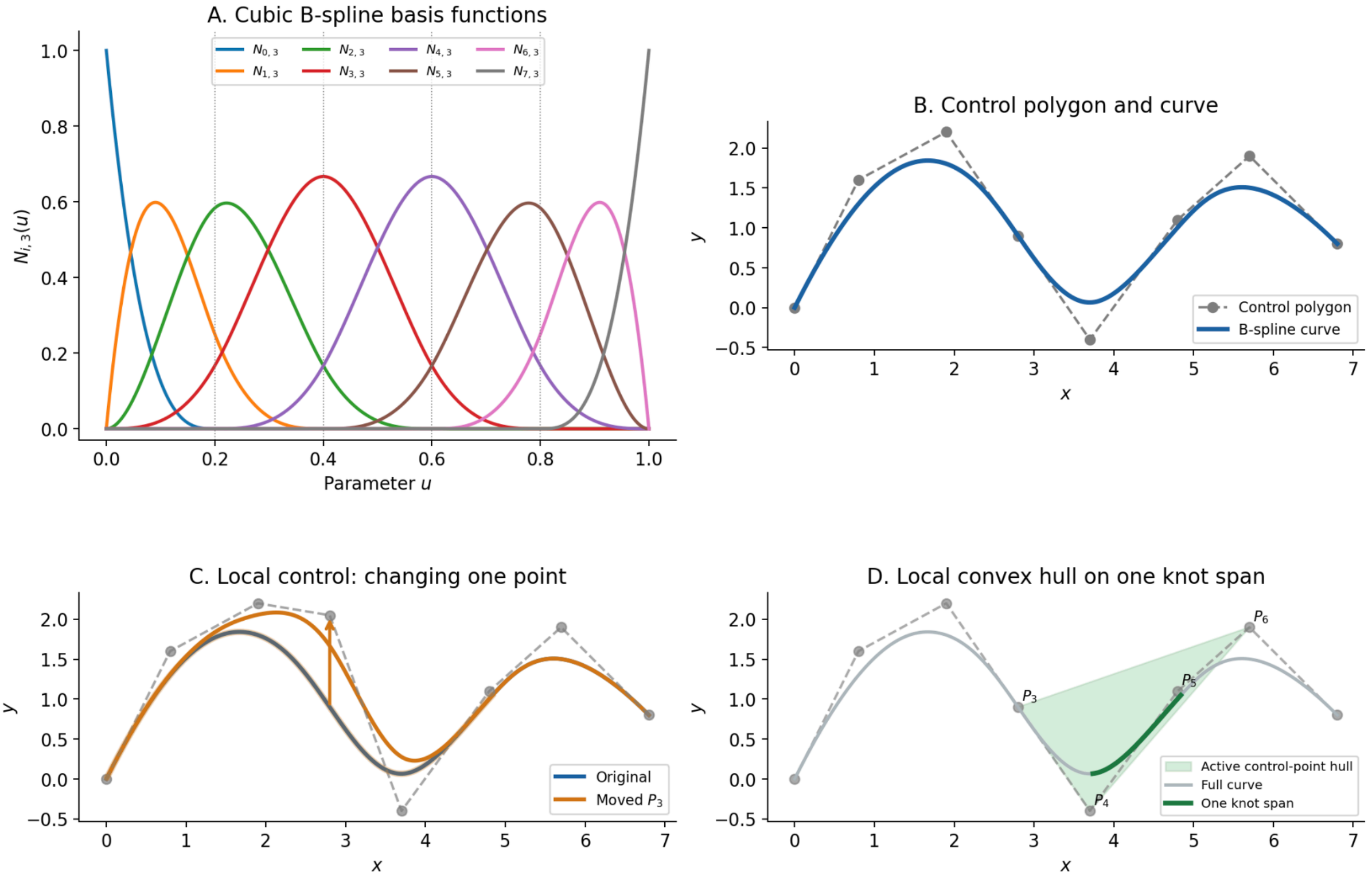
Quadratic



Cubic



Cubic B-Splines



Cubic B-Splines

Property 1: Local support

Modifying a single control point P_i alters the curve strictly within a localised region rather than affecting the whole parameter interval.

$\Delta s(u) = N_{i,p}(u)\Delta P_i$

Property 2: Partition of Unity

The basis functions sum to 1 and are non-negative, ensuring affine invariance so that transforming control points transforms the entire curve accordingly

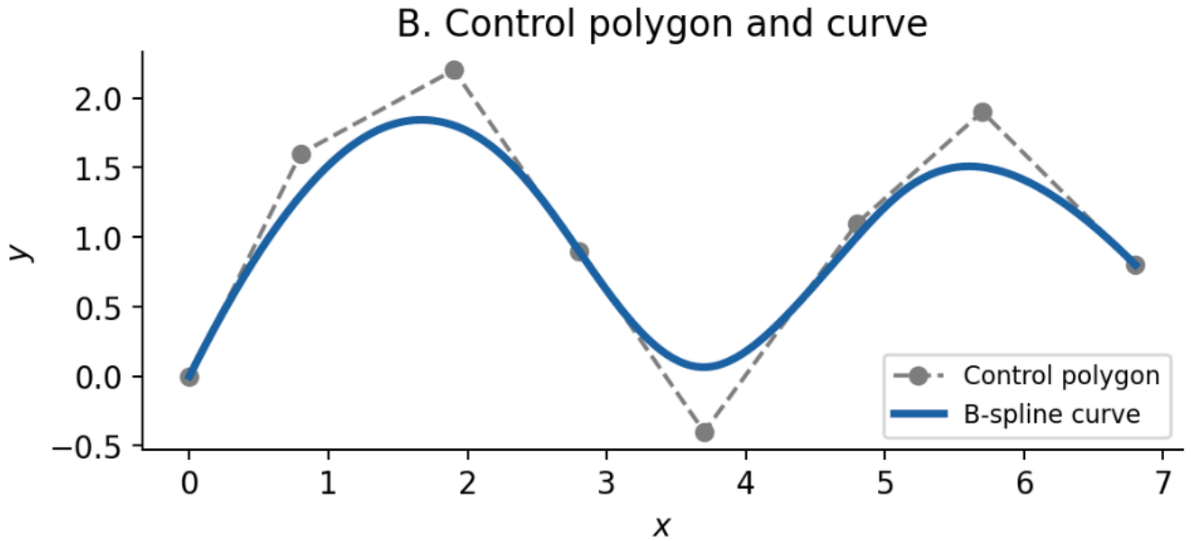
$\sum_i N_{i,p}(u) = 1$

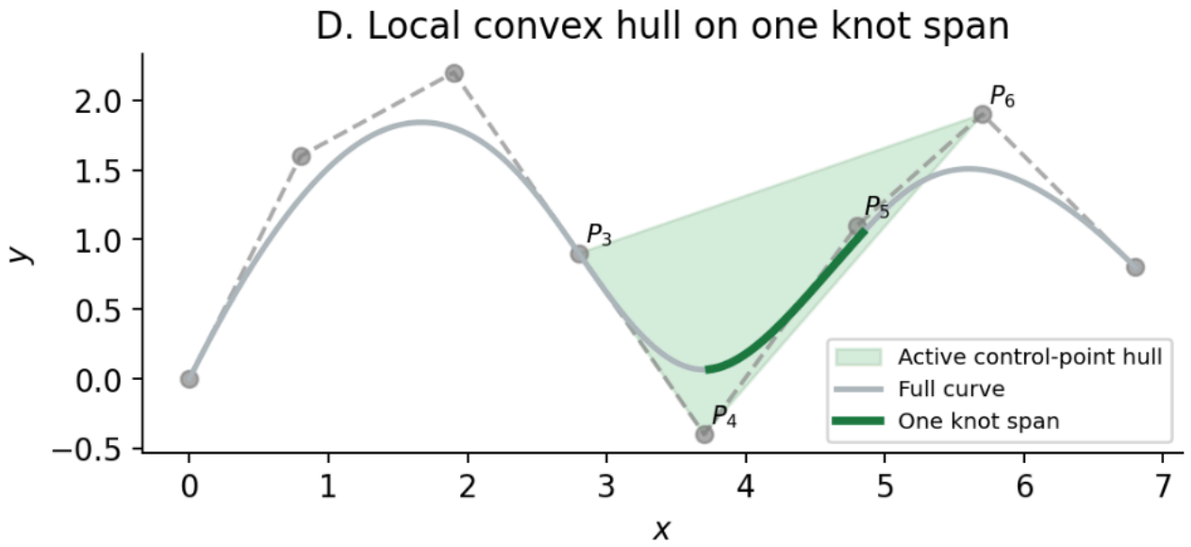
Property 3: Convex Hull

On any knot span $t \in [t_j, t_{j+1})$, the curve segment is strictly contained within the convex hull of its $p + 1$active control points.

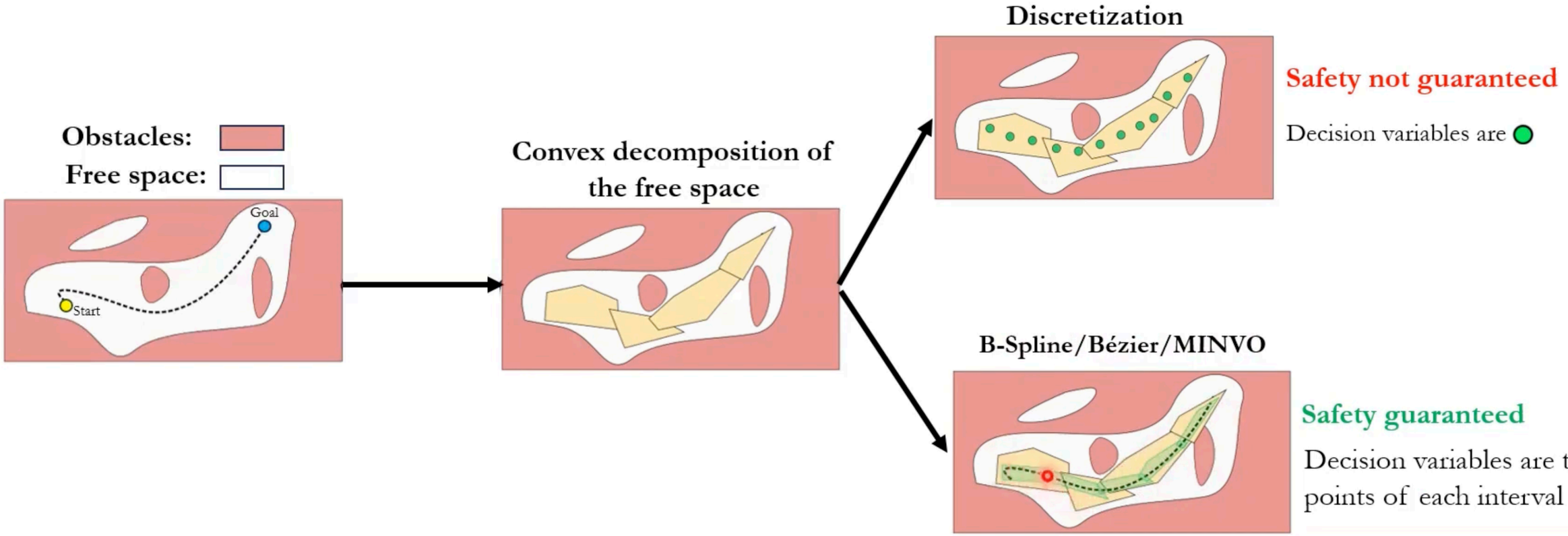
$s(t) \in \text{conv}\{P_{j-p}, \ldots, P_j\}$

Property 4: C^2 Continuity

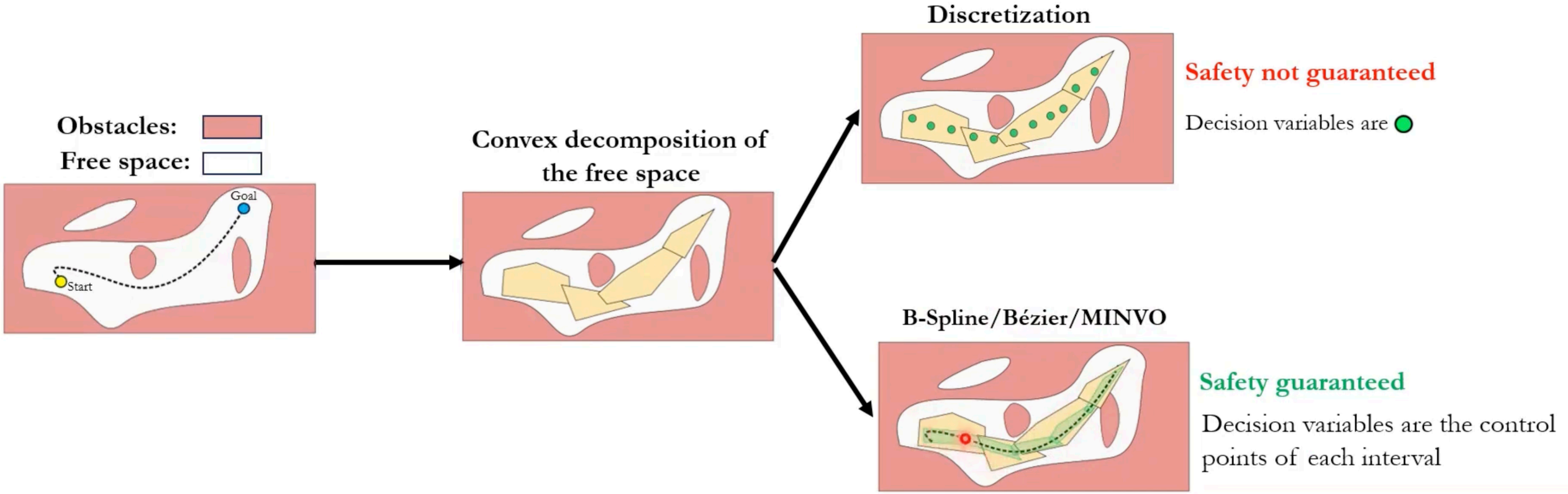




Use in Obstacle Environment



Use in Obstacle Environment



$$
\begin{aligned}
\min_{q(\cdot)} \quad & \int_0^T \|\ddot{q}(t)\|^2 \, dt \\
\text{s.t.} \quad & q(0) = q_{\text{start}}, \\
& q(T) = q_{\text{goal}}, \\
& q(t) \in \mathscr{C}_{\text{free}}, \\
& |\dot{q}_j(t)| \le v_{j,\max}, \\
& |\ddot{q}_j(t)| \le a_{j,\max}.
\end{aligned}
\quad \longrightarrow \quad
\begin{aligned}
\min_{\mathbf{P}} \quad & \mathbf{P}^\top (Q \otimes I_d) \mathbf{P} \\
\text{s.t.} \quad & P_0 = q_{\text{start}}, \\
& P_n = q_{\text{goal}}, \\
& A_i P_j \le b_i \quad \text{for relevant control points,} \\
& |\dot{q}_k(t)| \le v_{k,\max}, \\
& |\ddot{q}_k(t)| \le a_{k,\max}.
\end{aligned}
$$

Collocation

- Spline Approximations: Approximate the state and control using chosen spline bases:

$$x_h(t) = \sum_{i=0}^{n_x} P_i N_{i,p}(t) \quad \text{and} \quad u_h(t) = \sum_{j=0}^{n_u} U_j M_{j,r}(t)$$

where P_i and $U_j$are the optimisation variables.

- State Derivative: Calculate the time derivative of the state approximation using the derivative of the basis functions:

$$\dot{x}_h(t) = \sum_i P_i N'_{i,p}(t)$$

- Dynamics Residual: Define the residual function to measure how much the approximation deviates from the system dynamics:

$$r(t) = \dot{x}_h(t) - f(x_h(t), u_h(t))$$

- Collocation Constraints: Choose a finite set of collocation times $(t_0, t_1, \ldots, t_K)$and enforce the dynamics condition at each point to form nonlinear algebraic constraints:

$$\sum_i P_i N'_{i,p}(t_k) = f\left(\sum_i P_i N_{i,p}(t_k), \sum_j U_j M_{j,r}(t_k) \right)$$

- Nonlinear Programme Formulation: Convert the continuous problem into the final optimisation formulation:

$$\begin{aligned}
\min_{\mathbf{P},\mathbf{U}} \quad & \sum_{k=0}^{K} w_k L(x_h(t_k), u_h(t_k)) + \Phi(x_h(T)) \\
\text{s.t.} \quad & \dot{x}_h(t_k) = f(x_h(t_k), u_h(t_k)), \\
& k = 0,\ldots, K, \\
& x_h(0) = x_0, \\
& x_h(T) \in \mathcal{X}_f, \\
& \text{state and control constraints .}
\end{aligned}$$