Greedy-Verfahren zur Interpolation mit radialen Basisfunktionen

Diplomarbeit

vorgelegt von

Meike Eisenbrandt

aus

Cuxhaven

angefertigt im

Institut für Numerische und Angewandte Mathematik
der Georg–August–Universität Göttingen

2004

Inhaltsverzeichnis

Einleitung

Die Bedeutung radialer Basisfunktionen nimmt im Bereich der angewandten Mathematik immer mehr zu.
In vielen Bereichen der Wissenschaft und Technik werden radiale Basisfunktionen genutzt, um aus gegebenen endlichen Datenmengen $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ für $d \geq 1$ mit dazugehörigen Punkten $\{f_1, \ldots, f_n\} \subset \mathbb{R}$ Funktionen zu rekonstruieren, die alle gegebenen Daten durch $s_f(x_j) = f_j$ mit $j = 1, \ldots, n$ interpolieren.
Für eine eindeutige Lösung solcher Probleme ist es im multivariaten Fall notwendig, daß die Ansatzräume von den Daten abhängen müssen. Dies geschieht bei der Verwendung radialer Basisfunktionen, indem die Interpolante als Linearkombination der Translate einer festen radialen Funktion gebildet wird.
Eine Einführung in das Gebiet der radialen Basisfunktionen wird im ersten Kapitel gegeben.

Bei großen Datensätzen kann es ein ziemlich langwieriges Unterfangen werden, eine Funktion durch punktweise Anpassung zu rekonstruieren. Daher wird immer wieder nach neuen Auswegen gesucht, die Rekonstruktionsgeschwindigkeit zu verbessern.
Dies ist auch Hauptthema dieser Arbeit. Die im zweiten Kapitel untersuchte Idee besteht darin, die Interpolation mit einem Algorithmus zu verbinden, der nur bestimmte Punkte der gegebenen Datenmenge für den Anpassungsvorgang auswählt, dies schrittweise tut und dabei Aufdatierungsverfahren benutzt. Aufgrund seiner Punktewahl und der wachsenden Interpolationsmenge wird der Algorithmus *akkumulierender Greedy-Algorithmus* genannt werden.

Im dritten Kapitel ist die Umsetzung des entwickelten Algorithmus in das Softwaresystem Matlab beschrieben. Das Kapitel ist dabei durch eine grundlegende Einführung in diese Programmiersprache und zahlreiche Erläuterungen zu den dort aufgeführten Programmen so gehalten, daß es auch für Leser

verständlich ist, die Matlab noch nicht kennen.

Mit Hilfe dieser Programme werden die Vorteile des Verfahrens an konkreten Beispielen in Kapitel 4 getestet und ausgewertet.

Die gesamte Arbeit und alle in Kapitel 3 aufgeführten Programme sind auf einer Cd, die dieser Arbeit beigefügt ist, enthalten. Bei Interesse können so die im letzten Kapitel vorgestellten Ergebnisse der Testläufe vom Leser selbst noch einmal nachgestellt oder die Programme als Grundlage für weitere Untersuchungen genutzt werden.

Kapitel 1

Einführung und Überblick

Dieses Kapitel dient der Einführung in das Gebiet der radialen Basisfunktionen. Es werden kurz der Nutzen der Verwendung radialer Basisfunktionen erläutert, erste grundlegende Definitionen gegeben und gezeigt, wie die Interpolation mit radialen Basisfunktionen durchgeführt wird. Desweiteren werden die Energienorm, der Native Space und die Powerfunktion vorgestellt.

1.1 Radiale Basisfunktionen

Sind eine endliche Menge $X = \{x_1, \ldots, x_N\} \subset \Omega \subseteq \mathbb{R}^d$ paarweise verschiedener Punkte und Werte $f_i \in \mathbb{R}, 1 \leq i \leq N$, gegeben, so besteht das zugehörige Interpolationsproblem darin, aus einem Raum $\mathcal{S}$ von Funktionen $s : \mathbb{R}^d \to \mathbb{R}$ eine Funktion zu finden, für die gilt:

$$s_f(x_i) = f_i \text{ für } 1 \leq i \leq N.$$

Nach dem Satz von Mairhuber und Curtis läßt sich nur im eindimensionalen Fall (oder wenn die zugrunde liegende Menge nur aus einem Punkt besteht) ein datenunabhängiger Raum $\mathcal{S}$ finden (Vgl. hierzu [3]). Um im multivariaten Raum interpolieren zu können, muß der Raum $\mathcal{S}$ von den Datenpunkten abhängen. Dies gewährleistet die Verwendung *radialer Basisfunktionen*. Sie sind folgendermaßen definiert:

Definition 1.1 *Eine univariate, reellwertige Funktion $\phi : [0, \infty) \to \mathbb{R}$ heißt* **radiale Basisfunktion**, *falls sie als symmetrische multivariate Funktion $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ genutzt wird, wobei*

$$\Phi(x, y) = \phi(\|x - y\|_2) \text{ für alle } x, y \in \mathbb{R}^d. \tag{1.1}$$

Φ *wird - aufgrund des hier noch nicht wichtigen Zusammenhangs mit Hilberträumen - der* **reproduzierende Kern** *genannt.*
$\|.\|_2$ *bezeichnet hierbei die euklidische Norm im* $\mathbb{R}^d$.

Es sind also Funktionen, deren Funktionswerte nur vom euklidischen Abstand der Argumente abhängig sind. Sie lassen sich unterteilen in *positiv definite* und *bedingt positiv definite* radiale Basisfunktionen.
Die in der Definition hierzu zum ersten Mal in dieser Arbeit auftauchende Bezeichnung $I\!P_m^d$ stellt die Menge aller Polynome in d Variablen mit einem Gesamtgrad kleiner als m dar, die Dimension des Raumes errechnet sich als $Q = dim\ I\!P_m^d = \binom{m-1+d}{d}$.

Definition 1.2 *Eine stetige Funktion* $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *heißt* **bedingt positiv definit** *der Ordnung* m *auf* $\mathbb{R}^d$*, wenn bei beliebiger Wahl von finiten Untermengen* $X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^d$ *aus* N *paarweise verschiedenen Punkten und für alle Vektoren* $\alpha = \{\alpha_1, \ldots, \alpha_N\} \in \mathbb{R}^N$ *mit*

$$\sum_{j=1}^{N} \alpha_j p(x_j) = 0 \text{ für alle } p \in I\!P_m^d \tag{1.2}$$

die Ungleichung

$$\alpha^T A_{X,\Phi} \alpha = \sum_{j,k=1}^{N} \alpha_j \alpha_k \Phi(x_j, x_k) \geq 0 \tag{1.3}$$

erfüllt ist und die Gleichheit nur für $\alpha = 0$ *eintritt.*
Ist Φ *bedingt positiv der Ordnung 0, so wird die Funktion als* **positiv definit** *bezeichnet.*

In der folgenden Tabelle sind die in dieser Arbeit relevanten radialen Funktionen $\phi(r)$ mit $r := \|x-y\|_2, x, y \in \mathbb{R}^d$ aufgeführt. Die beiden erst genannten Funktionen sind bedingt positiv definit, die übrigen Funktionen positiv definit.

Name	$\phi(r)$	BPD Ordn. m	max. Dim. d
Thin Plate Splines	$r^2 \log r$	2	∞
Multiquadrics	$\sqrt{1+r^2}$	1	∞
Inverse Multiquadrics	$(1+r^2)^{-1/2}$	0	∞
Gaußglocke	$\exp(-r^2)$	0	∞
Wendlandfunktion	$(1-r)_+^4(1+4r)$	0	3

1.2 Skalierung radialer Basisfunktionen

In späteren Kapiteln dieser Arbeit wird mit durch einen Parameter $c > 0$ skalierten radialen Basisfunktionen

$$\Phi_c(.) = \Phi(./c)$$

gearbeitet werden. Dies wird den Vorteil haben, daß die "Breite" der Funktion beeinflußbar sein wird, und somit bei nicht-glatten Funktionen durch einen großen Skalierungsfaktor der Interpolationsfehler gesenkt werden kann. Bei analytischen Funktionen ist allerdings dabei darauf zu achten, daß die Kondition der Interpolationsmatrix

$$\text{cond}(A) = \|A\|\|A^{-1}\|$$

durch ein zu groß gewähltes c nicht zu schlecht wird, wenn A mit Φ_c gebildet wird und die Einträge $\Phi_c(x_j, x_k)$ hat.

1.3 Interpolation mit radialen Basisfunktionen

Im Falle multivariater Funktionen sieht das Interpolationsproblem unter Verwendung radialer Basisfunktionen wie folgt aus:
Gegeben seien eine endliche Menge $X = \{x_1, \ldots, x_N\} \subset \Omega \subseteq \mathbb{R}^d$ paarweise verschiedener Punkte und reelle Werte $f_k \in \mathbb{R}, 1 \leq k \leq N$. Mit Hilfe einer radialen Basisfunktion $\Phi : \Omega \times \Omega \to \mathbb{R}$ läßt sich nun ein datenabhängiger Funktionenraum konstruieren

$$\mathcal{S}_{X,\Phi} := \text{span}\{\Phi(x_k, x) : 1 \leq k \leq N\} \tag{1.4}$$

und gesucht wird bei der Verwendung einer positiv definiten radialen Basisfunktion eine Funktion $s_{f,X} \in \mathcal{S}_{X,\Phi}$, die folgendes Gleichungssystem löst:

$$f(x_k) \stackrel{!}{=} s_{f,X}(x_k) = \sum_{j=1}^{N} \alpha_j \Phi(x_k, x_j) \quad \text{für } k = 1, \ldots, N \; . \tag{1.5}$$

In Matrix-Vektor-Schreibweise hat das Problem folgende Form:

$$A_X \cdot \alpha_X = f_X \; , \tag{1.6}$$

mit der Interpolationsmatrix $A_X = (\Phi(x_k, x_j))_{1\le j,k\le N}$, den zu interpolierenden Funktionswerten $f_X = (f(x_1), \ldots, f(x_N))^T$ und dem Lösungsvektor $\alpha_X = (\alpha_1 \ldots, \alpha_N)^T$.
Die Lösung des Interpolationsproblems ist im Falle positiv definiter Basisfunktionen eindeutig, da die Interpolationsmatrix dann auch invertierbar ist, die Determinante der Matrix $A_{X,\Phi}$ also nicht verschwindet.
Bei bedingt positiv definiten radialen Basisfunktionen der Ordnung $m > 0$ hat die Interpolante für $k = 1, \ldots, N$ die Form

$$s_{f,X}(x_k) = \sum_{j=1}^{N} \alpha_j \Phi(x_j, x_k) + \sum_{l=1}^{Q} \beta_l p_l(x_k) \tag{1.7}$$

mit einem Vektor $\alpha \in \mathbb{R}^N$, der folgende zusätzliche Bedingung erfüllt:

$$\sum_{j=1}^{N} \alpha_j p_i(x_j) = 0, 1 \le i \le Q, \text{ für alle } p \in I\!P_m^d \,, \tag{1.8}$$

einem Vektor $\beta \in \mathbb{R}^Q$ und $p_1, \ldots, p_Q$ als eine Basis des $I\!P_m^d$. Der datenabhängige Raum hat die Form

$$\mathcal{S}_{X,\Phi} := \left\{ \sum_{j=1}^{M} \alpha_j \Phi(x, x_j) \text{ mit } (1.7) \right\} + I\!P_m^d \,. \tag{1.9}$$

Hier sei kurz angemerkt, daß im Falle $m = 0$ der Raum $I\!P_m^d$ Null und die Bedingung (1.8) leer ist, das System sich also auf den positiv definiten Fall reduziert.
Das Interpolationsproblem (1.7) hat in Matrizenschreibweise folgende Gestalt:

$$\begin{pmatrix} A_X & P_X \\ P_X^T & 0 \end{pmatrix} \begin{pmatrix} \alpha_X \\ \beta_X \end{pmatrix} = \begin{pmatrix} f_X \\ 0 \end{pmatrix}, \tag{1.10}$$

mit den Matrizen

$$\begin{aligned} A_X &= (\Phi(x_i, x_j))_{1\le i,j\le N} \\ P_X &= (p_k(x_j))_{1\le k\le Q, 1\le j\le N} \end{aligned}$$

und den Vektoren

$$\begin{aligned} \alpha_X &= (\alpha_1, \ldots, \alpha_N)^T \\ \beta_X &= (\beta_1, \ldots, \beta_Q)^T \\ f_X &= (f(x_1), \ldots, f(x_N))^T. \end{aligned}$$

Die eindeutige Lösbarkeit des Interpolationsproblems (1.10) zeigt der folgende Satz.

Theorem 1.1 *Ist $\Phi : \mathbb{R}^d \to \mathbb{R}$ eine bedingt positiv definite radiale Basisfunktion der Ordnung m und enthält die aus paarweise verschiedenen Punkten bestehende Menge $X = \{x_1, \ldots, x_N\}$ eine unisolvente Teilmenge, das heißt, die Matrix P_X ist injektiv, dann ist das System (1.10) eindeutig lösbar.*

Beweis. Aus den Gleichungen $A_X\alpha_X + P_X\beta_X = 0$ und $P_X^T\alpha_X = 0$ folgt $\alpha_X^T A_X\alpha_X + \alpha_X^T P_X\beta_X = \alpha_X^T A_X\alpha_X = 0$ und aufgrund bedingt positiver Definitheit von Φ gerade $\alpha_X = 0$. Da P_X injektiv ist, erhält man dann auch $\beta_X = 0$.

□

1.4 Die Energienorm

Ein wichtiges Instrument wird die *Native Space Norm* $\|.\|_{\mathcal{N}_\Phi}$ sein:
Für eine positiv definite Funktion der Form (1.5) führt folgende Definition eine Norm und für eine bedingt positiv definite Funktion der Form (1.7) eine Seminorm mit dem Nullraum $\mathbb{P}_m^d$ ein:

$$\|s_{f,X}\|_{\mathcal{N}_\Phi}^2 := \alpha^T A_X \alpha = \sum_{i,j=1}^{N} \alpha_i\alpha_j\Phi(x_i, x_j) \; . \tag{1.11}$$

Das (Semi-)Skalarprodukt zweier solcher Funktionen $f, g : \Omega \to \mathbb{R}^d$ auf zwei endlichen Teilmengen $X = \{x_1, \ldots, x_M\}$ und $Y = \{y_1, \ldots, y_N\}$ von $\Omega \subset \mathbb{R}^d$ ist

$$\langle s_{\alpha,X}, s_{\beta,Y}\rangle_{\mathcal{N}_\Phi} := \sum_{i=1}^{M}\sum_{j=1}^{N} \alpha_i\beta_j\Phi(x_i, y_j) \; , \tag{1.12}$$

vorausgesetzt, die beiden Koeffizientenvektoren $\alpha \in \mathbb{R}^M, \beta \in \mathbb{R}^N$ erfüllen die Nebenbedingung (1.8) sinngemäß.

Verwendet man als radiale Basisfunktion die *Thin Plate Splines* $\Phi(x, y) = \|x-y\|_2 \log \|x-y\|_2$ im $\mathbb{R}^2$, so beschreibt der Wert $\|s_{f,X}\|_{\mathcal{N}_\Phi}^2$ die Energie, die nötig ist, um ein dünnes Blech in die Form der Interpolante $s_{f,X}$ zu biegen. Daher wird sie auch *Energienorm* genannt.
Der *Native Space* selbst ist folgendermaßen definiert:

Definition 1.3 *Der* **Native Space** $\mathcal{N}_\Phi$ *für eine bedingt positiv definite Funktion* Φ *der Ordnung* m *auf* $\mathbb{R}^d$ *ist die direkte Summe*

$$\mathcal{N}_\Phi := I\!P_m^d \oplus \mathcal{H} \ ,$$

wobei $\mathcal{H}$ *den Hilbertraum bezeichnet, der sich aus der Vervollständigung aller Funktionen der Form* $s_{f,X}$ *bezüglich dem inneren Produkt* $\langle \ , \ \rangle_{\mathcal{N}_\Phi}$*, mit der Nebenbedingung*

$$\sum_{j=1}^{N} \alpha_j p_i(x_j) = 0 \ , 1 \le i \le Q \ , \ \textit{für alle } p \in I\!P_m^d \ ,$$

ergibt.

Diese kurze Einführung bezüglich des Native Space möge hier genügen, weiteres zu diesem Thema kann bei Interesse zum Beispiel in [4] nachgelesen werden.

Ein Schlüsselpunkt dieser Arbeit ist folgende Orthogonalitätsbeziehung und die sich daraus ergebende Konsequenz der Energieaufteilung:

Theorem 1.2 *Sei* $s_{f,X}$ *die Interpolante zu einer Funktion* $f \in \mathcal{N}_\Phi$ *auf einer endlichen Teilmenge* $X \subset \Omega$*. Dann gilt die Orthogonalitätsbeziehung*

$$\langle s_{f,X}, f - s_{f,X} \rangle_{\mathcal{N}_\Phi} = 0 \tag{1.13}$$

und der Satz des Pythagoras impliziert

$$\|f\|^2_{\mathcal{N}_\Phi} = \|f - s_{f,X}\|^2_{\mathcal{N}_\Phi} + \|s_{f,X}\|^2_{\mathcal{N}_\Phi} \ . \tag{1.14}$$

Beweis. Die Interpolante zur Funktion $f \in \mathcal{N}_\Phi$ hat die Form

$$s_{f,X} = \sum_{j=1}^{N} \alpha_j \Phi(x_j, .) + \sum_{l=1}^{Q} \beta_l p_l(x)$$

und erfüllt für $m > 0$ die Zusatzbedingung (1.8). Diese in (1.13) eingesetzt ergibt

$$\begin{aligned} \langle s_{f,X}, f - s_{f,X} \rangle_{\mathcal{N}_\Phi} &= \left\langle \sum_{j=1}^{N} \alpha_j \Phi(x_j, .), f - s_{f,X} \right\rangle_{\mathcal{N}_\Phi} \\ &= \sum_{j=1}^{N} \alpha_j \underbrace{(f - s_{f,X})(x_j)}_{= 0, \text{ Int.auf } X} \\ &= 0 \ , \end{aligned}$$

und es folgt

$$\begin{aligned}
\|f\|^2_{\mathcal{N}_\Phi} &= \|s_{f,X} + f - s_{f,X}\|^2_{\mathcal{N}_\Phi} \\
&= \langle s_{f,X} + f - s_{f,X}, s_{f,X} + f - s_{f,X}\rangle_{\mathcal{N}_\Phi} \\
&= \|s_{f,X}\|^2_{\mathcal{N}_\Phi} + 2\underbrace{\langle s_{f,X}, f - s_{f,X}\rangle_{\mathcal{N}_\Phi}}_{=0} + \|f - s_{f,X}\|^2_{\mathcal{N}_\Phi} \\
&= \|f - s_{f,X}\|^2_{\mathcal{N}_\Phi} + \|s_{f,X}\|^2_{\mathcal{N}_\Phi} \ .
\end{aligned}$$

□

Die *Energie* der Funktion $f \in \mathcal{N}_\Phi$ teilt sich also auf in die der Interpolante $s_{f,X}$ und die der Fehlerfunktion $f - s_{f,X}$.

1.5 Die Powerfunktion

Führt man das System

$$\begin{pmatrix} A_X & P_X \\ P_X^T & 0 \end{pmatrix} \begin{pmatrix} u_X(x) \\ v_X(x) \end{pmatrix} = \begin{pmatrix} \Phi_X(x) \\ p(x) \end{pmatrix} \tag{1.15}$$

mit den Notationen

$$\begin{aligned}
\Phi_X(x) &= (\Phi(x, x_1), \ldots, \Phi(x, x_N))^T \\
p(x) &= (p_1(x), \ldots, p_Q(x))^T \\
u_X(x) &= (u_1(x), \ldots, u_N(x))^T \\
v_X(x) &= (v_1(x), \ldots, v_Q(x))^T
\end{aligned}$$

ein, so erkennt man sofort die Polynome reproduzierende Lagrange-Basis

$$\begin{aligned}
u_i(x_j) &= \delta_{ij}, 1 \le i, j \le N \\
p_k(x) &= \sum_{j=1}^{N} u_j(x) p_k(x_j), 1 \le k \le Q \ ,
\end{aligned}$$

und

$$f \mapsto s_{f,X}(x) = \sum_{j=1}^{N} u_j(x) f(x_j)$$

ist ein lineares Funktional, ebenso das in $\mathcal{N}_\Phi^*$ (Dualraum des Native Space) liegende Fehlerfunktional

$$\epsilon_x : f \mapsto f(x) - s_{f,X}(x) \tag{1.16}$$

mit $\epsilon_x(p) = 0$ für alle $p \in I\!P_m^d$.

Definition 1.4 *Die* **Powerfunktion** P_X *ist definiert durch*

$$\begin{aligned} P_X^2(x) &:= \|\epsilon_x\|_{\mathcal{N}_\Phi^*}^2 && (1.17)\\ &= \left\| \Phi(x,.) - \sum_{j=1}^{N} u_j(x)\Phi(x_j,.) \right\|_{\mathcal{N}_\Phi}^2 && (1.18)\\ &= \Phi_X(x,x) - 2u_X^T(x)\Phi_X(x) + u_X^T(x)A_X u_X(x) && (1.19) \end{aligned}$$

für alle $x \in \Omega$.

Es läßt sich eine erste Fehlerabschätzung machen:

Theorem 1.3 *Mit der Powerfunktion ist der Fehler bei Rekonstruktion einer beliebigen Funktion* $f \in \mathcal{N}_\Phi$ *durch die Interpolante* $s_{f,X}$ *begrenzt durch*

$$|f(x) - s_{f,X}(x)| \le P_X(x)\|f\|_{\mathcal{N}_\Phi} \quad \textit{für alle } x \in \Omega\ . \tag{1.20}$$

Beweis. Für jede beliebige Funktion $f \in \mathcal{N}_\Phi$ gilt

$$\begin{aligned} |f(x) - s_{f,X}(x)| &= |\epsilon_x(f)| = \\ &\le \|\epsilon_x\|_{\mathcal{N}_\Phi^*}\|f\|_{\mathcal{N}_\Phi} = P_X(x)\|f\|_{\mathcal{N}_\Phi}\ . \end{aligned}$$

□

Zum Abschluß dieses Abschnitts seien noch zwei Konsequenzen des System (1.15) genannt, die hier noch nicht weiter relevant erscheinen, die sich für das folgende Kapitel aber als nützlich erweisen.
Aus dem System (1.15) ergeben sich die beiden Gleichungen

$$\begin{aligned} A_X u_X(x) + P_X v_X(x) &= \Phi_X(x)\\ P_X^T u_X(x) + 0 &= p(x)\ . \end{aligned}$$

Multipliziert man die obere Gleichung mit $u_X^T(x)$ und setzt die untere Gleichung dort ein, so erhält man

$$u_X^T(x)A_X u_X(x) + p^T(x)v_X(x) = u_X^T(x)\Phi_X(x)\ ,$$

und für (1.19) ergibt sich daraus:

$$P_X^2(x) = \Phi_X(x,x) - u_X^T(x)\Phi_X(x) - v_X^T(x)p(x)\ . \tag{1.21}$$

Eine weitere Konsequenz des Systems (1.15) ist folgende Gleichung:

$$\left(u_X^T(x), v_X^T(x)\right) \begin{pmatrix} A_X & P_X \\ P_X^T & 0 \end{pmatrix} \begin{pmatrix} u_X(x) \\ v_X(x) \end{pmatrix} = u_X^T(x)\Phi_X(x) + v_X^T(x)p(x)\ . \tag{1.22}$$

Unter [5] können weitere Ausführungen zur Powerfunktion nachgelesen werden.

Kapitel 2

Der Greedy-Algorithmus

Da die Matrix $A_{X,\Phi}$ in der Praxis oft sehr groß ausfällt und das Lösen des obigen Interpolationsproblems so sehr viel Zeit in Anspruch nimmt, wird versucht, durch Interpolation auf einer Teilmenge in sehr viel schnellerer Zeit eine ausreichend gut angenäherte Lösung für solch ein größeres Problem zu finden.
Die Grundidee dieser Arbeit besteht nun darin, sich die Aufteilung der Energie zu Nutze zu machen und mit Hilfe eines iterativen Algorithmus auf einer langsam wachsenden Teilmenge zu interpolieren und dabei der Fehlerfunktion je Iterationsschritt möglichst viel Energie zu entziehen, so daß die Interpolante - unter geeigneten Voraussetzungen - möglichst schnell bezüglich der Energienorm gegen eine gegebene Funktion $f \in \mathcal{N}_\Phi$ konvergiert.

2.1 Die Interpolationsmenge

Seien wiederum eine endliche Menge $X = \{x_1, \ldots, x_N\} \subset \Omega \subseteq \mathbb{R}^d$ paarweise verschiedener Punkte und reelle Werte $f_k \in \mathbb{R}, 1 \leq k \leq N$ gegeben. Dann wird die Interpolationsmenge nun folgendermaßen gewählt:
Zur Initialisierung wird eine Teilmenge $Y_0 \subseteq X$ gewählt, deren Ordnung der Dimension des Polynomraumes entspricht[1], und auf dieser Teilmenge interpoliert. Für $j = 1, 2, \ldots$ wird je Iterationsschritt der Punkt $\tilde{x} \in X$ zu der

[1] Im Falle der Verwendung der Thin Plate Splines ist darauf zu achten, daß P_{Y_0} injektiv ist.

Teilmenge hinzugenommen, an dem noch der größte Fehler vorliegt, der Abstand zwischen den Werten f_k und der Interpolante also (noch) maximal ist[2]. Zu der neuen Teilmenge wird dann die neue Interpolante berechnet.
Das Verfahren bricht ab, sobald mit der auf Y_j berechneten Interpolanten s_{f,Y_j} auf X eine maximal gewünschte Fehlergrenze erreicht wird.

Diese Vorgehensweise hat folgende Vorteile:

- Durch Interpolation nimmt die Fehlerfunktion an den Stellen x_k, mit $1 \leq k \leq N$, den Wert Null an, so daß im nächsten Iterationschritt automatisch ein anderer Punkt ausgewählt wird.
- Es ist keine Vorbearbeitung der großen Datenmenge notwendig.
- Der Interpolanten wird je Iterationsschritt möglichst viel Energie hinzugefügt, so daß sie - unter geeigneten Voraussetzungen - möglichst schnell bezüglich der Native-Space-Norm gegen eine gegebene Funktion $f \in \mathcal{N}_\Phi$ konvergiert.

Der letzte Punkt bedarf noch einer Erläuterung: wie später in diesem Kapitel bewiesen werden wird, ist die Energie der Interpolante $s_{f,Y\cup\{y\}}$ zu der um einen Punkt $y \notin Y$ erweiterten Interpolationsmenge $Y \cup \{y\}$ gegeben durch

$$\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} = \|s_{f,Y}\|^2_{\mathcal{N}_\Phi} + \frac{(f(y) - s_{f,Y}(y))^2}{P^2_Y(y)} \tag{2.1}$$

und demzufolge die Abnahme der Fehlerfunktion durch

$$\|f - s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} = \|f - s_{f,Y}\|^2_{\mathcal{N}_\Phi} - \frac{(f(y) - s_{f,Y}(y))^2}{P^2_Y(y)} \,. \tag{2.2}$$

Der Zuwachs der Energie der Interpolanten ist bei einer Punkterweiterung der Interpolationsmenge also maximal, wenn beim zweiten Term der rechten Seite der Gleichung (2.1) der Zähler maximal und der Nenner minimal ist. Dies läßt sich, wenn überhaupt, nicht so einfach gleichzeitig realisieren. Die Maximierung des Zählers erfolgt zwar durch die bereits beschriebene Wahl von $\tilde{x}$, dadurch ist aber nicht automatisch die maximal mögliche Minimierung des Nenners gewährleistet. Hierfür bedarf es noch weiterer Bedingungen an das gesamte Verfahren - und ob dabei die Maximierung des Zählers erhalten bleiben kann, ist nicht sicher.

[2]Die lineare Unabhängigkeit und die (bedingt) positive Definitheit des Interpolationssystems bleibt erhalten, da $\tilde{x} \in X$.

Deshalb wird in dieser Arbeit "nur" die Maximierung des Zählers der Gleichung (2.1) genutzt werden, um der Interpolanten je Iterationsschritt möglichst viel Energie zuzufügen, beziehungsweise der Fehlerfunktion je Iterationschritt möglichst viel Energie zu entziehen.

Dieser Algorithmus wird aufgrund der Eigenschaften der langsam wachsenden Teilmenge und der "Gierigkeit" nach dem Punkt mit dem größten verbliebenen Fehler *akkumulierender Greedy-Algorithmus* genannt.
Eine genaue Formulierung des Algorithmus folgt in Abschnitt 2.3.

2.2 Rang-1-Erweiterungen

Die Interpolante s_{f,Y_j} kann entweder mit Hilfe der Cholesky-Zerlegung[3] oder direkt über die Berechnung der Inversen der Koeffizientenmatrix $A_{Y_j,\Phi}$ ermittelt werden.
Da die Interpolationsmenge im Algorithmus je Iterationsschritt um einen Punkt erweitert wird, muß auch entweder die Cholesky-Zerlegung oder die Inverse B um diesen einen Punkt, je nachdem, welche Methode genutzt wird, aktualisiert werden. Um an Operationen zu sparen, wird eine rekursive Vorgehensweise verwendet.
Es folgen Sätze zu beiden Methoden.

Theorem 2.1 (Rang-1-Erweiterung einer Cholesky-Zerlegung)
Sei zu einer positiv definiten symmetrischen Matrix $A \in \mathbb{R}^{N\times N}$ die Cholesky-Zerlegung durch $A = L^T L$ gegeben.
Wird die Matrix A um einen Punkt erweitert, der das System weiterhin linear unabhängig und positiv definit bleiben läßt, dann hat die aktualisierte Cholesky-Zerlegung für die erweiterte Matrix

$$\tilde{A} = \begin{pmatrix} A & a \\ a^T & g \end{pmatrix}$$

mit einem Vektor a und einem Skalar g, die Form:

$$\tilde{A} = \tilde{L^T}\tilde{L} = \begin{pmatrix} L^T & 0 \\ l^T & \lambda \end{pmatrix} \begin{pmatrix} L & l \\ 0 & \lambda \end{pmatrix}$$

mit $l := (L^T)^{-1}a$ und $\lambda := \sqrt{g - l^T l}$ mit $g \geq l^T l$ für $\tilde{A}$ positiv definit.

[3]Eine Cholesky-Zerlegung ist für bedingt positive Matrizen nicht ohne eine vorherige Projektion auf einen Unterraum, die die positive Definitheit der Matrix sichert, möglich. Daher wird auf die Zerlegung nur im Falle positiv definiter Matrizen weiter eingegangen.

Beweis. Für eine positiv definite symmetrische Matrix $A = L^T L$, $l := (L^T)^{-1}a$ und $\lambda := \sqrt{g - l^T l}$ läßt sich

$$\tilde{A} = \begin{pmatrix} A & a \\ a^T & g \end{pmatrix}$$

schreiben als

$$\tilde{A} = \begin{pmatrix} A & L^T(L^T)^{-1}a \\ a^T L^{-1} L & l^T l + g - l^T l \end{pmatrix} = \begin{pmatrix} L^T L & L^T l \\ l^T L & l^T l + \lambda^2 \end{pmatrix} = \begin{pmatrix} L^T & 0 \\ l^T & \lambda \end{pmatrix} \begin{pmatrix} L & l \\ 0 & \lambda \end{pmatrix} .$$

□

Die Inverse der Koeffizientenmatrix läßt sich sowohl für positiv definite, als auch für bedingt positiv definite Matrizen ohne zusätzliche Einführungen von Projektionen rekursiv berechnen.

Theorem 2.2 (Rang-1-Erweiterung der Inversen einer positiv definiten Matrix) *Sei zu einer positiv definiten symmetrischen Matrix $A \in \mathbb{R}^{N \times N}$ die Inverse gegeben als $A^{-1} = B$. Wird die Matrix A um einen Punkt erweitert, der das System weiterhin linear unabhängig und positiv definit bleiben läßt, dann gilt für die erweiterte Matrix*

$$\tilde{A} = \begin{pmatrix} A & a \\ a^T & g \end{pmatrix}$$

mit einem Vektor a und einem Skalar g:

$$\tilde{A}^{-1} = \begin{pmatrix} B - xb^T & b \\ b^T & \beta \end{pmatrix}$$

mit

$$\begin{aligned} x &:= B.a \\ \beta &:= \tfrac{1}{g - a^T x} \\ b &:= -\beta x \, . \end{aligned}$$

Beweis. Durch $A^{-1} = B$ und

$$\begin{pmatrix} A & a \\ a^T & g \end{pmatrix} \begin{pmatrix} C & b \\ b^T & \beta \end{pmatrix} = \begin{pmatrix} E & 0 \\ 0 & 1 \end{pmatrix}$$

ergeben sich:

$$\begin{array}{rcrcrcl} AC & + & ab^T & = & E & \Rightarrow & C = B - Bab^T \,, \\ Ab & + & \beta a & = & 0 & \Rightarrow & b = -\beta Ba \,, \\ a^T C & + & gb^T & = & 0 & \Rightarrow & b^T = -\frac{1}{g - a^T Ba} a^T B \,, \\ a^T b & + & g\beta & = & 1 & \Rightarrow & \beta = \frac{1}{g - a^T Ba} \,, \end{array}$$

und man erhält

$$\begin{pmatrix} A & a \\ a^T & g \end{pmatrix}^{-1} = \begin{pmatrix} B - xb^T & b \\ b^T & \beta \end{pmatrix} .$$

□

Im Falle bedingt positiv definiter radialer Basisfunktionen sieht das Aktualisieren etwas komplizierter aus:

Theorem 2.3 (Rang-1-Erweiterung der Inversen einer bedingt positiv definiten Matrix) *Seien eine bedingt positiv definite symmetrische Matrix M und deren Inverse gegeben durch*

$$M = \begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \text{ und } M^{-1} = \begin{pmatrix} B & C \\ C^T & D \end{pmatrix}$$

mit $A, B \in \mathbb{R}^{N \times N}$, $P, C \in \mathbb{R}^{N \times Q}$ und $D \in \mathbb{R}^Q$.
Wird die Matrix M um einen Punkt erweitert, ohne daß das System die lineare Unabhängigkeit und bedingt positive Definitheit verliert, so hat die Inverse zu der um einen Punkt erweiterten Matrix

$$\tilde{M} = \begin{pmatrix} A & a & P \\ a^T & g & p^T \\ P^T & p & 0 \end{pmatrix}$$

mit Vektoren a, p und einem Skalar g, folgende Form:

$$\tilde{M}^{-1} = \begin{pmatrix} \tilde{B} & \tilde{C} \\ \tilde{C}^T & \tilde{D} \end{pmatrix}$$

mit den Matrizen

$$\tilde{B} := \begin{pmatrix} B - xb^T & b \\ b^T & \beta \end{pmatrix} , \; \tilde{C} := \begin{pmatrix} C - by^T \\ c^T \end{pmatrix} , \; \tilde{D} := \left(D - yc^T\right)$$

und

$$\begin{aligned} x &:= Ba + Cp \\ y &:= C^T a + Dp \\ \beta &:= \tfrac{1}{g - a^T x - p^T y} \\ b &:= -\beta x \\ c &:= -\beta y \ . \end{aligned}$$

Beweis. Sei E die Einheitsmatrix. Mit Hilfe der durch M und M^{-1} gegebenen Gleichungen

$$\begin{aligned} AB + PC^T &= E \\ AC + PD &= 0 \\ P^T B &= 0 \\ P^T C &= E \end{aligned}$$

und der Verwendung der im Satz gegebenen Definitionen von x, y, β, b und c ergeben sich folgende Gleichungssysteme:

$$\begin{aligned} 0 &= -Axb^T + Axb^T \\ &= -Axb^T + ABab^T + ACpb^T \\ &= -Axb^T + (E - PC^T)ab^T + \underbrace{AC}_{=-PD} pb^T \ \Leftrightarrow \\ E &= -Axb^T + ab^T - \underbrace{PC^T ab^T - PDpb^T}_{=-Pyb^T} + \underbrace{AB + PC^T}_{=E} \\ &= A(B - xb^T) + ab^T + P(C^T - yb^T) \\ &= (A, a, P)\left(B - xb^T, b^T, C^T - yb^T\right)^T \ , \end{aligned}$$

$$\begin{aligned} 0 &= -\beta a + \beta a \\ &= -\beta(AB + PC^T)a + \beta a - \beta ACp + \beta ACp \\ &= Ab + \beta a + Pc \\ &= (A, a, P)\,(b, \beta, c)^T \ , \end{aligned}$$

$$
\begin{aligned}
0 &= Aba^TC + Abp^TD^T - Aba^TC - Abp^TD^T \\
&= -\beta ABaa^TC - \beta ABap^TD^T - \beta ACpa^TC - \beta ACpp^TD \\
&\quad -Aba^TC - Abp^TD^T \\
&= ABac^T + ACpc^T - Aba^TC - Abp^TD^T \\
&= \underbrace{AC + PD}_{=0} - Aby^T + ac^T - Pyc^T \\
&= A(C - by^T) + ac^T + P(D - yc^T) \\
&= (A, a, P)\left(C - by^T, c^T, D - yc^T\right)^T \ ,
\end{aligned}
$$

$$
\begin{aligned}
0 &= x^T + \beta^{-1}(-\beta x^T) \\
&= x^T + \beta^{-1}b^T \\
&= a^TB + p^TC^T - a^Txb^T + gb^T - p^Tyb^T \\
&= a^TB - a^Txb^T + gb^T + p^TC^T - p^Tyb^T \\
&= \left(a^T, g, p^T\right)\left(B - xb^T, b^T, C^T - yb^T\right)^T \ ,
\end{aligned}
$$

$$
\begin{aligned}
1 &= \beta\beta^{-1} \\
&= -a^T \underbrace{\beta x}_{=b} + \beta g - p^T \underbrace{\beta y}_{=c} \\
&= \left(a^T, g, p^T\right)(b, \beta, c)^T \ ,
\end{aligned}
$$

$$
\begin{aligned}
0 &= y^T - \beta\beta^{-1}y^T \\
&= y^T - \beta(g - a^Tx - p^Ty)y^T \\
&= a^TC + p^TD + \beta a^Txy^T - \beta gy^T + \beta p^Tyy^T \\
&= a^TC - a^Tby^T + gc^T + p^TD - p^Tyc^T \\
&= \left(a^T, g, p^T\right)\left(C - by^T, c^T, D - yc^T\right)^T \ ,
\end{aligned}
$$

$$
\begin{aligned}
0 &= -pb^T + pb^T \\
&= -\underbrace{P^TB}_{=0}ab^T - \underbrace{P^TC}_{=E}pb^T + P^TCpb^T \\
&= \underbrace{P^TB}_{=0} - P^Txb^T + pb^T \\
&= P^T(B - xb^T) + pb^T \\
&= \left(P^T, p, 0\right)\left(B - xb^T, b^T, C^T - yb^T\right)^T \ ,
\end{aligned}
$$

$$\begin{aligned} 0 &= -\beta p + \beta p \\ &= -\beta \underbrace{P^T B}_{=0} a - \beta \underbrace{P^T C}_{=E} p + \beta p \\ &= P^T b + \beta p \\ &= \left(P^T, p, 0\right)(b, \beta, c)^T \ , \end{aligned}$$

$$\begin{aligned} 0 &= -\beta p y^T + \beta p y^T \\ &= -\beta \underbrace{P^T B}_{=0} a y^T - \beta \underbrace{P^T C}_{=E} p y^T + \beta p y^T \\ &= -P^T \beta x y^T + p \beta y^T \\ &= -P^T b y^T + p c^T \ \Leftrightarrow \\ E &= \underbrace{P^T C}_{=E} - P^T b y^T + p c^T \\ &= \left(P^T, p, 0\right)\left(C - b y^T, c^T, D - y c^T\right)^T \ . \end{aligned}$$

Schreibt man diese Gleichungen in ein Matrizensystem, so erhält man

$$\begin{pmatrix} E & 0 & 0 \\ 0 & E & 0 \\ 0 & 0 & E \end{pmatrix} = \begin{pmatrix} A & a & P \\ a^T & g & p^T \\ P^T & p & 0 \end{pmatrix} \begin{pmatrix} B - x b^T & b & C - b y^T \\ b^T & \beta & c^T \\ C^T - y b^T & c & D - y c^T \end{pmatrix}$$

und unter Verwendung der im Satz definierten Matrizen $\tilde{B}, \tilde{C}$ und $\tilde{D}$ folgt

$$\begin{pmatrix} A & a & P \\ a^T & g & p^T \\ P^T & p & 0 \end{pmatrix}^{-1} = \begin{pmatrix} \tilde{B} & \tilde{C} \\ \tilde{C}^T & \tilde{D} \end{pmatrix} \ .$$

□

2.3 Der akkumulierende Greedy-Algorithmus

Gegeben seien die Menge der Datenpunkte $X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^d$, mit x_i paarweise verschieden, und die Werte $f_i \in \mathbb{R}, 1 \leq i \leq N$ an diesen Punkten. Zusätzlich werde eine radiale Basisfunktion gewählt, auf der interpoliert werden soll, und eine Fehlergrenze η, für die die Approximation ausreichend gut ist.

Gestartet wird für $j = 0, 1, 2, \ldots$ mit der Teilmenge $Y_0 \subseteq X$, deren Anzahl an Einträgen der Dimension des Polynomraumes entspricht[4], und der Nullinterpolation, also der Interpolanten $s_0 \equiv 0$. Die Fehlerfunktion $r_j := f - s_j$ ist am Anfang die gegebene Funktion selbst.
Im j-ten Schritt wird mit Hilfe der Maximumsnorm der Punkt $\tilde{x} \in X$ ausgewählt, an dem noch der größte Fehler vorliegt:

$$\tilde{x} : \|r_j(x)\|_{\infty,X} = \max_{x \in X} |r_j(x)| \ .$$

Die Teilmenge $Y_j \subset X$ wird um diesen Punkt erweitert

$$Y_{j+1} := Y_j \cup \{\tilde{x}\}$$

und die Interpolante berechnet, indem f auf der neuen Teilmenge $Y_{j+1} \subset X$ interpoliert wird:

$$s_{j+1} := s_{f,Y_{j+1}}(y) = \sum_{k=0}^{j+1} \alpha_k^{j+1} \Phi(., y_k) + \sum_{l=1}^{Q} \beta_l^{j+1} p_l(y) \ .$$

Als letztes ist die neue Fehlerfunktion zu berechnen:

$$r_{j+1} := f - s_{j+1} \ .$$

Die Iteration wird mit $j + 1$ anstelle von j solange wiederholt, bis entweder die Fehlergrenze erreicht ist ($\|r_j\|_{\infty,X} \leq \eta$) oder $Y_j = X$.

2.4 Konvergenzuntersuchung

Es folgen einige Konvergenzaussagen bezüglich des Algorithmus.

[4] Wird auf den Thin Plate Splines interpoliert, so ist darauf zu achten, daß P_{Y_0} injektiv ist.

Theorem 2.4 *Die Energienormen der Interpolanten s_j aus dem Algorithmus sind monoton steigend. Es gilt für alle $j, k \geq 0$:*

$$\|s_{j+k}\|^2_{\mathcal{N}_\Phi} \geq \|s_j\|^2_{\mathcal{N}_\Phi} \ . \tag{2.3}$$

Beweis. Der Beweis hierzu ist analog zu dem Beweis zu Theorem 1.2. Wird dort $s_{j+k} = f$ gesetzt, so erhält man die Gleichung

$$\|s_{j+k}\|^2_{\mathcal{N}_\Phi} \quad = \quad \|s_{j+k} - s_j\|^2_{\mathcal{N}_\Phi} + \|s_j\|^2_{\mathcal{N}_\Phi}$$

und die Richtigkeit der Behauptung (2.3) ist sofort zu erkennen.

□

Zurückgreifend auf (1.21) und (1.22) läßt sich der Zuwachs der Energie der Interpolanten aus dem Algorithmus, wenn die Interpolationsmenge um einen Punkt $y := \tilde{x}$ erweitert wird, sogar konkret darstellen:

Theorem 2.5 *Für die Energienorm der Interpolanten $s_{f,Y\cup\{y\}}$ zu f auf einer um einen Punkt $y \notin Y$ erweiterten Teilmenge $Y \cup \{y\} \subseteq X$ aus dem Algorithmus gilt:*

$$\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} = \|s_{f,Y}\|^2_{\mathcal{N}_\Phi} + \frac{\left(f(y) - s_{f,Y}(y)\right)^2}{P^2_Y(y)} \ . \tag{2.4}$$

Beweis. Es ist bekannt, daß für

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} f_Y \\ 0 \end{pmatrix} \tag{2.5}$$

mit den Matrizen $A = (\Phi(y_j, y_k))_{1\leq j,k\leq n}, P = (p_k(y_j))_{1\leq k\leq Q, 1\leq j\leq n}$ und den Vektoren $\gamma = (\gamma_1, \ldots, \gamma_n)^T, \delta = (\delta_1, \ldots, \delta_Q)^T$ und $f_Y = (f(y_1), \ldots, f(y_n))^T$ gilt:

$$\|s_{f,Y}\|^2_{\mathcal{N}_\Phi} = \gamma^T A\gamma = \gamma^T(f_Y - P\delta) = \gamma^T f_Y \ . \tag{2.6}$$

Ist y der Punkt, um den die Menge Y erweitert wird, dann gilt für

$$\begin{pmatrix} A & a_y & P \\ a_y^T & g & p_y^T \\ P^T & p_y & 0 \end{pmatrix} \begin{pmatrix} \alpha(y) \\ \alpha_0(y) \\ \beta(y) \end{pmatrix} = \begin{pmatrix} f_Y \\ f(y) \\ 0 \end{pmatrix} \tag{2.7}$$

mit $a_y = (a(y_1, y), \ldots, a(y_N, y))$, $g = a_{y,y}$ und $p_y = p(y)$ folglich:

$$\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} = \begin{pmatrix} \alpha(y) \\ \alpha_0(y) \\ \beta(y) \end{pmatrix}^T \begin{pmatrix} f_Y \\ f(y) \\ 0 \end{pmatrix} . \tag{2.8}$$

Schreibt man die Systeme (2.7) und (2.5) um in

$$\begin{aligned} A\alpha(y) + \alpha_0(y)a_y + P\beta(y) &= f_Y \quad = A\gamma + P\delta && (2.9)\\ a_y^T\alpha(y) + \alpha_0(y)g \ + p_y^T\beta(y) &= f(y) && (2.10)\\ P^T\alpha(y) + \alpha_0(y)p_y \qquad\quad &= 0 \quad = P^T\gamma\ , && (2.11) \end{aligned}$$

so folgt aus den Gleichungen (2.9) und (2.11)

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \alpha(y) - \gamma \\ \beta(y) - \delta \end{pmatrix} = \begin{pmatrix} -\alpha_0(y)a_y \\ -\alpha_0(y)p_y \end{pmatrix} ,$$

woraus man durch Umformung wiederum

$$\begin{pmatrix} \alpha(y) \\ \beta(y) \end{pmatrix} = \begin{pmatrix} \gamma \\ \delta \end{pmatrix} - \alpha_0(y) \begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} a_y \\ p_y \end{pmatrix} \tag{2.12}$$

erhält. Dies in (2.10) eingesetzt ergibt

$$\alpha_0(y)g + \begin{pmatrix} a_y \\ p_y \end{pmatrix}^T \left(\begin{pmatrix} \gamma \\ \delta \end{pmatrix} - \alpha_0(y) \begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} a_y \\ p_y \end{pmatrix} \right) = f(y) ,$$

und man erhält unter Anwendung von (1.21) und (1.22)

$$\alpha_0(y) = \frac{f(y) - \begin{pmatrix} a_y \\ p_y \end{pmatrix}^T \begin{pmatrix} \gamma \\ \delta \end{pmatrix}}{g - \begin{pmatrix} a_y \\ p_y \end{pmatrix}^T \begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} a_y \\ p_y \end{pmatrix}} = \frac{f(y) - s_{f,Y}(y)}{P_Y^2(y)} . \tag{2.13}$$

Zurückgreifend auf (2.8) ergibt sich nun durch Umformen und Einsetzen:

$$\begin{aligned}
\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} &= \left(\alpha^T(y), \alpha_0(y), \beta^T(y)\right)(f_Y, f(y), 0)^T \\
&= \alpha_0(y)f(y) + \begin{pmatrix}\alpha(y)\\ \beta(y)\end{pmatrix}^T \begin{pmatrix} f_Y \\ 0\end{pmatrix} \\
&\overset{(2.12)}{=} \alpha_0(y)f(y) + \\
&\qquad \left(\begin{pmatrix}\gamma\\ \delta\end{pmatrix}^T - \alpha_o(y)\begin{pmatrix}a_y\\ p_y\end{pmatrix}^T \begin{pmatrix} A & P\\ P^T & 0\end{pmatrix}^{-1}\right)\begin{pmatrix} f_Y \\ 0\end{pmatrix} \\
&= \gamma^T f_Y + \alpha_0(y)f(y) - \alpha_0(y)\begin{pmatrix}a_y\\ p_y\end{pmatrix}^T \begin{pmatrix} A & P\\ P^T & 0\end{pmatrix}^{-1}\begin{pmatrix} f_Y \\ 0\end{pmatrix} \\
&\overset{(2.6)}{=} \|s_{f,Y}\|^2_{\mathcal{N}_\Phi} + \alpha_0(y)\left(f(y) - \begin{pmatrix}a_y\\ p_y\end{pmatrix}^T \begin{pmatrix}\gamma\\ \delta\end{pmatrix}\right) \\
&\overset{(2.13)}{=} \|s_{f,Y}\|^2_{\mathcal{N}_\Phi} + \frac{\left(f(y) - s_{f,Y}(y)\right)^2}{P_Y^2(y)}\,.
\end{aligned}$$

□

Der Fehler $\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} - \|s_{f,Y}\|^2_{\mathcal{N}_\Phi}$ ist also eine positive Zahl, die abhängig ist von der Funktion f, ihrer Interpolanten $s_{f,Y}$ auf Y und der Powerfunktion an der Stelle y.

Über die Fehlerfunktionen r_j für $j = 1, \ldots, N$ läßt sich folgende Aussage machen:

Theorem 2.6 *Die Fehlerfunktionen $r_j = f - s_j$ für $j = 1, \ldots, N$ aus dem Algorithmus sind in der Maximumsnorm begrenzt durch*

$$\|f\|^2_{\mathcal{N}_\Phi} \geq \|s_{f,Y_N}\|^2_{\mathcal{N}_\Phi} - \|s_{f,Y_1}\|^2_{\mathcal{N}_\Phi} \geq \sum_{j=1}^{N} \frac{\|r_j\|^2_{\infty,X}}{c} \tag{2.14}$$

und - falls die Menge X unendlich ist - quadratsummierbar, d. h. es gilt für $N \to \infty$:

$$\sum_{j=1}^{\infty} \frac{\|r_j\|^2_{\infty,X}}{c} \leq \infty \tag{2.15}$$

und die Folge der quadrierten Energienormen der Fehlerfunktionen konvergiert für $j \to \infty$ gegen Null.

Für den Beweis wird die Konsequenz des folgenden, aus [1] entnommenen Lemmas benötigt (der Beweis hierzu wird nicht weiter angeführt, er kann aber ebenfalls in [1] eingesehen werden).

Lemma 2.1 *Die Powerfunktion hat die Maximaleigenschaft*

$$P_X(x) = \sup\{f(x) : f \in \mathcal{N}_\Phi, \|f\|_{\mathcal{N}_\Phi} \leq 1, f(X) = \{0\}\}$$

für alle $x \in \Omega$.

Daraus erhält man die für den Beweis des Theorems wichtige Folgerung:

Folgerung 2.1 *Ist* $|\hat{X}| = Q$ *und* $Y \supseteq X \supseteq \hat{X}$*, so gilt die Ungleichungskette*

$$P_Y^2(x) \leq P_X^2(x) \leq P_{\hat{X}}^2(x) \leq \|P_{\hat{X}}^2\|_\Omega = c \tag{2.16}$$

für alle $x \in \Omega$ *und* c *konstant.*

Beweis von Theorem 2.6. Aus Theorem 2.5 ist bekannt, daß

$$\|s_{f,Y_{j+1}}\|_{\mathcal{N}_\Phi}^2 = \|s_{f,Y_j}\|_{\mathcal{N}_\Phi}^2 + \frac{(f(y_{j+1}) - s_{f,Y_j}(y_{j+1}))^2}{P_{Y_j}^2(y_{j+1})}$$

gilt. Unter Verwendung der Gleichheit $\|r_j(x)\|_{\infty,X} = |r_j(y_{j+1})|$ und der Folgerung 2.1 ergibt sich:

$$\begin{aligned} \|s_{f,Y_{j+1}}\|_{\mathcal{N}_\Phi}^2 &= \|s_{f,Y_j}\|_{\mathcal{N}_\Phi}^2 + \underbrace{\frac{\|r_j\|_{\infty,X}^2}{P_{Y_j}^2(y_{j+1})}}_{\leq c} \\ &\geq \|s_{f,Y_j}\|_{\mathcal{N}_\Phi}^2 + \frac{\|r_j\|_{\infty,X}^2}{c} . \end{aligned}$$

Hieraus folgt

$$\begin{aligned} \frac{\|r_j\|_{\infty,X}^2}{c} &\leq \|s_{f,Y_{j+1}}\|_{\mathcal{N}_\Phi}^2 - \|s_{f,Y_j}\|_{\mathcal{N}_\Phi}^2 \\ \sum_{j=1}^{N} \frac{\|r_j\|_{\infty,X}^2}{c} &\leq \|s_{f,Y_N}\|_{\mathcal{N}_\Phi}^2 - \|s_{f,Y_1}\|_{\mathcal{N}_\Phi}^2 , \end{aligned}$$

beziehungsweise für eine unendliche Menge X

$$\begin{aligned} \frac{\|r_j\|^2_{\infty,X}}{c} &\leq \|s_{f,Y_{j+1}}\|^2_{\mathcal{N}_\Phi} - \|s_{f,Y_j}\|^2_{\mathcal{N}_\Phi} \\ \sum_{j=1}^{\infty} \frac{\|r_j\|^2_{\infty,X}}{c} &\leq \|s_{f,Y_\infty}\|^2_{\mathcal{N}_\Phi} - \|s_{f,Y_1}\|^2_{\mathcal{N}_\Phi} \ . \end{aligned}$$

Die Begrenzung durch $\|f\|^2_{\mathcal{N}_\Phi}$ ergibt sich durch Übertragung der Energieaufteilung aus (1.14) hierauf.

□

Bemerkung 2.1 *Satz 2.6 läßt sich durch nähere Betrachtung des Verhaltens der Powerfunktion sicher noch verschärfen, da $P^2_{Y_j}(y_{j+1})$ durch die Konstante c sehr großzügig abgeschätzt wird.*

Theorem 2.7 *Die Folge der Energienormen der Interpolanten s_j aus dem Algorithmus konvergiert gegen die Energienorm der Interpolanten s_X auf ganz X:*

$$\|s_j\|_{\mathcal{N}_\Phi} \to \|s_{f,X}\|_{\mathcal{N}_\Phi} \tag{2.17}$$

Beweis. Nach der Energieaufteilung (1.14) gilt für $j = 1, \ldots, N$

$$\|s_{f,X}\|^2_{\mathcal{N}_\Phi} \stackrel{!}{=} \|f\|^2_{\mathcal{N}_\Phi} = \|r_j\|^2_{\mathcal{N}_\Phi} + \|s_j\|^2_{\mathcal{N}_\Phi} \ .$$

Da nach Theorem (2.6) die Folge der Fehlerfunktionen gegen Null konvergiert, ergibt sich die Behauptung dieses Satzes.

□

Kapitel 3

Umsetzung in Matlab

In diesem Kapitel wird die Umsetzung des im vorigen Kapitel vorgestellten Verfahren ins Programm beschrieben.
Diesem ist eine Einführung in das für diese Zwecke vorteilhafte numerische Softwaresystem *Matlab* vorangestellt, deren Grundzüge aus [2] entnommen sind. Matlab ist ein Matrix-orientiertes System, in dem sich Probleme und Lösungen in vertrauter mathematischer Schreibweise darstellen lassen, und es verfügt über eine umfangreiche Sammlung von mathematischen Algorithmen und Funktionen, auf die zurückgegriffen werden kann.

3.1 Einführung in Matlab

3.1.1 Vektoren und Matrizen

Da Matlab vor allem für numerische Berechnungen konstruiert worden ist, wird fast alles in Form von Matrizen beziehungsweise Vektoren realisiert. So können Zeilenvektoren (oder $1 \times n$-Matrizen) mit Hilfe eckiger Klammern sehr einfach eingegeben werden. Die Komponenten werden durch ein Leerzeichen oder durch ein Komma getrennt. Durch ein Semikolon am Ende der Eingabe kann die Ausgabe unterdrückt werden.

```
>> x = [1 2 3]

x =
```

```
     1     2     3
>> x = [1,2,3];
```

Spaltenvektoren und $n \times 1$-Matrizen werden durch ein Semikolon getrennt oder mit Hilfe des Transponiertzeichens (') eingegeben.

```
>> x = [1;2;3]

x =

     1
     2
     3
>> x = [1 2 3]'

x =

     1
     2
     3
```

Zeilenvektoren mit Schrittweite 1 oder einer beliebigen anderen Schrittweite können mit Hilfe des Colon-Operators (dem Doppelpunkt) eingegeben werden - bei einer Schrittweite von 1 durch `Anfang:Ende`, andernfalls durch `Anfang:Schrittweite:Ende`.

```
>> x = 1:5

x =

     1     2     3     4     5
>> y = 1:0.5:2

y =

    1.0000    1.5000    2.0000
```

Matrizen werden zeilenweise in eckigen Klammern eingegeben. Dabei wird durch ein Semikolon eine neue Zeile angekündigt; Leerzeichen oder Kommata trennen die Elemente in einer Zeile. Durch ein Semikolon am Ende

der Eingabe kann die Ausgabe der Matrix auf dem Bildschirm unterdrückt werden.

```
>> A = [1 2 3;4 5 6]

A =

     1     2     3
     4     5     6
>> A = [1 2 3;4 5 6];
```

`ones(m,n)` erzeugt die $m \times n$-Matrix mit Elementen 1:

```
>> ones(2,4)

ans =

     1     1     1     1
     1     1     1     1
```

Entsprechendes gilt für `zeros(m,n)`, durch das die Nullmatrix abgebildet wird, und `eye(n)`, das die $n \times n$-Einheitsmatrix erzeugt.
Der schon oben genannte Doppelpunkt hat nun eine ganz andere Bedeutung, wenn er innerhalb runder Klammern benutzt wird. Dann ist er ein Platzhalter für alle Einträge. So gibt `a = A(:)` alle Einträge von A als Spaltenvektor an.

```
>> a = A(:)

a =

     1
     4
     2
     5
     3
     6
```

Auch lassen sich einzelne Zeilen oder Elemente mit Hilfe des Doppelpunktes ansprechen oder sogar verändern:
Zum Beispiel erhält man die zweite Zeile der Matrix A durch

```
>> A(2,:)

ans =

     4     5     6
```

Das folgende Kommando ersetzt die erste Zeile der Matrix A durch den Zeilenvektor (7, 8, 9).

```
>> A(1,:) = [7 8 9]

A =

     7     8     9
     4     5     6
```

3.1.2 Rechen-Operationen

Die folgende Auflistung zeigt die Matrixoperationen in Matlab.

Symbol	*Operation*	*Matlab Syntax*	*Mathematische Syntax*
`+`	Addition	`A+B`	$A + B$
`-`	Subtraktion	`A-B`	$A - B$
`*`	Multiplikation	`A*B`	AB
`\`	linke Division	`A\B`	Löst $AX = B$ nach X
`^`	Potenzieren	`A^x`	A^x
`.'`	Transponiert	`A.'`	A^T

Bei Verwendung dieser Operatoren ist es wichtig, daß die Größen der beteiligten Komponenten kompatibel sind, da sonst eine Fehlermeldung von Matlab zurückgegeben wird. Ebenfalls ist es wichtig zu wissen, daß für den Fall, daß ein Operand ein Skalar ist, dieser Skalar auf jedem Matrixelement operiert. Durch die Matrixdivisionen können Gleichungssysteme der Form $Ax = b$ einfach gelöst werden. Ist A eine $n \times n$-Matrix und b ein n-Spaltenvektor, so löst die Linksdivision das Gleichungssystem mit Hilfe des Gauß-Verfahrens. Im Fall, daß A eine $m \times n$-Matrix und b ein n-Spaltenvektor ist, wird die Methode der kleinsten Quadrate zur Lösung des Gleichungssystems benutzt (auch im Falle eines überbestimmten Gleichungssystems).

Die Matrixaddition und -subtraktion sind elementweise definiert. Soll eine der übrigen oben angeführten Operationen (bis auf das Transponieren) komponentenweise durchgeführt werden, so wird Matlab dies durch einen Punkt vor dem Operator signalisiert. Zum Beispiel werden die Quadrate der Komponenten von $x = (1, 2, 3, 4, 5)$ berechnet, indem man folgendes eingibt:

```
>> z = x.*x

z =

     1     4     9    16    25
```

3.1.3 Kontrollstrukturen

Als Kontrollstrukturen existieren in Matlab folgende Verzweigungen und Schleifen, die wie folgt zu gebrauchen sind:

- `if`: Die Anweisung `if` wertet einen logischen Ausdruck aus und läßt eine Gruppe von Anweisungen ausführen, wenn der Ausdruck wahr ist. Ist der Ausdruck unwahr, wird die Anweisungen nach `else` (bzw. `elseif`) ausgeführt.

  ```
  if (a==2)
      x=2a;
  elseif (a==4)
      x=3a;
  else
      x=a;
  end
  ```

- `for`: `For`-Schleifen werden benutzt, wenn eine Gruppe von Anweisungen so oft ausgeführt werden soll, wie eine vorgegebene Anzahl festlegt ist, zum Beispiel:

  ```
  for i=1:3
      einefunktion(i);
  end
  ```

- `while`: Die `while`-Schleife wertet einen gegebenen Testausdruck aus und führt die Anweisungen zwischen `while` und `end` nur aus, wenn der Testausdruck wahr ist. Andernfalls wird im Programm mit dem ersten Befehl nach dem Ende der Schleife fortgefahren. Ein Beispiel:

```
a=0;
while (a<5)
    a=a+1
end
```

- `switch`: Die Anweisung `switch` läßt Gruppen von Anweisungen entsprechend dem Wert einer Variablen oder eines Ausdrucks ausführen. Die Schlüsselwörter `case` und `otherwise` begrenzen diese Gruppen. Es wird nur die erste Übereinstimmung ausgeführt.

```
switch x
case 1
    a=1;
case 2
    a=-1;
otherwise (a=0)
end
```

Jede dieser Anweisungen und Verzweigungen muß mit einem `end` beendet werden.

3.1.4 Nützliche Funktionen

In Matlab gibt es eine Vielzahl Funktionen, die dem Anwender das Programmieren sehr erleichtern können. Die Bedeutungen der Matlab-Funktionen, die in dieser Arbeit verwendet werden, sind in der folgenden Auflistung knapp beschrieben. Ausführliche Erläuterungen und Anwendungsbeispiele können gegebenenfalls in der Matlab-eigenen Hilfe - aufrufbar durch `help` - nachgelesen werden.

Funktion	*Bedeutung*
`abs`	Absolutwert
`break`	Abbruch einer Schleifen- oder Verzweigungsausführung
`ceil`	Rundet zur nächst größeren ganzen Zahl auf
`cond`	Kondition einer Matrix
`exp`	Exponentialfunktion
`figure`	Öffnet Grafikfenster
`find`	Gibt Indizes und Werte zurück
`global`	Macht Variable systemweit verfügbar
`grid`	Erzeugt ein Gitter

`hold on`	Läßt weitere Plots in bestehende Grafiken einfügen
`hold off`	Gibt Grafikfenster wieder "frei" nach `hold on`
`intersect`	Gibt Indizes gesuchter Einträge zurück
`inv`	Inverse einer Matrix
`length`	Länge eines Vektors
`log`	Natürlicher Algorithmus
`loglog`	Logarithmisches Koordinatensystem
`max`	Berechnet größte Komponente
`mesh`	Erzeugt eine 3D-Darstellung (offenes Gitter) der Fläche
`plot`	Erzeugt Grafikfenster
`polyval`	Berechnet Polynomwerte
`repmat`	Erzeugt Blockmatrix
`reshape`	Ändert die Struktur
`return`	Beendet Schleifen- oder Verzweigungsausführung
`size`	Matrixgröße
`sprintf`	Gibt Variablen aus
`sqrt`	Quadratwurzelfunktion
`subplot`	Unterteilt ein Grafikfenster in mehrere Figuren
`sum`	Berechnet die Summe der Komponenten
`surf`	Erzeugt eine 3D-Darstellung (schattiertes Gitter) der Fläche
`title`	Erzeugt eine Überschrift im Grafikfenster

3.2 Elementare Funktionen zum Rechnen mit RBF's in Matlab

Für die Umsetzung des im zweiten Kapitel beschriebenen Algorithmus ist es sinnvoll, zunächst elementare Funktionen, die das Rechnen mit radialen Basisfunktionen in Matlab vereinfachen, einzuführen. Die in diesem Abschnitt im folgenden vorgestellten Programme sind freundlicherweise von Herrn Prof. Dr. R. Schaback zur Verfügung gestellt worden, wobei sie teilweise zur Anpassung an diese Arbeit umgeschrieben wurden.

3.2.1 Zweidimensionale RBF-Funktionen

Zunächst werden in einer Funktion `fct2D.m` die für diese Arbeit wichtigen radialen Basisfunktionen zusammengefaßt. Hierbei dient die `switch`-Anweisung dazu, die radialen Basisfunktionen als einzelne Anweisungen aufzuführen, auf

die durch die Variable `Genfct2Dchoice` dann auch einzeln zurückgegriffen werden kann. Durch die Globalisierung dieser Variablen sind die Funktionen für das ganze System zugänglich.

```
function z = fct2D(x,y)

% Eingabe: x = N-Vektor
%          y = N-Vektor
% Ausgabe: z = N-Vektor

global Genfct2Dchoice

switch Genfct2Dchoice
case 1                              % Multiquadric
    r=(x.^2+y.^2).^(0.5);
    z=(1+r.*r).^(1/2);
case 2                              % Gaussian
    r=(x.^2+y.^2).^(0.5);
    z=exp(-r.*r);
case 3                              % Inverse Multiquadric
    r=(x.^2+y.^2).^(0.5);
    z=(1+r.*r).^(-1/2);
case 4                              % Wendland
    r=(x.^2+y.^2).^(0.5);
    z=(1+4*r).*max(0,(1-r)).^4;
case 5                              % TPS
    r=(x.^2+y.^2).^(0.5);
    z=r.*r.*log(r+eps);
otherwise
    error('2D function not implemented')
end
```

3.2.2 Datenpunkte

Die Funktion `getpoints.m` erzeugt Datenpunkte in für das jeweilige Hauptprogramm gewünschter Dimension. Sie läßt sich vom Hauptprogramm die Anzahl der geforderten Punkte (`numpoints`), deren Dimension (`dim`) und die Art, wie sie erzeugt werden sollen (`nrand`) - zufällig oder geordnet im Intervall $[-1,1]^{dim}$ - übergeben und gibt die "fertigen" Punkte (`points`) an

das Hauptprogramm zurück. Ebenfalls der Index-Vektor der Gittereckpunkte `IP` wird an das Hauptprogramm übergeben.
Im geordneten Fall erzeugt die Funktion ein $n \times n$ - Gitter. Ist `numpoints` eine Quadratzahl, so ist $n^2 =$ `numpoints`, andernfalls berechnet `getpointsnD.m` die nächstgrößere Quadratzahl und gibt dem Hauptprogramm mehr als geforderte, nämlich `numpoints` $+ x = n^2$, $x, n \in \mathbb{N}$ Gitterpunkte zurück.
Die Schrittweite zwischen den einzelnen Punkten beträgt $h = \frac{2}{\sqrt{n}-1}$.

```
function [points,IP] = getpoints(numpoints, dim, nrand)

% Eingabe: numpoints = Skalar
%          dim = Skalar
%          nrand = Skalar
% Ausgabe: points = numpoints x dim-Matrix
%          IP = Spaltenvektor

if nrand==1 % Zufall
    rand('state',0);
    points=-1+2*rand(numpoints,dim);
    return
end
% geordneter Fall:
np_oneD=2*ceil(numpoints.^(1/dim)/2); %behandelt alle
numpoints=np_oneD^dim;                %dim gleich
h=2/(sqrt(numpoints)-1);
if dim==1
    points=(-1:h:1)';
    B=[-1;1];
    [S,IP,IB]=intersect(points,B,'rows');
    return
end
if dim==2
    [X,Y]=meshgrid(-1:h:1,-1:h:1);
    points=[X(:) Y(:)];
    B=[-1,-1;-1,1;1,-1;1,1];
    [S,IP,IB]=intersect(points,B,'rows');
    return
end
% generelle Dimension
countvec=zeros(1,dim);
points=zeros(numpoints,dim);
```

```
ind=1;
for n=1:2^dim
    B(n,:)=compute_grid_vector(n-1,dim);
end
[S,IP,IB]=intersect(points,B,'rows');
while ind<=numpoints
    points(ind,:)=-1+countvec*h;
    for j=1:dim  % lexikographische Inkrementation von countvec
        if countvec(1,j)==np_oneD-1
            countvec(1,j)=0;
        else
            countvec(1,j)=countvec(1,j)+1;
            break;
        end
    end
    ind=ind+1;
end
```

Mit Hilfe des Programms `compute_grid_vector.m` können die Eckpunkte des Gitters ermittelt werden. Dieses erzeugt die für die genutzte Dimension möglichen Gittereckpunkte, welche in `getpoints` in eine Matrix `B` geschrieben werden und dann mit den von `getpoints` erzeugten Punkten `points` verglichen werden, um schließlich die Indizes der (vorhandenen) Gittereckpunkte zu ermitteln. Das Gittereckpunkte erzeugende Programm ist folgendermaßen programmiert:

```
function y = compute_grid_vector(n,d);

% Eingabe: n = Anzahl der zu entwickelnden Gittervektoren
%          d = Dimension des Gittervektors
% Ausgabe: y = Gittervektor als Zeilenvektor

y=zeros(1,d);
z = n;
for i=d:-1:1
    y(i) = 2*mod(z,2)-1;
    z = floor(z/2);
end
```

3.2.3 RBF-Matrizen

Um RBF-Interpolations-Matrizen darstellen zu können, werden zunächst die drei Hilfsfunktionen `getdistancesquaredmatrix.m`, `r2bf.m` und `polynomials.m` formuliert. Die erstgenannte Funktion wandelt "Punktmatrizen" in die für die Rechnung mit radialen Basisfunktionen notwendige Form der "Abstandsmatrizen" um. Dabei wird ausgenutzt, daß

$$\|x_j - x_i\|_2^2 = \|x_j\|_2^2 + \|x_i\|_2^2 - 2(x_j, x_i)_2$$

gilt, und der dadurch gegebene quadrierte Abstand berechnet wird, wie in der letzten Gleichung des Programms zu sehen ist.
Die drei aufeinanderfolgenden Punkte, die in dieser Funktion zum ersten Mal auftauchen, sind das Matlab-Symbol für einen Zeilenumbruch.

```
function fullmat = ...
             getdistancesquaredmatrix(evalpoints, datapoints)

% Eingabe: evalpoints = N x dim-Matrix
%          datapoints = N x dim-Matrix
% Ausgabe: fullmat = (#evalpoints) x (#datapoints)-Matrix
%                     + Polynome

[numdata, datadim]=size(datapoints);
[numeval, evaldim]=size(evalpoints);
if evaldim~=datadim
    error('Dimensions must agree')
end
evalsquares=sum(evalpoints.^2,2);
datasquares=sum(datapoints.^2,2);
fullmat=repmat(evalsquares,1,numdata)+...     % quadrierter
        repmat(datasquares,1,numeval)'-...    % Abstand
        2*evalpoints*datapoints';
```

Nun wird noch eine Skalarfunktion benötigt, die auf eine Abstandsmatrix (`rr`) angewendet eine radiale Basisfunktion-Interpolationsmatrix (`y`) erzeugt. Ähnlich wie in `fct2D.m` ist hierbei ein m-File `r2bf.m` sinnvoll, durch das später in einem Hauptprogramm durch einfache Änderung einer Variablen `RBFtype=k` zwischen verschiedenen Skalarfunktionen variiert werden kann. Auch diese Funktion ist durch die Quadrierung des hinzukommenden Skalierungsfaktors `RBFscale` extra auf große Abstandsmatrizen ausgerichtet.

```
function y =  r2bf(rr)

% Eingabe: rr = N x N-Matrix
% Ausgabe: y = N x N-Matrix

global RBFscale;
global RBFtype;

r=rr./(RBFscale^2);
if RBFtype==1
    y=(1+r).^(1/2);                  % Multiquadric
    return;
end
if RBFtype==2
    y=exp(-r);                       % Gaussian
    return;
end
if RBFtype==3
    y=(1+r).^(-1/2);                 % Inverse Multiquadric
    return;
end
if RBFtype==4
    r=sqrt(r);
    y=(1+4*r).*max(0,(1-r)).^4;      % Wendland 2
    return;
end
if RBFtype==5
    y=0.5*r.*log(r+eps);             % TPS
    return;
end
error('RBF type not implemented')
```

Mit dieser und der vorherigen Funktion können bereits Interpolationsmatrizen zu positiv definiten radialen Basisfunktionen ausgedrückt werden durch

```
rbfmatrix = ...
       r2bf(getdistancessquaredmatrix(evalpoints, datapoints))}.
```

Für Matrizen bedingt positiv definiter radialer Basisfunktionen wird für einen entsprechenden Ausdruck noch eine Funktion benötigt, die Polynome in passender Ordnung erzeugt. Die Funktion `polynomials.m` tut dies:

```
function polvalues = polynomials(points,order)

% Eingabe: points = N x dim-Matrix
%          order =  Skalar
% Ausgabe: polvalues = Polynome

[numpoints,dim]=size(points); % trivialer Fall
if order==0
    polvalues=[];
    return
end
if order==1
    polvalues=ones(numpoints,1);
    return
end
if order==2
    polvalues=[ones(numpoints,1) points];
    return
end
if dim==1
  polvalues=zeros(numpoints,order);
  polvalues(:,1)=ones(numpoints,1);
  polvalues(:,2)=points(:,1);
  for i=3:order
      polvalues(:,i)=polvalues(:,i-1).*points(:,1);
  end
  return
end
% allgemeiner Fall:
 polvalues=[polynomials(points,order-1)...
        polynomials_exact_order(points,order)];
```

Hier wird in den letzten beiden Zeilen - dem allgemeinen Fall - die rekursiv gehaltene Funktion `polynomials_exact_order` zu Hilfe genommen, sie berechnet die Monomauswertung. Es wird die Matrix mit den Werten sämtlicher Monome einer vorgegebenen Ordnung (=Grad+1) m in den Datenpunkten geliefert.

```
function polvalues = polynomials_exact_order(points,order)

% Eingabe: points    = N x dim-Matrix
%          order     = Skalar
% Ausgabe: polvalues =

[numpoints,dim]=size(points); % trivialer Fall
if order==1
    polvalues=ones(numpoints,1);
    return
end
if order==2
    polvalues=points;
    return
end
if dim==1
    polvalues=points(:,1).^(order-1);
    return
end
% Rekursion findet ueber Raumdimension statt,
% dadurch ausreichend perfomant:
polvalues= points(:,dim).^(order-1);
for i=2:order-1
    pe=polynomials_exact_order(points(:,1:dim-1),i);
    pp=points(:,dim).^(order-i);
    [rpes cpes]=size(pe);
    for j=1:cpes
        polvalues=[polvalues pe(:,j).*pp];
    end
end
polvalues=[polvalues polynomials_exact_order...
                              (points(:,1:dim-1),order)];
```

Folgende Funktion erzeugt nun - unter Verwendung der eben vorgestellten Hilfsfunktionen - ein allgemeines RBF-System (`fullmat`) in der vom Hauptprogramm durch `evalpoints` und `datapoints` vorgegebenen Größe.

```
function fullmat = getrbfmatrix(evalpoints, datapoints)

% Eingabe: evalpoints = N x dim-Matrix
%          datapoints = N x dim-Matrix
% Ausgabe: fullmat = (#evalpoints) x (#datapoints)-Matrix
%                    + Polynome

[numdata,  datadim]=size(datapoints);
[numeval,  evaldim]=size(evalpoints);
if datadim~=evaldim
    error('Dimensions must agree')
end
global RBForder

if RBForder==0
    fullmat=r2bf(getdistancesquaredmatrix...
                    (evalpoints,datapoints));
    return
end
if RBForder>0
    polevalmat=polynomials(evalpoints, RBForder);
    [nrpe, ncpe]=size(polevalmat);
    if datapoints==evalpoints
        fullmat=[r2bf(getdistancesquaredmatrix...
                (evalpoints, datapoints))polevalmat;
            polevalmat' zeros(ncpe, ncpe)];
        return
    else
        poldatamat=polynomials(datapoints,RBForder);
        [nrpd, ncpd]=size(poldatamat);
        fullmat=[r2bf(getdistancesquaredmatrix...
                  (evalpoints,datapoints))polevalmat;
            poldatamat' zeros(ncpd, ncpe)];
    end
    return
end
error('RBForder must be positive')
```

Das Programm `values` entwickelt in `evalpoints` eine RBF-Approximation, definiert durch Zentren in `datapoints` und `coefficients`.

```
function values = rbfevallarge(evalpoints, datapoints, coeff)

% Eingabe: evalpoints = Matrix
%          datapoints = Matrix
%          coeff = Vektor
% Ausgabe: values = Matrix

global RBForder;
[numeval,evaldim]=size(evalpoints);
[numdata,datadim]=size(datapoints);
[numcoeff,coeffdim]=size(coeff);
if evaldim~=datadim
    error('Unequal space dimensions')
end
if coeffdim~=1
    error('Coefficients must be a vector')
end
values=zeros(numeval,1);
if RBForder>0
    values=values+polynomials(evalpoints,RBForder)*...
                    coeff(numdata+1:end);
end
datadistsq=points2normsquares(datapoints);
evaldistsq=points2normsquares(evalpoints);
for ip=1:numdata
    values=values+coeff(ip,1)*...
        r2bf(evaldistsq+datadistsq(ip,1)-...
            2*evalpoints*(datapoints(ip,:)'));
```

Die hier benutzte Funktion `points2normsquares` weist Punkte normquadrierten Vektoren zu:

```
function normsquares = points2normsquares (points)

% Eingabe: points = Matrix
% Ausgabe: normsquares = Matrix

normsquares=sum(points.^2,2);
```

3.3 Weitere Hilfsfunktionen

In diesem Abschnitt werden nun noch diejenigen Funktionen aufgeführt, die für die Umsetzung und Auswertung des Algorithmus ebenfalls sehr hilfreich sein werden, aber nicht speziell auf das Rechnen mit radiale Basisfunktionen zugeschnitten sind.

3.3.1 Rang-1-Erweiterungen

Den folgenden drei Programmen dienen die Sätze 2.1, 2.2 und 2.3, deren Matrizen- und Vektorenbezeichnungen vollständig und unverändert übernommen wurden, als Grundlage. Anlehnend an die in den Sätzen aufgezeigten Schemata sind die Rang-1-Erweiterung einer Cholesky-Zerlegung in `cholesky0.m`, die Rang-1-Erweiterung der Inversen zu einer positiv definiten RBF-Matrix in `matuptdate0.m` sowie zu einer bedingt positiv definiten RBF-Matrizen in `matupdate.m` programmiert.
Es sei vorweg noch angemerkt, daß im weiteren Verlauf dieser Arbeit die Funktion `matupdate0.m` der Funktion `cholesky0.m` vorgezogen werden wird, um die Rang-1-Erweiterung-Berechnung bei positiv definiten und bedingt positiv definiten Matrizen im Programm "einheitlich" zu halten. Im Anhang ist jedoch ein Programm zu finden, welches `cholesky0.m` benutzt.

```
function L = cholesky0(L,a,g)

% Eingabe: a = N-Vektor
%          g = Skalar
%          L = N x N-Matrix
% Ausgabe: L = (N+1) x (N+1)-Matrix

[M,N]=size(L);
l = (inv(L'))*a;
u = sqrt(g-sum(l.*l));
L=[L l;zeros(1,N) u];
```

Die Rang-1-Erweiterung einer Inversen B einer positiv definiten Matrix ist folgendermaßen umgesetzt:

```
function B = matupdate0(B,a,g)

% Eingabe: B = N x N-Matrix
%          a = N-Vektor
%          g = Skalar
% Ausgabe: B = (N+1) x (N+1)-Matrix

x=B*a;
beta=1./(g-a'*x);
b=-beta.*x;
B=[B-x*b' b; b' beta];
```

Schließlich ist noch die Rang-1-Erweiterung der Inversen `[B C;C^T D]` einer bedingt positiv definiten Matrix anzuführen:

```
function [B,C,D] = matupdate(B,C,D,a,p,g)

% Eingabe: B = N x N-Matrix
%          C = N x Q-Matrix
%          D = Q x Q-Matrix
%          a = N-Vektor
%          p = N-Vektor
%          g = Skalar
% Ausgabe: B = (N+1) x (N+1)-Matrix
%          C = (N+1) x Q-Matrix
%          D = Q x Q-Matrix

x=B*a+C*p;
y=C'*a+D*p;
beta=1./(g-a'*x-p'*y);
b=-beta.*x;
c=-beta.*y;
B=[B-x*b' b;b' beta];
C=[C-b*y' ; c'];
D=D-y*c';
```

3.3.2 Ordnungsschätzung mittels linearer Regression

Mit Hilfe der Funktion `linreg.m` kann die Ordnung eines Verfahrens ermittelt werden. Die Ordnung x und der unbekannte Koeffizient a_0 einer Abschätzung der Form

$$y = a_0 t^x$$

lassen sich mit der Methode der kleinsten Quadrate und dem Backslash-Operator berechnen, wie es in dem Programm `linreg` auch geschieht.
Diese Funktion wird im Hauptprogramm dazu dienen, die Ordnung der Interpolation mit Hilfe des Greedy-Algorithmus zu ermitteln.

```
function [c,alpha] = linreg(x,y)

% Eingabe: x     = N-Vektor
%          y     = N-Vektor
% Ausgabe: c     = Skalar
%          alpha = Skalar

if (length(x)~=length(y))
    error('x und y sind unterschiedlich gross!');
end

x=right(x);
y=right(y);

X = [ones(length(x),1) log(x)];
p = X\ log(y);
c = exp(p(1));
alpha = p(2);
```

Die hier hinzugezogene Funktion `right.m` überprüft, ob der gegebene Vektor ein Spaltenvektor ist. Falls es ein Zeilenvektor ist, wird er transponiert zurückgegeben:

```
function x = right(x)

% Eingabe: x = N-Vektor
% Ausgabe: x = N-Spaltenvektor

[m n ]=size(x);
if (n>m), x=x'; end
```

3.4 Der akkumulierende Greedy-Algorithmus

Mit Hilfe aller bisher beschriebenen Funktionen läßt sich nun der Algorithmus aus dem Abschnitt 2.3 in den beiden wesentlichen Programmen dieser Arbeit, `greedyschleife.m` und `rahmen.m`, umsetzen.
Da sie beide sehr umfangreich sind, wird in diesem Abschnitt nur ihr Aufbau erläutert werden, bei Bedarf können sie jedoch im Anhang dieser Arbeit in voller Länge eingesehen werden.

3.4.1 Die Greedyschleife

Die Interpolation mit Hilfe des akkumulierenden Greedy-Algorithmus ist in `greedyschleife` umgesetzt. Wie schon der Name des Programms andeutet, ist der in 2.3 beschriebene Algorithmus selbst in eine `while`-Schleife eingebunden. Sie wird solange wiederholt, bis eines der drei folgenden Abbruchkriterien erfüllt wird:

- Es wird die durch `etabound` gesetzte Grenze für den maximalen Fehler der Interpolation erreicht bzw. unterschritten (Interpolation wurde also erfolgreich durchgeführt).
- Die Kondition der Interpolationsmatrix überschreitet einen gesetzten Grenzwert `K`.
- Die Anzahl der Punkte der Menge Y stimmt mit einem gesetzten Grenzwert `maxnumactive` überein.

Den Großteil der benötigten Werte, die im Hauptprogramm `rahmen.m` zugewiesen werden, wird der Funktion `greedyschleife` durch globale Deklaration der entsprechenden Variablen zugänglich gemacht: hierunter fallen nicht nur `K` und `maxnumactive`, sondern auch die Dimension des Raumes `(dim)`, die endliche Menge paarweise verschiedener Punkte $X = \{x_1, \ldots, x_N\}$ `(SDpoints)`, deren Dimension `(SDdim)` und Anzahl `(SDnp)`, die Werte f_i `(funct)`, weiterhin die RBF-Funktion `RBFtype`, mit der die Interpolationsmatrix erzeugt wird, deren Ordnung `(RBForder)`, die Dimension des Polynomraumes `(RBFPolDim)` und der Skalierungsfaktor `(RBFscale)`.
Der Wert `etabound` wird extra übergeben. Dies hat seinen Grund:
`etabound` ist im Hauptprogrammm kein Skalar, sondern ein folgendermaßen erzeugter Vektor:

```
etabound(1)=max(abs(resid(:))); % Maximum der Fehlerfunktion
                                % auf X
i=1;
while etabound(i)*7/10>=eta_end  % >= gesetzter Fehlergrenze
    etabound(i+1)=etabound(i)*7/10;
    i=i+1;
end
```

und die Funktion `greedyschleife` wird im Hauptprogramm selbst in einer - hier nur auszugsweise angeführten - Schleife aufgerufen:

```
l=1;
while l<=length(etabound)
          [energy_new,numactive, eta, eta3, energy3, flag]=...
            greedyschleife(etabound(l),numactive,l,IP);
            ...
end
```

Daher müssen der Funktion `greedyschleife` noch weiterhin die aktuelle Anzahl der Punkte der Menge Y (`numactive`) und der Zähler der Iterationen (`l`) übergeben werden, damit die hier zu berechnenden Werte den einzelnen Vektoren richtig zugeordnet werden können.
Wie bereits erwähnt, beinhaltet `IP` die Indizes der Gittereckpunkte. Bei Verwendung der Thin Plate Splines stellt `IP` nun durch entsprechende Zuordnung in `greedyschleife` sicher, daß P_{Y_0} injektiv ist.
Nach Durchführung der Schleife gibt `greedyschleife` dem Hauptprogramm wiederum die nun aktuelle Anzahl der Punkte der Menge Y (`numactive`), dann sowohl das je Interpolation erreichte Fehlermaximum (`eta3`) und die jeweils zugehörige Energie der Interpolanten (`energy3`), als auch das durch die letzte Interpolation erreichte Fehlermaximum (`eta`) und die zuletzt erreichte Energie der Interpolanten (`energynew`) zurück. Desweiteren wird noch eine Variable `flag` übergeben, die im entsprechenden Fall angibt, daß die Kondition der Interpolationsmatrix nicht akzeptabel ist. Dies möge als Vorbeschreibung genügen.

Eine gute Übersicht über den Aufbau des Programms bietet nun der Pseudo-Code auf der folgenden Seite.
Das gesamte Programm ist in Abschnitt A.0.2 angeführt.

Funktion 1 greedyschleife.m

1: Deklariere die globalen Variablen.
2: **if** $i == 1$ **then**
3: **if** `RBFPolDim`> 0 **then**
4: Berechne die Inverse der Interpolationsmatrix (als Vorbereitung auf `matupdate.m`).
5: **end if**
6: **end if**
7: **while** $0==0$ **do**
8: Finde das Fehlermaximum und den zugehörigen Punkt $\tilde{x}$ auf X.
9: **if** Fehlermaximum $\leq$ `etabound` **then**
10: **break**
11: **end if**
12: **if** Kondition der Interpolationsmatrix $> K$ **then**
13: `flag`$= 1$ **break**
14: **end if**
15: **if** $|Y| =$ `maxnumactive` **then**
16: **break**
17: **end if**
18: **if** `mod(numactive,`$20)== 0$ **then**
19: Setze Inverse 'unbekannt' (zur Sicherung der Genauigkeit der Inversen).
20: **end if**
21: Erweitere Y um den Punkt $\tilde{x}$.
22: **if** Inverse der Interpolationsmatrix auf Y unbekannt **then**
23: Berechne die Inverse der Interpolationsmatrix auf $Y \cup \{\tilde{x}\}$. Ermittle mit ihrer Hilfe die Interpolante auf $Y \cup \{\tilde{x}\}$ und die Kondition der erweiterten Interpolationsmatrix.
24: **else**
25: Berechne die Inverse der Interpolationsmatrix auf $Y \cup \{\tilde{x}\}$ durch Rang-1-Erweiterung der Inversen der Interpolationsmatrix auf Y mit `matupdate0.m` bzw. `matupdate.m`. Ermittle mit ihrer Hilfe die Interpolante auf $Y \cup \{\tilde{x}\}$ und die Kondition der erweiterten Interpolationsmatrix .
26: **end if**
27: Berechne die Energie der Interpolanten auf Y.
28: Berechne die Fehlerfunktion auf X.
29: Berechne die Interpolante auf X.
30: **end while**

3.4.2 Das Rahmenprogramm

In dem Hauptprogramms `rahmen.m`, das schon auszugsweise vorgestellt wurde, werden einerseits die nötigen Vorbereitungen für `greedyschleife.m` getroffen und andererseits nach Durchführung der Interpolation die Ergebnisse visualisiert. Wiederum sei hier der Aufbau kurz in einem Pseudo-Code skizziert:

Funktion 2 rahmen.m

1: Deklariere die globalen Variablen.
2: Initialisiere die für alle Interpolationen gleichbleibenden Werte.
3: **for** $j = 1 : 5$ **do**
4: Wähle eine radiale Basisfunktion aus `fct2D.m` (für die Erzeugung der Werte $f_i \in \mathbb{R}, 1 \leq i \leq N$).
5: **for** $k = 1 : 5$ **do**
6: Wähle eine radiale Basisfunktion aus `r2bf` (für die Erzeugung der Interpolationsmatrix).
7: Lösche Ergebnisse - sofern durch vorherige Schleifendurchläufe vorhanden - durch leere Feldzuweisungen.
8: Initialisiere weitere gegebene Anfangswerte der Interpolation.
9: **while** etabound(i)$*$Konstante>= Fehlerminimum **do**
10: Konstruiere den Vektor `etabound` (dessen Einträge werden ein Abbruchkriterium für `greedyschleife` sein).
11: **end while**
12: **while** l<=length(etabound) **do**
13: Führe das Programm `greedyschleife` aus.
14: **if** Kondition der Interpolationsmatrix zu schlecht **then**
15: Setze die Skalierung der RBF-Funktion herab, lösche alle bisherigen Ergebnisse dieser Schleife, weise den Variablen wieder die Anfangswerte der Schleife zu und beginne diese Schleife von vorn.
16: **end if**
17: **end while**
18: Berechne die Abhängigkeit des Fehlers von der Anzahl der zu interpolierenden Punkte in der Funktion `linreg`.
19: Gebe gewisse Ergebnisse numerisch aus.
20: Plotte die Ergebnisse in der Funktion `plots`.
21: **end for**
22: **end for**

Im Anhang dieser Arbeit kann `rahmen.m` in Abschnitt A.0.1 in voller Länge eingesehen werden, ebenso ist die noch nicht vorgestellte Funktion `plots` im Anhang, in Abschnitt A.0.4, zu finden. Durch sie werden die Ergebnisse der Durchläufe visualisiert.

Kapitel 4

Numerische Versuche

In diesem Kapitel wird nun das in den vorigen Kapiteln entwickelte Verfahren unter Verwendung der in Abschnitt 1.1 vorgestellten radialen Basisfunktionen untersucht.
Nutzt man diese Funktionen einerseits als Wertelieferant für die Menge der zu interpolierenden Punkte und andererseits als "Interpolationswerkzeug", so ergeben sich aus den fünf vorgestellten Funktionen fünfundzwanzig Kombinationsmöglichkeiten zur Untersuchung der Interpolation mit Hilfe des Greedy-Algorithmus.
Vorweg sei erwähnt, daß aufgrund der Anzahl der Interpolationen auf die Darstellung aller durch die Funktion `plots` erzeugten Abbildungen verzichtet wird und stattdessen nur auf einige markante Testdurchläufe eingegangen werden wird.
Die hier nicht aufgeführten visualisierten Ergebnisse der verschiedenen Interpolationen können bei Interesse mit Hilfe der beigefügten Cd leicht erzeugt werden.

4.1 Die Testdurchläufe

Die Versuche finden auf einem quadratischen Gitter im Intervall $[-1,+1]^2$ mit $N = (142) \cdot (142) = 20.164$ Datenpunkten statt. Dabei wurde $h = \frac{2}{\sqrt{N-1}}$ gesetzt und das Gitter so gelegt, daß der Nullpunkt in der Mitte einer Gittermasche liegt, damit er nie mit einem Gitterpunkt zusammenfallen und somit nicht als Interpolationspunkt genutzt werden kann. Die dazugehörigen Werte $f_i \in \mathbb{R}, 1 \leq i \leq 20.164$ werden durch die jeweils genutzte radiale Basisfunktion geliefert. Zusätzlich wird pro Durchlauf eine radiale Basisfunktion als

die Funktion gewählt, auf der mit Hilfe des Greedy-Algorithmus interpoliert werden soll. Die Interpolation wird erfolgreich beendet, wenn der maximale Fehler von $r_j = f - s_{f,Y_j}$ die gegebene Fehlergrenze $\eta_{i+1} = \eta_i \cdot \frac{7}{10} \geq 0.005$[1] mit $\eta_0 = \max_{x \in X} |r_j(x)|$ erreicht beziehungsweise unterschreitet und zum Erreichen dieser Fehlergrenze nicht mehr als 200 Interpolationspunkte benötigt werden. Die Anfangsskalierung von $c_0 = 100$ wird nur herabgesetzt, wenn die Kondition der Interpolationsmatrix einen Wert von $4 \cdot 10^9$ überschreitet oder der maximal gewünschte Fehler auch bei 200 Interpolationspunkten nicht erreicht wird[2]. Für $k = 0, 1, \ldots$ und $c_k > 1$ geschieht die Herabsetzung durch $c_{k+1} = \left(\sqrt{c_k} - 0.5\right)^2$, ab einem Skalierungsfaktor von $c_k = 1$ geht es in einer geringeren Schrittweite hinab: $c_{k+1} = \left(\sqrt{c_k} - 0.1\right)^2$.
Die folgende Tabelle zeigt die im Algorithmus bei einer erfolgreichen Interpolation genutzte Skalierung der radialen Basisfunktionen Φ bei den jeweiligen Kombinationen von f und Φ.

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	20.25	16.00	20.25	0.49	0.81
G	12.25	6.25	6.25	0.16	0.16
IMQ	30.25	25.00	30.25	0.64	1.0
W	100.00	100.00	100.00	100.00	100.00
TPS	100.00	100.00	100.00	100.00	100.00

Wie zu erwarten, ist für die nicht-glatten Funktionen Φ, wie die Wendlandfunktion und die Thin Plate Splines, der Anfangswert des Skalierungsfaktors erhalten geblieben, während bei den analytischen Funktionen Φ (Multiquadrics, Inverse Multiquadrics und Gaußglocke) der Skalierungsfaktor herabgesetzt werden mußte, um eine zu schlechte Kondition der Interpolationsmatrix zu vermeiden. Insbesondere bei Kombinationen mit der Wendlandfunktion und den Thin Plate Splines mußte die der Skalierungsfaktor extrem herabgesetzt werden, um die Kondition der Interpolationsmatrix akzeptabel zu halten.
Um bei weiteren Durchläufen dieser Art Zeit zu sparen, kann folgende Bemerkung helfen:

[1] Eine Erklärung für die Wahl von η_i mit $i = 0, \ldots, n$ als stufenweise sinkende Fehlergrenze wird in 4.2.1 geliefert werden.

[2] Es sei daran erinnert, daß der Skalierungsfaktor bei der Erzeugung der Interpolationsmatrix in `r2bf` quadriert wird. Will man also einen Skalierungsfaktor von 100 benutzen, ist im Rahmenprogramm c=10 zu setzen.

Bemerkung 4.1 *Die Wahl des Skalierungsfaktor hängt von der genutzten Interpolationsmatrix ab. Im Falle nicht-glatter Funktionen ist eine hohe Skalierung sinnvoll. Bei analytischen Funktionen hingegen sollte der Skalierungsfaktor geringer ausfallen - insbesondere bei der Rekonstruktion nicht-glatter Funktionen wie in diesem Kapitel, also bei* $(142) \cdot (142)$ *Daten im Intervall* $[-1, 1]$*, einen Wert von 1 nicht überschreiten, um die Kondition der Interpolationsmatrix nicht in die Höhe schnellen zu lassen.*

Die nächste Tabelle zeigt die Anzahl der Punkte der Teilmenge $Y_j \subset X = \{x_1, \ldots, x_N\}$ für $N = 20.164$, die nötig ist, um den maximalen Fehler der Interpolation von $r_j = f - s_{f,Y_j}$ auf das gegebene Fehlermaximum auf der Menge X von 0.005 zu reduzieren.

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	21	24	21	120	73
G	21	21	26	155	114
IMQ	21	21	21	140	111
W	17	17	13	48	43
TPS	21	29	13	47	35

Auf den ersten Blick wirkt es überraschend, daß in der linken Hälfte der Tabelle keine großen Differenzen bei der Anzahl der nötigen Interpolationspunkte zwischen der Interpolation mit glatten und der Interpolation mit nicht-glatten Funktionen zu geben scheint. Berücksichtigt man aber die Tabelle der Skalierungsfaktoren, so lassen sich die erstaunlich geringen Werte in den unteren beiden Zeilen durch die Skalierung erzeugte "Breite" der Funktionen erklären, während bei den analytischen Funktionen der Vorteil der Glattheit durch einen (aufgrund der Kondition) geringen Skalierungsfaktor zunichte gemacht wird. Sehr deutlich ist die Auswirkung eines sehr geringen Skalierungsfaktors auf die Anzahl der nötigen Interpolationspunkte in der rechten oberen Ecke zu sehen.

Dennoch ist das Ergebnis insgesamt sehr zufriedenstellend:

Bemerkung 4.2 *Anstelle der bei einer gewöhnlichen Interpolation erforderlichen* $\mathcal{O}(N^3)$ *Rechenoperationen werden bei der Interpolation mit Hilfe des akkumulierenden Greedy-Algorithmus nur* $\mathcal{O}(n^3) + \mathcal{O}(n^2 \cdot N)$ *Rechenoperationen benötigt.*

In diesem konkreten Fall sind das anstelle von $\mathcal{O}(20.164^3)$ nur maximal $\mathcal{O}(155^3) + \mathcal{O}(155^2 \cdot 20.164)$. Ein durchaus sehr gutes Ergebnis.

Interessant ist diesbezüglich noch die Frage, ob es einen Unterschied macht, die Daten in Form von Gitterpunkten (mit gleichmäßigem Abstand voneinander) oder zufällig verteilt (im gleichen Intervall) gegeben zu haben. Ein Testdurchlauf mit zufällig gewählten Punkten[3] - durchgeführt unter denselben Nebenbedingungen wie der bereits vorgestellte Testdurchlauf - liefert jedoch kaum abweichende Ergebnisse (in A.1 ist eine Gegenüberstellung der Ergebnisse beider Varianten der gegebenen Datenmengen aufgeführt).
Bei einer genügend groß gegebenen Datenmenge ist die Art der Gegebenheit, ob gitterpunktförmig oder zufällig verteilt, also nicht weiter relevant.

4.2 Zwei detaillierte Beispiele

Zur näheren Untersuchung werden nun zwei Interpolationsläufe ausgewählt. Da die Ergebnisse bezüglich der zum Erreichen der Fehlergrenze notwendigen Interpolationspunkte bereits bekannt sind, können für die nähere Betrachtung folgende Durchläufe ausgewählt werden:

- einer der beiden mit der geringsten Anzahl benötigter Interpolationspunkte
  (f =Inverse Multiquadrics und Φ =Wendlandfunktion)
- der mit der größten Anzahl benötigter Interpolationspunkte
  (f =Wendland und Φ =Gaußglocke).

Ziel dieser Betrachtungen soll sein, eine Aussage über einen möglichen Zusammenhang zwischen der Anzahl der Interpolationspunkte n und dem Fehler η beziehungsweise den Fehlerschranken η_i machen zu können.
Desweiteren wird auch die Entwicklung der im zweiten Kapitel ausführlich untersuchten Interpolanten $\|s_{f,Y}\|$, die im folgenden mit σ bezeichnet werden wird, durch die folgenden Bilder veranschaulicht.

[3] Hierfür ist nur die Variable `nrand` auf 0 zu setzen.

4.2.1 Erstes Beispiel

Die Werte $f_k \in \mathbb{R}, 1 \leq k \leq 20.164$ werden in diesem Beispiel durch die Inversen Multiquadrics geliefert und die Wendlandfunktion liefert die Interpolationsmatrix.

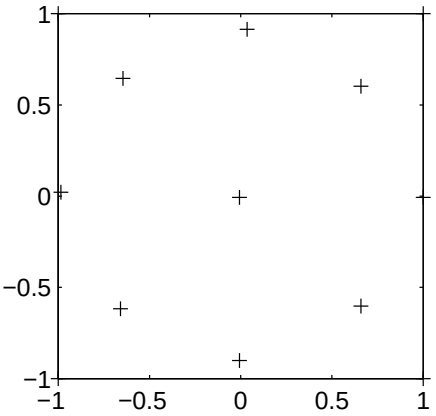

Abbildung 4.1: $f = \mathrm{IMQ}$

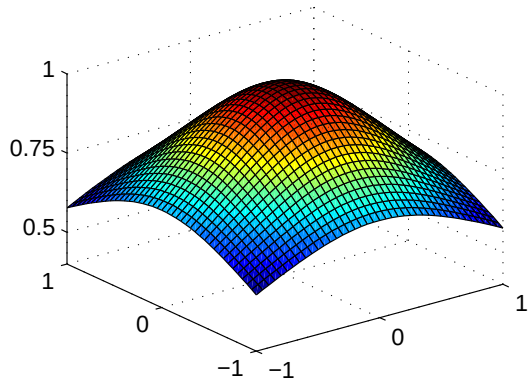

Abbildung 4.2: $\Phi = \mathrm{W}$

In Abbildung 4.1 sind die zur Interpolation genutzten zugrundeliegenden Stützstellen zu sehen, während Abbildung 4.2 die Interpolationsmatrix selbst darstellt.
Detaillierte Plots bezüglich der Beziehung zwischen dem Verhalten des Fehlers η und der Anzahl der Interpolationspunkte n sind auf der folgenden Seite in der linksseitigen Spalte zu sehen, wobei Abbildung 4.5 "nur" die Umkehrfunktion von Abbildung 4.3 darstellt, während Abbildung 4.7 (sie ist von rechts nach links zu betrachten) einen Zusammenhang zwischen den Fehlerschranken und der dafür benötigten Interpolationspunkten zeigt. Hier sind die Kosten, die eine bestimmte Fehlerstufe mit sich bringt, besonders deutlich erkennbar: es ist ablesbar, wieviel Punkte mehr benötigt werden, um einen geringeren Fehler zu erhalten, und dadurch abwägbar, ob sich ein geringerer Fehler "lohnt".
Es sind folgende Proportionalitätsbeziehungen erkennbar, die in den Abbildungen 4.3 und 4.7 durch rote Linien gekennzeichnet sind:

$$\eta = c \cdot n^{\alpha} \tag{4.1}$$

$$n_i = c^{-\gamma} \cdot \eta_i^{\gamma} \,. \tag{4.2}$$

Die rechtsspaltigen Abbildungen zeigen die Entwicklung der Energie der Interpolanten $\sigma := \|s_{f,Y}\|^2_{\mathcal{N}_\Phi}$ in verschiedenen Zusammenhängen. Wie nach Satz 2.3 zu erwarten, ist die Energiekurve der Interpolanten in bezug auf die Anzahl der Interpolationspunkte in Abbildung 4.4 monoton steigend.

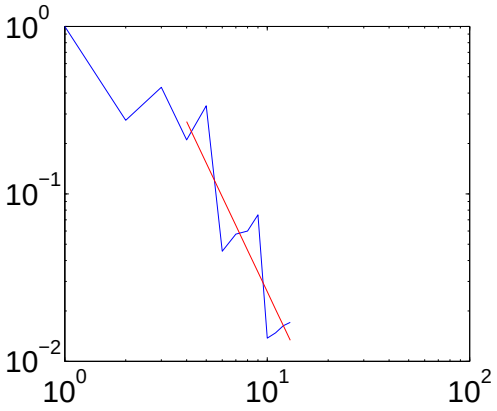

Abbildung 4.3: η versus n

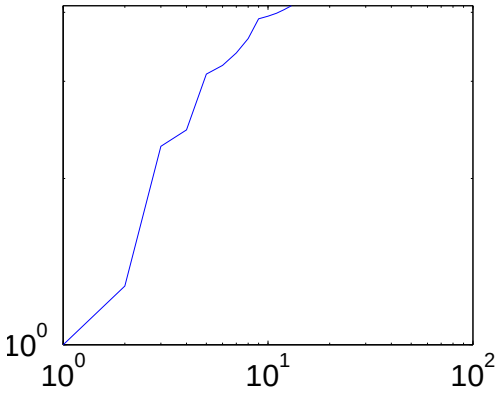

Abbildung 4.4: σ versus n

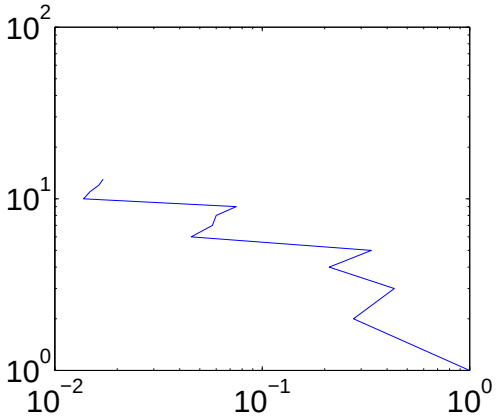

Abbildung 4.5: n versus η

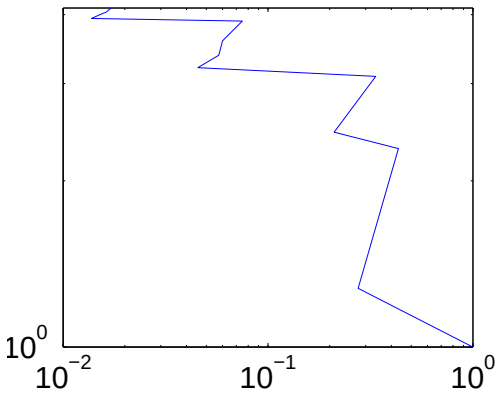

Abbildung 4.6: σ versus η

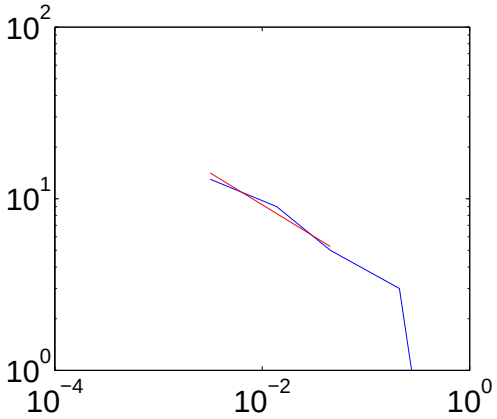

Abbildung 4.7: n_i versus η_i

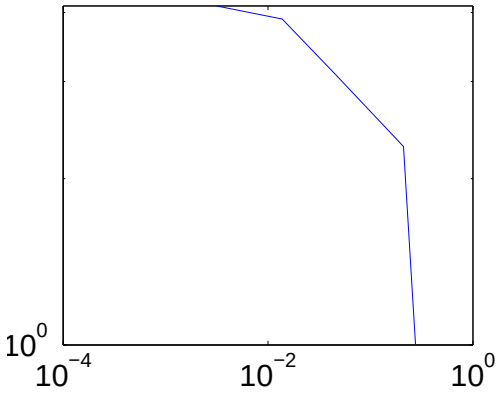

Abbildung 4.8: σ_i versus η_i

4.2.2 Zweites Beispiel

In diesem Beispiel werden die Werte $f_i \in \mathbb{R}, 1 \leq i \leq 20.164$ durch die Wendlandfunktion geliefert und die Interpolationsmatrix basiert auf der Gaußglocke.

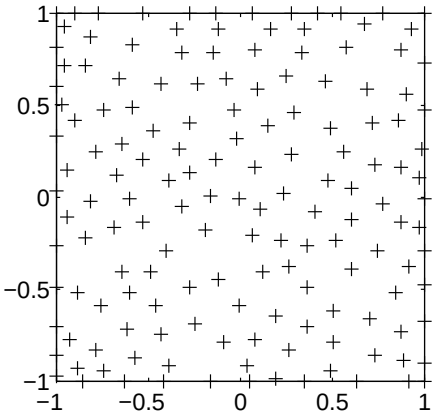

Abbildung 4.9: $f = \mathsf{W}$

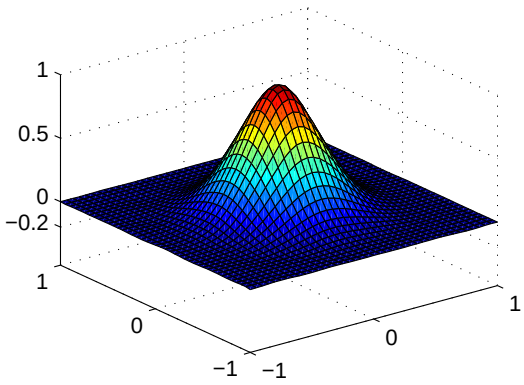

Abbildung 4.10: $\Phi = \mathsf{G}$

Die auf der folgenden Seite gezeigten Abbildungen sind wie im vorigen Beispiel aufgebaut.
Die hier teilweise erkennbare starke Oszillation der Kurven ist auf die schlechte Kondition der Interpolationsmatrix zurückzuführen.
Auffällig ist weiterhin, daß die Energie der Interpolanten anfangs kaum an Zuwachs gewinnt und am Ende fast senkrecht ansteigt. Der Fehlerfunktion wird also erst mit den letzten Interpolationpunkten die nötige Energie entzogen.
Auch hier sind die schon eben aufgezeigten Proportionalitätsbeziehungen zwischen dem Fehler und der "zugehörigen" Anzahl an Interpolationspunkten erkennbar und durch die roten Linien gekennzeichnet.
Betrachtet man die übrigen Testdurchläufe[4], kommt man zu der gleichen Feststellung, welche im nächsten Abschnitt noch einmal in bezug auf alle Testdurchlaüfe näher betrachtet wird.

[4]Wie schon erwähnt, wird auf die Abbildung der Bilder aller Testdurchläufe verzichtet, sie können aber mit Hilfe der Cd leicht rekonstruiert werden.

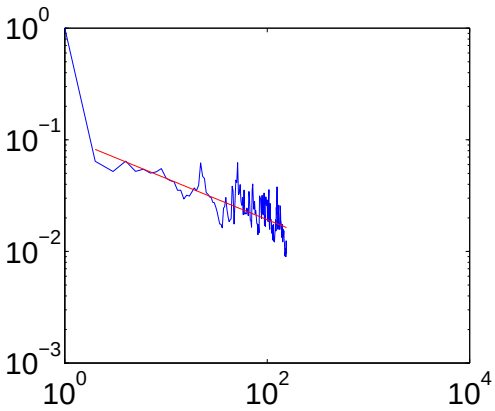

Abbildung 4.11: η versus n

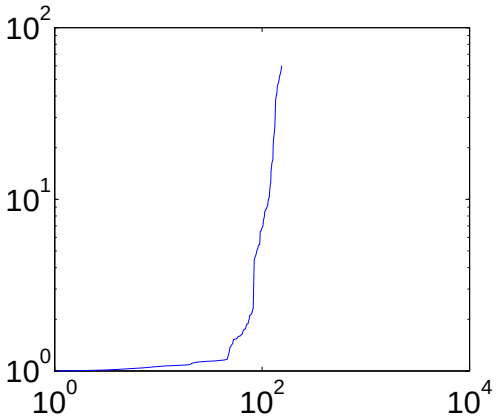

Abbildung 4.12: σ versus n

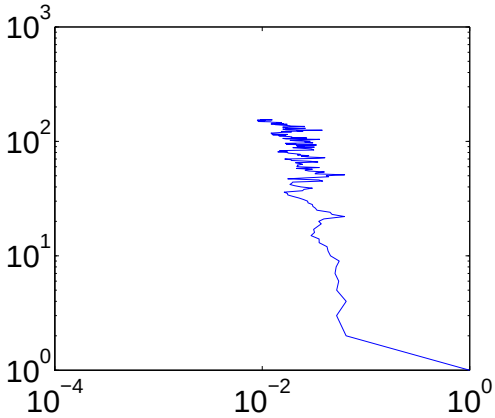

Abbildung 4.13: n versus η

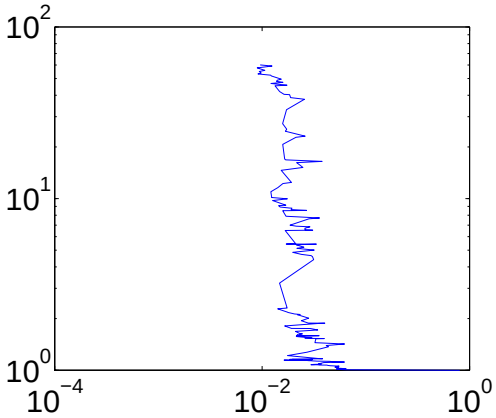

Abbildung 4.14: σ versus η

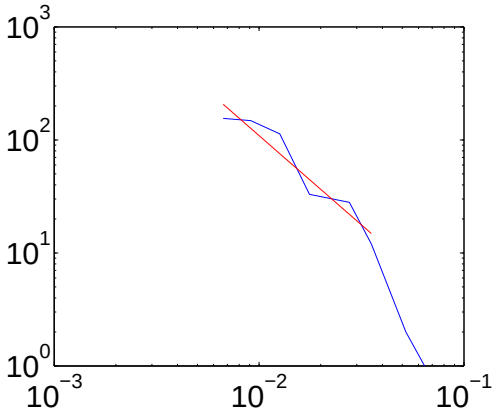

Abbildung 4.15: n_i versus η_i

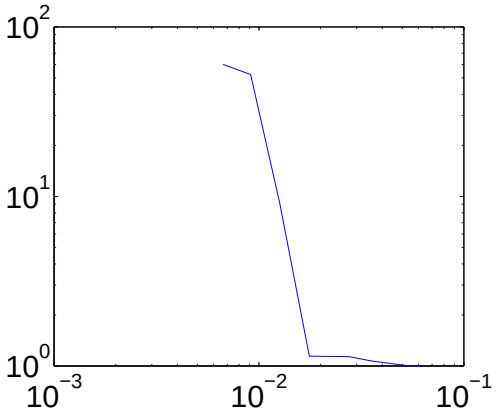

Abbildung 4.16: σ_i versus η_i

4.3 Bestimmung der Ordnung des Greedy-Verfahrens

Die in den Beispielen festgestellte Propotionalität zwischen dem Fehler η und der Anzahl der Interpolationspunkte n macht neugierig. Nimmt man anstelle der Anzahl der Punkte n die Dichte des Gebietes, gegeben durch $h := n^{-1/2}$, so ergibt sich die Beziehung

$$\eta = c \cdot h^{\alpha}$$

mit einem hier nicht weiter betrachteten Faktor c.
Für die einzelnen Kombinationen der verschiedenen f und Φ berechnet das Programm `linreg` folgende Ordnungen der Proportionalität:

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	3.36	5.23	2.39	2.72	2.81
G	4.25	4.00	2.48	0.74	2.82
IMQ	4.83	4.48	3.05	2.40	2.08
W	6.09	5.55	5.10	5.90	4.59
TPS	2.57	2.65	2.01	3.84	4.85

Die hier angeführten Werte sind leider nicht exakt, sie können sogar einen Fehler von bis zu 20 Prozent beinhalten. Betrachtet man die zugehörigen Abbildungen in Beispiel 1 und 2, so wird schnell klar, daß aufgrund der oszillierenden Kurven kein exakter Wert entstehen kann. Desweiteren hängen die Werte für α auch stark davon ab, welcher Anfangswert für die lineare Regression gewählt wird (in diesem Fall ist der erste gewertete Punkt für den Vektor x in `linreg.m` derjenige, für den gilt: die Differenz der zwei aufeinanderfolgenden Punkte ist negativ und der Betrag der beiden ist kleiner 0.1).

Dennoch kann aus der Tabelle folgendes Ergebnis herausgefiltert werden:
Die Ordnungen bei Kombinationen glatter Funktionen untereinander sind, wie erwartet, relativ gut. Es überrascht aber zunächst widerum, daß bei der Wendlandfunktion und den Thin Plate Splines als Interpolationsmatrix teilweise noch höhere Ordnungen bzw. keine niedrigen Ordnungen auftauchen. Aber es läßt sich wieder durch den hohen Skalierungsfaktor begründen, der die Funktionen ausreichend "breit" macht, so daß gute Werte erreicht werden können.

Durch die gesetzten Fehlerschranken $\eta_i = \eta_{i-1} \cdot \frac{7}{10} \geq 0.005$ mit $\eta_0 = \max_{x \in X} |r_j(x)|$ ist eine weitere Ordnungsbestimmung möglich. Die Proportionalität zwischen den Fehlerschranken η_i und der Dichte des Gebietes h_i läßt folgende Ordnungsbestimmung zu:

$$h_i = c^{-\gamma} \cdot \eta_i^{\gamma} \ .$$

Hier spielt nun auch die Größe des Faktors c eine entscheidende Rolle, da auch er mit γ potenziert ist. Ist zu der Konstanten c auch der Wert γ bekannt, ist dies eine entscheidende Information für den Benutzer: der zusätzliche Rechenaufwand, der mit einem geringeren "Fehlerwunsch" verbunden ist, ist ermittelbar. Der Benutzer kann also abwägen, ob sich ein geringer Fehler für ihn wirklich lohnt.
Die folgende Tabelle zeigt die Werte für γ, die sich für die in diesem Kapitel untersuchten vorgestellten Testdurchläufe ergeben, wenn man die Fehlerschranken $\eta_i < 0.05$ und die zugehörigen Gebietsdichten h_i mittels linearer Regression auswertet.

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	1.01	0.60	0.76	1.51	1.33
G	1.44	0.63	0.55	3.16	0.79
IMQ	1.52	0.66	0.78	1.88	1.64
W	1.27	0.64	0.74	0.82	0.98
TPS	1.46	1.24	1.47	1.06	0.72

Die Werte liegen größtenteils unter 1,5. Auffallend hoch ist der Wert für γ wiederum in dem am schlechtesten abschneidenden Lauf mit $f = W$ und $\Phi = G$.

4.4 Ausblick

Insgesamt erweist sich der akkumulierende Greedy-Algorithmus als sehr gute Methode, die Interpolation einer gegebenen, endlich großen Datenmenge und den dazugehörigen Werten zu beschleunigen. Akzeptiert man einen nur minimalem Restfehler, kann eine große Menge Rechenschritte und somit auch sonst benötigte Zeit eingespart werden.
Durchaus interessant wäre es, den in Kapitel 2 in Satz 2.5 aufgezeigten Sachverhalt

$$\|s_{f,Y\cup\{y\}}\|^2_{\mathcal{N}_\Phi} = \|s_{f,Y}\|^2_{\mathcal{N}_\Phi} + \frac{(f(y) - s_{f,Y}(y))^2}{P^2_Y(y)}$$

weiter zu untersuchen und eine weitere Optimierung des Verfahrens zu entwickeln, indem die Powerfunktion und ihr Verhalten bei weiteren Bedingungen an das Verfahren einer näheren Betrachtung unterzogen würde.
In diesem Zusammenhang ließe sich auch die in Satz 2.6 gezeigte Abschätzung

$$\|s_{f,Y_N}\|^2_{\mathcal{N}_\Phi} - \|s_{f,Y_1}\|^2_{\mathcal{N}_\Phi} \geq \sum_{j=1}^{N} \frac{\|r_j\|^2_{\infty,X}}{c}$$

durch eine genauere Aussage über c verschärfen.

Anhang A

Matlab-Programme

A.0.1 Das Rahmenprogramm

```
clear all;                    % leert Workspace
global RBFscale
global RBFtype
global RBForder
global RBFPolDim
global SDDim
global SDnp
global Genfct2Dchoice
global SDpoints
global activeset
global funct
global resid
global coeff
global fullcoeff
global B
global C
global D
global have_inverse
global maxnumactive           % Punkteschranke
global K                      % Grenzwert von cond(L)

dim=2;                        % Raumdimension
nrand=0;                      % 0=Gitter, 1=Zufall
num=20164;                    % Anzahl der Punkte
```

```
K=4e+9;
maxnumactive=200;
eta_end=0.005;                % minimaler gesuchter Fehler

[SDpoints,IP]=getpoints(num,dim,nrand);
alpha1=[];gamma1=[];delta1=[];numfinal=[];etafinal=[];
scale1=[];IP=[];
r=0;                          % Interpolationszaehler
for j=1:5
        Genfct2Dchoice=j;
    for k=1:5
        RBFtype=k;
        if k==1
            RBForder=1;
            RBFPolDim=1;
        elseif k==5
            RBForder=2;
            RBFPolDim=3;
        else
            RBForder=0;
            RBFPolDim=0;
        end

        resid=[];etabound=[];
        coeff=[];eta2=[];eta4=[];energy4=[];energy_final=[];
        numactive=[];numactive_final=[];
        B=[];C=[];D=[];ei=[];vi=[];

        r=r+1;
        RBFscale=10;              % wird in r2bf quadriert
        [SDnp,SDDim]=size(SDpoints); %[Anzahl, Dimension]
        activeset=zeros(SDnp,1);  % Liste der Indizes
                                  % der aktiven Punkte
        numactive=RBFPolDim;      % Anzahl der aktiven Punkte
        have_inverse=0;
        fullcoeff=zeros(SDnp+RBFPolDim,1);
        funct=fct2D(SDpoints(:,1),SDpoints(:,2));
        resid=funct; % Start mit Null-Interpolation
        fval=zeros(size(X));
        energy=0;
        numactive_final = numactive;
```

```
flag=0;

etabound(1)=max(abs(resid(:)));
i=1;
while etabound(i)*7/10>=eta_end
    etabound(i+1)=etabound(i)*7/10;
    i=i+1;
end

l=1;
while l<=length(etabound)
    [energy_new,numactive,eta,eta3,energy3,flag]=...
        greedyschleife(etabound(l),numactive,l,IP);
    if energy_new>0
        energy=energy_new;
    end
    energy_final(l)=energy;
    energy4=[energy4 energy3];
    eta2(l)=eta;
    eta4=[eta4 eta3];
    numactive_final(l)=numactive;
    if flag==0
        l=l+1;
    else
        RBFscalenew=RBFscale;

        if RBFscalenew<=1;
           RBFscale=RBFscalenew-0.1;
        else RBFscale=RBFscalenew-0.5;
        end
        if RBFscale==0;
            RBFscale=0
            break;
        end
        fullcoeff=[];
        coeff=[];eta4=[];energy4=[];energy_final=[];
        eta3=[];eta=[];energy3=[];energy_new=[];
        eta2=[];B=[];C=[];D=[];
        l=1;
        activeset=zeros(SDnp,1);
        energy=0; have_inverse=0;
```

```
                numactive=RBFPolDim;
                numactive_final=numactive;
                resid=funct;
            end;
        end
        scale1=[scale1 RBFscale];
        eta5=find(diff(eta4)<0);
        eta6=find(abs(eta4(eta5))<0.1);
        eta7=(eta6(1):length(eta4));
        [c, alpha]=linreg(eta7, eta4(eta7));

        halpha=alpha*(-2)
        alpha1=[alpha1 alpha];

        ei=find(eta4<0.05);
        [e, delta]=linreg(eta4(ei),ei);
        delta1=[delta1 delta];

        vi=find(eta2<0.05);
        [d, gamma]=linreg(eta2(vi),numactive_final(vi));

        hgamma=gamma*(-2)
        gamma1=[gamma1 gamma];
        numfinal=[numfinal numactive];
        eta1=eta;
        etafinal=[etafinal eta1];

        sprintf('Figure No.: %d \nf= %d \nPhi= %d \n...
          RBFPolDim= %d \nRBForder= %d \nRBFscale= %.1f \n...
          num= %d \nnumactive= %d \neta= %.4f \n...
          alpha= %.2f \ngamma= %f\.2n',...
          r,Genfct2Dchoice,RBFtype, RBFPolDim, RBForder,...
          RBFscale,num,numactive, eta1, alpha, gamma)

        fval=rbfevalmeshgrid(X,Y,SDpoints(activeset...
                             (1:numactive,1),:),coeff); % plot
        plots(eta2,eta4,eta7,energy4,energy_final,numactive,...
                numactive_final,alpha,gamma,c,d,vi,fval,r)
    end
end
```

A.0.2 Die Greedyschleife mit Hilfe der Inversenberechnung

```
function [energy,numactive,eta, eta1, energy1,flag] = ...
         greedyschleife(etabound,numactive,i,IP)
% Eingabe: etabound  = Skalar
%          numactive = Skalar
%          i         = Skalar
% Ausgabe: energy    = Skalar
%          energy1   = Vektor
%          numactive = Skalar
%          eta       = Skalar
%          eta1      = Vektor
%          flag      = Skalar

dim=2;
global K
global maxnumactive
global RBFtype
global RBForder
global RBFPolDim
global RBFscale
global Genfct2Dchoice
global funct
global SDpoints
global SDDim
global SDnp

global activeset   % Liste der Indizes (i_1,i_2,\ldots,i_n)
global resid       % Fehlerfunktion auf X
global coeff       % L"osungsvektor alpha auf Y
global fullcoeff   % L"osungsvektor alpha auf X
global B
global C
global D
global have_inverse

% Im Falle einer bedingt positiv definiten Interpolations-
% matrix wird im ersten Schleifendurchlauf die Inverse hierzu
% als Vorbereitung fuer das Programm matupdate.m berechnet:
```

```
if i==1
    if RBFPolDim>0
        activeset(1:RBFPolDim)=IP(2:RBFPolDim+1);
        have_inverse=1;
        ainv=inv(getrbfmatrix...
               (SDpoints(activeset(1:numactive,1),:),...
               SDpoints(activeset(1:numactive,1),:)));
        B=ainv(1:numactive,1:numactive);
        C=ainv(1:numactive,numactive+1:end);
        D=ainv(numactive+1:end,numactive+1:end);
    end;
end;
prhs=zeros(RBFPolDim+1,1); %Polynom
prhs(1,1)=1;
eta1=[];energy1=[];L=[];
energy=0;
while 0==0                                  % GREEDYSCHLEIFE:
        flag=0;
        [eta, maxind]=max(abs(resid));      % Fehlermaximum
        if eta<etabound                     % Abbruchkriterien:
            break;                          % Fehlergrenze er-
        end;                                % reicht,
        if K<cond(L)                        % Kondition der
            flag=1;                         % Interpolations-
            Kond=cond(L);                   % matrix zu
            break;                          % schlecht
        end;
        if numactive==maxnumactive          % |Y|=geg. Grenze
            numactive;
            break;
        end;
        if mod(numactive,20)==0             % Sicherung der Ge-
            have_inverse=0;                 % nauigkeit der
        end;                                % Inversen
        eta1=[eta1 eta];
        numactive=numactive+1;              % Erweiterung von
        activeset(numactive,1)=maxind;      % Y um Punkt maxind
        if have_inverse==0                  % Berechnung der Inv.
            have_inverse=1;                 % und Interpolation
            ainv=inv(getrbfmatrix...
                  (SDpoints(activeset(1:numactive,1),:),...
```

```
              SDpoints(activeset(1:numactive,1),:)))
        B=ainv(1:numactive,1:numactive);
        if RBFPolDim>0
            C=ainv(1:numactive,numactive+1:end);
            D=ainv(numactive+1:end,numactive+1:end);
            coeff=[B ; C']*funct(activeset(1:numactive,1));
            L=[B C; C' D];
        else
            coeff=B*funct(activeset(1:numactive,1));
            L=[B];
        end
    else
        v=rbfevallarge(SDpoints(activeset...
            (1:numactive,1),:),SDpoints(maxind,:),prhs);
        a=v(1:numactive-1,1);
        g=v(numactive,1);
        if RBFPolDim>0
            p=polynomials(SDpoints(maxind,:),RBForder)';
            [B, C, D]=matupdate(B,C,D,a,p,g);
            coeff=[B ; C']*funct(activeset(1:numactive,1));
            L=[B C; C' D];
        else
            B=matupdate0(B,a,g);
            coeff=B*funct(activeset(1:numactive,1));
            L=[B];
        end
    end
    energy=[abs(coeff(1:numactive,1)'*...   % Energie d.
          funct(activeset(1:numactive,1)))];% Interpolanten
    energy1=[energy1 energy];
    resid=funct-rbfevallarge(SDpoints,...   % verbl. Fehlerfkt.
      SDpoints(activeset(1:numactive,1),:), coeff);% auf X
    fullcoeff(activeset(1:numactive,1),1)=...
                                  coeff(1:numactive,1);
    if RBFPolDim>0
        fullcoeff(SDnp+1:end,1)=coeff(numactive+1:end,1);
    end
end
```

A.0.3 Die Greedyschleife mit Hilfe des Cholesky-Verfahrens

Das Programm `greedyschleifechol.m` ist wie die gerade vorgestellte Funktion `greedyschleife.m` aufgebaut, nur mit dem Unterschied, daß die Rang-1-Erweiterung der Interpolationsmatrix hier nicht mit Hilfe der Inversen, sondern mit Hilfe der Cholesky-Zerlegung und (zwei linken Divisionen) berechnet wird und dadurch auf positiv definite Interpolationsmatrizen (also auf der Gaußglocke, den Inversen Multiquadrics oder der Wendlandfunktion basierenden Matrizen) beschränkt ist.

```
function [energy,numactive,eta, eta1, energy1, flag]=...
    greedyschleifechol(etabound,numactive)

global maxnumactive
global RBFscale
global RBFtype
global SDDim
global SDnp
global Genfct2Dchoice
global SDpoints
global activeset
global funct
global resid
global coeff
global fullcoeff
global K
global have_chol
global T

prhs=zeros(1,1);
prhs(1,1)=1;
eta1=[];energy1=[];L=[];
energy=0;
while 0==0                                  % GREEDYSCHLEIFE
    flag=0;
    [eta, maxind]=max(abs(resid));
    if eta<etabound
        break;
    end;
```

```
    if mod(numactive,20)==0
        have_chol=0;
    end;
    if K<cond(L)
        flag=1;
        break;
    end;
    if numactive==maxnumactive
        flag=1;
        break;
    end
    eta1=[eta1 eta];
    numactive=numactive+1
    activeset(numactive,1)=maxind;
    if have_chol==0
        have_chol=1;
        an=getrbfmatrix(SDpoints(activeset(1:numactive,1),:)...
            ,SDpoints(activeset(1:numactive,1),:));
        A=an(1:numactive,1:numactive);
        T=chol(A);
        coeff=T\(T'\funct(activeset(1:numactive,1)))
        L=[T'*T];
    else
        v=rbfevallarge(SDpoints(activeset(1:numactive,1),:),...
            SDpoints(maxind,:),prhs);
        a=v(1:numactive-1,1);
        g=v(numactive,1);
        T=cholesky0(T,a,g);
        coeff=T\(T'\funct(activeset(1:numactive,1)));
        L=[T'*T];
    end
    energy=[abs(coeff(1:numactive,1)'*...
                            funct(activeset(1:numactive,1)))];
    energy1=[energy1 energy];
    resid=funct-rbfevallarge(SDpoints,...
        SDpoints(activeset(1:numactive,1),:), coeff);
    fullcoeff(activeset(1:numactive,1),1)=coeff(1:numactive,1);
end
```

A.0.4 Das Plot-Programm

```
function plots(eta2,eta4,eta7,energy4,energy_final,...
    numactive,numactive_final,alpha,gamma,c,d,vi,fval,r)

global Genfct2Dchoice
global RBFtype
global SDpoints
global activeset

[X,Y]=meshgrid(-1:0.05:1,-1:0.05:1);
nframes=1; % Bildzaehler

h=figure;
scrsz = get(0,'ScreenSize'); % Bildgroesse
my_figure = figure('Position',[ 10 10 scrsz(3)-350 scrsz(4)-80 ]);
figure(my_figure);

subplot(4,2,1);             % f
    plot(SDpoints(activeset(1:numactive,1),1),...
                  SDpoints(activeset(1:numactive,1),2),'r+');
    pbaspect([1 1 1]);
    switch(Genfct2Dchoice)
        case 1
            title('f=MQ');
        case 2
            title('f=Gauss');
        case 3
            title('f=IMQ');
        case 4
            title('f=Wendland');
        case 5
            title('f=TPS');
        case 6
            title('f=Peaks');
        otherwise
            title('f=untitled');
        end
subplot(4,2,2);           % Phi
    surf(X,Y,fval);
    switch(RBFtype)
```

```
        case 1
            title('phi=MQ');
        case 2
            title('phi=Gauss');
        case 3
            title('phi=IMQ');
        case 4
            title('phi=Wendland');
        case 5
            title('phi=TPS');
        otherwise
            title('untitled');
        end
subplot(4,2,3);          % eta versus numactive
    loglog(1:length(eta4),eta4);
    eta4_fit = c*(eta7).^alpha;
    hold on;             % Ordnung
    loglog( eta7, eta4_fit,'r-');
    hold off;
subplot(4,2,4);          % Int.-energie versus numactive
    loglog(1:length(energy4), energy4);
subplot(4,2,5);          % numactive versus eta
    loglog(eta4,1:length(eta4))
subplot(4,2,6);          % Int.-Energie versus eta
    loglog(eta4,energy4)
subplot(4,2,7);          %numactive_i versus eta_i
    loglog(eta2,numactive_final);
    numactive_fit= d*(eta2(vi)).^(gamma);
    hold on;             % Ordnung
    loglog(eta2(vi), numactive_fit,'r-');
    hold off;
subplot(4,2,8);          % Int.-Energie versus eta_i
    loglog(eta2, energy_final);
F(nframes)=getframe(my_figure);
nframes=nframes+1;
```

A.1 Testläufe mit Gitter- und Zufallspunkten im Vergleich

Die folgenden durch (G) markierten Tabellen beinhalten die Ergebnisse der in 4.1 vorgestellten Testläufe, die zugrunde liegende Datenmenge besteht hier aus Gitterpunkten (gekennzeichnet durch (G)) wie bereits in 4.1 beschrieben. Die mit (Z) markierten Tabellen beinhalten die Ergebnisse der gleichen Testdurchläufe, also mit denselben Nebenbedingungen wie in 4.1 beschrieben, mit dem einzigen Unterschied, daß die zugrunde liegende Datenmenge hier aus zufällig gewählten Punkten besteht.

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	20.25	16.00	20.25	0.49	1.00
G	12.25	6.25	4.00	0.09	0.25
IMQ	36.00	25.00	36.00	0.81	1.00
W	100.00	100.00	100.00	100.00	100.00
TPS	100.00	100.00	100.00	100.00	100.00

Abbildung A.1: Skalierung (G)

$\Phi \backslash f$	MQ	G	IMQ	W	TPS
MQ	20.25	16.00	20.25	0.49	0.81
G	12.25	6.25	6.25	0.16	0.16
IMQ	30.25	25.00	30.25	0.64	1.0
W	100.00	100.00	100.00	100.00	100.00
TPS	100.00	100.00	100.00	100.00	100.00

Abbildung A.2: Skalierung (Z)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	21	24	21	120	73
G	21	21	26	155	114
IMQ	21	21	21	140	111
W	17	17	13	48	43
TPS	21	29	13	47	35

Abbildung A.3: Anzahl nötiger Interpolationspunkte (G)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	21	21	22	117	87
G	21	21	23	123	154
IMQ	21	21	21	132	109
W	17	19	13	41	42
TPS	26	30	14	50	43

Abbildung A.4: Anzahl nötiger Interpolationspunkte (Z)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	3.36	5.23	2.39	2.72	2.81
G	4.25	4.00	2.48	0.74	2.82
IMQ	4.83	4.48	3.05	2.40	2.08
W	6.09	5.55	5.10	5.90	4.59
TPS	2.57	2.65	2.01	3.84	4.85

Abbildung A.5: Ordnung α (G)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	4.40	4.21	3.46	2.70	2.77
G	4.21	3.49	3.83	0.87	2.56
IMQ	4.26	4.00	3.19	2.69	2.52
W	6.07	5.94	5.16	6.49	4.70
TPS	3.22	3.12	2.52	3.56	4.40

Abbildung A.6: Ordnung α (Z)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	1.01	0.60	0.76	1.51	1.33
G	1.44	0.63	0.55	3.16	0.79
IMQ	1.52	0.66	0.78	1.88	1.64
W	1.27	0.64	0.74	0.82	0.98
TPS	1.46	1.24	1.47	1.06	0.72

Abbildung A.7: Ordnung γ (G)

$\Phi\backslash f$	MQ	G	IMQ	W	TPS
MQ	1.02	1.20	0.62	1.74	1.64
G	1.62	0.68	1.22	3.55	1.41
IMQ	1.03	0.55	0.65	1.73	1.63
W	1.33	0.75	0.74	0.81	0.99
TPS	1.08	1.08	1.01	1.03	0.89

Abbildung A.8: Ordnung γ (Z)

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. Robert Schaback für die interessante Themenstellung, die hervorragende Betreuung während der Anfertigung dieser Arbeit und insbesondere auch für die Bereitstellung der von ihm geschriebenen Programme bedanken.
Mein Dank gilt weiterhin Jörg Dittmar, der für meine Fragen immer ein offenes Ohr hatte.
Bei Sören Lemke und meiner Familie möchte ich mich für die Unterstützung in allen "nichtmathematischen" Lebensbereichen während des gesamten Studiums bedanken.

Literaturverzeichnis

[1] Stefano De Marchi, Robert Schaback und Holger Wendland: Optimal Data-Independent Point Locations for Radial Basis Function Interpolation. Preprint Göttingen/Verona, 2003.

[2] Günter Gramlich und Wilhelm Werner: *Numerische Mathematik mit Matlab.* dpunkt.verlag, Heidelberg, 2000.

[3] J. C. Mairhuber: On Haar's Theorem Concerning Chebychev Approximation Problems Having Unique Solutions. *Proc. Amer. Math. Soc. 7* S. 609–615, 1956.

[4] Robert Schaback: Native Spaces of Radial Basis Functions 1. In: *New Developments in Approximatios Theory (International Series of Numerical Mathematics)*, herausgegeben von M. W. Mueller, M. D. Buhmann, D. H. Mache und M. Felten, Bd. 132, S. 255–282, Birkhaeuser, Basel, 1999.

[5] Robert Schaback: Reconstruction of Multivariate Functions from Scattered Data. Manuscript,
http://www.num.math.uni-goettingen.de/schaback/teaching.html,
1997.

Versicherung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Göttingen, den 12.03.2004