Interpolation in geographischen Informationssystemen

Mathematischer Hintergrund

Zuerst gehalten im Wintersemester 2002/2003

Tomas Sauer

Version 0.0
Letzte Änderung: 2.12.2002

Statt einer Leerseite ...

Die Mathematik ist die Wissenschaft der Gestalt und der Menge. Mathematische Überlegungen sind nur auf die Beobachtung von Gestalt und Menge angewandte Logik. Der große Irrtum liegt in der Annahme, daß eben die Wahrheiten, die man "reine" Algebra nennt, abstrakte oder allgemeine Wahrheiten sind.

E. A. Poe, *Der entwendete Brief*

Die simple Schreibart ist schon deswegen zu empfehlen, weil kein rechtschaffener Mann an seinen Ausdrücken künstelt und klügelt.

G. Chr. Lichtenberg

1 Einführung

Unter *Interpolationsmethoden* versteht man mathematische Verfahren, die versuchen, aus der Kenntnis der Werte einer unbekannten Funktion auf den Wert dieser Funktion an einer anderen Stelle zu schließen. Der Begriff selbst wurde 1655 von Wallis eingeführt (siehe [2, 4]).

1.1 Ein Beispiel

Tatsächlich diente Interpolation zu dieser Zeit im wesentlichen zur Vevollständigung von Tabellen, beispielsweise für die Logarithmusfunktion; die Werte dieser Funktion an bestimmten Stellen lagen in gedruckter Version vor[1], wer aber den Wert an einer anderen Stelle wissen wollte, der musst sich anders behelfen:

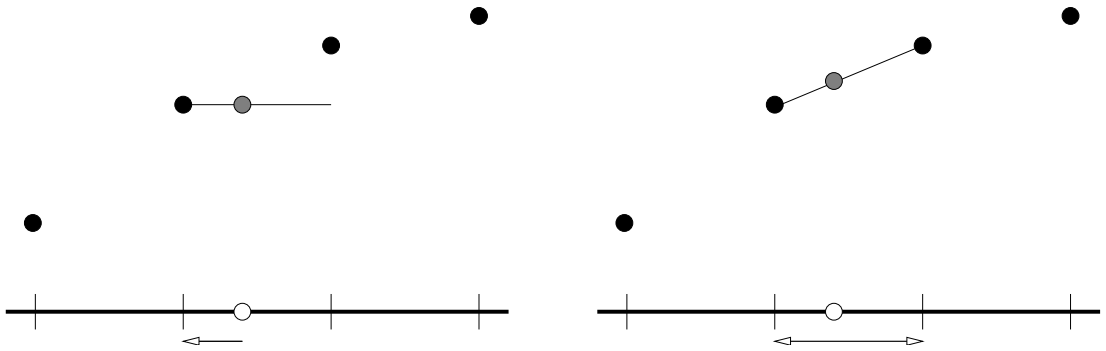

Abbildung 1.1: Zwei Möglichkeiten, einen Zwischenwert zu interpolieren, indem man entweder den Wert an der nächstgelegenen Stelle wiederholt (links), oder *linear interpoliert* (rechts).

- Die einfachste Idee, wäre es sicherlich, den Wert an der *nächstgelegenen* Position zu verwenden. Abgesehen davon, daß dies schwierig wird, wenn man sich den Punkt

[1] Die bis vor nicht allzu langer Zeit, genauer, bis zum Aufkommen der Taschenrechner, recht gebräuchlichen Logarithmentafeln

genau in der Mitte zwischen zwei "Abtaststellen" ansehen will, wird das Ergenis Sprünge aufweisen.

- Um das zu vermeiden kann man je zwei benachbarte Werte durch ein Linienstück verbinden und den Wert dieses Streckenzugs verwenden. Dies ist die einfachste Methode, ein Ergebnis ohne Sprünge aber immer noch mit "Knicken" zu bekommen.

- Und noch etwas "besser"[2]: wir legen durch die zwei linken und den rechten Randpunkt, oder, wenn man so will, durch die beiden benachbarten Punkte und dann den nächsten, einen Kreisbogen und verwendet dessen Wert, siehe Abb. 1.2.

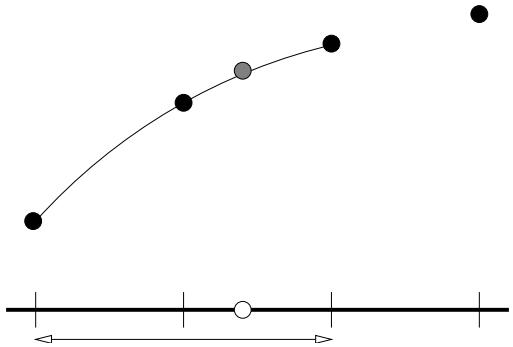

Abbildung 1.2: Schon etwas komplizierter: Wir verwenden den Wert des Kreisbogens durch drei Punkte (rechts)..

Damit war man zu Tabellenzeiten auch meistens schon am Ende: Wer eine Tabelle verwendete, der wollte ja gerade die Werte *ohne* große Rechnerei erhalten können.

Daß diese Verfahren von ganz unterschiedlicher Qualität sind, erkannt man, wenn wir beispielsweise aus der Kenntnis des (natürlichen) Logarithmus an den Stellen 1.1, 1.2 und 1.3 den Logarithmus von 1.23 bestimmen wollen. Die Werte sind laut `Matlab`

x	1.1	1.2	1.3
$\log x$	`0.095310`	`0.18232`	`0.26236`

Die erste Näherung für $\log 1.23$ ist nun einfach der Wert an der Stelle 1.2, die zweite Näherung ergibt sich als

$$\frac{1.3-1.23}{1.3-1.2}\log 1.2+\frac{1.23-1.2}{1.3-1.2}\log 1.3=0.23835$$

und für die dritte Näherung erhalten wir mit Methoden, die wir gleich noch kennenlernen werden, den Wert 0.20707. Und was ist nun am besten?

[2]Zumindest einmal komplizierter!

	Methode 1	Methode 2	Methode 3	Exakt
Wert	0.18232	0.20633	0.20707	0.20701
Fehler	-11.9 %	-0.3 %	+.02 %	

Das ist kein Wunder: Methoden *hoher Ordnung* sind für glatte Funktionen, und zu diesen gehört der Logarithmus, einfach genauer.

1.2 Problemstellung

Ein wichtiger Grundsatz der Mathematik besteht darin, erst einmal genau zu sagen, was man zu tun beabsichtigt, indem man das Problem und seine Ziele genau definiert.

Also gut, wir nehmen an, es wären *endlich viele* Punkte $x_1, \ldots, x_N$ vorgegeben, zu denen wir Meßwerte einer ansonsten unbekannten Funktion f kennen, die wir mit $f_1, \ldots, f_n$ bezeichnen wollen, wobei

$$f_j = f(x_j), \qquad j = 1, \ldots, N$$

ist. Unser Ziel ist es, aus diesen Daten eine Funktion ϕ zu konstruieren, die f "fortsetzt" und die uns eine Auswertung an beliebigen Punkten gestattet. Dazu ein paar Bemerkungen:

1. Da die Werte f_j alles sind, was wir über f wissen, ist die Versuchung natürlich groß, diese Information so vollständig und so restriktiv wie möglich zu verwenden, das heißt, wir definieren ϕ als *interpolierende* Funktion:

   $$\phi(x_j) = f_j = f(x_j), \qquad j = 1, \ldots, N.$$

   Manchmal ist es aber auch sinnvoll, zu fordern, daß ϕ nur im Mittel möglichst wenig abweicht, also z.B.

   $$\sum_{j=1}^{N} |f(x_j) - \phi(x_j)|^2 = \min,$$

   wodurch "Ausreisser" bei schlampig ermittelten Daten "geglättet" werden können.

2. Selbst wenn wir annehmen, daß ϕ interpolieren soll, ist ϕ natürlich in keinster Weise durch diese Bedingungen bestimmt – man denke nur an den Fall $N = 1$, den unsere Theorie ja auch abdecken sollte.

3. Diese Freiheitsgrade kann man nutzen, indem man sich klar wird, wofür man die Funktion ϕ eigentlich braucht: Die Berechnung von Höhenlinien stellt beispielsweise ganz andere Anforderungen an ϕ als ein schnelles *Rendering*, also die dreidimensionale Darstellung auf dem Bildschirm.

4. Information über die Funktion f kann nie schaden. Je mehr man über die *Natur* der Funktion[3] weiss, desto angepasster kann man das Interpolationsverfahren wählen.

[3] In der geographischen Realität also beispielsweise das Wissen, ob es sich um Höhendaten, Bevölkerungsdichte oder Niederschlagsmengen handelt.

Und selbst wenn wir den "einfachsten" Weg gehen und verlangen, daß die Funktion ϕ durch Interpolation der Werte f_j an den Stellen x_j definiert ist, stehen wir immer noch vor einer ganzen Menge von Fragen:

- **Wie** können wir konkret (algorithmisch) diese ominöse Funktion ϕ bestimmen?
- Interpoliert diese Funktion auch wirklich. Bei numerischer Rechnung am Computer treten ja normalerweise die allgegenwärtigen *Rundungsfehler* auf, die für den Übergang
$$\phi(x_j) = f_j \qquad \rightarrow \qquad \phi(x_j) \sim f_j$$
sorgen. Und das "$\sim$" hat mit "$=$" in der Realität oft genug absolut nichts mehr zu tun.
- Wie gut "reproduziert" die Interpolationsfunktion eine unbekannte Funktion. Genauer: Wenn die Werte f_j tatsächlich $f(x_j)$ sind, kann man dann irgendwelche Vorhersagen der Form $\phi(x) \sim f(x)$ für beliebige Punkte x machen?
- Man kann sich leicht vorstellen, daß diese "globalen" Vorhersagen sowohl vom Interpolationsverfahren[4] als auch von bestimmten Eigenschaften der Funktion f abhängt. Diese Zusammenhänge sind aber entscheidend, um zu einem gegebenen Problem das "richtige" Interpolationsverfahren auswählen zu können.

Um diese Fragen richtig angehen zu können ist es aber leider unvermeidlich, sich ein bißchen auch mit dem mathematischen Hintergrund der verwendeten Verfahren auseinanderzusetzen. Aber keine Angst: so schlimm wird's auch wieder nicht ...

[4] Also der Frage, was für eine Funktion ϕ man da so produziert.

Labor est materia virtutis et gloriae:
hunc qui reiicit, illas reiicit.
Die Arbeit ist das Material für
Tugend und Ruhm: ihr absagen heißt
ihnen absagen.

Petrarca, *"De otio et quiete"*, 1366

2 Interpolation mit linearen Räumen

In diesem Abschnitt sehen wir uns mal in ganz "allgemeiner" Theorie den generellen Ansatz zur Interpolation an. Diese Verallgemeinerung ist nicht unbedingt mathematischer Abstraktionswahn, sondern soll das Gemeinsame der gebräuchlichsten Interpolationsverfahren herausarbeiten, so daß man die Ideen nur einmal zu verstehen braucht, um sie für eine Vielzahl von Methoden einsetzen zu können.

2.1 Was sind lineare Räume und wofür sind die gut

Ein *linearer Raum* oder *Vektorraum* ist, abstrakt gesagt, nur eine Menge von Objekten, die man addieren und mit beliebigen Zahlen multiplizieren darf, ohne aus dem Raum "herauszufallen". Etwas formaler:

> Eine Menge $\mathcal{F}$ heißt *linearer (Funktionen-) Raum*, wenn mit f, f' auch die Funktion $\alpha\, f + \beta\, f'$ existiert und zu $\mathcal{F}$ gehört, wobei $\alpha, \beta \in \mathbb{R}$ beliebige Zahlen sind.

Lineare Räume besitzen eine *Dimension* – das ist die Menge an Information, die man benötigt, um ein beliebiges Element in diesem linearen Raum darzustellen. Interessant für uns sind natürlich *endlichdimensionale* lineare Räume, denn endlich viel Information gibt die Hoffnung, die Sache praktisch und wenn möglich auf dem Rechner zu behandeln.

Wie sieht das nun mit der "endlichen Information" aus? Ganz einfach: $\mathcal{F}$ hat genau dann *Dimension* N wenn es eine *Basis* $\Phi = \{\phi_1, \ldots, \phi_n\}$ von $\mathcal{F}$ gibt, so daß man jede Funktion $\phi \in \mathcal{F}$ *eindeutig* als

$$\phi = \sum_{j=1}^{N} a_j\, \phi_j, \qquad a_1, \ldots, a_N \in \mathbb{R},$$

darstellen kann. Rekapitulieren wir noch mal: Die Eigenschaft "linearer Raum" garantiert uns, daß die obige Summe wieder zu $\mathcal{F}$ gehört, die Basiseigenschaft hingegen sorgt dafür, daß wir, um eine beliebige Funktion $\phi \in \mathcal{F}$ darzustellen nur noch die *Parameter* $a_1, \ldots, a_N$ variieren müssen – und die sind einfach Zahlen, keine Funktionen mehr!

Beispiel 2.1 *(Lineare Räume)*

1. *Der Klassiker der linearen Räume sind sicherlich die guten alten* Polynome. *Ein Polynom ist eine Funktion der Form*

$$p(x) = \sum_{j=0}^{n} a_j \, x^j.$$

Die Menge aller *Polynome ist ein unendlichdimensionaler Raum (hier darf der* Grad *n beliebig wachsen), die Menge aller Polynome vom Grad* $\leq N$ *hingegen bilden einen linearen Raum der Dimension* N *– jedes solche Polynom lässt sich eindeutig durch die Koeffizienten* $a_0, \ldots, a_N$ *beschreiben.*

Auch eine Basis ist schnell angegeben: die Polynome $\phi_j(x) = x^j$. *Doch das ist natürlich nicht alles, eine andere, genauso sinnvolle Basis wäre beispielsweise*

$$\phi_j(x) = (x - x_{j-1}) \cdots (x - x_0) \, , x_0, \ldots, x_{j-1} \in \mathbb{R}.$$

Wir sehen also: Es gibt nicht ***die*** *Basis, es gibt nur* ***eine*** *Basis.*

2. *Eine sehr angenehme Praxis besteht darin, einen linearen Raum einfach dadurch vorzugeben, daß man einfach die Elemente einer Basis auflistet. So könnten wir beispielsweise*

$$\Phi = \{\sin x, \sin 2x, \sin 3x, \ldots, \sin Nx\}$$

wählen.

3. *Eine etwas interessantere Wahl, die dafür in* beliebigen *Dimensionen funktioniert, ist die folgende. Es sei* $|\cdot|$ *ein Maß für den Abstand zwischen zwei Punkten und es seien Punkte* $\xi_1, \ldots, \xi_N$ *vorgegeben*[5]. *Dann sind für eine beliebige Funktion* ψ *die Funktionen*

$$\phi_j(x) = \phi\left(|x - \xi_j|\right), \qquad j = 1, \ldots, N,$$

immer auf solchen Kurven konstant, die vom Center ξ *denselben Abstand haben – deswegen bezeichnet man diese Funktionen als* radial. *Eine beliebige Kombination solcher Funktionen hat dann die Form*

$$\phi(x) = \sum_{j=1}^{N} a_j \, \phi_j(x) = \sum_{j=1}^{N} a_j \, \phi\left(|x - \xi_j|\right).$$

Im Prinzip haben wir nun eine ganze Menge Parameter, an denen wir drehen können:

- *Die Funktion* ψ*, die ja auch wieder aus einem linearen Raum kommen und dort durch passende Parameter modelliert werden kann.*
- *Die Zentren* ξ_j*,* $j = 1, \ldots, N$.

[5]Diese Punkte könnten natürlich die Interpolationspunkte $x_1, \ldots, x_N$ sein, die wir immer im Hinterkopf haben wollen, müssen es aber nicht. Daher das andere Symbol.

- *Die Koeffizienten a_j, $j = 1, \ldots, N$.*

Aber Vorsicht: Linearität ist nur bezüglich der Koeffizienten vorhanden, die anderen Parameter, insbesondere die Zentren, tragen normalerweise auf nichtlineare *Weise bei!*

2.2 Interpolation und Gleichungssysteme

Daß man so begeistert mit linearen Räumen interpoliert, hat einen sehr einfachen Grund – das Interpolationsproblem lässt sich sehr einfach auf etwas zurückführen, das man recht gut behandeln kann[6], nämlich auf lineare Gleichungssysteme. Nehmen wir also jetzt an, wir wollen das Interpolationsproblem

$$f(x_j) = \phi(x_j), \qquad j = 1, \ldots, N, \tag{2.1}$$

mit einer Funktion ϕ aus dem M–dimensionalen linearen Raum $\mathcal{F}$ lösen. Nach dem, was wir im vorherigen Abschnitt gelernt haben, ist

$$\phi = \sum_{k=1}^{m} a_k \phi_k, \qquad a_k \in \mathbb{R}, \tag{2.2}$$

wobei die ϕ_j, $j = 1, \ldots, M$, die vielbeschworene *Basis* von $\mathcal{F}$ bilden. Tja, dann setzen wir eben mal (2.2) in (2.1) ein und erhalten so, daß

$$f(x_j) = \phi(x_j) = \sum_{k=1}^{m} a_k \phi_k(x_j), \qquad j = 1, \ldots, N.$$

Stapeln wir diese Gleichungen übereinander, so erhalten wir, daß

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \ldots & \phi_M(x_1) \\ \vdots & \ddots & \vdots \\ \phi_1(x_N) & \ldots & \phi_M(x_N) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_M \end{bmatrix}, \tag{2.3}$$

in Kurzform

$$\boldsymbol{f} = \boldsymbol{F}\,\boldsymbol{a}, \qquad \boldsymbol{f} \in \mathbb{R}^N, \quad \boldsymbol{F} \in \mathbb{R}^{N \times M}, \quad \boldsymbol{a} \in \mathbb{R}^M; \tag{2.4}$$

soviel für all diejenigen, die sich schon immer gefragt haben, wofür lineare Gleichungssysteme denn so gut sein sollen. Also:

> Kennen wir eine Basis Φ, so müssen wir "nur" die *Kollokationsmatrix*
>
> $$\boldsymbol{F} = [\phi_k(x_j) \ : \ j, k = 1, \ldots, N]$$
>
> aufstellen.

[6] Denkt man zumindest!

Aber Achtung: j ist hier der *Zeilenindex* und k der *Spaltenindex* – man sollte die tunlichst **nicht** durcheinanderbringen!
Die Frage nach der Lösbarkeit des Interpolationsproblems ist dann nichts anderes als die Frage nach der Lösbarkeit des linearen Gleichungssystems und das "entscheiden" die Lösungsverfahren dann gleich mehr order weniger mit.

Jetzt wird es aber Zeit für Beispiele und die Frage, wie wir mit einer geeigneten Basis einfach mal herumspielen können. Dazu verwenden wir das "Public–Domain"–Programm `Octave`[7], mit dem man interaktiv Mathematik in *numerischer Rechnung*[8] betreiben kann. Die Methode ist vergleichsweise einfach: Zuerst definieren wir uns einen Spaltenvektor von Interpolationspunkten

```
octave> X = (1:3)'
X =

  1
  2
  3
```

hier also den Vektor $[1, 2, 3]^T$, und dann basteln wir uns mit drei Basisfunktionen, sagen wir mal mit $\sin x, \sin 2x, \sin 3x$, die Kollokationsmatrix, indem wir uns erst eine 3×3–Matrix holen, die mit Nullen vorbelegt ist[9],

```
octave> F = zeros(3,3);
```

und deren Spalten dann passend eintragen:

```
octave> for j=1:3 F( :,j ) = sin( j*X ); end
```

Die Matrix $\boldsymbol{F}$ schauen wir uns dann schließlich noch an:

```
octave> F
F =

   0.84147   0.90930   0.14112
   0.90930  -0.75680  -0.27942
   0.14112  -0.27942   0.41212
```

Fein, nun möchten wir die Werte $0, 1, 2$ an den drei Interpolationspunkten vorschreiben, die rechte Seite unseres Gleichungssystems ist also der Spaltenvektor

```
octave> f = [ 0;1;2 ]
f =
```

[7] Siehe `www.octave.org` für Downloads, Dokumentation und Erweiterungen, wer zu viel Geld hat, kann auch gerne das Programm `Matlab` käuflich erwerben.

[8] Im Gegensatz zur *exakten* Arithmetik, die Programme wir `Maple`, `Mathematica` oder `MuPAD` bieten.

[9] Das Semikolon am Ende des Befehls unterdrückt lediglich die Ausgabe.

```
0
1
2
```

und wir erhalten unseren Koeffizientenvektor $\boldsymbol{a}$ als

```
octave> a = F\f
a =

   0.97682
  -1.45239
   3.53377
```

Die Funktion sieht man in Abb. 2.1. Wie man sieht interpoliert sie, aber ob das Ergebnis wirklich das ist, was man erwartet hätte, das ist natürlich eine andere Frage.

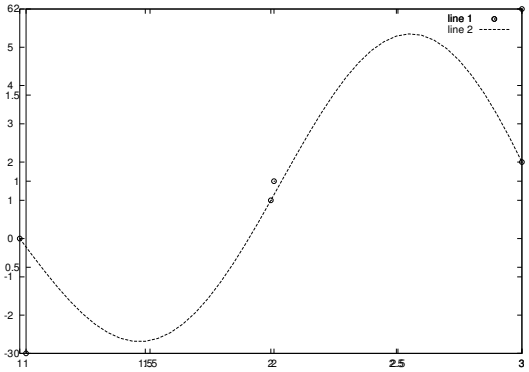

Abbildung 2.1: Interpolation mit $\Phi = \{\sin x, \sin 2x, \sin 3x\}$. Sieht fast aus wie eine einzelne Sinuskurve – aber eben nur fast.

Beispiel 2.2 *Kehren wir jetzt nochmal zu unserem Logarithmus zurück und interpolieren wir mit* $\Phi = \left\{1, x, \ldots, x^k\right\}$. *Den Fall* $k = 2$ *haben wir uns ja schon angesehen, jetzt aber wissen wir sogar, was wir tun:*

```
octave> X = [ 1.1;1.2;1.3 ];
octave> F = zeros( 3,3 ); for j=1:3 F( :,j ) = X.^(j-1); end
octave> a = F\log(X); ( 1.23.^(0:2) )*a
ans = 0.20707
```

Stellt sich natürlich die Frage: Bekommen wir vielleicht sogar noch mehr Genauigkeit, indem wir k *erhöhen? Probieren wir's doch einfach mal mit* $k = 5$ *und den* rechten *Nachbarn:*

```
octave> X = ( 1.2:.1:1.7 )';
octave> F = zeros( 6,6 ); for j=1:6 F( :,j ) = X.^(j-1); end
octave> ( 1.23.^(0:5) )*( F\log(X) )
ans = 0.20701
```

Und die Genauigkeit ist nun wirklich nicht schlecht:

```
octave> ( ans-log(1.23) ) / log(1.23)
ans =  -1.7237e-06
```

Also ist großes k immer besser? Probieren wir es mit $k = 8$, dann erhalten wir sogar

```
octave> X = ( 1.1:.1:1.9 )';
octave> F = zeros( 9,9 ); for j=1:9 F( :,j ) = X.^(j-1); end
octave> ( 1.23.^(0:8) )*( F\log(X) )
ans = 0.20701
octave> ( ans-log(1.23) ) / log(1.23)
ans =  1.5517e-08
```

aber wenn wir uns die linken Nachbarn vornehmen, dann ist die Begeisterung nicht mehr ganz so groß:

```
octave> X = ( .5:.1:1.3 )';
octave> F = zeros( 9,9 ); for j=1:9 F( :,j ) = X.^(j-1); end
octave> ( 1.23.^(0:8) )*( F\log(X) )
ans = 0.20701
octave> ( ans-log(1.23) ) / log(1.23)
ans =  3.1305e-06
```

Irgendwas scheint da zu passieren ...

2.3 Noch mehr Beispiele

Um einmal ein paar Methoden miteinander vergleichen zu können, verwenden wir zwei *Testfunktionen*, einen "Dreieckszacken" namens `Dach`[10] und eine Exponentialfunktion $f(x) = e^{-5x^2}$, die wir `Expo` nennen wollen, siehe Abb. 2.2. Entscheidend ist hierbei, daß eine Funktion sehr glatt ist (`Expo`) während die andere "Zacken" hat (`Dach`). Beide Funktionen wollen wir auf dem Intervall $[-2, 2]$ interpolieren und zwar an dem Vektor aus 11 Punkten bestehenden Vektor `X`, definiert durch

```
octave> X = (-2:.4:2)';
```

Beispiel 2.3 *Beginnen wir mit den bereits bewährten, zumindest aber bekannten Polynomen und machen uns, in altgewohnter Manier frisch ans Werk*[11]*:*

[10] Namen im `Typewriter`–Font sollen immer für Namen eines Octave–Programms stehen.

[11] Wir brauchen unsere Matrizen gar nicht unbedingt mit Nullen zu initialisieren, bei so kleinen Matrizchen macht das keinen so riesigen Unterschied. Eine gute Angewohnheit ist es aber allemal, denn wird `F` größer, dann tut sich `Matlab` oder `Octave` nicht mehr so leicht beim Erweitern – steht auch in den Handbüchern.

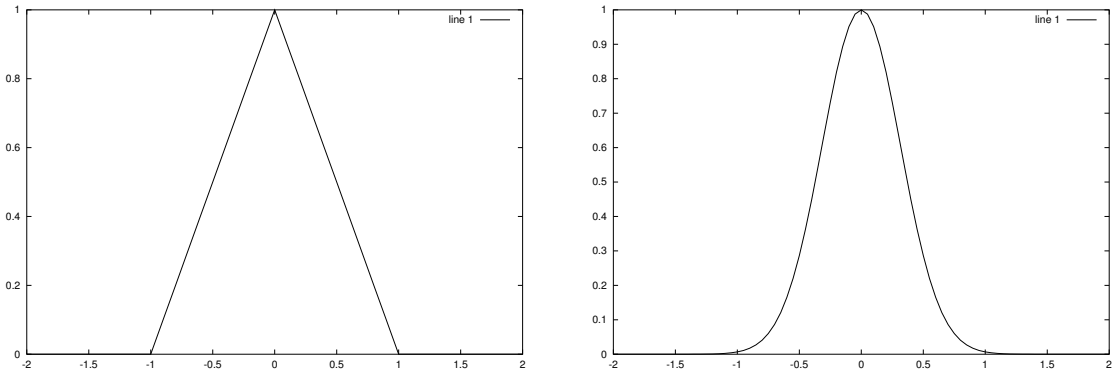

Abbildung 2.2: Die Testfunktionen `Dach` (links) und `Expo` (rechts).

```
octave> for j=1:11 F(:,j) = X.^(j-1); end
octave> aDach = F\Dach(X); aExpo = F\Expo(X);
```

Das nicht sonderlich begeisternde Ergebnis dieser Operation findet sich in Abb. 2.3.

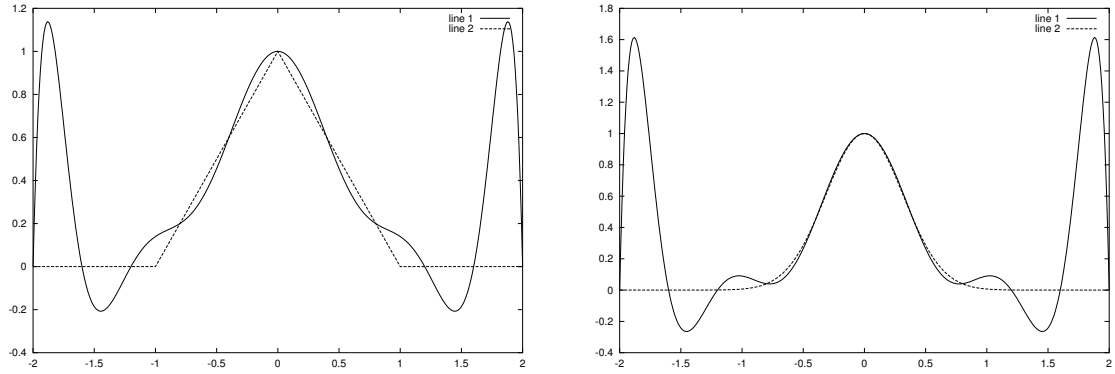

Abbildung 2.3: Interpolation der beiden Funktionen mit Polynomen. Während die Funktion `Expo` wenigstens im Inneren noch halbwegs gut wiedergegeben wird, haben wir mit der Funktion `Dach` wahrlich nichts als Scherereien.

.

Beispiel 2.4 *Probieren wir's doch mal mit Schwingungen, also mit den* trigonometrischen Polynomen

$$\{1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots\}$$

und basteln uns unsere Kollokationsmatrix als

```
octave> F = []; F( :,1 ) = X.^0;
octave> for j=1:5 F( :,2*j ) = sin( j*X ); F( :,2*j+1 ) = cos( j*X ); end
```

Der erste Befehl `F = [];` *"löscht" die Matrix* `F`*. Das Ergebnis dieser Operation findet sich in Abb. 2.4.*

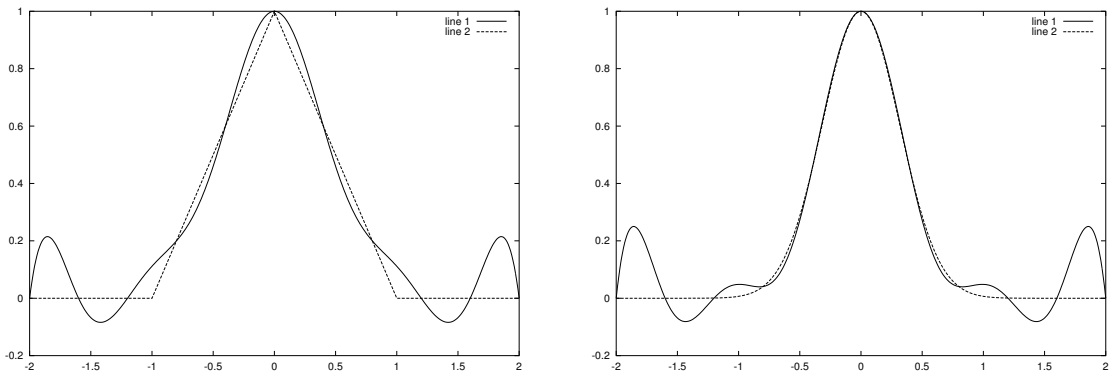

Abbildung 2.4: Hier versuchen wir's zur Abwechslung mal mit trigonometrischen Polynomen.Zumindest schlagen die am Rande nicht so stark aus, dafür sind sie aber im Innren nicht so gut.

Um mal was ganz anderes zu machen, interpolieren wir jetzt mit gestauchten und verschobenen Versionen von `Dach` und `Expo`, genauer, wir setzen

$$\phi_j(x) = \psi\left(3x - (j-6)\right), \qquad j = 1, \ldots, 11, \tag{2.5}$$

wobei ψ eine der beiden Funktionen ist. Wie haben wir uns das nun wieder vorzustellen? Nun, wenn die Funktion ψ in etwa auf $[-1, 1]$ "lebt", dann lebt die Funktion $\psi(3x)$ auf $\left[-\frac{1}{3}, \frac{1}{3}\right]$ und die verschobene Funktion $\psi(3x - k)$ auf $\left[\frac{k}{3} - \frac{1}{3}, \frac{k}{3} + \frac{1}{3}\right]$. Läuft nun j wie in (2.5), dann laufen die Zentren also von $-\frac{5}{3}$ bis $\frac{5}{3}$, was eigentlich ganz gut aussieht.

Beispiel 2.5 *Fangen wir mit der* `Dach`*–Funktion an und versuchen, so* `Dach` *und* `Expo` *zu bekommen. Die alte Prozedur liefert*

```
octave> F = []; for j=1:11 F( :,j ) = Dach( 3*X .- (j-6) ); end
octave> aDach = F\Dach(X); aExpo = F\Expo(X);
warning: matrix singular to machine precision, rcond = 0
warning: matrix singular to machine precision, rcond = 0
```

Huch – eine Warnung! Eigentlich ist es sogar eine ***Fehlermeldung****, denn wenn wir es uns genau überlegen, dann verschwinden die Funktionen, mit denen wir interpolieren alle am rechten und linken Rand – wir müssten den Skalierungsfaktor eigentlich* < 3 *wählen. Und obwohl es mathematisch gar nicht funktionieren dürfte, ist das Ergebnis in diesem Fall sogar sehr gut, siehe Abb. 2.5.*

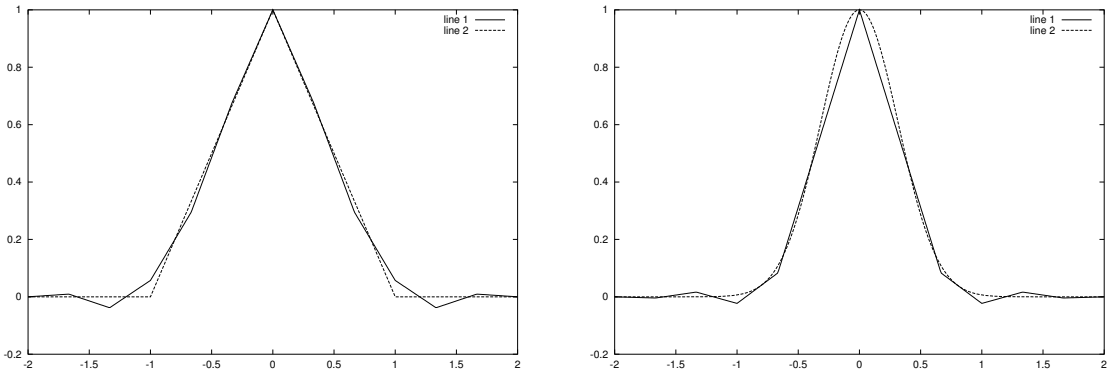

Abbildung 2.5: Interpolation mit verschobenen und gestauchten Funktionen der `Dach`–Funktion. Mit dem runden Übergang rechts gibts natürlich Probleme.

Beispiel 2.6 *Und weil's so schön war machen wir dasselbe Spiel auch mit der Funktion* ***`Expo`****:*

```
octave> F = []; for j=1:11 F( :,j ) = Expo( 3*X .- (j-6) ); end
octave> aDach = F\Dach(X); aExpo = F\Expo(X);
```

Nein, das ist ***kein*** *Versehen – hier gibt's keine Warnungen, denn die Funktion* ***`Expo`*** *hat nie den Wert Null. Und die Bilder? Klar, die finden sich in Abb. 2.6 – sind sie nicht hübsch?*

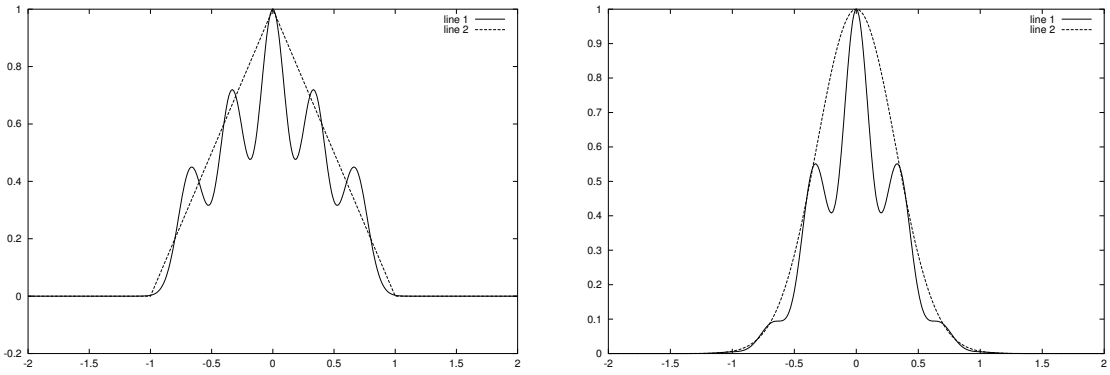

Abbildung 2.6: Hier versuchen wir unser Glück mit der Exponentialfunktion. Ob wir es damit gefunden haben – wer weiss.

Wer nun meint, daß man mit den Translaten der beiden Funktionen immer ein ganz ordentliches Resultat bekommen könnte, der braucht sich nur mal die beiden Interpolanten

zu der Funktion

$$f(x) = 1 + \texttt{Dach}(x)$$

ansehen, die in Abb. 2.7 geplottet sind, um von solch trügerischen Hoffnungen kuriert zu sein. Wählt man allerdings einen Skalierungsfaktor < 3 für die Funktion `Dach`, beispielsweise 2.5, dann sieht die Sache plötzlich richtig gut aus, siehe Abb. 2.8

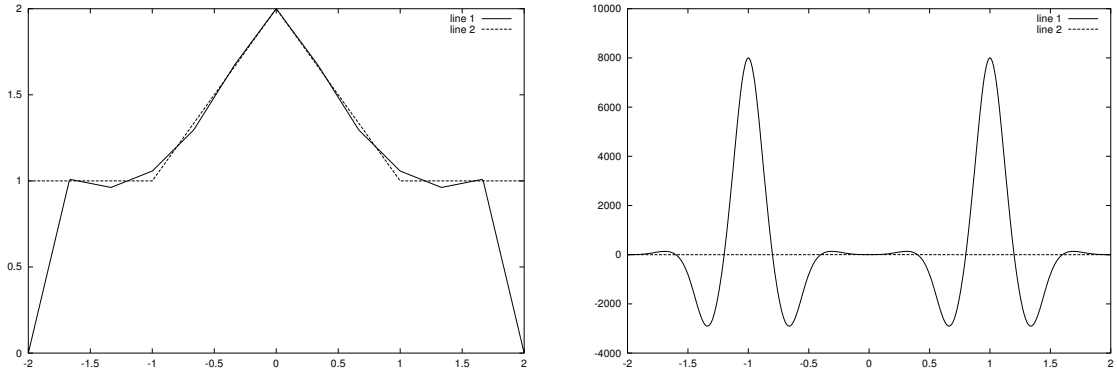

Abbildung 2.7: Was so alles passieren kann, wenn der Wert am Rande nicht Null ist. Man beachte im Übrigen die Skalierung auf der rechten Seite.

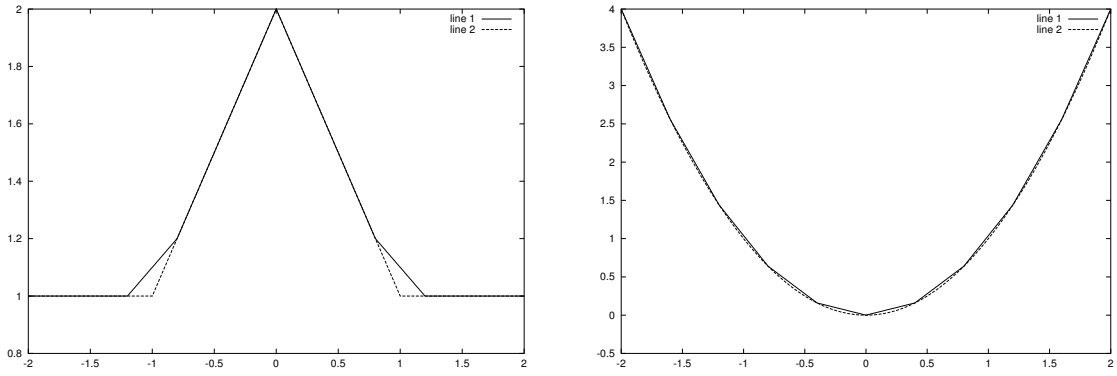

Abbildung 2.8: Kaum hat man den Skalierungsfaktor auf 2.5 heruntergedreht, sieht die Sache schon viel besser aus.

Zum Abschluss aber noch ein Beispiel, das zeigt, daß der Ansatz sehr wohl auch für Flächen funktioniert. Dazu berechnen wir eine "Trendfläche" (also ein Polynom) vom *Totalgrad* 4 zur Funktion

$$f(x,y) = e^{5\left(x^2+y^2\right)} = \texttt{Expo}\left(x^2+y^2\right).$$

Unsere Ansatzfunktionen sind die 21 Polynome

$$\left\{x^j\, y^k \;:\; j+k \le 4\right\} = \left\{1, x, y, x^2, xy, y^2, \ldots, x^4, x^3y, x^2y^2, xy^3, y^4\right\}.$$

In `Octave` sieht das dann wie folgt aus: Zuerst holen wir uns 21 zufällige Punkte

```
octave> X = ( rand( 21,2 ) .- .5 ) * 2;
```

und bestimmen und lösen dann das lineare Gleichungssystem

```
octave> j=1; for n=0:4 for k=0:n F( :,j ) = X(:,1).^k .* X(:,2).^(n-k); j=j+1;
> end end
octave> f = Expo( X(:,1).^2 + X(:,2).^2 ); a = F\f;
```

Zur Darstellung des Ergebnisses[12] verwenden wir die Funktionen[13]

```
octave> PlotMatrixPS( EvalPoly2D ( 4, a, (-1:.1:1), (-1:.1:1 ) ) );
```

Immerhin – das Bild in Abb. 2.9 kann sich halbwegs sehen lassen, auch wenn man in den Ecken schon wieder das typische "Wackeln" der Polynome erkennen kann.

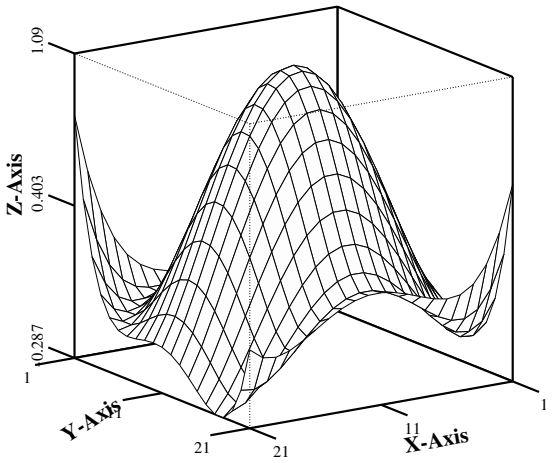

Abbildung 2.9: Interpolation von `Expo` $\left(x^2+y^2\right)$ durch Polynome vom Totalgrad 4 an 21 zufälligen Punkten.

[12] Die dreidimensionalen Plotfunktionen von Octave, zumindest in den Versionen `2.0.x` sind, vorsichtig gesagt, verbesserungsfähig.

[13] Diese Funktionen bilden eine selbstgebastelte `Octave`–Schnittstelle zum Programm `PDraw`, einer alten (Stand 1990) Public–Domain–Software zum Plotten dreidimensionaler Funktionen.

2.4 Nochmals Theorie – die Theorie der Praxis

Wir haben gesehen, daß der Ansatz mit den linearen Räumen und den Gleichungssystemen im Prinzip funktioniert - oder eben auch nicht. Um zu verstehen, warum die Dinge so funktionieren, wie sie es tun, brauchen wir ein bißchen Information über Basen und lineare Gleichungssysteme.
Das erste, was wir wissen müssen, ist die Tatsache, daß wir einen Preis dafür bezahlen, daß wir uns mit dem Hilfsmittel Computer eingelassen haben, nämlich die *numerische* Rechnung mit *endlicher Genauigkeit*. Das führt dazu, daß in jedem Rechenschritt kleine Fehler, eben Rundungsfehler dazukommen[14], die sich gemeinerweise auch noch anhäufen können.

Beispiel 2.7 *Fangen wir einfach an. Die einfache* 2×2*–Matrix*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$$

ist nicht invertierbar, das heißt, das Gleichungssystem ist nicht *immer eindeutig lösbar. In der Tat ist*

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + 3y \\ 2\,(x + 3y) \end{bmatrix},$$

"passende" rechte Seiten müssen also von der Form $[x, 2x]^T$ *sein – mit* `[1;1]`[15] *kann es also nicht funktionieren. Oder? Schauen wir mal nach:*

```
octave> A = [ 1 2; 3 6 ]; b = [1 ; 1]; x = A\b
warning: matrix singular to machine precision, rcond = 5.55112e-18
x =

  0.080000
  0.160000
```

Wir erhalten also zweierlei: eine Warnung und ein Ergebnis. Nur taugt das Ergebnis nicht viel, wie wir ganz einfach sehen können

```
octave> A*x - b
ans =

  -0.60000
   0.20000
```

Offenbar will uns `Octave` *mit der Warnung bezüglich* `rcond` *etwas sagen.*

[14] Das kann man auch tatsächlich untersuchen und durchaus vernünftige Aussagen darüber machen, aber diese Thematik langweilt sogar Mathematikstudenten.

[15] In `Octave`–Notation.

Man kann es gar nicht oft genug sagen:

> Eigentlich ist jede Matrix invertierbar: Mit beliebig kleinen Störungen wird **jedes** Gleichungssystem **theoretisch** eindeutig lösbar.

Nachdem nun aber unser Computer nicht wirklich unterscheiden kann, ob die Störungen von Rechenfehlern stammen[16], ob es Eingabe- oder Meßfehler sind, oder aber Absicht war, müssen wir damit leben, daß alle unsere Daten, sogar die Matrix unseres linearen Gleichungssystems *unscharf* sind. Ein Maß dafür, wie empfindlich das Gleichungssystem gegen solche Störungen ist, ist die *Konditionszahl* $\kappa(A)$ der Matrix A, die von `Octave` dankenswerterweise gleich mitberechnet wird[17] Genauer gesagt, das bereits erwähnte `rcond` steht für die *reverse condition number*, also $\kappa^{-1}(A)$.

> **Faustregeln:**
>
> 1. Je größer die Konditionszahl, desto "singulärer" ist die Matrix.
> 2. Die Konditionszahl ist ein Maß für die "Unlösbarkeit" des Gleichungssystem.
> 3. Ist `rcond` kleiner als die Rechengenauigkeit (heutzutage in *double precision* 10^{-16}, dann ist es hoffnungslos und man läßt besser die Finger davon!

Preisfrage:

> Ist garantiert, daß sich ein "Black–Box–System" wie beispielsweise `Idrisi` an die numerischen Faustregeln hält? Kann man erkennen, ob es das tut?

Aber natürlich sind diese Probleme doch nur "akademisches" Numerikergeschwätz – oder?

Beispiel 2.8 *Schauen wir uns doch wieder einmal unsere Polynome an, und zwar an den 21 Punkten `(0:.1:1)`. Also, wieder mal das wohlbekannte Procedere*

```
octave> X = (0:.05:1)';
octave> F = []; for j=1:length(X) F( :,j ) = X.^(j-1); end
```

Nun versuchen wir mal, die konstante Funktion und das Polynom x^13 zu interpolieren:

```
octave> a0 = F\(X.^0); a13 = F\(X.^13);
warning: matrix singular to machine precision, rcond = 5.96375e-18
warning: matrix singular to machine precision, rcond = 5.96375e-18
```

[16] Das Zauberwort hier heißt "Rückwärtsfehler"

[17] Das ist so nicht richtig! Die Bestimmung der *"exakten"* Konditionszahl benötigt einen Rechenaufwand, der ebenso groß ist wie der für das Lösen des Gleichungssystems, und das is zu viel. Was berechnet wird ist eine *Abschätzung* für die Konditionszahl, die mit *vertretbarem* Aufwand bestimmt werden kann, siehe [1].

Und da ist sie wieder, unsere Meldung. Insbesondere `a13` *ist weit davon entfernt, vernünftig zu sein, denn wir erhalten*

```
octave> a13(11:17)
ans =

   0.033901
  -0.078806
   0.145588
   0.786678
   0.246193
  -0.220892
   0.150789
```

Der Wert `0.786678` *sollte übrigens laut unserer Theorie "in Wirklichkeit"* 1 *sein und alle anderen* 0*. Trotzdem stimmen auf* $[0, 1]$ *die Plots noch sehr gut überein – man sieht eigentlich keinen Fehler; auch die Interpolation ist noch sehr gut gewährleistet:*

```
octave> disp( [ norm( F*a0 - X.^0 ),  norm( F*a13 - X.^13 ) ] )
   1.9953e-15   3.2433e-16
```

Das ist ***wirklich*** *erstaunlich gut! Sind die Daten auf der rechten Seite allerdings unregelmäßiger verteilt, dann hält sich der Jubel in Grenzen:*

```
octave> f = rand( size(X) ); a = F\f; norm( F*a - f )
warning: matrix singular to machine precision, rcond = 5.96375e-18
ans = 0.050912
```

Wenn nun unser Interpolationsproblem eindeutig lösbar ist, dann können wir ja auch spezielle rechte Seiten untersuchen, indem wir an einer Stelle den Wert 1, an allen anderen den Wert Null interpolieren. Damit erhalten wir dann *kardinale* Funktionen

$$\psi_j = \sum_{k=1}^{n} a_{jk}\, \phi_k, \qquad j = 1, \ldots, n,$$

mit der Eigenschaft

$$\psi_j\,(x_k) = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$$

Mit kardinalen Funktionen interpoliert sich's besonders einfach:

$$\phi = \sum_{j=1}^{n} f_j\, \psi_j.$$

Die Funktionen Ψ bilden ebenfalls eine *Basis* von $\mathcal{F}$!

Frage: Wie sieht die Kollokationsmatrix für die kardinale Basis aus?
Aber lohnt es sich, kardinale Funktionen zu berechnen. Schauen wir uns halt mal ein paar an, beispielsweise für `X = (0:.1:1)'`. Die polynomialen kardinalen Funktionen schwingen wieder einmal ziemlich schnell ziemlich heftig, wie man in Abb. 2.10 deutlich sieht. Was

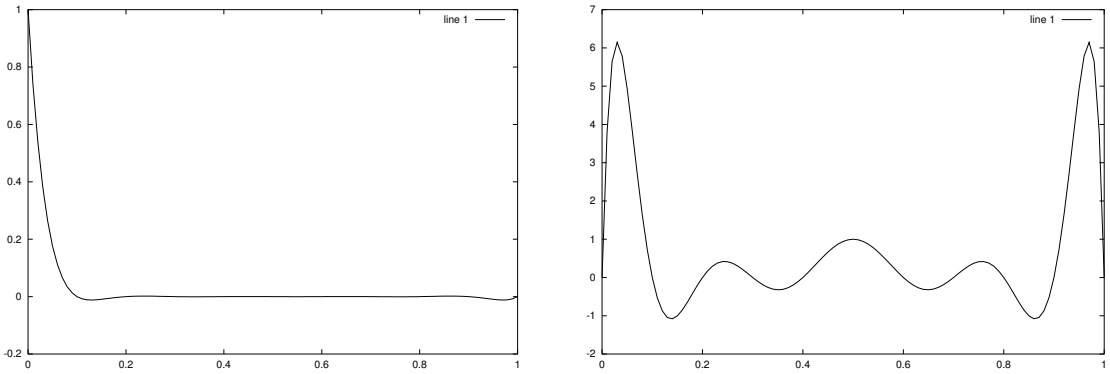

Abbildung 2.10: Zwei der kardinalen Interpolationspolynome und zwar das für $x = 0$ (links) und das für $x = 0.5$ (rechts).

aber viel schlimmer ist, das sind die Werte der Koeffizientenvektoren, nämlich

$x =$	a_0	a_1/a_6	a_2/a_7	a_3/a_8	a_4/a_9	a_5/a_{10}
0	1.00e+00	-2.93e+01	3.51e+02	-2.32e+03	9.42e+03	-2.49e+04
		4.35e+04	-5.00e+04	3.64e+04	-1.52e+04	2.76e+03
0.5	0.00e+00	5.04e+02	-1.38e+04	1.50e+05	-8.69e+05	3.01e+06
		-6.51e+06	8.89e+06	-7.43e+06	3.47e+06	-6.94e+05

Und hier sind zwei Probleme deutlich sichtbar: Die Koeffizienten wechseln das Vorzeichen[18] und werden ziemlich schnell ziemlich groß[19]. Und was das bedeutet, das sieht man ziemlich gut, wenn man sich diese beiden Funktionen mal auf dem etwas größeren Intervall $[-0.1, 1.1]$ plotten läßt, siehe Abb. 2.11 Daß es auch anders geht, das zeigt uns die Funktion `Dach`, deren Translate kardinal an den ganzzahligen Stellen sind, das heißt, die Funktion

$$\phi(x) = \sum_{j=-\infty}^{\infty} f(x)\,\mathrm{Dach}\,(x-j)$$

interpoliert an *allen* ganzzahligen Punkten. Machen wir das mit der "fast kardinalen" Funktion `Expo`, dann sieht das Ergebnis, das wir mittels

```
octave> X = (-10:10)';
octave> F = []; for j=1:length(X) F( :,j ) = Expo( X.- j + 11 ) ; end
octave> f = zeros( size(X) ); f(11) = 1; a = F\f;
```

[18] Das müssen sie.
[19] Das sollten sie besser nicht tun.

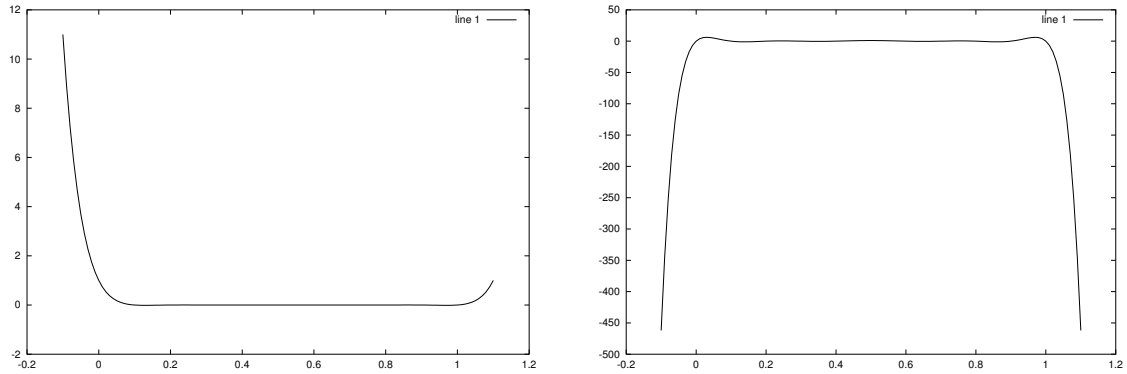

Abbildung 2.11: Die beiden Funktionen aus Abb. 2.10 auf einem etwas größerem Intervall. Man achte auf die Skalierung der y–Achse, vor allem rechts!

erhalten, ein klein wenig anders aus, siehe Abb. 2.12. "Verschmieren" wir hingegen die Funktion `Expo`, dann ergibt sich

```
octave> F = []; for j=1:length(X) F( :,j ) = Expo( .5* ( X.- j + 11 ) ) ; end
octave> f = zeros( size(X) ); f(11) = 1; a = F\f;
```

was sich ebenfalls in Abb. 2.12 bewundern läßt. Trotzdem: Hier sind die kardinalen Funktionen "brav".

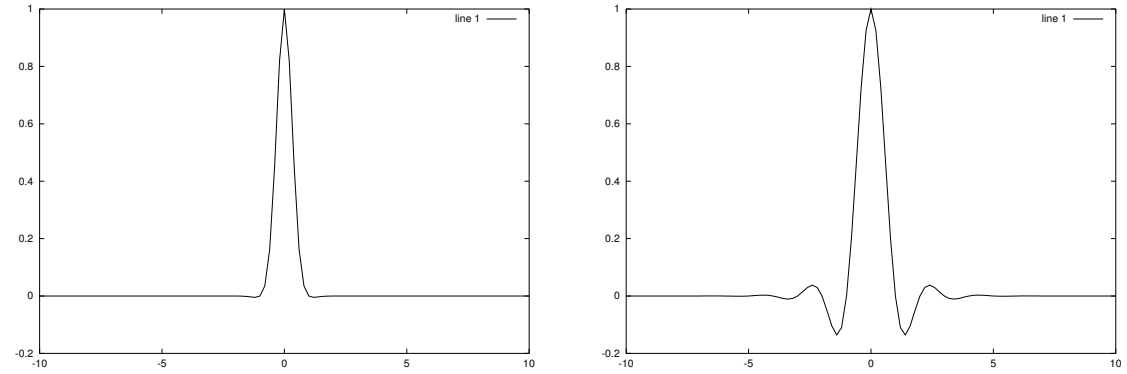

Abbildung 2.12: Die kardinalen Funktionen zu den ganzzahligen Translaten der Funktion `Expo`, links "normal", rechts die "verschmierte" Version. Aber Achtung: Der Schein trügt, diese Funktionen werden **nie** konstant Null sein.

Fassen wir zusammen:

- Interpolation mit linearen Räumen läßt sich immer als Gleichungssystem schreiben.
- Dieses Gleichungssystem kann man mit geeigneter Software lösen.
- Die Konditionszahl des Gleichungssystems ist ein Indikator, ob die Lösung auch brauchbar ist.
- Die Konditionszahl ist *optimal*, wenn man die *kardinalen Funktionen* verwendet.
- Die kardinalen Funktionen machen das Lösen des Interpolationsproblems sehr einfach, sind aber nicht immer "brav".
- Auch gut konditionierte Gleichungssysteme bedeuten nicht, daß die Lösung gut aussieht.

Welcher aber ... durch die Geometria sein Ding beweist und die gründliche Wahrheit anzeigt, dem soll alle Welt glauben. Denn da ist man gefangen.

Albrecht Dürer

3 Univariate Interpolation

In diesem Abschnitt wollen wir uns erst einmal Methoden ansehen, um *univariate* interpolierende Funktionen, also "Kurven" zu generieren. Natürlich sind auch solche Methoden in GIS nötig – irgendwie möchte man ja beispielsweise auch "schöne" Höhenlinien *automatisch* ermitteln und plotten können. Außerdem wird's für Flächen sowieso nochmal komplizierter. Dabei werden wir uns vor allem für die folgenden Fragen interessieren:

1. Kann man die Interpolationspunkte für ein bestimmtes Schema frei wählen?
2. Was sind *gute* Interpolationspunkte für ein vorgegebenes Schema?
3. Für welche Typen von Funktionen "im Hintergrund" ist das Schema geeignet.

Doch zuerst einmal ein bißchen Theorie, nämlich die Frage "Was sind Kurven eigentlich?"

3.1 Funktional, parametrisch und implizit

Es gibt drei Möglichkeiten, Kurven darzustellen:

Funktional: Eine *funktionale*[20] Kurve ist eine Abbildung[21] $x \mapsto f(x)$. Zu dieser Familie zählen alle die Interpolationsprobleme, die wir bisher betrachtet haben.

Parametrisch: Man kann sich eine Kurve anschaulich sehr schön als ein "verbogenes" Stück Draht vorstellen; das Stück Draht selbst, in "ausgestreckter" Form, ist dann aber nichts anderes als ein Streckenstück, ein "Intervall". Also ist eine Kurve hier eine Abbildung

$$[a,b] \quad \ni t \to \boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} \quad \in \mathbb{R}^n;$$

[20] Das hat nichts damit zu tun, daß diese Kurve in einer bestimmten Anwendung besonders gut funktionieren würde

[21] Gut, nachdem eine Kurve eine Punktmenge ist, ist sie eigentlich der Graph davon.

Die Verwendung der "Zeitvariablen" t legt noch eine andere Interpretation nahe, nämlich $\boldsymbol{x}(t)$ als Position[22] zu einem bestimmten Zeitpunkt. Das Interpolationsproblem ist dann von der Form

$$\boldsymbol{\phi}(t_j) = \boldsymbol{f}_j \qquad \text{oder} \qquad \begin{bmatrix} \phi_1(t_j) \\ \vdots \\ \phi_n(t_j) \end{bmatrix} = \begin{bmatrix} f_{j1} \\ \vdots \\ f_{jn} \end{bmatrix}, \qquad j = 1, \ldots, N. \tag{3.1}$$

Ist doch gar nicht so wild: Wir müssen "nur" n Interpolationsprobleme, nämlich eines für jede Komponente der interpolierenden Kurve, lösen.

Implizit: Eine *implizite* Kurve ist definiert als

$$\left\{(x, y) \in \mathbb{R}^2 \ : \ F(x, y) = 0\right\}, \qquad F \ : \ \mathbb{R}^2 \to \mathbb{R}.$$

Das Interpolationsproblem besteht dann darin, zu vorgegebenen Punkten (x_j, y_j), $j = 1, \ldots, N$, eine Funktion F zu bestimmen, so daß

$$F(x_j, y_j) = 0, \qquad j = 1, \ldots, N.$$

Jede funktionale Kurve kann parametrisch geschreiben werden, indem man ganz einfach die x/y–Kurve

$$\boldsymbol{f}(x) = \begin{bmatrix} x \\ f(x) \end{bmatrix}$$

betrachtet, die Umkehrung stimmt natürlich nicht. Denn wenn eine Kurve *funktional* ist, dann gibt es zu jedem x–Wert höchstens einen zugehörigen y–Wert, was aber von ganz einfachen parametrischen Kurven wie der in Abb. 3.1 offensichtlich nicht erfüllt ist.
Trotzdem heißt das, daß wir die funktionalen Kurven eigentlich vergessen und uns auf parametrische Kurven beschränken können – abgesehen davon kann man ja jede Komponente einer parametrischen Kurve auch wieder funktional darstellen. Und so können wir ja auch, wie in (3.1) gesehen, das parametrische Interpolationsproblem auf n funktionale Interpolationsprobleme zurückführen.
Mit der Konvertierung zwischen parametrischen und impliziten Kurven hingegen ist es *nicht* so einfach: So etwas existiert zwischen bestimmten Klassen von Funktionen, aber abgesehen davon, daß die Sache mathematisch ziemlich schwierig[23] und aufwendig ist, hat man sich auch noch mit der Nicht–Eindeutigkeit dieser *Parametrisierungen* bzw. *Implizitierungen* herumzuschlagen. Um das ganze richtig interessant zu machen, sind die beiden Darstellungen auch noch ziemlich komplementär:

Problem	parametrisch	implizit
Plotten der Kurve	leicht	schwer
(x, y) auf der Kurve?	schwer	leicht
Schnittberechnung	eher schwer	eher leicht
Punkte auf *unterschiedlichen* Seiten der Kurve	schwer	"leicht"[24]

[22] Eines Wanderers entlang der Kurve.
[23] Zu schwierig für einen Kontext wie hier, sowas wäre eine eigene mathematische Spezialvorlesung wert.

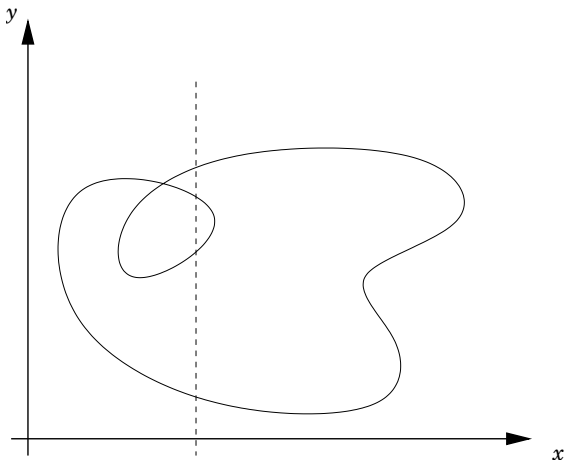

Abbildung 3.1: Eine parametrische, aber offensichtlich nicht funktionale Kurve: zu einem Wert von x gibt es mehrere y–Werte. .

3.2 Polynome

Mit Polynomen haben wir uns ja schon im Kontext der linearen Räume ziemlich exzessiv beschäftigt. Als *Polynome vom Grad n* bezeichnen wir den *linearen Raum*

$$\Pi_n := \left\{ p(x) = \sum_{j=0}^{n} p_j \, x^j \;\; : \;\; p_j \in \mathbb{R} \right\};$$

jedes Polynom wird also durch die $n+1$ Koeffizienten $p_0, \ldots, p_n$ repräsentiert. Daß Polynome nun so beliebt sind[25], liegt an der folgenden Tatsache:

Polynome interpolieren *universell*: Zu jeder Wahl von $n+1$ *verschiedenen* Punkten $x_0, \ldots, x_n$ und Werten $f_0, \ldots, f_n$ gibt es *genau ein* Polynom p, so daß

$$p\left(x_j\right) = f_j, \qquad j = 1, \ldots, n.$$

Diese Tatsache sieht man am schnellsten dadurch ein, daß man die *kardinalen* Funktionen

$$p_j\left(x\right) = \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \, \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_0}{x_j - x_n}, \qquad j = 0, \ldots, n,$$

angibt. Trotzdem:

[25] Oder zumindest waren.

Die kardinalen Funktionen sind numerisch **extrem** instabil und die Matrizen[a] des linearen Gleichungssystem sind lausig konditioniert. Ein Beispiel: liegen alle $n+1$ Punkte im Intervall $[0,1]$, dann ist

$$\kappa(A) \geq (n+1)\,2^n.$$

[a] Man bezeichnet sie als *Vandermonde–Matrizen*.

Man braucht also etwas anderes und daß man das braucht, war schon relativ früh[26] klar; die Antwort heißt *Newton–Ansatz* und schreibt das Interpolationspolynom als

$$p(x) = \sum_{j=0}^{n} [x_0, \ldots, x_j]\, f\ (x - x_0) \cdots (x - x_{j-1})\,, \tag{3.2}$$

wobei

$$\begin{aligned} [x_0]\, f &= f(x_0) \\ [x_0, \ldots, x_j]\, f &= \frac{[x_0, \ldots, x_{j-1}]\, f - [x_1, \ldots, x_j]\, f}{x_0 - x_j} \end{aligned}$$

die *dividierten Differenzen* bezeichnet. Was ist nun der Clou bei der Geschichte?

- Der Newton–Ansatz "lernt" die Interpolation "Punkt für Punkt", wobei immer ein Polynom hinzugefügt wird, das an allen "früheren" Interpolationspunkten den Wert Null hat – was wir einmal erhalten haben, das wird auch beibehalten!
- Der Wert, der "hinzugelernt" wird, ist der *Fehler*, den die vorherige "Schätzung" am neuen Interpolationspunkt macht.
- Ist also f "ungefähr" ein Polynom von kleinem Grad, so werden auch die Koeffizienten von höherem Grad recht klein sein, denn Fehler sind ja ziemlich klein.

Der letzte Punkt ist ziemlich wichtig, denn er sagt uns, wann und wo wir Polynome wirklich effizient einsetzen können:

Interpolation mit Polynomen ist dann gut, wenn die zu interpolierende Funktion sehr "polynomähnlich" ist.

Beispiel 3.1 *Ein Beispiel zum letzten Punkt: Wir interpolieren die Funktion $f(x) = e^{-x^2}$, einmal an den 11 Punkten* `(-1:.2:1)`*, and einmal an* `(-2:.4:2)` *und an* `(-3:.6:3)`*. Dazu verwenden wir ein paar* `Octave`*–Routinen aus [6], insbesondere eine Funktion* `PlotNewton`*, die einen Interpolanten nach der Newton–Methode berechnet und plottet:*

[26] Im 17. Jahrhundert!

```
octave> X = (-1:.2:1); Y = (-1:.01:1); f = exp( -X.^2 );
octave> clearplot; PlotNewton( f,X,Y ); hold on; plot( Y',exp(-Y.^2)')
octave> X = (-2:.4:2); Y = (-2:.01:2); f = exp( -X.^2 );
octave> clearplot; PlotNewton( f,X,Y ); hold on; plot( Y',exp(-Y.^2)')
```

Für den anderen Fall braucht man wohl nicht mehr anzugeben, wie die `Octave`*–Befehle aussehen. Das Ergebnis ist in Abb. 3.2 und Abb. 3.3 darsgestellt. Was man sieht ist, daß die Qualität der Polynominterpolation nicht nur von der "Glattheit" der Funktion abhängt. Faszinierend wird es nun, wenn man sich mal die Koeffizientenvektoren plotten läßt wie ebenfalls in Abb. 3.3, denn man kann diese Phänomen fast von den Koeffizienten ablesen.*

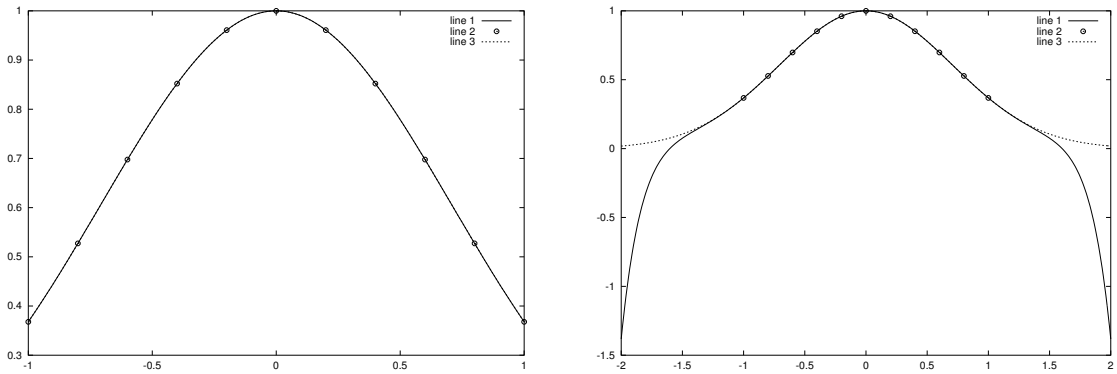

Abbildung 3.2: Der Interpolant an `(-1:.2:1)`, einmal auf dem "Interpolationsintervall", wo er sehr gut aussieht, einmal außerhalb. Daß das Ergebnis so "schlecht" wird ist nicht ganz so überraschend: Das Polynom "sieht" nur die Krümmung im Interpolationsbereich und die "Wendepunkte" liegen bestenfalls am Rand.

Aus diesem Beispiel können wir auch tatsächlich etwas lernen: Die Koeffizienten aus (3.2) scheinen etwas mit der Interpolationsqualität zu tun zu haben. Das wird klarer, wenn wir uns an die etwas vage Aussage erinnern, beim Newton–Ansatz würden sukzessive *Fehler* interpoliert. In der Tat, wenn wir mit p_k das Interpolationspolynom an den ersten $k+1$ Punkten $x_0, \ldots, x_k$ bezeichnen, dann erhalten wir, daß

$$[x_0, \ldots, x_k, x]\, f = f(x) - p_k(x), \qquad k = 0, \ldots, n.$$

Mit anderen Worten: Ist p_k eine gute *Approximation* an f, das heißt, ist $|f(x) - p_k(x)|$ klein, dann ist auch die dividierte Differenz mit $x = x_{k+1}$ klein, damit ist aber $p_{k+1} \sim p_k$ und somit ist auch p_{k+1} eine gute Approximation und die nächste dividierte Differenz wird auch wieder klein sein. Anders gesagt:

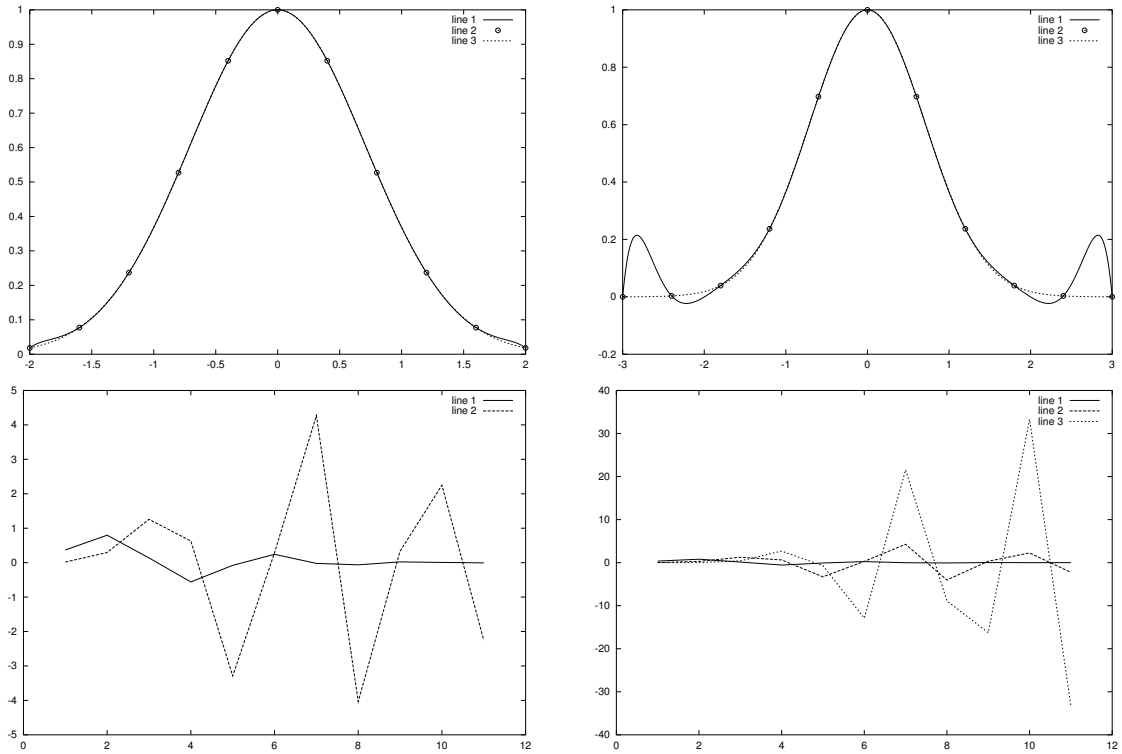

Abbildung 3.3: **Oben:** Vergrößern wir den Iterpolationsbereich, nämlich zu `(-2:.4:2)` (links) oder `(-3:.6:3)` (rechts), dann gibt es stärkere und stärkere Wackler – was niemanden mehr überraschen sollte.

Unten: Die Koeffizientenvektoren dazu. Links die ersten beiden Fälle, rechts ist der dritte Fall dabei. Wie man sieht sieht man dann vom ersten Fall gar nichts mehr.

Die dividierten Differenzen, also das Anwachsen der Koeffizienten in der Newton–Darstellung (3.2), sagen uns etwas über die Qualität polynomialer Interpolation – damit kann man beispielsweise versuchen, "Sprünge" oder "Zacken" zu lokalisieren.

Jetzt müssen wir uns nur noch darüber klarwerden, daß es die Polynome *hohen* Grades sind, die begeistert oszillieren, um zu der Feststellung zu kommen, daß Interpolation mit Polynomen dann gut ist, wenn diese Funktion durch Polynome *moderaten* Grades gut approximiert werden kann. Und nochmals für's Poesiealbum:

Wenn Polynome eine Funktion "mögen", dann funktioniert auch die Interpolation ganz gut, wenn sie die Funktion nicht mögen, dann benehmen sie sich auch bei der Interpolation zickig.

So ein Polynom ist eben auch nur ein Mensch ...

Nun aber zu dem, was die Nachteile von Polynomen ausmacht, nämlich ihre immense Störungsanfälligkeit[27].

Beispiel 3.2 *Wir betrachten Interpolation der Funktion $f(x) = x^2$ an den Punkte* `(-1:.1:1)` *und verwenden dazu unseren Newton–Ansatz. Das sieht dann richtig einfach aus:*

```
octave> X = (-1:.1:1); f = (1.+X).*(1.-X); Y = (-1:.01:1);
octave> PlotNewton ( f,X,Y );
```

Stören wir hingegen die ganze Sache zufällig um 0.5%,

```
octave> g = f .* (1 + 10^(-3)*( rand( size(f) ) .- .5 ) );
octave> PlotNewton (g,X,Y)
```

dann erhalten wir in Abb. 3.4 ein wesentlich weniger begeisterndes Bild – und ein Blick auf die Koeffizienten[28] *zeigt uns auch, warum!*

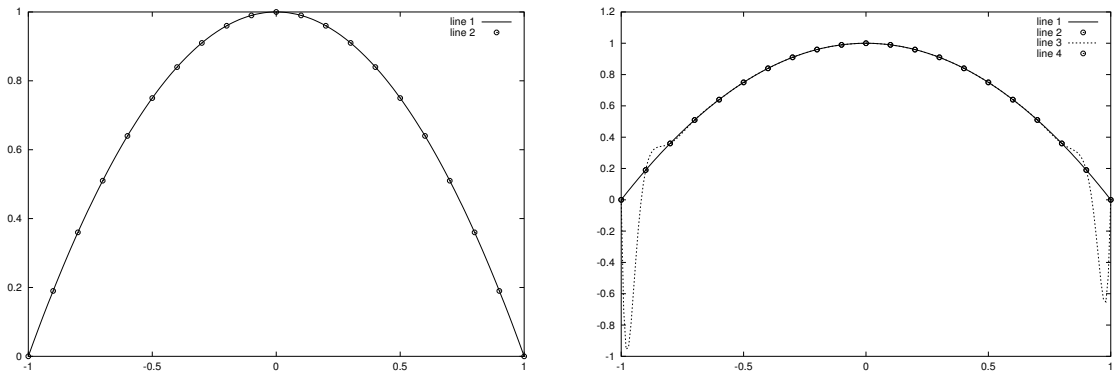

Abbildung 3.4: Interpolation einer Parabel an 21 Punkten, links ohne Störung, rechts mit zufälliger Störung um lediglich 0.5%

Und weil das an schlechten Nachrichten noch nicht reicht, noch eine weitere, vielleicht sogar schlimmere Eigenschaft der Polynome. Wir haben ganz am Anfang gesagt, daß Polynome an *beliebigen* Punkten interpolieren können, solange diese Punkte nur alle unerschiedlich sind. Nur leider sind "ob" und "wie" oftmals zwei ziemlich unerschiedliche Dinge.

> Die Qualität des polynomialen Interpolanten hängt **dramatisch** von der der Lage der Interpolationspunkte ab.

[27] Wen wundert's nach der letzten Bemerkung noch, daß Polynome ziemliche Sensibelchen sind.
[28] Mit dem Kommando `NewtonKoeff1 ( g,X )`!

Beispiel 3.3 *Wir interpolieren jetzt mal parametrisch, und zwar die* geschlossene *Kurve durch die* gleichverteilten *Punkten auf dem Einheitsquadrat, die in Abb. 3.5 zu sehen sind. In Matlab wird dieser Vektor durch*

```
octave> f1 = [ 0 .25 .5 .75 1; 0 0 0 0 0 ]; f2 = [ 1 1 1 1; .25 .5 .75 1];
octave> f3 = [ .75 .5 .25 0; 1 1 1 1 ]; f4 = [ 0 0 0 0; .75 .5 .25 0 ];
octave> f = [ f1 f2 f3 f4 ];
```

repräsentiert. Da jeder Punkt von seinem Nachfolger gleich weit entfernt ist, liegt es nahe, auch die Parameterwerte gleichverteilt *zu wählen, was mit dem Befehl*

```
octave> clearplot; auxPlotNewton( f, (-1:1/8:1), (-1:.01:1) )
```

das linke Bild in Abb. 3.6 liefert. Verwendet man hingegen bestimme, optimale[29] *Parameterwerte, die sogenannten* Tschebyscheff–Knoten*, dann ergibt sich plötzlich ein ganz anderes Bild.*

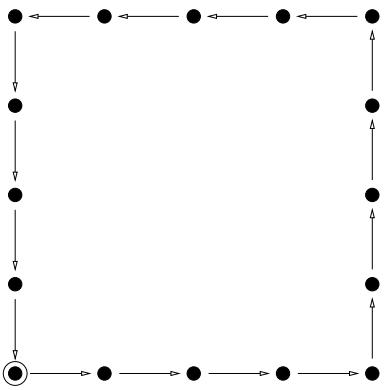

Abbildung 3.5: Die *parametrischen* Interpolationspunkte zu Beispiel (3.3); die Wahl der zugehörigen Parameterwerte oder "Zeitpunkte" ist in keinster Weise vorgegeben – leider oder glücklicherweise!

3.3 Splines

Um den Schwierigkeiten mit Polynomen zu entgehen, muß man also etwas anderes tun. Und am besten fängt man mal damit an, sich zu überlegen, **warum** es die Probleme gibt. Der Grund ist nämlich die *Globalität* der Polynome: kennt man ein Polynom auch nur auf einem winzigen Bereich, dann kennt man es überall, versucht also ein Polynom lokal einen Charakterzug einer Funktion nachzubilden, dann wird sie woanders scheitern, und zwar dort, wo sich diese Funktion anders benimmt. Es geht halt nur entweder so oder so.

[29] Und zwar optimal unabhängig davon was für eine Funktion man zu interpolieren gedenkt!

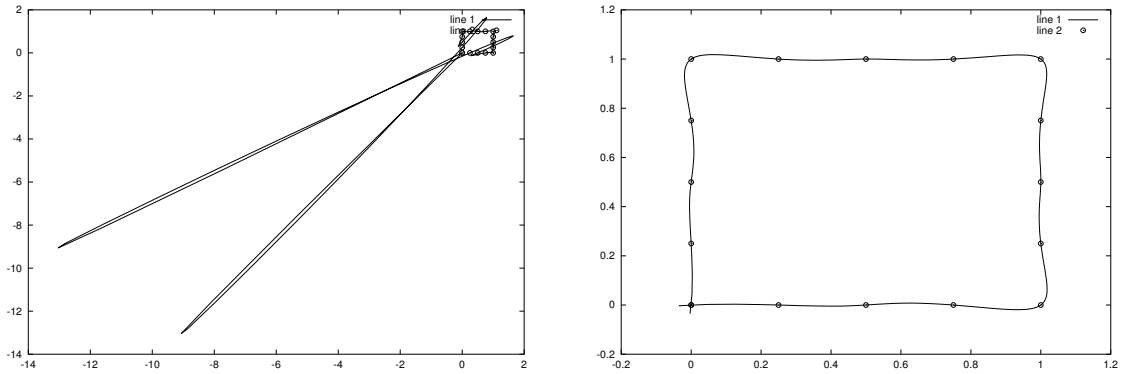

Abbildung 3.6: Interpolation des Einheitsquadrats mit gleichverteilten (links) und "magischen" (rechts) Parameterwerten.

Wenn also Globalität ein Problem darstellt, dann sollte man es mit etwas *lokalem* versuchen, also beispielsweise *stückweisen* Polynomen. Um stückweise arbeiten zu können, müssen wir zuerst spezifizieren, was die "Stücke" denn eigentlich sind; dafür gibt man sich sogenannte *Knoten* vor, das heißt Werte

$$t_0 \leq t_1 \leq \cdots \leq t_{M-1} \leq t_M, \qquad M =?, \tag{3.3}$$

die *aufsteigend* geordnet sind, sich aber durchaus wiederholen[30] dürfen, allerdings nicht zu oft, doch dazu werden wir noch kommen. Ein *(polynomialer) Spline* der Ordnung[31] m ist eine Funktion, die, eingeschränkt auf das *offene Intervall*[32] (t_j, t_{j+1}), ein Polynom vom Grad m ist. Ein bißchen mehr sollten wir allerdings schon fordern, nämlich daß die Polynomstücke "ordentlich" zusammengesetzt sind, also etwas miteinander zu tun haben. Zu diesem Zweck benutzt man die sogenannten *B–Splines*.

Denken wir mal an den allereinfachsten Fall, nämlich $m = 0$. Dann haben wir es mit Funktionen zu tun, die auf den Intervallen (t_j, t_{j+1}) *konstant* sind. Wären diese Funktionen nun auch noch "global" stetig, dann hätten wir es mit einer einzigen konstanten Funktion, also einem Polynom vom Grad 0 zu tun, was zugegebenermaßen ein bißchen langweilig wäre. Also gönnen wir uns *stückweise* konstante Funktionen und einfache Knoten und

[30] Diese Wiederholungen, anders gesagt, die *Vielfachheit* der Knoten ist ein wichtiger Designparameter, wie wir gleich sehen werden.

[31] Vorsicht beim Lesen mathematischer Literatur, z.B. [3, 5, 7]! Der Begriff der Ordnung ist nicht eindeutig definiert, es gibt hier unterschiedliche "Schulen".

[32] Das ist ein Intervall, bei dem die Endpunkte *nicht* dazugehören. Sind die beiden Endpunkte des Intervalls identisch, dann ist das zugehörige Intervall die leere Menge und auf der leeren Menge ist sogar die Exponentialfunktion ein Polynom vom Grad 0 oder 27 – ein fundementaler Grundsatz der formalen Logik.

erhalten für $M = n + 1$ Funktionen der Form

$$\phi(x) = \sum_{j=0}^{n} a_j \, N_j^0 \, (x) \, , \qquad N_j^0(x) = \begin{cases} 1, & x \in [t_j, t_{j+1}) \, , \\ 0, & x \notin [t_j, t_{j+1}) \, . \end{cases}$$

Achtung: Das *halboffene Intervall*[33] brauchen wir, damit wir auch an den Knoten selbst einen Wert unserer Funktion ϕ haben; daß wir die Funktionen linksseitig fortsetzen ist willkürlich, aber zulässig.
Die B–Splines höherer Ordnung $m > 1$ definiert man nun als

$$N_j^m(x) = \frac{x - t_j}{t_{j+m} - t_j} \, N_j^{m-1} \, (x) + \frac{t_{j+m+1} - x}{t_{j+m+1} - t_{j+1}} \, N_{j+1}^{m-1} \, (x) \tag{3.4}$$

und die zugehörige *Splinefunktion* ist

$$\phi(x) = \sum_{j=0}^{n} a_j \, N_j^m \, (x) \, ,$$

mit den Knoten $t_0 \leq \cdots \leq t_{n+m+1}$ – also hat das "unbekannte" M in (3.3) den Wert $M = n+m+1$. Natürlich sollten wir die komplizierte Formel (3.4) sofort wieder vergessen und lieber nachsehen, was für Eigenschaften diese Splinekurven haben:

B–Splines der Ordnung m sind, eingeschränkt auf die[a] Knotenintervalle, Polynome vom Grad m, also gilt das auch für die Splinekurve. Außerdem:

- Der B–Spline N_j^m "lebt" auf dem Intervall $[t_j, t_{j+m+1}]$, das heißt, er hat außerhalb dieses Intervalls immer den Wert Null. Die Basis ist also wirklich *lokal*.
- Ein B–Spline der Ordnung m ist an einem einfachen Knoten $m - 1$ mal stetig differenzierbar,
- und an einem k–fachen Knoten $m - k$ mal.

Mit B–Splines kann man also bewußt Knicke und Sprünge modellieren.

[a] Jetzt sogar halboffenen.

Da weniger als -1–fache Differenzierbarkeit, also eine unstetige Funktion mit Sprüngen, bei denen benachbarte Funktionsstücke **nichts** miteinander zu tun haben, sowieso nicht möglich ist, oder zumindest in diesem Kontext wenig Sinn macht[34], können wir noch eine Regel vorgeben:

Für Splines der Ordnung m darf kein Knoten eine größere Vielfachheit als $m+1$ haben.

[33] Bei dem *ein* Endpunkt, in unserem Fall der linke, dazugehört.
[34] "Richtige Mathematiker" finden allerdings in jedem noch so abgefahrenen Konzept Sinn!

Wir verwenden eine[35] `Octave`–Funktion namens `BSpline`, um solche B–Splines auszuwerten; diese bekommt den Knotenvektor und einen Vektor von Punkten, an denen der B–Spline ausgewertet werden soll, als Argumente übergeben. Das Plotten dieser Funktionen wird dann ganz einfach zu

```
octave> T = [ 0 1 2 ]; X = (-.5:.01:2.5); plot( X', BSpline (X,T)' );
```

um beispielsweise den *linaren* ($m = 1$) B–Spline mit den Knoten $0, 1, 2$ zu zeichnen.

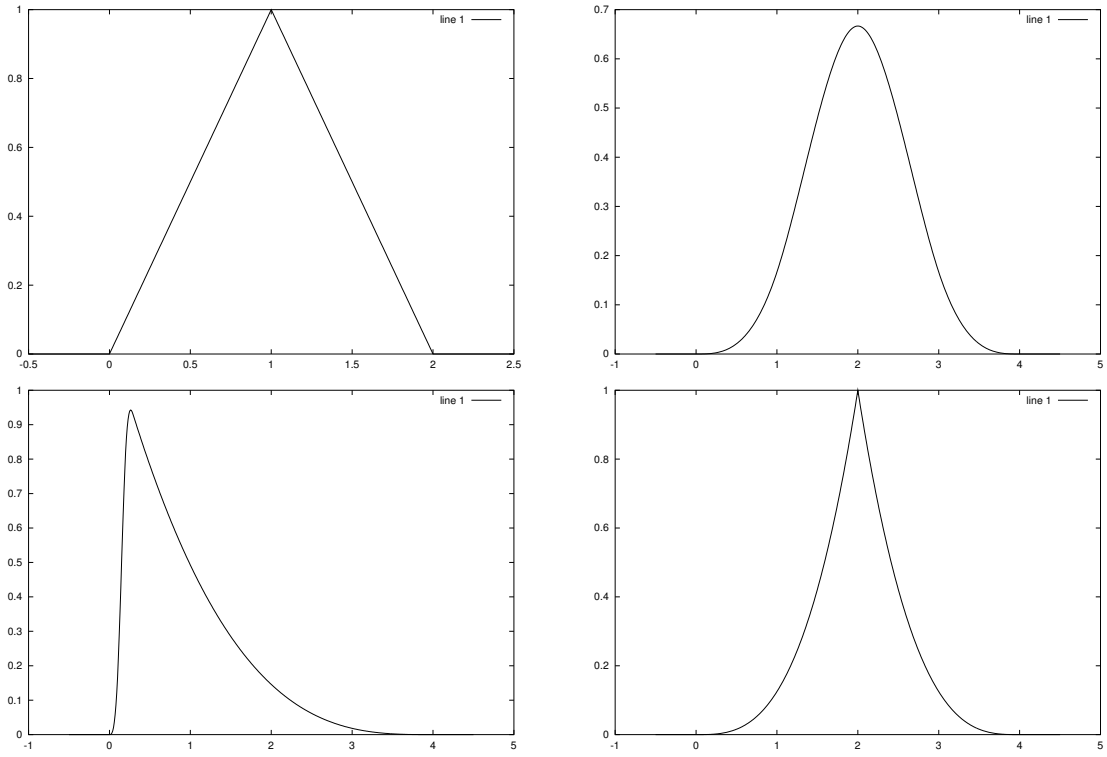

Abbildung 3.7: Eine kleine Kollektion von B–Splines: oben links der lineare B–Spline zu den Knoten `[0 1 2]`, daneben der kubische ($m = 3$) B–Spline mit Knotenvektor `[0 1 2 3 4]`, uten links ein B–Spline mit immer noch verschiedenen, aber am linken Rand ziemlich dichten Knoten `[0 .1 .2 .3 4]` und schließlich ein B–Spline mit einem *dreifachen* Knoten, genauer, dem Knotenvektor `[0 2 2 2 4]`.

Ein paar Beispiele für B–Splines finden sich in Abb. 3.7. Wie man sieht, ist unser gutes altes "Dach" wieder mit von der Partie, interessanter aber sind die beiden Beispiele in der unteren Zeile, die zeigen, was man mit Splines wirklich alles anstellen kann. Liegen nämlich Knoten relativ dicht beisammen (wie unten links), dann hat die Kurve zwar immer noch maximale Differenzierbarkeitsordnung $3 - 1 = 2$, steht allerdings unter einem

[35] selbstgebastelte

ziemlichen "Zug". An einem dreifachen Knoten hingegen (unten rechts) erhält man nur noch Differenzierbarkeitsordnung $3 - 3 = 0$, also nur Stetigkeit. Und in der Tat ist der Knick deutlich sichtbar.

Das ist ja nun alles nett, schön und gut, aber hilft uns das auch für die Interpolation? Die Antwort ist natürlich "ja"[36], allerdings gibt es eine Einschränkung: Zwischen den Interpolationspunkten und den Knoten muß eine gewisse Beziehung bestehen[37]

> Interpolation mit Splines ist genau dann möglich, wenn die Interpolationspunkte und die Splines geschachtelt sind:
>
> $$t_j < x_j < t_{j+m+1}$$

Eine besonders einfache Form der Interpolation wäre Interpolation an den Knoten; sofern diese einfach sind, gibt es viele Möglichkeiten, beispielsweise $x_j = t_{j+m/2}$, $j = 0, \ldots, n$ – diese Punkte erfüllen die obige Bedingung[38] solange die Knoten einfach sind. Etwas "fortgeschrittener" sind die *Greville–Abszissen*

$$x_j = \frac{1}{m+2} \sum_{k=0}^{m+1} t_{j+k}, \qquad j = 0, \ldots, m.$$

Diese Punkte sind wohldefiniert und erlauben **immer** Interpolation; schließlich haben wir ja vorausgesetzt, daß alle Knoten höchstens die Vielfachheit $m + 1$ haben.

Beispiel 3.4 *Nehmen wir doch mal unsere 17 Punkte auf dem Quadrat und sehen uns an, was* kubische *Spline–Interpolation hier liefert. Dafür brauchen wir $n = 16$, also $M = n + m + 1 = 16 + 3 + 1 = 20$, also* 21 *Knoten. Machen wir's uns erst mal einfach und nehmen einen gleichverteilten Knotenvektor* `(0:20)`*. Die Kollokationsmatrix ergibt sich also mit dem folgenden "Verfahren":*

```
octave> T = (0:20); X = ( 2:18 )';
octave> F = []; for j=1:17 F( :,j ) = BSpline( X,T(j:j+4) ); end
octave> a = F\f';
```

Zum Plotten des Ergebnisses verwenden wir unser altbewährtes Verfahren

```
octave> Y = ( 0:.05:20 )';
octave> G = []; for j=1:17 G( :,j ) = BSpline( Y,T(j:j+4) ); end
octave> Z = G*a; plot( Z(:,1),Z(:,2) )
```

Entsprechend können wir auch an den Greville-Abszissen interpolieren, indem wir

```
octave> X = Greville ( 3, T )';
```

[36] Sonst hätten es die Splines ja wohl kaum in dieses Skript geschafft, oder?

[37] Es ist in der Mathematik oft genug wie im richtigen Leben: Kein Objekt existiert für sich allein.

[38] Die als *Schoenberg–Whitney–Bedingung* bekannt ist.

verwenden. Das Ergebnis findet sich in Abb 3.8, die beiden Bilder sind nicht unterscheidbar.
Legen wir etwas mehr "Spannung" in die Knoten, indem wir sie am Anfang dichter und am Ende weiter entfernt wählen

```
octave> T = (0:1/20:1).^6; X = Greville( 3,T )';
octave> F = []; for j=1:17 F( :,j ) = BSpline( X,T(j:j+4) ); end
octave> a = F\f';Z = G*a; plot( Z(:,1),Z(:,2) )
```

dann kann das schon ganz anders aussehen. Und mit einer richtig "chaotischen" Knotenwahl, beispielsweise

```
octave> T = [ (0:.1:.5), (1:10), (10.1:.1:10.5) ]; X = Greville( 3,T )';
```

erhält man auch einen etwas chaotischeren Kurvenverlauf.

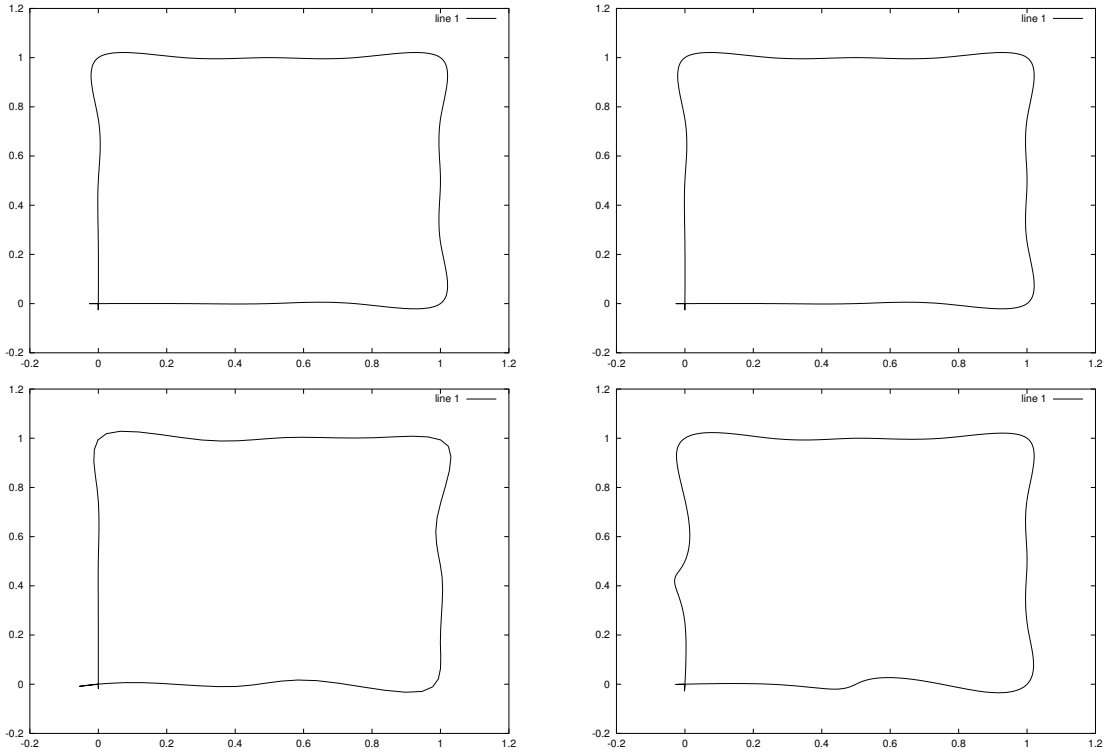

Abbildung 3.8: Interpolation des "Quadrats" mit kubischen Splines und gleichverteilten Knoten, wahlweise an Knoten (links) oder an den Greville-Abszissen (rechts). Unterschied ist keiner zu sehen. In der unteren Zeile hingegen sind die Knoten ungleichmäßig gewählt. Links häufen sie sich am linken Rand, rechts sind sie an den beiden Rändern dichter als im Inneren.

Beispiel 3.5 *Wir können aber auch die Punkte am Rand weit und im Inneren dicht wählen*

```
octave> T = [ (0:5), (5.1:.1:5.9), (6:11) ]; X = Greville( 3,T )';
```

was die Kurve links in Abb. 3.9 liefert. Als krönenden Höhepunkt nehmen wir schließlich

```
octave> T = [ 0,0,0,1,1,1,2,2,2,3,3,4,4,5,5,(6:11) ]; X = Greville( 3,T )';
```

also eine Mischung aus dreifachen, doppelten und einfachen Knoten. Was wir da in Abb. 3.9 zu sehen bekommen ist allerdings nicht so begeisternd.

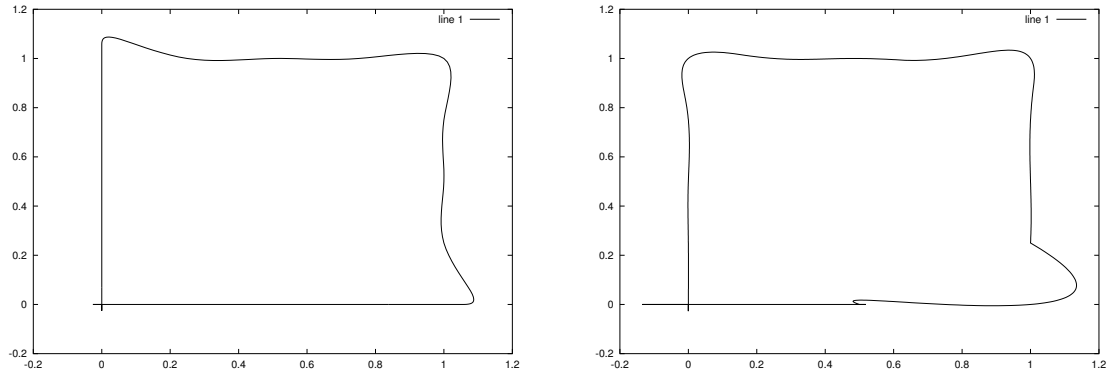

Abbildung 3.9: Noch mehr Beispiele. Einmal "außen" weit und innen dicht und einmal 3 dreifache, 3 doppelte und 6 einfache Knoten.

Interpolierende Splines mit einfachen Knoten zeigen immer eine Tendenz zum "Überschießen" and den Ecken einer Kurve. Das ist auch klar, wie die folgende physikalische Analogie zeigt: Ein kubischer Spline mit einfachen Knoten hat eine stetige zweite Ableitung, was einer stetigen *Beschleunigung* entsprechen würde. Auf diese Art und Weise kann man aber nicht um "richtige" Ecken kommen, da sich da die Fahrtrichtung abrupt ändert!
Trotzdem gibt es einen kubischen Spline, der das Quadrat *exakt* nachbilden kann, dieser muß aber an den Ecken dreifache Knoten[39] haben.

> Splines liefern Kurven von hoher Flexibilität, allerding muß man recht genau wissen, wie und wo man seine Knoten wählt.

Ach ja, woher kommt eigentlich der Name "Spline"? Ursprünglich ist ein Spline nämlich ein Kurvenlineal, das aus einem biegsamen Streifen und Gewichten besteht, mit denen dieser Streifen an den Interpolationspunkten festgehalten wird, siehe Abb. 3.10. Und wie

[39] Und damit Reduzierung der "Glattheit" auf das "Mindestmaß" Stetigkeit.

Abbildung 3.10: Ein "richtiger" Spline, das heißt, das Kurvenlineal, das dem mathematischen Objekt seinen Namen gegeben hat. Vielen Dank hierfür an Herrn Dr. Hollenhorst vom Hochschulrechenzentrum der JLU Gießen.

die Natur so ist, legt sich so ein Streifen automatisch so, daß die Biegeenergie *minimiert* wird[40]. Diese Biegeenergie könnte man nun durch das Integral über die zweite Ableitung annähern[41] und dann stellt sich heraus, daß es unter allen Lösungen eines gegebenen Interpolationsproblems immer einen bestimmten kubischen Spline mit Interpolation an bestimmten Knoten gibt, den sogenannten *natürlichen Spline*, der das Integral über die zweite Ableitung minimiert.

> Splines sind keine Splines!

3.4 Variationsminimierende Interpolation

Die Tatsache, daß Splines ein gewisses *Variationsfunktional* (die "Biegeenergie") minimieren, hat eine ganze Familie von Interpolationsverfahren motiviert, die darauf basieren, daß man mit einem "zu großen" linearen Raum interpoliert und dann die verbleibenden Freiheitsgrade darauf verwendet, das "Energiefunktional" zu minimieren. Dies ist manchmal etwas aufwendig, weil man dabei im Normalfall *nichtlineare* Funktionen minimieren muß[42], liefert aber oftmals sehr ansprechende, "organische" Kurven.

[40] Das liegt natürlich auch am Material, aus dem der Spline gefertigt ist. Wäre das Material ein Gummiband, so hätte man einen stückweise linearen Interpolanten, wäre das Material dagegen "mittelweich", dann kann man schöne geschwungene Kurven erwarten.

[41] Diese Näherung ist normalerweise sehr grob!

[42] Wofür es eine ganze Menge von Methoden gibt, die aller mehr oder weniger gut funktionieren.

If your wish is to become really a man of science and not merely a petty experimentalist, I should advise you to apply to every branch of natural philosophy, including mathematics.

M. Shelley, *Frankenstein*

4 Flächen

Nach den langen[43] Vorreden kommen wir nun endlich zum eigentlichen Thema, nämlich zur Interpolation mit *Flächen*. Zu diesem Zweck werden wir "nur" *parametrische* Flächen betrachten[44]; in Analogie zu den Kurven können wir diese Flächen als "verbogene" Version eines – ja wovon eigentlich? – betrachten. Flächen sind Abbildungen, die einen "Bereich" Ω in den $\mathbb{R}^N$ "verzerren", wobei natürlich $N > 2$ sein sollte. Dieser Bereich Ω bestimmt dann auch die Natur der Fläche und diese ist unterschiedlich, wenn beispielsweise Ω

- ein Kreis,
- ein Quadrat oder Rechteck,
- ein Dreieck

ist. Der Grund: es gibt keine "vernünftige"[45] Transformation, die aus einem dieser Gebilde das andere macht. Der Einfachheit halber werden wir Rechtecksflächen betrachten – aus der Sicht des "lokalen" Kartographen vielleicht gar nicht so abwegig.

4.1 Tensorproduktflächen

Es gibt ein einfaches Verfahren, mit dessen Hilfe wir aus Kurven Flächen machen können, ist die Konstruktion von *Tensorproduktflächen*. Die Idee dabei ist die folgende:

> Eine Fläche ergibt sich, indem sich eine Kurve durch den Raum bewegt und dabei auch noch verändert.

Dieses Schema ist in Abb. 4.1 schematisch dargestellt. Modellieren wir die Idee doch einmal mathematisch, indem wir wieder einmal von unserem Konzept des *linearen Raums* Gebrauch machen. Seien $\phi_1, \ldots, \phi_m$ und $\psi_1, \ldots, \psi_n$ die Basen von zwei linaren Räumen.

[43] Aber größtenteils notwendigen.

[44] Funktionale Flächen können genauso wie bei den Kurven parametrisch dargestellt werden, implizite Flächen sind richtig schwierig.

[45] Das heißt "glatte".

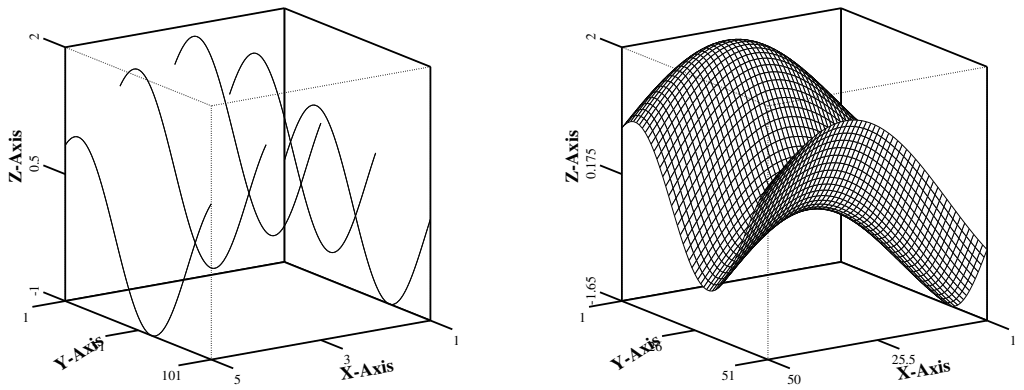

Abbildung 4.1: Links die sich "bewegenden" Kurven und rechts die Fläche, die auf diese Art und Weise entsteht.

Für vorgegebenes y setzt man die "sich bewegende" Kurve $f(\cdot, y)$ – der Punkt "$\cdot$" steht hierbei für das Argument, das sich verändern darf – als

$$f(x,y) = \sum_{j=1}^{m} a_j(y)\, \phi_j(x).$$

Das "Verändern" drückt sich darin aus, daß die Koeffizienten jetzt von y abhängen dürfen. Um auch diese Abhängigkeit modellieren zu können, verwenden wir die Basis Ψ und setzen somit

$$a_j(y) = \sum_{k=1}^{n} a_{j,k}\, \psi_k(y), \qquad j = 1, \ldots, n.$$

Einsetzen in die vorherige Gleichung liefert uns auch schon den

Tensorproduktansatz:

$$f(x,y) = \sum_{j=1}^{m} \sum_{k=1}^{n} a_{jk} \underbrace{\phi_j(x)\, \psi_k(y)}_{=:\phi_{jk}} \tag{4.1}$$

Dabei erhalten wir die *Basisfunktionen* ϕ_{jk}, $j = 1, \ldots, m$, $k = 1, \ldots, n$, für einen endlichdimensionalen Raum von Funktionen – wir können also unseren Ansatz "linearer Raum" voll übertragen.

Dabei ist es völlig egal, ob wir hier Tensorprodukte von gleichen oder verschiedenen Funktionen bilden, und ob wir die "Auflösung" in x–Richtung höher oder niedriger als in y–Richtung wählen, ob wir also $m < n$ oder $m > n$ wählen. Auch so kann man eine problemangepassten Interpolationsraum modellieren.

Besonders gut geeignet ist der Tensorproduktansatz für "Gitterdaten", das heißt, für Interpolationspunkte der Form

$$(x_j, y_k)\,, \qquad j = 1, \ldots, m,\ k = 1, \ldots, n, \qquad x_1, \ldots, x_m, y_1, \ldots, y_n \in \mathbb{R}.$$

Beispiel 4.1 *Schauen wir noch mal auf unsere "guten alten" Polynome. Dann ergibt sich mit* $\Phi = \{1, x, \ldots, x^m\}$ *und* $\Psi = \{1, y, \ldots, y^n\}$ *der Tensorproduktraum als* $(n+1)(m+1)$*–dimensionaler linearer Raum mit der Basis*

$$\left\{p(x,y) = x^j\, y^k \ :\ j = 0, \ldots, m,\ k = 0, \ldots, n\right\}.$$

Wir können also hoffen, an $(n+1)(m+1)$ *Punkten zu interpolieren*[46]*. Nur noch mal zum Vergleich: Der* Totalgrad–Interpolationsraum *hat hingegen die Form*

$$\left\{p(x,y) = x^j\, y^k \ :\ j + k \le n\right\}$$

und erlaubt Interpolation an $(n+1)(n+2)/2$ *Punkten, siehe Abb. 4.2.*

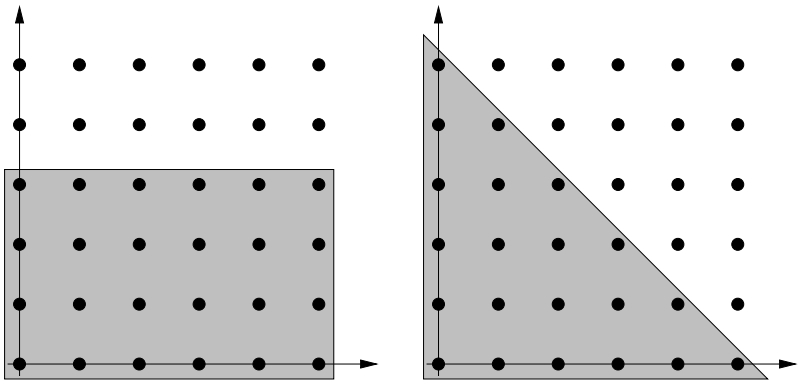

Abbildung 4.2: Die Exponenten (j, k) eine Tensorproduktraums mit $m = 5$, $n = 3$ (links) und des Totalgradraums mit $n = 5$ (rechts). Man hat es also entweder mit "Rechtecken" oder "Dreiecken" zu tun. Trotzdem: Mit einem Bereich, in dem man interpoliert, hat das ganz und gar nichts zu tun.

Beispiel 4.2 *Wegen der hohen Flexibilität der Kurven auf der einen und dem sehr einfachen Übergang zur Fläche auf der anderen Seite gehören* Tensorprodukt–Splineflächen *mit Sicherheit zu den am weitesten verbreiteten "Ansatzfunktionen" zur Flächenmodellierung oder -interpolation.*

[46] Nicht vergessen: Damit eine Matrix überhaupt ein vernünftiges (das heißt eindeutig lösbares) Gleichungssystem liefert, muß sie quadratisch sein!

Ein Ansatz, der zu optisch besonderes glatten Kurven führt, besteht darin, wesentlich mehr Freiheitsgrade als Interpolationsbedingungen zuzulassen und diese "freien" Parameter so zu bestimmen, daß beispielsweise die Biegeenergie der entstehenden Fläche minimal wird. Dieser Ansatz, oft auch als Fairing *bezeichnet führt zu sehr "organischen" Flächen.*

Wir könnten mit dem, was wir bereits kennen, nun eine Unmenge von Tensorprodukt–Interpolanten bestimmen, beliebige Funktionen miteinander kombinieren und dabei feststellen, daß diese Ergebnisse all die Stärken und Schwächen ihrer univariaten Gegenstücke erben – kein Wunder, wenn man bedenkt, wie wir die Dinge zusammengebastelt haben. Das wollen wir aber nicht tun, sondern uns vielmehr klarmachen, daß und warum Interpolation von Flächen doch etwas ganz anderes ist, als Interpolation von Kurven.

4.2 Der Fluch der Fläche

Wir erinnern uns, daß bei der Kurveninterpolation die Polynome eine faszinierende Eigenschaft hatten: Ungeachtet der numerischen und sonstigen Schwierigkeiten, die sie uns bereitet haben, waren Polynome *universale* Interpolanten, das heißt, solange die Punkte nur aller verschieden waren[47], war die Existenz des Interpolationspolynoms gesichert. Diese "universellen" Räume führen zu einer sehr starken mathematischen Theorie und liefern Eigenschaften, die man gerne auch für Flächen hätte. Aber leider ...

Für Flächen gibt es keine universellen Interpolationsräume.

Diese Tatsache kann man erstaunlich einfach einsehen[48]. Dazu brauchen wir lediglich zwei "schöne" Kurven, entlang derer wir die beiden Punkte (x_1, y_1) und (x_2, y_2) vertauschen können ohne daß die beiden dabei einander oder einem anderen Punkt zu nahe kommen, siehe Abb. 4.3. Dieser Vorgang soll eine Sekunde dauern[49] und die beiden Punkte zum Zeitpunkt t sollen als $(x_1(t), y_1(t))$ bzw. $(x_2(t), y_2(t))$ bezeichnet werden, wobei $t \in [0, 1]$ und

$$\begin{array}{rclcrcl} (x_1(0), y_1(0)) & = & (x_1, y_1) & \qquad & (x_1(1), y_1(1)) & = & (x_2, y_2) \\ (x_2(0), y_2(0)) & = & (x_2, y_2) & & (x_2(1), y_2(1)) & = & (x_1, y_1) \end{array}$$

sein soll. Zu jedem Zeitpunkt $t \in [0, 1]$ sind dabei alle Interpolationspunkte

$$(x_1(t), y_1(t)), (x_2(t), y_2(t)), (x_3, y_3), \ldots, (x_N, y_N)$$

voneinander verschieden, so haben wir ja gerade die Vertauschungswege gewählt. Die

[47] Und das ist generell eine Minimalforderung! Trotzdem kann man Polynominterpolanten sogar vernünftig für *mehrfache* Interpolationspunkte einführen, was zur sogenannten *Hermite–Interpolation* führt.

[48] Und dabei auch lernen, warum es schiefgeht, ja schiefgehen *muss*.

[49] Heutzutage, wo die Computer immer schneller werden kein Problem mehr.

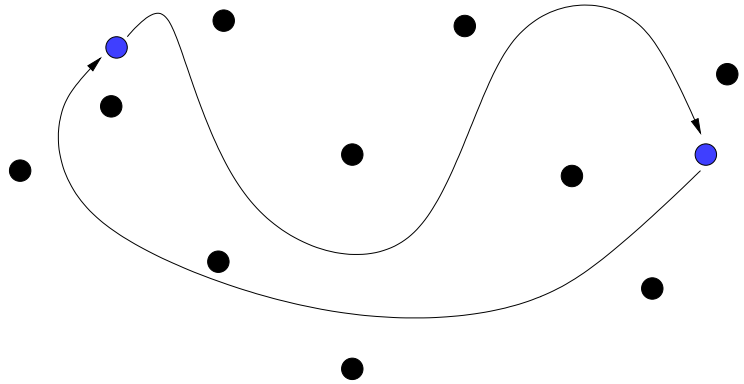

Abbildung 4.3: Vertauschung der beiden blauen Punkte ohne einander oder einen anderen Interpolationspunkt zu treffen.

Kollokationsmatrizen $\boldsymbol{F}(t)$ haben nun die Eigenschaft, daß

$$\boldsymbol{F}(0) = \begin{bmatrix} \phi_1(x_1, y_1) & \dots & \phi_N(x_1, y_1) \\ \phi_1(x_2, y_2) & \dots & \phi_N(x_2, y_2) \\ \vdots & \ddots & \vdots \\ \phi_1(x_N, y_N) & \dots & \phi_N(x_N, y_N) \end{bmatrix}$$

und

$$\boldsymbol{F}(1) = \begin{bmatrix} \phi_1(x_2, y_2) & \dots & \phi_N(x_2, y_2) \\ \phi_1(x_1, y_1) & \dots & \phi_N(x_1, y_1) \\ \vdots & \ddots & \vdots \\ \phi_1(x_N, y_N) & \dots & \phi_N(x_N, y_N) \end{bmatrix}$$

fast identisch sind, nur die ersten beiden Zeilen sind vertauscht. Das heißt aber, daß $\det \boldsymbol{F}(0) = -\det \boldsymbol{F}(1)$ ist und da die Vertauschungskurven brav[50] sind, ist $t \mapsto \det \boldsymbol{F}(t)$ eine stetige Funktion, weswegen es mindestens ein $t \in [0, 1]$ geben muß, so daß $\det \boldsymbol{F}(t) = 0$, also das zugehöre Interpolationsproblem nicht lösbar ist.
Und jetzt sollte es auch klar sein, warum es universelle Interpolationsräume wohl nur in einer Variablen geben kann und gibt: Auf der rellen Achse kann man Punkte nicht **stetig** aneinander "vorbeirangieren", ohne daß sie zusammenfallen.
Das gibt uns einen weiteren Spruch für das Poesiealbum:

> Bei der Flächeninterpolation **muß** der Interpolationsraum auf irgendeine Weise von den Interpolationspunkten abhängen.

[50] Stetigkeit reicht, denn die Determinante ist eine stetige Funktion, ja sogar ein Polynom, in den Matrixkoeffizienten.

And let me assure you, let me say to–whom—it–may–concern, that of the truth of these further stories there can be no doubt whatsoever. So finally it is not for me to judge, but for you.

S. Rushdie, *The Moor's Last Sigh*

5 Flächeninterpolation

Jetzt ist es endlich so weit, daß wir uns einige Methoden ansehen können, wie man Flächen interpolieren kann, das heißt, wie man zu vorgegebenen Punkten

$$(x_j, y_j)\,, \qquad j = 1, \ldots, n,$$

eine funktionale[51] Fläche ϕ findet, so daß

$$\phi\,(x_j, y_j) = f_j, \qquad j = 1, \ldots, n.$$

5.1 Die Shepard–Methode

Bei dieser Methode versucht man, sich kardinale Funktionen vorzugeben, deren Wert nur vom **inversen** Abstand vom jeweiligen Interpolationspunkt abhängt, also von Funktionen der Form

$$\psi_k(x, y) = \left((x - x_k)^2 + (y - y_k)^2\right)^{-\mu/2}, \qquad \mu > 0.$$

Diese Funktion ψ_k hat einen *Pol* an der Stelle (x_k, y_k), das heißt, sie wird dort **unendlich** groß.

Shepard–Interpolant:

$$\phi(x, y) = \frac{\displaystyle\sum_{k=1}^{n} f_k\, \psi_k(x, y)}{\displaystyle\sum_{k=1}^{n} \psi_k(x, y)} \tag{5.1}$$

Der Trick in (5.1) liegt im "Invers"! Ist nämlich $(x, y) \sim (x_k, y_k)$, dann wird der zugehörige Wert $\psi_k(x, y)$ groß im Vergleich zu den anderen Werten $\psi_j(x, y)$, $j \neq k$, und damit ist

$$(x, y) \sim (x_k, y_k) \qquad \Longrightarrow \qquad \phi(x, y) \sim \frac{f_k\, \psi_k(x, y)}{\psi_k(x, y)} \sim f_k,$$

[51] Denn auch für parametrische Kurven war das *Interpolationsproblem* ja "nur" funktionaler Natur.

weswegen dieses Ding tatsächlich ein Interpolant ist.

Der Shepard–Interpolant ist ein Interpolant.

Wenn das nun also alles so toll und einfach ist, wo ist dann das Problem? Sehen wir uns einmal die kardinalen Funktionen an, nämlich

$$\phi_k(x,y) = \frac{\psi_k(x,y)}{\sum_{k=1}^{n} \psi_k(x,y)},$$

dann fallen uns zwei Dinge auf:

1. Weil die Summe im Nenner größer als die Summe im Zähler ist, erhalten wir, daß $\psi_k \leq 1$ ist – immer und überall.
2. Weil alle beteiligten Funktionen positiv sind, ist auch $\psi_k \geq 0$.

Beispiel 5.1 *Das sehen wir uns nun einmal an, indem wir die* `Octave`*–Funktionen zur Bestimmung eines Shepard–Interpolanten via*

```
octave> f = [ 1, zeros( 1,14 ) ];X = rand( 2,15 );
octave> Y = ShepardGrid( (0:.03:1), (0:.03:1), X, f', 2 );
```

bestimmen; hierbei ist `ShepardGrid` *eine Funktion, die den Shepard–Interpolanten auf einem Rechtecksgitter auswertet und dieser Werte in einer Matrix speichert – diese Funktion basiert auf einer Funktion* `Shepard` *zur Berechnung des Shepard–Interpolanten. Das Ergebnis, das wir dann mittels*

```
octave> PlotMatrixBG( Y );
```

erhalten. Eine typische kardinale Shepard–Funktion, die wir so erhalten, hat die Gestalt wir in Abb. 5.1.

Jetzt aber zurück zu den kardinalen Funktionen? Eigentlich sind die doch ein Traum, denn sie liegen immer zwischen 0 und 1, also im **minimalen** Wertebereich, den so eine kardinale Funktion überhaupt nur haben kann. Trotzdem ist gerade diese "Stärke" in Wirklichkeit ihre Schwäche! Es bedeutet nämlich, daß jede kardinale Funktion ψ_k an den Interpolationspunkten (x_j, y_j) entweder ein Minimum ($j \neq k$) oder ein Maximum ($j = k$) annimmt, weswegen dort die Ableitung[52] den Wert Null haben muß. Damit ist aber jede kardinale Funktion an den Interpolationspunkten "flach" und diese *"Flat Spots"* sieht man den Shepard–Interpolanten leider auch an, siehe Abb. 5.1.

Man hat versucht, die zu vermeiden, indem man bei Shepard–Interpolanten zusätzlich *Ableitungswerte* an den Interpolationspunktion vorschreibt. Dann gibt es zwar keine *Flat Spots* mehr, dafür steht man aber vor dem Problem, aus den diskreten Werten, die man interpolieren möchte, den Wert einer Ableitung zu schätzen. Und das ist weder besonders einfach noch besonders lustig.

[52] Genauer: der Gradient der Funktion.

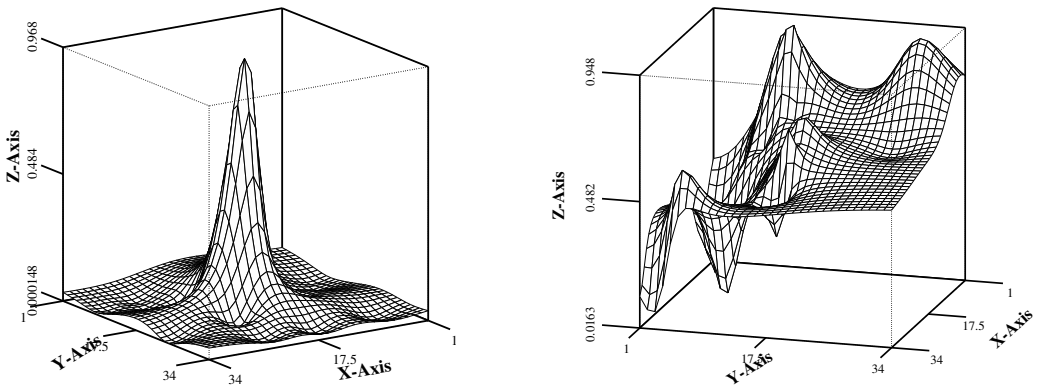

Abbildung 5.1: Eine kardinale Shepard–Funktion (links) und ein Shepard–Interpolant zu zufälligen Werten (rechts). Sieht nicht schlecht aus, hat aber leider erkennbar die *"Flat Spots"*.

5.2 Radiale Funktionen

Trotzdem ist die Idee mit den Funktionen, die im wesentlichen nur vom Abstand von einem Interpolationspunkt abhängen, in keinster Weise schlecht, ganz im Gegenteil! Generell können wir uns ja immer eine *univariate* Funktion $\psi : \mathbb{R} \to \mathbb{R}$ vorgeben und mit dem linearen Raum an die Sache herangehen, der von den Funktionen

$$\phi_k(x,y) = \psi\left(d_k(x,y)\right), \qquad d_k(x,y) = \sqrt{(x-x_k)^2 + (y-y_k)^2}, \qquad k = 1,\ldots,N, \quad (5.2)$$

erzeugt wird. Hier vermeiden wir übrigens den "Fluch der Fläche", denn die Funktionen, mit denen wir interpolieren wollen, hängen ja ganz direkt von den Interpolationspunkten ab. Daß diese Funktionen nun nicht mehr kardinal sind spielt hier keine Rolle mehr – wir haben ja immer noch unsere linearen Gleichungssysteme. Und die sind jetzt sogar besoders "schön" und einfach: Da

$$d_k\left(x_j,y_j\right) = \sqrt{(x_j-x_k)^2 + (y_j-y_k)^2} = d_j\left(x_k,y_k\right),$$

können wir feststellen:

Die Kollokationsmatrix zu radialen Funktionen ist immer symmetrisch.

Bleibt nur die Frage der Fragen: Wie wählen wir die "Ausgangsfunktion" ψ, die natürlich großen Einfluß auf die Qualität des Interpolanten haben wird. Die einfachste Wahl wäre $\psi(x) = x^2$ – man hebt also einfach die Wurzel wieder auf. Die "Basisfunktionen"

$$\phi_k(x,y) = d_k^2(x,y) = (x-x_k)^2 + (y-y_k)^2$$

sind dann *Parabelstücke*, die am Interpolationspunkt (x_k,y_k) verschwinden, also in gewissem Sinne genau das Gegenteil einer kardinalen Funktion. Für unsere `Octave`–Experimente

verwenden wir eine Funktion `DistMat`, die aus einer Matrix

$$X = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{bmatrix}$$

die zugehörige *Distanzmatrix*

$$D = \begin{bmatrix} d_1(x_1, y_1) & \dots & d_n(x_1, y_1) \\ \vdots & \ddots & \vdots \\ d_1(x_n, y_n) & \dots & d_n(x_n, y_n) \end{bmatrix}$$

bestimmt.

Beispiel 5.2 *Versuchen wir's doch einfach mal mit $\psi(x) = x^2$. Dann erhalten wir*

```
octave> X = [ [.5;.5], rand( 2,14 ) ]; F = DistMat ( X ).^2;
octave> a = F \ [ 1 ; zeros( 14,1 ) ];
warning: matrix singular to machine precision, rcond = 2.55834e-18
```

sind also in Schwierigkeiten. Probieren wir's hingegen mit $\psi = 1$, dann sieht es schon besser aus:

```
octave> X = [ [.5;.5], rand( 2,14 ) ]; F = DistMat ( X );
octave> a = F \ [ 1 ; zeros( 14,1 ) ];
octave> [G,xs,ys] = DistGridVec( (0:1/20:1), (0:1/20:1), X );
octave> PlotMatrixBG( GridToMat( G*a, xs, ys ) );
```

In der Tat ist die Bestimmung der Basisfunktion offenbar eine "kritische" Angelegenheit. Aber es gibt doch ein paar "klassische" Fälle:

Beispiel 5.3 ***(Hardy–Multiquadrik:)***
Die Basisfunktion ist $\psi(x) = (x^2 + r^2)^{1/2}$, wobei r ein vom Benutzer frei wählbarer Parameter zur "Lokalisierung" ist. Mit den `Octave`–Befehlen

```
octave> X = GridVec( (0:1/4:1), (0:1/4:1 ) );
octave> [Y,xs,ys] = DistGridVec( (0:1/40:1), (0:1/40:1), X );
octave> f = [ zeros( 12,1 ); 1 ; zeros( 12,1 ) ];
```

wählen wir ein rechteckiges Gitter und betrachten mittels

```
octave> r = 0; F = sqrt( DistMat( X ).^2 + r^2 ); G = sqrt( Y.^2 + r^2 );
octave> PlotMatrixBG( GridToMat( G*(F\f), xs, ys ) );
```

was uns die verschiedenen Wahlen von r so an kardinalen Funktionen liefern. Ein paar Beispiele sind in Abb 5.2 aufgelistet.

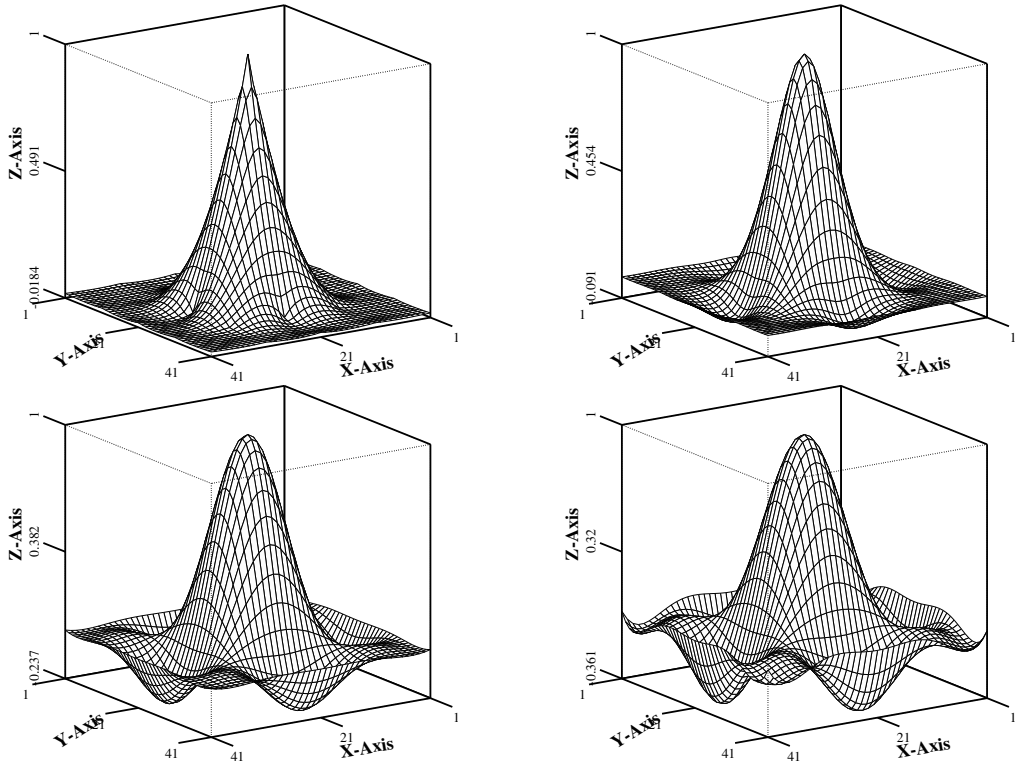

Abbildung 5.2: Kardinale Funktion für Hardysche Multiquadriken auf $[0,1]^2$ zu den Parametern $r = 0$ (oben links), $r = 0.1$ (oben rechts), $r = 0.5$ (unten links) und $r = 1$ (unten rechts).

Beispiel 5.4 ***(Thin–Plate–Splines:)***
Eine andere, eher willkürlich erscheinende Wahl ist die Funktion[53] $\psi(x) = x^2 \log x$. *Ihre Existenzberechtigung rührt daher, daß die zugehörigen Interpolationsfunktionen die Energie einer "dünnen Platte" minimieren, und zwar in demselben Sinne, wie der Spline die Biegeenergie des Streifens minimiert hat. In* `Octave` *bekommen wir den Thin–Plate–Spline wieder auf sehr einfache Art und Weise, nämlich indem wir*

```
octave> D = DistMat( X );
octave> F = D.^2 .* log( D+(D==0) ); G = Y.^2 .* log( Y+(Y==0) );
```

verwenden. Das Ergebnis des Plot–Befehls

```
octave> PlotMatrixBG( GridToMat( G*(F\f), xs, ys ) );
```

ist in Abb. 5.3 zu bewundern.

[53] Ja, hier taucht wirklich ein Logarithmus auf!

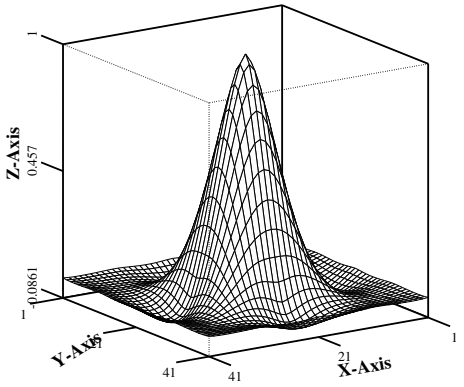

Abbildung 5.3: Kardinaler Thin–Plate–Spline.

Als "krönenden" Abschluß wollen wir schließlich noch diese Funktionen an einem etwas komplexeren Beispiel vergleichen, sozusagen an einer "richtigen" Funktion. Dazu wählen wir, um ein klein wenig Oszillation hineizubekommen, eine Überlagerung von `Expo` und `'Dach`, beispielsweise

```
octave> X = GridVec( (0:1/4:1), (0:1/4:1 ) );
octave> [Y,xs,ys] = DistGridVec( (0:1/40:1), (0:1/40:1), X );
octave> f = Expo( ( X(1,:) .- .5 ).^2 + ( X(2,:) .- .5 ).^2 )' + \
> Dach( 10 * (( X(1,:) .- .3 ).^2 + ( X(2,:) .- .3 ).^2 ) )';
```

Die Funktion ist in Abb. 5.4 geplottet. Zuerst wählen wir nun die Multiquadrik mit $r = 0$, also

```
octave> r = 0; F = sqrt( DistMat( X ).^2 + r^2 ); G = sqrt( Y.^2 + r^2 );
octave> PlotMatrixBG( GridToMat( G*(F\f), xs, ys ) );
```

dann mit $r = 1$

```
octave> r = 1; F = sqrt( DistMat( X ).^2 + r^2 ); G = sqrt( Y.^2 + r^2 );
octave> PlotMatrixBG( GridToMat( G*(F\f), xs, ys ) );
```

was schon wesentlich glattere Ergebnisse liefert, und schließlich setzen wir auch noch den Thin–Plate–Spline an:

```
octave> D = DistMat( X );
octave> F = D.^2 .* log( D+(D==0) ); G = Y.^2 .* log( Y+(Y==0) );
```

Das Ergebnis dieser Interpolationsprozesse ist in Abb. 5.4 aufgelistet. Was man ganz gut erkennen kann, sind die "Zacken", die die Multiquadrik mit $r = 0$ hat[54] und daß

[54]Das sind die "Spitzen" der Wurzelfunktion an der Stelle 0

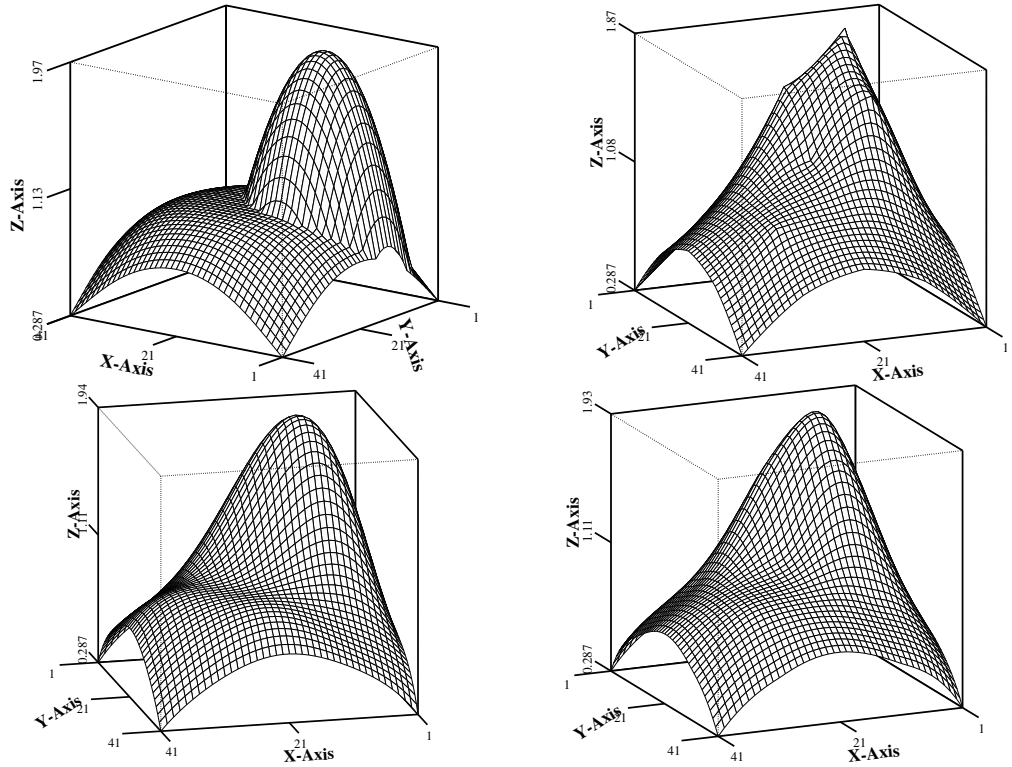

Abbildung 5.4: Interpolation der "Testfunktion" (oben links) mit Multiquadriken mit $r = 0$ (oben rechts) und $r = 1$ (unten links), sowie einem Thin–Plate–Spline, das Ganze auf einem "einfachen" Rechtecksgitter.

der Thin–Plate–Spline irgendwie ganz besonders unter "Zug" steht – das ist wieder die Minimierung des Energiefunktionals. Natürlich kann man auch bei den Multiquadriken noch mit weiteren Parametern experimentieren; generell gibt es zu den Radialen Basisfunktionen natürlich wesentlich mehr Theorie, als man hier anführen kann, beispielsweise

- Efiiziente Verfahren zur Berechnung des Interpolanten an vielen *Gitterpunkten*[55].
- "Passende" Verfahren zur Behandlung der linearen Gleichungssysteme, die sich bei RBFs ergeben.
- Schnelle Methoden zur Auswertung der RBFs; hat man es mal mit mehreren Tausend dieser Funktionen zu tun, dann spielt das sehr wohl eine Rolle.

Eigentlich haben wir hier nur die halbe Wahrheit zu Radialen Basisfunktionen kennengelernt – oftmals hat der Interpolant nämlich noch einen polynomialen Anteil, der allerdings

[55]Die dafür verwendeten Algorithmen nutzen dann auch wirklich aus, daß man es mit Interpolationspunkten zu tun hat, die auf einem Rechtecksgitter liegen.

sehr kleinen Grad hat. Was dahintersteckt ist der Begriff der *bedingten Definitheit*. Im einfachsten Fall, heißt das, daß

$$\sum_{j,k=1}^{n} c_j c_k \, \phi_j \left(x_k, y_k\right) > 0, \qquad \sum_{j=1}^{n} c_j = 0,$$

ist. Das steckt übrigens auch hinter der recht guten Lösbarkeit der Gleichungssysteme.

5.3 Kriging

Jetzt zu einer ganz anderen Interpolationsmethode, die die zu interpolierende Fläche als Realisierung eines *Zufallsprozesses* auffasst. Dieser Zufallsprozess, $Z(x)$ soll gewisse Eigenschaften haben, die wichtigste ist die der *Stationarität*[56]:

$$V\left(Z(x) - Z(y)\right) = 2\gamma\left(x - y\right),$$

wobei die Funktion 2γ als *Variogramm* und die Funktion γ als *Semivariogramm* bezeichnet werden.
Was heißt das? Der *Unterschied* zwischen $Z(x)$ und $Z(y)$ für zwei Punkte[57] $x, y \in \mathbb{R}^2$ ist ja ebenfalls ein Zufallsprozess und dessen *Varianz* (also die Abweichung vom Mittelwert) hängt nur von der Differenz der beiden Punkte ab; der Mittelwert selbst ist dabei von der Position unabhängig! Nun geht man noch einen Schritt weiter und nimmt an, daß der Prozess auch noch *isotrop* wäre, das heißt, daß die Richtung des "Unterschieds" ebenfalls keine Rolle spielt, sondern nur die Distanz, daß also

$$\gamma\left(x - y\right) = \gamma'\left(\left\|x - y\right\|_2\right)$$

ist, und dieses γ' ist es nun, das man zu modellieren versucht. Dazu betrachtet man den "heuristischen" Wert

$$\gamma'(d) = \sum_{\|x-y\|=d} \left(Z(x) - Z(y)\right)^2,$$

also die Varianz des diskreten Prozesses. x und y laufen dabei über alle unsere Interpolationspunkte. Natürlich gibt es für die meisten Werte von d gar kein Paar x, y, so daß $\|x - y\| = d$ ist, man hat es also mit einem univariaten Interpolationsproblem zu tun. Das darf man aber nicht so einfach nach dem Hau–Ruck–Verfahren lösen, sondern man muß beachten, daß das Variogramm immer die Forderung

$$\sum_{j,k=1}^{n} c_j c_k \, \gamma\left(x_j - x_k\right) \leq 0, \qquad \sum_{j=1}^{n} c_j = 0, \tag{5.3}$$

[56] Vorsicht! Wenn man genau hinsieht, dann gibt es eine Vielzahl von Stationaritätsbegriffen mit sehr feinen und subtilen Unterschieden.

[57] Ja, wir haben die Notation geändert!

erfüllen muß[58]; diese Eigenschaft folgt aus der Tatsache, daß das Variogramm eine Varianz darstellt. Nun setzt man einen "Praediktor"

$$P(x_1, \ldots, x_n) := \sum_{j=1}^{n} \lambda_j Z(x_j), \qquad \sum_{j=1}^{n} \lambda_j = 1,$$

an[59] und minimiert, für beliebiges x, den Ausdruck

$$E\left[Z(x) - P(x_1, \ldots, x_n)\right]^2 - 2m\left(\sum_{j=1}^{n} \lambda_j - 1\right)$$

bezüglich $\lambda_1, \ldots, \lambda_n$ und m; das "m" erzwingt, daß die Optimallösung die Bedingung $\sum \lambda_j = 1$ erfüllt. Das führt zu einem Gleichungssystem

$$\begin{aligned} \sum_{k=1}^{n} \lambda_k \gamma(x_j - x_k) + m &= \gamma(x - x_j), \qquad j = 1, \ldots, n \\ \sum_{k=1}^{n} \lambda_k &= 1, \end{aligned}$$

oder

$$\begin{bmatrix} \gamma(x_1 - x_1) & \ldots & \gamma(x_1 - x_n) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \gamma(x_n - x_1) & \ldots & \gamma(x_n - x_n) & 1 \\ 1 & \ldots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \\ m \end{bmatrix} = \begin{bmatrix} \gamma(x - x_1) \\ \vdots \\ \gamma(x - x_n) \\ 1 \end{bmatrix}$$

was wegen (5.3) immer eindeutig lösbar ist. Genau genommen, ist die Lösung ja ein $\lambda(x)$, das eine *Linearkombination* der rechten Seite ist, also eine Linearkombination der Funktionen

$$\gamma(x - x_1), \ldots, \gamma(x - x_n), 1,$$

und wegen des Ansatzes $\sum \lambda_j(x) Z(x_j)$ sind diese Gesellen die zugehörigen *kardinalen Funktionen*, die sich wegen der Annahme $\sum \lambda_j = 1$ auch anständig benehmen. Was auf diese Art minimiert wird, ist die *Kriging–Varianz* oder *Vorhersage–Varianz*

$$\sum_{j=1}^{n} \lambda_j \gamma(x - x_j) - \frac{1}{2} \sum_{j,k=1}^{n} \lambda_j \lambda_k \gamma(x_j - x_k).$$

Setzen wir alles richtig zusammen, so erhalten wir die folgende Beobachtung:

> Kriging ist RBF–Interpolation, bei der die zugrundeliegende Funktion ψ über statistisches Raten definiert wird.

[58] Kommt uns irgendwie bekannt vor, oder?

[59] Die Bedingung an die Werte λ_j sorgt dafür, daß der Mittelwert erhalten bleibt.

5.4 Die "richtige" RBF–Interpolation

Und in der Tat: Das mit den konstanten Funktionen kommt auch bei den Radialen Basisfunktionen ins Spiel! Erweitern wir nämlich die Matrix nicht, nehmen wir also nicht künstlich die konstante Funktion zu unserem Interpolationsraum hinzu, dann haben wir ebenfalls ein kleines Problem, das in Abb. 5.5 deutlich wird. Tatsächlich erhalten die ra-

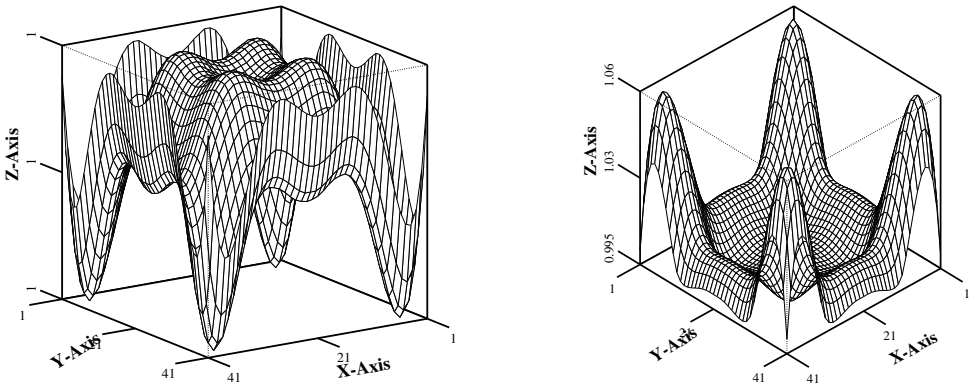

Abbildung 5.5: Interpolation der konstanten Funktion mit radialen Basisfunktionen, links Hardy mit $r = 1$, rechts Thin–Plate–Spline.

dialen Basisfunktionen nämlich **keine** konstanten Funktionen, weswegen man sie halt zu ihrem Glück zwingen müssen, indem wir die konstante Funktion zum Interpolationsraum hinzunehmen und das Gleichungssystem

$$\begin{bmatrix} \psi(\|x_1 - x_1\|) & \dots & \psi(\|x_1 - x_n\|) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \psi(\|x_n - x_1\|) & \dots & \psi(\|x_n - x_n\|) & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \\ a_* \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \\ 0 \end{bmatrix} \tag{5.4}$$

lösen – die interpolierende Funktion ist dann

$$a_* + \sum_{j=1}^{n} a_j \, \psi(x - x_j), \qquad \sum_{j=1}^{n} a_j = 0.$$

Beispiele für derartige Funktionen sind in Abb. 5.6 aufgeführt.

> Der Unterschied zwischen Kriging und RBF–Interpolation besteht in der Wahl der radialen Funktion!

Übrigens läßt weder Kriging noch RBF beliebige "Basisfunktionen" zu! In beiden Fällen muß die Funktion ψ besondere Eigenschaften besitzen, und zwar dieselbe.

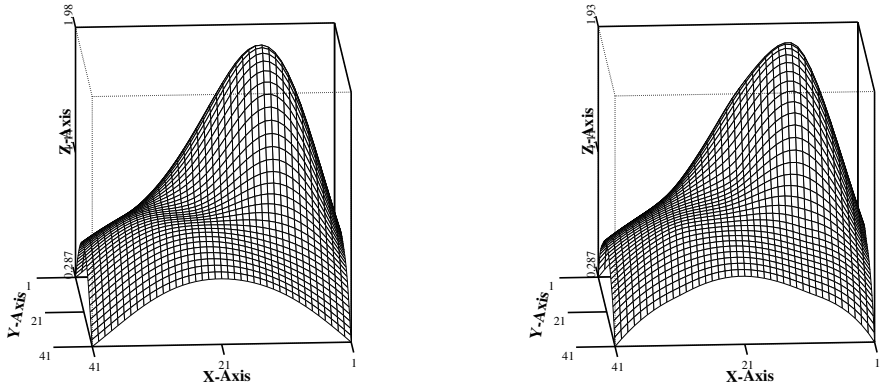

Abbildung 5.6: Zwei radiale Interpolanten mit "Erhaltung der Konstanten". Links eine Multiquadrik mit $r = 0.3$, rechts ein Thin–Plate–Spline.

Ob es vorteilhafter ist, die Funktion ψ aufgrund stochastischem Ratens oder *a priori* aufgrund numerischer Eigenschaften zu bestimmen, ist eine offene Frage.

Uns ist in alten mæren
wunders viel geseit
von Helden lobebæren
von grôzer arebeit

Das Nibelungenlied

Literatur

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK user's guide*, second ed., SIAM, 1995.

[2] J. Bauschinger, *Interpolation*, Encyklopädie der Mathematischen Wissenschaften, Band I, Teil 2, B. G. Teubner, Leipzig, 1900, pp. 800–821.

[3] C. de Boor, *Splinefunktionen*, Lectures in Mathematics, ETH Zürich, Birkhäuser, 1990.

[4] M. Gasca and T. Sauer, *On the history of multivariate polynomial interpolation*, J. Comput. Appl. Math. **122** (2000), 23–35.

[5] G. Nürnberger, *Approximation by spline functions*, Springer–Verlag, 1989.

[6] T. Sauer, *Numerische Mathematik I*, Vorlesungsskript, Universität Erlangen–Nürnberg, Universität Gießen, 2000.

[7] L. L. Schumaker, *Spline funtions: Basic theory*, Pure and Applied Mathematics: A Wiley–Interscience Series of Texts, Monographs and Tracts, John Wiley & Sons, 1981.