Numerische Analysis im SS 2019

Frank Wübbeling

24. April 2019

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der vorliegende Text entstand als Begleitmaterial zur Vorlesung Numerische Analysis im Sommersemester 2019. Die Vorlesung richtet sich an Studierende des Bachelorstudiengangs Mathematik im vierten Semester sowie Studierende in den Lehramtsstudiengängen Mathematik. Für die Korrektheit des Textes wird keinerlei Garantie übernommen, vermutlich sind noch reichlich Schreibfehler enthalten. Für Bemerkungen und Korrekturen bin ich dankbar.

Macht es Sinn, der großen, bereits existierenden Zahl von Skripten zu Einführungsveranstaltungen der Numerischen Mathematik noch ein weiteres hinzuzufügen? Die Antwort ist wohl ja, denn zumindest die Auswahl der Themen und vor allem Schwerpunkte im breiten Spektrum geschieht subjektiv durch den Dozenten.

Da der Großteil der Studierenden heute kaum noch einen physikalischen Hintergrund hat, habe ich auf die Darstellung der Beziehungen zwischen Angewandter Mathematik und Physik, wie sie in den klassischen Lehrbüchern und Vorlesungen üblich war, größtenteils verzichtet. Übungen zu den einzelnen Kapiteln finden sich im Netz, ebenso eine ausführliche (subjektive) Literaturliste.

Ich habe mich bemüht, zu allen vorgestellten Algorithmen eine Beispiel–Implementation in Matlab zu liefern. Einige Programme nutzen dabei die Imaging–Toolbox oder die SymbolicMath–Toolbox. Die zugehörigen Dateien sind in der PDF–Datei enthalten. Klick auf die jeweilige Textstelle öffnet die Beispielimplementation in Matlab. Ebenso sind alle Bilder beigefügt, Klick liefert jeweils das zugehörige Bild. Dies funktioniert in Acrobat (Reader) und in einigen anderen PDF–Readern, in vielen PublicDomain–Readern aber nicht.

In 2019 wird diese Vorlesung komplettiert durch eine Sammlung von Python–Notebooks, in der alle wesentlichen Algorithmen implementiert und dargestellt

sind. Diese Notebooks sind als Extra–Download auf der Seite https://www.uni-muenster.de/AMM/Veranstaltungen/SS19/NumAna/.

Billerbeck, im Frühjahr 2019

Frank Wübbeling

1.1 Einführung in die Numerische Mathematik

Buchübersicht. Grundlegende Begriffe.

Kapitel 2

Interpolation

Als Interpolation bezeichnen wir die Aufgabe, vorgegebene Funktionswerte auf sinnvolle Weise zu einer Funktion zu ergänzen. Die Interpolation tritt in vielen praktischen Varianten auf. Einige Beispiele:

1. Gegeben seien $(N+1)$ Punkte x_i, $i = 0 \ldots N$, in der Ebene. Verbinde die Punkte durch eine glatte Kurve, d.h. finde eine (differenzierbare) Funktion $s : [0, N] \mapsto \mathbb{R}^2$: $s(i) = x_i$, $i = 0 \ldots n$.

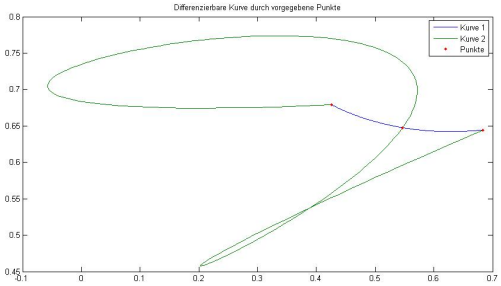


Abbildung 2.1: Interpolationsfunktionen

Klick für Bild interpol

2. Wir nehmen eine Holzlatte (**Straklatte**, engl. **Spline**) und biegen sie so, dass sie durch einige vorgegebene Punkte geht. Welche Form nimmt die Holzlatte an, bzw. welche Funktion stellt sie dar? Mathematisch: Die Energie, die in der Biegung einer Funktion steckt, ist proportional zum Integral über das Quadrat der zweiten Ableitung. Es sollte also die Norm der zweiten Ableitung der Formfunktion minimiert werden.

Sei $x_i \in [a,b]$, $i = 0 \dots N$. Finde $s : [a,b] \mapsto \mathbb{R}$, $s \in C^2([a,b])$, so dass $s(x_i) = y_i$, $i = 0 \dots N$ und

$$\int_a^b (s''(x))^2 \mathrm{dx} \leq \int_a^b g''(x)^2 \mathrm{dx}$$

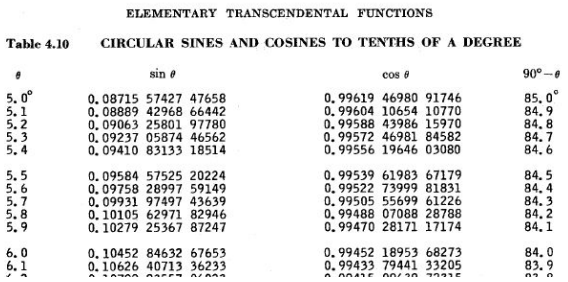
für alle Funktionen $g : [a,b] \mapsto \mathbb{R}$ mit $g(x_i) = y_i$, $i = 0 \dots N$, $g \in C^2([a,b])$.

Abbildung 2.2: Straklatte im Schiffsbau

Klick für Bild straklatte

Bemerkung: Außerhalb des Intervalls, in dem die Stützwerte liegen, ist die Funktion linear.

3. Eine Funktion h sei vertafelt (z.B. in dem Buch von Abramowitz and Stegun [1965]), und es seien die Werte $h(x_i) = y_i$ bekannt. Finde eine möglichst gute Näherung für h an einer Zwischenstelle ξ. Wie groß ist der maximale Fehler?



ELEMENTARY TRANSCENDENTAL FUNCTIONS

Table 4.10 CIRCULAR SINES AND COSINES TO TENTHS OF A DEGREE

θ	$\sin\theta$	$\cos\theta$	$90° - \theta$
5.0°	0.08715 57427 47658	0.99619 46980 91746	85.0°
5.1	0.08889 42968 66442	0.99604 10654 10770	84.9
5.2	0.09063 25801 97780	0.99588 43986 15970	84.8
5.3	0.09237 05874 46562	0.99572 46981 84582	84.7
5.4	0.09410 83133 18514	0.99556 19646 03080	84.6
5.5	0.09584 57525 20224	0.99539 61983 67179	84.5
5.6	0.09758 28997 59149	0.99522 73999 81831	84.4
5.7	0.09931 97497 43639	0.99505 55699 61226	84.3
5.8	0.10105 62971 82946	0.99488 07088 28788	84.2
5.9	0.10279 25367 87247	0.99470 28171 17174	84.1
6.0	0.10452 84632 67653	0.99452 18953 68273	84.0
6.1	0.10626 40713 36233	0.99433 79441 33205	83.9

Abbildung 2.3: Vertafelter sinus im Buch von Abramowitz und Stegun

Klick für Bild Abramo

4. Eine (transzendente) Funktion h soll auf einem Rechner ausgewertet werden, der nur die Grundrechenarten beherrscht. Gesucht ist eine berechenbare Funktion g, so dass

$$||h(x) - g(x)|| \leq \epsilon.$$

Dies ist ausdrücklich **keine** Interpolationsaufgabe. Wir schreiben nicht vor, dass die gesuchte Funktion durch feste Punkte gehen soll, sondern verlangen, dass sie sich einer gegebenen Kurve möglichst gut annähert. Dies ist das Problem der **Approximation** und wird in der Vorlesung Numerische Lineare Algebra ausführlich behandelt.

5. Ein Bild $f : \{1 \dots N\} \times \{1 \dots M\} \mapsto \mathbb{R}$ soll möglichst platzsparend abgespeichert werden. Hierzu bestimmen wir zunächst Koeffizienten a_{ik}, so dass

$$f(l,j) = \sum_{i,k} a_{ik} f_{ik}(l,j)$$

ist (Interpolationsschritt). Hierbei sind die f_{ik} z.B. trigonometrische Funktionen. Beim Abspeichern des Bildes ersetzen wir a_{ik} durch 0, falls sein Betrag klein ist, hierdurch wird eine Komprimierung erreicht. Zum Anzeigen wird die Näherung

$$\widetilde{f}(l,j) = \sum_{i,k} \widetilde{a}_{ik} f_{ik}(l,j)$$

berechnet, hierbei sind $\widetilde{a}$ die abgespeicherten Koeffizienten. Diese Methode ist die Standardmethode zur **Komprimierung in der Bildverarbeitung** (JPEG, MPEG, MP3).

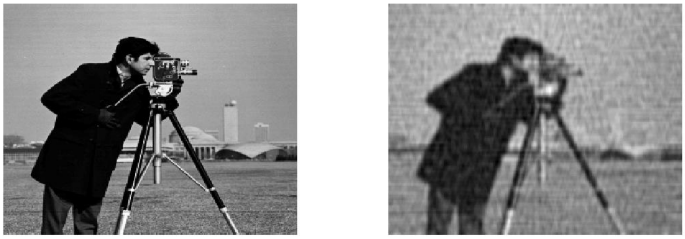


Abbildung 2.4: Original, zu stark komprimiertes Bild
Klick für Bild man
Klick für Bild man2

```
function comprimg
%COMPRIMG take cameraman.tif and delete small
%coefficients in the cosine transform
A=imread('cameraman.tif');
A=double(A);
B=dct2(A);
```

Listing 2.1: Bildkompression (interpolation/comprimg.m)

Klicken für den Quellcode von interpolation/comprimg.m

2.1 Polynominterpolation

Definition 2.1 (Allgemeine Interpolationsaufgabe)
Sei $N \in \mathbb{N}$. Seien $x_0, \ldots, x_N$ paarweise verschiedene Werte in $\mathbb{C}$ oder $\mathbb{R}$. Seien weiter $y_0, \ldots, y_N$ Elemente in $\mathbb{C}$ oder $\mathbb{R}$. Dann ist die **allgemeine Interpolationsaufgabe***:*
Suche eine Funktion f mit der Eigenschaft, dass $f(x_i) = y_i \, \forall i = 0..N$.
Die x_i heißen **Stützpunkte***, die y_i heißen* **Stützwerte***.*

In dieser Allgemeinheit, also ohne Einschränkung des zulässigen Funktionenraums, besitzt die Aufgabe offensichtlich unendlich viele Lösungen. Wir suchen f daher immer in einem angegebenen Funktionenraum, z.B den Polynomen.

Definition 2.2 (Aufgabe der Polynominterpolation, Polynomraum)
Sei $N \in \mathbb{N}$. Dann ist $\mathcal{P}_N$ der Raum der Polynome vom Grad kleiner oder gleich N.
Seien $x_0, \ldots, x_N$ paarweise verschieden, $y_0, \ldots, y_N$ gegeben. Dann ist die Aufgabe der **Polynominterpolation***:*
Finde ein $p \in \mathcal{P}_N$ mit $p(x_i) = y_i \, \forall i = 0 \ldots N$.

Damit gilt:

Satz 2.3 *Die Aufgabe der Polynominterpolation ist* **eindeutig** *lösbar.*

Beweis:

1. Formel von Lagrange, **Existenz** einer Lösung: Sei

$$w_j(x) := \prod_{\substack{k=0 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k}, \; j = 0 \ldots N.$$

   Dann ist $w_j(x) \in \mathcal{P}_N$, und $w_j(x_i) = \delta_{ji}$ für $j, i = 0 \ldots N$, denn für $j \neq i$ ist x_i Nullstelle des Zählers. Dabei ist

$$\delta_{ji} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

   das Kronecker–Delta. Sei

$$p(x) := \sum_{j=0}^{N} y_j w_j(x).$$

Dann ist $p \in P_N$, und es gilt

$$p(x_i) = \sum_{j=0}^{n} y_j w_j(x_i) = y_i w_i(x_i) = y_i$$

für alle $i = 0 \dots N$.

2. **Eindeutigkeit** der Lösung: Seien p_1 und p_2 Lösungen der Polynominterpolationsaufgabe. Sei $p = p_1 - p_2$. Dann ist $p \in \mathcal{P}_N$, und es gilt

$$p(x_i) = p_1(x_i) - p_2(x_i) = y_i - y_i = 0$$

für alle $i = 0 \dots N$. Also ist p ein Polynom vom Grad kleiner oder gleich N mit $N + 1$ Nullstellen, also ist nach dem Fundamentalsatz der Algebra $p = 0$, und damit $p_1 = p_2$.

□

Die Formel von Lagrange sichert die Existenz einer Lösung und gibt sie konstruktiv an, zur Auswertung eignet sie sich aber nicht, denn zur (naiven) Auswertung von $p(\widetilde{x})$ nach dieser Formel werden $N(N+1)$ Divisionen und Multiplikationen benötigt. Alternativ kann man die Koeffizienten des Interpolationspolynoms mit Hilfe der Vandermondematrizen bestimmen.

Definition 2.4 (Vandermondematrizen)
Es seien x_i, $i = 0 \dots N$ paarweise verschieden. Die Matrix $V \in \mathbb{C}^{n \times n}$, $V_{ik} = (x_i)^k$, $i, k = 0 \dots N$, heißt Vandermondematrix zu $x_0, \dots, x_N$.

Also:

$$V(x_0, \dots, x_N) = \begin{pmatrix} x_0^0 & \dots & x_0^N \\ \vdots & \ddots & \vdots \\ x_N^0 & \dots & x_N^N \end{pmatrix}$$

Satz 2.5 (Invertierbarkeit der Vandermondematrizen)
Seien $x_0, \dots, x_N$ paarweise verschiedene Zahlen, $y_0, \dots, y_N$ in $\mathbb{R}$ oder $\mathbb{C}$. Sei $p(x) = \sum_{k=0}^{N} a_k x^k$. Sei $y = (y_0, \dots, y_N)^t$, $a = (a_0, \dots, a_N)^t$, $V = V(x_0, \dots, x_N)$ Vandermonde–Matrix zu $x_0, \dots, x_N$. Dann gilt:

1. *p ist genau dann Lösung des Polynominterpolationsproblems, wenn $Va = y$.*
2. *V ist invertierbar.*

Beweis:

1. Es gilt $(Va)_j = p(x_j)$ und $p \in \mathcal{P}_N$.

2. Die Interpolationsaufgabe besitzt eine eindeutige Lösung nach 2.3, also ist V injektiv und surjektiv, also invertierbar.

□

Bemerkung: Da V quadratisch ist, reicht schon injektiv oder surjektiv allein zum Beweis der Invertierbarkeit. Dies ist einer der einfachsten Beweise der Abschätzung der Zahl der Nullstellen nach oben im Fundamentalsatz der Algebra: Mit Lagrange folgt, dass V surjektiv ist, also injektiv, also gibt es nur ein Polynom in $\mathcal{P}_N$ mit $(N+1)$ Nullstellen, das Nullpolynom.

Damit lassen sich die Koeffizienten eines Interpolationspolynoms durch Lösung eines linearen Gleichungssystems der Ordnung $(N+1)$ bestimmen, der Aufwand dazu beträgt $N^3/2$ Rechenoperationen (siehe Numerische LA), wobei wir eine Addition und eine Multiplikation zu einer Rechenoperation zusammenfassen. Das Polynom kann dann mit N Additionen und N Multiplikationen ausgewertet werden mit Hilfe des **Horner–Schemas**

$$p(x) = a_0 + x(a_1 + x(a_2 + (\ldots + a_n x))).$$

p lässt sich auch rekursiv aufbauen. Dies ist von Vorteil, wenn nachträglich eine Stützstelle hinzugefügt werden soll, bei Berechnung mit der Vandermonde–Matrix müsste in diesem Fall komplett neu gerechnet werden.

Satz 2.6 (Formel von Neville)
Gegeben sei die Polynominterpolationsaufgabe mit Stützstellen $x_0, \ldots, x_N$ und Stützwerten $y_0, \ldots, y_N$. Sei $p_{i\ldots k}$ die Lösung der Aufgabe für die Stützstellen $x_i, \ldots, x_k$ und Stützwerte $y_i, \ldots, y_k$, also

$$p_{i\ldots k} \in \mathcal{P}_{k-i},\ p(x_j) = y_j,\ j = i \ldots k.$$

Dann ist $p = p_{0\ldots N}$ die Lösung der vollen Aufgabe, und es gilt

$$p_{i\ldots k+1}(x) = \frac{1}{x_i - x_{k+1}} \left((x - x_{k+1})\, p_{i\ldots k}(x) + (x_i - x) p_{i+1\ldots k+1}(x) \right).$$

Beweis: Sei q das Polynom auf der rechten Seite. Dann ist $q \in \mathcal{P}_{k+1-i}$, und durch Einsetzen sieht man $q(x_j) = y_j$ für $j = i \ldots k+1$. Also ist q die eindeutige Lösung der Polynominterpolationsaufgabe und damit $q = p_{i\ldots k+1}$. □

Die Formel von Neville erlaubt die rekursive Berechnung von $p_{i\ldots k+1}$ aus $p_{i\ldots k}$ und $p_{i+1\ldots k+1}$ mit Hilfe des folgenden Schemas (N=2):

$$
\begin{array}{llll}
x_0 & y_0 = p_0 & & \\
 & & p_{01} & \\
x_1 & y_1 = p_1 & & p_{012} \\
 & & p_{12} & \\
x_2 & y_2 = p_2 & &
\end{array}
$$

Kommt nun nachträglich eine weitere Stützstelle x_3 mit Stützwert y_3 hinzu, so wird einfach an dieses Schema unten eine weitere Reihe angehängt.

```
function out = neville( x,y )
%NEVILLE
%Compute interpolating polynomial through x,y
%Using cell arrays

if (nargin<1)
```

Listing 2.2: Polynomberechnung mit dem Neville–Schema (interpolation/neville.m)

Klicken für den Quellcode von interpolation/neville.m

Die Formel von Neville kann auch eingesetzt werden, um den Wert des Interpolationspolynoms an einer Stelle $x = z$ direkt auszurechnen (ohne explizite Berechnung der Koeffizienten des Polynoms). Hierzu wird im Neville–Schema jeweils direkt z eingesetzt (s. Beispiele).

```
function [out,out1] = nevilleeval( x,y,z )
%NEVILLEEVAL
%Evaluate interpolating polynomial through x,y at z
if (nargin<1)
    x=[-1 0 2];
    y=[1 2 3];
```

Listing 2.3: Auswertung mit dem Neville–Schema (interpolation/nevilleeval.m)

Klicken für den Quellcode von interpolation/nevilleeval.m

Nachteil bei der Berechnung des Interpolationspolynoms mit dem Neville–Schema ist, dass alle Einträge im Schema Polynome sind. Dies umgehen wir mit der zweiten Art der rekursiven Berechnung des Interpolationspolynoms mit der Form von Newton. Mit den Bezeichnungen aus der Formel von Neville gilt die

Satz 2.7 (Formel von Newton)

$$
p_{i\ldots k}(x) = p_{i\ldots k-1}(x) + \frac{(y_k - p_{i\ldots k-1}(x_k))}{(x_k - x_i)\cdots(x_k - x_{k-1})}(x - x_i)\cdots(x - x_{k-1}).
$$

Beweis: Die rechte Seite hat die richtige Ordnung. Da der zweite Summand verschwindet für $x_i \dots x_{k-1}$ und $p_{i\dots k-1}$ das Interpolationspolynom für $x_i \dots x_{k-1}$ ist, liefert die rechte Seite für diese Stützstellen den korrekten Wert. Der zweite Summand ist dann gerade so gebaut, dass er auch für x_k den richtigen Wert liefert, also ist die rechte Seite das gesuchte Interpolationspolynom $p_{i\dots k}$. □

Definition 2.8 (Dividierte Differenzen)
Der Koeffizient $[y_i, \dots, y_k] = \frac{(y_k - p_{i\dots k-1}(x_k))}{(x_k - x_i)\cdots(x_k - x_{k-1})}$ *in der Formel von Newton heißt dividierte Differenz von* $y_i, \dots, y_k$.

Satz 2.9 (Interpolation nach Newton, Rekursion der dividierten Differenzen)

1. $[y_i, \dots, y_k]$ *ist der Höchstkoeffizient (Koeffizient von* x^{k-i}*) in* $p_{i\dots k}$.
2. *Es gilt* $[y_i] = y_i$ *und*
$$[y_i, \dots, y_{k+1}] = \frac{1}{x_i - x_{k+1}}([y_i, \dots, y_k] - [y_{i+1}, \dots, y_{k+1}].$$
3. *Es gilt*
$$p(x) = \sum_{j=0}^{N} [y_0, \dots, y_j](x - x_0)\cdots(x - x_{j-1}).$$

Beweis:

1. Folgerung aus der Formel von Newton.
2. Folgerung aus 1. und der Formel von Neville.
3. Folgerung aus der Formel von Newton.

□

Ähnlich wie beim Neville–Schema wird auch die Newton–Form rekursiv berechnet:

$$\begin{array}{llll} x_0 & y_0 = [y_0] & & \\ & & [y_0, y_1] & \\ x_1 & y_1 = [y_1] & & [y_0, y_1, y_2] \\ & & [y_1, y_2] & \\ x_2 & y_2 = [y_2] & & \end{array}$$

Hierbei sind die Einträge im Schema, die dividierten Differenzen, anders als im Neville–Schema natürlich nur Zahlen, was die Berechnung erheblich vereinfacht.

```
function out = divdiff( x,y)
%divdiff
%compute divided differences of x and y
if (nargin<1)
    x=[-1 0 2];
    y=[1 2 3];
```

Listing 2.4: Berechung der Dividierten Differenzen (interpolation/divdiff.m)

Klicken für den Quellcode von interpolation/divdiff.m

Die Newton–Form lässt sich ähnlich wie das Horner–Schema auswerten:

$$p(x) = [y_0] + (x - x_0)([y_0, y_1] + (x - x1)([y_0, y_1, y_2] + \ldots)).$$

Beispiel 2.10 *Sei* $N = 2$, $x_0 = -1$, $x_1 = 0$, $x_2 = 2$, $y_0 = 1$, $y_1 = 2$, $y_2 = 3$.

Lagrange

$$\begin{aligned} w_0(x) &= \frac{(x-0)(x-2)}{(-1-0)(-1-2)} \\ w_1(x) &= \frac{(x-(-1))(x-(-2))}{(0-(-1))(0-2)} \\ w_2(x) &= \frac{(x-(-1))(x-0)}{(2-(-1))(2-0)} \\ p(x) &= 1w_0(x) + 2w_1(x) + 3w_2(x) = -x^2/6 + 5/6x + 2 \end{aligned}$$

Neville

$$\begin{array}{lll} -1 & 1 & 1 \\ & & \frac{1}{-1}(x + 2(-1-x) = 2 + x \\ 0 & 2 & 2 \\ & & \frac{1}{-2}(2(x-2) + 3(-x)) = 2 + x/2 \\ 2 & 3 & 3 \end{array} \qquad \frac{1}{-3}((x-2)(x+2) + (-1-x)(2+x/2)) = -x^2/6 + 5/6x + 2$$

Neville, ausgewertet für $x = 1$*:*

$$\begin{array}{lll} -1 & 1 & 1 \\ & & \frac{1}{-1}(1 + (-2) \cdot 2) = 3 \\ 0 & 2 & 2 \qquad\qquad \frac{1}{-3}(-1 \cdot 3 + (-2) \cdot (\frac{5}{2})) = \frac{8}{3} \\ & & \frac{1}{-2}(-1 \cdot 2 + (-1) \cdot 3) = \frac{5}{2} \\ 2 & 3 & 3 \end{array}$$

Newton

$$\begin{array}{lll} -1 & 1 & 1 \\ & & \frac{1}{-1}(1 - 2) = 1 \\ 0 & 2 & 2 \qquad\qquad \frac{1}{-3}(\frac{1}{2}) = -1/6 \\ & & \frac{1}{-2}(2 - 3) = 1/2 \\ 2 & 3 & 3 \end{array}$$

und damit

$$p(x) = p_{012}(x) = 1 + (x + 1) - 1/6x(x + 1) = -x^2/6 + 5/6x + 2.$$

Vandermonde

$$V(-1, 0, 2) = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 2 & 4 \end{pmatrix}, \; y = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

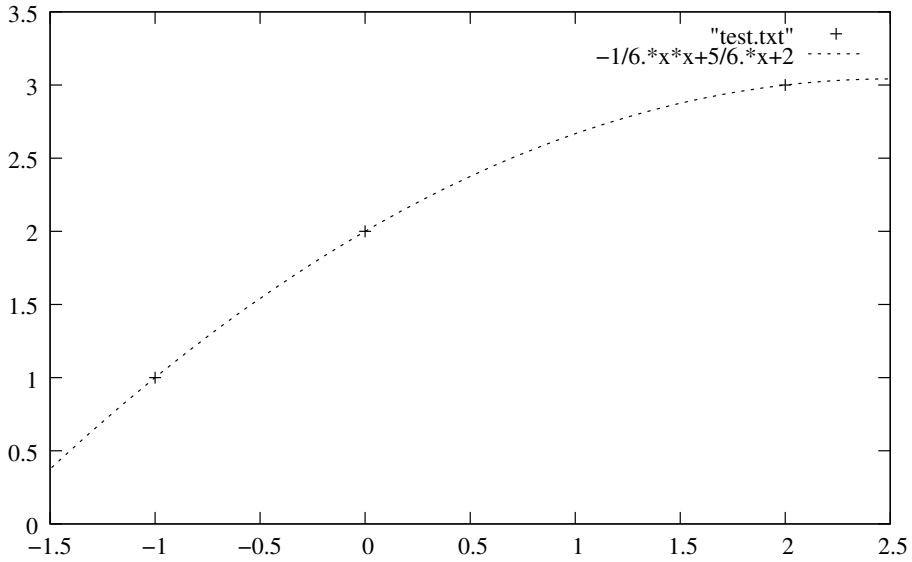
Es ergibt sich sofort $a_0 = 2$ *und*

$$-2a_1 + 2a_2 = -2, \; 2a_1 + 4a_2 = 1$$

und damit $a_2 = -1/6$*,* $a_1 = 5/6$*.*

Natürlich erzeugen alle Rechenvorschriften dasselbe Interpolationspolynom.

Im Diagramm erkennen wir, dass das Polynom die Punkte glatt verbindet (natürlich, denn es ist vom Grad 2).



Matlab-Code zur Berechnung der Koeffizienten des Interpolationspolynoms:

```
function p=interpolate(x,y)

%Find coefficients of interpolating polynomial
%by solving the linear equation with the Vandermonde matrix.
%Plot the result.
n=numel(x);
```

Listing 2.5: Polynominterpolation (interpolation/interpolate.m)

Klicken für den Quellcode von interpolation/interpolate.m

2.2 Interpolationsfehler bei Polynomen

Wir können die Polynominterpolation nutzen, um Funktionen zu approximieren. Sei in diesem Abschnitt f eine Funktion auf dem Intervall $[a, b] \subset \mathbb{R}$ nach $\mathbb{R}$. Wir werten f an den paarweise verschiedenen Stellen x_i, $i = 0 \dots N$, aus und betrachten das Interpolationspolynom p_N mit den Stützstellen x_i und Stützwerten $f(x_i)$, $i = 0 \dots N$.

$$p_N - f$$

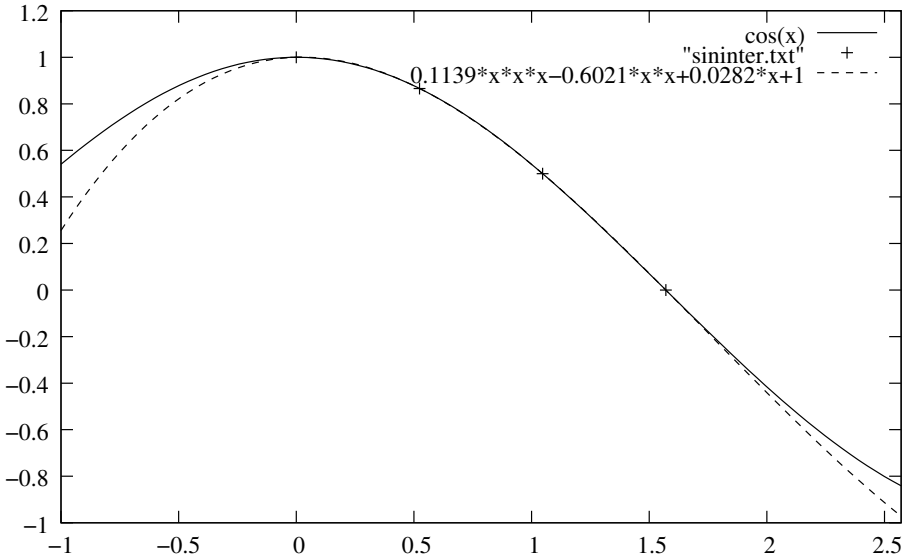
heißt **Interpolationsfehler**.

Ist p_N eine gute Approximation an f, also der Betrag des Interpolationsfehlers klein? Konvergiert p_N für wachsendes N und im Intervall $[a, b]$ gleichverteilte (äqui-

distante) Stützstellen $x_0, \ldots, x_N$ punktweise gegen f, wie wir es sicherlich erwarten?

Leider ist die Antwort häufig negativ (abhängig von f), die Polynominterpolation eignet sich nur begrenzt zur Approximation von Funktionen. Zwei typische Beispiele für äquidistante Stützstellen $x_k = a + k * (b - a)/N$, $k = 0 \ldots N$, folgen.

Beispiel 2.11 (Interpolation des Cosinus)

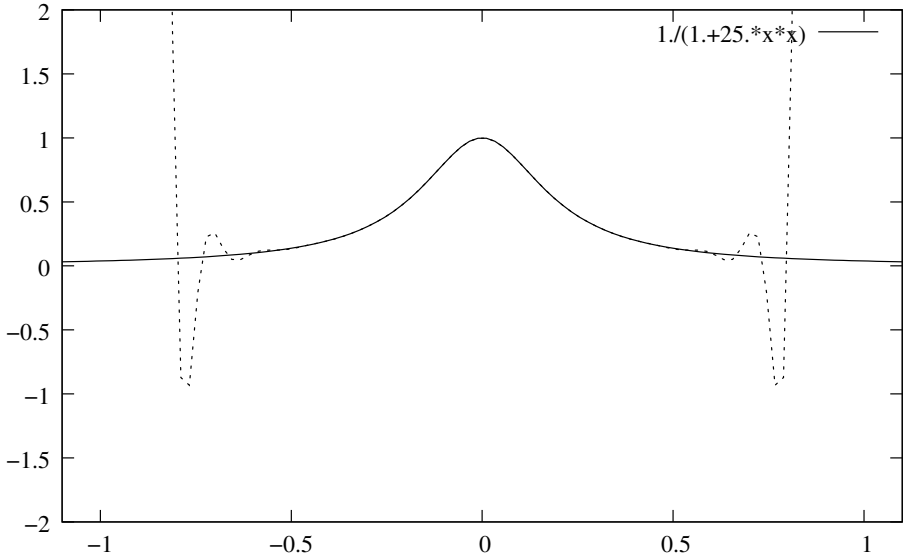


Interpolation des Cosinus auf $[0, \pi/2]$ mit vier Stützpunkten. Die Approximation ist bereits so exakt, dass innerhalb des von den Stützstellen abgedeckten Intervalls kaum ein Unterschied zwischen dem Cosinus und dem Interpolationspolynom vom Grade 3 sichtbar ist. Außerhalb steigt dagegen der Fehler schnell dramatisch an.

Beispiel 2.12 (Runge–Beispiel) *Runge and König [1925]*
Leider sind die Verhältnisse nicht immer so gut. Von Carl Runge stammt das Beispiel der Funktion

$$f(x) = \frac{1}{1 + 25x^2}$$

auf dem Intervall $[-1, 1]$: Für steigende Zahl der Stützstellen nimmt der maximale Fehler schnell zu.



Interpolation von $f(x) = 1/(1 + 25x^2)$ auf dem Einheitsintervall mit 30 äquidistanten Stützstellen. Die Approximation in der Nähe der o ist gut, am Rand beliebig schlecht.

Beispiel 2.13 Interpolation des Cosinus mit Auswertungsfehlern
Wir betrachten noch einmal die Interpolation des Cosinus. Diesmal nehmen wir aber an, dass wir den Cosinus nicht exakt berechnen, sondern dabei einen kleinen Fehler machen (wie es bei Messungen notwendig der Fall ist).

Bei niedrigen Polynomgraden hat dieser Fehler nur geringe Auswirkungen, bei höheren Polynomgraden bekommen wir dagegen Oszillationen wie im Beispiel von Runge.

Also: (Kleine) Störungen des Cosinus führen dazu, dass auch hier hohe Polynomgrade zu keiner vernünftigen Interpolation führen. Dies lässt bereits vermuten, dass die guten Eigenschaften des Cosinus bei der Polynominterpolation eine Ausnahme bilden.

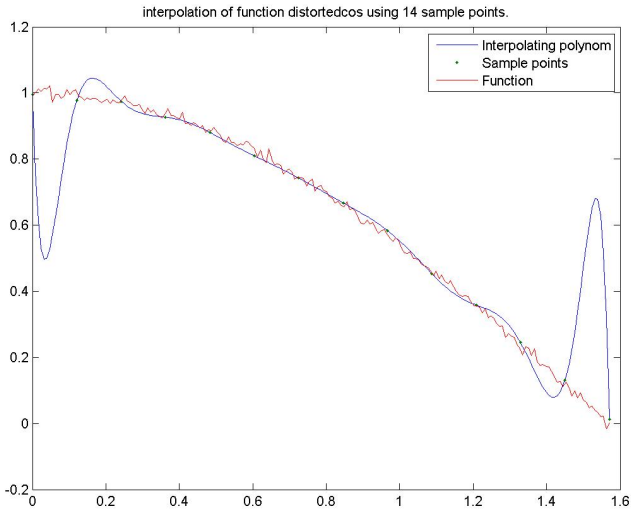


Abbildung 2.5: Interpolation des Cosinus mit kleinem Messfehler, mit äquidistanten Stützstellen

Klick für Bild distortedcos
Klick für Matlab Figure distortedcos

```
function p = polapprox( f,a,b,n,x )
%POLAPPROX Approximate a function f by polynomial
%interpolation in (n+1) equidistant sample points
if (nargin<5)
x=(0:n)/n*(b-a)+a;
end
```

Listing 2.6: Polynomapproximation (interpolation/polapprox.m)

Klicken für den Quellcode von interpolation/polapprox.m

```
function rungebeispiel(N)
%RUNGEBEISPIEL
if (nargin<1)
N=10;
end
close all;
```

Listing 2.7: Cosinus und Runge–Beispiel (interpolation/rungebeispiel.m)

Klicken für den Quellcode von interpolation/rungebeispiel.m

Wir stellen fest:

Bei steigender Zahl der Stützstellen sinkt der Betrag des Interpolationsfehlers nicht notwendig monoton. Er konvergiert nicht einmal notwendig gegen Null, wie wir es heuristisch erwarten würden.

Dieses Phänomen sorgte zu Beginn des 20. Jahrhunderts für großes Aufsehen. Insbesondere, weil es nicht eine in der Praxis nie auftauchende, obskure Funktion benutzt, sondern eine völlig unscheinbare, extrem glatte gebrochenrationale Funktion.

Polynominterpolation ist für uns das erste Beispiel eines Algorithmus, der scheinbar sinnvoll ist, aber tatsächlich viel größere Fehler liefert als er müsste. Es gibt nämlich einen viel einfacheren, konkurrierenden Algorithmus, bei dem offensichtlich die Interpolationsfunktion mit steigender Zahl der Nullstellen gegen die (stetige) zu interpolierende Funktion konvergiert: Dies ist die lineare Interpolation benachbarter Stützstellen, die wir mit ihren verwandten Algorithmen im Abschnitt über Splines noch betrachten werden.

Es gibt auch noch ein zweites Problem. Scheinbar liefert der Interpolationsalgorithmus für den Cosinus gute Ergebnisse (dies werden wir durch den nächsten Satz auch erklären). Tatsächlich ist er aber auch hier völlig unbrauchbar: Macht man kleine Fehler bei der Berechnung des Cosinus, wirken diese sich extrem auf das Ergebnis aus, und wir erhalten wieder hochoszillierende Funktionen. Auch hier würde die einfache lineare Interpolation wieder Abhilfe schaffen.

Algorithmen mit diesen Eigenschaften (das Ergebnis ist erheblich schlechter, als es sein müsste, ist typischerweise oszillierend, liefert betragsmäßig große, unsinnige Ergebnisse, reagiert sehr sensitiv auf Eingabefehler) nennen wir instabil. Eine genauere Definition werden wir im Abschnitt über Mehrschrittverfahren bei gewöhnlichen Differentialgleichungen kennenlernen.

Der folgende Satz schätzt den maximal zu erwartenden Interpolationsfehler ab.

Satz 2.14 (Abschätzung des Interpolationsfehlers)
Sei $f \in \mathbb{C}^{N+1}([a,b])$, $f : [a,b] \mapsto \mathbb{R}$. Seien x_i paarweise verschieden in $[a,b]$, $i = 0 \dots N$, und sei $p \in \mathcal{P}_N$ das zugehörige Interpolationspolynom mit $p(x_i) = f(x_i)$. Dann gilt:

$$\forall \overline{x} \in [a,b] \exists \widetilde{x} \in [a,b] \text{ mit } f(\overline{x}) - p(\overline{x}) = w(\overline{x}) \frac{f^{(N+1)}(\widetilde{x})}{(N+1)!},\ w(x) := \prod_{j=0}^{N} (x - x_j).$$

Insbesondere gilt

$$|f(x) - p(x)| \leq |w(x)| \frac{||f^{(N+1)}||_\infty}{(N+1)!}$$

und

$$||f - p||_\infty \leq ||w||_\infty \frac{||f^{(N+1)}||_\infty}{(N+1)!}$$

mit der Maximumnorm $||f||_\infty = \max_{x \in [a,b]} |f(x)|$.

Beweis:

1. Sei $\overline{x} = x_i$ für ein i. Dann ist $f(\overline{x}) = p(\overline{x})$, $w(\overline{x}) = 0 \Rightarrow$ Behauptung.
2. Sei $\overline{x} \neq x_i$ für alle $i = 0 \dots N$, also $w(\overline{x}) \neq 0$. Wir betrachten den Interpolationsfehler. Dieser hat bereits $(N+1)$ Nullstellen an den interpolierenden Punkten. Wir modifizieren die Fehlerfunktion nun leicht so, dass sie noch eine zusätzliche Nullstelle bei $\overline{x}$ hat. Sei also

   $$F(x) := (f(x) - p(x)) - Kw(x),\ K = \frac{f(\overline{x}) - p(\overline{x})}{w(\overline{x})}.$$

   F hat mindestens die $(N+2)$ verschiedenen Nullstellen $\overline{x}$ und x_i, $i = 0 \dots N$. Nach dem Satz von Rolle hat F' mindestens $(N+1)$ verschiedene Nullstellen, F'' mindestens N Nullstellen und $F^{(N+1)}$ hat mindestens eine Nullstelle $\widetilde{x}$ im Intervall $[a, b]$. $p \in \mathcal{P}_N$, also verschwindet $p^{(N+1)}$. Der Höchstkoeffizient von $x^{(N+1)}$ in w ist 1, also gilt

   $$p^{(N+1)}(x) = (N+1)!$$

   und damit insgesamt

   $$0 = F^{(N+1)}(\widetilde{x}) = f^{(N+1)}(\widetilde{x}) - K(N+1)! \Rightarrow K = \frac{f^{(N+1)}(\widetilde{x})}{(N+1)!}$$

   und damit

   $$0 = F(\overline{x}) = f(\overline{x}) - p(\overline{x}) - \frac{f^{(N+1)}(\widetilde{x})}{(N+1)!} w(\overline{x}).$$

□

Dieser Satz erklärt unsere einführenden Beispiele komplett. Alle Ableitungen des $\cos$ sind beschränkt, deshalb fällt hier der maximale Approximationsfehler schnell. Außerhalb des Intervalls $[a, b]$ steigt die Funktion w schnell an, deshalb bekommen

wir dort keine guten Approximationen. Die Ableitungen des Runge–Beispiels wachsen wie 50^n auch für kleine n rasant an, deshalb bekommen wir hier keine guten Approximationen. Wenn wir zufällige Auswertungsfehler berücksichtigen, so sind die dadurch entstehenden Funktionen sicherlich nicht mehr differenzierbar, und wir können den Satz nicht einmal mehr anwenden.

Bemerkung: Der Interpolationsfehler kann also durch eine Schranke abgeschätzt werden, die von der $(N+1)$. Ableitung von f und der Verteilung der Stützstellen (durch die Funktion $w(x)$) abhängt. Da wir f nicht beeinflussen können, sollten wir die x_i so wählen, dass $||w||_\infty$ möglichst klein wird.

Bemerkung: Die Voraussetzung, dass f glatt ist (also die $(N+1)$. Ableitung existiert), ist notwendig. Falls f nur k–mal differenzierbar ist, bringt eine Erhöhung des Polynomgrads jenseits von $k-1$ nichts mehr (siehe Übungen).

Beispiel 2.15

1. *Äquidistante Stützstellenwahl.*

   Ohne Einschränkung sei $[a,b] = [0,1]$ *und* $h = 1/N$ *der Abstand zwischen zwei Punkten. Sei* $f \in C^\infty([0,1]$. *Dann ist für* $j = \lfloor \frac{x}{h} \rfloor$ *bzw.* $j = \lceil \frac{x}{h} \rceil$

   $$\frac{|w(x)|}{(N+1)!} = h^{N+1} \frac{\prod_{i=0}^{N} |x/h - i|}{(N+1)!} < h^{N+1} \frac{j!(N+1-j)!}{(N+1)!} \leq \frac{1}{N^{N+1}}$$

   (Beweis durch Ausmultiplizieren des Zählers).

   Damit man für große N **keine** *Konvergenz des Interpolationsfehlers gegen* 0 *bekommt, muss die Supremumsnorm der* n*. Ableitung also schneller wachsen als* N^N.

   w *nimmt sein Betragsmaximum zwischen* x_0 *und* x_1 *bzw. zwischen* x_{N-1} *und* x_N *an, die Abschätzung wird also am Rand besonders schlecht.*

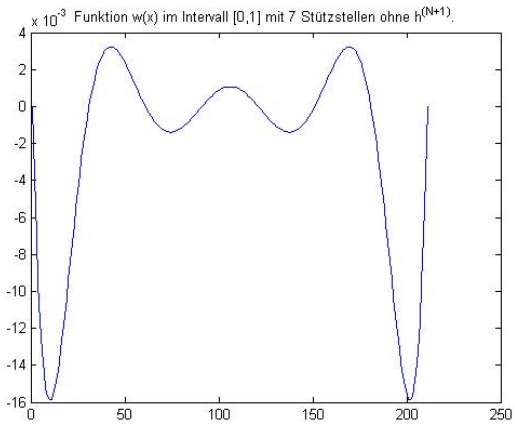


Abbildung 2.6: $w(x)$ ohne den Faktor h^{N+1} für $N = 7$

Klick für Bild approtest

Die inhaltliche Erklärung dafür, dass an den Rändern die Approximation schlecht ist: Bei äquidistanter Verteilung sind in der Nähe der Mitte des Intervalls doppelt so viele Stützpunkte wie am Rand. Für eine gleichmäßige Approximation sollten wir also die Stützstellen in der Nähe des Randes verdichten.

Diese Betrachtung gilt nicht bei der Interpolation periodischer Funktionen über die gesamte Periode: Hier sind alle Punkte des Intervalls gleichberechtigt, und tatsächlich zeigt eine einfache Symmetriebetrachtung, dass in diesem Fall die äquidistante Verteilung optimal ist.

Trefethen zeigt in einem Artikel, in dem er die Stützstellen als elektrisch geladene Teilchen interpretiert, die Optimalität der Dichteverteilung $1/(1-x^2)$ (Trefethen [2000]).

Unabhängig von all dem bedeutet ein hoher Polynomgrad ja auch noch einen hohen Aufwand bei der Auswertung des Polynoms. Grundsätzlich gilt die Regel: Ein hoher Polynomgrad bei der Interpolation (jenseits von ca. 8) ist im allgemeinen nicht zu empfehlen.

```
function out = approtest( N )
%APPROTEST
if (nargin<1)
    N=7;
end
close all;
```

Listing 2.8: Approximationsfehler (interpolation/approtest.m)

Klicken für den Quellcode von interpolation/approtest.m

2. *Wir betrachten wieder das Intervall $[a, b]$, teilen es aber in M Teile auf. In jedem Teilintervall führen wir eine Polynominterpolation an N äquidistanten Stützstellen durch, N fest. Um den Wert der Approximation an einer Stelle z zu berechnen, stellen wir zunächst fest, in welchem Teilintervall z liegt, und werten dort das Interpolationspolynom in diesem Intervall aus. In diesem Fall konvergiert für $M \mapsto \infty$ die Approximation gegen die Originalfunktion punktweise, und es gilt*

$$||f - p||_\infty \leq C \frac{||f^{(N+1)}||_\infty}{M^N}.$$

Für $N = 1$ wird in jedem Intervall durch eine Zahl approximiert, für $N = 2$ erhält man den Polygonzug.

Vorteil: Wir erhalten garantierte Konvergenz, der Aufwand zur Auswertung bleibt konstant.

Nachteil: Die entstehende Interpolationsfunktion ist zwar stetig (für $N > 1$), aber nicht mehr differenzierbar.

Diese Idee (Aufteilung der Approximationsaufgabe auf kleine Intervalle) wird die zentrale Idee für die Spline-Interpolation sein.

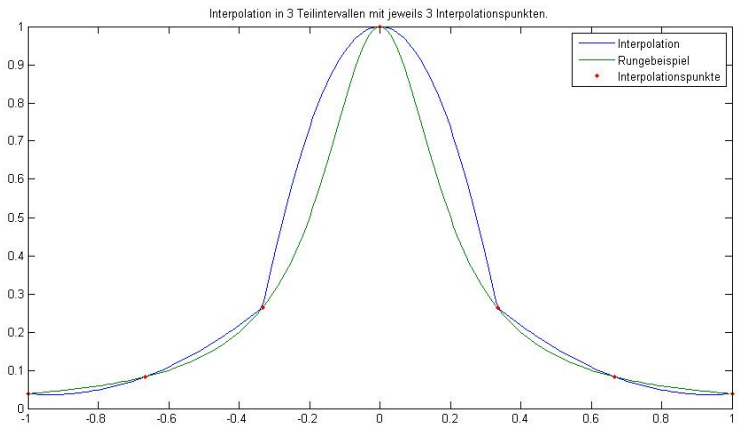


Abbildung 2.7: Interpolation in Teilintervallen

Klick für Bild partinter

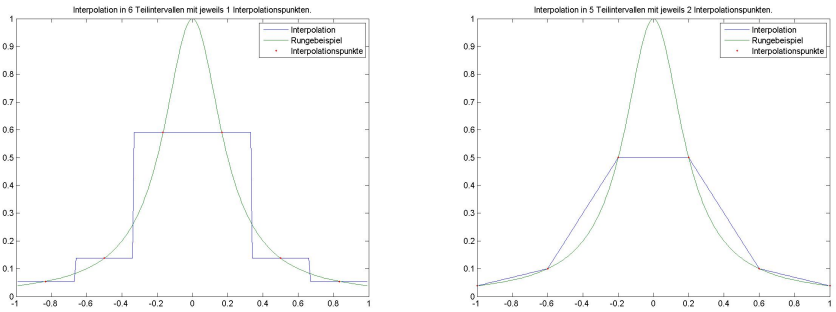


Abbildung 2.8: Auswertung und Polygonzug–Approximation
Klick für Bild partintero
Klick für Bild partinter1

```
function  partinter (M,N)
%PARTINTER
    function y=interpval(x)
        k=floor((x-a)/(b-a)*M)+1;
        y=polyval(Q(k,:),x);
    end
```

Listing 2.9: Interpolation in Teilintervallen (interpolation/partinter.m)

Klicken für den Quellcode von interpolation/partinter.m

Wir halten noch als ein wichtiges Ergebnis dieses Kapitels fest:

Korollar 2.16 *Die Polynominterpolation konvergiert bei steigendem Polynomgrad und äquidistanter Stützstellenwahl nicht notwendig gegen die zu interpolierende Funktion.*

Vorlesungsnotiz: 12.4.2013

2.3 Optimale Wahl der Stützstellen, Tschebyscheff–Interpolation

Im Licht von Satz 2.14 stellt sich die Frage: Falls wir frei sind in der Wahl der Stützstellen, welche Wahl liefert die beste Fehlerabschätzung, also den kleinsten Wert für $||w||_\infty$? Hierzu bestimmen wir das in der Maximumnorm kleinste Polynom p in $\mathcal{P}_{n+1}$ mit Höchstkoeffizient 1.

Definition 2.17 *(***Tschebyscheff–Polynome)**

$$T_n : [-1,1] \mapsto \mathbb{R},\ T_n(x) := \cos(n \arccos x),\ n \in \mathbb{N}$$

heißt **Tschebyscheff–Polynom** *der Ordnung* n*.*

Satz 2.18 Eigenschaften der Tschebyscheff–Polynome
Für die Tschebyscheff–Polynome T_n *gilt:*

1. $T_n \in \mathcal{P}_n$*. Für* $n > 0$ *hat* $T_n(x)/2^{n-1}$ *den Höchstkoeffizienten 1.*
2. *Die* T_n *bilden ein Orthogonalsystem im Vektorraum der stetigen Funktionen auf dem Intervall* $[-1,1]$ *bezüglich des Skalarprodukts*

$$(p,q) = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} p(x)q(x)dx.$$

3. *Die Nullstellen von* T_{n+1} *sind*

$$x_k^n = \cos\left(\frac{2k+1}{2(n+1)}\pi\right), k = 0 \ldots n.$$

4. *Es gilt*

$$||\frac{1}{2^{n-1}} T_n(x)||_\infty = \frac{1}{2^{n-1}} \leq ||p||_\infty$$

   für alle $p \in \mathcal{P}_n$ *mit Höchstkoeffizient 1.*

5. *Wählt man für eine Polynominterpolation vom Grad* n *die Stützstellen* x_k^n*,* $k = 0 \ldots n$*, so ist*

$$w(x) = \prod_{k=0}^{n} (x - x_k^n) = \frac{1}{2^n} T_{n+1}(x).$$

Beweis: Siehe Numerische Lineare Algebra .
Kurzer Beweis zu 4.: Sei

$$q(x) = T_n(x)/2^{n-1}.$$

Angenommen, $p \in \mathcal{P}_n$ mit Höchstkoeffizient 1 und

$$||p||_\infty < 1/2^{n-1} = ||q||_\infty.$$

Dann ist

$$r := q - p \in \mathcal{P}_{n-1}.$$

q nimmt sein Betragsmaximum mit wechselnden Vorzeichen an an den Stellen

$$z_k = \cos(k\pi/n),\ k = 0 \ldots n.$$

Wegen

$$|p(z_k)| \leq ||p||_\infty < ||q||_\infty = |q(z_k)|$$

gilt

$$\operatorname{sgn}(r(z_k)) = \operatorname{sgn}(q(z_k) - p(z_k)) = \operatorname{sgn}(q(z_k)).$$

r wechselt also ebenfalls mindestens n-Mal sein Vorzeichen, hat also n Nullstellen, also gilt $r = 0$ und damit $p = q$ im Widerspruch zur Annahme. □

Die Polynominterpolation, bei der wir die Stützstellen $x_0 \dots x_N$ als Nullstellen des Tschebyscheff–Polynoms T_{N+1} wählen, nennen wir **Tschebyscheff–Interpolation**. Wir erhalten für die Tschebyscheff–Interpolation nach 2.18 und 2.14 die Abschätzung

$$||f - p||_\infty \leq \frac{||f^{(n+1)}||_\infty}{2^n(n+1)!}.$$

Bemerkung: Durch die Abbildung

$$x \mapsto a + (x+1)\frac{b-a}{2}$$

wird das Intervall $[-1, 1]$ auf $[a, b]$ abgebildet. Für ein allgemeines Intervall $[a, b]$ lauten die Tschebyscheff–Stützstellen also

$$\widetilde{x_k^n} = a + (x_k^n + 1)\frac{b-a}{2}.$$

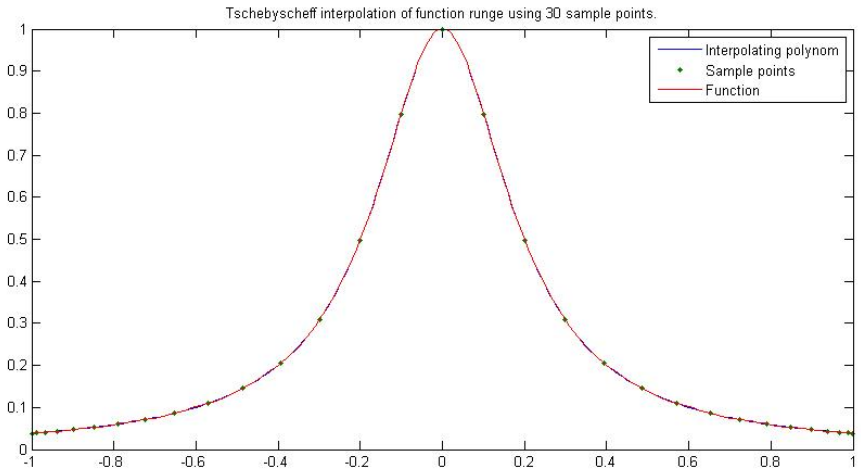


Abbildung 2.9: Tschebyscheff–Interpolation für das Runge–Beispiel

Klick für Bild tschebyscheff

```
function tscheb( N )
%TSCHEB
a=-1;
b=1;
f=@runge;
if (nargin<1)
```

Listing 2.10: Tschebyscheff–Interpolation (interpolation/tscheb.m)

Klicken für den Quellcode von interpolation/tscheb.m

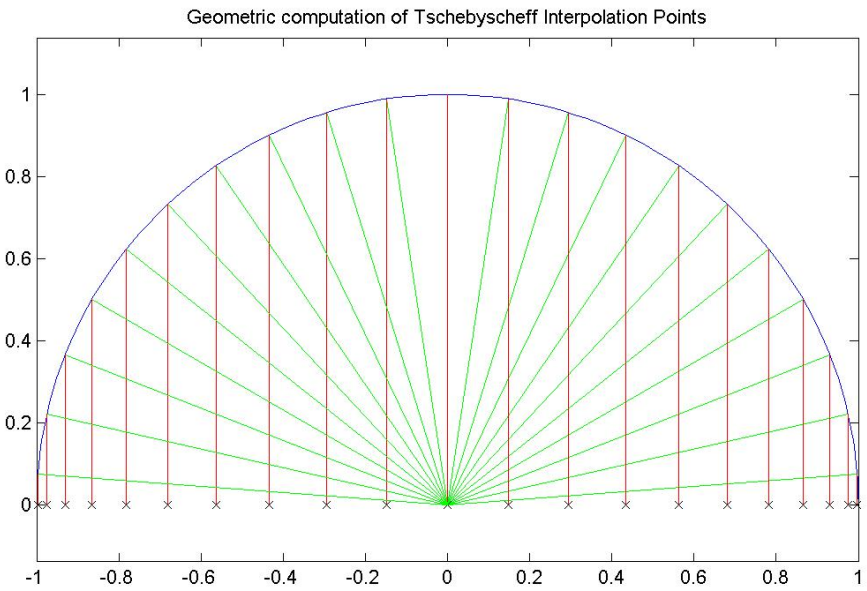


Abbildung 2.10: Geometrische Interpretation der Tschebyscheff–Interpolationspunkte als Abszissen der $n.$ komplexen Einheitswurzeln

Klick für Bild tschebgeom
Klick für Matlab Figure tschebgeom

```
function [ output_args ] = drawtscheb( input_args )
%DRAWTSCHEB
```

```
x=0:0.01:1;
hold off;
plot(cos(x*pi),sin(x*pi));
hold on;
```

Listing 2.11: Tschebyscheff: Geometrische Interpretation (integration/drawtscheb.m)

Klicken für den Quellcode von integration/drawtscheb.m

2.4 Hermite–Interpolation

Ein weiterer Spezialfall der Polynominterpolation ist die Hermite–Interpolation. Hier werden nicht nur die Werte der Funktion, sondern auch ihre Ableitungen spezifiziert. Es gilt der Satz:

Satz 2.19 *Gegeben seien die Stützstellen x_k, $k = 0 \dots N$, und die Stützwerte y_k^i, $k = 0 \dots N$, $i = 0 \dots N_k$. Weiter sei $M = \sum_{k=0}^{N}(N_k + 1) - 1$. Dann gibt es genau ein Polynom $p \in \mathcal{P}_M$ mit*

$$p^{(i)}(x_k) = y_k^i,\ k = 0 \dots N,\ i = 0 \dots N_k.$$

Beweis: Übungen. □

Definition 2.20 *Interpolationsaufgaben, bei denen an einigen Stützstellen zusätzlich zum Wert der Funktion auch der Wert von Ableitungen vorgeschrieben ist, nennen wir* **Hermite–Interpolationsaufgaben**.

Bemerkung: Für die Hermite–Interpolation gilt die Fehlerabschätzung 2.14 mit $w(x) = \prod_k (x - x_k)^{N_k}$, der Beweis verläuft wörtlich wie der Beweis zu 2.14.

Beispiel 2.21 *Sei $N = 2$, $x_0 = -1$, $x_1 = 0$, $x_2 = 1$, $y_0^0 = 1$, $y_0^1 = 0$, $y_1^0 = 0$, $y_2^0 = 0$, $y_2^1 = 0$. Wir suchen also ein Interpolationspolynom durch $(-1, 1)$, $(0, 0)$ und $(1, 1)$, dessen Ableitung an den Rändern verschwindet. Die Koeffizienten lassen sich durch eine entsprechend angepasste Vandermonde–Matrix berechnen, das resultierende Polynom ist*

$$p(x) = x^2(2 - x^2).$$

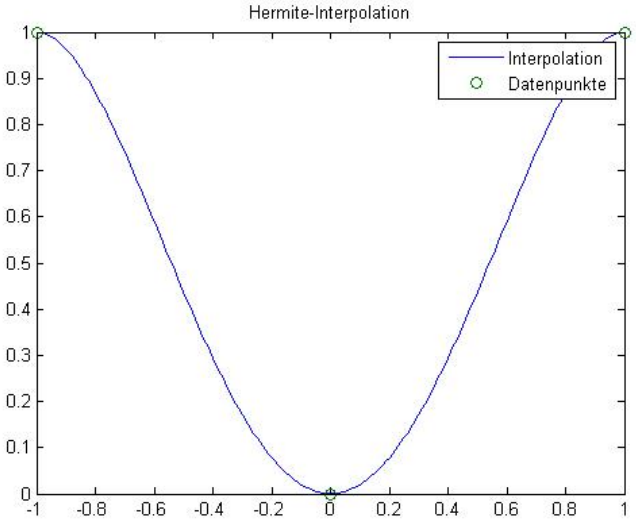


Abbildung 2.11: Beispiel zur Hermite–Interpolation

Klick für Bild hermite

```
function hermitebeispiel(x,N,y)
%HERMITEBEISPIEL
if (nargin<1)
    x=[-1 0 1];
    N=[2 1 2];
    y={[1 0] [0] [1 0]};
```

Listing 2.12: Programm zur Hermite–Interpolation (interpolation/hermitebeispiel.m)

Klicken für den Quellcode von interpolation/hermitebeispiel.m

2.5 Richardson–Extrapolation

Es sei der Grenzwert a_0 einer Funktion f für $h \mapsto 0$ zu bestimmen, die an der Stelle 0 nicht direkt auswertbar ist. Zur Verfügung stehen Auswertungen der Form $y_k = f(h_k)$, $k = 0 \ldots N$. Es sei bekannt, dass f mindestens eine Taylorreihe bis zum Grad N in h^p besitzt, also

$$f(h) = a_0 + \sum_{k=1}^{N} a_j h^{pk} + C(h)h^{p(N+1)} \tag{2.1}$$

mit (unbekannten) Koeffizienten a_j und einer (unbekannten) beschränkten Funktion $C(h)$ mit $|C(h)| \leq C$, aber bekanntem p (typischerweise ist $p = 1$ oder $p = 2$). Dies ist sicher richtig, falls f $(N+1)$-mal stetig differenzierbar ist auf einem Intervall um die 0. Wie erreicht man eine möglichst gute Approximation an den Grenzwert?

Wir bemerken zunächst, dass für kleine h der Restterm hinter der Summe keine Rolle mehr spielt, denn er geht auf der rechten Seite am schnellsten gegen Null. Vernachlässigen wir ihn für den Augenblick, so ist $p(h) = f(h^{1/p})\,(h > 0) \in \mathcal{P}_N$. p ist als Polynominterpolation der Punkte $(h_k^p, f(h_k))$ eindeutig bestimmt. Den gesuchten Wert $f(0)$ approximieren wir durch $p(0)$. Aufgrund unserer Herleitung erwarten wir dabei den Fehler $Ch_{\max}^{p(N+1)}$.

Kurz: Wir nehmen einige Auswertepunkte, legen ein Polynom hindurch, und werten das Polynom an der gesuchten, nicht berechenbaren Stelle aus. Dazu nutzen wir natürlich die Formel und das Schema von Neville 2.6.

Beispiel 2.22 (Richardson–Extrapolation)
Gesucht sei der Grenzwert 1 *der Funktion* $\operatorname{sinc}(x) = sinx/x$ *für* $x \mapsto 0$. *Der* sinc *ist gerade, hat also eine Taylorreihenentwicklung in* x^2, *also gilt* $p = 2$. *Wir interpolieren also an den Stellen* $(h_k^2, f(h_k)$ *durch Polynome. Zur Verfügung stehen Auswertungen von* $\operatorname{sinc}(2^{-k})$, $k = 0 \ldots N$. *Wir werten aus für* $x = 0$ *mit dem Schema von Neville 2.6. Angegeben ist jeweils der Fehler, also die Differenz des berechneten Werts zum korrekten Ergebnis 1:*

$$
\begin{array}{llllll}
-1.5853e-001 & -2.0222e-003 & -3.0442e-006 & -6.6472e-010 & -2.3759e-014 & -1.1102e-016 \\
-4.1149e-002 & -1.2924e-004 & -4.8220e-008 & -2.6202e-012 & -1.1102e-016 & \\
-1.0384e-002 & -8.1229e-006 & -7.5602e-010 & -1.0325e-014 & & \\
-2.6021e-003 & -5.0839e-007 & -1.1823e-011 & & & \\
-6.5091e-004 & -3.1785e-008 & & & & \\
-1.6275e-004 & & & & &
\end{array}
$$

```
function [ output_args ] = richardson1( input_args )
%RICHARDSON1
f=@sinc;
N=5;

format short e;
```

Listing 2.13: Richardson–Extrapolation der Sinc–Funktion (interpolation/richardson1.m)

Klicken für den Quellcode von interpolation/richardson1.m

Aus Folgegliedern mit der Genauigkeit 10^{-4} wird hier also durch die Richardson–Extrapolation ein Grenzwert mit der Genauigkeit 10^{-16} berechnet. Andere Beispiele sind etwa Differenzenquotienten (Grenzwert von $(f(x+h) - f(x))/h$ für $h \mapsto 0$)

oder die näherungsweise Berechnung von Integralen (siehe **??**). Zum Verständnis dieses Verhaltens führen wir zunächst das Landau–Symbol $O()$ ein.

Definition 2.23 (Landau–Symbol)
Seien $f, g : X \mapsto \mathbb{R}$, $X = \mathbb{R}$ oder $\mathbb{C}$. f heißt von der Ordnung g ($f = O(g)$) genau dann wenn:

1. $f(N) = O(g(N))$ *für große N, falls es Konstanten N_0 und C gibt mit*

$$|f(N)| \leq C|g(N)| \, \forall N > N_0.$$

2. $f(h) = O(g(h))$ *für kleine h, falls es Konstanten h_0 und C gibt mit*

$$|f(h)| \leq C|g(h)| \, \forall h < h_0.$$

Beispiel 2.24 *(zum Landau–Symbol)*

1. $N^2 = O(N^3)$, *allgemein* $N^a = O(N^b)$ *für* $a < b$.
2. $h^3 = O(h^2)$, *allgemein* $h^a = O(h^b)$ *für* $a > b$.
3. $\sin(x) = O(1)$ *für alle* $x \in \mathbb{R}$.

Die Ordnung schätzt ab, wie sich ein Term für sehr große N oder sehr kleine h verhält. Für h gilt insbesondere: $f(h) = a_0 + O(h^n)$ konvergiert umso schneller gegen a_0, je größer n ist.

Für die Richardson–Extrapolation, also die erste Zeile im Neville–Schema, gilt: Die erste Spalte hat eine Genauigkeit von $O(h^p)$, die zweite von $O(h^{2p})$ usw.

Satz 2.25 (Richardson–Extrapolation)
$f(x)$ besitze an der Stelle 0 eine Taylorreihe bis zum Grad N in x^p. Sei $g(x) = f(x^{1/p})$, $x \geq 0$. Berechne das Neville–Schema für die Stützstellen $x_k = (a^k h)^p$, $0 < a < 1$, $h > 0$, und Stützwerte $y_k = g(x_k) = f(a^k h)$. Dann gilt für die Einträge $p_{i\ldots i+k-1}(0)$ in der k. Spalte des Tableaus

$$p_{i\ldots i+k-1}(0) = a_0 + \sum_{j=k}^{N} a_{k,j}((a^i h)^p)^j$$

mit Zahlen $a_{k,j}$. Insbesondere gilt für die Zahlen in der ersten Zeile des Tableaus

$$p_{0\ldots k} = a_0 + O(h^{pk)}).$$

Beweis: Es sei zunächst $p = 1$, also $f = g$. In der ersten Spalte des Tableaus stehen die Stützwerte

$$p_i(0) = y_i = f(a^i h) = a_0 + \sum_{j=0}^{N} a_j (a^i h)^j + O(h^{N+1}),$$

das ist die Aussage für $k = 1$.
Sei der Satz nun richtig für die Spalte k. Wir zeigen, dass in der ersten Zeile in Spalte $k + 1$ die Exponenten bis h^k herausfallen. Der Eintrag wird nach der Formel von Neville 2.6 berechnet durch

$$\begin{aligned}
&p_{0\ldots k}(0) \\
&= \frac{1}{x_0 - x_k}((0 - x_k)p_{0\ldots k-1}(0) + (x_0 - 0)p_{1\ldots k}(0)) \\
&= \frac{1}{h(1 - a^k)}((-ha^k)(a_0 + a_{k,k}h^k + O(h^{k+1}) + (h)(a_0 + a_{k,k}(ah)^k + O(h^{k+1})))) \\
&= a_0 + O(h^{k+1}).
\end{aligned}$$

Die Rechnung kann so für alle Zeilen durchgeführt werden, das gibt die Aussage des Satzes für $p = 1$.
Für $p \neq 1$ wird die Rechnung für g statt f durchgeführt. Nur für $p \neq 1$ benötigen wir die Voraussetzung $h > 0$. □
Die Richardson–Interpolation berechnet also eine Linearkombination der bekannten Werte, so dass sich die Ordnung der Konvergenz erhöht. Wichtig: Die Koeffizienten der Taylorentwicklung a_k müssen nicht bekannt sein (ansonsten könnte man den Fehler auch gleich direkt abziehen), geschickte Linearkombination lässt die Fehler herausfallen.

Man kann sich sehr schnell klarmachen, wie die Richardson–Extrapolation funktioniert. Nehmen wir an, dass uns die Auswertungen $y_0 = f(h)$ und $y_1 = f(h/2)$ zur Verfügung stehen, um $f(0)$ auszurechnen, und dass f eine Taylorentwicklung in h^2 hat. Es gilt also:

$$\begin{aligned}
y_0 &= f(0) + a_0 h^2 && +O(h^4) \\
y_1 &= f(0) + a_0 (h/2)^2 && +O(h^4)
\end{aligned}$$

Wir bilden nun eine günstige Linearkombination von y_0 und y_1, so dass der störende Term in h^2 herausfällt. Also

$$\tilde{y} = 4y_1 - y_0 = 3f(0) + O(h^4)$$

und damit

$$\frac{1}{3}\tilde{y} = f(0) + O(h^4)$$

und wir haben eine Formel zur Berechnung von $f(0)$ mit der Ordnung h^4 gefunden, dies ist gerade die, die auch Richardson liefert.

Wir wollen Richardson an zwei ganz kleinen Beispielen testen. Es soll die Ableitung der Funktion f mit Taylorentwicklung an der Stelle 0 ausgerechnet werden. Hierzu betrachten wir die Funktion

$$F(h) = \begin{cases} \frac{f(h)-f(0)}{h} & h \neq 0 \\ f'(0) & h = 0 \end{cases}.$$

$f(h)$ besitze um 0 eine Taylorentwicklung in h mit Koeffizienten a_k. Dann besitzt auch $F(h)$ um 0 eine Taylorentwicklung in h. Wir suchen $F(0)$, das nicht direkt ausgerechnet werden kann. Statt dessen werten wir F an einigen Stellen aus und versuchen, darüber $F(0)$ mit Richardson zu approximieren.

Wir wählen zunächst $x_0 = -h$, $x_1 = h$. Es stehen also die Werte $y_0(h) = F(-h)$ und $y_1(h) = F(h)$ zur Verfügung. Neville liefert die Auswertung der interpolierenden Geraden an der Stelle 0, d.h.

$$\tilde{y}(h) = \frac{1}{2}(y_1(h) + y_0(h)) = \frac{f(h) - f(-h)}{2h}.$$

$\tilde{y}(h)$ hat die Taylorentwicklung

$$\tilde{y}(h) = f'(0) + \sum_{k=1}^{\infty} a_{2k+1} h^{2k}$$

und damit einen Fehler von $O(h^2)$, während $y_0(h)$ und $y_1(h)$ für sich jeweils einen Fehler in $O(h)$ aufweisen.

Jetzt wählen wir $x_0 = h/2$, $x_1 = h$. Das interpolierende Polynom ist in diesem Fall mit Lagrange

$$p_h(x) = y_0 \frac{x - h}{h/2 - h} + y_1 \frac{x - h/2}{h - h/2}$$

und wir erhalten für $x = 0$

$$\tilde{y}(h) = 2y_0 - y_1 = \frac{1}{h}\left(4f\left(\frac{h}{2}\right) - f(h) - 3f(0)\right) = f'(0) + O(h^2).$$

Bemerkung: Ein Spezialfall der Richardson–Interpolation ist die Romberg–Integration 3.3.

2.6 Rationale Interpolation

Natürlich können wir auch mit gebrochenrationalen Funktionen interpolieren.

Definition 2.26 (Rationale Interpolation)
Seien $x_k \in \mathbb{C}$ paarweise verschiedene Stützstellen, y_k Stützwerte, $k = 0 \dots N$. Es sei $P \geq 0$, $Q \geq 0$. Die Aufgabe

$$\textit{Bestimme } p \in \mathcal{P}_P,\ q \in \mathcal{P}_Q \textit{ mit } \frac{p(x_k)}{q(x_k)} = y_k,\ k = 0 \dots N$$

heißt rationale Interpolationsaufgabe.

Erste Frage ist natürlich, wie N zu wählen ist. p hat $P + 1$ Koeffizienten, q hat $Q + 1$ Koeffizienten, also insgesamt $P + Q + 2$ Freiheitsgrade, man könnte also $N = P + Q + 1$ wählen, um $P + Q + 2$ Bedingungen zu erfüllen.

Offensichtlich kann man aber mit einer Zahl erweitern, ohne dass sich der Wert des Bruchs ändert, das reduziert die Zahl der Freiheitsgrade um 1. Wir wählen also $N = P + Q$. Selbst dann kann man aber noch mit Polynomen von höherem Grad in Zähler und Nenner erweitern, ohne den Wert zu ändern. Existenz und Eindeutigkeit sind für die rationale Interpolation mit dieser Wahl also nicht gesichert (siehe Übungen).

Rationale Interpolation hat häufig bessere Interpolationseigenschaften als die normale Polyominterpolation (das Runge–Beispiel etwa kann sie natürlich exakt approximieren!).

Zur **Berechnung** von p und q multiplizieren wir 2.26 mit $q(x_k)$ und erhalten

$$y_k q(x_k) = p(x_k),\ k = 0 \dots N.$$

Häufig normiert man der Einfachheit halber den Koeffizienten von 1 in p auf 1, also $p(0) = 1$. Korrekter wäre hier: Sei k_0 ein Index mit $y_{k_0} \neq 0$. Dann setzen wir $p(x_{k_0}) = 1$ und können hieraus den Koeffizienten der 1 eliminieren.

In jedem Fall erhalten wir ein lineares Gleichungssystem für die restlichen $P + Q + 1 = N + 1$ Koeffizienten mit $N + 1$ Gleichungen, das wir zur Berechnung der Koeffizienten nutzen, sofern die Aufgabe lösbar ist.

Wegen seiner guten Glattheitseigenschaften ist die rationale **Approximation** mit Splines die Grundlage vieler Algorithmen in der Computergrafik (etwa NURBS, nonuniform rational B-Splines).

```
function [a,b] = ratinterp( x,y,n,m )
%RATINTERP rational interpolation
%assumes that po\not=o.
M=n+m+1;
if M~=numel(x)
    'Incorrect Size.'
```

Listing 2.14: Rationale Interpolation (interpolation/ratinterp.m)

Klicken für den Quellcode von interpolation/ratinterp.m

```
function y = rateval( a,b,x )
%RATEVAL
y=polyval(a,x)./polyval(b,x);
end
```

Listing 2.15: Rationale Auswertung (interpolation/rateval.m)

Klicken für den Quellcode von interpolation/rateval.m

```
function [ output_args ] = ratdemo( n,m )
%RATDEMO
if (nargin<1)
    n=2;
end
if (nargin<2)
```

Listing 2.16: Beispiel zur rationalen Interpolation (interpolation/ratdemo.m)

Klicken für den Quellcode von interpolation/ratdemo.m

2.7 Interpolation mit allgemeinen Ansatzfunktionen

Bei der Polynominterpolation versucht man, eine Funktion f durch eine Linearkombination der Monome x^k, $k = 0 \dots N$, zu approximieren. Diese Wahl ist nicht immer optimal. Falls etwa bekannt ist, dass die Funktion f stark (exponentiell) wächst, sollte man f durch eine Linearkombination von stark wachsenden Funktionen approximieren. Entsprechend: Falls f periodisch ist, etwa mit der Periode 2π, sollte es durch eine Linearkombination von periodischen Funktionen approximiert werden, also am einfachsten durch e^{ikx} in $\mathbb{C}$ oder $\sin(kx)$, $\cos(kx)$ in $\mathbb{R}$. Das Interpolationsproblem mit allgemeinen Ansatzfunktionen lautet

Definition 2.27 (Interpolation mit allgemeinen Ansatzfunktionen)
Seien $f_j : I \mapsto \mathbb{C}$, $j = 0 \dots N$, Ansatzfunktionen. Gegeben seien die paarweise verschiedenen Stützstellen x_k und Stützwerte y_k, $k = 0 \dots N$. Finde Koeffizienten a_j so dass

$$y_k = \sum_{j=0}^{N} a_j f_j(x_k),\ k = 0 \dots N.$$

Die Wahl der Ansatzfunktionen bietet also eine Möglichkeit, zusätzliches Wissen (a priori-Wissen) über die zu approximierende Funktion mit einzubringen: Der von den Ansatzfunktionen aufgespannte Raum sollte dieselben Eigenschaften haben wie die Funktion f. Weiß man etwa, dass f unstetig ist im Punkt z, so sollte zumindest eine der Ansatzfunktionen ebenfalls unstetig sein in z.

Das zugehörige Gleichungssystem für die a_j ergibt sich mit der Vandermonde–ähnlichen Matrix

$$V = \begin{pmatrix} f_0(x_0) & \cdots & f_N(x_0) \\ \vdots & \ddots & \vdots \\ f_0(x_N) & \cdots & f_N(x_N) \end{pmatrix} \tag{2.2}$$

und das allgemeine Interpolationsproblem ist eindeutig lösbar, falls V invertierbar ist. Für $f_k(x) = x^k$ erhalten wir die Polynominterpolation zurück.

Wir betrachten als Spezialfälle die Interpolation mit trigonometrischen Funktionen und die Splines.

2.8 Trigonometrische Interpolation: Diskrete Fouriertransformation

2.8.1 Eindimensionale diskrete Fouriertransformation

Als einen Spezialfall der Polynominterpolation und der Interpolation mit allgemeinen Ansatzfunktionen betrachten wir die trigonometrische Interpolation. Bei der Definition der Polynominterpolation hatten wir ausdrücklich Stützstellen und –werte in $\mathbb{C}$ zugelassen. Für die trigonometrische Interpolation wählen wir als Stützstellen x_k die $n.$ Einheitswurzeln, also

$$x_k = e^{2\pi i k/N} = \omega_N^k,\ \omega_N = e^{2\pi i/N},\ (x_k)^N = 1,\ k = 0 \dots N-1.$$

Die x_k liegen äquidistant auf dem Rand des Einheitskreises in der komplexen Ebene (deshalb auch “Interpolation am Kreis”).

Satz 2.28 Inverse der Vandermondematrix für Einheitswurzeln
Seien $(x_k) = \omega_N^k$, $k = 0 \dots N-1$, $W = V(x_0, \dots, x_{N-1})$ *die zugehörige Vandermonde–Matrix. Dann gilt*

$$W\overline{W} = NI \text{ und damit } W^{-1} = \frac{1}{N}\overline{W}.$$

Beweis: Durch Ausrechnen. Der Einfachheit halber wählen wir für W die Indizierung von 0 bis $N-1$.

$$\begin{aligned}
(W\overline{W})_{jk} &= \sum_{l=0}^{N-1} W_{jl}\overline{W}_{lk} \\
&= \sum_{l=0}^{N-1} \omega_N^{jl}\omega_N^{-lk} \\
&= \sum_{l=0}^{N-1} (\omega_N^{j-k})^l \\
&= \begin{cases} N & j = k \\ \frac{(\omega_N^{j-k})^N - 1}{\omega_N^{j-k} - 1} = 0 & j \neq k \end{cases} \\
&= n\delta_{jk}
\end{aligned}$$

□

Bemerkung:

$$W = \left(\omega_N^{jk}\right)_{jk} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_n^{2(N-1)} \\ \vdots & \vdots & & \vdots & \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{pmatrix}.$$

W ist symmetrisch, aber nicht selbstadjungiert.

Mit Hilfe dieser Formel für die Inverse der Vandermonde–Matrix lassen sich die Koeffizienten des Interpolationspolynoms der trigonometrischen Interpolation sofort angeben.

Korollar 2.29 (Interpolation am Kreis)
Sei $x_k = \omega_N^k$, $\omega_N = e^{2\pi i/N}$, $y_k \in \mathbb{C}$, $k = 0 \dots N-1$. *Dann ist das Interpolationspolynom* $p \in P_{N-1}$ *mit* $p(x_k) = y_k$ *gegeben durch*

$$p(x) = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{y}_k x^k,\ \widehat{y}_k = \sum_{j=0}^{N-1} y_j \overline{\omega}_N^{kj} = \sum_{j=0}^{N-1} y_j e^{-2\pi i kj/N},\ k = 0 \dots N-1. \qquad (2.3)$$

Umgekehrt gilt

$$y_j = p(x_j) = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{y}_k x_j^k = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{y}_k \omega_N^{kj} = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{y}_k e^{2\pi i kj/N},\ j = 0 \dots N-1. \tag{2.4}$$

Definition 2.30 (Diskrete Fouriertransformation)
Es sei $y_k \in \mathbb{C}$, $k = 0 \dots N-1$, *und*

$$\widehat{y}_k := \sum_{j=0}^{N-1} y_j e^{-2\pi i kj/N},\ k = 0 \dots N-1.$$

Dann heißt $(\widehat{y}_0, \dots, \widehat{y}_{N-1})$ *diskrete Fouriertransformation von* $(y_0, \dots, y_{N-1})$.

Es sei

$$\widetilde{y}_k := \frac{1}{N}\sum_{j=0}^{N-1} y_j e^{2\pi i kj/N},\ k = 0 \dots N-1.$$

Dann heißt $(\widetilde{y}_0, \dots, \widetilde{y}_{N-1})$ *inverse diskrete Fouriertransformation von* $(y_0, \dots, y_{N-1})$. *Üblicherweise wird die Fouriertransformation eines Vektors* y *mit* $\widehat{y}$ *bezeichnet, die inverse Fouriertransformation mit* $\widetilde{y}$. *Damit gilt*

$$\widetilde{\widehat{y}} = \widehat{\widetilde{y}} = y.$$

Bemerkung: In der Literatur finden sich viele Varianten: In der Ingenieurliteratur sind häufig die Definition von $\widehat{y}$ und $\widetilde{y}$ vertauscht. Den Faktor N kann man natürlich auch der jeweils anderen Transformation zuschlagen, oder als $1/\sqrt{N}$ auf beide Transformationen aufteilen. Im letzten Fall wird W unitär.

Beispiel 2.31
$N = 2$, $\omega_2 = -1$, $\overline{\omega}_2 = -1$:

$$\begin{aligned} \widehat{y}_0 &= y_0 + y_1 \\ \widehat{y}_1 &= y_0 - y_1 \end{aligned}$$

$N = 4$, $\omega_4 = i$, $\overline{\omega}_4 = -i$:

$$\begin{array}{llllll} \widehat{y}_0 & = y_0 & +y_1 & +y_2 & +y_3 \\ \widehat{y}_3 & = y_0 & -iy_1 & -y_2 & +iy_3 \\ \widehat{y}_2 & = y_0 & -y_1 & +y_2 & -y_3 \\ \widehat{y}_1 & = y_0 & +iy_1 & -y_2 & -iy_3 \end{array}$$

Bemerkung: Die naive Berechnung der Fouriertransformierten anhand der Formel benötigt N^2 Additionen und Multiplikationen.

Die diskrete Fouriertransformation wird genutzt zur Interpolation von periodischen Funktionen. Sei etwa

$$f : \mathbb{R} \mapsto \mathbb{C},\ f \in C^{(N)}(\mathbb{R}),$$

periodisch mit der Periode 2π. Seien

$$x_k = 2\pi k/N,\ k = 0 \ldots N-1,$$

gleichverteilte Stützstellen im Intervall $[0, 2\pi]$ (wegen $f(x_0) = f(0) = f(2\pi) = f(x_N)$ bringt die $N.$ Stützstelle keine zusätzliche Information). Als Ansatzfunktionen für die allgemeine Interpolationsaufgabe wählen wir die periodischen Funktionen $f_j(x) = e^{ijx}$. Dann liefert die Fouriertransformation die zugehörigen Entwicklungskoeffizienten der Interpolationsfunktion, bis auf den Faktor $1/N$.

Für praktische Rechnungen ist es ungünstig, dass sogar für reelle y_k die Fouriertransformierte komplexe Werte annimmt. In der Praxis rechnet man daher eher mit der Cosinus– oder Sinus–Transformation, die eng mit der Fouriertransformation zusammenhängen.

Sei also y reell und N z.B. gerade. Dann gilt

$$\begin{aligned}
\widehat{y}_k &= \sum_{j=0}^{N-1} y_j e^{-2\pi ijk/N} \\
&= \sum_{j=0}^{N-1} (y_j \cos(2\pi kj/N) - iy_j \sin(2\pi kj/N)) \\
&= y_0 + (-1)^k y_{N/2} + \sum_{j=1}^{N/2-1} (y_j \cos(2\pi kj/N) + y_{N-j} \cos(2\pi k(N-j)/N)) \\
&\quad - i \sum_{j=1}^{N/2-1} (y_j \sin(2\pi kj/N) - y_{N-j} \sin(2\pi k(N-j)/N)) \\
&= y_0 + (-1)^k y_{N/2} + \sum_{j=1}^{N/2-1} (y_j + y_{N-j}) \cos(2\pi kj/N) \\
&\quad - i \sum_{j=1}^{N/2-1} (y_j - y_{N-j}) \sin(2\pi kj/N)
\end{aligned}$$

(denn $\cos(x_k) = \cos(x_{N-k})$ und $\sin(x_k) = -\sin(x_{N-k})$).

Ist y gerade, also $y_j = y_{N-j}$, so ist in diesem Fall $\widehat{y}_k$ reell. Damit ist

$$\gamma_k := \widehat{y}_k/2 = \frac{y_0 + (-1)^k y_{N/2}}{2} + \sum_{j=1}^{N/2-1} \cos(2\pi kj/N) y_j,\ k = 0 \dots N/2.$$

$(\gamma_0, \dots, \gamma_{N/2})$ heißt dann **Cosinustransformation** von $(y_0, \dots, y_{N/2})$.

Ist y ungerade, also $y_j = -y_{N-j}$ so ist in diesem Fall $\widehat{y}_k$ rein imaginär, Damit ist

$$\sigma_k := \frac{i\widehat{y}_k}{2} = \sum_{j=1}^{N/2-1} \sin(2\pi kj/N) y_j,\ k = 1 \dots N/2 - 1.$$

$(\sigma_1, \dots, \sigma_{N/2-1})$ heißt dann **Sinustransformation** von $(y_1, \dots, y_{N/2-1})$.

Die genauen Definitionen der Cosinustransformation und der Sinustransformation sind noch stärker vom Anwendungsfall abhängig als die Fouriertransformierte, siehe etwa die Definition in Wikipedia mit der DCT I-IV. Die hier vorgestellte entspricht der DCT I.

Vorlesungsnotiz: 18. April 2013

2.8.2 Höherdimensionale Fouriertransformation

Häufig wird die diskrete Fouriertransformation auf Bilder angewandt, d.h. y ist eine (n_1, n_2)–Matrix. Hierzu wird die Multiplikation im Exponenten von 2.3 als Skalarprodukt interpretiert.

Definition 2.32 (Zweidimensionale Fouriertransformation)
Sei $k = (k_1, k_2)$, $j = (j_1, j_2)$, $0 \le k_1, j_1 \le N_1 - 1$, $0 \le k_2, j_2 \le N_2 - 1$, *und* $y \in \mathbb{C}^{N_1 \times N_2}$. *Dann ist die Fouriertransformation* $\widehat{y}$ *von* y *definiert durch*

$$\begin{aligned}\widehat{y}_k &= \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} y_{j_1,j_2} e^{-2\pi i(k_1 j_1/N_1 + k_2 j_2/N_2)} \\ &= \sum_{j_1=0}^{N_1-1} e^{-2\pi i k_1 j_1/N_1} \sum_{j_2=0}^{N_2-1} y_{j_1,j_2} e^{-2\pi i k_2 j_2/N_2}.\end{aligned}$$

Die zweidimensionale Fouriertransformation entspricht einer Fouriertransformation auf den Zeilen von y, gefolgt von einer Fouriertransformation auf den Spalten. Die Formeln für die inverse Fouriertransformation, $\cos$– und $\sin$–Transformationen gelten entsprechend, ebenso wird die FT für höhere Dimensionen erweitert.

2.8.3 FT in der Bild- und Signalverarbeitung: Faltungssatz

Einer der wichtigsten Sätze über die Fouriertransformation ist der Faltungssatz.

Definition 2.33 (Faltung von Vektoren)
Seien $y \in \mathbb{C}^N$, $z \in \mathbb{C}^M$. Dann ist die **diskrete Faltung** $(y\widehat{*}z) \in \mathbb{C}^{N+M-1}$ *von y und z definiert durch*

$$(y\widehat{*}z)_k = \sum_{j=0}^{N-1} y_j z_{k-j} = \sum_{l=k-N+1}^{k} y_{k-l} z_l, k = 0 \ldots N+M-2$$

wobei $z_p := 0$ für $p < 0$.

Seien $y, z \in \mathbb{C}^N$. Dann ist die **symmetrische diskrete Faltung** $(y * z) \in \mathbb{C}^N$ *von y und z definiert durch*

$$(y * z)_k = \sum_{j=0}^{N-1} y_j z_{k-j} = \sum_{l=0}^{N-1} y_l z_{k-l},\ k = 0 \ldots N-1$$

wobei für z der Index modulo N genommen wird, also $z_{-1} = z_{N-1}$ usw.

Die Wirkung der Faltung auf einen Vektor machen wir uns an einem kleinen Beispiel klar. Sei $y \in \mathbb{R}^N$, $z \in \mathbb{R}^2 = (1, -1)$. Dann gilt

$$(y\widehat{*}z)_k = y_k z_0 + y_{k-1} z_1 = y_k - y_{k-1},\ k = 0 \ldots N.$$

Dabei ist $y_{-1} = 0$, $y_N = 0$ für die echte Faltung und $y_{-1} = y_{n-1}$, $y_N = y_0$ für die symmetrische Faltung. Die Faltung bildet also bei festem Faltungsvektor z für jeden Wert im Ergebnis immer dieselbe Linearkombination von Werten im Signalvektor y, nur der Punkt, an dem diese Linearkombination genommen wird, wird jeweils verschoben. Die Koeffizienten sind dabei die Einträge von z.

Für unseren einfachen Vektor z bedeutet das: Bilde die Differenz aus dem aktuellen Wert y_k und dem letzten Wert y_{k-1}.

Die Faltung kann durch Anhängen von Nullen an y und z mit Hilfe der symmetrischen Faltung berechnet werden (Übungen).

Wieder ist die höherdimensionale Faltung entsprechend definiert, indem man die Indizes als Vektoren interpretiert.

Faltungen spielen eine große Rolle in der Signal- und Bildanalyse. Meist wählt man z fest aus einem kleinen Raum (Filter) und faltet dann ein Eingangssignal y mit z. Einige Beispiele:

1. $z = (-1, 0, 1)$: Approximation der Ableitung, verstärkt Kanten im Signal.
2. $z = (1, -2, 1)$: Approximation der 2. Ableitung, verstärkt Krümmung.
3. $z = (1, 2, 1)$: Glättung, Kanten werden geschwächt.
4. $z = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$: Kantenschärfer in 2D

```
function faltung1D
%FALTUNG1D
N=128;
x=(0:N)/N*2*pi;
y=cos(x)+0.2*randn(size(x));
z=[1 2 3 2 1];
```

Listing 2.17: Eindimensionale Faltung (interpolation/faltung1D.m)

Klicken für den Quellcode von interpolation/faltung1D.m

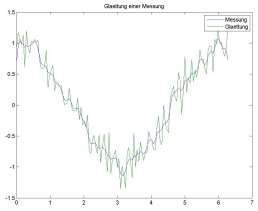


Abbildung 2.12: 1D–Faltung mit glattem Vektor, Glättung

Klick für Bild glatt1D

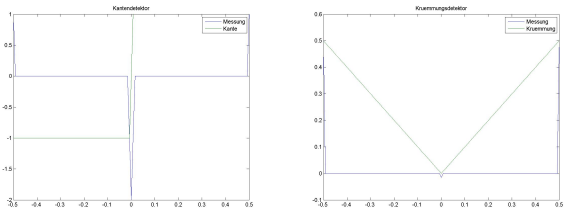


Abbildung 2.13: 1D–Faltung mit einem Kanten– bzw. Krümmungsdetektor
Klick für Bild kant1D
Klick für Bild kruem1d

```
function faltung2D
%FALTUNG2D
Y=imread('cameraman.tif');
Y=double(Y);
Z=[0 -1 0;-1 0 1; 0 1 0];
F=conv2(Y,Z);
```

Listing 2.18: Zweidimensionale Faltung (interpolation/faltung2D.m)

Klicken für den Quellcode von interpolation/faltung2D.m

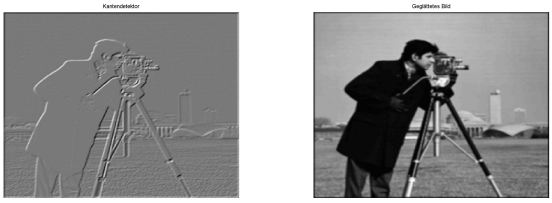


Abbildung 2.14: 2D Kantendetektor, Glättung
Klick für Bild kante2d
Klick für Bild glatt2d

Alle diese Filter sind aus Bildbearbeitungsprogrammen wie Photoshop bekannt. Dort gibt es auch die Option, Filter rückgängig zu machen. Dahinter steckt die Idee, ein unscharfes Foto nachträglich zu schärfen. Die mathematische Grundlage dafür und für die Realisierung der Faltung mit Hilfe von Fouriertransformationen gibt der Faltungssatz:

Satz 2.34 *Seien $y, z \in \mathbb{C}^N$. Dann gilt für die symmetrische diskrete Faltung von y und z*

$$\widehat{y}_p \widehat{z}_p = \widehat{(y * z)}_p.$$

Beweis: Sei $\omega_N = e^{-2\pi i/N}$ (beachte das Vorzeichen im Exponenten!).

$$\begin{aligned}
\widehat{y}_p \widehat{z}_p &= \left(\sum_{j=0}^{N-1} y_j \omega_N^{pj} \right) \left(\sum_{l=0}^{N-1} z_l \omega_N^{pl} \right) \\
&= \sum_{j=0}^{N-1} \sum_{l=0}^{N-1} y_j z_l \omega_N^{p(j+l)} \quad k = j + l \\
&= \sum_{j=0}^{N-1} \sum_{k=j}^{j+N-1} y_j z_{k-j} \omega_N^{pk} \\
&= \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} y_j z_{k-j} \omega_N^{pk} \qquad \text{(Summand ist } N\text{-periodisch in } k\text{)} \\
&= \sum_{k=0}^{N-1} \omega_N^{pk} \sum_{j=0}^{N-1} y_j z_{k-j} \\
&= \sum_{k=0}^{N-1} \omega_N^{pk} (y * z)_k \\
&= \widehat{(y * z)}_p.
\end{aligned}$$

□

Diser Satz ist sehr wichtig in der Bildverarbeitung. Falls ein unscharfes Bild $y * z$ und die Unschärfefunktion z bekannt sind, so können wir deren Fouriertransformierte berechnen, und es gilt

$$\widehat{y}_p = \widehat{(y * z)}_p / \widehat{z}_p.$$

Hieraus lässt sich das scharfe Bild y durch inverse Fouriertransformation berechnen. Dies ist gültig, solange die Fourierkoeffizienten von z nicht verschwinden.

Theoretisch lassen sich also mit dieser Formel unscharfe Bilder scharfrechnen, wenn man die Filterfunktion z kennt (unscharf maskieren, s. Übungen). Praktisch funktioniert das nur sehr eingeschränkt, da die Fourierkoeffizienten von z sehr schnell gegen 0 gehen und man durch sehr kleine Zahlen teilen muss.

2.8.4 FT in der Signalkompression

Aus der Analysis 1 (oder dem Forster) ist bekannt, dass sich eine stetige 2π-periodische Funktion f auf $\mathbb{R}$ als Linearkombination der trigonometrischen Funktionen schreiben lässt, also

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx} = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx))$$

mit den Fourierkoeffizienten

$$\begin{aligned} c_k &= \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} \mathsf{dx},\ k = -\infty \dots \infty, \\ a_k &= \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(kx) \mathsf{dx},\ k = 0 \dots \infty, \\ b_k &= \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(kx) \mathsf{dx},\ k = 1 \dots \infty \end{aligned}$$

(Fourierreihe). Die Fourierreihe stellt also periodische Signale als Linearkombination von periodischen Funktionen unterschiedlicher Frequenz dar.

Sei nun f ein aufgenommener, periodischer Ton. Dann sind typischerweise nur wenige c_k im Betrag groß. In umfangreichen Studien (zuerst wohl in Mayer [1896], Originalveröffentlichung 1894 (!Doubtful!)) wurden Aussagen etabliert wie: Falls für eine Frequenz k $|c_k|$ groß ist, die Frequenz k also mit großem Anteil im Signal vorkommt, und die Frequenzen k' in der Nähe von k mit $|k' - k| < d$ mit sehr kleinem Anteil, so sind die Anteile in k' nicht hörbar.

Konsequenterweise müssen diese Frequenzen k' bei der Speicherung oder Übertragung von Tönen auch nicht berücksichtigt werden.

Natürlich ist in diesen Fällen nicht die Funktion selbst bekannt, es stehen nur äquidistante Messwerte zu Zeitpunkten x_j zur Verfügung. Approximieren wir das Integral in der Formel aber etwa durch die Riemannsumme

$$c_k \sim \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) e^{-ikx_j},$$

so erhalten wir (bis auf Konstante) die Formel für die diskrete Fouriertransformation. Falls alle Werte reell sind, kann man alternativ auch die diskrete Cosinustransformation nutzen.

Beispiel 2.35 *Sei $f(x) = |x|$ im Intervall $[-\pi, \pi]$, periodisch fortgesetzt auf $\mathbb{R}$. Die Abbildung zeigt die Approximation an f für verschiedene Interpolationsgrade.*

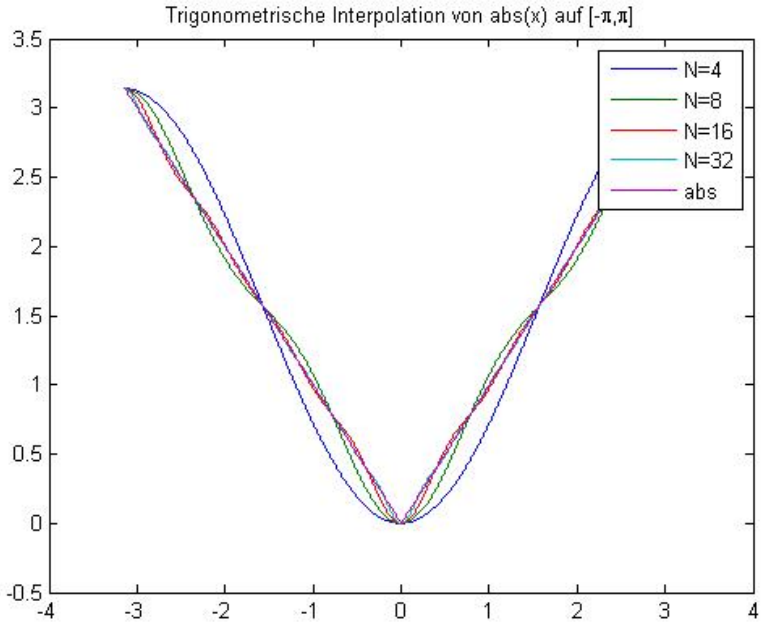


Abbildung 2.15: Trigonometrische Interpolation

Klick für Bild simplefour

```
function simplefour
%SIMPLEFOUR

    function y1=compute(N)
        x=(0:N-1)/N*2*pi;
        y=f(x-pi);
```

Listing 2.19: Trigonometrische Interpolation (interpolation/simplefour.m)

Klicken für den Quellcode von interpolation/simplefour.m

Beispiel 2.36 *Das gleiche Beispiel mit der Cosinus–Transformation. Das Ergebnis ist natürlich dasselbe, nur die Programme unterscheiden sich.*

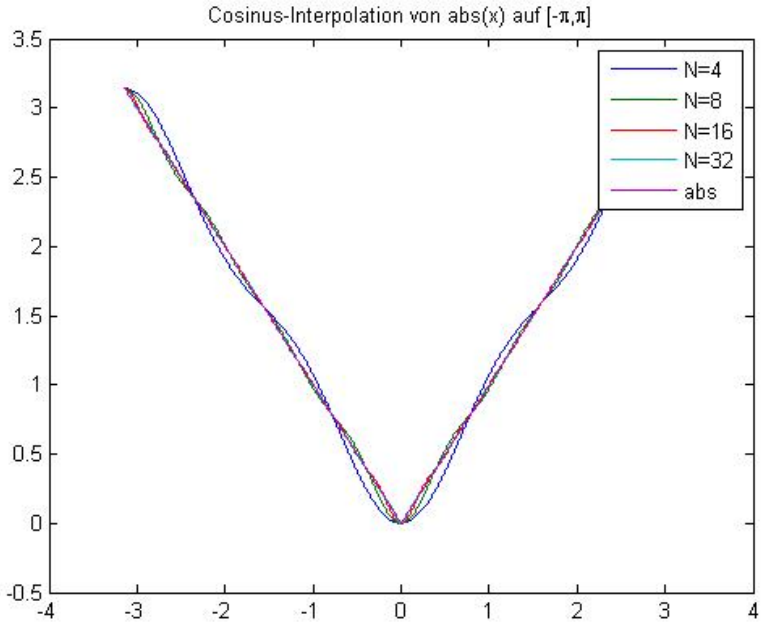


Abbildung 2.16: Interpolation mit der Cosinus-Transformation

Klick für Bild simplecos

```
function simplecostrans
%SIMPLECOSTRANS

    function y1=compute(N)
        x=(0:N-1)/N*pi;
        y=f(x-pi);
```

Listing 2.20: Cosinus-Transformation (interpolation/simplecostrans.m)

Klicken für den Quellcode von interpolation/simplecostrans.m

Technisch passiert nun das folgende:

1. Ein Signal wird aufgenommen.
2. Das Signal wird in seine Frequenz–Bestandteile zerlegt mit der diskreten Cosinus–Transformation.
3. Unhörbare Bestandteile werden identifiziert und $= 0$ gesetzt.
4. Die restlichen Koeffizienten der Cosinus-Transformation werden übermittelt.

5. Das Signal wird mit der inversen Cosinus–Transformation wieder zusammengesetzt.

Der letzte Schritt passiert dabei im MP3–Player auf sehr bescheidener Hardware, das muss also sehr einfach durchzuführen sein. Die Formeln in 2.3 und 2.4 sind dazu nicht geeignet, der Aufwand zur Transformation von N Werten wäre N^2 Additionen und Multiplikationen. Wir werden deutlich einfachere Formeln herleiten, die mit $O(N \log N)$ Rechenoperationen auskommen.

Der Ehrlichkeit halber sei gesagt, dass der tatsächliche Prozess im Größenordnungen komplizierter ist. Tatsächlich bildet aber die DCT die Basis dieser Komprimierungen.

Diese Idee lässt sich auch für Bilder durchführen. Mit Hilfe der zweidimensionalen Fouriertransformation schreiben wir unsere Funktion als Linearkombination von trigonometrischen Funktionen, wieder ist die Idee, dass der Eindruck des Bildes durch relativ wenige Entwicklungskoeffizienten beschrieben wird. Dies ist die Grundlage der JPEG– und MPEG–Kompression (wieder mit der Cosinustransformation an Stelle der Fouriertransformation).

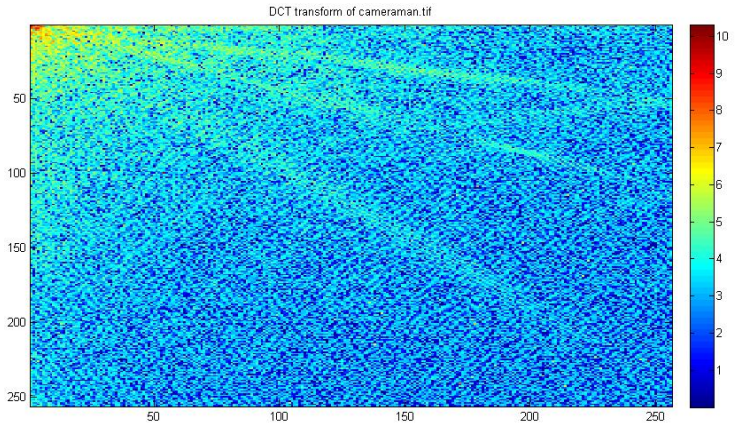


Abbildung 2.17: DCT des Kameramann-Bildes (abs val of log)

Klick für Bild cameramandct

Im Beispielbild wurde die zweidimensionale Cosinus–Transformation (DCT entlang Zeilen, dann Spalten) auf dem Kameramann–Bild ausgeführt. Der Betrag des log des Betrags der DCT wurde geplottet. Deutlich sichtbar ist, dass die relevanten Koeffizienten in der linken oberen Ecke stehen (niederfrequente Anteile).

2.8.5 FFT: Schnelle Fourier–Transformation nach Cooley und Tukey

Wir beginnen mit zwei kurzen Beispielen. Sei wieder $N = 2$, also $\omega_N = -1$. Damit ergibt sich für die $\widehat{y}_k$:

$$\begin{array}{rcrcl} \widehat{y}_0 & = & y_0 & + & y_1 \\ \widehat{y}_1 & = & y_0 & - & y_1. \end{array}$$

Für $N = 4$ ist $e^{-2\pi i/4} = -i$, also

$$\begin{array}{rcrcrcrcrcrcrcr} \widehat{y}_0 & = & y_0 & + & & y_1 & + & & y_2 & + & & y_3 & = & (y_0+y_2) & + & & (y_1+y_3) \\ \widehat{y}_1 & = & y_0 & + & (-i) & y_1 & + & (-1) & y_2 & + & (i) & y_3 & = & (y_0-y_2) & + & (-i) & (y_1-y_3) \\ \widehat{y}_2 & = & y_0 & + & (-1) & y_1 & + & & y_2 & + & (-1) & y_3 & = & (y_0+y_2) & - & & (y_1+y_3) \\ \widehat{y}_3 & = & y_0 & + & i & y_1 & + & (-1) & y_2 & + & (-i) & y_3 & = & (y_0-y_2) & - & (-i) & (y_1-y_3) \end{array}$$

Durch naives Ausrechnen der Formel in 2.3 für $N = 4$ bräuchten wir 12 (komplexe) Additionen. Offensichtlich lässt sich diese Anzahl verringern, indem man zunächst jeweils eine Fouriertransformation halber Länge auf den Fourierkoeffizienten mit geradem und ungeradem Index durchführt, und auf den Ergebnissen wieder zwei Fouriertransformationen der Länge zwei ausführt.

Dies ist die Grundidee der schnellen Fouriertransformation (FFT) von Cooley und Tukey (1965), einer der Top 10-Algorithms of the century.

Satz 2.37 (FFT nach Cooley und Tukey)
Sei $N = pq$. Dann kann die Fouriertransformation eines Vektors $(y_0, \ldots, y_{N-1})$ durch p Fouriertransformationen der Länge q, q Fouriertransformationen der Länge p und N Multiplikationen berechnet werden.

Beweis: Sei wieder

$$\omega_N = e^{-2\pi i/N}.$$

Wir bemerken zunächst, dass

$$\omega_N^p = \omega_q,\ \omega_N^q = \omega_p.$$

Sei $j \in \{0, \ldots, N-1\}$. Dann können wir j eindeutig schreiben als

$$j = ps + r,\ s \in \{0, \ldots, q-1\},\ r \in \{0, \ldots, p-1\}.$$

Hierbei ist $r = j \mod p$ und $s = (j - r)/p$. Die Abbildung $j \mapsto (r, s)$ ist bijektiv auf den angegebenen Mengen.
Sei nun $k \in \{0, \ldots, N-1\}$. Dann können wir k entsprechend schreiben als

$$k = k_1 q + k_2,\ k_1 = 0 \ldots p-1,\ k_2 = 0 \ldots q-1$$

und es gilt

$$\begin{aligned}
\widehat{y}_k &= \sum_{j=0}^{N-1} \omega_N^{kj} y_j \\
&= \sum_{r=0}^{p-1} \sum_{s=0}^{q-1} \omega_N^{k(ps+r)} y_{ps+r} \\
&= \sum_{r=0}^{p-1} \omega_N^{kr} \sum_{s=0}^{q-1} \omega_q^{ks} y_{ps+r} \\
&= \sum_{r=0}^{p-1} \omega_p^{k_1 r} \underbrace{\omega_N^{k_2 r} \underbrace{\sum_{s=0}^{q-1} \omega_q^{k_2 s} \underbrace{y_{ps+r}}_{=:C_{r,s}}}_{=:A_{r,k_2}}}_{=:B_{r,k_2}}.
\end{aligned}$$

Diese Summe berechnen wir nach der folgenden Vorschrift:

1. Setze
$$C_{r,s} = y_{ps+r},\ r = 0 \ldots p-1,\ s = 0 \ldots q-1.$$

2. Für jedes r von $0 \ldots p-1$ und k_2 von $0 \ldots q-1$ berechne
$$A_{r,k_2} = \sum_{s=0}^{q-1} \omega_q^{k_2 s} y_{ps+r} = \sum_{s=0}^{q-1} \omega_q^{k_2 s} C_{r,s}.$$
Für festes r ist das jeweils eine FT der Länge q. Insgesamt sind das p Fouriertransformationen der Länge q, ausgeführt auf den Vektoren C_r.

3. Für jedes r von $0 \ldots p-1$ und jedes k_2 von $0 \ldots q-1$ berechne
$$B_{r,k_2} = \omega_N^{k_2 r} A_{r,k_2}.$$
Das sind N Multiplikationen.

4. Für jedes k_1 von $0 \dots p-1$ und k_2 von $0 \dots q-1$ berechne

$$\widehat{y}_{k_1 q + k_2} = \sum_{r=0}^{p-1} \omega_p^{k_1 r} B_{r,k_2}.$$

Für jedes feste k_2 ist das eine FT der Länge p. Insgesamt sind das q Fouriertransformationen der Länge p, ausgeführt auf den Vektoren $B_{\cdot,k_2}$.

□

Beispiel 2.38 (Fouriertransformation von gerader Länge)
Wir betrachten den Fall $q = 2$, also $N = p \cdot 2$. Dann sind $s, k_2 \in \{0, 1\}$. Wir erhalten

$$C_{r,0} = y_r,\ C_{r,1} = y_{r+p}.$$

Auf diesen ist nun eine Fouriertransformation durchzuführen, also

$$A_{r,0} = C_{r,0} + C_{r,1},\ A_{r,1} = C_{r,0} - C_{r,1}.$$

Weiter

$$B_{r,0} = A_{r,0},\ B_{r,1} = \omega_N^r A_{r,1}.$$

Auf den zwei Vektoren $B_{r,0}$ und $B_{r,1}$, $r = 0 \dots p-1$, führen wir nun jeweils eine Fouriertransformation der Länge p durch und erhalten $\widehat{y}_{2r}$ bzw. $\widehat{y}_{2r+1}$.

Falls p wieder gerade ist, so können wir die Formel rekursiv anwenden.

Nun betrachten wir den Fall $p = 2$, also $N = 2q$. Dann sind $r, k_1 \in \{0, 1\}$. Wir erhalten

$$C_{0,s} = y_{2s},\ C_{1,s} = y_{2s+1}.$$

C_0 enthält also die Elemente von y mit geradem Index, C_1 die mit ungeradem Index. Auf diesen führen wir nun eine Fouriertransformation der Länge q durch und erhalten A_{0,k_2} bzw. A_{1,k_2}. Weiter

$$B_{0,k_2} = A_{0,k_2},\ B_{1,k_2} = \omega_N^{k_2} A_{1,k_2}.$$

Unser Ergebnis erhalten wir nun durch

$$y_{k_2} = B_{0,k_2} + B_{1,k_2},\ y_{k_2+q} = B_{0,k_2} - B_{1,k_2}.$$

Wir erhalten zwei grundsätzlich unterschiedliche Implementierungen. Wieder können wir natürlich, falls q gerade ist, die Formel rekursiv nutzen.

Damit können Fouriertransformationen schnell berechnet werden, falls N ein Produkt kleiner Primzahlen ist, weil dann dieser Satz möglichst oft rekursiv genutzt werden kann. Beispielsweise gilt für $N = 2^p$:

Satz 2.39 Aufwand der diskreten Fouriertransformation für Zweierpotenzen
Sei M_p die Anzahl der Multiplikationen, A_p die Anzahl der Additionen für eine Fouriertransformation der Länge $N = 2^p$. Dann gilt

$$M_p \leq p2^p = N \log_2 N,\ A_p \leq p2^p = N \log_2 N.$$

Beweis: Zur Berechnung einer Fouriertransformation der Länge 2^{p+1} werden nach Satz 2.37 höchstens 2 Fouriertransformationen der Länge 2^p, 2^p Fouriertransformationen der Länge 2 und 2^{p+1} Multiplikationen benötigt.
Es gilt $M_1 = 0$, $A_1 = 2$.

$$\begin{aligned} M_{p+1} &\leq 2M_p + 2^{p+1} \\ &\leq 2p2^p + 2^{p+1} \\ &= 2^{p+1}(p+1). \end{aligned}$$

$$\begin{aligned} A_{p+1} &\leq 2A_p + 2^p A_1 \\ &\leq p2^{p+1} + 2^{p+1} \\ &= (p+1)2^{p+1} \end{aligned}$$

□

Machen wir uns klar, was dieser Satz bedeutet: Mit naivem Ausrechnen würde eine Fouriertransformation der typischen Länge $N = 1024 = 2^{10}$ ca. $N^2 \sim 10^6$ Additionen und Multiplikationen benötigen. Durch Nutzung der Cooley–Tukey–FFT benötigen wir nur noch $N \log_2 N = 10 \cdot 1024 \sim 10^4$ Additionen und Multiplikationen, wir haben also den benötigten Aufwand um den Faktor 100 reduziert. Dies ist tatsächlich auch für eingebettete Systeme mit sehr schwacher Rechenleistung berechenbar.

Damit ist die Theorie der schnellen FT aber noch nicht beendet. Völlig unklar etwa ist, ob es auch schnelle Algorithmen gibt, wenn N eine Primzahl ist und wir den Algorithmus nicht anwenden können.

Tatsächlich gibt es für alle Längen spezielle Algorithmen, die eine schnelle Ausführung ermöglichen (siehe etwa Beth [1984] oder die Primzahl–FFT in Winograd [1978]). Die schnelle Fouriertransformation ist beispielsweise in der quelloffenen Bibliothek FFTW, Fastest Fourier Transform in the West, implementiert, sie gilt

allgemein als die effizienteste Implementation. Einer der dort angewandten Tricks ist, dass die Cooley–Tukey–FFT mit unterschiedlichen Zerlegungen angewandt und auf dem jeweiligen Prozessor getestet wird, welche die schnellste ist.

Eine schöne Übersicht über FFT–Algorithmen findet sich im aktuellen Buch Burrus. Die Ideen der Fouriertransformation können auf allgemeinere orthogonale (Wavelet–) Transformationen erweitert werden, siehe hierzu die klassischen Bücher Daubechies et al. [1992] oder Louis et al. [1998]. Von großer Bedeutung für die praktische Arbeit ist die Nicht–äquidistante schnelle Fouriertransformation NFFT, die sich ebenfalls aus der schnellen Fouriertransformation ableiten lässt. Eine Übersicht über mehrere Verfahren zu effizienten Algorithmen, auch zu schnellen Matrix–Multiplikationen und insbesondere den engen Beziehungen zur Algebra, finden sich im lesenswerten (alten) Vorlesungsskript Natterer [1994].

```
function yhat = myfft( y )
%MYFFT
%This is not really fast - it shows how the FFT
%computes its way. Do not use for real computations!
%This is he Cooley-Tukey algorithm.
format compact;
```

Listing 2.21: Einfache Implementierung der FFT (interpolation/myfft.m)

Klicken für den Quellcode von interpolation/myfft.m

2.9 Spline–Interpolation

Die kurze Behandlung innerhalb dieses Abschnitts wird der tatsächlichen Bedeutung der Spline–Interpolation nicht gerecht, tatsächlich ist sie das Standardwerkzeug zur Interpolation und in jedem Werkzeugkasten zur Numerik implementiert. Für eine umfassende Behandlung verweisen wir auf de Boor [2001]. Eine umfassende Würdigung von de Boor und Splines findet sich in DeVore and Ron [2005].

2.9.1 Eindimensionale Splines und B–Splines

Im Kapitel über Interpolationsfehler haben wir gesehen, dass es von Vorteil ist, das zugrundeliegende Intervall für die Interpolation aufzuteilen und die Interpolation mit niedrigem Polynomgrad auf jedem Teilintervall einzeln durchzuführen. Leider geht dabei die Stetigkeit/Differenzierbarkeit an den Intervallgrenzen verloren.

Splines beheben diesen Mangel: Sie teilen zunächst das Intervall $[a, b]$ an Knotenpunkten s_i auf. Auf jedem Einzelintervall $[s_i, s_{i+1}]$ sind die Splines (der Ordnung k)

Polynome p_i vom Grad $k-1$, mit der zusätzlichen Forderung, dass an den Knoten die zusammengesetzte Funktion $(k-2)$–mal differenzierbar ist, die Polynome von links und rechts also bis zur $(k-2)$–ten Ableitung übereinstimmen.

Beispiel 2.40
Gegeben seien die Stützstellen $0,1,2,3$ *und die Stützwerte* $0,0,0,1$.
Das interpolierende Polynom ist mit Lagrange

$$p(x) = \frac{x(x-1)(x-2)}{6}.$$

Der lineare interpolierende Polygonzug ist nicht differenzierbar.
Wir wählen nun die Knotenpunkte $s_k = x_k$. *Dann gilt: Die Funktion*

$$s(x) = \max(x-2,0)^2$$

interpoliert die Stützwerte und ist stetig differenzierbar auf dem gesamten Intervall $[0,3]$, *in jedem Teilintervall* $[s_i, s_{i+1}]$ *liegt sie in* $\mathcal{P}_2$, *sie ist also ein Spline der Ordnung* 3.

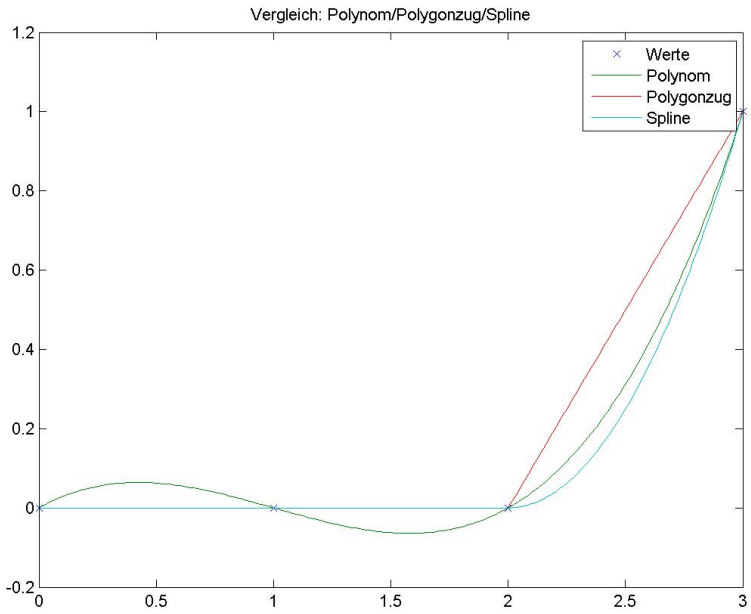


Abbildung 2.18: Vergleich Polynom/Polygon/Spline

Klick für Bild splinetest
Klick für Matlab Figure splinetest

Definition 2.41 (Splines)
Seien $s_0 < s_1 < \ldots < s_n$ reelle Zahlen. Eine Funktion

$$s : [s_0, s_n] \mapsto \mathbb{R}$$

heißt Spline der Ordnung k (zu den Knoten $s_0 \ldots s_n$), falls

1. $s \in C^{(k-2)}([s_0, s_n])$ *für* $k > 1$.
2. $s|_{[s_i, s_{i+1})} \in \mathcal{P}_{k-1}$, $i = 0 \ldots n-1$.

Üblicherweise wird der Spline über sein eigentliches Definitionsgebiet hinaus fortgesetzt, z.B. linear oder periodisch.

Beispiel 2.42

1. *Sei $p \in \mathcal{P}_{k-1}$. Dann ist p Spline der Ordnung k.*
2. *Sei s_i ein Knotenpunkt, k fest. Sei*

   $$s_i^+(x) = (\max(x - s_i, 0))^{k-1}$$

   für $k > 1$ und für $k = 1$

   $$s_i^+(x) = \begin{cases} 0 & x < s_i \\ 1 & x \geq s_i. \end{cases}$$

   Dann gilt

   $$s_i^+ \in C^{(k-2)}([s_0, s_n]), s_i^+ \notin C^{(k-1)}([s_0, s_n]) \, (k \geq 2).$$

   $s_i^+ \in \mathcal{P}_{k-1}$ auf jedem Intervall $[s_j, s_{j+1})$, also ist s_i^+ Spline der Ordnung k.
3. *Splines der Ordnung 1 sind genau die Funktionen, die konstant sind in jedem Intervall $[s_i, s_{i+1})$. Sie sind nicht notwendig stetig.*
4. *Sei s linear in jedem Intervall $[s_i, s_{i+1}]$ und stetig auf $[s_0, s_n]$, also ein Polygonzug. Genau dann ist s Spline der Ordnung 2.*

Sei s ein Spline der Ordnung k. Wir haben n Intervalle. Auf jedem Intervall ist $s \in \mathcal{P}_{k-1}$, also gibt es insgesamt nk Polynomkoeffizienten. An den Schnittstellen s_1 bis s_{n-1} müssen die Ableitungen links und rechts bis zum Grad $(k-2)$ übereinstimmen, macht insgesamt $(k-1)(n-1)$ lineare Bedingungen, von denen man mit dem Satz über die Eindeutigkeit der Hermite–Interpolation sieht, dass sie unabhängig sind. Es gilt also für die Dimension des Vektorraums V_s der Splines zu gegebener Knotenmenge $\{s_0, \ldots, s_n\}$ vom Grad $k-1$

$$\dim V_S \leq nk - (n-1)(k-1) = n + k - 1.$$

Wir können sofort eine Basis für diesen Vektorraum angeben: Nach Vorbemerkung sind alle Polynome in $\mathcal{P}_{k-1}$ Splines, dies ist ein Vektorraum der Dimension k. Für $i = 1 \dots n-1$ sind die s_i^+ linear unabhängige Splines, aber keine Polynome. Die Monome zusammen mit s_i^+, $i = 1 \dots n-1$, bilden also eine linear unabhängige Menge mit $(n+k-1)$ Elementen und damit eine Basis des Splineraums.

Dies gibt uns bereits eine einfache Möglichkeit, das Interpolationsproblem für Splines zu lösen: Wir wählen

$$N + 1 = n + k - 1$$

Stützstellen und Stützwerte und lösen das Interpolationsproblem mit allgemeinen Ansatzfunktionen 2.27 mit den Ansatzfunktionen in dieser Basis. Wir nutzen diesen Algorithmus zur Interpolation des Runge–Beispiels.

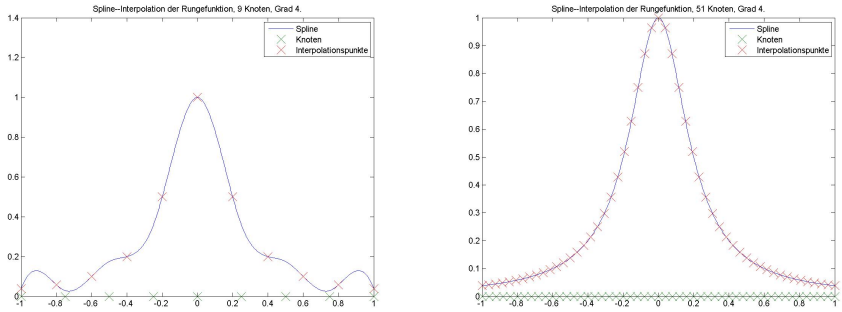


Abbildung 2.19: Interpolation des Rungebeispiels mit einfachen Ansatzfunktionen
Klick für Bild simplesplinerunge8-4
Klick für Matlab Figure simplesplinerunge8-4
Klick für Bild simplesplinerunge50-4
Klick für Matlab Figure simplesplinerunge50-4

```
function y = simplesplinebasisval( s,k,l,x )
%SIMPLESPLINEBASISVAL evaluate simple basis splines
%s – knots
%x – evaluation point
%k – degree of spline
%l – index of basis function, starting at o
```

Listing 2.22: Einfache Spline–Ansatzfunktionen (interpolation/simplesplinebasisval.m)

Klicken für den Quellcode von interpolation/simplesplinebasisval.m

```
function p = simplesplines( s,x,y,k )
%SIMPLESPLINES compute simple spline coefficients
%knots in s, evaluation points in x, values in y,
%order of spline in k (k=2 means linear)
%Returns vector of polynomial coefficients.
if (nargin==0)
```

Listing 2.23: Berechnung von Spline–Koeffizienten mit allgemeinen Ansatzfunktionen (interpolation/simplesplines.m)

Klicken für den Quellcode von interpolation/simplesplines.m

```
function [ output_args ] = simplesplinedemo( input_args )
%SIMPLESPLINEDEMO Summary of this function goes here
%   Detailed explanation goes here

    n=50;
    k=4;
```

Listing 2.24: Demo zu einfachen Spline–Ansatzfunktionen (interpolation/simplesplinedemo.m)

Klicken für den Quellcode von interpolation/simplesplinedemo.m

Dies ist leider doppelt ineffizient: Typischerweise ist haben wir viele Knoten, d.h. n ist groß, k klein. Mit diesen Ansatzfunktionen wäre die zugehörige Vandermonde–ähnliche Matrix 2.2 fast voll besetzt, d.h. das zugehörige Gleichungssystem zur Bestimmung der Koeffizienten wäre schwierig zu lösen. Zusätzlich verschwindet z.B. auf dem Intervall $[s_{n-1}, s_n]$ keine einzige Ansatzfunktion. Zur Auswertung des Splines müssten wir also alle Ansatzfunktionen auswerten und aufaddieren, der Aufwand dazu wäre mindestens $O(n)$, obwohl der Spline in diesem Intervall nur den Polynomgrad $k << n$ hat.

Wir werden eine günstigere Basis bestimmen, bei der die Ansatzfunktionen nur einen kleinen Träger haben, so dass die Matrix 2.2 nur wenige von Null verschiedene Zahlen enthält, und zusätzlich die resultierende Splinefunktion einfach auswertbar ist.

Definition 2.43 (B-Splines)
Seien $s_0 < s_1 < \ldots < s_n$ reelle Zahlen. Dann heißen

$$B_{i,1}(x) = \begin{cases} 1 & s_i \leq x < s_{i+1} \\ 0 & sonst \end{cases}$$

$$B_{i,k}(x) = \frac{x - s_i}{s_{i+k-1} - s_i} B_{i,k-1}(x) + \frac{s_{i+k} - x}{s_{i+k} - s_{i+1}} B_{i+1,k-1}(x)$$

B–Splines der Ordnung k.

Beispiel 2.44

1. *B-Splines der Ordnung $k = 1$ sind stückweise konstant, also Splines der Ordnung 1.*
2. *Sei $k = 2$. Für $x < s_i$ und $x > s_{i+2}$ ist*

   $$B_{i,1}(x) = B_{i+1,1}(x) = 0,$$

   daher ist auch $B_{i,2}(x) = 0$. Weiter gilt

   $$B_{i,2}(x) = \begin{cases} \frac{x - s_i}{s_{i+1} - s_i}, & s_i \leq x < s_{i+1} \\ \frac{s_{i+2} - x}{s_{i+2} - s_{i+1}}, & s_{i+1} \leq x < s_{i+2}. \end{cases}$$

   s ist also stetig und linear in jedem Intervall $[s_i, s_{i+1}]$. Also ist $B_{i,2}$ der lineare Polygonzug mit

   $$B_{i,2}(s_j) = \delta_{i+1,j}.$$

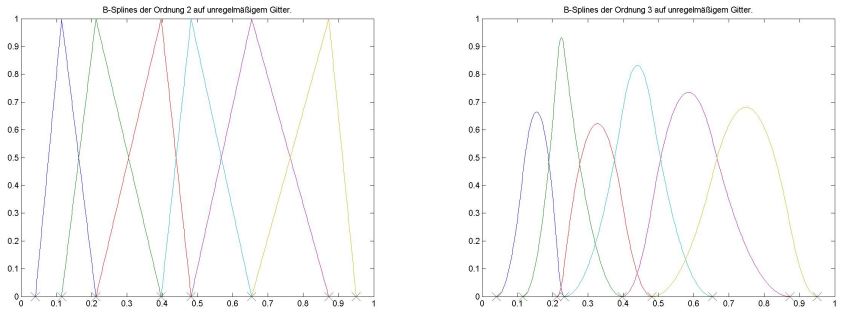


Abbildung 2.20: B–Splines der Ordnungen 2 und 3 auf unregelmäßigen Gittern

Klick für Bild bspline-8-2
Klick für Matlab Figure bspline-8-2
Klick für Bild bspline-9-3
Klick für Matlab Figure bspline-9-3

Bemerkung: Diese Definition kann auch für mehrfache Knoten ($s_i = s_{i+1}$) erweitert werden, in diesem Fall ist B definiert als der Grenzwert für $s_{i+1} \mapsto s_i$.

Das Beispiel lässt sich fortsetzen, es gilt der Satz:

Satz 2.45 *(Eigenschaften der B–Splines)*

1. $B_{i,k}$ *ist Spline der Ordnung* k*.*
2. *Der Träger von* $B_{i,k}$ *ist* (s_i, s_{i+k}) *(*$[s_i, s_{i+k})$ *für* $k = 1$*).*
3. *Falls links von* s_0 *und rechts von* s_n *noch jeweils* $(k-1)$ *zusätzliche paarweise verschiedene Knoten* s_{-k+1} *bis* s_{-1} *bzw.* $s_{n+1} \ldots s_{n+k-1}$ *eingefügt werden, so sind die B–Splines* $B_{j,k}$*,* $j = -k-1 \ldots n-1$*, Basis des Spline–Ansatzraums.*

Beweis: Durch Darstellung der B–Splines als dividierte Differenzen, siehe z.B. de Boor [2001], S. 87ff, oder Freund and Hoppe [2007].
Zum letzten Punkt: Zusammen mit den eingefügten Knoten gibt es gerade $(n+k-1)$ B–Splines $B_{i,k}$, $i = -k+1 \ldots n-1$, mit Träger in (s_0, s_n). Wegen der Träger–Bedingung sind sie linear unabhängig. □

```
function y = bspline( i,k,xo,x )
%BSPLINE
    function y=Bint(i,k,xo)
        if (k==1)
            y=zeros(size(xo));
            y((xo>x(i))&(xo<=x(i+1)))=1;
```

Listing 2.25: Berechnung von B–Splines (interpolation/bspline.m)

Klicken für den Quellcode von interpolation/bspline.m

```
function y = dbspline( i,k,xo,x,j )
%DBSPLINE
% j is order of differentiation
if (j==0)
    y=bspline(i,k,xo,x);
    return
```

Listing 2.26: Berechnung der Ableitung (interpolation/dbspline.m)

Klicken für den Quellcode von interpolation/dbspline.m

```
function bsplinedemo2
%BSPLINEDEMO2
x=0:10;
x=[0 0.5 1 2.5 3.5 4 4.5 7 8 8.2 10];
P=100;
xo=(0:P*10)/P;
```

Listing 2.27: B–Splines (interpolation/bsplinedemo2.m)

Klicken für den Quellcode von interpolation/bsplinedemo2.m

```
function y = splinecoeff( x,i,k,j )
%SPLINECOEFF Coefficients of spline B(i,k)
%in interval starting at x_j
if (k==1)
    if (i==j)
        y=1;
```

Listing 2.28: Berechnung der Koeffizienten von B–Splines (interpolation/splinecoeff.m)

Klicken für den Quellcode von interpolation/splinecoeff.m

```
function [ output_args ] = testsplinecoeff( input_args )
%TESTSPLINECOEFF
N=50;
i=1;
k=4;
p=k+1;
```

Listing 2.29: Auswertung von B–Splines anhand der Koeffizienten (interpolation/testsplinecoeff.m)

Klicken für den Quellcode von interpolation/testsplinecoeff.m

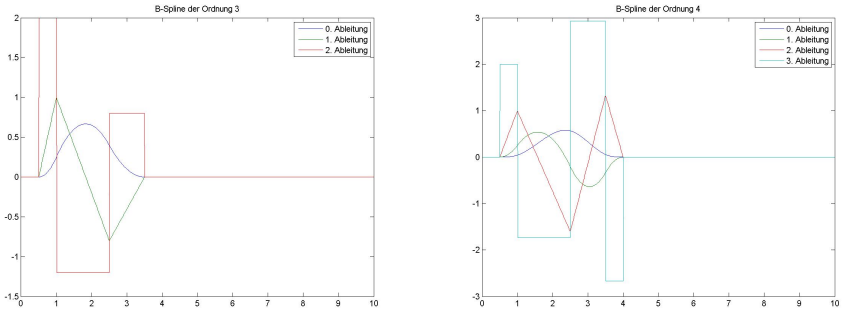


Abbildung 2.21: Splines der Ordnung 3 und 4
Klick für Bild spline3
Klick für Bild spline4

Es bleibt noch zu beweisen, dass das Interpolationsproblem eindeutig lösbar ist, d.h. ob die zugehörige Vandermonde–artige Matrix V invertierbar ist. Dies muss natürlich von der Wahl der Stützstellen abhängen: Es ist sicherlich nicht möglich, alle Stützstellen in einem einzigen Intervall $[s_i, s_{i+1}]$ zu wählen, die Stützstellen müssen sich gleichmäßig auf $[s_0, s_n]$ verteilen.

Tatsächlich gilt der Satz:

Satz 2.46 *(B–Splines sind Basis des Splineraums)*
Seien $s_0 < s_1 < \ldots < s_n$ reelle Zahlen. Gegeben seien Stützstellen x_j mit Stützwerten y_j, $j = 0 \ldots n-k$ und es gelte

$$x_j \in (s_j, s_{j+k}),\ j = 0 \ldots n-k.$$

Dann gibt es genau einen Spline s der Ordnung k,

$$s(x) = \sum_{i=0}^{n-k} a_i B_{ik}(x)$$

mit Koeffizienten $a_i \in \mathbb{R}$, so dass

$$s(x_j) = y_j,\ j = 0 \ldots n-k.$$

Die Koeffizienten lassen sich wieder mit Hilfe der Vandermonde–ähnlichen Matrix 2.2 berechnen.

Beweis: Schoenberg and Whitney [1953] mit der Definition der B–Splines über dividierte Differenzen, De Boor [1976] mit linearer Algebra, einfachster Beweis in de Boor [2001], Seite 97f. □

Die Dimension des Ansatzraums unterscheidet sich von der bereits berechneten, denn die B–Splines sind nur dann eine Basis, wenn wir links und rechts vom Interpolationsintervall noch jeweils $(k-1)$ Knoten einfügen. Tun wir das, kommen wir auf $n-k+1+2(k-1) = n+k-1$ Ansatzfunktionen, wie bereits berechnet.

Wir haben also eine alternative Basis des Spline–Ansatzraums gefunden, und können eine Interpolation mit allgemeinen Ansatzfunktionen durchführen. Dies ist jetzt deutlich effizienter als in unserem ersten Ansatz: Wählen wir die Knoten wie im Satz, so befinden sich genau k Knoten im Intervall $[s_i, s_{i+k}]$. Die k. Spalte von V enthält also höchstens k von Null verschiedene Einträge, es ist eine Bandmatrix. Das zugehörige lineare Gleichungssystem ist also leicht auflösbar.

Sei nun $x \in [s_0, s_n]$. Dann sind höchstens k Ansatzfunktionen an dieser Stelle ungleich Null, auch die Auswertung von $s(x)$ ist also einfach möglich.

Oft wählt man als Stützstellen gleich einige der Knoten, dies ist nach dem Satz jedoch nicht notwendig.

De Boor berechnet auch die Kondition der Matrizen, sie ist klein, falls die Interpolationspunkte einigermaßen gleichmäßig verteilt sind.

Vorlesungsnotiz: 25.4.2013

Wir wollen nun noch eine wichtige Minimaleigenschaft der kubischen Splines beweisen und beginnen mit

Lemma 2.47 *(Eine unendlich oft differenzierbare Funktion mit beschränktem Träger) Seien $\widetilde{x} \in \mathbb{R}$, $\epsilon > 0$. Dann ist die Funktion*

$$g_{\epsilon,\widetilde{x}}(x) = \begin{cases} \exp(\frac{1}{(x-\widetilde{x})^2-\epsilon^2}) & |x-\widetilde{x}| < \epsilon \\ 0 & \textit{sonst} \end{cases}$$

unendlich oft stetig differenzierbar. Der Träger der Funktion ist $(\widetilde{x}-\epsilon, \widetilde{x}+\epsilon)$.

Beweis: An den Enden des Intervalls geht die Exponentialfunktion von innen mit allen ihren Ableitungen exponentiell gegen 0, bei den Ableitungen kommen gebrochenrationale Funktionen hinzu, die polynomial gegen ∞ konvergieren. Insgesamt haben die Ableitungen also den Wert 0 auf dem Rand, und $g_{\epsilon,\widetilde{x}}$ ist unendlich oft differenzierbar. □

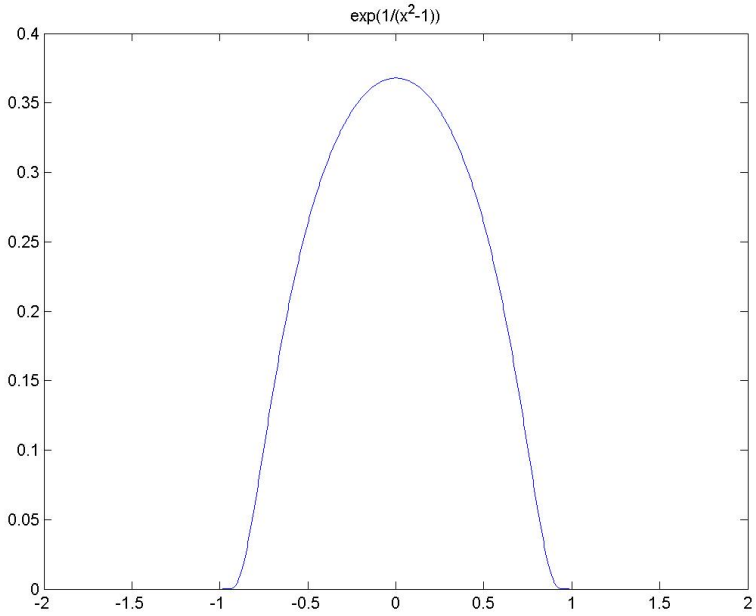


Abbildung 2.22: C^∞–Funktion mit kompaktem Träger

Klick für Bild expfrac

Die Funktion $g_{\epsilon,\widetilde{x}}$ gibt uns die Möglichkeit, eine Funktion in einem sehr kleinen Intervall um $\widetilde{x}$ zu ändern, ohne die Glattheitseigenschaften zu beeinflussen.

Die am häufigsten benutzten Splines sind die kubischen Splines der Ordnung 4. Sie sind zweimal stetig diffbar, also hinreichend glatt, und lösen unser Straklattenproblem aus der Einleitung.

Die Biegeenergie einer Latte mit Formfunktion v, die an den Punkten (x_i, y_i), $i = 0 \dots n$, eingespannt ist, kann approximiert werden durch (siehe z.B. Hanke-Bourgeois [2006], p. 372f)

$$E(u) := (u,u)^{1/2},\ (u,v) := \int_{x_0}^{x_n} u''(x)v''(x)\mathrm{dx}.$$

Sei V der Raum der möglichen Formfunktionen

$$\begin{aligned} V \quad = \{ \quad & v \in C^2(\mathbb{R}) : v \in C^\infty(\,(x_j, x_{j+1})\,),\ v(x_j) = y_j,\ j = 0 \dots n, \\ & v \text{ linear außerhalb } [x_0, x_n] \}\,. \end{aligned}$$

Dies beinhaltet insbesondere, dass die zweite Ableitung der Funktionen in V in x_0 und x_n verschwindet.

Ohne zusätzliche Kräfte nimmt die Latte eine Form $u \in V$ an, die diese Energie unter allen zugelassenen Formfunktionen minimiert, also

$$E(u) \leq E(v) \, \forall v \in V.$$

Wir behaupten: u ist ein Spline.

Satz 2.48 *(Minimaleigenschaft der kubischen B–Splines) Seien $x_0 \dots x_n$ der Größe nach geordnete, paarweise verschiedene Stützstellen und $y_0 \dots y_n$ Stützwerte. Sei*

$$\begin{aligned} V \quad = \{ \quad & v \in C^2(\mathbb{R}) : v \in C^\infty(\,(x_j, x_{j+1}\,),\, v(x_j) = y_j,\, j = 0 \dots n, \\ & v \text{ linear außerhalb } [x_0, x_n]\} \,. \end{aligned}$$

Auf $C^2(\mathbb{R})$ (linear außerhalb $[x_0, x_n]$) seien das Halbskalarprodukt (positiv semidefinite Bilinearform)

$$(u, v) = \int_{\mathbb{R}} u''(x) v''(x) dx$$

und die Halbnorm

$$E(v) = (v, v)^{1/2}$$

definiert. Sei $u \in V$ und

$$E(u) \leq E(v) \forall v \in V.$$

Dann ist u kubischer Spline der Ordnung $k = 4$ zu den Knoten $x_0 \dots x_n$.

$v''(x_0) = v''(x_n) = 0$ liefert uns zusammen mit der Interpolationsbedingung gerade $n + 1 + 2 = n + 3 = n + k - 1$ Bedingungen, d.h. die Funktion ist auch eindeutig bestimmt.

Beweis: Wir setzen $s_i := x_i$. Zu zeigen ist nur noch: u ist auf jedem Intervall (s_i, s_{i+1}) ein Polynom in $\mathcal{P}_3$.
Wir machen den üblichen Ansatz: Bei Minimierungsproblemen über (Halb–) Skalarprodukte wird immer wieder dieselbe Orthogonalitätseigenschaft genutzt (z.B. auch in der Vorlesung Numerische Lineare Algebra bei überbestimmten linearen Gleichungssystemen).
Sei $v \in V$ beliebig. Dann ist auch $u + \alpha(v - u) \in V$ für jedes $\alpha \in \mathbb{R}$. Da u $E(u)$ auf V minimiert, hat die Funktion

$$g(\alpha) := E(u+\alpha(v-u))^2 = (u+\alpha(v-u), u+\alpha(v-u)) = E(u)^2 + 2\alpha(u, v-u) + \alpha^2 E(v-u)^2$$

ein Minimum bei $\alpha = 0$. Also ist

$$g'(0) = 0$$

und damit

$$(u, v - u) = 0 \, \forall v \in V.$$

Angenommen, u ist nicht in $\mathcal{P}_3$ auf (s_i, s_{i+1}). Dann

$$\exists \widetilde{x} \in (s_i, s_{i+1}) : \; u^{(4)}(\widetilde{x}) \neq 0.$$

Wir zeigen: Dann können wir die Funktion u so modifizieren mit Hilfe von $g_{\epsilon,\widetilde{x}}$, dass die Modifikation eine kleinere Halbnorm hat, aber trotzdem noch in V liegt.
Sei also $\epsilon > 0$ so klein, dass die ϵ–Umgebung von $\widetilde{x}$ noch ganz in (s_i, s_{i+1}) liegt, und so klein, dass $u^{(4)}$ sein Vorzeichen in der ϵ–Umgebung nicht ändert ($u^{(4)}$ ist stetig auf (s_i, s_{i+1})!). Wir setzen

$$v = u + g_{\epsilon,\widetilde{x}}.$$

Dann ist wegen $u \in V$ auch $v \in V$. Es gilt mit partieller Integration

$$\begin{aligned} 0 &= (u, v - u) \\ &= \int_{\widetilde{x}-\epsilon}^{\widetilde{x}+\epsilon} u'' g''_{\epsilon,\widetilde{x}} \mathrm{dx} \\ &= -\int_{\widetilde{x}-\epsilon}^{\widetilde{x}+\epsilon} u^{(3)} g'_{\epsilon,\widetilde{x}} \mathrm{dx} \\ &= \int_{\widetilde{x}-\epsilon}^{\widetilde{x}+\epsilon} u^{(4)} g_{\epsilon,\widetilde{x}} \mathrm{dx}. \end{aligned}$$

Die Randterme fallen dabei jeweils weg, weil $g_{\epsilon,\widetilde{x}}$ mit allen seinen Ableitungen bei $\widetilde{x} - \epsilon$ und $\widetilde{x} + \epsilon$ verschwindet.
Nun ändert aber $u^{(4)}$ sein Vorzeichen nicht in $(\widetilde{x} - \epsilon, \widetilde{x} + \epsilon)$, und $g_{\epsilon,\widetilde{x}}$ ist dort positiv. Das Integral kann also nicht verschwinden, die Annahme war falsch, es gilt $u^{(4)} = 0$ auf (s_i, s_{i+1}), und damit ist $u \in \mathcal{P}_3$ auf diesem Intervall. □

```
function b = splineinterpol4
%SPLINEINTERPOL natural cubic splines
K=4;  %Splinegrad
N=8; %Stuetzstellen
M=N+K-1;
P=100;
```

Listing 2.30: Interpolation mit B–Splines (interpolation/splineinterpol4.m)

Klicken für den Quellcode von interpolation/splineinterpol4.m

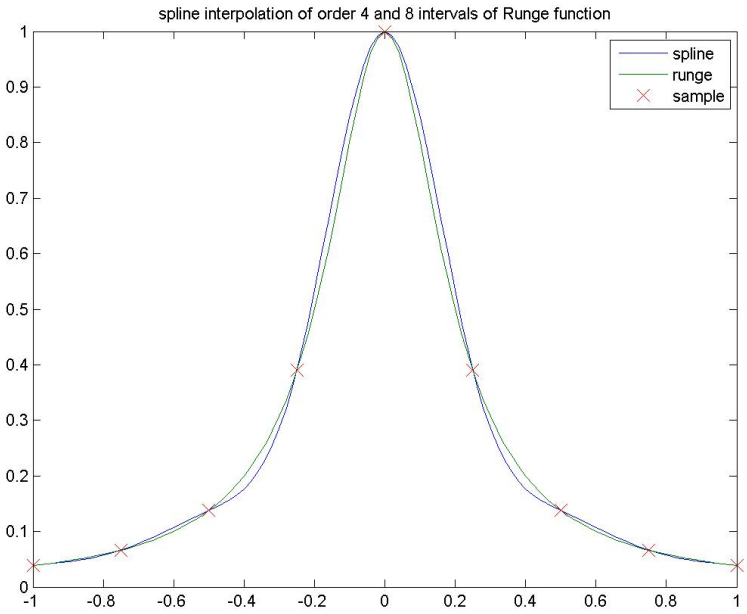


Abbildung 2.23: Spline–Interpolation der Runge–Funktion

Klick für Bild splineinterpol

2.9.2 Spline–Interpolation in höheren Dimensionen

Falls die Auswertepunkte in höheren Dimensionen auf einem rechteckigen Gitter liegen, kann man in so vorgehen wie bei der Fouriertransformation: Für ein Bild, das auf eine höhere Auflösung hochinterpoliert werden soll, wird erst entlang der Zeilen, dann entlang der Spalten mit Splines interpoliert. Dieses Vorgehen führt nicht zum Erfolg, falls die Auswertepunkte unregelmäßig verteilt sind. In diesem Fall benötigt man echte Splines in höheren Dimensionen. In der Numerik der partiellen Differentialgleichungen wird genau dies benutzt, wir geben nur einen ganz kurzen, informellen Ausblick.

Eine natürliche Erweiterung von Splines in höhere Dimensionen sind Finite Elemente. Wir betrachten ein Beispiel im $\mathbb{R}^2$. Es sei eine Funktion auf einem Gebiet $G \subset \mathbb{R}^2$ zu approximieren. Hierzu teilen wir G zunächst in endlich viele Teilgebiete G_k auf. Für jedes Teilgebiet wählen wir einen linearen Raum von Ansatzfunktionen V_k, etwa lineare Funktionen in den Raumkoordinaten x_1, x_2. Wir definieren Funktionen f auf G mit $f|G_k \in V_k \, \forall k$. Damit wäre allerdings nicht einmal die Stetigkeit gesichert. Hierzu fordern wir noch zusätzliche lineare Bedingungen, die die Glattheit sicherstellen.

Beispiel 2.49 Lineare Dreiecke
Sei $G = \bigcup_k \overline{G}_k$, G_k jeweils das Innere eines Dreiecks, und keine Ecke eines Dreiecks falle auf eine Seite eines anderen Dreiecks. Als Ansatzraum auf jedem G_k wählen wir die linearen Funktionen.
Sei f eine Funktion auf $\bigcup_k G_k$, die linear ist auf jedem G_k. Für alle Ecken P, an denen mehrere Dreiecke zusammenstoßen, gelte, dass die Fortsetzungen von $f|_{G_k}$ in P für alle Dreiecke denselben Wert hat. Dann hat f eine stetige Fortsetzung. f heißt Finite–Elemente–Funktion für lineare Dreiecke.

Beweis: *Zu zeigen ist: Die Fortsetzung von f ist entlang aller Kanten stetig. Sei K eine Kante, die durch P und Q begrenzt ist und die Dreiecke G_k und G_j trennt. $f|G_k$ ist linear, also durch die Werte $f(P)$ und $f(Q)$ bereits eindeutig bestimmt. Gleiches gilt aber auf für $f|G_j$, also sind die Fortsetzungen von beiden Seiten gleich.* □

Die Finite–Elemente–Funktionen können wir nun zur Approximation einer Funktion g auf G nutzen. Seien P_k alle Ecken. Dann gibt es genau eine Finite–Elemente–Funktion f mit $f(P_k) = g(P_k)$.

2.10 Approximation und Ausgleichspolynome

In vielen Situationen ist es günstiger, statt einer Interpolation eine Approximation zu wählen, bei der wir die Bedingung fallen lassen, dass die resultierende Funktion exakt durch die vorgegebenen Punkte gehen muss. Dies ist die Methode der Ausgleichspolynome: Wir suchen ein Polynom in $\mathcal{P}_M$ mit $p(x_k) = y_k$, $k = 0 \dots N$, mit $M < N$. Diese Aufgabe ist so offensichtlich im allgemeinen nicht lösbar, mit der Methode der kleinsten Quadrate lassen sich aber optimale Näherungen finden (siehe Numerische Lineare Algebra), siehe Beispiel für $N = 2M$ und das Rungebeispiel.

```
function ausgleichspol( N,M )
%AUSGLEICHSPOL
if (nargin<1)
    N=100;
end
close all;
```

Listing 2.31: Approximation durch Ausgleichspolynome (interpolation/ausgleichspol.m)

Klicken für den Quellcode von interpolation/ausgleichspol.m

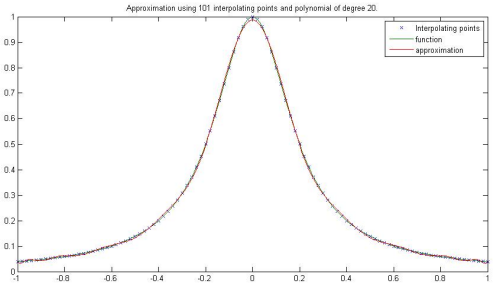


Abbildung 2.24: Approximation durch Ausgleichspolynome

Klick für Bild ausgleichspol

Alternativ können Methoden der Funktionsapproximation gewählt werden. Falls die komplette Funktion f bekannt ist, definieren wir als die Bestapproximation von f bezüglich der ∞–Norm in $\mathcal{P}_N$ das Polynom p_N in $\mathcal{P}_N$, für das $||f - p||_\infty$ minimal ist. Der Satz von Weierstrass garantiert die punktweise Konvergenz von p_N gegen f.

```
> with(numapprox);

  [chebdeg, chebmult, chebpade, chebsort, chebyshev, confracform,

        hermite_pade, hornerform, infnorm, laurent, minimax, pade,
```

Listing 2.32: Bestapproximation in Maple (interpolation/minimax.m)

Klicken für den Quellcode von interpolation/minimax.m

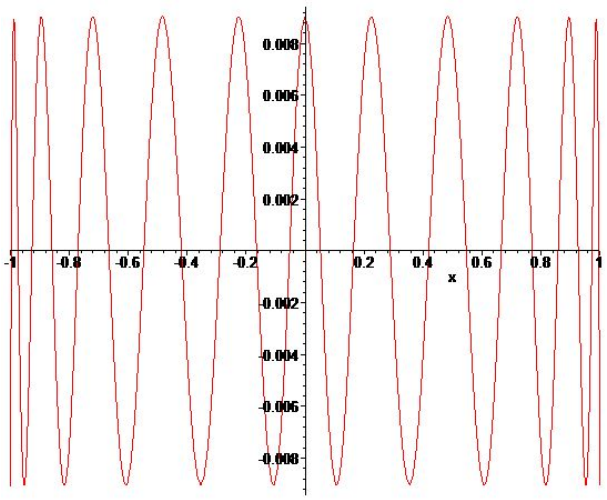


Abbildung 2.25: Fehler der Bestapproximation des Rungebeispiels, berechnet mit Maple

Klick für Bild minimax

```
function y1=matlabminmax(n)
%MATLABMINMAX
%Achtung: Ab Matlab R2009 benoetigt dieses
%Programm die maple-Toolbox
%(wird beim Installieren von Maple nach Matlab
% gleich mitinstalliert)
```

Listing 2.33: Bestapproximation in Matlab (interpolation/matlabminmax.m)

Klicken für den Quellcode von interpolation/matlabminmax.m

Diese Methode wird ausführlich in der Vorlesung Numerische Lineare Algebra behandelt.

Zum Abschluss noch ein Hinweis auf eine Methode, die wir nicht behandelt haben: In unserem Sinn ist der Bèzier–Spline der Computergrafik kein Spline, denn die dort benutzten Kontrollpunkte fordern keine Interpolation, sondern nur eine Approximation.

Literaturverzeichnis

Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables (is an Outgrowth of a Conference on Mathematical Tables Held at Cambridge, Mass., on 1954)*. Applied mathematics series. Dover Publ., 1965. ISBN 9780486612720. URL http://people.math.sfu.ca/~cbm/aands/abramowitz_and_stegun.pdf.

H.W. Alt. *Lineare Funktionalanalysis:*. Springer London, Limited, 2007. ISBN 9783540341871. URL http://books.google.de/books?id=TzeMiSx8_l4C.

Thomas Beth. *Verfahren der schnellen Fourier-Transformation: die allgemeine diskrete Fourier-Transformation–ihre algebraische Beschreibung, Komplexität und Implementierung*. Teubner-Studienbücher. Philologie. Teubner, 1984. ISBN 9783519023630. URL http://books.google.de/books?id=hk7vAAAAMAAJ.

Fischer Black and Myron S Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–54, May-June 1973. URL http://www.jstor.org/stable/1831029.

C. Sidney Burrus. *Fast Fourier Transforms*. Rice University. URL http://cnx.org/content/col10550/latest/.

Ingrid Daubechies et al. *Ten lectures on wavelets*, volume 61. SIAM, 1992.

Carl De Boor. Total positivity of the spline collocation matrix. *Indiana Univ. Math. J*, 25(6):541–551, 1976. URL http://www.math.tamu.edu/~rdevore/publications/47.pdf.

Carl de Boor. *A Practical Guide to Splines*. Number Bd. 27 in Applied Mathematical Sciences. Springer, 2001. ISBN 9780387953663. URL http://books.google.de/books?id=m0QDJvBI_ecC.

Ronald A. DeVore and Amos Ron. Developing a Computation-Friendly Mathematical Foundation for Spline Functions. *SIAM News*, 38(4):1–2, May 2005. URL http://www.siam.org/pdf/news/56.pdf.

L.C. Evans. *Partial Differential Equations*. Graduate Studies in Mathematics. American Mathematical Society, 2010. ISBN 9780821849743. URL http://books.google.de/books?id=Xnu0o_EJrCQC.

R.W. Freund and R.H.W. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. Springer-Lehrbuch. Springer London, Limited, 2007. ISBN 9783540453901. URL http://link.springer.com/book/10.1007/978-3-540-45390-1/page/1.

E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Solving Ordinary Differential Equations. Springer, 1993. ISBN 9783540566700. URL http://books.google.de/books?id=F93u7VcSRyYC.

M. Hanke-Bourgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Mathematische Leitfäden. Teubner, 2006. ISBN 9783835100909. URL http://books.google.de/books?id=tKrhTUmYNEoC.

P. Henrici. *Discrete variable methods in ordinary differential equations*. Wiley, 1962. URL http://books.google.de/books?id=j-5QAAAAMAAJ.

P. Henrici. *Elements of numerical analysis*. Wiley international edition. Wiley, 1964. URL http://ia700700.us.archive.org/23/items/ElementsOfNumericalAnalysis/Henrici-ElementsOfNumericalAnalysis.pdf.

Alfred K. Louis, Peter Maaß, and Andreas Rieder. *Wavelets:*. Teubner Studienbücher: Mathematik. Vieweg+Teubner Verlag, 1998. ISBN 9783519120940. URL http://books.google.de/books?id=QN2fg4sM5oIC.

Alfred Marshall Mayer. *Researches in Acoustics*, volume 151. 1896. URL http://www.ajsonline.org/content/s4-1/2/81.full.pdf+html?sid=9ff838fa-4010-4f5c-ad78-88212b258bf0.

J.D. Murray. *Mathematical Biology: I. An Introduction*. Interdisciplinary Applied Mathematics. Springer, 2002. ISBN 9780387952239. URL http://books.google.de/books?id=4WbpP90Gk1YC.

Frank Natterer. *Vorlesungsskript zur Vorlesung Effiziente Algorithmen im WS 94/95*. Institut für Numerische und Angewandte Mathematik der Universität Münster, 1994. URL http://wwwmath.uni-muenster.de/num/Vorlesungen/EffAlg_WS94/.

J.W. Prüß, R. Schnaubelt, and R. Zacher. *Mathematische Modelle in der Biologie: Deterministische homogene Systeme*. Mathematik Kompakt. Birkhäuser Basel, 2008. ISBN 9783764384364. URL http://books.google.de/books?id=IRH_k7vukdsC.

Carl Runge and Hermann König. *Vorlesungen über numerisches Rechnen*. Springer Göttingen, 1925. URL http://resolver.sub.uni-goettingen.de/purl?PPN373207646.

Isaac Jacob Schoenberg and Anne Whitney. On polya frequency function. iii. the positivity of translation determinants with an application to the interpolation problem by spline curves. *Transactions of the American Mathematical Society*, 74 (2):pp. 246–259, 1953. ISSN 00029947. URL http://www.jstor.org/stable/1990881.

J. Stoer and R. Bulirsch. *Numerische Mathematik 2: Eine Einführung - unter Berücksichtigung von Vorlesungen von F.L.Bauer*. Numerische Mathematik: eine Einführung - unter Berücksichtigung von Vorlesungen von F. L. Bauer. Springer, 2005. ISBN 9783540237778. URL http://books.google.de/books?id=_TPRZ9pabGcC.

Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Software, Environments and Tools Series. Cambridge University Press, 2000. ISBN 9780898714654. URL http://books.google.de/books?id=pB4xiZKZ4ecC.

W. Walter. *Gewöhnliche Differentialgleichungen: Eine Einführung*. Springer-Lehrbuch Series. Springer Singapore Pte. Limited, 2000. ISBN 9783540676423. URL http://books.google.de/books?id=tyAdMH69NRYC.

Shmuel Winograd. On computing the discrete fourier transform. *Mathematics of Computation*, 32(141):pp. 175–199, 1978. ISSN 00255718. URL http://www.jstor.org/stable/2006266.

Abbildungsverzeichnis

Listings