# Das Regularized-Principal-Manifold-Prinzip

Carsten Wasserfuhr

Geboren am 23. März 1993 in Troisdorf

1. September 2016

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jochen Garcke

Zweitgutachter: Prof. Dr. Michael Griebel

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

# Inhaltsverzeichnis

## Danksagung

An dieser Stelle möchte ich mich zunächst einmal bei allen bedanken, die mir bei der Anfertigung dieser Arbeit geholfen und mich unterstützt haben. Besonders bedanken möchte ich mich bei Herrn Prof. Jochen Garcke, der mir die Möglichkeit gegeben hat in einem solch spannenden Themengebiet zu arbeiten und mich dabei intensive betreut hat.
Des Weiteren möchte ich mich bei Bastian Bohn für die ideenreichen Diskussionen bedanken, die den Verlauf der Bearbeitung und den Inhalt meiner Arbeit wesentlich beeinflusst haben. Meinen Kommilitonen Simon Hunold, Jonas Röhrig und Tobias Bornheim danke ich für die hilfreichen Diskussionen und die Korrektur der ersten Fassung dieser Arbeite. Nicht zuletzt danke ich meiner Familie für das Korrekturlesen und die moralische Unterstützung.

# 1 Einleitung

In vielen realen Problemen der Datenanalyse treten große Mengen hochdimensionalen Daten auf. Zur Analyse dieser Daten benötigt man leistungsfähige Verfahren und Algorithmen, die diese Datenmengen in kurzer Zeit schnell und effizient auswerten.
Viele dieser Verfahren und Algorithmen aus den Bereichen des Data-Minings, des maschinellen Lernens und der statistischen Lerntheorie beruhen auf der Tatsache, dass hochdimensionale Daten häufig auf einer niedrigdimensionalen Mannigfaltigkeit liegen. Dies bedeutet, dass man die Daten einfacher beschreiben kann. Häufig lassen sich diese Daten sogar als Teilmenge des zwei- oder dreidimensionalen Raums beschreiben. Ein Beispiel dafür ist die Bewertung von Crashtest-Simulationen. Der Raum der Simulationsergebnisse kann dabei eine Dimension von 150.000.000 (für 3.000.000 Finite-Elemente-Konten und 50 simulierten Zeitschritten) annehmen und kann, je nachdem welches Bauteil des Autos analysiert wird, auf zwei oder drei Dimensionen reduziert werden [Sim]. Durch diese Reduktion der Dimension der Ergebnisse kann man beispielsweise mehrere Simulationen durchführen um herauszufinden, welche Koordinaten die Aufhängung einer Stoßstange haben sollte. Dabei lassen sich die gewonnenen Ergebnisse wesentlich leichter im zwei- oder dreidimensionalen Raum vergleichen, als in 150.000.000 Dimensionen. Auch das Einschränken der Simulation auf ein Model mit 60.000 Knotenpunkte und einem Zeitschritt ergibt zwar eine deutlich moderatere Anzahl an Dimensionen, trotzdem sind 60.000 Dimensionen nicht praktikabel genug. Dieses Beispiel zeigt, dass man Verfahren der Datenanalyse auf das Lernen von Mannigfaltigkeiten zurückführen kann.
Das Ziel dieses Lernprozesses soll es sein, eine $d$-dimensionale Mannigfaltigkeit $\mathcal{M}$ (eingebettet in den $\mathbb{R}^n$), auf der die gegebenen Daten Daten liegen, zu erfassen. Genauer gesagt möchte man aus den Daten zum einen Informationen und Regelmäßigkeiten erkennen und zum anderen Vorhersagen für mögliche neue Daten erstellen. Um die genannten Ziele zu erreichen sucht man häufig nach einer Funktion, welche die hochdimensionalen Daten in einen niedrigdimensionalen Raum transformiert. Diese Funktion liefert dann eine niedigdimensionale Darstellung der Daten. Eine solche Funktion, welche die Dimension der Daten verringert, wird in den Bereich der Dimensionsreduktion eingeordnet.
Wenn man auf diese Weise eine niedrigdimensionale Darstellung der Daten erreicht, hat man viele Möglichkeiten mit diesen Daten weiterzuarbeiten. Beispielsweise hat man die Möglichkeit, die Daten zu clustern, also in verschiedene Klassen einzuteilen und somit vorhandene Strukturen der Daten sichtbar zu machen. Eine weitere Möglichkeit ist das Interpolieren der Daten. Dies kann zur Vorhersage neuer Daten genutzt werden, da man dadurch neue Datenpunkte erhält und ungefähr einschätzen kann, welches Verhalten für diese neuen Punkten eintritt.
Aus diesen Überlegungen ergibt sich eine weitere Möglichkeit des Lernens von Daten. Diese nutzt den Ansatz der Rekonstruktion der Daten auf Grundlage des niedrigdimensionalen Raums. Ein großer Vorteil ist der Erhalt einer niedrigdimensionalen Darstellung der Daten

und einer Funktion, welche neue Daten produzieren kann. Diese kann zur Vorhersage weiterer Daten genutzt werden. An dieser Stelle setzt meine Arbeit an.
Grundlage meiner Arbeit sind die Resultat der Arbeit „Regularized principal manifolds“ [SMSW01] aus dem Jahre 2001 von Alexander J. Smola, Sebastian Mika, Bernhard Schölkopf und Robert C. Williamson. Mein Ziel ist es, ein Verfahren vorzustellen, mit dem eine solche Funktion berechnet werden kann. Dazu werden zunächst die theoretischen Grundlagen des Verfahrens beschrieben (Kapitel 2). Dabei werden Methoden aus dem Bereich des Hilbertraums mit reproduzierendem Kern (engl. reproducing kernel hilbert space) genutzt, um zum einen die gesuchte Funktion darzustellen und zum anderen einen Minimierungsterm zu erstellen. Dieser wird genutzt, um eine Funktion zu berechnen, die einen möglichst kleinen Rekonstruktionsfehler zulässt. Danach wird der Algorithmus inklusive der Implementierung beschrieben (Kapitel 3 und 4). Im Anschluss daran wird das Verfahren auf synthetische Daten angewandt und getestet (Kapitel 5). Dabei wird zunächst versucht, die in [SMSW01] präsentierten Ergebnisse nachzuvollziehen und nachzustellen. Gleichzeitig wird das Verfahren auf weitere Datenmengen angewendet und analysiert. Zum Schluss wird in Kapitel 6 die Tauglichkeit des Algorithmus eingeordnet und bewertet.

# 2 Theoretische Grundlagen

Ein Ziel dieser Arbeit ist die Erarbeitung und Analyse eines Verfahrens der Datenanalyse. Um dieses Verfahren in allen Einzelheiten genauer zu verstehen, soll zunächst eine grobe Einordnung des Verfahrens in den Bereich der Datenanalyse und des maschinellen Lernens vorgenommen werden. Dazu formulieren wir die komplexe Aufgabenstellung der Datenanalyse in ein mathematisches Konzept um. Auf Grund der Tatsache, dass hochdimensionale Daten oft auf einer niedrigdimensionalen Mannigfaltigkeit liegen, kann man sich auf das Lernen von Mannigfaltigkeiten beschränken. Das bedeutet, dass die Daten im hochdimensionalen Raum häufig eine wesentlich einfachere Struktur besitzen und durch wenige Dimensionen dargestellt werden können. Diese Idee wollen wir nun mit einer Bijektion $\phi$ zwischen einer $d$-dimensionalen Mannigfaltigkeit $\mathcal{M}$ und einem $d$-dimensionale Raum $\mathcal{D}$ (z.B $[0,1]^d$) umsetzen, also

$$\phi : \mathcal{M} \leftrightarrow \mathcal{D}.$$

Dabei soll jeder Datenpunkt durch einen Punkt im niedrig-dimensionalen Raum $\mathcal{D}$ repräsentiert werden und die Struktur der Mannigfaltigkeit möglichst erhalten bleiben. Allerdings ist das Berechnen einer solchen Bijektion in der Realität nicht möglich. Eine weitverbreitete und häufig genutzte Idee dieses Problem zu umgehen ist die Bijektion in zwei Funktionen $g : \mathbb{R}^n \to \mathcal{D}$ und $f : \mathcal{D} \to \mathbb{R}^n$ aufzuteilen. Dabei soll die Funktion $g$ die Funktion $\phi$ und $f$ die inverse Funktion $\phi^{-1}$ simulieren.

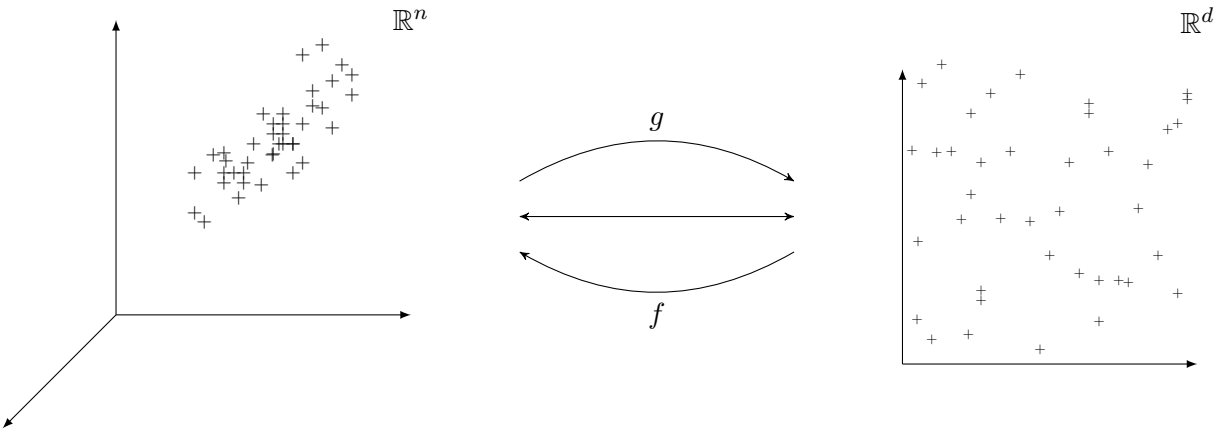



Abbildung 2.1: Graphische Darstellung der Funktionen $g : \mathcal{M} \to \mathcal{D}$ und $f : \mathcal{D} \to \mathcal{M}$.

Anschaulich sollen dabei die hochdimensionalen Daten durch die Funktion $g$ so in einen $d$-dimensionalen Raum abgebildet werden, dass die Daten dort bezüglich der Koordinaten der

Mannigfaltigkeit dargestellt werden. Der Vorteil der niedrigdimensionalen Darstellung ist die Tatsache, dass man in wenigen Dimensionen genauer arbeiten kann und die Ergebnisse des Verfahrens leichter kontrollieren kann.
All diese Verfahren, die versuchen, eine solche Funktion $g$ zu konstruieren, werden dem Bereich der Dimensionsreduktion zugeordnet. Das wohl bekannteste und am häufigsten genutzte Verfahren in diesem Bereich ist die Hauptkomponentenanalyse (principal component analysis (PCA)) [LV07; Hot33], welches lineare Zusammenhänge erfasst. Oftmals treten in der Praxis aber nichtlineare Zusammenhänge auf, so dass eine lineare Dimensionsreduktion nicht ausreichend ist um einen solchen Datensatz hinreichend gut in einem niedrigdimensionalen Raum zu beschreiben. Ein anschauliches Beispiel dafür ist der „Swiss Roll“-Datensatz, welcher in der nachfolgenden Abbildung 2.2 dargestellt ist.

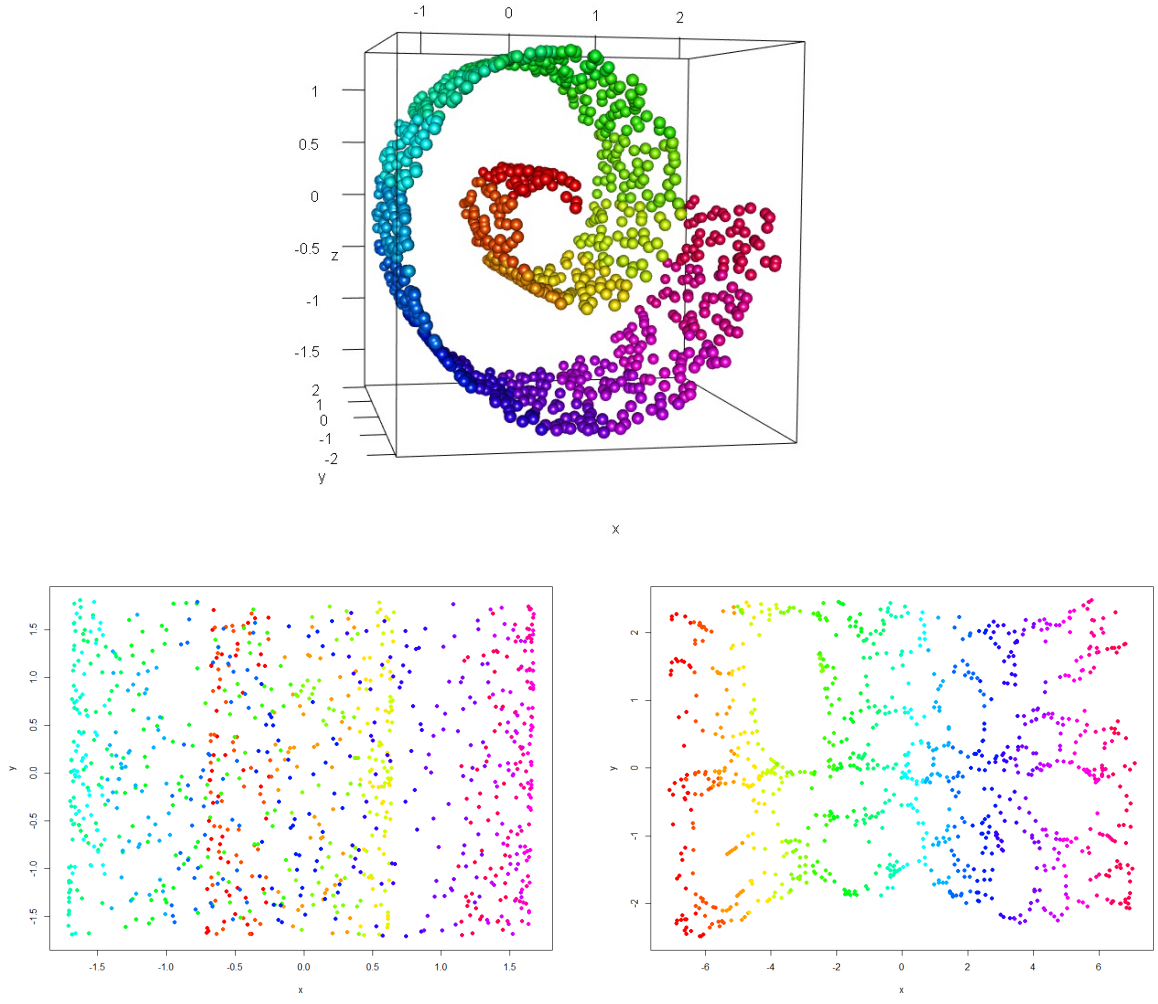



Abbildung 2.2: Dargestellt ist die Swiss Roll mit 1000 Datenpunkten im $\mathbb{R}^3$ (oben) und ihre zweidimensionale Darstellung mit der PCA (links) und dem Isomap-Algorithmus (rechts). Abbildungen aus [Swi]

Der “Swiss Roll“-Datensatz besteht aus 1000 Testpunkten, welche auf einer spiralförmigen Oberfläche liegen. Diese Oberfläche wird als eine zweidimensionale Mannigfaltigkeit in einem dreidimensionalen Raum aufgefasst und wird mit einem dimensionsreduzierenden Verfahren in einen zweidimensionalen Raum eingebetten. Dazu wurden in Abbildung 2.2 zum einen eine lineare Dimensionsreduktion in Form der PCA [LV07] und zum andern eine nichtlineare Dimensionsreduktion in Form des Isomap-Algorithmus [LV07] verwendet. Betrachtet man den Datensatz im obrigen Bild, kann man einen flüssigen Farbverlauf entlang der Spiraloberfläche erkennen. Ziel einer Dimensionsreduktion ist es die Struktur der Mannigfaltigkeit zu erhalten, in diesem Fall den Farbverlauf. Wie man erkennt, berechnet das Isomap-Verfahren das gewünschte Ergebnis. Die PCA hingegen führt zu keinem brauchbaren Ergebnis, da zusammengehörende Daten nicht zusammengehörig dargestellt sind. Dies ist in Abbildung 2.2 durch keinen eindeutigen Farbverlauf sichtbar.
Weitere bekannte Beispiele für nichtlineare Dimensionsreduktionen sind unter anderem Isomap [TSL00], Kernel-PCA [SSM99] und Laplacian Eigenmaps [BN03].
Anders als die gerade beschriebenen Verfahren lässt sich das Principal-Manifold-Learning-Verfahren (kurz PML-Verfahren) in die Klasse der Verfahren einordnen, welche versuchen, eine Funktion $f : \mathcal{D} \to \mathcal{M}$ zu finden. Dabei versucht die Funktion $f$, auf Grundlage des niedrigdimensionalen Raums $\mathcal{D}$, die Daten zu rekonstruieren. Das heißt, dass man für einen Datenpunkt $x \in \mathbb{R}^n$ ein $\zeta \in \mathcal{D}$ sucht, sodass der Fehler zwischen dem Datenpunkt und der Rekonstruktion $f(\zeta)$ möglichst gering ist.

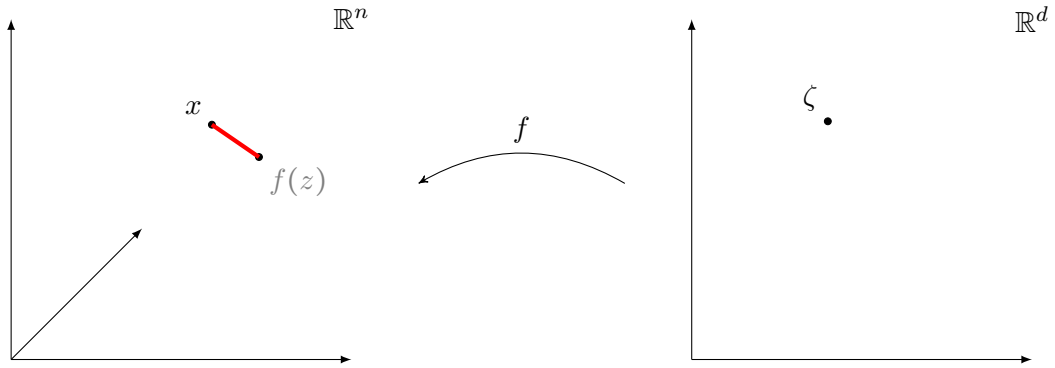



Abbildung 2.3: Graphische Darstellung der Approximation des Datenpunktes $x$ durch $f(\zeta)$. Die rote Linie stellt den Fehler der Approximation $f(\zeta)$ an $x$ dar.

Sucht man für alle Datenpunkte $x$ eine solche Approximation $f(\zeta)$, so bildet die Menge aller $\zeta$ auch eine niedrigdimensionale Darstellung der Daten, da man aus $x = f(g(x))$ den Zusammenhang $\zeta \approx g(x)$ herleiten kann. Das bedeutet insbesondere, dass man sowohl eine niedrigdimensionale Darstellung der Daten, als auch eine Funktion erhält, die zudem neue Datenpunkte berechnet und daher Vorhersagen treffen kann. Demnach ist diese Möglichkeit der Datenanalyse in der Theorie deutlich weitreichender als die zuvor beschriebene Idee der Dimensionsreduktion, aber in der Praxis auch deutlich schwerer umzusetzen.
Da die Rekonstruktion einer Funktion aus gegeben Daten ein schlecht gestelltes Problem ist [Gar04; TA77], werden wir die Regularisierungstheorie nach Tikhonov [Wah90; A.63] nutzen. Ziel der nächsten Abschnitte wird es sein, diese Theorie auf unsere Situation anzupassen.

## 2.1 Das Quantisierungsfehler-Funktional

Wir möchten nun ausführlicher die Theorie betrachten, welche die Grundlage des PML-Verfahrens sein wird. Dabei orientieren wir uns nachhaltig an den Kapiteln 2 und 3 aus [SMSW01]. In dieser Arbeit werden hauptsächlich zwei Räume betrachtet. Zunächst definiert man einen (euklidischen) Ausgangsraum $\mathcal{X} \subseteq \mathbb{R}^n$ mit $X := \{x_1, \ldots, x_m\} \subset \mathcal{X}$. Das heißt, wir erhalten gegebene Datenpunkte $\{x_i\}$ aus einem $n$-dimensionalen Raum als $n$-dimensionale Vektoren. Diese Datenpunkte werden dabei als iid[1] angenommen. Des Weiteren betrachten wir für $d \ll n$ einen Zustandsraum $\mathcal{Z} \subseteq \mathbb{R}^d$ und Funktionen vom Typ $f : \mathcal{X} \to \mathcal{Z}$. Man bezeichnet mit $\mathcal{F}$ die Klasse dieser Funktionen der Form

$$\mathcal{F} := \{f : \mathcal{X} \to \mathcal{Z}\}.$$

Heuristisch gesprochen möchte man nun eine solche Funktion $f \in \mathcal{F}$ finden, welche die Datenmenge $X$ erfasst, analysiert und anschließend zur Vorhersage weitere Vorgänge auswertet. Genauer gesagt sucht man eine Funktion $f : D \to M$, wobei $M \subset \mathbb{R}^n$ eine $d$-dimensionale Mannigfaltigkeit und $D$ ein $d$-dimensionaler Raum ist, welche die Mannigfaltigkeit $M$ aus der Menge $D$ rekonstruiert. Um die Genauigkeit einer solchen Rekonstruktion beschreiben zu können, benötigt man ein Model, welches die mangelnde Genauigkeit als eine Art Fehler ausdrückt.
Dieses Model wird ein Quantisierungsfehler-Funktional sein, auf dem auch das PML-Verfahren aufbaut. Dazu betrachten wir den Ausgangsraum $\mathcal{X}$ und definieren den Quantisierungsfehler $R[f]$:

$$R[f] := \int_{\mathcal{X}} \min_{z \in \mathcal{Z}} c(x, f(z)) \mathrm{d}\mu(x). \tag{2.1}$$

Hierbei ist $c(x, f(z))$ eine Kostenfunktion und $\mathrm{d}\mu(x)$ eine unbekannte Wahrscheinlichkeitsverteilung. In dieser Arbeit wird als Kostenfunktion die euklidische Distanz gewählt:

$$c(x, f(z)) = \|x - f(z)\|_2^2 .$$

Schaut man sich $R[f]$ genauer an, so fällt auf, dass die Bezeichnung „Rekonstruktionsfehler“ eine bessere Wortwahl wäre, da man für eine feste Funktion $f$ und jeden Wert $x \in \mathcal{X}$ die bestmögliche Approximation $f(z)$ mit $z \in \mathcal{Z}$ berechnen möchte. Es fällt auf, dass die bestmögliche Approximation stark von der Funktion $f$ abhängt. Daher möchte man den Rekonstruktionsfehler $R[f]$ nicht nur für eine konkrete Funktion $f$ berechnen, sondern für alle möglichen Funktionen $f$ minimieren und diejenige Funktion $f^*$ finden, so dass gilt:

$$R[f^*] \leq R[f] \quad \forall f \in \mathcal{F}. \tag{2.2}$$

Man hat zwar ein Werkzeug gefunden, um zum einen anzugeben, wie gut die Rekonstruktion ist, und um zum anderen den Erhalt einer optimalen[2] Funktion $f$ zu sichern. Allerdings treten dabei mehrere Probleme auf, welche das Quantisierungsfehler-Funktional nur zu einem theoretischen Modell machen. Eines der zentralen Probleme des Quantisierungsfehler-Funktionals

[1] Das ist geläufige Abkürzung aus dem Englischen für unabhängig identisch verteilt
[2] Hier ist „optimal“ im Sinne von (2.2) zu verstehen.

ist die Annahme, dass man den ganzen Raum $\mathcal{X}$ als etwas kontinuierliches betrachten kann. Dies ist aber eine sehr optimistische Annahme, da man in der Praxis nur die Datenmenge $X$ kennt. Daher muss das kontinuierliche Modell derart ummodelliert werden, so dass es auch auf einen diskreten Raum angewendet werden kann. Dazu ersetzt man zunächst das unbekannte Maß $\mathrm{d}\mu(x)$ durch das empirische Maß

$$\mu_m(x) := \frac{1}{m} \sum_{i=1}^{m} \delta(x - x_i).$$

Anschließend wird in (2.1) das unbekannte Maß $\mathrm{d}\mu(x)$ durch $\mu_m(x)$ ersetzt und man erhält den empirischer Quantisierungsfehler

$$R_{\mathrm{emp}}[f] := \int_{\mathcal{X}} \min_{z \in \mathcal{Z}} c(x, f(z)) \mathrm{d}\mu_m(x) = \frac{1}{m} \sum_{i=1}^{m} \min_{z \in \mathcal{Z}} c(x_i, f(z)). \tag{2.3}$$

Die Idee hinter $R_{\mathrm{emp}}[f]$ ist dabei die selbe wie bei $R[f]$: Man finde für jeden gegebenen Datenpunkt $x_i$ eine beste Approximation $f(z)$ zu $x_i$. Analog wird dabei $f(z)$ eine beste Approximation genannt, falls die Kosten $c(x, f(z))$ minimal sind.

## 2.2 Regularisierer

Bislang wurde stillschweigend angenommen, dass die Funktion $f \in \mathcal{F}$ „schöne“ Eigenschaften hat, also dass sämtliche Fehler-Funktionale wohldefiniert sind. Dies ist im Allgemeinen nicht der Fall, da an die Menge $\mathcal{F}$ keinerlei Restriktionen gestellt wurde. Somit sind weder $R[f]$ noch $R_{\mathrm{emp}}[f]$ immer wohldefiniert. Insbesondere dann nicht, wenn $f$ unstetig ist. Um dieses Problem zu lösen, wird nun ein Regularisierer $Q[f]$ eingeführt.
Nun wird also nicht mehr das empirische Quantisierungsfehler-Funktional bezüglich $f$ minimiert, sondern eine regularisierte Version davon. Dazu definieren wir das regularisierte Quantisierungsfehler-Funktional $R_{\mathrm{reg}}[f]$ wie folgt:

$$R_{\mathrm{reg}}[f] := R_{\mathrm{emp}}[f] + \lambda Q[f]. \tag{2.4}$$

In diesem Kontext bezeichnet $Q[f]$ einen nicht negativen konvexen Regularisierungsterm und $\lambda > 0$ eine Konstante, welche -grob gesagt- den Anteil der Regularisierung steuert. Genauer gesagt beschreibt $\lambda$, wie stark die Funktion $f$ im Vergleich zu Funktionen mit kleinem empirischen Quantisierungsfehler verändert werden muss [SMSW01].
Damit stellt sich die Frage, wie ein solches $Q[f]$ aussieht. Wir werden in dieser Arbeit nur eine quadratische Version des Regularisierungsterms diskutieren.
Die folgenden theoretischen Überlegungen basieren auf der Theorie des Hilbertraums mit reproduzierendem Kern. Im nächsten Abschnitt werden daher Grundlagen und Resultate aus diesem Bereich grob skizziert.

## 2.3 Hilbertraum mit reproduzierendem Kern

Im Laufe der Herleitung des PML-Verfahrens werden wir auf Hilberträume von Funktionen zurückgreifen. Als eine gute Wahl solcher Räume erweisen sich Hilberträume mit reprodu-

zierendem Kern[3]. Dieser bieten den Vorteil, dass man in diesen speziellen Räumen einige hilfreiche Resultate, wie zum Beispiel das Repräsentierungstheorem 2.1, nutzen kann und diese einen einfach zu beschreibenden Algorithmus ermöglichen.
Ein RKHS ist dabei ein Raum von Funktionen $\mathcal{H}$, welche auf einem Gebiet $\Omega$ definiert sind und die folgenden Bedingungen erfüllen:

$$(1) \qquad f(y) = \delta_y(f) = \langle f, K(y,\cdot)\rangle_{\mathcal{H}} \qquad \text{für alle} \quad f \in \mathcal{H}, y \in \Omega \tag{2.5}$$

und

$$(2) \qquad K(x,y) = \langle K(x,\cdot), K(y,\cdot)\rangle_{\mathcal{H}} = \langle \delta_x, \delta_y\rangle_{\mathcal{H}^*} \qquad \text{für alle} \quad x, y \in \mathcal{H}. \tag{2.6}$$

Hierbei bezeichnet $K : \Omega \times \Omega \to \mathbb{R}$ eine Kernfunktion, $\delta_x$ die Punktauswertung an der Stelle $x$ und $\mathcal{H}^*$ den Dualraum von $\mathcal{H}$. Eine Kernfunktion, welche die Bedingungen (2.5) und (2.6) bezüglich des Raumes $\mathcal{H}$ erfüllt, wird reproduzierender Kern genannt.
Ein erstes Resultat beschreibt den Zusammenhang zwischen der Kernfunktion und einem Hilbertraum von Funktionen. Es kann gezeigt werden, dass zu jedem Hilbertraum von Funktionen eine Kernfunktion $K$ existiert, welche die Bedingungen (2.5) und (2.6) erfüllt. Umgekehrt kann ein solcher Hilbertraum zu einer festen Kernfunktion $K$ gefunden werden, so dass $K$ ein reproduzierender Kern ist. Wir nehmen diese Tatsache als gegeben an, ohne weiter genauer darauf einzugehen. Weitere Resultate, wie wir gleich sehen werden, nutzen diese Tatsache beziehungsweise setzen diese Tatsache voraus.
Eine weitere nützliche Eigenschaft des RKHS ist es, dass man eine kernbasierte Darstellung der Funktion $f$ herleiten kann, siehe [Sch11]. Dazu definiert man eine Menge $Z \subset \mathcal{Z}$ mit $|\, Z \,| = M < \infty$ und betrachtet den Unterraum

$$\mathcal{H}_Z = \overline{\operatorname{span}\{K(z,\cdot) \mid z \in Z\}} \subseteq \mathcal{H}.$$

Nun teilen wir den Raum $\mathcal{H}$ auf in den Unterraum $\mathcal{H}_Z$ und dessen orthogonales Komplement $\mathcal{H}_Z^\perp$; also $\mathcal{H} = \mathcal{H}_Z \oplus \mathcal{H}_Z^\perp$. Zusammen mit dieser Zerlegung kann nun eine Funktion $f \in \mathcal{H}$ in

$$f = f_Z + f_Z^\perp \tag{2.7}$$

aufgeteilt werden mit $f_Z \in \mathcal{H}_Z$ und $f_Z^\perp \in \mathcal{H}_Z^\perp$. Aus diesen Überlegungen kann man folgern, dass die Interpolierende $f_Z \in \mathcal{H}_Z$, definiert als

$$f(\cdot) := f_Z(\cdot) = \sum_{j=1}^{M} \alpha_j K(z_j,\cdot), \quad \alpha_j \in \mathbb{R}^n, \tag{2.8}$$

die beste Approximation an eine Funktion $\tilde{f} \in \mathcal{H}$ auf $Z$ ist, siehe [Sch11]. Man beachte an dieser Stelle, dass es sich bei den $\alpha_i$ für $1 \le i \le M$ nicht um Skalare handelt, sondern um Vektoren. Analog berechnet die Kernfunktion $K(z_j,\cdot)$ keinen Vektor, sondern einen Skalar. An dieser Stelle verleitet diese Notation gerne dazu, umgekehrt zu denken, da man klassischer Weise zunächst ein Skalar und anschließend eine vektorielle Basisfunktion in der Funktionsdarstellung betrachtet.
Diese Zerlegung einer Funktion $f$ lässt auch einen einfachen Beweis eines bekannten und

[3] Eine übliche Abkürzung ist RKHS (repruducing kernel hilbert space). Diese Bezeichnung wird im weiteren Verlauf benutzen

wichtigen Theorems zu. Auf Grundlage von Wahba's klassischem Repräsentierungstheorem [Wah90] wurde 2001 das, im Bereich der kernbasierten Verfahren bekannte, Repräsentierungstheorem [SHS01] formuliert und bewiesen. Einige der benötigten Begriffe werden hier nicht näher erläutert und sind alle in [SS02; Gar16] zu finden.

**Theorem 2.1** (Repräsentierungstheorem)**.** *Sei $s : [0, \infty] \to \mathbb{R}$ eine streng monoton fallende Funktion, $\lambda > 0$, $\Omega$ eine Menge, $Y \subset \mathbb{R}$, $\mathcal{H}$ eine RKHS auf der Menge $\Omega$ und $c : \Omega \times Y \times \mathbb{R} \to [0, \infty)$ eine Verlustfunktion. Dann gilt für die Daten $D := \{(x_i, y_i)\}_{i=1}^N, x_i \in \Omega, y_i \in Y$, dass jede minimierende Funktion $f \in \mathcal{H}$ des regularisierten empirischen Fehlerfunktionals*

$$R_{l,reg}[f] = \frac{1}{N} \sum_{i=1}^{N} l(x_i, y_i, f(x_i)) + \lambda \, s(\|f\|_{\mathcal{H}})$$

*eine Darstellung der Form*

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x)$$

*für $f \in \mathcal{H}_X$, $X = \{x_1, \ldots, x_N\}$, zulässt.*

Ein ausführlicher Beweis dazu kann in [SS02; Gar16] gefunden werden. Der Beweis beruht hauptsächlich auf der Zerlegung (2.7) der minimierenden Funktion $f$. Man nutzt die Tatsache, dass $f_Z$ auf der Menge $Z$ die beste Approximation ist und zeigt, dass $f_Z^{\perp} = 0$ ist. Aus dem Repräsentierungstheorem folgt dann, dass die beste Approximation $f$ unter allen Funktionen in $\mathcal{H}$, welche (2.4) minimiert, schon exakt durch die Darstellung (2.8) bestimmt ist.

## 2.4 Quadratischer Regularisierer

Als eine gute Wahl eines solchen Regulierers hat sich ein quadratisches Funktional erwiesen. Dazu betrachtet man zunächst einen Regularisierungsoperator $P$ und definieren diesen Operator wie in [SS02] als eine lineare Funktion auf dem Funktionenraum $\{\psi \mid \psi : \Omega \to \mathbb{R}\}$ in einen Raum $\Gamma$, welcher ein Skalarprodukt besitzt.
Zusammen mit dem Regularisierungsoperator $P$ definieren wir den Regularisierungsterm $Q[f]$ als

$$Q[f] = \frac{1}{2}\langle Pf, Pf\rangle = \frac{1}{2}\|Pf\|^2.$$

Der Operator $P$ überführt dabei nichtglatte Funktionen in einen Hilbertraum. Genauer gesagt wird nun als Hilbertraum ein *repruducing kernel hilbert space* (RKHS) $\mathcal{H}$ gewählt.
Nachdem wir in Abschnitt 2.3 einige Eigenschaften des RKHS dargestellt haben, wollen wir diese nun mit der Theorie des Quantisierungsfehlers kombinieren. Dazu betrachten wir zunächst die Darstellung von $f$ in (2.8). Dazu sei eine Menge $Z := \{z_1, \cdots, z_M\} \subset \mathcal{Z}$, welche

zur Funktionsauswertung der Kernfunktion genutzt wird, gegeben. Wir definieren nun die Darstellung der gesuchten Funktion $f$ als

$$f(z) = \sum_{i=1}^{M} \alpha_i K(z_i, z) \text{ mit } \alpha_i \in \mathcal{X}, K : \mathcal{Z}^2 \to \mathbb{R}. \tag{2.9}$$

Durch die Darstellung der gesuchten Funktion $f$ als Summe von Kernfunktionen können wir besser beschreiben, wie die Funktion $f$ aussieht.

Wir betrachten nun den Regularisierungsterm $Q[f]$ genauer. Das Einsetzen der Darstellung von $f$ aus 2.9 ergibt

$$\begin{aligned}
\|Pf\|^2 &= \langle Pf, Pf \rangle \\
&= \langle P\left(\sum_{i=1}^{M} \alpha_i K(z_i, \cdot)\right), P\left(\sum_{j=1}^{M} \alpha_i K(z_j, \cdot)\right) \rangle \\
&= \langle \sum_{i=1}^{M} \alpha_i PK(z_i, \cdot), \sum_{j=1}^{M} \alpha_i PK(z_j, \cdot) \rangle \\
&= \sum_{i=1}^{M} \sum_{j=1}^{M} \langle \alpha_i, \alpha_j \rangle \langle PK(z_i, \cdot), PK(z_j, \cdot) \rangle .
\end{aligned}$$

Zwischen Regularisierungsoperatoren und Hilberträumen mit reproduzierendem Kern wurde 1998 ein Zusammenhang gefunden und bewiesen [SS02; SSM98].

**Theorem 2.2.** *Für jeden RHKS $\mathcal{H}$ mit reproduzierendem Kern $K$ existiert ein dazugehöriger Regularisierungsoperator $P : \mathcal{H} \to \mathcal{D}$, sodass für alle $f \in \mathcal{H}$ gilt:*

$$f(x) = \langle PK(x, \cdot), Pf(\cdot) \rangle_{\mathcal{D}} . \tag{2.10}$$

*Insbesondere gilt für $f(\cdot) = K(\cdot, x')$*

$$\left\langle PK(x, \cdot), PK(x', \cdot) \right\rangle_{\mathcal{D}} = K(x, x'). \tag{2.11}$$

*Umgekehrt gibt es für jeden Regularisierungsoperator $P : \mathcal{L} \to \mathcal{D}$, wobei $\mathcal{L}$ ein Funktionenraum zusammen mit einem Skalarprodukt und einer zugehörigen Green-Funktion für $P^\star P$ ist, einen dazugehörigen RKHS $\mathcal{H}$ mit Kern $K$, so dass (2.10) und (2.11) erfüllt sind.*

Ein ausführlicher Beweis findet sich in [SS02]. Wenden wir Theorem 2.2 auf die Regularisierungstheorie von oben an, so ergibt sich

$$K(z_i, z_j) := \langle PK(z_i, \cdot), PK(z_j, \cdot) \rangle \tag{2.12}$$

für alle $z_i, z_j \in \mathcal{Z}$. Nutzen wir diesen Zusammenhang aus, ergibt sich als Regularisierungsterm

$$\|Pf\|^2 = \sum_{i,j=1}^{M} \langle \alpha_i, \alpha_j \rangle K(z_i, z_j)$$

Nun haben wir sowohl für unsere unbekannte Funktion $f$, als auch für unseren Regularisierungsterm $Q[f]$ eine diskrete Funktionsdarstellung gefunden. Das Einsetzen der oben erarbeiteten Aussagen in das regularisierte Quantisierungsfehler-Funktional ergibt:

$$\begin{aligned} R_{\text{reg}}[f] :&= R_{\text{emp}}[f] + \frac{\lambda}{2}\|Pf\|^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}\min_{z\in\mathcal{Z}}\|x_i - f(z)\|^2 + \frac{\lambda}{2}\|Pf\|^2. \\ &= \frac{1}{m}\sum_{i=1}^{m}\min_{z\in\mathcal{Z}}\|x_i - \sum_{i=1}^{M}\alpha_i K(z_i, z)\|^2 + \frac{\lambda}{2}\sum_{i,j=1}^{M}\langle\alpha_i, \alpha_j\rangle K(z_i, z_j). \end{aligned}$$

# 3 Das Verfahren

In diesem Kapitel wird unter Verwendung der theoretischen Resultate aus Kapitel 2 ein Verfahren vorgestellt, welches den Regularisierungsterm $R_{\text{reg}}[f]$ minimiert. Dabei bewegt man sukzessive Vektoren im Raum $\mathcal{Z}$ und passt anschließend die Hilfsvektoren im Ausgangsraum an. Das PML-Verfahren verhält sich dabei analog zu dem EM-Algorithmus aus der mathematischen Statistik, siehe [DLR77].
Der große Vorteil dieses Verfahrens ist seine Einfachheit, da man es zum einen sehr gut beschreiben, zum anderen gut in ein Programm umsetzen kann.
Man beachte, dass an dieser Stelle bewusst nicht von einem optimalen Verfahren im Sinne der Konvergenz oder Geschwindigkeit gesprochen wird [SMSW01]. Das Hauptaugenmerk liegt darauf einen Algorithmus zu präsentieren, welcher in das Konzept der kernbasierten Funktionsdarstellung passt und in der Praxis solide umsetzbar ist.
Die nun folgenden Abschnitte orientieren sich sowohl inhaltlich als auch in ihrer Reihenfolge an Kapitel 5 aus [SMSW01].

## 3.1 Der Minimierungsterm

Im Folgenden nehmen wir ohne Beschränkung der Allgemeinheit an, dass unsere Datenmenge $X$ zentriert ist. Dies bietet den Vorteil, auf diverse Konstanten verzichten zu können und vereinfacht somit das Verfahren. Wir haben in Kapitel 2 das regularisierte Quantisierungsfehler-Funktional

$$\begin{aligned} R_{\text{reg}}[f] &:= R_{\text{emp}}[f] + \lambda Q[f] \\ &= \frac{1}{m}\sum_{i=1}^{m} \min_{z\in\mathcal{Z}} \|x_i - f(z)\|^2 + \frac{\lambda}{2}\|Pf\|^2. \end{aligned}$$

eingeführt und hergeleitet. Des Weiteren haben wir gesehen, dass sich unter Ausnutzung verschiedener Resultate aus dem Bereich des RKHS die Funktion $f$ von der Form (2.8) schreiben lässt. Aus dieser Darstellung von $f$ konnten wir außerdem eine Darstellung des Regularisierungsterms herleiten, s. (2.12). Betrachtet man diese Darstellungen genauer, so stellt man fest, dass beide Terme von den Variablen $\{\alpha_1, \ldots, \alpha_M\}$ abhängen. Daher lässt sich nun die Minimierung des regularisierten Quantisierungsfehler-Funktionals schreiben als

$$\min_{\substack{\{\zeta_1,\ldots,\zeta_m\}\subset\mathcal{Z}\\ \{\alpha_1,\ldots,\alpha_M\}\subset\mathcal{X}}} \left[\frac{1}{m}\sum_{i=1}^{m} c(x_i, f_{(\alpha_1,\ldots,\alpha_M)}(\zeta_i)) + \lambda Q(\alpha_1,\ldots,\alpha_M)\right] \tag{3.1}$$

Dies ergibt eine doppelte Minimierung. Zum einen möchte man für die Datenpunkte $x_i \in X$ die beste Approximation $\zeta_i \in Z$ bezüglich der Funktion $f$ finden (Projektion), zum anderen

den Regularisierungsfehler möglichst klein halten (Adaption). Dies führt dazu, dass sowohl $\{\zeta_1, \ldots, \zeta_m\} \subseteq \mathcal{Z}$ als auch $\{\alpha_1, \ldots, \alpha_M\} \subseteq \mathcal{X}$ in ihren zugrundeliegenden Räumen verschoben werden.

## 3.2 Der Algorithmus

In diesem Abschnitt wird nun genauer auf den Aufbau und die Grundbausteinen des PML-Algorithmus eingegangen. Hauptsächlich ist der Algorithmus aus einem Projektions- und einem Adaptionschritt aufgebaut und iteriert abwechselnd zwischen diesen zwei Schritten. Das bedeutet, dass in einer Iteration sowohl ein Projektions- als auch ein Adaptionsschritt durchgeführt wird. Um die erste Iteration überhaupt zu starten, benötigt man zunächst gewisse $\alpha_i$ als eine Art Startzustand. Eine genauere Wahl wird in 3.2.3 beschrieben. Zusammengefasst ergibt sich folgender grundlegender Algorithmus:

**Algorithmus 1 :** Grundschema des PML-Algorithmus

**Eingabe** : $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n$, $Z = \{z_1, \ldots, z_M\} \subseteq \mathbb{R}^d$
**Ausgabe** : $\alpha_{aktuell}$, Approximationen $Xeta$
berechne gute Startvektoren $\alpha_i$
**while** *ABBRUCHBEDINGUNG nicht erfüllt* **do**
  (1) Projektion
  (2) Adaption

Ausgabe wird die Menge $\alpha = \{\alpha_1, \ldots, \alpha_M\} \subseteq \mathbb{R}^n$ und die Menge $\{\zeta_1, \ldots, \zeta_m\} \subseteq \mathbb{R}^d$ sein. Diese werden zum einen benötigt um die Funktion $f$ durch 2.9 als Linearkombination aus den $\alpha_i$ und den Kernfunktionen darzustellen und zum anderen um eine niedrigdimensionale Darstellung der Daten zu erhalten.

In den folgenden Abschnitten werden nun die einzelnen Bausteine des PML-Algorithmus genauer erläutert, wobei einige Rechnungen analog zu denen in Kapitel 3.2 aus [Gar04] geführt werden.

### 3.2.1 Projektion

Um die oben beschriebene doppelte Minimierung genauer zu untersuchen, betrachten wir die Vektoren $\{\alpha_1, \ldots, \alpha_M\}$ zunächst einmal als fest, und minimieren über die Variablen $\{\zeta_1, \ldots, \zeta_m\}$. Dies hat zur Folge, dass der Regularisierungsterm $Q[f]$ konstant bleibt und sich bezüglich einer Minimierung nach $\{\zeta_1, \ldots, \zeta_m\}$ nicht ändert, da $Q[f]$ nur von den Vektoren $\{\alpha_1, \ldots, \alpha_M\}$ abhängig ist. Daher kann dieser in dem ersten Schritt vernachlässigt werden. Wir minimieren nun (3.1) nicht komplett, sondern berechnen für jeden Datenpunkt $x_i$ ein Approximation, das heißt für jedes $i \in \{1, \ldots, m\}$ wähle man

$$\zeta_i := \operatorname*{argmin}_{\zeta \in \mathcal{Z}} c(x_i, f(\zeta)).$$

Wie üblich wird als Kostenfunktion die $L_2$-Norm benutzt. In der praktischen Umsetzung wird als Minimierungsverfahren ein niedrigdimensionaler, nichtlinearer Standard-Minimierer genutzt.

### 3.2.2 Adaption

Nachdem diese besten Approximationen berechnet wurden, kann nun die Minimierung über die Menge $\{\alpha_1, \ldots, \alpha_M\}$ beginnen. Analog zum ersten Schritt sehen wir jetzt die Variablen $\{\zeta_1, \ldots, \zeta_m\}$ als fest an. Dies ergibt auf natürliche Weise Sinn, da man im Algorithmus als vorherigen Schritt die Variablen $\zeta_i$ neu berechnet hat. Um nun (2.4) zu minimieren, betrachten wir den Fehlerterm

$$\frac{1}{m}\sum_{i=1}^{m}\left\|x_i - \sum_{j=1}^{M}\alpha_j k(z_j,\zeta_i)\right\|_2^2 + \frac{\lambda}{2}\sum_{i,j=1}^{M}\langle\alpha_i,\alpha_j\rangle k(z_i,z_j). \tag{3.2}$$

Diesen möchten wir nun nach den Vektoren $\alpha_k$ für $k = 1, \ldots, M$ minimieren. Dazu leiten wir (3.2) nach $\alpha_k$ ab und suchen einen KKT-Punkt. Wir erhalten damit

$$0 = \frac{\partial R_{\text{reg}}[f]}{\partial \alpha_k} = -\frac{2}{m}\sum_{i=1}^{m}\left(x_i - \sum_{j=1}^{M}\alpha_j K(\zeta_i,z_j)\right)K(\zeta_i,z_k) + \lambda\sum_{j=1}^{M}\alpha_j K(z_j,z_k), \tag{3.3}$$

wenn wir (3.2) nach der $k$-ten Variable $\alpha_k$ differenzieren. Das Ausmultiplizieren von (3.3) ergibt für $k = 1, \ldots, M$

$$\begin{aligned}
0 &= -\frac{2}{m}\sum_{i=1}^{m}\left(x_i - \sum_{j=1}^{M}\alpha_j K(\zeta_i,z_j)\right)K(\zeta_i,z_k) + \lambda\sum_{j=1}^{M}\alpha_j K(z_j,z_k)\\
&= -\frac{2}{m}\sum_{i=1}^{m}\left(x_i K(\zeta_i,z_k) - \sum_{j=1}^{M}\alpha_j K(\zeta_i,z_j)K(\zeta_i,z_k)\right) + \lambda\sum_{j=1}^{M}\alpha_j K(z_j,z_k)\\
&= -\frac{1}{m}\sum_{i=1}^{m}x_i K(\zeta_i,z_k) + \frac{1}{m}\sum_{j=1}^{M}\alpha_j\sum_{i=1}^{m}K(\zeta_i,z_j)K(\zeta_i,z_k) + \frac{\lambda}{2}\sum_{j=1}^{M}\alpha_j K(z_j,z_k).
\end{aligned}$$

Durch u erhält man

$$\sum_{j=1}^{M}\alpha_j\left(\sum_{i=1}^{m}K(\zeta_i,z_j)K(\zeta_i,z_k) + \frac{\lambda m}{2}K(z_j,z_k)\right) = \sum_{i=1}^{m}x_i K(\zeta_i,z_k).$$

Um erstens eine kompaktere Schreibweise dieser Ausdrücke zu erhalten und zweitens gleichzeitig über alle $k = 1, \ldots, M$ zu differenzieren, führen wir verschiedene Matrizen ein. Wir möchten die Daten, die Variablen $z_j$, die Vektoren $\alpha_j$ und die Kernauswertungen $K(\cdot,\cdot)$ zusammenfassen. Dazu bezeichnen wir mit $X$ die Matrix der Daten, wobei in jeder Zeile ein Datenpunkt als n-dimensionaler Vektor gespeichert ist, also

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} \in \mathbb{R}^{m\times n}.$$

Analog definieren wir

$$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_M \end{pmatrix} \in \mathbb{R}^{M \times n}, Z = \begin{pmatrix} z_1 \\ \vdots \\ z_M \end{pmatrix} \in \mathbb{R}^{M \times d}.$$

Des Weiteren definieren wir die Kernmatrizen $K_z$ und $K_\zeta$ durch $(K_z)_{i,j} := K(z_i, z_j)$ und $(K_\zeta)_{i,j} := K(\zeta_i, z_j)$ als eine $M \times M$ bzw. $m \times M$ dimensionale Matrix.

Führen wir nun die oben angestellten Überlegungen für alle $k = 1, \ldots, M$ zusammen, ergibt sich das Gleichungssystem

$$\left(\frac{\lambda m}{2} K_z + K_\zeta^\top K_\zeta\right) \alpha = K_\zeta^\top X,$$

welches nach $\alpha$ gelöst werden muss. Man beachte, dass die Lösung kein Vektor, sondern eine $M \times n$ Matrix ist.

### 3.2.3 Initialisierung

Wie oben schon erwähnt, benötigt man für die erste Iteration die Menge $\{\alpha_1, \ldots, \alpha_M\}$, um den ersten Projektionschritt durchzuführen. Die nun beschrieben Initialisierung entspricht dabei exakt der Initialisierung in [MS98].
Die Überlegung ist, als eine erste Annäherung die PCA zu nutzen und die ersten $d$ Hauptkomponenten $v_1, \ldots, v_d$ der Daten zu betrachten. Man möchte also zunächst die Vektoren $\alpha_i$ derart wählen, dass die Approximation an die gesuchte Funktion $f$ in die Richtung der ersten $d$ Hauptkomponenten zeigt. Wir betrachten dazu die Hauptkomponenten, die zu den $d$ größten Eigenwerten korrespondieren. In Formeln ausgedrückt wählt man die Menge $\alpha_{start}$ durch

$$\alpha_{start} = \underset{\alpha=(\alpha_1,\ldots,\alpha_M)\subset\mathcal{X}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} c\left(V(z_i - z_0), f_{(\alpha_1,\ldots,\alpha_M)}(z_i)\right) + \lambda Q[f].$$

In unserem Fall kann man dieses Problem ähnlich wie in 3.2.2 derart umformulieren, dass diese Minimierung zu einem Lösen des Gleichungssystem

$$\left(\frac{\lambda}{2}\mathbf{1} + K_z\right) \alpha = V(Z - Z_0)$$

transformiert werden kann. $K_z$ ist wieder die Kernmatrix, Z die Matrix der Hilfsvektoren, $Z_0$ die Matrix, welche aus $M$ Kopien des Mittelpunktes $z_0$ der Hilfsvektoren $z_i$ besteht und $V = (v_1, \ldots, v_d)$ die Matrix der Hauptkomponenten.

### 3.2.4 Abbruchkriterium

Nachdem nun das Grundprinzip dieses Verfahrens beschrieben wurde, stellt sich die Frage, wie lange diese abwechselnden Schritte durchgeführt werden sollen. Wir suchen also nach

einem geeigneten Abbruchkriteriums.
In der zugrunde liegenden Arbeit [SMSW01] gehen die Autoren auf dieses Problem nicht genauer ein. Um allerdings mit dem Verfahren praktisch zu arbeiten, ist ein solches Kriterium unabdingbar. Eine einfache, aber nicht notwendig gute Wahl ist es die Anzahl der Iterationen durch eine konstante $C \in \mathbb{N}$ zu beschränken, also $Anzahl\ Iterationen < C$. Dies kann dazu führen, dass man das (lokale) Optimum nicht erreicht bzw. die Lösung noch nicht exakt genug ist und man noch mehr Iterationen benötigt, um Konvergenz zu erreichen.
Ein anderer Ansatz folgt aus dem Verfahren, das drauf angelegt ist, den Regularisierungsfehler pro Iteration zu verringern. Die Überlegung geht dahin, den Regularisierungsfehler der $(i+1)$-ten Iteration mit dem der $i$-ten Iteration zu vergleichen. Anders ausgedrückt iteriert man solange, bis der Unterschied des Fehlers kleiner als ein fest gewähltes $\varepsilon > 0$ ist. Dies ist deshalb sinnvoll, weil man im Falle von Konvergenz[4] weiß, dass nicht mehr wesentlich besser approximiert werden kann.

[4] An dieser Stelle versteht man Konvergenz im Sinne des vorherigen Abbruchkriteriums.

# 4 Programmierung

Nachdem wir den PML-Algorithmus aus theoretischer Sicht beleuchtet haben, wollen wir uns mit der praktischen Umsetzung des Verfahrens auseinandersetzen. Dazu wird zum einen auf bestimmte programmiertechnische Details und zum anderen auf die konkrete Implementierung eingegangen.
Zunächst einmal haben wir in Algorithmus 1 ein grobes Verständnis für das Grundprinzip des Verfahrens erhalten. Beziehen wir nun unsere vorangegangenen Überlegungen aus Kapitel 3 mit ein, so erhalten wir folgenden Algorithmus:

**Algorithmus 2 :** PML-Algorithmus

**Eingabe** : $X = \{x_1, \dots, x_m\} \subseteq \mathbb{R}^n$, $Z = \{z_1, \dots, z_M\} \subseteq \mathbb{R}^d$, Fehlertoleranz $\varepsilon$
**Ausgabe** : $\alpha_{aktuell}$, Approximationen $Xeta$
berechne gute Startvektoren $\alpha_i$
**while** *Fehler* $> \varepsilon$ **do**

(1) $\forall i \in \{1, \dots, m\}$ :

$$\zeta_i = \underset{\zeta \in \mathcal{Z}}{\operatorname{argmin}} \, \|x_i - f(\zeta)\|_2^2 ;$$

(2) Löse das LGS

$$\left(\frac{\lambda m}{2} K_z + K_\zeta^\top K_\zeta\right) \alpha = K_\zeta^\top X$$

## 4.1 Aufbau der Implementierung

Unser Ziel im nächsten Abschnitt ist es, den Algorithmus 2 praktisch umzusetzen. Dazu wird die Programmiersprache Python -Version 3.5.1- genutzt. Sämtliche Programmiercodes und alle Graphiken, die im Folgenden beschrieben und gezeigt werden, sind mit Python erstellt und berechnet worden. Python wurde als Programmiersprache ausgewählt, da es dafür Python eine große Anzahl an leistungsfähigen Bibliotheken und Paketen gibt.
Grundlage der Implementierung des PML-Verfahrens ist das Paket NumPy[5] (Numerical Python), welches viele mathematische und numerische Routinen enthält. Wir nutzen NumPy

[5] Eine übliche Abkürzung von NumPy, welche hauptsächlich in der Programmierung benutzt wird, ist np

hauptsächlich zum Speichern der Datensätze als mehrdimensionale Arrays und zum Gebrauch einiger Routinen der linearen Algebra.
In Kapitel 2 haben wir den Zusammenhang zwischen der gesuchten Funktion $f$ und einer Kernfunktion hergeleitet. In der Umsetzung des Verfahrens nutzen wir den Gaußschen RBF-Kern[6]

$$K(x,y) := \exp\left(-\frac{\|x-y\|^2}{2\sigma}\right).$$

Umgesetzt wurde diese Kernauswertung durch die in NumPy implementierte Exponential- und Normfunktion. Wichtig dabei ist, dass die implementierte Kernfunktion 4.1 als Eingabe nur einen Vektor bekommt, da $K$ ein RBF-Kern ist und daher umgeschrieben werden kann als $K(x,y) = \Phi(\|x-y\|)$, für eine Funktion $\Phi : [0,\infty] \to \mathbb{R}$ [VTS04; Gar16].

```
def gauss_kern(x,sig): return np.exp(-((np.linalg.norm(x))**2)/(2*sig**2))
```

Programmcode 4.1: Gaußscher RBF-Kern

Zusammen mit dem Gaußschen RBF-Kern kann nun die Berechnung der Kernmatrizen $K_z$ und $K_\zeta$ implementiert werden.

```
def erstelle_kernmatrix_z(Stuetzvektor,sig,M):
    Kernmat_z=np.zeros((M,M))
    for i in range(0,M):
        for j in range(0,M):
            tmp=Stuetzvektor[i][0]-Stuetzvektor[j][0]
            Kernmat_z[i][j]=gauss_kern(tmp,sig)
    return Kernmat_z

def erstelle_kernmatrix_xeta(Xeta,Stuetzvektor,sig,m,M):
    Kernmat_xeta=np.zeros((m,M))
    for i in range(0,m):
        for j in range(0,M):
            tmp=Xeta[i]-Stuetzvektor[j]
            Kernmat_xeta[i][j]=gauss_kern(tmp,sig)
    return Kernmat_xeta
```

Programmcode 4.2: Kernmatrizen

Da sich im PML-Algorithmus die Menge der Stützvektoren nicht ändert, bleibt die Kernmatrix $K_z$ konstant und kann daher vor Beginn des Algorithmus einmal berechnet werden und anschließend als Argument übergeben werden. Dies gilt nicht für $K_\zeta$, da sich die Menge der Approximationen in jedem Projektionsschritt ändert.

[6] RBF ist eine übliche Abkürzung für 'radial basis function'

```
def alogorithmus(Data,Stuetzvektor,Alphas,Kernmat_z,lambd,sig,n,d,m,M):
    reg_error_old=0
    iterator=1
    Xeta=np.zeros((m,d))

    while True :

        #Projektion

        #Adaption

        reg_error_new=reguar_error(Data,Alphas,Kernmat_xeta,Kernmat_z,lambd,m
            ,M)

        if np.abs(reg_error_old-reg_error_new)< 1e-6:
            return Alphas,Xeta
        if iterator >= 500:

            print('ACHTUNG ACHTUNG: Zu viele Iterationen!!!')
            print('Dies kann zu unbrauchbaren Resultaten fuehren')

            return Alphas,Xeta

        reg_error_old=reg_error_new
        iterator+=1
```

Programmcode 4.3: Algorithmus in Python

Als Grundgerüst der Implementierung ist eine *do-while*-Schleife implementiert worden, da man nach der Initialisierung eine komplette Iteration durchführen möchte, um im Anschluss daran die Abbruchbedingung zu überprüfen. In Kapitel 3.2.4 wurde eine Möglichkeit beschrieben, wie man die Anzahl der Iterationen des PML-Algorithmus steuern kann. Dabei verstehen wir unter einer Iteration wieder die Durchführung eines Projektions- und eines Adaptionsschrittes. Die Idee dieser Abbruchbedingung war der Vergleich zweier aufeinander folgender Regularisierungsfehler. Dazu benötigt man eine Funktion, die diesen Fehler berechnet.

```
def regular_error(Data,Alphas,Kernmat_xeta,Kernmat_z,lambd,m,M):
    error=0
    for i in range(0,m):
        tmp=np.copy(Data[i,:])
        for j in range(0,M):
            tmp-=Kernmat_xeta[i][j]*Alphas[j,:]
        error+=(np.linalg.norm(tmp))**2
    for i in range(0,M):
        for j in range(0,M):
            error+=(lambd/2)*(Kernmat_z[i][j])*(Alphas[i,:].dot(Alphas[j,:]))
    return error
```

Programmcode 4.4: Fehlerberechnung

Analog werden der empirische Fehler $R_{\text{emp}}[f]$ und der Regularisierungsterm $Q[f]$ berechnet, indem man den jeweiligen Term aus 4.4 extrahiert. Um zu verhindern, dass der Algorithmus im Falle der Nicht-Konvergenz in eine Endlosschleife gerät, wird dieser nach einer gewissen Anzahl an Iterationen vorzeitig beendet und eine Fehlerwarnung ausgegeben, mit der Aussage, dass die berechneten Ergebnisse eventuell unbrauchbar sein könnten.
Es sei erwähnt, dass der Algorithmus selbst in Python als eine Funktion definiert ist und über eine Main-Funktion aufgerufen wird. Dies bietet die Möglichkeit, den Programmcode universell zu gestalten und ausschließlich die Gegebenheiten und Parameter in der Main-Funktion der Aufgabe anzupassen ohne den Programmcode des Algorithmus zu manipulieren.
Nachdem wir uns der praktischen Umsetzung des Grundschemas des Algorithmus gewidmet haben, wollen wir im Folgenden die Implementierung der Hauptbestandteile des PML-Algorithmus, der Adaption und der Projektion, genauer betrachten.

## 4.2 Umsetzung der Adaption

Zunächst beschreiben wir die Umsetzung der Adaption genauer. Aus 3.2.2 ist bekannt, dass das Gleichungssystem

$$\left(\frac{\lambda m}{2} K_z + K_\zeta^\top K_\zeta\right) \alpha = K_\zeta^\top X \tag{4.1}$$

gelöst werden muss. Wie im vorherigen Abschnitt erwähnt ist dabei die Matrix $K_z$ konstant und wurde dem Algorithmus als Argument übergeben. Die Matrix $K_\zeta$ hingegen hängt von der vorherigen Projektion ab. Daher wird zunächst die Matrix *Kernmat_xeta* mit der vorher definierten Routine *erstelle_kernmatrix_xeta* berechnet. Um das Gleichungssystem 4.1 zu lösen, werden zunächst zwei Matrizen $T_1$ und $T_2$ erstellt und anschließend das Gleichungssystem

$$T_1 \alpha = T_2$$

mit dem *solve*-Befehl des NumPy-Pakets *linalg* gelöst. Dieser Befehl bietet den Vorteil, dass auf schnelle und effiziente Implementierungen des Lösers in C zurückgegriffen wird [Oli06; Num16].

```
Kernmat_xeta=erstelle_kernmatrix_xeta(Xeta,Stuetzvektor,sig,m,M)

T1=(lambd*m)/2 * Kernmat_z + (Kernmat_xeta.T).dot(Kernmat_xeta)
T2=(Kernmat_xeta.T).dot(Data)

Alphas=np.linalg.solve(T1,T2)
```

Programmcode 4.5: Adaption

## 4.3 Umsetzung der Projektion

Im Gegensatz zum Adaptionsschritt, der in der Implementierung klar durch die Theorie vorgegeben ist, lässt der Projektionschritt viel mehr Freiraum in der Implementierung zu. In [SMSW01] wird kaum bis gar nicht auf weitere Details diesbezüglich eingegangen. Einzig der Gebrauch eines nichtlinearen niedrigdimensionalen Standardminimierers zur Umsetzung der Minimierung wird empfohlen und auf das Buch [PTVF92] verwiesen, welches einen ausführlichen Überblick über grundlegende numerische Verfahren gibt.
Um mit Python die Minima $\zeta_i$ bezüglich der Datenpunkte zu berechnen werden wir das SciPy-Pakete (Scientific Python) nutzen. Wir können damit auf sehr leistungsfähige und effiziente Implementierungen von bekannten numerischen Verfahren zugreifen. Dabei interessieren wir uns hauptsächlich für die implementierten Minimierungsverfahren aus dem Bereich der Optimierer. Zur Verfügung stehen dort elf verschiedene Verfahren wie das CG- oder BFGS-Verfahren [JOP+01; Sci].
Bevor wir den genutzten Minimierer genauer beschreiben, definieren wir zunächst eine Funktion, welche nach $\zeta_i$ minimiert werden kann. Dazu betrachten wir erneuert die Kostenfunktion $c(x_i, f(\zeta))$ und formulieren diesen Ausdruck unter Zuhilfenahme von 2.8 ein wenig um:

$$c(x_i, f(\zeta)) = \|x_i - f(\zeta)\|_2^2 = \left\| x_i - \sum_{j=1}^{M} \alpha_j K(z_j, \zeta) \right\|_2^2 .$$

Diese Darstellung lässt nun eine einfache Implementierung der zu minimierenden Funktion zu. Dazu definiert man zunächst eine Funktionsauswertung *evaluation_f_Alphas* 4.6 der Funktion $f$ als Summe von Kernfunktionen. Diese wird sowohl zur Berechnung der zu minimierenden Funktion als auch bei der Erstellung diverser Graphiken benötigt.

```
def evaluation_f_Alphas(x,Stuetzvektor,Alphas,sig,n,M):
    result=np.zeros(n)
    for i in range(0,M):
        result+=gauss_kern(Stuetzvektor[i]-x,sig)*Alphas[i,:]
    return result
```

Programmcode 4.6: Funktionsauswertung

Diese Funktionsauswertung nutzen wir, um die zu minimierende Funktion *funct* zu definieren, siehe Programmcode 4.7. Des Weiteren verwenden wir aus dem NumPy-Paket die Norm-Subroutine aus der linearen Algebra. Da wir für einen Datenpunkt eine Minimierung bezüglich des $d$-dimensionalen Vektors $\zeta$ durchführen, wird in der Implementierung über die $d$-dimensionale Variable *xeta* minimiert. Die restlichen Parameter werden in Form von Matrizen oder Skalaren übergeben. Man bemerke noch, dass die Norm-Funktion aus NumPy die gewöhnliche euklidische Norm berechnet. Dies bedeutet insbesondere, dass dort die Wurzel gezogen wird. Aus diesem Grund wurde in der Implementierung die Norm quadriert.

```
def funct(xeta,Xi,Alphas,Stuetzvektor,sig,n,d,m,M):
    return np.linalg.norm(Xi-evaluation_f_Alphas(xeta,Stuetzvektor,Alphas,sig
        ,n,M))**2
```

Programmcode 4.7: Minimierungsfunktion

Insgesamt haben wir damit ein unrestringiertes, nichtlineares Optimierungsproblem formuliert. Um dieses Problem zu lösen, wollen wir das Newton-Verfahren nutzen [GK99; UU12]. Angesichts der Komplexität der Minimierungsfunktion wollen wir auf die Berechnung der Hessematrix möglichst verzichten und dies durch symmetrisch positiv definite Matrizen approximieren. Dieser Ansatz führt zu den Quasi-Newton-Verfahren [GK99; UU12]. Aus dieser Klasse der Quasi-Newton-Verfahren werden wir das BFGS-Verfahren als Minimierungsverfahren nutzen.
Genauer gesagt werden wir eine Variante des BFGS-Verfahrens, das L-BFGS-B-Verfahren, nutzen, welches häufig im Bereich des Maschinellen Lernens angewandt wird [Hag14]. Dabei wird der benutzte Speicher im Rechner begrenzt, indem nur eine begrenzte Anzahl an Vektoren zur Approximation der Hessematrix genutzt werden. Weitere Details sind in [Ska10; BNSNB94] zu finden. Auch werden dabei die Wertebereiche der Variablen sowohl von unten als auch von oben begrenzt [ZBLN97]. Diese Einschränkung wird benötigt, da wir nicht immer über den kompletten $\mathbb{R}^d$ minimieren möchten, sondern auch $d$-dimensionalen Teilräume betrachten.
Bei der Implementierung in Python nutzen wir den SciPy-Minimierer *minimize* mit der Methode *L-BFGS-B*. Eine Ausnahme bildet dabei der eindimensionale Fall des Raumes $\mathcal{D}$, da dort der eindimensionale Minimierer *minimize_scalar* mit der Methode *bounded* verwendet wird [7].

```
for i in range(0,m):
    Xi=np.copy(Data[i,:])

    #Im Falle von d>1 muss der Startwert-Vektor und die Grenzen auf d-
        Dimensionen erweitert werden. Hier ist ein Beispiel der Dimensiona d
        =2 angegeben.

    if d==1:
        solution = minimize_scalar(funct, bounds=[-1,1], args=(Xi, Alphas,
            Stuetzvektor,sig,n,d,m,M), method='bounded')
    else:
        solution = minimize(funct,(0.01,0.01), args=(Xi, Alphas, Stuetzvektor
            ,sig,n,d,m,M), method='L-BFGS-B' , bounds=((-1,1),(-1,1)))

    Xeta[i,:]=solution.x
```

Programmcode 4.8: Projektion

[7] Dies ist eigentlich keine Ausnahme, da man auch mit dem normalen *minimize* den eindimensionalen Fall lösen kann. Es hat sich in der Praxis gezeigt, dass der Unterschied zwischen den Minimieren im eindimensionalen Fall im Bereich von $10^{-4}$ bis $10^{-5}$ gelegen hat.

Die nötigen Parameter und Datenmengen werden als Argumente zusammengefasst und als Menge *args* dem Minimierer übergeben. Zudem müssen noch die Grenzen der einzelnen Variablen und deren Startwerte angegeben werden. Dabei werden in den Experimenten in Kapitel 5 zwei verschiedene Möglichkeiten eines Startwertes genutzt. Eine Möglichkeit ist eine Wahl aus einer Umgebung der 0. Eine zweite Möglichkeit ist, die in der vorherigen Iteration berechnete niedrigdimensionale Darstellung der Datenpunkte als geeignete Startwerte zu nutzten.
Die SciPy-Minimierer geben viele Informationen nach dessen Ausführung zurück. Um nun den Lösungsvektor aus diesen Informationen zu extrahieren, nutzt man den Punktoperator, in unserer Implementierung also *solution.x*. Da wir für jeden Datenpunkt einen solchen Lösungsvektor erhalten, speichern wir diese für den $i$-ten Datenpunkt in der $i$-ten Zeile der Matrix *Xeta* ab. Die berechneten Vektoren bilden nun eine niedrigdimensionale Darstellung der höherdimensionalen Daten.
Um zum einen das Verfahren immer auf gleiche Datenmengen zu testen und zum anderen überflüssige Berechnungen zu vermeiden, werden sowohl die synthetischen Daten und die Stützvektoren, als auch die Menge der initialisierten $\alpha_i$ extern berechnet, abgespeichert und im Verfahren als Datei eingebunden.

# 5 Resultate

In Kapitel 3 haben wir ein Verfahren kennengelernt, welches eine niedrigdimensionale Darstellung hochdimmensionaler Daten berechnet und gleichzeitig eine Funktion $f : \mathcal{Z} \to \mathcal{X}$ liefert. Wir haben gesehen, dass man den PML-Algorithmus 2 mit den Methoden der Kern-basierten Funktionsdarstellung und einem Quantisierungsfehler-Funktional herleiten kann. Im Anschluss daran wurde in Kapitel 4 die Implementierung des PML-Verfahren erläutert, mit der das PML-Verfahren getestet werden kann.
Ferner wurde gezeigt, dass die Resultate des PML-Verfahrens von vielen Faktoren beeinflusst werden. Unter anderem sind die Resultate des PML-Verfahren von

- der Initialisierung der Vektoren $\{\alpha_1, \ldots, \alpha_M\}$
- der Parameterwahl des Regularisierungsparameters $\lambda$ und der Gaußbreite $\sigma$
- der Anzahl an Basisfunktionen $M$ zur Funktionsdarstellung
- der Wahl des Optimierungsverfahren
- der Initialisierung des Optimierungsverfahren
- und dem Abbruchkriterium

anhängig. Diese Überlegungen führen zu der Frage, welche Konfiguration des PML-Verfahrens für die gegebene Situation und die jeweiligen Daten gewählt werden sollte um sinnvolle Resultate zu erzielen. Insbesondere soll der Einfluss einiger der eben genannten Faktoren auf das PML-Verfahren untersucht werden. Dazu werden in diesem Kapitel vier Experimente beschrieben, in denen der PML-Algorithmus auf verschiedene Daten angewendet wird. Im Anschluss daran werden die Ergebnisse dieser Experimente unter verschiedenen Gesichtspunkten analysiert. Ein weiteres Ziel dieses Kapitel ist die Bestätigung der in [SMSW01] gezeigten Resultate.

## 5.1 Experimente und deren Datensätze

Um die Wirksamkeit des PML-Algorithmus zu belegen wurde in [SMSW01], in drei verschiedenen Experimenten, mit verschieden Daten das Verfahren getestet. In diesem Abschnitt werden diese Experimente beschrieben. Dabei wird versucht, die Experimente beziehungsweise die betrachteten Datensätze bestmöglich zu imitieren, da in [SMSW01] nicht eindeutig beschrieben wird wie die Daten entstanden sind.

### 5.1.1 Polynomrekonstruktion

Wir möchten das Verfahren für das Problem der zweidimensionalen Polynomrekonstruktion nutzen. Dazu betrachten wir den Graph eines Polynoms $p$ als eine eindimensionale Mannigfaltigkeit im zweidimensionalen Raum und möchten diesen Graphen auf Grundlage der Daten, welche entlang des Graphen liegen, rekonstruieren.
Um nun diese Daten zu erzeugen, betrachten wir das kubische Polynom $p(x) = 3.2x^3 - 2.2x$ auf dem Intervall $[-1, 1]$. Durch diese Wahl erhalten wir zwei Ausschläge (Hoch- und Tiefpunkt) in unserem Datensatz, welche optimalerweise durch den Algorithmus erkannt werden. Des Weiteren definieren wir, genau wie in [SMSW01], den Zustandsraum $\mathcal{Z} = [-1, 1]$. Selbstverständlich tauchen solche Daten in der Realität nicht als eine Mannigfaltigkeit auf. Daher wird auf die synthetischen Daten noch ein Rauschen aufaddiert.
Eine Abschätzung aus den dargelegten Resultaten in [SMSW01] ergab eine Anzahl von 70 Datenpunkten, auf dessen Grundlage ein Datensatz aus $m = 70$ Datenpunkten erzeugt wurde. Analog wurde ein gleicher Datensatz mit $m = 200$ Daten erhoben. Des Weiteren wurden zwei weitere Datensätze für je 70 und 200 Daten erzeugt, indem jeweils die y-Koordinate auf ein Drittel gestaucht wurde (s. im Folgenden 5.2). Die Anzahl an Kernfunktionen $M$ im Algorithmus wird auf 10 gesetzt. Die Menge $Z \subset [-1, 1]$ wurde durch ein eindimensionales Gitter mit 10 gleichverteilten Gitterpunkten erzeugt.

### 5.1.2 Oberflächenrekonstruktion

Im vorherigen Experiment haben wir eine eindimensionale Mannigfaltigkeit im zweidimensionalen Raum betrachtet. Nun wollen wir diese Situation um eine Dimension erhöhen, dass heißt, wir möchten eine zweidimensionale Mannigfaltigkeit im dreidimensionalen Raum rekonstruieren. Dazu betrachten wir den Graph der Funktion $h(x, y) = x^3 - x - \frac{1}{2}y^2$ auf dem Quadrat $[-1, 1]^2$ und generieren daraus 400 Datenpunkte. Auf diese Datenpunkte wird wieder ein Rauschen aufaddiert. In den Experimenten wurden jeweils 36 Kernfunktionen, $\mathcal{Z} = [-1, 1]^2$ und auf $\mathcal{Z}$ ein zweidimensionales Gitter mit 36 gleichverteilten Gitterpunkten genutzt.
Ziel dieses Experiments ist die Rekonstruktion der Funktion $h$. Dabei soll die Struktur des Graphen erkennbar sein, dass heißt in Richtung der x-Achse sollte ein wellenförmiger Verlauf und in Richtung der y-Achse ein parabelförmiger Verlauf erkennbar sein.

### 5.1.3 Swiss Roll

Ähnlich wie in der zuvor beschriebenen Rekonstruktion des zweidimensionalen Polynoms möchten wir die in Kapitel 1 erwähnte Swiss Roll als eine spiralförmige Oberfläche ansehen und das PML-Verfahren darauf anwenden. Diese Oberfläche wird dabei durch den Graphen der Funktion $\varphi(x, y) = (x\cos(x), y, x\sin(x))$ erzeugt. Der Swiss Roll Datensatz enthält nun 500 Datenpunkte, welche auf dieser Oberfläche liegen. Um den PML-Algorithmus auf diesen Datensatz anzuwenden verwenden wir 36 Kernfunktionen auf dem Zustandsraum $\mathcal{Z} = [-1, 1]^2$ und ein gleichverteiltes Gitter mit 36 Gitterpunkten auf dem Raum $[-1, 1]^2$.
Das Ziel dieses Experiments sollte es sein, diese spiralförmige Struktur in der Rekonstruktion der Daten wiederzuerkennen.

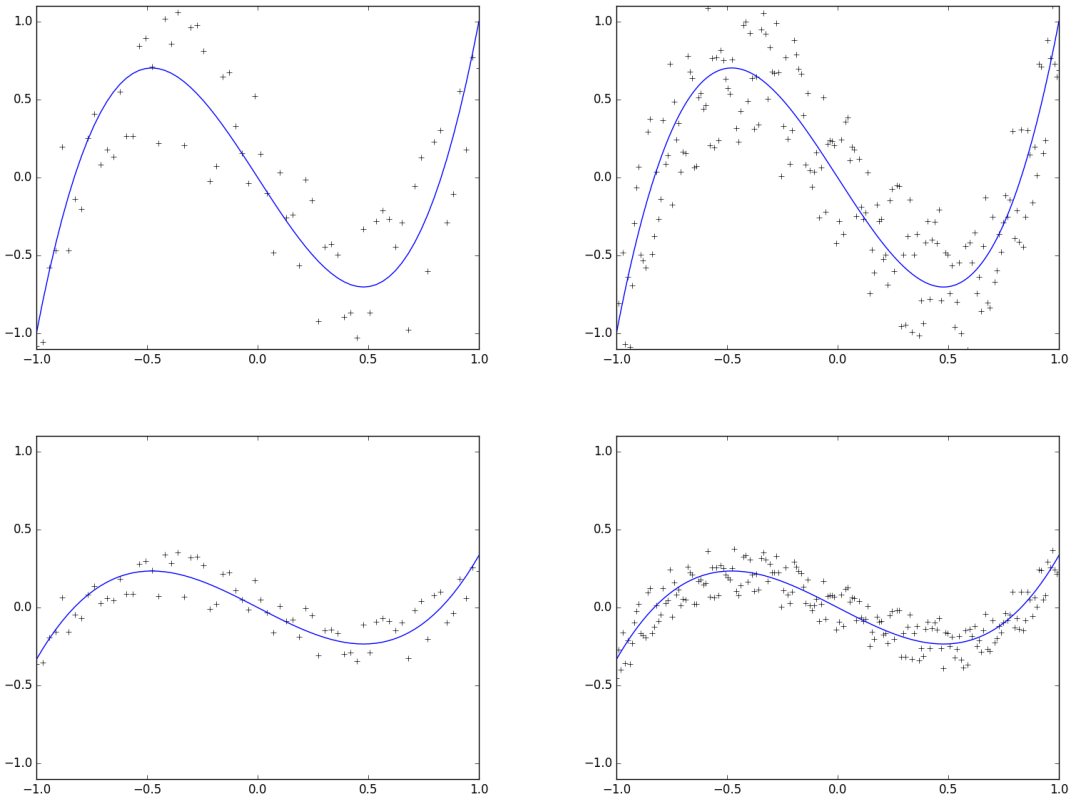


Abbildung 5.1: Graphische Darstellung der Polynomdaten für 70 (links) und 200 (rechts) Datenpunkte.

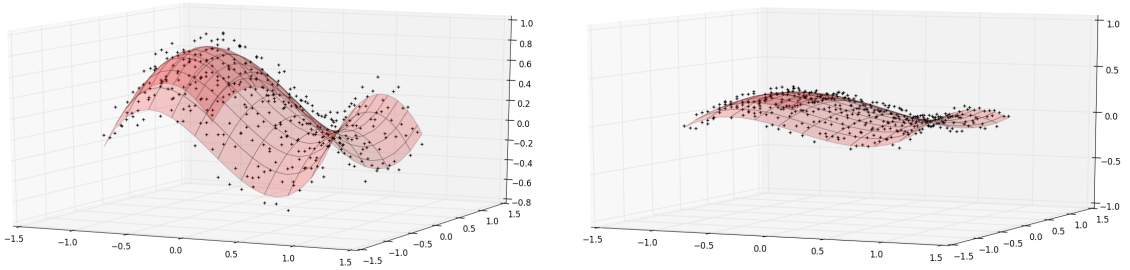


Abbildung 5.2: Graphische Darstellung der Oberflächendaten für 400 Datenpunkte. Ungestaucht (links), gestaucht (rechts)

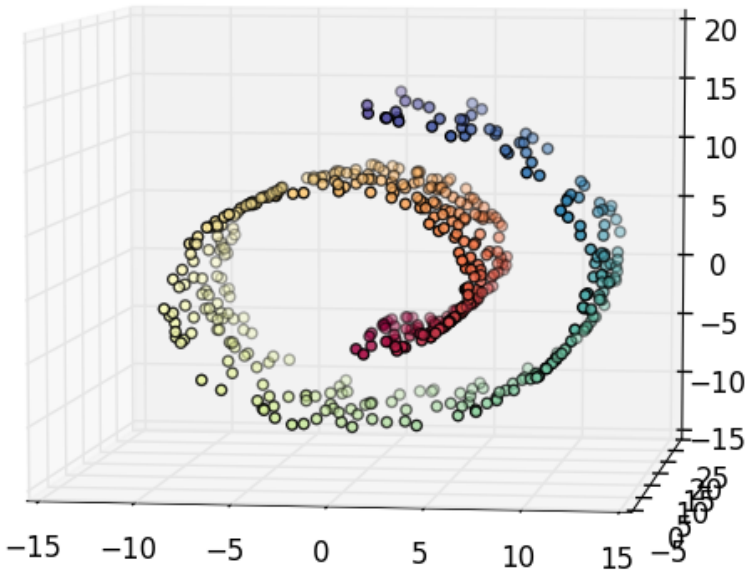



Abbildung 5.3: Graphische Darstellung der Swiss Roll mit 500 Datenpunkte.

### 5.1.4 Der Öl-Fluss-Datensatz

Um zu zeigen, dass das PML-Verfahren nicht nur für Instanzen mit geringer Dimensionalität sinnvolle Ergebnisse liefert, wird das PML-Verfahren auch auf den so genannten Öl-Fluss-Datensatz angewandt. Die Idee das PML-Verfahren mit diesen Datensatz zu testen, stammt aus der Nähe des PML-Verfahrens zu der GTM. In [MS98] wurde der GTM-Algorithmus auch auf den Öl-Fluss-Datensatz angewandt und liefert nützliche Ergebnisse.
Der Öl-Fluss-Datensatz enthält 1000 Datenpunkte aus einem 12-dimensionalen Raum und beschreibt den Fluss einer Mixtur aus Öl, Wasser und Gas in einer Pipeline [LYL14; Law16]. Dabei können diese Daten, genauer gesagt der Fluss in der Pipeline, je einen von drei Zuständen (horizontal geschichteter Fluss, ringförmig verschachtelter Fluss oder gleichmäßig gemischter Fluss) einnehmen. Daraus folgt, dass die Daten in drei Klassen eingeteilt werden können. Jede Klasse wiederum besitzt zwei Freiheitsgrade (Wasserdruck und Öldruck), so dass jede Klasse eine zweidimensionale Darstellung besitzt [Law16].
Aus diesen Überlegungen folgt, dass man die 12-dimensionalen Daten mit Hilfe des PML-Verfahren in zwei Dimensionen darstellen und dort klassifizieren kann. Falls der PML-Algorithmus wie gewünscht arbeitet, sollten die drei verschiedenen Klassen der Daten klar im zweidimensionalen Raum voneinander getrennt dargestellt sein.

## 5.2 Initialisierung

In Kapitel 3.2.3 haben wir gesehen, dass eine Initialisierung der Vektoren$\{\alpha_1, \ldots, \alpha_M\}$ berechnet werden muss. Betrachten wir dazu die Polynomdaten und die in Kapitel 3.2.3 be-

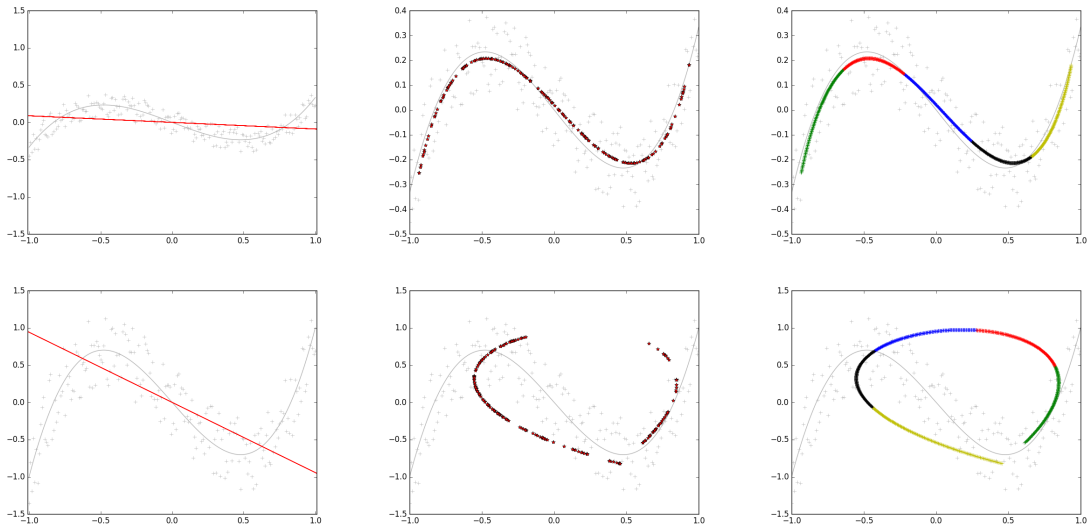


Abbildung 5.4: Darstellung der Initialisierung des PML-Algorithmus (links), der Approximationen der Datenpunkte (Mitte) und der Verlauf der rekonstruierten Funktion auf dem Intervall $[-1, 1]$ für die gestauchten Daten (obere Reihe) und die ungestauchten Daten (untere Reihe). Das PML-Verfahren wurde dabei mit den Parametern $\lambda = 0.001$ und $\sigma = 1$ ausgeführt.

schriebene Initialisierung entlang der ersten Hauptkomponente. In Abbildung 5.4 kann man erkennen, dass die Initialisierung (rote Linie) im ungestauchten Fall deutlich steiler als die Initialisierung im gestauchten Fall ist. Dies führt im PML-Algorithmus zu stark unterschiedlichen Resultaten.
Lässt man nun den Algorithmus (bei gleicher Parameterwahl) mit den beiden Datensätzen laufen, so erhält man in beiden Fällen Konvergenz. Allerdings unterscheiden sind die Resultate deutlich voneinander. Wie man in Abbildung 5.4 sieht, gelingt sowohl die Approximation der Daten als auch der Erhalt einer glatten Funktion $f$ bei einer flacheren Initialisierung der Vektoren $\{\alpha_1, \ldots, \alpha_M\}$. Hingegen sind die Resultate bei einer zu steilen Initialisierung nicht brauchbar. In Abbildung 5.5 ist eine Initialisierung entlang zweier Hauptkomponenten für verschiedene Regularisierungsparameter dargestellt. Zunächst kann man erkennen, das die Initialisierung der dreidimensionalen Vektoren $\{\alpha_1, \ldots, \alpha_M\}$ dem entspricht, was in Abschnitt 3.2.3 beschrieben wurde. Betrachtet man die Auswertung der berechneten Funktion $f$ nach der Initialisierung auf dem Raum $[-1, 1]^2$, so ergibt sich eine Ebene, die entlang der ersten und zweiten Hauptkomponenten der Datenmenge verläuft. Ein weiterer Aspekt, der in Abbildung 5.5 sichtbar wird, ist der Einfluss des Regularisierungsparameters auf die Initialisierung. Betrachtet man die berechneten Ebenen entlang der z-Achse, so kann man für verschiedene Werte des Regularisierungsparameters verschieden große Ebenen erkennen. Je kleiner dabei der Regularisierungsparameter gewählt wird, umso mehr tendiert die Form dieser Ebene von einem Oval hin zu einem Rechteck. Die Tatsache, dass für unterschiedliche Regularisierungsparameters unterschiedliche Ergebnisse in der Initialisierung erzielt werden, wirft die Frage auf, in wie weit dieser Parameter auch die Ergebnisse des gesamten Algorithmus beeinflusst. Dieser Einfluss des Regularisierungsparameters soll im nächsten Abschnitt

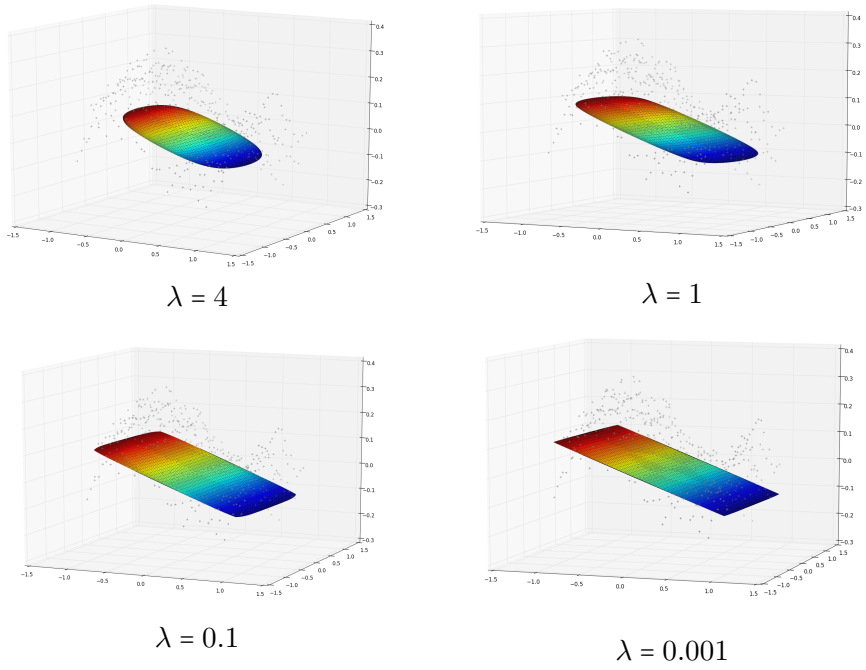



Abbildung 5.5: In dieser Abbildung ist die Auswertung der berechneten Funktion auf dem Raum $[-1,1]^2$ für verschiedene Regularisierungsparameter nach der Initialisierung dargestellt.

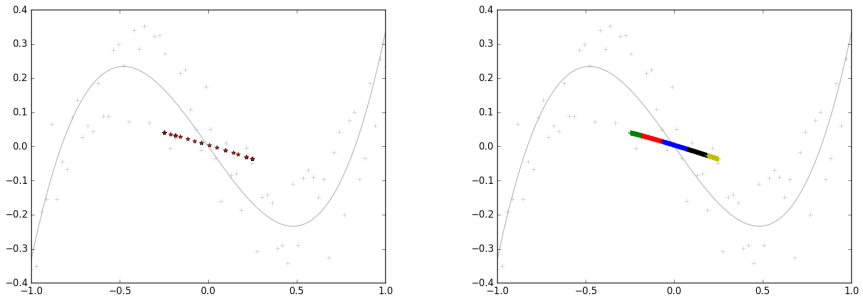
genauer untersucht werden.

## 5.3 Einfluss des Regularisierungsparameter

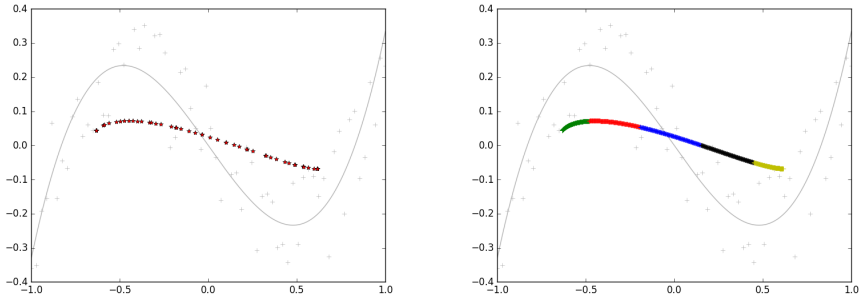
In Abschnitt 2.2 haben wir den Regularisierungsparameter $\lambda$ definiert. Vereinfacht ausgedrückt gibt dieser an, wie stark die Funktion $f$ glatter gemacht werden muss und hat damit einen großen Einfluss auf die berechneten Resultate. Es wird daher untersucht, wie genau dieser Parameter Einfluss auf die Resultate nimmt und welche Wahl dieses Parameters akzeptable Ergebnisse liefert.

### 5.3.1 Beispiel Polynomrekonstuktion

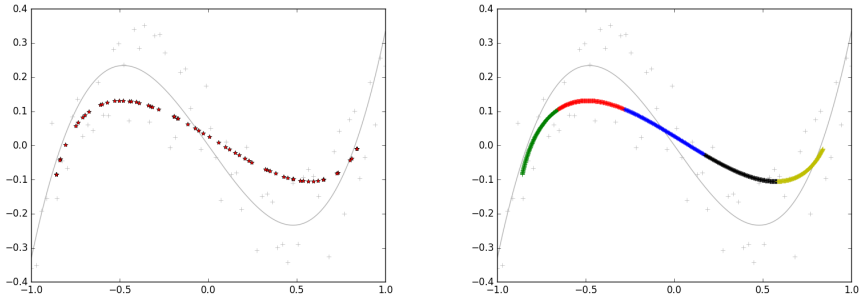
Man betrachte zunächst in Abbildung 5.6 die berechneten Approximationen und den Verlauf der Polynomrekonstruktion für verschiedene Regularisierungsparameter. Man kann erkennen, dass für $\lambda = 1$ sowohl die berechneten Approximationen der Daten als auch die Funktion, ausgewertet auf dem Intervall $[-1,1]$, nicht ausreichend ist, um die Ursprungsfunktion darzustellen. Je kleiner allerdings den Regularisierungsparameter gewählt wird, umso besser werden die berechneten Approximationen und die gesuchte Funktion $f$. Dabei steigt die Genauigkeit mit negativen Zehnerpotenz an. Ein Regularisierungsparameter um 0.001 liefert die gewünschten Ergebnisse, also die Rekonstruktion des Polynoms $p$ auf dem Intervall



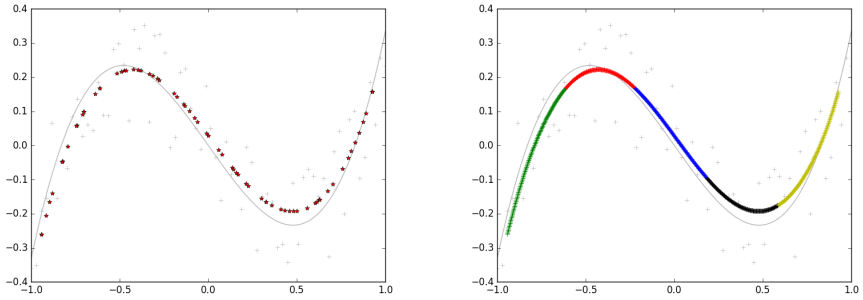
$\lambda = 1$



$\lambda = 0.1$



$\lambda = 0.01$



$\lambda = 0.001$

Abbildung 5.6: Darstellung der Rekonstruktion der Datenmengen für $\lambda = 1, 0.1, 0.01, 0.001$. In der linken Spalte sind durch rote Sternchen die Approximationen der Datenpunkte (graue Kreuze im Hintergrund) gekennzeichnet, in der rechten Spalte ist der Verlauf der rekonstruierten Funktion $f$ auf dem Intervall $[-1, 1]$ zu sehen. Hierbei wurden die Parameter $m = 70$, $\sigma = 1$ gewählt und der Algorithmus angehalten, falls die Fehlerdifferenz kleiner als $10^{-6}$ ist.

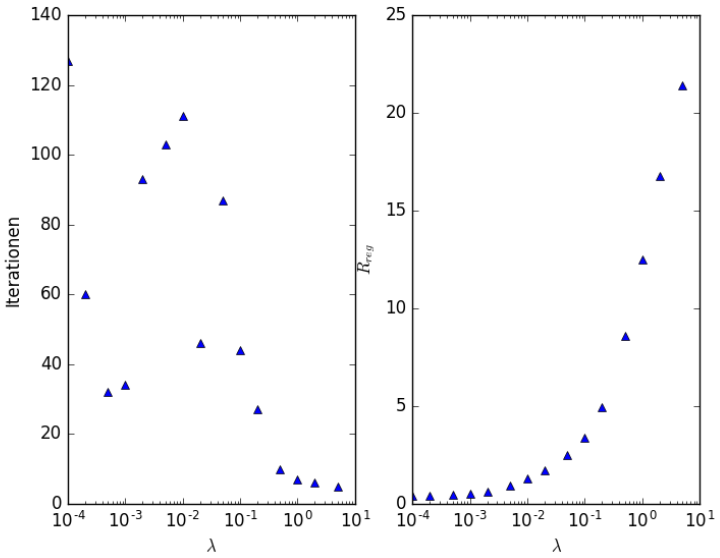
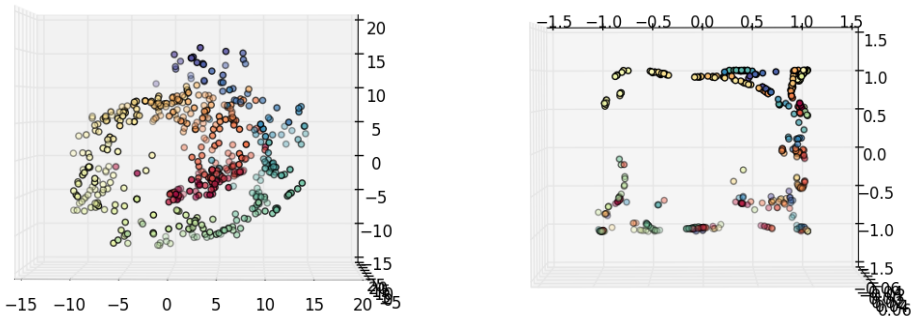



Abbildung 5.7: Verhältnis $\lambda$ zu der Anzahl der Iterationen und dem regularisiertem Fehler.

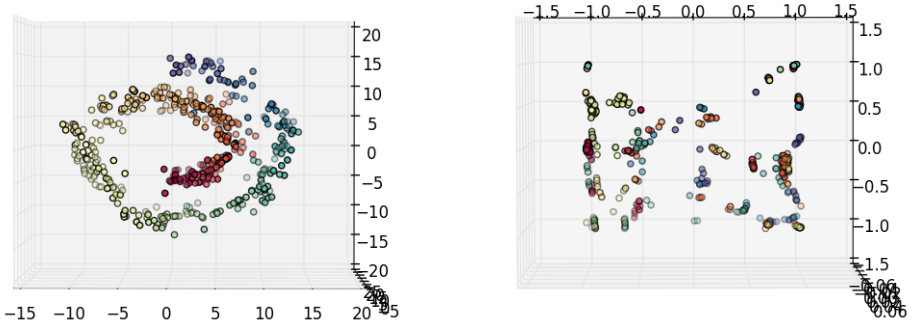
$[-1, 1]$ und eine gute Approximation der Daten.
Allerdings folgt auf eine noch kleinere Wahl des Regularisierungsparameters nicht zwingend eine höhere Genauigkeit. Betrachtet man die Abbildung 5.7, so kann man erkennen, dass der Regularisierungsfehler für kleinere Regularisirungsparameter kleiner wird. Allerdings kann man für Regularisierungsparameter kleiner als $10^{-3}$ keine signifikanten Fehlerverbesserungen erreichen. Beispielsweise liegt der Regularisierungsfehler für $\lambda = 10^{-3}$ ungefähr bei 0.51 und für $\lambda = 10^{-6}$ ungefähr bei 0.47. Des Weiteren steigt für kleinere Regularisierungsparamter die Anzahl an Iterationen stark an, so dass es für kleinere Regularisierungsparamter deutlich länger dauern kann, bis Konvergenz erreicht wird. Daher ist ein zu kleiner Regularisierungsparametrers $\lambda$ nicht zwingend eine gute Wahl, um sinnvolle Resultate am Beispiel der Polynomrekonstruktion zu erzielen.

### 5.3.2 Beispiel Swiss Roll

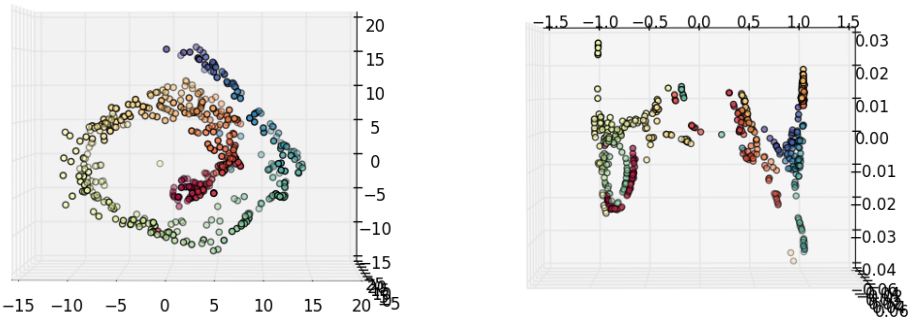
In dem Experiment zur Polynomrekonstruktion konnte gezeigt werden, dass eine geeignete Parameterwahl für $\lambda$ um $10^{-3}$ liegt und für diese Wahl Konvergenz erreicht werden kann. Nutzt man das PML-Vefahren für den Swiss Roll Datensatz, so zeigen sich gegenteilige Resultate. Betrachtet man in Abbildung 5.8 sowohl die Approximationen an die Datenpunkte als auch die zweidimensionale Darstellung der Swiss Roll für die Regularisierungsparameter $4, 1, 0.01, 0.0001$ so kann man erkennen, dass für $\lambda = 0.0001$ die Struktur der Swiss Roll nicht wieder gegeben werden kann und die Datenpunkte auf getrennten Ebenen und Geraden liegen. Ein ähnliches Resultat erhält man für $\lambda = 0.1$. Hierbei kann man zwar die spiralförmige Struktur etwas deutlicher erkennen, aber betrachtet man die dreidimensionale Darstellung der approximierten Datenpunkte in einem anderen Blickwinkel, so können wieder Ebenen



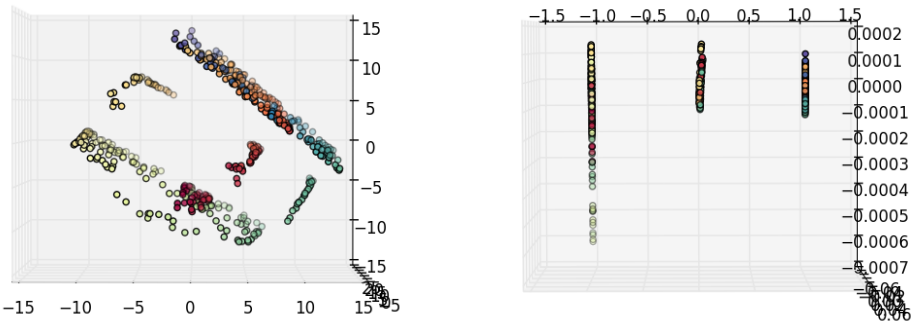
$\lambda = 4$



$\lambda = 1$



$\lambda = 0.01$



$\lambda = 0.0001$

Abbildung 5.8: Dargestellt sind jeweils die Approximationen (links) und die zweidimensionale Darstellung eingebettet in den dreidimensionalen Raum (rechts) für die Regularisierungsparameter $4, 1, 0.01, 0.0001$ des Swiss Roll Datensatzes für 500 Datenpunkte nach 100 Iterationen.

und Geraden, auf denen ein Großteil der Datenpunkte liegen, erkannt werden. Betrachtet man zudem die zweidimensionale Darstellungen der Swiss Roll für $\lambda = 0.0001$ und 0.01, so sind diese Darstellung der Swiss Roll nicht brauchbar, da diese bezüglich der y-Achse eine starke Skalierung aufweisen.
Je größer man den Regularisierungsparamter wählt, umso klarer wird eine zufällige Anordnung der Datenpunkte erkennbar. Damit ist gemeint, dass die berechneten Approximationen immer seltener auf einer gemeinsamen Geraden oder Ebene liegen und die spiralförmige Struktur der Swiss Roll sowohl in Höhe und Breite als auch in der Tiefe besser erkennbar wird. Trotzdem kann eine Vermischung verschiedenfarbiger Datenpunkte erkannt werden. Betrachtet man zusätzlich die berechnete zweidimensionale Darstellung, so liefert das PML-Verfahren für jeden der betrachteten Regularisierungsparameter keine nützliche zweidimensionale Darstellung, da kein gleichmäßiger Farbverlauf erkennbar ist. Allerdings wird für $\lambda = 1$ und $\lambda = 4$ immerhin der gesamte Raum $[-1, 1]^2$ ausgeschöpft. Für die beiden kleineren Parameter ist die y-Achse deutlich skaliert worden, um eine gewissen Struktur sichtbar zu machen.

### 5.3.3 Beispiel Oberflächenrekonstruktion

Für das Experiment der Oberflächenrekonstruktion tritt für verschiedene Regularisierungsparameter Konvergenz ein. Allerdings ist dabei keine eindeutige Tendenz erkennbar, welche Größenordnung eine gute Wahl der Parameters $\lambda$ haben sollte. Betrachtet man die rekonstruierten Oberflächen in Abbildung 5.9, so ähneln sich die berechneten Oberflächen sehr stark, auch wenn man punktuell kleinere Unterschiede im Randverhalten erkennen kann. Dieses Resultat unterscheidet sich an dieser Stelle von den präsentierten Ergebnissen in [SMSW01], da dort für verschiedene Regularisierungsparameter auch verschiedene Oberflächen gezeigt werden.

Insgesamt kann man festhalten, dass die Wahl eines geeigneten Regularisierungsparameters von der jeweiligen Situation abhängt. Aus diesem Grund lässt sich keine genaue Vorhersage für einen sinnvolle Wahl des Regularisierungsparameters treffen.

## 5.4 Einfluss der Gaußbreite

Nachdem wir den Einfluss des Regularisierungsparameters analysiert haben, wollen wir uns nun dem zweiten Parameter, der Gaußbreite, widmen. Die Gaußbreite $\sigma$ tritt hauptsächlich in der Auswertung des Gaußschen RBF-Kerns auf und hat damit auch einen großen Einfluss auf das PML-Verfahren und dessen Resultate. Um diesen Einfluss deutlich zu machen betrachtet man das Experiment der Polynomrekonstruktion. In Abbildung 5.10 ist der Verlauf der rekonstruierten Funktion $f$ auf dem Intervall $[-1, 1]$ für den Regularisierungsparameter $\lambda = 0.001$ und für verschiedene Gaußbreiten dargestellt. Für $\sigma > 1$ tritt eine deutliche Einschränkung der Funktionsapproximation ein. Exemplarisch kann man für $\sigma = 2$ erkennen, dass die Verformung der Graden, welche durch die Initialisierung berechnet worden ist, zu steif ist und die Ursprungsfunktion nicht ausreichend genug rekonstruiert wird. Dagegen wird

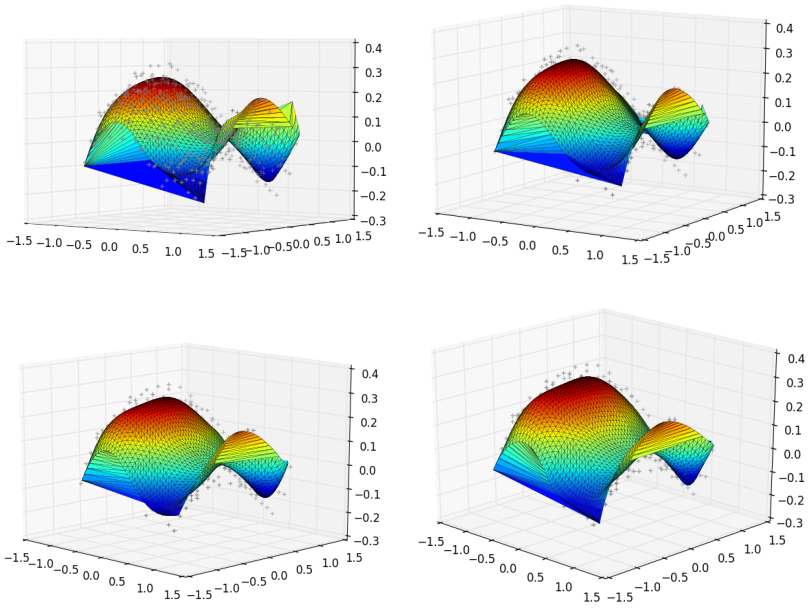


Abbildung 5.9: Oberflächenrekonstruktion für die Regularisierungsparameter $0.1, 0.01, 0.001, 0.0001$. Als Gaußbreite ist $\sigma = 1$ gewählt. Es wurde jeweils Konvergenz, im Sinne einer Fehlerdifferenz von $10^{-2}$, erreicht.

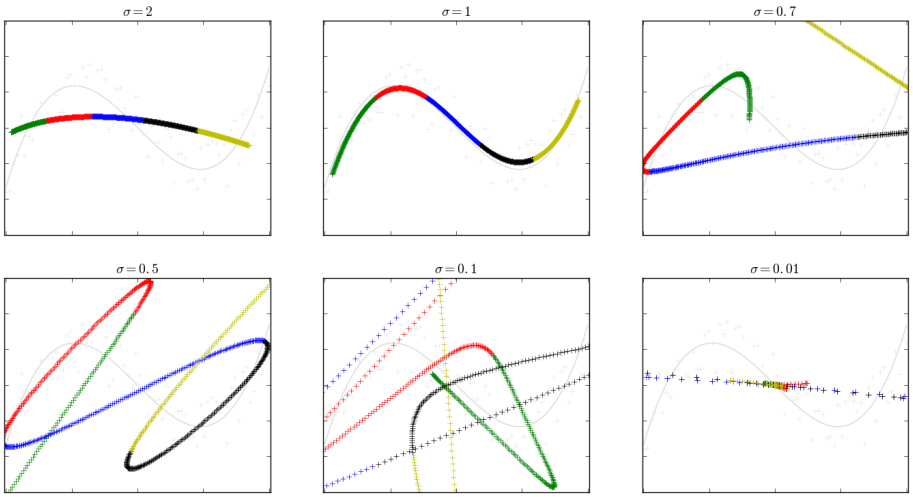



Abbildung 5.10: Auswertung der berechneten Funktion $f$ auf dem gesamten Zielraum $[-1, 1]$ für verschiedene Gaußbreiten $\sigma$. Dabei wurde bei allen Gaußbreiten Konvergenz erreicht mit $\lambda = 0.001$

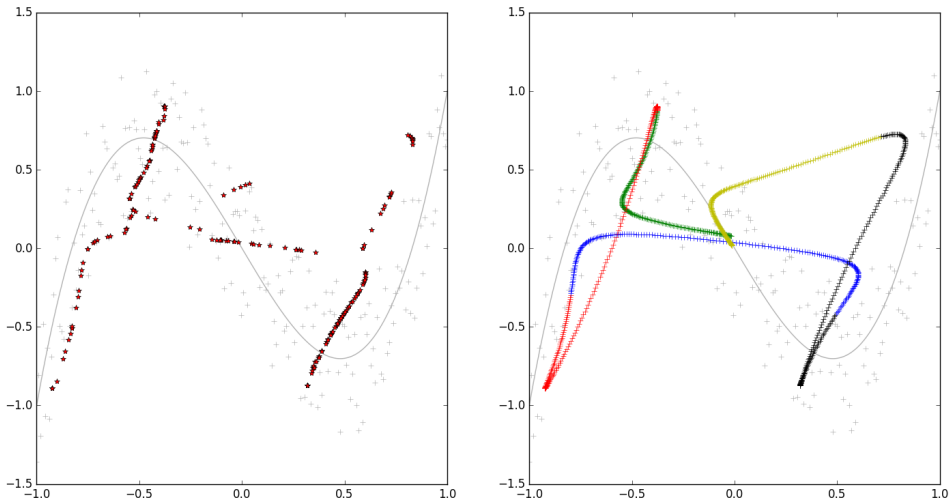



Abbildung 5.11: Abgebildet sind die ungestauchten Polynomdaten und deren Approximation (links) und der Verlauf der Funktion $f$ ausgewertet auf dem Intervall $[-1, 1]$ mit den Parametern $\lambda = 0.01$ und $\sigma = 0.1$.

das Polynom $p$ für $\sigma = 1$ nahezu optimal rekonstruiert. Im Gegensatz dazu tritt für $\sigma < 1$ ein starkes Overfitting ein. Für $\sigma = 0.5$ oder 0.1 kann man erkennen, dass der Verlauf der Funktion zwar eine eingeschränkte Datenapproximation zulässt, aber die berechnete Funktion selber keine Funktion mehr ist. Selbst bei kleinen Abweichungen, wie zum Beispiel für $\sigma = 0.7$, erhält man keine Funktion mehr. Je kleiner also die Gaußbreite gewählt wird, umso stärker tritt merkwürdiges Verhalten für die berechnete Funktion auf und liefert in Bezug zum Erhalt einer Funktion $f$ nur inakzeptable Resultate.
Wie gerade beschrieben tritt für kleine Werte der Gaußbreite merkwürdiges Verhalten in der rekonstruierten Funktion $f$ auf. Lassen wir diesen Aspekt mal außen vor, so kann der PML-Algorithmus trotzdem eine Approximationen an die Datenpunkte berechnen, welche nicht optimal, aber annehmbar ist. Dies wird in Abbildung 5.11 sichtbar. Betrachtet man für die Parameter $\lambda = 0.01$ und $\sigma = 0.1$ den Verlauf der berechneten Funktion, so ist dieser unbrauchbar. Trotzdem liegen die Approximationen ungefähr in der Nähe der Originaldaten. Ein selber Effekt kann für das Swiss Roll Experiment gezeigt werden.

## 5.5 Konvergenzen

Neben dem Einfluss der Parameter $\lambda$ und $\sigma$ auf die Resultate des PML-Verfahren spielt auch die Konvergenz der Resultate eine wichtige Rolle. Dabei kann man zwei Aspekte untersuchen: die Art der Konvergenz und die Schnelligkeit der Konvergenz. Unter den ersten Aspekt fallen alle Überlegungen bezüglich eines geeigneten Abbruchkriteriums, da der Begriff Konvergenz bezüglich des Abbruchkriteriums eingeführt wurde. Die Schnelligkeit spielt dahingehend eine Rolle, da man im Rahmen der Datenanalyse zügig nützliche Resultate benötigt. Ziel dieses

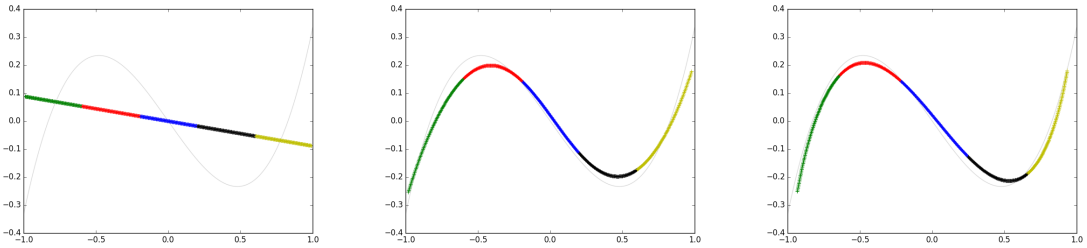


Abbildung 5.12: Darstellung des Verlaufs der Funktion $f$ ausgewertet auf dem Intervall $[-1, 1]$ zu verschieden Zeitpunkten. Initialisierung (links), erste Iteration (mitte) und Konvergenz (rechts)

Abschnittes ist die Analyse diese zwei Aspekte.
Dazu betrachte man zunächst die Abbildung 5.12. Man kann erkennen, dass nach einer Iteration der Verlauf des betrachteten Polynoms fast exakt rekonstruiert wird. Vergleicht man den Verlauf nach einer Iteration mit dem Verlauf nach Eintreten der Konvergenz, so lassen sich nur sehr kleine Unterschiede feststellen. Daraus kann man schließen, dass in diesem Experiment eine Iteration für eine ausreichende Rekonstruktion genügt. Ähnliche Effekte können auch für die dreidimensionale Oberflächenrekonstruktion erzielt werden. Dabei muss man allerdings beachten, dass zwar wenige Iterationen ausreichen, um akzeptable Ergebnisse zu erhalten, aber eine falsche Abbruchbedingung dazu führen kann, dass der PML-Algorithmus nicht nach diesen wenigen ausreichenden Iterationen terminiert, sondern plötzlich schlechtere Ergebnisse berechnet. Betrachtet man in Abbildung 5.13 den Verlauf des Regularisierungsfehlers $R_{\text{reg}}[f]$, so wird zunächst die gewünschte Minimierung des Fehlers $R_{\text{reg}}[f]$ pro Iteration erreicht, aber ab der 16. Iteration steigt der Fehler sprunghaft an und die Berechnung konvergiert nicht mehr. Dabei wurde als Abbruchbedingung der Fehlerunterschied zwischen zwei aufeinanderfolgenden Iterationen genutzt und der Algorithmus beendet, falls dieser Unterschied kleiner als $10^{-5}$ ist. Diese Größenordnung der Fehlerdifferenz erweist sich als zu klein und demnach als zu genau um ein geeignetes Resultat zu erhalten. Eine Fehlerdifferenz in der Größenordnung von $10^{-3}$ bis $10^{-2}$ führt dabei rechtzeitig zu einer gewünschten Konvergenz.

Ein weiteres Beispiel für diesen Effekt ist in Abbildung 5.14 dargestellt. Hierbei wurden sowohl der Regularisierungsfehler, der empirische Fehler und der Regularisierungsterm pro Iteration für das Swiss Roll Experiment dargestellt. Zunächst wird innerhalb der ersten Iterationen der Regularisierungsfehler verringert, bevor dieser immer wieder enorm ansteigt. Untersucht man die Gründe für diese plötzlichen Anstiege des Regularisierungsfehlers, so kann man feststellen, dass weder der Projektions- noch der Adaptionsschritt alleine verantwortlich sind. Des Weiteren erkennt man, dass der Regularisierungsterm $Q[f]$ einen geringen Einfluss auf den Regularisierungsfehler $R_{\text{reg}}[f]$ hat, da dieser zum einen im Verhältnis und zum anderen auch absolut gesehen sehr klein ist. Daraus folgt insbesondere, dass der Regularisierungsfehler $R_{\text{reg}}[f]$ und der empirische Fehler $R_{\text{emp}}[f]$ sich kaum voneinander unterscheiden.

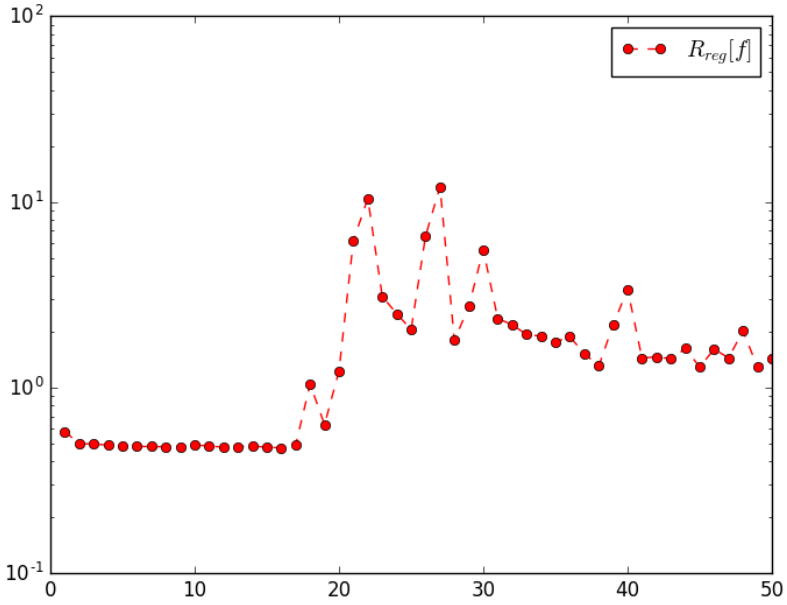



Abbildung 5.13: Verlauf der Fehler und Regularisierungsterme der ersten 50 Iterationen für die Rekonstruktion der Oberfläsche des zweidimensionalen Polynoms mit den Parametern $\lambda = 0.0001$, $\sigma = 1$ und als Startwert des Optimierers die vorherige niedrig-dimensionale Darstellung des Datenpunktes.

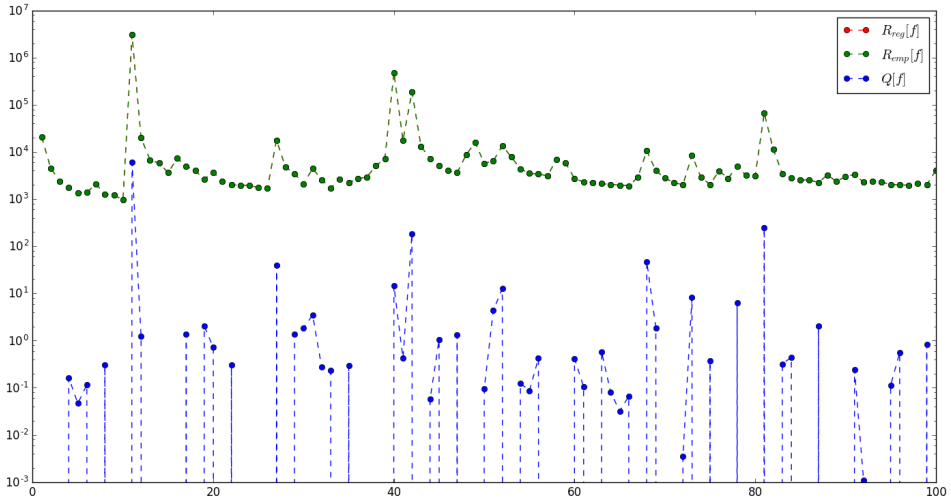



Abbildung 5.14: Verlauf der Fehler und des Regularisierungsterms der ersten 100 Iterationen für den Swiss Roll Datensatz mit den Parametern $\lambda = 1$ und $\sigma = 1$. Als Initialisierung des Optimierers wird die vorherige niedrig-dimensionale Darstellung des Datenpunktes genutzt.

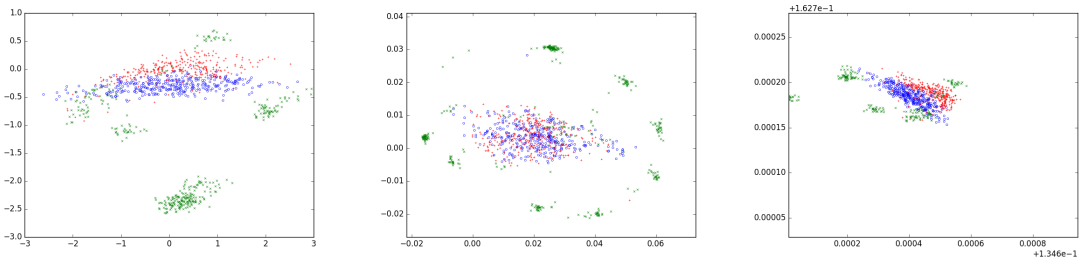


Abbildung 5.15: In dieser Abbildung sind die zweidimensionalen Darstellungen des Öl-Fluss Datensatzes für verschiedene Konfigurationen dargestellt. In der oberen Graphik wurde die PCA zum Erhalt einer zweidimensionalen Darstellung genutzt. In der mittleren Graphik ist die Darstellung nach 18.099 Iterationen mit dem modifiziertem PML-Verfahren dargestellt. In der unteren Graphik ist die Darstellung, die das reguläre PML-Verfahren nach 200 Iterationen berechnet hat, zu sehen.

## 5.6 Resultate für höherdimensionale Daten am Beispiel des Öl-Fluss Experiments

Für den Öl-Fluss-Datensatz zeigt das PML-Verfahren mit den Parameterwahlen, die in den Experimenten in [SMSW01] gewählt wurden, keine überzeugende Ergebnisse, s. Abbildung 5.15. Nutzt man zunächst die PCA, so kann man erkennen, dass es in der niedrigdimensionalen Darstellung Bereiche gibt, in denen keine klare Klassifikation der drei Klassen möglich ist. Dieses Resultat, welches auch in [SMSW01] beschrieben ist, kann bestätigt werden. Im Gegensatz dazu kann eine eindeutige Klassifizierung der Daten in drei Klassen durch den PML-Algorithmus nicht bestätigt werden. In Abbildung 5.15 ist zu erkennen, dass es in bestimmten Bereichen zu eine Vermischung aller Klassen kommt. Allerdings kann man recht gut erkennen, dass jeweils die Klasse, dargestellt durch grünen Kreuze, relative gut getrennt dargestellt werden kann. Insbesondere die unterste Abbildung lässt die Vermutung zu, dass nach ausreichend Iterationen das gewünschte Ergebnis erreicht werden kann.
Führt man den Algorithmus mit anderen Parametern oder mit einer Variante des Projektionsschrittes durch, in der anstatt für alle Datenpunkte nur für eine zufällige Anzahl an Datenpunkten eine Minimierung durchgeführt wird, so zeigt sich der gleiche Effekt wie zuvor, s. mittlere Abbildung in 5.15.

# 6 Fazit

Die Arbeit baut hauptsächlich auf der Arbeit [SMSW01] auf. Es wird versucht, unüberwachtes Lernen unter Betrachtung einer Minimierung eines Fehlerfunktionals in überwachtes Lernen zu transformieren [SMSW01]. Dazu haben wir ein Fehlerfunktional hergeleitet und versucht, dieses zu minimieren. Dies führte zur Suche einer Funktion $f$, welche die Daten möglichst gut approximiert.
Um nun einen Algorithmus herzuleiten, der eine solche Funktion berechnet, wurde als Funktionenraum ein Hilbertraum mit reproduzierendem Kern gewählt. Diese Wahl bringt den Vorteil mit sich, dass sich die gesuchte Funktion als Summe von Kernfunktionen schreiben lässt. Neben der Anwendung des Repräsentierungstheorems nutzen wir den Zusammenhang zwischen Kernen und Regularisierungsoperatoren. Aus diesen Überlegungen heraus konnte man relativ leicht ein Algorithmus formulieren. Alles in allem lässt sich dieses Verfahren sehr einfach mit den Methoden aus dem Bereich der Kern-Verfahren beschreiben und fasst dabei alle wesentlichen Resultate aus dem Bereich des RKHS zusammen. Dies bedeutet, dass das PML-Verfahren gut in den Rahmen der kernbasierten Funktionsdarstellung und der Theorie des RKHS passt.
Neben diesen guten Aspekten des PML-Verfahren gibt es aber auch kritische Aspekte, weshalb das PML-Verfahren nicht die gewünschten Ergebnisse leistet. Dazu wurde in Kapitel 5 gezeigt, dass einige der in [SMSW01] dargestellten Resultate nur zum Teil bestätigt werden können. Beispielsweise kommen die Autoren zu dem Schluss, dass die Gaußbreite keinen großen Einfluss auf die Resultate hat. Wir haben hingegen einen starken Einfluss der Gaußbreite $\sigma$ auf die Resultate zeigen können. Auch die Wahl des Regularisierungsparameters $\lambda$ kann nicht verallgemeinert werden und ist demnach von der betrachteten Situation abhängig, das heißt je nachdem aus welcher praktischen Anwendung die betrachteten Daten stammen, muss der Wert des Parameters $\lambda$ anders gewählt werden. Insbesondere konnten Unterschiede in der Wahl des Regularisierungsparameters nur für andere Werte, also nicht für die in [SMSW01] beschriebenen Werte, erreicht werden.
Eine weitere Schwierigkeit, welche die in [SMSW01] dargestellten Resultate eher künstlich erscheinen lassen, ist das Fehlen von präzisen Angaben der Datensätze, auf die die Autoren das PML-Verfahren angewandt haben. Es wurde gezeigt, dass nur unter günstigen Voraussetzungen einige der Resultate bestätigt werden können. Daraus kann man Folgern, dass die Autoren möglicherweise ihre Daten derart skaliert haben, dass die Tauglichkeit des Verfahrens nachgewiesen werden kann. Ein weiterer Anhaltspunkt für diese These sind die Resultate der Oberflächenrekonstruktion und des Öl-Fluss-Experiments. In beiden Fällen kann man nach einer gewissen Anzahl an Iterationen bessere und schlechtere Resultate erzielen, ohne dass dabei eine Konvergenz zustande kommt. Daher liegt es Nahe, dass die Autoren möglicherweise bestimmte Zeitpunkte eines Experiments darstellen, in denen der PML-Algorithmus das gewünschte liefert. Dies kann vor allem am Verlauf des Regularisierungsfehlers belegt werden, da die Autoren lediglich einen Verlauf eines, nicht weiter erläuterten und daher

unbekannten, Experiments für die ersten 10 Iterationen zeigen. Betrachtet man den dargestellten Fehlerverlauf des Swiss-Roll Experiment für die ersten 10 Iterationen, so können die in [SMSW01] gezeigten Fehlerverläufe bestätigt werden, obwohl ab der 11. Iteration das Verfahren instabiles Verhalten aufweist. Dies unterstützt die These, dass die Autoren nur Resultate zeigen, die die Korrektheit des Verfahrens zeigen soll.
Des Weiteren haben wir in den Kapiteln 3 und 4 gesehen, dass der Projektions-Schritt einen wesentlichen Einfluss auf die Resultate des Verfahrens hat. Genau an dieser Stelle beschreiben die Autoren nicht ausführlich, wie man dort das gesuchte Minimum berechnen kann und verweisen den Leser auf ein Standardwerk über numerische Verfahren. Es bleibt daher völlig offen, wie genau die Minimierung und die Konvergenz der dargestellten Resultate zustande gekommen sind. Daher liegt auch hier die Vermutung nahe, dass ausschließlich sinnvolle Resultate gezeigt werden.
Eine mögliche Fehlerquelle, nämlich eine fehlerhafte Programmierung des PML-Verfahren, ist nahezu auszuschließen, da man beispielsweise für die Polynomrekonstruktion die gewünschte Resultate erzielen kann. Zudem zeigt sich bei der Ausführung der Programmierung eine gewisse Stabilität. Allerdings kann man, je nachdem wie die Parameter gewählt werden und wie die Minimierung im Projektionsschritt umgesetzt wird, andere Resultate erhalten. Alles in allem besitzt das PML-Verfahren sehr viele Freiheitsgerade, welche für die jeweilige Situation angepasst werden müssen um brauchbare Ergebnisse zu erhalten.
Ein weiteres Argument, weshalb das PML-Verfahren in der Anwendung nicht praktikabel genug ist beruht auf der Tatsache, dass es wenige Folgearbeiten von Wissenschaftlern gibt, die dieses Verfahren in der Praxis anwenden. Wissenschaftliche Arbeiten, die die Arbeit von Smola, Mika, Schölkopf und Williamson referenzieren, erwähnen meist nur, dass es diese Möglichkeit der Datenanalyse gibt oder skizzieren grob, was dort gemacht wird. Auch die Tatsache, dass man mit diesem Verfahren lediglich ein lokales Minimum finden kann ist unbefriedigend.
In dieser Arbeit wurden einige Faktoren diskutiert, für die gezeigt werden kann, dass diese einen Einfluss auf die Resultate des PML-Verfahrens haben. Man könnte diese Analyse nun fortführen und weiter Faktoren, wie beispielsweise die Anzahl an Kernfunktionen, analysieren.

# Literatur


[A.63] Tikhonov A. „Solution of Incorrectly Formulated Problems and the Regularization Method“. In: *Soviet Math. Dokl.* 5 (1963), S. 1035–1038.

[BN03] Mikhail Belkin und Partha Niyogi. „Laplacian Eigenmaps for Dimensionality Reduction and Data Representation“. In: *Neural Comput.* 15.6 (Juni 2003), S. 1373–1396. ISSN: 0899-7667. DOI: 10.1162/089976603321780317. URL: http://dx.doi.org/10.1162/089976603321780317.

[BNSNB94] Richard H. Byrd, Jorge Nocedal, Robert B. Schnabel, Richard H. Byrd Jorge Nocedal und Robert B. *Representations Of Quasi-Newton Matrices And Their Use In Limited Memory Methods*. 1994.

[DLR77] A. P. Dempster, N. M. Laird und D. B. Rubin. „Maximum likelihood from incomplete data via the EM algorithm“. In: *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 39.1 (1977), S. 1–38.

[Gar04] J. Garcke. „Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern“. Doktorarbeit. Institut für Numerische Simulation, Universität Bonn, 2004.

[Gar16] J. Garcke. *“Scientific Computing II - Kernel Methods and Nonlinear Dimensionality Reduction” Lecture Notes*. http://www.ins.uni-bonn.de/teaching/vorlesungen/WissRech2SS16/lecnotes/lecture-notes.pdf. 2016.

[GK99] C. Geiger und C. Kanzow. *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Springer-Lehrbuch. Springer Berlin Heidelberg, 1999. ISBN: 9783540662204.

[Hag14] Aria D. Haghighi. *Numerical Optimization: Understanding L-BFGS*. [Online; Stand 15. August 2016]. 2014. URL: http://aria42.com/blog/2014/12/understanding-lbfgs.

[Hot33] H. Hotelling. „Analysis of a Complex of Statistical Variables with Principal Components“. In: *Journal of Educational Psychology* 24 (1933), S. 417–441.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson u. a. *SciPy: Open source scientific tools for Python*. [Online; Stand 13. August 2016]. 2001–. URL: http://www.scipy.org/.

[Law16] Neil Lawrence. *3 Phase Oil Data*. [Online; Kopie der Originalwebsite von Markus Svensén aus dem Jahr 1996; Stand 15. August 2016]. 2016. URL: http://inverseprobability.com/3PhaseData.html.

[LV07] John A. Lee und Michel Verleysen. *Nonlinear Dimensionality Reduction*. 1st. Springer Publishing Company, Incorporated, 2007. ISBN: 0387393501, 9780387393506.

[LYL14] Xudong Luo, Jeffrey Xu Yu und Zhi Li. *Advanced Data Mining and Applications: 10th International Conference, ADMA 2014, Guilin, China, December 19-21, 2014, Proceedings*. Springer Publishing Company, Incorporated, 2014.

[MS98] Christopher K.I. Williams Markus Svensén Christopher Bishop. „GTM: The Generative Topographic Mapping“. Diss. Jan. 1998, 215–234. URL: https://www.microsoft.com/en-us/research/publication/gtm-the-generative-topographic-mapping/.

[Oli06] Travis E. Oliphant. *Guide to NumPy*. Provo, UT, März 2006. URL: http://web.mit.edu/dvp/Public/numpybook.pdf.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling und Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992. ISBN: 0-521-43108-5.

[Sch11] Robert Schaback. *“Kernel-Based Meshless Methods.” Lecture Notes*. http://num.math.uni-goettingen.de/schaback/teaching/AV_2.pdf. 2011.

[Sci] *SciPy Documentation*. http://docs.scipy.org/doc/scipy/reference/optimize.html. [Online; Stand 13. August 2016].

[SHS01] B. Schölkopf, R. Herbrich und AJ. Smola. „A Generalized Representer Theorem“. In: *Lecture Notes in Computer Science, Vol. 2111*. LNCS 2111. Berlin, Germany, 2001, S. 416–426.

[Sim] *SIMDATA-NL: Nichtlineare Charakterisirung und Analyse von FEM-Simulationsergebnissen für Autobauteile und Crash-Tests*. [Online; Stand 18. August 2016]. URL: http://garcke.ins.uni-bonn.de/research/projects/SIMDATA/.

[Ska10] Anders Skajaa. „Limited Memory BFGS for Nonsmooth Optimization“. Magisterarb. Courant Institute of Mathematical Science, New York University, 2010.

[SMSW01] AJ. Smola, S. Mika, B. Schölkopf und RC. Williamson. „Regularized principal manifolds“. In: *Journal of Machine Learning Research* 1 (Juni 2001), S. 179–209.

[SS02] B. Schölkopf und AJ. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Dez. 2002, S. 644.

[SSM98] AJ. Smola, B. Schölkopf und K-R. Müller. „The connection between regularization operators and support vector kernels.“ In: *Neural Networks* 11.4 (Juni 1998), S. 637–649.

[SSM99] Bernhard Schölkopf, Alexander J. Smola und Klaus-Robert Müller. „Advances in Kernel Methods“. In: Hrsg. von Bernhard Schölkopf, Christopher J. C. Burges und Alexander J. Smola. Cambridge, MA, USA: MIT Press, 1999. Kap. Kernel Principal Component Analysis, S. 327–352.

[Swi] http://blog.albert2005.co.jp/2014/12/11. (Besucht am 04. 08. 2016).

[TA77] A.N. Tikhonov und V.I.A. Arsenin. *Solutions of ill-posed problems*. Winston, 1977.

[TSL00] Joshua B. Tenenbaum, Vin de Silva und John C. Langford. „A Global Geometric Framework for Nonlinear Dimensionality Reduction“. In: *Science* 290.5500 (2000), S. 2319–2323.

[UU12] M. Ulbrich und S. Ulbrich. *Nichtlineare Optimierung*. Mathematik Kompakt. Springer Basel, 2012. ISBN: 9783034601429.

[VTS04] JP. Vert, K. Tsuda und B. Schölkopf. „A Primer on Kernel Methods“. In: *Kernel Methods in Computational Biology*. Cambridge, MA, USA: MIT Press, 2004, S. 35–70.

[Wah90] Grace Wahba. *Spline models for observational data*. CBMS-NSF regional conference series in applied mathematics. Philadelphia: Society for industrial und applied mathematics, 1990.

[ZBLN97] Ciyou Zhu, Richard H. Byrd, Peihuang Lu und Jorge Nocedal. „Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-scale Bound-constrained Optimization“. In: *ACM Trans. Math. Softw.* 23.4 (Dez. 1997), S. 550–560. ISSN: 0098-3500.

[Num16] NumPy Development Team. *NumPy: fundamental package for scientific computing with Python*. [Online; Stand 15. August 2016]. 2016. URL: http://www.numpy.org/index.html.