Numerisches Programmieren (IN0019)

Frank R. Schmidt

Winter Semester 2016/2017

5. Interpolation

Interpolation

Motivation

Bis jetzt sind wir immer davon ausgegangen, dass eine Funktion $f\colon \mathbb{R} \to \mathbb{R}$ bekannt ist und dass wir lediglich die Berechnung von $f(x)$ durchführen wollen.

Im Folgenden wollen wir $f(x)$ berechnen, ohne f komplett zu kennen. Stattdessen sind für gewisse **Stützstellen** $x_0 < \ldots < x_n$ die Funktionswerte $y_i = f(x_i)$ bekannt.

Sind die Stützstellen und Funktionswerte gegeben, so nennt man die Schätzung von $f(x)$

- **Interpolation**, wenn $x \in [x_0; x_n]$
- **Extrapolation**, wenn $x \notin [x_0; x_n]$

Dieses Szenario ist sehr realistisch in Situationen in denen Sensoren benutzt werden und wir daher nur wenige Messwerte haben.

Bildinterpolation (Beispiel)

Stückweise Konstant

Glatte Interpolation

Funktionen-Vektorraum

Die Menge $\mathrm{Abb}(\mathbb{R}, \mathbb{R})$ der Funktionen $f\colon \mathbb{R} \to \mathbb{R}$ bildet einen $\mathbb{R}$-Vektorraum, zusammen mit den Operationen

$$\begin{aligned} (f+g)(x) &= f(x) + g(x) && \text{(Addition)} \\ (\lambda \cdot f)(x) &= \lambda \cdot f(x) && \text{(Skalar-Multiplikation)} \end{aligned}$$

Um Interpolationen durchzuführen, definieren wir linear unabhängige **Basisfunktionen** $g_0, \ldots, g_n$. Das Interpolationspoblem besteht dann darin, Parameter $\lambda_0, \ldots \lambda_n \in \mathbb{R}$ zu finden, so dass für $g = \sum_{j=0}^{n} \lambda_j \cdot g_j$ die **Interpolationsbedingungen** erfüllt sind

$$g(x_i) = f(x_i) = y_i \qquad \text{für } i = 0, \ldots, n$$

Wir müssen also $n+1$ Gleichungen mit $n+1$ Unbekannten lösen.

Allgemeine Problemstellung

Gegeben seien **Punktepaare** (x_i, y_i) für $i = 0, \ldots, n$, wobei die Stützstellen x_i paarweise verschieden sind. Darüber hinaus sind $n+1$ linear unabhängige Funktionen $g_0, \ldots, g_n \in \mathrm{Abb}(\mathbb{R}, \mathbb{R})$ gegeben.

Gesucht sind Koeffizienten λ_j für $j = 0, \ldots, n$, so dass

$$\underbrace{\begin{pmatrix} g_0(x_0) & \ldots & g_n(x_0) \\ \vdots & & \vdots \\ g_0(x_n) & \ldots & g_n(x_n) \end{pmatrix}}_{A} \cdot \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}$$

Insgesamt muss also ein Gleichungssystem mit $n+1$ Variablen und $n+1$ Gleichungen gelöst werden. Insbesondere muss also $\det(A) \neq 0$ gelten.

Polynome

Besonders einfache Funktionen in einer Variablen sind die **Polynome**:

$$p(x) = a_n x^n + \ldots + a_1 \cdot x + a_0 \qquad a_n \neq 0$$

Wir bezeichnen mit $n =: \deg(p)$ den **Grad** von p. Der Grad der Nullfunktion wird mit $-\infty$ definiert und es gilt $\deg(p \cdot q) = \deg(p) + \deg(q)$.

Wir bezeichnen mit

$$\Pi_n := \{p \colon \mathbb{R} \to \mathbb{R} | p \text{ ist Polynom} \wedge \deg(p) \leqslant n\}$$

den Menge aller Polynome, die einen Grad von maximal n haben.

Π_n ist ein Untervektorraum von $\mathrm{Abb}(\mathbb{R}, \mathbb{R})$ und besitzt die Dimension $n + 1$. Eine Basis von Π_n ist die Menge der Monome $\{1, x, x^2, \ldots, x^n\}$.

Polynominterpolation

Benutzen wir Polynome zur Interpolation, müssen wir zeigen, dass die **Vandermonde-Matrix**

$$V_n := \begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix}$$

invertierbar ist.

Da man zeigen kann, dass

$$\det(V_n) = \prod_{0 \leqslant i < j \leqslant n} (x_j - x_i),$$

können wir sehen, dass **Polynominterpolationen immer eindeutig lösbar** sind, wenn die Stützstellen paarweise verschieden sind.

Lagrange-Polynome

Da das Invertieren von Matrizen mit $\mathcal{O}(n^3)$ recht zeitaufwendig ist, reicht es, solche Polynome $L_j \in \Pi_n$ zu definieren, die Folgendes erfüllen:

$$L_j(x_i) = \begin{cases} 1 & \text{, wenn } i = j \\ 0 & \text{, wenn } i \neq j \end{cases}$$

Diese Polynome sind eindeutig durch die Stützstellen bestimmt

$$L_j(x) := \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i} \in \Pi_n$$

und heißen **Lagrange-Polynome**.
Das Interpolationsproblem wird dann durch folgendes Polynom eindeutig gelöst

$$p(x) = \sum_{j=0}^{n} y_j \cdot L_j(x)$$

Online-Berechnung

Die explizite Berechnung ist sehr teuer und kann wegen der $\mathcal{O}(n^2)$ wiederholten Differenzen numerisch instabil werden.

Wenn wir das interpolierende Polynom nur an wenigen Stellen berechnen wollen, bietet es sich an, interpolierende Polynome induktiv zu berechnen, die immer mehr Stützstellen berücksichtigen.

Definiere hierzu $p_{i,\ell}(x) \in \Pi_\ell$ als das Polynom vom Grad ℓ, das genau an den Stellen $x_i, \ldots, x_{i+\ell}$ die Interpolations-Bedingungen erfüllt.

Zur Berechnung von $p_{i,\ell}$ benutzen wir $p_{i,\ell-1}$ und $p_{i+1,\ell-1}$, d.h. wir können $p(x)$ rekursiv bestimmen.

Rekursive Berechnung

Wir können folgende **rekursive Berechnungsformel** benutzen

$$\begin{aligned} p_{i,0}(x) &= y_i \\ p_{i,\ell}(x) &= \frac{(x - x_i)p_{i+1,\ell-1}(x) - (x - x_{i+\ell})p_{i,\ell-1}(x)}{x_{i+\ell} - x_i} \end{aligned}$$

Die Interpolationsbedingungen sind für $p_{i,0}$ offensichtlich erfüllt.

Für den Rekursionsschritt gilt

$$\begin{aligned} p_{i,\ell}(x_i) &= p_{i,\ell-1}(x_i) = y_i \\ p_{i,\ell}(x_{i+\ell}) &= p_{i+1,\ell-1}(x_{i+\ell}) = y_{i+\ell} \\ p_{i,\ell}(x_{i+j}) &= \frac{(x_{i+j} - x_i)y_{i+j} - (x_{i+j} - x_{i+\ell})y_{i+j}}{x_{i+\ell} - x_i} = y_{i+j} \end{aligned}$$

Neville-Tableau

Wegen der Eindeutigkeit des interpolierenden Polynoms ist jedes der Polynome $p_{i,\ell}$ die eindeutige Lösung des jeweiligen Interpolationsproblems.

Wir können nun $p(x)$ mit Hilfe des **Neville-Tableaus** berechnen:

Grad	0	1	2
x_0	$p_{0,0}(x) = y_0$		
		$p_{0,1}(x)$	
x_1	$p_{1,0}(x) = y_1$		$p_{0,2}(x) = p(x)$
		$p_{1,1}(x)$	
x_2	$p_{2,0}(x) = y_2$		

Ein Vorteil des Neville-Tableaus ist, dass es einfach ist, $p(x)$ zu aktualisieren, wenn weitere Stützstellen bekannt werden.

Polynominterpolation (Lagrange)

Betrachten wir folgendes Interpolationsproblem:

$$(x_0, y_0) = (0, 1) \qquad (x_1, y_1) = (1, 3) \qquad (x_2, y_2) = (3, 7)$$

Unter Benutzung der Lagrangepolynome lässt sich $p(x)$ wie folgt berechnen

$$\begin{aligned}
L_0(x) &= \frac{(x-1)(x-3)}{(0-1)(0-3)} = \frac{1}{3}(x^2 - 4x + 3) \\
L_1(x) &= \frac{(x-0)(x-3)}{(1-0)(1-3)} = -\frac{1}{2}(x^2 - 3x) \\
L_2(x) &= \frac{(x-0)(x-1)}{(3-0)(3-1)} = \frac{1}{6}(x^2 - x) \\
p(x) &= \frac{1}{3}(x^2 - 4x + 3) - \frac{3}{2}(x^2 - 3x) + \frac{7}{6}(x^2 - x) \\
&= 2x + 1
\end{aligned}$$

Polynominterpolation (Neville)

Betrachten wir folgendes Interpolationsproblem:

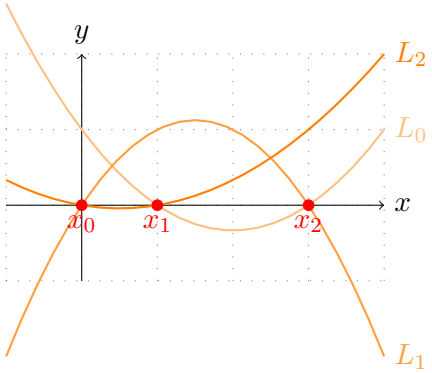
$$(x_0, y_0) = (0, 1) \qquad (x_1, y_1) = (1, 3) \qquad (x_2, y_2) = (3, 7)$$

Die Berechnun des Neville-Tableaus erfolgt mit der Rekursionsvorschrift

$$p_{i,\ell}(x) = \frac{(x - x_i)p_{i+1,\ell-1}(x) - (x - x_{i+\ell})p_{i,\ell-1}(x)}{x_{i+\ell} - x_i}$$

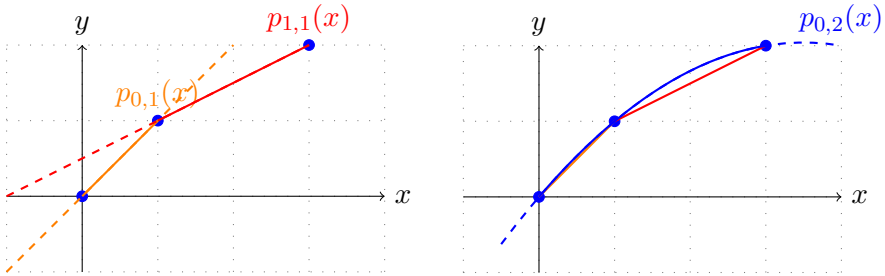
Grad	0		1	2
x_0 0	$p_{0,0}(x)$	=		
	y_0 1			
			$p_{0,1}(x)$ 2x +	
			1	
x_1 1	$p_{1,0}(x)$	=		$p_{0,2}(x)$ 2x +
	y_1 3			1
			$p_{1,1}(x)$ 2x +	
			1	
x_2 3	$p_{2,0}(x)$	=		
	y_2 7			

Lagrangepolynome (Beispiel)



Lagrangepolynome sind immer vom maximalen Grad. Falls eine Funktion von niedrigerem Grad interpoliert werden soll, kann es zu Auslöschungen kommen.

Neville-Polynome (Beispiel)



Das Neville-Tableau verbessert die Interpolation in jedem Schritt.

Man kann die Berechnung früher abbrechen, wenn man nur an Funktionen interesiert ist, die stückweise linear, quadratischen, etc. sind.

Dies wird (im Wesentlichen) bei der Interpolation von Bildern benutzt.

Fehler bei Polynominterpolation

Lemma 1. *Gegeben $n+1$ Stützstellen $x_0 < \ldots < x_n$ und $f \in C^{n+1}(\mathbb{R})$, d.h. $f : \mathbb{R} \to \mathbb{R}$, so dass die ersten $n+1$ Ableitungen existieren und stetig sind. Weiter sei $p \in \Pi_n$ das bzgl. der Stützstelle eindeutige Interpolationspolynom. Dann gilt für jedes $\overline{x} \in \mathbb{R}$*

$$f(\overline{x}) - p(\overline{x}) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(\overline{x} - x_0) \cdot \ldots \cdot (\overline{x} - x_n),$$

für ein $\xi \in [\min(\overline{x}, x_0); \max(x_n, \overline{x})]$.

Beweis. Für $\overline{x} \in \{x_0, \ldots, x_n\}$ ist nicht zu zeigen. Sei also $\overline{x} \notin \{x_0, \ldots, x_n\}$. Wählen wird $K := \frac{f(\overline{x}) - p(\overline{x})}{\prod_{i=0}^{n} \overline{x} - x_i}$ so hat die Funktion

$$g(x) = f(x) - p(x) - K \prod_{i=0}^{n} x - x_i$$

mindestens $n+2$ Nullstellen und zwar bei $\overline{x}$ und $x_0, \ldots, x_n$.

Fehler bei Polynominterpolation

Beweis (Fort.) Nach dem Satz von Rolle hat die i-te Ableitung mindestens $n+2-i$ Nullstellen. Damit gilt für ein $\xi \in [\min(\overline{x}, x_0); \max(x_n, \overline{x})]$ gerade

$$\begin{aligned} 0 =& g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - K \left(\frac{\mathrm{d}}{\mathrm{dx}}\right)^{n+1} \left(\prod_{i=0}^{n} x - x_i\right)\Bigg|_{x=\xi} \\ =& f^{(n+1)}(\xi) - K(n+1)! \end{aligned}$$

□

Für das Interpolationsproblem bedeutet das Lemma gerade

$$|f(x) - p(x)| \leqslant \frac{M}{(n+1)!} \prod_{i=0}^{n} |x - x_i|\,,$$

wobei $M := \max_{x \in [x_0, x_n]} \left|f^{(n+1)}(x)\right|$ den maximalen Absolutwert der $(n+1)$-ten Ableitung beschreibt.

Fehleranalyse

Wir haben also insgesamt den folgenden Fehler

$$|f(x) - p(x)| \leqslant \frac{M}{(n+1)!} \left| \prod_{i=0}^{n} (x - x_i) \right|,$$

Auf $(n+1)!$ haben wir **keinen Einfluß**, da es sich um eine Konstante handelt.

Auf M haben wir **keinen Einfluß**, da es sich bei gegebenem f um eine Konstante handelt.

Auf $w_n(x) := \prod_{i=0}^{n}(x - x_i)$ können wir Einfluß nehmen, indem wir die **Stützstellen x_i geschickt wählen**.

Optimale Stützstellen

Wir suchen ein Polynom $T_n \in \Pi_n$ mit führendem Koeffizienten $a_n = 1$, so dass

$$\max_{x \in [a,b]} |T_n(x)|$$

minimal ist. Die optimalen Stützstellen sind dann die n Nullstellen dieses Polynoms.

Ohne Einschränkung können wir davon ausgehen, dass $a = -b$ gilt. Den allgemeinen Fall erhalten wir dann durch Verschieben des Polynoms $T_n(x) \rightsquigarrow T_n(x - \Delta)$ und der Nullstellen $x_i \rightsquigarrow x_i + \Delta$.

Ohne Einschränkung können wir davon ausgehen, dass $[-b, b] = [-1, 1]$ gilt. Den allgemeinen Fall erhalten wir dann durch Skalierung des Polynoms $T_n(x) \rightsquigarrow T_n\left(\frac{x}{b}\right)$ und der Nullstellen $x_i \rightsquigarrow x_i \cdot b$.

Tschebyscheff-Polynome

Wir sind also auf der Suche nach Polynomen $T_n \in \Pi_n$ mit führendem Koeffizienten $a_n = 1$, so dass

$$\max_{x \in [-1,1]} |T_n(x)|$$

minimal ist. Diese Polynome sind die normierten **Tschebyscheff-Polynome**.

Für $T_1(x) = x + a$ gilt

$$\max_{x \in [-1,1]} |x + a| = \max(|1 + a|, |a - 1|) = \begin{cases} 1 + |a| & a > 0 \\ 1 + |a| & a < 0 \end{cases}$$

Somit ist also $a = 0$ und $T_1(x) = x$.

Es stellt sich heraus, dass im Allgemeinen Folgendes gilt

$$T_n(x) = \frac{1}{2^{n-1}} \cos\left(n \cdot \cos^{-1}(x)\right) \qquad \forall n > 0$$

Tschebyscheff-Rekursion

Für die normierten Tschebyscheff-Polynome T_n gilt die Rekursionsvorschrift

$$T_0(x) = 1 \qquad T_1(x) = x \qquad T_2(x) = x^2 - \frac{1}{2}$$

$$T_{n+1}(x) = xT_n(x) - \frac{1}{4}T_{n-1}(x)$$

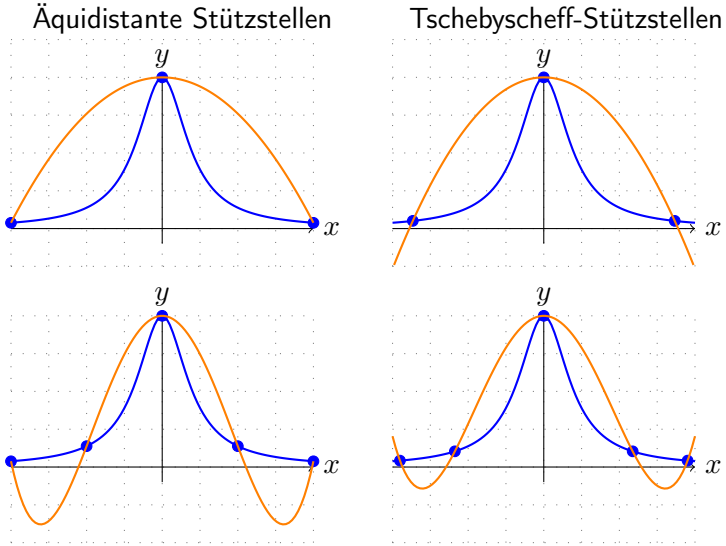
Daher kann man sehen, dass $T_n \in \Pi_n$ ist und $a_n = 1$ gilt.
Für die optimalen Stützstellen gilt also

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right) \qquad i = 0, \ldots, n$$

und wir haben für das Interpolationsproblem auf dem Intervall $[-1, 1]$ gerade

$$|f(x) - p(x)| \leqslant \frac{M}{2^n(n+1)!}$$

Interpolation (Beispiel)



Zusammenfassung

Man sollte äquidistante Stützstellen vermeiden, da sie zu großen Fehlern am Rand führen können.

Stattdessen sollte man mehr Stützstellen am Rand benutzen, um diesen Effekt zu minimieren.

Ein Beispiel hierfür sind die Tschebyscheff-Stützstellen.

Tschebyscheff-Stützstellen haben den Vorteil, dass sie die obere Schranke des möglichen Interpolationsfehler minimieren.

Hermite-Interpolation

Um Oszillationen zu vermeiden, kann man neben den Funktionswerten auch die Ableitungen an den Stützstellen vorgeben.

Im einfachsten Fall definiert man also (x_i, y_i, y_i') für $i = 0, \ldots, n$ und sucht ein Polynom $p \in \Pi_{2n+1}$ mit

$$p(x_i) = y_i \qquad p'(x_i) = y_i' \qquad i = 0, \ldots, n$$

Dies führt zu einem Gleichungssystem

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{2n+1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{2n+1} \\ 0 & 1 & 2x_0 & \ldots & (2n+1)x_0^{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & 2x_n & \ldots & (2n+1)x_n^{2n} \end{pmatrix} \cdot \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{2n+1} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \\ y_0' \\ \vdots \\ y_n' \end{pmatrix}$$

Hermite-Interpolation

Um zu sehen, dass dieses Gleichungssystem immer eindeutig lösbar ist, müssen wir nur zeigen, dass $y = y' = 0$ nur die Lösung $\lambda = 0$ zulässt.

Falls $y = y' = 0$ ist, muss jedes x_i eine doppelte Nullstelle sein und das Interpolationspolynom muss durch

$$\prod_{i=0}^{n}(x - x_i)^2 \in \Pi_{2n+2}$$

teilbar sein. Das einzige Polynom $p \in \Pi_{2n+1}$, das dies erfüllt, ist $p = 0$.

Analog kann man zeigen, dass auch höhere Ableitungen ohne Probleme in ein Interpolationsproblem integriert werden können.

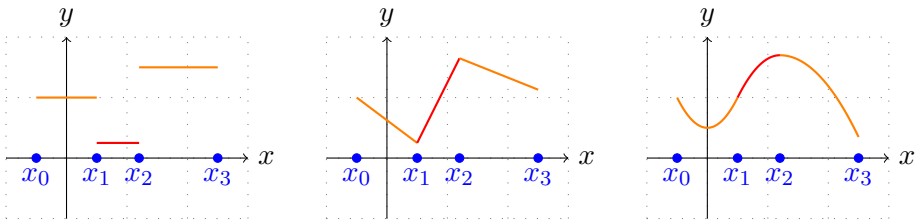
Allerdings gibt es für eine Verallgemeinerung von Lagrange-Polynomen keine geschlosssenen Lösungen. Daher sind diese Probleme üblicherweise schwerer als das klassische Interpolationsproblem.

Spline-Interpolation

Eine andere Möglichkeit, um Oszillationen zu verhindern, besteht darin, eine Funktion zu benutzen, die stückweise aus Polynomen niedrigen Grades besteht, d.h. zu den Stützstellen $x_0 < \ldots < x_n$ gilt

$$S \in C^{k-1}((x_0, x_n), \mathbb{R})$$
$$S|[x_i, x_{i+1}] \in \Pi_k$$

Solche Funktionen nennen wir **Splines**.



$k = 0$ $k = 1$ $k = 2$

Spline-Interpolationsproblem

Gegeben $n + 1$ Stützstelle $x_0 < \ldots < x_n$ besteht das Spline-Interpolationsproblem darin n Polynome vom Grad $k > 0$ zu finden ($n \cdot (k + 1)$ Parameter), so dass Folgendes gilt

$$S(x_i) = y_i \qquad (2n \text{ Nebenbedingungen})$$
$$S^{(\kappa)}(x_i) = y_i^{(\kappa)} \qquad 0 < \kappa < k \qquad ((n-1)(k-1) \text{ Nebenbedingungen})$$

Insgesamt haben wir also $n(k + 1) + (1 - k)$ Nebenbedingungen. Um eine eindeutige Lösung zu erhalten, benötigen wir $k - 1$ weitere Nebenbedingungen:

- Für lineare Splines werden keine weiteren Nebenbedingungen benötigt.
- Durch Forderung von Periodizität ($S^{(\kappa)}(x_0) = S^{(\kappa)}(x_n)$) erhalten wir genau $k - 1$ Nebenbedingungen.
- Man kann $k - 1$ weitere Nebenbedingunen (z.B. Ableitungen bei x_0 und/oder x_n) einführen.

Bernstein-Polynome

Das Lösen des oben beschriebene Interpolationsproblem ist recht aufwändig.

Eine Alternative besteht darin, $k+1$ linear unabhängige Polynome $B_{i,k} \in \Pi_n$ zu wählen, die man als Basisfunktionen benutzen kann. Hierfür eignen sich die **Bernsteinpolynome**

$$\begin{aligned} B_{i,k} \colon [0;1] &\to [0;1] \\ t &\mapsto \binom{k}{i} t^i (1-t)^{k-i} \end{aligned}$$

Bernsteinplynome nehmen nur Werte zwischen 0 und 1 an, was Oszillationen vermeidet. Außerdem gilt für $p(x) = \sum_{i=0}^{k} \lambda_i B_{i,k}(x)$ gerade

$$p(0) = \lambda_0 \qquad\qquad p(1) = \lambda_k$$

Kubische Interpolation (Bernstein-Polynome)

Für die Ableitung von Bernstein-Polynomen gilt gerade Folgendes

$$B'_{i,k}(t) = k \cdot (B_{i-1,k-1} - B_{i,k-1})$$

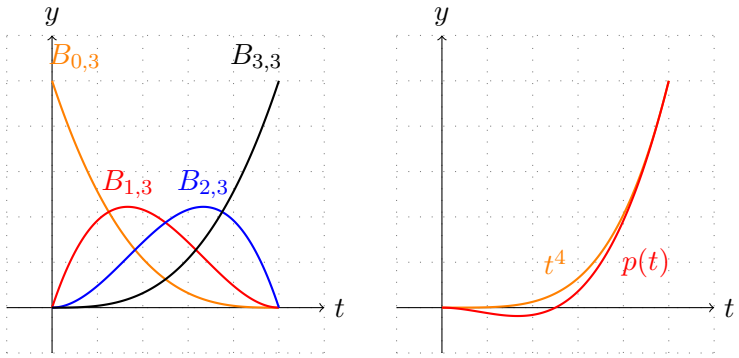
Das heißt wir können uns folgende Funktion ansehen

$$\begin{aligned} p(t) =& x_0 B_{0,3} + \left(x_0 + \frac{1}{3} y_0\right) B_{1,3} + \left(x_1 - \frac{1}{3} y_1\right) B_{2,3} + x_1 B_{3,3} \\ p'(t) =& y_0 B_{0,2} + (3(x_1 - x_0) + (y_0 + y_1)) B_{1,2} + y_1 B_{2,2} \end{aligned}$$

Dies löst gerade das Iterpolationsproblem

$$p(0) = x_0 \qquad p(1) = x_1 \qquad p'(0) = y_0 \qquad p'(1) = y_1$$

Bernstein-Polynome (Beispiel)



Wir wollen das Monom t^4 durch ein kubisches Bernsteinpolynom annähern:

$$p(0) = 0 \qquad p(1) = 1 \qquad p'(0) = 0 \qquad p'(1) = 4$$

Insgesamt gilt also $p(t) = -\frac{1}{3}B_{2,3} + B_{3,3}$.

Bézier-Kurve

Eine natürliche Erweiterung von Funktioninterpolationen sind Interpolationen von Kurven

$$c\colon [0;1] \to \mathbb{R}^k$$

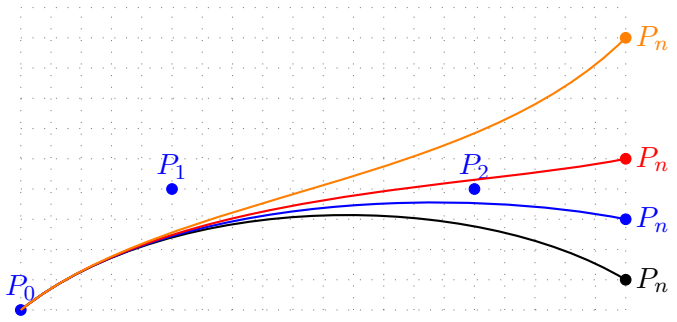
Hier kann man die Projektion von c auf ihre Komponenten $c_1, \ldots, c_k\colon [0;1] \to \mathbb{R}$ als Funktionen auffassen.

Eine **Bézier-Kurve** $C\colon [0;1] \to \mathbb{R}^k$ vom Grad n hat die folgende Darstellung

$$C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t),$$

wobei $B_{i,n}$ die Bernsteinpolynome vom Grad n sind. $P_i \in \mathbb{R}^k$ heißen **Kontrollpunkte**, da sie nicht notwendigerweise Interpolationspunkte sind, aber die Funktionswerte (und Ableitungen) von C *kontrollieren*.

Bézier-Kurven (Beispiel)



Eine Bézier-Kurve ist eine gewichtete Summe von Vektoren, die bei P_0 beginnt und bei P_n endet.

Dabei überwiegt in einem Teilbereich von $[0;1]$ jeweils eines der Bernsteinpolynome $B_{i,n}$ die anderen, und sorgt dafür, dass die Kurve zu dem entsprechenden Kontrollpunkt P_i gezogen wird.

B-Splines

Anstelle von Bernsteinpolynomen kann man andere Basisfunktionen wählen, die nicht auf dem gesamten Intervall $(0;1)$ positiv sind. Damit verändert sich die Kurve nur lokal, wenn einer der Kontrollpunkte verändert wird.

Ein Beipiel hierfür sind **B-Splines**. Gegeben n Knoten $0 \leqslant t_0 < \ldots < t_n \leqslant 1$, definiert man die linear unabhängige Splines $N_{i,k}$ vom Grad k, die folgende Eigenschafen erfüllen

$0 \leqslant N_i(t) \leqslant 1$	$t \in [0;1]$	(beschränkt)
$N_i(t) = 0$	$t \notin [t_{i-k}; t_{i+k}]$	(lokal)
$\sum_i N_i \equiv 1$		(Partition der Eins)

B-Splines und Erweiterungen davon werden in den meisten CAD-Systemen benutzt. (CAD=Computer Aided Design)

Zusammenfassung

Polynom-Interpolation berechnet für n Paare (x_i, y_i) das Polynom $p \in \Pi_n$, so dass $p(x_i) = y_i$.

Hermite-Interpolation berechnet für n Trippel (x_i, y_i, y_i') das Polynom $p \in \Pi_{2n+1}$, so dass $p(x_i) = y_i$ und $p'(x_i) = y_i'$.

Spline-Interpolation berechnet $p \in C^{k-1}$, das stückweise aus Polynome vom Grad k besteht, so dass die Interpolationsbedingungen erfüllt sind.

Bézier-Kurve $p(t) = \sum_{i=0}^{n} P_j B_j(t)$ wird durch die P_j kontrolliert. Es handelt sich nicht um eine Interpolation.

B-Splines $p(t) = \sum_{i=0}^{n} P_j N_j(t)$ nutzen Basisfunktionen, so dass Kontrollpunkte die resultierende Kurve nur lokal beeinflussen.