Karlsruher Institut für Technologie

Angewandte Geometrie (CAGD)
Prof. Dr. Hartmut Prautzsch
Dipl.-Ing. Yijun Xu
B.Sc. Louis Lutzweiler

MARS-Basispraktikum **WS 2019/20**

Versuch 1

B-Spline-Kurven

Inhaltsverzeichnis

1 Allgemeines

Der Praktikumsteilnehmer wird mit den Grundlagen interpolierender Splines vertraut gemacht. Im Versuch werden dazu einige Methoden implementiert.

Durchführung

1. Lesen Sie sich diese Versuchsbeschreibung gründlich durch. Machen Sie sich mit den Grundlagen der B-Splines vertraut. Verschaffen Sie sich einen Überblick über die bereits implementierten Klassen und Methoden.

   Im Verzeichnis `cagd` befinden sich die Datei `spline.py`, in der in diesem Versuch Methoden zu implementieren sind.

2. Um das Programm auszuführen, rufen sie `python3 interpolation.py` auf. Zu Beginn werden noch Fehler auftreten, da die entsprechenden Methoden noch nicht implementiert sind.

3. Programmieren Sie die Methoden *knot_index* und *de_boor* der Klasse *spline*. Schauen Sie sich den Beispielspline des Programms `interpolytion.py` an.

4. Implementiert die Methode *interpolate_cubic* der Klasse *spline* und aktiviert den zweiten Teil der Aufgabe in `interpolation.py` durch Entfernen der Kommentarzeichen.

5. Führen Sie die Ergebnisse dem Tutor vor.

2 Theoretische Grundlagen

Zum Modellieren von beliebigen Oberflächen werden so genannte *Freiformflächen* verwendet. Dies wird vor allem im Bereich des Karosseriebaus, CAD, ... eingesetzt. Im folgenden Versuch beschäftigen wir uns mit zweidimensionalen *Kurven*.

Gegeben sei eine Folge von Punkten $\mathbf{p}_i = (x_i, y_i)$ in der Ebene mit $i = 0, \ldots, m$ mit zugehörigen Parameterwerten $t_i < t_{i+1}$. Gesucht ist eine Kurve, die möglichst *glatt* durch diese Punkte verläuft. Unter einer *glatten Kurve* versteht man eine Kurve, die keine Ecken besitzt und kein ungewünscht welliges Verhalten aufweist.

Versucht man, die Kurve durch ein *Polynom* zu interpolieren, so muss dieses die $m+1$ Bedingungen $\mathbf{s}(t_i) = \mathbf{p}_i$ erfüllen. Im allgemeinen Fall muss also das Polynom mindestens vom Grad m sein.

Bei der *Hermite–Interpolation* gibt man nicht nur die Interpolationspunkte, sondern auch die ersten Ableitungen vor. Bei $m + 1$ Punkten hat man jetzt $2m + 2$ Bedingungen, die ein Interpolationspolynom zu erfüllen hat. Es muss also im allgemeinen Fall mindestens vom Grad $2m + 1$ sein.

Hat man nun eine größere Anzahl von Punkten, so kann es in beiden Fällen bei höheren Graden zu großen Schwingungen kommen. Daher verwendet man stückweise polynomiale Kurven niedrigen Grades.

Die Idee bei *Splines* ist es, zwischen je zwei *Stützpunkten* $\mathbf{p}_i$ und $\mathbf{p}_{i+1}$ $(i = 0, \ldots, m - 1)$ ein Polynomstück niedrigen Grades zu legen. Dabei versucht man, an den Stützpunkten möglichst glatte Übergänge zu bekommen.

Nimmt man bei Splinekurven Monome (x^i) als Basisfunktionen, sind die Koeffizienten nicht anschaulich interpretierbar. Daher benutzt man andere Basisfunktionen, mit denen die Koeffizienten (dann Kontrollpunkte genannt) eine geometrisch anschauliche Bedeutung bekommen. Die wichtigsten auftretenden Basisfunktionen sind *Bernstein-Polynome* und *B-Splines*. Wir werden uns speziell mit B-Splines vom Grad 3 beschäftigen. Diese Splines sind C^2-stetig, falls der (innere) Knotenvektor keine Mehrfachknoten aufweist.

2.1 Spline-Kurven

Da die Spline-Kurven hinlänglich aus Vorlesungen bekannt sein dürften, soll an dieser Stelle nur eine kurze Zusammenfassung gegeben werden. Ausführlichere Darstellungen finden sich in der angegebenen Literatur.

Die *normalisierten B-Splines* $N_i^n(t)$ sind aus Polynomsegmenten zusammengesetzt, deren Trennstellen durch die *Knoten* t_i des *Knotenvektors* bestimmt werden, und werden definiert über

$$
\begin{aligned}
N_i^k(t) &= \frac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) \quad + \quad \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t) \\
N_i^0(t) &= \begin{cases} 1 & \text{für} \quad t \in [t_i, t_{i+1}) \\ 0 & \phantom{\text{für}} \quad t \notin [t_i, t_{i+1}) \end{cases} .
\end{aligned}
\tag{1}
$$

Es folgt, dass $N_i^n(t) = 0$ für $t \notin [t_i, t_{i+n+1})$, dem *Träger* von $N_i^n(t)$.

Eine *Spline-Kurve n–ten Grades* ist eine *stückweise polynomiale Kurve*

$$\mathbf{s}(t) = \sum_i \mathbf{d}_i N_i^n(t) \ . \tag{2}$$

Die Koeffizienten $\mathbf{d}_i$ bilden das *Kontrollpolygon*. Über $[t_{i+n}, t_{i+n+1})$ verschwinden somit alle B-Splines außer $N_i^n(t), \ldots, N_{i+n}^n(t)$, d. h. das zugehörige *Kurvensegment* wird nur von $\mathbf{d}_i, \ldots, \mathbf{d}_{i+n}$ beeinflusst. Die zu $\mathbf{d}_0, \ldots, \mathbf{d}_m$ gehörende Kurve ist also über $[t_n, t_{m+1})$ parametrisiert.

$t_n, \ldots, t_{m+1}$ bilden den *inneren Knotenvektor*, der somit den gültigen *Parameterbereich* der Kurve festlegt, wohingegen der gesamte Knotenvektor aus $t_0, \ldots, t_{n+m+1}$ besteht.

2.2 Der De-Boor-Algorithmus

Zur Berechnung von Punkten auf einer Spline-Kurve kann der *De-Boor-Algorithmus* verwendet werden, der eine Verallgemeinerung des *de Casteljau-Schemas* ist. Außer einem günstigeren Aufwand ($O(n^2)$ statt $O(2^n)$ bei Verwendung obiger Rekursionsformel (1)) ist dieses Verfahren auch grundlegend für die Berechnung von Ableitungen und beim Unterteilen bzw. beim Einfügen von Knoten.

Für das Segment einer kubischen Spline-Kurve ($n = 3$) mit den Kontrollpunkten $\mathbf{d}_0^0 = \mathbf{d}_0, \ldots, \mathbf{d}_3^0 = \mathbf{d}_3$ und dem zugehörigen Knotenvektor $t_1, \ldots, t_6$ ergibt sich folgendes Schema:

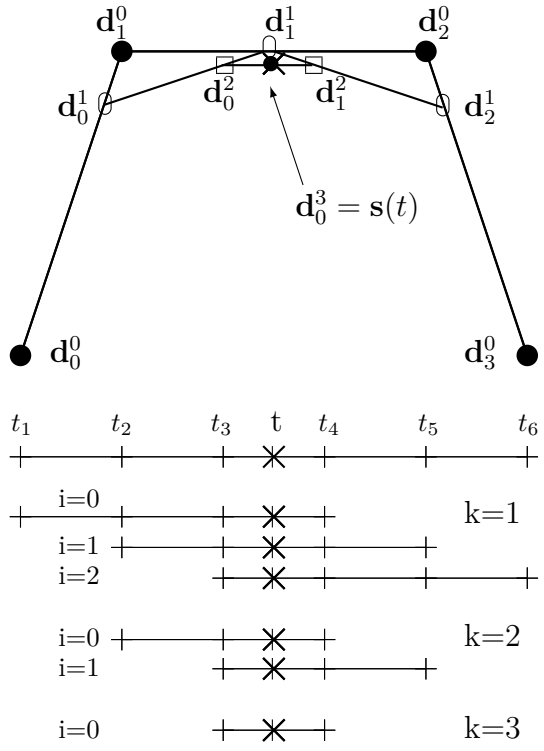
$$\begin{array}{ccccccc}
\mathbf{d}_0^0 & & & & & & \\
 & \mathbf{d}_0^1 & & & & & \\
\mathbf{d}_1^0 & & \mathbf{d}_0^2 & & & & \\
 & \mathbf{d}_1^1 & & \mathbf{d}_0^3 = \mathbf{s}(t) & & & \\
\mathbf{d}_2^0 & & \mathbf{d}_1^2 & & & & \\
 & \mathbf{d}_2^1 & & & & & \\
\mathbf{d}_3^0 & & & & & &
\end{array} \tag{3}$$

$$\begin{aligned}
\mathbf{d}_i^k &= (1 - \alpha_i^k)\ \mathbf{d}_i^{k-1} + \alpha_i^k\ \mathbf{d}_{i+1}^{k-1} \\
\alpha_i^k &= \frac{t - t_{i+k}}{t_{i+n+1} - t_{i+k}} \ ,
\end{aligned}$$

wobei $t \in [t_3, t_4)$.

Für $t \in [t_j, t_{j+1})$ und $n = 3$ ist der zugehörige Knotenvektor $t_{j-2}, \ldots, t_{j+3}$, und die zugehörigen Kontrollpunkte sind $\mathbf{d}_{j-3}, \ldots, \mathbf{d}_j$, was entsprechende Indextransformationen in obigem Schema (3) nach sich zieht.

Graphisch ergibt sich (für $n = 3$) folgender Eckenschnittalgorithmus:



Aufgabe: *Betrachten Sie eine kubische B-Spline-Kurve mit Knotenvektor $t_1 = t_2 = t_3 = 0, t_4 = t_5 = t_6 = 1$. Wie sieht in diesem Fall der Algorithmus von de Boor aus? Bestimmen Sie graphisch die Werte einer solchen Kurve für die Parameterwerte $t = 0, 1/4, 1/2, 3/4, 1$.*

Für Spline-Kurven beliebigen Grades n und $t \in [t_{i+n}, t_{i+n+1})$ hat der De-Boor-Algorithmus die folgende Gestalt:

$$
\begin{array}{ccccccc}
\mathbf{d}_i^0 & & & & & & \\
& \mathbf{d}_i^1 & & & & & \\
\mathbf{d}_{i+1}^0 & & \mathbf{d}_i^2 & & & & \\
& \mathbf{d}_{i+1}^1 & & & \ddots & & \\
\mathbf{d}_{i+2}^0 & & \vdots & & & \mathbf{d}_i^n = \mathbf{s}(t) & \\
& \vdots & & & & & \\
\vdots & & \mathbf{d}_{i+n-2}^2 & & & & \\
& \mathbf{d}_{i+n-1}^1 & & & & & \\
\mathbf{d}_{i+n}^0 & & & & & &
\end{array}
$$

$$
\begin{aligned}
\mathbf{d}_j^0 &= \mathbf{d}_j \\
\mathbf{d}_j^k &= (1 - \alpha_j^k)\, \mathbf{d}_j^{k-1} + \alpha_j^k\, \mathbf{d}_{j+1}^{k-1} \\
\alpha_j^k &= \frac{t - t_{j+k}}{t_{j+n+1} - t_{j+k}}
\end{aligned}
$$

Aufgabe: *Zeigen Sie (beispielsweise graphisch), dass für quadratische Spline-Kurven die Kurvenpunkte in den Knoten auf dem Kontrollpolygon liegen.*

2.3 Interpolation mit kubischen Spline-Kurven

Zur Interpolation einer gegebenen Punktemenge $\mathbf{p}_0, \ldots, \mathbf{p}_m$ müssen diesen zunächst Parameter $\hat{t}_j$ zugewiesen werden, so dass

$$
\mathbf{s}(\hat{t}_j) = \sum_i \mathbf{d}_i N_i^3(\hat{t}_j) = \mathbf{p}_j \ , \qquad j = 0, \ldots, m. \tag{4}
$$

Für die Wahl der Parameter ergeben sich verschiedene Möglichkeiten:

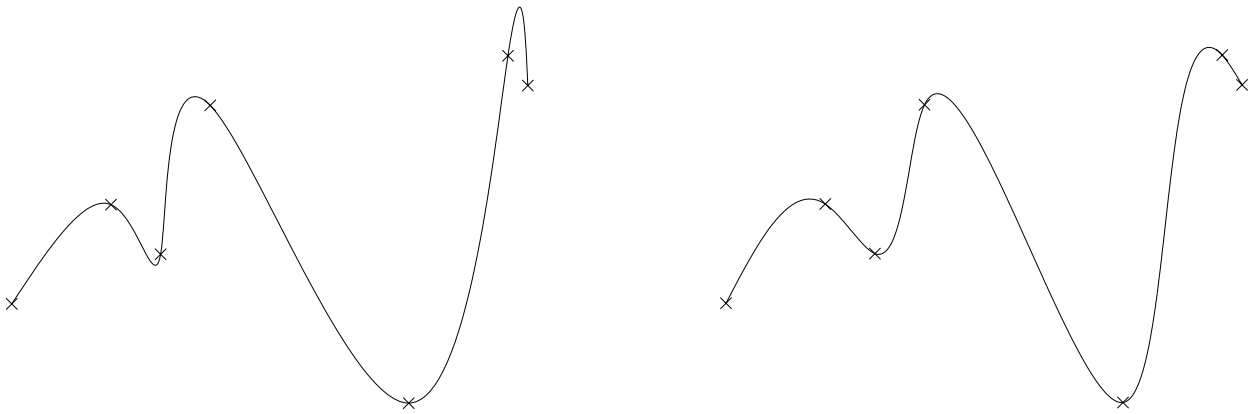
- **Äquidistante Parametrisierung**
  Bei äquidistanter (oder uniformer) Parametrisierung werden die Abstände $\hat{t}_{i+1} - \hat{t}_i$ für den inneren Knotenvektor konstant gewählt. Der Einfachheit halber wird hierbei $\hat{t}_i = i$ gesetzt. Diese Parameterwahl führt zu sehr einfachen Berechnungen, berücksichtigt aber nicht die geometrische Anordnung der zu interpolierenden Punkte.

- **Chordale Parametrisierung**
  Die chordale Parametrisierung approximiert die Bogenlängenparametrisierung der interpolierenden Kurve, d. h. eine Parametrisierung, bei der die Kurve mit konstanter Geschwindigkeit durchlaufen wird. Dazu werden die Längen der Parameterintervalle proportional zum Abstand aufeinander folgender Interpolationspunkte gewählt.

  Hierbei werden die Längenverhältnisse der Parameterintervalle zueinander näherungsweise den Verhältnissen der Bogenlängen der zugehörigen Kurvensegmente angepasst. Die Bogenlänge der Kurvensegmente wird durch die Länge der Verbindungsstrecke aufeinander folgender Interpolationspunkte approximiert.

  Die $\hat{t}_i$ werden demzufolge so bestimmt, dass

$$\hat{t}_{i+1} - \hat{t}_i = || \mathbf{p}_{i+1} - \mathbf{p}_i || \, . \tag{5}$$



- **Zentripetale Parametrisierung**
  Analog zur chordalen Parametrisierung werden als Längen der Parameterintervalle

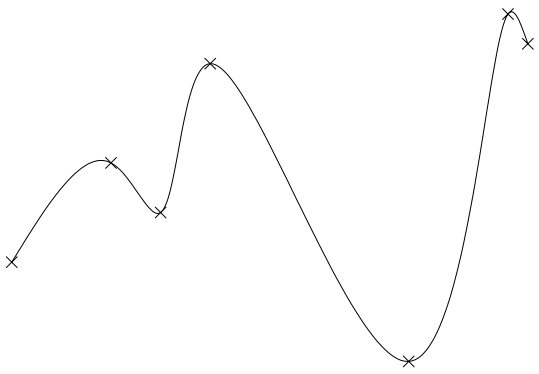
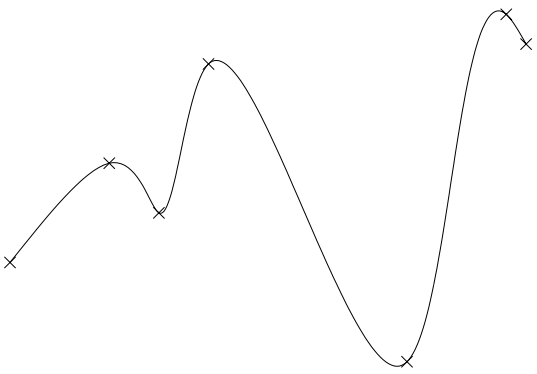
$$\hat{t}_{i+1} - \hat{t}_i = \sqrt{|| \mathbf{p}_{i+1} - \mathbf{p}_i ||} \tag{6}$$

  gewählt. Diese Parametrisierung entspricht physikalisch einer Fahrt längs eines Slalomkurses, bei der der Fahrer darauf achtet, dass die Normalbeschleunigung nicht zu groß wird. Dies wird beispielsweise dadurch erreicht, dass die Normalkräfte näherungsweise proportional zur Winkeländerung gewählt werden.

- **Parametrisierung nach Foley**
  Foley berücksichtigt in seinem Vorschlag zur Parametrisierung nicht nur die Abstände, sondern auch die Winkeländerungen in den Interpolationspunkten. Er schlägt als Parametrisierung vor (siehe auch [HOS]):

$$\hat{t}_{i+1} - \hat{t}_i = d_i \left(1 + \frac{3}{2} \frac{\hat{\Theta}_i d_{i-1}}{d_{i-1} + d_i} \; + \; \frac{3}{2} \frac{\hat{\Theta}_{i+1} d_{i+1}}{d_{i+1} + d_i} \right), \quad \hat{\Theta}_i = \min \left\{ \pi - \Theta_i, \frac{\pi}{2} \right\}, \tag{7}$$

  mit der Nielson-Metrik $d_i = M[\boldsymbol{p}](\mathbf{p}_i, \mathbf{p}_{i+1})$ und Θ_i als Winkel zwischen $(\mathbf{p}_{i-1}, \mathbf{p}_i)$ und $(\mathbf{p}_i, \mathbf{p}_{i+1})$. Zur Vereinfachung kann hier auch die chordale oder zentripetale Metrik eingesetzt werden, d. h. $d_i = ||\mathbf{p}_{i+1} - \mathbf{p}_i||$ ($d_{-1} := 0$, $d_m := 0$). Dieser Ansatz ist nur für ebene Kurven möglich!

Gewöhnlich werden dann die Knoten t_i in Übereinstimmung mit dieser Parametrisierung gewählt, so dass die Segmentgrenzen mit den zu interpolierenden Punkten übereinstimmen, d. h.

$$t_{i+3} = \hat{t}_i \ . \tag{8}$$

Um zu erreichen, dass die Randpunkte mit den Endpunkten des Kontrollpolygons zusammenfallen, wählt man dreifache Randknoten

$$\begin{aligned} t_1 = \quad t_2 \quad &= t_3 \ , \\ t_{m+3} = \ t_{m+4} \ &= t_{m+5} \ . \end{aligned} \tag{9}$$

Zu dem so bestimmten Knotenvektor gehören die Kontrollpunkte $\mathbf{d}_0, \ldots, \mathbf{d}_{m+2}$ (s. 2.1) und wegen (9) gilt

$$\begin{aligned} \mathbf{d}_0 = \quad \mathbf{s}(t_3) \quad &= \mathbf{p}_0 \ , \\ \mathbf{d}_{m+2} = \ \mathbf{s}(t_{m+3}) \ &= \mathbf{p}_m \ . \end{aligned} \tag{10}$$

Diese wie auch die folgenden Gleichungen erhält man beispielsweise unter Benutzung des De-Boor-Schemas.

Mit

$$\alpha_i = \frac{t_{i+2} - t_i}{t_{i+3} - t_i} \ , \qquad \beta_i = \frac{t_{i+2} - t_{i+1}}{t_{i+3} - t_{i+1}} \ , \qquad \gamma_i = \frac{t_{i+2} - t_{i+1}}{t_{i+4} - t_{i+1}} \tag{11}$$

ist dann

$$\begin{array}{rcl} (1-\beta_i) & \Big((1-\alpha_i) \ \mathbf{d}_{i-1} \ + \ \alpha_i \mathbf{d}_i \quad \Big) & \\ + \quad \beta_i & \Big((1-\gamma_i) \ \mathbf{d}_i \quad + \ \gamma_i \mathbf{d}_{i+1} \ \Big) & = \mathbf{s}(t_{i+2}) = \mathbf{p}_{i-1} \end{array} \tag{12}$$

für $i = 2, \ldots, m$.

(10) und (12) liefern uns also, der Anzahl der $\mathbf{p}_i$ entsprechend, $m+1$ Gleichungen für die $m+3$ unbekannten $\mathbf{d}_i$.

Um das Gleichungssystem eindeutig lösen zu können, müssen für die zwei fehlenden Gleichungen noch zwei Randbedingungen vorgegeben werden, was beispielsweise durch Vorgabe der beiden Randtangentenvektoren geschehen kann.

Eine weitere Möglichkeit ist die Forderung, dass die zweiten Ableitungen am Rand verschwinden. Die dadurch definierte Kurve wird *natürlicher Spline* genannt. Man erhält so die beiden Bedingungen

$$\begin{aligned} \mathbf{d}_0 - 2\mathbf{d}_1 + (1-\alpha_2)\mathbf{d}_1 + \alpha_2 \mathbf{d}_2 \quad &= 0 \ , \\ (1-\gamma_m)\mathbf{d}_m + \gamma_m \mathbf{d}_{m+1} - 2\mathbf{d}_{m+1} + \mathbf{d}_{m+2} \ &= 0 \ . \end{aligned} \tag{13}$$

Mit (10), (12) und (13) lässt sich dann ein tridiagonales Gleichungssystem aufstellen:

$$\begin{pmatrix} 1 & 0 & & & & & \\ -1 & 1+\alpha_2 & -\alpha_2 & & & 0 & \\ & * & * & * & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & * & * & * & \\ & 0 & & & -1+\gamma_m & -\gamma_m+2 & -1 \\ & & & & & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{d}_0^{\mathrm{t}} \\ \\ \\ \vdots \\ \\ \\ \mathbf{d}_{m+2}^{\mathrm{t}} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0^{\mathrm{t}} \\ \mathbf{0}^{\mathrm{t}} \\ \mathbf{p}_1^{\mathrm{t}} \\ \vdots \\ \mathbf{p}_{m-1}^{\mathrm{t}} \\ \mathbf{0}^{\mathrm{t}} \\ \mathbf{p}_m^{\mathrm{t}} \end{pmatrix} . \qquad (14)$$

Die Koeffizienten '$*$' lassen sich mit (12) bestimmen. Dieses Gleichungssystem muss jeweils für jede Koordinate gelöst werden.

2.4 Lösung linearer Gleichungssysteme mit tridiagonalen Matrizen

Oftmals reduziert sich (wie im vorliegenden Fall) das Problem der Berechnung der kubischen Splines mit natürlichen Randbedingungen auf das Problem der Lösung von linearen Gleichungssystemen mit tridiagonalen Matrizen. Im Folgenden werden solche Systeme mit Hilfe des so genannten *konzentrierten Gaußschen Algorithmus* gelöst.

Da die Kurven mit Hilfe eines Rechners erstellt werden sollen, müssen vorab einige Probleme erläutert werden, die auftreten können. Trotz exakter Algorithmen kann es vorkommen, dass man völlig falsche Ergebnisse bekommt, die durch Rundungen und Approximationen entstehen.

Man sagt, ein Algorithmus für ein numerisches Verfahren ist *stark stabil* bzw. *schwach stabil*, wenn ein im n-ten Rechenschritt zugelassener Rundungsfehler bei exakter Rechnung in den Folgeschritten abnimmt, bzw. von gleicher Größenordnung bleibt. Sonst ist der Algorithmus instabil.

Systeme, bei denen kleine Änderungen der Koeffizienten große Änderungen der Lösungen bewirken, nennt man *schlecht konditionierte Systeme*. Sind für die Koeffizienten nur gerundete Werte gegeben, können unbrauchbare Lösungen entstehen. Die Instabilität rührt daher, dass die Gleichungen *fast* linear abhängig sind. Durch Betrachten der Determinanten können Konditionsmaße definiert werden, die eine relative Aussage über die Kondition eines Systems machen.

Bei der Berechnung von kubischen Splines entstehen ganz spezielle Gleichungssysteme, die gut konditioniert sind. Außerdem ist in diesem Falle der Gaußsche Algorithmus ohne Pivotisierung numerisch stabil.

Gegeben sei ein lineares Gleichungssystem mit $A\mathbf{x} = \mathbf{d}$,

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \\ \vdots \\ \\ d_n \end{pmatrix},$$

wobei die $n \times n$–Matrix folgende Eigenschaften hat:

- tridiagonal: $a_{ij} = 0$ für $\mid i - j \mid > 1$
- diagonal–dominant $\mid b_{ii} \mid \geq \mid c_i \mid + \mid a_i \mid, i = 1, \ldots, n$
- für mindestens ein i gilt $\mid b_{ii} \mid > \mid c_i \mid + \mid a_i \mid, i = 1, \ldots, n$
- positive Hauptdiagonalelemente

Das bedeutet, dass die Matrix A positiv definit und invertierbar ist. Die Lösung des Gleichungssystems ist somit eindeutig bestimmt und auch einfach sowie numerisch stabil zu berechnen.

Der Gaußsche Algorithmus kann in Matrizenschreibweise wie folgt beschrieben werden: Die Matrix A wird zerlegt in das Produkt einer unteren Dreiecksmatrix mit einer normierten oberen Dreiecksmatrix

$$A = LU.$$

Für das Gleichungssystem erhält man

$$(LU)\mathbf{x} = \mathbf{d}.$$

Danach werden nacheinander die Systeme $L\mathbf{y} = \mathbf{d}$ mit $\mathbf{y} = U\mathbf{x}$ nach $\mathbf{y}$ und $U\mathbf{x} = \mathbf{y}$ nach $\mathbf{x}$ aufgelöst.

Für tridiagonale Matrizen ergibt sich folgende Zerlegung:

$$L = \begin{pmatrix} \beta_1 & & & & \\ \alpha_2 & \beta_2 & & 0 & \\ & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \\ & 0 & & \alpha_{n-1} & \beta_{n-1} \\ & & & \alpha_n & \beta_n \end{pmatrix}$$

und

$$U = \begin{pmatrix} 1 & v_2 & & & \\ & 1 & v_3 & & 0 \\ & & 1 & v_4 & \\ & & & \ddots & \ddots \\ & 0 & & & 1 & v_n \\ & & & & & 1 \end{pmatrix}.$$

Algorithmus zur Lösung von Gleichungssystemen mit tridiagonalen Matrizen:

$$LU = \begin{pmatrix} \beta_1 & \beta_1 v_2 & & & & \\ \alpha_2 & \alpha_2 v_2 + \beta_2 & \beta_2 v_3 & & 0 & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \alpha_{n-1} & \alpha_{n-1} v_{n-1} + \beta_{n-1} & \beta_{n+1} v_n \\ & & & & \alpha_n & \alpha_n v_n + \beta_n \end{pmatrix}$$

Vergleicht man die Elemente von LU mit denen der Matrix A, so hat man:

1. Schritt: Bestimmung der Koeffizienten von L und U:

$$\begin{array}{ccc} & \beta_1 = b_1 & v_2 = \frac{c_1}{\beta_1} \\ \alpha_2 = a_2 & \beta_2 = b_2 - \alpha_2 v_2 & v_3 = \frac{c_2}{\beta_2} \\ \vdots & \vdots & \vdots \\ \alpha_{n-1} = a_{n-1} & \beta_{n-1} = b_{n-1} - \alpha_{n-1} v_{n-1} & v_n = \frac{c_{n-1}}{\beta_{n-1}} \\ \alpha_n = a_n & \beta_n = b_n - \alpha_n v_n & \end{array}$$

2. Schritt: Auflösung von $L\mathbf{y} = \mathbf{d}$ nach $\mathbf{y}$:

$$\downarrow \begin{array}{rcl} y_1 & = & \frac{d_1}{\beta_1} \\ y_2 & = & \frac{d_2 - a_2 y_1}{\beta_2} \\ & \vdots & \\ y_n & = & \frac{d_n - a_n y_{n-1}}{\beta_n} \end{array}$$

3. Schritt: Auflösung von $U\mathbf{x} = \mathbf{y}$ nach $\mathbf{x}$:

$$\left\downarrow \begin{array}{rcl} x_n & = & y_n \\ x_{n-1} & = & y_{n-1} - v_n x_n \\ & \vdots & \\ x_1 & = & y_1 - v_2 x_2 \end{array} \right.$$

Da einige Variablen nur zwischenzeitlich auftreten, kann der Algorithmus in kompakter Form geschrieben werden:

$$\begin{array}{rcl} v_1 & := & a_1 := c_n := y_0 := 0 \\ \left.\begin{array}{rcl} z_k & := & \frac{1}{b_k - a_k v_k} \\ v_{k+1} & := & z_k \cdot c_k \\ y_k & := & z_k \cdot (d_k - a_k y_{k-1}) \end{array}\right\} & & k = 1, \ldots, n \\ x_n := y_n & & \\ x_k := y_k - v_{k+1} x_{k+1} & & k = n-1, n-2, \ldots, 1 \end{array}$$

3 Praktischer Teil

3.1 Programmierung

In `spline.py` fehlen die Methoden *knot_index* sowie *de_boor* und *interpolate_cubic*. Diese Methoden müssen noch programmiert werden.

knot_index soll für ein t den Index i liefern, für den $t \in [t_i, t_{i+1}), n \leq i < m+1$ gilt. Entsprechend diesem Index müssen dann die Indizes im De-Boor-Schema transformiert werden.

de_boor soll für ein t das De-Boor-Schema berechnen. In weiteren Aufgaben wird nicht nur der letzte Wert benötigt werden sondern auch Zwischenergebnisse. Deswegen gibt der zweite Parameter *stop* an, wann das De-Boor-Schema abgebrochen werden soll. Wenn in einer Spalte nur noch *stop* Werte stehen, sollen diese in einer Liste zurückgegeben werden.

Für die Methode *interpolate_cubic* muss ein tridiagonales Gleichungssystem gelöst werden. Dafür soll die Methode *solve_tridiagonal_equation* in der Datei `utils.py` implementiert werden.

Literatur

[BOE] *W. Boehm, G. Farin, J. Kahmann*, A survey of curve and surface methods in CAGD, Computer Aided Geometric Design 1, July 1984.

[FAR] *G. Farin*, Curves and Surfaces for Computer Aided Design, Second Edition 1990, Academic Press.

[HOS] *J. Hoschek, D. Lasser*, Grundlagen der geometrischen Datenverarbeitung, 2. Auflage 1992, Teubner Verlag, Stuttgart.

[PRA] *H. Prautzsch, W. Boehm, M. Paluszny*, Bézier- and B-Spline Techniques, 2002, Springer-Verlag, Berlin.