Numerik Starschnitt
Interpolation

Interpolationsaufgabe:
Tabellarisch definierte Funktionen zwischen den (paarweise verschiedenen) Stützstellen angenähert darstellen, durch z.B. ein Interpolationspolynom P_n höchstens n-ten Grades, das die Interpolationsbedingung $P_n(x_i) = y_i$ für alle i=0,1,...,n erfüllt.

Lagrangesche Interpol.formel
Linearkombination
von Lagrange-Polynomen
$P_n(x) = \Sigma_{i=0 \text{ bis } n}\, y_i\, L_i(x)$

Existenz

Zu beliebigen n Stützpunkten mit paarweise verschiedenen Stützstellen **existiert genau ein** Interpol.polynom, dessen Grad höchstens n ist.

Eindeutigkeit

Fundamentalsatz der Algebra:
Jedes Polynom ist als Produkt seiner Linearfaktoren darstellbar, d.h.
$P_n(x) = (x-x_0)(x-x_1)\dots(x-x_n)$

Lagrange-Polynom
$L_i(x) = \Pi_{j=0 \text{ bis } n}\, {}^{(x-x_j)} / {}_{(x_i-x_j)}$
mit $j \neq i$ für alle i=0,1,...,n

zu aufwendig

$L_i(x_i) = 1$ und $L_i(x_k) = 0$
mit $k \neq i$

Berechnung zu aufwendig und bei Hinzunahme weiterer Stützstellen die gesamte Rechnung neu durchgeführt werden muss. Diese Nachteile vermeidet das Newton.-Interpol.verfahren.

Einführung
von Hilfsgrößen

Baryzentrische Formel
der Lagrange-Interpo.
$P_n(x) = \Sigma_{i=0 \text{ bis } n}\, \mu_i y_i \,/\, \Sigma_{i=0 \text{ bis } n}\, \mu_i$
mit den Stützkoeffizienten
$\lambda_i = \Pi_{j=0 \text{ bis } n}\, 1/(x_i-x_j)$
und $\mu_i = \lambda_i/(x-x_i)$
Rechenaufwand für die Stützkoeffizienten λ_i

1/2 n (n+1)

äquidistant

2 n

hat auch die Darstellung (n meint den Grad)
$P_n(x) = \Sigma_{i=0 \text{ bis } n}\, y_i\, \lambda_i^{(n)}\, \Pi_{j=0 \text{ bis } n}\, (x-x_j)$
mit $j \neq i$

λ_i läßt sich für äquidistante Stützstellen x_0, $x_i = x_0 + ih$ für alle i=1,2,..,n vereinfachen
$\lambda_i = (-1)^{n-i} / h^n n!$ (n über i)

Numerische Differentiation
n-fache Ableitung des Interpolationspolynoms approximiert die n-te Ableitung der Funktion f:
$d^n P_n(x) \,/\, d x^n = \Sigma_{i=0 \text{ bis } n}\, y_i\, \lambda_i^{(n)}\, n!$
$= (1/h^n)\, \Sigma_{i=0 \text{ bis } n}\, (-1)^{n-i} y_i$ (n über i)
für äquidistante Stützstellen

n-ter Differenzenquotient
Für n=2 hätte man dann z.B. die 2. Ableitung, usw. Die Approximation ist aber nur etwa in der Mitte des Intervalls gut.

Ist die zu approximierende Funktion bekannt und kann sie als konvergierende Taylorreihe an der Stelle x_M entwickelt werden, dann kann durch Auflösen nach dem Quotienten der Fehler bestimmt werden, d.h.
$f(x_M+h) - f(x_M-h) \,/\, 2h = f'(x_M)$
liefert nicht genau die Ableitung in x_M von f, wenn man für f Taylorreihen verwedent.
Und es zeigt sich, das die Approximation durch den ***zentralen*** **Differenzenquotienten** besser ist.

Bei kleiner werdender Schrittweite nimmt der relative Fehler zu.

Zentraler Differenzenquotient
Man nützt aus, das die Approximation in der Mitte des Intervalls/der Intervalle x_M besser ist.
Die Stützstellen müssen geeignet gewählt werden.
Dann leitet man das n-te Interpol.polynom - für die p-te Ableitung - p-mal ab und setzt die Mitte $x_M = (x_0 + x_n)/2$ ein.
Für n=2 (Parabel) und p=1 ergibt sich die Sekante von x_0 bis x_2, die in x_M der Ableitung entspricht.
Der Satz von Rolle (Mittelwertsatz) sichert die Existenz einer solchen Stelle.

Extrapolation auf Null
Oft kann eine gesuchte Größe A nur durch einen berechenbare Größe B(t) approximiert werden, wenn diese von einem Parameter abhängt ist,
$B(t) = A + c_1 t + c_2 t^2 + \dots$
Wenn z.B. B(t)= 1/t ist, kann t=0 nicht bestimmt werden.
Extrapolation auf Null besteht darin mit $t_0 > t_1 > .. > t_n > 0$ und $B(t_k)$ Stützpunkte für ein Interpolationspolyn. $P_n(t)$ zu berechnen und dieses an der Stelle t=0 auszuwerten.

Newtonsche Interpol.formel
Linearkombination
von Newton-Polynomen
$P_n(x) = \Sigma_{k=0 \text{ bis } n}\, c_k\, N_k(x)$

Newton-Polynom
$N_k(x) = \Pi_{j=0 \text{ bis } k-1}\, (x-x_j)$

Ansatz für paarweise verschiedene Stützstellen:
$P_n(x) = c_0 + c_1 (x-x_0) + c_2 (x-x_0)(x-x_1) + \dots + c_n (x-x_0)(x-x_1)\dots(x-x_n)$
Bestimmung der Koeffizienten: Das LGS lässt sich durch Vorwärtseinsetzen lösen
$P_n(x_0) = c_0 = y_0$
$P_n(x_1) = c_0 + c_1 (x-x_0) = y_1$
$P_n(x_2) = c_0 + c_1 (x-x_0) + c_2 (x-x_0)(x-x_1) = y_2$
Wir betrachten Interpol.poly. für Teilmengen der Stützpunkte
$P^*_{i_0 i_1 .. i_k}(x)$ mit $1 < k \leq n$
um eine Rekursionsformel definieren zu können

$$P^*_{i_0 i_1 .. i_k}(x) = \frac{(x-x_{i_0})\, P^*_{i_1 i_2 .. i_k}(x) - (x-x_{i_k})\, P^*_{i_0 i_1 .. i_{k-1}}(x)}{x_{i_k} - x_{i_0}}$$

, die die Interpolationsbedingung $P^*_{\dots}(x_i) = y_i$ erfüllt. Für k=1 gilt $P^*_i(x) = y_i$. Für k=n folgt das obige Interpolationspolynom.

folgt

k-t dividierte Differenz c_k
Das i-te Lagrangesch Polynom hatte angenehme die Eigenschaft für x_i 1 zu sein, so dass die Koeffizienten nicht berechnet werden mussten. Der k-te Koeffizient ist hier von den Stützpunkten 0,1,..,k abhängig. Man schreibt deshalb
$c_k := f[x_0 x_1 .. x_k]$
Rekursive Berechnung der Koeffizienten (=Steigungen)
$c_k := f[x_0 x_1 .. x_k] = f[x_1 x_2 .. x_k] - f[x_0 x_1 .. x_{k-1}] \,/\, x_k - x_0$
für k=0,1,..,n erfolgt mit dem Schema der divid. Differenz:
x_0 $y_0 = f[x_0] = c_0$
$f[x_0 x_1] = c_1$
x_1 $y_1 = f[x_1]$ $f[x_0 x_1 x_2] = c_2$
$f[x_1 x_2]$
x_2 $y_2 = f[x_2]$
Berechnet man das Schema von unten nach oben und von links nach rechts, dann beträgt der benötigte Speicherplatz nur n+1.
Gleicher Rechenaufwand wie die Stützkoeffizienten des Lagrange-Polynoms.

1/2 n (n+1) Divisionen

Neustelle
Die Anzahl der Multiplikationen **n** zur Berechnung einer Neustelle vereinfacht sich deutlich (weniger als die Hälfte als bei der Lagrange-Interpoltion), wenn das Interpolationspolynom stark verschachtelt wird. Newton Interpol. ist also vom Aufwand her am besten!
$P_n(x) = \Big[.. \big[c_{n-3} + (x-x_{n-3}) [c_{n-2} + (x-x_{n-2})\, [c_{n-1} + (x-x_{n-1})\, c_n]] \big] .. \Big]$

äquidistante Stützstellen

Newton-Gregory-Interpolationsformel
für äquidistante Stützstellen gilt
$P_n(x) = y_0 + \Sigma_{i=0 \text{ bis } n} \binom{t}{i} \Delta_{i/2}^{\,i}$

Vereinfachung wegen der Äquidistanz $x_i = x_0 + ih$.
Die k-te Differenz $f[x_i, x_{i+1}, .., x_{i+k}]$ wird gesetzt als
$1/(k!h^k)\, \Delta_{i+(k/2)}^{\,k}$
Durch Definition der relativen Distanz $t = (x-x_0)/h$ und Verwendung des Binomialkoeffizienten entsteht diese kompakte Formel.
Anzahl der wesentlichen Operationen für die Berechnung einer Neustelle hat sich allerdings verdoppelt

2 n + 1

äquid. und gleiche Stützstellen

Hermitesche Interpolation
zusammenfallende äquidistante Stützstellen

Zusammenfallende Stützstellen führen bei der dividierten Differenz zum Verschwinden des Nenners! Man betrachtet deshalb den Grenzwert von h gegen 0.
$f[x_k, x_k, .., x_k] = {}^1/{}_{m!}\, f^{(m)}(x_k)$ m=1,2,..
Im Schema der div. Differenz stehen dann entsprechen der Vielfachheit der Stützstellen m die Ableitungen $f^{(m)}$.
$f[x_0]$
$f'(x_0) = c_1$
$f[x_0]$ ${}^1/{}_2\, f''(x_0) = c_2$
$f'(x_0)$
$f[x_0]$
Wenn Interpolationspolynome, die an zusammenfallenden Sützstellen auch die Ableitungen interpolieren, spricht man von Hermitecher Interpolation.

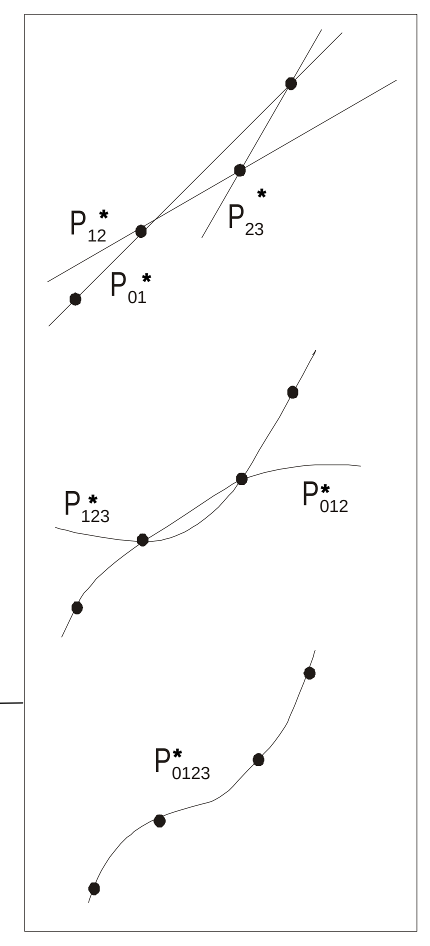
Interpolation nach Aitken-Neville
keine explizite Darstellung des Inter.Poly.

Algorithmus von Aitken
hat sich nicht durchgesetzt. Die Rekursionsformel wird auch auf nicht nebeneinander liegende Stützstellen angewendet. Z.B. lauten die Indizes der 2ten Spalte des Schema 01,02,03,04,...

Algorithmus von Neville
Unter Verwendung der Rekursionsformel, auf der auch das Newton-Verfahren beruht wird analog zum Schema der dividierten Differenz folgendes Schema verwendet:
x_0 $y_0 = P_0^*$
x_1 $y_1 = P_1^*$ P_{01}^*
x_2 $y_2 = P_2^*$ P_{12}^* P_{012}^*
x_3 $y_3 = P_3^*$ P_{23}^* P_{123}^* $P_{0123}^* = P_3(x)$
Direkte Berechnung des interplierten Wertes ohne explizite Darstellung des Inter.Polynoms, d.h. der Diagonale können keine Koeffizienten entnommen werden, sondern rechts steht der Wert der Neustelle x.

Benötiger Speicherplatz n+1. Aufwand für die Berechnung einer Neustelle doppelt so aufwendig wie beim Newton-Verfahren, dafür einfach und kompakt.

n (n+1)



Extrapolation auf Null
Sie ist hier auch möglich. (Wie bei der Lagrangesche Interpolation beschrieben)

Thielescher Kettenbruch
Interpolation durch eine gebrochen rationale Funktion
$R(x) := P_k(x) \,/\, Q_l(x)$

Rationale Interpolation leistet oft eine bessere Aproximation als polynomiale.

Ist die Zahl der Stützstellen n gerade haben Zähler und Nennerpolynom den Graf n/2, sonst ist der des Zählerpolynoms um 1 größer.
Die k-te inverse dividierte Differenz ist definiert als

$$\varphi_k(x_i, x_{k-1}, .., x_0) := \frac{x_i - x_{k-1}}{\varphi_{k-1}(x_i, x_{k-2}, .., x_0) - \varphi_{k-1}(x_{k-1}, x_{k-2}, .., x_0)}$$

Für k=Spalte und i=Zeile entsteht dann das Schema der inversen dividierten Differenz.
Der endlich, abgebrochene Thielesche Kettenbruch entsteht so: Man ersetzt in allen inv. div. Diff. der Diagonale x_i durch x, löst jede nach $\varphi_{k-1}(x,..)$ auf und setzt die k+1-te Diff. die k-te ein.
Z.B. für n=3

$$f(x) = f(x_0) + \cfrac{x-x_0}{\varphi_1(x_1,x_0) + \cfrac{x-x_1}{\varphi_2(x_2,x_1,x_0) + \cfrac{x-x_2}{\varphi_3(x_3,x_2,x_1,x_0)}}}$$

Es finden nur die Diagonalelement des Schemas verwendung:
$c_i := \varphi_i(x_i, x_{i-1}, .., x_1, x_0)$
Die Reihenfolge der Stützstellen ist anders als beim Schema der dividierten Differzent.
f löst die Interpolationsaufgabe. Zu beachten sind eventuelle Nullstellen im Nenner.

Auswert des T.K. für Neustellen geschieht durch rekursive Berechnung der Teilkettenbrüche $R_k(x)$ für k=2,3,..,n
$p_k = c_k p_{k-1} + (x-x_{k-1}) p_{k-2}$ (Zähler)
$q_k = c_k q_{k-1} + (x-x_{k-1}) q_{k-2}$ (Nenner)
mit den Startwerten
$p_0 = c_0$, $p_1 = c_0 c_1 + (x-x_0)$ und $q_0 = 1$, $q_1 = c_1$

Spline-Interpolation

Idee: Interpolationspolynome niedrigen Grades (2p-1) werden für Teilintervalle definiert, wobei die Interpolierende glatt an den Grenzen der Teilintervalle sein soll.
Der Grad gibt den Namen, z.B.
p=2 -> (2 2-1) = 3 kubische-Splinefunktion,
p=3 -> (2 3-1) = 5 quintische Splinefunktion.
Problem: Interpolierende höheren Grades oszillieren stärker.

Die Spline-interpolierende s(x) ist Lösung der Variationsaufgabe:

a) s(x) erfüllt die Interpolationsbedingung $s(x_i) = y_i$ i=0,1,..,n
b) s(x) minimiert $J = 1/2 \int_{x_0}^{x_n} s^{(p)}(x)^2\, dx$
d.h. die splines (=Latten) nehmen einen energetisch günstige Form an. Die Ableitung von s(x) legt $p \geqslant 2$ fest.
c) s(x) an allen inneren Stützstellen (2p-2)-mal stetig diff.bar, d.h. glatt, die Ableitungen sind stetig.
d) s(x) ist zwischen den Stützstellen 2p-mal diff.bar

1. Das Integral entsteht aus physikalischen Überlegungen.
2. Differenzieren nach s''(x), weil wir das Minimum suchen
3. Variationsrechnung: Multiplizieren mit $\delta s''(x)$
4. Summenregel der Integration
5. Zweimalige partielle Integration
6. Alle Terme mit $\delta s(x)$ verschwinden, weil die erste Variation $\delta s(x)$ an den Stürzstellen 0 sein muß, wegen der Int.Beding.
7. Aus dem entstandenen Ausdruck lassen sich die Bedingungen teilweise (links) ablesen.

natürliche Bedingung:
$s^{(p)}(x), s^{(p+1)}(x), .., s^{(2p-2)}(x)$ sind an den Endpunkten x_0 und x_n Null.
Man spricht dann von einer natürlichen Splinefunktion.

Berechnung der kubischen Spline-Interpolierenden
Die kubischen Polynome haben die Gestalt
$s_i(x) = a_i (x-x_i)^3 + b_i (x-x_i)^2 + c_i (x-x_i) + d_i$ und wir setzen
$h_i = x_{i+1} - x_i$
Aus $s_i(x_i) = y_0$, $s_i(x_{i+1}) = y_{i+1}$ und deren 1. und 2. Ableitungen gewinnt man durch Subtraktion, Einsetzen und Auflösen die a_i, b_i, c_i, d_i. Die Koeffizienten nur noch von den y_i'' abhängig.

natürliche

Aus der Stetigkeit der 1. Ableitung an den inneren Stützstellen ($s'_{i-1}(x_i) = s'_i(x_i)$ für alle i=1,2,..,n-1) folgt ein GS mit n-1 Gleichungen und n-1 Unbekannten. Denn gilt die natürliche Randbedingung, dann sind y_0'' und y_1'' bekannt.
Die Matrix ist symmetrisch, tridiagonal und diagonal-dominant. Der Gauss-Algo. unter Verwendung der Diagonalstrategie führt zu einer eindeutigen Lösung.

allgemeine

Um das oszillieren an den Rändern zu vermeiden:
Entweder ist die 2. Abl. an den Rändern vorgeben mit
$y_0'' = \alpha\, y_1''$ und $y_n'' = \beta\, y_{n-1}''$
oder not-a-knote Bedingung:
Auch die 3. Abl. soll in x_1 und x_{n-1} stetig sein, d.h.
$s'''_0(x_1) = s'''_1(x_1)$ und $s'''_{n-2}(x_{n-1}) = s'''_{n-1}(x_{n-1})$
Man rechnet wie oben und bestimmt anschliessend y_0'' und y_n'' mit den zwei zusätzlichen Gleichungen, weil sonst das GS seine günstigen Eigenschaften verlöre.
Alternativ: Oft kann die 1. Abl. in den Endpunkten vorgegeben werden $s'_0(x_0) = y'_0$ und $s'_n(x_n) = y'_n$
Diese Gleichungen können dem GS hinzugefügt werden, ohne die günstigen Eigenschaften zu verlieren. Das GS hat dann n+1 Gleichungen und bestimmt y_i'' für i=0,1,..,n.

periodische

Zu approximieren ist jetzt eine periodische Funktion. Für die Periode T gilt $x_n = x_0 + T$. Die Funktion hat die Eigenschaften
$y_0 = y_n$, $y'_0 = y'_n$, $y''_0 = y''_n$
Somit sind nur die y''_i für alle i=0,1,...n-1 unbekannt. Es entsteht ein GS mit n Gleichungen.
Wegen der Periodizität ist jetzt der Index -1 interpretierbar als n-1, so dass auch die erste Gleichung definiert ist.
Das GS ist zwar symmetrisch und diagonal dominant, aber nicht mehr tridiagonal. Da die Matrix pos. def. ist, eignet sich besonders die Methode von Cholesky zur Lösung.

Damit sind auch die y_i'' bekannt.
Für das GS ist keine Nachiteration nötig, weil die Konditionszahl der Matrix nicht sehr groß ist.

Die Koeffizienten lassen sich errechnen und die Spline Polynome formulieren.

Rechenaufwand ~ n

Bezier-Technik
für Kurven und Flächen.

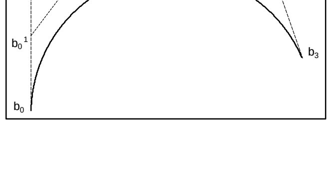
Idee: Kurven und Flächen durch eine möglichst einfache Parameterdarstellung zu definieren, wobei sie sich effizient punktweise berechnen lassen.

Weil Bernstein-Poly. eine Basis des Vektorraums P_n der reellen Polynome vom Grad n bilden, lassen sich Raumkoordinaten (Vektoren) der Dimension d als Linearkombination von Bernstein-Poly. darstellen:
$x(t) = (x_1(t), .., x_d(t))^T$ mit $b_i = (b_{i1}, .., b_{id})^T \in R^d$ für alle i=0,1,..,n , so gelangt man zur Bezier-Darstellung einer polynomialen Kurve: $x(t) := \Sigma_{i=0 \text{ bis } n}\, b_i\, B_i^n(t;a,b)$ mit $t \in [a,b]$
Die Koeffizientenvektoren $b_1, .., b_n \in R^d$, welche die Bezier-Kurve festlegen, heißen Bezier-Punkte. Verbindet man sie durch Geraden, so entsteht das Bezier-Polygon. Die Bezier-Kurve liegt in der konvexen Hülle des Bezier-Polygons, weil sie aus einer linearen Konvexkombination der Bezier-Punkte entstehen. Hier zeigt sich die erste Besonderheit: Die Punkte liegen bis auf Start- und Endpunkt nicht auf der Kurve. [a,b] nennt man das Parameterintervall.

Eine Kuve mit gewünschten Eigenschaften wird stückweise durch Zusammensetzung von Bezier-Kurvensegmenten $x_j(t_j)$ approximiert. Die Stetigkeit wird erreicht durch:
$x_j(t_j) = x_{j+1}(t_j)$ mit $t \in [t_{j-1}, t_j]$ und $[t_j, t_{j+1}]$
die Glattheit durch:
$x_j'(t_j) = x_{j+1}'(t_j)$ es reicht $x_j'(1) = x_{j+1}'(0)$ zu untersuchen.
Somit ist $b_{n,j} = b_{0,j+1}$ eine lineare Konvexkombination der Bezier-Punkte $b_{n-1,j}$ und $b_{1,j+1}$. Die Die Tangentenrichtung in den Randpunkten ist jeweils durch den Randpunkt und den benachbarten Bezier-Punkt bestimmt.

Bernstein-Polynome

Nach binomischen Lehrsatz läßt sich die 1-Funktion
$1 = [\lambda + (1-\lambda)]^n = \Sigma_{i=0 \text{ bis } n} \binom{n}{i} \lambda^i (1-\lambda)^{n-1} =: \Sigma_{i=0 \text{ bis } n}\, B_i^n(\lambda)$
in n+1 Bernstein-Polynome vom Grad n bzgl. des Intervalls [0,1] zerlegen.
Jedes Intervall kann durch affine Transformation $\varphi^{-1}(t)$ auf das Intervall [0,1] abgebildet werden. Für das Intervall [a,b] gilt dann für alle i=0,1,..,n
$B_i^n(\varphi^{-1}(t)) = B_i^n(t;a,b) =$
$B_i^n({}^{t-a}/{}_{b-a}) = {}^1/{}_{(b-a)^n} \binom{n}{i} (t-a)^i (b-t)^{n-i}$ mit $t \in [a,b]$
Mit Hilfe des Additionstheorems des Binomialkoeffizienten
$\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}$ läßt sich eine Rekursionsformel bestimmen:
$B_i^n(\lambda) = \lambda\, B_{i-1}^{n-1}(\lambda) + (1-\lambda)\, B_i^{n-1}(\lambda)$ für i=1,..,n
$B_0^{n+1}(\lambda) = (1-\lambda)\, B_0^n(\lambda)$ und $B_n^{n+1}(\lambda) = \lambda\, B_n^n(\lambda)$
Ein Bernstein-Poly. vom Grad n läßt überführen in eine lineare Konvexkombination zweier Bernstein-Polv. vom Grad n-1.



Algo. von de Casteljau
Der Wert $x(\lambda)$ wird durch fortgesetzte lineare Konverkombination aus den Bezier-Punkten mit Hilfe der Rekursionsformel ermittelt.
Schema für n=3 (von links nach rechts).

i\n	0	1	2	3
0	b_0	b_0^1	b_0^2	$b_0^3 = x(\lambda)$
1	b_1	b_1^1	b_1^2	
2	b_2	b_2^1		
3	b_3			

, z.B. $b_1^1 = (1-\lambda)\, b_1 + \lambda\, b_2$. Gleichzeitg ergibt sich auch die 1. Ableitung, weil b_0^2 und b_1^2 auf der Tangente von $x(\lambda)$ liegen:
$x'(\lambda) = n\, (\, b_1^{n-1}(\lambda) - b_0^{n-1}(\lambda)\,)$