Neue Verfahren zur Approximation mit radialen Basisfunktionen und numerische Untersuchung der Dagum-Funktionen

Dissertation
zur Erlangung des akademischen Grades
„Doktor der Naturwissenschaften“
(Dr. rer. nat.)

am Fachbereich
Mathematik und Informatik, Physik, Geographie
der Justus-Liebig-Universität Gießen

vorgelegt von
Dipl. Math. René Hofmann

14. Mai 2013

Disputation am 23. Juli 2013

Für Katja

Diese Arbeit enstand während meiner Tätigkeit an der Justus-Liebig-Universität Gießen und hat meinen Alltag über 5 Jahre geprägt.

Für die hervorragende Betreuung in dieser Zeit als Doktorvater und Vorgesetzter möchte ich mich bei Herrn Professor Buhmann bedanken, dessen Engagement, hilfreiche Anregungen und gute Ideen mir immer sehr geholfen haben.

Abschließend möchte ich mich noch ganz herzlich bei den Mitarbeitern und Mitarbeiterinnen des Lehrstuhls für numerische Mathematik an der Justus-Liebig Universität Gießen für die spannende und schöne Zeit bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In dieser Arbeit werden zwei neue Verfahren zur Approximation mit radialen Basisfunktionen eingeführt und deren numerische Ergebnisse miteinander verglichen. Außerdem werden wir eine neue Familie von radialen Basisfunktionen, die Dagum-Funktionen, untersuchen und ebenfalls numerisch auswerten.

Wir werden uns dabei in den numerischen Untersuchungen auf das folgende Approximationsproblem konzentrieren.

> *Zu gegebenen Daten, bestehend aus Datenpunkten* $x_1, \ldots, x_n$ *des* $\mathbb{R}^d$ *und dazu entsprechenden Werten* $f_i = f(x_i)$, $i = 1, \ldots, n$, *aus* $\mathbb{R}$, *suchen wir eine Funktion (Approximante)* $s : \mathbb{R}^d \mapsto \mathbb{R}$ *zu einer Funktion* $f : \mathbb{R}^d \mapsto \mathbb{R}$ *für die* $|s(x_i) - f(x_i)| < \delta$, $i = 1, \ldots, n$, *zu einem* $\delta > 0$ *gilt.*

Um die Approximante s zu berechnen, werden wir zum einen Interpolations-, zum anderen Quasi-Interpolationsalgorithmen verwenden. Für die Interpolation einer Funktion f zu den Daten $(x_i, f_i) \in \mathbb{R}^d \times \mathbb{R}$ wird die Approximante s zu

$$s(x) = \sum_{j=1}^{n} c_j B_j(x),$$

für entsprechende Koeffizienten c_j, $j = 1, \ldots, n$ und Basisfunktionen B_j, $j = 1, \ldots, n$. Außerdem fordern wir dann $s(x_i) = f(x_i)$ für alle $i = 1, \ldots, n$, das heißt wie oben für $\delta = 0$. Die Basisfunktionen werden in unserem Fall radial symmetrische Basisfunktionen folgender Form sein

$$B_j(x) := \phi(\| \cdot - x_j\|_2),\ x_j \in \mathbb{R}^d,\ j = 1, \ldots, n,$$

bzw. Linearkombinationen davon. Damit ist die Approximante s eine endliche Linearkombination von Translaten der Form $\phi(\| \cdot - x_i\|_2)$, $x_i \in \mathbb{R}^d$. Die verwendete Norm im Argument der Funktion ϕ ist standardmäßig die euklidische Norm. Diese Wahl ist

nicht beliebig, da für bestimmte p-Normen das Approximationsproblem nicht lösbar sein kann [3]. Die Basisfunktionen sind *radial symmetrisch*, da der Funktionswert von $\phi(\|\cdot\|_2)$ nur von der euklidischen Distanz des Arguments zum Nullpunkt abhängt und damit insbesondere rotationsinvariant ist. Die Datenpunkte x_i, $i = 1, \ldots, n$, werden zweierlei verwendet, als Translationen und als Interpolationspunkte. Formales zur Interpolation und allgemeine Eigenschaften radialer Basisfunktionen behandeln wir ausführlich in Kapitel 2.

Als zusätzlichen numerischen Algorithmus führen wir die Quasi-Interpolation ein, das heißt wir verwenden den Approximante

$$s_h(x) = \sum_{j \in \mathbb{Z}^d} f(jh)\psi(x/h - j),\ x \in \mathbb{R}^d, \tag{1.0.1}$$

zu einer Gitterfeinheit $h > 0$ und

$$\psi(x) = \sum_{k \in \mathbb{Z}^d} c_k \phi(\|x - k\|),\ x \in \mathbb{R}^d. \tag{1.0.2}$$

Die Herleitung der Berechnung der Koeffizienten c_k, $k \in P \subset \mathbb{Z}^d$, wird ebenfalls in Kapitel 2 formal eingeführt und in Kapitel 5 für die Thin-Plate-Splines und shifted-Thin-Plate-Splines konkretisiert.

In Kapitel 3 führen wir die Dagumfunktionen

$$\varphi_{\beta,\gamma}(r) = 1 - \left(\frac{r^\beta}{1 + r^\beta}\right)^\gamma,\ r \geq 0,$$

als radiale Basisfunktionen $\varphi_{\beta,\gamma}(\|\cdot - x_i\|_2)$ ein und beschreiben die bereits bekannten notwendigen und hinreichenden Bedingungen an die Parameter β und γ, damit $\varphi_{\beta,\gamma}$ vollständig monoton ist [4]. Zusätzlich werden wir als Neuerung die hinreichenden Aussagen bzgl. der Ableitung der Dagumfunktionen für einen Spezialfall erweitern.

Wir werden sehen, dass die Interpolation bei der Verwendung von radialen Basisfunktionen numerischen Einschränkungen in der Anzahl der Interpolationspunkte und - bei bestimmten radialen Basisfunktionen - in der Wahl des Glättungsparameters unterliegen kann. Für große n ($\gg 100$) kann die Interpolationsmatrix numerisch instabil und deren Konditionszahl in der Größenordnung 10^{20} (oder größer) sein. Der Glättungsparameter $\varepsilon > 0$ des Gaußkerns $e^{-\varepsilon^2 r^2}$, der inversen Multiquadrics $(1 + \varepsilon^2 r^2)^{-1}$ oder der shifted-Thin-Plate-Splines $(1 + \varepsilon^2 r^2) \log \sqrt{1 + \varepsilon^2 r^2}$ führt für $\varepsilon \to 0$ ebenfalls zu schlecht konditionierten Interpolationsmatrizen. Der Contour-

Padé-Algorithmus von Fornberg et. al. [7, 2004] kann mit sehr kleinen ε-Werten in den radialen Basisfunktionen umgehen, ist aber nicht auf viele Interpolationspunkte anwendbar. Der RBF-QR-Algorithmus, ebenfalls von Fornberg et. al. [5, 2009], kann auch für eine große Anzahl an Interpolationspunkte (> 100), sowie für sehr kleine ε-Werte ($< 10^{-10}$) verwenden werden. Dieser Algorithmus ist allerdings nur für den Gaußkern sowie für Besselfunktionen im $\mathbb{R}^2$ konstruiert. Die Erweiterung für $\mathbb{R}^1$ und $\mathbb{R}^3$ ist seit kurzem in [6, 2011] verfügbar. Wir werden in Kapitel 4 einen neuen Algorithmus einführen, der auf dem RBF-QR-Algorithmus für $\mathbb{R}^2$ aufbaut und für die inverse Multiquadrics, deren Verallgemeinerung und die Multiquadrics für kleine ε-Werte und viele Interpolationspunkte eine stabile Auswertung ermöglicht.

In Kapitel 6.2 werden wir die Ergebnisse für die Dagumfunktionen aus Kapitel 3 verwenden, um $\varphi_{\beta,\gamma}$ für unterschiedliche Parameterwerte von β und γ auf ihre Interpolationsfehler zu untersuchen. Der Vergleich der numerischen Ergebnisse des RBF-QR- und RBF-QR-GQ-Algorithmus folgt in Kapitel 6.3 und die numerische Betrachtung des RBF-QI-Algorithmus für die shifted-Thin-Plate-Splines in Kapitel 6.4.

Kapitel 2

Grundlagen

Radiale Basisfunktionen $\Phi : \mathbb{R}^d \mapsto \mathbb{R}$ werden zu einer Funktion $\varphi : \mathbb{R}^+ \mapsto \mathbb{R}$ über $\Phi(x) := \varphi(\|x\|_2)$ definiert. Für das Argument r von φ wird nur die Länge des Vektor $x \in \mathbb{R}^d$ übertragen, nicht aber dessen Richtung. Realisiert wird dies durch die Anwendung der euklidischen Norm. Wir erhalten so einen geometrischen Bezug zur Radialität der Funktion, wie in Abbildung 2.1 am Beispiel des Gaußkerns

$$\varphi(x) = e^{-\varepsilon^2\|x\|_2^2},\ \varepsilon > 0,\ x \in \mathbb{R}^d, \tag{2.0.1}$$

zu sehen ist.

Linear unabhängige Basisfunktion Φ_i, $i = 1, \ldots, n$, erhalten wir, indem die Funktion φ an paarweise verschiedenen Centerpunkte x_i, $j = 1, \ldots, n$, zentriert wird. Der Gaußkern selbst in (2.0.1) ist am Nullpunkt zentriert.

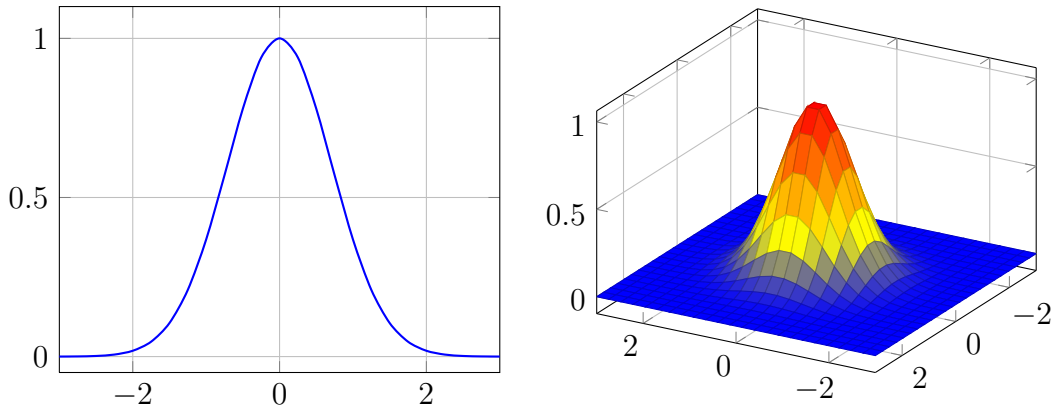

Abbildung 2.1: Gaußkern $e^{-\|x\|_2^2}$, $x \in \mathbb{R}^d$, für $d = 2, 3$

Die Approximante zu den radialen Basisfunktionen $\Phi_i(x) := \varphi(\|x - x_i\|_2)$, $i = 1, \ldots, n$, ist von der Form

$$s(x) = \sum_{j=1}^{n} c_j \Phi_j(x) = \sum_{j=1}^{n} a_j \varphi(\|x - x_j\|_2),$$

für geeignete Koeffizienten c_j, $j = 1, \ldots, n$. Für die Interpolation werden zur Bestimmung der Koeffizienten die Centerpunkte auch als Interpolationspunkte verwendet, so dass folgende symmetrische Matrix entsteht

$$\{\varphi(\|x_i - x_j\|_2)\}_{i,j=1}^{n}. \tag{2.0.2}$$

Für die Funktion $e^{-\varepsilon^2 \|x\|_p^2}, \varepsilon > 0$, ist die Interpolationsmatrix für beliebige paarweise verschiedene Interpolationspunkte nichtsingulär, falls p-Normen mit $p \in (1, 2]$ verwendet werden (siehe [3, Buhmann]). Dies ist nicht immer für alle $p \in (1, 2]$ gegeben. Beispiele für Interpolationspunkte und p-Normen, die zu singulären Matrizen führen können, finden sich in [2, Baxter].

Für die weiteren Betrachtungen verwenden wir standardmäßig die euklidische Norm und setzen dazu $\|\cdot\| := \|\cdot\|_2$. Wir werden im nächsten Kapitel Bedingungen an die radiale Basisfunktion aufzeigen, die gewährleisten, dass die Interpolationsmatrix (2.0.2) nichtsingulär ist. Dies ist insbesondere für vollständig monotone Funktionen g mit $g(t) = \varphi(\sqrt{t})$ gegeben. Diese Funktionen erfüllen folgende Bedingung

$$(-1)^{\ell} g^{(\ell)}(t) \geq 0 \text{ für alle } t \in \mathbb{R}_+. \tag{2.0.3}$$

Beispiele dafür sind der Gaußkern (2.0.1) mit $g(t) = \exp -\varepsilon^2 t$ und die inverse Multiquadrics

$$\varphi_\varepsilon(r) = \frac{1}{\sqrt{1 + \varepsilon^2 r^2}},\ \varepsilon > 0. \tag{2.0.4}$$

mit $g(t) = 1/\sqrt{1 + \varepsilon^2 t}$. Wir werden außerdem zeigen, dass auch die Interpolationsmatrizen von Basisfunktionen, deren Ableitung vollständig monoton ist, nicht aber sie selbst, nichtsingulär sind. Dies gilt beispielsweise für die Multiquadrics

$$\varphi(r) = \sqrt{1 + \varepsilon^2 r^2},\ \varepsilon > 0. \tag{2.0.5}$$

Für Basisfunktionen, die erst ab der k-ten Ableitung vollständig monoton sind, wird

der Interpolant zu

$$s(x) = \sum_{j=1}^{n} \lambda_j \varphi(\|x - x_j\|_2) + \sum_{|\alpha|<k} \lambda_\alpha x^\alpha,$$

wobei α ein Multiindex ist. Die Interpolationsmatrizen haben in diesem Fall die Gestalt

$$A = \begin{pmatrix} \{\varphi(\|x_i - x_j\|_2)\}_{i,j=1}^{n} & \{(x_i^\alpha)\}_{i,|\alpha|<k}^{n} \\ \left(\{(x_i^\alpha)\}_{i,|\alpha|<k}^{n}\right)^T & 0 \end{pmatrix}.$$

Alternativ zur vollständigen Monotonie können obige Eigenschaften der Interpolationsmatrix auch über Bedingungen an die Fouriertransformation $\hat{\varphi}$ von φ hergeleitet werden. Dies werden wir ebenfalls im folgenden Kapitel konkretisieren.

2.1 Eigenschaften radialer Basisfunktionen

Wir betrachten zunächst wichtige Definitionen und Eigenschaften radialer Basisfunktionen.

Definition 2.1.1 (Radiale Basisfunktion)**.** *Eine Funktion* $\Phi : \mathbb{R}^d \mapsto \mathbb{R}$ *heißt radiale Basisfunktion, falls eine Funktion* $\varphi : [0, \infty) \mapsto \mathbb{R}$ *existiert, so dass*

$$\Phi(x) = \varphi(r), \text{ mit } r = \|x\|,$$

wobei $\|\cdot\|$ *die euklidische Norm des* $\mathbb{R}^d$ *ist.*

Die Funktionen $\varphi(r)$ und $\Phi(x)$ können dabei als zusätzliches Argument einen Glättungsparameter $\varepsilon > 0$ oder $c > 0$ enthalten

$$\varphi_\varepsilon(r) := \varphi(r, \varepsilon) \tag{2.1.6}$$

bzw.

$$\varphi_c(r) := \varphi(r, c). \tag{2.1.7}$$

Die unterschiedliche Notation der Glättungsparameter verwenden wir aufgrund unterschiedlicher Grenzbetrachtung

$$\varepsilon \to 0 \text{ und } c \to \infty.$$

Diese treten zum Beispiel in der Multiquadrics (2.0.4) in der Form $\sqrt{1 + (\varepsilon r)^2}$ und $\sqrt{c^2 + r^2}$ auf oder können benutzt werden, um andere radiale Basisfunktionen zu glätten. So werden beispielsweise die *Thin-Plate-Splines*

$$\varphi(r) = r^2 \log r \tag{2.1.8}$$

mit Glättungsparameter $c > 0$ zu (*shifted-Thin-Plate-Splines*)

$$\varphi_c(r) = (c^2 + r^2) \log (c^2 + r^2). \tag{2.1.9}$$

bzw. mit $\varepsilon > 0$ zu

$$\varphi_\varepsilon(r) = (1 + r^2\varepsilon^2) \log (1 + r^2\varepsilon^2). \tag{2.1.10}$$

Alle bisher erwähnten radiale Basisfunktionen besitzen insbesondere keinen kompak-

ten Träger. Für Basisfunktionen mit kompaktem Träger kann zum Beispiel auf die Wendlandfunktionen zurückgegriffen werden. Sie besitzen folgende allgemeine Form

$$\varphi(r) = \begin{cases} p(r), & \text{falls } 0 \leq r \leq 1, \\ 0, & \text{falls } r > 1, \end{cases}$$

wobei $p(r)$ ein univariates Polynom ist. Abbildung 2.2 zeigt die Wendlandfunktionen $(1-r)_+^2$ und $(1-r)_+^8(32r^3+25r^2+8r+1)$, wobei $(1-r)_+$ die abgeschnittene Potenz mit $(1-r)$ für $0 \leq r \leq 1$ und 0 für $r > 1$ und $r < 0$ ist. Für ausführlichere Informationen siehe auch [24, Wendland].

Um Aussagen zur Existenz und Eindeutigkeit der Lösung von Interpolationsproblemen mit radialen Basisfunktionen machen zu können, führen wir noch den Begriff der vollständigen Monotonie ein. Eine Funktion $g \in C^\infty(0,\infty)$ heißt vollständig monoton (vm) genau dann, wenn sie für $\ell = 0, 1, \ldots,$ Bedingung (2.0.3) erfüllt. Als bedingt vollständig monoton der Ordnung k (bvm^k) bezeichnen wir eine stetige Funktion $g : \mathbb{R}^+ \mapsto \mathbb{R}$, falls $g \in C^\infty(0,\infty)$ und

$$(-1)^{(\ell)} g^{(\ell)} \geq 0 \text{ für alle } \ell \geq k. \tag{2.1.11}$$

Beispiele vollständig und bedingt vollständig monotoner Funktionen sind:

1. Mit $g(t) = e^{-\varepsilon^2 t}$ und $t := r^2$ gilt für den *Gaußkern*

$$\varphi(r) = e^{-\varepsilon^2 r^2}, \ \varepsilon > 0, \tag{2.1.12}$$

dass

$$g^{(\ell)}(t) = (-1)^\ell \alpha^\ell e^{-\alpha t}.$$

Damit ist g vollständig monoton für $\varepsilon > 0$, $t \in [0,\infty)$, und die Interpolations-

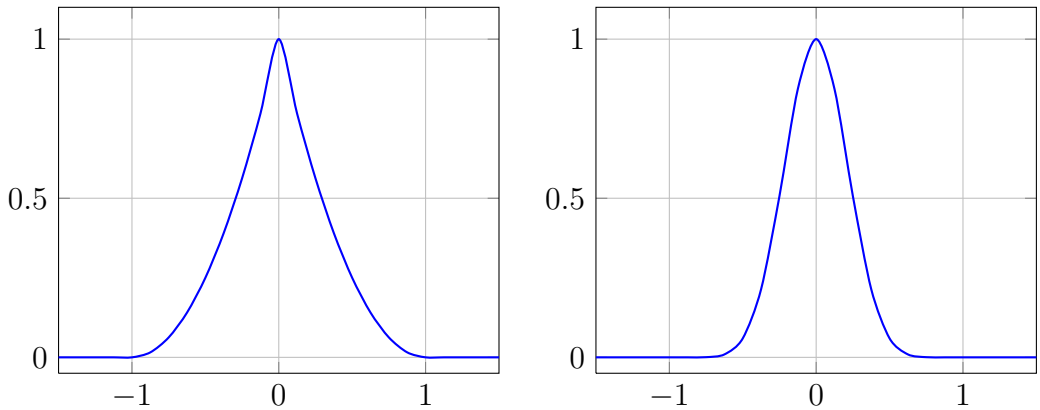

Abbildung 2.2: Wendlandfunktionen $(1-r)_+^2$ und $(1-r)_+^8(32r^3+25r^2+8r+1)$

matrix (2.0.2) bezüglich $\varphi(r)$ nichtsingulär.

2. Für die inverse Multiquadrics

$$\varphi(x) = (1 + (\varepsilon r)^2)^{-1/2} \tag{2.1.13}$$

folgt mit $g(t) = (1 + \varepsilon^2 t)^{-1/2}$ und dessen Ableitungen

$$g^{(\ell)}(t) = (-1)^\ell \varepsilon^{2\ell} (1 + \varepsilon^2 t)^{-\frac{2\ell+1}{2}} 2^{-\ell} \prod_{j=1}^{\ell} (2j - 1)\,,$$

ebenfalls die Nichtsingularität der Interpolationsmatrix (2.0.2).

3. Die Multiquadrics

$$\varphi(x) = -(1 + (\varepsilon r)^2)^{1/2} \tag{2.1.14}$$

ist wegen $g(t) = -(1+\varepsilon^2 t)^{1/2}$ und $g'(t) = -\frac{1}{2}\varepsilon^2(1+\varepsilon^2 t)^{-1/2}$ bedingt vollständig monoton der Ordnung 1.

4. Die Thin-Plate-Splines (2.1.8) sind bedingt vollständig monoton der Ordnung 2, da mit $g(t) = \frac{1}{2} t \log(t)$ gilt

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} g(t) = \frac{1}{2} t^{-1}.$$

Gleiches gilt auch für die shifted-Thin-Plate-Splines (2.1.10).

Für vollständig monotone Funktionen, die nicht konstant sind, ist die Interpolationsmatrix (2.0.2) nichtsingulär ist. Die folgende Proposition geht zurück auf Schönberg [18, 1938].

Proposition 2.1.2. *Sei $g : \mathbb{R}^+ \mapsto \mathbb{R}$ stetig und vollständig monoton, dann ist die Matrix*

$$\left\{\varphi\left(\|x_i - x_j\|_2\right)\right\}_{i,j=1}^{m}. \tag{2.1.15}$$

für alle endlichen Teilmengen $\{x_i\}_{i=1}^m \subset \mathbb{R}^d$ paarweise verschiedener Punkte positiv definit für $\varphi(r) = g(r^2)$, es sei denn g ist konstant. Die Matrix ist insbesondere nichtsingulär.

Der Gaußkern $\varphi(r) = e^{-\varepsilon^2 r^2}$, $\varepsilon > 0$, ist der Prototyp für alle anderen vollständig monotonen Funktionen. Dies bedeutet, das mit $g(t) = e^{-\alpha t}$, $\alpha > 0$, als Kernfunkti-

on in der Laplacetransformation alle vollständig monotonen Funktionen bzgl. eines Maßes μ dargestellt werden können. Dazu das folgende Theorem aus [3, Buhmann].

Theorem 2.1.3 (Bernstein-Widder Darstellung)**.** *Eine Funktion g ist vollständig monoton genau dann, wenn die Laplace-Transformation gilt, das heißt*

$$g(t) = \int_0^\infty e^{-\alpha t} \, \mathrm{d}\mu(\alpha), \ \alpha > 0,$$

für $t \geq 0$ und ein nicht fallendes Maß μ, das nach unten beschränkt ist, das heißt $\mathrm{d}\mu \geq 0$.

Der folgende Satz aus [15, Micchelli] ist eine unmittelbare Konsequenz von Proposition 2.1.2.

Satz 2.1.4. *Sei $g \in C^\infty[0, \infty)$ derart, dass g' vollständig monoton ist, jedoch nicht konstant. Angenommen $g(0) \geq 0$. Dann ist A nichtsingulär für $\varphi(r) = g(r^2)$.*

Mit Satz 2.1.4 ist insbesondere auch die Interpolationsmatrix der Multiquadrics (2.1.14) nichtsingulär.

Wir benötigen noch eine zu Proposition 2.1.2 analoge Aussage für bedingt vollständig monotone Funktionen der Ordnung k. Dazu brauchen wir noch eine weitere Eigenschaft, die bedingte positive Definitheit für Funktionen. Dabei heißt eine Funktion $F : \mathbb{R}^d \mapsto \mathbb{R}$ bedingt positiv definit der Ordnung k (bpd^k), $k \in \mathbb{Z}_+$, auf $\mathbb{R}^d$, falls für alle endlichen Teilmengen paarweise verschiedener Punkte $\{x_j\}_{j=1}^n \subset \mathbb{R}^d$, die quadratische Form

$$\lambda^T \{F(x_i - x_j)\}_{i,j=1}^n \lambda \tag{2.1.16}$$

positiv ist für alle $\lambda \in \mathbb{R}^m \backslash \{0\}$ mit

$$\sum_{i=1}^m \lambda_i p(x_i) = 0 \text{ für alle } p \in \mathbb{P}_d^{k-1}. \tag{2.1.17}$$

Die Bedingung an die quadratische Form (2.1.16) ist eine Bedingung an die Funktion F, nicht an eine Matrix. Mit folgendem Satz (aus [3]) erhalten wir die Verbindung zwischen bedingter vollständigen Monotonie der Ordnung k und bedingter positiver Definitheit der Ordnung k.

Satz 2.1.5. *Ist $\varphi(\sqrt{\cdot})$, $\varphi : \mathbb{R}^+ \mapsto \mathbb{R}$, bvm^k, so ist $\varphi(\|\cdot\|_2)$ bpd^k, $k \in \mathbb{Z}_+^0$.*

Beziehen wir nun die Bedingung (2.1.17) an die Koeffizienten in die Interpolationsmatrix (2.1.15) mit ein, dann erhalten wir die folgende Aussage analog zur Proposition 2.1.2 für bedingt positiv definite Funktionen der Ordnung k und damit auch für bedingt vollständig monotone Funktionen gleicher Ordnung.

Proposition 2.1.6. *Sei F bpd^k, $\{x_j\}_{j=1}^m \subset \mathbb{R}^d$ paarweise verschieden. Außerdem sei $p = 0$ das einzige Polynom $p \in \mathbb{P}_d^{k-1}$, das an allen x_j verschwindet. Dann ist*

$$\begin{pmatrix} \{F(x_i - x_j)\}_{i,j=1}^m & \{x_i^\alpha\}_{i=1,|\alpha|<k}^m \\ \left(\{x_i^\alpha\}_{i=1,|\alpha|<k}^m\right)^T & 0 \end{pmatrix} \tag{2.1.18}$$

nichtsingulär.

Die bedingte positive Definitheit können wir auch über die Fouriertransformation $\hat{\varphi}$ von φ herleiten. Dazu führen wir zuerst die Fouriertransformation für absolut integrierbare Funktionen $f \in L_1(\mathbb{R})$ ein und - da nicht alle radialen Basisfunktionen, die wir verwenden, in L_1 sind - erweitern dies dann auf verallgemeinerte Fouriertransformationen von Distributionen.

Die Fouriertransformation $\hat{f}$ zu einer absolut integrierbaren Funktion $f \in L_1(\mathbb{R}^d)$ ist definiert durch

$$\hat{f}(x) = \int_{\mathbb{R}^d} f(t) e^{-ix^T t} dt, \ x \in \mathbb{R}^d.$$

Das Integral ist wegen $f \in L_1(\mathbb{R}^d)$ endlich, da für absolut integrierbare Funktionen gilt

$$\int_{\mathbb{R}^d} |f(x)| dx < \infty.$$

Beispiel 1: Der Gaußkern (2.0.1) ist absolut integrierbar und besitzt folgende Fouriertransformation

$$\hat{\varphi}(\omega) = \frac{1}{2^{d/2} \|\omega\|_2^d} e^{-\|\omega\|_2^2/(4\varepsilon^2)}, \ \omega \in \mathbb{R}^d.$$

Nicht jede radiale Basisfunktion ist absolut integrierbar. Beispielsweise sind die Thin-Plate-Splines (2.1.8) nicht in L_1. Um auch von diesen eine Fouriertransformation zu erhalten, benötigen wir verallgemeinerte Funktionen (Distributionen) und einen Raum 'guter' Testfunktionen, den Schwartz-Raum. Damit definieren wir dann

temperierte Distributionen, für die verallgemeinerte Fouriertransformationen existieren.

Für ein Gebiet $\Omega \subset \mathbb{R}^d$ ist der Raum der Testfunktionen $\mathcal{D} := \mathcal{D}(\Omega)$ die Menge aller Funktionen $\tau : \Omega \mapsto \mathbb{R}$, die beliebig oft differenzierbar sind und einen beschränkten Träger $\text{supp}(\tau)$ in Ω besitzen. Der Träger von τ ist die im $\mathbb{R}^d$ abgeschlossene Hülle der Menge $\{x \in \mathbb{R}^d | \tau(x) \neq 0\}$.

Bevor wir zur Definition der Distributionen kommen, benötigen wir noch einen Konvergenzbegriff für die Testfunktionen. Eine Folge $\{\tau_n\}_{n\in\mathbb{N}}$ von Testfunktionen in $\mathcal{D}$ konvergiert gegen die Nullfunktion genau dann, wenn es eine kompakte Teilmenge von Ω gibt, die alle Träger $\text{supp}(\tau_n)$ enthält, und wenn alle Ableitungen der τ_n gleichmäßig gegen die Nullfunktion konvergieren, das heißt wenn für $(k_1, \ldots, k_d) \in \mathbb{N}_0^d$ gilt

$$\sup_{x\in\Omega} \left| \frac{\partial^{k_1+k_2+\ldots+k_d}}{\partial x_1^{k_1} \partial x_2^{k_2} \cdot \ldots \cdot x_d^{k_d}} \tau_n(x) \right| \overset{n\to\infty}{\longrightarrow} 0.$$

Eine Distribution T ist eine stetige lineare Abbildung $T : \mathcal{D} \mapsto \mathbb{R}$, das heißt für $a, b \in \mathbb{R}$, τ_1, τ_2 und $\tau = \lim_{n\to\infty} \tau_n$ in $\mathcal{D}$ gelten

$$T(a\tau_1 + b\tau_2) = aT(\tau_1) + bT(\tau_2) \text{ und } T(\tau) = \lim_{n\to\infty} T(\tau_n).$$

Die Menge aller Distributionen wird mit $\mathcal{D}'$ bezeichnet. Für die Anwendung von T auf τ schreiben wir auch

$$T(\tau) = \langle T, \tau \rangle = \langle T(t), \tau(t) \rangle = \int_{-\infty}^{\infty} T(t)\tau(t)dt.$$

Für temperierte Distributionen verwenden wir als Raum der Testfunktionen den Schwartzschen Raum $\mathcal{S}(\mathbb{R}^d)$.

Definition 2.1.7. *$\tau : \mathbb{R}^d \mapsto \mathbb{R}$ gehört zu $\mathcal{S}(\mathbb{R}^d)$, wenn τ unendlich oft differenzierbar ist und wenn für beliebige $k \in \mathbb{Z}^+$ und $\alpha \in (\mathbb{Z}^+)^d$ gilt*

$$\lim_{\|x\|\to\infty} \|x\|^k \frac{\partial^\alpha}{\partial x^\alpha} \tau(x) = 0.$$

Der Schwartzschen Raum wird auch als Raum der schnell fallenden Funktionen bezeichnet. Statt $\mathcal{S}(\mathbb{R}^d)$ schreiben wir auch kurz $\mathcal{S}$.

Jede stetige lineare Abbildung $T : \mathcal{S} \mapsto \mathbb{C}$ heißt temperierte Distribution. Die

Menge aller temperierten Distributionen wird mit $\mathcal{S}'$ bezeichnet. Die verallgemeinerte Fouriertransformierte $\hat{T}$ einer Distribution $T \in \mathcal{S}'$ ist für $\tau \in \mathcal{S}$ definiert durch

$$\left\langle \hat{T}, \tau \right\rangle = \langle T, \hat{\tau} \rangle .$$

Damit können wir das folgende Theorem aus [3, Buhmann] übernehmen und haben eine Verbindung zwischen der bedingt positiven Definitheit der Ordnung k von φ und deren Fouriertransformation $\hat{\varphi}$.

Theorem 2.1.8. *Die Funktion $\varphi(\|\cdot\|) : \mathbb{R}^d \mapsto \mathbb{R}$ ist bedingt positiv definit der Ordnung k, falls $\varphi(\|\cdot\|) : \mathbb{R}^d \mapsto \mathbb{R}$ eine verallgemeinerte Fourier-Transformation besitzt die folgende Bedingungen erfüllt*

$$\hat{\varphi}(r) > 0 \text{ für alle } r > 0 \quad \text{oder} \quad \hat{\varphi}(r) < 0 \text{ für alle } r > 0 \tag{A1}$$

$$\int_1^\infty \hat{\varphi}(r) r^{d-1} dr < \infty, \tag{A2}$$

$$\int_0^1 r^{2k} \hat{\varphi}(r) r^{d-1} dr < \infty. \tag{A3}$$

Ein wichtiger Vorteil dieser Charakterisierung der bedingten positiven Definitheit der Ordnung k zeigen wir am Beispiel $\varphi(r) = r$. Theorem 2.1.8 liefert dazu die bedingte positive Definitheit der Ordnung 1 für $\varphi(r) = -r$.

Proposition 2.1.9. *Die Funktion $\varphi(r) = -r$ ist bedingt positiv Definit der Ordnung 1.*

Beweis: Aus [13, Jones] erhalten wir als verallgemeinerte Fouriertransformation

$$\hat{\varphi}(\|\omega\|) = -\frac{\Gamma\left(\frac{1+d}{2}\right) 2^d \pi^{\frac{d-1}{2}}}{\|\omega\|^{d+1}}, \ \omega \in \mathbb{R}^d.$$

Dabei bezeichnet Γ die Gammafunktion mit der Definition

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \, \mathrm{d}t \quad \text{für } z \in \mathbb{C} : \mathcal{R}e(z) > 0. \tag{2.1.19}$$

Setzen wir $B_d := -\Gamma\left(\frac{1+d}{2}\right) 2^d \pi^{\frac{d-1}{2}}$ und $r := \|\omega\|$, dann ist

$$(1) \quad \hat{\varphi}(r) = \frac{B_d}{r^{d+1}} < 0 \text{ für alle } r > 0. \tag{2.1.20a}$$

$$(2) \quad \int_1^\infty \hat{\varphi}(r) r^{d-1} dr = B_d \int_1^\infty r^{-(d+1)} r^{d-1} dr = B_d \int_1^\infty r^{-2} dr < \infty. \tag{2.1.20b}$$

(3) $$\int_0^1 r^{2k}\hat{\varphi}(r)r^{d-1}dr = B_d \int_0^1 r^{2k} r^{-(d+1)} r^{d-1} dr$$
$$= B_d \int_0^1 r^{2k-2} dr < \infty \text{ für alle } k \geq 1. \tag{2.1.20c}$$

Damit folgt die Behauptung. □

Zusammengefasst haben wir Bedingungen für radiale Basisfunktionen beschrieben, die uns die Existenz der Koeffizient zur Lösung des Interpolationsproblems liefert. Für bpd Funktionen der Ordnung 0 und 1 haben wir die Nichtsingularität der Interpolationsmatrix (2.1.15), für Ordnung größer 1 erhalten wir mit (2.1.18) nichtsinguläre Matrizen zur Interpolation.

2.2 Interpolation

Für die Interpolation betrachten wir paarweise verschiedene Datenpunkte $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$ mit dazu gehörenden Datenwerte $\{f_i\}_{i=1}^n \subset \mathbb{R}$. Gesucht ist der Interpolant $s : \mathbb{R}^d \mapsto \mathbb{R}$ der Form

$$s(x) = \sum_{i=1}^{n} a_i \varphi(\|x - x_i\|_2),\ x \in \mathbb{R}^d,\ a_i \in \mathbb{R},\ i = 1, \ldots, n, \tag{2.2.21}$$

der die Interpolationsbedingung erfüllt, das heißt für den gilt

$$s(x_i) = f_i, \text{ für alle } i = 1, \ldots, n. \tag{2.2.22}$$

Die Interpolationspunkte benutzen wir als Verschiebung des Argumentes von s innerhalb der radialen Basisfunktion φ, wodurch n linear unabhängige Basisfunktionen entstehen. Aus diesem Grund bezeichnet man die Punkte $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$ auch als Centerpunkte. Da die Centerpunkte und Interpolationspunkte gleich sind, ist die Interpolationsmatrix M insbesondere symmetrisch mit

$$M = \left\{\varphi\left(\|x_i - x_j\|_2\right)\right\}_{i,j=1}^n. \tag{2.2.23}$$

Die Koeffizienten erhalten wir durch Lösen des Gleichungssystems $Ma = f$, mit $a = (a_1, \ldots, a_n)^T$ und $f = (f_1, \ldots, f_n)^T$.

Für vollständig monotone und bedingt vollständig monotone Funktionen der Ordnung 1 können wir die (2.2.23) direkt übernehmen. Im Falle bedingt vollständig monotone Funktionen größerer Ordnung erweitern wir die Interpolationsmatrix (2.2.23) um entsprechende Bedingungen an die Koeffizienten, um die Lösbarkeit des Gleichungssystems zu gewährleisten.

Die Existenz und Eindeutigkeit der Lösung eines gegebenen Interpolationsproblems für den Gaußkern (2.0.1), die inverse Multiquadrics (2.0.4) oder die Multiquadrics (2.0.5) erhalten wir automatisch.

Für bpd^k Funktionen mit $k \geq 2$ müssen wir nach Proposition 2.1.6 die Bedingungen (2.1.17) an die Koeffizienten mit einbeziehen. Folgendes Korollar fasst dies nochmal zusammen.

Korollar 2.2.1. *Ist $\varphi(\|\cdot\|)$ bpd^k, $\{x_i\}_{i=1}^m$ paarweise verschieden und*

$$\dim \mathbb{P}_d^{k-1}\big|_{\{x_i\}_{i=1}^m} = \binom{d+k-1}{d},$$

so ist das Interpolationsproblem

$$s(x_i) = \sum_{j=1}^{m} \lambda_j \varphi(\|x_i - x_j\|_2) + \sum_{|\alpha|<k} \hat{\lambda}_\alpha x_i^\alpha = f_i, \text{ für alle } i = 1, \ldots, m, \qquad (2.2.24)$$

mit

$$\sum_{j=1}^{m} \lambda_j x_j^\alpha = 0 \text{ für alle } |\alpha| < k,$$

eindeutig lösbar für alle $\{f_j\}_{j=1}^m \subset \mathbb{R}$.

Die Approximante unter Verwendung der Thin-Plate-Splines (2.1.8) wird damit zu

$$s(x_i) = \sum_{j=1}^{m} a_j \varphi(\|x_i - x_j\|_2) + \sum_{|\alpha|<2} \hat{a}_\alpha x_i^\alpha, \ i = 1, \ldots, m,$$

wobei $\alpha \in \mathbb{R}^d$, $|\alpha| := \alpha_1 + \ldots + \alpha_d$ und $x^\alpha := x_1^{\alpha_1} \cdot \ldots \cdot x_d^{\alpha_d}$. Zur Bestimmung der Koeffizienten müssen wir für $d = 2$ folgendes Gleichungssystem lösen

$$\begin{pmatrix} \varphi(\|x_1 - x_1\|_2) & \cdots & \varphi(\|x_1 - x_m\|_2) & 1 & x_{1,1} & x_{1,2} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \\ \varphi(\|x_m - x_1\|_2) & \cdots & \varphi(\|x_m - x_m\|_2) & 1 & x_{m,1} & x_{m,2} \\ 1 & \cdots & 1 & 0 & 0 & 0 \\ x_{1,1} & \cdots & x_{m,1} & 0 & 0 & 0 \\ x_{1,2} & \cdots & x_{m,2} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_m \\ \hat{a}_{00} \\ \hat{a}_{10} \\ \hat{a}_{01} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Dabei bezeichnet $x_{i,j}$ die j-te Komponente des Vektors x_i, $i = 1, \ldots, m$, $j = 1, 2$.

2.3 Quasi-Interpolation

Wir erweitern die Approximante (2.2.21) zunächst auf das Gitter $\mathbb{Z}^d$. Sei dazu

$$s(x) = \sum_{j \in \mathbb{Z}^d} f(j)\psi(x-j),\ x \in \mathbb{R}^d, \tag{2.3.25}$$

für $f : \mathbb{R}^d \mapsto \mathbb{R}$ und

$$\psi(x) = \sum_{k \in \mathbb{Z}^d} c_k \varphi(\|x-k\|),\ x \in \mathbb{R}^d, \tag{2.3.26}$$

mit von f unabhängigen Koeffizienten c_k und einer radialen Basisfunktion $\varphi : \mathbb{R}^+ \mapsto \mathbb{R}$. Für eine Konvergenz auf ganz $\mathbb{R}^d$ skalieren wir das Gitter $\mathbb{Z}^d$ mit einem entsprechend kleinen Parameter $h > 0$ auf $h\mathbb{Z}^d$

$$s_h(x) = \sum_{j \in \mathbb{Z}^d} f(jh)\psi(x/h-j),\ x \in \mathbb{R}^d. \tag{2.3.27}$$

Ausschlaggebend ist die Wahl von ψ. Verlangen wir, dass s_h an den Stellen hj, $j \in \mathbb{Z}^d$, exakt ist, dann wählen wir ψ mit Hilfe der Interpolation und erhalten eindeutig bestimmte, absolut summierbare Koeffizienten und damit auch eine eindeutige Approximante.

Um dies umzusetzen, wählen wir ψ als Lagrangefunktion

$$L(x) = \sum_{k \in \mathbb{Z}^d} c_k \varphi(\|x-k\|),\ x \in \mathbb{R}^d, \tag{2.3.28}$$

mit der Eigenschaft

$$L(j) = \begin{cases} 1, & \text{falls } j = 0, \\ 0, & \text{falls } j \in \mathbb{Z}^d \backslash \{0\}. \end{cases} \tag{2.3.29}$$

Damit interpoliert die Approximante (2.3.27) die Funktion f automatisch auf $h\mathbb{Z}^d$

$$s_h(\ell h) = \sum_{j \in \mathbb{Z}^d} f(jh)\psi(\ell h/h - j) = \sum_{j \in \mathbb{Z}^d} f(jh)L(\ell - j) = f(\ell h), \quad \forall\, \ell \in \mathbb{Z}^d,$$

bzw.

$$s_h \equiv f \quad \text{auf ganz } h\mathbb{Z}^d.$$

Allerdings benötigen wir nun Bedingungen an φ derart, dass für die Funktion L die Koeffizienten c_k, $k \in \mathbb{Z}^d$ existieren. Aus [3, Buhmann] übernehmen wir dazu folgende Proposition.

Proposition 2.3.1. *Sei φ derart, dass $|\varphi(r)| \leq C(1+r)^{-d-\varepsilon}$ für ein $C > 0$, konstant, und $\varepsilon > 0$, so dass insbesondere $\varphi(\|\cdot\|) \in L^1(\mathbb{R}^d)$. Angenommen für das Symbol*

$$\sigma(\theta) = \sum_{j \in \mathbb{Z}^d} \varphi(\|j\|) e^{-i\theta^T j}$$

gilt $\sigma(\theta) \neq 0 \; \forall \, \theta \in \mathbb{T}^d$. Dann gibt es eindeutige, absolut summierbare Koeffizienten $\{c_k\}_{k \in \mathbb{Z}^d}$, so dass die kardinale Funktion L (2.3.28) die Eigenschaft (2.3.29) erfüllt. Darüber hinaus genügt folgende Bedingung an die Fouriertransformation $\hat{\varphi}(\|\cdot\|)$

$$|\hat{\varphi}(r)| \leq C(1+r)^{-n-\varepsilon}$$

mit $\hat{\varphi}$ positiv, damit das Symbol σ überall positiv ist.

Hier tritt wieder das Problem der Einschränkung $\varphi(\|\cdot\|) \in L^1(\mathbb{R}^d)$ auf. Für temperierte Distributionen schreiben wir das Symbol und die Koeffizienten um

$$\sigma(\theta) = \sum_{j \in \mathbb{Z}^d} \hat{\varphi}(\|\theta - 2\pi j\|), \; \theta \in \mathbb{T}^d \backslash \{0\}, \tag{2.3.30}$$

sowie

$$c_k = \frac{1}{(2\pi)^d} \int\limits_{\mathbb{R}^d} \frac{e^{i\theta^T k}}{\sigma(\theta)} \, \mathrm{d}\, \theta, \; k \in \mathbb{Z}^d. \tag{2.3.31}$$

Wir halten fest, dass die Koeffizienten (2.3.31) existieren, falls das Symbol σ wohldefiniert und nullstellenfrei ist. Betrachten wir dieses nur noch in der Form (2.3.30), dann garantieren folgende Bedingungen an die verallgemeinerte Fouriertransformation $\hat{\varphi}(\|\cdot\|) : \mathbb{R}^d \backslash \{0\} \mapsto \mathbb{R}$ die Existenz der Koeffizienten:

$$\hat{\varphi}(r) > 0, \; \forall \, r > 0 \quad \text{oder} \quad \hat{\varphi}(r) < 0, \; \forall \, r > 0, \tag{B1}$$

$$\hat{\varphi}(r) = \mathcal{O}(r^{-d-\varepsilon}) \text{ für } r \to \infty \text{ und } \varepsilon > 0, \tag{B2}$$

$$\hat{\varphi}(r) = Cr^{-\mu} + o(r^{-\mu}) \text{ für } r \to 0_+ \text{ mit } C \neq 0 \text{ konstant}, \; \mu \geq 0. \tag{B3}$$

Dabei gibt (B1) die Nullstellenfreiheit des Symbols und damit die Existenz von $\frac{1}{\sigma}$. Es würde auch genügen zu fordern, dass die Fouriertransformation nicht-negativ ist und

keine 2π-periodischen Nullstellen besitzt. Die Bedingungen (B2) und $\hat{\varphi} \in C(\mathbb{R}_{>0})$ stellen zusätzlich sicher, dass die Summe in (2.3.30) wohldefiniert ist.

Damit das Reziproke des Symbols $\frac{1}{\sigma}$ glatter wird, können wir zusätzlich zu (B2) und (B3) weitere Bedingungen an die Ableitungen von $\hat{\varphi}(r)$ fordern. Für festes $\varepsilon > 0$ und $\ell = 0, \ldots, M$, $M \in \mathbb{Z}^+$, sei

$$\hat{\varphi}^{(\ell)}(r) = \mathcal{O}(r^{-d-\varepsilon}) \text{ für } r \to \infty, \text{ insbesondere } \hat{\varphi}^{(\ell)} \in C^M(0,\infty), \tag{B2a}$$

und

$$\hat{\varphi}^{(\ell)}(r) = \tilde{C} r^{-\mu-\ell} + \mathcal{O}(r^{-\mu-\ell+\varepsilon}) \text{ für } r \to 0_+, \ \tilde{C} \neq 0 \text{ und ein } \mu \geq 0. \tag{B3a}$$

Theorem 2.3.2 (aus [3]). *Angenommen für φ gelte* (B1), (B2a) *und* (B3a). *Sei $M > \lceil d + \mu \rceil$, dann erfüllen die Koeffizienten* (2.3.31) *für $k \neq 0$ und für $\varepsilon > 0$*

$$|c_k| = \begin{cases} C\|k\|^{-d-\mu} + O(\|k\|^{-d-\mu-\varepsilon}), & \textit{falls } \mu \notin 2\mathbb{Z}, \\ O(\|k\|^{-d-\mu-\varepsilon}), & \textit{sonst.} \end{cases}$$

Dabei bezeichne $\lceil \mu \rceil := \min\{m \in \mathbb{Z} | m \geq \mu\}$ die nächst höhere ganze Zahl.

Insgesamt erhalten wir für die Lagrangefunktion (2.3.28) bzw. die Quasi-Interpolante (2.3.25) - (2.3.27) Polynomreproduktion für alle Polynome in d Variablen mit Totalgrad kleiner als μ.

Theorem 2.3.3 (aus [3]). *Unter den Voraussetzungen von Theorem 2.3.2 gilt die Polynomreproduktionseigenschaft für die Interpolation*

$$\sum_{j \in \mathbb{Z}^d} p(j) L(x - j) = p(x), \quad x \in \mathbb{R}^d, \tag{2.3.32}$$

für alle Polynome p in d Variablen mit Totalgrad kleiner als μ. Weiterhin gibt es ein Polynom vom Grad mindestens μ, das nicht durch obige Summe reproduziert wird, auch wenn die unendliche Summe in (2.3.32) *für dieses Polynom wohldefiniert ist.*

Wir übernehmen den Beweis aus [3, Buhmann], da Einzelheiten aus diesem später noch zur Anwendung kommen. Vorher benötigen wir noch die *Poissonsche Summationsformel* aus [20, Stein und Weiss].

Lemma 2.3.4. *Sei $s \in L^1(\mathbb{R}^n)$ derart, dass die Fouriertransformation $\hat{s}$ ebenfalls*

absolut integrierbar ist. Dann gilt

$$\sum_{j\in\mathbb{Z}^n} s(j)e^{-i\theta\cdot j} = \sum_{\ell\in\mathbb{Z}^n} \hat{s}(\theta + 2\pi\ell), \quad \theta \in \mathbb{T}^n, \tag{2.3.33}$$

mit Konvergenz in $L^1(\mathbb{T}^n)$*, mit* $\mathbb{T} = [0, 2\pi)$*. Falls* s *folgende Abschätzungen für ein* $\varepsilon > 0$ *erfüllt*

$$|s(x)| = O((1 + \|x\|)^{-n-\varepsilon}) \text{ und } |\hat{s}(x)| = O((1 + \|x\|)^{-n-\varepsilon}),$$

dann sind beide Summen absolut konvergent und die Grenzfunktionen stetig. Damit gilt obige Identität punktweise.

Beweis (von Theorem 2.3.3)**:** Sei p ein Polynom in n Unbekannten vom Grad kleiner als μ, dann ist mit der Poissonschen Summationsformel (2.3.33) die linke Seite von (2.3.32) die Summe der partiellen Ableitungen einer Exponentialfunktion *mal* die Fouriertransformation der Lagrangefunktion

$$\sum_{j\in\mathbb{Z}^n} p(iD)\{e^{-ix\cdot t}\hat{L}(t)\}_{t=2\pi j},$$

mit

$$D = D_t = \left(\frac{\partial}{\partial t_1}, \frac{\partial}{\partial t_2}, \ldots, \frac{\partial}{\partial t_n}\right)^T.$$

Das multivariate Polynom p, angewendet auf iD, führt zu einer Linearkombination von partiellen Ableitungen. Der Index $t = 2\pi j$ bedeutet, dass solche partiellen Ableitungen zuerst berechnet werden und erst danach ausgewertet werden. Wir separieren den Term mit Index $j = 0$ vom Rest der Summe. Damit erhalten wir

$$\sum_{j\in\mathbb{Z}^n\setminus\{0\}} p(iD)\{e^{-ix\cdot t}\hat{L}(t)\}_{t=2\pi j} + \hat{L}(0)p(x). \tag{2.3.34}$$

Wir behaupten nun, dass obiger Ausdruck $p(x)$ ist, wie angenommen. Dies gilt, da [3]

$$L(x) = \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} e^{ix\cdot y}\frac{\hat{\varphi}(\|y\|)}{\sum_{\ell\in\mathbb{Z}^n}\hat{\varphi}(\|y - 2\pi\ell\|)}\,\mathrm{d}y$$

folgendes impliziert

$$\hat{L}(0) = 1,$$

sowie

$$\left(D^\alpha \hat{L}\right)(2\pi j) = 0 \; \forall \; j \in \mathbb{Z}^n \backslash \{0\}, \; |\alpha| < \mu,$$

und

$$\left(D^\alpha \hat{L}\right)(0) = 0 \; \forall \; \alpha, \; 0 < |\alpha| < \mu,$$

wobei $\alpha \in (\mathbb{Z}^+)^n$. All diese Eigenschaften der Fouriertransformation der Lagrangefunktion L kommen von der Singularität der Fouriertransformation von $\hat{\varphi}$ am Ursprung, da diese Singularität, die in dem Nenner von $\hat{L}$ auftritt, die Nullstellen von $\hat{L}$ an allen Stellen $2\pi k$, $k \in \mathbb{Z}\backslash\{0\}$, induziert. Wir können zum Beispiel $\hat{L}$ in einer Umgebung von Null oder $2\pi k$, $k \in \mathbb{Z}\backslash\{0\}$, entwickeln, um obige Nullbedingungen direkt aus der Bedingung (B3) zu erhalten. Damit ist der erste Teil des Theorems bewiesen.

Es ist klar, dass die Summe (2.3.32) nicht wohldefiniert ist, falls $\mu \notin 2\mathbb{Z}^+$. Damit wäre

$$L(x) \sim \|x\|^{-n-\mu}$$

und p vom Totalgrad $\geq \mu$. Allerdings ist (2.3.32) wohldefiniert, falls $\mu \in 2\mathbb{Z}^+$ ist und L dadurch schneller abklingt. Sei $p(x) = \|x\|^\mu$, $\mu \in 2\mathbb{Z}^+$. Dann ist p ein Polynom vom Totalgrad μ. In diesem Fall enthält (2.3.34) Terme wie

$$e^{-i2\pi j \cdot x} \|iD\|^\mu \hat{L}(2\pi j), \quad j \in \mathbb{Z}^n, \tag{2.3.35}$$

wobei der rechte Faktor eine Konstante ungleich Null ist. Damit kann (2.3.32) nicht gelten und es gibt keine Auslöschung in der Summe in Ausdruck (2.3.35) wegen der linearen Unabhängigkeit der Exponentialfunktionen für verschiedene Argumente. □

Wir verallgemeinern diese Aussagen nun und fordern für ψ nicht mehr die Eigenschaft (2.3.29), sondern dass die Koeffizientenfolge $\{c_k\}_{k\in\mathbb{Z}^d}$ der Linearkombination (2.3.26) endlichen Träger besitzt. Diese Wahl macht für die Approximationsordnung und die Polynomreproduktion keinen Unterschied zur Lagrangefunktion [3]. Analog zur Lagrangefunktion (2.3.28) betrachten wir wieder die Fouriertransformation

$$\hat{\psi}(t) = \hat{\varphi}(\|t\|) \sum_{k\in\mathbb{Z}^d} c_k e^{-it\cdot k}, \quad t \in \mathbb{Z}^d, \tag{2.3.36}$$

von ψ aus (2.3.26) und fordern für $\mu \in \mathbb{Z}^+$ die Strang und Fix Bedingungen für

$\mu \in \mathbb{Z}^+$ (siehe auch [3, 14])

$$\hat{\psi}(0) = 1, \tag{C1}$$

$$\left(D^\alpha \hat{\psi}\right)(0) = 0 \;\forall\; \alpha,\; 0 < |\alpha| < \mu, \tag{C2}$$

$$\left(D^\alpha \hat{\psi}\right)(2\pi j) = 0 \;\forall\; j \in \mathbb{Z}^d \backslash \{0\},\; 0 \le |\alpha| < \mu. \tag{C3}$$

Unter diesen Bedingungen erhalten wir Polynomreproduktion vom Grad μ.

Theorem 2.3.5. *Angenommen ψ in* (2.3.26) *besitzt eine verallgemeinerte Fouriertransformation $\hat{\psi}$, die* (C1) - (C3) *erfüllt. Dann gilt*

$$\sum_{j \in \mathbb{Z}^d} (j)^\alpha \psi(x - j) = x^\alpha \quad \textit{für alle } \alpha \in (\mathbb{Z}^+)^d : |\alpha| \le \mu,$$

das heißt Polynomreproduktion vom Grad μ.

Beweis: Mit der Poissonschen Summationsformel (2.3.33) und $D = (\partial/\partial t_1, \ldots, \partial/\partial t_d)^T$ folgt für alle $\alpha \in \mathbb{Z}^d : |\alpha| \le \mu$

$$\begin{aligned}
\sum_{j \in \mathbb{Z}^d} (j)^\alpha \psi(x - j) &= \sum_{j \in \mathbb{Z}^d} (iD)^\alpha \left(e^{-ixt} \hat{\psi}(t)\right)_{t=2\pi j} \\
&= \sum_{j \in \mathbb{Z}^d \backslash \{0\}} (iD)^\alpha \left(e^{-ixt} \hat{\psi}(t)\right)_{t=2\pi j} + \left(i^\alpha D^\alpha e^{-ixt} \hat{\psi}(t)\right)_{t=0} \\
&= \sum_{j \in \mathbb{Z}^d \backslash \{0\}} (iD)^\alpha \left(e^{-ixt} \hat{\psi}(t)\right)_{t=2\pi j} + \left(i^\alpha D^\alpha e^{-ixt}\right)_{t=0} \hat{\psi}(0) \\
&\quad + \left(i^\alpha e^{-ixt} D^\alpha \hat{\psi}(t)\right)_{t=0} \\
&= \sum_{j \in \mathbb{Z}^d \backslash \{0\}} (iD)^\alpha \left(e^{-ixt} \hat{\psi}(t)\right)_{t=2\pi j} + \left(i^\alpha D^\alpha e^{-ixt}\right)_{t=0} \hat{\psi}(0) \\
&\quad + i^\alpha \underbrace{\left(D^\alpha \hat{\psi}(t)\right)_{t=0}}_{=0,\ (C2)} \\
&= \sum_{j \in \mathbb{Z}^d \backslash \{0\}} (iD)^\alpha \left(e^{-ixt} \hat{\psi}(t)\right)_{t=2\pi j} + x^\alpha \underbrace{\hat{\psi}(0)}_{=1,\ (C1)} \\
&= \sum_{j \in \mathbb{Z}^d \backslash \{0\}} \left((-i)^\alpha i^\alpha x^\alpha e^{-ix2\pi j} \underbrace{\hat{\psi}(2\pi j)}_{=0,\ (C3)} + i^\alpha e^{-ix2\pi j} \underbrace{D^\alpha \hat{\psi}(2\pi j)}_{=0,\ (C3)}\right) + x^\alpha \\
&= x^\alpha.
\end{aligned}$$

Dabei haben wir folgende Eigenschaft der Fouriertransformation benutzt:

$$\widehat{x_k^n \varphi(x)} = i^k \frac{\partial^n}{\partial \omega_k^n} \hat{\varphi}(\omega).$$

□

Nun müssen wir die Bedingungen (C1) - (C3) an ψ noch auf Bedingungen an die radiale Basisfunktion φ überführen.

Theorem 2.3.6. *Die Funktion φ erfülle* (B1), (B2a) *und* (B3a) *für $\mu \in 2\mathbb{Z}^+$. Dann gibt es ein multivariates trigonometrisches Polynom g, so dass die Funktion ψ in* (2.3.26) *exakt für Polynome vom Grad kleiner als μ ist und folgende asymptotische Eigenschaft besitzt*

$$|\psi(x)| = O(\|x\|^{-d-\mu}) \quad \textit{für große } \|x\|.$$

Beispiel 2 (Quasi-Interpolation)**:** Wir betrachten $\varphi(r) = -r$ und $\{x_k\}_{k=1}^{\ell} \subset \mathbb{R}^d$ für ein $\ell \in \mathbb{N}$. Dann ist

$$\psi(x) = \sum_{k=1}^{\ell} c_k \varphi(\|x - x_k\|), \quad x \in \mathbb{R}^d, \tag{2.3.37}$$

und

$$\hat{\psi}(x) = \sum_{k=1}^{\ell} c_k e^{-ix\cdot k} \hat{\varphi}(\|x\|). \tag{2.3.38}$$

Die Fouriertransformation für $d = 3$ ist wegen (2.1.20a)

$$\hat{\varphi}(\|x\|) = B_3 \|x\|^{-4}, \text{ mit } B_3 < 0. \tag{2.3.39}$$

Aus (C1) folgt nun

$$\sum_{k=1}^{\ell} c_k = 0, \tag{2.3.40}$$

und mit (C2) erhalten wir

$$\sum_{k=1}^{\ell} c_k k^{\alpha} = 0 \; \forall \; 0 < |\alpha| < 4 \tag{2.3.41}$$

sowie mit (C3)

$$\sum_{k=1}^{\ell} c_k k^{\alpha} e^{-2\pi i k \cdot j} = 0, \ \forall\, j \in \mathbb{Z}^d \backslash \{0\}, \ 0 \leq |\alpha| < 4. \tag{2.3.42}$$

Die Bestimmung der Koeffizienten c_k behandeln wir ausführlich in Kapitel 5.1.

Kapitel 3

Die Dagumfunktionen

Die Dagumfunktionen sind definiert durch

$$\varphi_{\beta,\gamma}(r) = 1 - \left(\frac{r^\beta}{1+r^\beta}\right)^\gamma,\ r \geq 0, \tag{3.0.1}$$

für gegebene Parameter $\beta, \gamma > 0$. Eingeführt wurden diese als radial symmetrische Korrelationsfunktionen in [16, Porcu et al.]. Insbesondere sind sie nach Definition 2.1.1 radial symmetrische Basisfunktionen mit $\Phi(x) = \varphi_{\beta,\gamma}(\|x\|)$.

Die Dagumfunktionen werden im Anschluss in die numerischen Untersuchungen als Basisfunktion der Interpolation miteinbezogen. Dazu werden noch notwendige und hinreichende Bedingungen an β und γ für die vollständige Monotonie und die damit verbundene Nichtsingularität der Interpolationsmatrix benötigt.

3.1 Eigenschaften

Zunächst zum aktuellen Stand. Sei dazu für $1 < \beta < 2$ und

$$\psi_\beta(t) = 1 - \frac{2}{\beta}\exp\left(t\cos\frac{\pi}{\beta}\right)\cos\left(t\sin\pi\beta\right) \tag{3.1.2}$$

$$- \frac{\sin(\beta\pi)}{\pi}\int_0^\infty \frac{e^{-ts}s^{\beta-1}}{1+2s^\beta\cos(\beta\pi)+s^{2\beta}}\,\mathrm{d}s, \quad t \geq 0. \tag{3.1.3}$$

Damit ist die Funktion

$$\beta \to \Psi(\beta) = \max_{t\geq 0}\psi_\beta(t)$$

stetig auf dem Intervall $[1, 2]$. Die Funktion

$$\ell(\beta) = \beta\left(\Psi(\beta) - 1\right), \quad 1 \leq \beta \leq 2,$$

ist ebenfalls stetig und streng monoton steigend. Wegen $\ell(1) = 0$, $\ell(2) = 2$ setzen wir noch β_* als die eindeutige Zahl im Intervall $(1, 2)$ mit $\ell(\beta_*) = 1$.

Theorem 3.1.1. *Sei $\beta, \gamma > 0$ und $\mathcal{C}$ die Menge der vollständig monotonen Funktionen. Dann gilt*

$$\varphi_{\beta,\gamma}(r) \in \mathcal{C} \iff \frac{r^{\beta\gamma-1}}{(1+r^\beta)^{\gamma+1}} \in \mathcal{C}.$$

Beweis: Die Behauptung folgt sofort aus der Tatsache, dass $\varphi_{\beta,\gamma}(r)$ eine positive $\mathcal{C}^\infty$ Funktion ist und

$$\frac{\mathrm{d}}{\mathrm{d}r}\varphi_{\beta,\gamma}(r) = -\gamma\beta\frac{r^{\beta\gamma-1}}{(1+r^\beta)^{\gamma+1}},\ \gamma, \beta > 0. \tag{3.1.4}$$

□

Theorem 3.1.2 (aus [4])**.** *Angenommen*

$$\frac{r^{\beta\gamma-1}}{(1+r^\beta)^{\gamma+1}} \in \mathcal{C},$$

dann ist

$$\beta\gamma \leq 1 \text{ und } \beta \leq 2.$$

Für die andere Richtung gilt

(i) Wenn $\beta\gamma \leq 1$ und $\beta \leq 1$ dann $\varphi_{\beta,\gamma} \in \mathcal{C}$.

(ii) Für $\beta\gamma = 1$ ist

$$\varphi_{\beta,1/\beta} \in \mathcal{C} \iff \beta \leq 1.$$

(iii) Falls $\beta\gamma < 1$, $1 < \beta < \beta_*$ und zusätzlich

$$0 < \gamma \leq \frac{1-\ell(\beta)}{\beta+\ell(\beta)},$$

dann ist

$$\varphi_{\beta,\gamma}(r) \in \mathcal{C}.$$

Insbesondere folgt aus Theorem 3.1.2 die positive Definitheit der Interpolations-

matrix für die Bedingungen $(i) - (iii)$.

Theorem 3.1.2 schließt den Fall $b = 2$ als notwendige Bedingung nicht mit ein, da $\beta_* \in (1, 2)$. Mit dem folgenden neuen Theorem erhalten wir noch eine Aussage für die Ableitung (3.1.4) mit erweitertem Zähler und $\beta = 2$, $\gamma \leq 1/2$.

Theorem 3.1.3. *Sei $\beta = 2$ und $\gamma \leq 1/2$, dann ist die Interpolationsmatrix von*

$$\varphi_{2,\gamma}(r) = \frac{\left(\sqrt{\delta^2 + r^2}\right)^{2\gamma-1}}{(1+r^2)^{\gamma+1}}$$

mit $r = \|x - x_i\|_2$ positiv definit für alle paarweise verschiedenen x_i, $i = 1, \ldots, n$, und kleines $\delta > 0$.

Beweis: Sei $M_{2,\gamma}$ die Interpolationsmatrix von $\varphi_{2,\gamma}$ zu den Punkten $x_i \in \mathbb{R}^d$, $i = 1, \ldots, n$. Dann gilt

$$M_{2,\gamma} = A_{2,\gamma} \circ B_{2,\gamma}$$

mit $A_{2,\gamma}$ als Interpolationsmatrix von $\tilde{\varphi}_{2,\gamma}(r) = \left(\sqrt{\delta^2 + r^2}\right)^{2\gamma-1}$ und $B_{2,\gamma}$ als Interpolationsmatrix von $\bar{\varphi}_{2,\gamma}(r) = (1+r^2)^{-(\gamma+1)}$ zu den Punkten $x_i \in \mathbb{R}^d$, $i = 1, \ldots, n$. Für das Hadamard-Produkt $\circ$ gilt insbesondere, dass, wenn $A_{2,\gamma}$ und $B_{2,\gamma}$ positiv definit sind, dann ist auch $M_{2,\gamma}$ positiv definit (aus [11, Theorem 5.2.1]).

Zunächst zur positiven Definitheit von $B_{2,\gamma}$: Für $\gamma > 0$ folgt mit $g_{2,\gamma}(r) := \bar{\varphi}_{2,\gamma}(\sqrt{r})$, dass

$$(-1)^\ell g_{2,\gamma}^{(\ell)}(r) = (1+r)^{-(\gamma+1+\ell)} \prod_{j=0}^{\ell-1} (\gamma + 1 + j)\,.$$

Damit ist $\bar{\varphi}_{2,\gamma}$ nach (2.0.3) vollständig monoton und die Matrix $B_{2,\gamma}$ insbesondere positiv definit für alle $\gamma > 0$.

Zur positiven Definitheit von $A_{2,\gamma}$: Für $\gamma < 1/2$ ist $2\gamma - 1 < 0$ und die positive Definitheit folgt analog zu $B_{2,\gamma}$. Damit folgt die Behauptung für $\gamma < 1/2$.

Für den Fall $\gamma = 1/2$ erhalten wir $M_{2,1/2} = B_{2,1/2}$ und es folgt ebenfalls die Behauptung. □

Abbildung 3.1 zeigt einige ausgewählte Dagumfunktionen.

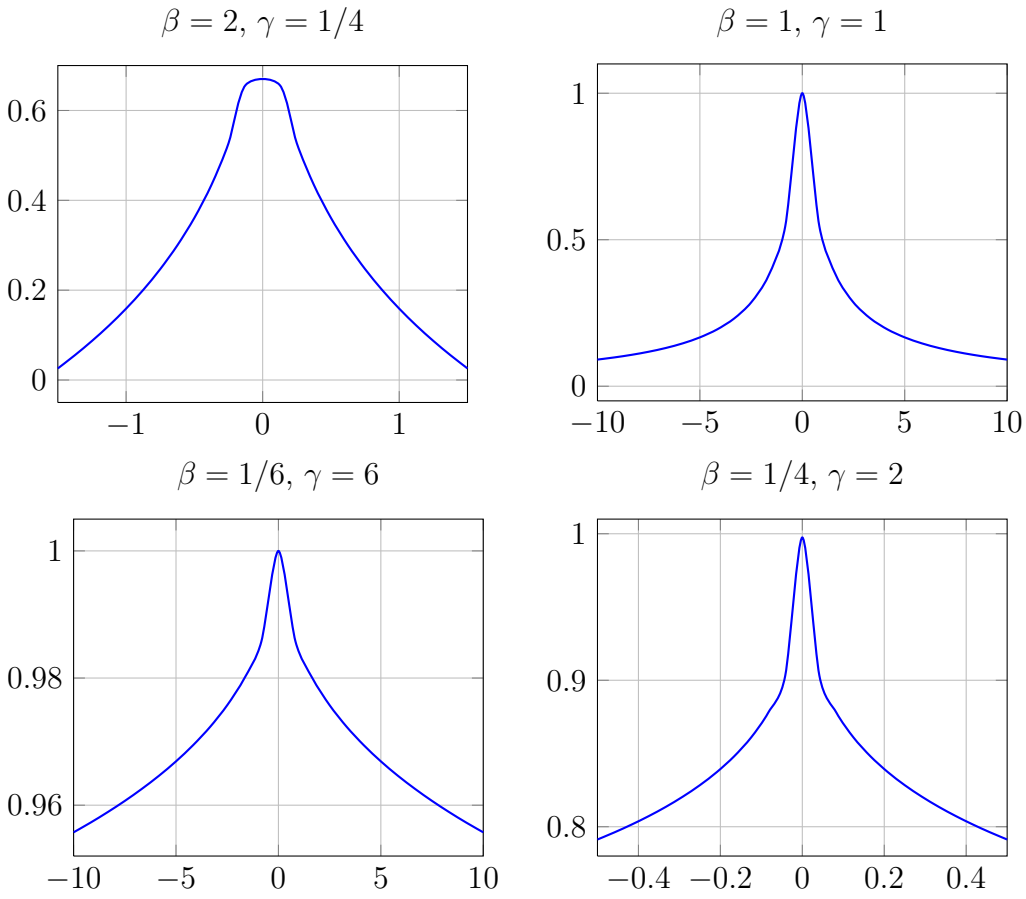

Abbildung 3.1: Dagumfunktionen für ausgewählte β und γ.

Kapitel 4

Interpolation mit radialen Basisfunktionen

In Kapitel 2.2 haben wir die Interpolation mit radialen Basisfunktionen eingeführt. Insbesondere haben wir zwischen positiv definiten und bedingt positiv definiten Basisfunktionen 2.1.6 unterschieden. Für bpd^k wird der Interpolant um ein Polynom vom Grad $k-1$ erweitert zu

$$s(x) = \sum_{j=1}^{m} a_j \varphi(\|x_i - x_j\|_2) + \sum_{|\alpha|<k} \hat{a}_\alpha x_i^\alpha, \tag{4.0.1}$$

wobei $\alpha \in \mathbb{R}^d$. In Kapitel 2.2 haben wir Bedingungen an die Interpolationsmatrix (2.2.23) betrachtet, um singuläre Matrizen zu vermeiden. Daneben können instabile Interpolationsmatrizen auch bei sehr kleinen ε-Werten auftreten. Lassen wir den Glättungsparameter ε in den entsprechenden radialen Basisfunktionen (2.1.6) gegen 0 gehen, dann streben die Funktionswerte der radialen Basisfunktionen durch die numerische Auslöschung gegen einem konstanten Wert – im Fall des Gaußkerns gilt beispielsweise $e^{-\varepsilon^2 r^2} \to 1$ für $\varepsilon \to 0$ und r fest. Gleiches gilt für radiale Basisfunktionen mit Glättungsparameter $c > 0$ (2.1.7) und $c \to \infty$.

Eine numerisch stabile Berechnung der Koeffizienten des Interpolationsproblems gelingt mit speziellen Algorithmen, wie dem Contour-Padé-Algorithmus [7] und dem RBF-QR Algorithmus [5, 6]. Der Contour-Padé-Algorithmus ist allerdings aufwendig, langsam und nur auf verhältnismäßig wenige Interpolationspunkte anwendbar. So kann – je nach Verteilung der Interpolationspunkte – ab $n > 80$ die Interpolationsmatrix durch numerische Auslöschung ebenfalls singulär werden. Der RBF-QR Algorithmus fängt auch diese numerische Instabilität ab und kann auf eine größere Anzahl von Punkten ($n > 100$) angewendet werden. Zudem ist er relativ einfach zu implementieren (weniger als 100 Zeilen Quellcode) und vergleichsweise schnell.

In dieser Arbeit werden wir einen neuen Algorithmus entwickeln, der die Einschränkungen an n und ε für die *inverse Multiquadrics*, deren Verallgemeinerung und die Multiquadrics beseitigt. Dieser neue Algorithmus, der RBF-QR-GQ-Algorithmus, verwendet die Tatsache, dass jede vollständig monotone Funktion, über die Bernstein-Widder-Darstellung, als Laplacetransformation mit dem Gaußkern als Kernfunktion dargestellt werden kann. Die Idee in dieser Arbeit ist es, dass, wenn die Bernstein-Widder-Darstellung folgende Form hat

$$\int_0^\infty u^\beta e^{-u} e^{-u\varepsilon^2 r^2} \, \mathrm{d}u, \ \beta \in \mathbb{R},$$

dann können wir das Integral mittels Gauß-Quadratur approximieren. Die erhaltene Summe können wir, analog zur Vorgehensweise des RBF-QR-Algorithmus, umrechnen und erhalten insgesamt eine stabile Interpolationsmethode. Für die Multiquadrics (2.0.5) existiert zwar keine Bernstein-Widder-Darstellung, dafür gibt es eine Integraldarstellung, die ebenfalls eine Umrechnung erlaubt.

Im Folgenden beschränken wir uns auf die Betrachtung des $\mathbb{R}^2$, da wir den RBF-QR-GQ-Algorithmus auf den RBF-QR-2D-Algorithmus aus [5] aufbauen, und zum Anderen $\mathbb{R}^2$ eine gute Wahl für Testfunktionen und deren anschließende Visualisierung zur Fehlerbetrachtung der Algorithmen in Kapitel 6 ist. Dazu betrachten wir ohne Einschränkung den Einheitskreis $B(0,1)_2 \subset \mathbb{R}^2$ und das Quadrat $[0,1]^2$.

$$
\begin{aligned}
f_1(x,y) &= \frac{25}{25 + (x-0.2)^2 + 2y^2} \\
f_2(x,y) &= \frac{\exp\left((x-0.1)^2 + 0.5y^2\right)}{\exp(1.21)} \\
f_3(x,y) &= \frac{\arctan\left(2(x+3y-1)\right)}{\arctan\left(2(\sqrt{10}+1)\right)} \\
f_4(x,y) &= \sin(2\pi(x-y)) \\
f_5(x,y) &= \frac{3}{4}\exp^{-\frac{1}{4}((9x-2)^2+(9y-2)^2)} + \frac{3}{4}\exp^{-\frac{1}{49}(9x+1)^2-\frac{1}{10}(9y+1)} \\
&\quad + \frac{1}{2}\exp^{-\frac{1}{4}((9x-7)^2+(9y-3)^2))} - \frac{1}{5}\exp^{-(9x-4)^2-(9y-7)^2} \\
f_6(x,y) &= (1-2|x|)_+ + (1-2|y|)_+.
\end{aligned}
$$

Tabelle 4.1: Testfunktionen auf $B(0,1)_2$ und $[0,1]^2$.

In Tabelle 4.1 sind einige Testfunktionen aufgeführt. Zur Anschauung und Motivation sind in Abbildung 4.2 die relativen Fehler des Gaußkerns und der inversen Multiquadrics für den RBF-Direkt-Algorithmus zu 64 Interpolationspunkten für $\varepsilon \in (0, 1)$ bzgl. der Testfunktionen $f_1, \ldots, f_6$ und 10.000 Auswertungspunkte in $B(0, 1)_2$ abgebildet. Für $\varepsilon = 10^{-1}$ erhalten wir zu den Testfunktionen f_1, f_2 und f_5 noch relativ gute Ergebnisse, allerdings wird die Interpolationsmatrix bei kleinerem Glättungsparameter numerisch instabil, die Konditionszahl wird entsprechend groß (Abbildung 4.1).

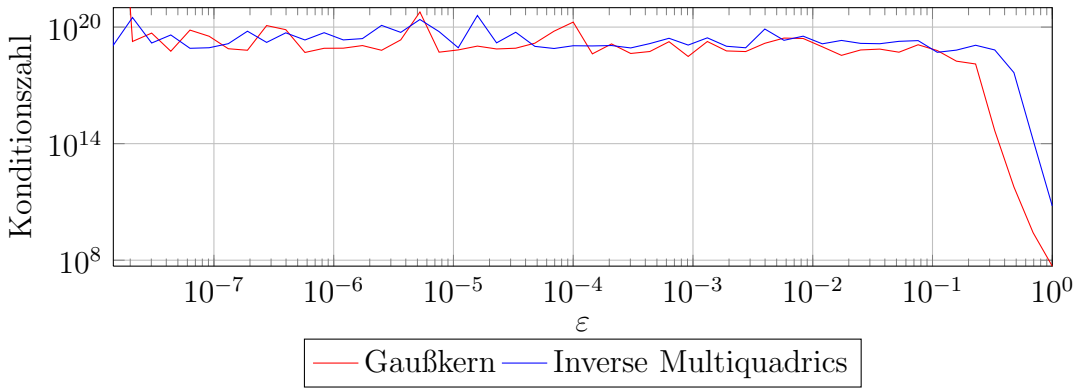

Abbildung 4.1: Konditionszahl bzgl. der GA- und IMQ-RBF für 64 Interpolationspunkte und dem RBF-Direkt-Algorithmus.

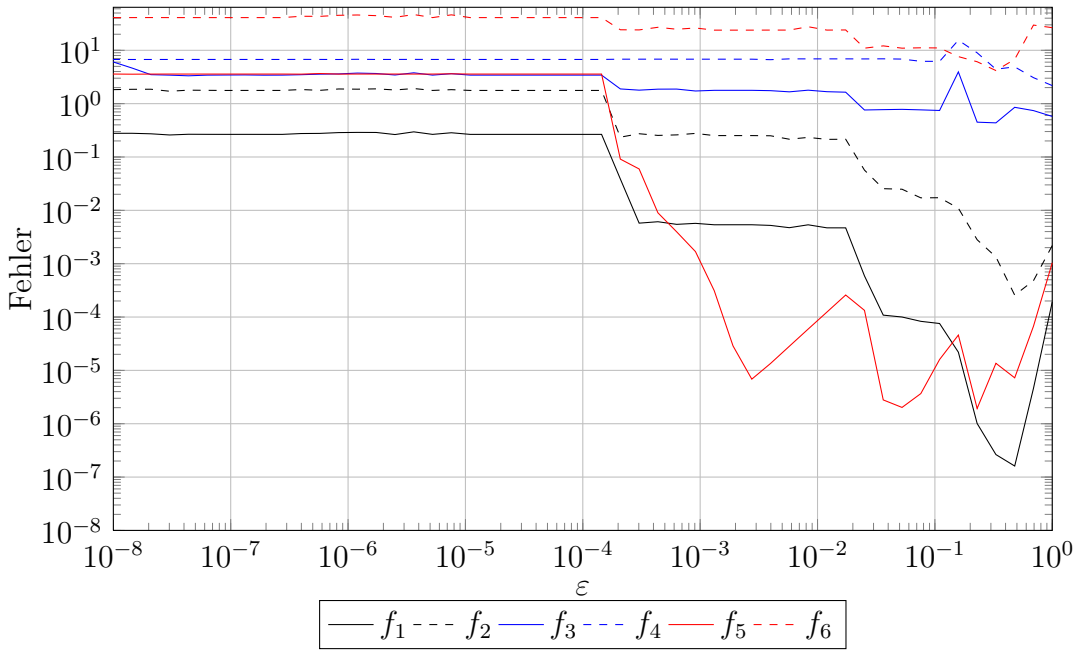

Abbildung 4.2: Relativer Interpolationsfehler des RBF-Direkt-Algorithmus mit Gaußkern für 64 Interpolationspunkte in $B(0, 1)_2$ und 10.000 Auswertungspunkte.

4.1 Der RBF-QR-Algorithmus

Wir beschreiben zunächst den RBF-QR-Algorithmus nach Fornberg [5] und erweitern diesen im Anschluss zum RBF-QR-GQ-Algorithmus.

Zunächst halten wir fest, dass die Eigenwerte der Interpolationsmatrix mit beliebiger Verteilung nicht-periodischer Interpolationspunkte in $\mathbb{R}^2$ von ε, abhängen. Insbesondere können die Eigenwerte in Mengen $O_k := \{c\varepsilon^k | c \in \mathbb{R}^+\}$ eingeteilt werden. Dabei steigt die Mächtigkeit der Mengen mit zunehmendem Exponenten von ε an (siehe Tabelle 4.2, [8]). Betrachtet man hingegen variable ε_ℓ, so hängen diese Mengen von dem jeweils kleinsten $\varepsilon_{\tilde{\ell}}$ ab [8].

	$\lvert\{O_0\}\rvert$	$\lvert\{O_2\}\rvert$	$\lvert\{O_4\}\rvert$	$\lvert\{O_6\}\rvert$	...
GA, IMQ, MQ	1	2	3	4	...

Tabelle 4.2: Anzahl der Eigenwerte entsprechender Ordnung für nicht-periodische Gebiete in $\mathbb{R}^2$.

Der RBF-QR-Algorithmus teilt nun die Interpolationsmatrix in ein Produkt von Matrizen auf, die explizit in Auswertungspunkte, Interpolationspunkte und ε-Terme getrennt sind. Durch anschließende QR-Faktorisierung werden die zur Instäbiliät beitragenden Eigenwerte numerisch eliminiert. Während für periodische Gebiete aus $\mathbb{R}^d$, $d = 1, 2, 3$, die Anwendung für mehrere RBF's vorliegt [8], gibt es eine solche im nicht-periodischen Fall bisher nur für den Gaußkern und Besselfunktionen.

RBF-QR für $\varphi(r) = e^{-\varepsilon^2 r^2}$ (aus [5])

Wir zentrieren den Gaußkern $\varphi(r) = e^{-\varepsilon^2 r^2}$ mit der euklidischen Norm am Interpolationspunkt $(\tilde{x}, \tilde{y})$

$$\varphi(\|(x, y) - (\tilde{x}, \tilde{y})\|_2) = e^{-\varepsilon^2(x^2+y^2)} e^{-\varepsilon^2(\tilde{x}^2+\tilde{y}^2)} e^{2\varepsilon^2(x\tilde{x}+y\tilde{y})},\ (x, y) \in \mathbb{R}^2. \tag{4.1.2}$$

Die Faktoren $e^{-\varepsilon^2(x^2+y^2)}$ und $e^{-\varepsilon^2(\tilde{x}^2+\tilde{y}^2)}$ sind für $\varepsilon \to \infty$ vernachlässigbar, da sie entweder Interpolations- oder Auswertungspunkt enthalten. Es tritt zwar ebenfalls Auslöschung auf, allerdings ist der Informationsgehalt im verbliebenen Faktor

$$e^{2\varepsilon^2(x\tilde{x}+y\tilde{y})} \tag{4.1.3}$$

ausreichend für eine sehr gute Interpolation. Dazu muss der Term $e^{2\varepsilon^2(x\tilde{x}+y\tilde{y})}$ angepasst werden, indem zunächst die kartesischen Koordinaten in (4.1.3) in Polarkoordinaten umgeschrieben und davon anschließend die Taylorreihe gebildet wird. Sei

dazu (r,θ) die Polarkoordinaten von (x,y) und $(\tilde{r},\tilde{\theta})$ die von $(\tilde{x},\tilde{y})$, dann ist

$$e^{2\varepsilon^2(x\tilde{x}+y\tilde{y})} = e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})}. \tag{4.1.4}$$

Das folgende Theorem bildet den Hauptteil des RBF-QR-Algorithmus, da es insbesondere Tschebyscheffpolynome als Basisfunktionen für die Entwicklung von (4.1.4) einführt. Dazu benötigen wir noch die Definition der verallgemeinerten hypergeometrischen Reihen [9]

$${}_rF_s\left(a_1,a_2,\ldots,a_r;b_1,b_2,\ldots,b_s;z\right) = \sum_{n=0}^{\infty}\frac{(a_1)_n(a_2)_n\cdots(a_r)_n}{(b_1)_n(b_2)_n\cdots(b_s)_n}\frac{z^n}{n!},\ z\in\mathbb{C}, \tag{4.1.5}$$

wobei

$$(a)_0 = 1,\ (a)_n = \prod_{q=0}^{n-1}(a+q),\ a\in\mathbb{R}^+,\ n\in\mathbb{N}, \tag{4.1.6}$$

das Pochhammer-Symbol bezeichnet.

Außerdem benötigen wir noch folgendes Lemma.

Lemma 4.1.1. *Für $j,\ell,n\in\mathbb{N}_0$, $p=j\ \ (mod\ \ 2)$ gilt*

$$\binom{j-2m+2\ell}{\ell}\frac{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!}{2^{2\ell}\left(\frac{j+p+2m+2\ell}{2}\right)!\left(\frac{j-p-2m+2\ell}{2}\right)!} = \frac{\left(\frac{j+p-2m+1}{2}\right)_\ell}{\ell!\,(j-2m+1)_\ell\left(\frac{j+p+2m+2}{2}\right)_\ell}.$$

Beweis: Die Gleichung ergibt sich aus

$$\begin{aligned}
&\binom{j-2m+2\ell}{\ell}\frac{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!}{2^{2\ell}\left(\frac{j+p+2m+2\ell}{2}\right)!\left(\frac{j-p-2m+2\ell}{2}\right)!}\\
&= \frac{(j-2m+2\ell)!}{2^{2\ell}\ell!(j-2m+\ell)!}\prod_{q=0}^{\ell-1}\frac{1}{\left(\frac{j+p+2m+2}{2}+q\right)\left(\frac{j-p-2m+2}{2}+q\right)}\\
&= \frac{1}{2^{2\ell}\ell!}\prod_{q=0}^{\ell-1}\frac{(j-2m+\ell+1+q)}{\left(\frac{j+p+2m+2}{2}+q\right)\left(\frac{j-p-2m+2}{2}+q\right)}\\
&= \frac{2^{2\ell}\prod_{q=\ell+1}^{2\ell}(j-2m+q)}{2^{2\ell}\ell!\prod_{q=0}^{\ell-1}(j+p+2m+2+2q)\,(j-p-2m+2+2q)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\prod_{q=1}^{2\ell}(j-2m+q)}{\ell!\prod_{q=0}^{\ell-1}(j-2m+1+q)\,(j+p+2m+2+2q)\,(j-p-2m+2+2q)} \\
&= \frac{\prod_{q=0}^{\ell-1}(j-2m+2q+1)(j-2m+2q+2)}{\ell!\prod_{q=0}^{\ell-1}(j-2m+1+q)\,(j+p+2m+2+2q)\,(j-p-2m+2+2q)} \\
&= \frac{1}{2^\ell \ell!\,(j-2m+1)_\ell\left(\frac{j+p+2m+2}{2}\right)_\ell}\underbrace{\frac{\prod_{q=0}^{\ell-1}(j-2m+2q+1)(j-2m+2q+2)}{\prod_{q=0}^{\ell-1}(j-p-2m+2+2q)}}_{=2^\ell\prod_{q=0}^{\ell-1}\left(\frac{j+p-2m+1}{2}+q\right)} \\
&= \frac{\left(\frac{j+p-2m+1}{2}\right)_\ell}{\ell!\,(j-2m+1)_\ell\left(\frac{j+p+2m+2}{2}\right)_\ell}.
\end{aligned}
$$

□

Kommen wir nun zum angesprochenen Theorem.

Theorem 4.1.2. *Sei T_n das n-te Tschebyscheff-Polynom, $j,m \in \mathbb{N}_0$ und $p = j \pmod 2$. Für Punkte $(x,y) \in \mathbb{R}^2$ und $(\tilde{x},\tilde{y}) \in \mathbb{R}^2$ mit Polarkoordinaten $(r,\theta) \in \mathbb{R}^+ \times [0,2\pi)$ bzw. $(\tilde{r},\tilde{\theta}) \in \mathbb{R}^+ \times [0,2\pi)$ gilt*

$$
\begin{aligned}
&e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} \\
&= \sum_{j=0}^{\infty}\varepsilon^{2j}\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!}\,{}_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2), \qquad (4.1.7)
\end{aligned}
$$

mit $\varepsilon > 0$ und

$$
\alpha_{j,m} = \frac{j-2m+p+1}{2}, \quad \beta_{j,m} = \left(j-2m+1, \frac{j+2m+p+2}{2}\right). \qquad (4.1.8)
$$

Außerdem ist

$$
b_n = \begin{cases} 1, & \text{falls } n = 0, \\ 2, & \text{sonst,} \end{cases}
$$

und

$$t_m = \begin{cases} 1, & m > 0, \\ 1/2, & m = 0, , \\ 0, & m < 0, \end{cases}$$

sowie $\Theta_n = \cos(n\theta)\cos(n\tilde{\theta}) + \sin(n\theta)\sin(n\tilde{\theta})$.

Beweis: Zunächst gilt mit dem Variablenwechsel $m \mapsto m + \frac{1}{2}(j+p)$

$$\begin{aligned}
e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} &= \sum_{j=0}^{\infty} \frac{2^j\varepsilon^{2j}}{j!} r^j\tilde{r}^j \cos^j(\theta-\tilde{\theta}) \\
&= \sum_{j=0}^{\infty} \frac{2^j\varepsilon^{2j}}{j!} r^j\tilde{r}^j \sum_{m=0}^{j} \binom{j}{m} \frac{1}{2^j} \cos\left((j-2m)(\theta-\tilde{\theta})\right) \\
&= \sum_{j=0}^{\infty} \frac{2^j\varepsilon^{2j}}{j!} r^j\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \binom{j}{\frac{j+p}{2}+m} \frac{b_{2m+p}}{2^j} \cos\left((2m+p)(\theta-\tilde{\theta})\right) \\
&= \sum_{j=0}^{\infty} \frac{2^j\varepsilon^{2j}}{j!} r^j\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \binom{j}{\frac{j+p+2m}{2}} \frac{b_{2m+p}}{2^j} \Theta_{2m+p} \\
&= \sum_{j=0}^{\infty} \varepsilon^{2j} r^j\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \binom{j}{\frac{j+p+2m}{2}} \frac{b_{2m+p}}{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} \Theta_{2m+p}.
\end{aligned}$$

Dabei sind insbesondere, wegen der Definition von j und p, $j+p$ und $j-p$ gerade und die Fakultäten im Nenner existieren. Mit folgender Identität aus [21]

$$r^j\Theta_{2m+p} = r^{2m}\Theta_{2m+p}2^{-(j-2m-1)} \sum_{\ell=0}^{\frac{j-p-2m}{2}} \binom{j-2m}{\ell} t_{j-2m-2\ell}T_{j-2m-2\ell}(r), \qquad (4.1.9)$$

mit t_m aus (4.1.9). Es folgt nun

$$\begin{aligned}
e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} = \sum_{j=0}^{\infty}\sum_{m=0}^{\frac{j-p}{2}} \varepsilon^{2j}\tilde{r}^j \frac{b_{2m+p}}{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} r^{2m}\Theta_{2m+p}2^{-(j-2m-1)} \\
\sum_{\ell=0}^{\frac{j-p-2m}{2}} \binom{j-2m}{\ell} t_{j-2m-2\ell}T_{j-2m-2\ell}(r).
\end{aligned}$$

Wegen $t_m = 0$ für $m < 0$ erhalten wir für $j \mapsto j + 2\ell$

$$e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} = \sum_{\ell=0}^{\infty}\sum_{j=-2\ell}^{\infty} \varepsilon^{2j}\tilde{r}^j \sum_{m=0}^{\frac{j-p+2\ell}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} \frac{\varepsilon^{4\ell}\tilde{r}^{2\ell}}{2^{2\ell}}\binom{j-2m+2\ell}{\ell}\frac{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!}{\left(\frac{j+p+2m+2\ell}{2}\right)!\left(\frac{j-p-2m+2\ell}{2}\right)!}.$$

Für $j \in \{-2\ell, -2\ell+1, \ldots, -1\}$ ist $j-2m < 0$ für alle $m, \ell \geq 0$ und damit $t_{j-2m} = 0$ nach (4.1.9). Damit sind alle Terme der Summe über j gleich 0 für $j < 0$ oder $m \geq (j-p+2)/2$. Es folgt

$$e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} = \sum_{\ell=0}^{\infty}\sum_{j=0}^{\infty} \varepsilon^{2j}\tilde{r}^j \sum_{m=0}^{\frac{j-p+2\ell}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} \frac{\varepsilon^{4\ell}\tilde{r}^{2\ell}}{2^{2\ell}}\binom{j-2m+2\ell}{\ell}\frac{\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!}{\left(\frac{j+p+2m+2\ell}{2}\right)!\left(\frac{j-p-2m+2\ell}{2}\right)!}. \tag{4.1.10}$$

Insgesamt folgt nun mit Lemma 4.1.1 für (4.1.10)

$$e^{2\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} = \sum_{j=0}^{\infty} \varepsilon^{2j}\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} \sum_{\ell=0}^{\infty}\frac{\varepsilon^{4\ell}\tilde{r}^{2\ell}}{\ell!}\frac{\left(\frac{j+p-2m+1}{2}\right)_\ell}{(j-2m+1)_\ell\left(\frac{j+p+2m+2\ell}{2}\right)_\ell}. \tag{4.1.11}$$

Mit (4.1.5) folgt

$$\sum_{\ell=0}^{\infty}\frac{\varepsilon^{4\ell}\tilde{r}^{2\ell}}{\ell!}\frac{\left(\frac{j+p-2m+1}{2}\right)_\ell}{(j-2m+1)_\ell\left(\frac{j+p+2m+2\ell}{2}\right)_\ell} = {}_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2) \tag{4.1.12}$$

für

$$\alpha_{j,m} = \frac{j-2m+p+1}{2}, \quad \beta_{j,m} = \left(j-2m+1, \frac{j+2m+p+2}{2}\right). \tag{4.1.13}$$

und damit die Behauptung. □

Mit Theorem 4.1.2 folgt nun für (4.1.2) in Polarkoordinaten

$$\tilde{\varphi}(r,\theta,\tilde{r},\tilde{\theta}) = \sum_{j=0}^{\infty}\sum_{m=0}^{\frac{j-p}{2}} \frac{\varepsilon^{2j}b_{2m+p}t_{j-2m}e^{-\varepsilon^2\tilde{r}^2}\tilde{r}^j {}_1F_2(\alpha_{j,m},\beta_{j,m},\varepsilon^4\tilde{r}^2)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} \left(e^{-\varepsilon^2\tilde{r}^2}r^{2m}T_{j-2m}(r)\Theta_{2m+p}\right). \tag{4.1.14}$$

Wir setzen nun

$$
\begin{aligned}
d_{j,m} &= \frac{\varepsilon^{2j}}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!},\\
c_{j,m}(\tilde{r},\tilde{\theta}) &= b_{2m+p}t_{j-2m}e^{-\varepsilon\tilde{r}^2}\tilde{r}^j\cos((2m+p)\tilde{\theta})_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2),\\
s_{j,m}(\tilde{r},\tilde{\theta}) &= b_{2m+p}t_{j-2m}e^{-\varepsilon\tilde{r}^2}\tilde{r}^j\sin((2m+p)\tilde{\theta})_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2),\\
T^c_{j,m}(r,\theta) &= \cos((2m+p)\theta)e^{-\varepsilon^2r^2}r^{2m}T_{j-2m}(r),\\
T^s_{j,m}(r,\theta) &= \sin((2m+p)\theta)e^{-\varepsilon^2r^2}r^{2m}T_{j-2m}(r),
\end{aligned}
$$

und erhalten

$$
\begin{aligned}
&\tilde{\varphi}(r,\theta,\tilde{r},\tilde{\theta})\\
&=\sum_{j=0}^{\infty}\sum_{m=0}^{\frac{j-p}{2}}d_{j,m}c_{j,m}(\tilde{r},\tilde{\theta})T^c_{j,m}(r,\theta)+\sum_{m=1-p}^{\frac{j-p}{2}}d_{j,m}s_{j,m}(\tilde{r},\tilde{\theta})T^s_{j,m}(r,\theta).
\end{aligned}
\tag{4.1.15}
$$

In der Entwicklung (4.1.15) sind nun Interpolations- und Auswertungspunkte getrennt. Insbesondere werden nur die Auswertungspunkte in den neuen Basisfunktionen verwendet. Die numerische Instabilität wird nun über eine QR-Zerlegung der Interpolationsmatrix abgefangen. Dazu schreiben wir obige Summe in Matrixform. Seien (r_i,θ_i), $i=1,\dots,n$, die Interpolationspunkte und (r,θ) ein Auswertungspunkt, jeweils in Polarkoordinaten. Außerdem seien $j_{\max}$ der Index, an dem die Summe über j abgeschnitten wird und $m_{\max}:=(j_{\max}-p)/2$. Für die Bestimmung von $j_{\max}$ gehen wir in der späteren numerischen Berechnung analog zu [5, Fornberg et al.] vor. Wir setzen

$$
C(r_i,\theta_i)=\left(c_{0,0}(r_i,\theta_i),c_{1,0}(r_i,\theta_i),\dots,s_{j_{\max},m_{\max}}(r_i,\theta_i)\right), \tag{4.1.16}
$$

$$
C=\left\{C(r_i,\theta_i)\right\}_{i=1}^{n}, \tag{4.1.17}
$$

$$
D=\begin{pmatrix}
d_{0,0} & 0 & \cdots & \cdots & 0\\
0 & d_{1,0} & \ddots & & \vdots\\
\vdots & \ddots & d_{1,0} & \ddots & \vdots\\
\vdots & & \ddots & \ddots & 0\\
0 & \cdots & \cdots & 0 & d_{j_{\max},m_{\max}}
\end{pmatrix}, \tag{4.1.18}
$$

sowie

$$
T(r,\theta)=\left(T^c_{0,0}(r,\theta),\ T^c_{1,0}(r,\theta),\ T^s_{1,0}(r,\theta),\ \cdots,\ T^s_{j_{\max},m_{\max}}(r,\theta)\right), \tag{4.1.19}
$$

und erhalten

$$\underbrace{\begin{pmatrix} \varphi(\|(x,y)-(x_1,y_1)\|) \\ \varphi(\|(x,y)-(x_2,y_2)\|) \\ \vdots \\ \varphi(\|(x,y)-(x_n,y_n)\|) \end{pmatrix}}_{=:\Phi(x,y)} = C \cdot D \cdot T(\varrho(x,y))^T \tag{4.1.20}$$

bzw. für die Interpolationsmatrix

$$(\Phi(x_j,y_j)\|))_{j=1}^n = C \cdot D \cdot \begin{pmatrix} T(\varrho(x_1,y_1)) \\ T(\varrho(x_2,y_2)) \\ \vdots \\ T(\varrho(x_n,y_n)) \end{pmatrix}^T, \tag{4.1.21}$$

wobei $\varrho : \mathbb{R}^2 \mapsto \mathbb{R}^+ \times [0, 2\pi)$ die Polarkoordinatentransformation bezeichnet. Wir betrachten wieder (4.1.20) und bilden die QR-Zerlegung der Koeffizientenmatrix C:

$$\begin{aligned} \Phi(x,y) &= Q \cdot R \cdot D \cdot T(\varrho(x,y))^T = Q \cdot [\ R_1\ R_2\ ] \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \cdot T(\varrho(x,y))^T \\ &= Q \cdot [\ R_1 D_1\ R_2 D_2\ ] \cdot T(\varrho(x,y))^T, \end{aligned} \tag{4.1.22}$$

wobei R_1 eine obere Dreiecksmatrix ist und $R_1,\ D_1 \in \mathbb{R}^{M\times M}$. Nun bilden wir die neue Basisfunktionen

$$\begin{aligned} \Psi(x,y) &= D_1^{-1} R_1^{-1} Q^H \Phi(x,y) \\ &= \left[\ I \quad D_1^{-1} R_1^{-1} R_2 D_2\ \right] \cdot T(\varrho(x,y))^T = \left[I \ \ \tilde{R}\ \right] \cdot T(\varrho(x,y))^T. \end{aligned} \tag{4.1.23}$$

Dies ist legitim, da wir jede beliebige lineare unabhängige Kombination der $\{\varphi(\|x - x_i\|)\}_{i=1}^N$ benutzen können, ohne den Approximationsraum zu ändern. Die Interpolationsmatrix A berechnen wir nun über

$$\begin{pmatrix} \Psi(x_1,y_1) & \cdots & \Psi(x_N,y_N) \end{pmatrix}^T = \begin{pmatrix} T(x_1,y_1) \\ \vdots \\ T(x_N,y_N) \end{pmatrix} \begin{pmatrix} I \\ \tilde{R}^T \end{pmatrix} = T_1 + T_2 \tilde{R}^T,$$

wobei T_1 die ersten, M und T_2 die restlichen Basisfunktionen enthält, ausgewertet jeweils an (x_i, y_i), $i = 1, \ldots, N$. Wir erhalten damit die Koeffizienten λ und können an jeder beliebigen Stelle (x, y) auswerten mit

$$s(x,y) = \Psi(x,y)^T \lambda.$$

Algorithmus 1 (RBF-QR)**.** Die Entwicklung (4.1.15) und deren Anwendung über die Matrixformulierung, mit anschließender Umwandlung der Basisfunktionen und QR-Faktorisierung, bezeichnen wir als RBF-QR-Algorithmus.

Der Quelltext des Algorithmus und eine detailliertere Beschreibung der Herleitung findet sich in [5], der Programmcode kann unter [23] heruntergeladen werden.

4.2 Der RBF-QR-GQ-Algorithmus

Mit dem RBF-QR Algorithmus ist eine stabile Methode gegeben, um sehr kleine ε-Werte sowie eine sehr große Anzahl von Interpolationspunkten zu handhaben. Wir werden nun, ausgehend vom RBF-QR Algorithmus 1, einen Algorithmus für andere vollständig monotone RBFs angeben, den RBF-QR-GQ-Algorithmus. Idee ist es, Theorem 2.1.3 anzuwenden, dass heißt wir schreiben die vollständig monotone RBF φ über die Bernstein-Widder-Darstellung um, in der Form

$$\varphi(s) = \int_0^\infty e^{-\alpha s} \, \mathrm{d}\mu(\alpha), \ \alpha > 0, \tag{4.2.24}$$

für $s \geq 0$ und ein nicht fallendes, nach unten beschränkte Maß μ.

Nun nutzen wir aus, dass der Gaußkern $\varphi(r) = e^{-\varepsilon^2 r^2}$, $\varepsilon > 0$, durch diese Darstellung als Prototyp für vollständig monotonen Funktion dient. Mit anderen Worten, alle vollständig monotonen Funktionen können mit $g(t) = e^{-\alpha t}$, $\alpha > 0$, als Kernfunktion in der Laplacetransformation (4.2.24), bzgl. eines Maßes μ, dargestellt werden. Wir werden dies zunächst an der inversen Multiquadrics vorführen.

4.2.1 Die inverse Multiquadrics

Die inverse Multiquadrics (IMQ)

$$\varphi(r) = \frac{1}{\sqrt{1 + \varepsilon^2 r^2}} \tag{4.2.25}$$

ist nach Beispiel 2, Proposition 2.1.2, vollständig monoton und somit über die Bernstein-Widder-Darstellung aus Theorem 2.1.3 für ein entsprechendes Maß mithilfe des Gaußkerns darstellbar.

Zur Berechnung der Maßfunktion der Bernstein-Widder-Darstellung 2.1.3 benötigen wir wieder die Gammafunktion Γ für positive reelle Zahlen z aus (2.1.19), sowie deren folgende Eigenschaften

1. Für $a \in \mathbb{R}^+$, $n \in \mathbb{N}_0$ ergibt sich das Pochhammer-Symbol (4.1.6) durch

$$(a)_n = \frac{\Gamma(a+n)}{\Gamma(n)}. \tag{4.2.26}$$

2. Für die Gammafunktion (2.1.19) an der Stelle $1/2$ gilt mit Substitution $t = su$,

$s > 0$,

$$\Gamma(1/2) = \int_0^\infty t^{-1/2} e^{-t} \, \mathrm{d}t = s^{1/2} \int_0^\infty u^{-1/2} e^{-su} \, \mathrm{d}u. \tag{4.2.27}$$

Lemma 4.2.1. *Die IMQ* (4.2.25) *besitzt in der Bernstein-Widder-Darstellung* (4.2.24) *folgende Maßfunktion*

$$\mu(u) = \frac{1}{\Gamma(1/2)} u^{-1/2} \, \mathrm{d}u.$$

Beweis: Wir setzen in (4.2.27) $s := 1 + r^2\varepsilon^2$ und erhalten

$$\begin{aligned}
& \Gamma(1/2) = \left(1 + r^2\varepsilon^2\right)^{1/2} \int_0^\infty u^{-1/2} e^{-(1+r^2\varepsilon^2)u} \, \mathrm{d}u \\
\iff \quad & \left(1 + r^2\varepsilon^2\right)^{-1/2} = \frac{1}{\Gamma(1/2)} \int_0^\infty u^{-1/2} e^{-(1+r^2\varepsilon^2)u} \, \mathrm{d}u \\
& \qquad = \frac{1}{\Gamma(1/2)} \int_0^\infty u^{-1/2} e^{-u} e^{-u(r^2\varepsilon^2)} \, \mathrm{d}u \\
\implies \quad & \varphi(r) = \frac{1}{\Gamma(1/2)} \int_0^\infty u^{-1/2} e^{-u(1+\varepsilon^2 r^2)} \, \mathrm{d}u.
\end{aligned}$$

□

Das Integral in der Darstellung

$$\varphi(r) = \frac{1}{\Gamma(1/2)} \int_0^\infty u^{-1/2} e^{-u(1+\varepsilon^2 r^2)} \, \mathrm{d}u \tag{4.2.28}$$

nähern wir zunächst mittels Gauß-Laguerre-Quadratur an. Die Gewichtsfunktion ist in diesem Fall (u.a. zu finden in [17])

$$\omega(u) = u^{-1/2} e^{-u}.$$

Insgesamt gilt

$$\int_0^\infty \omega(u) e^{-u\varepsilon^2 r^2} \, \mathrm{d}u = \sum_{q=1}^{N_Q} \omega_q e^{-\hat{x}_q \varepsilon^2 r^2} + R_{N_Q}, \tag{4.2.29}$$

für ein $N_Q \in \mathbb{N}$ und Fehlerterm

$$R_{N_Q} = \frac{N_Q! \, \Gamma(N_Q - 1/2)}{(2N_Q)!} \frac{\mathrm{d}^{2N_Q}}{\mathrm{d}u^{2N_Q}} e^{-u\varepsilon^2 r^2} \Big|_{u=\xi}, \ \xi \in \mathbb{R}^+. \tag{4.2.30}$$

Die Gewichte ω_q zu den Knoten $\hat{x}_q$, $q = 1, \ldots, N_Q$, sind gegeben durch

$$\omega_q = \frac{\Gamma(N_Q + 1/2)\hat{x}_q}{N_Q!(N_Q+1)^2 \left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2}. \tag{4.2.31}$$

Die Knoten bestimmen sich als Nullstellen des Laguerre-Polynoms $L_{N_Q}^{-1/2}$ mit

$$L_N^{\alpha}(z) = \sum_{m=0}^{N} \frac{\Gamma(N+\alpha+1)}{\Gamma(N-m+1)\Gamma(\alpha+m+1)} \frac{(-x)^m}{m!},\ \alpha > -1. \tag{4.2.32}$$

Zusammen haben wir für den Interpolationspunkt $(\tilde{x}, \tilde{y})$ und Auswertungspunkt (x, y)

$$\varphi(\|(x,y)-(\tilde{x},\tilde{y})\|_2) = \frac{1}{\Gamma(1/2)} \sum_{q=1}^{N_Q} \omega_q e^{-\hat{x}_q\varepsilon^2(\tilde{x}^2+\tilde{y}^2)} e^{-\hat{x}_q\varepsilon^2(x^2+y^2)} e^{2\hat{x}_q\varepsilon^2(x\tilde{x}+y\tilde{y})} + R_{N_Q}.$$

Mit Theorem 4.1.2 folgt für die Polarkoordinatendarstellung von

$$e^{2\hat{x}_q\varepsilon^2(x\tilde{x}+y\tilde{y})}$$

mit (r, θ) für (x, y) und $(\tilde{r}, \tilde{\theta})$ für $(\tilde{x}, \tilde{y})$ dass

$$\begin{aligned}
&e^{2\hat{x}_q\varepsilon^2 r\tilde{r}(\cos\theta\cos\tilde{\theta}+\sin\theta\sin\tilde{\theta})} \\
=&\sum_{j=0}^{\infty} \varepsilon^{2j}\hat{x}_q^j\tilde{r}^j \sum_{m=0}^{\frac{j-p}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} {}_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2\tilde{x}_q^2),
\end{aligned}$$

für $q = 1, \ldots, N_Q$ und $\alpha_{j,m}$, $\beta_{j,m}$ aus (4.1.8).

Wir erhalten als Näherung an die *inverse Multiquadrics* (4.2.25) mit Fehlerterm (4.2.30) insgesamt

$$\begin{aligned}
&\tilde{\varphi}(r,\theta,\tilde{r},\tilde{\theta}) \\
\approx&\frac{1}{\Gamma(1/2)} \sum_{q=1}^{N_Q} \omega_q \sum_{j=0}^{\infty} \varepsilon^{2j}\hat{x}_q^j\tilde{r}^j \\
&\sum_{m=0}^{\frac{j-p}{2}} \frac{b_{2m+p}t_{j-2m}r^{2m}\Theta_{2m+p}T_{j-2m}(r)}{2^{j-2m-1}\left(\frac{j+p+2m}{2}\right)!\left(\frac{j-p-2m}{2}\right)!} {}_1F_2(\alpha_{j,m};\beta_{j,m};\varepsilon^4\tilde{r}^2\tilde{x}_q^2) \\
=&\frac{\Gamma(N_Q+1/2)}{\Gamma(1/2)\ N_Q!\ (N_Q+1)^2} \sum_{q=1}^{N_Q}\sum_{j=0}^{\infty}
\end{aligned}$$

$$\sum_{m=0}^{\frac{j-p}{2}} \varepsilon^{2j} \tilde{r}^j \frac{\hat{x}_q^{j+1}}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} \frac{b_{2m+p} t_{j-2m} r^{2m} \Theta_{2m+p} T_{j-2m}(r)}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!}$$
$$e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2} e^{-\hat{x}_q \varepsilon^2 r^2} {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \tilde{x}_q^2).$$

Als nächstes stellen wir die Matrixdarstellung $\Phi(x, y)$ in (4.1.20) der IMQ für Interpolationspunkte (x_i, y_i), $i = 1, \ldots, n$, und den Auswertungspunkt (x, y) auf. Dazu setzen wir wieder

$$d_{j,m} := \frac{\varepsilon^{2j}}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!}, \tag{4.2.33}$$

sowie

$$c_{q,j,m}(\tilde{r}, \tilde{\theta}) := \frac{\hat{x}_q^{j+1}}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} b_{2m+p} t_{j-2m} \cos((2m+p)\theta_i) \tilde{r}^j e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2}$$
$${}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \tilde{x}_q^2), \tag{4.2.34}$$
$$s_{q,j,m}(\tilde{r}, \tilde{\theta}) := \frac{\hat{x}_q^{j+1}}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} b_{2m+p} t_{j-2m} \sin((2m+p)\theta_i) \tilde{r}^j e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2}$$
$${}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \tilde{x}_q^2), \tag{4.2.35}$$
$$T^c_{q,j,m}(r, \theta) := r^{2m} \cos((2m+p)\theta) e^{-\hat{x}_q \varepsilon^2 r^2} T_{j-2m}(r), \tag{4.2.36}$$
$$T^s_{q,j,m}(r, \theta) := r^{2m} \sin((2m+p)\theta) e^{-\hat{x}_q \varepsilon^2 r^2} T_{j-2m}(r). \tag{4.2.37}$$

Damit und mit der Identität der verallgemeinerten hypergeometrischen Reihen (4.1.12) mit Argumenten (4.1.8) folgt

$$\tilde{\varphi}(r, \theta, \tilde{r}, \tilde{\theta}) \approx \frac{\Gamma(N_Q + 1/2)}{\Gamma(1/2)\, N_Q!\, (N_Q+1)^2} \left(\sum_{q=1}^{N_Q} \sum_{j=0}^{\infty} \sum_{m=0}^{\frac{j-p}{2}} d_{j,m}\, c_{q,j,m}(\tilde{r}, \tilde{\theta}) T^c_{q,j,m}(r, \theta) \right.$$
$$\left. + \sum_{q=1}^{N_Q} \sum_{j=0}^{\infty} \sum_{m=1-p}^{\frac{j-p}{2}} d_{j,m} s_{q,j,m}(\tilde{r}, \tilde{\theta}) T^s_{q,j,m}(r, \theta) \right). \tag{4.2.38}$$

In der Entwicklung (4.2.38) sind, wie in (4.1.15), Interpolations- und Auswertungspunkte getrennt. Dadurch ist es wieder möglich die numerische Instabilität über eine QR-Zerlegung aufzufangen.

Wir schneiden die Summe über j wieder bei $j_{\max}$ ab und setzen $m_{\max} := (j_{\max} - p)/2$. Außerdem verwenden wir wieder die Polarkoordinatendarstellung der Interpolations-

und Auswertungspunkte. Für ein festes q aus $\{1,\ldots,N_Q\}$ setzen wir

$$C_q(r_i,\theta_i) = \Big(c_{q,0,0}(r_i,\theta_i),\quad c_{q,1,0}(r_i,\theta_i),\quad s_{q,1,0}(r_i,\theta_i),\quad \cdots,\quad s_{q,j_{\max},m_{\max}}(r_i,\theta_i)\Big),$$

und

$$T_q(x,y) = \Big(T^c_{q,0,0}(r,\theta),\quad T^c_{q,1,0}(r,\theta),\quad T^s_{1,0}(r,\theta),\quad \cdots,\quad T^s_{q,j_{\max},\frac{j_{\max}-p}{2}}(r,\theta)\Big). \tag{4.2.39}$$

Damit erhalten wir

$$C(r_i,\theta_i) = \Big(C_1(r_i,\theta_i),\quad \cdots,\quad C_{N_Q}(r_i,\theta_i)\Big), \tag{4.2.40}$$

$$C = \{C(r_i,\theta_i)\}_{i=1}^{N}, \tag{4.2.41}$$

$$T(r,\theta) = \Big(T_1(r,\theta),\quad \cdots,\quad T_{N_Q}(r,\theta)\Big), \tag{4.2.42}$$

sowie

$$D_{N_Q,N_Q} = I_{N_Q,N_Q} \otimes \tilde{D} \tag{4.2.43}$$

mit

$$\tilde{D} = \begin{pmatrix} d_{0,0} & 0 & \cdots & \cdots & 0 \\ 0 & d_{1,0} & \ddots & & \vdots \\ \vdots & \ddots & d_{1,0} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & d_{j_{\max},m_{\max}} \end{pmatrix} \tag{4.2.44}$$

und I_{N_Q,N_Q} als Einheitsmatrix der Dimension $N_Q \times N_Q$ und $\otimes$ als Kronecker-Produkt.

Mit den Matrizen C, $T(r,\theta)$ und D_{N_Q,N_Q} erhalten wir analog zu (4.1.20)

$$\underbrace{\begin{pmatrix} \varphi(\|(x,y)-(x_1,y_1)\|) \\ \varphi(\|(x,y)-(x_2,y_2)\|) \\ \vdots \\ \varphi(\|(x,y)-(x_n,y_n)\|) \end{pmatrix}}_{=:\Phi(x,y)} \approx C \cdot D \cdot T(\varrho(x,y))^T \tag{4.2.45}$$

und für die Interpolationsmatrix

$$(\varphi(\|(x_i,y_i)-(x_j,y_j)\|))_{i,j=1}^{n} \approx C \cdot D \cdot \begin{pmatrix} T(\varrho(x_1,y_1)) \\ T(\varrho(x_2,y_2)) \\ \vdots \\ T(\varrho(x_n,y_n)) \end{pmatrix}^T . \tag{4.2.46}$$

Die Vorfaktoren

$$\frac{\Gamma(N_Q+1/2)}{\Gamma(1/2)\ N_Q!\ (N_Q+1)^2}$$

haben wir dabei ohne Einschränkung für die Interpolation weggelassen. Im Vergleich mit (4.1.22) haben wir nun eine um den Faktor N_Q erhöhte Anzahl an Spalten. Die Anordnung der Spalten ist für die Invertierung der Dreiecksmatrix R_1 aus $C = Q[R_1R_2]$ nicht trivial. Da die Gewichte und Knoten der Gauß-Quadratur-Formel unabhängig von N und ε sind, kann eine andere Anordnung zu einer singulären Matrix R_1 führen. Aus diesem Grund erfolgt die Summation in (4.2.38) in der entsprechenden Reihenfolge. Bei Vertauschen der Summen zu

$$\sum_{j=0}^{\infty}\sum_{m=0}^{\frac{j-p}{2}}\sum_{q=1}^{N_Q}$$

unterscheiden sich die ersten N_Q-Spalten nicht in den neuen Basisfunktionen oder ε, sondern nur in den Knoten und Gewichten und sind nicht linear unabhängig voneinander, was sich zur Matrix R_1 fortsetzt und deren Invertierung nicht ermöglicht.

Beispiel 3: Sei $N_Q = 2$, $j_{\max} = 1$ und die Punkte $(0,0)$ und $(1,0)$ gegeben, dann haben wir die Knoten

$$\hat{x}_1 = 2.72474,\ \hat{x}_2 = 0.27526.$$

1. Für die Summationsreihenfolge $\sum_{q=1}^{2}\sum_{j=0}^{1}\sum_{m=0}^{\frac{j-p}{2}}$ erhalten wir mit (4.2.40)

$$\begin{aligned} C &= \begin{pmatrix} c_{1,0,0}(r_1,\theta_1) & c_{1,1,0}(r_1,\theta_1) & s_{1,1,0}(r_1,\theta_1) & c_{2,0,0}(r_1,\theta_1) & c_{2,1,0}(r_1,\theta_1) & s_{2,1,0}(r_1,\theta_1) \\ c_{1,0,0}(r_2,\theta_2) & c_{1,1,0}(r_2,\theta_2) & s_{1,1,0}(r_2,\theta_2) & c_{2,0,0}(r_2,\theta_2) & c_{2,1,0}(r_2,\theta_2) & s_{2,1,0}(r_2,\theta_2) \end{pmatrix} \\ &= \begin{pmatrix} 0.27515 & 1.92279 & 1.92279 & 6.06119 & 10.24514 & 10.24514 \\ 1.10102 & 0 & 0 & 10.89898 & 0 & 0 \end{pmatrix} \end{aligned}$$

mit QR-Zerlegung

$$Q = \begin{pmatrix} -0.24245 & -0.97016 \\ -0.97016 & 0.24245 \end{pmatrix},$$
$$R = \begin{pmatrix} -1.13488 & -0.46618 & -0.46618 & -12.04333 & -2.48393 & -2.48393 \\ 0 & -1.86542 & -1.86542 & -3.23789 & -9.93946 & -9.93946 \end{pmatrix}.$$

Für die Teilmatrix von $R_1 = \{r_{ij}\}_{i,j=1}^{2}$ von $R = \{r_{ij}\}_{i,j=1}^{2\,6}$ gilt $\text{cond}(R_1) = 1.7988$.

2. Für die Summationsreihenfolge $\sum_{j=0}^{1} \sum_{m=0}^{\frac{j-p}{2}} \sum_{q=1}^{2}$ erhalten wir

$$C = \begin{pmatrix} c_{1,0,0}(r_1,\theta_1) & c_{2,0,0}(r_1,\theta_1) & c_{1,1,0}(r_1,\theta_1) & c_{2,1,0}(r_1,\theta_1) & s_{1,1,0}(r_1,\theta_1) & s_{2,1,0}(r_1,\theta_1) \\ c_{1,0,0}(r_2,\theta_2) & c_{2,0,0}(r_2,\theta_2) & c_{1,1,0}(r_2,\theta_2) & c_{2,1,0}(r_2,\theta_2) & s_{1,1,0}(r_2,\theta_2) & s_{2,1,0}(r_2,\theta_2) \end{pmatrix}$$
$$= \begin{pmatrix} 0.27515 & 6.06119 & 1.92279 & 10.24514 & 1.92279 & 10.24514 \\ 1.10102 & 10.89898 & 0 & 0 & 0 & 0 \end{pmatrix}$$

mit QR-Zerlegung

$$Q = \begin{pmatrix} -0.24245 & -0.97016 \\ -0.97016 & 0.24245 \end{pmatrix},$$
$$R = \begin{pmatrix} -1.13488 & -12.04333 & -0.46618 & -2.48393 & -0.46618 & -2.48393 \\ 0 & -3.23789 & -1.86542 & -9.93946 & -1.86542 & -9.93946 \end{pmatrix}.$$

Für die Teilmatrix von $R_1 = \{r_{ij}\}_{i,j=1}^{2}$ gilt diesmal $\text{cond}(R_1) = 42.651$, eine deutlich schlechtere Konditionierung für die anschließende Invertierung von R_1 im Vergleich zur Summationsreihenfolge in 1.

Analog zu (4.1.22) erhalten wir für (4.2.45) mit QR-Zerlegung der Koeffizientenmatrix C:

$$\Phi(x,y) = Q \cdot [\ R_1 D_1\ R_2 D_2\ ] \cdot T(\varphi(x,y))^T, \tag{4.2.47}$$

wobei R_1 wieder obere Dreiecksmatrix und R_1, $D_1 \in \mathbb{R}^{n\times n}$ ist. Für die neuen Basisfunktionen folgt analog zu (4.1.23)

$$\Psi(x,y) = \left[I \ \ \tilde{R} \right] \cdot \hat{T}(\varphi(x,y))^T. \tag{4.2.48}$$

Die Interpolationsmatrix A berechnen wir über

$$\begin{pmatrix} T(\varphi(x_1, y_1)) \\ \vdots \\ T(\varphi(x_N, y_N)) \end{pmatrix} \begin{pmatrix} I \\ \tilde{R}^T \end{pmatrix} = T_1 + T_2 \tilde{R}^T, \tag{4.2.49}$$

und erhalten damit die Koeffizienten λ. Die Auswertung erfolgt wieder über

$$s(x, y) = \Psi(x, y)^T \lambda.$$

Der Algorithmus ist im Programm `RBf_QR_GQ_2D.m` (Quelltext A.4) implementiert, wobei die Teile des ursprünglichen RBF-QR-Algorithmus aus [5] übernommen sind. Die hinzugefügten neuen Teile sind insbesondere die Berechnung der Indexmengen und hinzufügen von Gewichten und Knotenpunkten. Die Berechnung von N_Q sowie der Gewichte und Knoten sind im Programm `RBF_QR_GQ_quad_part.m` (Quelltext A.5) implementiert.

4.2.2 Verallgemeinerung der inversen Multiquadrics

Als nächstes betrachten wir

$$\varphi(s) = (1 + \varepsilon^2 s^2)^{-\beta}, \quad \beta \in \mathbb{R}^+. \tag{4.2.50}$$

Diese Verallgemeinerung der inversen Multiquadrics (4.2.50) bezeichnen wir als β-*inverse Multiquadrics* (β-IMQ). Es gilt

$$(1 + \varepsilon^2 s^2)^{-\beta} = \frac{1}{\Gamma(\beta)} \int_0^\infty u^{\beta-1} e^{-u(1+\varepsilon^2 s^2)} \, \mathrm{d}u,$$

mit der Maßfunktion $\mu(u) = \frac{1}{\Gamma(\beta)} u^{\beta-1} \, \mathrm{d}u$. Damit ist $\varphi(\sqrt{t})$ nach dem Bernstein-Widder-Theorem 2.1.3 vollständig monoton und die Interpolationsmatrix der β-IMQ (4.2.50) insbesondere positiv definit.

Als Gewichtsfunktion wählen wir diesmal

$$\omega_\beta(u) = u^{\beta-1} e^{-u},$$

und haben somit

$$\int_0^\infty \omega^{(\beta)}(u) e^{-u\varepsilon^2 r^2} \, \mathrm{d}u = \sum_{q=1}^{N_Q} \omega_q^{(\beta)} e^{-\hat{x}_q^{(\beta)} u \varepsilon^2 r^2} + R_{N_Q}^{(\beta)}, \tag{4.2.51}$$

für $N_Q \in \mathbb{N}$ und dem Fehlerterm

$$R_{N_Q}^{(\beta)} = \frac{N_Q!(N_Q+\beta-1)!}{(2N_Q)!} \left. \frac{\mathrm{d}^{2N_Q}}{\mathrm{d}u^{2N_Q}} e^{-u\varepsilon^2 r^2} \right|_{u=\xi}, \ \xi \in \mathbb{R}.$$

Die Gewichte ω_q zu den Knoten $\hat{x}_q^{(\beta)}$, $q = 1, \ldots, N_Q$, erhalten wir aus [17] als

$$\omega_q^{(\beta)} = \frac{\Gamma(N_Q+\beta)\hat{x}_q^{(\beta)}}{N_Q!(N_Q+1)^2 \left(L_{N_Q+1}^{\beta-1}(\hat{x}_q^{(\beta)})\right)^2}.$$

Die Knoten sind wieder die Nullstellen des Laguerre-Polynoms $L_{N_Q}^{\beta-1}$.

Wir erhalten analog zum Vorgehen zur IMQ für die β-IMQ mit Theorem 4.1.2 in Polarkoordinatendarstellung

$$\begin{aligned}
&\tilde{\varphi}(r,\theta,\tilde{r},\tilde{\theta}) \\
\approx &\frac{1}{\Gamma(\beta)} \sum_{q=1}^{N_Q} \omega_q^{(\beta)} \sum_{j=0}^{\infty} \varepsilon^{2j} \left(\hat{x}_q^{(\beta)}\right)^j \tilde{r}^j \\
&\sum_{m=0}^{\frac{j-p}{2}} \frac{b_{2m+p} t_{j-2m} r^{2m} \Theta_{2m+p} T_{j-2m}(r)}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!} {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \left(\hat{x}_q^{(\beta)}\right)^2) \\
= &\frac{\Gamma(N_Q+\beta)}{\Gamma(\beta)\ N_Q!\ (N_Q+1)^2} \sum_{q=1}^{N_Q} \sum_{j=0}^{\infty} \sum_{m=0}^{\frac{j-p}{2}} \varepsilon^{2j} \tilde{r}^j \frac{\left(\hat{x}_q^{(\beta)}\right)^{j+1}}{\left(L_{N_Q+1}^{\beta-1}(\hat{x}_q^{(\beta)})\right)^2} \frac{b_{2m+p} t_{j-2m} r^{2m} \Theta_{2m+p} T_{j-2m}(r)}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!} \\
&e^{-\hat{x}_q^{(\beta)} \varepsilon^2 \tilde{r}^2} e^{-\hat{x}_q^{(\beta)} \varepsilon^2 r^2} {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \left(\hat{x}_q^{(\beta)}\right)^2).
\end{aligned}$$

Wir setzen

$$d_{j,m} := \frac{\varepsilon^{2j}}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!}, \tag{4.2.52}$$

$$\begin{aligned}
c_{q,j,m}^{(\beta)}(\tilde{r},\tilde{\theta}) := &\frac{\left(\hat{x}_q^{(\beta)}\right)^{j+1}}{\left(L_{N_Q+1}^{\beta-1}(\hat{x}_q^{(\beta)})\right)^2} b_{2m+p} t_{j-2m} \cos((2m+p)\tilde{\theta}) \tilde{r}^j e^{-\hat{x}_q^{(\beta)} \varepsilon^2 \tilde{r}^2} \\
&{}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \left(\hat{x}_q^{(\beta)}\right)),
\end{aligned} \tag{4.2.53}$$

$$\begin{aligned}
s_{q,j,m}^{(\beta)}(\tilde{r},\tilde{\theta}) := &\frac{\left(\hat{x}_q^{(\beta)}\right)^{j+1}}{\left(L_{N_Q+1}^{\beta-1}(\hat{x}_q^{(\beta)})\right)^2} b_{2m+p} t_{j-2m} \sin((2m+p)\tilde{\theta}) \tilde{r}^j e^{-\hat{x}_q^{(\beta)} \varepsilon^2 \tilde{r}^2} \\
&{}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \left(\hat{x}_q^{(\beta)}\right)^2),
\end{aligned} \tag{4.2.54}$$

$$T_{q,j,m}^{c,\beta}(r,\theta) := r^{2m} \cos((2m+p)\theta) e^{-\hat{x}_q^{(\beta)} \varepsilon^2 r^2} T_{j-2m}(r), \tag{4.2.55}$$

$$T^{s,\beta}_{q,j,m}(r,\theta) := r^{2m}\sin((2m+p)\theta)e^{-\hat{x}_q^{(\beta)}\varepsilon^2 r^2}T_{j-2m}(r). \tag{4.2.56}$$

und erhalten analog zu (4.2.38) eine in Interpolations- und Auswertungspunkte aufgeteilte Entwicklung

$$\begin{aligned}\tilde{\varphi}_\beta(r,\theta,\tilde{r},\tilde{\theta}) \approx & \frac{\Gamma(N_Q+\beta)}{\Gamma(\beta)N_Q!(N_Q+1)^2}\left(\sum_{q=1}^{N_Q}\sum_{j=0}^{\infty}\sum_{m=0}^{\frac{j-p}{2}} d_{j,m}\, c^{(\beta)}_{q,j,m}(\tilde{r},\tilde{\theta})T^{c,\beta}_{q,j,m}(r,\theta)\right. \\ & \left. +\sum_{q=1}^{N_Q}\sum_{j=0}^{\infty}\sum_{m=1-p}^{\frac{j-p}{2}} d_{j,m}s^{(\beta)}_{q,j,m}(\tilde{r},\tilde{\theta})T^{s,\beta}_{q,j,m}(r,\theta)\right).\end{aligned} \tag{4.2.57}$$

Die Argumente der hypergeometrischen Reihe sind wieder aus (4.1.8). Seien (r_i,θ_i), $i=1,\dots,n$, die Interpolationspunkte und (r,θ) Auswertungspunkt und $j_{\max}$ und $m_{\max}$ wie im vorherigen Fall. Wir setzen wieder für festes q aus $\{1,\dots,N_Q\}$ und β

$$C^{(\beta)}_q(r_i,\theta_i) = \left(c^{(\beta)}_{q,0,0}(r_i,\theta_i),\quad c^{(\beta)}_{q,1,0}(r_i,\theta_i),\quad \cdots,\quad s^{(\beta)}_{q,j_{\max},m_{\max}}(r_i,\theta_i)\right), \tag{4.2.58}$$

sowie

$$T^{\beta}_q(r,\theta) = \left(T^{c,\beta}_{0,0}(r,\theta),\quad T^{c,\beta}_{1,0}(r,\theta),\quad \cdots,\quad T^{s,\beta}_{j_{\max},m_{\max}}(r,\theta)\right). \tag{4.2.59}$$

Damit

$$C_\beta(r_i,\theta_i) = \left(C^{(\beta)}_1(r_i,\theta_i),\quad \cdots,\quad C^{(\beta)}_{N_Q}(r_i,\theta_i)\right), \tag{4.2.60}$$

$$C = \{C(r_i,\theta_i)\}_{i=1}^N, \tag{4.2.61}$$

$$T_\beta(r,\theta) = \left(T^{(\beta)}_1(r,\theta),\quad \cdots,\quad T^{(\beta)}_{N_Q}(r,\theta)\right), \tag{4.2.62}$$

und mit D_{N_Q,N_Q} aus (4.2.43) folgt für die Interpolationsmatrix

$$(\varphi_\beta(\|(x_i,y_i)-(x_j,y_j)\|))_{i,j=1}^n \approx C_\beta\cdot D\cdot \begin{pmatrix} T_\beta(\varrho(x_1,y_1)) \\ T_\beta(\varrho(x_2,y_2)) \\ \vdots \\ T_\beta(\varrho(x_n,y_n))\end{pmatrix}^T. \tag{4.2.63}$$

Die Vorfaktoren

$$\frac{\Gamma(N_Q+\beta)}{\Gamma(1/2)\ N_Q!\ (N_Q+1)^2}$$

haben wir dabei ohne Einschränkung für die Interpolation weggelassen.

Wir erhalten wieder mit anschließender QR-Zerlegung der Koeffizientenmatrix $C_\beta = Q_\beta \cdot [\ R_{\beta,1}\ R_{\beta,2}\ ]$ für die neuen Basisfunktionen

$$\Psi_\beta(x,y) = \begin{bmatrix} I & \tilde{R}_\beta \end{bmatrix} \cdot P \cdot T_\beta(x,y)^T, \tag{4.2.64}$$

mit $\tilde{R}_\beta = D_1^{-1} R_{\beta,1}^{-1} R_{\beta,2} D_2$. Die Interpolationsmatrix berechnen wir analog zu (4.2.49), berechnen die Koeffizienten λ und können wieder an jeder beliebigen Stelle (x,y) auswerten mit

$$s(x,y) = \Psi_\beta(x,y)^T \lambda.$$

Der Algorithmus ist analog zum vorherigen zur IMQ im Programm `RBf_QR_GQ_2D.m` (Quelltext A.4) implementiert, die Gewichte und Knoten werden wieder über das Programm `RBF_QR_GQ_quad_part.m` (Quelltext A.5) berechnet.

4.2.3 Die Multiquadrics

Die Multiquadrics (MQ)

$$\varphi(s) = \left(1 + \varepsilon^2 s^2\right)^{1/2},\ \varepsilon > 0, \tag{4.2.65}$$

ist zwar nur vollständig monoton der Ordnung 1, jedoch ist die Interpolationsmatrix nach Satz 2.1.4 nichtsingulär. Insbesondere gilt [24]

$$\varphi(s) = 1 + \frac{1}{2\sqrt{\pi}} \int_0^\infty u^{-3/2} e^{-u} \left(1 - e^{-u\varepsilon^2 s^2}\right) \mathrm{d}u,\ \varepsilon > 0. \tag{4.2.66}$$

Wir werden zunächst das Integral wieder numerisch mittels verallgemeinerte Gauß-Laguerre-Quadratur annähern. Dazu nehmen wir wieder die Gewichtsfunktion

$$\omega(u) = u^{-1/2} e^{-u},$$

und erhalten mit $\alpha = -1/2$ in (4.2.32) die Gewichte [17]

$$\omega_q = \frac{\Gamma(N_Q + 1/2)\hat{x}_q}{N_Q!(N_Q+1)^2 \left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2}.$$

Die Knoten sind dabei wieder die Nullstellen von $L_{N_Q}^{-1/2}$. Wir setzen

$$\int_0^\infty u^{-3/2} e^{-u} \left(1 - e^{-u\varepsilon^2 s^2}\right) \mathrm{d}u = \sum_{q=0}^{N_Q} \omega_q \hat{x}_q^{-1} \left(1 - e^{-\hat{x}_q \varepsilon^2 s^2}\right) + R_{N_Q},$$

mit Fehlerterm (4.2.30). Der Unterschied zu (4.2.29) besteht in dem zusätzlichen Faktor $\hat{x}_q^{-1}$, der wegen der Definition des Laguerre-Polynoms (4.2.32), $\alpha > -1$, entsteht.

Insgesamt haben wir nun

$$\varphi(s) \approx 1 + \frac{1}{2\sqrt{\pi}} \sum_{q=0}^{N_Q} \omega_q \hat{x}_q^{-1} \left(1 - e^{-\hat{x}_q \varepsilon^2 s^2}\right), \ \varepsilon > 0. \tag{4.2.67}$$

In der Polarkoordinatendarstellung folgt

$$\begin{aligned}
\tilde{\varphi}(r,\theta,\tilde{r},\tilde{\theta}) \approx 1 + \frac{1}{2\sqrt{\pi}} \sum_{q=0}^{N_Q} \omega_q \hat{x}_q^{-1} \Bigg(& 1 - \sum_{j=0}^{\infty} \sum_{m=0}^{\frac{j-p}{2}} \varepsilon^{2j} \tilde{r}^j \hat{x}_q^j \\
& \frac{b_{2m+p} t_{j-2m} r^{2m} \Theta_{2m+p} T_{j-2m}(r)}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!} e^{-\hat{x}_q \varepsilon^2 r^2} e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2} {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \hat{x}_q^2) \Bigg) \\
= 1 + \frac{1}{2\sqrt{\pi}} \sum_{q=0}^{N_Q} \omega_q \hat{x}_q^{-1} & \\
- \frac{1}{2\sqrt{\pi}} \frac{\Gamma(N_Q + 1/2)}{\Gamma(1/2)\ N_Q!\ (N_Q+1)^2} & \sum_{q=0}^{N_Q} \sum_{j=0}^{\infty} \sum_{m=0}^{\frac{j-p}{2}} \varepsilon^{2j} \tilde{r}^j \frac{\hat{x}_q^j}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} \\
& \frac{b_{2m+p} t_{j-2m} r^{2m} \Theta_{2m+p} T_{j-2m}(r)}{2^{j-2m-1} \left(\frac{j+p+2m}{2}\right)! \left(\frac{j-p-2m}{2}\right)!} e^{-\hat{x}_q \varepsilon^2 r^2} e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2} {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \hat{x}_q^2).
\end{aligned}$$

Der Rest folgt mit den Gleichungen (4.2.33) sowie

$$\begin{aligned}
c_{q,j,m}(\tilde{r},\tilde{\theta}) := & \frac{\hat{x}_q^j}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} b_{2m+p} t_{j-2m} \cos((2m+p)\theta_i) \tilde{r}^j e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2} \\
& {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \tilde{x}_q^2), \\
s_{q,j,m}(\tilde{r},\tilde{\theta}) := & \frac{\hat{x}_q^j}{\left(L_{N_Q+1}^{-1/2}(\hat{x}_q)\right)^2} b_{2m+p} t_{j-2m} \sin((2m+p)\theta_i) \tilde{r}^j e^{-\hat{x}_q \varepsilon^2 \tilde{r}^2} \\
& {}_1F_2(\alpha_{j,m}; \beta_{j,m}; \varepsilon^4 \tilde{r}^2 \tilde{x}_q^2),
\end{aligned}$$

und (4.2.36) - (4.2.37). Die Matrixdarstellung erfolgt mit (4.2.40) - (4.2.42) zu obigen Elementen $c_{q,j,m}$ und $s_{q,j,m}$ sowie (4.2.43). Außerdem können wir

$$1 + \frac{1}{2\sqrt{\pi}} \sum_{q=0}^{N_Q} \omega_q \hat{x}_q^{-1}$$

sowie die Faktoren

$$-\frac{1}{2\sqrt{\pi}}\frac{\Gamma(N_Q+\frac{1}{2})}{\Gamma(1/2)N_Q!(N_Q+1)^2}$$

wegen der Unabhängigkeit zu den Interpolations- und Centerpunkte weglassen. Insgesamt gilt

$$\varphi(\|(x,y)-(x_i,y_i)\|_2) \approx C(x_i,y_i)\cdot D\cdot T(\varrho(x,y))^T,\ i=1,\ldots,n, \tag{4.2.68}$$

Wir erhalten wieder mit QR-Zerlegung der Koeffizientenmatrix C für die neuen Basisfunktionen

$$\Psi_{-1/2}(x,y) = \begin{bmatrix} I & \tilde{R} \end{bmatrix}\cdot T(\varrho(x,y))^T. \tag{4.2.69}$$

Die Interpolationsmatrix berechnen wir analog zu (4.2.49), erhalten damit die Koeffizienten λ und können wieder an jeder beliebigen Stelle (x,y) auswerten mit

$$s(x,y) = \Psi(x,y)^T\lambda.$$

Die Berechnung erfolgt wieder über die Programme `RBf_QR_GQ_2D.m` (Quelltext A.4) und `RBF_QR_GQ_quad_part.m` (Quelltext A.5).

4.3 Fazit

Wir haben in diesem Kapitel einen neuen Algorithmus, den RBF-QR-GQ-Algorithmus, entwickelt und für die inverse Multiquadrics, die β-inverse Multiquadrics und die Multiquadrics direkt angegeben. Dabei haben wir ausgenutzt, dass in der Bernstein-Widder-Darstellung (IMQ, β-IMQ) bzw. Integraldarstellung (MQ) der vorgestellten radialen Basisfunktionen der Term

$$e^{-\varepsilon^2 r^2}$$

auftaucht. Solange dies gegeben ist, kann der RBF-QR-GQ-Algorithmus auch für andere vollständig monotone radiale Basisfunktionen verwendet werden.

Kapitel 5

Quasi-Interpolation mit radialen Basisfunktionen

In Kapitel 4 haben wir einen neuen Algorithmus zur Interpolation mit der IMQ, β-IMQ und MQ für $\varepsilon \to \infty$ eingeführt, um die numerischen Berechnungen zu verbessern, insbesondere auch für eine große Anzahl n an Interpolationspunkten. Eine weitere Möglichkeit etwaige numerische Instabilitäten bei großen n zu umgehen, bietet die Quasi-Interpolation. Zwar haben wir nicht mehr den Vorteil, auch für kleine ε gute numerische Ergebnisse zu erhalten, allerdings wird die Approximation für moderate ε gut sein. Wir verwenden nun die Quasi-Interpolante (2.3.27)

$$s_h(x) = \sum_{j \in \mathbb{Z}^d} f(jh)\psi(x/h - j),\ x \in \mathbb{R}^d, \tag{5.0.1}$$

zu einer zu approximierenden Funktion f, einer Gitterfeinheit h und mit

$$\psi(x) = \sum_{k \in \mathbb{Z}^d} c_k \varphi(\|x - k\|),\ x \in \mathbb{R}^d, \tag{5.0.2}$$

für eine radiale Basisfunktion φ und Koeffizienten c_k, $k \in \mathbb{Z}^d$.

Zwar können wir die numerische Instabilität für sehr kleine ε durch eine Vergrößerung des Abstands der Gitterpunkten $k \in Z^d$ von (5.0.2) zueinander abfangen, dieser wächst allerdings um den Faktor $\lceil \varepsilon^{-2} \rceil$ und führt ebenfalls für sehr kleine ε zu numerischen Instabilität. Wir werden insbesondere in diesem Kapitel nicht die gleichen radialen Basisfunktionen verwenden wie in Kapitel 4, sondern bedingt positiv definite Basisfunktionen der Ordnung 2, im Einzelnen die Thin-Plate-Splines (2.1.8) und shifted-Thin-Plate-Splines (2.1.9) sowie (2.1.10). Diese radialen Basisfunktionen ermöglichen uns, für die in der numerischen Berechnung verwendeten Teilmengen des $\mathbb{R}^2$, eine Reproduktion von Polynomen bestimmten Grades.

Für die in Kapitel 4 verwendeten Basisfunktion kann man im Einzelnen zeigen (siehe auch [12, Jackson]), dass

1. die Gaussian keine,
2. die Inverse- und die Multiquadrics nur für ungerade Dimensionen,
3. die verallgemeinerte β-Inverse-Multiquadrics mit

$$\varphi(r) = \left(c^2 + r^2\right)^{-\beta/2},\ c > 0,\ r \in \mathbb{R}^+,$$

   nur für $d + \beta \in 2\mathbb{N}$, $\beta \in \mathbb{N}$.

Polynomreproduktion nach Theorem 2.3.6 ermöglicht.

Insbesondere erhalten wir für die Thin-Plate-Splines (TPS) und shifted-Thin-Plate-Splines (sTPS) keine alternative Darstellung durch die Bernstein-Widder-Darstellung 2.1.3, da diese weder vollständig monoton noch bvm[1] sind.

Für die Verwendung dieser bedingt positiv definiten Funktionen in der Quasi-Interpolation spricht, dass bei Quasi-Interpolation keine zusätzlichen niedriggradigen Polynome erforderlich sind, weil nicht interpoliert wird.

5.1 Koeffizientenbestimmung

Zunächst benötigen wir die Koeffizienten $\{c_k\}_{k=1}^{\ell}$ in

$$\psi(x) = \sum_{k=1}^{\ell} c_k \varphi(\|x - x_k\|),\ x \in \mathbb{R}^d, \tag{5.1.3}$$

zur Basisfunktion φ. Um diese zu berechnen, schreiben wir die verallgemeinerte Fouriertransformation

$$\hat{\psi}(t) = \hat{\varphi}(\|t\|) \sum_{k \in \mathbb{Z}^d} c_k e^{-it\cdot k}, \quad t \in \mathbb{Z}^d, \tag{5.1.4}$$

in der Nähe von $\omega = 0$ um in eine Entwicklung der Form

$$\hat{\varphi}(\omega) = \frac{1}{P_r(\omega) + P_{r+1}(\omega) + \ldots + P_u(\omega)} + h(\omega), \tag{5.1.5}$$

mit $r \geq 1$ und P_s, ein homogenes Polynom vom Grad s, $s = r, \ldots, n$, wobei $P_r \not\equiv 0$ und h eine $(u - 2r + \varepsilon)$-regulär differenzierbare Funktion für $\varepsilon \in (0, 1)$ ist. Bevor wir weitermachen noch folgende Definition.

Definition 5.1.1. *$h : \mathbb{R}^d \mapsto \mathbb{R}$ ist eine b-regulär differenzierbare Funktion für $b \in \mathbb{R}$, falls alle partiellen Ableitungen h von 0 entfernt existieren und falls*

$$\frac{\partial^\alpha h(\omega)}{\partial \omega^\alpha} = O(\|\omega\|^{b-|\alpha|}) \quad \textit{für} \quad \|\omega\| \to \infty,$$

für alle $\alpha \in \mathbb{Z}_+^d$.

Diese Funktionen haben folgende nützliche Eigenschaften:

Proposition 5.1.2. *Angenommen p_s, q_s, $s = 0, 1, 2, \ldots$, sind homogene Polynome vom Grad s. Dann ist*

1. *$p_u(\cdot)(q_r(\cdot))^{-1}$ eine $(u - r)$-reguläre differenzierbare Funktion und*
2. *$\left(\sum_{s=u}^{\infty} p_s(\cdot)\right)\left(\sum_{s=r}^{\infty} q_s(\cdot)\right)^{-1}$ ist ebenfalls $(u - r)$-reguläre differenzierbare Funktion, sofern die Summen in einer Umgebung um 0 konvergieren.*

Sei außerdem g eine u-regulär differenzierbare Funktion und h eine r-regulär differenzierbare Funktion, dann ist

1. *gh eine $(u + r)$-reguläre differenzierbare Funktion und*
2. *$g + h$ ist eine $\min(u, r)$-reguläre differenzierbare Funktion.*

Zudem ist $\log(\|\cdot\|)$ *eine* $(-\delta)$*-regulär differenzierbare Funktion für ein* $\delta > 0$*.*

Für die Koeffizientenbestimmung setzen wir voraus, dass $\varphi : \mathbb{R}^d \mapsto \mathbb{R}$ eine verallgemeinerte Fouriertransformation $\hat{\varphi}(\omega)$ besitzt, die (B1), (B2a) und (B3a) für ein $\mu \in 2\mathbb{Z}_+$ erfüllt. Weiterhin besitze $\hat{\varphi}(\omega)$ in der Nähe von $\omega = 0$ eine Entwicklung der Form (5.1.5). Damit existieren dann $\ell \in \mathbb{N}$, Koeffizienten $\{c_k\}_{k=1}^{\ell}$ und Punkte $\{x_k\}_{k=1}^{\ell} \subset \mathbb{Z}^d$ derart, dass ψ in der Form (5.1.3) benutzt in (2.3.25) alle Polynome vom Grad kleiner oder gleich $\mu - 1$ reproduziert, woraus folgt, dass wir, wie oben bemerkt, keine Polynome zur Approximante addieren müssen.

Mit Theorem 2.3.6 folgt, dass es ein trigonometrisches Polynom g gibt der Form

$$g(t) = \sum_{k=1}^{\ell} c_k e^{-it^T x_k}, \quad t \in \mathbb{Z}^d. \tag{5.1.6}$$

Wir entwickeln nun $e^{-it^T y}$ um $t = 0$

$$e^{-it^T y} = \sum_{s=0}^{\infty} \frac{(-it^T y)^s}{s!}$$

und setzen dies in (5.1.6) ein

$$g(t) = \sum_{k=1}^{\ell} c_k \sum_{s=0}^{\infty} \frac{(-it^T x_k)^s}{s!} = \sum_{s=0}^{\infty} \underbrace{\frac{(-it)^s}{s!} \sum_{k=1}^{\ell} c_k x_k^s}_{=:\tilde{q}_s(t)} = \sum_{s=0}^{\infty} \tilde{q}_s(t),$$

wobei $\tilde{q}_s(t)$ ein homogenes Polynom vom Grad s in t ist. Mit $\alpha \in \mathbb{Z}_+^d : |\alpha| = s$ erhalten wir für die Koeffizienten von t^α in $\tilde{q}_s(t)$

$$(-i)^s \sum_{k=1}^{\ell} \frac{c_k x_k^\alpha}{s!} \frac{(|\alpha|)!}{\alpha!} = \frac{(-i)^{|\alpha|}}{\alpha!} \sum_{k=1}^{\ell} c_k x_k^\alpha \tag{5.1.7}$$

Da die Monome linear unabhängige Funktionen sind, können wir $\ell \in \mathbb{N}$, Koeffizienten $\{c_k\}_{k=1}^{\ell}$ und Punkte $\{x_k\}_{k=1}^{\ell} \subset \mathbb{Z}^d$ wählen, so dass

$$\begin{array}{rcll} \tilde{q}_s(t) & \equiv & 0 & \text{für} \quad s < r, \\ \tilde{q}_s(t) & = & \tilde{P}_s(t) & \text{für} \quad r \le s \le u', \end{array} \tag{5.1.8}$$

mit $u' = \min(2r - 1, u)$ und $\tilde{P}_s$ aus (5.1.5).

Wir bestimmen nun die Polynome $\tilde{q}_s(t)$ aus (5.1.8) und benutzen für die Darstellung

der Fouriertransformationen die Besselfunktionen I_n [1, 9.6.10] und K_n [1, 9.6.11] n-ter Ordnung, $n \in \mathbb{N}_0$,

$$I_n(z) = \left(\frac{1}{2}z\right)^n \sum_{k=0}^{\infty} \frac{\left(\frac{1}{4}z^2\right)^k}{k!(n+k)!}, \tag{5.1.9}$$

$$K_n(z) = \frac{1}{2}\left(\frac{1}{2}z\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!}\left(-\frac{1}{4}z^2\right)^k + (-1)^{n+1}\log\left(\frac{1}{2}z\right) I_n(z)$$
$$+ (-1)^n \frac{1}{2}\left(\frac{1}{2}z\right)^n \sum_{k=0}^{\infty} \{\Psi(k+1) + \Psi(n+k+1)\} \frac{\left(\frac{1}{4}z^2\right)^k}{k!(n+k)!}, \tag{5.1.10}$$

mit der Digamma-Funktion [1, § 6.3]

$$\Psi(z) = \frac{\mathrm{d}}{\mathrm{d}z}\log_2 \Gamma(z). \tag{5.1.11}$$

Dabei sind

- $I_n(z)$ eine (n)-regulär differenzierbare Funktion,
- $h_1(z) := (-1)^{n+1}\log\left(\frac{1}{2}z\right) I_n(z)$ für $\delta > 0$ eine $(-\delta+n)$-regulär differenzierbare und
- $h_2(z) := (-1)^n \frac{1}{2}\left(\frac{1}{2}z\right)^n \sum_{k=0}^{\infty} \{\psi(k+1) + \psi(n+k+1)\} \frac{\left(\frac{1}{4}z^2\right)^k}{k!(n+k)!}$ eine (n)-regulär differenzierbare Funktion.

$h_3(z) := h_1(z) + h_2(z)$ ist somit eine $(-\delta+n)$-regulär differenzierbare Funktion und wir können insgesamt schreiben

$$K_n(z) = \frac{1}{2}\left(\frac{1}{2}z\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!}\left(-\frac{1}{4}z^2\right)^k + h_3(\omega).$$

Aus [12] übernehmen wir noch folgende Gleichung

$$\sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!}(-x)^k = \frac{1}{\sum_{k=0}^{n-1} A_k x^k} + \overline{h}(x), \quad x \in \mathbb{R}^+, \tag{5.1.12}$$

wobei $\overline{h}$ eine (n) - regulär differenzierbare Funktion ist und

$$A_0 = 1,\ A_k = \frac{1}{\left(\frac{b+d}{2}-1\right)!} \sum_{w=1}^{k} (-1)^w \sum_{\{\alpha \in \mathbb{Z}_+^\omega \setminus \{0\} : |\alpha| = k\}} \frac{1}{\alpha!\left(1-\frac{b+d}{2}\right)_\alpha}. \tag{5.1.13}$$

Damit erhalten wir nun für $x \in \mathbb{R}^+$

$$K_n(x) = \frac{1}{2}\left(\frac{1}{2}x\right)^{-n}\left(\frac{1}{\sum_{k=0}^{n-1} A_k \left(2^{-2}x^2\right)^k} + \overline{h}(x)\right) + h_3(x) \tag{5.1.14}$$

$$= \frac{2^{n-1}x^{-n}}{\sum_{k=0}^{n-1} A_k \left(2^{-2}x^2\right)^k} + h_4(x), \tag{5.1.15}$$

mit $h_4(x) := \frac{1}{2}\left(\frac{1}{2}x\right)^{-n}\overline{h}(x) + h_3(x)$ als $(-\delta+n)$-regulär differenzierbare Funktion.

Wenden wir uns nun der Koeffizientenbestimmung von

$$\varphi_c(r) = (r^2+c^2)^{b/2}\log(r^2+c^2)^{1/2},\ b \in \mathbb{R}, \tag{5.1.16}$$

analog zu [12, Jackson], sowie der Koeffizientenbestimmung für

$$\varphi_\varepsilon(r) = (1+r^2\varepsilon^2)^{b/2}\log(1+r^2\varepsilon^2)^{1/2},\ b \in 2\mathbb{Z}_+, \tag{5.1.17}$$

die bisher noch nicht behandelt worden ist, zu.

shifted-Thin-Plate-Splines 1.Art $\varphi_c(r)$ (aus [12])

Wir betrachten zuerst die shifted-Thin-Plate-Splines mit Glättungsparameter $c > 0$:

$$\varphi_c(r) = (r^2+c^2)^{b/2}\log(r^2+c^2)^{1/2},\ b \in \mathbb{R}. \tag{5.1.18}$$

Für $b \notin 2\mathbb{Z}_+$ erhalten wir eine Fouriertransformation, die um $\omega = 0$ nicht in der Form (5.1.5) geschrieben werden kann [12]. Aus diesem Grund beschränken wir uns in der folgenden Betrachtung auf $b \in 2\mathbb{Z}_+$. Eine ausführliche Analyse für den Fall $b \notin \mathbb{Z}_+$ findet sich in [12]. Die Fouriertransformation von (5.1.18) für $b \in 2\mathbb{Z}_+$ ist

$$\hat{\varphi}_c(\|\omega\|) = c^{\frac{b+d}{2}}\mathcal{C}_{b,d}\frac{K_{\frac{b+d}{2}}(c\|\omega\|)}{\|\omega\|^{\frac{b+d}{2}}}, \quad \omega \in \mathbb{R}^d, \tag{5.1.19}$$

mit

$$\mathcal{C}_{b,d} := (-1)^{\frac{b}{2}+1}2^{\frac{b+d}{2}}\left(\frac{b}{2}\right)!\pi^{\frac{d}{2}}.$$

Weiterhin fordern wir $b+d \in 2\mathbb{Z}_+\backslash\{0\}$, da man ansonsten zeigen kann, dass (5.1.19) ebenfalls nicht in der Form (5.1.5) um $\omega = 0$ geschrieben werden kann, da zum einen ([1, 9.7.2])

$$K_\nu(x) \sim \sqrt{\frac{\pi}{2x}}e^{-x} \text{ für} x \to \infty,$$

sowie (C3) erfüllt ist, es damit aber ein $D \in \mathbb{R}$ gibt, so dass um $\omega = 0$ gilt $\hat{\varphi}(\omega) = D + o(1)$ mit $\hat{\varphi}(2\pi\beta) = 0$ für alle $\beta \in \mathbb{Z}^d\backslash\{0\}$ (siehe auch [12]). Wir erhalten in diesem Fall also noch nicht einmal die Reproduktion konstanter Funktionen.

Sei also $b+d \in 2\mathbb{Z}_+\backslash\{0\}$ und $b \in 2\mathbb{Z}_+$. Die Darstellung (5.1.5) gilt für $r = b+d$, $u = 2b+2d-1$ und $0 < \varepsilon < 1$. Um $\{c_j : j = 1, 2, \ldots, \ell\}$ und $\{x_j \in \mathbb{Z}^d : j = 1, 2, \ldots, \ell\}$ zu berechnen, schreiben wir (5.1.19) mit (5.1.15) um in

$$\begin{aligned}\hat{\varphi}_c(\|\omega\|) &= \frac{c^{\frac{b+d}{2}}\mathcal{C}_{b,d}}{\|\omega\|^{\frac{b+d}{2}}}\left(\frac{2^{\frac{b+d}{2}-1}\left(c\|\omega\|\right)^{-\frac{b+d}{2}}}{\sum_{k=0}^{\frac{b+d}{2}-1} A_k\left(2^{-2}c^2\|\omega\|^2\right)^k} + h_4(c\|\omega\|)\right)\\ &= \frac{1}{\sum_{k=0}^{\frac{b+d}{2}-1} A_k\left(2^{-2}c^2\|\omega\|^2\right)^k \|\omega\|^{b+d}2^{-\frac{b+d}{2}+1}\mathcal{C}_{b,d}^{-1}} + h_5(c\|\omega\|).\end{aligned}$$

Dabei ist $h_5(x) = x^{-\frac{b+d}{2}}\mathcal{C}_{b,d}h_4(x)$ eine $(-\delta)$-regulär differenzierbare Funktion. Mit

$$\tilde{\mathcal{C}}_{b,d} := 2^{\frac{b+d}{2}-1}\mathcal{C}_{b,d} \tag{5.1.20}$$

erhalten wir

$$\hat{\varphi}_c(\omega) = \frac{1}{\sum_{k=0}^{\frac{b+d}{2}-1} 2^{-2k}A_k c^{2k}\|\omega\|^{b+d+2k}\tilde{\mathcal{C}}_{b,d}^{-1}} + h_5(c\|\omega\|). \tag{5.1.21}$$

Der Nenner von (5.1.21) ist die Summe von Polynomen in $\|\omega\|$ mit Grad $b+d$, $b+d+2$, …, $2b+2d-2 \le b+d+m$ für $m = b+d-1$. Beachten wir noch, dass für $k = \frac{b+d}{2}-1$ der Grad $s = 2b+2d-2$ gilt, dann ist

$$k = \frac{b+d}{2} - 1 = \frac{b+d-2}{2} = \frac{2b+2d-2-b-d}{2} = \frac{s-b-d}{2}. \tag{5.1.22}$$

Damit erhalten wir unter Berücksichtigung von (5.1.8)

$$\tilde{q}_s(\omega) = \begin{cases}\frac{A_{\frac{s-b-d}{2}}c^{s-b-d}2^{b+d}}{2^s\tilde{\mathcal{C}}_{b,d}}\|\omega\|^s, & \text{für } s \text{ gerade und } b+d \le s \le b+d+m,\\ 0, & \text{für alle anderen } s \le b+d+m.\end{cases}$$

Da (5.1.7) $\tilde{q}_s(\omega) = B\|\omega\|^s$, $B \in \mathbb{R}$, für s gerade genau dann, wenn $\{c_j : j = 1, 2, \ldots, \ell\}$ und $\{x_j \in \mathbb{Z}^d : j = 1, 2, \ldots, \ell\}$ folgendes erfüllen [12, (4.3.8)]

$$\sum_{j=1}^{\ell} c_j x_j^{\alpha} = \begin{cases}(-1)^{\frac{s}{2}}B\left(\frac{s}{2}\right)!\frac{\alpha!}{\left(\frac{\alpha}{2}\right)!}, & \text{falls } \alpha \in \mathbb{Z}_+^d \text{ komponentenweise gerade,}\\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^d : |\alpha| = s,\end{cases}$$

dann folgt insgesamt

$$\sum_{j=1}^{\ell} c_j x_j^{\alpha}$$

$$= \begin{cases} \frac{(-1)^{\frac{|\alpha|}{2}} 2^{b+d} A_{\frac{|\alpha|-b-d}{2}} c^{|\alpha|-b-d} \left(\frac{|\alpha|}{2}\right)!}{2^{|\alpha|} \tilde{\mathcal{C}}_{b,d}} \frac{\alpha!}{\left(\frac{\alpha}{2}\right)!}, & \text{falls } b+d \leq |\alpha| \leq b+d+m, \\ & \alpha \in \mathbb{Z}_+^d \text{ komponentenweise gerade}, \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^d : |\alpha| \leq b+d+m. \end{cases} \tag{5.1.23}$$

Beispiel 4:
Sei $b = d = 2$, dann erhalten wir eine maximale Reproduktion für Polynome vom Grad $b + d - 1 = 3$.

- **Lineare Polynome:** Für $m = 1$ folgt mit (5.1.23)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} \frac{\alpha!}{4\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 4 \text{ und } \alpha \text{ komponentenweise gerade}, \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| \leq 5. \end{cases} \tag{5.1.24}$$

- **Quadratische Polynome:** Für $m = 2$ benötigen wir zusätzlich zu (5.1.24)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} \frac{-3c^2\alpha!}{8\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 6 \text{ und } \alpha \text{ komponentenweise gerade}, \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 6. \end{cases} \tag{5.1.25}$$

- **Kubische Polynome:** Für $m = 3$ benötigen wir zusätzlich zu (5.1.24) und (5.1.25)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = 0, \text{ für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 7. \tag{5.1.26}$$

shifted-Thin-Plate-Splines 2.Art $\varphi_\varepsilon(r)$

Die shifted-Thin-Plate-Splines mit Glättungsparameter $\varepsilon > 0$ werden in [12] nicht behandelt. Da wir aber in unseren Anwendungen glatte radiale Basisfunktion betrachten, holen wir dies nach und liefern im Anschluss die Berechnung der Koeffizienten von (5.1.3) für lineare, quadratische und kubische Polynomreproduktion. Mit

$$\varphi_\varepsilon(r) = (1 + r^2\varepsilon^2)^{b/2} \log(1 + r^2\varepsilon^2)^{1/2}, \; b \in 2\mathbb{Z}_+, \tag{5.1.27}$$

formen wir zuerst den Ausdruck (5.1.18) um mit $\varepsilon := \frac{1}{c}$

$$\begin{aligned}
\varphi_c(r) &= \left(r^2 + c^2\right)^{b/2} \log\left(r^2 + c^2\right)^{1/2} \\
&= c^b \left(1 + \varepsilon^2 r^2\right)^{b/2} \left(\log(c) + \log\left(1 + \varepsilon^2 r^2\right)^{\frac{1}{2}}\right) \\
&= c^b \log(c) \left(1 + \varepsilon^2 r^2\right)^{b/2} + c^b \left(1 + \varepsilon^2 r^2\right)^{b/2} \log\left(1 + \varepsilon^2 r^2\right)^{\frac{1}{2}} \\
&\Longleftrightarrow \\
\varphi_\varepsilon(r) &= \varepsilon^b \varphi_c(r) - \log(c) \left(1 + \varepsilon^2 r^2\right)^{\frac{b}{2}}. \qquad (5.1.28)
\end{aligned}$$

Damit ermitteln wir nun die verallgemeinerte Fouriertransformation von (5.1.27) mit $\Phi_\varepsilon(\omega) := \varphi_\varepsilon(\|\omega\|)$ und $\Phi_c(\omega) := \varphi_c(\|\omega\|)$

$$\begin{aligned}
\int_{\mathbb{R}^d} \Phi_\varepsilon(\omega)\hat{\gamma}(\omega)\,\mathrm{d}\omega &= \int_{\mathbb{R}^d} \left(\varepsilon^b \Phi_c(\omega) - \log(c)\left(1 + \varepsilon^2\omega^2\right)^{\frac{b}{2}}\right)\hat{\gamma}(\omega)\,\mathrm{d}\omega \\
&= \varepsilon^b \int_{\mathbb{R}^d} \Phi_c(\omega)\hat{\gamma}(\omega)\,\mathrm{d}\omega - \log(c) \underbrace{\int_{\mathbb{R}^d} \left(1 + \varepsilon^2\omega^2\right)^{\frac{b}{2}} \hat{\gamma}(\omega)\,\mathrm{d}\omega}_{=0} \\
&\Longleftrightarrow \qquad (5.1.29) \\
\int_{\mathbb{R}^d} \Phi_\varepsilon(\omega)\hat{\gamma}(\omega)\,\mathrm{d}\omega &= \int_{\mathbb{R}^d} \underbrace{\varepsilon^b \hat{\Phi}_c(\omega)}_{=\hat{\Phi}_\varepsilon(\omega)} \gamma(\omega)\,\mathrm{d}\omega. \qquad (5.1.30)
\end{aligned}$$

Die letzte Gleichung folgt, da $(1 + \varepsilon^2\omega^2)^{\frac{b}{2}}$ für $b \in 2\mathbb{Z}_+$ ein Polynom p ist und für diese gelten [24]

$$\int_{\mathbb{R}^d} p(\omega)\hat{\gamma}(\omega)\,\mathrm{d}\omega = 0.$$

Wir erhalten insgesamt

$$\hat{\varphi}_\varepsilon(\|\omega\|) = \varepsilon^{\frac{b-d}{2}} \mathcal{C}_{b,d} \frac{K_{\frac{b+d}{2}}\left(\varepsilon^{-1}\|\omega\|\right)}{\|\omega\|^{\frac{b+d}{2}}}, \quad \omega \in \mathbb{R}^d. \qquad (5.1.31)$$

Hier wird die Einschränkung auf $b \in 2\mathbb{Z}_+$ deutlich, da wir sonst analog zu (5.1.18) die verallgemeinerte Fouriertransformation nicht in der Form (5.1.5) schreiben können. Es folgt mit (5.1.20)

$$\hat{\varphi}_\varepsilon(\omega) = \frac{1}{\sum_{k=0}^{\frac{b+d}{2}-1} 2^{-2k} A_k \varepsilon^{-(b+2k)} \|\omega\|^{b+d+2k} \tilde{\mathcal{C}}_{b,d}} + h_5(\varepsilon^{-1}\|\omega\|).$$

Der Nenner im letzten Term ist wieder die Summe von Polynomen in $\|\omega\|$ mit Grad $b+d$, $b+d+2$, …, $2b+2d-2$. Beachten wir noch (5.1.22), dann erhalten wir mit

(5.1.8)

$$\tilde{q}_s(\omega) = \begin{cases} \frac{A_{\frac{s-b-d}{2}} \varepsilon^d 2^{b+d}}{\varepsilon^s 2^s \tilde{\mathcal{C}}_{b,d}} \|\omega\|^s, & \text{für } s \text{ gerade und } b+d \leq s \leq b+d+m, \\ 0, & \text{für alle anderen } s \leq b+d+m. \end{cases}$$

Wegen (5.1.7) ist $\tilde{q}_s(\omega) = B\|\omega\|^s$, $B \in \mathbb{R}$, für s gerade genau dann, wenn $\{\mu_j : j = 1, 2, \ldots, \ell\}$ und $\{x_j \in \mathbb{Z}^d : j = 1, 2, \ldots, \ell\}$ folgendes erfüllen [12, 4.3.8]

$$\sum_{j=1}^{\ell} \mu_j x_j^\alpha = \begin{cases} (-1)^{\frac{s}{2}} B \left(\frac{s}{2}\right)! \frac{\alpha!}{(\frac{\alpha}{2})!}, & \text{falls } \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^d : |\alpha| = s. \end{cases}$$

Insgesamt

$$\begin{aligned} &\sum_{j=1}^{\ell} \mu_j x_j^\alpha \\ &= \begin{cases} \frac{(-1)^{\frac{|\alpha|}{2}} A_{\frac{|\alpha|-b-d}{2}} \varepsilon^d 2^{b+d} \left(\frac{|\alpha|}{2}\right)!}{\varepsilon^{|\alpha|} 2^{|\alpha|} \tilde{\mathcal{C}}_{b,d}} \frac{\alpha!}{(\frac{\alpha}{2})!}, & \text{falls } b+d \leq |\alpha| \leq b+d+m, \\ & \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen} \\ & \alpha \in \mathbb{Z}_+^d : |\alpha| \leq b+d+m. \end{cases} \end{aligned} \tag{5.1.32}$$

Beispiel 5:
Sei wieder $b = d = 2$, dann erhalten wir eine maximale Reproduktion für Polynome vom Grad $b + d - 1 = 3$.

- **Lineare Polynome:** Für $m = 1$ folgt mit (5.1.32)

$$\sum_{j=1}^{\ell} \mu_j x_j^\alpha = \begin{cases} \frac{\alpha!}{4\varepsilon^2 \pi (\frac{\alpha}{2})!}, & \text{falls } |\alpha| = 4 \text{ und } \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| \leq 5. \end{cases} \tag{5.1.33}$$

- **Quadratische Polynome:** Für $m = 2$ folgt mit (5.1.32) zusätzlich zu (5.1.33)

$$\sum_{j=1}^{\ell} \mu_j x_j^\alpha = \begin{cases} -\frac{3\alpha!}{8\varepsilon^4 \pi (\frac{\alpha}{2})!}, & \text{falls } |\alpha| = 6 \text{ und } \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 6. \end{cases} \tag{5.1.34}$$

- **Kubische Polynome:** Für $m = 3$ benötigen wir zusätzlich

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = 0 \text{ für alle } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 7.$$

5.2 Quasi-Interpolation

Wir benutzen die Approximante (2.3.27) mit eingeschränktem Gitter für Punkte aus $\Omega \subset \mathbb{R}^2$

$$s_h(x) = \sum_{j \in \Omega \cap \mathbb{Z}^2} f(jh)\psi(x/h - j),\ x \in \Omega. \tag{5.2.35}$$

Dabei gilt analog zu (2.3.26)

$$\psi(x) = \sum_{k \in P} c_k \varphi(\|x - k\|),\ x \in \Omega, \tag{5.2.36}$$

für eine Punktemenge $P \subset \mathbb{Z}^2$ und entsprechenden Koeffizienten c_k, $k \in P$, zur Basisfunktion φ.

Da wir wieder Punkte aus $\mathbb{R}^2$ betrachten, sind wir in der Quasi-Interpolation auf die Thin-Plate-Splines und shifted-Thin-Plate-Splines der 1. (5.1.18) und 2. Art (5.1.27) beschränkt.

5.2.1 shifted-Thin-Plate-Splines 1. Art

Nach Kapitel 2.3 können wir die Koeffizienten c_k von (5.1.18)

$$\varphi_c(r) = (r^2 + c^2)^{b/2} \log(r^2 + c^2)^{1/2},\ b \in 2\mathbb{Z}_+, \tag{5.2.37}$$

zur Punktemenge P mit folgender Formel (5.1.23) bestimmen

$$\begin{aligned} &\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} \\ &= \begin{cases} \frac{(-1)^{\frac{|\alpha|}{2}} 2^{b+d} A_{\frac{|\alpha|-b-d}{2}} c^{|\alpha|-b-d} \left(\frac{|\alpha|}{2}\right)!}{2^{|\alpha|} \tilde{\mathcal{C}}_{b,d}} \frac{\alpha!}{\left(\frac{\alpha}{2}\right)!}, & \text{falls } b+d \le |\alpha| \le b+d+m, \\ & \alpha \text{ komponentenweise gerade}, \\ 0, & \text{für alle anderen} \\ & \alpha \in \mathbb{Z}_+^d : |\alpha| \le b+d+m. \end{cases} \end{aligned} \tag{5.2.38}$$

Für $b = 2$ und zweidimensionale Punkte erhalten wir maximale Reproduktion für Polynome vom Grad $b + d - 1 = 3$ und mit Beispiel 4 die Bedingungen

- **Lineare Polynome:** Für $m = 1$ folgt mit (5.2.38)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} \frac{\alpha!}{4\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 4 \text{ und } \alpha \text{ gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| \leq 5. \end{cases} \tag{5.2.39}$$

- **Quadratische Polynome:** Für $m = 2$ benötigen wir zusätzlich zu (5.2.39)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} \frac{-3c^2\alpha!}{8\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 6 \text{ und } \alpha \text{ gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 6. \end{cases} \tag{5.2.40}$$

- **Kubische Polynome:** Für $m = 3$ benötigen wir zusätzlich zu (5.2.39) und (5.2.40)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = 0, \text{ für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 7. \tag{5.2.41}$$

Wir halten fest, dass der Nullpunkt in P enthalten sein muss, da sonst die Matrix des linearen Gleichungssystem zur Bestimmung der Koeffizienten aufgrund $\alpha_1 = (0,0)^T$ eine Spalte mit Nullen besitzt und das Gleichungssystem nicht lösbar ist. Für die Punkte

$$P_1 := \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \pm 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 1 \end{pmatrix}, \begin{pmatrix} \pm 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 2 \end{pmatrix}, \begin{pmatrix} \pm 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \pm 3 \end{pmatrix}, \begin{pmatrix} \pm 1 \\ \pm 1 \end{pmatrix}, \begin{pmatrix} \pm 2 \\ \pm 2 \end{pmatrix} \right\} \tag{5.2.42}$$

und kubischer Polynomreproduktion erhalten wir nun aus Symmetriegründen folgende Gleichungen

$$c_{\binom{1}{0}} = c_{\binom{-1}{0}} = c_{\binom{0}{1}} = c_{\binom{0}{-1}}, \tag{5.2.43a}$$

$$c_{\binom{2}{0}} = c_{\binom{-2}{0}} = c_{\binom{0}{2}} = c_{\binom{0}{-2}}, \tag{5.2.43b}$$

$$c_{\binom{3}{0}} = c_{\binom{-3}{0}} = c_{\binom{0}{3}} = c_{\binom{0}{-3}}, \tag{5.2.43c}$$

$$c_{\binom{1}{1}} = c_{\binom{-1}{1}} = c_{\binom{-1}{1}} = c_{\binom{-1}{-1}}, \tag{5.2.43d}$$

$$c_{\binom{2}{2}} = c_{\binom{-2}{2}} = c_{\binom{-2}{2}} = c_{\binom{-2}{-2}}, \tag{5.2.43e}$$

sowie

$$\begin{pmatrix} 1 & 4 & 4 & 4 & 4 & 4 \\ 0 & 2 & 8 & 18 & 4 & 16 \\ 0 & 2 & 32 & 162 & 4 & 64 \\ 0 & 0 & 0 & 0 & 4 & 64 \\ 0 & 2 & 128 & 1458 & 4 & 256 \\ 0 & 0 & 0 & 0 & 4 & 256 \end{pmatrix} \begin{pmatrix} c_{\binom{0}{0}} \\ c_{\binom{1}{0}} \\ c_{\binom{2}{0}} \\ c_{\binom{3}{0}} \\ c_{\binom{1}{1}} \\ c_{\binom{2}{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \frac{3}{\pi} \\ \frac{1}{\pi} \\ -\frac{45c^2}{2\pi} \\ -\frac{9c^2}{2\pi} \end{pmatrix}. \tag{5.2.44}$$

Als Koeffizienten erhalten wir

$$c_{\binom{0}{0}} = \frac{5(70+51c^2)}{96\pi}, \tag{5.2.45a}$$

$$c_{\binom{1}{0}} = c_{\binom{-1}{0}} = c_{\binom{0}{1}} = c_{\binom{0}{-1}} = -\frac{142+117c^2}{96\pi}, \tag{5.2.45b}$$

$$c_{\binom{2}{0}} = c_{\binom{-2}{0}} = c_{\binom{0}{2}} = c_{\binom{0}{-2}} = \frac{5(10+9c^2)}{192\pi}, \tag{5.2.45c}$$

$$c_{\binom{3}{0}} = c_{\binom{-3}{0}} = c_{\binom{0}{3}} = c_{\binom{0}{-3}} = -\frac{2+3c^2}{96\pi}, \tag{5.2.45d}$$

$$c_{\binom{1}{1}} = c_{\binom{-1}{1}} = c_{\binom{1}{-1}} = c_{\binom{-1}{-1}} = \frac{8+9c^2}{24\pi}, \tag{5.2.45e}$$

$$c_{\binom{2}{2}} = c_{\binom{-2}{2}} = c_{\binom{2}{-2}} = c_{\binom{-2}{-2}} = -\frac{2+9c^2}{384\pi}. \tag{5.2.45f}$$

Die gleichen Bedingungen (5.2.39) - (5.2.41) und Punkte P_1 (5.2.42) werden auch in [12] verwendet, allerdings mit anderem Ergebnis. Die hier berechneten Koeffizienten erfüllen, im Gegensatz zu den in [12, Jackson], auch die Bedingung $\sum_{k\in P} c_k = 0$.

Mit (5.2.36) folgt

$$\psi_c(x) = \sum_{k\in P_1} c_k \varphi_c(\|x-k\|),\ x \in \Omega, \tag{5.2.46}$$

und insgesamt

$$s_h(x) = \sum_{j\in\Omega} f(jh)\psi(x/h-j) = \sum_{j\in\Omega} f(jh) \sum_{k\in P_1} c_k \varphi_c(\|x/h-j-k\|). \tag{5.2.47}$$

Abbildung 5.1 zeigt die Interpolante $\psi_c(x)$ für einige ausgewählte c. Insbesondere werden die Interpolanten für große c zu der gewählten Punkteverteilung P_1 (5.2.42)

instabil. Dies folgt aus dem Gleichungssystem zur Berechnung der Koeffizienten (5.2.44), das für große c entsprechend schlecht konditioniert ist. Zwar kann dies durch eine Vergrößerung der Punkte um den Faktor c^2 abgefangen werden, allerdings ohne Verbesserung der Ergebnisse in der numerischen Berechnung im Vergleich zum c-Wert des kleinsten Fehlers. Für $c \to 0$ hingegen erhalten wir als Grenzwerte die Koeffizienten der Thin-Plate-Splines $\varphi(r) = r^2 \log(r)$ (siehe auch [12])

$$c_{\binom{0}{0}} = \frac{175}{48\pi}, \tag{5.2.48a}$$

$$c_{\binom{1}{0}} = c_{\binom{-1}{0}} = c_{\binom{0}{1}} = c_{\binom{0}{-1}} = -\frac{71}{48\pi}, \tag{5.2.48b}$$

$$c_{\binom{2}{0}} = c_{\binom{-2}{0}} = c_{\binom{0}{2}} = c_{\binom{0}{-2}} = \frac{25}{96\pi}, \tag{5.2.48c}$$

$$c_{\binom{3}{0}} = c_{\binom{-3}{0}} = c_{\binom{0}{3}} = c_{\binom{0}{-3}} = -\frac{1}{48\pi}, \tag{5.2.48d}$$

$$c_{\binom{1}{1}} = c_{\binom{-1}{1}} = c_{\binom{1}{-1}} = c_{\binom{-1}{-1}} = \frac{1}{3\pi}, \tag{5.2.48e}$$

$$c_{\binom{2}{2}} = c_{\binom{-2}{2}} = c_{\binom{2}{-2}} = c_{\binom{-2}{-2}} = -\frac{1}{192\pi}. \tag{5.2.48f}$$

Alternativ zur exakten Berechnung ist in Programm `RBF_QI_TPS_1_Koeff.m` (Quelltext A.10) die numerische Berechnung der Koeffizienten für (5.2.37) mit $b \in 2\mathbb{Z}_+$ zu einer beliebigen Punktemenge P umgesetzt.

5.2.2 shifted-Thin-Plate-Splines 2. Art

Für die Koeffizienten c_k von (5.1.27)

$$\varphi_\varepsilon(r) = (1 + r^2\varepsilon^2)^{b/2} \log(1 + r^2\varepsilon^2)^{1/2},\ b \in 2\mathbb{Z}_+, \tag{5.2.49}$$

zur Punktemenge P_1 (5.2.42) benutzen wir die Formel (5.1.32)

$$\begin{aligned}
&\sum_{j=1}^{\ell} \mu_j x_j^\alpha \\
&= \begin{cases}
\dfrac{(-1)^{\frac{|\alpha|}{2}} A_{\frac{|\alpha|-b-d}{2}} \varepsilon^d 2^{b+d} \left(\frac{|\alpha|}{2}\right)!}{\varepsilon^{|\alpha|} 2^{|\alpha|} \tilde{\mathcal{C}}_{b,d}} \dfrac{\alpha!}{(\frac{\alpha}{2})!}, & \text{falls } b+d \le |\alpha| \le b+d+m, \\
& \alpha \text{ komponentenweise gerade,} \\
0, & \text{für alle anderen} \\
& \alpha \in \mathbb{Z}_+^d : |\alpha| \le b+d+m.
\end{cases}
\end{aligned} \tag{5.2.50}$$

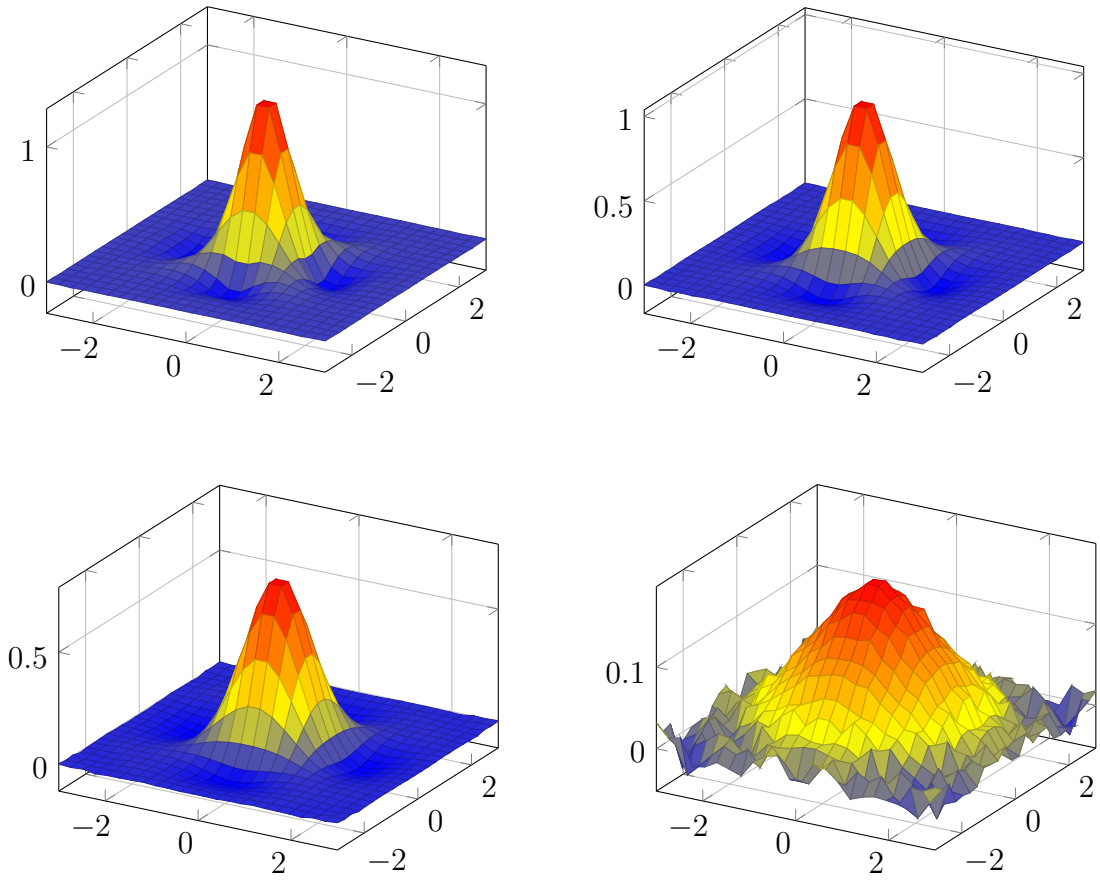

Abbildung 5.1: Quasi-Interpolant der shifted-Thin-Plate-Splines 1. Art (5.2.46) für $c = 10^{-6}, 0.5, 1$ und 4.

Für $b = 2$ und zweidimensionale Punkte folgt wieder die Reproduktion von Polynome vom maximalen Grad 3

- **Lineare Polynome:** Für $m = 1$ folgt mit (5.1.32)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} \frac{\alpha!}{4\varepsilon^2\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 4 \text{ und} \\ & \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| \leq 5. \end{cases} \tag{5.2.51}$$

- **Quadratische Polynome:** Für $m = 2$ folgt mit (5.1.32) zusätzlich zu (5.2.51)

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = \begin{cases} -\frac{3\alpha!}{8\varepsilon^4\pi\left(\frac{\alpha}{2}\right)!}, & \text{falls } |\alpha| = 6 \text{ und} \\ & \alpha \text{ komponentenweise gerade,} \\ 0, & \text{für alle anderen } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 6. \end{cases} \tag{5.2.52}$$

- **Kubische Polynome:** Für $m = 3$ benötigen wir zusätzlich

$$\sum_{j=1}^{\ell} \mu_j x_j^{\alpha} = 0 \text{ für alle } \alpha \in \mathbb{Z}_+^2 : |\alpha| = 7. \tag{5.2.53}$$

Dabei gelten wieder die gleichen Einschränkungen an die Punkte, wie bei den shifted-Thin-Plate-Splines der 1. Art. Für die Punkte P_1 (5.2.42) und kubischer Polynomreproduktion, erhalten wir wieder die Gleichungen (5.2.43), sowie

$$\begin{pmatrix} 1 & 4 & 4 & 4 & 4 & 4 \\ 0 & 2 & 8 & 18 & 4 & 16 \\ 0 & 2 & 32 & 162 & 4 & 64 \\ 0 & 0 & 0 & 0 & 4 & 64 \\ 0 & 2 & 128 & 1458 & 4 & 256 \\ 0 & 0 & 0 & 0 & 4 & 256 \end{pmatrix} \begin{pmatrix} c_{\binom{0}{0}} \\ c_{\binom{1}{0}} \\ c_{\binom{2}{0}} \\ c_{\binom{3}{0}} \\ c_{\binom{1}{1}} \\ c_{\binom{2}{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \frac{3}{\varepsilon^2\pi} \\ \frac{1}{\varepsilon^2\pi} \\ -\frac{45}{\varepsilon^4\pi} \\ -\frac{9}{\varepsilon^4\pi} \end{pmatrix}.$$

Als Koeffizienten erhalten wir

$$c_{\binom{0}{0}} = \frac{255 + 175\varepsilon^2}{48\varepsilon^4\pi}, \tag{5.2.54a}$$

$$c_{\binom{1}{0}} = c_{\binom{-1}{0}} = c_{\binom{0}{1}} = c_{\binom{0}{-1}} = -\frac{11 + 71\varepsilon^2}{48\varepsilon^4\pi}, \tag{5.2.54b}$$

$$c_{\binom{2}{0}} = c_{\binom{-2}{0}} = c_{\binom{0}{2}} = c_{\binom{0}{-2}} = \frac{5(9+5\varepsilon^2)}{96\varepsilon^4\pi}, \tag{5.2.54c}$$

$$c_{\binom{3}{0}} = c_{\binom{-3}{0}} = c_{\binom{0}{3}} = c_{\binom{0}{-3}} = -\frac{3+\varepsilon^2}{48\varepsilon^4\pi}, \tag{5.2.54d}$$

$$c_{\binom{1}{1}} = c_{\binom{-1}{1}} = c_{\binom{1}{-1}} = c_{\binom{-1}{-1}} = \frac{9+4\varepsilon^2}{12\varepsilon^4\pi}, \tag{5.2.54e}$$

$$c_{\binom{2}{2}} = c_{\binom{-2}{2}} = c_{\binom{2}{-2}} = c_{\binom{-2}{-2}} = -\frac{9+\varepsilon^2}{192\varepsilon^4\pi}. \tag{5.2.54f}$$

Mit (5.2.36) folgt

$$\psi_\varepsilon(x) = \sum_{k\in P_1} c_k \varphi_\varepsilon(\|x-k\|),\ x\in\Omega, \tag{5.2.55}$$

und insgesamt

$$s_h(x) = \sum_{j\in\Omega} f(jh)\psi(x/h-j) = \sum_{j\in\Omega} f(jh) \sum_{k\in P_1} c_k \varphi_\varepsilon(\|x/h-j-k\|). \tag{5.2.56}$$

Um den Zusammenhang zwischen den Koeffizienten von φ_c (5.2.45) und φ_ε (5.2.54) zu verdeutlichen betrachten wir deren Fouriertransformationen $\hat{\varphi}_c(\|\omega\|)$ bzw. $\hat{\varphi}_\varepsilon(\|\omega\|)$, die Gleichungen (5.1.28) und (5.1.30) sowie die Fouriertransformation (5.1.4) der entsprechenden Quasi-Interpolante (5.1.3). Es gilt dann mit $\varepsilon = \frac{1}{c}$

$$\hat{\varphi}_c(\|\omega\|) = \varepsilon^2 \hat{\varphi}_\varepsilon(\|\omega\|),\ \omega\in\mathbb{R}^d.$$

Koeffizientenvergleich der Fouriertransformationen der Quasi-Interpolante (5.1.4) mit (5.1.30) liefert

$$\lim_{c\to 0} c_{c,k} = \lim_{\varepsilon\to\infty} \varepsilon^2 c_{\varepsilon,k},\ k\in P_1.$$

Beispiel 6: Für $\binom{2}{2} \in P_1$ erhalten wir

$$\lim_{c\to 0} c_{c,\binom{2}{2}} = \lim_{c\to 0} -\frac{2+9c^2}{384\pi} = -\frac{1}{192\pi} \tag{5.2.57}$$

sowie

$$\lim_{\varepsilon\to\infty} c_{\varepsilon,\binom{2}{2}} = \lim_{\varepsilon\to\infty} -\frac{1+\frac{9}{\varepsilon^2}}{192\pi} = -\frac{1}{192\pi}. \tag{5.2.58}$$

Dies ist insbesondere der Koeffizient der Quasi-Interpolante des Thin-Plate-Splines $\varphi(r) = r^2\log(r)$ zu P_1 für den Punkt $\binom{2}{2}$ (vgl. (5.2.48)).

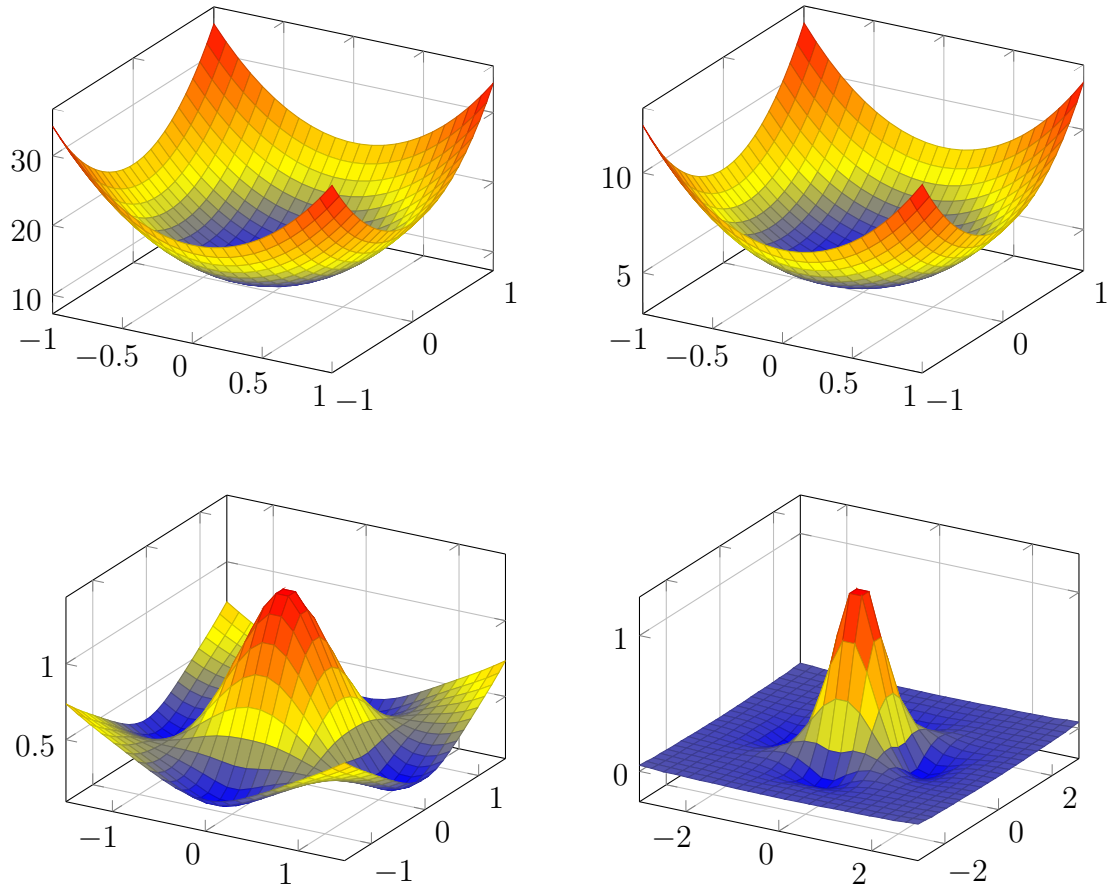

Abbildung 5.2: Quasi-Interpolant der shifted-Thin-Plate-Splines der 2. Art (5.2.55) für $\varepsilon = 0.5, 1, 10$ und 100.

Abbildung 5.2 zeigt die Interpolante $\psi_\varepsilon(x)$ für einige ausgewählte ε. Analog zu 5.2.47 werden hier die Interpolante für kleine ε zur gewählten Punkteverteilung P_1 (5.2.42) instabil. Dies folgt wieder aus dem Gleichungssystem zur Berechnung der Koeffizienten (5.1.32), das für kleine ε entsprechend schlecht konditioniert ist. Zwar kann dies wieder durch eine Vergrößerung der Punkte abgefangen werden, allerdings ebenfalls ohne Verbesserung der Ergebnisse in der numerischen Berechnung, im Vergleich zum ε-Wert des kleinsten Fehlers. Für $\varepsilon \to \infty$ hingegen, erhalten wir als Grenzwerte wieder die Koeffizienten (5.2.48) der Thin-Plate-Splines $\varphi(r) = r^2 \log(r)$. Alternativ zur exakten Berechnung ist in Programm `RBF_QI_TPS_2_Koeff.m` (Quelltext A.11) die numerische Berechnung der Koeffizienten für (5.2.49) mit $b \in 2\mathbb{Z}_+$ zu einer beliebigen Punktemenge P umgesetzt.

Zusammengefasst gehen wir nach folgendem Algorithmus vor.

Algorithmus 2 (RBF-QI)**.**

0. Gegeben:

   - Radiale Basisfunktion φ (TPS, sTPS 1. oder 2. Art) und Punktemenge P,
   - Feinheit h und Randerweiterung ω sowie
   - Auswertungspunkte (x, y) aus $G(1, 0)$, wobei $G(r, s) = B(0, r + s)_2$ oder $G(r, s) = [0-s, r+s]^2$, und zu approximierende Funktion $f(x) : G(1, h\omega) \mapsto \mathbb{R}$.

1. Bestimme Koeffizienten μ zu φ und P.

2. Bilde Gitter

$$\Omega_h(\omega) := \{j \in \mathbb{Z}^2 | jh \in G(1, h\omega)\}.$$

3. Interpoliere: Für alle $(x, y) \in G(1, 0)$ berechne

$$y = \sum_{j \in \Omega_h(\omega)} f(jh) \sum_{k \in P} \mu_k \varphi(\|x/h - j - k\|_2).$$

Der Algorithmus ist in den Programmen `RBF_QI_TPS_1.m` (Quelltext A.8) für die shifted-Thin-Plate-Splines 1. Art und `RBF_QI_TPS_2.m` (Quelltext A.9) für die shifted-Thin-Plate-Splines 2. Art.

Kapitel 6

Numerische Ergebnisse

In diesem Kapitel werden wir die Ergebnisse aus den Kapiteln 3 - 5 numerisch auswerten. Dazu werden wir zunächst das Interpolationsproblem aufstellen und das zugrunde liegende Gebiet definieren. Die Quelltext zu den verwendeten Programme sind in Anhang A zu finden. Diese sind für GNU Octave, version 3.6.3, optimiert. Für die Berechnung wurde ein Notebook mit Intel i5-2430M und 2.40GHz Taktfrequenz unter Ubuntu 12.10 LTS "Quantal Quetzal" verwendet.

Im ersten Abschnitt betrachten wir den direkten Interpolationsalgorithmus der Dagumfunktionen für ausgewählte Parameterwerte β und γ, welche die Nichtsingularität der Interpolationsmatrizen garantieren.

Im Anschluss werden wir den RBF-QR-GQ-Algorithmus für die inverse Multiquadrics, β-inverse Multiquadrics und Multiquadrics mit dem direkten Interpolationsalgorithmus vergleichen.

Zuletzt werden wir noch die Quasi-Interpolations mit dem RBF-QI-Algorithmus für die shiftes-Thin-Plate-Splines 1. und 2. Art verwenden und die Approximationsfehler vergleichen.

6.1 Interpolationsproblem

Zum besseren Vergleich und Visualisierung der numerischen Ergebnisse aus den vorherigen Kapiteln betrachten wir den Einheitskreis $B(0,1)_2 \subset \mathbb{R}^2$ und das Quadrat $[0,1]^2 \subset \mathbb{R}^2$. Für die Verteilung der Interpolationspunkte in $B(0,1)_2$ werden wir die x-y-Projektion der dreidimensionalen Eigenwert-Punkte aus [19] verwenden. Dazu werden die Punkte x_i, $i = 1, \ldots, n$, so gewählt, dass der kleinste Eigenwert der Matrix

$$G(x_i, x_j) := \sum_{\ell=0}^{n} \sum_{k=1}^{2\ell+1} Y_{\ell,k}(x_i) Y_{\ell,k}(x_j)$$

maximiert wird. Dabei bezeichne $Y_{\ell,k}$, $\ell = 0, \ldots, n$ und $k = 1, \ldots, 2+1$ die Kugelflächenfunktionen im $\mathbb{R}^3$. Für eine ausführliche Beschreibung des Algorithmus zur Berechnung der Eigenwert-Punkte sei auf [19, Sloan] verwiesen, die Punkte in 3D für spezielle n sind online verfügbar [22, UNSW]. Diese haben sich als besonders stabil für einen Vergleich variabler ε-Werte erwiesen. Für Interpolationspunkte im Quadrat $[0,1]^2$ werden wir die Halton-Punkte $H_{2,N}$ aus [10] zu den Primzahlen $p_1 = 7$ und $p_2 = 11$. Die Menge der Halton-Punkte ist dabei definiert durch

$$H_{s,N} = \{(h_{p_1}(n), \ldots, h_{p_s}(n)) : \ n = 0, \ldots, N\}.$$

Dazu gilt für die *van der Corput* Folgen

$$h_p(n) = \sum_{i=0}^{k} \frac{a_i}{p^{i+1}} \text{ mit } n = \sum_{i=0}^{k} a_i p^i,$$

wobei p eine geeignete Primzahlbasis ist. Die Verteilungen der Punkte sind in Abbildung 6.1 für 64 Punkte dargestellt.

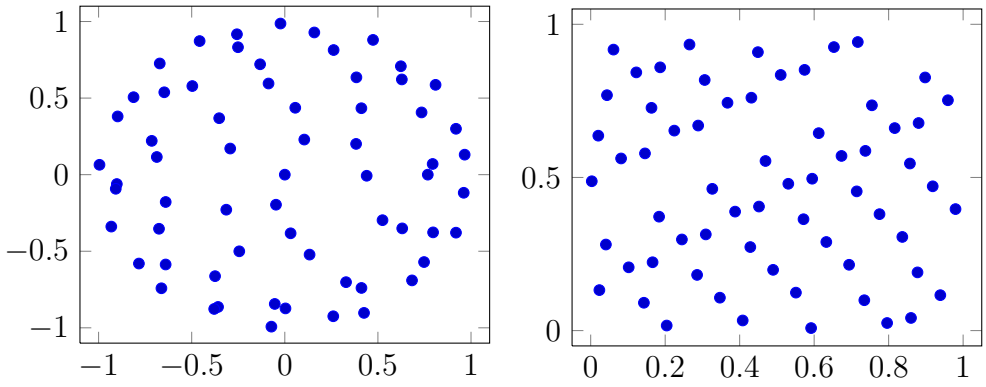

Abbildung 6.1: Verteilung von 64 Interpolationspunkte in $B(0,1)_2$ und $[0,1]^2$.

Als Testfunktionen verwenden wir die Funktionen aus Tabelle 4.1.

$$f_1(x,y) = \frac{25}{25 + (x-0.2)^2 + 2y^2}, \tag{6.1.1}$$

$$f_2(x,y) = \frac{\exp\left((x-0.1)^2 + 0.5y^2\right)}{\exp(1.21)}, \tag{6.1.2}$$

$$f_3(x,y) = \frac{\arctan\left(2(x+3y-1)\right)}{\arctan\left(2(\sqrt{10}+1)\right)}, \tag{6.1.3}$$

$$f_4(x,y) = \sin(2\pi(x-y)), \tag{6.1.4}$$

$$\begin{aligned} f_5(x,y) = {} & \frac{3}{4}\exp^{-\frac{1}{4}((9x-2)^2+(9y-2)^2)} + \frac{3}{4}\exp^{-\frac{1}{49}(9x+1)^2-\frac{1}{10}(9y+1)} \\ & + \frac{1}{2}\exp^{-\frac{1}{4}((9x-7)^2+(9y-3)^2))} - \frac{1}{5}\exp^{-(9x-4)^2-(9y-7)^2}, \end{aligned} \tag{6.1.5}$$

$$f_6(x,y) = (1-2|x|)_+ + (1-2|y|)_+. \tag{6.1.6}$$

In Abbildung 6.2 sind die Testfunktionen (6.1.1) - (6.1.6) für $B(0,1)_2$, ausgewertet an 625 Punkten, geplottet. Abbildung 6.3 zeigt die gleichen Funktionen für das Quadrat $[0,1]^2 \in \mathbb{R}^2$, ebenfalls ausgewertet an 625 Punkten. Die Graphen wurde mit der Funktion `PlotteFunktion.m` (siehe Quelltext A.17) erstellt.

Für einen qualifizierten Vergleich der Algorithmen verwenden wir als Referenzalgorithmus den in Kapitel 2.2 beschriebenen direkten Interpolationsalgorithmus (*RBF-Direkt*). Dazu werten wir die Testfunktionen $f_1 - f_6$ an 10.000 Auswertungspunkte aus und vergleichen die relativen Fehler

$$\epsilon_{rel} = \frac{\|y - f\|_2}{\|f\|_2}. \tag{6.1.7}$$

Dabei sei y der Vektor der Ergebnisse der Interpolation bzw. Quasi-Interpolation und f der Vektor der exakten Lösungen an den Auswertungspunkte.

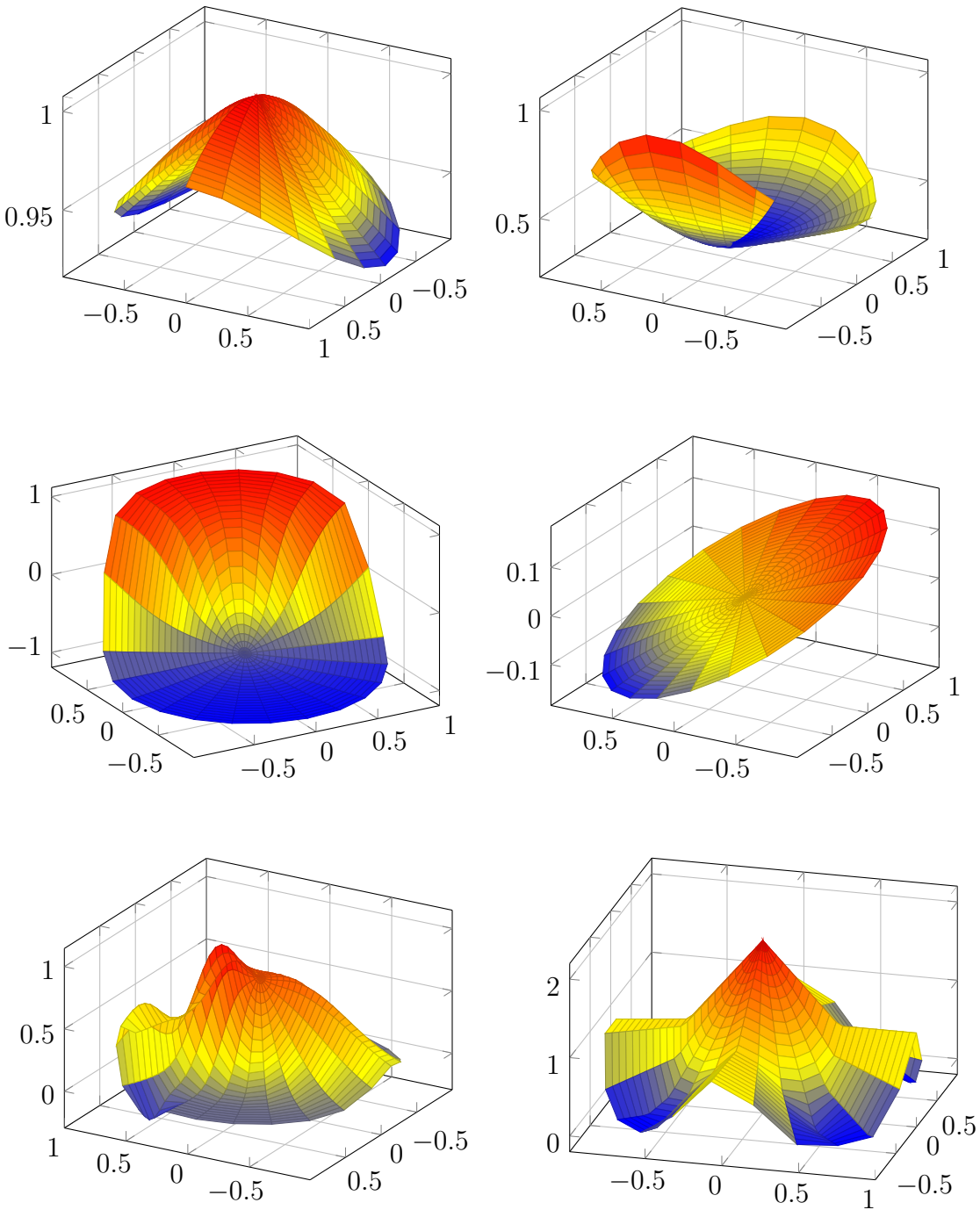

Abbildung 6.2: Testfunktionen für den Einheitskreis.

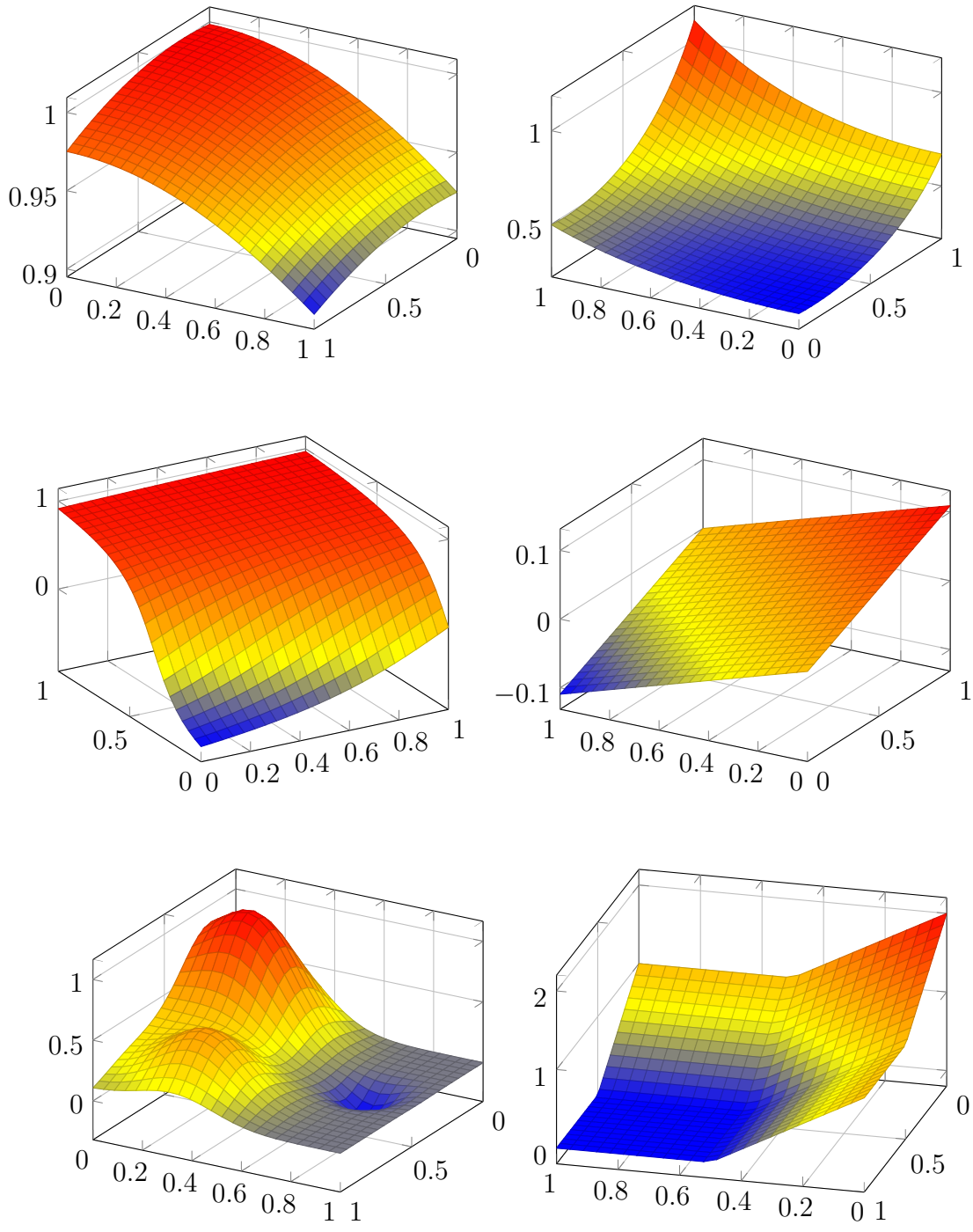

Abbildung 6.3: Testfunktionen für das Quadrat $[0,1]^2$.

6.2 Dagumfunktionen

Wir verwenden für die direkte Interpolation mit Dagumfunktionen zu unterschiedlichen β- und γ-Werten das Programm `RBFDirektDagumTest.m` (Quelltext A.2). Um die Nichtsingularität der Interpolationsmatrix sicherzustellen, verwenden wir Theorem 3.1.2 und 3.1.3 aus Kapitel 3, um β und γ optimal zu wählen. Insgesamt machen wir folgende Beobachtungen:

- Für $\beta = \gamma = 1$ erhalten wir die kleinsten relativen Fehler,
- für $\gamma = 1/\beta$ und $\beta = 1/4$ sowie $\beta = 1/10$ erhalten wir eine gute Approximation, jedoch für $\beta \to 0$ eine instabile Interpolationsmatrix,
- für $\beta = 2$ und $\gamma = 1/3$ sowie $\gamma = 1/10$ ist die Approximation ebenfalls gut, allerdings für $\gamma \to 0$ ebenfalls mit größeren relativen Fehlern, gleiches gilt auch
- für $\beta = \gamma$ mit $\beta = 1/2$ und $\beta = 1/7$.

Die vollständige Monotonie haben wir nach Theorem 3.1.2 für die Fälle Wie untersuchen zusätzlich noch Dagum-Funktionen mit $\beta = 2$. In Theorem 3.1.3

Für die Interpolationspunkte verwenden wir die bereits angesprochenen Eigenwertpunkte für $B(0,1)_2$ und Haltonpunkte für $[0,1]^2$. Die Verteilung ist in Abbildung 6.4 für 100, 400 und 900 Punkte geplottet.

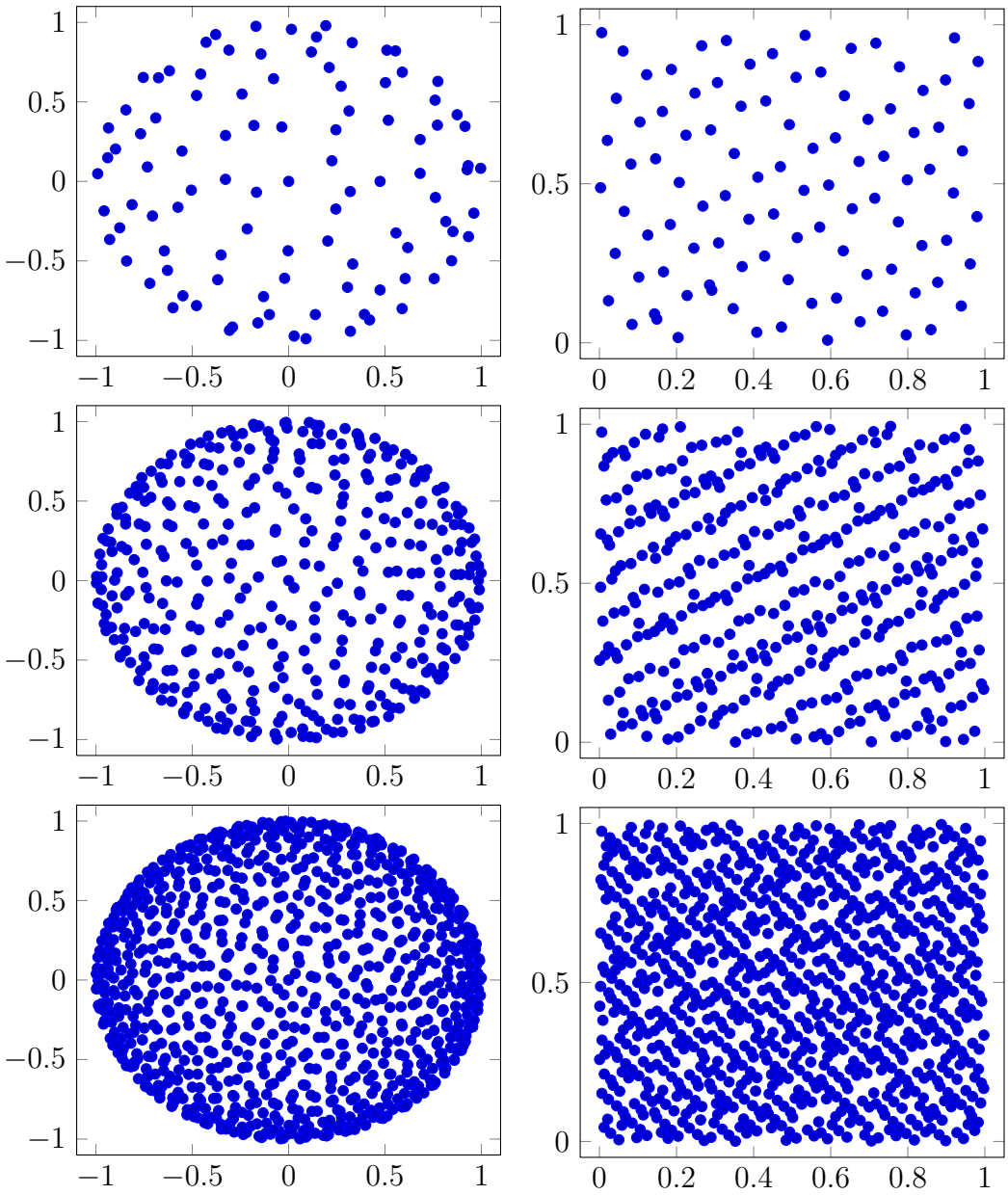

Abbildung 6.4: Verteilung von 100, 400 und 900 Interpolationspunkte für $B(0,1)_2$ und $[0,1]^2$.

Parameterwerte $\beta = \gamma = 1$

Die Funktion

$$\varphi_{1,1}(r) = 1 - \left(\frac{r}{1+r}\right), \ r \geq 0. \tag{6.2.8}$$

ist nach Theorem 3.1.2 (ii) vollständig monoton. Die Interpolation für $B(0,1)_2$ und $[0,1]^2$ wird in beiden Fällen auch für große n berechenbar und der relative Fehler beträgt für die Franke-Funktion f_5 und die stückweise lineare Funktion f_6 bei $n = 900$ weniger als 1%, für f_1 sogar weniger als 10^{-7}. Die relativen Fehler sind in Abbildung 6.5 bzw. 6.6 dargestellt.

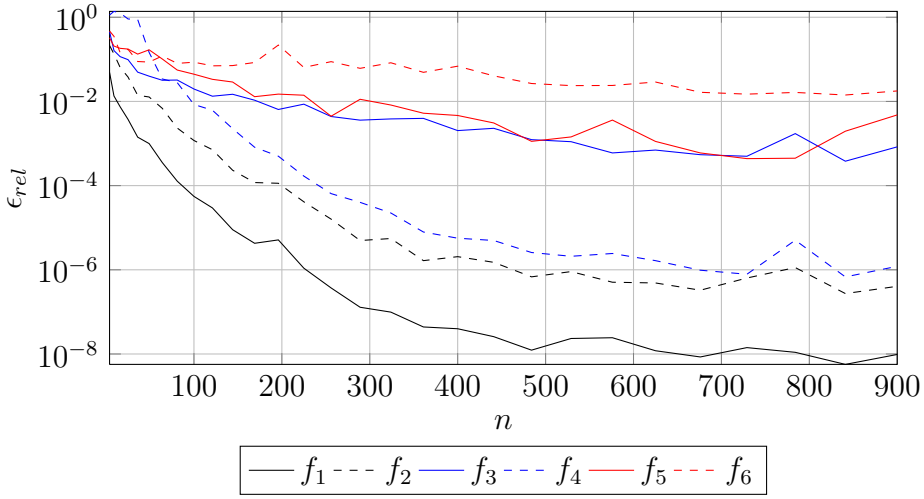

Abbildung 6.5: Relativer Fehler auf $B(0,1)_2$.

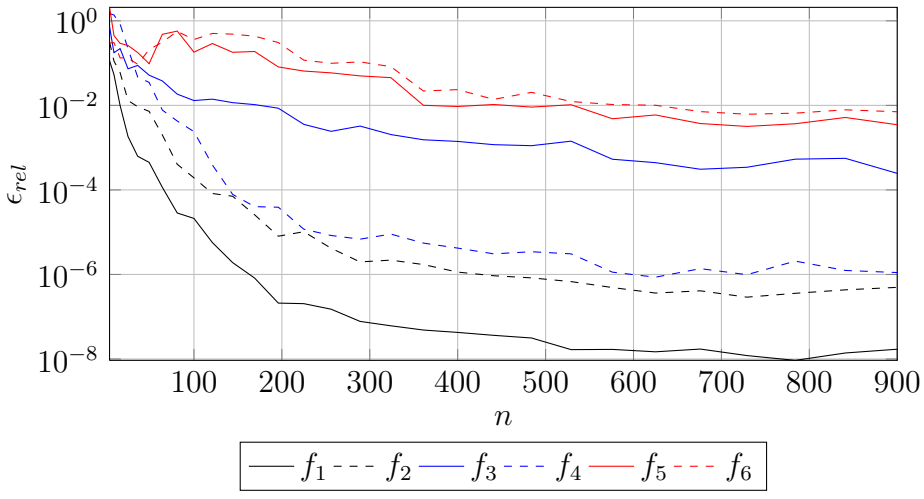

Abbildung 6.6: Relativer Fehler auf $[0,1]^2$.

Parameterwerte $\beta = 1/4$, $\gamma = 4$

Für die Funktion $\varphi_{1/4,4}(r)$ ist die Interpolation in beiden Fällen ($B(0,1)_2$ und $[0,1]^2$) ebenfalls für große n berechenbar und der relative Fehler beträgt für alle Funktionen bei $n = 900$ weniger als 1%, für f_1 mit $\epsilon_{rel} = 10^{-5}$ eine etwas schlechtere Approximation im Vergleich zum vorherigen Fall $\beta = \gamma = 1$.

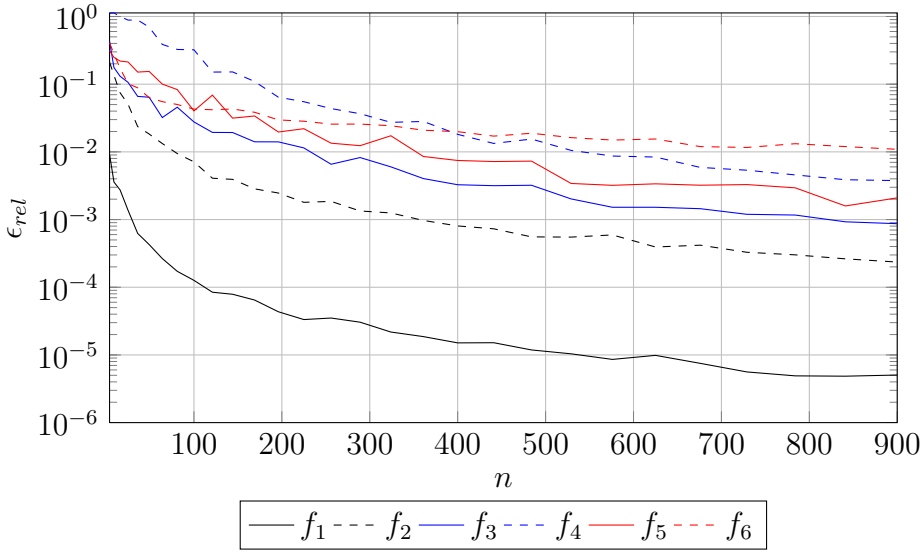

Abbildung 6.7: Relativer Fehler auf $B(0,1)_2$.

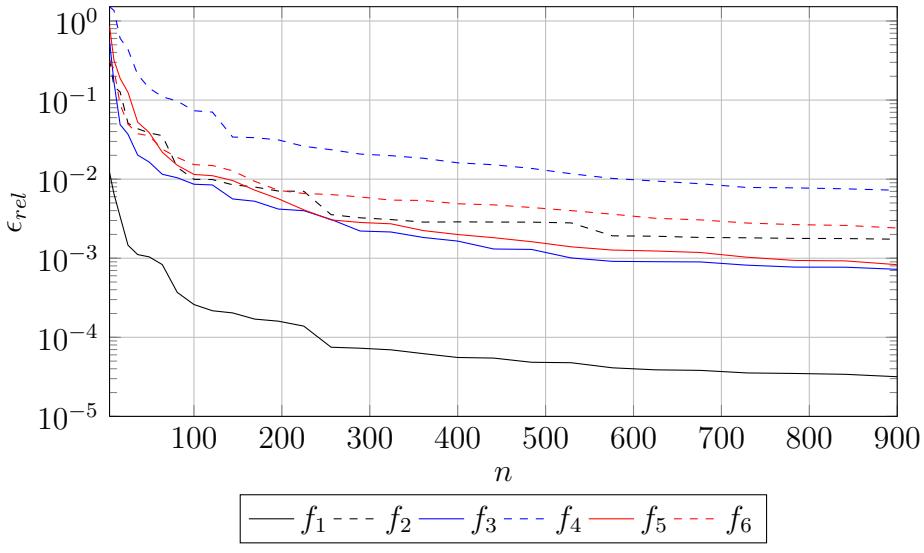

Abbildung 6.8: Relativer Fehler auf $[0,1]^2$.

Parameterwerte $\beta = 1/10$, $\gamma = 10$

Die Funktion $\varphi_{1/10,10}(r)$ interpoliert für $B(0,1)_2$ und $[0,1]^2$ insgesamt mit größerem relativen Fehler im Vergleich zu den vorherigen Fällen. Dieser Trend wird für weitere Verkleinerungen von β mit $\gamma = 1/\beta$ fortgesetzt und liegt vermutlich an der damit verbundenen numerischen Instabilität, da die Einträge der Interpolation in diesem Fall gegen 1 gehen. Für $\beta = 1/55$ ist keine Approximation mit der direkten Interpolation mehr zu erreichen.

Die Konditionszahl der Interpolationsmatrizen sind in den Abbildungen 6.11 und 6.12 in Abhängigkeit des β-Wertes geplottet.

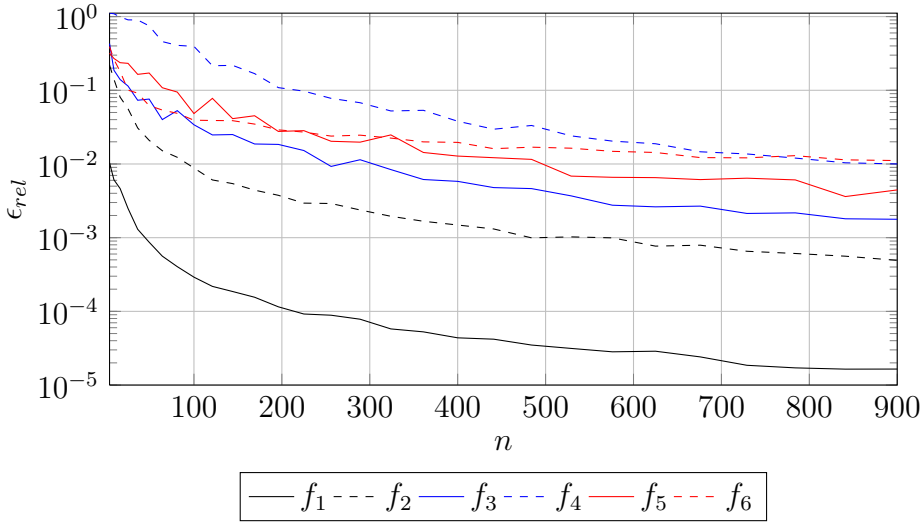

Abbildung 6.9: Relativer Fehler auf $B(0,1)_2$.

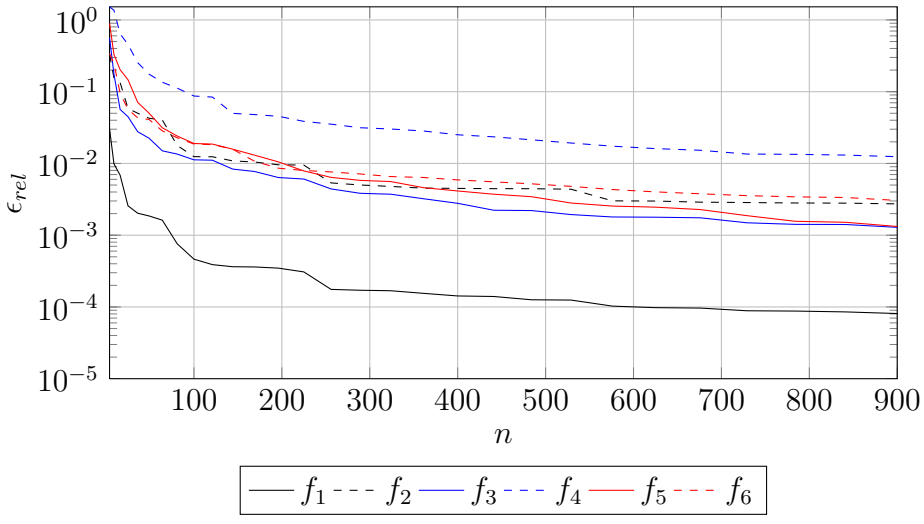

Abbildung 6.10: Relativer Fehler auf $[0,1]^2$.

Konditionszahl für $\beta = 1/\gamma$

Die Konditionszahlen der Interpolationsmatrizen für den Fall $\beta = 1/\gamma$ werden für große $1/\beta > 50$ ebenfalls sehr groß. Dies ist auch mit Blick auf die Fehlerbetrachtung der Abbildungen 6.5 - 6.10 zu erwarten. Alle nicht geplotteten Werte entsprechen dem ausgegebenen Wert `Inf` (numerisch ∞).

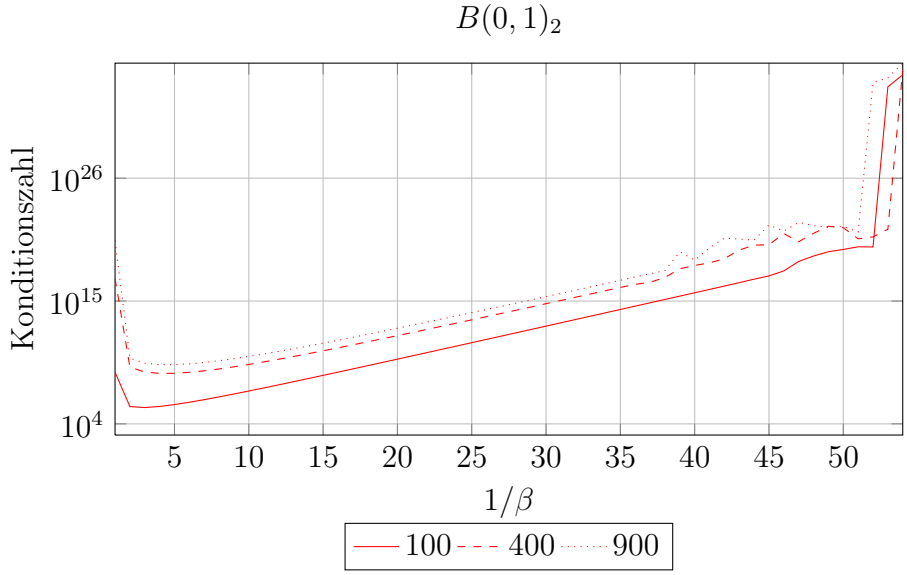

Abbildung 6.11: Konditionszahl der Interpolationsmatrix für $\beta = 1/\gamma$ - für alle größeren $1/\beta$ ist der berechnete Wert ∞.

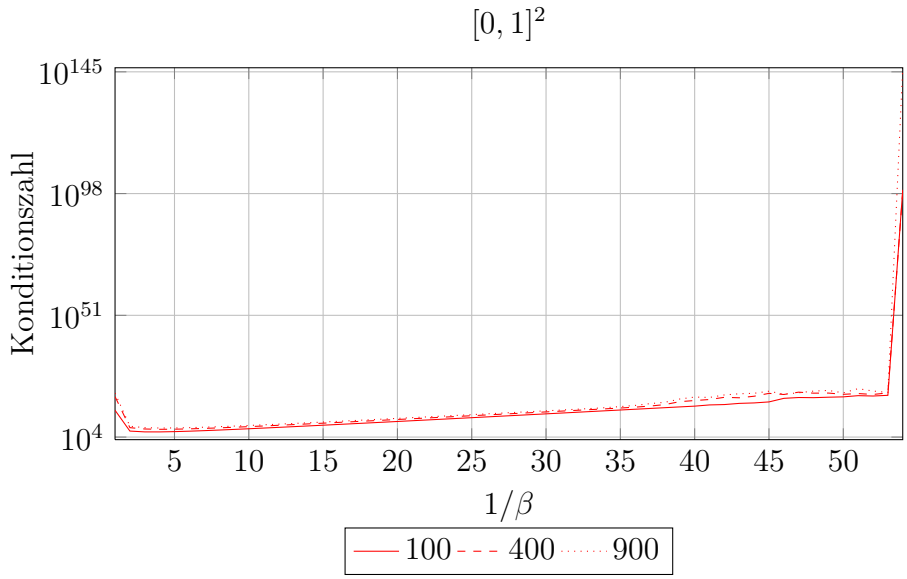

Abbildung 6.12: Konditionszahl der Interpolationsmatrix für $\beta = 1/\gamma$ - für alle größeren $1/\beta$ ist der berechnete Wert ∞.

Parameterwerte $\beta = 2$, $\gamma = 1/3$

Wir betrachten die Funktion

$$\varphi_{2,1/3}(r) = 1 - \left(\frac{r^2}{1+r^2}\right)^{1/3}, \; r \geq 0, \tag{6.2.9}$$

zur numerischen Untersuchung ohne direkten Nachweis der vollständigen Monotonie. Der relative Fehler der Interpolation wird in beiden Fällen ($B(0,1)_2$ und $[0,1]^2$) mit wachsendem n kleiner (Abbildung 6.13 bzw. 6.14).

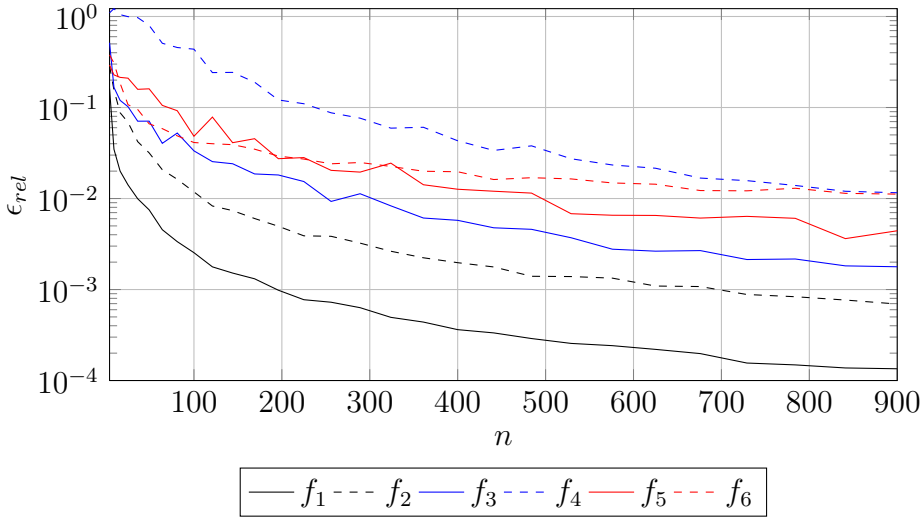

Abbildung 6.13: Relativer Fehler auf $B(0,1)_2$.

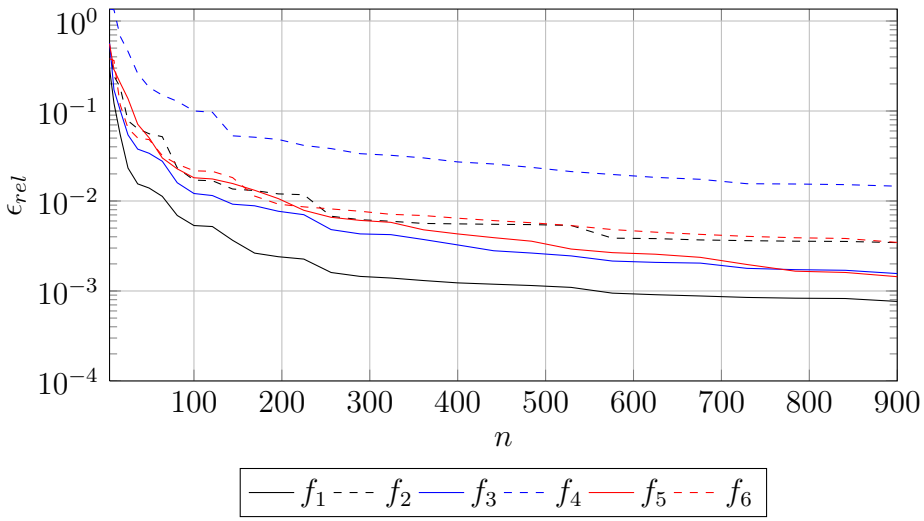

Abbildung 6.14: Relativer Fehler auf $[0,1]^2$.

Parameterwerte $\beta = 2$, $\gamma = 1/10$

Der relative Fehler der Interpolation für die Funktion

$$\varphi_{2,1/10}(r) = 1 - \left(\frac{r^2}{1+r^2}\right)^{1/10}, \; r \geq 0. \tag{6.2.10}$$

wird in beiden Fällen ($B(0,1)_2$ und $[0,1]^2$) mit wachsendem n zwar kleiner (Abbildung 6.15 bzw. 6.16), ist aber insgesamt für zu groß.

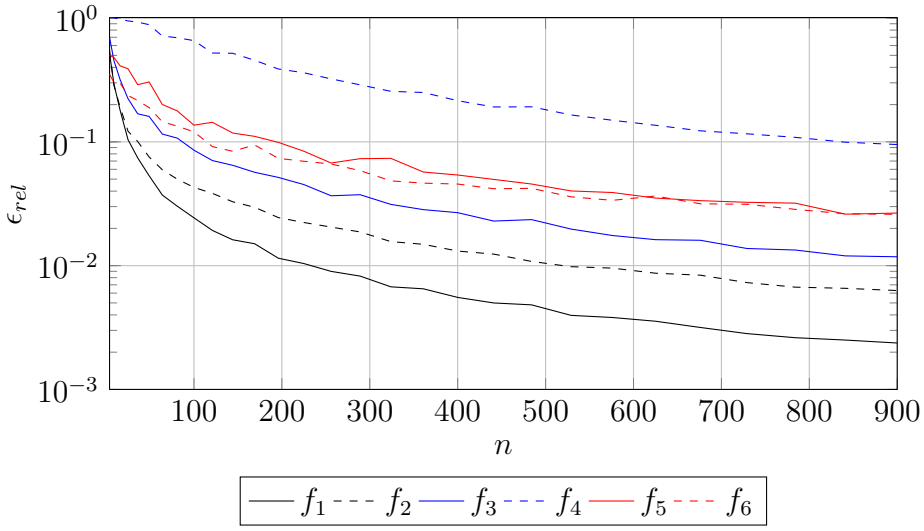

Abbildung 6.15: Relativer Fehler auf $B(0,1)_2$.

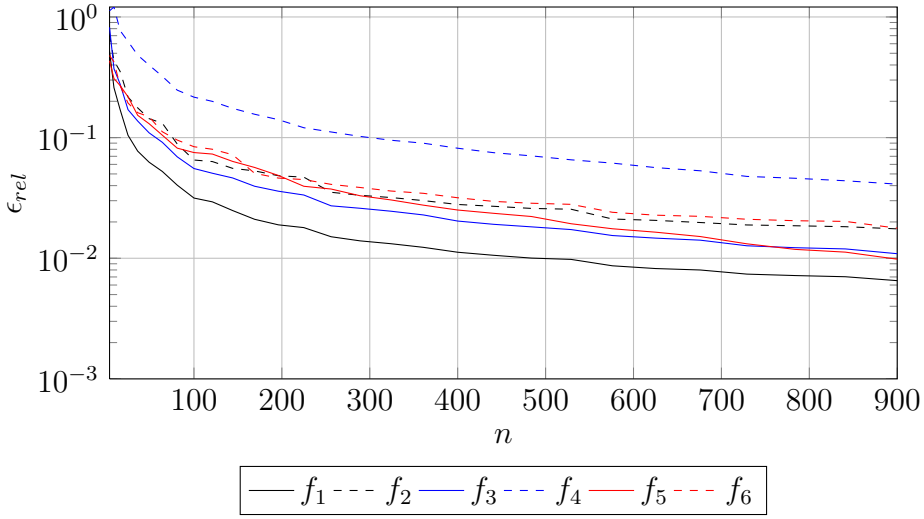

Abbildung 6.16: Relativer Fehler auf $[0,1]^2$.

Konditionszahl für $\beta = 2$

Die Konditionszahl der Interpolationsmatrix für den Fall $\beta = 2$ ist zwar für γ in der Nähe von 1 relativ groß, jedoch nicht so, dass die Interpolationsmatrix beeinflusst wird. Des weiteren fällt sie für kleiner werdendes γ sogar ab, wie in den Abbildungen 6.17 und 6.18 zu sehen. Der teilweise schlechte Interpolationsfehler, die in den Abbildungen 6.13 - 6.16 zu sehen sind, können nicht zwingend auf eine numerische Instabilität der Interpolationsmatrix zurück geführt werden. Hier scheint es für unterschiedliche Interpolationsprobleme unterschiedliche optimale β und γ Parameter zu geben.

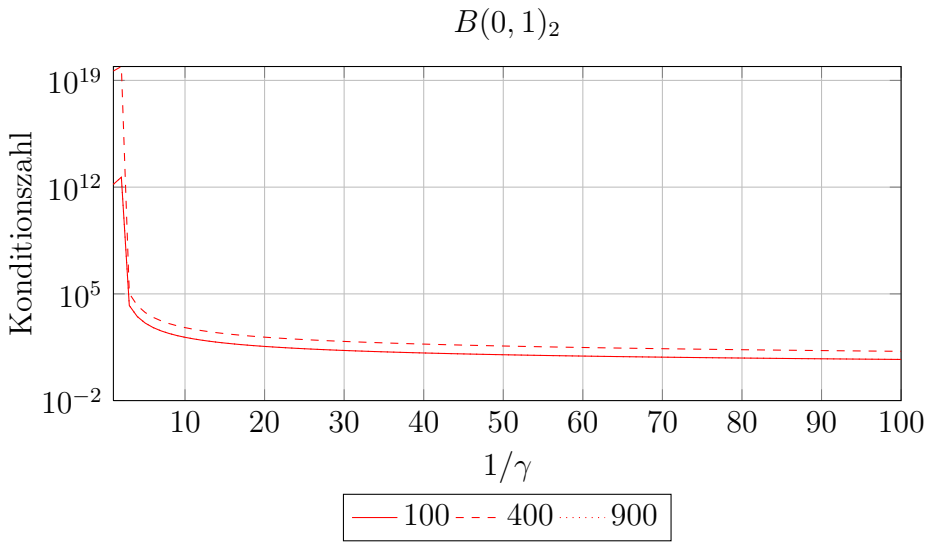

Abbildung 6.17: Konditionszahl der Interpolationsmatrix für $\beta = 2$ - für alle größeren $1/\gamma$ ist der berechnete Wert ∞.

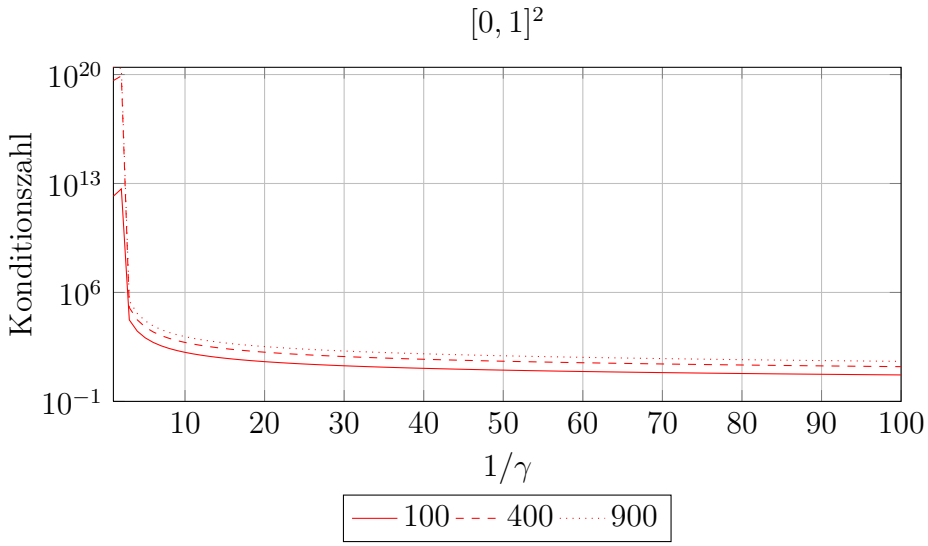

Abbildung 6.18: Konditionszahl der Interpolationsmatrix für $\beta = 2$ - für alle größeren $1/\gamma$ ist der berechnete Wert ∞.

Parameterwerte $\beta = 1/2$, $\gamma = 1/2$

Die Funktion

$$\varphi_{1/2,1/2}(r) = 1 - \left(\frac{r^{1/2}}{1 + r^{1/2}}\right)^{1/2}, \; r \geq 0. \tag{6.2.11}$$

ist nach Theorem 3.1.2 (i) vollständig monoton. Der relative Fehler der Interpolation wird in beiden Fällen ($B(0,1)_2$ und $[0,1]^2$) mit wachsendem n zwar kleiner (Abbildung 6.19 bzw. 6.20), ist aber insgesamt schlechter als $\beta = \gamma = 1$.

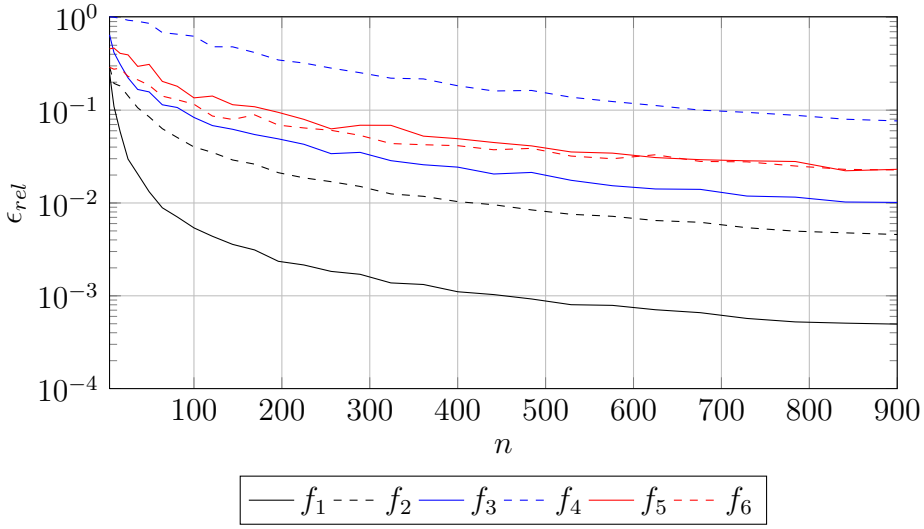

Abbildung 6.19: Relativer Fehler auf $B(0,1)_2$.

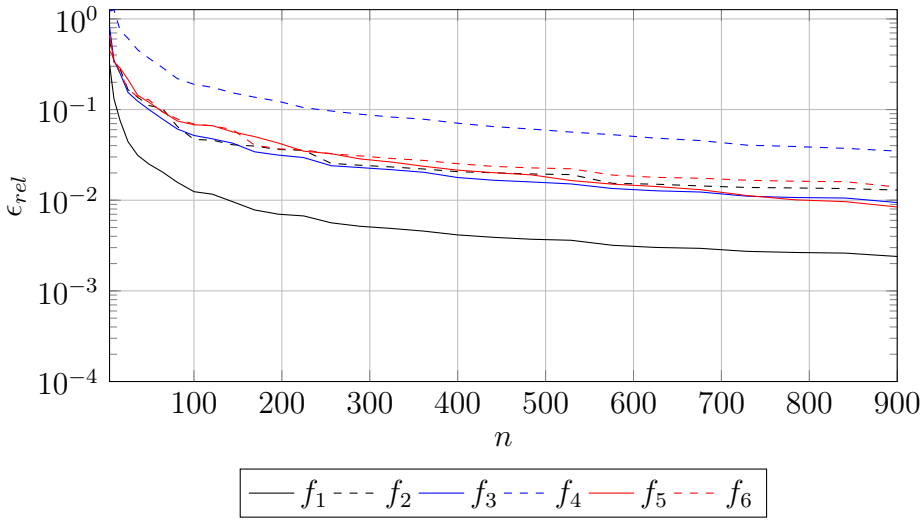

Abbildung 6.20: Relativer Fehler auf $[0,1]^2$.

Parameterwerte $\beta = 1/7$, $\gamma = 1/7$

Die Funktion

$$\varphi_{1/7,1/7}(r) = 1 - \left(\frac{r^{1/7}}{1+r^{1/7}}\right)^{1/7}, \; r \geq 0. \tag{6.2.12}$$

ist nach Theorem 3.1.2 (i) vollständig monoton. Der relative Fehler wird in beiden Fällen ($B(0,1)_2$ und $[0,1]^2$) mit wachsendem n zwar wiederum kleiner (Abbildung 6.21 bzw. 6.22), ist aber insgesamt noch größer als im vorherigen Fall und die Approximation nur für f_1 und große n verwendbar.

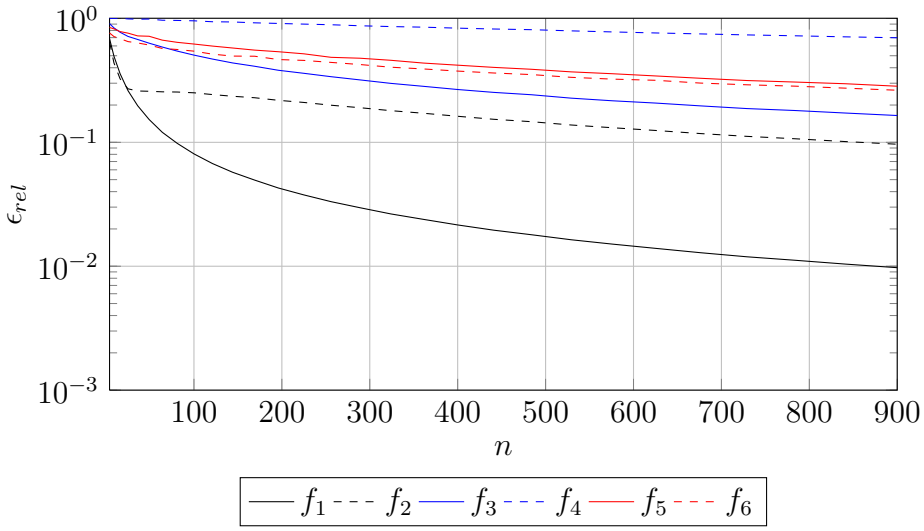

Abbildung 6.21: Relativer Fehler auf $B(0,1)_2$.

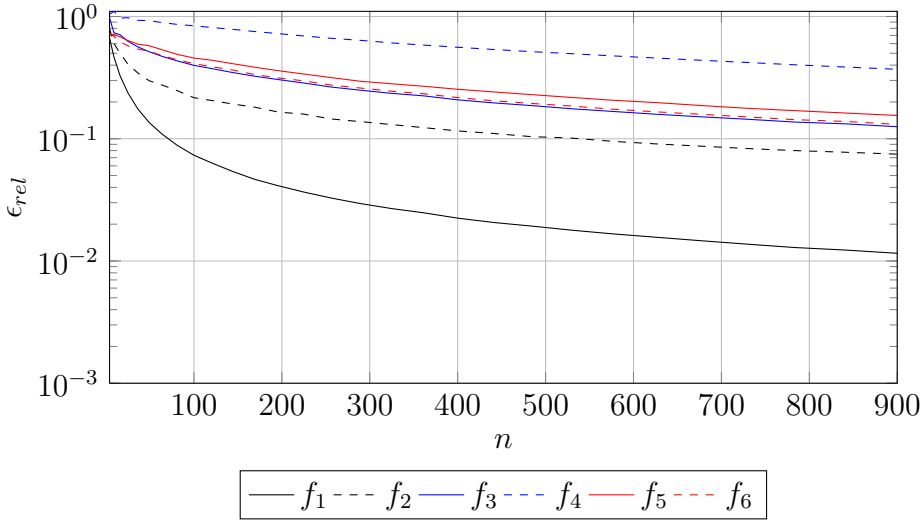

Abbildung 6.22: Relativer Fehler auf $[0,1]^2$.

Konditionszahl für $\beta = \gamma$

Für $\beta = \gamma$ und $\beta \to \infty$ verhält sich die Konditionszahl der Interpolationsmatrix wie im vorherigen Fall $\beta = 2$. Auch hier scheint der teilweise schlechte Interpolationsfehler in den Abbildungen 6.19 - 6.22 nicht zwingend auf eine numerische Instabilität der Interpolationsmatrix zurück geführt werden zu können. Hier könnte es ebenfalls für unterschiedliche Interpolationsprobleme unterschiedliche optimale β und γ Parameter zu geben.

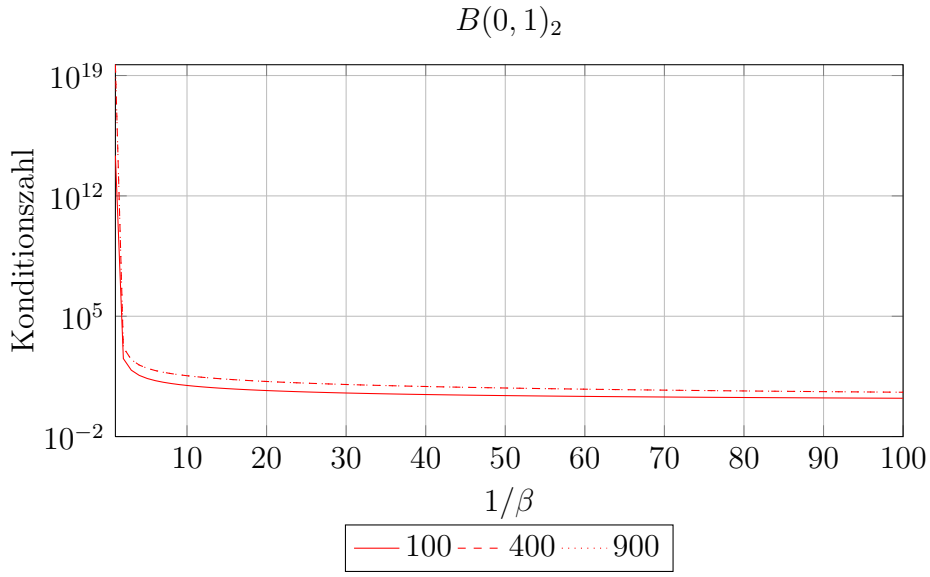

Abbildung 6.23: Konditionszahl der Interpolationsmatrix für $\beta = \gamma$ - für alle größeren $1/\beta$ ist der berechnete Wert ∞.

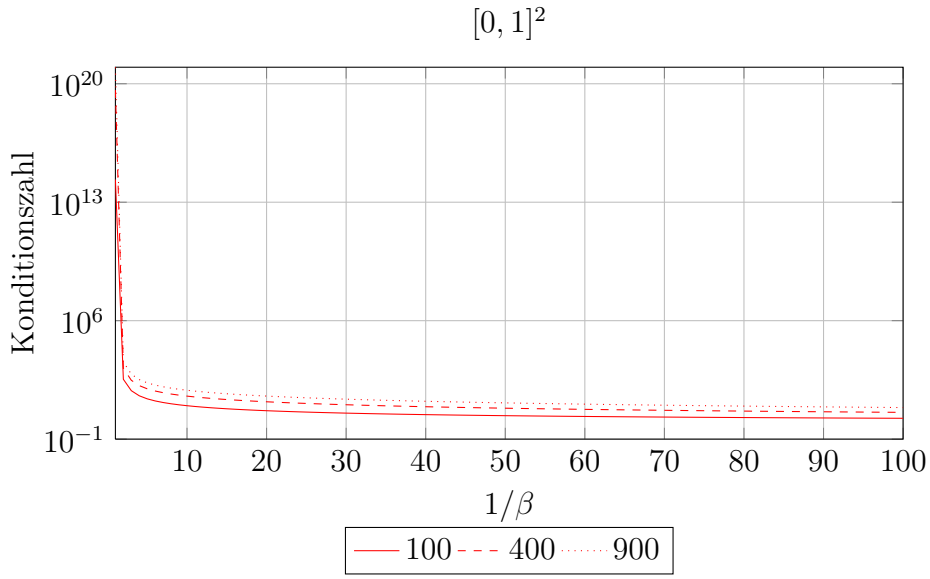

Abbildung 6.24: Konditionszahl der Interpolationsmatrix für $\beta = \gamma$ - für alle größeren $1/\beta$ ist der berechnete Wert ∞.

6.3 Der RBF-QR-GQ-Algorithmus

Wir verwenden wieder die Funktionen aus Tabelle 4.1, um den RBF-QR-GQ- mit dem direkten- und dem RBF-QR-Algorithmus zu vergleichen. Für die Interpolationspunkte verwenden wir wieder die Eigenwertpunkte für $B(0,1)_2$ und Haltonpunkte für $[0,1]^2$. Dabei beschränken wir uns für die Betrachtung von $\varepsilon \to 0$ auf 100, 400 und 900 Interpolationspunkte (siehe auch Abbildung 6.4). Alle nicht geplotteten Werte entsprechen wieder dem ausgegebenen Wert `Inf` (numerisch ∞).

Die inverse Multiquadrics

Durch den RBF-QR-GQ-Algorithmus wird die Konditionszahl der Interpolationsmatrix für die IMQ stabilisiert, wobei für $\varepsilon > 0.1$ die Konditionszahl auf gleichem Niveau wir die des direkten Algorithmus liegt. Für kleinere ε-Werte dagegen liegt sie mit der des RBF-QR-Algorithmus gleichauf. Diese Beobachtung setzt sich für alle betrachteten n fort, wie in Abbildung 6.25 zu sehen.

Der relative Fehler der IMQ für den RBF-QR-GQ-Algorithmus ist für $n = 100$ ab einem genug kleinen ε-Wert gleich dem des RBF-QR-Algorithmus. Für ε nahe bei 1 ist der Fehler hingegen sogar größer als der des direkten Algorithmus. Dies ergibt sich aus der Tatsache, dass der Fehlerterm (4.2.30) der Gauß-Quadratur

$$R_{N_Q} = \frac{N_Q!\,\Gamma(N_Q - 1/2)}{(2N_Q)!} \left.\frac{\mathrm{d}^{2N_Q}}{\mathrm{d}u^{2N_Q}} e^{-u\varepsilon^2 r^2}\right|_{u=\xi}, \ \xi \in \mathbb{R}^+. \tag{6.3.13}$$

für großes ε nicht klein genug gemacht werden kann, um eine gute Approximation an das Integral zu erhalten.

Für die folgenden numerischen Untersuchungen bestimmen wir die Anzahl der Knoten N_Q, indem wir N_Q derart wählen, dass R_{N_Q} kleiner der Maschinengenauigkeit ist. Dabei schätzen wir den Term

$$\left.\frac{\mathrm{d}^{2N_Q}}{\mathrm{d}u^{2N_Q}} e^{-u\varepsilon^2 r^2}\right|_{u=\xi}$$

ab durch

$$(\varepsilon\delta_{\max})^{4N_Q},$$

mit $\delta_{\max} = \max \|x - y\|_2$ und x, y Interpolationspunkte.

Dass die Fehler für sehr kleine ε fast gleich sind, ist nicht verwunderlich, da in

diesem Fall die Anzahl der Gauß-Quadraturknoten aufgrund der verwendeten Abschätzung über den Fehlerterm gleich 1 ist. Gleichwohl gibt es auch ε-Werte, für die sich die relativen Fehler unterscheiden und die IMQ zum Teil auch besser ist als der Gaußkern, so in Abbildung 6.26 für f_1 und f_2 zu sehen. Dies ist außerdem interessant, da sich f_2 aus zwei Exponentialfunktionen zusammensetzt.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100, 400 **und** 900 **Interpolationspunkte auf dem Kreis**

Während die Konditionszahlen der Interpolationsmatrizen für den direkten Algorithmus mit $\varepsilon \to 0$ sehr groß werden, stabilisieren sich die des RBF-QR und RBF-QR-GQ- Algorithmus. Zwar steigen die Konditionszahlen für größere n ebenfalls an, allerdings stabilisieren sich diese ebenfalls.

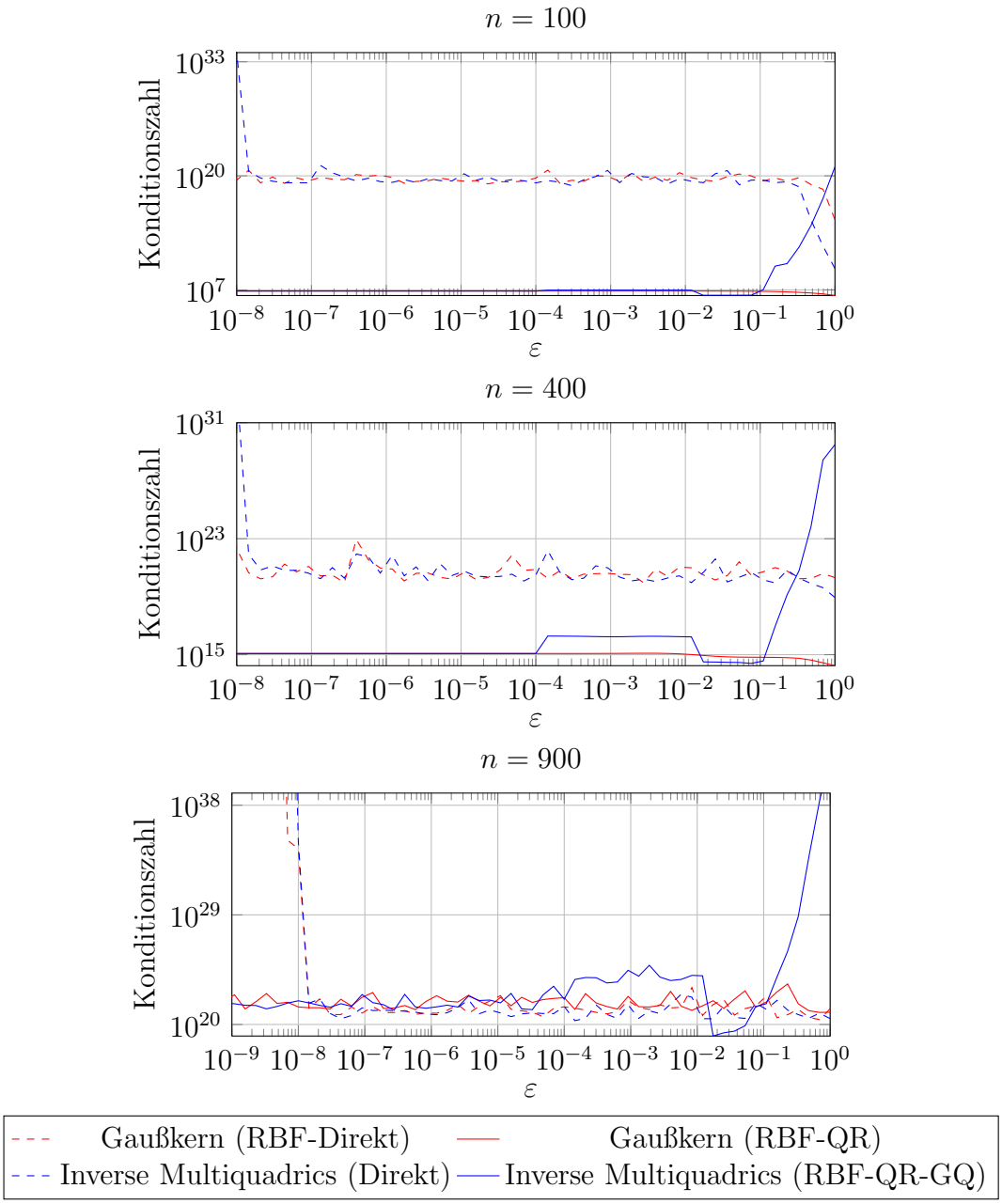

Abbildung 6.25: Konditionszahl bzgl. der GA- und IMQ-RBF für den Einheitskreis.

Relative Fehler der IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 100$

Die relativen Fehler der Interpolation mit dem RBF-QR-GQ-Algorithmus für die Testfunktionen f_1 und f_2 sind kleiner als 10^{-11} bzw. 10^{-7}, während die relativen Fehler für f_3, f_5 und f_6 zwar kleiner als die des direkten Algorithmus sind, im Gesamten jedoch unbrauchbar ($> 10\%$). Der relative Fehler für f_4 ist kleiner 10^{-1}.

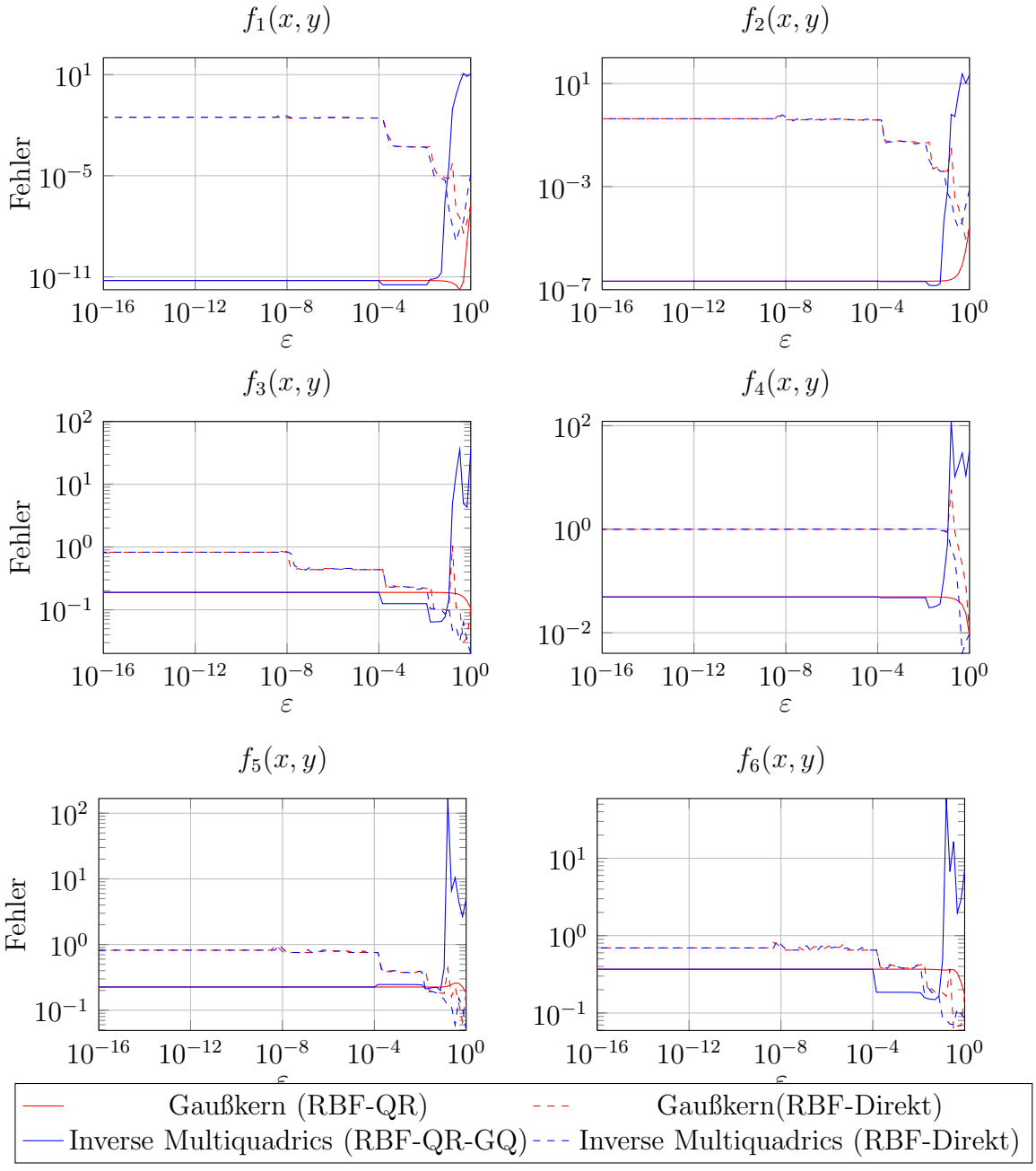

Abbildung 6.26: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Relative Fehler der IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 400$

Für 400 Interpolationspunkte sind die relativen Fehler der Interpolation mit dem RBF-QR-GQ-Algorithmus für die Testfunktionen f_1,f_2 und f_4 für kleines ε sehr gut, die für f_3 und f_5 kleiner als im Fall $n = 100$ ($< 10^{-1}$), und für f_6 wie im vorherigen Fall zu groß.

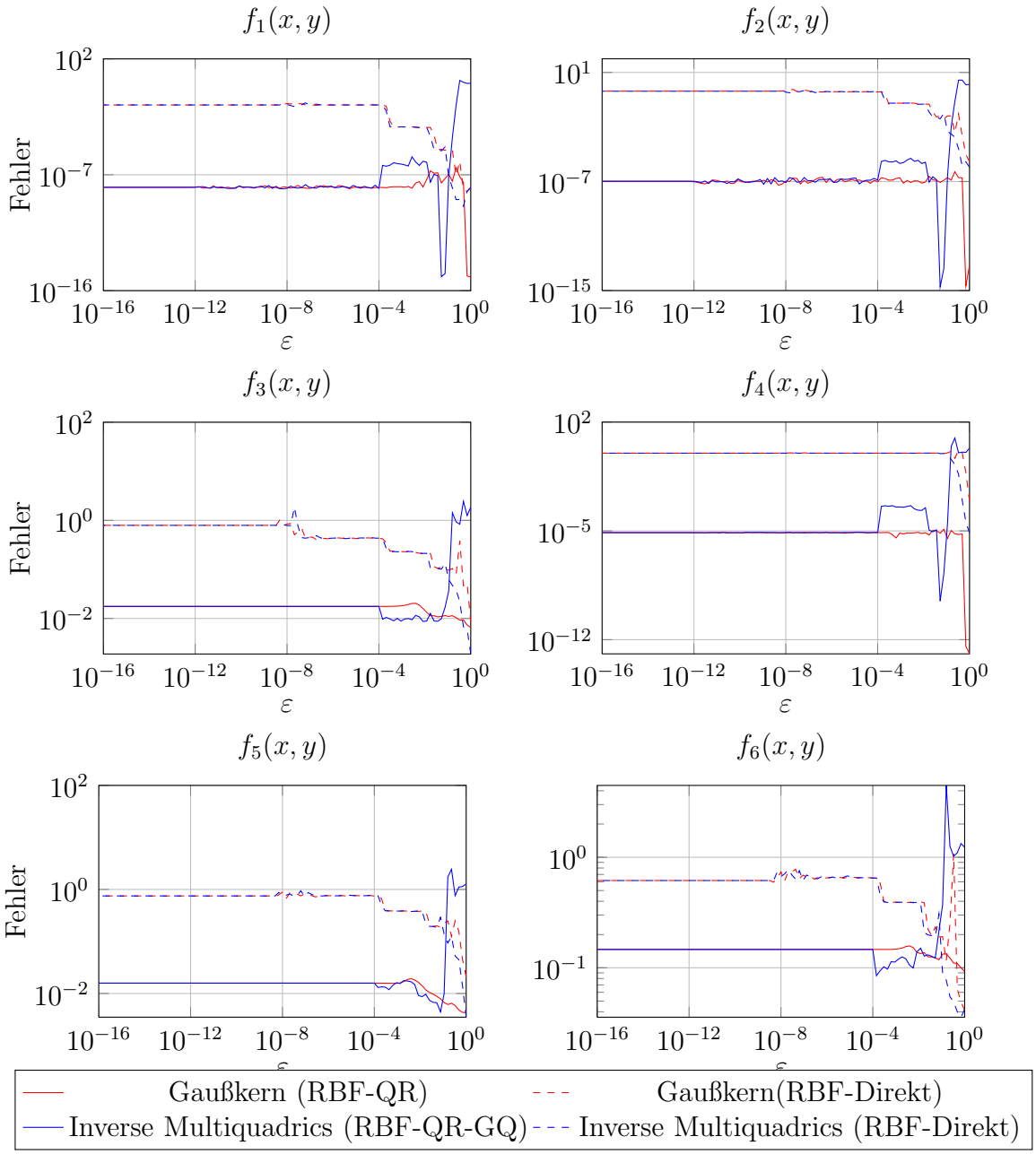

Abbildung 6.27: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 400 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Relative Fehler der IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 900$

Für 900 Interpolationspunkte verschlechtern sich die relativen Fehler der Interpolation mit dem RBF-QR-GQ-Algorithmus für die Testfunktionen f_1,f_2 und f_4 für kleines ε zwar wieder, allerdings erhalten wir für f_6 einen relativen Fehler im einstelligen Prozentbereich.

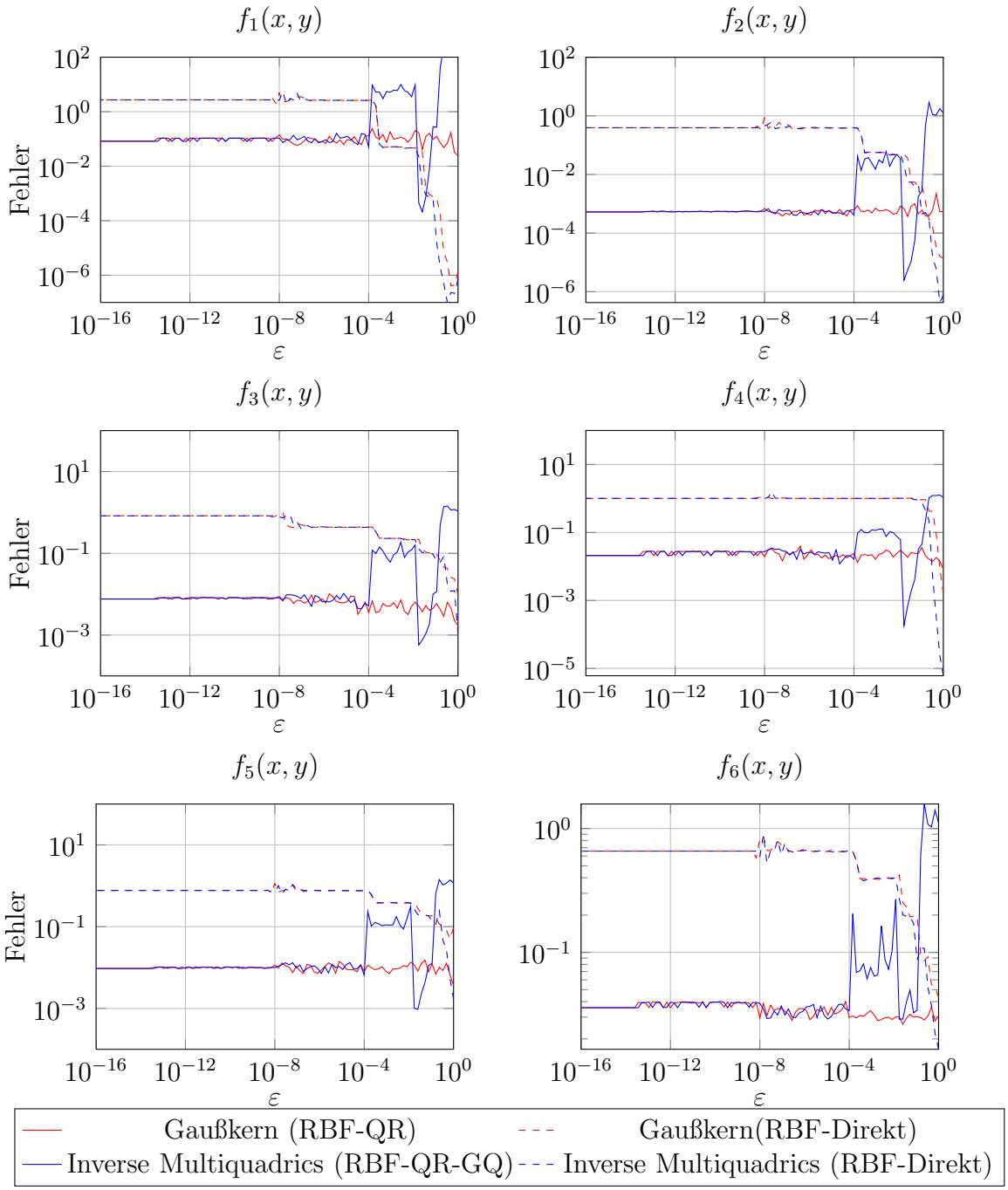

Abbildung 6.28: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 900 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100, 400 **und** 900 **Interpolationspunkte auf dem Quadrat**

Auf dem Quadrat werden die Konditionszahlen der Interpolationsmatrizen für den direkten Algorithmus und $\varepsilon \to 0$ ebenfalls sehr groß. Die Konditionszahlen des RBF-QR und RBF-QR-GQ- Algorithmus stabilisieren sich wie beim Kreis, sind für große n und $\varepsilon > 10^{-8}$ allerdings größer als die des direkten Algorithmus.

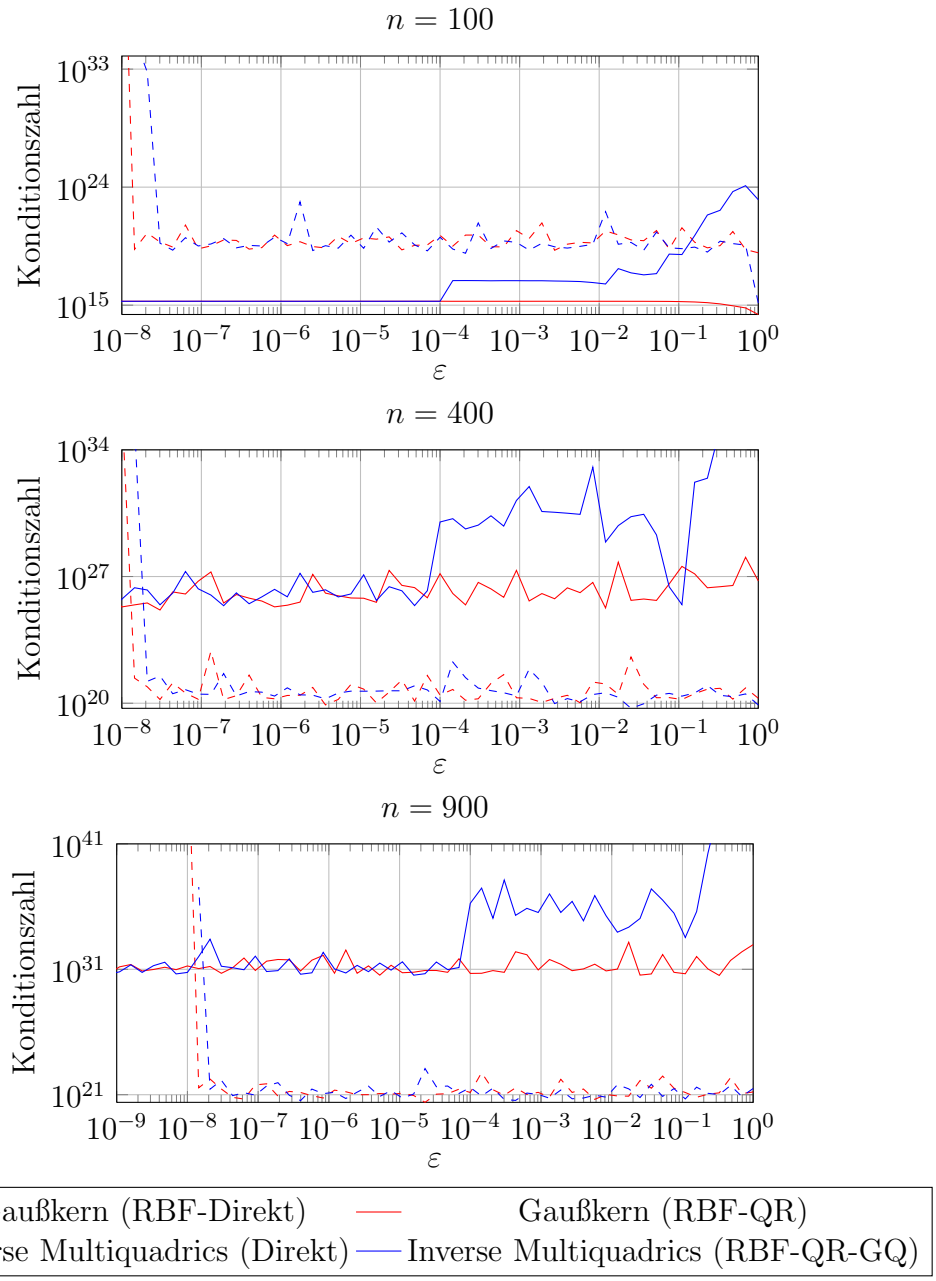

Abbildung 6.29: Konditionszahl bzgl. der GA- und IMQ-RBF für das Quadrat.

Relative Fehler der IMQ für $f_1 - f_6$ auf dem Quadrat und $\varepsilon \to 0$ mit $n = 100$

Für $n = 100$ zeigen sich auf dem Quadrat analoge Ergebnisse wie auf dem Kreis. Auffallend sind dabei die Einbrüche in den relativen Fehler für einzelne ε-Werte und die sehr hohen Fehler für f_3, f_5 und f_6.

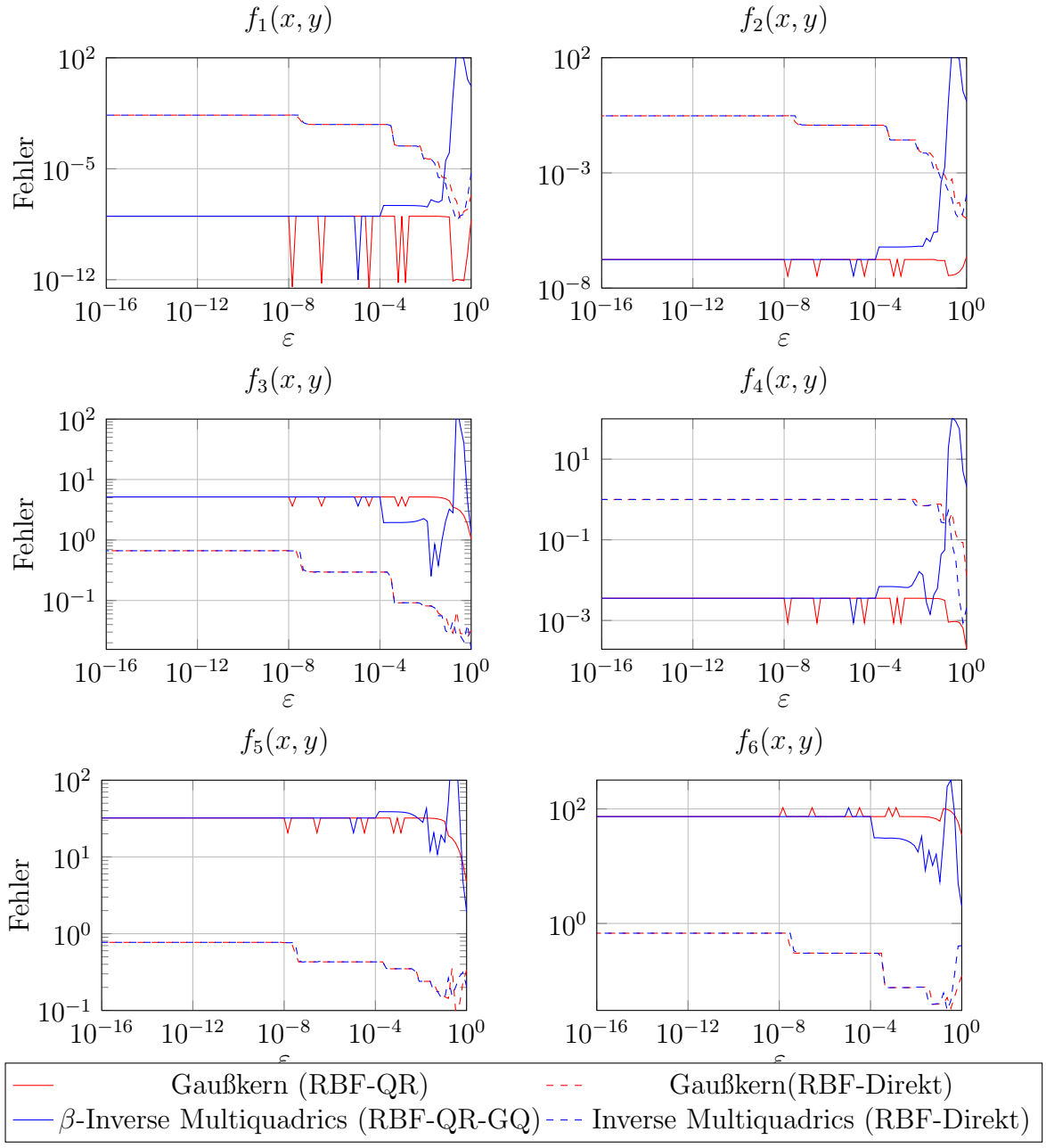

Abbildung 6.30: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $[0,1]^2$.

Die β-inverse Multiquadrics

Für die β-inverse Multiquadrics erhalten wir ähnliche Ergebnisse wie im Fall der IMQ. Für kleinere ε-Werte liefert der RBF-QR-GQ-Algorithmus eine gute Näherung an die exakten Funktionswerte. Für ε nahe bei 1 ist der Fehler wieder größer als der des direkten Algorithmus. Dies gilt für alle zulässigen $\beta \in \mathbb{R}^+$. Die Folgenden vorgestellten numerischen Ergebnisse beziehen sich auf $\beta = 9/2$. Danach folgt noch eine Zusammenstellung für unterschiedliche β-Wert zu einem festen ε.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100**,** 400 **und** 900 **Interpolationspunkte auf dem Kreis**

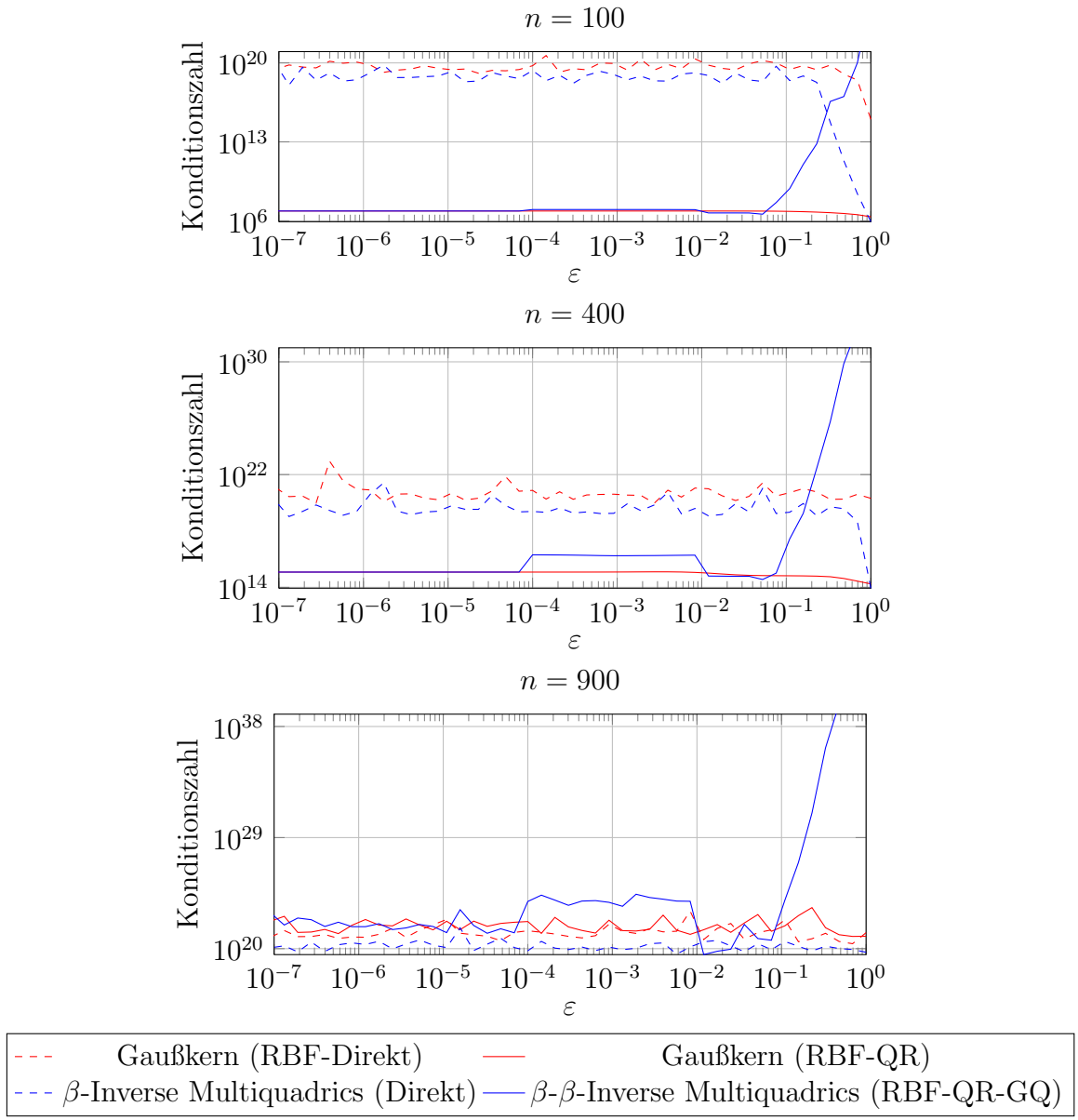

Abbildung 6.31: Konditionszahl bzgl. der GA- und β-IMQ-RBF, $\beta = 9/2$.

Relative Fehler der β-IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 100$ und $\beta = 9/2$

Die relativen Fehler der Interpolation mit dem RBF-QR-GQ-Algorithmus für die β-IMQ mit $\beta = 9/2$ sind für die Testfunktionen mit denen der IMQ vergleichbar. Abweichend ist die Stabilisierung des Fehlers auf einen festen Wert. Diese stellt sich erst bei kleinerem ε ein.

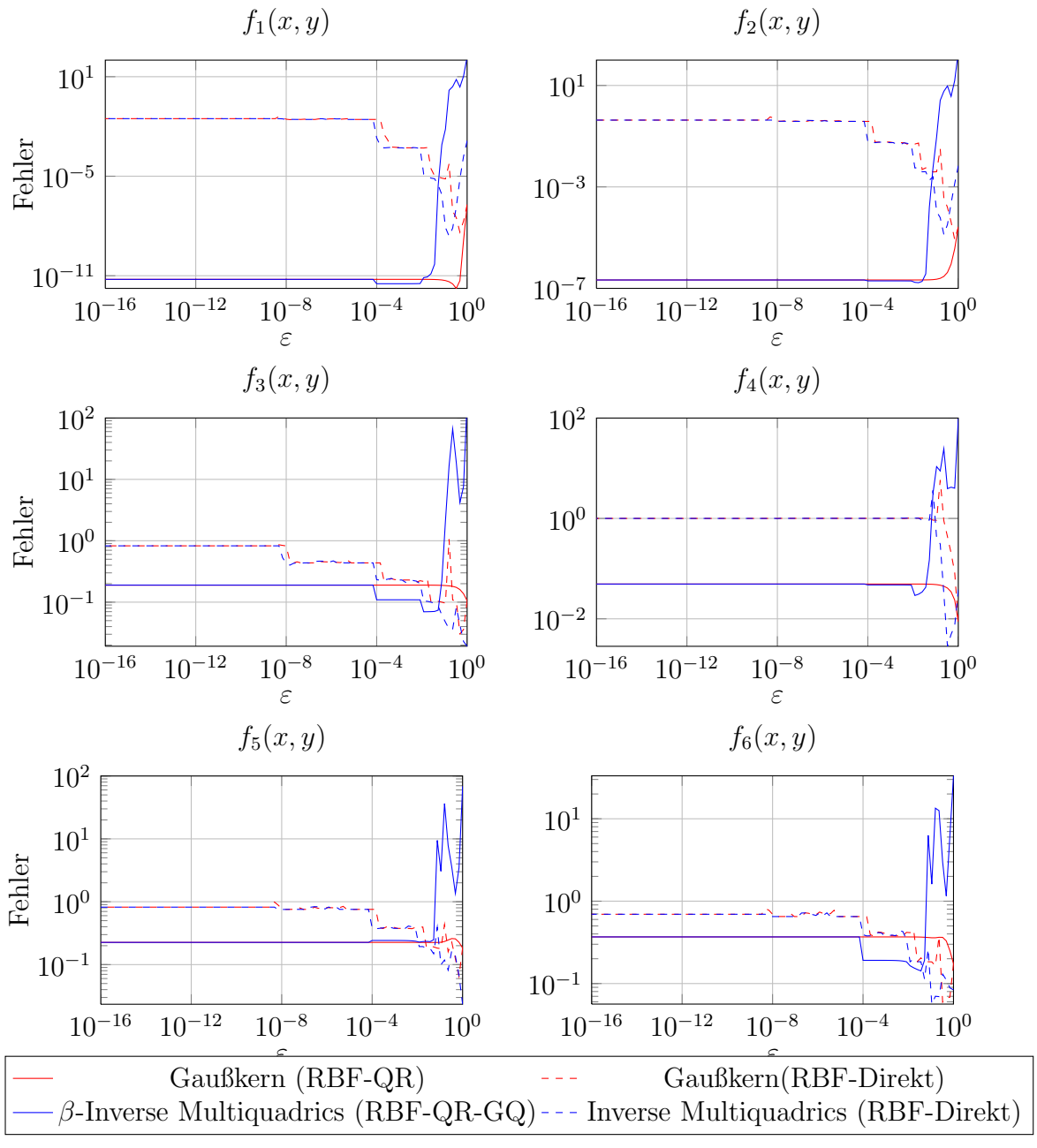

Abbildung 6.32: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Relative Fehler der β-IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 400$ und $\beta = 9/2$

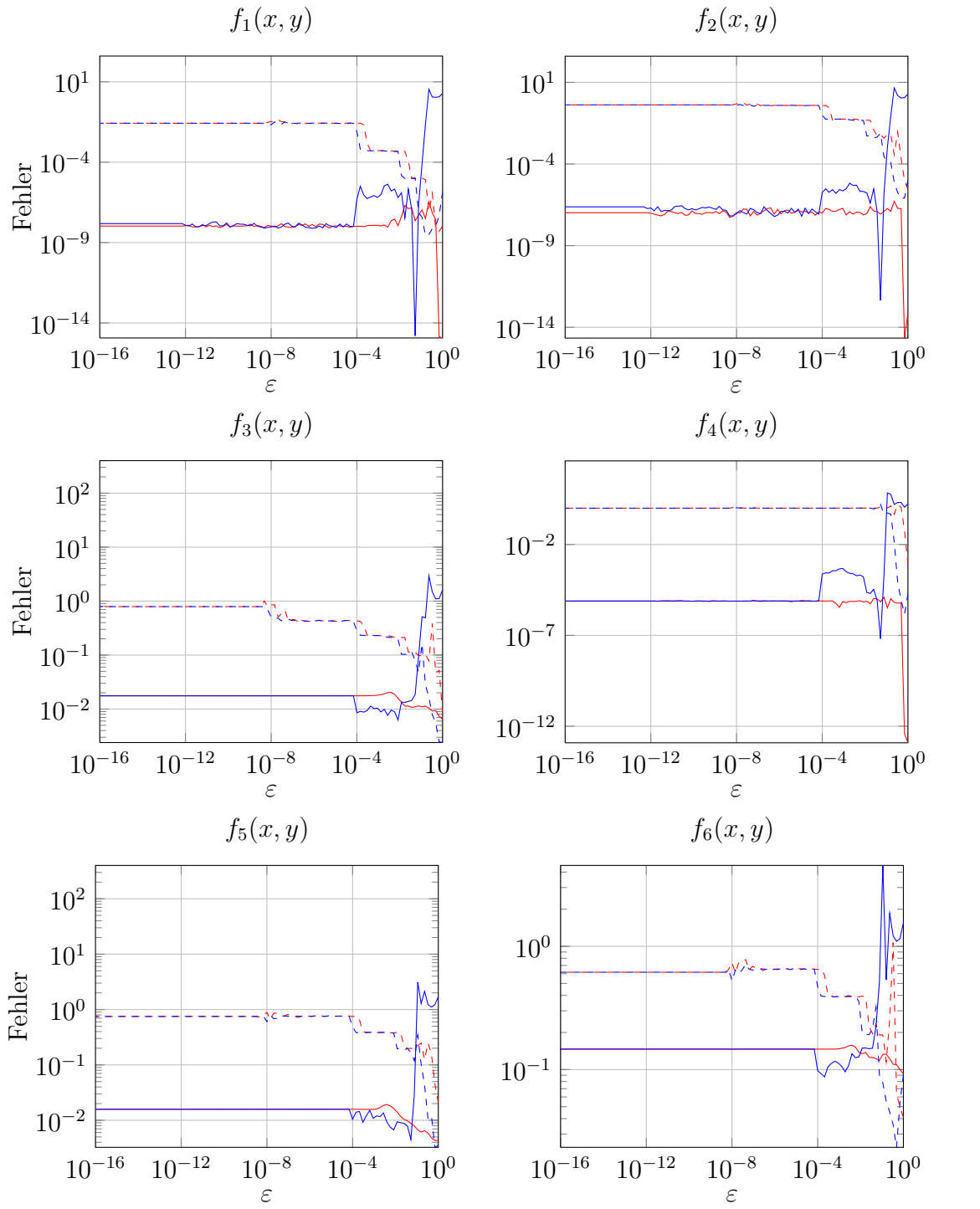

Abbildung 6.33: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 400 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Relative Fehler der β-IMQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 900$ und $\beta = 9/2$

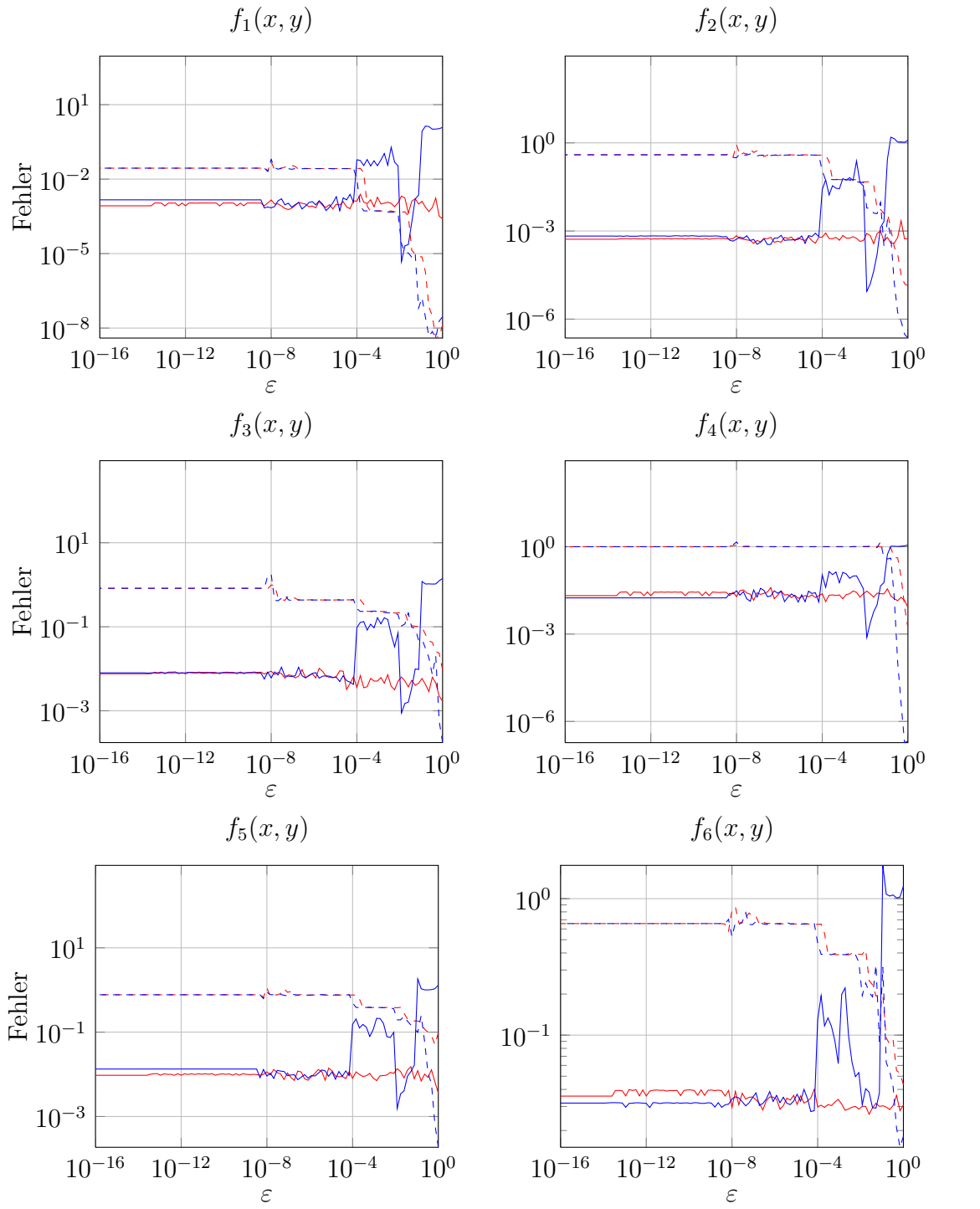

Abbildung 6.34: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 900 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100, 400 **und** 900 **Interpolationspunkte auf dem Quadrat**

Auf dem Quadrat werden die Konditionszahlen der Interpolationsmatrizen für den direkten Algorithmus und $\varepsilon \to 0$ ebenfalls sehr groß. Die Konditionszahlen des RBF-QR und RBF-QR-GQ- Algorithmus stabilisieren sich wie beim Kreis, sind für große n und $\varepsilon > 10^{-8}$ allerdings größer als die des direkten Algorithmus.

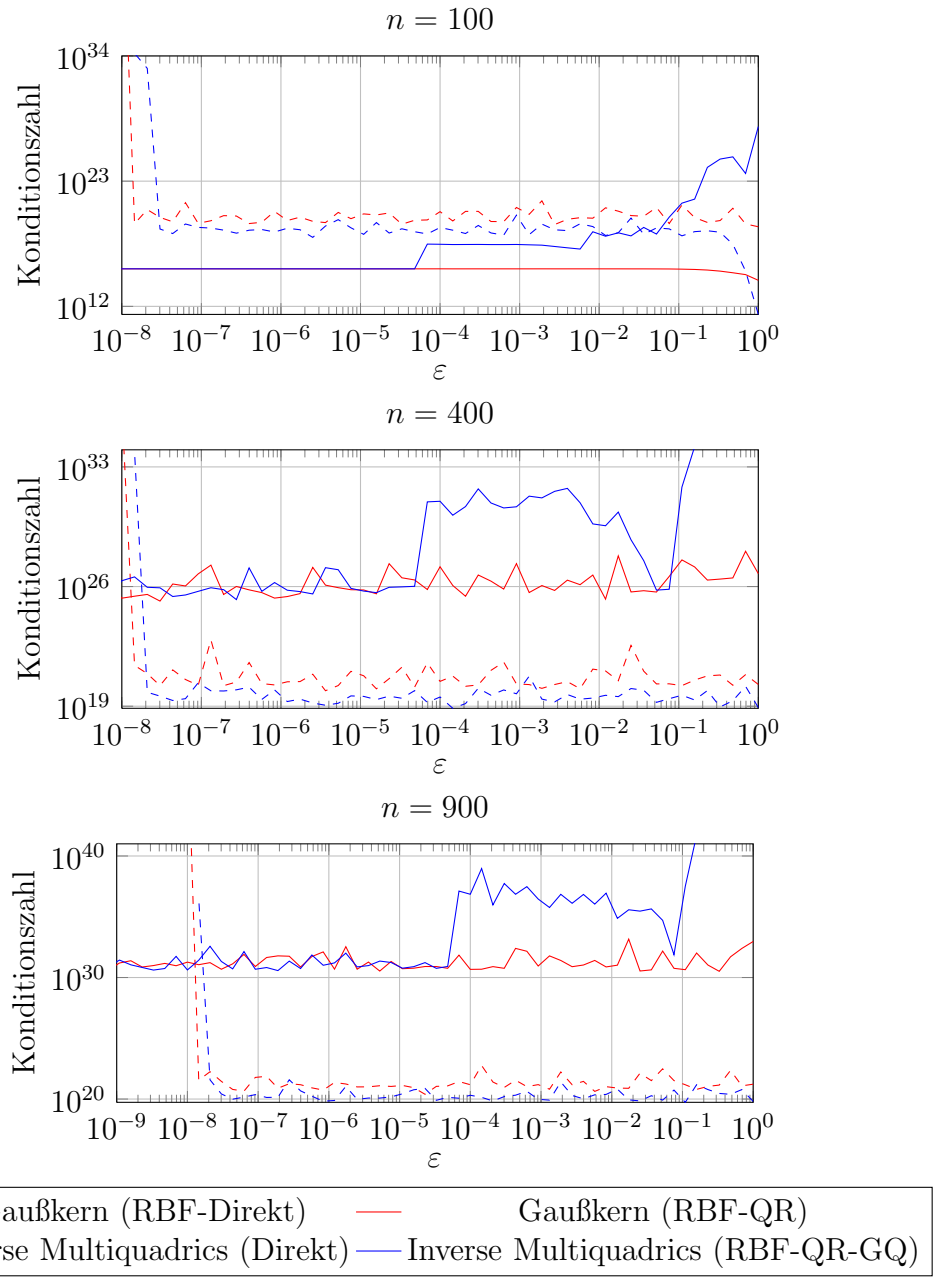

Abbildung 6.35: Konditionszahl bzgl. der GA- und bIMQ-RBF für das Quadrat.

Relative Fehler der β-IMQ für $f_1 - f_6$ auf dem Quadrat und $\varepsilon \to 0$ mit $n = 100$ und $\beta = 9/2$

Für $n = 100$ zeigen sich auf dem Quadrat analoge Ergebnisse wie auf dem Kreis. Auffallend sind dabei die Einbrüche in den relativen Fehler für einzelne ε-Werte und die sehr hohen Fehler für f_3, f_5 und f_6.

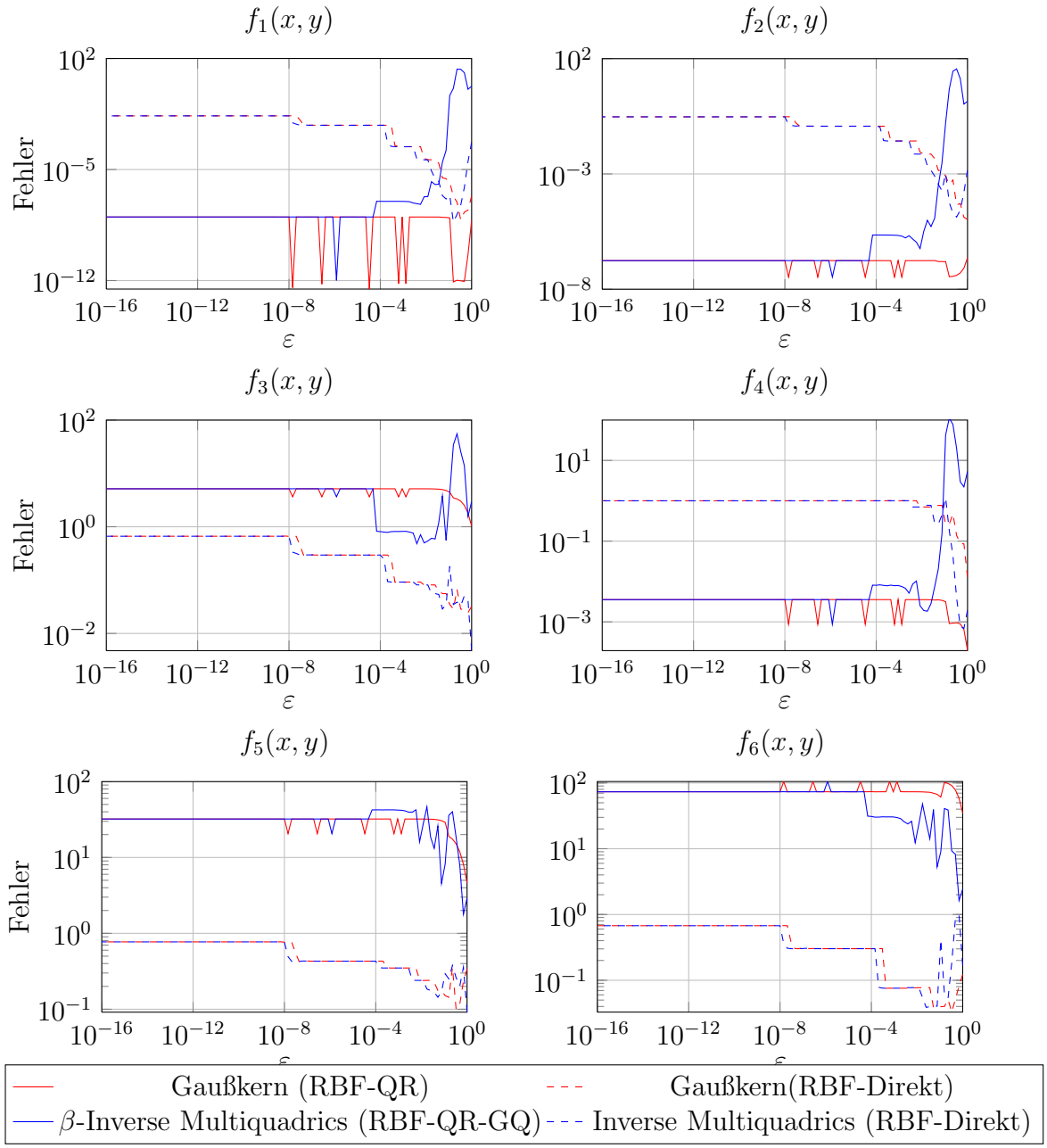

Abbildung 6.36: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 900 Interpolationspunkte und 10.000 Auswertungspunkte auf $[0,1]^2$.

Relative Fehler der β-IMQ für $f_1 - f_6$ und $\varepsilon = 10^{-3}$ mit $n = 100$

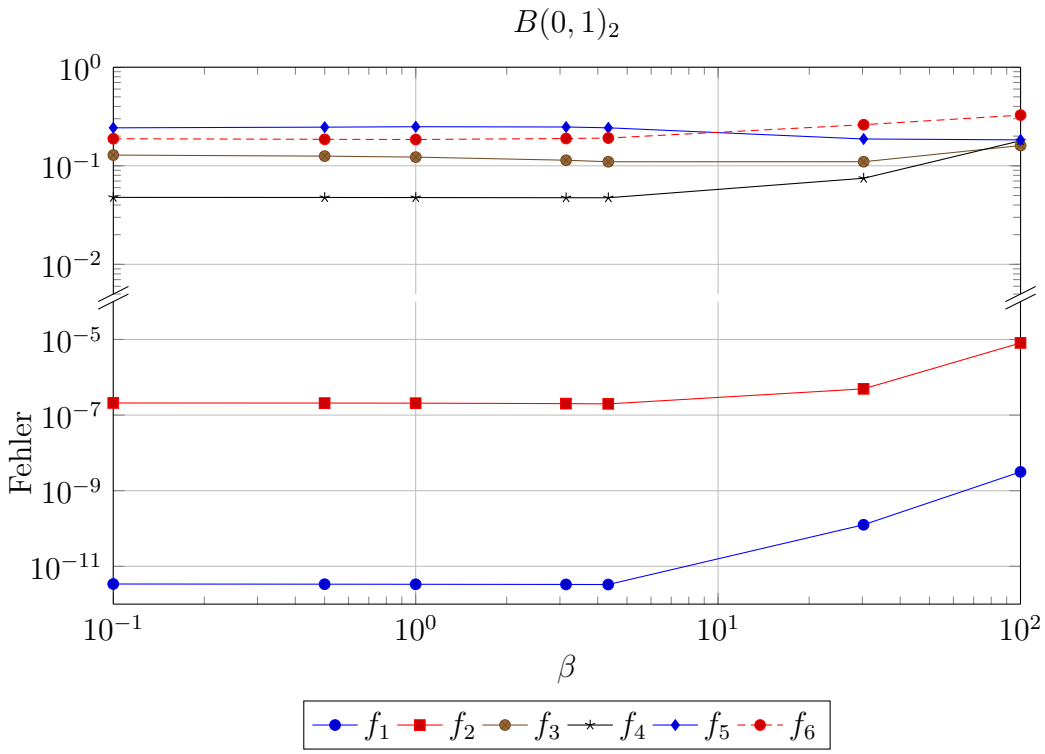

Abbildung 6.37: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $B(0,1)_2$ für verschiedene β-Werte.

Für $\varepsilon = 10^{-3}$ und größer werdendes β steigt auch der relative Fehler an. Wir erhalten für die einzelnen Testfunktionen analoge Ergebnisse wie für die IMQ für $n = 100$.

Abbildung 6.38 zeigt die gleichen Funktionen auf dem Quadrat $[0,1]^2$. Die relativen Fehler für f_1, f_2 und f_4 sind für relativ kleine β unter 1%, für die restlichen Funktionen ist die Interpolation nicht verwendbar für dieses gewählte ε.

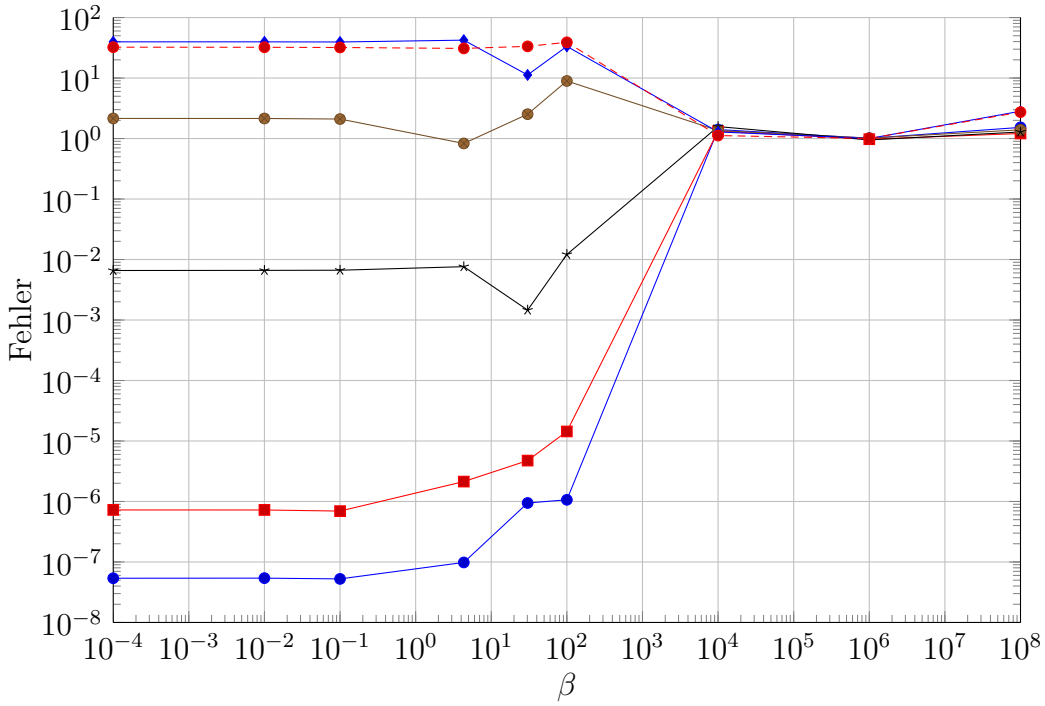

Abbildung 6.38: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $[0,1]^2$ für verschiedene β-Werte.

Die Multiquadrics

Der RBF-QR-GQ-Algorithmus liefert für die Multiquadrics ebenfalls gute Ergebnisse, insbesondere für $\varepsilon \to 0$.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100, 400 **und** 900 **Interpolationspunkte auf dem Kreis**

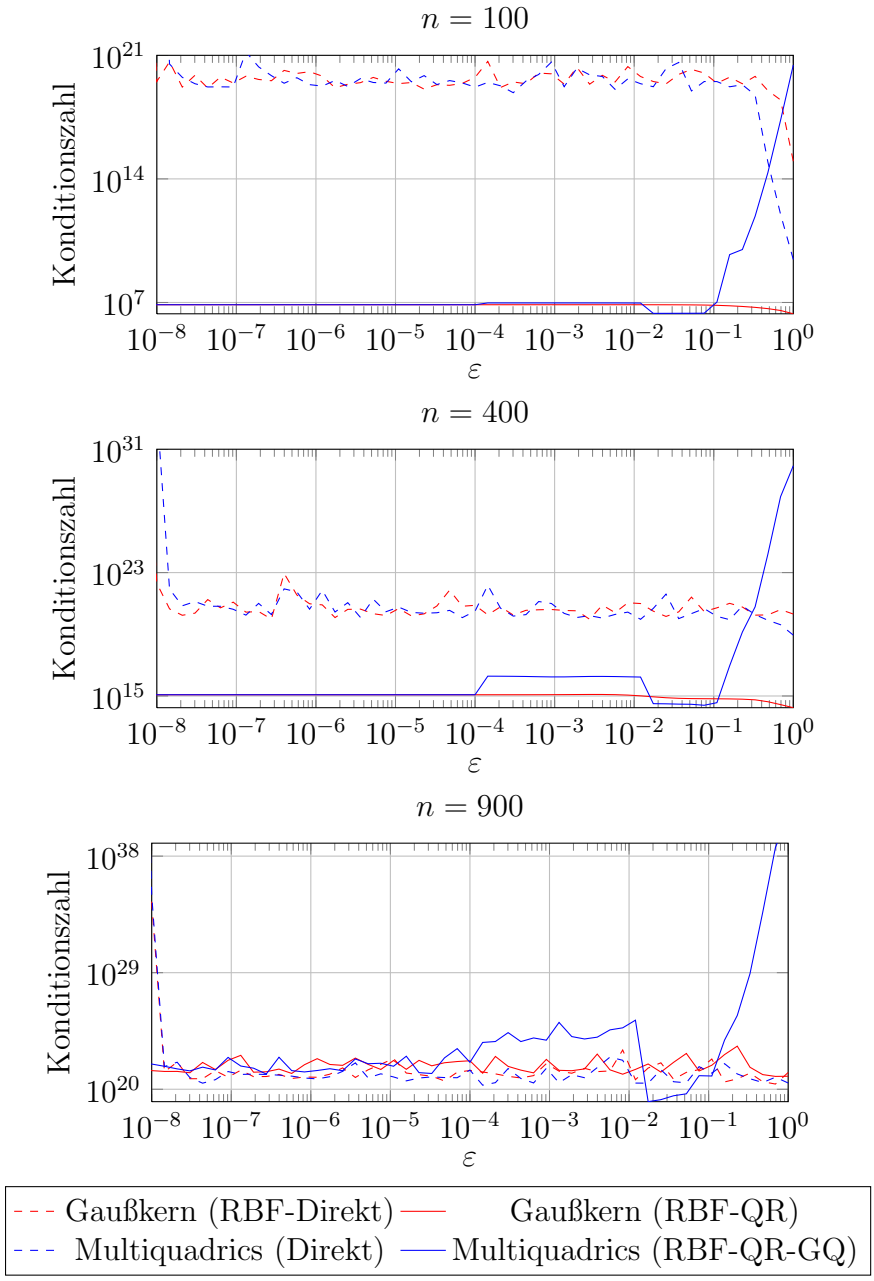

Abbildung 6.39: Konditionszahl bzgl. der GA- und MQ-RBF.

Relative Fehler der MQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 100$

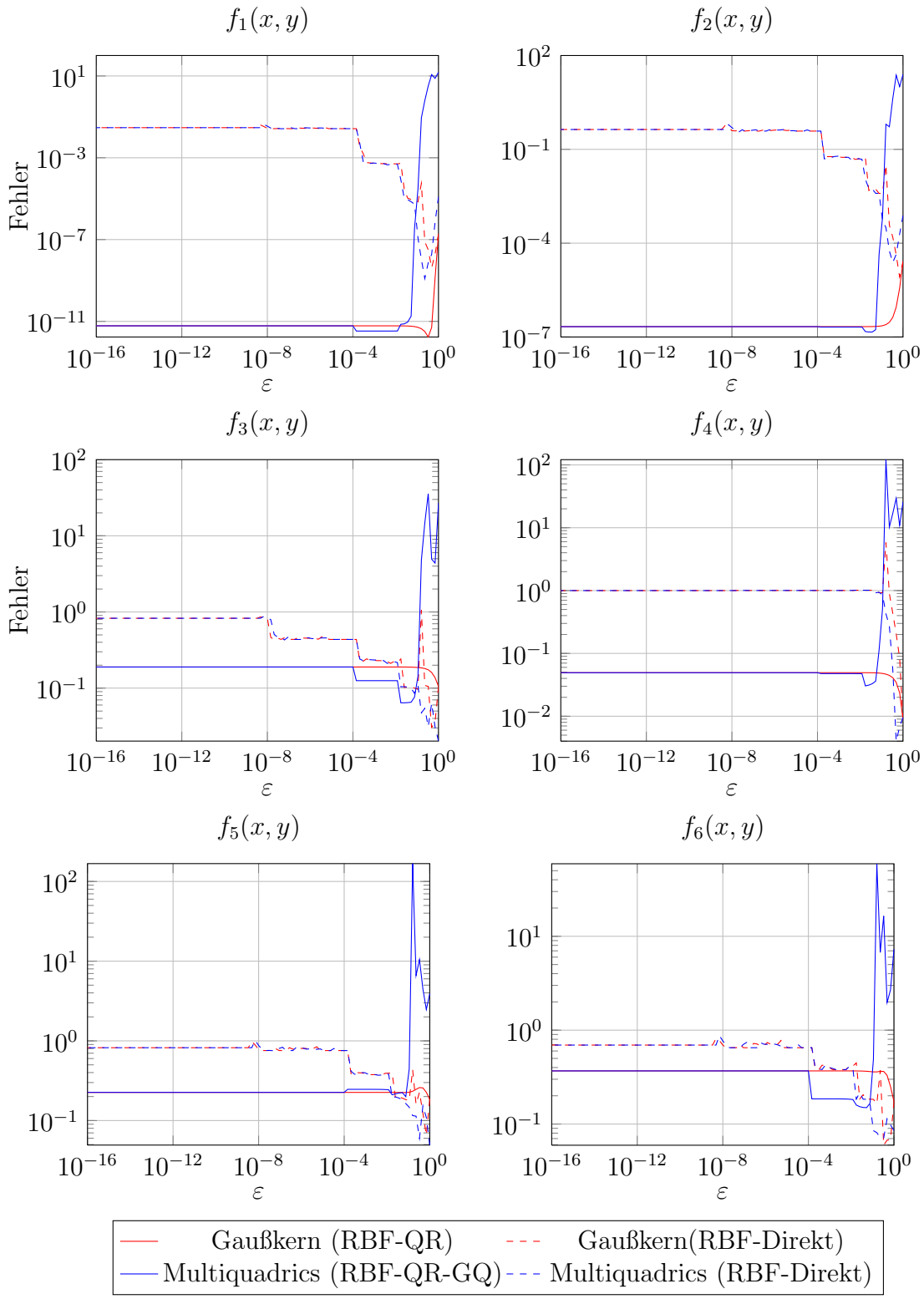

Abbildung 6.40: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte.

Relative Fehler der MQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 400$

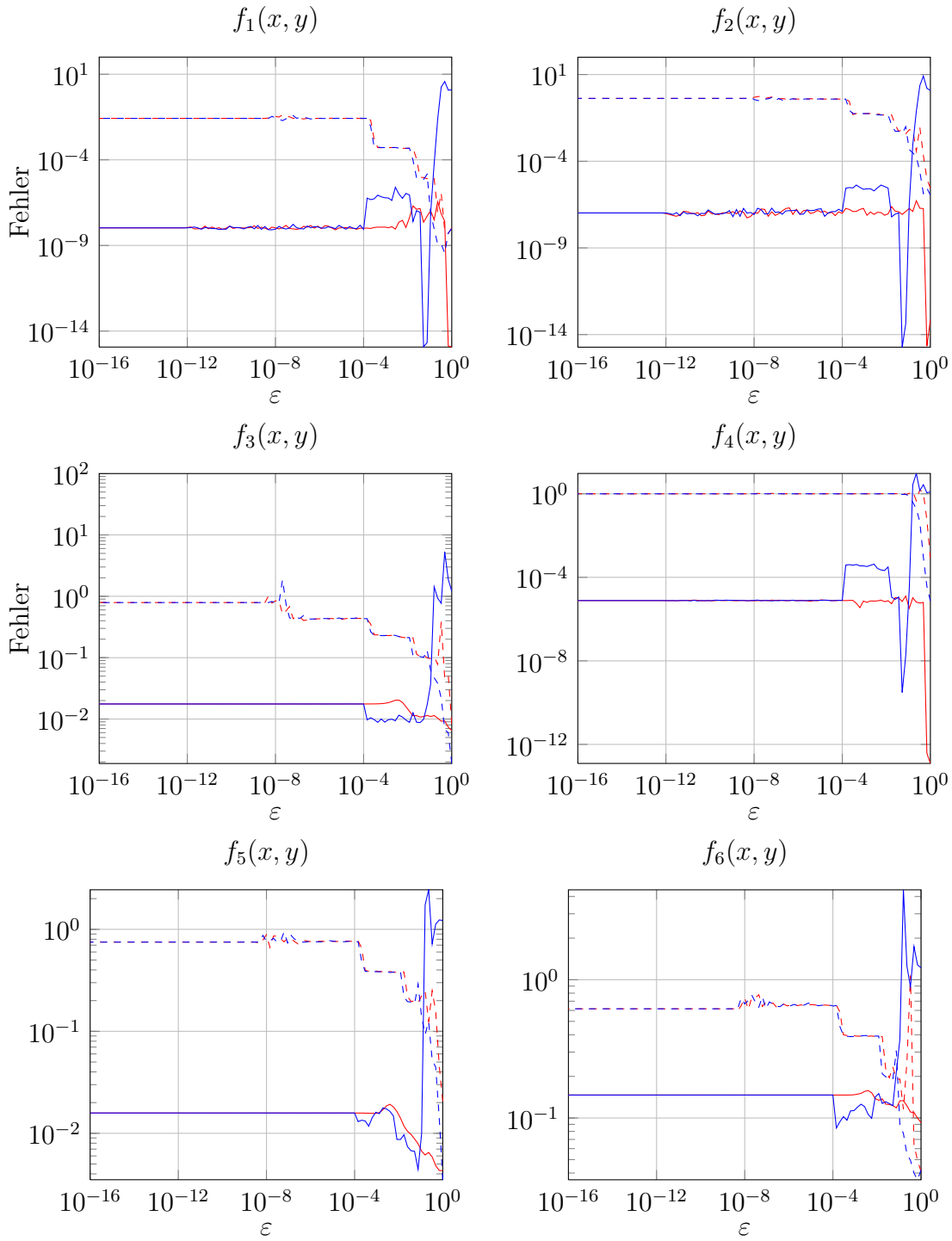

Abbildung 6.41: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 400 Interpolationspunkte und 10.000 Auswertungspunkte.

Relative Fehler der MQ für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$ mit $n = 900$

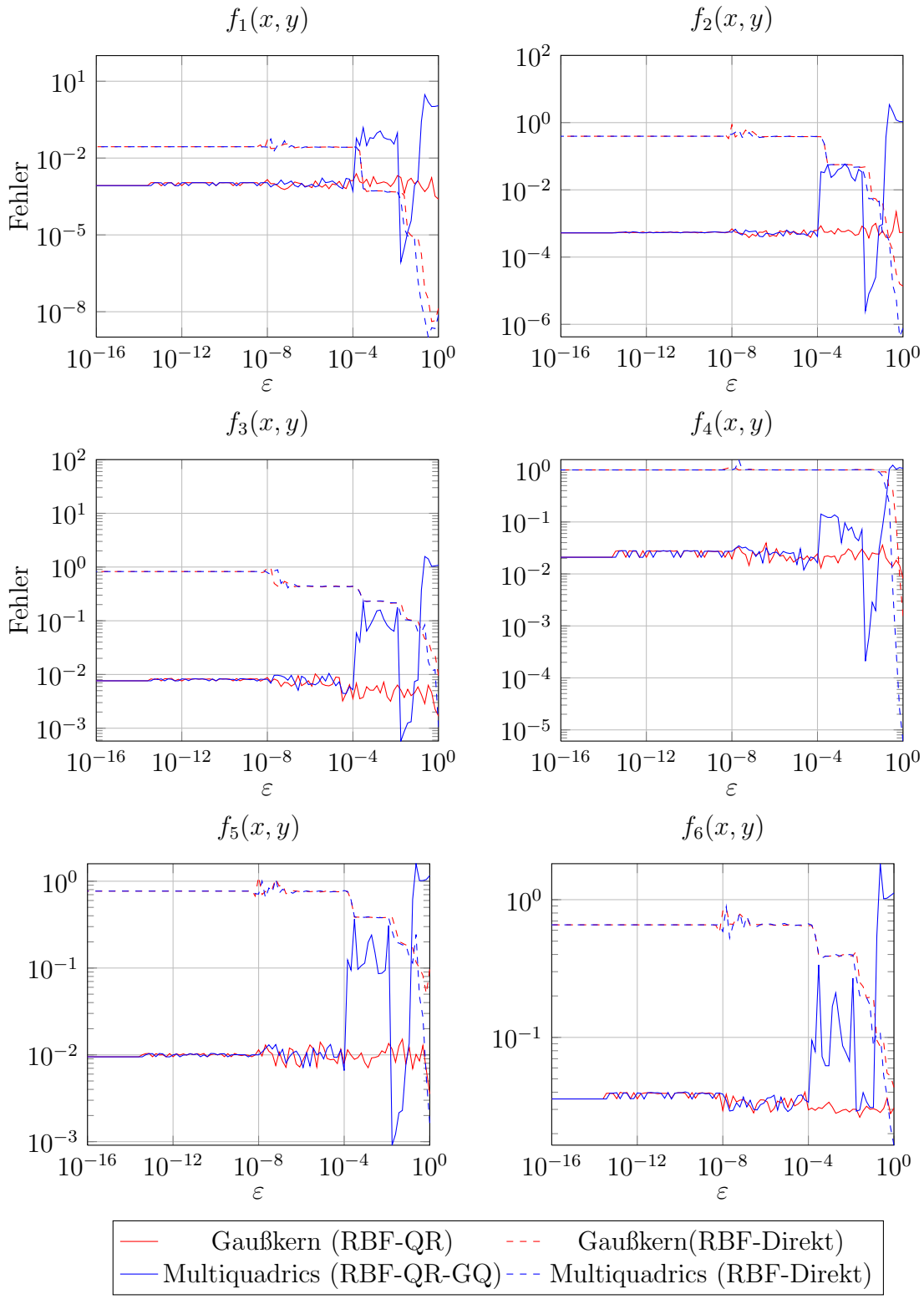

Abbildung 6.42: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 900 Interpolationspunkte und 10.000 Auswertungspunkte.

Vergleich der Konditionszahlen der Interpolationsmatrizen für 100, 400 **und** 900 **Interpolationspunkte auf dem Quadrat**

Auf dem Quadrat werden die Konditionszahlen der Interpolationsmatrizen für den direkten Algorithmus und $\varepsilon \to 0$ ebenfalls sehr groß. Die Konditionszahlen des RBF-QR und RBF-QR-GQ- Algorithmus stabilisieren sich wie beim Kreis, sind für große n und $\varepsilon > 10^{-8}$ allerdings größer als die des direkten Algorithmus.

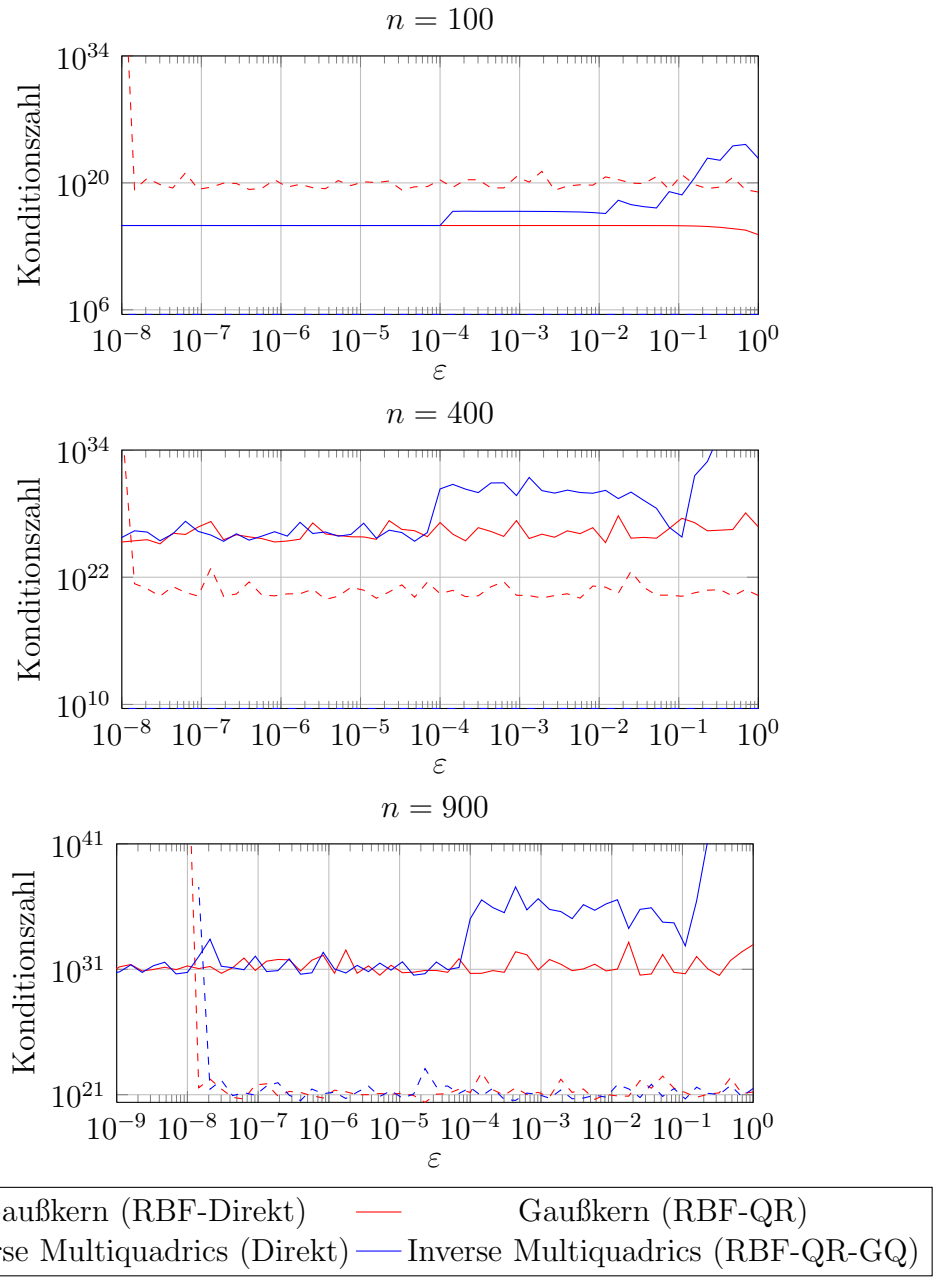

Abbildung 6.43: Konditionszahl bzgl. der GA- und MQ-RBF für das Quadrat.

Relative Fehler der MQ für $f_1 - f_6$ auf dem Quadrat und $\varepsilon \to 0$ mit $n = 100$

Für $n = 100$ zeigen sich auf dem Quadrat analoge Ergebnisse wie auf dem Kreis. Auffallend sind dabei wieder die Einbrüche in den relativen Fehler für einzelne ε-Werte und die sehr hohen Fehler für f_3, f_5 und f_6.

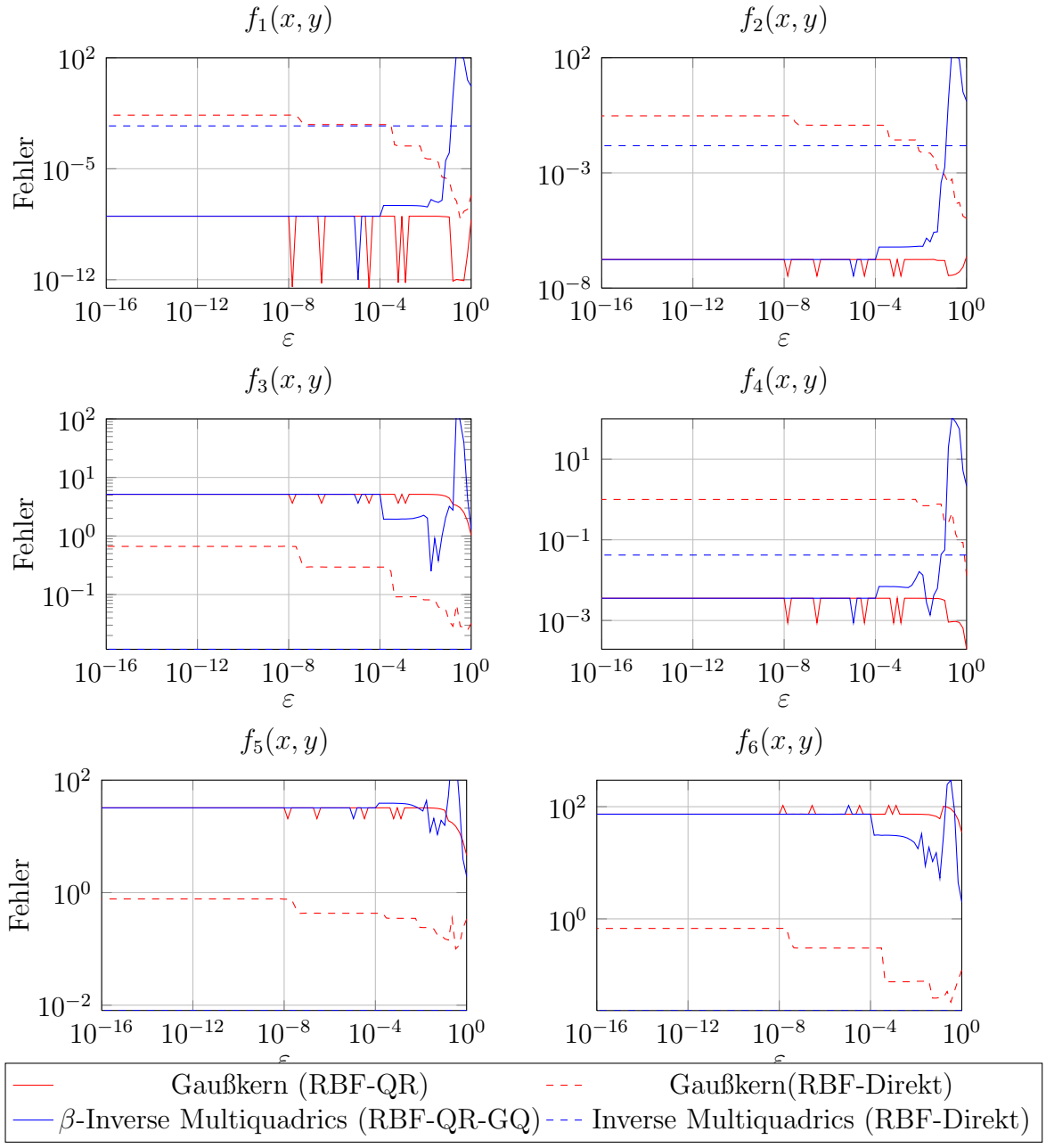

Abbildung 6.44: Interpolationsfehler in der 2-Norm für $f_1 - f_6$ mit 100 Interpolationspunkte und 10.000 Auswertungspunkte auf $[0,1]^2$.

6.4 Der RBF-QI-Algorithmus

Für die numerische Berechnung verwenden wir die gleichen Testfunktionen wie im vorherigen Kapitel (siehe Tabelle (4.1), Abbildungen 6.2 und 6.3). Wir beschränken uns in der Auswertung wieder auf den Einheitskreis $B(0,1)_2 \subset \mathbb{R}^2$ und das Quadrat $[0,1]^2$. Für die Quasi-Interpolation benötigen wir nun das Gitter

$$\Omega_h^K(\omega) := \{j \in \mathbb{Z}^2 | jh \in B(0, 1+\omega)_2\} \tag{6.4.14}$$

$$\text{bzw. } \Omega_h^Q(\omega) := \{j \in \mathbb{Z}^2 | jh \in [0-\omega, 1+\omega]^2\} \tag{6.4.15}$$

zur Feinheit $h \in (0,1)$ und Randerweiterung $\omega \in \mathbb{R}^+$. Die Feinheit gibt dabei die Entfernung der einzelnen Punkte zueinander an. Der Parameter ω erweitert den Kreis bzw. das Quadrat um ein zusätzliches Randgebiet, damit die Approximation am Rand stabilisiert und verbessert wird. Beispiele für die Punkteverteilung eines Gitters mit $h = \frac{1}{10}$ und $\omega = 1$ sowie $h = \frac{1}{8}$ und $\omega = 2$ sind in Abbildung 6.45 dargestellt. Die Gitterpunkte erhalten wir über das Programm `RBF_QI_Gitter.m` (siehe Quelltext A.7), die Auswertung erfolgt über `RBF_QI_TPS_1.m` bzw. `RBF_QI_TPS_2.m` zu den analytisch berechneten mit den Koeffizienten μ as 5.2.45 bzw. 5.2.54 zu den Punkten P (5.2.42).

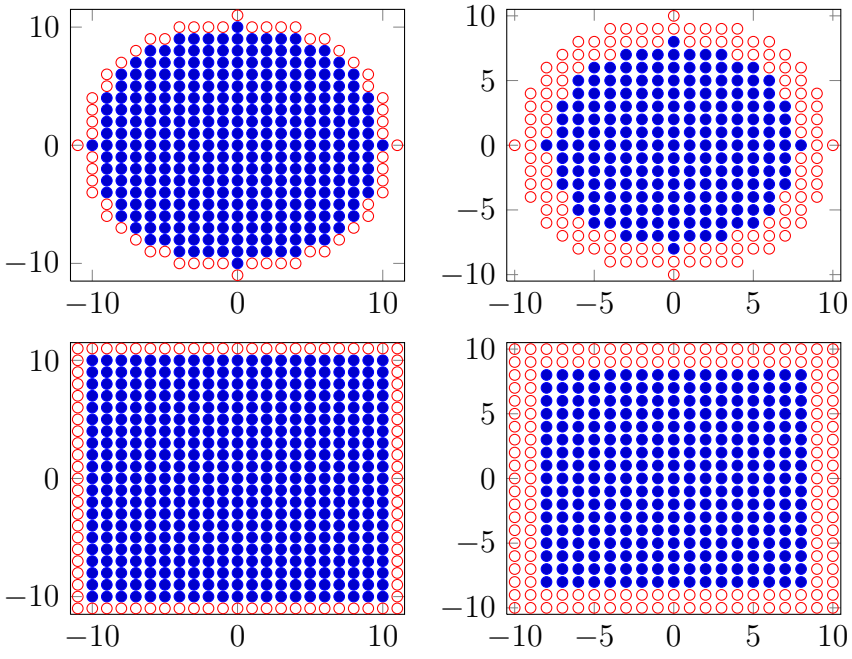

Abbildung 6.45: Verteilung der Gitterpunkte für $h = \frac{1}{10}$ und $\omega = 1$ sowie $h = \frac{1}{8}$ und $\omega = 2$ für $B(0,1)$ (blau) bzw. $\Omega_h^K \backslash B(0,1)$ (rot) und $[0,1]^2$ (blau) bzw. $\Omega_h^Q \backslash [0,1]^2$ (rot).

Für die Quasi-Interpolante benutzen wir (5.2.46) und (5.2.55) mit den Koeffizienten (5.2.45) bzw. (5.2.54) zu den Punkten P_1 (5.2.42) und gehen nach Algorithmus 2 vor. Wir setzen $h = \frac{1}{8}$ und $\omega = 1$ und vergleichen den Approximationsfehler wieder mit dem des direkten Interpolationsalgorithmus aus Kapitel 2.2 für $n = 100$ Interpolationspunkte mit den Verteilungen aus Abbildung 6.4 für $B(0,1)_2$ und $[0,1]^2$.

Folgende Werte erhalten wir für Testfunktion f_5 (Franke's Funktion) für den Kreis.

c	ϵ_1^Q	ϵ_1^I	ε	ϵ_2^Q	ϵ_2^I
10^2	9.79819×10^{-1}	2.57426	10^{-2}	9.64904×10^{-1}	2.52093×10^{1}
10	3.46869×10^{-1}	6.10972×10^{1}	10^{-1}	2.92705×10^{-1}	6.40837×10^{1}
1	2.07487×10^{-2}	1.27631×10^{3}	1	4.45080×10^{-2}	1.27631×10^{3}
10^{-1}	1.16473×10^{-2}	1.48695	10	1.13848×10^{-2}	8.52974×10^{-1}
10^{-2}	1.14828×10^{-2}	1.51378	10^2	1.14801×10^{-2}	6.22248×10^{-1}
10^{-4}	1.14810×10^{-2}	1.51361	10^4	1.14810×10^{-2}	5.03959×10^{-1}
10^{-6}	1.14810×10^{-2}	1.51361	10^6	1.14810×10^{-2}	4.68531×10^{-1}
10^{-8}	1.14810×10^{-2}	1.51361	10^8	1.14810×10^{-2}	4.51647×10^{-1}

Dabei bezeichne ϵ_1^Q den relativen Fehler der Quasi-Interpolation und ϵ_1^I den der direkten Interpolation für die sTPS 1. Art, ϵ_2^Q und ϵ_2^I sind die entsprechende Fehler der sTPS 2. Art.

Die Quasi-Interpolation approximiert die Funktion um einen konstanten Faktor im relativen Fehler besser als die direkte Interpolation. Außerdem ist die Stabilisierung für kleine c- und große ε-Werte erkennbar, die aufgrund der Definitionen dieser Basisfunktionen nicht überraschend sind.

Wir erhalten analoge Ergebnisse für das Quadrat $[0,1]^2$.

c	ϵ_1^Q	ϵ_1^I	ε	ϵ_2^Q	ϵ_2^I
10^2	9.78382×10^{-1}	3.22196×10^{-1}	10^{-2}	9.62702×10^{-1}	1.34197×10^{1}
10	4.41664×10^{-1}	1.71311×10^{-1}	10^{-1}	3.69573×10^{-1}	8.89833×10^{-1}
1	4.11064×10^{-2}	1.54289×10^{0}	1	5.36214×10^{-2}	1.54289×10^{0}
10^{-1}	2.35230×10^{-2}	1.62449×10^{-2}	10	2.29180×10^{-2}	1.23230×10^{-2}
10^{-2}	2.32003×10^{-2}	2.24056×10^{-2}	10^2	2.31941×10^{-2}	1.56203×10^{-2}
10^{-4}	2.31963×10^{-2}	2.27389×10^{-2}	10^4	2.31963×10^{-2}	1.52094×10^{-2}
10^{-6}	2.31963×10^{-2}	2.27390×10^{-2}	10^6	2.31963×10^{-2}	1.49787×10^{-2}
10^{-8}	2.31963×10^{-2}	2.27390×10^{-2}	10^8	2.31963×10^{-2}	1.48663×10^{-2}

Relative Fehler der sTPS 1. Art für $f_1 - f_6$ auf dem Kreis und $c \to 0$

Die relativen Fehler stabilisieren sich für $c \to 0$, sind aber für $c > 1$ größer als die der direkten Interpolation.

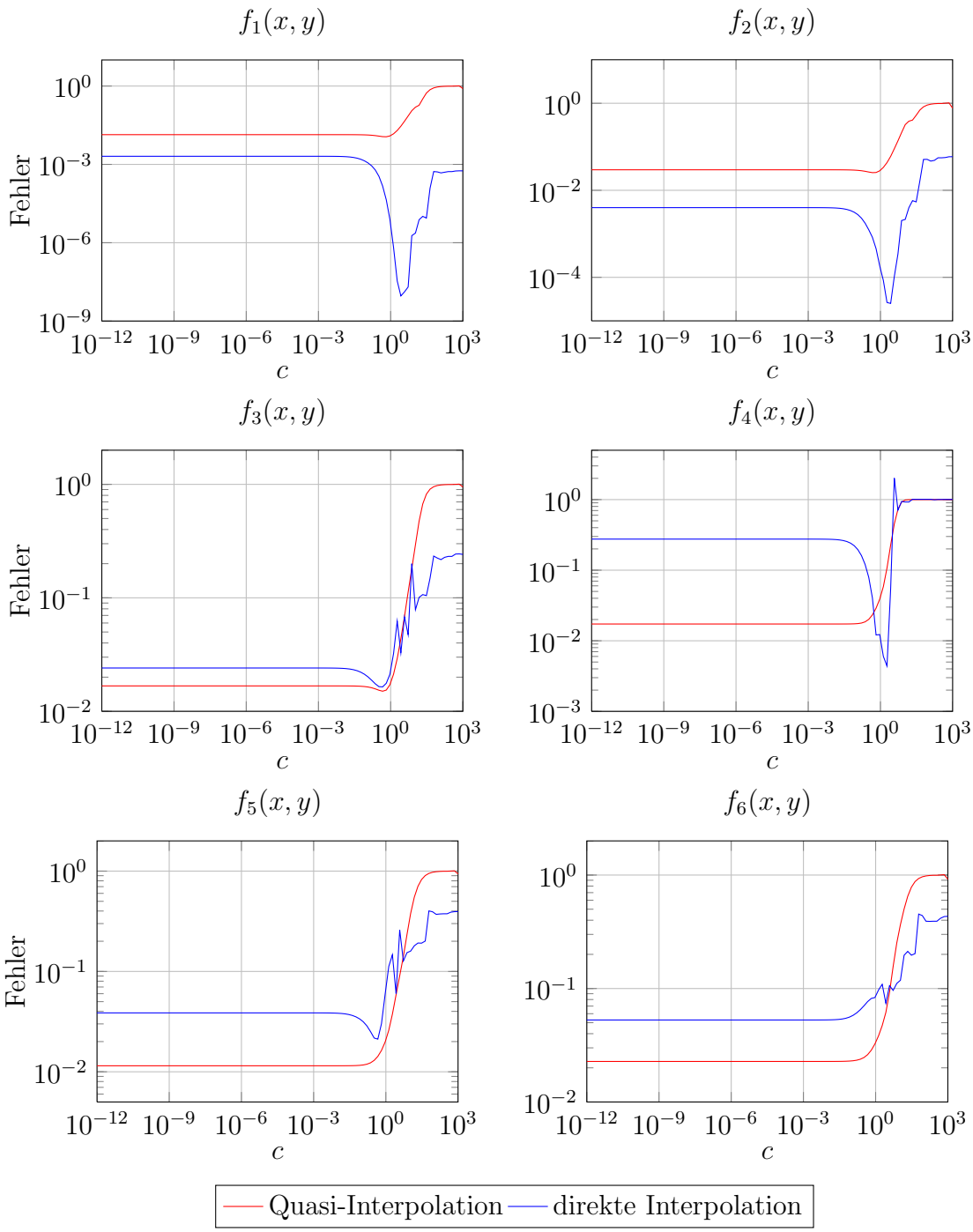

Abbildung 6.46: Relativer Fehler der Quasi-Interpolation mit $h = \frac{1}{8}$ und $\omega = 1$ und direkter Interpolation mit 100 Interpolationspunkte auf dem Kreis für $f_1 - f_6$ und 625 Auswertungspunkte.

Relative Fehler der sTPS 1. Art für $f_1 - f_6$ auf dem Quadrat und $c \to 0$

Auf dem Quadrat ergeben sich ähnliche relative Fehler im Vergleich zur direkten Interpolation wie auf dem Kreis.

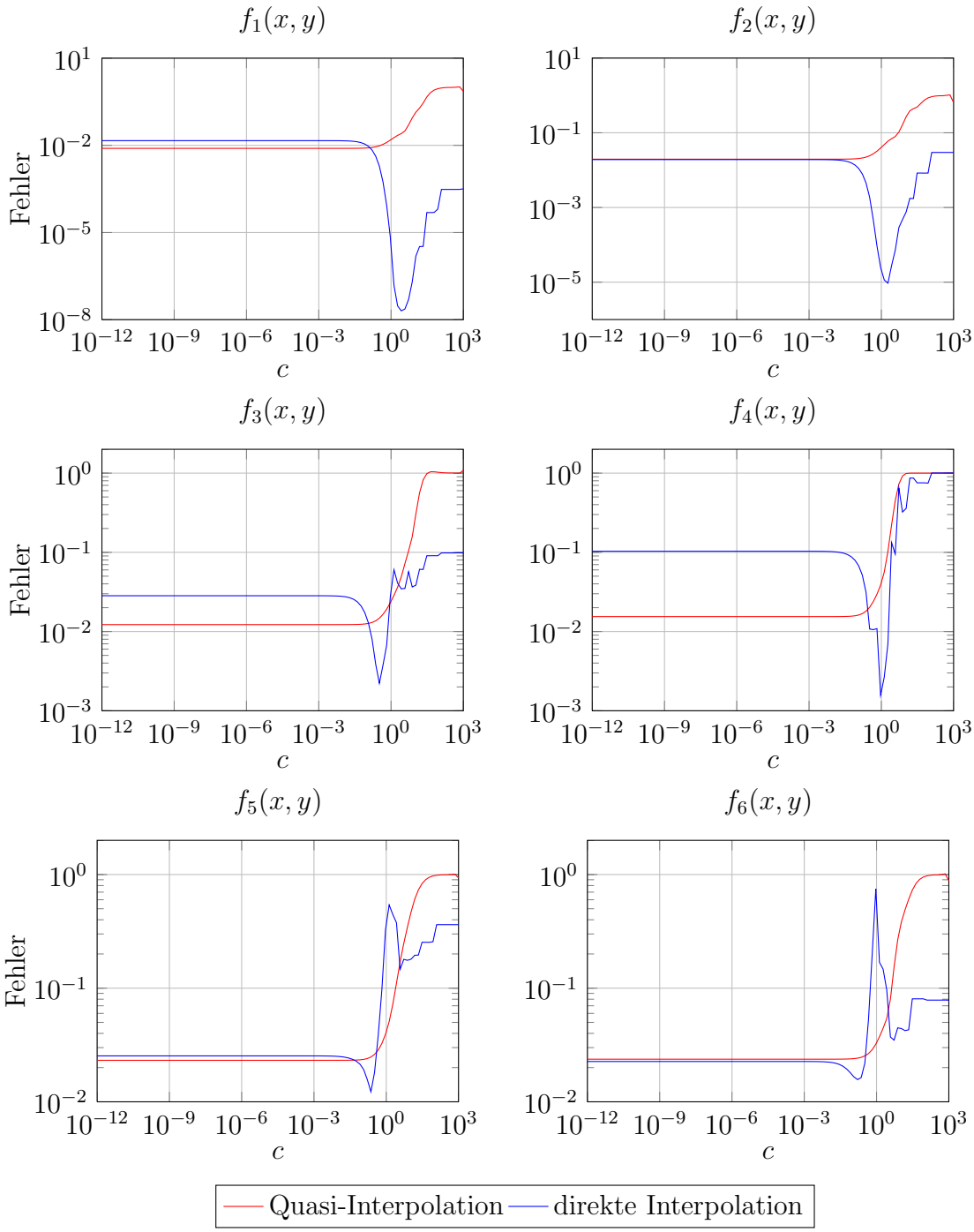

Abbildung 6.47: Relativer Fehler der Quasi-Interpolation mit $h = \frac{1}{8}$ und $\omega = 3$ und direkter Interpolation mit 100 Interpolationspunkte auf dem Quadrat für $f_1 - f_6$ und 625 Auswertungspunkte.

Relative Fehler der sTPS 2. Art für $f_1 - f_6$ auf dem Kreis und $\varepsilon \to 0$

Die relativen Fehler stabilisieren sich für $\varepsilon \to \infty$, sind aber für kleine $\varepsilon < 10^{-2}$ größer als die der direkten Interpolation.

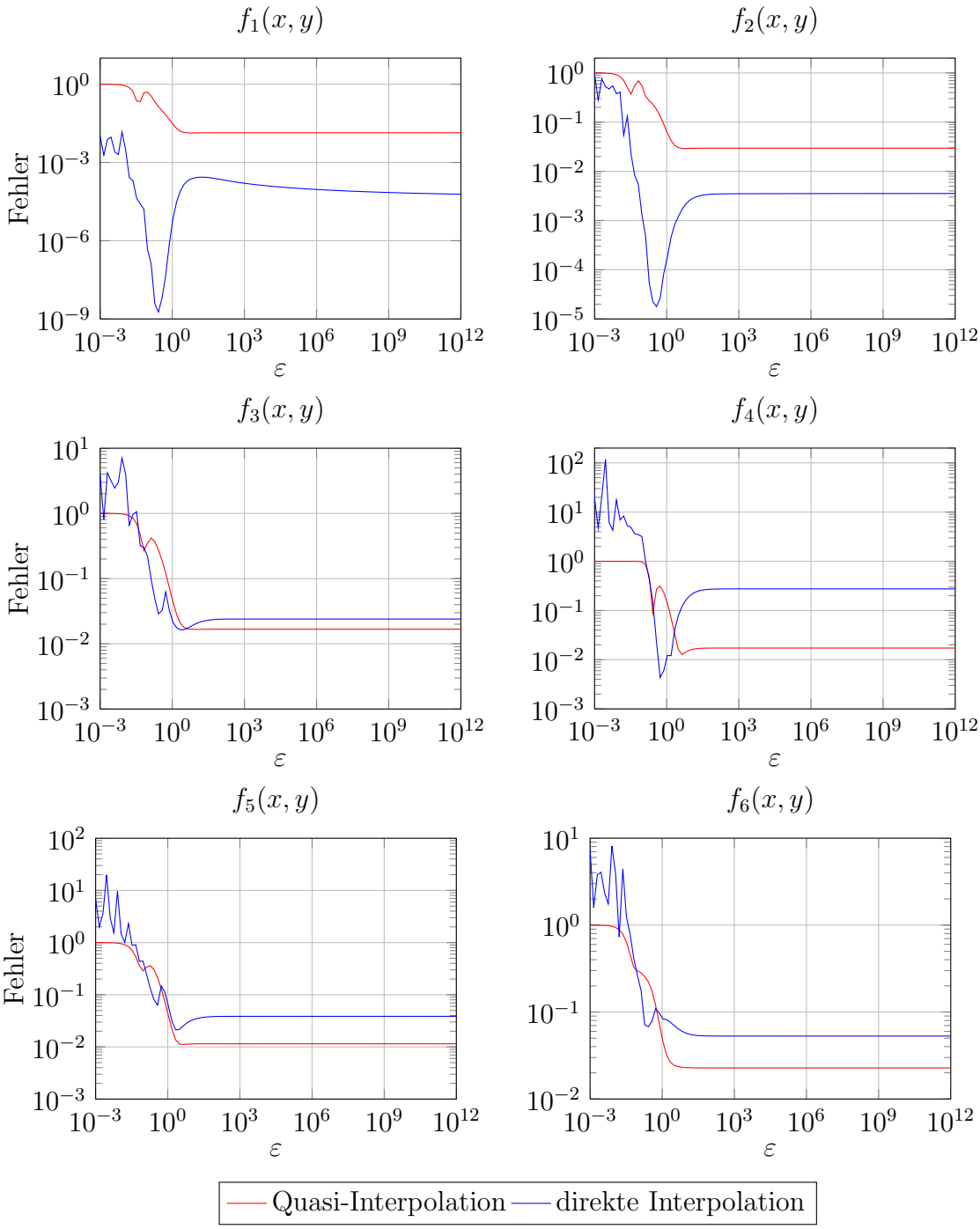

Abbildung 6.48: Relativer Fehler der Quasi-Interpolation mit $h = \frac{1}{8}$ und $\omega = 1$ und direkter Interpolation mit 100 Interpolationspunkte auf dem Kreis für $f_1 - f_6$ und 625 Auswertungspunkte.

Relative Fehler der sTPS 2. Art für $f_1 - f_6$ auf dem Quadrat und $\varepsilon \to 0$

Auf dem Quadrat ergeben sich ähnliche relative Fehler im Vergleich zur direkten Interpolation wie auf dem Kreis.

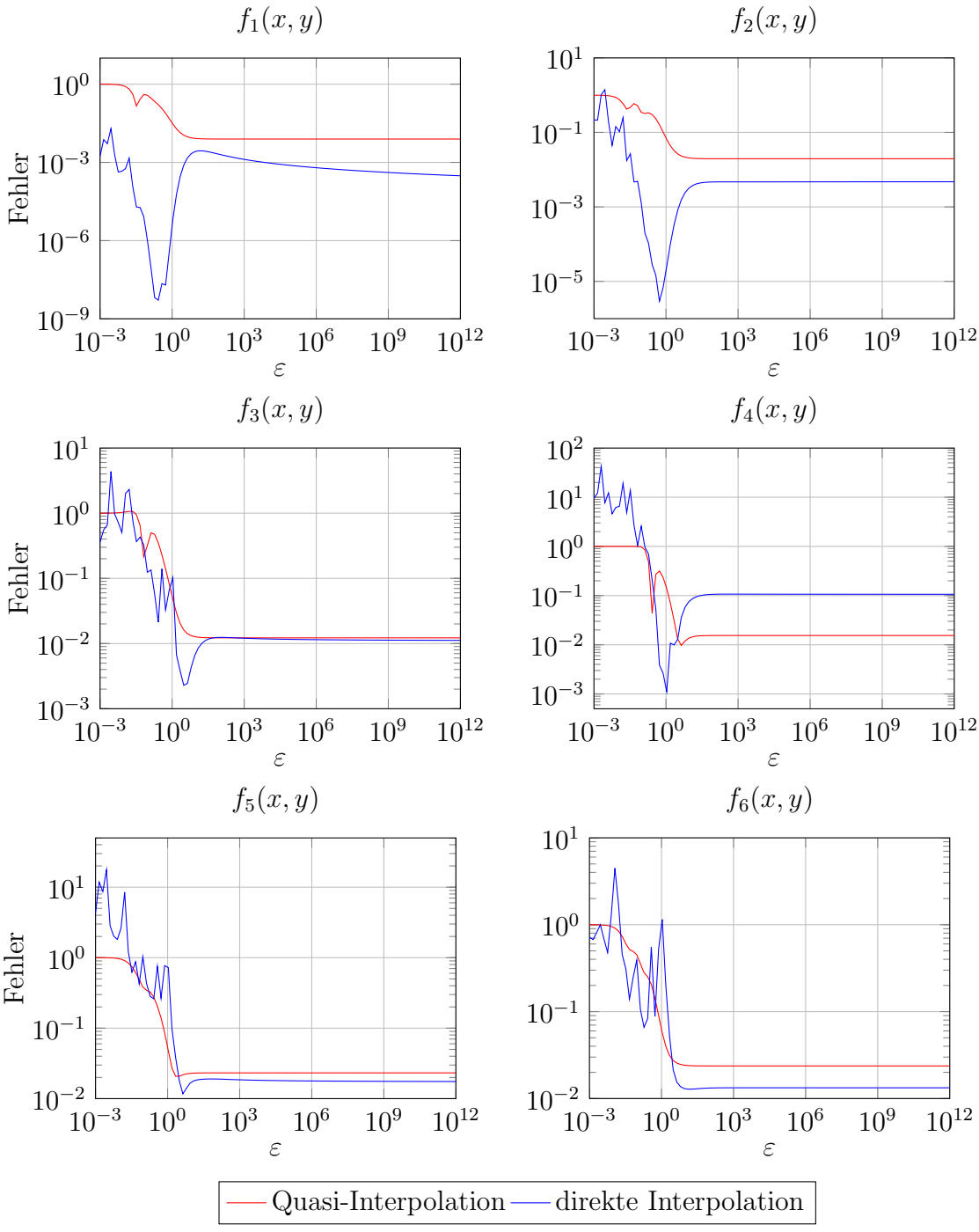

Abbildung 6.49: Relativer Fehler der Quasi-Interpolation mit $h = \frac{1}{8}$ und $\omega = 1$ und direkter Interpolation mit 100 Interpolationspunkte auf dem Quadrat für $f_1 - f_6$ und 625 Auswertungspunkte.

Kapitel 7

Zusammenfassung und Ausblick

Sowohl der RBF-QR-GQ-Algorithmus in Kapitel 4.2 als auch der RBF-QI-Algorithmus in Kapitel 5 zeigen einen geringen Approximationsfehler in der numerischen Berechnung im Vergleich zur direkten Interpolation. Außerdem sind durch die Verwendung der Dagumfunktionen $\varphi_{\beta,\gamma}$ neue Basisfunktionen zur Interpolation gegeben.

Für die Ableitung der Dagumfunktionen haben wir eine Erweiterung der hinreichenden Bedingungen für einen Spezialfall gezeigt. Insgesamt fehlt an dieser Stelle noch eine genau dann, wenn Aussage. Die numerische Untersuchungen zeigten außerdem, dass für die Interpolation eine stabile Methode für $\beta \to 0$ und $\gamma = 1/\beta$ benötigt wird.

Hinsichtlich des RBF-QR-GQ-Algorithmus kann sicherlich die Berechnung der zu berechnende Anzahl an Quadraturknoten in Abhängigkeit des vorliegenden ε-Wertes verbessert werden. Zudem fehlt noch eine Erweiterung auf weitere radiale Basisfunktionen, die eine ähnliche Maßfunktion wie die verwendeten Basisfunktionen besitzen, oder - und das wäre von größerem Vorteil, da auf mehr Basisfunktionen anwendbar - die Erweiterung des Algorithmus auch auf andere Maßfunktionen, die keine Terme der Form aus Theorem (4.1.2) besitzen. Dies würde aber auch eine Änderung der Reihenentwicklung des ursprünglichen RBF-QR-Algorithmus bedeuten. Des weiteren kann der RBF-QR-GQ-Algorithmus noch auf 1- und 3-dimensionale Gebiete erweitert werden, wenn die in [6] verwendete Reihenentwicklung mit Erweiterung der Basisfunktionen auf 1-dimensionale und sphärische Tschebyscheffpolynome entsprechend der Betrachtungen in Kapitel 4.2 angepasst wird. Zudem fehlt noch ein Kriterium zur Entscheidung, für welche ε-Werte der RBF-QR-GQ-Algorithmus stabiler und besser ist als die direkte Interpolation.

Der RBF-QI-Algorithmus kann ebenfalls zu kleineren Approximationsfehlern führen. Hier ist eine Erweiterung auf andere radiale Basisfunktionen, zum Beispiel auch

auf die Dagum-Funktionen, oder andere, bisher nicht betrachtete, radiale Basisfunktionen sinnvoll.

Anhang A

Quelltexte

A.1 RBF-Direkt-Algorithmus

```
% RBF_Direkt_2D.m
%
% Direkte Interpolation mit radialen Basisfunktionen.
%
% Eingabe:
%   APkt(1:2,1:N)  - Auswertungspunkte in kartesische Koordinaten
%   IPkt(1:2,1:Ni) - Interpolationspunkte in kartesische Koordinaten
%   f(1:Ni,1:Nf)   - Interpolationswerte zu Nf Funktionen
%   rbf(String)    - radiale Basisfunktion aus
%                    {'ga', 'imq', 'bimq', 'mq', 'dagum', 'tps', 'stps1', 'stps2'}
%   param(1-2)     - ein- oder zweidimensionaler Parameter der RBF
%                     * 'dagum' - [ beta, gamma ]
%                     * 'bimq'  - [ beta, epsilon ]
%                     * sonst Glaettungsparameter [epsilon] bzw. [c]
%
% Ausgabe:
%   y(1:N,1:Nf)    - Werte der TPS an den Punkten APkt
%   kond           - Konditionszahl der Interpolationsmatrix
function [ y, kond ] = RBF_Direkt_2D( APkt, IPkt, f, rbf, param )
  % RBF holen
  switch rbf
    case {'ga'}
      if( length(param) < 1 )
        error('Bitte Glaettungsparameter angeben!');
      endif
      RBF = @(param,MPkt) exp( - param(1)^2 * MPkt );
    case {'imq'}
```

```
28        if ( length(param) < 1 )
29          error('Bitte Glaettungsparameter angeben!');
30        endif
31        RBF = @(param,MPkt) ( 1 + param(1)^2 * MPkt ).^(-1/2);
32      case {'bimq'}
33        if ( length(param) < 2 )
34          error('Bitte Glaettungsparameter und beta angeben!');
35        endif
36        RBF = @(param,MPkt) ( 1 + param(2)^2 * MPkt ).^(-param(1));
37      case {'mq'}
38        if ( length(param) < 1 )
39          error('Bitte Glaettungsparameter angeben!');
40        endif
41        RBF = @(param,MPkt) ( 1 + param(1)^2 * MPkt ).^(-1/2);
42      case {'dagum'}
43        if ( length(param) < 2 )
44          error('Bitte beta und gamma angeben!');
45        endif
46        RBF = @(param,MPkt) 1 - ( ( MPkt.^param(1) )./...
47                                ( 1 + MPkt.^param(1) ) ).^param(2);
48      case {'tps'}
49        RBF = @(param,MPkt) MPkt .* log( sqrt( MPkt ) );
50      case {'stps1'}
51        if ( length(param) < 1 )
52          error('Bitte Glaettungsparameter angeben!');
53        endif
54        RBF = @(param,MPkt) ( param(1)^2 + MPkt ) .* ...
55                                log( sqrt( param(1)^2 + MPkt ) );
56      case {'stps2'}
57        if ( length(param) < 1 )
58          error('Bitte Glaettungsparameter angeben!');
59        endif
60        RBF = @(param,MPkt) ( 1 + param(1)^2 * MPkt ) .* ...
61                                log( sqrt( 1 + param(1)^2 * MPkt ) );
62      otherwise
63        error ('Unbekannte RBF!');
64    endswitch
65    % Anzahl an Punkte
66    N = columns( APkt );
67    Ni = columns( IPkt );
68    % Differenzenmatrix der Interpolationspunkte
```

```
69    Pkt_x = repmat( IPkt( 1, : ),  Ni,  1  );
70    Pkt_y = repmat( IPkt( 2, : ),  Ni,  1  );
71    Diff_Mat_IPkt = ( Pkt_x − Pkt_x' ).^2 + ( Pkt_y − Pkt_y' ).^2;
72    % Differenzenmatrix der Interpolations− und Auswertungspunkte
73    APkt_x = repmat( APkt( 1, : ), Ni, 1 );
74    APkt_y = repmat( APkt( 2, : ), Ni, 1 );
75    IPkt_x = repmat( IPkt( 1, : ),  N, 1  );
76    IPkt_y = repmat( IPkt( 2, : ),  N, 1  );
77    Diff_Mat_APkt = ( APkt_x' − IPkt_x ).^2 + ( APkt_y' − IPkt_y ).^2;
78    % Interpolationsmatrix
79    Int_Mat = RBF( param, Diff_Mat_IPkt );
80    % Konditionszahl der Interpolationsmatrix
81    kond = cond( Int_Mat );
82    % Koeffizienten
83    mu = Int_Mat \ f;
84    % Auswerten
85    Mat = RBF( param, Diff_Mat_APkt );
86    y = Mat * mu;
87 endfunction
```

Quelltext A.1: RBF_Direkt_2D.m

A.2 Dagum-Funktionen

```
% RBFDirektDagumTest.m
%
% Test der Interpolation der Dagumfunktionen
%
% Die Interpolation verwendet n Eigenwert- (Einheitskreis)
% oder Halton-Punkte (Einheitskreisquadrat).
%
% Benoetigte Funktionen:
%   LadeTestfunktionen        - Testfunktion 1 - 6
%   BildeAuswertungspunkte - 2-dimensionale Auswertungspunkte
%   RBF_Direkt_2D             - Direkte Interpolantion
%   LadeInterpolationspunkte - Interpolationspunkte
%
% Vorzugebende Werte:
%   dbeta  - beta-Paramater
%   dganna - gamma-Parameter
%   n      - Anzahl an Interpolationspunkten
%   gebiet - Einheitskreis oder -quadrat ('circle', 'cube')
%   res    - Aufloesung der Auswertungspunkte, egibt ein
%            ( res x res ) - Gitter
%   method - Art der Interpolantionspunkte ('ev','md','mn')
%   nbr    - Nummer der Interpolantionspunkte
%            ('09' = 100, '19' = 400, '29' = 900);
%
% Literatur
%  [1] - Rene Hofmann, "Approximation mit radialen Basisfunktionen",
%        Dissertation, 2013
%  [2] - UNSW - The University of New South Wales,
%        "Interpolation and Cubature on the Sphere",
%        http://web.maths.unsw.edu.au/~rsw/Sphere/index.html
%        zuletzt abgerufen am 27.04.2013.
if( not(exist('dbeta')) ) dbeta = 1; endif
if( not(exist('dgamma')) ) dgamma = 1; endif
if( not(exist('n')) ) n = 100; endif
if( not(exist('gebiet')) ) gebiet = 'circle'; endif
if( not(exist('res')) ) res = 100; endif
if( not(exist('p1')) && not(exist('p2')) ) p1 = 7; p2 = 11; endif
% Testfunktionen
LadeTestfunktionen;
```

```
40 % Aufloesung des Gitters fuer die Auswertungspunkte
41 [ kart, pol ] = BildeAuswertungspunkte( res, gebiet );
42 xe = kart(1,:); ye = kart(2,:);
43 % Exakte Loesungswerte
44 E = [ tf1(xe,ye); tf2(xe,ye); tf3(xe,ye); tf4(xe,ye); tf5(xe,ye); tf6(xe,ye) ];
45 % Interpolationspunkte
46 switch ( gebiet )
47   case 'cube'
48     method = 'hd'; p = [p1,p2]; N = [n,n];
49   case ' circle '
50     % Punkte analog zu [2]
51     if( not(exist('method')) ) method = 'ev'; endif
52     if( not(exist('nbr')) ) nbr = '09'; endif
53   otherwise
54     break;
55 end
56 LadeInterpolationspunkte;
57 [x,y] = pol2cart( theta, r );
58 % Testfunktionen an den Interpolationspunkten auswerten
59 TF_x_y = [ tf1(x,y)' tf2(x,y)' tf3(x,y)' tf4(x,y)' tf5(x,y)' tf6(x,y)' ];
60 % Interpolation Dagum
61 [ Y, kond ] = RBF_Direkt_2D( [xe;ye], [x;y], TF_x_y, 'dagum', [dbeta,dgamma] );
62 fehler_di = norm( E - Y', 2, 'rows' )' ./ norm( E, 2, 'rows' )';
```

Quelltext A.2: RBFDirektDagumTest.m

A.3 RBF-QR-GQ-Algorithmus

```
% RBFQRGQTest.m
%
% Test der Interpolation mit dem RBF-QR-GQ-Algorithmus
%
% Die Interpolation verwendet n Eigenwert- (Einheitskreis)
% oder Halton-Punkte (Einheitskreisquadrat).
%
% Benoetigte Funktionen:
%   LadeTestfunktionen       - Testfunktion 1 - 6
%   BildeAuswertungspunkte - 2-dimensionale Auswertungspunkte
%   RBF_Direkt_2D            - Direkte Interpolantion
%   LadeInterpolationspunkte - Interpolationspunkte
%
% Vorzugebende Werte:
%   rbf       - radiale Basisfunktion aus
%               {'ga', 'imq', 'bimq', 'mq'}Peaks
%   gebiet    - Einheitskreis oder -quadrat ('circle', 'cube')
%   res       - Aufloesung der Auswertungspunkte, egibt ein
%               ( res x res ) - Gitter
%   n         - Anzahl an Interpolationspunkten
%   method - Art der Interpolantionspunkte ('ev','md','mn','hd')
%   nbr       - Nummer der Interpolantionspunkte
%               ('09' = 100, '19' = 400, '29' = 900);
%   epsilon - Vektor mir den auszuwertenden epsilon-Werten
%   optpar  - beta-Parameter fuer 'bimq'
%
% Literatur
% [1] - Rene Hofmann,
%       Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%       numerische Untersuchung der Dagum-Funktionen',
%       Dissertation, 2013
% [2] - UNSW - The University of New South Wales,
%       "Interpolation and Cubature on the Sphere",
%       http://web.maths.unsw.edu.au/~rsw/Sphere/index.html
%       zuletzt abgerufen am 27.04.2013.
% [3] - Bengt Fornberg et al.,
%       'Stable Computations with Gaussian Radial Basis Functions in 2-D',
%       Uppsala University, Department of Information Technology, 2009
if( not(exist('rbf')) ) rbf = 'imq'; endif
```

```
40 if( not(exist('gebiet')) ) gebiet = 'circle'; endif
41 if( not(exist('res')) ) res = 100; endif
42 if( not(exist('n')) ) n = 100; endif
43 if( not(exist('p1')) && not(exist('p2')) ) p1 = 7; p2 = 11; endif
44 if( not(exist('epsilon')) ) epsilon = logspace(0,-16,101); endif
45 if( not(exist('optpar')) ) optpar = []; endif
46 % Testfunktionen
47 LadeTestfunktionen;
48 % Aufloesung des Gitters fuer die Auswertungspunkte
49 [ kart, pol ] = BildeAuswertungspunkte( res, gebiet );
50 xe = kart(1,:); ye = kart(2,:);
51 % Exakte Loesungswerte
52 E = [ tf1(xe,ye); tf2(xe,ye); tf3(xe,ye); tf4(xe,ye); tf5(xe,ye); tf6(xe,ye) ]';
53 % Interpolationspunkte
54 switch ( gebiet )
55   case 'cube'
56     method = 'hd'; p = [p1,p2]; N = [n,n];
57   case 'circle'
58     % Punkte analog zu [2]
59     if( not(exist('method')) ) method = 'ev'; endif
60     if( not(exist('nbr')) ) nbr = '09'; endif
61   otherwise
62     break;
63 end
64 LadeInterpolationspunkte;
65 [x,y] = pol2cart( theta, r ); n = length(x);
66 [theta_eval, r_eval] = cart2pol( xe, ye );
67 % Testfunktionen an den Interpolationspunkten auswerten
68 TF_x_y = [ tf1(x,y)' tf2(x,y)' tf3(x,y)' tf4(x,y)' tf5(x,y)' tf6(x,y)' ];
69 % auswerten
70 for it = 1 : length(epsilon)
71     [ f_rbf, kond_rbf ] = RBF_Direkt_2D( [xe;ye], [x;y], TF_x_y, rbf, ...
72                                 [optpar, epsilon(it)] );
73     [ f_rbf_QR, kond_rbf_QR,qn ] = RBF_QR_GQ_2D( epsilon(it), [ r', theta'], ...
74                                         [r_eval', theta_eval'], TF_x_y, rbf, optpar );
75     [ f_GA, kond_GA ] = RBF_Direkt_2D( [xe;ye], [x;y], TF_x_y, 'ga', epsilon(it) );
76     [ f_GA_QR, kond_GA_QR ] = RBF_QR_2D( epsilon(it), [ r', theta'], ...
77                                         [r_eval', theta_eval'], TF_x_y );
78     Error_rbf_inf(:, it) = norm( E - f_rbf, inf, 'cols' );
79     Error_rbf_2(:, it) = norm( E - f_rbf, 2, 'cols' );
80     Error_rbf_QR_inf(:, it) = norm( E - f_rbf_QR, inf, 'cols' );
```

```
81      Error_rbf_QR_2(:,it) = norm( E - f_rbf_QR, 2, 'cols' );
82      Error_GA_inf(:,it) = norm( E - f_GA, inf, 'cols' );
83      Error_GA_2(:,it) = norm( E - f_GA, 2, 'cols' );
84      Error_GA_QR_inf(:,it) = norm( E - f_GA_QR, inf, 'cols' );
85      Error_GA_QR_2(:,it) = norm( E - f_GA_QR, 2, 'cols' );
86      Konditionszahl(it,1) = kond_GA;
87      Konditionszahl(it,2) = kond_GA_QR;
88      Konditionszahl(it,3) = kond_rbf;
89      Konditionszahl(it,4) = kond_rbf_QR;
90      q(it) = qn;
91  end
92  % Aufraeumen
93  Konditionszahl(find(Konditionszahl==Inf)) = NaN;
94  % relative Fehler
95  Error_GA_2_rel = Error_GA_2 ./ repmat( norm( E, 2, 'cols' ), ...
96                          columns(Error_GA_2), 1 )';
97  Error_GA_QR_2_rel = Error_GA_QR_2 ./ repmat( norm( E, 2, 'cols' ), ...
98                              columns(Error_GA_2), 1 )';
99  Error_rbf_2_rel = Error_rbf_2 ./ repmat( norm( E, 2, 'cols' ), ...
100                         columns(Error_GA_2), 1 )';
101 Error_rbf_QR_2_rel = Error_rbf_QR_2 ./ repmat( norm( E, 2, 'cols' ), ...
102                             columns(Error_GA_2), 1 )';
```

Quelltext A.3: RBFQRGQTest.m

```
% RBF_QR_GQ_2D.m
%
% RBF-QR-GQ Algorithmus fuer IMQ, beta-IMQ und MQ
% interpoliert die Funktionen f = (f1 ,..., fn)
% an den Stellen xe mit Knoten und Glaettungsparameter ep.
%
% Eingabe:
%   ep            - Glaettungsparameter
%   xk(1:N,1:2)   - Interpolationspunkte in Polarkoordinaten (r,theta)
%   xe(1:Ne,1:2)  - Auswertungspunkte in Polarkoordinaten (r,theta)
%   f(1:N,1:Nf)   - Interpolationswerte von Nf unterschiedlichen Funktionen
%   rbf           - radiale Basisfunktion aus
%                   {'imq', 'bimq', 'mq'}
%   optpar        - beta-Parameter fuer 'bimq'
%
% Ausgabe:
%   u(1:Ne,1:Nf) - RBF-Interpolant an den Stellen xe fuer jede Funktion
%   kond         - Konditionszahl der Interpolationsmatrix
%   qn           - Anzahl an Knoten der Gauss-Quadratur
%
% Literatur
%  [1] - Rene Hofmann, 'Approximation mit radialen Basisfunktionen',
%        Dissertation, 2013
%  [2] - Bengt Fornberg et al.,
%        'Stable Computations with Gaussian Radial Basis Functions in 2-D',
%        Uppsala University, Department of Information Technology, 2009
function [ u, kond, qn ] = RBF_QR_GQ_2D( ep, xk, xe, f, rbf, optpar=0 )
  % Anzahl Interpolations- und Auswertungspunkte bestimmen
  N = size(xk,1); Ne = size(xe,1);
  % Maschinengenauigkeit
  mp = eps;
  % jmax zum Abschneiden der Summe bestimmen (jN+1)(jN+2)/2=N
  jN=ceil(-3/2+sqrt(9/4+2*N-2));
  jmax = 1; ratio = ep^2/2;
  while ( jmax < jN & ratio > 1 )
    jmax = jmax + 1;
    ratio = ep^2 / (jmax + mod( jmax, 2 )) * ratio;
  end
  if (ratio < 1)
    jmax = jN; ratio = 1;
  end
```

```
42  ratio = epˆ2 / (jmax + 1 + mod(jmax + 1, 2)) * ratio;
43  while (ratio * exp(0.223 * (jmax + 1) + 0.212 *...
44      (1 - 3.097 * mod(jmax + 1, 2))) > mp)
45    jmax = jmax + 1;
46    ratio = epˆ2 / (jmax + 1 + mod(jmax + 1, 2)) * ratio;
47  end
48  % ------------------------------------------------------------
49  % Gauss-Quadratur
50  % ------------------------------------------------------------
51  % Bestimmung Anzahl, Knoten und Gewichte
52  [ qn, knots, weights ] = RBF_QR_GQ_quad_part( rbf, ep, xk, optpar );
53  % Indizees und Vektoren mit Knoten und Gewichten
54  j = zeros(0,1); m=zeros(0,1); p=zeros(0,1); lq=zeros(0,1); odd=1;
55  hatX=[]; Omega=[]; co_si = zeros(0,1); P = zeros(0,0);
56  for k = 0 : jmax
57    odd = 1 - odd;
58    j = [ j; k * ones( k + 1, 1 ) ]; p = [ p; odd * ones( k + 1, 1 ) ];
59    q(1:2:k+1,1) = (0:(k-odd)/2)'; q(2:2:k+1,1) = ((1-odd):(k-odd)/2)';
60    m = [m;q(1:k+1)];
61    % Cosinus und Sinus Teil bestimmen
62    pos = find(2*q(1:k+1)+odd*ones( k + 1, 1 )>0);
63    co_si_help = zeros(k+1,1);
64    co_si_help(pos(1:2:end),1) = 1; co_si_help(pos(2:2:end),1) = 2;
65    co_si = [ co_si; co_si_help ];
66    if(qn>1)
67      hatX = [ hatX; knots(repmat((1:qn)',k+1,1))' ];
68      Omega = [ Omega; weights(repmat((1:qn)',k+1,1))' ];
69    else
70      hatX = [ hatX; knots(repmat((1:qn)',k+1,1)) ];
71      Omega = [ Omega; weights(repmat((1:qn)',k+1,1)) ];
72    end
73  end
74  % Indizees an Gauss-Quadratur anpassen
75  j = repmat( j', qn, 1 ); j = j(:);
76  m = repmat( m', qn, 1 ); m = m(:);
77  p = repmat( p', qn, 1 ); p = p(:);
78  co_si = repmat( co_si', qn, 1 );
79  co_si = co_si(:);
80  % Permutationsmatrix
81  P = eye( length( j ), length( j ) );
82  P1 = P(:,1:qn:end); P(:,1:qn:end) = [];
```

```
83  P = [ P1, P ];
84  % ----------------------------------------------------------------
85  % Berechne T_j(r), cos/sin(m*theta), und Exponenten von r
86  Tk = cos(acos(xk(:,1))*(0:jmax)); Te = cos(acos(xe(:,1))*(0:jmax));
87  Hkc = cos(xk(:,2)*(1:jmax)); Hec = cos(xe(:,2)*(1:jmax));
88  Hks = sin(xk(:,2)*(1:jmax)); Hes = sin(xe(:,2)*(1:jmax));
89  Pk = ones(N,jmax+1);
90  for k=1:jmax
91    Pk(:,k+1) = (xk(:,1).*Pk(:,k));
92  end
93  re2 = xe(:,1).^2;
94  Pe = ones(Ne,(jmax-p(end))/2+1);
95  for k=1:(jmax-p(end))/2
96    Pe(:,k+1) = re2.*Pe(:,k);
97  end
98  % Koeffizientenmatrix C bestimmen unter Beruecksichtigung der GQ-Knoten
99  M = length(j);
100 scale = exp(-ep^2*repmat(xk(:,1).^2,1,M).*repmat(hatX',N,1));
101 switch rbf
102   case {'mq'}
103     scale = scale .*repmat((hatX.^(j-1))',N,1);
104   otherwise
105     scale = scale .*repmat((hatX.^(j))',N,1);
106 endswitch
107 cscale = 2*ones(1,M); % Spaltenskalierung von C = b_{2m+p}t_{j-2m}
108 pos = find(2*m+p == 0); cscale(pos) = 0.5*cscale(pos);
109 pos = find(j-2*m == 0); cscale(pos) = 0.5*cscale(pos);
110 C = Pk(:,j+1); % Die Exponenten von r_k und der trigonometrische Teil
111 pos = find(co_si == 1); C(:,pos) = C(:,pos).*Hkc(:,2*m(pos)+p(pos));
112 pos = find(co_si == 2); C(:,pos) = C(:,pos).*Hks(:,2*m(pos)+p(pos));
113 % Gewichte einfuegen
114 C = C.*(scale.*repmat(cscale,N,1)).*repmat(Omega',N,1);
115 a = (j-2*m+p+1)/2; b=[j-2*m+1 (j+2*m+p+2)/2];
116 z = ep.^4*xk(:,1).^2;
117 % Hypergeometrische Reihe
118 for k=1:M
119     C(:,k) = C(:,k).*hyperg_1F2(a(k),b(k,1),b(k,2),z*hatX(k)^2);
120 end
121 % QR-Zerlegung von C (N x (jmax*qn))
122 [Q,R] = qr(C*P);
123 % Teilmatrix von R invertieren
```

```
124   Rt = R(1:N,1:N)\R(1:N,N+1:M);
125   p1 = (1:N); p2 = (N+1):M; [pp2,pp1] = meshgrid(p2,p1);
126   if (M>N)
127     D = EvalD(ep,pp1,pp2,j,m,p);
128     Rt = D.*Rt;
129   end
130   % Basisfunktionen an den Interpolationspunkten auswerten (N x (jmax*qn))
131   V = exp(-ep^2*repmat(xk(:,1).^2,1,M).*repmat(hatX',N,1)).* ...
132       ( Pk(:,2*m+1).*Tk(:,j-2*m+1));
133   pos = find(co_si == 1); V(:,pos) = V(:,pos).*Hkc(:,2*m(pos)+p(pos));
134   pos = find(co_si == 2); V(:,pos) = V(:,pos).*Hks(:,2*m(pos)+p(pos));
135   % Spalten wieder umorden
136   V = V*P;
137   % Interpolationsmatrix
138   A = V(:,1:N) + V(:,N+1:M)*Rt.';
139   % Konditionszahl
140   kond = cond(A);
141   % Koeffizienten
142   lambda = A\f;
143   % Basisfunktionen an den Auswertungspunkten auswerten (Ne x (jmax*qn))
144   Ve = exp(-ep^2*repmat(xe(:,1).^2,1,M).*repmat(hatX',Ne,1)).* ...
145       ( Pe(:,m+1).*Te(:,j-2*m+1));
146   pos = find(co_si == 1); Ve(:,pos) = Ve(:,pos).*Hec(:,2*m(pos)+p(pos));
147   pos = find(co_si == 2); Ve(:,pos) = Ve(:,pos).*Hes(:,2*m(pos)+p(pos));
148   % Spalten wieder umordnen
149   Ve = Ve*P;
150   % Loesung auswerten
151   B = Ve(:,1:N) + Ve(:,N+1:M)*Rt.';
152   u = B*lambda;
153 end
154
155 function D=EvalD(ep,p1,p2,jv,m,p)
156   % D_1^{-1} und D_2 berechnen
157   sz = size(p1);
158   p1 = p1(:); p2 = p2(:);
159   D = ep.^(2*(jv(p2)-jv(p1))./2.^(jv(p2)-jv(p1)-2*(m(p2)-m(p1)));
160   f1 = (jv(p2)+2*m(p2)+p(p2))/2; f2 = (jv(p2)-2*m(p2)-p(p2))/2;
161   f3 = (jv(p1)+2*m(p1)+p(p1))/2; f4 = (jv(p1)-2*m(p1)-p(p1))/2;
162   for k=1:length(D)
163     v1 = sort([(f1(k)+1):f3(k) (f2(k)+1):f4(k)]);
164     v2 = sort([(f3(k)+1):f1(k) (f4(k)+1):f2(k)]);
```

```
165       l1 = length(v1); l2 = length(v2);
166       v1 = [ones(1,l2-l1) v1]; v2 = [ones(1,l1-l2) v2];
167       D(k) = D(k)*prod(v1./v2);
168     end
169     D = reshape(D,sz);
170 endfunction
```

Quelltext A.4: RBF_QR_GQ_2D.m

```
% RBF_QR_GQ_quad_part.m
%
% Berechnet die Anzahl, Knoten und Gewichte zur Gauss-Quadratur
% fuer die gegebene RBF.
%
% Eingabe:
%   rbf           - radiale Basisfunktion aus
%                   {'imq', 'bimq', 'mq'}
%   ep            - Glaettungsparameter
%   xk(1:N,1:2)   - Interpolationspunkte in Polarkoordinaten (r,theta)
%   optpar        - beta-Parameter fuer 'bimq'
%
% Ausgabe:
%   qn            - Anzahl an Knoten der Gauss-Quadratur
%   knots         - Knoten der Gauss-Quadratur
%   weights       - Gewichte der Gauss-Quadratur
function [ qn, knots, weights ] = RBF_QR_GQ_quad_part( rbf, ep, xk, optpar=0 )
  % Maximale Anzahl an Knoten
  qn_max = 5;
  % Maschinengenauigkeit
  mp = eps;
  % Bestimmung Anteil Knoten
  [XX,YY] = meshgrid(xk(:,1), xk(:,1));
  qn = 0; testq = 1; max_dist = max(max(abs( XX - YY )));
  switch rbf
    case {'imq','mq'}
      while ( testq > mp & qn < qn_max )
        qn++;
        testq = factorial(qn) * gamma(qn - 1/2) / ...
                factorial (2 * qn) * (ep*max_dist)^(4 * qn);
      endwhile
      % Knoten als Nullstellen der Laguerre-Funktion
      [Y, Lag] = generalized_laguerre(-1/2, qn, 0); knots = roots(Lag)';
      % Gewichte
      weights = knots./generalized_laguerre( - 1/2, qn + 1, knots ).^2;
    case {'bimq'}
      if( optpar<=0 )
  error ('optpar for bIMQ has to be a nonnegative real number!');
      else
  while ( testq > mp & qn < qn_max )
    qn++;
```

```
42      testq = factorial(qn) * gamma(qn + optpar - 1) / ...
43                  factorial(2 * qn) * (ep*max_dist)^(4 * qn);
44    endwhile
45    % Knoten als Nullstellen der Laguerre-Funktion
46    [Y, Lag] = generalized_laguerre(optpar - 1, qn, 0); knots = roots(Lag)';
47    % Gewichte
48    weights = knots./(generalized_laguerre(optpar - 1, qn + 1, knots ).^2);
49        endif
50      otherwise
51        error ('Invalide RBF!');
52    endswitch
53  endfunction
```

Quelltext A.5: RBF_QR_GQ_quad_part.m

A.4 RBF-QI-Algorithmus

```
% RBFQuasiInterpolationTest.m
%
% Vergleicht die Quasi- und Interpolation der shifted-Thin-Plate-Splines
% 1. und 2. Art anhand der Testfunktionen 1 - 6.
%
% Fuer die Quasi-Interpolation werden die 21 Punkte aus [1] und die
% Koeffizienten [1] verwendet. Das Gitter wird aus der Feinheit h
% und der Randerweiterung omega je nach Gebiet (Einheitskreis oder -quadrat)
% erstellt .
%
% Die Interpolation verwendet n Eigenwert- (Einheitskreis)
% oder Halton-Punkte (Einheitskreisquadrat).
%
% Benoetigte Funktionen:
%   LadeTestfunktionen          - Testfunktion 1 - 6
%   BildeAuswertungspunkte      - 2-dimensionale Auswertungspunkte
%   RBF_QI_Gitter               - 2-dimensionale Gitterpunkte zu omega und h
%   RBF_QI_TPS_1                - Quasi-Interpolantion mit sTPS 1. Art
%   RBF_QI_TPS_2                - Quasi-Interpolantion mit sTPS 2. Art
%   RBF_Direct_2D_TPS_1_Polar - Direkte Interpolantion sTPS 1. Art
%   RBF_Direct_2D_TPS_2_Polar - Direkte Interpolantion sTPS 2. Art
%   getIntPnts                  - Interpolationspunkte
%
% Vorzugebende Werte:
%   c         - Glaettungsparameter der shifted-Thin-Plate-Splines 1. Art
%   ep        - Glaettungsparameter der shifted-Thin-Plate-Splines 2. Art
%   h         - Feinheit des QI-Gitter
%   omega     - Randerweiterung QI
%   n         - Anzahl an Interpolationspunkten
%   gebiet    - Einheitskreis oder -quadrat ('circle ',  'cube')
%   res       - Aufloesung der Auswertungspunkte, ergibt ein ( res x res ) - Gitter
%   method - Art der Interpolantionspunkte ('ev ','md','mn')
%   nbr       - Nummer der Interpolantionspunkte ('09' = 100, '19' = 400, '29' = 500);
%
% Literatur
%  [1] - Rene Hofmann,
%          'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%          numerische Untersuchung der Dagum-Funktionen',
%          Dissertation , 2013
```

```
40 if ( not(exist('c')) ) c = 1; endif
41 if ( not(exist('ep')) ) ep = 1; endif
42 if ( not(exist('h')) ) h = 1/8; endif
43 if ( not(exist('omega')) ) omega = 1; endif
44 if ( not(exist('n')) ) n = 100; endif
45 if ( not(exist('gebiet')) ) gebiet = 'circle'; endif
46 if ( not(exist('res')) ) res = 25; endif
47 if ( not(exist('p1')) && not(exist('p2')) ) p1 = 7; p2 = 11; endif
48 if ( not(exist('method')) ) method = 'ev'; endif
49 if ( not(exist('nbr')) ) nbr = '09'; endif
50 % Testfunktionen
51 LadeTestfunktionen;
52 % Koeffizienten der Quasi-Interpolanten sTPS 1. Art
53 mu1 = 1/(96*pi) * [ 5*(70+51*c^2), ...
54         kron( [ -(142+117*c^2), 5*(10+9*c^2)/2, ...
55         -(2+3*c^2), 4*(8+9*c^2), -(2+9*c^2)/4 ], ones( 1, 4 ) ) ];
56 % Koeffizienten der Quasi-Interpolanten sTPS 2. Art
57 mu2 = 1/(12*pi*ep^4) * [ 5*(51+35*ep^2)/4, ...
58        kron( [ -(117+71*ep^2)/4, 5*(9+5*ep^2)/8, ...
59        -(3+ep^2)/4, (9+4*ep^2), -(9+ep^2)/(16) ], ones( 1, 4 ) ) ];
60 % Punkte der Quasi-Interpolanten
61 Punkte = [0;0];
62 Punkte = [ Punkte, [ 1; 0 ], [-1; 0 ], [ 0; 1 ], [ 0;-1 ] ];
63 Punkte = [ Punkte, [ 2; 0 ], [-2; 0 ], [ 0; 2 ], [ 0;-2 ] ];
64 Punkte = [ Punkte, [ 3; 0 ], [-3; 0 ], [ 0; 3 ], [ 0;-3 ] ];
65 Punkte = [ Punkte, [ 1; 1 ], [-1; 1 ], [ 1;-1 ], [-1;-1 ] ];
66 Punkte = [ Punkte, [ 2; 2 ], [-2; 2 ], [ 2;-2 ], [-2;-2 ] ];
67 % Aufloesung des Gitters fuer die Auswertungspunkte
68 [ kart, pol ] = BildeAuswertungspunkte( res, gebiet );
69 xe = kart(1,:); ye = kart(2,:);
70 % Gitter fuer die Quasi-Interpolation
71 Gitter = RBF_QL_Gitter( h, omega, gebiet );
72 % Exakte Loesungswerte
73 E = [ tf1(xe,ye); tf2(xe,ye); tf3(xe,ye); tf4(xe,ye); tf5(xe,ye); tf6(xe,ye) ];
74 % Testfunktionen mittels sTPS der 1. Art approximieren
75 % und relative Fehler speichern
76 y_1_1 = RBF_QL_TPS_1( c, 'tf1', [xe; ye], mu1, Punkte, h, Gitter );
77 y_1_2 = RBF_QL_TPS_1( c, 'tf2', [xe; ye], mu1, Punkte, h, Gitter );
78 y_1_3 = RBF_QL_TPS_1( c, 'tf3', [xe; ye], mu1, Punkte, h, Gitter );
79 y_1_4 = RBF_QL_TPS_1( c, 'tf4', [xe; ye], mu1, Punkte, h, Gitter );
80 y_1_5 = RBF_QL_TPS_1( c, 'tf5', [xe; ye], mu1, Punkte, h, Gitter );
```

```
81  y_1_6 = RBF_QL_TPS_1( c, 'tf6', [xe; ye], mu1, Punkte, h, Gitter );
82  fehler_qi_1 = norm( E - [ y_1_1; y_1_2; y_1_3; y_1_4; y_1_5; y_1_6 ], 2, 'rows' )' ./ ...
83                norm( E, 2, 'rows' )';
84  % Testfunktionen mittels sTPS der 2. Art approximieren
85  % und relative Fehler speichern
86  y_2_1 = RBF_QL_TPS_2( ep, 'tf1', [xe; ye], mu2, Punkte, h, Gitter );
87  y_2_2 = RBF_QL_TPS_2( ep, 'tf2', [xe; ye], mu2, Punkte, h, Gitter );
88  y_2_3 = RBF_QL_TPS_2( ep, 'tf3', [xe; ye], mu2, Punkte, h, Gitter );
89  y_2_4 = RBF_QL_TPS_2( ep, 'tf4', [xe; ye], mu2, Punkte, h, Gitter );
90  y_2_5 = RBF_QL_TPS_2( ep, 'tf5', [xe; ye], mu2, Punkte, h, Gitter );
91  y_2_6 = RBF_QL_TPS_2( ep, 'tf6', [xe; ye], mu2, Punkte, h, Gitter );
92  fehler_qi_2 = norm( E - [ y_2_1; y_2_2; y_2_3; y_2_4; y_2_5; y_2_6 ], 2, 'rows' )' ./ ...
93                norm( E, 2, 'rows' )';
94  % Interpolationspunkte
95  switch ( gebiet )
96    case 'cube'
97      method = 'hd'; p = [p1,p2]; N = [n,n];
98    case ' circle '
99      % siehe Initiierung
100   otherwise
101     break;
102 end
103 LadeInterpolationspunkte;
104 [x,y] = pol2cart( theta, r );
105 % Testfunktionen an den Interpolationspunkten auswerten
106 TF_x_y = [ tf1(x,y)' tf2(x,y)' tf3(x,y)' tf4(x,y)' tf5(x,y)' tf6(x,y)' ];
107 % Interpolation sTPS 1. Art
108 Y1 = RBF_Direkt_2D( [xe;ye], [x;y], TF_x_y, 'stps1', c )';
109 fehler_di_1 = norm( E - Y1, 2, 'rows' )' ./ norm( E, 2, 'rows' )';
110 % Interpolation sTPS 2. Art
111 Y2 = RBF_Direkt_2D( [xe;ye], [x;y], TF_x_y, 'stps2', ep )';
112 %Y2 = RBF_Direct_2D_TPS_2( ep, [x;y], [xe;ye], TF_x_y )';
113 fehler_di_2 = norm( E - Y2, 2, 'rows' )' ./ norm( E, 2, 'rows' )';
```

Quelltext A.6: RBFQuasiInterpolationTest.m

```
% RBF_QI_Gitter.m
%
% Eingabe:
% h       - Feinheit des Gitters
% omega - Parameter fuer zusaetzliches Randgebiet
% gebiet  - Einheitskreis oder -quadrat ('circle', 'cube')
%
% Ausgabe:
% Gitter  - Gitterpunkte der Form (1:2,1:N) mit N = (2*(1/h+omega)+1)^2
function Gitter = RBF_QI_Gitter( h, omega, gebiet )
  h_inv = 1/h;
  % Gitterpunkte
  N = ( 2 * (h_inv + omega) + 1 )^2;
  % Gitter bauen
  max_radio = h_inv + omega;
  grided = linspace( -max_radio, max_radio, 2 * max_radio + 1);
  [ gx, gy ] = meshgrid( grided );
  Gx = gx( : ); Gy = gy( : );
  Kx = Gx; Ky = Gy;
  if( strmatch( gebiet, 'circle' ) )
    % Loesche alle die ausserhalb des Kreises mit zusaetzliches Rand liegen
    Index = find( norm( [ Gx, Gy ], 2, 'rows' ) > max_radio );
    Gx( Index ) = []; Gy( Index ) = [];
  endif
  Gitter = [ Gx, Gy ]';
endfunction
```

Quelltext A.7: RBF_QI_Gitter.m

```
% RBF_QI_TPS_1.m
%
% Quasi-Interpolation mit shifted-Thin-Plate-Splines der 1. Art zur Funktion fct.
% Fuer die Quasi-Interpolante werden die 21 Punkte aus [1]
% und die dazu bestimmten Koeffizienten mu aus [1] verwendet.
% Die Approximation ist fuer Polynome vom Grad 3 exakt.
%
% Benoetigte Funktionen:
% RBF_QI_Gitter - 2-dimensionale Gitterpunkte zu omega und h
%
% Eingabe:
%   c(1)                - Parameter der sTPS 1. Art, c > 0
%   fct(x,y)            - zu interpolierende Funktion, fct: R^2 -> R
%   xa(1:2,1:n_eval)    - Auswertungspunkte
%   mu(1,1:n_mu)        - Koeffizienten der Quasi-Interpolante
%   pkt(1:2,1:n_mu)     - Punkte der Quasi-Interpolante zu mu
%   h(1)                - Feinheit des Gitters
%   gpkt(1:2,1:n_gpkt)  - Gitterpunkte, falls [] muessen omega und gebiet gegeben sein
%   omega(1)            - optional, falls gesetzt wird grid ignoriert und
%                         Gitterpunkte zusaetzlich berechnet
%   gebiet              - optional, muss aus {'circle','cube'} sein
%
% Ausgabe:
%   y(1,1:n_eval)       - Werte der QI an den Punkten xa
%
% Literatur
% [1] - Rene Hofmann,
%       'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%       numerische Untersuchung der Dagum-Funktionen',
%       Dissertation, 2013
function y = RBF_QI_TPS_1( c, fct, xa, mu, pkt, h, gpkt, omega=[], gebiet=[] )
  % Falls nicht gegeben, Gitter fuer die Quasi-Interpolation bilden
  if( length(gpkt)==0 )
    gpkt = RBF_QI_Gitter( h, omega, gebiet );
  end
  % Anzahl Punkte, Gitter- und Auswertungspunkte
  nm = columns( pkt );
  ng = columns( gpkt );
  ne = columns( xa );
  % Quasi-Interpolation
  for col = 1 : ne
```

```
42      x = xa( :, col );
43      args = repmat( x/h, 1, ng * nm ) - kron( gpkt, ones( 1, nm ) ) - repmat( pkt, 1, ng );
44      s = repmat( mu, 1, ng ) .* TPS_1( c, args );
45      y( 1, col ) = feval( fct, gpkt(1,:) * h, gpkt(2,:) * h ) * sum( reshape( s, nm, ng ), 1 )';
46    end
47  endfunction
```

Quelltext A.8: RBF_QI_TPS_1.m

```
% RBF_QI_TPS_2.m
%
% Quasi-Interpolation mit shifted-Thin-Plate-Splines der 2. Art zur Funktion fct.
% Fuer die Quasi-Interpolante werden die 21 Punkte aus [1]
% und die dazu bestimmten Koeffizienten mu aus [1, ] verwendet.
% Die Approximation ist fuer Polynome vom Grad 3 exakt.
%
% Benoetigte Funktionen:
% RBF_QI_Gitter - 2-dimensionale Gitterpunkte zu omega und h
%
% Eingabe:
% ep(1)                 - Parameter der sTPS 2. Art, ep > 0
% fct(x,y)              - zu interpolierende Funktion, fct: R^2 -> R
% xa(1:2,1:n_eval)      - Auswertungspunkte
% mu(1,1:n_mu)          - Koeffizienten der Quasi-Interpolante
% pkt(1:2,1:n_mu)       - Punkte der Quasi-Interpolante zu mu
% h(1)                  - Feinheit des Gitters
% gpkt1:2,1:n_gpkt)     - Gitterpunkte, falls [] muessen omega und gebiet gegeben sein
% omega(1)              - optional, falls gesetzt wird grid ignoriert und
%                         Gitterpunkte zusaetzlich berechnet
% gebiet                - optional, muss aus {'circle', 'cube'} sein
%
% Ausgabe:
%  y(1,1:n_eval)        - Werte der QI an den Punkten xa
%
% Literatur
% [1] - Rene Hofmann,
%       'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%       numerische Untersuchung der Dagum-Funktionen',
%       Dissertation, 2013

function y = RBF_QI_TPS_2( ep, fct, xa, mu, pkt, h, gpkt, omega=[], gebiet=[] )
  % Falls nicht gegeben, Gitter fuer die Quasi-Interpolation bilden
  if( length(gpkt)==0 )
    gpkt = RBF_QI_Gitter( h, omega, gebiet );
  end
  % Anzahl Punkte, Gitter- und Auswertungspunkte
  nm = columns( pkt );
  ng = columns( gpkt );
  ne = columns( xa );
  % Quasi-Interpolation
```

```
42    for  col  = 1 : ne
43      x = xa( :,  col  );
44      args  = repmat( x/h, 1, ng * nm) - kron( gpkt, ones( 1, nm ) ) - repmat( pkt, 1, ng );
45      s  = repmat( mu, 1, ng ) .* TPS_2( ep, args );
46      y( 1,  col  ) = feval(  fct ,  gpkt(1,:)  * h,  gpkt(2,:)  * h ) * sum( reshape( s,  nm, ng ),  1  )';
47    end
48  endfunction
```

Quelltext A.9: RBF_QI_TPS_2.m

```
% RBF_QI_TPS_1_Koeff.m
%
% Berechnet die Koeffizienten der Quasi-Interpolation der
% shifted-Thin-Plate-Splines 1. Art.
%
% Eingabe:
%   c(1)                  - Glaettungsparameter der sTPS 1. Art, c > 0
%   b(1)                  - Parameter der sTPS 1. Art, b in 2R^+
%   Punkte(1:2,1:npkt) - Punkte der Quasi-Interpolante zu mu
%
% Ausgabe:
%   mu(1,1:npkt)        - Koeffizienten der QI
%
% Literatur
%  [1]  - Rene Hofmann,
%         'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%         numerische Untersuchung der Dagum-Funktionen', Dissertation, 2013
function mu = RBF_QI_TPS_1_Koeff( c, b, Punkte )
  % Dimension
  d = 2;
  % Maximale Polynomreproduktion vom Grad m
  m = b + d - 1;
  % Minimum von |alpha| bestimmen, ab dem sum_{j=1}^\ell \mu_j x_j^\alpha!=0
  alpha_min = b + d;
  % Maximum von |alpha| bestimmen, bis zu dem sum_{j=1}^\ell \mu_j x_j^\alpha existiert
  alpha_max = alpha_min + m;
  % Index mit |alpha| erstellen fuer das sum_{j=1}^\ell \mu_j x_j^\alpha!=0
  alpha_ind = [ ones( 1, alpha_min - 1 ), [ alpha_min : alpha_max ] ];
  alpha_ind_even = 1 - mod( alpha_ind, 2 );
  % Rechten Seite !=0
  alpha_ind_ungl_null = find( alpha_ind_even );
  % Multiindicees alpha bestimmen
  alpha = [];
  for it = 0 : alpha_max
    help_row = linspace(0,it,it + 1);
    alpha = [ alpha, [ help_row; fliplr ( help_row ) ] ];
  end
  % Interpolationsmatrix
  for jt = 1 : columns(alpha)
    for it = 1:columns(Punkte)
      M(jt,it) = prod(Punkte(:,it).^alpha(:,jt));
```

```
42      end
43    end
44    % rechte Seite
45    r = zeros( columns( alpha ), 1 );
46    alpha_sum = sum( alpha ); alpha_sum_ungl_null = [ b + d : 2 : b + d + m ];
47    % Indicees der alpha's dazu
48    % zuerst A_k
49    max_ind_A = alpha_sum_ungl_null(end)
50    A(1) = 0;
51    for it = 1 : max_ind_A
52      s = 0;
53      for jt = 1 : it + 1
54        a = 0;
55        for gt = find( alpha_sum == jt + 1 )(1) : find( alpha_sum == jt + 1 )(end)
56          for dt = 1 : jt - 1
57            vektor = (1 - (b+d)/2) + [ 0 : alpha( 1, gt ) - 1 ];
58            vektor(find(vektor==0)) = 1; pochh = prod( vektor );
59          end
60          a += 1 / ( prod(factorial( alpha( :, gt ) )) * );
61        end
62        s += (-1)^jt * a;
63      end
64      A(it+1) = s/factorial((b+d)/2-1);
65    end
66    % rechte Seite r berechnen
67    for it = 1 : length( alpha_sum_ungl_null )
68      ind = find( alpha_sum == alpha_sum_ungl_null(it) & !prod( mod( alpha, 2 ) ) );
69      for jt = 1 : columns( ind )
70        r( ind( jt ) ) = ( (-1)^alpha_sum( ind( jt ) ) * 2^(b+d) * ...
71          A( ( alpha_sum( ind( jt ) ) - b - d )/2 + 1 ) * ...
72          c^( alpha_sum( ind( jt ) ) - b - d ) * ...
73          prod( factorial ( sum(alpha( :, index_4( jt ) )) ./ 2 ) ) * ...
74          prod( factorial ( alpha( :, index_4( jt ) ) ) ) ) / ...
75          ( 2^alpha_sum( ind( jt ) ) * C(b,d) * prod( factorial (alpha (:, index_4( jt )) ./ 2)) );
76      end
77    end
78    % Koeffizienten berechnen
79    mu = M\r;
80  endfunction
```

Quelltext A.10: RBF_QI_TPS_1_Koeff.m

```
% RBF_QI_TPS_2_Koeff.m
%
% Berechnet die Koeffizienten der Quasi-Interpolation der
% shifted-Thin-Plate-Splines 1. Art.
%
% Eingabe:
%   ep(1)                 - Glaettungsparameter der sTPS 2. Art, ep > 0
%   b(1)                  - Parameter der sTPS 1. Art, b in 2R^+
%   Punkte(1:2,1:npkt) - Punkte der Quasi-Interpolante zu mu
%
% Ausgabe:
%   mu(1,1:npkt)          - Koeffizienten der QI
%
% Literatur
%  [1]  - Rene Hofmann,
%         'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%         numerische Untersuchung der Dagum-Funktionen',
%         Dissertation, 2013
function mu = RBF_QI_TPS_2_Koeff( ep, b, Punkte )
  % Dimension
  d = 2;
  % Maximale Polynomreproduktion vom Grad m
  m = b + d - 1;
  % Minimum von |alpha| bestimmen, ab dem sum_{j=1}^\ell \mu_j x_j^\alpha!=0
  alpha_min = b + d;
  % Maximum von |alpha| bestimmen, bis zu dem sum_{j=1}^\ell \mu_j x_j^\alpha existiert
  alpha_max = alpha_min + m;
  % Index mit |alpha| erstellen fuer das sum_{j=1}^\ell \mu_j x_j^\alpha!=0
  alpha_ind = [ ones( 1, alpha_min - 1 ), [ alpha_min : alpha_max ] ];
  alpha_ind_even = 1 - mod( alpha_ind, 2 );
  % Rechten Seite !=0
  alpha_ind_ungl_null = find( alpha_ind_even );
  % Multiindicees alpha bestimmen
  alpha = [];
  for it = 0 : alpha_max
    help_row = linspace(0,it,it + 1);
    alpha = [ alpha, [ help_row; fliplr ( help_row ) ] ];
  end
  % Interpolationsmatrix
  for jt = 1 : columns(alpha)
    for it = 1:columns(Punkte)
```

```
42          M(jt,it) = prod(Punkte(:,it).^alpha(:, jt ));
43        end
44      end
45      % rechte Seite
46      r = zeros( columns( alpha ), 1 );
47      alpha_sum = sum( alpha ); alpha_sum_ungl_null = [ b + d : 2 : b + d + m ];
48      % Indicees der alpha's dazu
49      % zuerst A_k
50      max_ind_A = alpha_sum_ungl_null(end)
51      A(1) = 0;
52      for it = 1 : max_ind_A
53        s = 0;
54        for jt = 1 : it + 1
55          a = 0;
56          for gt = find( alpha_sum == jt + 1 )(1) : find( alpha_sum == jt + 1 )(end)
57            for dt = 1 : jt - 1
58              vektor = (1 - (b+d)/2) + [ 0 : alpha( 1, gt ) - 1 ];
59              vektor(find(vektor==0)) = 1; pochh = prod( vektor );
60            end
61            a += 1 / ( prod(factorial( alpha( :, gt ) )) * );
62          end
63          s += (-1)^jt * a;
64        end
65        A(it+1) = s/factorial((b+d)/2-1);
66      end
67      % rechte Seite r berechnen
68      for it = 1 : length( alpha_sum_ungl_null )
69        ind = find( alpha_sum == alpha_sum_ungl_null(it) & !prod( mod( alpha, 2 ) ) );
70        for jt = 1 : columns( ind )
71          r( ind( jt ) ) = ( (-1)^alpha_sum( ind( jt ) ) * 2^(b+d) * ...
72            A( ( alpha_sum( ind( jt ) ) - b - d )/2 + 1 ) * ...
73            c^( alpha_sum( ind( jt ) ) - b - d ) * ...
74            prod( factorial ( sum(alpha( :, index_4( jt ) )) ./ 2 ) ) * ...
75            prod( factorial ( alpha( :, index_4( jt ) ) ) ) ) / ...
76            ( 2^alpha_sum( ind( jt ) ) * C(b,d) * prod( factorial ( alpha( :, index_4( jt ) ) ./ 2 ) ) );
77        end
78      end
79      % Koeffizienten berechnen
80      mu = M\r;
81  endfunction
```

Quelltext A.11: RBF_QI_TPS_2_Koeff.m

```
% TPS_1.m
%
% Berechnet zu gegebenen Punkten und Glaettungsparameter c
% die Werte der shiftes-Thin-Plate-Splines der 1. Art, d.h.
% phi(r) = (c^2 + r^2) * log( sqrt( c^2 + r^2 ) ).
%
% Eingabe:
% c(1)           - Glaettungsparameter, c > 0
% Pkt(1:2,1:N) - Auswertungspunkte
%
% Ausgabe:
% y(1,1:N)       - Werte der sTPS an den Punkten Pkt
function y = TPS_1( c, Pkt )
  cNPkt = c^2 + sum( Pkt.^2, 1 );
  y = cNPkt .* log( cNPkt.^(1/2) );
endfunction
```

Quelltext A.12: TPS_1.m

```
% TPS_2.m
%
% Berechnet zu gegebenen Punkten und Glaettungsparameter epsilon
% die Werte der shiftes-Thin-Plate-Splines der 2. Art, d.h.
% phi(r) = (1 + epsilon^2 * r^2) log( sqrt( 1 + epsilon^2 * r^2 ) ).
%
% Eingabe:
% ep(1)          - Glaettungsparameter, ep > 0
% Pkt(1:2,1:N) - Auswertungspunkte
%
% Ausgabe:
% y(1,1:N)       - Werte der sTPS an den Punkten Pkt
function Y = TPS_2( ep, Pkt )
  cNPkt = 1 + ep^2 * sum( Pkt.^2, 1 );
  Y = cNPkt .* log( cNPkt.^(1/2) );
endfunction
```

Quelltext A.13: TPS_2.m

A.5 Hilfsfunktionen

```
% LadeTestfunktionen.m
%
% Definition der Testfunktionen.
%
% Ausgabe:
%   tf1(x,y) = 25/(25 + ( x - 0.2 )^2 + 2 y^2 )
%   tf2(x,y) = exp( ( x - 0.1 )^2 + 1/2 y^2 )/exp( 1.21 )
%   tf3(x,y) = atan( 2 ( x + 3*y - 1 ) ) / atan( 2 ( sqrt( 10 ) + 1 ) )')
%   tf4(x,y) = sin( 2 pi ( x - y ) )')
%   tf5(x,y) = 3/4 exp( -1/4((9x-2)^2+(9y-2)^2) )
%              + 3/4 exp( -1/49(9x+1)^2-1/10(9y+1))')
%              + 1/2 exp( -1/4((9x-7)^2+(9y-3)^2)) )
%              - 1/5 exp( -(9x-4)^2-(9y-7)^2 )')
%   tf6(x,y) = max(0,(1-2|x|)) + max(0,(1-2|y|))')
disp('Lade Testfunktionen');
disp('tf1  = @(x,y) 25/(25 + ( x - 0.2 )^2 + 2 y^2 )');
function f = tf1( x, y )
  f = 25 * (25 + ( x - 0.2 ).^2 + 2 * y.^2 ).^(-1);
endfunction
disp('tf2  = @(x,y) exp( ( x - 0.1 )^2 + 1/2 y^2 )/exp( 1.21 )');
function f = tf2( x, y )
  f = exp((x .- 0.1).^2 .+ 0.5.*y.^2)./exp(1.21);
endfunction
disp('tf3  = @(x,y) atan( 2 ( x + 3*y - 1 ) ) / atan( 2 ( sqrt( 10 ) + 1 ) )');
function f = tf3( x, y )
  f = atan( 2 * ( x .+ 3 * y .- 1 ) ) ./ atan( 2 * ( sqrt( 10 ) + 1 ) );
endfunction
disp('tf4  = @(x,y) sin( 2 pi ( x - y ) )');
function f = tf4( x, y )
  f = sin(2*pi*(x-y));
endfunction
% Franke's Funktion
disp('tf5  = @(x,y) 3/4 exp( -1/4((9x-2)^2+(9y-2)^2) )');
disp('             + 3/4 exp( -1/49(9x+1)^2-1/10(9y+1))');
disp('             + 1/2 exp( -1/4((9x-7)^2+(9y-3)^2)) )');
disp('             - 1/5 exp( -(9x-4)^2-(9y-7)^2 )');
function f = tf5( x, y )
  f  = 0.75*exp(-((9*x-2).^2+(9*y-2).^2)/4);
  f += 0.75*exp(-((9*x+1).^2/49+(9*y+1).^2/10));
```

```
40    f += 0.5*exp(-((9*x-7).^2+(9*y-3).^2)/4);
41    f -= 0.2*exp(-((9*x-4).^2+(9*y-7).^2));
42  endfunction
43  % Stueckweise lineare Funktion
44  disp('tf6 = @(x,y) max(0,(1-2|x|)) + max(0,(1-2|y|))');
45  function f = tf6( x, y )
46    f = max(0,(1 - 2*abs(x))) + max(0,(1-2*abs(y)));
47  endfunction
```

Quelltext A.14: LadeTestfunktionen.m

```
% BildeAuswertungspunkte.m
%
% Bildet ( res x res ) Auswertungspunkte in Polarkoordinaten und kartesische
% Koordinaten fuer den Einheitskreis und das Einheitsquadrat.
%
% Eingabe:
%   res(1)           - Aufloesung des Gitters der Auswertungspunkte
%   gebiet(String) - Einheitskreis oder -quadrat ('circle', 'cube')
%
% Ausgabe:
%   kart(1:2,1:res^2) - Kartesische Koordinaten (x,y) der Form
%   pol(1:2,1:res^2)  - Polarkoordinaten (theta, r)
function [ kart, pol ] = BildeAuswertungspunkte( res, gebiet )
  % Anzahl Auswertungspunkte
  M = res^2;
  disp(sprintf('\n%d Auswertungspunkte in %s\n', M, gebiet));
  switch( gebiet )
    case 'circle'
      % Polarkoordinaten
      [ theta_grid, r_grid ] = meshgrid( linspace( 0, 2 * pi, res ), linspace( 0, 1, res ) );
      theta = reshape( theta_grid, 1, M ); r = reshape( r_grid, 1, M );
      % Kartesische Koordinaten
      [ x, y ] = pol2cart( theta, r );
    case 'cube'
      % Kartesische Koordinaten
      [ x_grid, y_grid ] = meshgrid( linspace( 0, 1, res ), linspace( 0, 1, res ) );
      x = reshape( x_grid, 1, M ); y = reshape( y_grid, 1, M );
      % Polarkoordinaten
      [ theta, r ] = cart2pol( x, y );
    otherwise
      break;
  end
  kart = [ x; y ];
  pol = [ theta; r ];
endfunction
```

Quelltext A.15: BildeAuswertungspunkte.m

```
% LadeInterpolationspunkte.m
%
% Gibt Interpolationspunkte zur Methode method in
% Polarkoordinaten.
%
% Eingabe:
%   method(string) - Methode, kann 'rd', 'hd', 'sp' oder
%                    {'ev','md','mn'} sein
%   nbr(string)    - Nummer der Interpolantionspunkte
%                    ('09' = 100, '19' = 400, '29' = 900);
%
% Ausgabe:
%   theta(N,1) - Winkel
%   r(N,1)     - Radius
%
% Literatur
%  [1] - Rene Hofmann,
%        'Neue Verfahren zur Approximation mit radialen Basisfunktionen und
%        numerische Untersuchung der Dagum-Funktionen',
%        Dissertation, 2013
%  [2] - UNSW - The University of New South Wales,
%        'Interpolation and Cubature on the Sphere',
%        http://web.maths.unsw.edu.au/~rsw/Sphere/index.html
%        zuletzt abgerufen am 27.04.2013.
switch (method)
  case 'rd'
      rand( 'seed', 4078 );
      theta = 2*pi*rand(1,N); r = rand(1,N);
  case 'hd'
      Points = Halton( p, N, 2 );
      [theta,r]=cart2pol(Points(1,:), Points(2,:));
  case 'sp'
    [X, Y, Z] = sphere(str2num(nbr));
    [theta,r]=cart2pol(X(:),Y(:)); theta = theta'; r = r';
  otherwise
      directory = sprintf('./Hilfsfunktionen/Punkte/%s%s.*', method, nbr);
      [output,filepath] = system(sprintf('ls %s', directory),1);
      X = load( strtrim(sprintf('%s', filepath ) ) );
      [theta,r]=cart2pol(X(:,1),X(:,2)); theta = theta'; r = r';
endswitch
```

Quelltext A.16: LadeInterpolationspunkte.m

```
% PlotteFunktion3D.m
%
% Plotte 3-dimensionale Datenpunkte (x,y,z) und speichere
% den Graph ggf. in filename.eps ab.
%
% Eingabe:
% x,y,z     - Koordinaten der Punkte in der Form (1, res^2) oder
%             im meshgrid-Format (res, res)
% res       - optional, Aufloesung der Punkte zu Umrechnung in das meshgrid-Format
% myview    - optional, view-Parameter
% titel     - Titel des Graphen
% filename  - Dateiname zum speichern des Graphen
function PlotteFunktion3D( x, y, z, res=0, myview=3, caption='', filename=[] )
  figure;
  % x, y, z im meshgrid-Format oder als Zeilen-/Spaltenvektor
  if( res )
    Plt = surf( reshape( x, res, res ), reshape( y, res, res ), reshape( z, res, res ) );
  else
    Plt = surf( x, y, z );
  endif
  % Graustufen
  colormap (flipud(gray(128)));
  % Balken der Farbstufen rechts
  % colorbar('EastOutside');
  colorbar('None');
  % Axen formatieren
  set( gca(), 'linewidth', 1, 'tickdir','out', 'ticklength' ,[0.005,0.005],
               'xtick' ,[-1:0.5:1],  'xticklabel' ,'' ,%  {'-1','-0.5','0','0.5','1'},
               'ytick' ,[-1:0.5:1],  'yticklabel' ,'' ,%  {'-1','-0.5','0','0.5','1'},
               'ztick' ,[0.29:0.4:1.11],   'zticklabel' ,{'0.29','0.70','1.11'},
       'zlim' ,[0.29,1.11]); %[min(min(z)),max(max(z))]);
  set( gca(), 'Fontsize',24 );
  % Titel setzen, falls vorhanden
   title (caption);
  view(myview);
  % Graphen speichern, sofern gewuenscht
  if( length(filename)>0 ) print( filename, '-deps' ); endif
endfunction
```

Quelltext A.17: PlotteFunktion3D.m

```
% Halton.m
%
% Halton.m
%
% Berechnet N d-dimensionale Haltonpunkte zu den Primzahlen p = (p1,...,pd).
%
% Eingabe:
%   p(1:d) - d Primzahlen
%   N(1:d) - Anzahl an Punkten je Dimension
%   d(1)   - Dimension
%
% Ausgabe:
  %   seq(1:d,1:max(N)) - Halton Punkte
function seq = Halton( p, N, d )
  seq = zeros( d, 1 );
  for dim = 1 : d
    prime = p( dim );
    for n = 1 : N( dim )
      Basis = n;
      max_exponent = floor( log( Basis ) / log( prime ) );
      a = [];
      exponent = [ max_exponent : -1 : 0 ];
      for ex = max_exponent : -1 : 0
  a = [ a, floor( Basis / ( prime^ex ) ) ];
  Basis = Basis - floor( Basis / ( prime^ex ) ) * prime^ex;
      endfor
      seq( dim, n + 1 ) = sum( a ./ ( prime .^ ( exponent + 1 ) ) );
    endfor
  endfor
  seq( :, find( norm(seq, 2, 'columns') == 0 ) ) = [];
endfunction
```

Quelltext A.18: Halton.m

```
% generalized_laguerre.m
%
% Berechnet die verallgemeinerte Laguerre Funktion L_alpha^n am Punkt x,
% gibt zudem die Koeffizienten des Polynoms aus.
%
% Eingabe:
%   alpha(1) - Aufloesung des Gitters der Auswertungspunkte
%   n(1)     - Einheitskreis oder -quadrat ('circle',  'cube')
%   x(1,1:d) - d eindimensionale Auswertungspunkte
%
% Ausgabe:
%   y(1,1:d) - Werte an den Stellen x
%   P        - n+1 Koeffizienten
function [y,P] = generalized_laguerre(alpha,n,x)
    d = columns(x);
    M = [n:-1:0]';
    P = (-1).^M .* bincoeff(n+alpha,n-M) ./ factorial(M);
    y = sum( repmat( P, 1, d ) .* repmat( x, n + 1, 1 ) .^ repmat( M, 1, d ), 1);
endfunction
```

Quelltext A.19: generalized_laguerre.m

```
% hyperg_1F2.m
%
% Berechnet die hypergeometrische Reihe 1F2(a1;b1,b2;z) am Punkt x
% und zu den Parametern a1, b1, b2.
%
% Eingabe:
%   alpha(1), b1(1), b2(1) - Parameter der hypergeometrischen Reihe
%   x(1,1:d) - d eindimensionale Auswertungspunkte
%
% Ausgabe:
%   y(1,1:d) - Werte an den Stellen x
function y = hyperg_1F2(a1, b1, b2, x)
    lmax = 11;
    l   = [0:lmax-1];
    L   = repmat(l', 1, columns(a1));
    A1 = repmat(a1, lmax, 1);
    B1 = repmat(b1, lmax, 1);
    B2 = repmat(b2, lmax, 1);
    coeff  = gamma(L + A1)./gamma(A1) .* ...
             gamma(B1)./gamma(L + B1) .* gamma(B2)./gamma(L + B2);
    coeff(1,1)  = 1;
    for  col  = 1:columns(x)
  X  = repmat(x(:,col),  1,  lmax);
  y(:, col)  = sum( coeff' * ( X.^repmat(l,rows(x),1)./ factorial (repmat(l,rows(x),1))  )',  1)';
    end
end
```

Quelltext A.20: hyperg_1F2.m

Symbolverzeichnis

$\mathbb{N}$, $\mathbb{N}_0$	Menge der natürlichen Zahlen, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
$\mathbb{N}^d$	Menge der d-dimensionalen natürlichen Zahlen
$\mathbb{Z}$, $\mathbb{Z}^+$	Menge der ganzen Zahlen, Menge der nichtnegativen ganzen Zahlen
$\mathbb{Z}^d$	Menge der d-dimensionalen ganzen Zahlen
$\mathbb{R}$,$\mathbb{R}^+$	Menge der reelen Zahlen, Menge der nichtnegattiven reelen Zahlen
$\mathbb{R}^d$	Menge der d-dimensionalen reelen Zahlen
$\mathbb{C}$	Menge der komplexen Zahlen
$L_1(\mathbb{R}^d)$	Menge der absolut integrierbare Funktionen $\mathbb{R}^d \mapsto \mathbb{R}$
φ	Basisfunktion $\varphi : \mathbb{R}^+ \mapsto \mathbb{R}$
φ_ε	Basisfunktion $\varphi_\epsilon : \mathbb{R}^+ \mapsto \mathbb{R}$ mit Glättungsparameter $\varepsilon > 0$
φ_c	Basisfunktion $\varphi_c : \mathbb{R}^+ \mapsto \mathbb{R}$ mit Glättungsparameter $c > 0$
$\varphi_{\beta,\gamma}$	Dagum-Funktion mit Parameter $\beta, \gamma > 0$
Φ	radiale Basisfunktion $\Phi : \mathbb{R}^d \mapsto \mathbb{R}$ mit $\Phi(x) = \varphi(\|\|x\|\|_2)$
$\|\|x\|\|_p$	p-Norm des Vektors x
$C^k(I)$	Menge der k-mal stetig differenzierbaren Funktionen, die auf dem Intervall I definiert sind
$\lceil a \rceil$	nächst höhere ganze Zahl, $\lceil \mu \rceil := \min\{m \in \mathbb{Z} \| m \geq a\}$
$(a)_n$	Pochhammer-Symbol zu $a \in \mathbb{R}^+$, $n \in \mathbb{N}_0$
${}_rF_s(\alpha; \beta; z)$	Hypergeometrische Reihe bzgl. $a \in (\mathbb{R}^+)^r$ und $a \in (\mathbb{R}^+)^s$ zum Argument $z \in \mathbb{C}$
$\circ$	Hadamard-Produkt zweier Matrizen
$\otimes$	Kroneckerprodukt zweier Matrizen

Literaturverzeichnis

[1] Abramowitz, Milton und Irene A. Stegun: *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. ninth Dover printing, 1964.

[2] Baxter, Brad: *Conditionally positive functions and p-norm distance matrices*. Constructive Approximation, 7:427 – 440, 1991.

[3] Buhmann, M. D.: *Radial basis functions: theory and implementations*. Cambridge University Press, Cambridge Monographs on Applied and Computational Mathematics, 2003.

[4] E. Porcu, J. Mateu, A. Zini und R. Pini: *The Dagum family for spatio-temporal modelling*. Advances in Applied Probability, 37:1–17, 2006.

[5] Fornberg, Bengt, Elisabeth Larsson und Natasha Flyer: *Stable Computations with Gaussian Radial Basis Functions in 2-D*. submitted to SIAM J. Sci. Comput. Also Technical Report 2009-020, Uppsala University, Department of Information Technology., 2009.

[6] Fornberg, Bengt, Elisabeth Larsson und Natasha Flyer: *Stable computations with Gaussian radial basis functions*. SIAM Journal on Scientific Computing, 33:869 – 892, 2011.

[7] Fornberg, Bengt und G. Wright: *Stable Computation of Multiquadric Interpolants for All Values of the Shape Parameter*. Computers and Mathematics with Applications, 48:853–867, 2004.

[8] Fornberg, Bengt und Julia Zuev: *The Runge Phenomenon and spatially variable shape parameters in RBF Interpolation*. Computers and Mathematics with Applications, 54:379 – 398, 2007.

[9] Gasper, George und Mizan Rahman: *Basic Hypergeometric Series*. Encyclopedia of Mathematics and its Applications, Volume 35, 1990.

[10] Halton, J. H.: *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals.* Numerische Mathematik, 2:84–90, 1960.

[11] Horn, Roger A. und Charles R. Johnson: *Topics in Matrix Analysis.* Cambridge University Press, 1991.

[12] Jackson, Ian R. H.: *Radial Basis Function Methods for Multivariable Approximation.* Dissertation, University of Cambridge, 1988.

[13] Jones, D. S.: *The Theory of Generalised Functions.* Cambridge University Press, 1982.

[14] Liu, Jian Ping und Zong Min Wu: *Generalized Strang-Fix condition for scattered data quasi-interpolation.* Adv. in Comp. Math., 23:201–214, 2005.

[15] Micchelli, C. A.: *Interpolation of scattered data: distance matrices and conditionally positive definite functions.* Constructive Approximation, 1:11–22, 1986.

[16] Porcu, E., J. Mateua, A. Zinib und R. Pini: *Modelling spatio-temporal data: a new variogram and covariance structure proposal.* Statist. Probab. Lett. 77 No. 1, Seiten 83–89, 2007.

[17] Rabinowitz, Philip und George Weiss: *Tables of Abscissas and Weights for Numerical Evaluation of Integrals of the Form* $\int_0^\infty e^{-x}x^n f(x)dx$. Mathematical Tables and Other Aids to Computation, Vol. 13, No. 68:285–294, 1959.

[18] Schoenberg, I.J.: *Metric spaces and completely monotone functions.* Ann. Math., 39:811 – 841, 1938.

[19] Sloan, Ian H. und Robert S. Womersley: *Extremal Systems of Points and Numerical Integration on the Sphere.* Advances in Computational Mathematics, 21:102–125, 2004.

[20] Stein, Elias M. und Guido Weiss: *Introduction to Fourier analysis on euclidean spaces.* Princeton mathematical series. Princeton Univ. Press, Princeton, 1971.

[21] Thatcher, Henry C.: *Conversion of a Power to a Series Of Chebyshev Polynomials.* Communications of te ACM, 7, Number 3:181–182, 1964.

[22] UNSW - The University of New South Wales: *Interpolation and Cubature on the Sphere* . `http://web.maths.unsw.edu.au/~rsw/Sphere/index.html`. zuletzt abgerufen am 27.04.2013.

[23] Uppsala Universitet - Department of Information Technology: *Radial Basis Function Interpolation - RBF-QR.* `http://www.it.uu.se/research/scicomp/software/rbf_qr`. zuletzt abgerufen am 27.04.2013.

[24] Wendland, Holger: *Scattered Data Approximation.* Cambridge University Press, Cambridge Monographs on Applied and Computational Mathematics, 2005.

Erklärung

Ich erkläre: Ich habe die vorgelegte Dissertation selbständig und ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt, die ich in der Dissertation angegeben habe.

Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, und alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht.

Bei den von mir durchgeführten und in der Dissertation erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis, wie sie in der "Satzung der Justus-Liebig-Universität Gießen zur Sicherung guter wissenschaftlicher Praxis" niedergelegt sind, eingehalten.

Gießen, Mai 2013

Unterschrift