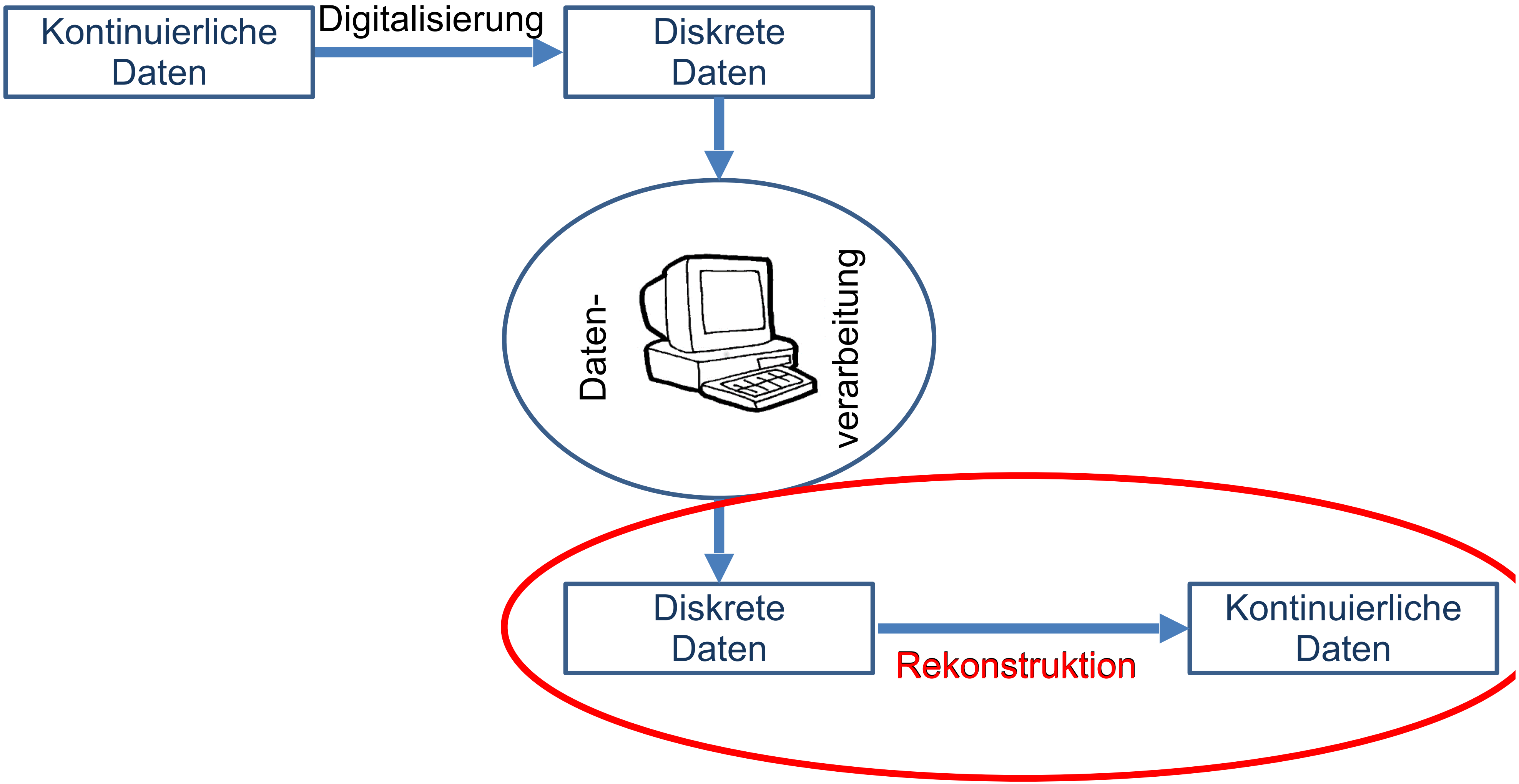
Algorithmik kontinuierlicher Systeme

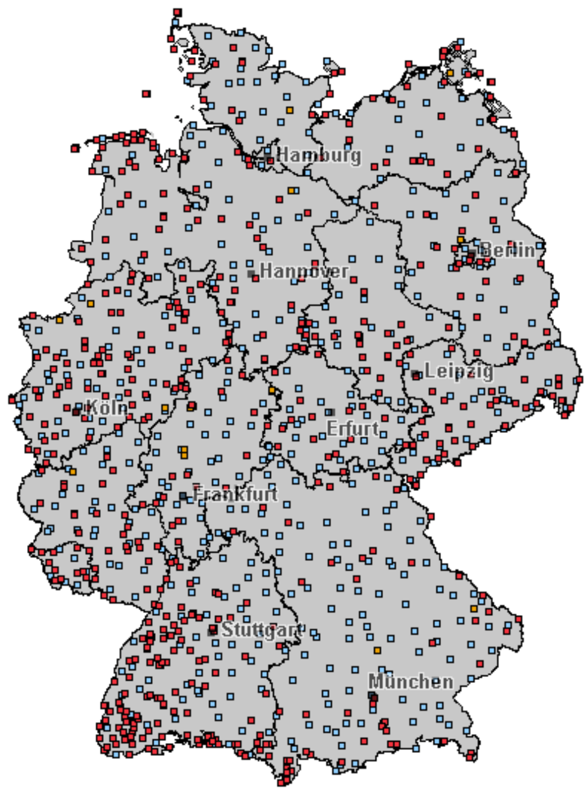
Rekonstruktion kontinuierlicher Daten – Interpolation 1D

Motivation

- Rekonstruktion kontinuierlicher Daten aus diskreten Daten



- Digital-Analog-Wandlung : D/A Wandler
- Äquidistante Abtastung: Abtast-Theorem
  - Falls Voraussetzungen erfüllt, Rekonstruktion mit sinc-Funktion
  - Die Voraussetzungen sind in konkreten Anwendungen meist nicht gegeben
  - Beispiel Wetterkarte



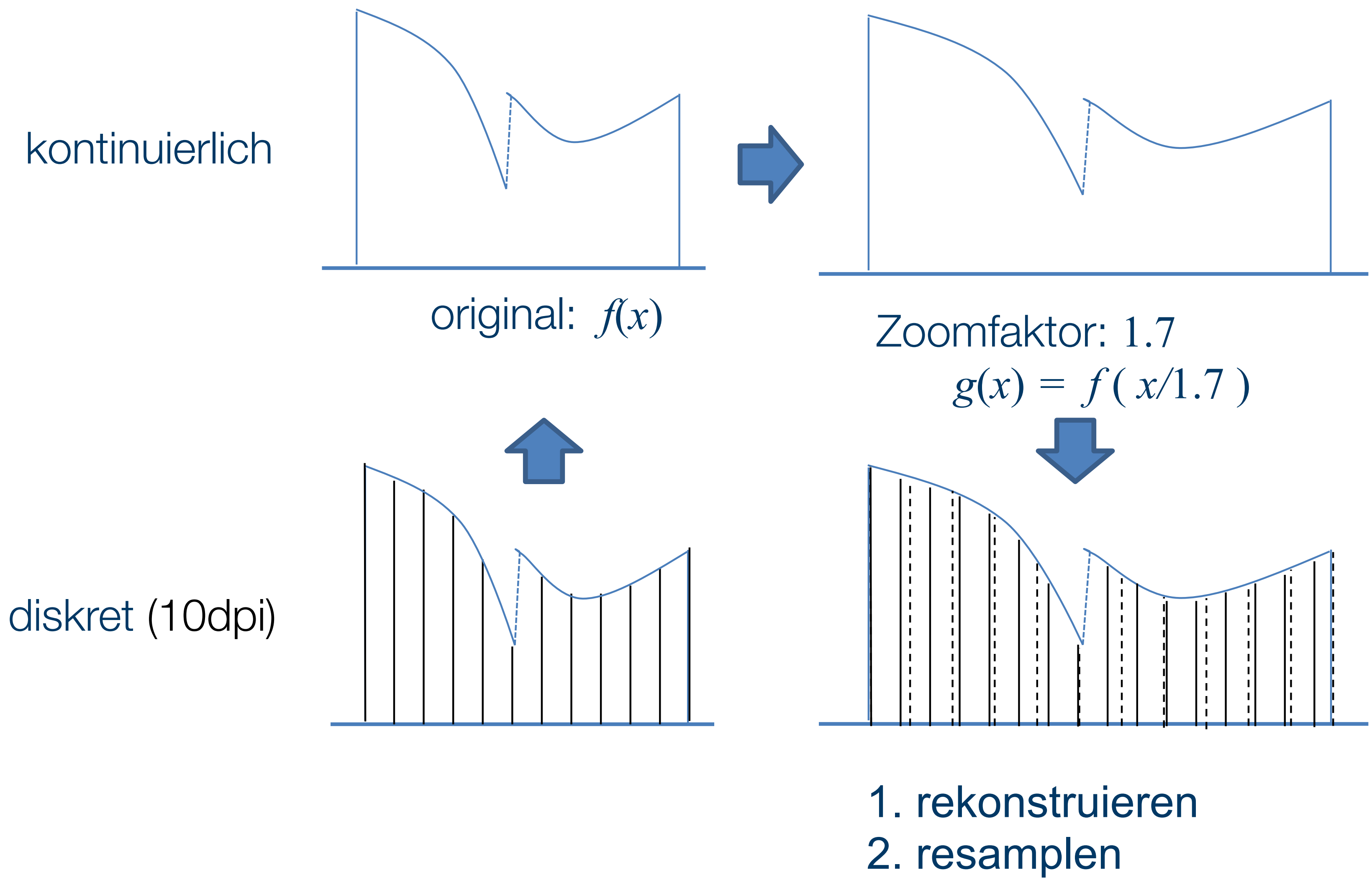
Wetterstationen Deutschland

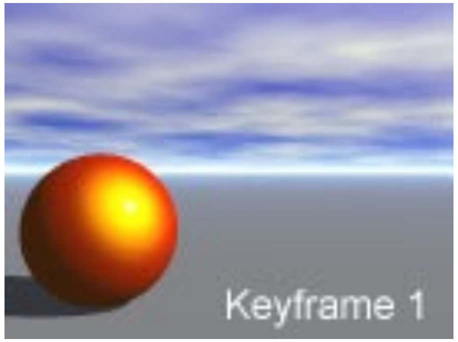
- Alternativen: Interpolation und Approximation

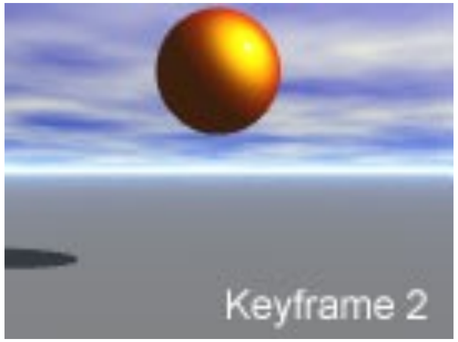
- Rekonstruktion „kontinuierlicher Daten“ aus diskreten
  - Gegeben „Punkte“
  - Gesucht stetige (glatte) „Kurve“ (oder Fläche, oder Körper)

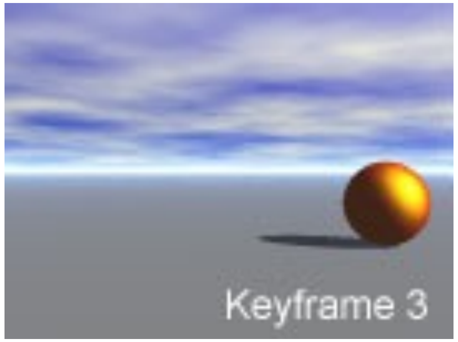
- Beispiele:
  - CAD-Daten aus Abtastwerten eines Modells errechnen
  - Vermessung: Geo-Informationssysteme (Digitale Höhenmodelle, Erstellung digitaler Landkarten)
  - Rückwandlung digitalisierter Signale in analoge Signale (Bilder, Audio, Video, Umrechnung von Bildauflösungen, Abtastraten, etc.),
  - „In-Betweening“ bei Animationen – Key frame Animation
  - „Zoomen“ digitaler Daten

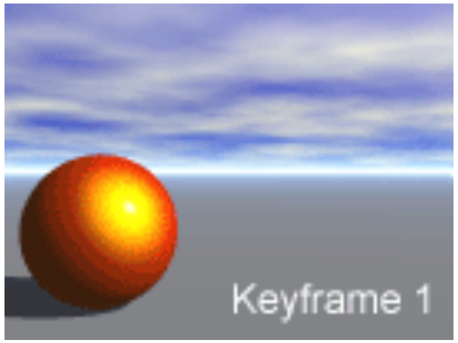
- Zoomen eines digitalen Bildes bei fester Auflösung
- anschaulich: Grauwertbild, nur eine Bildzeile



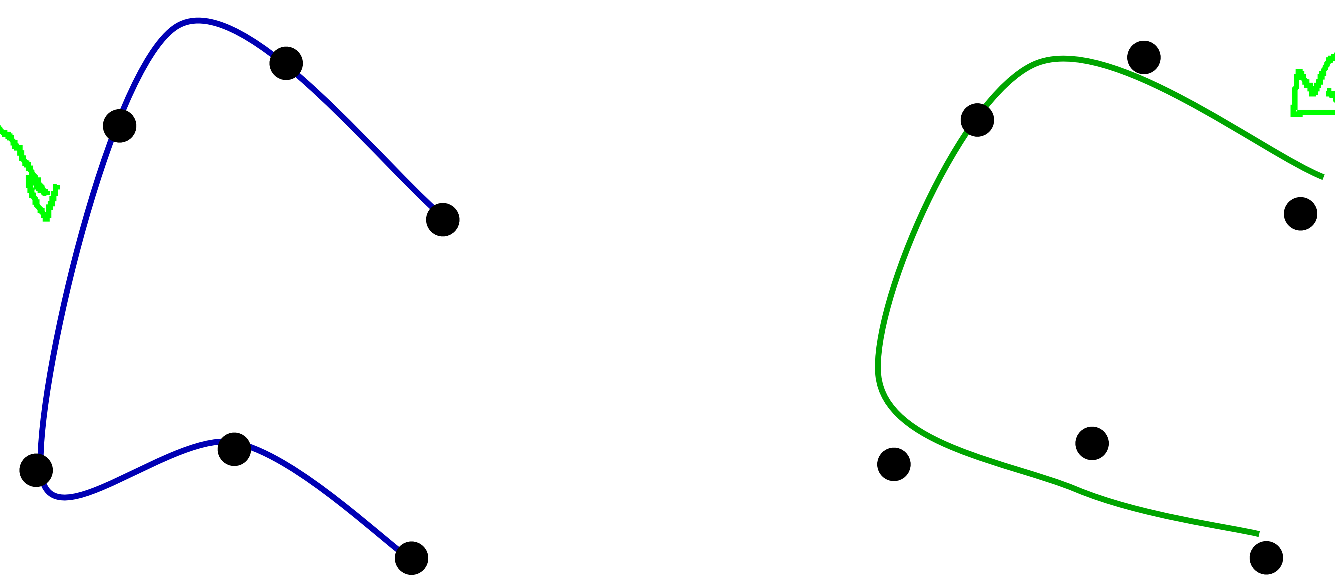

Keyframe 1


Keyframe 2

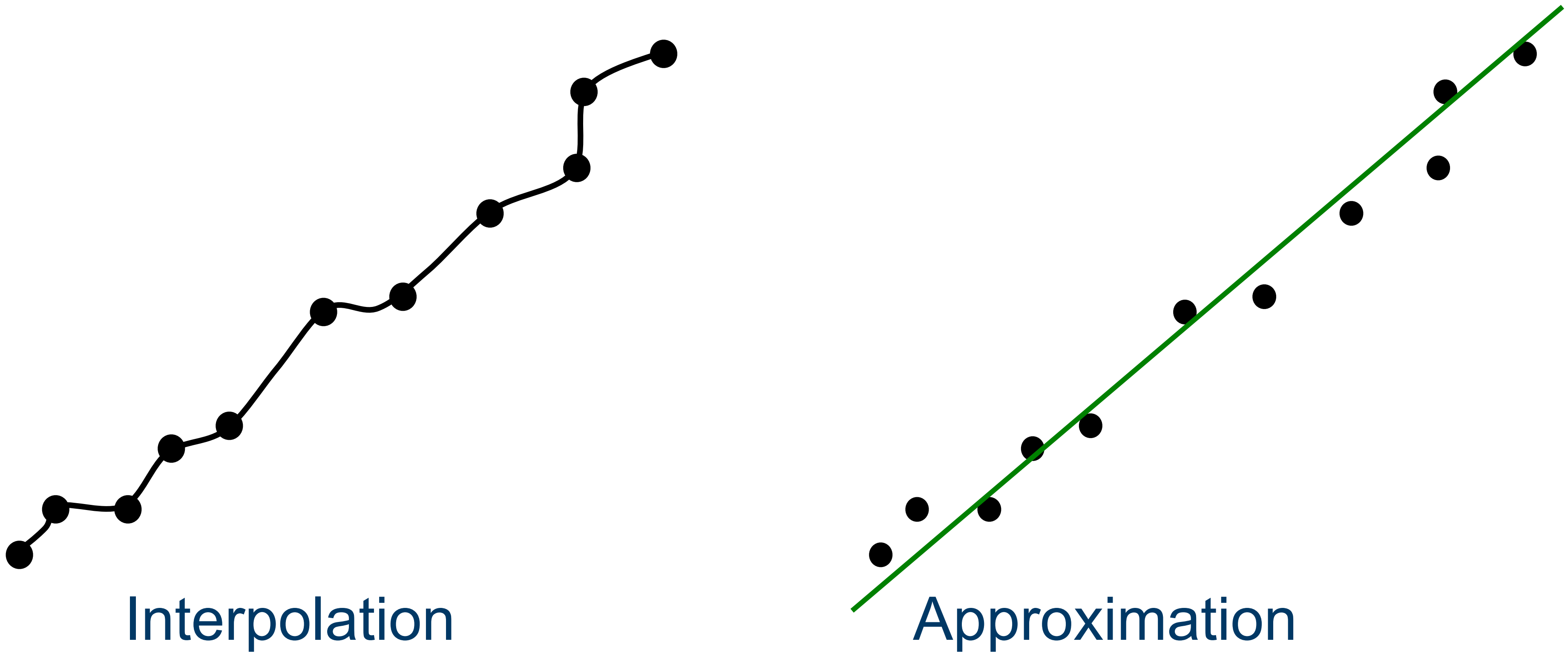

Keyframe 3


Keyframe 1

- Interpolation vs. Approximation
  - Soll die rekonstruierte Funktion (Kurve, Fläche, …) exakt durch die Punkte verlaufen, so sprechen wir von **Interpolation**. Das behandeln wir im Folgenden.
  - Soll die Kurve (Fläche) nur näherungsweise durch die Punkte verlaufen, so sprechen wir von **Approximation**. Das ist verwandt mit den Least-Squares-Problemen, (Ausgleichsgerade, etc. s.o.)



- Interpolation ist nicht in jedem Fall das genauere, bessere Verfahren
  - z.B. bei Messfehlern



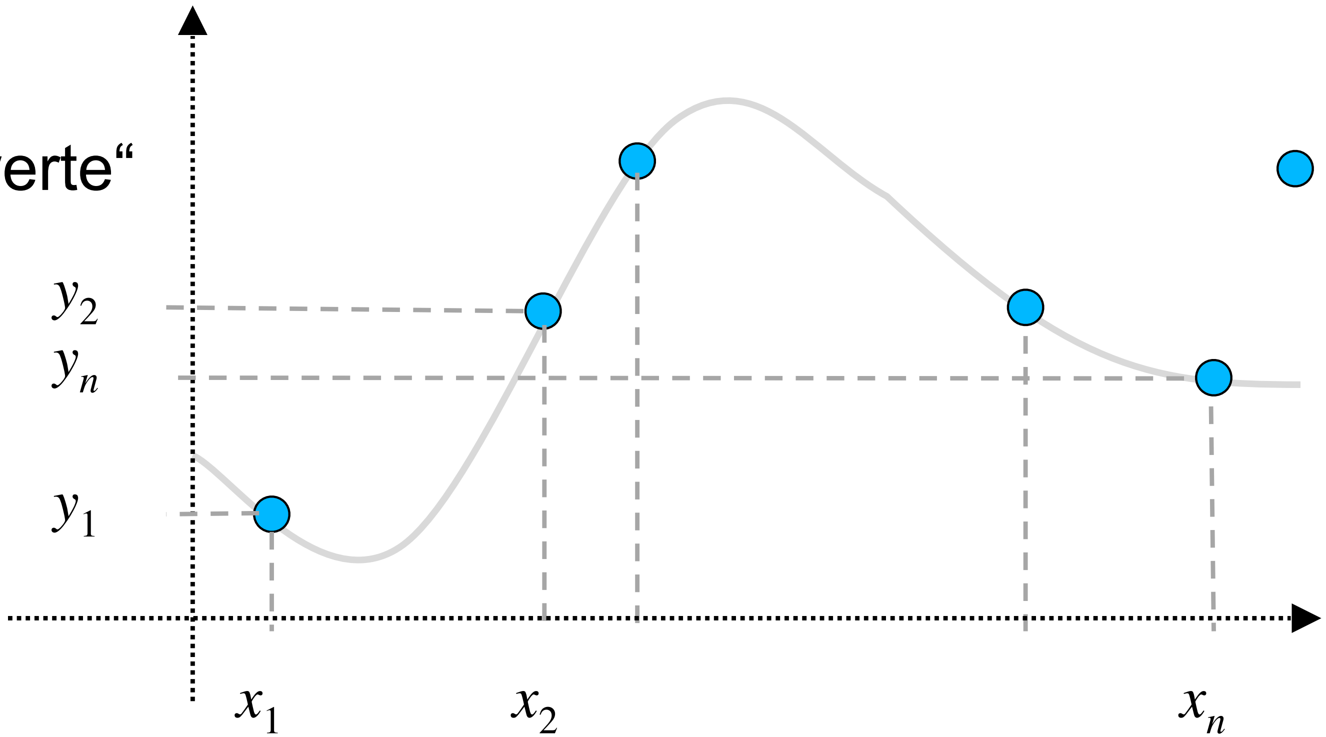
- Verwandte Aufgabenstellungen:
  - Interpolation/Approximation komplizierter Funktionen durch einfachere (z.B. Approximation der exp-Funktion durch Polynome)
    - „Komplizierte“ (transzendente) Funktionen werden intern mit Interpolation approximiert.

- Kontinuierliche Daten sind eigentlich Relationen, d.h. allgemeiner als „Funktionen“
  - Rekonstruktion von „Kurven“ anstelle von Funktionen
  - Aus Gründen der Einfachheit werden wir meistens trotzdem über Funktionen sprechen.

- Von einer unbekannten Funktion $f : \mathbb{R} \to \mathbb{R}$ kennt man nur die Werte an endlich vielen Punkten:

$$y_i = f(x_i) \quad \text{für} \quad i = 1, 2, \ldots, n$$

„Stützwerte“

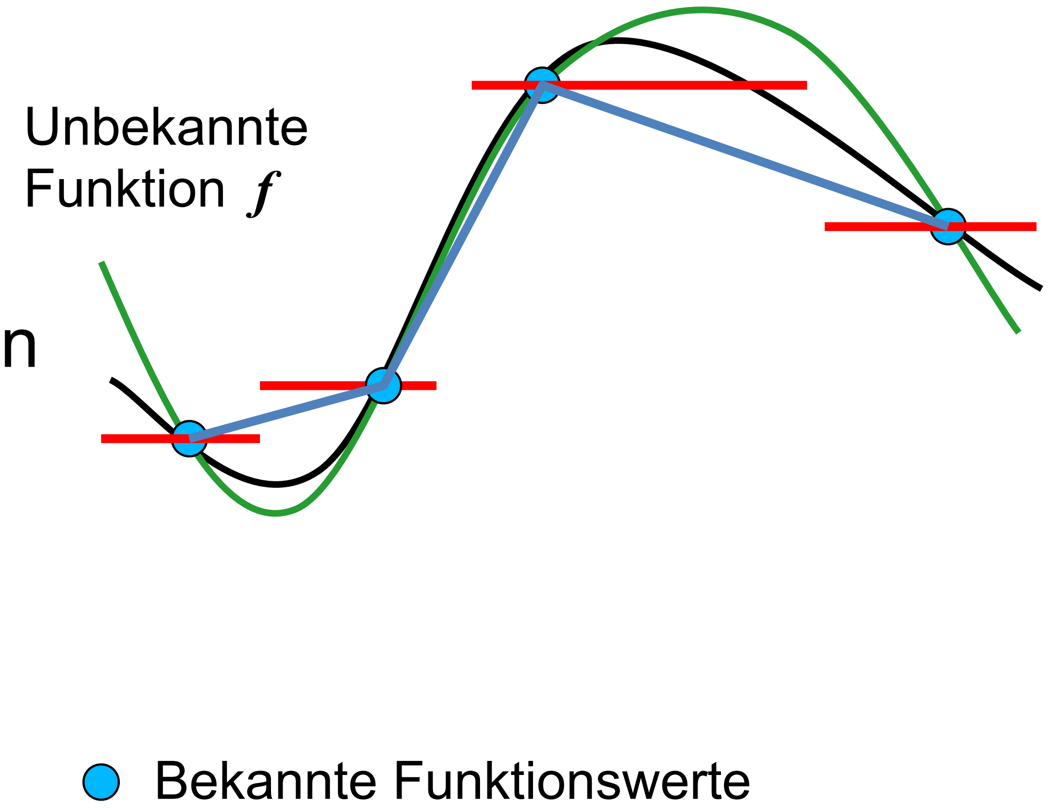


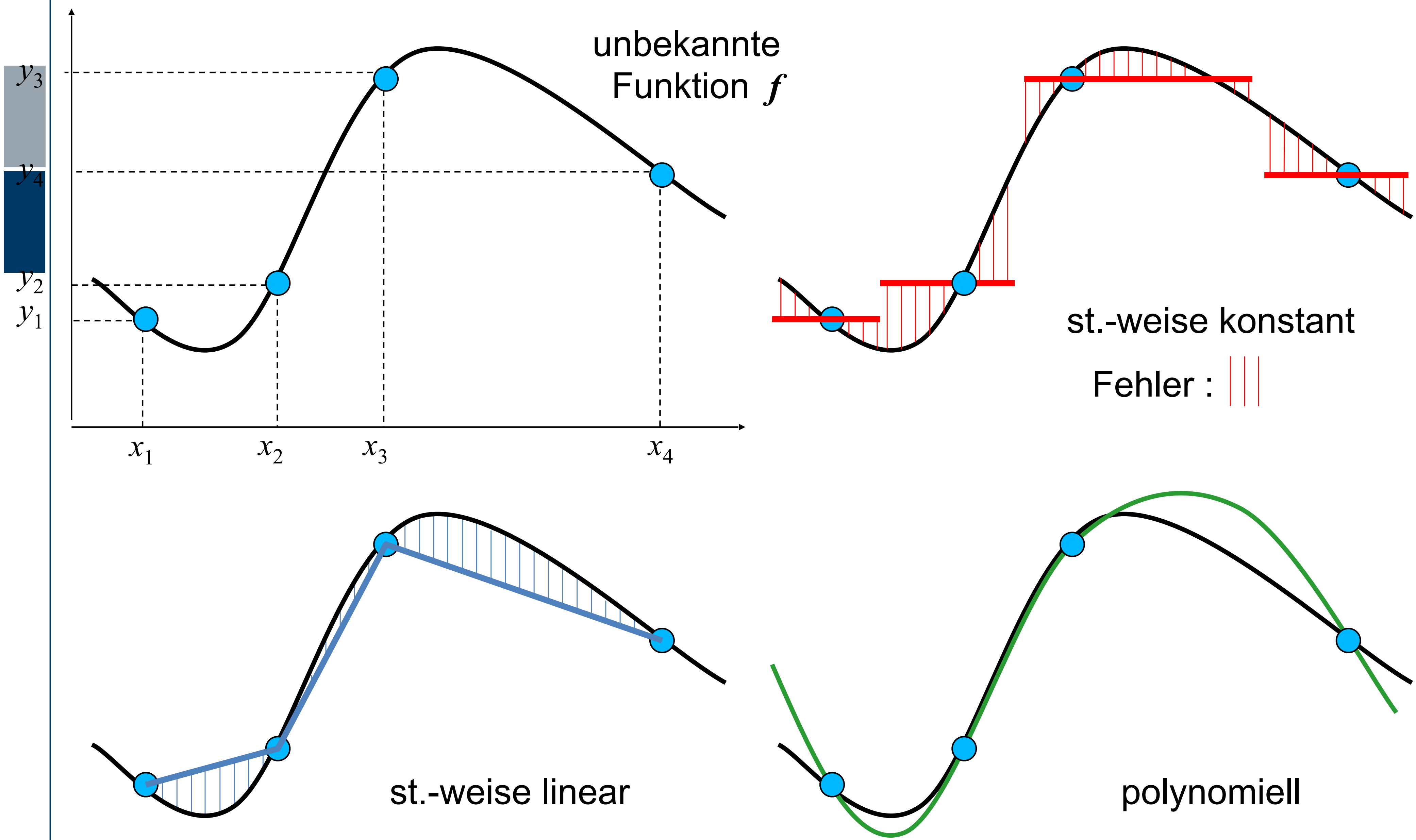
- Annahme: $x_1 < x_2 < \ldots < x_{n-1} < x_n$ „Stützstellen“

- Aus der Anwendung kennen wir gewisse Grundeigenschaften von f, z.B.
  - f ist eine Gerade, oder ein Polynom
  - f ist stetig (oder glatt, d.h. differenzierbar).

- Wir wählen eine Klasse (Menge) $\boldsymbol{K}$ von Funktionen, in der wir f aus den endlich vielen Punkten näherungsweise rekonstruieren.
- Kriterien:
  - f muss in der Funktionsklasse $\boldsymbol{K}$ gut approximierbar sein,
  - die Funktionsklasse $\boldsymbol{K}$ muss auf dem Rechner gut (d.h. effizient) darstellbar und manipulierbar sein.
  - gut darstellbar sind z.B, Polynome
  - Polynome werden oft „gestückelt", d.h. stückweise definiert verwendet

- Unbekannte Funktion
- Rekonstruktion durch stückweise konstante Funktion
- Rekonstruktion durch stückweise lineare Funktion
- Rekonstruktion durch (kubisches) Polynom




y_3
y_4
y_2
y_1
unbekannte Funktion f
x_1
x_2
x_3
x_4
st.-weise konstant
Fehler :
st.-weise linear
polynomiell

- Unterscheidung zwischen **lokalen** und **globalen** Verfahren
  - Lokal: interpolierter Wert $p(x)$ hängt nur von den „benachbarten“ Werten ab. Z.B.: $x_i < x < x_{i+1}$: nur y_i und y_{i+1} gehen ein.
  - Global: alle Werte y_i gehen ein
- Lokale Verfahren
  - Nearest neighbor (stückweise konstant)
  - Linear (stückweise linear)
  - Catmull-Rom (stückweise kubisches Polynom)

parameteranzahl wi

- Globale Verfahren
  - Polynom-Interpolation
  - B-Spline-Interpolation

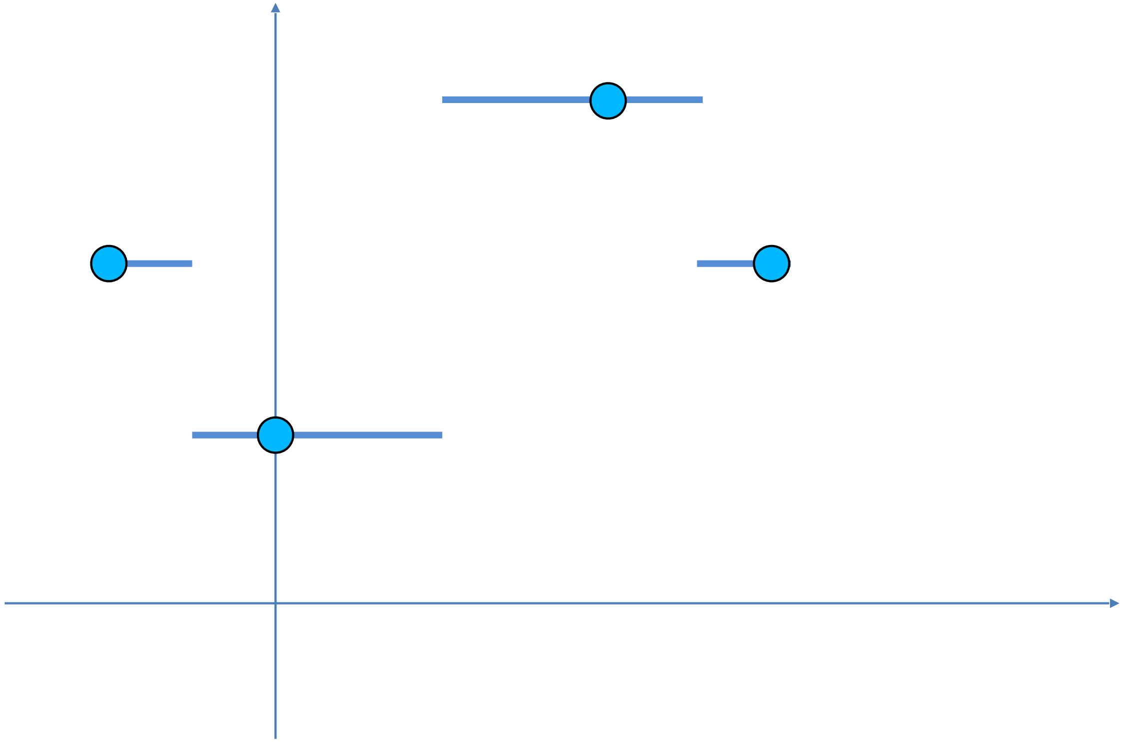
- Gegeben
  - **Stützstellen** $\{ x_1, x_2, \ldots , x_n\}$ und
    Stützwerte $\{ y_1, y_2, \ldots , y_n\}$ (Skalare oder Vektoren)
    (Abtastwerte einer unbekannten,
    zu rekonstruierenden Funktion f)
  - Gesucht: kontinuierliche Rekonstruktion $p(x)$
    (Näherung von $f(x)$)

- Stückweise konstante Interpolation:
  - Um den Wert an der Stelle x anzunähern, suche den *nearest neighbor* d.h. die nächst gelegene Stützstelle x_i
  - Interpolierter Wert ist: $p(x) = y_i$

- Bezeichnung: **nearest neighbor interpolation**

- Beispiel:

x_i	-1	0	2	3
y_i	2	1	3	2



- Allgemeine Formel:

$$p(x) = \begin{cases} y_1 & \text{falls} \quad x_1 \le x \le \frac{1}{2}(x_1 + x_2) \\ y_2 & \text{falls} \quad \frac{1}{2}(x_1 + x_2) < x \le \frac{1}{2}(x_2 + x_3) \\ \vdots & \vdots \quad \vdots \quad \vdots \\ y_n & \text{falls} \quad \frac{1}{2}(x_{n-1} + x_n) < x \le x_n \end{cases}$$

- Kein (Rechen-)Aufwand, nur Suchen der Nachbarn
  - äquidistant: $x_i = a + i\,h$ (h Schrittweite)
    einfache Suche $i = \mathrm{int}(\,(x\text{-}a)/h\,)$ $O(1)$ man kennt hier sofort den nearest neighbor
    Komplexität für die Suche
  - sind die Stützstellen schon geordnet:
    Aufwand für Suche $O(log(n))$ wenn nicht äquidistant
  - wenn nicht, dann $O(n{*}log(n))$

- in vielen Fällen zu ungenau, z.B.
  bei bekanntermaßen glattem f

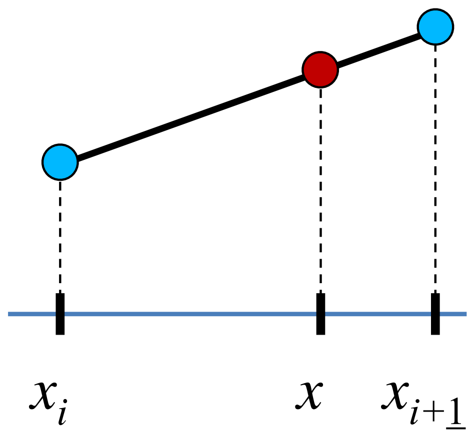
- Fehler bei glattem (d.h. differenzierbarem) f :

$$|p(x) - f(x)| \le \frac{h}{2} \cdot \max_{x_1 \le \xi \le x_n} \{|f'(\xi)|\} \quad \text{wobei} \quad h = \max_{1 \le i \le n} \{x_{i+1} - x_i\}$$

- Das am weitesten verbreitete Interpolationsverfahren.
  - Warum sind Graphikkarten so leistungsfähig?
  - Lineare Interpolation in Hardware + höchst parallel!

- Gegeben
  - **Stützstellen** $\{ x_1, x_2, \dots , x_n\}$ und **Stützwerte** $\{ y_1, y_2, \dots , y_n\}$ (Abtastwerte einer unbekannten zu rekonstruierenden Funktion f)
  - Gesucht: kontinuierliche Rekonstruktion $p(x)$ (Näherung von $f(x)$)

- Stückweise lineare Interpolation:
  - Um den Wert an der Stelle x anzunähern, suche die nächst gelegene linke und rechte Stützstelle $x_i \le x \le x_{i+1}$
  - Interpoliere im Intervall $[x_i , x_{i+1}]$ linear



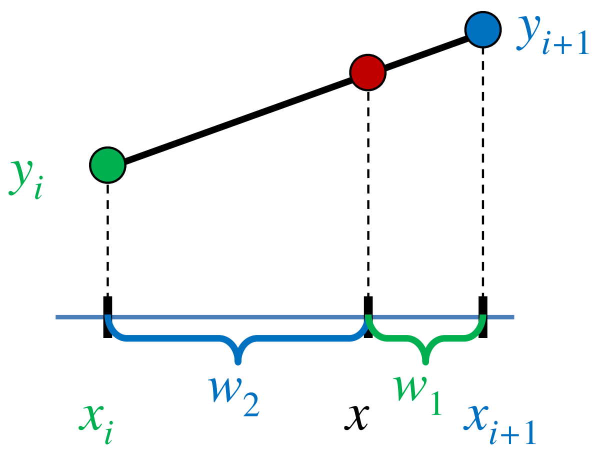
- Stückweise lineare Interpolation:
  - Um den Wert an der Stelle x anzunähern, suche die nächst gelegen Stützstellen $x_i \leq x \leq x_{i+1}$
  - Interpolierter Wert ist:

$$p(x) = m_i(x - x_i) + y_i \quad \text{mit} \quad m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

dy/dx

  - Alternativ: gewichtetes Mittel von y_i und y_{i+1} :

$$p(x) = w_1 y_i + w_2 y_{i+1} \quad \text{mit} \quad w_1 = \frac{x_{i+1} - x}{x_{i+1} - x_i} \quad \text{und} \quad w_2 = \frac{x - x_i}{x_{i+1} - x_i} = 1 - w_1$$



- Gegeben:
  Stützstellen $\{x_1, x_2, \ldots, x_n\}$ und -werte $\{y_1, y_2, \ldots, y_n\}$
- Idee:
  1. Schätze an jeder Stelle die erste Ableitung (Steigung):
     $\rightarrow \{ y_1', y_2', \ldots, y_n' \}$
  2. Finde auf jedem Teilintervall $[x_i, x_{i+1}]$ das (eindeutige) kubische Polynom p_i welches in den beiden Endpunkten die Stützwerte und die geschätzten Ableitungen interpoliert:

abstand nach rechts^2

$$p(x) = a_0(x_{i+1}-x)^3 + a_1(x_{i+1}-x)^2(x-x_i)$$
$$a_2(x_{i+1}-x)(x-x_i)^2 + a_3(x-x_i)^3$$

abstand links ^2

$$a_0 = \frac{y_i}{(x_{i+1}-x_i)^3}, \quad a_1 = 3a_0 + \frac{y_i'}{(x_{i+1}-x_i)^2},$$

$$a_3 = \frac{y_{i+1}}{(x_{i+1}-x_i)^3}, \quad a_2 = 3a_3 - \frac{y_{i+1}'}{(x_{i+1}-x_i)^2},$$

From Wikipedia

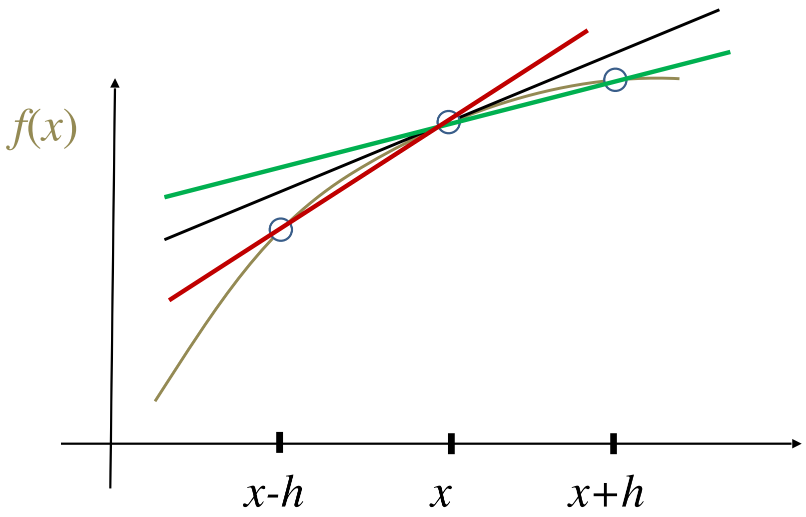
- Ed Catmull ist ein amerikanischer Informatiker, der zu vielen wichtigen Entwicklungen in der Computergraphik beigetragen hat.
- Er ist vierfacher Oscar-Preisträger.
- Er studierte Physik und Computer Science an der University of Utah
- 1979 Arbeit für George Lucas bei Lucasfilm
- 1986 Steve Jobs übernimmt *Lucasfilm's digital division* und *Pixar*.
- 2020 Turing Award
- Catmull ist Chef-Entwickler, u.a das rendering system RenderMan
- Animationsfilme Filme *Toy Story*, *Finding Nemo, …*
- zur Zeit Präsident von *Pixar* und *Disney Animation Studios*

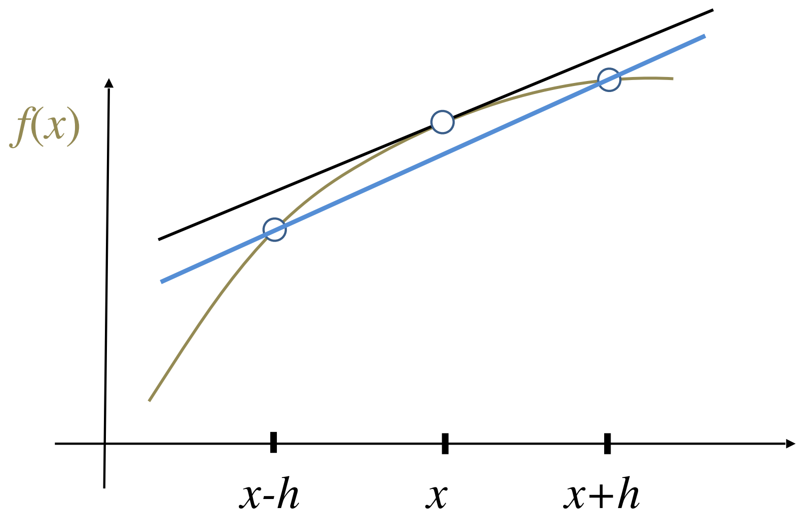
https://de.wikipedia.org/wiki/Edwin_Catmull

Wie schätzt man die Ableitungen?

- Mittels Differenzenquotienten:
  - Vorwärts-Differenz $$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$
  - Rückwärts-Differenz $$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h)$$
  - Zentrale Differenz $$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

h<1 führt zu h^2 be





Wie schätzt man die Ableitungen (allgemein)?

- Beispiel zu Differenzenquotienten

$$f(x) = \exp(x), \quad x_0 = 0.5, \quad f'(x_0) = 1.648721271\cdots$$

- Tabelle der absoluten Fehler für Schrittweiten $h=2^{-3}, 2^{-4}, \ldots$

Schrittweite	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}
Vorwärts-D.	.1075	.0526	.0260	$.129\cdot10^{-1}$	$.646\cdot10^{-2}$	$.322\cdot10^{-2}$	$.161\cdot10^{-2}$	$.804\cdot10^{-3}$
Rückwärts-D.	.0908	.0505	.0255	$.128\cdot10^{-1}$	$.642\cdot10^{-2}$	$.321\cdot10^{-2}$	$.161\cdot10^{-2}$	$.804\cdot10^{-3}$
Zentrale D.	.0043	.0011	.00026	$.671\cdot10^{-4}$	$.168\cdot10^{-4}$	$.411\cdot10^{-5}$	$.905\cdot10^{-6}$	$.137\cdot10^{-6}$

faktor 1000 besser

- Erkenntnis: Bei jeder Halbierung der Schrittweite:
  - Vorwärts- und Rückwärts-Differenz: halbiert sich der Fehler
  - Zentraler Differenz wird der Fehler jeweils geviertelt

Catmull-Rom-Interpolant: lokal und glatt

Schätzen der Ableitung bei Catmull-Rom (1. Schritt):

- ~~Vorwärtsdifferenz~~ $$y_i'{}_{fw} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

- ~~Rückwärtsdifferenz~~ $$y_i'{}_{bw} = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

- Zentrale Differenz (einfach) $$y_i' = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$$

- Zentrale Differenz (exakt):
  (gewichtetes Mittel von Vorwärts- und Rückwärts-D.):

also steigungsdreieck zwischen elementen

$$y_i' = \frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} \cdot y_i'{}_{fw} + \frac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} \cdot y_i'{}_{bw}$$

also korrektur durch distanz an den rändern der stützst

Im äquidistanten Fall sind die beiden zentralen D. identisch.

Zusammenfassung

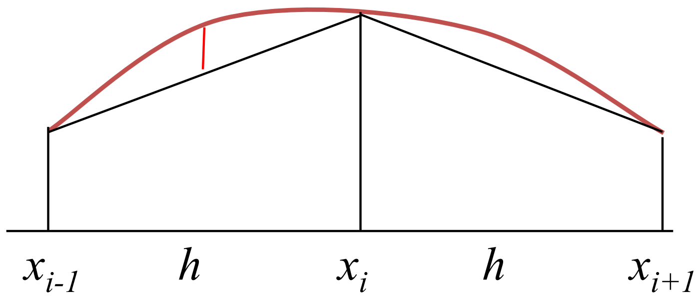
Gegeben: Stützstellen $\{x_1, x_2, \ldots, x_n\}$ und -werte $\{y_1, y_2, \ldots, y_n\}$

1. Schätze an inneren Stützstellen die Ableitung durch zentrale Differenz → $\{y_2', y_3', \ldots, y_{n-1}'\}$
2. Schätze die Ableitung in den beiden Endpunkten y_1', y_n'
   - zB Vorwärts- bzw. Rückwärts-Differenz
   - NB: es gibt bessere Verfahren
3. Finde auf jedem Teilintervall $[x_i, x_{i+1}]$ das (eindeutige) kubische Polynom p_i welches in den beiden Endpunkten die Stützwerte und die geschätzten Ableitungen interpoliert

Formeln siehe oben!

- Annahme äquidistante Stützstellen Schrittweite h und zu rekonstruierte Funktion f ist genügend differenzierbar
- Nearest Neighbor: $O(h)$ genauer: $\leq |f'(\xi)|/2 \cdot h$ (konstante Interpolation)
- Lineare Interpolation: $O(h^2)$ genauer: $\leq |f''(\xi)|/8 \cdot h^2$
- Catmull Rom: $O(h^3)$ genauer: $\leq |f'''(\xi)|/24 \cdot h^3$
- B-Spline: $O(h^4)$
- Auch im allgemeinen Fall gültig, wenn h = maximaler Abstand aufeinander folgender Stützstellen

- $|f(x) - p(x)| \leq h^2/8 \max_\xi \{ |f''(\xi)| \}$



- Anwendung:
  - Tabellierung von Funktionswerten
  - Look-up-tables

- Beispiel: Tabelle (look-up-table) für $\sin(x)$

  Gewünschte Genauigkeit: $\varepsilon_0 = 10^{-3}$, $\varepsilon_1 = 10^{-6}$

  Aus Fehlerabschätzung folgt für Schrittweite h :

  $h_0 \leq 9 \cdot 10^{-2}$ bzw. $h_1 \leq 2.8 \cdot 10^{-3}$

- Die Funktion $f(x) = \sin(\pi x)\ ,\ I = [0,1]$

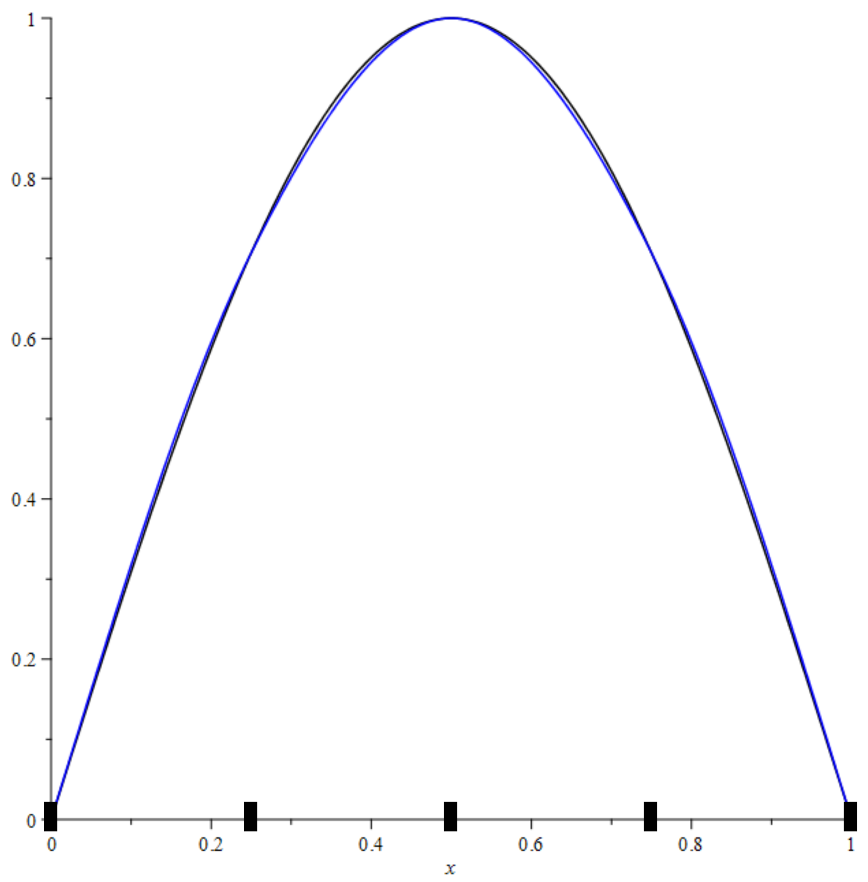
  wird äquidistant abgetastet mit Schrittweite h
- Der Interpolationsfehler
  für linearen, Catmull-Rom- und B-Spline-Interpolanten

estimated order of convergence

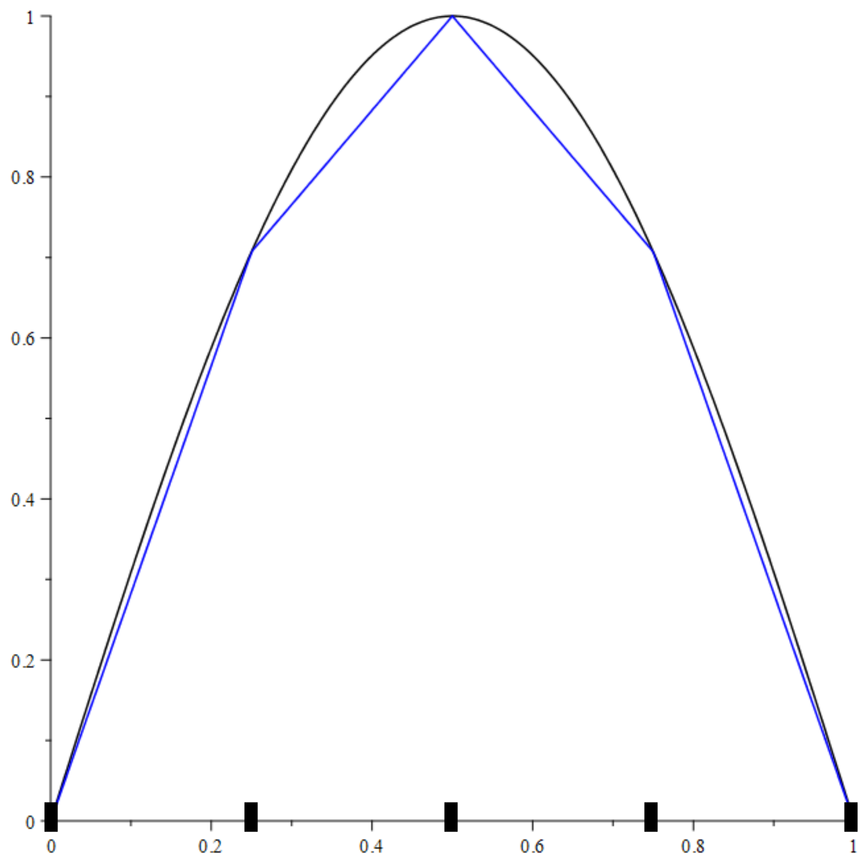
h	*linear*	*EOC*	*Catmull-R.*	*EOC*	*B-Spline*	*EOC*
	$O(h^2)$	**4**	$O(h^3)$	**8**	$O(h^4)$	**16**
0.25	0.0703		8.80E-03		1,06E-03	
0.125	0.0188	3.73	9.90E-04	8.88	6.31E-05	16.86
0.0625	4.79E-3	3.93	1.21E-04	8.13	3.89E-06	16.22
0.03125	1.20E-3	3.98	1.52E-05	8.03	2.42E-07	16.06
0.015625	3.01E-4	4.00	1.89E-06	8.01	1.51E-08	16.02

- Die Funktion $f(x) = \sin(\pi x)$, $I = [0,1]$
  samples: $\{ 0.0, 0.25, 0.5, 0.75, 1.0\}$ $(h = 1/4)$

Catmull-Rom-Interpolant



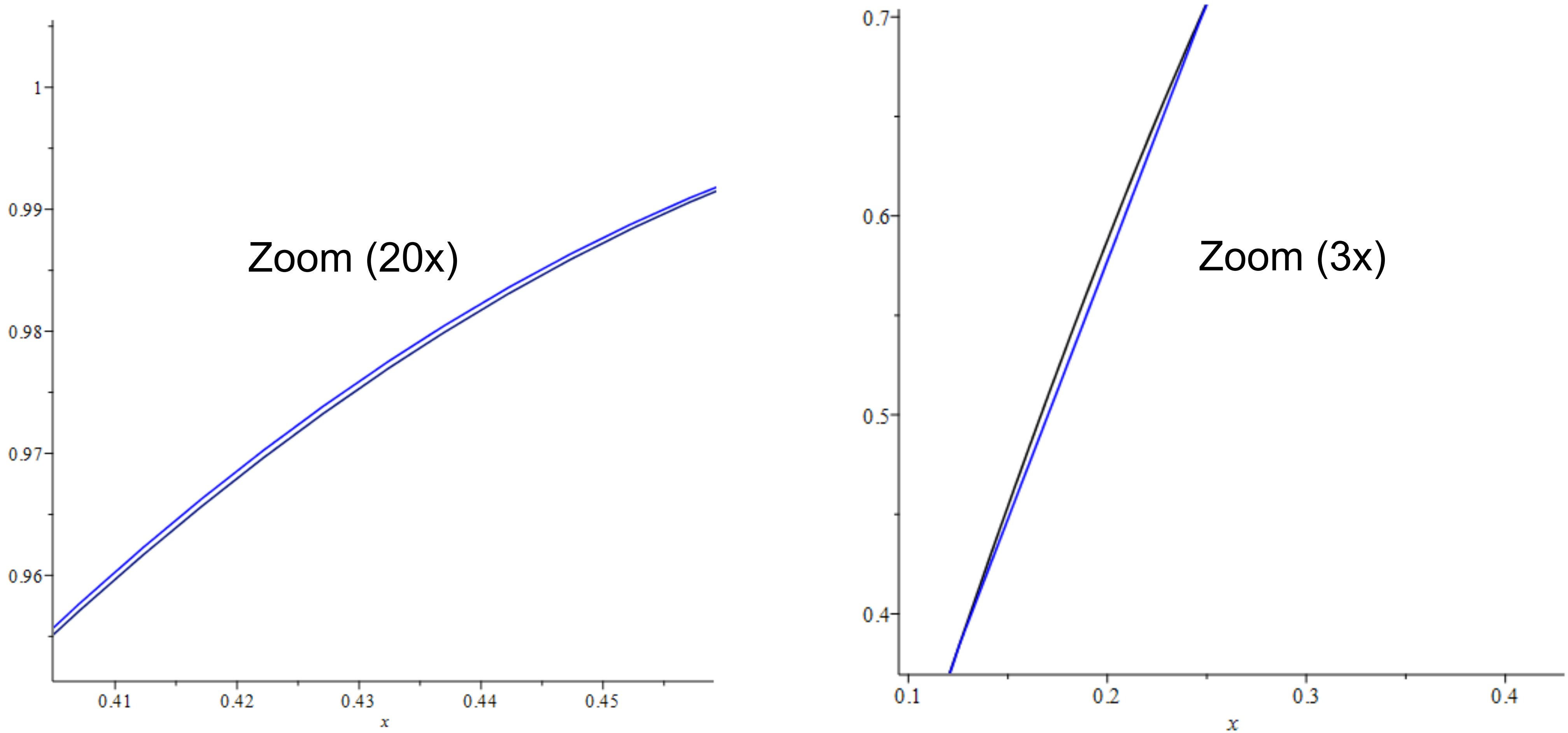
Linearer Interpolant



- Die Funktion $f(x) = \sin(\pi x)$, $I = [0,1]$
  samples: $\{ 0.0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75. 0.875, 1.0 \}$ $(h = 1/8)$

Catmull-Rom-Interpolant

Linearer Interpolant



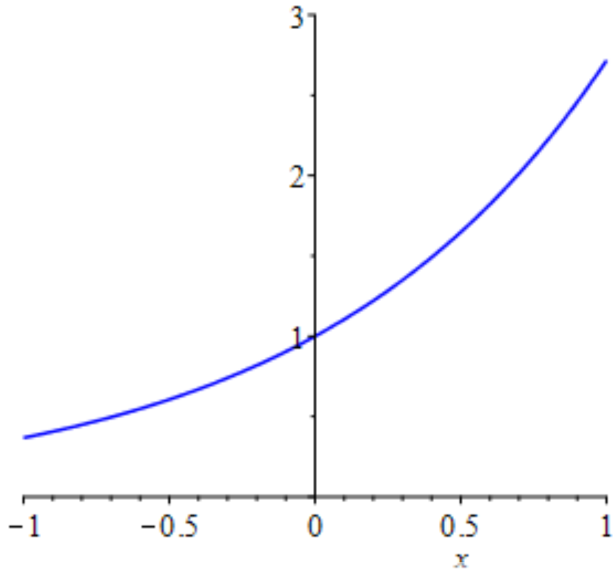
Die Funktion $f(x) = \sin(\pi x)\ ,\ I = [0,1]$

wird äquidistant abgetastet mit Schrittweite h

Interpolationsfehler
für linearen, Catmull-Rom- und B-Spline-Interpolanten

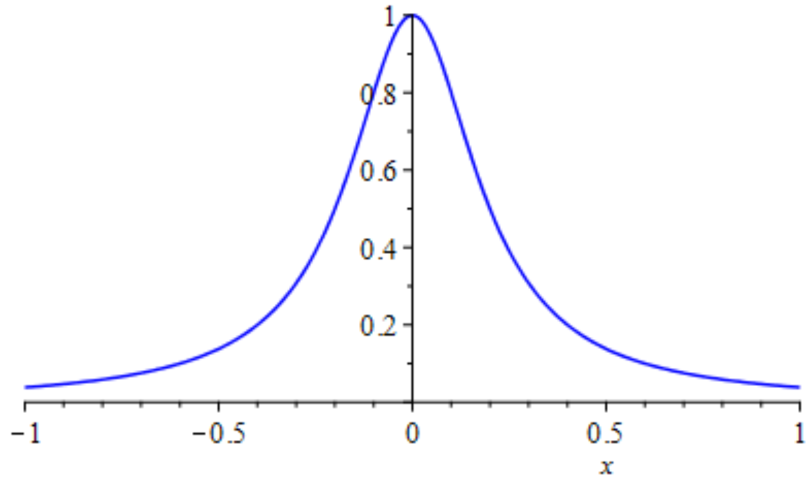
h	*linear*	*EOC*	*Catmull-R.*	*EOC*	*B-Spline*	*EOC*
	O(h^2)	*4*	*O(h^3)*	*8*	*O(h^4)*	*16*
2^{-2}	0.07		0.010		6,96E-05	
2^{-3}	1.9E-02	3.68	1.75E-03	5.7	4,21E-06	16,54
2^{-4}	4.8E-03	3.96	2.4E-04	7.3	2,60E-07	16,17
2^{-5}	1.2E-03	4.0	3.0E-05	8,0	1,75E-08	14,85
2^{-6}	3.0E-04	4.0	3.75E-06	8.0		

- $f(x) = \exp(x)\,,\ I = [-1,1]$
- wird äquidistant abgetastet mit Schrittweite h
- Interpolationsfehler: für linearen, Catmull-Rom- und B-Spline-Interpolanten



h	linear	EOC	Catmull-R.	EOC	B-Spline	EOC
	$O(h^2)$	**4**	$O(h^3)$	**8**	$O(h^4)$	**16**
0.4			2,20E-03		1,67E-04	
0.2	1,24E-02		3,11E-04	7,07	1,09E-05	15,32
0.1	3,24E-03	3,83	4,12E-05	7,55	6,94E-07	15,71
0.05	8,30E-04	3,90	5,30E-06	7,77	4,40E-08	15,77
0.025	2,10E-04	3,95	6,74E-07	7,86		

- $f(x) = 1/(1+25x^2)$, $I = [-1,1]$ wird äquidistant abgetastet mit Schrittweite h



- Interpolationsfehler für linearen, Catmull-Rom- und B-Spline-Interpolanten

h	linear		Catmull-R.		B-Spline	
	$O(h^2)$	**4**	$O(h^3)$	**8**	$O(h^4)$	**16**
0.4	0.5		4.50E-01		0.422	
0.2	6.75E-02	7.41	1.82E-02	24.78	2.20E-02	19.29
0.1	4.18E-02	1.61	1.12E-02	1.63	3.18E-03	6.90
0.05	1.41E-02	2.98	1.74E-03	6.43	2.77E-04	11.49
0.025	3.80E-03	3.69	1.58E-04	11.02	1.61E-05	17.21

- Lokale Verfahren vs. globale Verfahren
- Lokale Verfahren
  - *nearest neighbor* : unstetig, $O(h)$
  - linear: stetig, $O(h^2)$
  - Catmull-Rom: glatt (diff.-bar, C^1), $O(h^3)$
  - kubische Splines: glatt (diff.-, bar, C^2), $O(h^4)$

- Schätzen von Ableitungen : vorwärts, rückwärts, zentral
- Rechenaufwand:
  - Aufwand für Suche …
  - der Rest ist $O(1)$

- Globale Verfahren
  - Polynominterpolation
  - B-Spline-Interpolation

- Polynome vom Grad n-1 bilden einen n-dimensionalen Vektorraum.

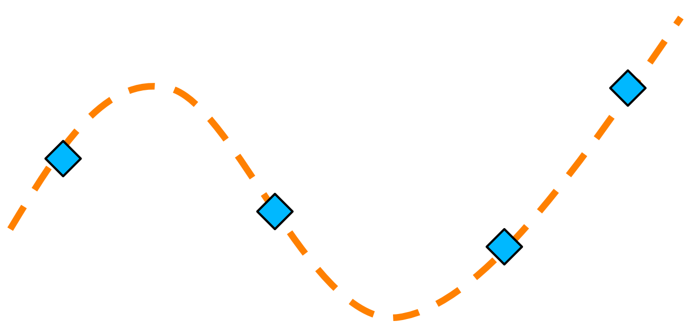
$$\mathbb{P}_n = \text{span}\left\{1, x, x^2, \ldots, x^{n-1}\right\} = \left\{\sum_{i=0}^{n-1} c_i x^i\right\}$$

Taylorbasis
Monombasis

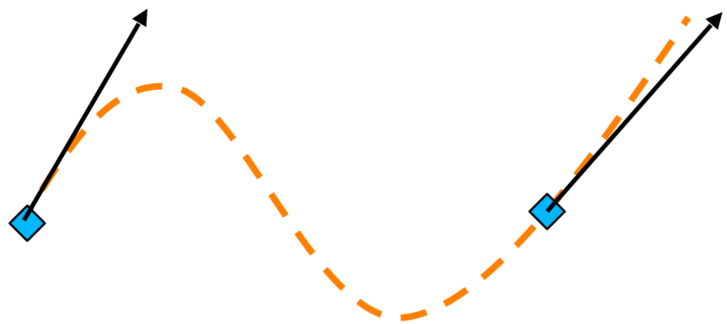
- Polynome werden in der Informatik sehr gerne verwendet, weil deren Auswertung für Computer einfach ist
(→ Hornerschema s.u.)
- im Gegensatz zu anderen mathematischen Funktionen.
- Polynome werden deshalb häufig zur Approximation anderer (komplizierterer) Funktionen eingesetzt
(→ Taylorentwicklung)
- Die Taylorbasis (oder Monombasis) oben ist nur eine Möglichkeit den Polynomraum darzustellen - Varianten später.

Interpolation: von der zu approximierenden Funktion werden nur Werte an (wenigen) Stützstellen verwendet, die Werte dazwischen werden durch ein „Polynominterpolation“ berechnet.

- Werden nur die Punkte selbst verwendet, spricht man von „Lagrange-Interpolation“.
- Werden zusätzliche auch Informationen über die Richtungen (Ableitung der Funktion) verwendet, dann spricht man von „Hermite-Interpolation“ bzw. von Interpolation mit doppelten Stützstellen.



Lagrange Interpolation



Hermite Interpolation

- Für die (Lagrange-)Polynominterpolanten sind gegeben:
  - Stütz**stellen** $\{ x_1, x_2, \dots , x_n\}$
  - Stütz**werte** $\{ y_1, y_2, \dots , y_n\}$

- Ein Polynom $p(x) = \sum_{i=0}^{n-1} c_i x^i$ interpoliert diese Werte

sofern $p(x_1) = y_1, \dots , p(x_n) = y_n$

das heißt:

$$
\begin{aligned}
p(x_1) &= \sum_{i=0}^{n-1} c_i x_1{}^i = y_1 \\
p(x_2) &= \sum_{i=0}^{n-1} c_i x_2{}^i = y_2 \\
&\vdots \\
p(x_n) &= \sum_{i=0}^{n-1} c_i x_n{}^i = y_n
\end{aligned}
$$

- $p(x) = c_0 + c_1 x + \ldots + c_{n\text{-}1} x^{n\text{-}1}$ eingesetzt in die n Interpolationsbedingungen führt auf ein lineares Gleichungssystem mit n Gleichungen für die n unbekannten Koeffizienten c_i.

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}}_{\boldsymbol{c}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\boldsymbol{y}}$$

- $\boldsymbol{A}$ heißt Vandermonde-Matrix,
- Vandermonde-Determinate $\det(\boldsymbol{A}) = \prod_{1 \le j < k \le n} (x_j - x_k) \neq 0$

- Die Matrix A ist die sogenannte Vandermonde-Matrix.
  - Sie ist quadratisch, voll besetzt, nicht singulär (sofern $x_i \neq x_j$)

    aber
  - schon bei moderatem n (und äquidistanten Stützstellen) sehr schlecht konditioniert

n	$\kappa(A_{\text{Vandermonde}})$		
3	13.91		
4	154.46		
5	2592.89		
6	57688.70		
7	1597316.16		
8	52937735		
9	2043725290		
10	9.00778E10		
11	4.46282E12		
12	2.45502E14		
13	1.48460E16		
14	9.79006E17		
15	7.02701E19		

- Das System $A\,c = y$ ist eindeutig lösbar, da $\det(A) \neq 0$:

- Folgerung:
Zu n Stützstellen gibt es ein eindeutig bestimmtes Interpolationspolynom vom Grad n-1

- Man löse das Gleichungssystem $A\,c = y$ mit der Vandermondematrix A z.B. mit LR-Zerlegung (oder Gauss-Elimination)
  - Aufwand $O(n^3)$
  - Numerisch sehr problematisch (da schlecht konditioniert)
  - *Gibt's, denn da nichts besseres?*

Divide-et-Impera Ansatz

Rekursion

Wie könnte es gehen?

- $p(x)$ interpoliert $x_1, x_2, \ldots, x_k$
- $q(x)$ interpoliert $x_{k+1}, x_{k+2}, \ldots, x_{2k}$
- $F(p,q)$ interpoliert $x_1, x_2, \ldots, x_k, x_{k+1}, x_{k+2}, \ldots, x_{2k}$

So geht es leider nicht!

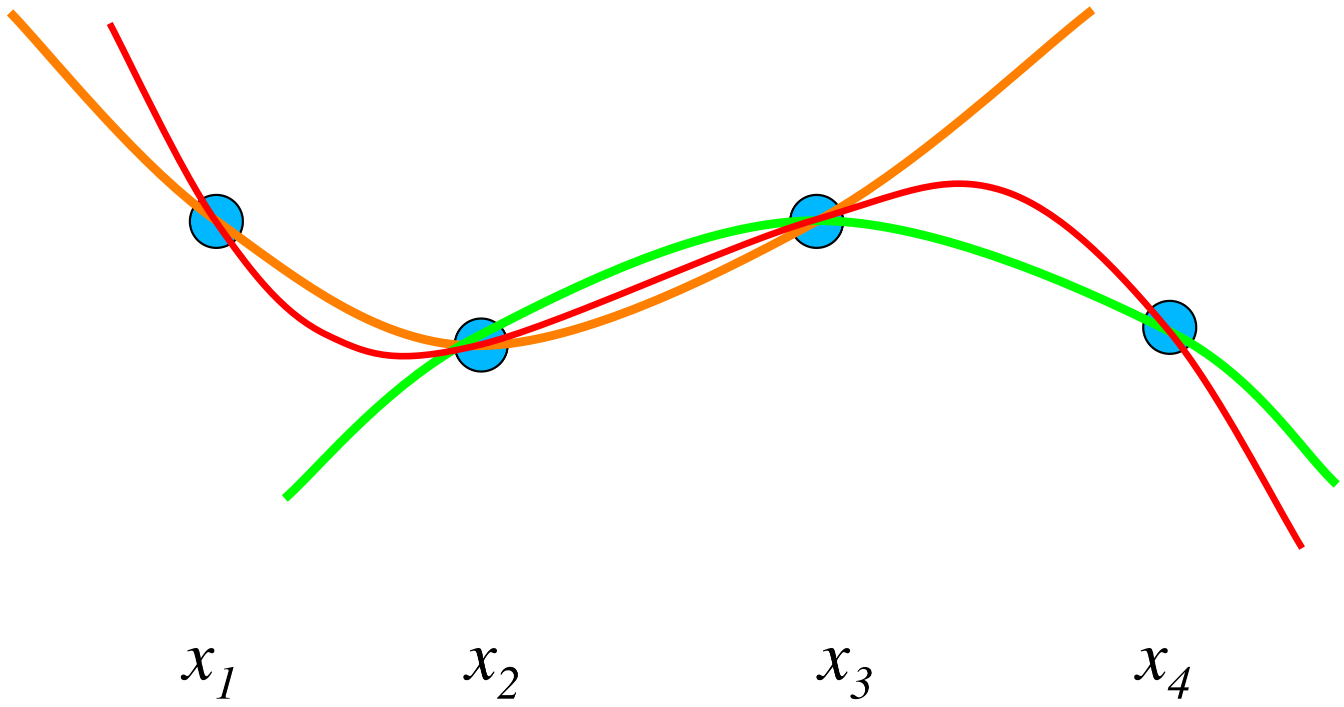
(Es wäre zu schön gewesen)

Ansatz

$p_{1,k\text{-}1}(x)$ interpoliert an $x_1, x_2, \ldots, x_k$ (Polynomgrad k-1)

$p_{2,k\text{-}1}(x)$ interpoliert an $x_2, x_3, \ldots, x_{k+1}$ (Polynomgrad k-1)

Brechne daraus $p_{1,k}$ das an $x_1, x_2, \ldots, x_{k+1}$ interpoliert (Polynomgrad k)



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

- Warum funktioniert das:
  - Ist $x=x_i$, dann „übernimmt“ das linke Polynom.
  - Ist $x=x_{i+k}$, dann „übernimmt“ das rechte Polynom.
  - Bei den Stützstellen dazwischen „kooperieren“ beide Polynome.
  - Die Sortierung der Stützstellen ist nicht relevant.
- Die Formel kann man als Rekursion für (reelle/komplexe) Zahlen, also die Werte der Polynome an der Stelle x lesen
- Es ist aber genau so gut eine Formel für Funktionen:
  - Eine Funktion, die aus zwei Funktionen eine dritte berechnet.
- Terminierung der Rekursion:
  - Die Interpolation einer einzigen Stützstelle (x_i, y_i) liefern trivialerweise die konstanten Polynome $p_{i,0}(x) = y_i$

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

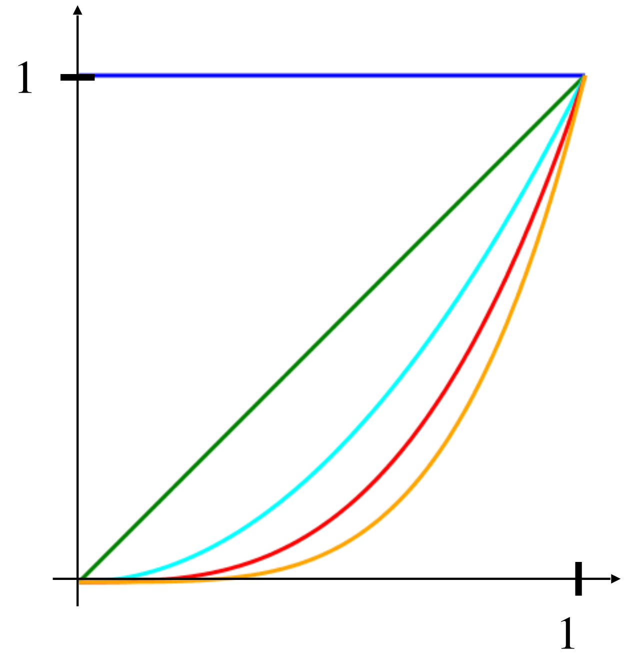
Wenn man einen Datentyp „Polynom“ (`class Polynom { .... }` in Python/C++) hat, kann man diese Formel zur Konstruktion des interpolierenden Polynoms verwenden.

- Besonders attraktiv wäre dies natürlich in funktionalen Programmierprachen (wie Lisp, Maple, ...).
- Am einfachsten und effizientesten geht es, wenn man eine geeignete Datenstruktur (mathematisch gesehen: Polynombasis) verwendet, nämlich die Polynomdarstellung nach Newton. Dann kommt man auf die Newtonsche Interpolationsformel (siehe heute am Ende der Vorlesung).

Aber zunächst ist die Rekursionsformel noch unbrauchbar:

- Der Aufwand explodiert, wenn man die Rekursionsformel naiv verwendet: Bei Polynomgrad n sind $O(2^n)$ Aufrufe erforderlich! Das ist viel zu teuer.
- Nur wenn man nutzt, dass die Rekursion immer wieder die gleichen Polynome berechnet, kann man ein effizienten Algorithmus erhalten.
- Das führt auf das Aitken-Neville-Dreiecksschema und die Newtoninterpolation: siehe unten.

- Bisher : Darstellung in der
  - Taylor-Basis bzw. Monom-Basis $\{1, x, x^2, x^3, ..., x^{n-1}\}$
  - zu speichern: Die n Koeffizienten
  - Effiziente Auswertung mit dem Horner-Schema (s.u.)



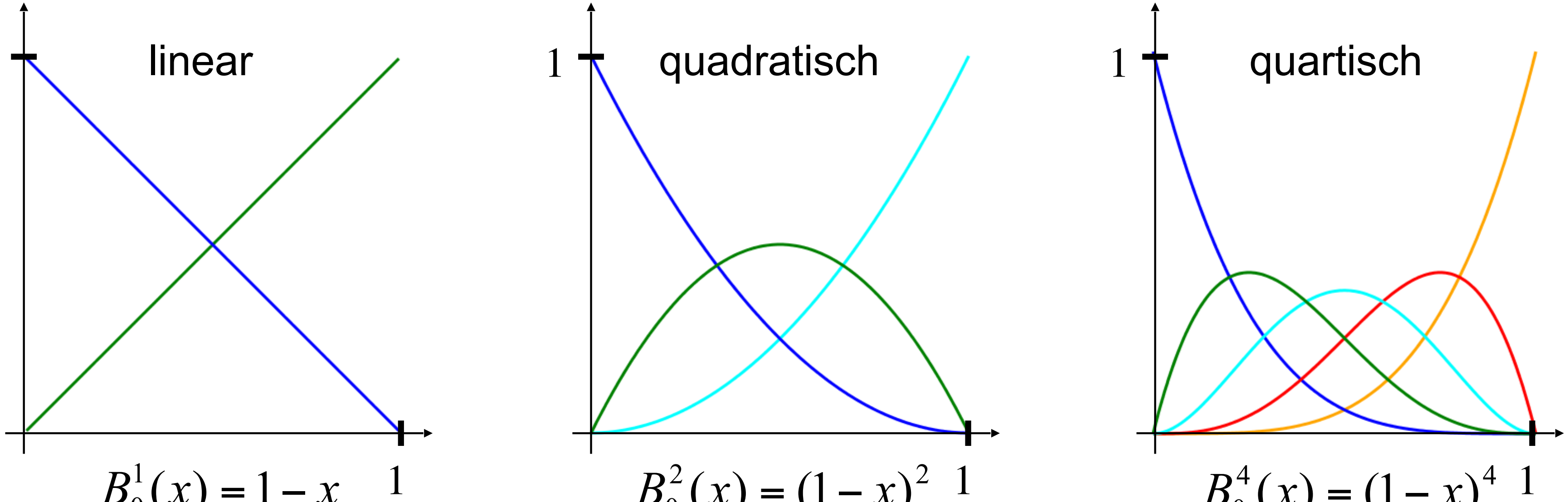
- **Alternativen:**

1. Bernstein-Polynome
2. Lagrange-Polynome
3. Newton-Polynome

- bilden jeweils eine Basis von $\mathbb{P}_n$

- Alternative 1: Basis aus **Bernstein**-Polynomen

$$B_i^{n-1}(x) = \binom{n-1}{i}(1-x)^{n-1-i}x^i \qquad \mathbb{P}_{n-1} = \operatorname{span}\left\{B_0^{n-1}(x), B_1^{n-1}(x), \ldots B_{n-1}^{n-1}(x)\right\}$$



$B_0^1(x) = 1-x$

$B_1^1(x) = x$

$B_0^2(x) = (1-x)^2$

$B_1^2(x) = 2(1-x)x$

$B_2^2(x) = x^2$

$B_0^4(x) = (1-x)^4$

$B_1^4(x) = 4(1-x)^3 x$

$B_2^4(x) = 6(1-x)^2 x^2$

$B_3^4(x) = 4(1-x)x^3$

$B_4^4(x) = x^4$

- **Eigenschaften:**
  - $B_0^n(x) \geq 0$ falls $0 < x < 1$
  - $\sum_{i=0}^n B_i^n(x) = 1$

- Alternative 1: Basis aus **Bernstein**-Polynomen

$$B_i^{n-1}(x) = \binom{n-1}{i}(1-x)^{n-1-i}x^i \quad \mathbb{P}_{n-1} = \mathrm{span}\left\{B_0^{n-1}(x), B_1^{n-1}(x), \ldots B_{n-1}^{n-1}(x)\right\}$$

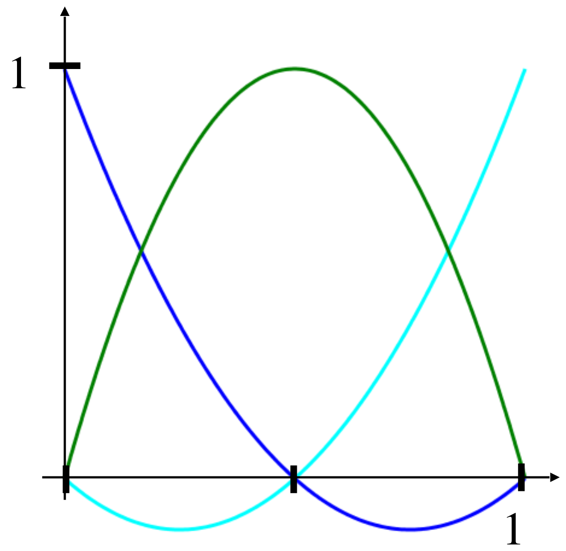
  - Basis für Polynome vom Grad n-1
  - Sehr beliebt in Graphik beim geometrischen Modellieren

    → Deshalb kommen wir später darauf noch mal zurück
  - Für Polynominterpolation eher **nicht** geeignet

- Alternative 2: Basis aus **Lagrange**-Polynomen zu gegebenen Stützstellen $\{ x_1, x_2, \ldots , x_n\}$

$$L_i(x) = \frac{(x-x_1)\cdot\ldots\cdot(x-x_{i-1})\cdot(x-x_{i+1})\cdot\ldots\cdot(x-x_n)}{(x_i-x_1)\cdot\ldots\cdot(x_i-x_{i-1})\cdot(x_i-x_{i+1})\cdot\ldots\cdot(x_i-x_n)} \quad (i=1..n)$$

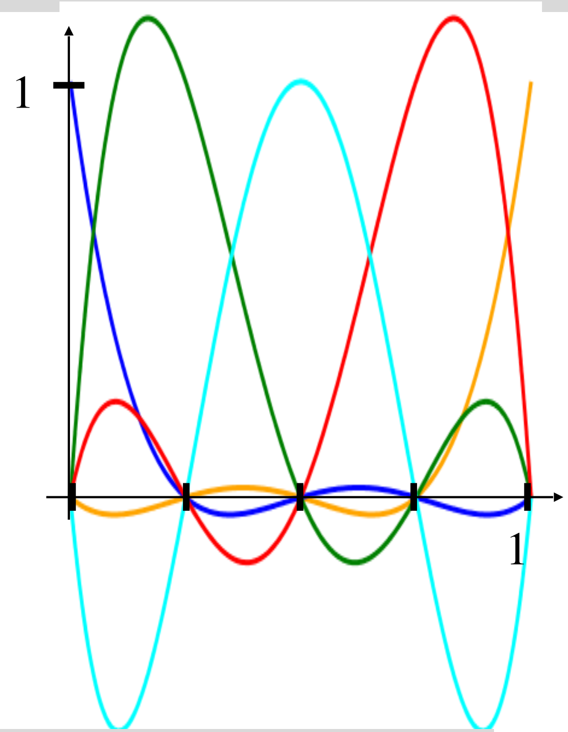
$x_1 = 0,$
$x_2 = 0.5,$
$x_3 = 1;$



$$L_1(x) = \frac{(x-0.5)(x-1)}{(0-0.5)(0-1)} = 2x^2 - 3x + 1$$

$$L_2(x) = \frac{(x-0)(x-1)}{(0.5-0)(0.5-1)} = -4x^2 + 4x$$

$$L_3(x) = \frac{(x-0)(x-0.5)}{(1-0)(1-0.5)} = 2x^2 - x$$



$x_1 = 0,$
$x_2 = 0.25,$
$x_3 = 0.5,$
$x_4 = 0.75,$
$x_5 = 1;$

$$L_i(x_j) = \begin{cases} 1 & \text{falls} \quad j = i \\ 0 & \text{sonst} \end{cases}$$

- Alternative 2: Basis aus **Lagrange**-Polynomen zu gegebenen Stützstellen $\{x_1, x_2, \ldots, x_n\}$

$$L_i(x) = \frac{(x-x_1)\cdot\ldots\cdot(x-x_{i-1})\cdot(x-x_{i+1})\cdot\ldots\cdot(x-x_n)}{(x_i-x_1)\cdot\ldots\cdot(x_i-x_{i-1})\cdot(x_i-x_{i+1})\cdot\ldots\cdot(x_i-x_n)} \quad (i=1..n)$$

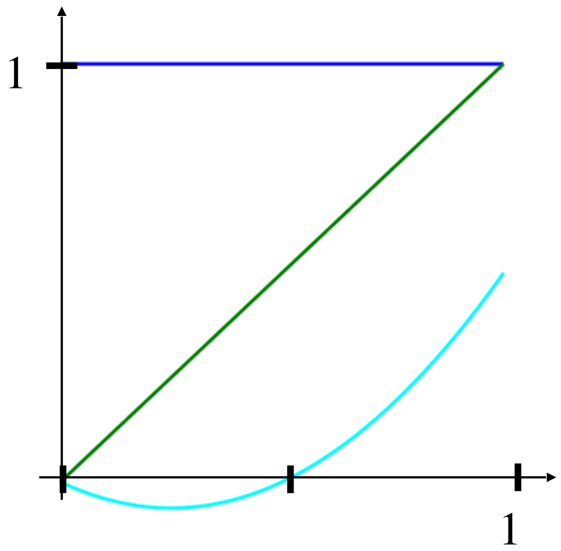
- Eigenschaften der Lagrange-Polynome:
  - Die L_i sind linear unabhängig und bilden eine Basis von $\mathbb{P}_n$
  - $L_i(x_j)=\delta_{ik}$ (= Kroneckersymbol)
  - → das (eindeutige) Interpolationspolynom ist gegeben durch

$$p(x) = \sum_{i=1}^{n} y_i L_i(x)$$

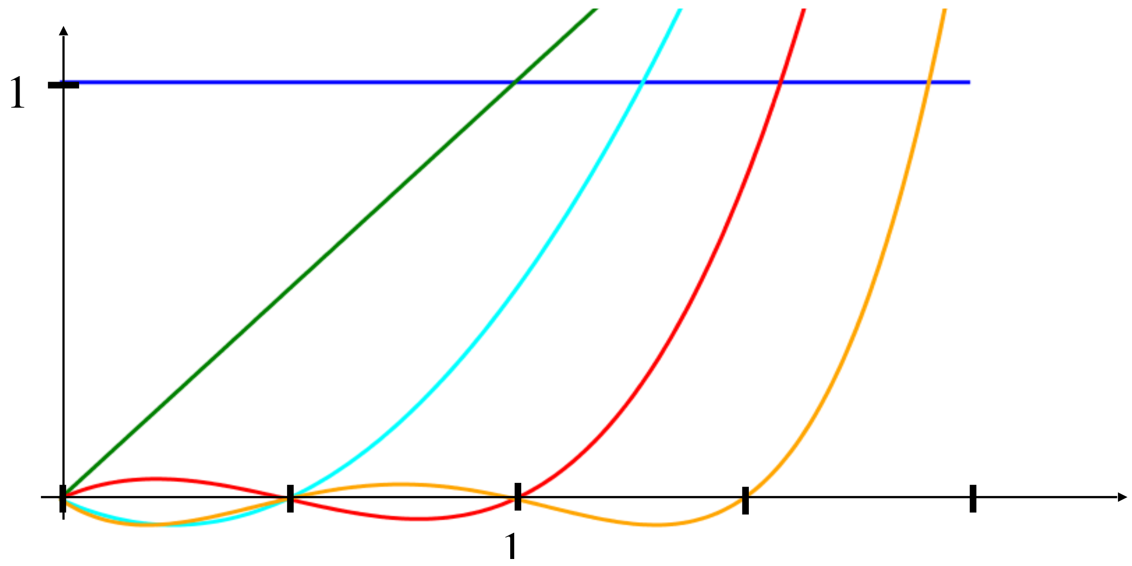
  - Hauptanwendung der Lagrange-Polynome sind **theoretische Überlegungen** (z.B. zur Entwicklung von speziellen Algorithmen). Für die algorithmische **Berechnung** des Interpolationspolynoms selbst sind sie eher schlecht geeignet!

- Alternative 3: Basis aus **Newton**-Polynomen zu gegebenen Stützstellen $\{ x_1, x_2, \dots , x_n\}$

$$N_i(x) = (x - x_1)\cdot(x - x_2)\cdot\ldots\cdot(x - x_{i-1}) \quad (i = 1..n)$$



$x_1 = 0,\ x_2 = 0.5,\ x_3 = 1;$



$x_1 = 0,\ x_2 = 0.5,\ x_3 = 1,\ x_4 = 1.5,\ x_5 = 2;$

$$N_1(x) = 1,$$
$$N_2(x) = (x - 0) = x,$$
$$N_3(x) = (x - 0)(x - 0.5) = x^2 - 0.5x;$$

$$\cdots,$$
$$N_4(x) = (x - 0)(x - 0.5)(x - 1),$$
$$N_5(x) = (x - 0)(x - 0.5)(x - 1)(x - 1.5);$$

- Alternative 3: Basis aus **Newton**-Polynomen
  zu gegebenen Stützstellen $\{ x_1, x_2, \ldots , x_n \}$

$$N_i(x) = (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_{i-1}) \quad (i = 1..n)$$

Verwende die Basis $\{ N_1(x) , N_2(x) , \ldots , N_n(x) \}$ zur Lösung der Interpolationsaufgabe, Ansatz:

$$p(x) = a_0 + a_1(x\text{-}x_1) + a_2(x\text{-}x_1)(x\text{-}x_2) + \ldots.\ a_{n\text{-}1}(x\text{-}x_1)(x\text{-}x_2)\ldots(x\text{-}x_{n\text{-}1})$$

- Mit dieser Basis läßt sich des Interpolationsproblem effizient und stabil lösen
  $\rightarrow$ Algorithmus von ***Aitken-Neville***
- Weiterer Vorteil:
  In dieser Basis ist das Interpolationspolynom effizient für alle x (mit einem Horner-artigen Schema) auswertbar.

- Alternative 3: Basis aus **Newton**-Polynomen zu gegebenen Stützstellen $\{ x_1, x_2, \dots , x_n\}$

$$p(x) = a_0+a_1(x-x_1)+a_2(x-x_1)(x-x_2)+\dots\; a_{n-1}(x-x_1)(x-x_2)\dots(x-x_{n-1})$$

- Die Interpolationsbedingungen $p(x_j) = y_j$ führen auf ein Gleichungssystem mit unterer Dreiecksmatrix:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_2 - x_1 & 0 & & 0 \\ 1 & x_3 - x_1 & (x_3 - x_1)(x_3 - x_2) & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & x_n - x_1 & (x_n - x_1)(x_n - x_2) & & N_n(x_n) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

- Der Algorithmus von **Aitken-Neville**

$$p_{i,k} = \frac{p_{i+1,k-1} - p_{i,k-1}}{x_{i+k} - x_i}$$

$$
\begin{array}{lllllll}
p_{1,0} = y_1 \;= \boldsymbol{a_0} & & & & & & \\
& \searrow & & & & & \\
& & p_{1,1} = \boldsymbol{a_1} & & & & \\
& \nearrow & & \searrow & & & \\
p_{2,0} = y_2 & & & & p_{1,2} = \boldsymbol{a_2} & & \\
& \searrow & & \nearrow & & \searrow & \\
& & p_{2,1} & & & & p_{1,3} = \boldsymbol{a_3} \\
& \nearrow & & \searrow & & & \ddots \\
p_{3,0} = y_3 & & & & & & \searrow \\
& & & & & & \qquad p_{1,n-1} = \boldsymbol{a_{n-1}} \\
\vdots & & & & & & \nearrow \\
p_{n-1,0} = y_{n-1} & & & & p_{n-2,2} & & \\
& \searrow & & \nearrow & & & \\
& & p_{n-1,1} & & & & \\
& \nearrow & & & & & \\
p_{n,0} = y_n & & & & & &
\end{array}
$$

- Der Algorithmus von Aitken-Neville

$$p(x) = a_0+a_1(x-x_1)+a_2(x-x_1)(x-x_2)+\dots.\ a_{n-1}(x-x_1)(x-x_2)\dots(x-x_{n-1})$$

- Pseudo-Code zur Berechnung der Koeffizienten a_i:

```
# Initialisierung:
for i=1..n
   p_{i,0}= y_i
# Spaltenweises Vorgehen:
for k=1..n-1
   # Durchlaufen der Spalte:
   for i=1..n-k-1
      p_{i,k} = (p_{i+1 k-1} - p_{i k-1})/(x_{i+k}-x_i)
# Rückgabe der Ergebnisse:

    a_i = p_{1,i} ; (i = 0..n-1)
```

- Aufwand: $O(n^2)$

- Effiziente Auswertung des Newton-Polynoms mit Horner-artigem Schema:
- Klassisches Hornerschema für

$$p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} = (\ldots(a_{n-1} x + a_{n-2})x + \cdots + a_1)x + a_0$$

  - PseudoCode :

```
sum = a[n-1]
for i=n-2..0
   sum = sum*x+a[i]
```

- Modifiziertes Hornerschema

$$p(x) = a_0 + a_1(x - x_1) + \ldots + a_{n-1}(x - x_1) \cdot \ldots \cdot (x - x_{n-1}) =$$

$$= (\ldots(a_{n-1}(x - x_{n-1}) + a_{n-2})(x - x_{n-2}) + \ldots + a_1)(x - x_1) + a_0$$

  - PseudoCode

```
sum = a[n-1]
for i=n-2..0
   sum = sum*(x-x[i+1])+a[i]
```

Konditionszahl (bzgl. Euklidischer Norm) für **Vandermonde-Matrix** und **„Newton"-Matrix** für n äquidistante Stützstellen: $\{0,1,2,\ldots,n\text{-}1\}$

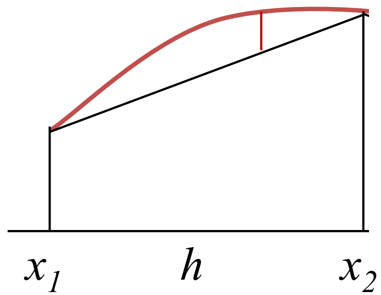
n	$\kappa(M_{\text{Vandermonde}})$	$\kappa(M_{\text{Newton}})$	$\kappa(M_N)$ / $\kappa(M_V)$
3	13.91	6.18	0.444
4	154.46	18.69	0.121
5	2592.89	74.17	0.029
6	57688.70	368.83	0.639E-2
7	1597316.16	2206.21	0.138E-2
8	52937735	15415.01	0.291E-3
9	2043725290	123173.74	0.603E-4
10	0.900778E11	1107665.81	0.123E-4
11	0.446282E13	11070260.73	0.123E-4
12	0.245502E15	121720944.9	0.496E-6
13	0.148460E17	1460178399	0.984E-7
14	0.979006E18	0.189775E11	0.194E-7
15	0.702701E20	0.265633E12	0.378E-8

- Gegeben n Stützstellen $\{x_1, x_2, \ldots, x_n\}$, die Werte sind Samples einer f : $y_i = f(x_i)$. Falls f n-mal differenzierbar ist, gilt für das Interpolationspolynom $p(x)$:

$$f(x) - p(x) = (x - x_1)\cdot(x - x_2)\cdot\ldots\cdot(x - x_n)\cdot\frac{f^{(n)}(\xi)}{n!}$$

dabei ist ξ ein Wert zwischen $\min\{x, x_1, x_2, \ldots, x_n\}$ und $\max\{x, x_1, x_2, \ldots, x_n\}$

- Speziell $n=2$: lineare Interpolation



$$f(x) - p(x) = (x - x_1)(x - x_2)\frac{f^{(2)}(\xi)}{2}$$ folglich

$$|f(x) - p(x)| \leq \frac{h^2}{8}\max\{|f^{(2)}(\xi)|\}$$

In dem Ausdruck für den Fehler kommt eine Funktion vor:

$$w(x) = (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_n)$$

- Verlauf der Funktion $|w(x)|$
- $w(x)$ oszilliert und wächst zu den Rändern hin (und über die Ränder hinaus, wenn man „Extrapolation“ machen möchte) stark an
- Dort muss mit sehr viel größeren Rekonstruktions-fehlern gerechnet werden!

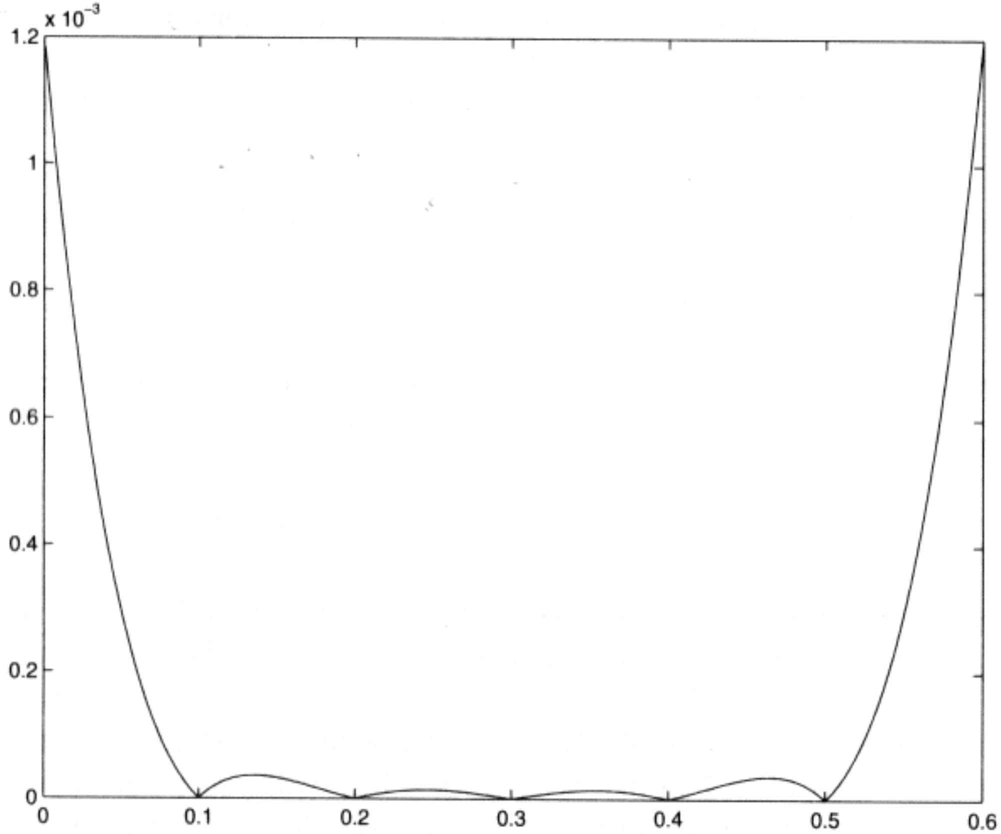
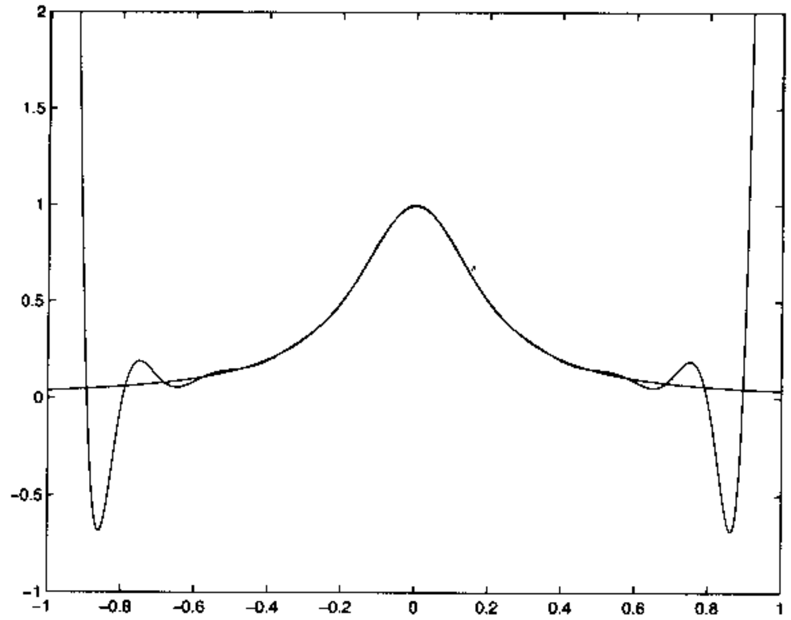


Abb. 14.5. Verlauf der Funktion $w(x)$ zu den Stützstellen 0.1, 0.2, 0.3, 0.4 und 0.5.

- Ein Beispiel : Runge Funktion $f(x) = \dfrac{1}{1+25x^2}, \quad (x \in [-1,1])$



  - 21 äquidistante Stützstellen im Intervall $[-1,1]$
  - Polynominterpolant vom Grad 20

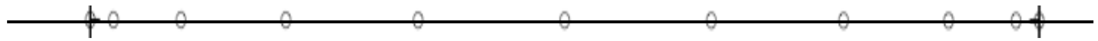
- Selbst bei Verwendung von mehr äquidistanten Stützstellen wird das Ergebnis nicht besser (**keine Konvergenz**!)

Polynominterpolation - Fehlerabschätzung

- Ein Beispiel : Runge Funktion $f(x) = \dfrac{1}{1+25x^2}, \quad (x \in [-1,1])$

★ Ausweg in diesem Fall: Verwendung nicht äquidistanter Stützstellen sog. Tschebycheff-Stützstellen (engl. Chebyshev)

$$x_i = -\cos\left(\frac{i-1}{n-1}\cdot\pi\right), \quad i = 1,2,\ldots n$$



- Chebyshev-Polynome (reichhaltige Literatur!)

- In der Praxis oft die Lösung:
  Vermeide Polynominterpolation für größeres n
- Verwende alternative Verfahren:
  - Catmull-Rom
  - B-Spline Interpolation

- Grundlage für geometrisches Modellieren
- Freiformkurven
- „Kontrollpunkte“ für den Designer anstelle von Interpolationspunkten

- Die Bernstein-Polynome vom Grad n

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \quad (i = 0,1,2,\ldots,n)$$

bilden ein Basis der Polynome (vom Grad n)

- Die Bernstein-Polynome vom Grad n

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \quad (i = 0,1,2,\ldots,n)$$

bilden ein Basis der Polynome (vom Grad n)

- Die Binomial-Koeffizienten $\binom{n}{i} = \frac{n!}{i! \cdot (n-i)!}$

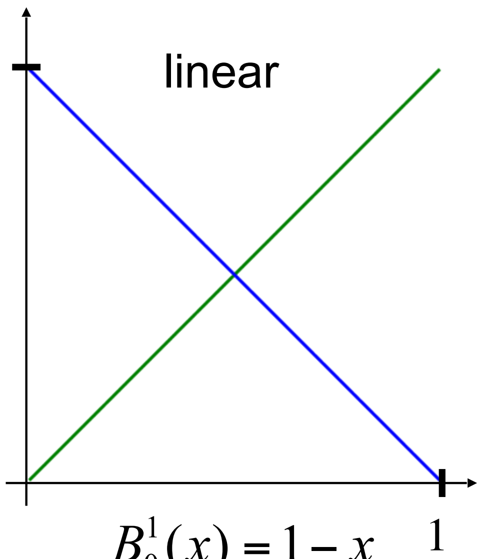
- Rekursive Definition (Pascalsches Dreieck) $\binom{n+1}{i} = \binom{n}{i} + \binom{n}{i-1}$

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

- Binomische Formel $(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} \cdot a^{n-i} b^i$

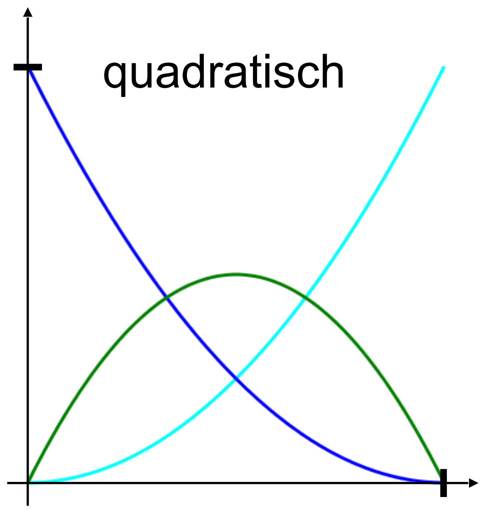
- Alternative 1: Basis aus **Bernstein**-Polynomen

$$B_i^{n-1}(x) = \binom{n-1}{i}(1-x)^{n-1-i}x^i \qquad \mathbb{P}_{n-1} = \operatorname{span}\left\{B_0^{n-1}(x), B_1^{n-1}(x), \dots B_{n-1}^{n-1}(x)\right\}$$



$B_0^1(x) = 1 - x$

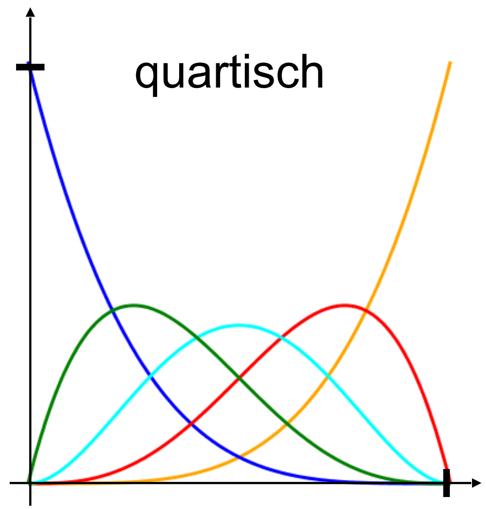
$B_1^1(x) = x$



$B_0^2(x) = (1-x)^2$

$B_1^2(x) = 2(1-x)x$

$B_2^2(x) = x^2$



$B_0^4(x) = (1-x)^4$

$B_1^4(x) = 4(1-x)^3 x$

$B_2^4(x) = 6(1-x)^2 x^2$

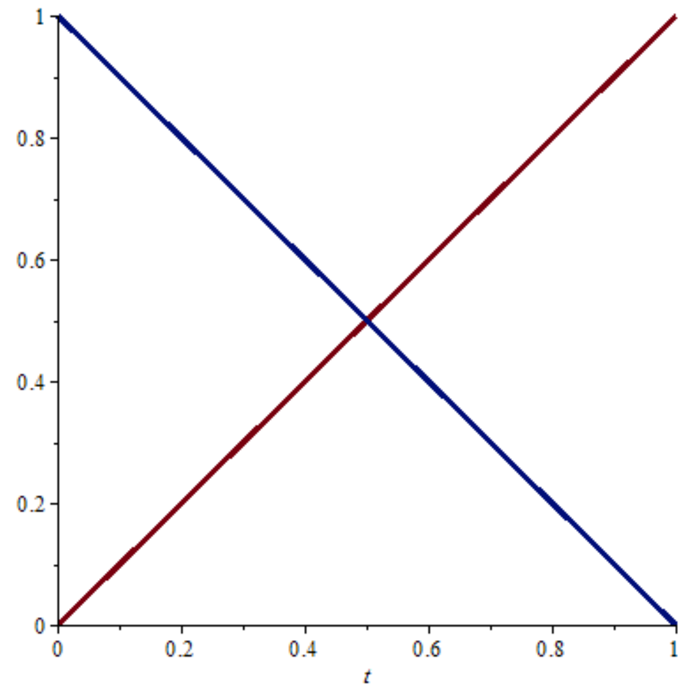
$B_3^4(x) = 4(1-x)x^3$

$B_4^4(x) = x^4$

- **Eigenschaften:**
  - ★ $B_0^n(x) \geq 0$ falls $0 < x < 1$
  - ★ $\sum_{i=0}^{n} B_i^n(x) = 1$

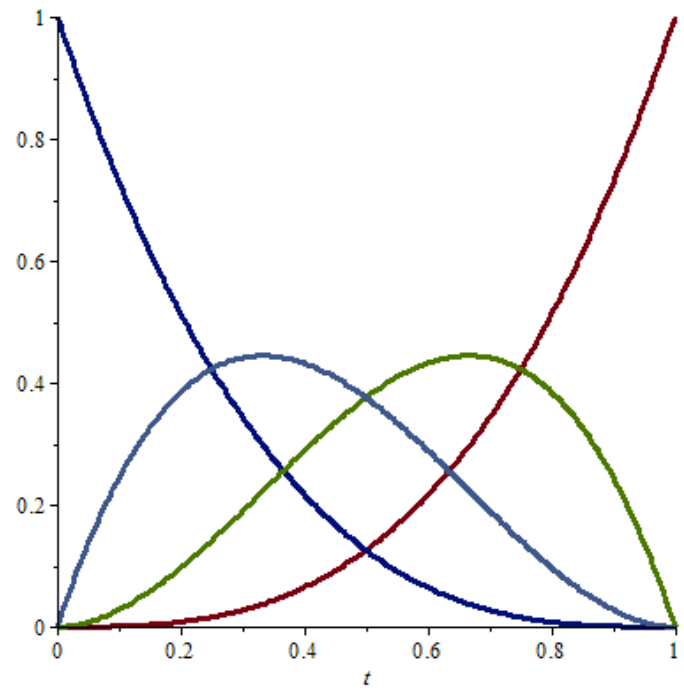
- Bernstein-Polynome vom Grad $n=1$

$$B_0^1(t) = 1 - t, \quad B_1^1(t) = t;$$



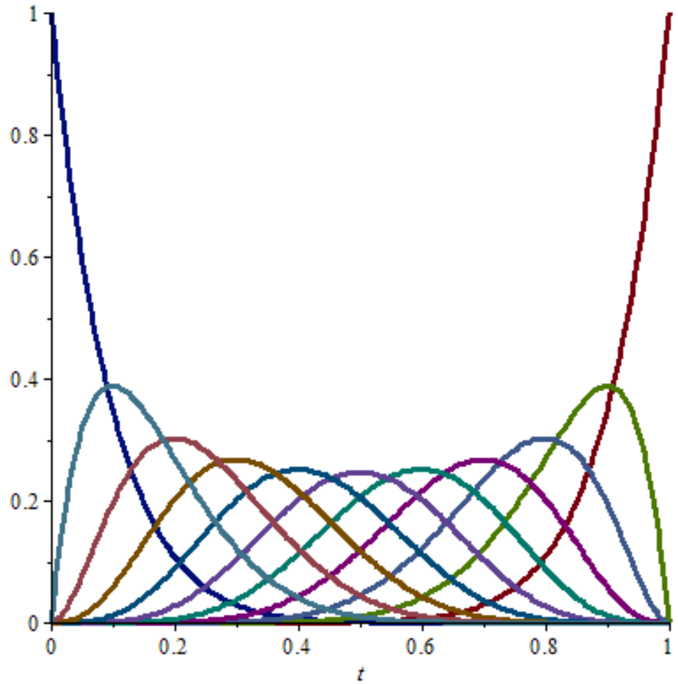
- Bernstein-Polynome vom Grad $n=3$
  (kubische Bernstein-Polynome)

$$B_0^3(t) = (1-t)^3,\ B_1^3(t) = 3(1-t)^2 t,\ B_2^3(t) = 3(1-t)t^2,\ B_3^3(t) = t^3;$$



- Bernstein-Polynome vom Grad $n=10$

$$B_0^{10}(t) = (1-t)^{10},\ B_1^{10}(t) = 10(1-t)^9 t,\ \ldots,\ B_{10}^{10}(t) = t^{10};$$



$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \qquad (i = 0,1,2,\dots,n)$$

- Es gilt

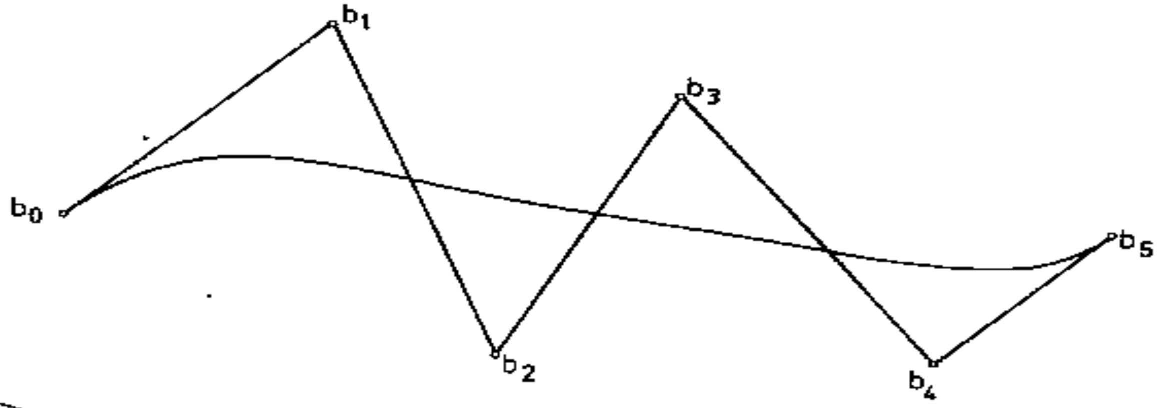
$0 \le B_i^n(t) \le 1$ für $t \in [0,1]$

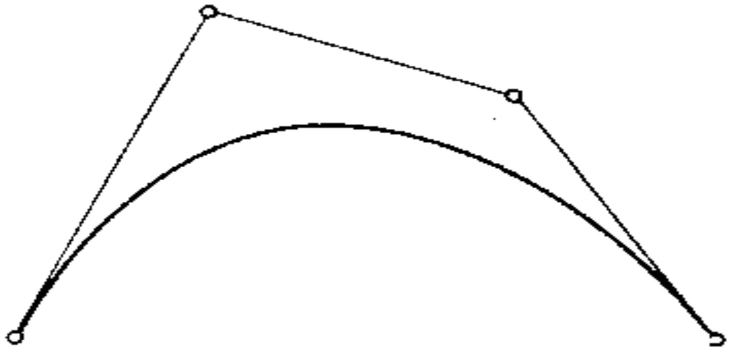
$B_i^n(t)$ hat eine *i*-fache Nullstelle in $t = 0$

$B_i^n(t)$ hat eine (*n*-*i*)-fache Nullstelle in $t = 1$

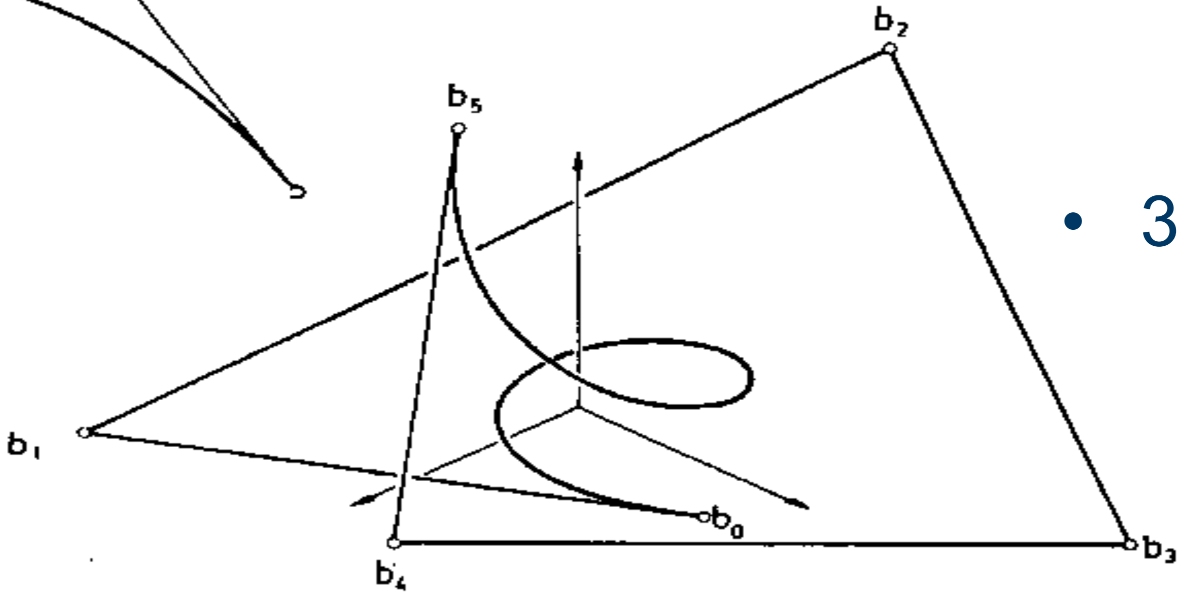
$\sum_{i=0}^{n} B_i^n(t) = 1$ für alle t

- 2D-Kurven





- 3D-Kurve



- Geometrie des Kontrollpolygons spiegelt (grob) die Geometrie der Kurve wider

- **Formeigenschaften**
  1. Interpolation der Endpunkte
  2. In den Endpunkten tangential an das Kontrollpolygon
  3. Bézier-Kurve liegt in der konvexen Hülle der Kontrollpunkte
  4. affine Invarianz
  5. Variationsreduzierend

- Konsequenz: Modifikation der Kontrollpunkte führt zu vorhersehbarer (intuitiver) Änderung der Bézier-Kurve.

- **Endpunkt-Interpolation**

$$C(0) = \sum_{i=0}^{n} \boldsymbol{b}_i \cdot B_i(0) = \boldsymbol{b}_0 \quad , \quad C(1) = \sum_{i=0}^{n} \boldsymbol{b}_i \cdot B_i(1) = \boldsymbol{b}_n$$

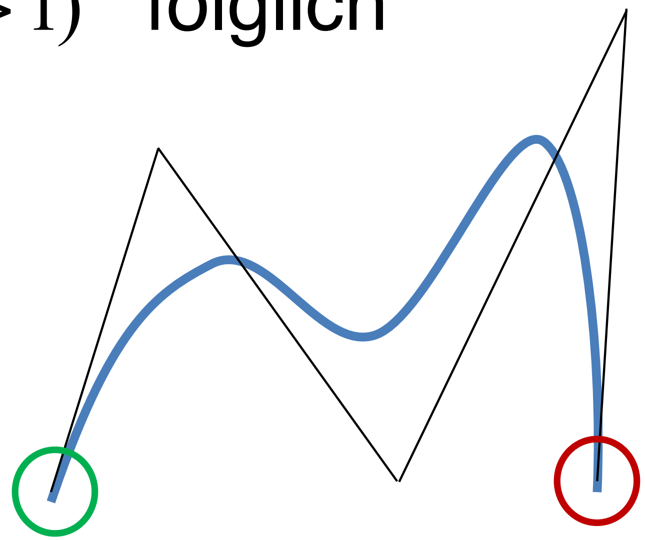
- **Tangentenbedingung**

$$C'(t) = \sum_{i=0}^{n} \boldsymbol{b}_i \cdot B_i'(t), \text{ dabei}$$

$$B_0'(0) = -n\,, \; B_1'(0) = n\,, \; B_i'(0) = 0 \; (i > 1) \quad \text{folglich}$$

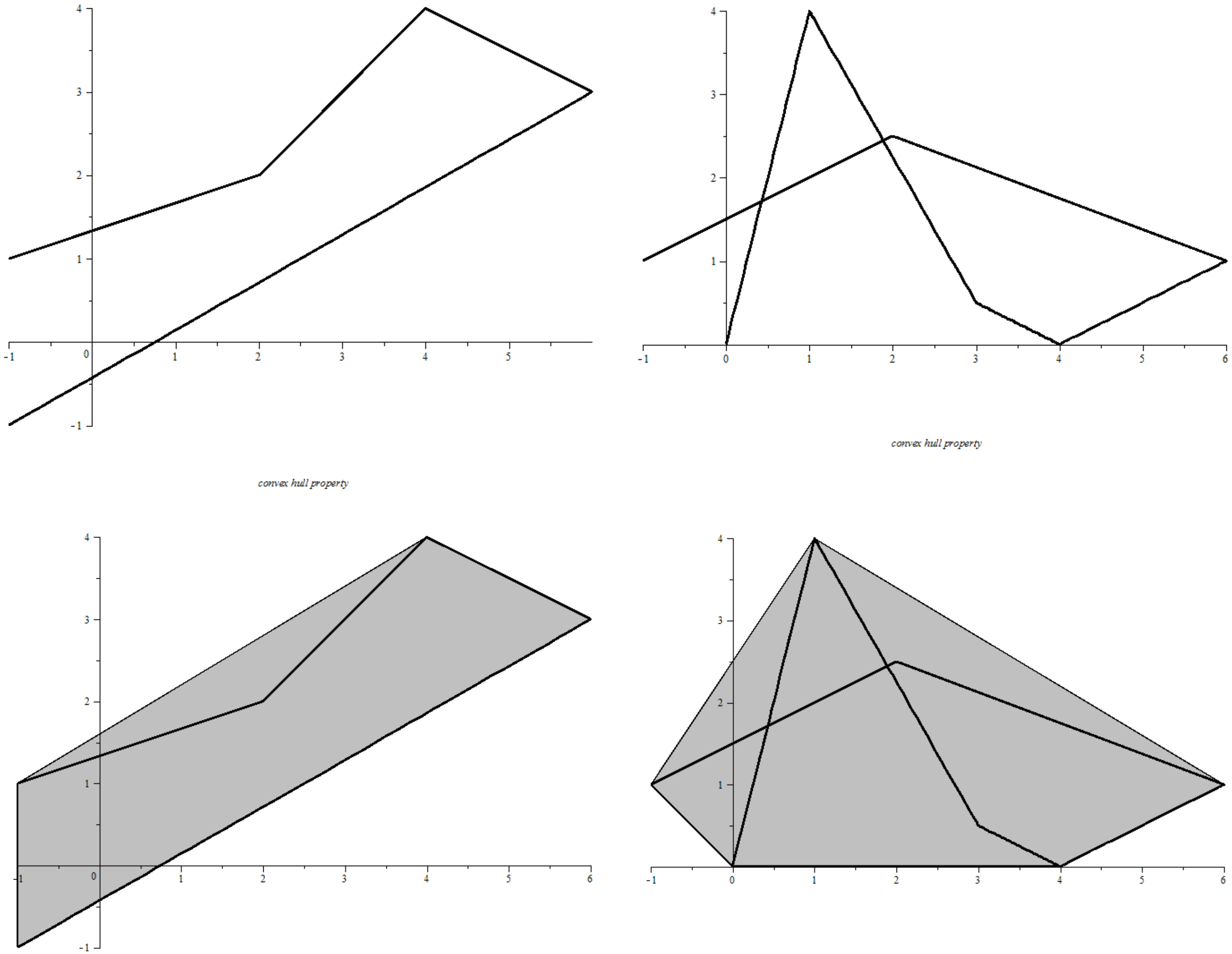
$$C'(0) = \sum_{i=0}^{n} \boldsymbol{b}_i \cdot B_i'(0) = n(\boldsymbol{b}_1 - \boldsymbol{b}_0)$$

$$C'(1) = \sum_{i=0}^{n} \boldsymbol{b}_i \cdot B_i'(1) = n(\boldsymbol{b}_n - \boldsymbol{b}_{n-1})$$



- Konvexe Hülle:
  Bézier-Kurve liegt in **konvexer Hülle** ihrer Kontrollpunkte

- Beispiele



- (Rekursive) Auswertung der Bernstein Polynome: teuer

$$B_i^{r+1}(t) = (1-t)B_i^r(t) + tB_{i-1}^r(t)$$

$$B_i^0(t) = \begin{cases} 1 & \text{falls} \quad i = 0 \\ 0 & \text{sonst} \end{cases}$$

- oder durch **Horner-Bézier**: effizient

$$C(t) = \sum_{i=0}^{n} \boldsymbol{b}_i B_i^n(t) = \sum_{i=0}^{n} \boldsymbol{b}_i \binom{n}{i} (1-t)^{n-i} t^i$$

$$= (1-t)^n \left(\tilde{\boldsymbol{b}}_0 + \tilde{\boldsymbol{b}}_1 \left(\frac{t}{1-t} \right)^1 + \ldots + \tilde{\boldsymbol{b}}_{n-1} \left(\frac{t}{1-t} \right)^{n-1} + \tilde{\boldsymbol{b}}_n \left(\frac{t}{1-t} \right)^n \right)$$

$$= t^n \left(\tilde{\boldsymbol{b}}_0 \left(\frac{1-t}{t} \right)^n + \tilde{\boldsymbol{b}}_1 \left(\frac{1-t}{t} \right)^{n-1} + \ldots + \tilde{\boldsymbol{b}}_{n-1} \left(\frac{1-t}{t} \right)^1 + \tilde{\boldsymbol{b}}_n \right)$$

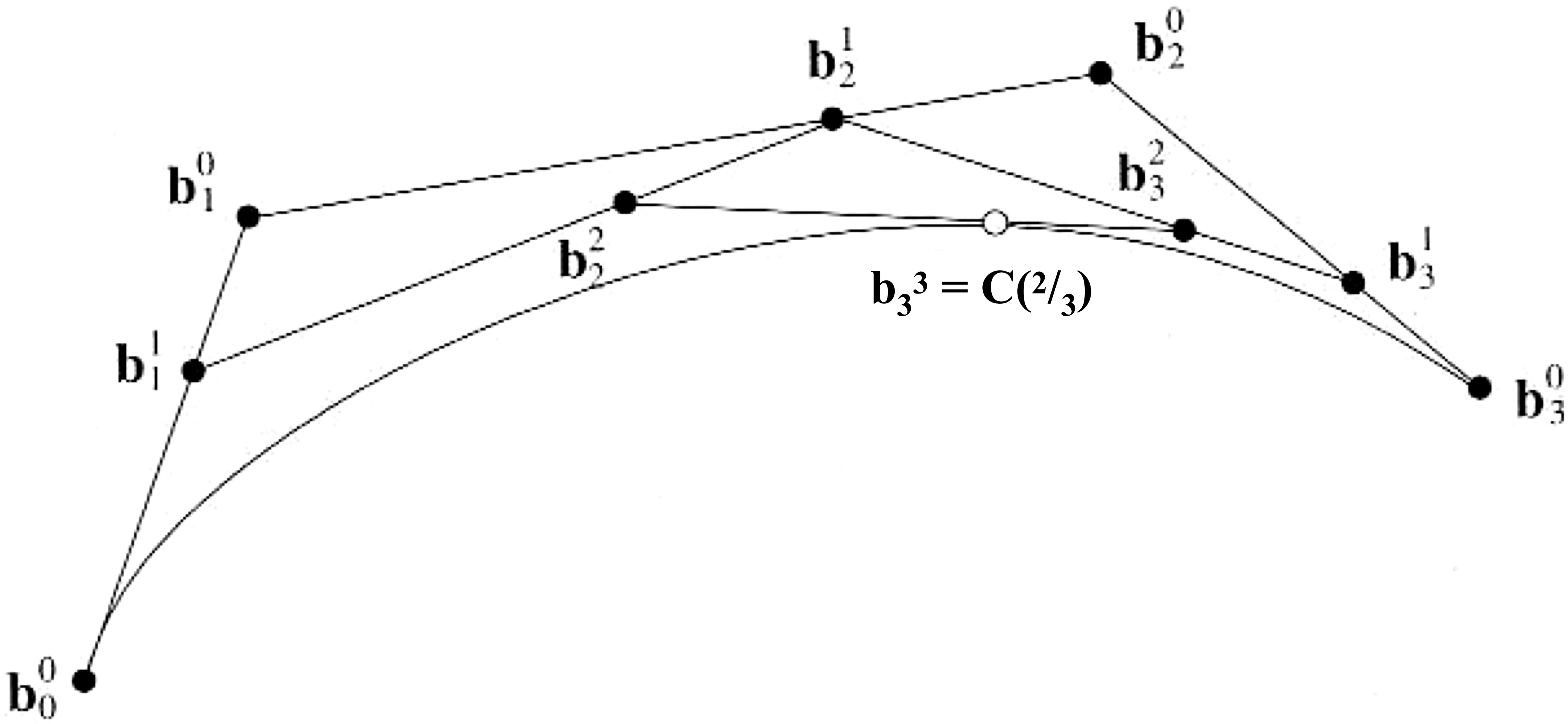
- Oder durch fortgesetzte lineare Interpolation: der **Algorithmus von *de Casteljau*** (stabil)

$$
\begin{tikzcd}
\mathbf{b}_0 = \mathbf{b}_0^0 \arrow[dr, "1-t"] & \\
\mathbf{b}_1 = \mathbf{b}_1^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_1^1 \\
\mathbf{b}_2 = \mathbf{b}_2^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_2^1 \\
\vdots & \vdots \\
\mathbf{b}_{n-1} = \mathbf{b}_{n-1}^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_{n-1}^1 \\
\mathbf{b}_n = \mathbf{b}_n^0 \arrow[r, "t"] & \mathbf{b}_n^1
\end{tikzcd}
$$

- Oder durch fortgesetzte lineare Interpolation:
  der **Algorithmus von *de Casteljau*** (stabil)

$$
\begin{tikzcd}
\mathbf{b}_0 = \mathbf{b}_0^0 \arrow[dr, "1-t"] & & & & \\
\mathbf{b}_1 = \mathbf{b}_1^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_1^1 \arrow[dr, "1-t"] & & & \\
\mathbf{b}_2 = \mathbf{b}_2^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_2^1 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_2^2 \ddots & & \\
\vdots & \vdots & \vdots & & \\
\mathbf{b}_{n-1} = \mathbf{b}_{n-1}^0 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_{n-1}^1 \arrow[r, "t"] \arrow[dr, "1-t"] & \mathbf{b}_{n-1}^2 \cdots & \mathbf{b}_{n-1}^{n-1} \arrow[dr, "1-t"] & \\
\mathbf{b}_n = \mathbf{b}_n^0 \arrow[r, "t"] & \mathbf{b}_n^1 \arrow[r, "t"] & \mathbf{b}_n^2 \cdots & \mathbf{b}_n^{n-1} \arrow[r, "t"] & \mathbf{b}_n^n = \mathbf{x}(t)
\end{tikzcd}
$$

- der **Algorithmus von *de Casteljau*** (geometrisch)



Algorithmus von *de Casteljau* für $t = {}^2/_3$ und $n = 3$

- Algorithmus von *de Casteljau*: PseudoCode

```
# Initialisierung
for i = 0..n
    b_i^0 = b_i;

# deCasteljau Pyramide/Dreieck
for k = 1..n
    for i = k..n
        b_i^k = (1-t)*b_{i-1}^{k-1} + t*b_i^{k-1}

return b_n^n  # Kurvenpunkt C(t)
```

- Polynome sind zur Interpolation/Rekonstruktion geeignet, aber nur wenn der Polynomgrad nicht zu hoch ist.
- Andernfalls
  - Wird das Gleichungssystem für die Polynomkoeffizienten sehr schlecht konditioniert
  - Der Fehler oszilliert und wird an den Rändern des Interpolationsintervalls sehr groß
  - Man kann diese Probleme durch geeignete andere Wahl der Stützstellen (teilweise) kompensieren.
- Lagrange-Polynome für theoretische Untersuchungen hilfreich aber ungeeignet zur numerischen Behandlung
- Andere Polynom-Basen (Bernstein, Lagrange, Newton)
- Praktische Berechnung: Algorithmus von Aitken-Neville
  - numerisch stabil und Aufwand $O(n^2)$
  - Effiziente Auswertung mittels Hornerschema