

# Numerisches Programmieren, Übungen

## 4. Übungsblatt: Stückweise Interpolation

### 1) Hermite-Interpolation

Ziel der Interpolation nach Hermite ist es, eine Funktion  $p(x)$  zu erhalten, die überall stetig differenzierbar ist ( $p \in \mathcal{C}^1 =$  keine Knicke). Um dies zu erreichen, müssen zusätzlich zu Stützpunkten  $(x_0, y_0), \dots, (x_n, y_n)$  auch noch die Werte der ersten Ableitung  $y'_0, \dots, y'_n$  an den Stützstellen vorgegeben werden. Damit ist es möglich, für  $p(x)$  auf jedem Teilintervall  $[x_i; x_{i+1}]$ ,  $i = 0, \dots, n - 1$ , ein kubisches Polynom zu erzeugen und diese Einzeldarstellungen stetig differenzierbar an den  $x_i$  zu „verkleben“.

- i) Betrachten Sie den einfachen Fall nur eines Teilintervalls mit  $x_0 = 0$  und  $x_1 = 1$ . Zusätzlich zu den Funktionswerten  $y_0, y_1$  seien auch die ersten Ableitungen  $y'_0, y'_1$  bei  $x_0$  und  $x_1$  gegeben.

Bestimmen Sie das kubische Polynom  $p(x)$ , das durch die Vorgabe dieser Werte

$$p(x_0) = y_0, \quad p(x_1) = y_1 \quad (1)$$

$$p'(x_0) = y'_0, \quad p'(x_1) = y'_1 \quad (2)$$

festgelegt ist! Nützen Sie hierfür den allgemeinen Ansatz für kubische Polynome

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \quad t \in [0; 1] = [x_0; x_1].$$

Bestimmen Sie anschließend mittels Koeffizientenvergleich die kubischen Basispolynome  $H_0, \dots, H_3$  des Hermite-Ansatzes

$$p(t) = y_0 \cdot H_0(t) + y_1 \cdot H_1(t) + y'_0 \cdot H_2(t) + y'_1 \cdot H_3(t), \quad t \in [0; 1] = [x_0; x_1].$$

- ii) Um die stückweise Hermite-Interpolation durchführen zu können, muss der Spezialfall  $x_0 = 0$  und  $x_1 = 1$  von Teilaufgabe i) auf ein beliebiges Teilintervall  $[x_i, x_{i+1}]$  übertragen werden. Geben Sie dazu zuerst die passenden vier Bedingungen für das lokale kubische Polynom an!

Überlegen Sie dann, wie mit Hilfe einer Transformationsfunktion  $t_i(x)$  sowie der bereits in i) berechneten Basispolynome  $H_0(t)$  bis  $H_3(t)$  eine passende Interpolationsfunktion  $p_i(t_i(x))$  auf dem Teilintervall  $[x_i, x_{i+1}]$  konstruiert werden kann! Insgesamt gilt dann:

$$p(x)|_{[x_i, x_{i+1}]} = p_i(t_i(x)).$$

## 2) Interpolation mit kubischen Splines

Bei der kubischen Spline-Interpolation möchte man eine interpolierende Funktion  $s(x)$  erhalten, die ähnlich wie bei der Hermite-Interpolation aus kubischen Teilpolynomen  $p_i(x)$  auf den Teilintervallen  $[x_i, x_{i+1}]$  besteht. Allerdings soll die Splinefunktion  $s(x)$  überall zweimal stetig differenzierbar sein ( $s \in \mathcal{C}^2$ ) und dafür keine anderen Informationen benötigen als die  $n + 1$  zu interpolierenden Stützpunkte  $(x_0, y_0), \dots, (x_n, y_n)$ .

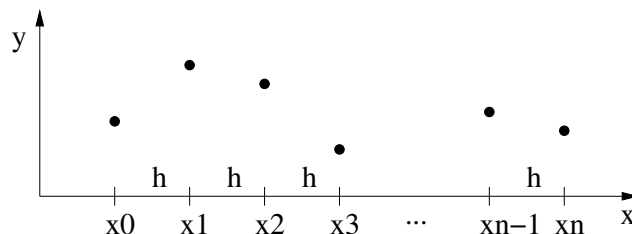


Abbildung 1: Visualisierung der äquidistanten Stützstellen  $x_i$ .

In dieser Aufgabe beschränken wir uns auf den Fall äquidistanter Stützstellen  $x_i$ , d.h. alle Teilintervalle haben dieselbe Länge (vgl. Abb. 1):

$$x_{i+1} - x_i = h := \frac{x_n - x_0}{n}, \quad i = 0, 1, \dots, n - 1.$$

- i) Geben Sie die Bedingungen an, die aus der Interpolation und der  $\mathcal{C}^2$ -Stetigkeit resultieren!
- ii) Berechnen Sie die zweiten Ableitungen  $H_i''(t)$ ,  $i = 0, \dots, 3$ , der Hermite-Basispolynome  $H_i(t)$  aus Aufgabe 1) und werten Sie sie an den Stellen  $t = 0$  und  $t = 1$  aus!
- iii) Zeigen Sie, dass  $s(x)$  mit Hilfe der Stützwerte  $y_i$  und folgendem linearem Gleichungssystem konstruiert werden kann:

$$\frac{1}{h} \begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & & 1 & 4 \end{pmatrix} \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \end{pmatrix} = \frac{3}{h^2} \begin{pmatrix} y_2 - y_0 - \frac{h}{3}y'_0 \\ y_3 - y_1 \\ \vdots \\ y_{n-1} - y_{n-3} \\ y_n - y_{n-2} - \frac{h}{3}y'_n \end{pmatrix}.$$

- iv) Bestimmen Sie die Spline-Funktion  $s(x)$  für die Stützpunkte

$$P_0 = (-1, 2), \quad P_1 = (0, 0), \quad P_2 = (1, 2), \quad P_3 = (2, 3)$$

und die Randbedingungen

$$s'(-1) = 9, \quad s'(2) = 0.$$

Zum Abschluss des Kapitels über Interpolation ist nachfolgend wieder eine Aufgabe aus einer Semestralklausur aufgeführt. Diese ist dazu gedacht, den Stoff noch einmal selbst üben zu können. Sie wird deshalb nicht in der Übung behandelt.

Die in der Angabe enthaltenen Code-Stücke entstammen einer behandelten Programmieraufgabe. Sie können jedoch auch unabhängig davon verstanden werden.

## Wiederholung: Interpolation

Betrachten Sie im Folgenden die Funktion

$$f : [0, 2] \rightarrow [0, 1], \quad f(x) = \sin\left(\frac{\pi x}{2}\right).$$

- a) Bestimmen Sie einen Polynom-Interpolanten  $p(x)$ , der die Funktion  $f$  interpoliert und die drei Stützpunkte  $P_0 = (x_0, y_0)$ ,  $P_1 = (x_1, y_1)$ , and  $P_2 = (x_2, y_2)$  mit den zugehörigen Stützstellen

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 2$$

besitzt.

Berechnen Sie dazu die dividierten Differenzen und geben Sie eine geschlossene Darstellung des Interpolanten  $p(x)$  an, indem Sie die Newton'sche Interpolationsformel verwenden.

- b) Um eine bessere Approximation zu erzielen, werden die Stützpunkte um einen neuen Punkt  $P_3 = (x_3, y_3)$  mit  $x_3 = \frac{1}{3}$  erweitert. Bestimmen Sie den neuen Interpolanten, der die vier Punkte  $P_0, P_1, P_2$  und  $P_3$  interpoliert.
- c) Sei  $p_1(u)$  ein Interpolant zu beliebig gegebenen Stützstellen  $\{u_0, u_1, u_2, u_3\}$  und  $p_2(u)$  ein Interpolant zu Stützstellen  $\{u_1, u_2, u_3, u_4\}$ . Bestimmen Sie einen Interpolanten  $p_3(u)$  zu den Stützstellen  $\{u_0, u_1, u_2, u_3, u_4\}$ , basierend auf  $p_1(u)$  und  $p_2(u)$ !
- d) Welche Methode eignet sich zur Interpolation von ca. 100 äquidistanten Stützpunkten? Geben Sie zwei Vorteile gegenüber der Polynom-Interpolation an!
- e) Die Methode `aitken` des folgenden Java-Codefragments soll die Polynominterpolation mit dem Schema nach Aitken-Neville berechnen. Dabei bezeichnen `xs` den Vektor der x-Koordinaten der Stützstellen, `f` den Vektor der Stützwerte und `x` die gewünschte Auswertungsstelle des Interpolationspolynoms. Zeigen Sie 2 Fehler auf und korrigieren Sie diese.

```

01 public static double aitken(double xs[], double f[], double x) {
02     int n = f.length;
03     assert n==xs.length;
04
05     for(int k = 1; k <= n; k++)
06         for(int i = 0; i < n-k; i++)
07             f[i] = ((x-xs[i])*f[i+1] - (x-xs[i+k])*f[i])/(xs[i]-xs[i+k]);
08
09     return f[n-1];
10 }

```

- f) Erklären Sie kurz, was die Methode `foo()` des folgenden Java-Codefragments unter Verwendung der Methode `aitken` aus Teilaufgabe e) berechnet.

```

01 public static double foo(double xs[], double ys[],
02                          double f[][] , double x, double y) {
03
04     assert xs.length == ys.length;
05     assert xs.length == f.length;
06     assert ys.length == f[0].length;
07
08     double pHat[] = new double[xs.length];
09
10     for(int i=0; i < xs.length; i++) pHat[i]=aitken(ys,f[i],y);
11     return aitken(xs,pHat,x);
12 }

```