Kronecker-Produkte und multivariate Tensorprodukt Splines auf gestreuten Daten

Dissertation

zur Erlangung des akademischen Grades
"Doktor der Naturwissenschaften"
(Dr. rer. nat.)

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

vorgelegt von

Dipl. Math. Frank Lamping

geb. in Dillenburg

Giessen, September 2015

Datum der Disputation: 20.07.2015

Erstgutachter:	Prof. Dr. M. Buhmann Lehrstuhl für Numerische Mathematik Universität Giessen
Zweitgutachter:	Prof. Dr. T. Sauer Lehrstuhl für Mathematik mit Schwerpunkt Digitale Bildverarbeitung Universität Passau
Prüfer:	Prof. Dr. Bernhard Mühlherr Universität Giessen
Prüfer:	Prof. Dr. Max Horn Universität Giessen

Für meine Eltern

und

für Tina.

Danksagung:

Ich möchte mich zuerst bei Herrn Prof. Dr. Sauer für die sehr gute Unterstützung und Betreuung im Laufe meiner Promotion, die anregenden wissenschaftlichen Diskussionen und hilfreichen Ideen bedanken. Mir haben die Besuche in Passau immer eine große Freude bereitet und mich während meiner Promotionszeit sehr motiviert.

Des Weiteren bin ich Herrn Prof. Dr. Buhmann für die Übernahme meiner Betreuung und für eine wirklich gewinnbringende Zeit in seiner Arbeitsgruppe sehr dankbar. Den Mitarbeiterinnen und Mitarbeitern der Arbeitsgruppe Numerik danke ich für eine sehr gute Zusammenarbeit und viele anregende sowie erhellende Diskussionen.

Mein ganz besonderer Dank gilt meiner Familie und meinen Freunden für die liebe Unterstützung und das große Verständnis im Verlauf meiner Promotion. Vorallem danke ich meiner Freundin Tina für ihre Geduld und Toleranz gegenüber meinem mathematischen Forscheralltag mit seinen Höhen und Tiefen. Auch meinem Bruder Matthias möchte ich für die vielen Gespräche über unsere Forschung und die gemeinsame Zeit danken.

Zusammenfassung

Bei der Bestimmung und Anwendung von Tensorprodukt Smoothing Splines auf gestreuten Daten treten große Datenmengen auf, welche nicht mit einem handelsüblichen PC bearbeitet werden können. Mit Hilfe von Summen von Kronecker-Produkten lässt sich diese Problematik effizient lösen. In der vorliegenden Arbeit wird dieser Zusammenhang erläutert und weitere Verfahren vorgestellt, um den Umgang mit Summen von Kronecker-Produkten zu optimieren.

Es wird eine auf Multilinearformen basierende Methode präsentiert, welche Summen von Kronecker-Produkten durch eine Summe mit weitaus weniger Summanden komprimiert bzw. approximiert. Das Verfahren wird mit Methoden aus der multilinearen Algebra verglichen, welche ebenfalls theoretisch anwendbar sind. Weiterhin wird mit Hilfe des Prokrustes-Problemes eine Möglichkeit entwickelt, um die Inverse eine Summe von Kronecker-Produkten erneut durch Kronecker-Produkte zu approximieren. Es wird an numerischen Beispielen gezeigt, dass diese Approximation als Vorkonditionierer für das GMRES-Verfahren verwendet werden kann. Abschließend wird ein Vorkonditionierer für das CG-Verfahren basierend auf Kronecker-Produkten vorgestellt.

Abstract

Using tensor product smoothing splines on scattered data will lead to a great amount of data, which extends the capacity of usual workstations. Sums of Kronecker products appear as a natural possibility to avoid this difficulty. The connection between the mentioned multivariate splines and Kronecker products will be shown. Furthermore several methods to optimize the handling of sums of Kronecker products will be presented.

This thesis introduces a method based on multilinear forms, which approximates sums of Kronecker products through sums with less summands. A comparison with procedures from multilinear algebra is represented. Based on the procrustes problem a possibility to approximate the inverse of a sum of Kronecker products through Kronecker products is developed. Numerical experiments show that this approximation can be used as a preconditioner for the GMRES method. At the end a preconditioner for the CG method, based on Kronecker products, is presented.

Inhaltsverzeichnis

1. Einleitung **9**
- 1.1. Gliederung der Arbeit 12
- 1.2. Neue Erkenntnisse der Arbeit 14

2. Grundlagen **17**
- 2.1. Kronecker-Produkte 17
- 2.2. Tensorprodukt Splines 27
- 2.3. Iterative Verfahren zum Lösen von linearen Gleichungssystemen 39
  - 2.3.1. CG-Verfahren & GMRES-Verfahren 40
  - 2.3.2. Vorkonditionierung 42
- 2.4. Multilinearformen & Tensoren 48

3. Komprimierung von Tensorprodukt Smoothing Splines **65**
- 3.1. Approximation einer Multilinearform 69
- 3.2. Algorithmus zur Approximation einer Multilinearform 73
- 3.3. Effizienz der Komprimierung bei multivariaten Smoothing Splines 76
- 3.4. Existierende Verfahren für Tensoren 82

4. Approximation einer Kronecker-Inversen **93**
- 4.1. Das Prokrustes-Problem und Kronecker-Produkte 95
- 4.2. Effiziente Umsetzung der Approximation einer Kronecker-Inversen 104
- 4.3. Konvergenzverhalten des vorgestellten Algorithmus 109

5. Vorkonditionierer **113**
- 5.1. SSOR-Vorkonditionierer 114
- 5.2. Prokrustes-Vorkonditionierer für das CG-Verfahren 120

6. Numerische Ergebnisse **123**
- 6.1. Komprimierung 124
- 6.2. Approximation der Kronecker-Inversen 125
- 6.3. SSOR-Vorkonditionierer 129

7. Zusammenfassung & Fazit **133**

8. Ausblick **137**

A. Implementierung **139**
A.1. Tensorprodukt Smoothing Splines 140
A.2. Iterative Verfahren zum Lösen von LGS 142
A.3. Komprimierung 144
A.4. Approximation der Kronecker-Inversen 145
A.5. Vorkonditionierer 146

Symbolverzeichnis **149**

Literatur **153**

1 Einleitung

In der heutigen Zeit wird die Erfassung von Daten immer schneller und einfacher. Dies bedeutet im Wesentlichen, dass die Datenmengen stark anwachsen und die zu lösenden Probleme immer mehr Dimensionen umfassen. Diese Zunahme an Daten und Dimensionen lässt sich in vielen Bereichen der numerischen Mathematik beobachten. Handelsübliche Computersysteme werden zwar leistungsstärker und haben ebenfalls mehr Speicher, allerdings sind die entwickelten Verfahren in vielen Fällen nicht für die Komplexität der heutigen Aufgabenstellungen ausgelegt oder die Umsetzung der Berechnung auf einem zur Verfügung stehenden Computersystem ist zu kompliziert.

Zur Bewältigung und Verbesserung dieser Problematik sind daher effizientere mathematische Verfahren erforderlich. Diese müssen mit den gestiegenen Anforderungen umgehen und dabei effektiv umgesetzt werden können. In vielen Fällen ist eine naive Umsetzung der Verfahren nicht zielführend, da der benötigte Speicher nicht ausreicht oder die Dauer der Auswertung bzw. der Bestimmung der Lösung nicht akzeptabel ist. Stellenweise lässt sich diese Problematik umgehen, wenn bei der Implementierung der Verfahren ein Kompromiss zwischen der Speicher- und der Rechenauslastung gewählt wird. Dieser Ansatz funktioniert allerdings nur in seltenen Fällen. In diesem Zusammenhang spielt der „Fluch der Dimensionen“ eine wesentliche Rolle. Diese Bezeichnung wurde zuerst von Bellmann in [1] eingeführt und drückt aus, dass mit jeder zusätzlichen Dimension der Speicher- bzw. Rechenaufwand der Verfahren exponentiell anwächst. Das Problem kann generell nicht nur auf die Daten zurückgeführt werden, son-

dern ist vielmehr ein Zusammenspiel zwischen den Daten und den angewendeten Verfahren. Viele Methoden können in niedrigen Dimensionen noch effizient verwendet werden, aber sobald die Anzahl der Dimensionen ansteigt, ist dies nicht mehr gegeben. Ein solches Verhalten ist bei den in dieser Arbeit betrachteten multivariaten Smoothing Splines zu erkennen. Das Ziel soll sein, die Verfahren in einem solchen Maße umzugestalten, dass sie in höheren Dimensionen überhaupt verwendbar sind und effizient umgesetzt werden können.

Das zugrundeliegende, hochdimensionale Problem, welches als Motivation für diese Arbeit dient, kommt aus dem Bereich der Chemie, siehe [44]. Es wurde bereits in meiner Diplomarbeit [24] bearbeitet. Diese bildet die Grundlage für die weitere Untersuchung. Bei der Problemstellung handelt es sich um die Approximation einer chemischen Reaktion an einem Katalysator. Diese Reaktion

$$f : \mathbb{R}^p \to \mathbb{R}$$

kann im Grunde beliebig viele Eingangsdaten haben, um das Verhalten des Katalysators zu beschreiben. Die Reaktion wird an N Punkten gemessen oder durch ein Modell simuliert. Allgemein formuliert, ist das Ziel diese komplizierte und schwer auszuwertende Funktion f durch eine einfachere Funktion s darzustellen. Eine exakte Darstellung durch die Funktion s ist in vielen Fällen nicht möglich, so dass eine Approximation ausreichen muss. Mit Hilfe der gegebenen N Punkte wird eine Annäherung durch eine Funktion s an f bestimmt.

In [24] wurden für die Approximation Tensorprodukt Smoothing Splines verwendet, da sie aufgrund ihrer Flexibilität und Struktur einen geeigneten Ansatz darstellen. Direkt zu Beginn wurde festgestellt, dass eine naive Implementierung nur in niedrigen Dimensionen ohne Bedenken umsetzbar ist, wenn die Datenmenge nicht zu groß wird. In höheren Dimensionen ist dieser Ansatz nicht praktikabel oder sehr ineffizient. Auf den ersten Blick scheint es nicht offensichtlich, dass die Umsetzung schon sehr schnell nicht mehr möglich ist. Das grundlegende Problem an dieser Stelle lässt sich erneut durch den „Fluch der Dimensionen“ beschreiben. Um dies zu veranschaulichen, soll kurz die Vorgehensweise bei der Approximation mit einem Tensorprodukt Spline erläutert werden. Ein Tensorprodukt Spline besteht aus einer Linearkombination von Basisfunktionen. Um einen für die vorgegebenen Daten passenden Spline zu bestimmen, müssen die Koeffizienten für die Basisfunktionen berechnet werden. Dies lässt sich durch das

Lösen eines linearen Gleichungssystems (LGS) bewerkstelligen. Die auftretende Matrix wächst mit jeder Dimension exponentiell, so dass diese nur bis zu einer bestimmten Größe explizit gespeichert werden kann. In der verwendeten Programmiersprache `Octave` hat eine Matrix $A \in \mathbb{R}^{10000\times 10000}$ einen Speicherbedarf von circa 762 Mb, wenn die Einträge in `double`-Genauigkeit gespeichert werden. Bei der betrachteten Problemstellung können laut [24] durchaus mehrere Matrizen in dieser Größenordnung und solche mit größeren Dimensionen auftreten, so dass ein handelsübliches Computersystem mit 4 Gb Arbeitsspeicher sehr schnell an seine Grenze stößt bzw. Daten auf die Festplatte ausgelagert werden, was sehr lange Zugriffszeiten zur Folge hat.

Ein entscheidender Punkt zur Vermeidung des zuvor beschriebenen Speicherengpasses ist die Struktur der Daten. Liegen die Daten auf einem Gitter, lässt sich der Tensorprodukt Spline effizient durch ein Kronecker-Produkt $A \in \mathbb{R}^{m\times n}$ von Matrizen $A_j \in \mathbb{R}^{m_j\times n_j}$

$$A = \bigotimes_{j=1}^{p} A_j = A_1 \otimes \cdots \otimes A_p, \text{ mit } m = \prod_{j=1}^{p} m_j \text{ und } n = \prod_{j=1}^{p} n_j$$

bestimmen. Falls die Daten interpoliert werden sollen, existiert in [6] ein Verfahren für diesen speziellen Fall. Diese strukturelle Voraussetzung ist in vielen Fällen nicht gegeben, aber die Vorgehensweise lässt sich hier als Inspiration für den vorliegenden Fall verwenden. Das Verfahren ist bei dem betrachteten Problem nicht anwendbar, da die verwendeten Daten nicht auf einem Gitter liegen und eine Interpolation bedingt durch verrauschte Daten nicht zielführend wäre. Zusätzlich ist aufgrund der Größe des Funktionenraums, in welchem eine geeignete Funktion gesucht wird, das Problem in vielen Fällen unterbestimmt. Das bedeutet, dass mehr als eine Lösung existiert. Dies hängt mit dem „Fluch der Dimensionen“ und der geringen Anzahl an Stützstellen zusammen. In [24] wurde deswegen anstelle eines Tensorprodukt Splines ein Tensorprodukt Smoothing Spline verwendet. Ein solcher Smoothing Spline versucht die Funktion f möglichst glatt zu approximieren, aber dabei möglichst nah an den Daten zu bleiben. Es wurde gezeigt, dass ein Tensorprodukt Smoothing Spline ebenfalls durch Kronecker-Produkte dargestellt werden kann. Im Gegensatz zu [6] handelt es sich in diesem Fall um eine Summe von Kronecker-Produkten. Mit Hilfe einer Eigenschaft von Kronecker-Produkten bei der Multiplikation mit einem Vektor lässt sich auf diese Weise eine effiziente Umsetzung der Tensorprodukt Smoo-

thing Splines realisieren. Allerdings fehlen in [24] aufgrund der Komplexität und des Umfangs des Themas noch wesentliche Verbesserungen des Verfahrens. Das Ziel dieser Arbeit soll eine effiziente Lösung dieser Problemstellung sein.

Bei den möglichen Verbesserungen handelt es sich um die Bestimmung eines Vorkonditionierers, die Approximation einer Inversen und die Abschätzung eines Glättungsparameters, welcher den Trade-Off zwischen der Glattheit und der Approximationsgüte des Smoothing Splines reguliert. Die Ursache dafür, dass diese Punkte noch nicht effizient gelöst werden konnten, liegt an der Kronecker-Struktur. Diese muss überall zugrundeliegen, da ansonsten der „Fluch der Dimensionen“ eine effiziente Umsetzung vereitelt.

1.1. Gliederung der Arbeit

Der Aufbau der vorliegenden Arbeit gliedert sich wie folgt: In Kapitel 2 werden die Grundlagen erläutert, insbesondere wie sich Tensorprodukt Smoothing Splines durch Kronecker-Produkte darstellen lassen und wie diese mit Hilfe dieser Darstellung effizient bestimmt werden können. Des Weiteren werden adäquate iterative Verfahren zum Lösen des LGS und benötigte Datenstrukturen vorgestellt. Bei Kapitel 2 handelt es sich um ein vorbereitendes Kapitel, insofern werden dort keine neuen Ergebnisse präsentiert.

Um die obigen Punkte anzugehen, ist es sinnvoll eine Komprimierung der Summe von Kronecker-Produkten zu bestimmen. Dies soll in Kapitel 3 mit Hilfe von Multilinearformen bzw. Tensoren und durch die Verwendung von bekannten Verfahren aus dem Bereich der multilinearen Algebra, siehe [18, 23] und [25], ermittelt werden. Die Methoden zur Komprimierung existieren bereits in Bereichen mit anderem Zusammenhang — wie zum Beispiel in der Psychologie oder der Physik, wo Tensoren mehrdimensionalen Matrizen entsprechen. Ein Kronecker-Produkt lässt sich durch eine Umstrukturierung als Multilinearform auffassen und eine Multilinearform ist in dem betrachteten Fall als Tensor darstellbar. Die Komprimierung wird durch eine Zerlegung von Multilinearformen realisiert, aber da diese Tensoren entsprechen, ist dies nicht als Neuerung aufzufassen. Ein wichtiges Konzept bei der Bestimmung der Komprimierung ist die „Alternating Least-squares“ (ALS) Methode aus [3]. Die Idee dahinter ist die abwechselnde Berechnung von „Least-squares“-Problemen in den einzelnen multivariaten Komponenten.

Die Neuerungen bzgl. der vorzustellenden Komprimierung bestehen im Wesentlichen aus der Anwendung der Verfahren auf Tensorprodukt Smoothing Splines auf gestreuten Daten, einer Untersuchung, wie diese Komprimierung im Fall von Tensorprodukt Smoothing Splines weiter verwendet werden kann, sowie aus Abschätzungen, wann die vorgestellte Komprimierung überhaupt effizient und sinnvoll ist. Dies ist bisher noch nicht erforscht worden. Weiterhin werden andere bekannte Verfahren für Tensoren vorgestellt und es wird aufgezeigt, wann diese für die vorliegende Problemstellung überhaupt geeignet erscheinen.

In Kapitel 4 soll eine mögliche Approximation einer Inversen der Matrix des Tensorprodukt Smoothing Splines mit Kronecker-Struktur vorgestellt werden. Dafür wird als Grundlage das Prokrustes-Problem aus [4] bzw. [42] verwendet:

$$\min_X \|AX - I\|_F, \text{ mit } A, X \in \mathbb{R}^{n \times n}.$$

Die Matrix I entspricht der $n \times n$ Einheitsmatrix. Außerdem wird eine Variante des bereits erwähnten ALS-Konzeptes, inspiriert durch [3] und [36], mit dem Prokrustes-Problem kombiniert. An dieser Approximation einer Inversen ist neu, dass es sich um eine allgemeinere Umsetzung des Prokrustes-Problems mit Kronecker-Produkten handelt und eine Verwendung der ALS-Idee in einer solchen Form noch nicht bekannt ist. Bisher wurde entweder das Prokrustes-Problem ohne Kronecker-Produkte für orthogonale Matrizen oder nur für spezielle Fälle von Kronecker-Produkten betrachtet — zum Beispiel haben die Kronecker-Produkte nur zwei Faktoren und zusätzlich wird Forderung gestellt, dass die Faktoren orthogonal zueinander sind. Die Verwendung der ALS-Idee ist bisher nicht umgesetzt worden. An numerischen Tests wird deutlich gemacht, dass die in Kapitel 3 vorgestellte Komprimierung effektiv für die Bestimmung der Approximation der Inversen eingesetzt werden kann. Die Annäherung der Inversen kann für verschiedene Zwecke verwendet werden — zum Beispiel als Vorkonditionierer für die „Generalized minimal residual method“ (GMRES-Verfahren) oder zur Bestimmung des Glättungsparameters des Smoothing Splines bei dem Verfahren der „Generalized Cross Validation“.

In Kapitel 5 werden zwei mögliche Vorkonditionierer für das Verfahren der konjugierten Gradienten (CG-Verfahren) vorgestellt. Diese erscheinen effizienter als die in [24] verwendeten Skalierungsvorkonditionierer. Es handelt sich bei den beiden Vorkonditionierern zum einen um den SSOR-Vorkonditionierer,

ebenfalls bekannt als symmetrisches Gauß-Seidel-Verfahren, und zum anderen um einen selbstentwickelten Vorkonditionierer auf Grundlage des Prokrustes-Problems. Die Neuerung bei dem SSOR-Vorkonditionierer umfasst im Wesentlichen die Realisierung ohne Verletzung der Kronecker-Struktur, die Ausnutzung der Struktur des Tensorprodukt Smoothing Splines und eine dadurch bedingte effiziente Implementierung. Im Gegensatz zu Skalierungsvorkonditionierern nutzen die zuvor beschriebenen Vorkonditionierer mehr der gegebenen Informationen über die Matrix, wodurch theoretisch bessere Ergebnisse erzielt werden sollten. Bei dem Vorkonditionierer auf Grundlage des Prokrustes-Problems soll nur die Idee vorgestellt werden, da dieser nur testweise zum Einsatz kommt und bisher keine stabilen numerischen Ergebnisse liefert. Die numerischen Ergebnisse zu den einzelnen Forschungsbereichen sind in Kapitel 6 zu finden, woran sich Kapitel 7 mit der Zusammenfassung der Arbeit und einem Fazit anschließt. In Kapitel 8 wird ein Ausblick auf offene Forschungspunkte gegeben.

Diese Arbeit verbessert an wesentlichen Stellen die Verfahren aus [24] und liefert ein umfassendes Paket an Methoden für Tensorprodukt Smoothing Splines in höheren Dimensionen auf gestreuten Daten. Dabei ist irrelevant, woher die Daten kommen. Solange sie in der beschriebenen Form auftreten, können die Tensorprodukt Smoothing Splines darauf angewendet werden. Dies deutet das große Potential der vorzustellenden Verfahren an. Die Untersuchungen haben darüber hinaus unabhängig von den Tensorprodukt Smoothing Splines Verwendungsmöglichkeiten in anderen Bereichen, in welchen Summen von Kronecker-Produkten Anwendungsfälle besitzen. Dies ist zum Beispiel bei „Stochastic Automata Networks“ der Fall, siehe [26]. Weitere Anwendungen von Kronecker-Produkten sind in [43] oder [28] zu finden.

1.2. Neue Erkenntnisse der Arbeit

Die Neuerungen dieser Arbeit wurden bereits vereinzelt im letzten Abschnitt erwähnt. Sie sollen an dieser Stelle kurz hervorgehoben werden, damit deutlich wird, was die vorliegende Arbeit an neuen Erkenntnissen beinhaltet:

- Die Anwendung des Verfahrens aus 3.2 auf Tensorprodukt Smoothing Splines und die Verwendung der resultierenden Komprimierung wurden in dieser Art und Weise bisher noch nicht vorgestellt. Weiterhin sind die

Abschätzungen in den Sätzen 3.11 und 3.13 bzgl. der Effizienz der Komprimierung bei den genannten Splines eine neue wissenschaftliche Erkenntnis.

- Das Verfahren zur Approximation einer Kronecker-Inversen für eine Summe von Kronecker-Produkten in Kapitel 4 und die Verwendung dieser Approximation als Vorkonditionierer für das GMRES-Verfahren (siehe Kapitel 6.2) werden in dieser Arbeit zum ersten Mal präsentiert. Weiterhin ist die Spezialisierung des Verfahrens für Tensorprodukt Smoothing Splines und die Untersuchung des Verhaltens des Verfahrens in den Unterkapiteln 4.2 und 4.3 eine Neuentwicklung.

- In Kapitel 5 entspricht die Vorstellung einer Möglichkeit zur Anwendung des SSOR-Vorkonditionierers für Tensorprodukt Smoothing Splines auf gestreuten Daten einer neuen Idee. Weiterhin ist die Formulierung eines möglichen Vorkonditionierers für das CG-Verfahren basierend auf dem Prokrustes-Problem ein Ansatz, der bisher noch nicht verfolgt wurde.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der vorliegenden Arbeit erläutert. Die wesentlichen Bestandteile sind Kronecker-Produkte, Tensorprodukt Smoothing Splines, iterative Methoden zum Lösen linearer Gleichungssysteme sowie Multilinearformen bzw. Tensoren. Die Kronecker-Produkte werden für die Darstellung der Tensorprodukt Smoothing Splines benötigt und zur Bestimmung dieser Art von Splines benötigt man iterative Verfahren für das Lösen der auftretenden linearen Gleichungssysteme, welche die Kronecker-Struktur erhalten. Dies ist eine essentielle Anforderung an die iterativen Verfahren und daher müssen aus der Vielzahl an Methoden — siehe z.B. [35] — adäquate Verfahren ausgewählt werden. Die Zusammenhänge wurden bereits ausführlich in der Diplomarbeit [24] beschrieben und die folgenden Erläuterungen basieren größtenteils auf diesen Erkenntnissen, falls nicht anderweitig angegeben. In den einzelnen Unterkapiteln werden außerdem weitere Literaturquellen zur Ergänzung aufgeführt.

2.1. Kronecker-Produkte

Das Kronecker-Produkt ist das grundlegende Element dieser Arbeit, da ohne dieses eine praktische Umsetzung von Tensorpordukt Splines in höheren Dimensionen $p > 4$ mit einem handelsüblichen Computersystem nur für eine sehr geringe Anzahl an Datenpunkten möglich ist. Dies entspricht nicht den in der Realität auftretenden Problemstellungen. Im folgenden Abschnitt wird dieser Umstand insbesondere für Tensorprodukt Smoothing Splines sehr deutlich. Alle

im Kontext dieser Arbeit angewendeten Verfahren müssen auf das Kronecker-Produkt zugeschnitten sein. Die hier aufgeführten Informationen sind im Wesentlichen aus [43] und [33] entnommen. Das Kronecker-Produkt ist eine spezielle Multiplikation von Matrizen und Vektoren. Diese Multiplikation lässt sich auf höherdimensionale Strukturen erweitern, wie später zu sehen sein wird.

Definition 2.1 (Kronecker-Produkt) *Seien $A \in \mathbb{R}^{m\times n}$ und $B \in \mathbb{R}^{p\times q}$. Das Kronecker-Produkt $\otimes$ zweier Matrizen ist definiert durch*

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \mathbb{R}^{mp\times nq}.$$

Die Anwendung eines Kronecker-Produkts zeigt das folgende Beispiel:

Beispiel 2.2 *Seien*

$$A = \begin{bmatrix} 2 & 3 \\ 7 & 4 \\ 1 & 6 \end{bmatrix} \in \mathbb{R}^{3\times 2} \text{ und } B = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

dann gilt für das Kronecker-Produkt der beiden Matrizen

$$A \otimes B = \begin{bmatrix} 2 & 4 & 3 & 6 \\ 8 & 6 & 12 & 9 \\ 7 & 14 & 4 & 8 \\ 28 & 21 & 16 & 12 \\ 1 & 2 & 6 & 12 \\ 4 & 3 & 24 & 18 \end{bmatrix} \in \mathbb{R}^{6\times 4}.$$

Um eine für diese Arbeit entscheidende Eigenschaft des Kronecker-Produkts bei Matrix-Vektor-Multiplikationen aufzeigen zu können, muss der *vec*-Operator definiert werden. Dieser Operator strukturiert eine Matrix zu einer Spalte bzw. Zeile um. In dieser Arbeit wird immer von einer Spalte ausgegangen.

Definition 2.3 (*vec*-Operator) *Sei $A \in \mathbb{R}^{m\times n}$ und a_i die i-te Spalte von A mit $i = 1, \ldots, n$. Der* vec*-Operator ist definiert durch*

$$vec(A) := \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^{mn}.$$

Um eine eindeutige Inverse des Operators angeben zu können, werden zusätzliche Informationen benötigt. Es müssen die Dimensionen der Matrix angegeben werden, in welche der Vektor $vec(A)$ umstrukturiert werden soll. Sei $a \in \mathbb{R}^{mn}$, dann lässt sich die Inverse des *vec*-Operators von $a = vec(A)$ durch

$$vec^{-1}(a) = vec^{-1}(a, m, n) := vec^{-1}(vec(A)) = A$$

definieren. Ein Beispiel hilft bei der Veranschaulichung des Umgangs mit dem *vec*-Operator und dessen Umkehrfunktion:

Beispiel 2.4 *Sei die Matrix A wie in Beispiel 2.2 gewählt. Es ist*

$$vec(A) = a = \begin{pmatrix} 2 & 7 & 1 & 3 & 4 & 6 \end{pmatrix}^t \in \mathbb{R}^6.$$

Hier lässt sich erkennen, warum die Inverse des Operators nicht eindeutig ist, wenn die entsprechenden Dimensionen der Matrix nicht angegeben werden:

$$vec^{-1}(a, 2, 3) = \begin{bmatrix} 2 & 1 & 4 \\ 7 & 3 & 6 \end{bmatrix} \neq \begin{bmatrix} 2 & 3 \\ 7 & 4 \\ 1 & 6 \end{bmatrix} = vec^{-1}(a, 3, 2).$$

Der *vec*-Operator ist relevant, wenn ein Kronecker-Produkt mit einem Vektor multipliziert werden soll. In diesem Fall besitzt das Kronecker-Produkt eine Eigenschaft, welche das explizite Berechnen des Kronecker-Produkts vermeidet. Der nächste Satz zeigt diese Eigenschaft.

Satz 2.5 *Seien $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$ und $C \in R^{p\times q}$. Damit kann das Matrix-Produkt ABC berechnet werden und es gilt*

$$vec(ABC) = (C^t \otimes A) \cdot vec(B).$$

Beweis: Der Beweis ist in [43] bzw. [33] zu finden. □

Beispiel 2.6 *Seien A und B wie in Beispiel 2.2 und*

$$C = \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}.$$

Es gilt

$$C^t \otimes A = \begin{bmatrix} 4 & 6 & 2 & 3 \\ 14 & 8 & 7 & 4 \\ 2 & 12 & 1 & 6 \\ 6 & 9 & 4 & 6 \\ 21 & 12 & 14 & 8 \\ 3 & 18 & 2 & 12 \end{bmatrix}$$

und weiter

$$(C^t \otimes A) \cdot vec(B) = \begin{pmatrix} 41 & 72 & 70 & 68 & 121 & 115 \end{pmatrix}^t.$$

Mit

$$ABC = \begin{bmatrix} 41 & 68 \\ 72 & 121 \\ 70 & 115 \end{bmatrix}$$

lässt sich die Aussage aus Satz 2.5 erkennen.

Das Kronecker-Produkt hat weitere relevante Eigenschaften, welche kurz aufgeführt werden sollen.

Satz 2.7 *Seien $A \in \mathbb{R}^{m \times n},\ B \in \mathbb{R}^{p \times q},\ C \in \mathbb{R}^{n \times r}$ und $D \in \mathbb{R}^{q \times s}$. Es gilt*

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

Für quadratische Matrizen $Q \in \mathbb{R}^{n \times n}$ und $P \in \mathbb{R}^{m \times m}$ gilt

1. $\det(Q \otimes P) = \det(Q) \cdot \det(P)$,
2. $(Q \otimes P)^{-1} = Q^{-1} \otimes P^{-1}$, *falls Q und P invertierbar sind,*
3. *sowie* $\mathrm{tr}(Q \otimes P) = \mathrm{tr}(Q) \cdot \mathrm{tr}(P)$, *wobei die* $\mathrm{tr}(Q)$ *die Spur der Matrix Q kennzeichnet. Diese ist für quadratische Matrizen definiert durch*

$$\mathrm{tr}(Q) := \sum_{j=1}^{n} q_{jj}.$$

Beweis: Die Beweise zu den obigen Eigenschaften sind in [43] nachzulesen. □

Das Kronecker-Produkt ist in vielen Bereichen zu finden und aufgrund seiner Eigenschaften für die effiziente Speicherung von Matrizen und die Umsetzung von Matrix-Vektor-Multiplikationen bekannt. Es ist daher wünschenswert für jede Matrix eine Darstellung durch Kronecker-Produkte angeben zu können. Dies wird nicht in jedem Fall gelingen, allerdings wird in [32] gezeigt, wie eine minimale Approximation an eine Matrix durch ein Kronecker-Produkt mit zwei Faktoren berechnet werden kann. Das Minimum wird bezüglich der Frobeniusnorm

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\mathrm{tr}(A^tA)}, \text{ wobei } A \in \mathbb{R}^{m\times n},$$

bestimmt. Das Verfahren kann so erweitert werden, dass es eine Darstellung der Matrix durch eine Summe von Kronecker-Produkten mit zwei Faktoren ermöglicht. Diese Ideen sind im späteren Verlauf bei der Bestimmung einer Komprimierung in Kapitel 3 von Interesse. Deswegen soll die Vorgehensweise aus [32] vorgestellt werden. Das Ziel ist es, den Ausdruck

$$\|A - B \otimes C\|_F \tag{2.1}$$

bezüglich $B \in \mathbb{R}^{m_1\times n_1}$ und $C \in \mathbb{R}^{m_2\times n_2}$ zu minimieren, wobei $A \in \mathbb{R}^{m_1m_2\times n_1n_2}$ gegeben ist. Für das Verfahren wird die Matrix A in Blockmatrizen unterteilt

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1n_1} \\ \vdots & \ddots & \vdots \\ A_{m_11} & \cdots & A_{m_1n_1} \end{bmatrix}, \text{ mit } A_{ij} \in \mathbb{R}^{m_2\times n_2}. \tag{2.2}$$

Mit Hilfe dieser Darstellung lässt sich die Matrix $A \in \mathbb{R}^{m_1m_2\times n_1n_2}$ umstrukturieren zu

$$\mathcal{R}(A) := \mathcal{R}(A, m_1, n_1) = \begin{bmatrix} A_1 \\ \vdots \\ A_{n_1} \end{bmatrix} \text{ wobei } A_j = \begin{bmatrix} v(A_{1j})^t \\ \vdots \\ v(A_{m_1j})^t \end{bmatrix}, \ j = 1, \ldots, n_1. \tag{2.3}$$

Der Operator $\mathcal{R}(\cdot)$ ist ein linearer Operator. Dies ist einfach nachzurechnen und

wurde bereits in [45] gezeigt. Es gilt folglich für $\lambda \in \mathbb{R}$ und zwei Matrizen A, B

$$\mathcal{R}(\lambda \cdot A) = \lambda \mathcal{R}(A) \text{ sowie } \mathcal{R}(A + B) = \mathcal{R}(A) + \mathcal{R}(B).$$

Des Weiteren gilt $\mathcal{R}(A \otimes B) = v(A)v(B)^t$, was direkt mit der Definition des Kronecker-Produkts, der Unterteilung in (2.2) und der Umstrukturierung in (2.3) folgt.

Beispiel 2.8 *Sei*

$$A = \begin{bmatrix} 4 & 2 & 8 & 1 \\ 8 & 6 & 7 & 6 \\ 12 & 1 & 4 & 0 \\ 9 & 8 & 2 & 2 \end{bmatrix}.$$

Für die Umstrukturierung aus (2.3) *gilt*

$$\mathcal{R}(A) = \begin{bmatrix} 4 & 8 & 2 & 6 \\ 12 & 9 & 1 & 8 \\ 8 & 7 & 1 & 6 \\ 4 & 2 & 0 & 2 \end{bmatrix}.$$

Es lässt sich die Vektorisierung und Stapelung der Teilmatrizen von A erkennen.

Für das weitere Vorgehen wird der Begriff der Rang-1 Matrix benötigt. Eine Rang-1 Matrix $A \in \mathbb{R}^{n \times n}$ hat, wie der Name schon sagt, den Rang 1 und lässt sich insbesondere durch ein Produkt zweier Vektoren $A = a \cdot b^t$, mit $a, b \in \mathbb{R}^n$ darstellen. Diese effiziente Darstellung einer Matrix wird in der folgenden Arbeit eine wichtige Rolle spielen und lässt sich bei dem Minimierungsproblem aus (2.1) direkt ausnutzen. Mit Hilfe der Umstrukturierung kann dieses Problem als eine Approximation der Matrix $\mathcal{R}(A)$ durch eine Rang-1 Matrix aufgefasst werden. Der folgende Satz aus [32] veranschaulicht dies.

Satz 2.9 *Sei $A \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$. Wenn $B \in \mathbb{R}^{m_1 \times n_1}$ sowie $C \in \mathbb{R}^{m_2 \times n_2}$, dann gilt*

$$\|A - B \otimes C\|_F = \|\mathcal{R}(A) - v(B)v(C)^t\|_F. \tag{2.4}$$

Beweis: Die Aussage lässt sich durch Umformen der Norm beweisen und ist in [32] zu finden. □

Um eine minimale Rang-1 Approximation bestimmen zu können, wird das Konzept einer Singulärwert-Zerlegung (SVD) benötigt. Dieses wird aus [39] übernommen und ist ursprünglich in [15] zu finden.

Satz 2.10 *Zu jeder Matrix* $A \in \mathbb{R}^{m\times n},\ m,\ n \in \mathbb{N}$*, gibt es orthogonale Matrizen* $U \in \mathbb{R}^{m\times m}$*,* $V \in \mathbb{R}^{n\times n}$ *und eine Diagonalmatrix* $\Sigma \in R^{m\times n}$*, so dass gilt*

$$A = U\Sigma V^t, \quad mit\ \Sigma = \mathrm{diag}(\sigma_{11}, \ldots, \sigma_{\ell}) \tag{2.5}$$

und $\sigma_{11} \geq \sigma_{22} \geq \cdots \geq \sigma_{\ell\ell} \geq 0,\ \ell := \min(m, n)$.

Beweis: Siehe [39]. □

Definition 2.11 (Singulärwert-Zerlegung) *Die Darstellung in (2.5) wird als Singulärwert-Zerlegung (SVD) bezeichnet, die Diagonalwerte von* $\Sigma,\ \sigma_j = \sigma_{jj},\ j = 1, \ldots, \ell$*, werden Singulärwerte von* A *genannt.*

Jede Matrix $A \in \mathbb{R}^{m\times n}$ lässt sich mit Hilfe ihrer SVD $A = U\Sigma V^t$ durch eine Summe von Rang-1 Matrizen

$$A = \sum_{i=1}^{r} \sigma_j U(:, i)V(:, i)^t \text{ mit } \mathrm{rang}(A) = r,$$

darstellen. Die Bezeichnung $U(:, j)$ beschreibt die j-te Spalte der Matrix U und diese Notation gilt analog für die Zeilen. Wenn die Matrix A den Rang r besitzt, gilt für die Singulärwerte $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_\ell = 0$. Die Matrix A kann gegebenenfalls mit Hilfe der SVD komprimiert werden und diese ist für eine solche Anwendung bekannt. Die Darstellung durch Rang-1 Matrizen spielt bei der Komprimierung in dieser Arbeit ebenfalls eine wichtige Rolle, wie später zu sehen sein wird. Mit Satz 2.9 folgt, dass

$$\min_{B,C} \|A - B \otimes C\|_F = \min_{v(B),v(C)} \|\mathcal{R}(A) - v(B)v(C)^t\|_F. \tag{2.6}$$

Laut [32] lässt sich das Minimum des Terms in (2.6) mit Hilfe der Singulärwert-Zerlegung

$$\mathcal{R}(A) = U\Sigma V^t$$

bestimmen. In [32] wird weiter gezeigt, dass

$$\min_{v(B),v(C)} \|\mathcal{R}(A) - v(B)v(C)^t\|_F = \|\mathcal{R}(A) - \sigma_1 U(:, 1)V(:, 1)^t\|_F, \tag{2.7}$$

wobei σ_1 der größte Singulärwert von $\mathcal{R}(A)$ ist. Die Vektoren $U(:,1)$ und $V(:,1)$ sind die zugehörigen Singulärvektoren. Daraus lässt sich folgern, dass $v(B) = \sigma_1 U(:,1)$ sowie $v(C) = V(:,1)$. Die Matrizen B und C lassen sich durch eine Umstrukturierung bestimmen. Aufgrund der Eindeutigkeit der Singulärwerte ist das Minimum ebenfalls eindeutig. Die Darstellung ist allerdings nicht eindeutig, da der Singulärwert zum Beispiel anstelle von B mit C multipliziert werden kann. Die Güte der Approximation ist abhängig von dem Rang der Matrix $\mathcal{R}(A)$. Ist dieser größer 1, kann die Approximation $A \approx B \otimes C$ natürlich sehr ungenau sein.

Beispiel 2.12 *Sei A wie in Beispiel 2.8. Damit ist $\mathcal{R}(\mathcal{A})$ ebenfalls bekannt und die SVD von $\mathcal{R}(A)$ entspricht bedingt durch Rundungsfehler ungefähr*

$$\begin{aligned}\mathcal{R}(A) &= U\Sigma V^t \\ &\approx \begin{bmatrix} -0.435 & 0.857 & 0.268 & -0.038 \\ -0.712 & -0.389 & 0.008 & -0.584 \\ -0.514 & -0.062 & -0.543 & 0.660 \\ -0.198 & -0.327 & 0.795 & 0.470 \end{bmatrix} \begin{bmatrix} 23.81 & 0 & 0 & 0 \\ 0 & 4.13 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &\quad \begin{bmatrix} -0.638 & -0.583 & -0.088 & -0.495 \\ -0.736 & 0.551 & 0.306 & 0.245 \\ -0.164 & -0.506 & -0.148 & 0.833 \\ 0.155 & -0.315 & 0.936 & 0.005 \end{bmatrix}.\end{aligned}$$

Es lässt sich erkennen, dass $\mathcal{R}(A)$ den Rang 2 hat und damit die Matrix A nur ungefähr angenähert werden kann. Mit (2.7) und (2.6) folgt

$$\min_{v(B),v(C)} \|\mathcal{R}(A) - v(B)v(C)^t\|_F \approx 4.13 \approx \min_{B,C} \|A - B \otimes C\|_F$$

und

$$A \approx B \otimes C \approx 23.81 \cdot \begin{bmatrix} -0.435 & -0.712 \\ -0.514 & -0.198 \end{bmatrix} \begin{bmatrix} -0.638 & -0.736 \\ -0.164 & 0.155 \end{bmatrix}.$$

Die Approximation ist noch nicht ausreichend, was daran liegt, dass der Rang der Matrix $\mathcal{R}(A)$ größer 1 ist.

Falls die Approximation zufriedenstellend sein sollte, kann die Inverse von A

ebenfalls effizient angenähert werden, da nach Satz 2.7 gilt

$$A^{-1} \approx (B \otimes C)^{-1} = B^{-1} \otimes C^{-1}.$$

Die Näherung der Inversen kann als Vorkonditionierer für iterative Methoden zum Lösen von LGS verwendet werden. Ein Vorkonditionierer soll helfen, ein LGS stabiler und schneller zu lösen, aber später mehr dazu. Eine solche Verwendung ist zum Beispiel bei „Stochastic Automata Networks“ zu finden, siehe [26]. Die Vorgehensweise ist allerdings nicht in jedem Fall zielführend, da die Abschätzung von A nicht sehr genau sein muss und dies Auswirkung auf die Approximation der Inversen hat. Dies wird in [27] verdeutlicht und es ist im Grunde zu erwarten, da die Matrix $\mathcal{R}(A)$ in den meisten Fällen einen größeren Rang als 1 besitzt.

Bemerkung 2.13 *Wenn eine Matrix genau einem Kronecker-Produkt mit zwei Faktoren entspricht, dann lässt sich eine Darstellung der Matrix durch ein Kronecker-Produkt mit dem oben beschriebenen Verfahren finden. Wie bereits erwähnt, ist dieses Kronecker-Produkt nicht eindeutig. Die Darstellung durch ein Kronecker-Produkt hat nicht nur Speichervorteile — die Bestimmung einer LU-Zerlegung ist zum Beispiel ebenfalls effizienter umzusetzen, da nach Satz 2.7 gilt*

$$B \otimes C = (L_B U_B) \otimes (L_C U_C) = (L_B \otimes L_C)(U_B \otimes U_C).$$

Es müssen nur die LU-Zerlegungen der Matrizen B und C bestimmt werden und nicht die LU-Zerlegung der gesamten Matrix $B \otimes C$.

Um die Approximation zu verbessern, falls $\mathcal{R}(A)$ nicht Rang 1 hat, lässt sich ausnutzen, dass $\mathcal{R}(\cdot)$ ein linearer Operator ist. Die Darstellung von $\mathcal{R}(A)$ durch eine Summe von Rang-1 Matrizen kann in eine Darstellung von A durch eine Summe von Kronecker-Produkten übertragen werden. Es gilt

$$\left\| A - \sum_{j=1}^{r} B_j \otimes C_j \right\|_F^2 = \left\| \mathcal{R}(A) - \sum_{j=1}^{r} v(B_j) v(C_j) \right\|_F^2, \tag{2.8}$$

wobei $r = \operatorname{rang}(\mathcal{R}(A))$, $v(B_j) = \sigma_j U(:, j)$ und $v(C_j) = V(:, j)$. Da die Approximation kein reines Kronecker-Produkt ist, sondern eine Summe mehrerer Kronecker-Produkte, kann sie zur Annäherung der Inversen nicht mehr ausgenutzt werden. Deswegen wird in Kapitel 4 eine alternative Annäherung an eine

Inverse vorgestellt.

Beispiel 2.14 *Seien die Matrizen A und $\mathcal{R}(A)$ wie in Beispiel 2.8. Da $\mathcal{R}(A)$ Rang 2 hat, folgt*

$$\mathcal{R}(A) = \sigma_1 U(:,1)V(:,1)^t + \sigma_2 U(:,2)V(:,2)^t,$$

wobei die Vektoren $U(:,1)$, $U(:,2)$, $V(:,1)$ und $V(:,2)$ sowie die Werte σ_1 und σ_2 aus der SVD von $\mathcal{R}(A)$ stammen. Für die Darstellung von A ergibt sich damit

$$\begin{aligned} A \approx & \underbrace{23.81 \cdot \begin{bmatrix} -0.435 & -0.712 \\ -0.514 & -0.198 \end{bmatrix}}_{=:B_1} \underbrace{\begin{bmatrix} -0.638 & -0.736 \\ -0.164 & 0.155 \end{bmatrix}}_{=:C_1} \\ & + \underbrace{4.13 \cdot \begin{bmatrix} 0.857 & -0.389 \\ -0.062 & -0.327 \end{bmatrix}}_{=:B_2} \underbrace{\begin{bmatrix} -0.583 & 0.551 \\ -0.506 & -0.315 \end{bmatrix}}_{=:C_2} \end{aligned}$$

und weiter

$$\|A - (B_1 \otimes C_1 + B_2 \otimes C_2)\|_F \approx 0 \approx \|\mathcal{R}(A) - (\sigma_1 U(:,1)V(:,1)^t + \sigma_2 U(:,2)V(:,2)^t)\|_F.$$

Da in dieser Arbeit Kronecker-Produkte mit mehr als zwei Faktoren benötigt werden, ist die obige Vorgehensweise auf den ersten Blick nicht ausreichend. Durch mehrfache Anwendung dieser Vorgehensweise lässt sich allerdings eine Annäherung mit mehr Faktoren bestimmen, wie später gezeigt wird. Diese Bestimmung ist in dem untersuchten Fall praktisch nicht umsetzbar und deswegen muss ein alternatives Verfahren verwendet werden. Die wesentliche Idee für den Ansatz mit Kronecker-Produkten, welche mehr als zwei Faktoren besitzen, basiert aber auf der Vorgehensweise über die SVD. Diese wird im Kapitel 3 erneut aufgegriffen. Das Ziel wird sein, eine Matrix durch ein Kronecker-Produkt mit mehreren Faktoren darzustellen. Dafür wird die Matrix zu einem Tensor umstrukturiert und eine Zerlegung ermittelt, welche einer mehrdimensionalen SVD entspricht.

In [26] wird für einen Spezialfall eine alternative Möglichkeit aufgezeigt, wie eine Approximation durch ein Kronecker-Produkt mit mehr als zwei Faktoren bestimmt werden kann. Dabei ist die Matrix $A \in \mathbb{R}^{m \times n}$ bereits durch eine

Summe von Kronecker-Produkten gegeben

$$A = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij}, \text{ mit } A_{ij} \in \mathbb{R}^{m_j \times n_j} \text{ und } m = \prod_{j=1}^{p} m_j, \; n = \prod_{j=1}^{p} n_j. \tag{2.9}$$

Diese Struktur liegt bei der hier untersuchten Problemstellung ebenfalls vor, wie später zu sehen sein wird. Die Faktoren der Kronecker-Produkte zur Approximation von A werden durch eine Linearkombination der jeweiligen Faktoren der einzelnen Kronecker-Produkte aus der Summe in (2.9)

$$B = \sum_{i=1}^{N} b_i A_{i1}, \; C = \sum_{i=1}^{N} c_i A_{i2}, \ldots$$

bestimmt. Da die Norm $\|A - B \otimes C \otimes \cdots\|_F$ minimiert werden soll, entsprechen die Koeffizienten der Linearkombinationen den Variablen eines nichtlinearen Optimierungsproblems und die Variablenanzahl ist von der Summandenanzahl der Summe von Kronecker-Produkten abhängig. Dieses Optimierungsproblem ist in dem hier betrachteten Fall zu komplex, da die Anzahl der Variablen $N \cdot p$ entspricht. Die Anzahl beträgt in der untersuchten Problemstellung mindestens $10000 \cdot 3$, wie in [24] nachzuvollziehen ist. Die eigentliche Approximation kann auf diese Weise nicht effizient bestimmt werden, weswegen in Kapitel 3 ein alternatives Vorgehen vorgestellt wird.

Die gezeigten Eigenschaften und Möglichkeiten von Kronecker-Produkten sind für die später vorgestellten Resultate ausreichend. Kronecker-Produkte haben allerdings wesentlich mehr Potential. Eine ausführlichere Beschreibung von Kronecker-Produkten ist in [43] oder [33] zu finden.

2.2. Tensorprodukt Splines

Wie in der Einleitung beschrieben, soll eine Funktion f durch eine einfachere Funktion s approximiert werden. Die Funktion f ist durch eine Menge von Punkten

$$\{(X_i, y_i) \; : \; i = 1, \ldots, N \text{ und } X_i \in \mathbb{R}^p, \; y_i \in \mathbb{R}\}$$

gegeben und die Punkte X_i werden im Folgenden als Stützstellen bezeichnet. Die Approximation s von f lässt sich zum Beispiel durch das Interpolationsproblem

$$s(X_i) = y_i = f(X_i),\ i = 1, \ldots, n \tag{2.10}$$

bestimmen. Als Grundlage für s wird in dieser Arbeit das Konzept der Splines verwendet. Aufgrund der Mehrdimensionalität des betrachteten Problems werden Tensorprodukt Splines verwendet. Das Kronecker-Produkt steht im natürlichen Zusammenhang mit Tensorprodukten. Daher lassen sich Kronecker-Produkte zur effizienten Darstellung von Tensorprodukt Splines verwenden. Diese Form des multivariaten Splines setzt sich aus den univariaten Splines in den jeweiligen Dimensionen zusammen. Diese univariaten Splines entsprechen wiederum stückweisen Polynomen. Aufgrund des Tensorprodukts bestehen folglich die Komponenten des Tensorprodukt Splines aus Produkten der univariaten Komponenten und damit aus stückweise multivariaten Polynomen. Wie später gezeigt wird, ist dies effizient mit Kronecker-Produkten darstellbar. Die Tensorprodukt Knotenfolge lässt sich analog aus den univariaten Knotenfolgen aufstellen.

Dieser Abschnitt basiert auf [5] und [41]. Ausführlicher wurde der Zusammenhang zwischen Kronecker-Produkten und Tensorprodukt Splines in [24] bearbeitet. Um die obigen Begriffe genau definieren zu können, wird der Begriff des Multiindexes aus [41] eingeführt.

Definition 2.15 (Multiindex) *Ein Vektor $\alpha = (\alpha_1, \ldots, \alpha_p) \in \mathbb{N}^p$ von positiven ganzen Zahlen heißt* Multiindex. *Die Länge von α entspricht der Zahl*

$$|\alpha| = \sum_{j=1}^{p} \alpha_j.$$

Der Operator $\leq$ definiert eine Halbordnug auf den Multiindizes. Für zwei Multiindizes $\alpha, \beta \in \mathbb{N}^p$ gilt

$$\alpha \leq \beta \Leftrightarrow \alpha_i \leq \beta_i, \quad i = 1, \ldots, n.$$

Definition 2.16 (Tensorprodukt Knotenfolge) *Seien μ, ν und $\alpha \in \mathbb{N}^p$. In der j-ten Dimension wird die Knotenfolge*

$$T_j = T^{(j)}_{\mu_j,\nu_j} := \left\{t_{j,1}, \ldots, t_{j,\nu_j+\mu_j+1}\right\}$$

verwendet. An diese Folge werden bestimmte Anforderungen gestellt: Sie muss monoton fallend, $t_{j,1} \leq \cdots \leq t_{j,\mu_j+\nu_j+1}$, sein und die einzelnen Knoten sind in der Vielfalt beschränkt durch $t_{j,k} < t_{j,k+\mu_j}$. Die Tensorprodukt Knotenfolge *ist das Produkt der univariaten Knotenfolgen*

$$T = T_{\mu\nu} := \bigotimes_{j=1}^{p} T_j = \left\{t_\alpha = \left(t_{\alpha_1}, \ldots, t_{\alpha_p}\right) : \ \alpha \leq \nu + \mu + 1\right\},$$

wobei μ als der Multigrad des Tensorprodukt Splines bezeichnet wird und ν als die Anzahl der inneren Knoten.

Die Tensorprodukt Knoten liegen aufgrund des Tensorprodukts immer auf einem Gitter und ihre Anzahl wächst mit Vergrößerung der Dimension exponentiell. Weiterhin soll erwähnt werden, dass zwischen inneren Knoten und Randknoten unterschieden wird. Die Anzahl der Randknoten in der Richtung j beträgt $\mu_j + 1$, wobei μ_j den Grad des univariaten Splines bezeichnet. Diese Anzahl an Randknoten ist dafür verantwortlich, dass der Spline an den Randpunkten interpoliert. Die inneren Knoten können als die Schnittstellen der stückweisen Polynome des Splines aufgefasst werden und ihre Anzahl in der Richtung j beträgt ν_j. Beispielhaft sind verschiedene Fälle von Stützstellen und Tensorprodukt Knoten in Abbildung 2.1 zu sehen.

Die Knotenwahl spielt bei dem Verhalten des Splines eine große Rolle und soll an dieser Stelle kurz erläutert werden. Prinzipiell ist die Wahl der univariaten Knotenfolge T_j beliebig, solange die Voraussetzungen aus Definition 2.16 erfüllt sind. Die Knoten können zum Beipiel äquidistant gewählt werden. In dem hier betrachteten Problem hat dies aber nicht zu guten Ergebnissen geführt und es wird die Knotenwahl aus [24] verwendet. Dabei werden die Komponenten der Stützstellen in der j-ten Dimension der Größe nach geordnet und dann in ν_j Mengen der gleichen Mächtigkeit eingeteilt. Die Mittelwerte dieser ν_j Mengen werden als Knoten verwendet. Dadurch kann garantiert werden, dass sich die Knoten an die Verteilung der Stützstellen anpassen.

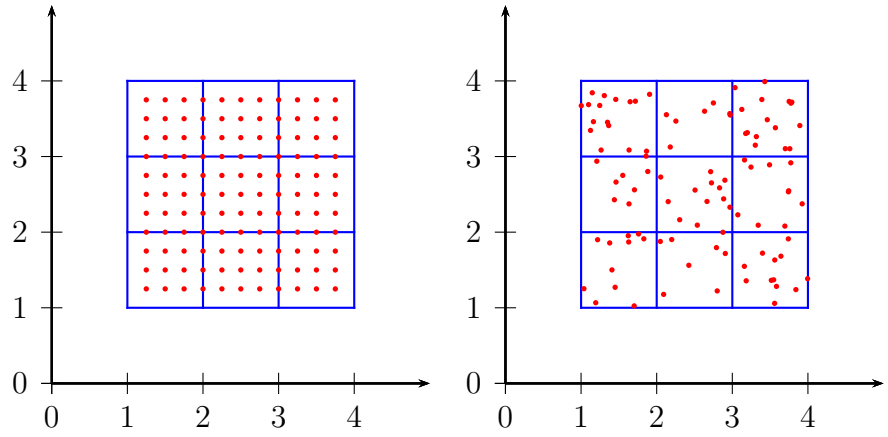

Abbildung 2.1.: Knoten und Stützstellen

Die linke Abbildung zeigt ein Gitter von zweidimensionalen Tensorprodukt Knoten (blau), mit $T_1 = \{1, 2, 3, 4\}$ *und* $T_2 = \{1, 2, 3, 4\}$*, und weiterhin Stützstellen (rot), welche ebenfalls auf einem Gitter liegen. Die rechte Abbildung zeigt dieselben Tensorprodukt Knoten, aber gestreute Stützstellen. Diese weisen keinerlei Struktur auf, was der vorliegenden Problemstellung aus [44] entspricht.*

Als nächstes wird der Aufbau von Tensorprodukt Splines erläutert. Diese lassen sich mit Hilfe von Tensorprodukt B-Splines umsetzen, wobei die mehrdimensionalen B-Splines analog zu den Knoten mit Hilfe der univariaten Komponenten dargestellt werden. Sie entsprechen dem Produkt von univariaten B-Splines.

Definition 2.17 (Tensorprodukt B-Splines) *Sei* χ *die* charakteristische Funktion

$$\chi_{[a,b]} := \chi_{[a,b]}(x) = \begin{cases} 1, & \textit{falls } x \in [a,b] \\ 0, & \textit{sonst.} \end{cases}$$

Der univariate B-Spline in der j*-ten Dimension lässt sich mit Hilfe der zugehörigen univariaten Knotenfolge durch folgende Rekursionsformel aufstellen*

$$N^0_{\kappa_j}(\cdot|T_j) := \chi_{[t_{j,i}, t_{j,i+1})},$$

$$N^{\mu_j}_{\kappa_j}(\cdot|T_j) := \frac{\cdot - t_{j,i}}{t_{j,i+\mu_j} - t_{j,i}} N^{\mu_j - 1}_{\kappa_j}(\cdot|T_j) + \frac{t_{j,i+\mu_j+1} - \cdot}{t_{j,i+\mu_j+1} - t_{j,i+1}} N^{\mu_j - 1}_{\kappa_j+1}(\cdot|T_j),$$

mit $i = 1, \ldots, \nu_j + \mu_j + 1$*. Der* Tensorprodukt B-Spline *ist das Produkt dieser*

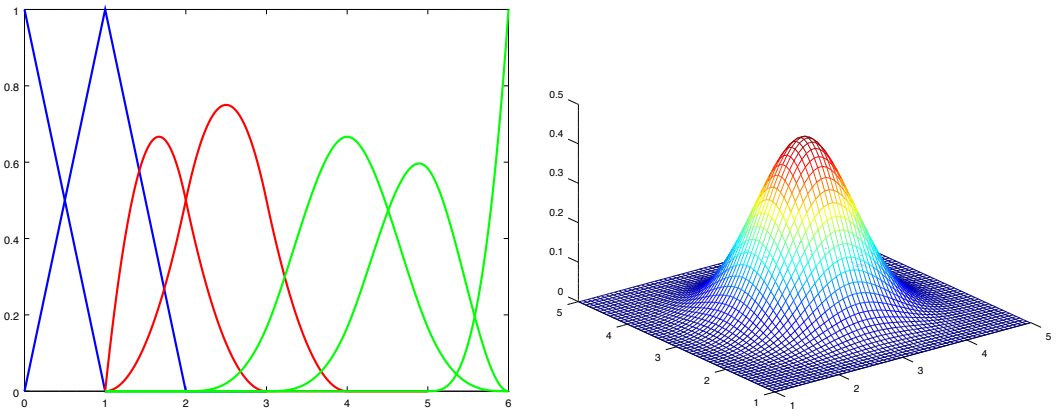

Abbildung 2.2.: B-Splines

Der linke Plot zeigt univariate B-Splines mit unterschiedlichen Graden. Die B-Splines auf der linken Seite (blau) haben Grad 1*, in der Mitte (rot) Grad* 2 *und die rechten B-Splines (grün) Grad* 3*. Die rechte Abbildung zeigt einen multivariaten B-Spline mit dem Grad* 2 *in jeder Dimension. Es lässt sich erkennen, wie sich dieser B-Spline aus dem Produkt der beiden univariaten B-Splines zusammensetzt.*

univariaten B-Splines

$$N_\kappa^\mu(\cdot|T) := \prod_{j=1}^{p} N_{\kappa_j}^{\mu_j}(\cdot|T_j).$$

Die univariaten B-Splines werden an der zugehörigen univariaten Komponente x_{ij} der Stützstelle X_j ausgewertet. Beispiele für B-Splines und ihre mehrdimensionale Variante sind in Abbildung 2.2 zu finden. Durch eine Linearkombination der Tensorprodukt B-Splines wird ein Tensorprodukt Spline konstruiert.

Definition 2.18 (Tensorprodukt Spline) *Sei $T_{\mu,\nu}$ eine Tensorprodukt Knotenfolge und $N_\kappa^\mu(\cdot|T_{\mu,\nu})$, $\kappa \leq \nu$, die zugehörigen Tensorprodukt B-Splines. Der* Tensorprodukt Spline *ist definiert durch*

$$s_\mu(\cdot) = s_\mu(\cdot|T_{\mu,\nu}) := \sum_{\kappa \leq \nu} d_\kappa N_\kappa^\mu(\cdot|T_{\mu,\nu}),$$

wobei $d_\kappa \in \mathbb{R}$ als die Kontrollpunkte und μ als der Grad des Splines bezeichnet werden.

In Abbildung 2.3 ist ein solcher Spline veranschaulicht. Die Kontrollpunkte wer-

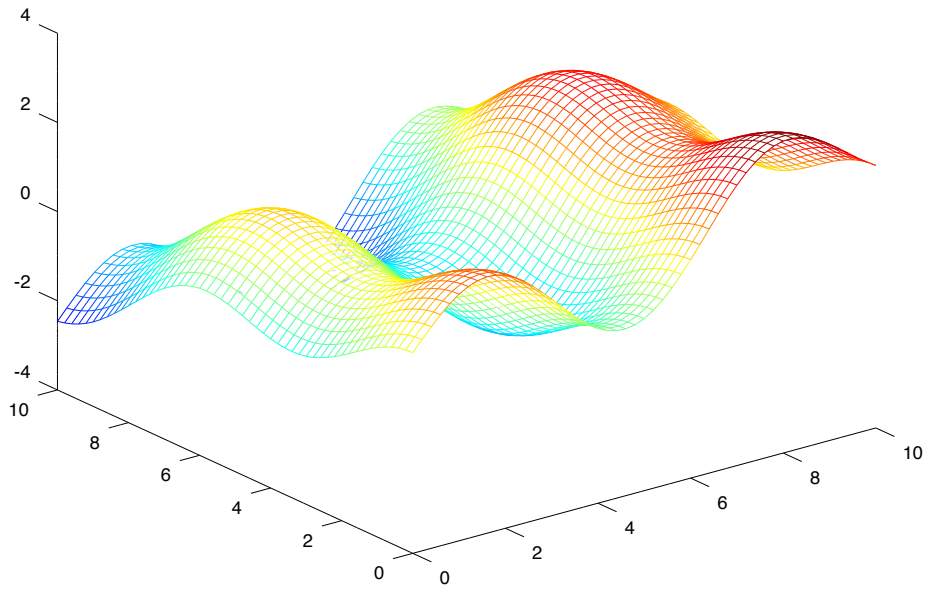

Abbildung 2.3.: Tensorprodukt Spline

Die Abbildung zeigt einen kubischen Tensorprodukt Spline, welcher die Funktion

$$f(x,y) = 1.2\sin(x) + 2\cos(y) + \left(\frac{x}{7}\right)^2 - \left(\frac{y}{7}\right)^2$$

auf dem Gebiet $[0,10] \times [0,10]$ *approximiert. Die Anzahl der Stützstellen beträgt in jeder Dimension* 13 *und diese werden äquidistant auf das Intervall verteilt. Die Anzahl der inneren Knoten beträgt* 13 *und die der Randknoten ist* 3*. Sie sind ebenfalls gleich verteilt. In [13] wird anschaulich erklärt, wie der Spline aufgebaut wird. Dies soll hier kurz wiedergegeben werden. Der dreidimensionale Tensorprodukt Spline setzt sich aus zwei univariaten Splines zusammen. Der eine Spline wird entlang des anderen Splines durch den Raum geführt, so dass eine Fläche entsteht, welche in der Abbildung zu sehen ist.*

den auch als Koeffizienten der B-Splines bezeichnet. Der Tensorprodukt Spline bzw. dessen Kontrollpunkte lassen sich für ein Interpolationsproblem

$$s_\mu(X_i) = y_i,\ i = 1, \ldots, N$$

durch ein lineares Gleichungssystem (LGS)

$$N^\mu(X|T)d = y \text{ mit } X = (X_1, \ldots, X_N) \text{ und } y = (y_1, \ldots, y_N)$$

bestimmen. Ein bekannter Satz von Schoenberg und Whitney macht eine Aussage darüber, welche Bedingungen für die Knoten und Stützstellen gelten müssen, damit eine eindeutige Lösung des LGS möglich ist. Er ist im Fall von Tensorprodukt Splines ins Mehrdimensionale übertragbar. Da dies prinzipiell nicht von Interesse ist, wird an dieser Stelle nicht weiter darauf eingegangen und es sei für nähere Informationen auf [5] sowie [41] verwiesen. Die Matrix des LGS entspricht einer Vandermonde-Matrix und lässt sich durch die Tensorprodukt B-Splines und die Stützstellen aufstellen

$$N^\mu(X|T) := \left[N_\kappa^\mu(x|T) : \begin{smallmatrix} x \in X \\ \kappa \leq \nu \end{smallmatrix}\right]. \tag{2.11}$$

Die Struktur dieser Matrix hängt von der Struktur der Stützstellen X und der Knoten ab. Die Knoten liegen, wie bereits erwähnt, in dem behandelten Fall auf einem Gitter. Wenn die Stützstellen ebenfalls auf einem Gitter liegen, lässt sich die Vandermonde-Matrix als Kronecker-Produkt der univariaten Vandermonde-Matrizen darstellen

$$N^\mu(X|T) = \bigotimes_{j=1}^{p} N^{\mu_j}(X_j|T_j).$$

In diesem Fall ist das lineare Gleichungssystem sehr schnell und effizient über die univariaten Matrizen lösbar, was bereits in [6] untersucht wurde. Oft liegen die Stützstellen allerdings nicht auf einem Gitter und daher ist eine solche Darstellung nicht möglich. Des Weiteren lässt sich die multivariate Vandermonde-Matrix praktisch nur in niedrigen Dimension explizit aufstellen, da der Speicheraufwand durch das Wachsen der Dimensionen exponentiell steigt und die Speicherkapazität eines herkömmlichen Computersystems in höheren Dimensionen nicht mehr ausreicht. In der verwendeten Programmiersprache `Octave` treten in vier Dimensionen bei 10 inneren Knoten in jede Richtung und 10000

Stützstellen zum Beispiel Matrizen in der Größenordnung von circa 760 Mb auf. Es ist nachvollziehbar, dass handelsübliche Computersysteme bei einer solchen Speicherauslastung schnell an ihre Grenzen stoßen.

Aufgrund der beschriebenen Problematik soll prinzipiell nur auf die univariaten Komponenten zugegriffen werden. Die Bandiertheit und die dünne Besetztheit der univariaten Vandermonde-Matrizen überträgt sich nicht auf die multivariate Vandermonde-Matrix. Diese besitzt aufgrund der gestreuten Daten keine ausnutzbare Struktur mehr. Lediglich die einzelnen Zeilen der multivariaten Vandermonde-Matrix lassen sich als Kronecker-Produkt der entsprechenden Zeilen der univariaten Vandermonde-Matrix darstellen, wie später im Detail zu sehen ist.

Die Tensorprodukt Splines bilden genau wie im Univariaten einen Vektorraum. Dieser wird im Folgenden Splineraum genannt. Zur Definition dieses Splineraums wird das Konzept der Vielfachheit eines Tensorprodukt Knotens benötigt. Die Vielfachheit φ eines Knotens $t_\kappa \in T$ lässt sich als vektorielle Größe analog zu [41] definieren durch

$$\varphi(t_\kappa) := (\varphi(t_{j,\kappa_j}) : \ j = 1, \ldots, p).$$

Damit ergibt sich folgende Definition des Splineraums in Anlehnung an [41]:

Definition 2.19 (Splineraum) *Der* Splineraum $\mathbb{S}_\mu(T)$ *der Tensorprodukt Splines vom Grad μ zur Knotenfolge T besteht aus der Menge aller Funktionen f, welche die beiden folgenden Eigenschaften erfüllen:*

1. *Die Funktion f muss auf dem Knoten-Intervall $\otimes_{j=1}^{p}(t_j, t_{j+1})$ einem Tensorprodukt Polynom der Ordnung $\kappa \leq \mu$ entsprechen*

$$f\Big|_{\otimes_{j=1}^{p}(t_j,t_{j+1})} \in \Pi_\mu \ \textit{mit}\ \Pi_\mu = \operatorname{span}\{x^\kappa \ : \ \kappa \leq \mu\}.$$

2. *Weiterhin muss die Funktion f eine gewisse Glattheit besitzen. D.h., an den Knoten der Vielfachheit $\varphi(t_\kappa)$ müssen alle partiellen Ableitungen*

$$\frac{\partial^{|\alpha|} f}{\partial x^\alpha} \ \textit{mit}\ \alpha \leq \mu - \varphi(t)$$

existieren und stetig sein. Diese Bedingung muss ebenfalls an den Übergängen der einzelnen Tensorprodukt Polynome erfüllt sein.

Ein Satz von Curry und Schoenberg besagt für den univariaten Fall, dass die B-Splines eine Basis des Splineraums bilden. Dies lässt sich in dem Fall von Tensorprodukt Splines ins Mehrdimensionale übertragen. Weitere Details dazu sind in [5] und [41] zu finden. Der Splineraum wird für das Konzept des Smoothing Splines benötigt und dieses wird aus dem folgenden Grund in Betracht gezogen: Das vorliegende Problem ist in vielen Fällen unterbestimmt, da die Anzahl der Knoten wesentlich größer ist als die Anzahl der Stützstellen. Ein möglicher Grund hierfür ist das nicht exponentielle Mitwachsen der Anzahl der Stützstellen, wenn die Dimension erhöht wird. Dies liegt daran, dass die Daten nicht auf einem Gitter liegen, sondern gestreut sind. Weiterhin sind die Daten verrauscht, wodurch eine Interpolation nicht sinnvoll ist. In diesem Fall macht es Sinn auf eine Interpolation zu verzichten und die Daten zu approximieren. Hier bietet sich die Verwendung eines Smoothing Splines an.

Definition 2.20 (Tensorprodukt Smoothing Spline) *Sei $\mathbb{S}_\mu(T)$ der Splineraum zum Multigrad μ und Knotenfolge T. Ein* Tensorprodukt Smoothing Spline *minimiert das Funktional*

$$\min_{s_\mu \in \mathbb{S}_\mu(T)} \sum_{i=1}^{N} |y_i - s_\mu(x_i)|^2 + \lambda \int_{\mathbb{R}^p} \|H(s_\mu)\|_F^2 dx_1 \cdots dx_p,$$

wobei $H(s)$ ein Differentialoperator ist und $\lambda \geq 0$ als Glättungsparameter bezeichnet wird.

Der erste Summand des Funktionals entspricht der Approximationsgüte und der zweite der Glattheit des Smoothing Splines. Der Glättungsparameter reguliert das Verhältnis der beiden zueinander. Wenn er gegen Unendlich geht, handelt es sich bei dem Smoothing Spline um eine Ausgleichshyperebene. Wenn er gegen Null geht, entspricht der Spline einem Interpolationsspline, falls eine Interpolation möglich ist. Ist dies nicht der Fall, wird ein „Least-squares“ Spline bestimmt. Veranschaulicht ist das Prinzip des Smoothing Splines in Abbildung 2.4. Bei der Wahl des Differentialoperators ist zu beachten, dass dessen Kern und der Kern des „Least-squares“ Problems nur eine triviale Schnittmenge haben dürfen, damit das Minimierungsproblem nicht singulär ist. In den bisherigen Anwendungen hat sich der folgende Differentialoperator für ein $s_\mu \in \mathbb{S}_\mu(T)$

$$\|H(s_\mu)\|_F^2 := \sum_{j=1}^{p} \sum_{k=1}^{p} \left(\frac{\partial^2}{\partial x_j \partial x_k} s_\mu(x) \right)^2$$

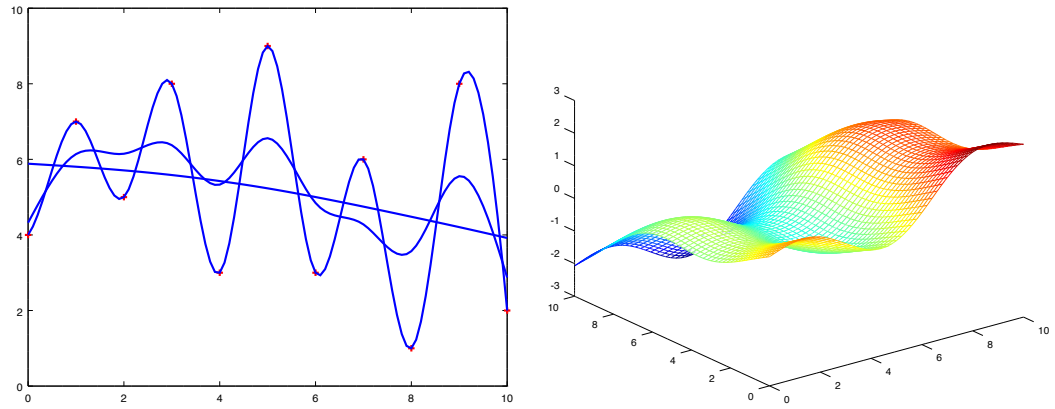

Abbildung 2.4.: Tensorprodukt Smoothing Spline

Die linke Grafik zeigt drei univariate Smoothing Splines mit unterschiedlichen Glättungsparametern. Dabei ist zu erkennen, dass bei $\lambda = 0$ der Smoothing Spline einem interpolierenden Spline entspricht. Um so größer der Glättungsparameter ist, um so mehr nähert sich der Smoothing Spline einer Ausgleichsgeraden an.

Die rechte Abbildung zeigt einen dreidimensionalen Smoothing Spline, welcher die Funktion aus Abbildung 2.3 approximiert. Es ist deutlich zu erkennen, dass der Smoothing Spline wesentlich glatter als der Spline ist. Man kann sich einen dreidimensionalen Smoothing Spline als biegsame Platte vorstellen, welche entweder durch bestimmte Punkte im Raum laufen muss oder diese aufgrund der Biegsamkeit (Glattheit) vernachlässigen darf.

als geeignet erwiesen. Dieser entspricht der Frobeniusnorm der Hesse-Matrix. Für die Bestimmung des Tensorprodukt Smoothing Splines müssen die Kontrollpunkte d berechnet werden. Das lässt sich durch ein lineares Gleichungssystem realisieren

$$\left(N^{\mu}(X|T)^t N^{\mu}(X|T) + \lambda G^{\mu}(T)\right) d = N^{\mu}(X|T)^t y, \tag{2.12}$$

wobei die Matrix $G^{\mu}(T)$ einer Gram-Matrix entspricht und für den Glättungsterm steht. Diese Gram-Matrix besteht aus einer Summe von Kronecker-Produkten

$$G^{\mu}(T) := \sum_{\kappa} \bigotimes_{j=1}^{p} G_{\kappa_j}^{\mu_j}(T_j),$$

wobei

$$G_{\kappa_j}^{\mu_j}(T_j) := \left[\int_{\mathbb{R}} \frac{d}{dx^{\kappa_j}} N_i^{\mu_j}(x|T_j) \frac{d}{dx^{\kappa_j}} N_k^{\mu_j}(x|T_j) dx : \ i,k = 1,\ldots,\nu_j \right]$$

für eine Gram-Matrix mit den univariaten Komponenten steht.

Bemerkung 2.21 *Im Folgenden ist an einigen Stellen vereinfacht von der Matrix des Tensorprodukt Smoothing Splines die Rede, damit ist immer die Matrix des LGS bzgl. der Bestimmung des Smoothing Splines in* (2.12) *gemeint.*

Aufgrund der gestreuten Stützstellen lässt sich die Vandermonde-Matrix, wie bereits erwähnt, nicht mehr in ein Kronecker-Produkt zerlegen. Da die Knoten auf einem Gitter liegen, sind zumindestens die Zeilen der Vandermonde-Matrix in ein Kronecker-Produkt zerlegbar

$$N^{\mu}(X|T) = \left[\bigotimes_{j=1}^{p} N^{\mu_j}(x_j|T_j) : \ x \in X \right],$$

der Ausdruck $N^{\mu_j}(x_j|T_j)$ steht für eine Zeile der entsprechenden univariaten Vandermonde-Matrix. Für eine alternative Darstellung dieser Matrix soll das Khatri-Rao-Produkt eingeführt werden.

Definition 2.22 (Khatri-Rao-Produkt) *Seien $A \in \mathbb{R}^{n\times k}$ und $C \in \mathbb{R}^{m\times k}$ Matrizen. Das* Khatri-Rao-Produkt *ist definiert als das spaltenweise Kronecker-Produkt zweier Matrizen mit der gleichen Spaltenzahl*

$$A \odot C := \begin{bmatrix} a_1 \otimes c_1 & \cdots & a_k \otimes c_k \end{bmatrix} \in \mathbb{R}^{nm\times k}.$$

Beispiel 2.23 *Gegeben seien die Matrizen*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \ \textit{sowie} \ B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

Für das Khatri-Rao-Produkt gilt

$$A \odot B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \odot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 7 & 16 \\ 15 & 24 \\ 21 & 32 \end{bmatrix}.$$

Die transponierte multivariate Vandermonde-Matrix lässt sich durch das Khatri-Rao-Produkt der transponierten univariaten Vandermonde-Matrizen darstellen:

$$N^{\mu}(X|T)^t = N^{\mu_1}(X_1|T_1)^t \odot \cdots \odot N^{\mu_p}(X_p|T_p)^t.$$

Es wird in [24] gezeigt, dass die Matrix $N^{\mu}(X|T)^t N^{\mu}(X|T)$ in eine Summe mit $\#X$ Kronecker-Produkten zerlegt werden kann

$$N^{\mu}(X|T)^t N^{\mu}(X|T) = \sum_{i=1}^{\#X} \bigotimes_{j=1}^{p} N^{\mu_j}(x_{ij}|T_j)^t N^{\mu_j}(x_{ij}|T_j). \tag{2.13}$$

Die Faktoren des Kronecker-Produkts sind symmetrische Rang-1 Matrizen, welche aus den Zeilen der univariaten Vandermonde-Matrizen berechnet werden. Insgesamt müssen folglich nur die univariaten Vandermonde- und Gram-Matrizen gespeichert werden. Die Matrix $N^{\mu}(X|T)^t N^{\mu}(X|T)$ ist im Gegensatz zur multivariaten Vandermonde-Matrix in (2.11) wieder bandiert, wobei diese Matrix mehrere Bänder um die Hauptdiagonale besitzt. Die Matrix ist gewissermaßen gestreift, was in Abbildung 2.5 zu erkennen ist. Diese Struktur lässt sich leider nicht ausnutzen, da zuviele Einträge von Null verschieden sind und der Speicherbedarf in höheren Dimensionen für eine explizite Speicherung zu groß wird. Aufgrund dessen sollte auf die obige Darstellung mit Kronecker-Produkten zurückgegriffen werden. Die Matrix des linearen Gleichungssystems in (2.12) lässt sich folglich als Summe von Kronecker-Produkten schreiben

$$Ax = \left(\sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} \right) x = b.$$

Das LGS ist effizient mit iterativen Verfahren lösbar, welche die Eigenschaften der Matrix-Vektor-Multiplikation bei Kronecker-Produkten aus Satz 2.5 ausnutzen. Die Anzahl der Summanden bewegt sich allerdings in der Größenordnung der Stützstellenanzahl. Da der Approximationsterm die meisten Summanden besitzt und nur aus Kronecker-Produkten von Rang-1 Matrizen besteht, ist diese große Anzahl an Summanden in bestimmten Fällen ineffizient. Aufgrund dessen wird in Kapitel 3 eine Komprimierungsmöglichkeit vorgestellt. Weiterhin müssen alle zusätzlichen Erweiterungen bzgl. der Bestimmung eines Splines eine Kronecker-Struktur besitzen, da es ansonsten zu Engpässen bei der Speicherkapazität kommen kann.

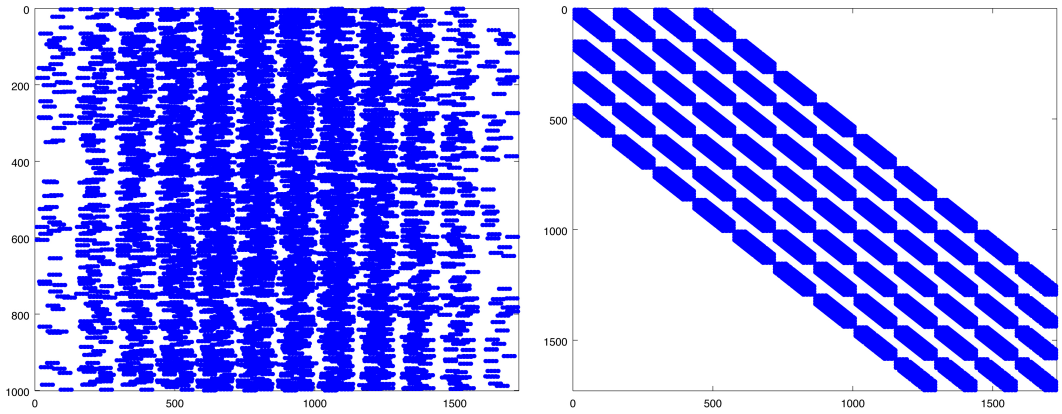

Abbildung 2.5.: Besetztheit der multivariaten Vandermonde-Matrix und der Matrix des Approximationsanteils

In der linken Abbildung ist ein Beispiel für die Besetztheit der multivariaten Vandermonde-Matrix $N^{\mu}(X|T)$ mit den vorliegenden Daten aus der Problemstellung der Chemie zu sehen. Es ist deutlich zu erkennen, dass keine Struktur vorhanden ist, da die Daten gestreut sind.

Auf der rechten Seite ist die Besetztheit der Matrix $N^{\mu}(X|T)^t N^{\mu}(X|T)$ des Approximationsteils des Smoothing Splines dargestellt. Sie weist eine deutliche Bandstruktur auf und ist außderdem symmetrisch.

2.3. Iterative Verfahren zum Lösen von linearen Gleichungssystemen

Der letzte Abschnitt verdeutlicht, dass die Bestimmung eines Tensorprodukt Smoothing Splines sehr aufwendig und eine naive Herangehensweise nicht zielführend ist. Zur Bestimmung des Splines muss ein lineares Gleichungssystem gelöst werden, auf dessen Koeffizientenmatrix nur als Summe von Kronecker-Produkten zugegriffen werden kann. Dieser Umstand muss bei der Wahl eines Lösungsverfahrens beachtet werden. Es kommen somit keine Splitting-basierten Verfahren wie das Jacobi- oder das Gauß-Seidel-Verfahren in Frage, da diese die Matrix aufteilen und bei ihnen die Kronecker-Struktur nicht erhalten bleibt. Die Verfahren benötigen folglich Zugriff auf die gesamte Matrix oder die einzelnen Elemente müssen für jeden Rechenschritt neu berechnet werden, was unverhältnismäßig viele Rechenoperationen erfordern würde. Solche Splitting-

basierten Verfahren lassen sich aber durchaus zur Vorkonditionierung einsetzen, wie später zu sehen sein wird. Weitere Informationen sind in [35] bzw. [21] zu finden.

Die Eigenschaft von Kronecker-Produkten bei Matrix-Vektor-Multiplikationen aus Satz 2.5 ist von großer Bedeutung beim Lösen des LGS und sollte ausgenutzt werden. Der Aufbau des ausgewählten Verfahrens sollte daher auf einer solchen Multiplikation basieren. Es bietet sich das Verfahren der konjugierten Gradienten (CG-Verfahren) an, da die Matrix des linearen Gleichungssystems in (2.12) symmetrisch und aufgrund der multivariaten Gram-Matrix positiv definit ist. Dabei ist zu beachten, dass die Matrix ihre positive Definitheit verlieren kann, falls für den Glättungsparameter $\lambda = 0$ gilt. In diesem Fall ist die Matrix allerdings immer noch positiv semidefinit.

Aufgrund der Problematik mit der positiven Definitheit soll zusätzlich die „Generalized minimal residual method“ (GMRES-Verfahren) betrachtet werden, da diese ebenfalls auf einer Matrix-Vektor-Multiplikation basiert. Das GMRES-Verfahren hat zwar eine schlechtere Konvergenzgeschwindigkeit im Vergleich zum CG-Verfahren, allerdings lassen sich für das GMRES-Verfahren mehr Anwendungsfälle finden. Dies liegt daran, dass es weniger Voraussetzungen benötigt. Die Matrix des LGS muss lediglich regulär sein. Weiterhin ist es einfacher einen Vorkonditionierer für das GMRES-Verfahren zu bestimmen. Die Vorkonditionierer für die jeweiligen Verfahren müssen ebenfalls die Kronecker-Struktur erhalten, weswegen am Ende dieses Abschnitts kurz das Prinzip der Vorkonditionierung erläutert wird und die Verfahren an die Verwendung von Vorkonditionierern angepasst werden.

2.3.1. CG-Verfahren & GMRES-Verfahren

Im Folgenden wird das Gleichungssystem

$$Ax = b \text{ mit } A = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} \in \mathbb{R}^{n \times n} \text{ mit } A_{ij} \in \mathbb{R}^{n_j \times n_j},\ b \in \mathbb{R}^n$$

betrachtet. Die Matrix A soll aufgrund des hohen Speicherbedarfs für große N und p nicht im Ganzen verwendet werden, sondern es soll nur auf die Faktoren der Kronecker-Produkte zugegriffen werden. Die Verfahren werden lediglich kurz vorgestellt. Die Theorie hinter den Verfahren und weitere Informationen sind in

Algorithmus 2.1 : CG-Verfahren

Input : $A \in \mathbb{R}^{n \times n}$ symmetrisch und positiv definit, $b \in \mathbb{R}^{n \times n}$
Output : Lösungsvektor x

begin
  $x_0 \in \mathbb{R}^n$,
  $p_0 := r_0 := b - Ax_0$,
  $\alpha_0 := r_0^t r_0,\ j := 0$
  while $\alpha_j \neq 0$ **do**
    $v_j := Ap_j,\ \lambda_j := \frac{\alpha_j}{v_j^t p_j}$
    $x_{j+1} := x_j + \lambda_j p_j$
    $r_{j+1} := r_j - \lambda_j v_j$
    $\alpha_{j+1} := r_{j+1}^t r_{j+1}$
    $p_{j+1} := r_{j+1} + \frac{\alpha_{j+1}}{\alpha_j} p_j$
    $j := j + 1$
  end
  $x := x_{j+1}$
end

[35], [21] und [15] zu finden. Das CG-Verfahren setzt voraus, dass die Matrix A symmetrisch und positiv definit ist. Die Vorgehensweise von der Methode ist in Algorithmus 2.1 beschrieben.

Die Matrix-Vektor-Multiplikation Ap_j ist bei dem CG-Verfahren die kritische Stelle. Sie lässt sich mit der Eigenschaft des Kronecker-Produkts aus Satz 2.5 realisieren. Da dies die einzige Operation ist, bei welcher die Kronecker-Struktur berücksichtigt werden muss, lässt sich das Verfahren effizient umsetzen. Die Voraussetzungen für das GMRES-Verfahren sind wesentlich schwächer. Es reicht, wie bereits erwähnt, dass die Matrix des LGS regulär ist. Dafür benötigt das GMRES-Verfahren in der Regel länger zum Lösen des LGS wie das CG-Verfahren. Dieser Nachteil kann aber stellenweise durch die schnellere Bestimmung eines Vorkonditionierers und dessen Verwendung kompensiert werden. An dieser Stelle soll kurz das Verfahren vorgestellt werden. Weitere Erläuterungen sind in [35] zu finden, woraus das Verfahren übernommen wurde.

Es wird das modifizierte GMRES-Verfahren mit “Restarts” verwendet. Das einfache GMRES-Verfahren baut eine orthogonale Basis auf und sucht darin den Lösungsvektor. Bei der Methode mit „Restarts“ wird nach einer Anzahl von Iterationen diese Basis verworfen, wenn der berechnete Lösungsvektor nicht die Toleranz erfüllt. Das Verfahren wird daraufhin erneut gestarten, wobei der bis-

her berechnete Lösungsvektor als Initialisierung für den neu zu berechneten Lösungsvektor verwendet wird. Es wird erneut eine Orthogonalbasis aufgebaut und das Verfahren setzt sich fort, solange die maximale Anzahl an Restarts noch nicht erreicht ist. Das Verfahren ist im Algorithmus 2.2 aufgeführt.

Nachteilig bei der Verwendung des GMRES-Verfahrens sind das zweimalige Auftreten einer Matrix-Vektor-Multiplikation und der erhöhte Speicherbedarf. Das Speichern der Basisvektoren v_j, $j = 1, \ldots, m$, kann dabei durchaus zu Speicherengpässen führen, da die Anzahl der zu speichernden Vektoren der Anzahl der Iterationen entspricht. Dies sollte bei der Implementierung beachtet werden. Weiterhin lässt sich bei genügend Speicherkapazität eine der beiden Matrix-Vektor-Multiplikationen vermeiden, indem die Vektoren $\tilde{v}_j = Av_j$ und $\tilde{x} = Ax_0$ gespeichert werden.

Die beiden vorgestellten Verfahren sind iterative Verfahren. Sie liefern allerdings nach n Schritten ein exaktes Ergebnis, wenn symbolisch gerechnet wird. Aufgrund von Rundungsfehlern ist dies in der Praxis nicht gegeben. Um die Anzahl der Rechenschritte zu verringern bzw. die Konvergenz zu verbessern, soll im nächsten Abschnitt das Prinzip der Vorkonditionierung erläutert werden.

2.3.2. Vorkonditionierung

Um die Idee der Vorkonditionierung zu erläutern, wird zuerst die Konditionszahl einer regulären Matrix definiert. Diese stammt aus [21].

Definition 2.24 (Konditionszahl) *Seien $\|\cdot\|$ eine beliebige Matrixnorm und $A \in \mathbb{R}^{n\times n}$ eine reguläre Matrix. Dann heißt*

$$\kappa(A) := \|A\|\|A^{-1}\|$$

die Konditionszahl *von A.*

Die Konditionszahl einer Matrix beschreibt, wie stark sich die Lösung eines LGS bei einer Störung ändert. Sie lässt sich als ein Maß für die Störungsanfälligkeit der Matrix auffassen.

Beispiel 2.25 *Sei*

$$A = \begin{bmatrix} 35 & 3 & -40 \\ 3 & 45 & -58 \\ -40 & -58 & 114 \end{bmatrix}.$$

Algorithmus 2.2 : GMRES-Verfahren

Input : $A \in \mathbb{R}^{n\times n}$ regulär, $b \in \mathbb{R}^n$, max. Anzahl von Restarts, max. Iter. m

Output : Lösungsvektor x

begin
  Wähle $x_0 \in \mathbb{R}^n$, $\varepsilon > 0$, $restart := 0$ und $r_0 := b - Ax$
  if $r_0 \neq 0$ **then**
    repeat
      $v_1 := \frac{r_0}{\|r_0\|_2},\ \gamma_1 := \|r_0\|_2$
      for $j = 1 : m$ **do**
        for $i = 1 : j$ **do**
          $h_{ij} := v_i^t A v_j$
        end
        $w_j := Av_j - \sum_{i=1}^{j} h_{ij} v_i,\ h_{j+1,j} := \|w_j\|_2$
        for $i = 1 : j-1$ **do**
          $\begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix} := \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix} \begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix}$
        end
        $\beta := \sqrt{h_{jj}^2 + h_{j+1,j}^2},\ s_j := \frac{h_{j+1,j}}{\beta},\ c_j := \frac{h_{jj}}{\beta}$
        $h_{jj} := \beta,\ \gamma_{j+1} := -s_j\gamma_j,\ \gamma_j := c_j\gamma_j$
        if $\gamma_{j+1} \leq \varepsilon$ **then**
          for $i = j : 1$ **do**
            $\alpha_i := \frac{1}{h_{ii}}(\gamma_i - \sum_{k=i+1}^{j} h_{ik}\alpha_k)$
          end
          $x := x_0 + \sum_{i=1}^{j} \alpha_i v_i$
          STOP
        else
          $v_{j+1} := \frac{w_j}{h_{j+1,j}}$
        end
      end
      for $i = m : 1$ **do**
        $\alpha_i := \frac{1}{h_{ii}}(\gamma_i - \sum_{k=i+1}^{j} h_{ik}\alpha_k)$
      end
      $x := x_0 + \sum_{i=1}^{m} \alpha_i v_i$
      $x_0 := x,\ r_0 := b - Ax_0$
      $restarts := restarts + 1$
    until *restart* $\leq$ *max. Anzahl von Restarts*;
  end
  $x := x_0$
end

Die Konditionszahl $\kappa(A)$ wird mittels der Maximumsnorn berechnet und es ergibt sich

$$\kappa(A) = \|A\|_\infty \|A^{-1}\|_\infty = 492.21.$$

Das Ziel der Vorkonditionierung ist die Verringerung der Konditionszahl der Matrix des Gleichungssystems, so dass das iterative Verfahren schneller konvergiert. Ein Vorkonditionierer darf die Lösung selbstverständlich nicht verändern, soll aber die Störungsanfälligkeit der Matrix reduzieren. Die Ideen, Definitionen und Algorithmen sind aus [2], [35] sowie [21] entnommen.

Definition 2.26 (Vorkonditioniertes System) *Seien P_L und $P_R \in \mathbb{R}^{n\times n}$ reguläre Matrizen. Dann heißt*

$$P_L A P_R x_P = P_L b \text{ mit } x = P_R x_P$$

das zu $Ax = b$ gehörige vorkonditionierte System.

Dabei wird P_R als der Rechtsvorkonditionierer bezeichnet, wenn $P_R \neq I$. Die Einheitsmatrix hat die bestmögliche Konditionszahl $\kappa(I) = 1$ und daher ist es wünschenswert, dass bei der Vorkonditionierung

$$P_L A \approx I,\ AP_R \approx I \text{ bzw. } P_L A P_R \approx I$$

als Resultat erzielt wird. Eine direkte Invertierung der Matrix A ist numerisch nicht sinnvoll und wäre bei dem vorliegenden Problem auch nicht effektiv umsetzbar. Die Berechnung eines Vorkonditionierers muss möglichst schnell sein, damit seine Anwendung überhaupt von Vorteil ist. Des Weiteren müssen die Vorkonditionierer für das vorliegende Problem immer eine Kronecker-Struktur aufweisen, da ansonsten keine Verwendung möglich ist. Für das GMRES-Verfahren wird in Kapitel 4 ein Vorkonditionierer der Form

$$P_L A \approx I \text{ bzw. } AP_R \approx I$$

vorgestellt. Die Matrizen P_L und P_R müssen lediglich regulär sein und die geforderte Kronecker-Struktur besitzen. Der Vorkonditionierer kann in den Algorithmus des GMRES-Verfahrens integriert werden, indem die Matrix-Vektor-Multiplikation Av_j durch $P_L A v_j$ bzw. $AP_R v_j$ ersetzt wird. Der Vektor b oder der Lösungsvektor x_P müssen entsprechend mit P_L oder P_R multipliziert werden. Die Anpassung des GMRES-Verfahren ist somit sehr einfach und bis auf

die Berechnung des Vorkonditionierers muss kaum ein zusätzlicher Aufwand betrieben werden, da die Anzahl der Rechenoperationen nur um eine Matrix-Vektor-Multiplikation erhöht wird. Diese Matrix-Vektor-Multiplikation ist bei dem später vorgestellten Vorkonditionierer wesentlich schneller zu berechnen als die eigentliche Multiplikation Ax. Dies liegt daran, dass die Matrix des Vorkonditionierers zwar eine Summe von Kronecker-Produkten ist, diese aber in der Regel sehr viel weniger Summanden als A besitzt.

Bei der Vorkonditionierung für das CG-Verfahren muss neben der Regularität und der Kronecker-Struktur beachtet werden, dass der Vorkonditionierer die Symmetrie und die positive Definitheit erhält, da ansonsten das CG-Verfahren nicht mehr angewendet werden kann. Die Symmetrie und die positive Definitheit können allerdings zur Bestimmung des Vorkonditionierers ausgenutzt werden. Mit Hilfe einer regulären Matrix P_L lässt sich für $Ax = b$ folgendes vorkonditioniertes System aufstellen

$$A_P = P_L A P_L^t,\ x_P = P_L^{-t} x \text{ und } b_P = P_L b,$$

welches anstelle des urspünglichen LGS gelöst wird. Da möglichst $P_L A P_L^t \approx I$ gelten soll, ist die Bestimmung von P_L für den vorliegenden Fall nur mit großem Aufwand überhaupt möglich, weil P_L meistens über Zerlegungen bestimmt wird. Für das CG-Verfahren muss die Matrix P_L allerdings nicht explizit bekannt sein. Es reicht stattdessen, wenn eine Matrix C existiert mit

$$C = P_L^t P_L.$$

Es ist weiterhin wichtig, dass die Matrix C theoretisch in eine solche Form zerlegbar ist. Diese Darstellung entspricht einer Cholesky-Zerlegung und garantiert, dass die Symmetrie und positive Definitheit erhalten bleiben. Ansonsten wäre das CG-Verfahren, wie bereits erwähnt, nicht mehr anwendbar. Mit der Matrix C lässt sich der Algorithmus 2.3 für das vorkonditionierte CG-Verfahren aus [35] aufstellen. Es wird ebenfalls eine zusätzliche Matrix-Vektor-Multiplikation benötigt. Bei dem später vorgestellten Vorkonditionierer wird diese aber nur implizit verwendet, da anstelle dessen ein LGS gelöst wird, um eine Invertierung der verwendeten Matrix zu vermeiden.

Algorithmus 2.3 : CG-Verfahren

Input : $A \in \mathbb{R}^{n\times n}$ und $C \in \mathbb{R}^{n\times n}$ beide symmetrisch und positiv definit, $b \in \mathbb{R}^n$

Output : Lösungsvektor x

begin
  Wähle $x_0 \in \mathbb{R}^n$.
  Berechne $r_0 := b - Ax_0,\ p_0 := Cr_0,\ \alpha_0 := r_0^t p_0,\ j := 0$
  while $\alpha_j \neq 0$ **do**
    $v_j := Ap_j,\ \lambda_j := \frac{\alpha_j}{v_j^t p_j}$
    $x_{j+1} := x_j + \lambda_j p_j$
    $r_{j+1} := r_j - \lambda_j v_j$
    $z_{j+1} := Cr_{j+1},\ \alpha_{j+1} := r_{j+1}^t z_{j+1}$
    $p_{j+1} := z_{j+1} + \frac{\alpha_{j+1}}{\alpha_j} p_j$
    $j := j + 1$
  end
  $x := x_{j+1}$
end

Es gibt mehrere Möglichkeiten C zu wählen. Eine sehr einfache Form stellen Skalierungen dar. Eine Skalierung ist wie folgt definiert:

Definition 2.27 (Skalierung) *Eine reguläre Diagonalmatrix*

$$D = \operatorname{diag}(d_{11}, \ldots, d_{nn}) \in \mathbb{R}^{n\times n}$$

heißt Skalierung.

Diese Definition stammt aus [35]. Gleiches gilt für die folgenden Vorschläge für mögliche Skalierungen, basierend auf einer regulären Matrix $A \in \mathbb{R}^{n\times n}$:

1. Skalierung mit den Reziprokwerten der Hauptdiagonalen, wobei $a_{ii} \neq 0$ für $i = 1, \ldots, n$:
$$d_{ii} := \frac{1}{a_{ii}},\ i = 1, \ldots, n. \tag{2.14}$$

2. Zeilen-/Spaltenskalierung bzgl. der Betragssummennorm:
$$d_{ii} := \frac{1}{\sum\limits_{j=1}^{n} |a_{ij}|} \text{ bzw. } d_{jj} := \frac{1}{\sum\limits_{i=1}^{n} |a_{ij}|} \tag{2.15}$$
für $i, j = 1, \ldots, n$.

3. Zeilen-/Spaltenskalierung bzgl. der euklidischen Norm:

$$d_{ii} := \frac{1}{\left(\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2}} \text{ bzw. } d_{jj} := \frac{1}{\left(\sum_{i=1}^{n} |a_{ij}|^2\right)^{1/2}} \tag{2.16}$$

   für $i, j = 1, \ldots, n$.

4. Zeilen-/Spaltenskalierung bzgl. der Maximumsnorm:

$$d_{ii} := \frac{1}{\max\limits_{j=1,\ldots,n} |a_{ij}|} \text{ bzw. } d_{jj} := \frac{1}{\max\limits_{i=1,\ldots,n} |a_{ij}|} \tag{2.17}$$

   für $i, j = 1, \ldots, n$.

Die Güte der Vorkonditionierung hängt von der jeweiligen Norm ab. Skalierungen erfüllen nicht zwingend die Anforderungen, welche an einen Vorkonditionierer für das CG-Verfahren gestellt werden. Dies muss insofern vorher überprüft werden. Skalierungen können ebenfalls für das GMRES-Verfahren verwendet werden. Vorteilhaft ist, dass sie sehr schnell zu berechnen sind, der Speicherbedarf sehr gering und aufgrund dessen die Kronecker-Struktur nicht zwingend nötig ist. Dies liegt daran, dass die Diagonale als Vektor gespeichert werden kann.

Beispiel 2.28 *Sei A wie in Beispiel 2.25 gewählt. Es soll in diesem Beispiel verdeutlicht werden, dass Skalierungen die Konditionszahl sowohl verschlechtern wie auch verbessern können. Zuerst wird die Skalierung aus* (2.14) *gewählt. Es gilt*

$$D = \begin{bmatrix} \frac{1}{35} & 0 & 0 \\ 0 & \frac{1}{45} & 0 \\ 0 & 0 & \frac{1}{114} \end{bmatrix}$$

und für die Konditionszahl folgt

$$\kappa(AD) = \|AD\|_\infty \|(AD)^{-1}\|_\infty = 738.17.$$

Die Konditionszahl verschlechtert sich. Eine Verbesserung wird mit der Skalie-

rung aus (2.17) *erzielt. Es gilt*

$$\tilde{D} = \begin{bmatrix} \frac{1}{40} & 0 & 0 \\ 0 & \frac{1}{58} & 0 \\ 0 & 0 & \frac{1}{114} \end{bmatrix}$$

und für die Konditionszahl folgt

$$\kappa(A\tilde{D}) = \|A\tilde{D}\|_\infty \|(A\tilde{D})^{-1}\|_\infty = 483.15.$$

Die Konditionszahl verbessert sich aber nicht wesentlich. Dies deutet den begrenzten Nutzen von Skalierungen an.

In [24] wurde die Skalierung aus (2.14) auf das aktuelle Problem angewendet und dort sind ebenfalls die zugehörigen Resultate zu finden. Sie konnten in diesem konkreten Fall durchaus überzeugen. In Kapitel 6 werden diese Resultate mit den später vorgestellten Vorkonditionierern verglichen. Der Nachteil von Skalierungen liegt in der geringen Verwendung von Informationen über die Matrix A. Aufgrund dessen soll in den Kapiteln 4 und 5 die Umsetzung von Vorkonditionierern vorgestellt werden, welche mehr Informationen verwenden. Die Kronecker-Struktur ist dabei von zentraler Bedeutung und entscheidend dafür, dass die meisten etablierten Vorkonditionierer (siehe [35] und [2]) — wie zum Beispiel die unvollständige LU-Zerlegung — nicht in Frage kommen.

2.4. Multilinearformen & Tensoren

Im Unterkapitel 2.1 wurde die Möglichkeit aufgezeigt, eine Matrix durch eine Summe von Kronecker-Produkten mit zwei Faktoren zu approximieren. Bei der betrachteten Problemstellung gleicht die Anzahl der Faktoren des Kronecker-Produkts der Anzahl der Dimensionen, da die Faktoren aus den univariaten Komponenten des Smoothing Splines gebildet werden. Deshalb werden bei der Approximation einer gegebenen Matrix $A \in \mathbb{R}^{m \times n}$ mit einer Summe von Kronecker-Produkten mehr als zwei Faktoren benötigt

$$A \approx \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} \text{ mit } m = \prod_{j=1}^{p} m_j,\ n = \prod_{j=1}^{p} n_j \text{ und } A_{ij} \in \mathbb{R}^{m_j \times n_j}.$$

Das Verfahren zur Bestimmung einer solchen Approximation wird in Kapitel 3 vorgestellt. Für dieses Verfahren wird die Matrix A ins Mehrdimensionale übertragen und die Approximation wird mit Hilfe der multilinearen Algebra bestimmt. Aufgrund dessen sollen an dieser Stelle vorbereitend benötigte Elemente aus der multilinearen Algebra vorgestellt werden. Bei diesen Elementen handelt es sich um Multilinearformen und Tensoren. Als Quellen für diese dienen [14, 23, 25] sowie [18]. Eine Multilinearform lässt sich, angelehnt an [14], wie folgt definieren:

Definition 2.29 (Multilinearform) *Eine Abbildung*

$$X^\flat : V_1 \times \cdots \times V_p \to K,$$

wobei K ein Körper ist und $V_1, \ldots, V_p$ Vektorräume über diesem, wird p-dimensionale Multilinearform *genannt, wenn für alle $u_i \in V_i,\ i = 1, \ldots, p$ erfüllt ist, dass:*

1. *Für alle $\lambda \in K$ gilt*

$$X^\flat[u_1, \ldots, \lambda u_i, \ldots, u_p] = \lambda X^\flat[u_1, \ldots, u_p],\ i = 1, \ldots, p,$$

2. *und für alle $w \in V_i$*

$$X^\flat[u_1, \ldots, u_i + w, \ldots, u_p] = X^\flat[u_1, \ldots, u_i, \ldots, u_p] + X^\flat[u_1, \ldots, w, \ldots, u_p],$$

für $i = 1, \ldots, p$.

Bemerkung 2.30 *In dieser Arbeit werden nur Multilinearformen auf endlichdimensionalen Vektorräumen $\mathbb{R}^{m_j n_j},\ j = 1, \ldots, p$, betrachtet. Die verwendete Multilinearform entspricht somit*

$$X^\flat : \mathbb{R}^{m_1 n_1} \times \cdots \times \mathbb{R}^{m_p n_p} \to \mathbb{R}.$$

Das ist relevant, da in diesem Fall eine Multilinearform mit einem Tensor identifiziert werden kann. Die Identifizierung wird im Folgenden näher ausgeführt und wurde bereits in [31] im Zusammenhang mit einer mehrdimensionalen Singulärwert-Zerlegung untersucht.

Um den Zusammenhang zwischen Multilinearformen und Tensoren erklären zu können, muss weiter der Begriff des Tensors definiert werden. Für eine bessere Übersicht wird die folgende Notation bzgl. der auftretenden Dimensionen festgelegt

$$\gamma = (m_1, \ldots, m_p),\ \eta = (n_1, \ldots, n_p) \text{ und } m = \prod_{j=1}^{p} m_j,\ n = \prod_{j=1}^{p} n_j$$

und weiterhin wird auf den Begriff des Multiindexes aus Definition 2.15 zurückgegriffen. Es sei

$$\Gamma_{\gamma\eta} := \{\alpha \in \mathbb{N}^p :\ \alpha_j \leq m_j n_j,\ j = 1, \ldots, p\}$$

eine Menge von Multiindizes. Die folgenden Bezeichnungen und Definitionen sind an [18], [23] sowie [16] angelehnt.

Definition 2.31 (Tensor) *Seien $m_1 n_1, \ldots, m_p n_p \in \mathbb{N}$. Ein* Tensor *ist definiert durch*

$$\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p} \ni \mathcal{X} := [x_\alpha = x_{\alpha_1,\ldots,\alpha_p} :\ \alpha \in \Gamma_{\gamma\eta}].$$

Ein Tensor kann als ein mehrdimensionales Array oder eine mehrdimensionale Matrix aufgefasst werden. Ein einfaches Beispiel für einen dreidimensionalen Tensor ist in Abbildung 2.6 zu finden. Da es nicht sinnvoll scheint auf eine solche Struktur mittels x_α nur elementweise zu zugreifen, soll auf die `Octave`-Notation zurückgegriffen werden. Bei einer Matrix A ist mit der Bezeichnung $A(i, :)$ die i-te Zeile gemeint und die Bezeichnung für die j-te Spalte ist analog. Dies lässt sich auf Tensoren erweitern, nur dass mehr als zwei Dimensionen ins Spiel kommen. Für einen dreidimensionalen Tensor $\mathcal{X} \in \mathbb{R}^{k \times m \times n}$ bezeichnet $\mathcal{X}(:, :, i)$ die i-te Matrix in der dritten Dimension. Wenn dies auf höhere Dimensionen übertragen wird, lässt sich erkennen, dass ein Tensor durch Tensoren mit einer Dimension weniger aufgebaut werden kann. Es gilt weiter, dass $x_{\alpha_1\alpha_2\alpha_3} = \mathcal{X}(\alpha_1, \alpha_2, \alpha_3)$. Die Abbildung 2.7 und das folgende Beispiel versuchen dies zu veranschaulichen.

Beispiel 2.32 *Eine Matrix ist ein einfaches Beispiel für einen zweidimensionalen Tensor. Tensoren in höheren Dimensionen setzen sich ebenfalls aus Matrizen zusammen und ein mögliches Beispiel für einen dreidimensionalen Tensor*

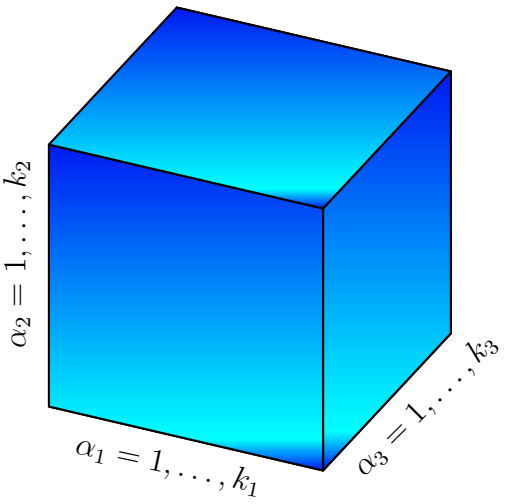

Abbildung 2.6.: Tensor im 3D

Die Abbildung zeigt einen dreidimensionalen Tensor. Die Anordnung der Elemente ist der einer Matrix sehr ähnlich. Es wird nur eine Dimension mehr benötigt $\mathcal{X}(\alpha_1, \alpha_2, \alpha_3) = x_{\alpha_1\alpha_2\alpha_3}$. Die Idee für die Darstellung stammt aus [23].

$\mathcal{X} \in \mathbb{R}^{3\times3\times2}$ *könnte wie wie folgt aussehen:*

$$\mathcal{X}(:,:,1) = \begin{bmatrix} 8 & 4 & 6 \\ 4 & 2 & 3 \\ 12 & 6 & 9 \end{bmatrix}$$

und

$$\mathcal{X}(:,:,2) = \begin{bmatrix} 16 & 8 & 12 \\ 8 & 4 & 6 \\ 24 & 12 & 18 \end{bmatrix}.$$

Die Zusammensetzung von Tensoren aus Matrizen lässt sich in drei Dimensionen anschaulich durch Stapeln der Matrizen (siehe Abbildung 2.7) visualisieren. In höheren Dimensionen ist dies aufgrund der Anzahl der Dimensionen nicht mehr möglich. Aber der Aufbau setzt sich analog fort. Ein Tensor kann durch Tensoren mit einer Dimension weniger zusammengesetzt werden. Die Dimensionen in den einzelnen Richtungen müssen allerdings übereinstimmen.

Um den Zusammenhang zwischen Tensoren und Multilinearformen explizit beschreiben zu können, müssen weitere Eigenschaften und Definitionen von Tenso-

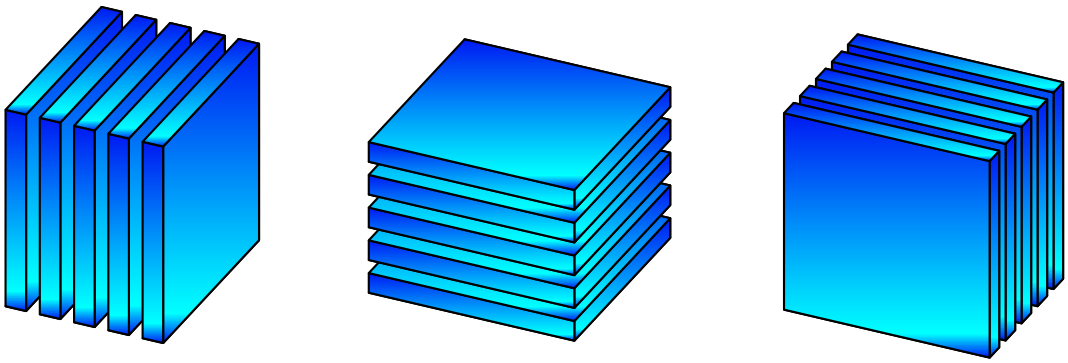

Abbildung 2.7.: Aufbau eines Tensor durch Matrizen

Die Grafik zeigt, wie sich ein dreidimensionaler Tensor durch Matrizen aufbauen lässt. Diese Vorstellung kann dabei helfen, die Umsetzung der Matricization aus Definition 2.39 besser zu veranschaulichen. Weiterhin setzt sich jeder Tensor durch niedrigdimensionale Tensoren zusammen. Die Skizze ist angelehnt an eine Abbildung aus [23].

ren betrachtet werden. Analog zu Matrizen lässt sich auch für Tensoren ein Rang angeben. In diesem Zusammenhang spielt das mehrdimensionale Äquivalent der Rang-1 Matrix eine Rolle. Um dieses einführen zu können, muss der Begriff des äußeren bzw. dyadischen Produkts ins Mehrdimensionale übertragen werden. Da der Begriff des äußeren Produkts nicht eindeutig und der des dyadischen Produkts im Mehrdimensionalen nicht mehr sinngemäß ist, wird dieses Produkt im Folgenden als tensorielles Produkt bezeichnet, wie es auch in der Literatur bekannt ist (siehe [14]). Die Bezeichnung sowie die Definition sind an [14] angelehnt.

Definition 2.33 (Tensorielles Produkt) *Das* tensorielle Produkt *von p reellen Vektoren $v_1 \in \mathbb{R}^{m_1 n_1}, \ldots, v_p \in \mathbb{R}^{m_p n_p}$ ist definiert als eine Verknüpfung der Form*

$$\otimes_t : \ \mathbb{R}^{m_1 n_1} \times \cdots \times \mathbb{R}^{m_p n_p} \to \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$$

mit

$$(v_1, \ldots, v_p) \mapsto v_1 \otimes_t \cdots \otimes_t v_p.$$

Die Bezeichnung $\otimes_t$ des gerade definierten Produktes soll Verwechslungen mit

dem Kronecker-Produkt vorbeugen. In der gängigen Literatur ist die Bezeichnung dieser beiden Produkte oftmals identisch. In der Definition ist das tensorielle Produkt nur auf Vektoren definiert, was dem Rang-1 Charakter entspricht. Dies ist allerdings nicht zwangsweise notwendig, da das Produkt assoziativ ist. Für zwei Matrizen $B = b_1 \otimes_t b_2 \in \mathbb{R}^{k_1 \times k_2}$ und $C = c_1 \otimes_t c_2 \in \mathbb{R}^{q_1 \times q_2}$ gilt

$$\begin{aligned} B \otimes_t C &= (b_1 \otimes_t b_2) \otimes_t (c_1 \otimes_t c_2) \\ &= [a_\alpha = b_{\alpha_1,\alpha_2} \cdot c_{\alpha_3,\alpha_4} : \ \alpha \leq (k_1, k_2, q_1, q_2)] = \mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times q_1 \times q_2}. \end{aligned}$$

Daraus folgt, dass das tensorielle Produkt zweier Matrizen einem vierdimensionalen Tensor entspricht. Das Ergebnis eines mehrdimensionalen, tensoriellen Produkts zwischen Vektoren ist ein Tensor, welcher als Analogon zu einer Rang-1 Matrix aufgefasst werden kann. Daher soll er im Folgenden als Rang-1 Tensor bezeichnet werden.

Definition 2.34 (Rang-1 Tensor) *Sei $v_j \in \mathbb{R}^{n_j m_j}$. Ein* Rang-1 Tensor *ist definiert durch*

$$\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p} \ni \mathcal{V} := v_1 \otimes_t \cdots \otimes_t v_p := [v_\alpha = v_{\alpha_1} \cdots v_{\alpha_p} : \ \alpha \in \Gamma_{\gamma\eta}].$$

Für Multilinearformen existiert eine identische Struktur, welche als 1-Form bezeichnet wird (siehe [25]). Da Multilinearformen in dieser Arbeit äquivalent zu Tensoren verwendet werden können, wie später zu sehen sein wird, werden diese Begriffe gleichbedeutend benutzt. Die Idee eines Rang-1 Tensors ist in Abbildung 2.8 skizziert und folgendes Beispiel veranschaulicht dessen Aufbau und Struktur:

Beispiel 2.35 *In zwei Dimensionen entspricht ein Rang-1 Tensor offensichtlich einer Rang-1 Matrix. In drei Dimensionen lässt sich ein Rang-1 Tensor mit dem tensoriellen Produkt darstellen. Der Tensor $\mathcal{X}$ aus Beispiel 2.32 kann zum Beispiel als folgender Rang-1 Tensor geschrieben werden:*

$$\mathcal{X} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} \otimes_t \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix} \otimes_t \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

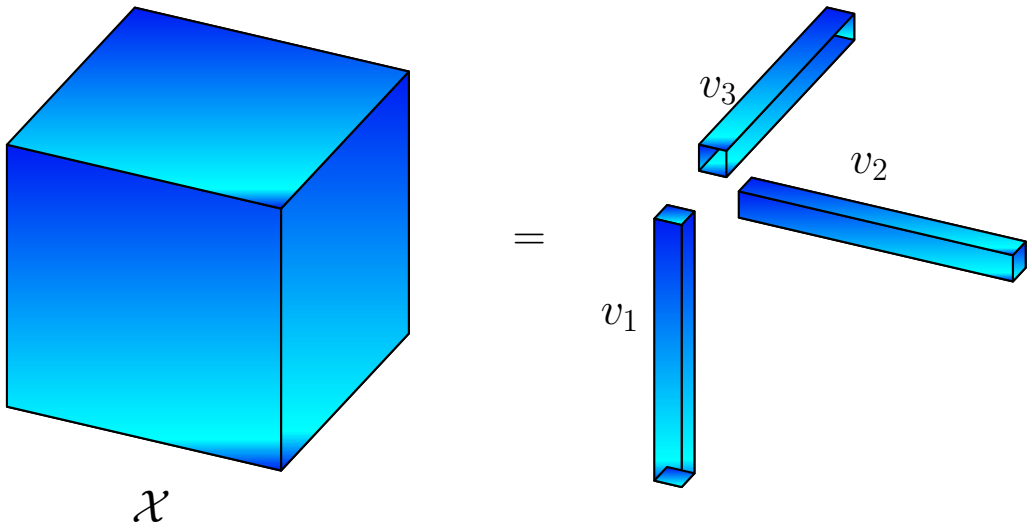

Abbildung 2.8.: Rang-1 Tensor

Die Abbildung zeigt einen dreidimensionalen Tensor, welcher durch einen Rang-1 Tensor beschrieben werden kann

$$\mathcal{X} = v_1 \otimes_t v_2 \otimes_t v_3.$$

Die Idee für die Darstellung eines dreidimensionalen Rang-1 Tensors stammt aus [23].

Jeder Tensor kann als Summe von Rang-1 Tensoren ausgedrückt werden. Der Rang eines Tensors entspricht der minimalen Anzahl von Rang-1 Tensoren, die für eine solche Darstellung benötigt werden.

Definition 2.36 (Rang eines Tensors) *Sei $\mathcal{X} \in \mathbb{R}^{m_1 \times \cdots \times m_p}$ ein Tensor. Der Rang* $\operatorname{rang}(\mathcal{X}) = r$ *des Tensors ist definiert durch die minimale Anzahl an Rang-1 Tensoren mit denen $\mathcal{X}$ dargestellt werden kann*

$$\mathcal{X} = \sum_{i=1}^{r} v_{i1} \otimes_t \cdots \otimes_t v_{ip} \text{ mit } v_{ij} \in \mathbb{R}^{m_j},\ j = 1, ..., p.$$

Die Darstellung bzgl. Rang-1 Tensoren muss allerdings nicht eindeutig sein. Eine solche Formulierung des Rangs ist auch für Multilinearformen möglich, indem diese durch Summen von 1-Formen dargestellt werden. Die Formulierung lässt sich aber nur anwenden, wenn die Multilinearform auf endlichdimensionalen Vektorräumen basiert. Weitere Bezeichnungen für einen Rang-1 Tensor sind

Elementartensor oder kanonischer Tensor, siehe [16, 18, 23].

Bemerkung 2.37 *Die Menge der Rang-*1 *Tensoren bildet eine echte Untermenge der Menge der Tensoren*

$$\{v_1 \otimes_t \cdots \otimes_t v_p : \ v_j \in \mathbb{R}^{m_j n_j}, \ j = 1, \ldots, p\} \subset \{\mathcal{X} : \ \mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}\}$$

*Wenn für die Vektoren des tensoriellen Produkts der Rang-*1 *Tensoren die Basisvektoren der einzelnen Vektorräume verwendet werden, lässt sich so eine Basis aufstellen. Dies ist aber weitläufig bekannt und zum Beispiel in [18] expliziter nachzulesen.*

Die weiteren Ausführungen bzgl. Tensoren gelten in dem hier vorliegenden Fall ebenfalls für Multilinearformen, was wiederum in [25] zu finden ist. Auf Tensoren $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ lässt sich laut [18] ein inneres Produkt

$$\langle \mathcal{X}, \mathcal{Y} \rangle := \sum_{\alpha \leq \gamma\eta} x_\alpha y_\alpha$$

und damit eine Norm

$$\|\mathcal{X}\|_F^2 = \sum_{\alpha \leq \gamma\eta} x_\alpha^2$$

definieren. Es gilt

$$\langle \mathcal{X}, \mathcal{X} \rangle = \|\mathcal{X}\|_F^2.$$

Für die effiziente Berechnung des inneren Produkts zweier Rang-1 Tensoren lässt sich deren Struktur ausnutzen, so dass nur das Produkt der univariaten Skalarprodukte berechnet werden muss. Seien $\mathcal{V} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ und $\mathcal{U} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ zwei Rang-1 Tensoren. Für das innere Produkt folgt:

$$\langle \mathcal{V}, \mathcal{U} \rangle = \prod_{j=1}^{p} v_j^t u_j \text{ mit } v_j, u_j \in \mathbb{R}^{m_j n_j}. \tag{2.18}$$

Diese Eigenschaft wird im Zusammenhang mit den Summen von Kronecker-Produkten bei den Smoothing Splines aus Kapitel 2.2 noch sehr wichtig sein. Weiterhin lässt sich mit Hilfe des inneren Produkts die Orthogonalität von Rang-1 Tensoren definieren. Dies ist aus [22] übernommen.

Definition 2.38 (Orthogonalität) *Seien $\mathcal{U}, \mathcal{V} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ zwei Rang-1 Tensoren. Die Rang-1 Tensoren sind orthogonal ($\mathcal{U} \perp \mathcal{V}$), wenn gilt*

$$\langle \mathcal{U}, \mathcal{V} \rangle = 0.$$

In [22] wird dies ausführlicher untersucht und es wird zwischen verschiedenen starken Orthogonalitäten unterschieden. An dieser Stelle reicht die einfache Orthogonalität aus, da sie nur kurz zu Beginn des Kapitels 3 benötigt wird. Um Multilinearformen bzw. Tensoren für die Komprimierung von den gegebenen Kronecker-Produkt Summen verwenden zu können, lässt sich ein Zusammenhang zwischen Kronecker-Produkten und Rang-1 Tensoren ausnutzen. Sei die Matrix A ein Kronecker-Produkt

$$\mathbb{R}^{m \times n} \ni A = \bigotimes_{j}^{p} A_j \text{ mit } A_j \in \mathbb{R}^{m_j \times n_j}.$$

Mit Hilfe des *vec*-Operators aus Definition 2.3

$$A \in \mathbb{R}^{m \times n} \mapsto vec(A) \in \mathbb{R}^{mn}$$

lässt sich obiges Kronecker-Produkt zu einem Rang-1 Tensor umstrukturieren

$$\underbrace{A_1 \otimes \cdots \otimes A_p}_{\text{Kronecker-Produkt}} \cong \underbrace{v(A_1) \otimes_t \cdots \otimes_t v(A_p)}_{p\text{-dim. Rang-1 Tensor}}. \tag{2.19}$$

Dies kann als das höherdimensionale Analogon der Umstrukturierung $\mathcal{R}(\cdot)$ von einem Kronecker-Produkt mit zwei Faktoren aus (2.3) aufgefasst werden. Dort wurde ebenfalls für die Komprimierung eine Umstrukturierung der zu approximierenden Matrix verwendet. Für die Umformung in (2.19) lässt sich in diesem Zusammenhang eine Bijektion zwischen den Zeilen- und Spaltenindizes der Matrix und den Multiindizes des Rang-1 Tensors aufstellen

$$\rho : \{1, \ldots, m\} \times \{1, \ldots, n\} \mapsto \Gamma_{\gamma\eta}. \tag{2.20}$$

Es gibt viele Möglichkeiten diese Bijektion zu wählen und in [23] sowie in vielen anderen Quellen wird sie nicht explizit angegeben. Wenn allerdings als Voraussetzung gefordert wird, dass sie ein Kronecker-Produkt auf einen Rang-1 Tensor abbilden und die Reihenfolge der Komponenten nicht verändert werden soll,

dann kann die Bijektion eindeutig angegeben werden. Die Funktion $\rho(i,j) = \alpha$ wird folgendermaßen realisiert: Zuerst werden die Indizes für die Kronecker-Struktur (i_1, j_1),...,(i_p, j_p) berechnet. Es ist dabei nicht relevant, ob die Matrix überhaupt eine Kronecker-Struktur besitzt. Diese erste Berechnung dient nur als Hilfestellung für die Berechnung der Indizes des Tensors. Es geht primär, um die Position der Elemente. Die Berechnung des Indexes i_k für den gegebenen Index i soll im Folgenden vorgestellt werden:

$$i_k = \left\lceil i^{(k)} \mod \prod_{\ell=k+1}^{p} m_\ell \right\rceil \text{ mit } k = 1, \ldots, p-1$$

wobei $i^{(1)} = i$ sowie

$$i^{(k)} = \begin{cases} i^{(k-1)} \mod \prod_{\ell=k}^{p} m_\ell, & \text{für } i^{(k-1)} \mod \prod_{\ell=k}^{p} m_\ell \neq 0 \\ \prod_{\ell=k}^{p} m_\ell, & \text{sonst} \end{cases}$$

für $k = 2, \ldots, p-1$. Der Operator $\lceil x \rceil := \min\{k \in \mathbb{Z} \mid k \geq x\}$ bezeichnet die Aufrundungsfunktion. Der letzte Index i_p für die Kronecker-Struktur berechnet sich durch

$$i_p = \begin{cases} i^{(p-1)} \mod m_p, & \text{für } i^{(p-1)} \mod m_p \neq 0 \\ m_p, & \text{sonst.} \end{cases}$$

Die Berechnung der Indizes j_k für die Spalten funktioniert analog. Mit Hilfe dieser Indizes lassen sich die einzelnen Werte des Multiindexes α durch

$$\alpha_k = i_k j_k + (j_k - 1) m_k$$

berechnen. Dies entpricht der Vektorisierung der Faktoren des Kronecker-Produkts. Wie bereits erwähnt, kann die Abbildung ρ auf die Indizes einer Matrix ohne Kronecker-Struktur angewendet werden, so dass gilt

$$R^{m \times n} \ni X \leftrightarrow \mathcal{X} \text{ mit } X(i,j) = x_{\rho(i,j)} = x_\alpha.$$

Später wird auch eine Umkehrung benötigt werden. Es muss dann ausgehend von einem Multiindex $\alpha \in \Gamma_{\gamma\eta}$ der zugehörige Spaltenindex berechnet werden. Gesucht ist folglich eine Abbildung, welche einen Multiindex auf einen eindi-

mensionalen Index abbildet

$$\psi : \ \Gamma_{\gamma\eta} \to \{1, \ldots, mn\}.$$

Die Abbildung entspricht im Grunde einer Umkehrfunktion von ρ, allerdings nur auf einer Komponente. Sie lässt sich angeben durch

$$\psi(\alpha) = \psi(\alpha, \gamma, \eta) := \sum_{i=1}^{p} \left((\alpha_i - 1) \prod_{j=1}^{i-1} m_j n_j \right).$$

Mit Hilfe dieser Umstrukturierungen lässt sich eine beliebige Matrix als Tensor bzw. Multilinearform auffassen, solange sich die Spalten- und Zeilendimensionen der Matrix in Produkte $m = \prod_{j=1}^{p} m_j$ und $n = \prod_{j=1}^{p} n_j$ aufteilen lassen. Weiterhin ist die Umstrukturierung ρ einer Matrix in einen Tensor analog zu dem Operator $\mathcal{R}(\cdot)$ in jeder Komponente linear. Dies lässt sich durch Nachrechnen beweisen.

Im letzten Teil dieses Abschnittes soll erläutert werden, inwiefern eine Matrix bzw. die Umstrukturierung in einen Tensor als Multilinearform aufgefasst werden kann. In [31] wurde bereits der Zusammenhang zwischen Multilinearformen und Tensoren bei der Zerlegung von Tensoren mittels einer höherdimensionalen SVD aufgeführt. Die Multilinearform $X^\flat[u_1, \ldots, u_p]$ einer Matrix $A \in \mathbb{R}^{m \times n}$ entspricht der Umstrukturierung dieser in einen Tensor $\mathcal{X}$ und dem Aufsummieren des komponentenweisen Produkts mit den Variablen. Dies wird verständlicher, wenn die Variablen als Rang-1 Tensor aufgefasst werden. Es gilt folglich

$$A \to \mathcal{X} \text{ sowie } (u_1, \ldots, u_p) \to u_1 \otimes_t \cdots \otimes_t u_p \text{ wobei } u_j \in \mathbb{R}^{m_j n_j}.$$

Die Multilinearform lässt sich ausdrücken durch

$$X^\flat[u_1, \ldots, u_p] = \sum_{\alpha \in \Gamma_{\gamma\eta}} x_\alpha u_\alpha.$$

Falls die Matrix A einem Kronecker-Produkt enspricht, also in einen Rang-1 Tensor umgewandelt werden kann

$$A \to \mathcal{X} = v_1 \otimes_t \cdots \otimes_t v_p \text{ wobei } v_j \in \mathbb{R}^{m_j n_j},$$

dann kann die Gleichung (2.18) ausgenutzt werden. Für die zugehörige Multili-

nearform gilt

$$X^\flat[u_1, \ldots, u_p] = \prod_{j=1}^{p} v_j^t u_j.$$

und eine solche Multilinearform wird 1-Form genannt. Die Umsetzung wird später wichtig sein. Es wurden zwei Möglichkeiten vorgestellt, wie eine Multilinearform im Kontext von Tensoren umzusetzen ist. Die naive Umsetzung entspricht der Aufsummierung der komponentenweisen Multiplikation des Tensors und des Rang-1 Tensors, welcher die Variablen enthält. Zur besseren und übersichtlicheren Umsetzung von Multilinearformen auf Tensoren soll der Begriff der multilinearen Multiplikation eingeführt werden. Es wird dabei in jeder Dimension ein Vektor bzw. eine Matrix an den Tensor multipliziert.

Eine Matrix ist ein gutes Beispiel, um diesen Sachverhalt zu verstehen. Es werden von links und rechts Vektoren an die Matrix multipliziert, in jeder Dimension ein Vektor, und dadurch erhält man eine Bilinearform. Die zu multiplizierenden Vektoren entsprechen den Argumenten der Bilinearform. Wenn anstelle von Vektoren Matrizen als Argumente betrachtet werden, handelt es sich um eine bilineare Abbildung. Für die mehrdimensionale Multiplikation braucht es zusätzlich eine Umstrukturierung des Tensors in verschiedene Matrizen, abhängig von den jeweiligen Dimensionen. Diese wird als Matricization bezeichnet und ist als „Plättung“ des Tensors in die verschiedenen Richtungen zu verstehen. Die Definition der beiden Begriffe stammt aus [23].

Definition 2.39 (Matricization) *Sei $\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ ein Tensor und sei $j = 1, \ldots, p$. Die* Matricization *in die j-te Dimension ist definiert durch*

$$\mathcal{M}_j : \ \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p} \to \mathbb{R}^{m_j n_j \times \tilde{m}}, \ \textit{mit } \tilde{m} = \prod_{\ell \neq j} m_\ell n_\ell,$$

und

$$(\mathcal{M}_j(\mathcal{X}))_{\alpha_j, \psi(\alpha_1, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_p)} := x_{\alpha_1, \ldots, \alpha_p}$$

wobei die Funktion ψ die übrigen Komponenten der Multiindizes auf den entsprechenden Spaltenindex abbildet.

Eine Matricization lässt sich durch eine Abbildung der Multiindizes auf den Zeilen- und Spaltenindex realisieren, wobei die bekannten Funktionen ρ und ψ verwendet werden. Diese „Plättung“ lässt sich einfacher nachvollziehen, wenn der Leser Abbildung 2.7 betrachtet und sich vorstellt, dass die einzelnen Schei-

ben des „Stapels“ von Matrizen in der gewählten Richtung nebeneinander gelegt werden. Folgendes Beispiel zeigt ebenfalls, wie diese Umstrukturierung umzusetzen ist:

Beispiel 2.40 *Der Tensor $\mathcal{X}$ aus dem Beispiel 2.32 lässt sich wie folgt in die verschiedenen Richtungen zu einer Matrix umstrukturieren:*

$$\begin{aligned}\mathcal{M}_1(\mathcal{X}) &= \begin{pmatrix}2\\1\\3\end{pmatrix} \otimes_t \left(\begin{pmatrix}4 & 2 & 3\end{pmatrix} \otimes \begin{pmatrix}1 & 2\end{pmatrix}\right)\\ &= \begin{bmatrix}8 & 16 & 4 & 8 & 6 & 12\\ 4 & 8 & 2 & 4 & 3 & 6\\ 12 & 24 & 6 & 12 & 9 & 18\end{bmatrix}.\end{aligned}$$

$$\begin{aligned}\mathcal{M}_2(\mathcal{X}) &= \begin{pmatrix}4\\2\\3\end{pmatrix} \otimes_t \left(\begin{pmatrix}2 & 1 & 3\end{pmatrix} \otimes \begin{pmatrix}1 & 2\end{pmatrix}\right)\\ &= \begin{bmatrix}8 & 16 & 4 & 8 & 12 & 24\\ 4 & 8 & 2 & 4 & 6 & 12\\ 6 & 12 & 3 & 6 & 9 & 18\end{bmatrix}.\end{aligned}$$

$$\begin{aligned}\mathcal{M}_3(\mathcal{X}) &= \begin{pmatrix}1\\2\end{pmatrix} \otimes_t \left(\begin{pmatrix}2 & 1 & 3\end{pmatrix} \otimes \begin{pmatrix}4 & 2 & 3\end{pmatrix}\right)\\ &= \begin{bmatrix}8 & 4 & 6 & 4 & 2 & 3 & 12 & 6 & 9\\ 16 & 8 & 12 & 8 & 4 & 6 & 24 & 12 & 18\end{bmatrix}.\end{aligned}$$

Es lässt sich an dieser Stelle deutlich die Ausnutzung der zugrundeliegenden Rang-1 Struktur von $\mathcal{X}$ erkennen.

Mit Hilfe der Matricization lässt sich eine multilineare Multiplikation auf Tensoren definieren.

Definition 2.41 (Multilineare Multiplikation) *Sei $\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$, $j = 1, \ldots, p$ und $U_j \in \mathbb{R}^{k_j \times m_j n_j}$. Dann ist die Multiplikation in der j-ten Dimension*

$U_j \circ_j X$ definiert durch

$$\mathcal{M}_j\left(U_j \circ_j \mathcal{X}\right) := U_j \mathcal{M}_j(\mathcal{X}) \in \mathbb{R}^{k_j \times \tilde{m}}, \text{ mit } \tilde{m} = \prod_{\ell \neq j} m_\ell n_\ell$$

sowie den Einträgen

$$\left(U_j \circ_j \mathcal{X}\right)_{\alpha_1,\ldots,\alpha_{j-1},k_j,\alpha_{j+1},\ldots,\alpha_p} := \sum_{\alpha_j=1}^{m_j n_j} (U_j)_{k_j,\alpha_j} X_{\alpha_1,\ldots,\alpha_{j-1},\alpha_j,\alpha_{j+1},\ldots,\alpha_p}.$$

Die multilineare Multiplikation *mit den Matrizen $U_j \in \mathbb{R}^{k_j \times m_j n_j}$, $j = 1, \ldots, p$, ist definiert durch*

$$(U_1, \ldots, U_p) \circ \mathcal{X} := U_1 \circ_1 \cdots \circ_{p-1} U_p \circ_p \mathcal{X} \in \mathbb{R}^{k_1 \times \cdots \times k_p},$$

wobei die Reihenfolge vernachlässigt werden kann.

Ein Tensor $\mathcal{X}$ lässt sich als Multilinearform $X^\flat$ verstehen, wenn er in jeder Dimension mit einem Vektor multipliziert wird

$$u_1 \circ_1 \cdots \circ_{p-1} u_p \circ_p \mathcal{X} = X^\flat[u_1, \ldots, u_p].$$

Diese Darstellung einer Multilinearform wird durch folgendes Beispiel deutlich:

Beispiel 2.42 *Der Tensor $\mathcal{X}$ aus Beispiel 2.32 soll zur Veranschaulichung der multilinearen Multiplikation mit den Vektoren*

$$\left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)$$

multipliziert werden

$$\left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \circ \mathcal{X} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \circ_1 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \circ_2 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \circ_3 \mathcal{X}.$$

Es wird zuerst der Vektor $\begin{pmatrix} 1 & 1 \end{pmatrix}^t$ in der dritten Dimension multipliziert und

mit dem Beispiel 2.40 ergibt sich

$$\left(\begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right) \circ \mathcal{X} = \begin{pmatrix}1 & 1 & 1\end{pmatrix} \begin{bmatrix}24 & 12 & 36\\12 & 6 & 18\\18 & 9 & 27\end{bmatrix} \begin{pmatrix}1\\1\\1\end{pmatrix} = 162.$$

Durch die Multiplikation mit einem Vektor verringert sich die Dimension des Tensors um 1 *und damit kann die multilineare Multiplikation mit den anderen beiden Vektoren durch eine Matrix-Vektor-Multiplikation ausgedrückt werden. Das Ergebnis der multilinearen Multiplikation lässt sich über die Rang-*1 *Darstellung effizienter berechnen*

$$\begin{pmatrix}2 & 1 & 3\end{pmatrix} \begin{pmatrix}1\\1\\1\end{pmatrix} \otimes_t \begin{pmatrix}4 & 2 & 3\end{pmatrix} \begin{pmatrix}1\\1\\1\end{pmatrix} \otimes_t \begin{pmatrix}1 & 2\end{pmatrix} \begin{pmatrix}1\\1\end{pmatrix} = 6 \cdot 9 \cdot 3 = 162.$$

Dieser Umstand lässt sich sehr effizient für die Summe von Kronecker-Produkten bei den Tensorprodukt Smoothing Splines ausnutzen. Es ist offensichtlich, dass

$$\left(\begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right) \circ \mathcal{X} = X^\flat \left[\begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right].$$

Ein Tensor entspricht aufgrund der multilinearen Multiplikation einer multilinearen Abbildung, wie es bereits für Matrizen aufgezeigt wurde. Eine multilineare Abbildung bildet im Gegensatz zu einer Multilinearform auf einen Vektorraum ab und nicht nur auf den zugrundeliegenden Körper. Jede Multilinearform auf Vektorräumen mit endlicher Basis kann folglich mit einem Tensor identifiziert werden. Diesen Zusammenhang hebt das folgende Beispiel nochmals hervor.

Beispiel 2.43 *Es soll in diesem Beispiel veranschaulicht werden, wie eine Determinante über einen Tensor ausgedrückt werden kann. Dies soll verdeutlichen, wie eng Multilinearformen mit Tensoren zusammenhängen. Es wird die Determinante einer quadratischen Matrix* $A \in \mathbb{R}^{2\times 2}$ *betrachtet. Es gilt*

$$\det(A) = \det\left(\begin{bmatrix}a_{11} & a_{12}\\a_{21} & a_{22}\end{bmatrix}\right) = a_{11}a_{22} - a_{12}a_{21}.$$

Mittels eines zweidimensionalen Tensors lässt sich folgende Multilinearform aufstellen, welche der Determinante entspricht

$$X^\flat\left[\begin{pmatrix}a_{11}\\a_{21}\end{pmatrix},\begin{pmatrix}a_{22}\\a_{21}\end{pmatrix}\right] = \begin{pmatrix}a_{11} & a_{21}\end{pmatrix}\underbrace{\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}}_{=\mathcal{X}}\begin{pmatrix}a_{22}\\a_{21}\end{pmatrix}$$
$$= \begin{pmatrix}a_{11}\\a_{21}\end{pmatrix} \circ_1 \begin{pmatrix}a_{22}\\a_{21}\end{pmatrix} \circ_2 \mathcal{X}.$$

Es soll noch erwähnt werden, dass das Konzept des Kronecker-Produkts auch auf Tensoren anwendbar ist, wie die folgende Bemerkung zeigt.

Bemerkung 2.44 *Die Definition des Kronecker-Produkts lässt sich intuitiv auf Tensoren und Multilinearformen erweitern. Seien* $\mathcal{X} \in \mathbb{R}^{m_1\times\cdots\times m_p}$ *und* $\mathcal{Y} \in \mathbb{R}^{n_1\times\cdots\times n_p}$ *Tensoren. Das Kronecker-Produkt von diesen Tensoren lässt sich wie folgt definieren:*

$$\mathcal{X}\otimes\mathcal{Y} := [x_\alpha\mathcal{Y} \ : \ \alpha \le (m_1,\ldots,m_p)] \in \mathbb{R}^{m_1n_1\times\cdots\times m_pn_p}.$$

*Die Umwandlung des Approximationsterms eines multivariaten Smoothing Splines auf gestreuten Daten zu einem Rang-*1 *Tensor oder einer* 1*-Form kann aufgrund der vorliegenden Struktur als Kronecker-Produkt von Rang-*1 *Tensoren bzw.* 1*-Formen dargestellt werden*

$$(a_1^ta_1)\otimes\cdots\otimes(a_p^ta_p) \cong (a_1\otimes a_1)\otimes_t\cdots\otimes_t(a_p\otimes a_p)$$
$$\cong (a_1\otimes_t\cdots\otimes_t a_p)\otimes(a_1\otimes_t\cdots\otimes_t a_p).$$

*Dies liegt daran, dass die Kronecker-Faktoren der „Least-squares“ Vandermonde-Matrix symmetrische Rang-*1 *Matrizen sind, welche aus den jeweiligen Zeilen der univariaten Vandermonde-Matrizen bestimmt werden.*

Abschließend lässt sich festhalten, dass jede Multilinearform mit einem Tensor identifiziert werden kann, solange die Multilinearform auf endlichdimensionalen Vektorräumen definiert ist. Eine Multilinearform stellt aufgrund der Definition 2.29 eine allgemeinere Struktur wie ein Tensor dar, allerdings wird dies bei der vorliegenden Problemstellung nicht ausgenutzt. In dieser Arbeit ist eine Multilinearform mit einem Tensor gleichzusetzen. Aufgrund der vorhandenen

Kronecker-Struktur ist eine schnelle und effiziente Auswertung einer Multilinearform möglich und es müssen nur die univariaten Komponenten betrachtet werden. Dies ermöglicht einen Verzicht auf die komplexe Indizierung bei den Tensoren. Im Folgenden werden die Begriffe Multilinearform und Tensor analog verwendet. Dies gilt ebenso für die 1-Form und den Rang-1 Tensor.

3

Komprimierung von Tensorprodukt Smoothing Splines

Das folgende Kapitel stellt eine Möglichkeit vor eine Summe von Kronecker-Produkten zu approximieren und mit Hilfe der Approximation zu komprimieren. Dies ist dadurch motiviert, da bei Tensorprodukt Smoothing Splines solche Summen mit einer großen Anzahl an Summanden auftreten und diese Darstellung nicht effizient scheint. Mit Hilfe der Approximation soll versucht werden, die wesentlichen Informationen in einer komprimierten Summe von Kronecker-Produkten wiederzugeben. Da die ersten drei Unterkapitel zum Großteil auf [25] basieren, wird dort hauptsächlich von Multilinearformen gesprochen. Im letzten Unterkapitel werden kurz existierende Verfahren für Tensoren aus [18] vorgestellt und daher wird dort von Tensoren die Rede sein. Wie im letzten Abschnitt zu sehen war, können die beiden Bezeichnungen in dieser Arbeit jedoch äquivalent verwendet werden. Im letzten Unterkapitel werden allerdings Verfahren erwähnt, die Tensoren als multilineare Abbildungen verwenden und damit eine Multilinearform ausschließen.

In Kapitel 2.1 wurde eine Möglichkeit vorgestellt, eine Matrix durch ein Kronecker-Produkt mit zwei Faktoren zu approximieren und in Kapitel 2.2 wurde gezeigt, dass bei Tensorprodukt Smoothing Splines die Anzahl der Faktoren eines Kronecker-Produkts mit der Anzahl der Dimensionen des betrachteten Pro-

blems zusammenhängt. Dies lässt sich gut durch die Darstellung des Splines als Summe von Kronecker-Produkten erkennen. Ein Kronecker-Produkt mit zwei Faktoren ist folglich für die Approximation in dem Fall $p > 2$ nicht mehr ausreichend. Es wird daher eine Variante mit mehreren Faktoren benötigt. Durch iterative Anwendung der Vorgehensweise aus Kapitel 2.1 mit den Formeln (2.6) und (2.8) lässt sich die Matrix $A^{m \times n}$, mit $m = \prod_{j=1}^{p} m_j$ und $n = \prod_{j=1}^{p} n_j$, in eine Summe von Kronecker-Produkten mit mehr als zwei Faktoren zerlegen. Diese Zerlegung ist nicht eindeutig, da A bei jedem Schritt in unterschiedliche Faktoren bzgl. eines Kronecker-Produkts zerlegt werden kann. Die Idee soll an einem kurzen Beispiel veranschaulicht werden.

Beispiel 3.1 *Sei $A \in \mathbb{R}^{m \times n}$, mit $m = \prod_{j=1}^{3} m_j$ und $n = \prod_{j=1}^{3} n_j$. Im ersten Schritt wird mit Hilfe von* (2.6) *und* (2.8) *eine Zerlegung in*

$$A = \sum_{i=1}^{r_1} B_i \otimes \tilde{C}_i \text{ mit } B_i \in \mathbb{R}^{m_1 \times n_1} \text{ und } \tilde{C}_i \in \mathbb{R}^{m_2 m_3 \times n_2 n_3}$$

bestimmt. Im zweiten Schritt werden dann die Matrizen $\tilde{C}_i$ in

$$\tilde{C}_i = \sum_{k=1}^{r_{2,i}} C_{ik} \otimes D_{ik} \text{ mit } C_{ik} \in \mathbb{R}^{m_2 \times n_2} \text{ und } D_{ik} \in \mathbb{R}^{m_3 \times n_3}$$

zerlegt. Insgesamt ergibt sich damit

$$A = \sum_{i=1}^{r_1} B_i \otimes \left(\sum_{k=1}^{r_{2,i}} C_{ik} \otimes D_{ik} \right) = \sum_{i=1}^{r_1} \sum_{k=1}^{r_{2,i}} B_i \otimes C_{ik} \otimes D_{ik}.$$

Die gewünschte Darstellung wird auf diesem Weg zwar erreicht, aber gerade der erste Schritt wirft Probleme auf, da vermieden werden soll die gesamte Matrix aufzustellen. Dies ist aber für die SVD von $\mathcal{R}(A)$ im ersten Schritt nötig. Die Matrizen $\tilde{C}_i$ können ebenfalls zu einer Speicherauslastung führen, wenn die entsprechende Dimension und die zugehörigen Parameter zu groß sind.

Des Weiteren existiert nur eine eingeschränkte Ordnung bezüglich der Gewichtung der einzelnen Kronecker-Produkte in der Summe. Dies ist für eine Approximation nicht hilfreich, da nachträglich die Summanden sortiert werden müssen. Es ist ebenfalls nicht eindeutig, in welcher Reihenfolge die Dimensionen zerlegt werden sollen. Die Zerlegung im ersten Schritt könnte auch mit $\tilde{B}_i \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$ und $C_i \in \mathbb{R}^{m_3 \times n_3}$ umgesetzt werden.

Das Beispiel deutet an, dass eine andere Vorgehensweise zur Bestimmung einer Approximation der Matrix A durch Kronecker-Produkte benötigt wird, da die skizzierte Vorgehensweise zu Speicherengpässen führen kann. Wie bereits in Kapitel 2.4 dargestellt, kann ein Kronecker-Produkt als Multilinearform oder als 1-Form aufgefasst werden. Analog können Summen von Kronecker-Produkten als Summen von Multilinearformen dargestellt werden. Diese Umstrukturierung kann als mehrdimensionale Variante der Umstrukturierung aus Kapitel 2.1 bzw. [33] verstanden werden. Es gilt im Folgenden für

$$A = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij}, \text{ dass } \mathcal{R}(A) = \sum_{i=1}^{N} \bigotimes_{i=1}^{p} vec(A_{ij}).$$

Die Umstrukturierung wird genutzt, um eine Approximation an die Matrix A durch ein Kronecker-Produkt mit mehreren Faktoren zu bestimmen. Mit Hilfe der bijektiven Umstrukturierung einer Matrix zu einer Multilinearform bzw. einem Tensor und dem Beispiel 3.1 lässt sich erkennen, dass für jede Multilinearform eine endliche, orthogonale Zerlegung in 1-Formen existiert. Diese Zerlegung muss nicht zwingend minimal sein, zeigt aber eine obere Schranke auf. Das Resultat wird bereits im Bereich der Tensoren erwähnt und bewiesen, siehe [22]. Es soll an dieser Stelle allerdings für das bessere Verständnis und auf eine andere Art, wie bisher bekannt, gezeigt werden.

Satz 3.2 *Sei $A^\flat$ eine Multilinearform auf $\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$, welche in eine Matrix $A \in \mathbb{R}^{m \times n}$ umstrukturierbar ist. Für $A^\flat$ existiert eine endliche, orthogonale Zerlegung in eine Summe von 1-Formen*

$$A^\flat = \sum_{i=1}^{s} v_{i1} \otimes_t \cdots \otimes_t v_{ip}.$$

Beweis: Durch iteratives Anwenden der Formeln (2.6) und (2.8) lässt sich für die Matrix A eine Zerlegung in eine Summe von Kronecker-Produkten realisieren

$$A = \sum_{i=1}^{s} \bigotimes_{j=1}^{p} V_{ij},$$

wie in Beispiel 3.1 veranschaulicht wird. Mit Hilfe der Umstrukturierung $\mathcal{R}(\cdot)$ wird die endliche Zerlegung in 1-Formen erkenntlich. Da nach Satz 2.9

$$\min \|A - B \otimes C\|_F = \min \|\mathcal{R}(A) - vec(B) \otimes vec(C)\|_F,$$

gilt und die Rang-1 Matrix aus dem größtem Singulärwert und den zugehörigen Singulärvektoren von $\mathcal{R}(A)$ nach [32] genau die Norm minimiert, folgt dass

$$v_{i1} \otimes_t \cdots \otimes_t v_{ip} \perp v_{k1} \otimes_t \cdots \otimes_t v_{kp},\ i \neq k,$$

weil die v_{ij} den einzelnen Singulärvektoren entsprechen. Anschaulich existiert in jeder 1-Form $v_{i1} \otimes_t \cdots \otimes_t v_{ip}$ mindestens ein Vektor v_{ij}, welcher zu den anderen Vektoren v_{kj} senkrecht ist. Dadurch sind alle 1-Formen zueinander orthogonal. □

Korollar 3.3 *Mit Satz 3.2 ergibt sich $s = \sum_{j=1}^{p} m_j n_j$ als obere Schranke für die Anzahl an Summanden einer orthogonalen Zerlegung bzgl. der Multilinearform $A^\flat$ aus Satz 3.2.*

Wie bereits erwähnt, ist diese Vorgehensweise praktisch für Tensorprodukt Splines nicht sinnvoll anwendbar. Deswegen soll zunächst eine Approximation der Multilinearform vorgestellt werden, welche sich in Matrizen umwandeln lässt und damit als Approximation für $A \in \mathbb{R}^{m \times n}$ dient. Im Anschluss daran wird auf ein Verfahren eingegangen, mit welchem diese Zerlegung bestimmt werden kann. Dieses Verfahren entspricht im Wesentlichen der „Higher Order Power Method“ aus [30] für Tensoren. Es soll weiterhin gezeigt werden, welche Information die Approximation liefert und in welchen Fällen sie sinnvoll ist.

Am Ende des Kapitels werden bereits existierende Verfahren basierend auf Tensoren vorgestellt. Diese Verfahren sollen der Vollständigkeit halber erwähnt werden, damit Vergleiche möglich sind und aufgezeigt werden kann, warum das zuerst vorgestellte Verfahren für Tensorprodukt Smoothing Splines besser geeignet erscheint. Theoretisch können sie ebenfalls verwendet werden, da Tensoren, wie in Kapitel 2.4 bereits gezeigt wurde, insbesondere Multilinearformen darstellen und sich Kronecker-Produkte in Tensoren umwandeln lassen.

Die meisten der vorgestellten Verfahren sind allerdings für die bei den betrachteten Splines auftretenden Datenstrukturen in der Praxis nutzlos, da bekannte Probleme — wie zum Beispiel die ineffiziente Speicherung — auftreten. Ein Verfahren existiert dennoch, welches durchaus in dieser speziellen Problemstellung angewendet werden könnte. Daher wird ein kurzer Vergleich zu dem für Multilinearformen vorgestellten Verfahren angestrebt, welcher zeigt, dass dieses Verfahren gewisse Nachteile bezüglich der Berechnung und der Darstellung hat.

3.1. Approximation einer Multilinearform

Die im Folgenden vorgestellte Annäherung einer Multilinearform ist aus [25] entnommen, welche sich wiederum auf die Approximation in [30] und [31] stützt. Die Vorarbeit zu Multilinearformen in Kapitel 2.4 wird dabei eine relevante Rolle spielen. Mit Hilfe dieser Näherung lässt sich eine Komprimierung durch Kronecker-Produkte realisieren. Durch Umstrukturierung einer Matrix in eine Multilinearform wird diese Aussage bestätigt, da mit dieser gilt, dass

$$\left\| A - \bigotimes_{j=1}^{p} X_j \right\|_F = \left\| A^\flat - vec(X_1) \otimes_t \cdots \otimes_t vec(X_p) \right\|_F . \tag{3.1}$$

Dies entspricht der Verallgemeinerung von Satz 2.9 für den höherdimensionalen Fall und lässt sich durch Ausnutzen der Umstrukturierung bzw. Nachrechnen zeigen, worauf an dieser Stelle verzichtet wird. Das Minimierungsproblem der Approximation einer Matrix durch Kronecker-Produkte lässt sich folglich als Approximation einer Multilinearform durch 1-Formen auffassen. Diese Erkenntnis wurde schon in [31] gewonnen und weiterentwickelt. Die Formulierung des Minimierungsproblems soll kurz vorgestellt werden. Für die weitere Vorgehensweise ist die Definition eines höherdimensionalen Torus notwendig

$$\mathbb{T}^{\gamma\eta} := \{(u_1, \ldots, u_p) \ : \ u_j \in \mathbb{R}^{m_j n_j}, \ \|u_j\| = 1, \ j = 1, \ldots, p\} .$$

Für die Minimierung der Norm in (3.1) lässt sich folgende Aussage im Mehrdimensionalen analog zu [25] für Multilinearformen oder [30] für Tensoren beweisen:

Satz 3.4 *Sei $A^\flat$ eine p-dimensionale Multilinearform auf $\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ sowie $X_j \in \mathbb{R}^{m_j \times n_j}$. Die Lösung des Minimierungsproblems*

$$\min_{v(X_1),\ldots,v(X_p)} \left\| A^\flat - vec(X_1) \otimes_t \cdots \otimes_t vec(X_p) \right\|_F \tag{3.2}$$

ist gegeben durch $\sigma \cdot v_1 \otimes \cdots \otimes v_p, \ \|v_j\| = 1$ und $v_j = v(X_j)$, mit

$$A^\flat[v_1, \ldots, v_p] = \max \left\{ A^\flat[u_1, \ldots, u_p] \ : \ (u_1, \ldots, u_p) \in \mathbb{T}^{\gamma\eta} \right\} =: \sigma.$$

Der Fehler der Bestapproximation beträgt

$$\min_{v_1,\ldots,v_p} \|A^\flat - v_1 \otimes_t \cdots \otimes_t v_p\|_F = \sqrt{\|A^\flat\|_F^2 - \sigma^2}.$$

Beweis: Der Beweis ist in [25] zu finden. □

Mit Hilfe dieses Minimierungsproblems, welches effektiv durch die Maximierung der Multilinearformen gelöst wird, lässt sich zeigen, dass die Multilinearform $A^\flat$ iterativ durch eine Summe von 1-Formen angenähert werden kann. Die Norm bezüglich dieser Summe nimmt mit jedem weiteren Summanden ab.

Satz 3.5 *Für eine Multilinearform $A^\flat$ auf $\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ existiert eine Folge von* 1*-Formen*

$$(v_{11} \otimes_t \cdots \otimes_t v_{1p}), \ldots, (v_{n1} \otimes_t \cdots \otimes_t v_{np}),$$

so dass gilt

$$\|A^\flat - (v_{11} \otimes_t \cdots \otimes_t v_{1p})\|_F \geq \cdots \geq \left\| A^\flat - \sum_{i=1}^{n} (v_{i1} \otimes_t \cdots \otimes_t v_{ip}) \right\|_F . \tag{3.3}$$

Beweis: Der Satz 3.4 zeigt, dass $v_1^* = (v_1^* \otimes_t \cdots \otimes_t v_p^*)$ existiert, so dass gilt

$$v_1^* = (v_{11}^* \otimes_t \cdots \otimes_t v_{1p}^*) = \arg\min_{v_1,\ldots,v_p} \|A^\flat - (v_1 \otimes_t \cdots \otimes_t v_p)\|_F.$$

Wird der Satz 3.4 iterativ auf die Differenz

$$A^\flat - \sum_{i=1}^{n-1} (v_{i1} \otimes_t \cdots \otimes_t v_{ip})$$

angewendet, ergibt sich aufgrund der zusätzlichen Freiheitsgrade und der Minimierung für jeden weiteren Schritt ein

$$v_n^* = (v_{n1}^* \otimes_t \cdots \otimes_t v_{np}^*) = \arg\min_{v_1,\ldots,v_p} \left\| \left(A^\flat - \sum_{i=1}^{n-1} (v_{i1}^* \otimes_t \cdots \otimes_t v_{ip}^*) \right) - (v_1 \otimes_t \cdots \otimes_t v_p) \right\|_F .$$

Die $v_1^*, \ldots, v_n^*$ sind eine Folge von 1-Formen

$$(v_{11} \otimes_t \cdots \otimes_t v_{1p}), \ldots, (v_{n1} \otimes_t \cdots \otimes_t v_{np})$$

mit

$$\|A^\flat - (v_{11} \otimes_t \cdots \otimes_t v_{1p})\|_F \geq \cdots \geq \left\| A^\flat - \sum_{i=1}^{n} (v_{i1} \otimes_t \cdots \otimes_t v_{ip}) \right\|_F ,$$

was zu zeigen war. □

Mittels des Minimierungsproblems in (3.2) und des nächsten Lemmas lässt sich eine Multilinearform in eine unendliche Summe von 1-Formen zerlegen. Das Lemma wurde bereits in [25] und [34] bewiesen.

Lemma 3.6 *Für jede Multilinearform $A^\flat$ existiert eine Konstante $0 < \tau < 1$, so dass die Abschätzung*

$$\max_{(v_1,\ldots,v_p)\in\mathbb{T}^{\gamma\eta}} A^\flat [v_1, \ldots, v_p] \geq \tau \|A^\flat\|_F$$

erfüllt ist.

Damit lässt sich zeigen, dass für jede Multilinearform eine unendliche Zerlegung existiert, welche auf jeden Fall konvergiert. Die Konvergenz ist relevant für die Verwendung der Approximation als Komprimierung. Diese Zerlegung bzw. Approximation lässt sich durch die wiederholte Anwendung des Satzes 3.4 aufstellen, wie bereits ansatzweise in Satz 3.5 umgesetzt. Damit ergibt sich der folgende Satz aus [25]:

Satz 3.7 *Für jede Multilinearform $A^\flat$ auf $\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ existieren nichtnegative Werte σ_k und normalisierte Vektoren $(v_1^k, \ldots, v_p^k) \in \mathbb{T}^{\gamma\eta}$, $k \in \mathbb{N}$, so dass gilt*

$$A^\flat = \sum_{k=1}^{\infty} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp} \text{ und } \sum_{k=1}^{\infty} \sigma_k^2 = \|A^\flat\|_F^2. \tag{3.4}$$

Beweis: Der Beweis ist aus [25] entnommen und soll an dieser Stelle ausgeführt werden, damit der exponentielle Abfall der Fehlerwerte ersichtlich wird. Wie bereits erwähnt, lässt sich die Zerlegung in (3.4) durch wiederholtes Anwenden des Satzes 3.4 in Form eines „greedy"-Algorithmuses bestimmen. Sei $(v_1^k, \ldots, v_p^k) \in \mathbb{T}^{\gamma\eta}$ eine Lösung des Minimierungsproblems aus (3.2). Für den Ausdruck

$$A^\flat - \sum_{k=1}^{\ell} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp}$$

gilt dann

$$\left\| A^\flat - \sum_{k=1}^{\ell} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp} \right\|_F^2 = \|A^\flat\|_F^2 - \sum_{k=1}^{\ell} \sigma_k^2.$$

Dies lässt sich induktiv zeigen, indem die Berechnung des Fehlerwerts aus Satz 3.4 sowie

$$\langle A^\flat - \sigma v_{k1} \otimes_t \cdots \otimes_t v_{kp}, \sigma v_{k1} \otimes_t \cdots \otimes_t v_{kp} \rangle = 0$$

verwendet wird. Mit Lemma 3.6 folgt, dass

$$\begin{aligned}
\|A^\flat\|_F^2 - \sum_{k=1}^{\ell} \sigma_k^2 &= \left\| A^\flat - \sum_{k=1}^{\ell-1} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp} \right\|_F^2 - \sigma_\ell^2 \\
&= \left\| A^\flat - \sum_{k=1}^{\ell-1} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp} \right\|_F^2 \\
&\quad - \left\langle A^\flat - \sum_{k=1}^{\ell-1} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp}, v_{\ell 1} \otimes_t \cdots \otimes_t v_{\ell p} \right\rangle \\
&\leq (1-\tau^2) \left\| A^\flat - \sum_{k=1}^{\ell-1} \sigma_k v_{k1} \otimes_t \cdots \otimes_t v_{kp} \right\|_F^2 \leq \cdots \leq (1-\tau^2)^\ell \|A^\flat\|_F^2.
\end{aligned}$$

Daher gilt

$$\|A^\flat\|_F^2 \geq \sum_{k=1}^{\ell} \sigma_k^2 \geq \|A^\flat\|_F^2 \left(1 - (1-\tau^2)^\ell\right), \tag{3.5}$$

was die Aussage beweist. □

Mit der Formel (3.5) folgt, dass

$$\sigma_k \leq (1-\tau^2)^{(k-1)/2} \|A^\flat\|_F. \tag{3.6}$$

Daraus wird ersichtlich, dass die σ_k exponentiell abfallen, auch wenn die Zerlegung unendlich ist. Der exponentielle Abfall ist allerdings asymptotisch zu verstehen und τ ist in der praktischen Anwendung in den meisten Fällen sehr klein, so dass die Konvergenzrate vergleichsweise langsam ausfällt. Diese Konvergenzrate ist in bestimmten Fällen für die Komprimierung durchaus ausreichend, wie die Ergebnisse in Kapitel 6.1 zeigen. Offensichtlich lassen sich die obigen Resultate ebenso für Matrizen und Kronecker-Produkte formulieren, analog zu [25].

Korollar 3.8 *Für jede Matrix* $A \in \mathbb{R}^{m_1 \cdots m_p \times n_1 \cdots n_p}$ *existieren Matrizen* $X_{kj} \in$

$\mathbb{R}^{m_j \times n_j}$, $j = 1, \ldots, p$ *und exponentiell abfallende Werte* $\sigma_k \geq 0$, *so dass*

$$\left\| A - \sum_{k=1}^{\ell} \sigma_k \bigotimes_{j=1}^{p} X_{kj} \right\|_F^2 = \sum_{k=\ell+1}^{\infty} \sigma_k^2 \leq \frac{(1-\tau^2)^\ell}{\tau^2} \|A\|_F^2$$

für $\tau \in (0, 1)$.

In der Praxis reichen in bestimmten Fällen wenige Summanden aus, im Vergleich zu der eigentlichen Summandenanzahl, um akzeptable Fehlerwerte zu erhalten. Die bestimmten 1-Formen lassen sich in Kronecker-Produkte umwandeln und damit ergibt sich eine Approximation für die Matrix A. In [25] wird weiterhin eine endliche, orthogonale Zerlegung für Multilinearformen aufgezeigt. Auf diese wird hier verzichtet, da sie für die benötigte Komprimierung nicht praxisrelevant ist. Eine orthogonale Zerlegung bringt keine weiteren Vorteile, da primär eine schnelle Approximation von Interesse ist und die orthogonale Zerlegung zusätzlich mehr Zeit zur Bestimmung benötigt. Der letzte Baustein, der an dieser Stelle noch fehlt, ist ein effizienter Algorithmus zur Bestimmung der Folge von 1-Formen in (3.3).

3.2. Algorithmus zur Approximation einer Multilinearform

Wie in Satz 3.4 deutlich wird, ist die Bestimmung der Bestapproximation einer Multilinearform durch eine 1-Form mit Hilfe der Maximierung der Multilinearform über den Torus $\mathbb{T}^{\gamma\eta} = \mathbb{S}^{m_1 n_1} \times \cdots \times \mathbb{S}^{m_p n_p}$ möglich, wobei der Ausdruck

$$\mathbb{S}^n := \{x \in \mathbb{R}^n \ : \ \|x\|_2 = 1\}$$

die n-dimensionale Einheitssphäre beschreibt. Für die Maximierung einer Multilinearform lässt sich nach [25] ein wichtiges Lemma aufstellen, mit welchem ein Algorithmus zur Bestimmung der Zerlegung und damit der Komprimierung einer Multilinearform formuliert werden kann. Die Maximierung der Multilinearform, aufgefasst als eine Abbildung nach $\mathbb{R}$, entspricht, wie bereits erwähnt, der Minimierung der Norm in (3.2). Das Lemma stammt aus [25] und ist für Tensoren in [30] zu finden.

Lemma 3.9 *Sei $A^\flat$ eine p-dimensionale Multilinearform auf $\mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ und $j \in \{1, \ldots, p\}$. Die Linearform*

$$A^\flat_{(j)}[z] := A^\flat[u_1, \ldots, u_{j-1}, z, u_{j+1}, \ldots, u_p]$$

besitzt ein Maximum auf $\mathbb{S}^{m_j n_j}$ bei $\|z_j\|^{-1} z_j$, wobei

$$z_j = \left[\sum_{\alpha_j = 1}^{m_j n_j} x_\alpha u_{1,\alpha_1} \cdots u_{j-1,\alpha_{j-1}} \cdot u_{j+1,\alpha_{j+1}} \cdots u_{p,\alpha_p} \right]. \tag{3.7}$$

Beweis: Siehe [25]. □

Mit Hilfe von diesem Lemma lässt sich ein Ansatz erkennen, um ein Maximum der Multilinearform zu berechnen, in dem die Lösungen der komponentenweisen Maximierungsprobleme verwendet werden. Dieser Ansatz ersetzt in jedem Schritt u_j durch $z_j / \|z_j\|$, wodurch die Zielfunktion nach Lemma 3.9 in jedem Schritt vergrößert wird. Für den in Kapitel 2.1 bzw. in [32] beschriebenen Fall mit zwei Faktoren entspricht dieses Verfahren der Potenzmethode, welche unter anderem in [15] erklärt wird.

Dieser Ansatz lässt sich ohne Probleme auf höhere Dimensionen und Multilinearformen erweitern. Damit kann das Maximum einer Multilinearform berechnet und mittels diesem eine Annäherung an die Multilinearform bestimmt werden. Bei dieser Methode handelt es sich um ein bekanntes Standardverfahren und sie ist in sehr ähnlicher Art als „Alternating Least-squares"-Verfahren (ALS) oder als „Higher Order Power Method" in der multilinearen Algebra bekannt, siehe [3, 30]. Die Idee des ALS-Verfahrens soll kurz beschrieben werden, sie wird später in Kapitel 4 genauer erläutert. Die Methode bestimmt einen kritischen Punkt eines mehrdimensionalen Minimierungsproblems, in dem sie immer nur eine Komponente als Variable und die anderen Komponenten als Parameter betrachtet. Sie minimiert dann das entstehende univariate Problem mit Hilfe eines „Least-squares"-Problems. Durch zyklisches Wechseln der Variablen wird so in den meisten Fällen ein globales Minimum bestimmt, wobei dies nicht garantiert werden kann. Gegenbeispiele lassen sich durch ein nicht lineares Optimierungsproblem konstruieren, indem alle Matrixeinträge als separate Variablen betrachtet werden und nicht die einzelnen Faktoren des Kronecker-Produktes als Ganzes. Die Bezeichnung „Alternating" bezieht sich auf das Wechseln der Variablen

Algorithmus 3.1 : Zyklische höherdimensionale Potenzmethode für Multilinearformen

Input : p-dimensionale Multilinearform $A^\flat$, $\varepsilon > 0$ Fehlertoleranz
Output : „Singulärwert“ $\sigma = A^\flat[u_1^\infty, \ldots, u_p^\infty]$ und „Singulärvektoren“ $v_j = u_j^\infty$.

begin
  Initialisierung von $u_j^0 \in \mathbb{S}^{m_j n_j}$, $j = 1, \ldots, p$, so dass $A^\flat[u_1^0, \ldots, u_p^0] > 0$.
  Laufindex $s := 1$ und $\tau_0 := A^\flat[u_1^0, \ldots, u_p^0] + \varepsilon$ sowie $\tau_1 := A^\flat[u_1^0, \ldots, u_p^0]$
  while $|\tau_s - \tau_{s-1}| > \varepsilon$ **do**
    for $j = 1, \ldots, p$ **do**
      Berechne $z^{sp+j} := z_j$, unter Verwendung von Lemma 3.9 mit $u_\ell := u_\ell^s$, $\ell = 1, \ldots, p$ und $\ell \neq j$.
      Setze $u_j^{s+1} := z_j / \|z_j\|$.
    end
    Setze $\tau_{s+1} := A^\flat[u_1^{s+1}, \ldots, u_p^{s+1}]$.
    $s := s + 1$
  end
  Setze $\sigma := \tau_\infty$ und $v_j := u_j^\infty$.
end

bei der Bestimmung der einzelnen „Least-squares“-Probleme. Die beiden oben genannten Algorithmen wurden im Zusammenhang mit Tensoren entwickelt, was erneut auf den engen Zusammenhang zwischen Multilinearformen und Tensoren hinweist. Mit den bewiesenen Aussagen ergibt sich der Algorithmus 3.1 der höherdimensionalen Potenzmethode für Multilinearformen. Wenn die Multilinearformen durch Tensoren ersetzt werden, entspricht dieser Algorithmus den erwähnten Algorithmen für Tensoren.

Mit Hilfe einer wiederholten Anwendung des angegebenen Algorithmus lässt sich eine Approximation durch eine Summe von Kronecker-Produkten bestimmen. Für die Bestimmung eines einzelnen Kronecker-Produkts macht die Reihenfolge des Laufindexes j keinen Unterschied. Es werden allerdings unterschiedlich viele Iterationen und Rechenoperationen verwendet. Empirische Tests deuten an, dass die zyklische Minimierung am effizientesten ist. In dem Fall, dass die zu approximierende Matrix bereits einer Summe von Kronecker-Produkten entspricht, ist der Algorithmus sehr effizient umsetzbar, wie folgende Bemerkung zeigt:

Bemerkung 3.10 *Der Algorithmus 3.1 ist im Bereich der Tensorprodukt Smoothing Splines sehr effizient, da die zugehörige Matrix des LGS in* (2.12) *bereits als Summe von Kronecker-Produkten darstellbar ist. Die entsprechende Multilinearform ergibt sich folglich als eine Summe von* 1*-Formen*

$$A^\flat = \sum_{k=1}^{N} a_{1k} \otimes_t \cdots \otimes_t a_{pk},$$

und dadurch müssen bei der Berechnung der z_j *nur die* $N \cdot p$ *Werte* $a_{jk}^t u_j$, $j = 1, \ldots, p,\ k = 1, \ldots, N$ *neu berechnet werden. Dies liegt daran, dass in jeder Iteration nur ein* u_j *aktualisiert wird und die anderen Werte erhalten bleiben. Bei der Auswertung einer Multilinearform handelt es sich in diesem Fall um eine Multiplikation von Skalarprodukten der univariaten Komponenten*

$$A^\flat [u_1, \ldots, u_p] = \sum_{k=1}^{N} \prod_{j=1}^{p} \langle a_{jk}, u_j \rangle.$$

Bezüglich der Maximierung lässt sich sagen, dass in der Zeile 7 des Algorithmuses 3.1 der Wert der Multilinearform immer erhöht wird. Es lassen sich aber keine Aussagen über die Konvergenz treffen, da es sich um ein ALS-Verfahren handelt. Dies wurde bereits in [36] gezeigt.

3.3. Effizienz der Komprimierung bei multivariaten Smoothing Splines

Der Algorithmus 3.1 lässt sich aufgrund der strukturellen Vorteile sehr effektiv auf Tensorprodukt Smoothing Splines anwenden. Die resultierende Komprimierung der Multilinearform kann zwar die Originaldaten zum Beispiel beim Lösen des LGS nicht direkt ersetzen, eignet sich aber zur Approximation einer Inversen oder zur Bestimmung eines Vorkonditionieres. Allerdings gibt es gewisse Einschränkungen, welche hier aufgezeigt werden sollen.

Die Faktoren des Kronecker-Produkts der Matrix des Tensorprodukt Smoothing Splines in (2.12) bestehen hauptsächlich aus symmetrischen Rang-1 Matrizen, wie in (2.13) deutlich gemacht wurde. Der Aufbau dieser Rang-1 Matrizen ist bekannt und für die Faktoren muss lediglich ein Vektor $a \in \mathbb{R}^{n_j}$ pro Matrix $A_j = aa^t$, mit $A_j \in \mathbb{R}^{n_j \times n_j}$ gespeichert werden. Aufgrund dieser

Darstellungsmöglichkeit müssen keine vollbesetzten Matrizen für die Faktoren der Kronecker-Produkte verwendet werden. Dadurch ist die Darstellung sehr speicher- und recheneffizient, womit die Menge an Speicherplatz bzw. die Anzahl an Rechenoperationen gemeint ist. Im Gegensatz dazu verwendet die Approximation vollbesetzte Matrizen für die Kronecker-Faktoren. Bei der Komprimierung muss daher beachtet werden, dass die Speicher- und Recheneffizienz erhalten bleibt. Damit die Komprimierung überhaupt sinnvoll ist, sollten die Menge an Speicherplatz und die Anzahl der Rechenoperationen kleiner sein als bei den ursprünglichen Daten.

Es lassen sich für beide Fälle obere Grenzen für die Anzahl der Summanden der Komprimierung bestimmen. Der Approximationsterm des Tensorprodukt Smoothing Splines spielt für diese Abschätzung eine wesentliche Rolle, da dieser so viele Summanden besitzt wie es Stützstellen gibt und zudem ausschließlich aus den zuvor beschriebenen symmetrischen Rang-1 Matrizen besteht. Dies ist in der Darstellung (2.13) nachzuvollziehen. Die Matrizen des Glättungsterms sind hingegen nicht aus Rang-1 Matrizen aufgebaut. Sie fallen aber bezüglich der Speicherung nicht ins Gewicht, da ihre Summandenanzahl nur von der Dimension abhängt. Diese Anzahl ist sehr gering im Vergleich zu dem Approximationsterm.

Satz 3.11 (Speichereffizienz) *Sei N die Anzahl der Stützstellen, n_j die Anzahl der Knoten pro Dimension, $j = 1, \ldots, p$, und k die Ordnung der Ableitung des Glättungsanteils sowie s die Anzahl der Summanden der Komprimierung. Die Komprimierung ist speichereffizient, wenn gilt*

$$\frac{n_+(N + kn_+)}{n_-^2} > s, \tag{3.8}$$

wobei $n_+ := \max\{n_j \ : \ j = 1, \ldots, p\}$ *und* $n_- := \min\{n_j \ : \ j = 1, \ldots, p\}$.

Beweis: Für die Speicherung eines Tensorprodukt Smoothing Splines muss folgende Anzahl an Matrixeinträgen gespeichert werden

$$\underbrace{\sum_{j=1}^{p} N \cdot n_j}_{\text{Approximationsterm}} \quad + \quad \underbrace{\sum_{j=1}^{p} k \cdot n_j^2}_{\text{Glättungsterm}}$$

und für die Anzahl der Einträge der Komprimierung gilt

$$\sum_{j=1}^{p} s \cdot n_j^2.$$

Die Komprimierung ist folglich genau dann speichereffizient, wenn gilt

$$\sum_{j=1}^{p} N \cdot n_j + \sum_{j=1}^{p} k \cdot n_j^2 > \sum_{j=1}^{p} s \cdot n_j^2. \tag{3.9}$$

Dieser Term lässt sich weiter abschätzen, um eine übersichtlichere Formel zu erhalten. Dafür wird der Term auf der linken Seite von (3.9) bzw. alle n_j nach oben mit $n_+ = \max\{n_j : \ j = 1, \ldots, p\}$ abgeschätzt. Es folgt

$$pn_+(N + kn_+) \geq \sum_{j=1}^{p} N \cdot n_j + \sum_{j=1}^{p} k \cdot n_j^2.$$

Weiterhin wird die rechte Seite von (3.9) bzw. alle n_j nach unten mit $n_- = \min\{n_j : \ j = 1, \ldots, p\}$ abgeschätzt. Damit ergibt sich

$$\sum_{j=1}^{p} s \cdot n_j^2 \geq psn_-^2.$$

Insgesamt lässt sich folgern, dass

$$\begin{aligned} & pn_+(N + kn_+) > psn_-^2 \\ \Rightarrow \ & \frac{n_+(N + kn_+)}{n_-^2} > s. \end{aligned}$$

Dies entspricht der Behauptung. □

Bemerkung 3.12 *Die Abschätzung bzgl. der Speichereffizienz ist unabhängig von der Dimension p. Dies hängt mit dem linearen Wachstum des Speichers aufgrund der Kronecker-Darstellung zusammen.*

Für die Recheneffizienz lässt sich eine ähnliche Aussage bzgl. der maximalen Anzahl der Summanden der Komprimierung treffen.

Satz 3.13 (Recheneffizienz) *Sei N die Anzahl der Stützstellen, n_j die Anzahl der Knoten pro Dimension, $j = 1, \ldots, p$, sowie s die Anzahl der Summanden der Komprimierung. Für den Glättungsterm wird die Frobeniusnorm der Hesse-*

matrix des Tensorprodukt Splines verwendet. Die Komprimierung ist recheneffizient, wenn gilt

$$\frac{n_+^p \left(N\left(2n_+ - 1 + \frac{1}{p}\right) + \frac{p(p+1)}{2}(2n_+ - 1)\right)}{n_-^p(2n_- - 1)} > s, \tag{3.10}$$

wobei $n_+ := \max\{n_j \ : \ j = 1, \ldots, p\}$ *und* $n_- := \min\{n_j \ : \ j = 1, \ldots, p\}$.

Beweis: Für die Anzahl der Rechenoperationen einer Matrix-Vektor-Multiplikation bei der Matrix des Tensorprodukt Smoothing Splines gilt nach [24]

$$N \cdot \left(\sum_{j=1}^{p}\left(\prod_{i=j+1}^{p} n_i(2n_j - 1)\right) + \prod_{j=1}^{p} n_j\right) + \frac{p(p+1)}{2}\prod_{j=1}^{p} n_j \left(2\sum_{j=1}^{p} n_j - p\right).$$

Der erste Summand entspricht der Anzahl der Rechenoperationen für die Multiplikation eines Vektors mit der Matrix des Approximationsterms. Der zweite Summand steht für eine Multiplikation mit dem Glättungsterm. Weiterhin gilt für die Anzahl der Rechenoperationen einer Matrix-Vektor-Multiplikation bei der Komprimierung

$$s\prod_{j=1}^{p} n_j \left(2\sum_{j=1}^{p} n_j - p\right).$$

Eine Komprimierung ist genau dann recheneffizient, wenn gilt

$$\begin{aligned} N \cdot \left(\sum_{j=1}^{p}\left(\prod_{i=j+1}^{p} n_i(2n_j - 1)\right) + \prod_{j=1}^{p} n_j\right) + \frac{p(p+1)}{2}\prod_{j=1}^{p} n_j \left(2\sum_{j=1}^{p} n_j - p\right) \\ > s\prod_{j=1}^{p} n_j \left(2\sum_{j=1}^{p} n_j - p\right). \end{aligned} \tag{3.11}$$

Bei der linken Seite der Ungleichung (3.11) werden alle n_j nach oben durch $n_+ = \max\{n_j : \ j = 1, \ldots, p\}$ abgeschätzt. Damit folgt

$$\begin{aligned} &N \cdot \left(\sum_{j=1}^{p}\left(\prod_{i=j+1}^{p} n_i(2n_j - 1)\right) + \prod_{j=1}^{p} n_j\right) + \frac{p(p+1)}{2}\prod_{j=1}^{p} n_j \left(2\sum_{j=1}^{p} n_j - p\right) \\ &\leq N\left(p(n_+^p(2n_+ - 1)) + n_+^p\right) + \frac{p(p+1)}{2} n_+^p(2pn_+ - p) \\ &= pn_+^p \left(N\left(2n_+ - 1 + \frac{1}{p}\right) + \frac{p(p+1)}{2}(2n_+ - 1)\right) \end{aligned}$$

Für die rechte Seite werden alle n_j nach unten durch $n_- = \min\{n_j : j = 1, \ldots, p\}$ abgeschätzt. Es folgt

$$s \prod_{j=1}^{p} n_j \left(2 \sum_{j=1}^{p} n_j - p \right) \geq s n_-^p (2pn_- - p) = spn_-^p (2n_- - 1).$$

Insgesamt folgt dann die Behauptung durch

$$\begin{aligned} & pn_+^p \left(N \left(2n_+ - 1 + \frac{1}{p} \right) + \frac{p(p+1)}{2} (2n_+ - 1) \right) > spn_-^p (2n_- - 1) \\ \Rightarrow & \frac{n_+^p \left(N \left(2n_+ - 1 + \frac{1}{p} \right) + \frac{p(p+1)}{2} (2n_+ - 1) \right)}{n_-^p (2n_- - 1)} > s. \end{aligned}$$

□

Die Abschätzung bzgl. der Recheneffizienz kann noch stärker formuliert werden, der Übersichtlichkeit halber wird jedoch darauf verzichtet.

Korollar 3.14 *Wenn in jeder Dimension die gleiche Anzahl $\tilde{n}$ an Knoten verwendet wird, können die Abschätzungen aus Satz 3.11 und 3.13 wie folgt vereinfacht werden: Die Komprimierung ist in dem Fall speichereffizient, wenn gilt*

$$\frac{N}{\tilde{n}} + k > s$$

und recheneffizient, wenn

$$\frac{p(p+1)}{2} + N \left(1 + \frac{1}{p(2\tilde{n} - 1)} \right) > s.$$

erfüllt ist.

Bemerkung 3.15 *Bei der Bestimmung des Maximums einer Multilinearform in Algorithmus 3.1 (Zeile 6) bzw. durch Anwendung von Lemma 3.9 kann bei den Tensorprodukt Smooting Splines ausgenutzt werden, dass pro Iterationsschritt nur ein Skalarprodukt mit dem zuletzt berechneten Faktor des Kronecker-Produkts berechnet werden muss. Die anderen Skalarprodukte mit den Kronecker-Faktoren können gespeichert werden, sodass Rechenoperationen eingespart werden.*

Weiterhin lässt sich die Rang-1 Struktur des Approximationsterms ausnutzen, wenn nicht genügend Speicherplatz zur Verfügung steht. Die Anzahl der Rechen-

*operationen des Skalarprodukts pro Iterationsschritt beträgt unter Ausnutzung der Rang-*1 *Struktur* $2n_j^2+3n_j-1$. *Wenn zuerst die Matrix des Kronecker-Faktors berechnet wird und damit das Skalarprodukt, dann werden* $3n_j^2 - 1$ *Rechenoperationen benötigt. Wenn die Kronecker-Faktoren gespeichert werden können, reduziert sich die Anzahl der Rechenoperationen auf* $2n_j^2 - 1$. *Dies sind weniger Rechenoperationen als bei der Ausnutzung der Rang-*1 *Struktur und bei genügend Speicherplatz sollte das ausgenutzt werden.*

In numerischen Tests lässt sich feststellen, dass sich die Komprimierung als sehr effizient herausstellt, wenn der Glättungsanteil mehr Einfluss als der Approximationsanteil hat. Dies belegen die Ergebnisse in Kapitel 6.1. Mit anderen Worten, wenn der Glättungsparameter eine bestimmte Grenze überschreitet, dann lässt sich zumindest zur approximativen Verwendung eine effiziente Komprimierung finden.

Vermutung 3.16 *Die Güte der Komprimierung hängt davon ab, wie stark der Approximations- bzw. Glättungsterm gewichtet wird. Dies steht im Zusammenhang mit der Wahl des Glättungsparameters* λ. *An dieser Stelle soll versucht werden, einen Schwellenwert anzugeben, sodass direkt entschieden werden kann, wann eine Komprimierung überhaupt sinnvoll scheint. Es wird zuerst die Matrix des Tensorprodukt Smoothing Splines aus* (2.12)

$$N^{\mu}(X|T)^t N^{\mu}(X|T) + \lambda G^{\mu}(T)$$

im Ganzen betrachtet. Die Matrix wird als Multilinearform aufgefasst und mittels Algorithmus 3.1 bzw. Lemma 3.9 wird ihr Maximum σ *bestimmt. Danach wird die Matrix in den Approximations- und Glättungsterm aufgeteilt. Die beiden Matrizen* $N^{\mu}(X|T)^t N^{\mu}(X|T)$ *und* $G^{\mu}(T)$ *werden analog als Multilinearformen aufgefasst und ihre Maxima bestimmt. Der Wert* σ_A *steht für das Maximum des Approximationsterms und* σ_G *für das des Glättungsterms. Insgesamt sollte ungefähr gelten*

$$\sigma \approx \sigma_A + \lambda \sigma_G.$$

Eine Gleichgewichtung des Approximations- und Glättungsterms würde eintreten, wenn für $\lambda = \frac{\sigma_A}{\sigma_G}$ *gilt. Numerische Tests haben gezeigt, dass die Komprimierung effizient ist, wenn der Glättungsterm stärker gewichtet wird. Dies lässt vermuten, dass eine Komprimierung nur Sinn macht, wenn* $\lambda > \frac{\sigma_A}{\sigma_G}$. *Ein sinnvolles Vorgehen wäre:*

1. *Bestimmung des kritischen Werts σ_A/σ_G.*
2. *Unter Ausnutzung der Komprimierung testen, ob dieser Wert für λ zufriedenstellende Ergebnisse bzgl. der Splineapproximation liefert.*
3. *In Abhängigkeit der Ergebnisse den Approximations- bzw. Glättungsterm stärker gewichten und abhängig davon die Komprimierung nutzen.*

Die Ergebnisse bezüglich der Komprimierung finden sich in Kapitel 6.1.

3.4. Existierende Verfahren für Tensoren

Wie bereits im Unterkapitel 2.4 gesehen, besteht eine enge Verbindung zwischen Kronecker-Produkten und Tensoren. Da zur Bestimmung der höherdimensionalen Varianten von Singulärwerten und -vektoren bzgl. Tensoren bereits eine Vielzahl von Verfahren existiert und diese zur Komprimierung von Tensoren und Kronecker-Produkten verwendet werden können, sollen sie an dieser Stelle vorgestellt werden. Dabei wird auf das Verfahren der „Higher Order Power Method“ verzichtet, da es dem Verfahren für Multilinearformen aus dem Unterkapitel 3.2 entspricht. Für die vorgestellten Methoden soll aufgezeigt werden, warum diese Verfahren für die Komprimierung im Zusammenhang mit Tensorprodukt Smoothing Splines nur theoretisch geeignet sind.

Für die Tensoren gelten die Notationen und Aussagen aus Kapitel 2.4. Zuerst sollen grundlegende Methoden vorgestellt werden, welche die Basis für ein späteres Verfahren darstellen. Da Tensoren in vielen Bereichen — zum Beispiel Big Data, der Physik und der Chemie (siehe [18]) — zu finden sind, gibt es zahlreiche Namen für ähnliche Methoden. Ein guter Überblick ist in [23] und [18] zu finden. Die folgenden Verfahren, Definition, Notationen und Aussagen sind daraus entnommen oder lehnen sich daran an, wenn nicht anderweitig gekennzeichnet.

Die erste Methode wird als CP-ALS bezeichnet und ist in ihrer ersten Form unter dem Namen CANDECOMP [8] bzw. PARAFAC [19] bekannt. Sie wurde später von de Lathauwer und anderen weiterentwickelt [29, 30] und ist allgemein als ein „Alternating Least-squares“-Verfahren zur Berechnung einer Rang-1 Zerlegung für einen Tensor bekannt. Die *CP*-Zerlegung für $\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ hat folgende Gestalt

$$\mathcal{X} = \sum_{i=1}^{R} \sigma_i (v_{i1} \otimes_t \cdots \otimes_t v_{ip}) \text{ mit } v_{ij} \in \mathbb{R}^{m_j n_j}.$$

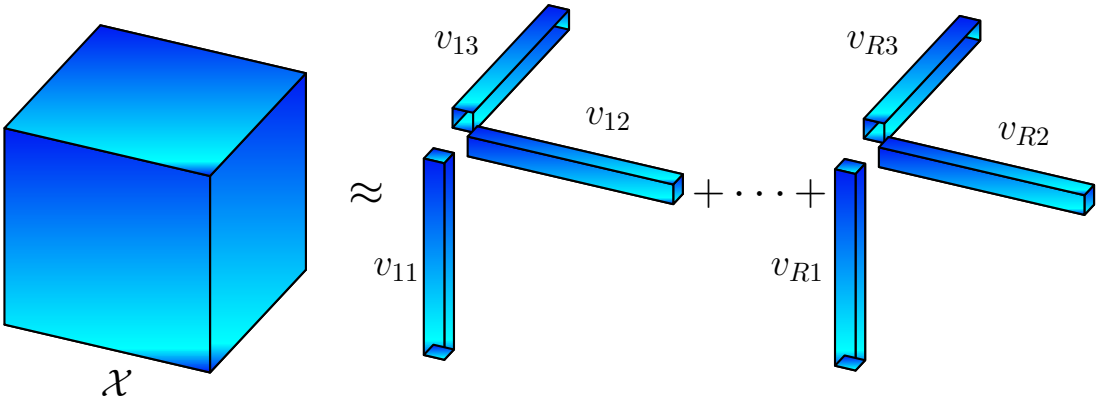

Abbildung 3.1.: Zerlegung mittels CP-ALS

Die Abbildung veranschaulicht die Zerlegung eines dreidimensionalen Tensors mittels des CP-ALS Verfahrens. Die Idee für die Abbildung stammt aus [23].

Bei $(v_{i1} \otimes_t \cdots \otimes_t v_{ip})$, $i = 1, \ldots, R$, handelt es sich um Rang-1 Tensoren. Anschaulich ist diese Zerlegung in Abbildung 3.1 dargestellt. Bevor der Algorithmus angegeben werden kann, bedarf es der Definition eines weiteren Matrix-Produkts.

Definition 3.17 (Hadamard-Produkt) *Seien $A \in \mathbb{R}^{n \times k}$ und $B \in \mathbb{R}^{n \times k}$ Matrizen. Das* Hadamard-Produkt *zweier Matrizen ist definiert durch das komponentenweise Produkt der Matrizen*

$$A \circ_H B := \begin{bmatrix} a_{11}b_{11} & \cdots & a_{1k}b_{1k} \\ \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & \cdots & a_{nk}b_{nk} \end{bmatrix} \in \mathbb{R}^{n \times k}.$$

Beispiel 3.18 *Gegeben seien die Matrizen*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ sowie } B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

Für das Hadamard-Produkt gilt

$$A \circ_H B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \circ_H \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 12 \\ 21 & 32 \end{bmatrix}.$$

Algorithmus 3.2 : Algorithmus für das CP-ALS Verfahren

Input : $\mathcal{X}$ Tensor, $R = \text{rang}(\mathcal{X})$
Output : $\lambda,\ V^{(1)}, \ldots,\ V^{(p)}$

begin
  Initialisiere $V^{(j)} \in \mathbb{R}^{m_j n_j \times R}$ für $j = 1, \ldots, p$, z.B. mit Zufallszahlen.
  while *Bis* λ *sich nicht mehr ändert* **do**
    for $j = 1, \ldots, p$ **do**
      $A := (V^{(1)})^t V^{(1)} \circ_H \cdots \circ_H (V^{(j-1)})^t V^{(j-1)} \circ_H (V^{(j+1)})^t V^{(j+1)} \circ_H \cdots \circ_H (V^{(p)})^t V^{(p)}$
      $V^{(j)} := \mathcal{M}_j(\mathcal{X}) \left(V^{(p)} \odot \cdots \odot V^{(j+1)} \odot V^{(j-1)} \odot \cdots \odot V^{(1)}\right) A^\dagger$
      $\lambda := \|V^{(j)}\|_F$.
    end
  end
end

Des Weiteren wird für den Algorithmus die Moore-Penrose-Inverse benötigt.

Definition 3.19 (Moore-Penrose-Inverse) *Die* Moore-Penrose-Inverse *einer Matrix* $A \in \mathbb{R}^{m \times n}$ *mit der SVD* $A = U\Sigma V^t$ *aus Satz 2.10 ist definiert als*

$$A^\dagger := V\Sigma^\dagger U^t, \quad \Sigma^\dagger \in \mathbb{R}^{n \times m},$$

wobei

$$(\Sigma^\dagger)_{jk} := \begin{cases} \sigma_{jk}^{-1}, & \sigma_{jk} \neq 0, \\ 0 & \sigma_{jk} = 0, \end{cases} \quad j = 1, \ldots, n,\ k = 1, \ldots, m.$$

Damit lässt sich der Algorithmus 3.2 für das CP-ALS Verfahren aus [23] angeben. Mit $\odot$ ist das Khatri-Rao-Produkt aus Definition 2.22 gemeint.

Wie bereits erwähnt, basieren die „Higher Order Power Method“ und der Algorithmus 3.1 ebenfalls auf dem Prinzip des ALS. Eine weitere Alternative zur Komprimierung eines Tensors ist die „Higher Order SVD“ (HOSVD). Für dieses Verfahren existieren ebenfalls viele Namen, da es in verschiedenen Bereichen Anwendung findet, wie zu Beginn des Unterkapitels erwähnt wurde. Eine sehr geläufige Bezeichnung für diese Zerlegung ist die Tucker-Zerlegung (siehe [23]). Eine alternative Definition des Tensorrangs zu der in Kapitel 2.4 (siehe Definition 2.36) muss eingeführt werden, um die Zerlegung beschreiben zu können. Dieser Tensorrang wird j-Rang genannt und bezieht sich auf die einzelnen Dimensionen des Tensors.

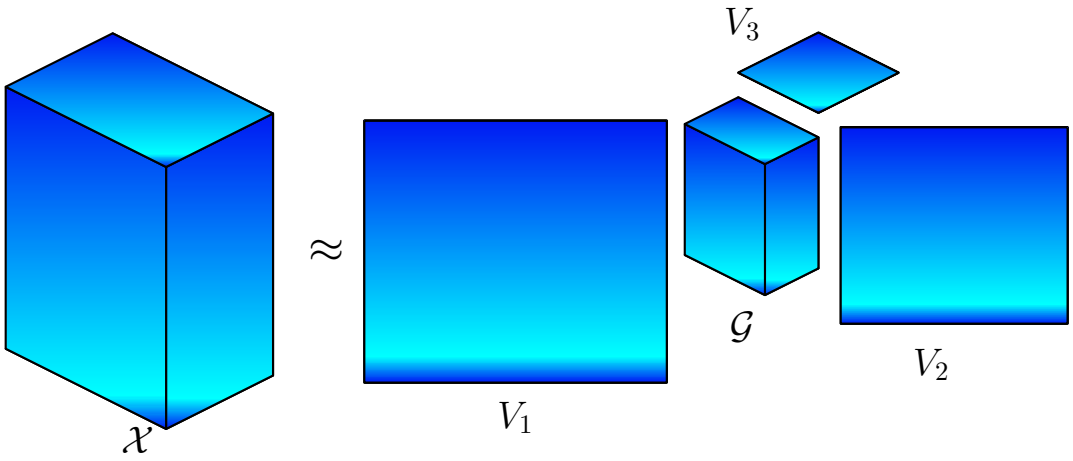

Abbildung 3.2.: Zerlegung mittels HOSVD

Die Grafik skizziert die Zerlegung eines dreidimensionalen Tensors mittels des Verfahrens für die HOSVD. Sie ist angelehnt an eine Skizze aus [23].

Definition 3.20 (j-Rang) *Sei $\mathcal{X} \in \mathbb{R}^{m_1 \times \cdots \times m_p}$. Der j-Rang eines Tensors ist definiert als*

$$rang_j(\mathcal{X}) := \operatorname{rang}\left(\mathcal{M}_j(\mathcal{X})\right).$$

Für Matrizen, also zweidimensionale Tensoren, ist der Rang und j-Rang identisch. Weitere Zusammenhänge bzgl. der unterschiedlichen Rang-Definitionen eines Tensors sind in [23] zu finden. Die Tucker-Zerlegung für einen Tensor $\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ hat folgende Gestalt

$$\mathcal{X} \approx \mathcal{G} \circ_1 V^{(1)} \circ_2 V^{(2)} \circ_3 \cdots \circ_p V^{(p)},$$

mit $V^{(j)} \in \mathbb{R}^{m_j n_j \times R_j}$ und $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_p}$. Dabei sind $V^{(j)}$, $j = 1, \ldots, p$, die „Faktor-Matrizen“, sie entsprechen der höherdimensionalen Variante der Singulärvektoren, also dem Analogon zu U und V in zwei Dimensionen. Die Variable $\mathcal{G}$ wird als „Core-Tensor“ bezeichnet, welcher als das höherdimensionale Analogon von Σ aufzufassen ist und die entsprechenden Singulärwerte des Tensors beinhaltet. Der „Core-Tensor“ ist dabei komplett besetzt und nicht nur wie im Zweidimensionalen auf der Diagonalen. Die Tucker-Zerlegung ist für einen dreidimensionalen Tensor anschaulich in Abbildung 3.2 dargestellt. Mit Hilfe der SVD aus Satz 2.10 ergibt sich der Algorithmus 3.3 der HOSVD aus [23].

Algorithmus 3.3 : Algorithmus für die „Higher Order SVD“

Input : $\mathcal{X}$ Tensor, $R_1, \ldots, R_p$ gewünschte Mächtigkeit für die j-Ränge der einzelnen Dimensionen mit $R_j \leq \operatorname{rang}_j(\mathcal{X})$

Output : $\mathcal{G}$, $V^{(1)}, \ldots, V^{(p)}$

1 **begin**
2 **for** $j = 1, \ldots, p$ **do**
3 $V^{(j)} := U(:, 1 : R_j)$ von $\mathcal{M}_j(\mathcal{X}) = U\Sigma V^t$.
4 **end**
5 $\mathcal{G} := \mathcal{X} \circ_1 (V^{(1)})^t \circ_2 \cdots \circ_p (V^{(p)})^t$.
6 **end**

Dieser Algorithmus wurde bereits in [37] auf eine Summe von Kronecker-Produkten angewendet, allerdings handelte es sich dabei um den Fall $p = 3$ und es wurden wesentlich kleinere Datenmengen untersucht. Die Ergebnisse dieser Untersuchung lassen sich folglich nicht auf die Problemstellung in dieser Arbeit übertragen. Die beiden vorgestellten Algorithmen können nur theoretisch auf den betrachteten Problemfall angewendet werden. Dies liegt daran, dass die Matricization $\mathcal{M}_j(\mathcal{X})$ zwar durch die Kronecker-Struktur dargestellt werden kann, aber eine SVD von dieser die Kronecker-Struktur nicht erhält und somit die Speicherkapazität zur Berechnung der obigen Algorithmen nicht ausreicht. In Algorithmus 3.2 kann auch aufgrund von möglichen Speicherengpässen die Zeile 6 nicht effizient umgesetzt werden. Die Verfahren bieten folglich einen theoretischen Ansatz, können aber nicht umgesetzt werden.

Es soll weiterhin eine — zur Zeit sehr aktuelle — Zerlegung vorgestellt werden. Dabei stützt sich diese Arbeit im Wesentlichen auf [16–18] und die folgenden Notationen, Definitionen und Algorithmen sind daraus entnommen. Bei der Zerlegung handelt es sich um die „Hierarchical HOSVD“ für Tensoren. Diese ist eine Weiterentwicklung der HOSVD und deshalb müssen manche Begriffe umdefiniert werden. Die Zerlegung kann im Grunde als eine mehrstufige Variante der HOSVD aufgefasst werden. Für den mehrstufigen Aufbau wird eine Hierarchie der einzelnen Richtungen bzw. Dimensionen $1, \ldots, p$ aufgestellt.

Definition 3.21 (Dimensionsbaum) *Ein* Dimensionsbaum T_p *für die Anzahl der Dimension* $p \in \mathbb{N}$ *entspricht einem Baum mit der Wurzel* $W(T_p) = \{1, \ldots, p\}$ *und einer Tiefe*

$$d = \lceil \log_2(p) \rceil := \min\{i \in \mathbb{N}_0 \mid i \geq \log_2(p)\},$$

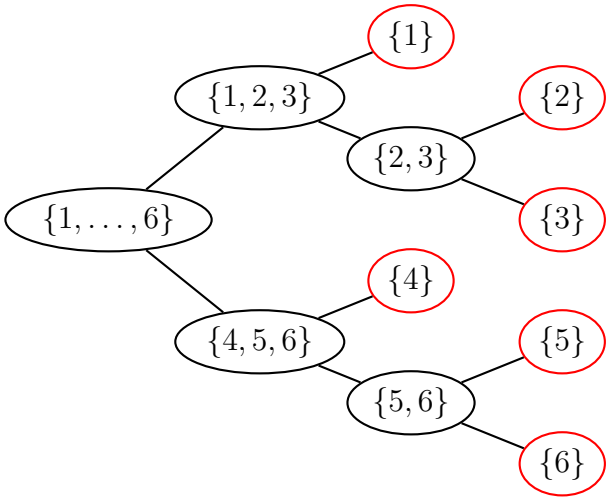

Abbildung 3.3.: Abbildung eines Dimensionsbaums

Die Grafik zeigt ein Beispiel eines Dimensionsbaums für $p = 6$, welches aus [16] stammt.

so dass jeder Knoten $t \in T_p$ entweder

1. *ein Blatt und eine einelementige Menge $t = \{i\},\ i = 1, \ldots, p$, mit dem Abstand $\ell \in \{d-1, d\}$ zur Wurzel oder*
2. *die disjunkte Vereinigung ($\dot\cup$) zweier Nachfolger $s(t) := \{t_1, t_2\} : t = t_1 \dot\cup t_2$ ist.*

Die Menge der Blätter des Baumes wird mit $\mathcal{L}(T_p)$ bezeichnet und die Menge der inneren Knoten mit $\mathcal{I}(T_p)$. Ein Knoten des Baumes wird auch als Dimensionsauswahl bezeichnet, da er verschiedene Dimensionen beinhaltet.

Zur Veranschaulichung ist ein Beispiel eines Definitionsbaums in Abbildung 3.3 zu finden. Im Folgenden wird die Definition der Matricization verallgemeinert, so dass mehrere Dimensionen bei der Matricization zusammengefasst werden können. Zur besseren Unterscheidung wird diese Form der Matricization H-Matricization genannt im Bezug auf die „Hierarchical HOSVD“.

Definition 3.22 (H-Matricization) *Sei t ein Knoten aus einem Dimensionsbaum T_p. Das Komplement t' ist definiert durch*

$$t' := \{1, \ldots, p\} \setminus t.$$

Die zugehörige H-Matricization *ist weiterhin definiert durch*

$$\mathcal{M}_t^H \ : \ \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p} \to \mathbb{R}^{\tilde{m} \times \tilde{n}}, \quad \tilde{m} = \prod_{i \in t} m_i n_i, \ \tilde{n} = \prod_{i \in t'} m_i n_i,$$

$$\text{mit } (\mathcal{M}_t^H(\mathcal{X}))_{(\alpha_i)_{i \in t}, (\alpha_i)_{i \in t'}} := x_{\alpha_1, \ldots, \alpha_p},$$

und dem Spezialfall $\mathcal{M}_\emptyset^H(\mathcal{X}) := \mathcal{M}_{\{1,\ldots,p\}}^H(\mathcal{X}) = \mathcal{X}$.

Des Weiteren soll der hierarchische Rang eingeführt werden. Dieser ähnelt im Wesentlichen dem zu Beginn des Unterkapitels definierten j-Rang. Der hierarchische Rang erweitert dieses Konzept durch die Verwendung der H-Matricization.

Definition 3.23 (Hierarchischer Rang) *Der* hierarchische Rang k_t *für ein* $t \in T_p$ *eines Tensors* $\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p}$ *ist definiert durch*

$$k_t := \operatorname{rang}(\mathcal{M}_t^H(\mathcal{X})) \in \mathbb{N}, \quad \forall t \in T_p.$$

Die Menge $(k_t)_{t \in T_p}$ *wird* Rang-Verteilung *bzgl. des Dimensionsbaums* T_p *genannt. Die Menge aller Tensoren mit höchstens hierarchischem Rang* k_t *für* $t \in T_p$ *wird mit*

$$\operatorname{rang}_H((k_t)_{t \in T_p}) := \{\mathcal{X} \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_p n_p} \mid \forall t \in T_p \ : \ \operatorname{rang}(\mathcal{M}_t^H(\mathcal{X})) \leq k_t\}$$

bezeichnet.

Eine hierarchische HOSVD lässt sich mit Hilfe des hierarchischen Rangs als SVD der Matrizen $\mathcal{M}_t^H(\mathcal{X})$ bestimmen. Im Grunde ist dies sehr ähnlich zu der HOSVD und daher treten im Fall der Tensorprodukt Smoothing Splines ähnliche Komplexitätsprobleme auf, da ein Zugriff auf die gesamte Matrix zur Speicherauslastung führen kann. Es wird sich jedoch später zeigen, dass dies bei Tensorprodukt Smoothing Splines zum Teil vermeidbar ist. Zuerst soll die eigentliche Zerlegung eingeführt werden, wofür eine letzte Definition nötig ist.

Definition 3.24 (Basisbaum, Basismatrix, Transfertensor) *Sei* $t \in T_p$ *eine Dimensionsauswahl und* $(k_t)_{t \in T_p}$ *eine Rang-Verteilung. Eine Matrix* $U_t \in \mathbb{R}^{\tilde{m} \times k_t}$, *mit* $\tilde{m} = \prod_{i \in t} m_i n_i$, *wird* Basismatrix *genannt und ein Tupel* $(U_t)_{t \in T_p}$ *solcher Matrizen wird als* Basisbaum *bezeichnet. Eine Basismatrix ist orthogonal, wenn alle Spalten orthonormal sind. Ein Basisbaum ist orthognal, wenn alle enthaltenen Basismatrizen orthogonal sind. Weiterhin ist ein Basisbaum*

verschachtelt, wenn für jede innere Dimensionsauswahl des Dimensionsbaums mit den Nachfolgern $s(t) = \{t_1, t_2\}$ gilt

$$Bild(U_t) \subset Bild(U_{t_1}) \otimes Bild(U_{t_2}).$$

Die Einträge des Tensors $B_t \in \mathbb{R}^{k_t \times k_{t_1} \times k_{t_2}}$ sind die Koeffizienten für die Darstellung der i-ten Spalte $(U_t)_i$ von U_t durch die Spalten von U_{t_1} und U_{t_2},

$$(U_t)_i := \sum_{j=1}^{k_{t_1}} \sum_{\ell=1}^{k_{t_2}} (B_t)_{i,j,\ell} (U_{t_1})_j \otimes (U_{t_2})_\ell.$$

Dieser Tensor B_t wird Transfertensor *genannt.*

Die Transfertensoren fungieren als Bindeglieder für die einzelnen Basismatrizen und stehen in den inneren Knoten der Zerlegung. Mit Hilfe der angegebenen Strukturen lässt sich eine hierarchische, geschachtelte Zerlegung für Tensoren definieren.

Definition 3.25 (Hierarchische Tucker-Zerlegung) *Sei $(k_t)_{t \in T_p}$ eine Rang-Verteilung für den Dimensionsbaum T_p und $\mathcal{X} \in \text{rang}_H((k_t)_{t \in T_p})$. Sei $(U_t)_{t \in \mathcal{L}(T_p)}$ ein verschachtelter Basisbaum mit dem Transfertensor $(B_t)_{t \in \mathcal{I}(T_p)}$ und*

$$\forall t \in T_p : \; Bild(\mathcal{M}_t^H(\mathcal{X})) = Bild(U_t), \quad \mathcal{X} = U_{\{1,..,p\}}.$$

Die Darstellung $\mathcal{X}_H := ((B_t)_{t \in \mathcal{I}(T_p)}, \; (U_t)_{t \in \mathcal{L}(T_p)})$ ist die hierarchische Tucker-Zerlegung *von $\mathcal{X}$.*

Der Aufbau einer solchen Zerlegung wird in Abbildung 3.4 beispielhaft dargestellt. Die in [17] angegebenen Algorithmen zum Berechnen der hierarchischen Tucker-Zerlegung benötigen die SVD der verschiedenen Matricizations. Dies ist aufgrund der Größenordnung der Daten praktisch bei dem betrachteten Fall nicht umsetzbar. Der Vollständigkeit halber soll das Tensor-Train Format aus [38] erwähnt werden. Diese Zerlegung ist ein Sonderfall der hierarchischen HOSVD und entspricht einer hierarchischen Tucker-Zerlegung mit einem degenerierten Dimensionsbaum, d.h. einer linearen Liste, deren Höhe proportional zu p ist. Da bei dieser vergleichbare Komplexitätsprobleme auftauchen, wird die Zerlegung nicht weiter betrachtet.

Bei Tensorprodukt Smoothing Splines handelt es sich, wie bereits erwähnt, um einen Spezialfall, da diese als Summe von Kronecker-Produkten darstellbar sind

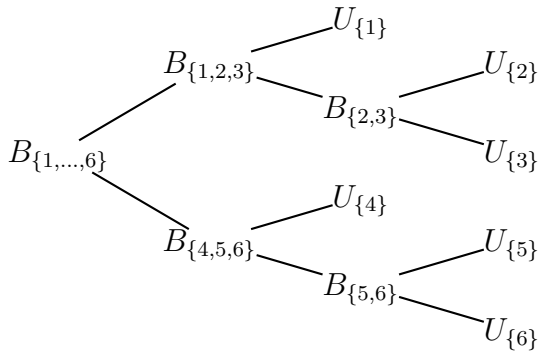

Abbildung 3.4.: Hierarchische Tucker-Zerlegung

Die Abbildung skizziert den verschachtelten Aufbau einer Hierarchischen Tucker-Zerlegung. Es lässt sich erkennen, dass die Transfertensoren als Bindeglieder für die Basismatrizen fungieren. Aufgrund dessen müssen alle Transfertensoren berechnet werden.

(siehe 2.2). Diese Summe kann zu einer Summe von 1-Formen bzw. Rang-1 Tensoren umstrukturiert werden. In [17] wird mit der Bemerkung 37 eine Möglichkeit vorgestellt, wie diese Summe direkt in der hierarchische Tucker-Zerlegung dargestellt werden kann. Diese Möglichkeit soll kurz vorgestellt werden. Eine Summe von Rang-1 Tensoren

$$\mathcal{X} = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} v_{ij}, \quad v_{ij} \in \mathbb{R}^{m_j n_j}$$

kann für alle Blätter $t = \{j\} \in \mathcal{L}(T_p)$ durch die Basismatrizen

$$(U_t)_i := v_{ij},\ i = 1, \dots, k,\ k_j := k$$

und für alle inneren Knoten $t \in \mathcal{I}(T_p) \setminus W(T_p)$ die Transfertensoren

$$(B_t)_{i,j,\ell} := \begin{cases} 1 & , \text{ wenn } i = j = \ell \\ 0 & , \text{ sonst} \end{cases},$$

$B_t \in \mathbb{R}^{k \times k \times k},\ k_t := k$, direkt in die hierarchische Tucker-Zerlegung überführt

Algorithmus 3.4 : Orthogonalisierung der hierarchischen Tucker Darstellung

Input : $\mathcal{X}_H \in \mathrm{rang}_H((k_t)_{t\in T_p})$ dargestellt durch $((B_t)_{t\in\mathcal{I}(T_p)},\ (U_t)_{t\in\mathcal{L}(T_p)})$.
Output : verschachtelte orthogonale Basismatrizen $(U_t)_{t\in\mathcal{L}(T_p)}$ und Transfertensoren $(B_t)_{t\in\mathcal{I}(T_p)}$

begin
  forall the $t \in \mathcal{L}(T_p)$ **do**
    Berechne die QR-Zerlegung von U_t und setze $U_t := Q$ sowie
    $$B_f := \begin{cases} (I, I, R) \circ B_f & , \text{ wenn } t \text{ der zweite Nachfolger ist,} \\ (I, R, I) \circ B_f & , \text{ wenn } t \text{ der erste Nachfolger ist,} \end{cases},$$
    für den Vaterknoten f von t.
  end
  forall the $t \in \mathcal{I}(T_p)$ **do**
    Berechne die QR-Zerlegung von $\mathcal{M}^H_{(\{1,2\})}(B_t)$ mit $\mathcal{M}^H_{(\{1,2\})}(B_t) := \mathcal{M}^H_{(\{1,2\})}(Q_t)R$ und setze $B_t := Q_t$ sowie
    $$B_f := \begin{cases} (I, I, R) \circ B_f & , \text{ wenn } t \text{ der zweite Nachfolger ist,} \\ (I, R, I) \circ B_f & , \text{ wenn } t \text{ der erste Nachfolger ist,} \end{cases},$$
    für den Vaterknoten f von t.
  end
end

werden. Der Wurzel-Transfertensor entspricht

$$(B_{\{1,\dots,p\}})_{1,j,\ell} := \begin{cases} 1 & , \text{ wenn } j = \ell \\ 0 & , \text{ sonst} \end{cases},$$

$B_{\{1,\dots,p\}} \in \mathbb{R}^{1\times k\times k}$, $k_{\{1,\dots,d\}} = 1$. Diese Darstellung benötigt zwar mehr Speicher, aber dieser fällt laut [17] nicht ins Gewicht, da das Sparse-Format verwendet werden kann. Das Sparse-Format speichert alle von Null verschiedenen Einträge und ihre zugehörigen Indizes. Dies ist nur effizient, wenn weniger als $\frac{1}{p}\prod_{j=1}^{p} m_j n_j$ Einträge verschieden von Null sind. Bei Tensorprodukt Smoothing Splines ist diese Speicherung nur bedingt effizient, wie später erläutert wird. Es wird weiterhin ein Algorithmus angegeben, um diese Darstellung in eine effiziente hierarchische Tucker-Zerlegung zu überführen. Die obige Umstrukturierung einer Summe von Kronecker-Produkten kann mit dem Algorithmus 3.4 aus [17] orthogonalisiert und komprimiert werden, was einer Optimierung bzgl. der hierarchischen Tucker-Zerlegung entspricht.

Da für den Algorithmus 3.4 in jedem Schritt nur eine QR-Zerlegung von den univariaten Daten berechnet wird, sollte dieser Algorithmus praktisch umsetzbar sein. Es ist aber von Problemen mit der Speicherung der Transfertensoren $B \in \mathbb{R}^{k \times k \times k}$, $k = N$, bei der Umwandlung der Summe von Kronecker-Produkten in die hierarchische Tucker-Zerlegung auszugehen, da bei Tensorprodukt Smoothing Splines die Dimensionen des Transfertensors von der Anzahl der Stützstellen abhängt und diese in der betrachteten Problemstellung aus [44] durchaus mehr als 10.000 betragen kann. Selbst wenn der Transfertensor im Sparse-Format gespeichert wird, reicht ab einem gewissen Punkt der Speicher nicht aus, da durch die Optimierung mehr Elemente besetzt werden. Dies scheint der Grund zu sein, warum in [17] in der Bemerkung 37 nur von einer kurzen Summe bzgl. der Umwandlung die Rede ist.

Bei den betrachteten Splines entspricht die Anzahl der Summanden aber der Anzahl der Stützstellen. Selbst wenn die Speicherproblematik der Transfertensoren wegfallen würde, hat die Zerlegung durch die verschachtelte Darstellung gewisse Nachteile gegenüber der Darstellung mit Multilinearformen, zumindest im Zusammenhang mit der Verwendung bei Tensorprodukt Smoothing Splines. Dies liegt im Wesentlichen daran, dass die Darstellung nach dem Dimensionsbaum geordnet ist und bedingt dadurch nachträglich eine Ordnung, welche der Darstellung aus Algorithmus 3.1 entspricht, gefunden werden muss. Die Zerlegung ist zwar orthogonal, was als Vorteil zu werten ist, aber dieser fällt bei der benötigten Verwendung nicht weiter ins Gewicht.

Abschließend ist zu sagen, dass bei Tensorprodukt Smoothing Splines nur eine grobe Approximation durch Kronecker-Produkte gefunden werden soll, um diese beispielsweise zum Vorkonditionieren oder zur Approximation der Inversen zu verwenden. Die Variante mittels Multilinearformen in Unterkapitel 3.2 ist durch die Ordnung bzgl. der Größe der σs und ihre schnelle Auswertung im Vorteil und geeigneter für das Problem, da die Bestimmung wesentlich schneller und übersichtlicher ist. Weiterhin müssen nur die univariaten Komponenten — explizit handelt es sich dabei um Vektoren — betrachtet werden, da die bereits vorhandene Kronecker-Struktur ausgenutzt werden kann. Bei den gegen Ende dieses Kapitels vorgestellten Verfahren kann diese Struktur nicht erhalten werden, sodass der Algorithmus 3.1 als einzige Möglichkeit übrig bleibt. In den folgenden Kapiteln werden mögliche Verwendungen für die Komprimierung vorgestellt.

4

Approximation einer Kronecker-Inversen

Die Motivation für die Approximation einer Kronecker-Inversen liegt bei Tensorprodukt Smoothing Splines darin begründet, dass sie zum Beispiel für die Optimierung des Glättungsparameters λ bei der „Generalized Cross Validation“ oder als Vorkonditionierer für das GMRES-Verfahren verwendet werden kann. Aufgrund der Komplexität und der Größenordnung des betrachteten Problems ist es in vielen Fällen weder sinnvoll noch möglich, die Inverse der Summe von Kronecker-Produkten explizit aufzustellen. Falls die Matrix

$$\mathbb{R}^{n\times n} \ni A = \bigotimes_{j=1}^{p} A_j, \text{ mit } A_j \in \mathbb{R}^{n_j\times n_j} \text{ und } n = \prod_{j=1}^{p} n_j,$$

nur aus einem Kronecker-Produkt besteht, ist die Inverse von A sehr effizient durch die Inverse der einzelnen Kronecker-Faktoren A_j gegeben

$$A^{-1} = \left(\bigotimes_{j=1}^{p} A_j\right)^{-1} = \bigotimes_{j=1}^{p} A_j^{-1}.$$

Dies folgt durch mehrfache Anwendung des Satzes 2.7. Bei einer Summe von Kronecker-Produkten ist dies nicht möglich, da die Inverse sich nicht durch die Invertierung der einzelnen Summanden berechnen lässt. Durch die Summation geht weiterhin die Kronecker-Struktur verloren. Da die Matrix in den meisten Fällen nicht explizit angegeben werden sollte, müssen viele alternative Verfah-

ren zur Annäherung der Inversen von der Untersuchung ausgeschlossen werden. Daher erscheint eine Approximation an die Inverse, welche die Summen- und Kronecker-Struktur mit einbezieht, als einzige sinnvolle Möglichkeit für den Umgang mit der Inversen.

Im Folgenden sei entsprechend der betrachteten Problemstellung bei Tensorprodukt Smoothing Splines

$$\mathbb{R}^{n\times n} \ni A = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij}, \text{ wobei } A_j \in \mathbb{R}^{n_j \times n_j} \text{ und } n = \prod_{j=1}^{p} n_j.$$

Im Zusammenhang mit „Stochastic Automatic Networks“ (SANS) wurde in [26] bereits eine Methode vorgestellt, um die Inverse einer Summe von Kronecker-Produkten zu bestimmen. Dabei wird die Summe von Kronecker-Produkten durch ein einzelnes Kronecker-Produkt approximiert

$$\min_{\substack{X_j,\\ j=1,\dots,p}} \left\| A - \bigotimes_{j=1}^{p} X_j \right\|_F, \text{ mit } X_j \in \mathbb{R}^{n_j \times n_j}, \tag{4.1}$$

und dessen Inverse als Approximation für die Inverse der gesamten Summe verwendet. Da bei SANS wesentlich weniger Summanden auftreten als bei der Aufgabenstellung dieser Arbeit, funktioniert die Vorgehensweise in diesem besonderen Fall verhältnismäßig gut. In [27] werden allerdings Fälle aufgezeigt, in welchen diese Vorgehensweise nicht funktioniert, da sie zu wenig Informationen mit einbezieht. Dies entspricht im Grunde wieder der Problematik der Rang-1 Approximation, welche im Abschnitt 2.1 aufgezeigt wurde. Bei Tensorprodukt Smoothing Splines ist diese Approximation ebenfalls nicht zufriedenstellend, da aufgrund der sehr großen Anzahl an Summanden die Information in der Kronecker-Produkt-Approximation nicht ausreichend ist.

In dieser Arbeit soll ein alternatives Verfahren zur Approximation der Inversen vorgestellt werden. Dieses basiert im Wesentlichen auf dem Prokrustes-Problem aus [4, 42] und [15]

$$\min_{X} \|AX - I\|_F, \text{ wobei } A, X \text{ und } I \in \mathbb{R}^{n\times n}$$

und einer Verwendung der „Alternating Least-squares“-Idee (ALS). Es handelt sich dabei um eine Vorgehensweise zur Minimierung eines multivariaten

Problems, indem abwechselnd auf die univariaten Komponenten das „Least-squares“-Verfahren angewendet wird. Im nächsten Abschnitt wird erklärt, wie die Idee in dieser Arbeit Verwendung findet. Eine ausführliche Untersuchung der ALS-Idee ist in [3] zu finden. Die vorgestellte Anwendung in [3] bezieht sich allerdings auf ein anderes Problem. Die approximierte Inverse kann, wie bereits erwähnt, als Vorkonditionierer für das GMRES-Verfahren oder für die Bestimmung des Glättungsparameters λ verwendet werden. Weiterhin ist es möglich die Komprimierung aus Kapitel 3 in bestimmten Fällen als Grundlage zur Bestimmung der Annäherung zu verwenden.

4.1. Das Prokrustes-Problem und Kronecker-Produkte

Das Prokrustes-Problem kann allgemein beschrieben werden durch die Idee der Annäherung einer gegebenen Struktur an eine weitere Struktur. Es ist in der linearen Algebra als Problem einer Matrix-Approximation bekannt und in vielen Fällen wird zusätzlich gefordert, dass die Annäherung orthogonal sein soll. Dadurch wird diese Approximation häufig als „orthogonales Prokrustes-Problem“ bezeichnet. Zuerst wurde der Begriff in [42] eingeführt. Weitere allgemeine Informationen sind in [20] und [15] zu finden.

Konkret lässt sich das Prokrustes-Problem wie folgt formulieren: Zu zwei gegebenen Matrizen $A \in \mathbb{R}^{n\times n}$ und $B \in \mathbb{R}^{n\times n}$ soll eine Matrix $X \in \mathbb{R}^{n\times n}$ approximiert werden, so dass gilt

$$\min_X \|AX - B\|_F.$$

In dieser Arbeit wird für die Matrix X außerdem eine Kronecker-Struktur gefordert. Dies wurde zum Beispiel bereits in [4] betrachtet. Dabei wurde allerdings nur der Fall mit einem zweidimensionalen Kronecker-Produkt und orthogonalen Matrizen untersucht. Solche Einschränkungen sind an dieser Stelle nicht relevant und es wird lediglich die Kronecker-Struktur mit einer beliebigen Anzahl an Faktoren verlangt, damit X im Rahmen dieser Arbeit überhaupt Verwendung finden kann. Es wird im Grunde eine Rechtsinverse der Matrix A approximiert. Wenn eine Linksinverse bevorzugt wird, muss das Minimierungsproblem zu $\|XA - I\|_F$ umgestellt werden. Die folgende Vorgehensweise lässt sich darauf analog umsetzen. Die Kronecker-Struktur fließt folgendermaßen in die Formu-

lierung des Prokrustes-Problems mit ein:

$$\min_{\substack{X_j,\\ j=1,\dots,p}} \left\| A \bigotimes_{j=1}^{p} X_j - B \right\|_F, \text{ wobei } X_j \in \mathbb{R}^{n_j \times n_j},\ n = \prod_{j=1}^{p} n_j.$$

Für die Approximation der Inversen einer Matrix, welche als Summe von Kronecker-Produkten darstellbar ist, ergibt sich somit

$$\min_{\substack{X_j,\\ j=1,\dots,p}} \left\| \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} \bigotimes_{j=1}^{p} X_j - I \right\|_F, \text{ mit } A_{ij}, X_j \in \mathbb{R}^{n_j \times n_j},\ n = \prod_{j=1}^{p} n_j. \tag{4.2}$$

Die Matrix I steht für die $n \times n$ Einheitsmatrix. Die erste Idee für diesen Ansatz stammt aus [40]. Die Kronecker-Faktoren X_j lassen sich komponentenweise durch ein ALS-Verfahren bestimmen. Eine ähnliche Anwendung des ALS-Verfahrens kann in [3] gefunden werden. Der dort betrachtete Fall entspricht der Verwendung des ALS-Verfahrens, wie es in Kapitel 3 vorgestellt wird. Es ist mit der Anwendung der ALS-Idee auf die Formel (4.1) gleichzusetzen. In diesem Kapitel wird das ALS-Verfahren wie folgt verwendet: Die Komponenten X_j von X werden zyklisch durch das folgende Minimierungsproblem

$$\min_{X_j} \|\tilde{B} X_j - Y\|_F, \quad \tilde{B}, Y \in \mathbb{R}^{n_j \times n_j}$$

bestimmt. Dabei wird immer eine Komponente X_j als Variable betrachtet und die anderen Komponenten als Parameter. Die Matrizen $\tilde{B}$ und Y setzen sich aus den übrig gebliebenen Komponenten von X und den jeweiligen Komponenten von A zusammen. Der genaue Aufbau der beiden Matrizen wird später ausführlicher erläutert. Es handelt es sich um ein spaltenweises „Least-squares“-Problem

$$\min_{X_j} \|\tilde{B} X_j(:, i) - Y(:, i)\|_2,$$

für $i = 1, \dots, n_j$, wobei dieses aber als matrixwertiges LGS gelöst werden kann. Die Komponenten X_j des Kronecker-Produkts werden solange durchgewechselt, bis das Ergebnis zufriedenstellend ist oder sich nicht mehr ändert. Das „Alternating“ in diesem Verfahren bezieht sich auf das Wechseln der Variablen.

Ein Nachteil des ALS-Konzepts ist die nicht vorhandene Garantie, dass durch die Bestimmung der einzelnen Komponenten zwingend das globale Minimum gefun-

den wird. Dies kann mit Gegenbeispielen belegt werden. Diese sind aber für die Darstellung an dieser Stelle nicht geeignet. Sie können durch ein nichtlineare Optimierungsproblem erzeugt werden. Dafür werden die einzelnen Matrixeinträge als Variablen betrachtet und nicht nur die Faktoren der Kronecker-Produkte. Es wird bei der Anwendung in diesem Kapitel in jedem Fall ein kritischer Punkt angenommen. Dieser entspricht einem Sattelpunkt oder einem lokalen Minimum bezüglich der partiellen Ableitungen entlang der Koordinatenachsen. Dies wird später zu sehen sein. In der Praxis wird allerdings in den meisten Fällen bei der Verwendung bzgl. der gegebenen Problemstellung ein globales Minimum gefunden. Das „Least-squares"-Problem für die einzelnen Komponenten lässt sich durch die partielle Ableitung der Norm aufstellen. Dafür sollen zuerst einzelne Teile des Ausdrucks in (4.2) betrachtet werden. Die Differenz $AX - I$ erlaubt mit Satz 2.7 die Umformung

$$AX - I = \left(\sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij}\right) \bigotimes_{j=1}^{p} X_j - I = \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} X_j - I. \tag{4.3}$$

Zur Erinnnerung: Für die Frobenius-Norm gilt, dass

$$\|A\|_F = \sqrt{\operatorname{tr}(A^t A)}.$$

Um die Frobenius-Norm partiell abzuleiten, wird eine weitere Umformung benötigt. Es gilt

$$\begin{aligned}
(AX - I)^t (AX - I) = &\sum_{i,k=1}^{N} \left(\bigotimes_{j=1}^{p} A_{ij} X_j\right)^t \left(\bigotimes_{j=1}^{p} A_{kj} X_j\right) \\
&- \sum_{i=1}^{N} \bigotimes_{j=1}^{p} X_j^t A_{ij}^t - \sum_{j=1}^{N} \bigotimes_{j=1}^{p} A_{ij} X_j + I \\
= &\sum_{i,k=1}^{N} \bigotimes_{j=1}^{p} \left(X_j^t A_{ij}^t A_{kj} X_j\right) - \sum_{j=1}^{N} \bigotimes_{j=1}^{p} X_j^t A_{ij}^t \qquad (4.4a) \\
&- \sum_{i=1}^{N} \bigotimes_{j=1}^{p} A_{ij} X_j + I. \qquad (4.4b)
\end{aligned}$$

Daraus lässt sich für die Norm folgern, dass

$$\begin{aligned}\|AX - I\|_F^2 &= \operatorname{tr}\left((AX-I)^t(AX-I)\right) \\ &= \sum_{i,k=1}^{N}\prod_{j=1}^{p}\operatorname{tr}\left(X_j A_{ij} A_{kj} X_j\right) - 2\sum_{i=1}^{N}\prod_{j=1}^{p}\operatorname{tr}\left(A_{ik}X_j\right) + \underbrace{\operatorname{tr}(I)}_{=n}, \qquad (4.5)\end{aligned}$$

wobei die Eigenschaft einer Spur von Kronecker-Produkten aus Satz 2.7 verwendet wird. Die Berechnung der Spur eines Matrix-Produkts lässt sich mit Hilfe des *vec*-Operators effizient als Matrix-Vektor-Multiplikation umsetzen.

Lemma 4.1 *Für zwei Matrizen $X, A \in \mathbb{R}^{n\times n}$ gilt*

$$\operatorname{tr}(X^tAX) = vec(X)^t(I\otimes A)vec(X),$$

wobei I die $n \times n$ Einheitsmatrix ist. Weiterhin gilt ebenfalls

$$\operatorname{tr}(AX) = vec(X)^t vec(A^t).$$

Beweis: Es folgt für den ersten Fall, dass

$$\operatorname{tr}(X^tAX) = \sum_{i=1}^{n} X^t(i,:)AX(:,i) = vec(X)^t(I\otimes A)vec(X),$$

wobei mit der Bezeichnung $X(i,:)$ die i-te Zeile bzw. für $X(:,i)$ die i-te Spalte von X gemeint ist. Analog folgt für den zweiten Fall, dass

$$\operatorname{tr}(AX) = \sum_{i=1}^{n} A(i,:)X(:,i) = vec(X)^t vec(A^t).$$

□

Die Berechnung der Spur eines Matrixprodukts ist auf diese Weise wesentlich effizienter als die direkte Berechnung. Mit Lemma 4.1 ergibt sich für (4.5)

$$\begin{aligned}\left\| A\bigotimes_{j=1}^{p} X_j - I\right\|_F &= \sum_{i,k=1}^{N}\prod_{j=1}^{p} vec(X_j)^t(I\otimes A_{ij}^tA_{kj})vec(X_j) \qquad (4.6a)\\ &\quad - 2\sum_{i=1}^{N}\prod_{j=1}^{p} vec(X_j)^t vec(A_{ij}^t) + n \qquad (4.6b)\end{aligned}$$

Das eigentliche Ziel ist immer noch den Ausdruck in (4.6) komponentenweise

bzgl. der Matrizen $X_r,\ r = 1, ..., p$, zu minimieren. Es wird dafür X_r als Variable betrachtet und die restlichen Matrizen $X_j,\ j \neq r$, werden als Konstanten festgehalten. Damit lässt sich der Ausdruck in (4.6) wie folgt formulieren:

$$vec(X_r)^t(I \otimes B_r)vec(X_r) - 2vec(X_r)^t y_r + n, \tag{4.7}$$

wobei

$$B_r := \sum_{i,k=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_j^t A_{ij}^t A_{kj} X_j\right) A_{ir}^t A_{kr}$$

sowie

$$y_r := \sum_{i=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(A_{ij} X_j\right) v(A_{ir}^t).$$

Bei dem Ausdruck in (4.7) handelt es sich um ein quadratisches Funktional. Weiterhin ist die Matrix B_r im Fall von Tensorprodukt Smoothing Splines symmetrisch und positiv semidefinit, da sie aus einer Summe von symmetrischen und positiv semidefiniten Matrizen besteht. Das Kronecker-Produkt $I \otimes B_r$ ist laut [43] ebenfalls symmetrisch und positiv semidefinit. Damit folgt nach einem Resultat aus [7], dass das quadratische Funktional konvex ist, da die zugehörige Hesse-Matrix $I \otimes B_r$ positiv semidefinit ist. Mit einem weiteren Satz aus [7] folgt, dass für das quadratische Funktional (4.7) bzgl. X_r nur ein globales Minimum existieren kann. Dieses wird bei

$$\frac{\partial}{\partial X_r}\left(vec(X_r)^t(I \otimes B_r)vec(X_r) - 2vec(X_r)^t y_r + n\right) = 0$$

angenommen. Um die Norm in (4.2) bezüglich X_r zu minimieren, wird der Term in (4.7) nach X_r abgeleitet

$$\begin{aligned} &\frac{\partial}{\partial X_r}\left(vec(X_r)^t(I \otimes B_r)vec(X_r) - 2vec(X_r)^t y_r + n\right) \\ &\qquad = 2(I \otimes B_r)v(X_r) - 2y_r. \end{aligned} \tag{4.8}$$

Die Formel (4.8) lässt sich durch Null setzen und mit Hilfe des inversen *vec*-Operators in eine kompaktere Form umformen zu

$$B_r X_r = Y_r \text{ mit } Y_r = \sum_{i=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}(A_{ij} X_j) A_{ir}^t.$$

Algorithmus 4.1 : Algorithmus zur Approximation der Inversen durch ein Kronecker-Produkt

Input : $\mathbb{R}^{n\times n} \ni A = \sum_i^N \bigotimes_j^p A_{ij}$, mit $A_{ij} \in \mathbb{R}^{n_j \times n_j}$ und A_{ij} symmetrisch positiv semidefinit, $\varepsilon > 0$

Output : Kronecker-Produkt $\bigotimes_j^p X_j \approx A^{-1}$

begin
  Initialisiere $X_j := \left(\sum_k^N A_{kj}\right)^{-1}$ mit $j = 1, \ldots, p$
  while $\|AX - I\|_F > \varepsilon$ **do**
    for $r = 1, \ldots, p$ **do**
      $B_r := \sum_{i,k=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_j^t A_{ij}^t A_{kj} X_j\right) A_{ir}^t A_{kr}$
      $Y_r := \sum_{i=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(A_{ij} X_j\right) A_{ir}^t$
      Löse $B_r X_r = Y_r$ mittels „Least-squares“.
    end
  end
end

Um das Minimum bzgl. X_r zu bestimmen, muss das obige LGS gelöst werden. Es entspricht dem zu Beginn vorgestellten komponentenweisen „Least-squares“-Problem

$$\min_{X_r} \|B_r X_r - Y_r\|_F \text{ bzw. } \min_{X_r(i,:)} \|B_r X_r(i,:) - Y_r(i,:)\|_2$$

für $i = 1, \ldots, n_r$. Mit Hilfe dieser Idee lässt sich für die komponentenweise Minimierung der Norm der zyklische Algorithmus 4.1 aufstellen. Mit diesem Algorithmus lässt sich ein kritischer Punkt finden, welcher aber, wie zuvor erwähnt, nicht zwingend ein globales oder lokales Minimum sein muss. Dies lässt sich durch Gegenbeispiele belegen und wurde bereits in [36] ausführlicher untersucht.

Weiterhin ist diese Approximation der Inversen lediglich bei Summen von Kronecker-Produkten mit wenig Summanden effektiv anwendbar. Dies lässt sich in [27] und an numerischen Beispielen gut beobachten. Es ist vergleichbar mit der Problematik bei SANS, da zu wenige Informationen verwendet werden. Um die Idee des Algorithmus 4.1 auf das vorgestellte Problem anwenden zu können, muss dieser daher erweitert werden, so dass X iterativ verbessert werden kann. Die erste Idee war die Matrix X als Produkt aufzufassen $X = \prod_\ell X_\ell$. Dies funktioniert, wenn X nicht als Kronecker-Produkt darstellbar sein muss. Sobald gefordert wird, dass $X_\ell = \prod_\ell \bigotimes_j^p X_{\ell j}$, ist dieser Ansatz nicht zielführend.

Aufgrund der Eigenschaft des Kronecker-Produkts aus Satz 2.7

$$(C \otimes D)(E \otimes F) = (CE) \otimes (DF) \text{ mit } C, D, E, F \in \mathbb{R}^{n \times n}$$

stagniert der Algorithmus, da das komponentenweise Minimum nicht verbessert werden kann. Dies liegt daran, dass sich X zu $X = \bigotimes_{j=1}^{p} X_{1j} X_{2j}$ umschreiben lässt und eine Minimierung nach X_{2j} keine Verbesserung bringen kann, da X_{1j} bereits ein globales Minimum bzgl. des komponentenweisen Minimums ist

$$\min_{X_{2r}} \| \underbrace{B_r X_{1r}}_{\approx Y_r} X_{2r} - Y_r \|_F.$$

In anderen Worten: Die Variable X_{2j} approximiert im Grunde genommen die Einheitsmatrix. Dies bringt offensichtlich keinen Vorteil. Ein anderer Ansatz besteht darin, die Matrix X ebenfalls als Summe von Kronecker-Produkten zu betrachten

$$A^{-1} \approx X = \sum_{\ell=1}^{g} \bigotimes_{j=1}^{p} X_{\ell j}.$$

Die Matrix X wird schrittweise um einen Summanden mit Kronecker-Struktur erweitert. Dieser zusätzliche Summand wird im Wesentlichen mit dem obigen Algorithmus berechnet, wobei die vorher berechneten X_ℓ mit A multipliziert und von I abgezogen werden. Diese Multiplikation sowie die Subtraktion werden allerdings nicht ausgeführt, da die Kronecker-Struktur erhalten bleiben muss und eine direkte Multiplikation zu Speicherengpässen führen kann. Eine ähnliche Vorgehensweise ist in [3] zu finden. Das Minimierungsproblem aus (4.2) ändert sich ab dem zweiten Summanden X_ℓ, $\ell \geq 2$, zu

$$\min_{X_g} \left\| AX_g - \underbrace{\left(I - \left(A \sum_{\ell=1}^{g-1} X_\ell \right) \right)}_{\text{Konstante bzgl. } X_g} \right\|_F^2 \text{ mit } X_\ell = \bigotimes_{j=1}^{p} X_{\ell j},\ \ell = 1, \ldots, g.$$

Dadurch ändert sich das LGS, welches zur komponentenweisen Minimierung benötigt wird. Es lässt sich analog zum obigen Vorgehen durch die partielle Ableitung bzgl. der Komponenten von X_g aufstellen, wofür die Norm zuerst

wieder entsprechend umgeformt werden soll. Es gilt

$$\begin{aligned}
&\left\| A \bigotimes_{j=1}^{p} X_{gj} - \left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right\|_F^2 \\
&\qquad = \operatorname{tr}\left(\left(A \bigotimes_{j=1}^{p} X_{gj} - \left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right)^t \right. \\
&\qquad\quad \left. \cdot \left(A \bigotimes_{j=1}^{p} X_{gj} - \left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right) \right) \\
&\qquad \underset{\star}{=} \operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t A \bigotimes_{j=1}^{p} X_{gj} \right) && (4.9) \\
&\qquad\quad - 2 \operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t \left(I - \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right) && (4.10) \\
&\qquad\quad + \operatorname{tr}\left(\left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right)^t \left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right), && (4.11)
\end{aligned}$$

wobei $\star$ aufgrund von $\operatorname{tr}(C^t) = \operatorname{tr}(C)$ und $\operatorname{tr}(CD) = \operatorname{tr}(DC)$ für quadratische Matrizen $C, D \in \mathbb{R}^{n \times n}$ folgt. Zur besseren Übersicht werden im nächsten Schritt die einzelnen Terme getrennt voneinander betrachtet. Der erste Term (4.9) lässt sich analog zum Vorgehen am Anfang des Kapitels umformen

$$\operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t A \bigotimes_{j=1}^{p} X_{gj} \right) = \sum_{i,k=1}^{n} \prod_{j=1}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj} \right).$$

Für die Ableitung nach X_{gr} folgt

$$2 \sum_{i,k=1}^{N} \prod_{\substack{j=1 \\ j \neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj} \right) A_{ir}^t A_{kr} X_{gr}.$$

Der zweite Term (4.10) kann weiter vereinfacht werden zu

$$\begin{aligned}
&2 \operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t \left(I - A \left(\sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right) \right) \right) \\
&\qquad = 2 \operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t - \bigotimes_{j=1}^{p} X_{gj}^t A^t A \sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j} \right)
\end{aligned}$$

$$= 2\left(\operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t\right) - \operatorname{tr}\left(\bigotimes_{j=1}^{p} X_{gj}^t A^t \sum_{\ell=1}^{g-1} \bigotimes_{j=1}^{p} X_{\ell j}\right)\right)$$
$$= 2\left(\sum_{i=1}^{N}\prod_{j=1}^{p}\operatorname{tr}\left(A_{ij}X_{gj}\right) - \sum_{\ell=1}^{g-1}\sum_{i,k=1}^{N}\prod_{j=1}^{p}\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right)\right). \tag{4.12}$$

Für die Ableitung nach X_{gr} folgt

$$2\left(\sum_{i=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(A_{ij}X_{gj}\right)A_{ir}\right) - 2\left(\sum_{\ell=1}^{g-1}\sum_{i,k=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right)A_{ir}^t A_{kr} X_{gr}\right),$$

dies ergibt sich analog zum Vorgehen zu Beginn des Kapitels. Der dritte Term (4.11) der Gleichung muss nicht genauer betrachtet werden, da es sich bei diesem um eine Konstante bzgl. X_{gj} handelt. Er fällt bei der Ableitung weg und ist für das LGS folglich nicht relevant. Das LGS zur Bestimmung der X_ℓ, $\ell \geq 2$, ergibt sich durch Null setzen der Ableitung:

$$B_{gr}X_{gr} = Y_{gr},$$

wobei

$$B_{gr} := \sum_{i,k=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj}\right)A_{ij}^t A_{kj}$$

und

$$Y_{gr} := \sum_{i=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(A_{ij}X_{gj}\right)A_{ir} - \sum_{\ell=1}^{g-1}\sum_{i,k=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right)A_{ir}^t A_{kr} X_{\ell r}.$$

Die Existenz von nur einem Minimum bzgl. der komponentenweisen Minimierung nach X_{gr} lässt sich auf die gleiche Art und Weise begründen wie zuvor. Mittels der aus dem ersten Schritt bekannten, zyklischen Minimierung durch das ALS-Konzept lässt sich somit ein kritischer Punkt für

$$\min_{X_g}\left\|AX_g - \left(I - A\sum_{\ell=1}^{g-1}X_\ell\right)\right\|_F$$

berechnen. Dieser muss genau wie bei der Bestimmung des ersten Kronecker-Produkts nicht zwingend ein Minimum sein, ist es aber in vielen Fällen. Ins-

Algorithmus 4.2 : Algorithmus zur Approximation der Inversen durch eine Summe von Kronecker-Produkten

Input : $\mathbb{R}^{n\times n} \ni A = \sum_i^N \bigotimes_j^p A_{ij}$, mit $A_{ij} \in \mathbb{R}^{n_j \times n_j}$ und A_{ij} symmetrisch positiv semidefinit, $\varepsilon > 0$

Output : Kronecker-Produkt $\sum_{\ell=1}^{g} \bigotimes_{j=1}^{p} X_{\ell j} \approx A^{-1}$

begin
  Bestimme X_1 mittels Algorithmus 4.1 und setze $g = 1$
  while $\|A \sum_\ell^g X_\ell - I\|_F > \varepsilon$ **do**
    Setze $g := g + 1$ und initialisiere $X_{gj} := I_{n_j \times n_j}$
    for $r = 1 : p$ **do**
      $$B_{gr} := \sum_{i,k=1} N \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj}\right) A_{ir}^t A_{kr}$$
      $$Y_{gr} := \sum_{i=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(A_{ij} X_{gj}\right) A_{ir} - \sum_{\ell=1}^{g-1} \sum_{i,k=1}^{n} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right) A_{ir}^t A_{kr} X_{\ell r}$$
      Löse $B_{gr} X_{gr} = Y_{gr}$ mittels „Least-squares“
    end
  end
end

gesamt lässt sich der Algorithmus 4.2 zur Bestimmung der Kronecker-Inversen angeben.

Im Weiteren sollen die Effizienz und das Verhalten bzgl. der Konvergenz und der Abklingrate des Algorithmus 4.2 untersucht werden, da der Algorithmus gerade bei Matrizen mit sehr vielen Summanden bereits nach wenigen Iterationen sehr rechenlastig wird. Bei Tensorprodukt Smoothing Splines lässt sich allerdings die symmetrische Rang-1 Struktur des Approximationsteils $(N^\mu)^t N^\mu$ ausnutzen.

4.2. Effiziente Umsetzung der Approximation einer Kronecker-Inversen

Aufgrund der großen Summandenanzahl wird die Berechnung mit jedem Summanden, welcher der Kronecker-Inversen hinzugefügt wird, aufwendiger. Die Matrix A hat bei Tensorprodukt Smoothing Splines allerdings eine gewisse

Struktur (siehe 2.2), welche zur effizienten Berechnung ausgenutzt werden kann. Wie bereits erwähnt, sind alle Faktoren der Kronecker-Produkte symmetrisch und bei den meisten Kronecker-Produkten bestehen die Faktoren aus Rang-1 Matrizen. Diese Struktur kann ausgenutzt werden, um die Anzahl der Rechenoperationen des Algorithmus 4.2 in zwei Schritten zu verbessern. Da die Summe zweier symmetrischer Matrizen wieder symmetrisch ist

$$(C+D)^t = D^t + C^t = D + C = C + D, \quad C, D \in \mathbb{R}^{n\times n}$$

und Matrizen mit den passenden Dimensionen bei der Berechnung der Spur vertauscht werden können, lässt sich die Berechnung der linken Seite im ersten Schritt wie folgt verbessern:

$$\begin{aligned} B_{gr} &= \sum_{i,k=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj}\right) A_{ir}^t A_{kr} \\ &= \sum_{i=1}^{N}\sum_{k=i}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{gj}\right) \left(\underbrace{A_{ir}^t A_{kr}}_{=:\tilde{A}} + \underbrace{A_{kr}^t A_{ir}}_{=:\tilde{A}^t}\right). \end{aligned} \tag{4.13}$$

Es muss nur noch die Hälfte an Matrixprodukten sowie an Spuren berechnet werden. Dies kann zum Teil auch bei der rechten Seite des Gleichungssystems ausgenutzt werden. Es gilt analog zu (4.13), dass

$$\begin{aligned} &-\sum_{\ell=1}^{g}\sum_{i,k=1}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right) A_{ir}^t A_{kr} X_{\ell r} = \\ &\quad = -\sum_{\ell=1}^{g} \underbrace{\left(\sum_{i=1}^{N}\sum_{k=i}^{N} \prod_{\substack{j=1 \\ j\neq r}}^{p} \operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right)\left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir}\right)\right)}_{=:\tilde{Y}_{\ell r}} X_{\ell r}. \end{aligned} \tag{4.14}$$

Es können die gleichen Einsparungen wie für die linke Seite des LGS ausgenutzt und die Matrix $\tilde{Y}_{\ell r}$ kann einzeln aufgestellt werden. Aufgrund der gleichen Struktur kann die Berechnung der Summen in (4.13) und (4.14) bei der Implementierung zusammengezogen werden. Pro Iterationsschritt verändert sich nur die Matrix X_{gj}. Wenn genügend Speicherplatz zur Verfügung steht, lässt sich dies ausnutzen, indem die Matrizen $(A_{ir}^t A_{kr} + A_{kr}^t A_{ir})$ zwischengespeichert

und nur die benötigten Koeffizienten bzw. Spuren neu berechnet werden. Die größte Einsparung an Rechenoperationen lässt sich im zweiten Schritt durch die Ausnutzung der Struktur der symmetrischen Rang-1 Matrizen bei dem Approximationsterm des Smoothing Splines erreichen. Genauer gesagt bei der „Least-squares"-Vandermonde-Matrix, da diese für den Großteil der Summanden verantwortlich ist. Die oben verwendeten Matrizen A_{ir} und A_{kr} lassen sich in diesem Fall schreiben als

$$A_{ij} = a_{ir}^t a_{ir} \text{ sowie } A_{kr} = a_{kr}^t a_{kr}. \tag{4.15}$$

Die linke Seite des LGS lässt sich damit wie folgt vereinfachen:

$$\begin{aligned}
&\sum_{i=1}^{N}\sum_{k=i}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p} \operatorname{tr}\left(\underbrace{\underbrace{X_{gj}^t a_{ij}^t}_{\text{Spalte}}\underbrace{a_{ij}a_{kj}^t}_{\text{Skalar}}\underbrace{a_{kj}X_{gj}}_{\text{Zeile}}}_{\text{Rang-1 Matrix } \tilde{x}_{gij}\tilde{x}_{gkj}^t}\right)\left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir}\right)\\
&\qquad = \sum_{i=1}^{N}\sum_{k=i}^{N}\left(\prod_{j=1}^{p} a_{ij}a_{kj}^t\right)\left(\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(\tilde{x}_{gij}\tilde{x}_{gkj}^t\right)\right)\left(a_{ir}^t a_{kr} + a_{kr}^t a_{ir}\right)\\
&\qquad = \sum_{i=1}^{N}\sum_{k=i}^{N}\left(\prod_{j=1}^{p} a_{ij}a_{kj}^t\right)\left(\prod_{\substack{j=1\\ j\neq r}}^{p}\left(\tilde{x}_{gkj}^t\tilde{x}_{gij}\right)\right)\left(a_{ir}^t a_{kr} + a_{kr}^t a_{ir}\right).
\end{aligned}$$

Dies lässt sich analog auf die rechte Seite des LGS anwenden und es ergibt sich

$$\begin{aligned}
&\sum_{i=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(A_{ij}X_{gj}\right)A_{ir}\\
&\qquad -\sum_{\ell=1}^{g-1}\left(\sum_{i=1}^{N}\sum_{k=i}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj}X_{\ell j}\right)\left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir}\right)X_{\ell r}\right)\\
&\quad = \sum_{i=1}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(a_{ij}^t\underbrace{a_{ij}X_{gj}}_{\text{Zeile }\tilde{x}_{igj}^t}\right)a_{ir}^t a_{ir}\\
&\qquad -\sum_{\ell=1}^{g-1}\left(\sum_{i=1}^{N}\sum_{k=i}^{N}\prod_{\substack{j=1\\ j\neq r}}^{p}\operatorname{tr}\left(\underbrace{X_{gj}^t a_{ij}^t}_{\text{Spalte }\tilde{x}_{igj}}\underbrace{a_{ij}^t a_{kj}}_{\text{Skalar}}\underbrace{a_{kj}X_{\ell j}}_{\text{Zeile }\tilde{x}_{\ell kj}^t}\right)\right.
\end{aligned}$$

$$\left(a_{ij}^t \underbrace{a_{ij} a_{kj}^t}_{\text{Skalar}} a_{kj} + a_{kj}^t \underbrace{a_{kj} a_{ij}^t}_{\text{Skalar}} a_{ij} \right) X_{\ell r} \Bigg)$$

$$= \sum_{i=1}^{N} \prod_{\substack{j=1 \\ j \neq r}}^{p} \left(\tilde{x}_{igj}^t a_{ij} \right) a_{ir}^t a_{ir}$$

$$- \sum_{\ell=1}^{g-1} \left(\sum_{i=1}^{N} \sum_{k=i}^{N} \left(\prod_{j=1}^{p} a_{ij} a_{kj}^t \right) \prod_{\substack{j=1 \\ j \neq r}}^{p} \tilde{x}_{\ell kj}^t \tilde{x}_{igj} \left(a_{ij}^t a_{kj} + a_{kj}^t a_{ij} \right) X_{\ell r} \right).$$

Mit diesen Umformungen lässt sich der Algorithmus 4.2 effizient umsetzen. Es soll kurz verdeutlicht werden, dass diese Umstellung wirklich eine Verbesserung darstellt. Dafür wird die Anzahl der Rechenoperationen bei vollbesetzten und Rang-1 Matrizen für den Term

$$\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j} \right) \left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir} \right) X_{\ell r}, \text{ mit } A_{ij},\ X_{\ell,j} \in \mathbb{R}^{n_j \times n_j} \tag{4.16}$$

für $j = 1, \ldots, p$, verglichen. Dieser Term wird gewählt, da er insgesamt am meisten berechnet werden muss. Andere Terme fallen nicht weiter ins Gewicht, da diese sich nicht deutlich verbessern lassen. Aufgrund dessen reicht es aus nur diesen Term zu betrachten. Es wird speziell der Fall untersucht, der bei Tensorprodukt Smoothing Splines auftritt. Es ist folglich eine bekannte Rang-1 Struktur vorhanden. Für vollbesetzte Matrizen bzw. die vollständig aufgestellten Rang-1 Matrizen werden $8n_j^3 - 1$ Operationen zur Berechnung benötigt. Dies erklärt sich wie folgt: Die Berechnung von

$$\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j} \right) = vec\left(X_{\ell j}^t A_{kj}^t \right)^t vec\left(X_{gj}^t A_{ij}^t \right)$$

benötigt durch die Verwendung von Lemma 4.1 insgesamt $2n_j^2(2n_j - 1) + 2n_j^2 - 1$ Operationen. In

$$\left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir} \right) X_{\ell r}$$

muss nur ein Summand berechnet werden, da gilt $(A_{ir}^t A_{kr})^t = A_{ir}^t A_{kr}$. Für diesen Teil ergeben sich somit $2n_j^2(2n_j - 1) + n_j^2$ Rechenoperationen. Für die Multiplikation mit der Spur kommen nochmal n_j^2 Multiplikationen hinzu und insgesamt ergibt sich folgende Anzahl an Rechenoperationen

$$4n_j^2(2n_j - 1) + 2n_j^2 - 1 + 2n_j^2 = 8n_j^3 - 1.$$

Für die Umsetzung mittels bekannten Rang-1 Matrizen in (4.15) werden lediglich $10n_j^2 + 3n_j - 3$ Operationen benötigt. Dies lässt sich folgendermaßen erkennen: Die Spur kann umgestellt werden zu

$$\operatorname{tr}\left(X_{gj}^t A_{ij}^t A_{kj} X_{\ell j}\right) = \operatorname{tr}\left(X_{gj}^t a_{ij}^t a_{ij} a_{kj}^t a_{kj} X_{\ell j}\right),$$

so dass nur zwei Matrix-Vektor-Multiplikationen und äußere bzw. innere Produkte berechnet werden müssen. Unter Ausnutzung von Lemma 4.1 ergeben sich damit $(n_j + 1)(2n_j - 1) + 2n_j - 1$ Rechenoperationen. Für den zweiten Teil

$$\left(A_{ir}^t A_{kr} + A_{kr}^t A_{ir}\right) X_{\ell r} = \left(a_{ir}^t a_{ir} a_{kr}^t a_{kr} + a_{kr}^t a_{kr} a_{ir}^t a_{ir}\right) X_{\ell r}$$

ergeben sich $2n_j - 1 + 2n_j(2n_j - 1) + 2n_j^2$ Rechenoperationen. Inklusive der Multiplikation mit der Spur ergeben sich ingesamt

$$(n_j + 1)(2n_j - 1) + 2n_j - 1 + 2n_j - 1 + 2n_j(2n_j - 1) + 3n_j^2 = 10n_j^2 + 3n_j - 3$$

Rechenoperationen. Dieser Vergleich zeigt deutlich, dass es von Vorteil ist die Rang-1 Struktur auszunutzen, da die Berechnung in (4.16) genau $\left(N \cdot \frac{N}{2}(p-1)\right)$-mal für ein komponentenweises Minimierungsproblem auftritt. D.h. pro Iteration des obigen Prokrustes-Problem tritt die Berechnung $\left(N \cdot \frac{N}{2}(p-1) \cdot p\right)$-mal auf. Zur Erinnerung: Der Wert N bezeichnet die Stützstellenanzahl und der Wert p die Dimension. Diese liegen bei der untersuchten Problemstellung durchaus bei $N = 10000$ und $p = 6$. Ein Vorberechnen und Zwischenspeichern der Rang-1 Matrizen wäre somit von Nachteil.

Unter Verwendung der Komprimierung aus Kapitel 3 lässt sich die Approximation ebenfalls schneller umsetzen, dies hängt allerdings von der Güte der Komprimierung ab. In bestimmten Fällen kann es zu einer sehr langsamen Konvergenz bzw. Divergenz kommen, wie im nächsten Abschnitt beschrieben werden soll. Zur Bestimmung eines Vorkonditionierers für das GMRES-Verfahren ist diese Vorgehensweise allerdings bei guter Komprimierung vielversprechend. Die Ergebnisse dazu finden sich in Kapitel 6.2.

4.3. Konvergenzverhalten des vorgestellten Algorithmus

Es wurde mit Algorithmus 4.2 ein Verfahren zur Bestimmung einer Kronecker-Inversen vorgestellt und gezeigt, wie dieses Verfahren für den betrachteten Fall effizient umgesetzt werden kann. In diesem Abschnitt soll das Konvergenzverhalten des entwickelten Algorithmus untersucht werden. Zuerst soll gezeigt werden, dass für die Folge von Kronecker-Produkten $X_\ell,\ \ell = 1, 2, \ldots, g$, welche durch den Algorithmus erzeugt wird, die Norm

$$\left\| A \sum_{\ell=1}^{g} X_\ell - B \right\|_F^2$$

mit jedem weiteren Summanden abnimmt.

Satz 4.2 *Sei $A = \sum_{i=1}^{N} \otimes_{j=1}^{p} A_{ij}$, mit $A_{ij} \in \mathbb{R}^{m_j \times n_j}$ sowie symmetrisch und positiv semidefinit. Sei weiterhin $X_\ell = \otimes_{j=1}^{p} X_{\ell j}$, mit $X_{\ell j} \in \mathbb{R}^{m_j \times n_j}$ und die Summe $\sum_{\ell=1}^{g} \otimes_j X_{\ell j}$ durch den Algorithmus 4.2 bestimmt. Dann gilt*

$$\|AX_1 - I\|_F^2 \geq \|A(X_1 + X_2) - I\|_F^2 \geq \cdots \geq \left\| A \sum_{\ell=1}^{\infty} X_\ell - I \right\|_F^2 .$$

Beweis: Es soll gezeigt werden, dass die Norm

$$\left\| A \sum_{\ell=1}^{g} X_\ell - I \right\|_F^2$$

mit jeder Anwendung des Algorithmus 4.2 abnimmt. Anschaulich wird in jedem Schritt der Rest des vorherigen Schritts approximiert

$$\min_{X_{g+1}} \left\| A \sum_{\ell=1}^{g+1} X_\ell - I \right\|_F^2 = \min_{X_{g+1}} \left\| AX_{g+1} - \left(I - A \sum_{\ell=1}^{g} X_\ell \right) \right\|_F^2 .$$

Durch das Hinzufügen von X_{g+1} werden neue Freiheitsgrade verfügbar und die Matrix X_{g+1} wird mit Hilfes des Algorithmus 4.2 bestimmt, so dass für ein

$$X^* = \bigotimes_{j=1}^{p} X_j^* = \arg\min_X \left\| AX - \left(I - A \sum_{\ell=1}^{g} X_\ell \right) \right\|_F^2 ,$$

welches nicht zwingend ein globales Minimum sein muss, gilt

$$\left\| AX^* - \left(I - A \sum_{\ell=1}^{g} X_\ell \right) \right\|_F^2 \leq \left\| A \sum_{\ell=1}^{g} X_\ell - I \right\|_F^2 .$$

Es ist nicht weiter problematisch, dass X^* nicht zwingend ein globales Minimum ist, da es nur um die Minimierung des Abstands zum Residuum geht. Dies wird aufgrund der Symmetrie und der positiven Semidefinitheit in jedem Fall erreicht, da das komponentenweise „Least-squares"-Problem aus Algorithmus 4.2 in diesem Fall ein globales Minimum bzgl. der univariaten Komponente liefert. Mit der trivialen Wahl $X_{g+1} = 0$ ist die Ungleichung immer erfüllt. Mit $X_{g+1} = X^*$ für den jeweiligen Iterationsschritt folgt die Behauptung. □

Da es sich bei dem Algorithmus 4.2 um ein ALS-Verfahren handelt, wird die Aussage bereits in [3] und [36] für eine ähnliche Problemstellung erwähnt und bewiesen. Die Folge $(x_g)_{g\in\mathbb{N}}$ mit

$$x_g = \left\| A \sum_{\ell=1}^{g} X_\ell - I \right\|_F^2$$

ist nach unten durch Null beschränkt und monoton fallend. Dies reicht leider nicht aus, um zu zeigen, dass sie eine Nullfolge ist. Des Weiteren wird in [36] aufgezeigt, dass diesbezüglich keine Aussagen getroffen werden können. Numerische Tests für das betrachtete Problem deuten darauf hin, dass die Folge beliebig langsam abfallen kann. Dies ist abhängig von der zu invertierenden Matrix.

Vermutung 4.3 *Die Konvergenz der Summe $\sum_\ell X_\ell$ gegen die Inverse von A kann wahrscheinlich nicht gezeigt werden, was die Untersuchung der Folge x_g bereits vermuten lässt. Dies wird durch Aussagen in [36] ebenfalls unterstrichen und numerische Tests in Kapitel 6.2 deuten höchstens auf eine sehr langsame Konvergenz hin. Eine explizites Gegenbeispiel konnte allerdings bisher nicht gefunden werden.*

Wenn für die Matrizen des Kronecker-Produkts zusätzlich Orthogonalität gefordert wird, sollte die Summe gegen eine Inverse von A konvergieren. Diese Forderung führt allerdings zu einer größeren Komplexität des Problems und scheint in der Praxis nicht von Vorteil zu sein.

Für eine sinnvolle Verwendung als Vorkonditionierer des GMRES-Verfahrens reichen in bestimmten Fällen wenige Iterationen aus, so dass die nicht vorhandene Konvergenz bei der praktischen Anwendung kein wesentlicher Nachteil ist. Für diese explizite Verwendung müssen die beteiligten Matrizen allerdings symmetrisch und positiv semidefinit sein, damit der Algorithmus 4.2 verwendet werden kann. Dies ist bei Tensorprodukt Smoothing Splines der Fall und dadurch sind insbesondere die Voraussetzungen für das GMRES-Verfahren erfüllt. Entscheidend für die Güte der Approximation ist an dieser Stelle das Verhältnis zwischen Approximations- und Glättungsterm des Smoothing Splines. Ähnlich wie bei der Komprimierung in Kapitel 3 liefert die Approximation gute Ergebnisse und ist verwendbar, wenn der Glättungsterm stärker gewichtet wird als der Approximationsterm. Dies liegt daran, dass sich in diesem Fall die Gram-Matrix durchsetzt.

Der Glättungsparameter λ muss folglich über einem gewissen Schwellwert liegen, wie bereits in der Vermutung 3.16 für die Komprimierung formuliert wurde. Der Schwellwert bzgl. λ für eine effiziente Komprimierung und Approximation der Inversen ist damit identisch zu wählen. Liegt λ über dem erwähnten Schwellwert, ist ein weiterer Vorteil, dass anstelle der Originaldaten die Komprimierung zur Bestimmung einer Approximation einer Kronecker-Inversen verwendet werden kann. Dadurch können massiv Rechenoperationen eingespart werden. Der Algorithmus liefert allerdings keine befriedigenden Ergebnisse, wenn der Approximationsterm wesentlich stärker gewichtet wird als der Glättungsterm. Es fällt dabei auf, dass die Folge x_g in diesem Fall sehr langsam abfällt und ab einem gewissen Punkt scheinbar stagniert. Dies zeigen die Ergebnisse in Kapitel 6.2.

5 Vorkonditionierer

Dieses Kapitel soll Möglichkeiten zur Vorkonditionierung des CG-Verfahrens vorstellen, welche die Kronecker-Struktur bei Tensorprodukt Smoothing Splines erhalten. Viele allgemeine Vorkonditionierer für das CG-Verfahren bzw. GMRES-Verfahren, welche gute Ergebnisse liefern, sind von vorneherein ausgeschlossen. Dies liegt an der Kronecker-Struktur, welche von zentraler Wichtigkeit ist und in jedem Fall erhalten bleiben muss. Im Unterkapitel 2.3.2 wird mit den Skalierungsvorkonditionierern eine Möglichkeit zur Vorkonditionierung für das CG-Verfahren bzw. GMRES-Verfahren und im Kapitel 4 mit der Approximation der Inversen eine Möglichkeit für das GMRES-Verfahren vorgestellt.

Diese beiden Vorkonditionierer können für das CG-Verfahren im Fall der Tensorprodukt Smoothing Splines nur bedingt überzeugen, da die Skalierungsvorkonditionierer zu wenig Informationen umfassen. Außerdem erfüllt die Approximation der Inversen prinzipiell nicht die Voraussetzung als Vorkonditionierer für das CG-Verfahren, da sie im Fall der betrachteten Smoothing Splines nicht positiv definit ist. Weiterhin kann die Annäherung der Inversen nur unter bestimmten Bedingungen als Vorkonditionierer für das GMRES-Verfahren gute Ergebnisse erzielen. Dies hängt im Wesentlichen von der Güte der Approximation ab. Wenn der Approximationsterm des Smoothing Splines stärker gewichtet wird als der Glättungsterm, nimmt die Qualität der Approximation der Inverse schnell ab. Dieses Verhalten ist analog bei der Komprimierung zu beobachten. Das LGS bzgl. des Tensorprodukt Smoothing Splines für das untersuchte Problem erfüllt die Bedingungen zur Verwendung des CG-Verfahrens, wobei der

Glättungsparameter $\lambda \neq 0$ sein muss. Da eine Interpolation bei den betrachteten Splines nicht sinnvoll scheint, ist dies immer der Fall. Weiterhin scheint das CG-Verfahren generell effizienter als das GMRES-Verfahren. Aufgrund dieser beiden Gründe ist es von Interesse das CG-Verfahren anzuwenden.

Aufgrund der oben genannten Defizite soll ein möglicher Vorkonditionierer für das CG-Verfahren vorgestellt werden. Dieser hat den Vorteil, dass er sowohl mit einer guten Komprimierung aus Kapitel 3 umgesetzt werden kann als auch separat relativ effizient verwendbar ist, falls die Güte der Komprimierung nicht den Ansprüchen genügt. Am Ende wird zusätzlich eine Idee für einen Vorkonditionierer skizziert, welcher auf dem Prokrustes-Problem basiert.

5.1. SSOR-Vorkonditionierer

Es wurde bereits erwähnt, das sogenannte Splitting-Verfahren nicht zur iterativen Lösung des LGS verwendet werden können, da sie die Kronecker-Struktur nicht ausnutzen und prinzipiell Zugriff auf die gesamte Matrix benötigen. Sie bieten allerdings in einer abgewandelten Form die Möglichkeit als Vorkonditionierer für das CG-Verfahren verwendet zu werden. Dies wird in [21, 35] und [2] beschrieben und erläutert. Es kommen wieder nicht alle Verfahren in Frage. Eine mögliche Umsetzung für einen Vorkonditionierer stellt jedoch das symmetrische Gauß-Seidel-Verfahren dar, ebenfalls bekannt als „Symmetric Successive Overrelaxation“(SSOR)-Vorkonditionierer. Prinzipiell hält dieses Verfahren die Kronecker-Struktur nicht intakt. Aufgrund der Struktur der Matrix von Tensorprodukt Smoothing Splines auf gestreuten Daten lässt sich dieses Problem aber umgehen, wie später zu sehen sein wird. Die Grundlagen des Verfahrens sind aus [35] und [21] entnommen.

Für den Algorithmus 2.3 des vorkonditionierten CG-Verfahrens muss eine Matrix C bestimmt werden, welche die Vorkonditionierung umsetzt (siehe Unterkapitel 2.3.2). Diese Matrix $C \in \mathbb{R}^{n\times n}$ kann bei dem SSOR-Vorkonditionierer wie folgt gewählt werden:

$$C = \left((D+L)D^{-1}(D+L)^t\right)^{-1}, \text{ mit } L, D \in \mathbb{R}^{n\times n},$$

wobei $A \in \mathbb{R}^{n\times n}$ die Matrix des LGS und in $A = L + D + L^t$ zerlegbar ist. Es steht L für eine linke, untere Dreiecksmatrix und D für eine Diagonalmatrix.

Die beiden Matrizen sind in der betrachteten Problemstellung regulär. Es gilt insbesondere $C = KK^t$ mit $K = (D+L)^{-t}D^{\frac{1}{2}}$. Dies ist relevant, damit die Symmetrie und die positive Definitheit für das CG-Verfahren nicht verletzt werden. Eine direkte Umsetzung von

$$h_{k+1} = Cr_{k+1}, \tag{5.1}$$

siehe Algorithmus 2.3, ist bedingt durch den Verlust der Kronecker-Struktur bei der Invertierung nicht möglich und wäre numerisch nicht sinnvoll. Eventuell könnte die Approximation einer Kronecker-Inversen aus Kapitel 4 an dieser Stelle weiterhelfen, aber aus Zeitgründen konnte dies nicht untersucht werden. Die Gleichung (5.1) kann zu $C^{-1}h_{k+1} = r_{k+1}$ umgestellt werden. Dies hat zur Folge, dass ein lineares Gleichungssystem gelöst werden muss, bei dem jedoch die Kronecker-Struktur erhalten bleibt.

Da C^{-1} aus einem Produkt von oberen und unteren Dreiecksmatrizen besteht, lässt sich dieses lineare Gleichungssystem bekanntermaßen durch eine Vorwärtselimination und eine Rücksubstitution effizient lösen, siehe [35]. Diese wird zeilenweise berechnet und es lässt sich die zugrundeliegende Kronecker-Struktur ausnutzen. Die Matrix L wird aus einer Summe von linken, unteren Dreiecksmatrizen aufgebaut, welche durch Kronecker-Produkte darstellbar sind. Diese Dreiecksmatrizen stammen aus der Aufteilung der Matrix $(N^\mu)^t N^\mu + \lambda G_\mu$ in $L+D+L^t$ und es werden die unteren Dreiecksmatrizen der einzelnen Kronecker-Faktoren verwendet, wodurch die obige Zerlegung zustande kommt. Es müssen weiterhin nur die Summanden der Matrix $(N^\mu)^t N^\mu + \lambda G_\mu$ betrachtet werden, welche für die untere Dreiecksmatrix und die jeweilige Zeile bei der Vorwärtselimination und Rücksubstitution relevant sind. In der Praxis bringt die Unterscheidung zwischen unterer und oberer Dreiecksmatrix bei der untersuchten Problemstellung für die Auswahl der relevanten Summanden nur wenig Vorteile, sodass auf diese verzichtet werden kann. Das sollte bei der Berechnung von h_k beachtet werden. Diese lässt sich folgendermaßen umsetzen: Im ersten Schritt wird die Vorwärtselimination durch

$$C^{-1}h_k = r_k \Leftrightarrow (D+L)\underbrace{D^{-1}(D+L)^t h_k}_{=\tilde{h}_k} = r_k \Leftrightarrow (D+L)\tilde{h}_k = r_k. \tag{5.2}$$

realisiert. Um die elementweise Berechnung von h_k übersichtlicher darstellen zu

können, wird analog zu den vorherigen Kapiteln die `Octave`-Schreibweise verwendet. Zur Erinnerung, diese hat folgende Bezeichnungen: Das i-te Element eines Vektors a wird mit $a(i)$ bezeichnet und analog für eine Matrix A soll gelten, dass $A(i,j)$ das Element in der i-ten Zeile und der j-ten Spalte bezeichnet. Mit der Notation $A(i,1:j)$ sind die Elemente $A(i,1)$ bis $A(i,j)$ der Matrix A gemeint. Folglich entspricht $A(i,1:j)$ einem Vektor. Die Bezeichung $A(i,end)$ meint das letzte Element der i-ten Zeile. Damit lassen sich die einzelnen Elemente $\tilde{h}_k(i)$ aus (5.2) durch folgende Formel berechnen:

$$\tilde{h}_k(i) = \frac{r_k(i) - L(i,1:i-1)\tilde{h}_k(1:i-1)}{D(i,i)},$$

wobei $\tilde{h}_k(1:i-1)$ die bisher berechneten Elemente von $\tilde{h}_k$ sind. Das Skalarprodukt

$$L(i,1:i-1) \cdot \tilde{h}(1:i-1)$$

lässt sich mit Hilfe des Kronecker-Produkts bestimmen, da $L(i,1:i-1)$ als Summe von zeilenweisen Kronecker-Produkten darstellbar ist. Es macht daher Sinn, nicht jede Zeile direkt auszurechnen, sondern die rekursive Matrix-Vektor-Multiplikation bei Kronecker-Produkten auszunutzen. Wenn der Speicher allerdings ausreicht, kann zur Vereinfachung der Implementierung jede Zeile direkt berechnet und nur der relevante Teil verwendet werden. Ein Kompromiss zwischen Speicher- und Rechenaufwand wäre an dieser Stelle ratsam. Im zweiten Schritt folgt die eigentliche Berechnung von h_k mittels Rücksubstitution

$$(D+L)^t h_k = D\tilde{h}_k.$$

Damit ergibt sich wiederum eine Formel für die zu bestimmenden Elemente von h_k

$$\begin{aligned} h_k(i) &= \frac{D(i,i)\tilde{h}_k(i) - L(i+1:end)h_k(i+1:end)}{D(i,i)} \\ &= \tilde{h}_k(i) - \frac{L(i+1:end)h_k(i+1:end)}{D(i,i)}. \end{aligned} \tag{5.3}$$

Die Struktur von $L(i+1:end)$ lässt sich an dieser Stelle wiederum ausnutzen. Ein wesentliches Problem bei der Umsetzung ist die große Anzahl von Summanden bei der Vandermonde-Matrix. Wenn eine gute Komprimierung vorliegt, lässt

sich diese verwenden. Somit wird das Problem mit der großen Anzahl an Summanden umgangen. Dies hängt jedoch davon ab, wie gut die Komprimierung aus Kapitel 3 ist.

Falls die Komprimierung den Ansprüchen nicht genügt, um damit den SSOR-Vorkonditionierer zu approximieren, dann lässt sich die große Anzahl von Summanden auf eine andere Art und Weise umgehen. Dabei ist relevant, dass eine Zeile einer univariaten Vandermonde-Matrix in den meisten Fällen nicht voll besetzt ist, was durch den Träger der B-Splines begründet werden kann, wie später zu sehen sein wird. Wegen dem Kronecker-Produkt mit mehreren Zeilen von univariaten Vandermonde-Matrizen sind daher die meisten Einträge einer Zeile der multivariaten Vandermonde-Matrix gleich Null. Aufgrund des Aufbaus von $(N^\mu)^t N^\mu$ ist damit nur ein geringer Anteil der Summanden an einer Zeile der Matrix $(N^\mu)^t N^\mu$ beteiligt. Dies soll im Folgenden genauer untersucht werden:

Eine Zeile der Matrix $(N^\mu)^t$ entspricht der Auswertung der Stützstellen an einem Tensorprodukt B-Spline. Die von Null verschiedenen Einträge entsprechen den Stützstellen, welche im Träger des Tensorprodukt B-Splines liegen. Aufgrund der Rang-1 Struktur der Matrix

$$(N^\mu)^t N^\mu = \sum_{i=1}^{\#X} \bigotimes_{j=1}^{p} N^{\mu_j}(x_{ij}|T_j)^t N^{\mu_j}(x_{ij}|T_j)$$

entspricht die Anzahl der relevanten Summanden für die j-te Zeile der Matrix $(N^\mu)^t N^\mu$ der Anzahl der von Null verschiedenen Elemente aus der j-ten Zeile von $(N^\mu)^t$. Damit ist nur eine grobe Abschätzung möglich, da für eine solche die univariaten Komponenten der Stützstellen in den Trägern der univariaten B-Splines betrachtet werden müssen. Wie viele Stützstellen im Träger eines univariaten B-Splines liegen, ist von den Stützstellen und der Wahl der Knoten abhängig. Bei einem B-Spline vom Grad m umfasst der Träger $2m + 1$ Knotenintervalle. Bei einem p-dimensionalen multivariaten B-Spline, der sich aus solchen univariaten B-Splines vom Grad m zusammensetzt, umfasst der Träger $(2m + 1)^p$ Hyperwürfel.

Liegen die Daten auf einem Gitter, würde die Anzahl der Stützstellen im Träger des Tensorprodukt B-Splines ebenfalls exponentiell anwachsen. Aufgrund der gestreuten Daten im betrachteten Fall ist die Anzahl der Stützstellen im Träger des multivariaten B-Splines wesentlich geringer. Dieser kann sogar weniger Stütz-

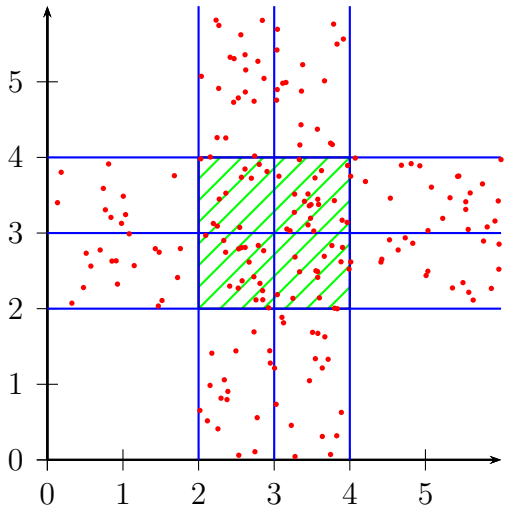

Abbildung 5.1.: Träger eines multivariaten B-Splines bei gestreuten Daten

Die Abbildung verdeutlicht, dass bei gestreuten Daten im Träger (schraffierter Bereich) eines multivariaten B-Splines weniger Stützstellen liegen können als in den Trägern der zugehörigen univariaten B-Splines, welche dem Intervall $[2, 4]$ *entsprechen.*

stellen enthalten als der univariate Träger. Die Unterbestimmtheit des betrachteten Problems liefert einen weiteren Hinweis auf die geringe Anzahl von Stützstellen im Träger des Tensorprodukt B-Splines. Es kann beobachtet werden, dass sich dieses Phänomen bei höheren Dimensionen verstärkt. Der beschriebene Sachverhalt ist in Abbildung 5.1 veranschaulicht.

Diese Beobachtung lässt sich ausnutzen, indem bei der ersten Verwendung des Vorkonditionierers für jede Zeile die beteiligten Summanden bestimmt werden. Im nächsten Durchgang werden dann nur diese für die jeweilige Zeile bei der Berechnung verwendet. Dadurch verringert sich der Rechenaufwand massiv und die Verwendung des SSOR-Vorkonditionierers lässt sich effizient umsetzen. In Abbildung 5.2 ist ein Diagramm zu finden, welches an einem Beispiel die Anzahl der relevanten Summanden pro Zeile der Matrix $(N^\mu)^t N^\mu$ zeigt. Der Vorkonditionierer ist weiterhin vorteilhaft, da er wenig zusätzlichen Speicher benötigt und ohne Mehraufwand bestimmt werden kann. Nachteilig ist, dass im Vergleich zum ursprünglichen Verfahren bei seiner Anwendung pro Iteration mehr Rechenaufwand entsteht.

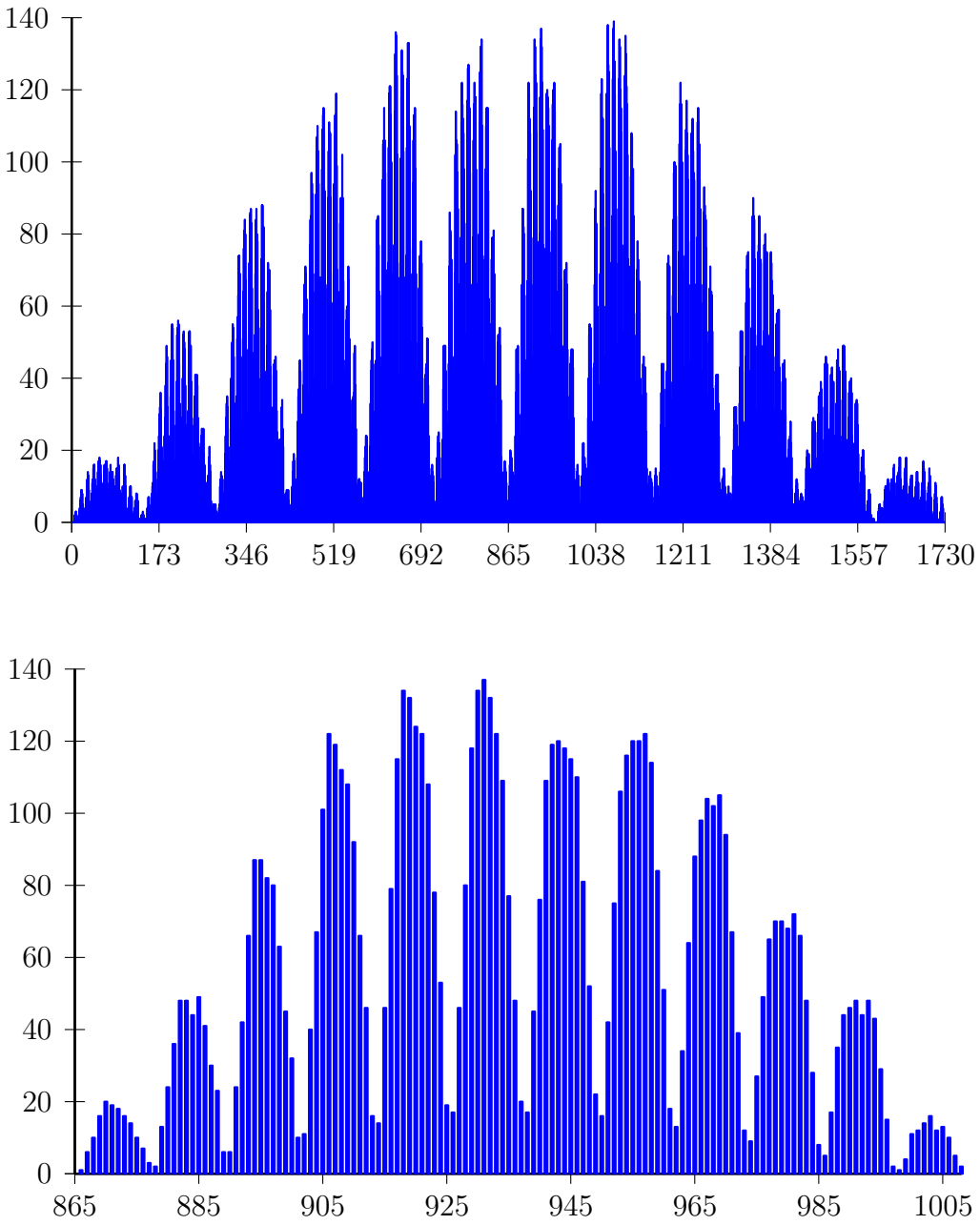

Abbildung 5.2.: Anzahl relevanter Summanden pro Zeile der „Least-squares“ Vandermonde-Matrix

Das obige Diagramm zeigt den Anteil von relevanten Summanden pro Zeile der Matrix $N^{\mu}(X|T)^t N^{\mu}(X|T)$. *Das Beispiel bezieht sich auf den Datensatz* NH_3 *in vier Dimensionen aus [44]. Es wurden* 12 *innere Knoten pro Dimension gewählt. Es ist zu sehen, dass die Anzahl der inneren Knoten der Anzahl der Spitzen im Diagramm entspricht. In dem unteren Diagramm wird eine Spitze des obigen Diagramms einzeln und vergrößert dargestellt. Es sind wieder* 12 *Spitzen zu erkennen, was mit dem Tensorprodukt der Knoten zusammenhängt. Insgesamt sind es* 1009 *Summanden, aber pro Zeile sind maximal* 140 *Summanden involviert.*

Eine mögliche Verbesserung des SSOR-Vorkonditionierers wäre die Bestimmung eines Relaxations-Parameters ω, sodass die Matrix

$$C = \left(\frac{1}{2-\omega}\left(\frac{1}{\omega}D + L\right)\left(\frac{1}{\omega}D\right)^{-1}\left(\frac{1}{\omega}D + L\right)^{t}\right)^{-1}, \text{ mit } \omega \in (0,2),$$

für die Vorkonditionierung verwendet werden kann. Dies scheint allerdings nicht effizient, da die mögliche Verbesserung nicht in Relation zu dem Aufwand der Bestimmung des Parameters ω steht. Die Ergebnisse zum SSOR-Vorkonditionierer finden sich in Kapitel 6.3.

5.2. Prokrustes-Vorkonditionierer für das CG-Verfahren

Es soll ein zweiter Vorkonditionierer bzgl. der Matrix $A \in \mathbb{R}^{n\times n}$, $n = \prod_{j=1}^{p} n_j$ für das CG-Verfahren vorgestellt werden, welcher im Zusammenhang mit Tensorprodukt Smoothing Splines ein gewisses Potenzial besitzt. Er basiert auf der Idee des Prokrustes-Problems und des ALS-Verfahrens analog zu der Approximation der Inversen in Kapitel 4. Allerdings werden die Anforderungen an einen Vorkonditionierer für das CG-Verfahren mit einbezogen. Es ergibt sich folgendes Minimierungsproblem

$$\min_{X,Y} \|YAX - I\|_F^2 + \omega\|X - Y^t\|_F^2, \text{ mit } \omega > 0 \tag{5.4}$$

wobei

$$\mathbb{R}^{n\times n} \ni X = \bigotimes_{j}^{p} X_j, \; X_j \in \mathbb{R}^{n_j\times n_j} \text{ sowie } \mathbb{R}^{n\times n} \ni Y = \bigotimes_{j}^{p} Y_j, Y_j \in \mathbb{R}^{n_j\times n_j},$$

und

$$A = \sum_{i=1}^{N}\bigotimes_{j}^{p} A_{ij}, \; A_{ij} \in \mathbb{R}^{n_j\times n_j},$$

für $j = 1,\ldots,p$. Der zweite Summand in (5.4) dient als Strafterm, um eine Symmetrie zwischen den Matrizen X und Y zu fordern. Die Minimierung des Ausdrucks in (5.4) erfolgt über die partielle Ableitung der einzelnen Komponenten X_j und Y_j. Dies soll nicht weiter ausgeführt werden, da es analog zu der Vorgehensweise in Kapitel 4 durchgeführt werden kann.

Für die komponentenweise Minimierung von X_ℓ muss folgendes LGS gelöst werden:

$$B_\ell^x X_\ell = Z_\ell^x$$

mit

$$B_\ell^x := \sum_{j,i} \prod_{k \neq \ell} \operatorname{tr}\left(X_k^t A_{jk}^t Y_k^t Y_k A_{ik} X_k\right) A_{j\ell}^t Y_\ell^t Y_\ell A_{i\ell} + \omega \prod_{k \neq \ell} \operatorname{tr}(X_k^t X_k) I \tag{5.5}$$

und

$$Z_\ell^x := \sum_{j} \prod_{k \neq \ell} \operatorname{tr}\left(X_k^t A_{jk}^t Y_k^t\right) A_{j\ell}^t Y_\ell^t + \omega \prod_{k \neq \ell} \operatorname{tr}\left(X_k^t Y_k^t\right) Y_\ell^t. \tag{5.6}$$

Das LGS zur Minimierung für Y_ℓ ergibt sich analog durch

$$B_\ell^y Y_\ell = Z_\ell^y,$$

wobei

$$B_\ell^y := \sum_{j,i} \prod_{k \neq \ell} \operatorname{tr}\left(Y_k^t A_{jk}^t X_k^t X_k A_{ik} Y_k\right) A_{j\ell}^t X_\ell^t X_\ell A_{i\ell} + \omega \prod_{k \neq \ell} \operatorname{tr}(Y_k^t Y_k) I \tag{5.7}$$

und

$$Z_\ell^y := \sum_{j} \prod_{k \neq \ell} \operatorname{tr}\left(Y_k^t A_{jk}^t X_k^t\right) X_\ell A_{j\ell} + \omega \prod_{k \neq \ell} \operatorname{tr}\left(Y_k X_k\right) X_\ell. \tag{5.8}$$

Mittels ALS ergibt sich der Algorithmus 5.1. Die beiden Matrizen X und Y lassen sich als Vorkonditionierer für das CG-Verfahren verwenden. Dabei ist zu beachten, dass nicht das vorkonditionierte CG-Verfahren verwendet wird, da die Matrix $C = YX$ nicht den an das vorkonditionierte CG-Verfahren gestellten Anforderungen genügt. Dies liegt daran, dass C nicht wie in Unterkapitel 2.3.2 gefordert in $C = K^t K$ zerlegt werden kann. Die Matrix-Vektor-Multiplikation Av des CG-Verfahrens wird durch $(YAX)v$ ersetzt. Der zusätzliche Rechenaufwand für das CG-Verfahren ist dadurch minimal. Die Bestimmung der Matrizen X und Y ist allerdings sehr aufwendig. Weiterhin kann bei praktischen Tests festgestellt werden, dass die Vorkonditionierung nur gut funktioniert, wenn der Glättungsparameter in einem bestimmten Intervall liegt. Dieses Intervall ist allerdings vorher nicht zu erkennen.

Bei der Implementierung des Algorithmus 5.1 lassen sich einige Optimierungen vornehmen, auf welche an dieser Stelle nicht weiter eingegangen wird, da die

Algorithmus 5.1 : Algorithmus zur Bestimmung eines Prokrustes-Vorkonditionierers für das CG-Verfahren

Input : $\mathbb{R}^{n\times n} \ni A = \sum_i^N \bigotimes_j^p A_{ij}$, mit $A_{ij} \in \mathbb{R}^{n_j \times n_j}$, $\varepsilon, \omega > 0$
Output : Kronecker-Produkte $\bigotimes_j X_j, \bigotimes_j Y_j$

```
1  begin
2      Initialisiere X_j = (Σ_k A_kj)^{-1} und Y_j = (Σ_k A_kj)^{-t} mit j = 1,...,p
3      while ‖YAX − I‖_F + ω‖X − Y^t‖_F > ε do
4          for j = 1 : p do
5              Berechne B_ℓ^x sowie Z_ℓ^x mit (5.5) und (5.6).
6              Löse B_ℓ^x X_ℓ = Z_ℓ^x mittels „Least-squares“.
7              Berechne B_ℓ^y sowie Z_ℓ^y mit (5.7) und (5.8).
8              Löse B_ℓ^y Y_ℓ = Z_ℓ^y mittels „Least-squares“.
9          end
10     end
11 end
```

theoretischen Defizite derzeit überwiegen. Die Bestimmung des Parameters ω wird zum Beispiel bisher nur heuristisch durchgeführt. Trotzdem hat die Idee ein gewisses Potential und sollte deswegen vorgestellt werden. Weiterhin sollte es möglich sein die Komprimierung aus Kapitel 3 zu verwenden. Im Rahmen dieser Arbeit konnten aber nur bedingt befriedigende Resultate erzielt werden und die Vorstellung soll nur als möglicher Anreiz für weitere Untersuchungen dienen.

6 Numerische Ergebnisse

Im folgenden Kapitel sollen die numerischen Ergebnisse der einzelnen Abschnitte vorgestellt werden. Die Verfahren wurden in `Octave` programmiert, wobei an manchen Stellen aus Effizienzgründen auf `oct-files` zurückgegriffen wurde. `Oct-files` bieten die Möglichkeit Quellcode mittels `C` zu implementieren und diesen in `Octave` einzubinden. Dies hat den Vorteil `for`-Schleifen sehr schnell durchlaufen zu können. In `Octave` sind `for`-Schleifen sehr langsam umgesetzt. Dort sollten diese vermieden und durch Matrix- bzw. Vektor-Multiplikationen umgesetzt werden, da `Octave` dafür optimiert ist. Dies ist aber nicht in jedem Fall zu realisieren, weswegen als Alternative `oct-files` existieren.

Die gesamten Berechnungen der Ergebnisse wurden auf einem Desktop-PC mit „AMD Phenom II X4 965“-Prozessor, 3.4 Ghz, 16 GiB Arbeitspeicher und Ubuntu 14.04, 64 Bit LTS, als Betriebssystem realisiert. Als Programmiersprache diente `Octave` 3.8.1. Die Datensätze der chemischen Reaktionen aus [44], welche bereits in [24] als Testgrundlage dienten, wurden als Testdaten in dieser Arbeit verwendet. Der Quellcode ist online unter

`https://subversion.forwiss.uni-passau.de/publications/Dissertation-Lamping`

zu finden. Eine ausführlichere Erläuterung der verwendeten Datenstrukturen und des Quellcodes sowie dessen Anwendung ist im Anhang A beschrieben. Die jeweiligen Tests und die verwendeten Fehlerberechnungen sind in den einzelnen Unterkapiteln erläutert.

6.1. Komprimierung

Die beschriebenen Verfahren wurden an allen Daten der chemischen Problemstellung getestet. Es werden aber nur exemplarische Ergebnisse von bestimmen Daten vorgestellt, da sich die Resultate in allen Fällen ähneln. Zum besseren Verständnis: Der Algorithmus 3.1 wird iterativ angewendet. Zur Veranschaulichung werden immer die ersten 50 Singulärwerte aus Algorithmus 3.1 sowie die ersten 50 Fehlerwerte ($\|A - \sum_i^k X_i\|_F$) bzgl. Algorithmus 3.1 nebeneinander geplottet. Da für die vierdimensionalen Daten nur Datensätze der NH_3-Daten zur Verfügung stehen, werden die Datensätze `NH3_1000A4D` und `NH3_100004D` zu Beginn betrachtet. Zum besseren Vergleich werden noch die fünfdimensionalen Datensätze `NH3_1000A5D` und `NH3_100005D` hinzugefügt. Um zu verdeutlichen, dass bei den anderen Daten ähnliche Resultate vorliegen, werden im Anschluss die Ergebnisse für die fünfdimensionalen Datensätze `N2_1000A` und `N2_10000` vorgestellt.

Es wurde wie folgt vorgegangen: Mit den erwähnten Datensätzen wurde die Matrix für das LGS des Tensorprodukt Smoothing Splines als Summe von Kronecker-Produkten aufgestellt. Die Anzahl der inneren Knoten beträgt 12, da diese Anzahl laut [24] für gute Ergebnisse ausreichend ist. Die Umformung in eine Multilinearform ist trivial und es kann der Algorithmus 3.1 wiederholt darauf angewendet werden, um eine Approximation zu bestimmen. In Abbildung 6.1 sind die Ergebnisse für die NH_3-Daten zu finden. Der Glättungsparameter beträgt in diesem Fall $\lambda = 1$, wodurch der Glättungsanteil im Fall der betrachteten Daten stärker als der Approximationsanteil gewichtet ist. Dies deuten die separat berechneten $\sigma_A \approx 6.7$ des Approximations- und $\sigma_G \approx 4871.9$ des Glättungsterms für `NH3_1000A4D` an. In der Abbildung lässt sich ein schnelles Abfallen der beschriebenen Werte erkennen. Dies wirkt sich auf die Komprimierung aus. Die ersten 20 Summanden sind für eine gute Komprimierung ausreichend, da sie die meisten Informationen beinhalten. Dies lässt sich deutlich in der Abbildung erkennen.

In Abbildung 6.2 sind die Ergebnisse für dieselben Daten mit dem Glättungsparameter $\lambda = 0.0001$ abgebildet. Ein sehr langsames Abfallen ist zu erkennen, es ist nicht mehr so deutlich wie in der ersten Abbildung. Die Begründung dafür liefert der Wert des Glättungsparameters. Durch diesen ist der Glättungsanteil nicht stärker als der Approximationsanteil gewichtet. Die Singulär- und Fehler-

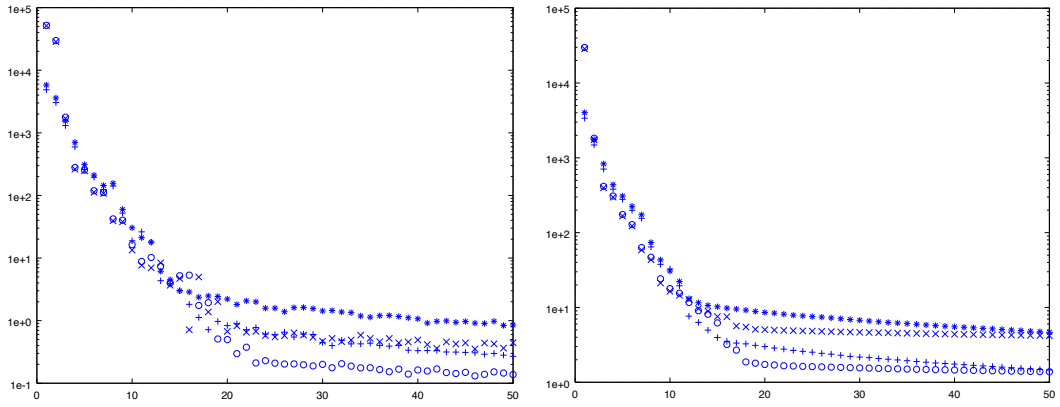

Abbildung 6.1.: Abfallen der Singulär- und Fehlerwerte bei einer Anwendung von Algorithmus 3.1 bei $NH3$-Daten und $\lambda = 1$

Die linke Abbildung zeigt die ersten 50 *Singulärwerte und die rechte die entsprechenden Fehlerwerte* $\|A - \sum_i^k X_i\|_F$ *bzgl. der iterativen Anwendung des Algorithmus 3.1. Aufgrund der Rechenungenauigkeit kann es zu kleinen Sprüngen bei den Werten kommen. Der Glättungsparameter hat den Wert* 1 *und die Daten entsprechen den folgenden Punkten:* $+$: `NH3_1000A4D`, $\circ$: `NH3_1000A5D`, $*$: `NH3_100004D` *sowie* $\times$: `NH3_100005D`.

werte verändern sich lediglich minimal, wenn der Wert des Glättungsparameters kleiner als 0.0001 gewählt wird. Der Approximationsanteil ist bei $\lambda = 0.0001$ schon so stark gewichtet, dass im Grunde nur eine Komprimierung von diesem berechnet wird. Die Komprimierung braucht in diesem Fall wesentlich mehr Summanden und selbst dann ist sie in vielen Fällen nicht gut genug, wie sich später zeigen wird. In der Abbildungen 6.3 sind der Vollständigkeit halber die Ergebnisse für die $N2$-Daten zu finden. In den folgenden beiden Unterkapiteln soll durch numerische Tests veranschaulicht werden, dass die Komprimierung effizient für weitere Verfahren verwendet werden kann, wenn der Glättungsterm stark genug einfließt.

6.2. Approximation der Kronecker-Inversen

Zuerst werden die Ergebnisse bezüglich der Approximation der Kronecker-Inversen für die Originaldaten und die komprimierten Daten vorgestellt. Im zwei-

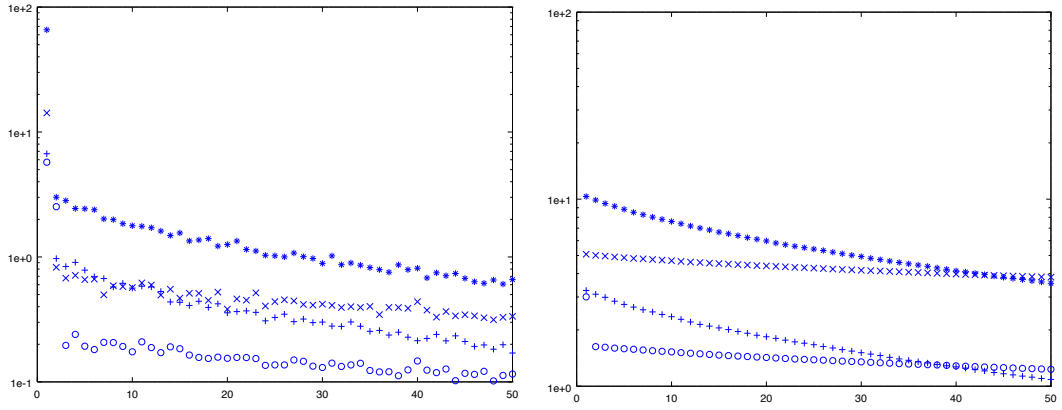

Abbildung 6.2.: Abfallen der Singulär- und Fehlerwerte bei einer Anwendung von Algorithmus 3.1 bei $NH3$-Daten und $\lambda = 0.0001$

Die linke Abbildung zeigt wieder die ersten 50 *Singulärwerte und die rechte die entsprechenden Fehlerwerte* $\|A - \sum_i^k X_i\|_F$ *bzgl. Algorithmus 3.1. Aufgrund der Rechenungenauigkeit kann es zu kleinen Sprüngen bei den Werten kommen. Der Glättungsparameter hat den Wert* 0.0001 *und die Daten entsprechen den folgenden Punkten:* $+$: `NH3_1000A4D`, $\circ$: `NH3_1000A5D`, $*$: `NH3_100004D` *sowie* $\times$: `NH3_100005D`.

ten Teil wird die approximierte Inverse zur Vorkonditionierung des GMRES-Verfahrens verwendet und die zugehörigen Resultate präsentiert. Als Datengrundlage wird der `NH3_1000A4D`-Datensatz verwendet. Mit diesem wird die Matrix des LGS für den Smoothing Spline aufgestellt und die entsprechende Summe von Kronecker-Produkten betrachtet. Die Anzahl der inneren Knoten beträgt wieder 12 (nach [24]), für die Bestimmung der approximierten Inversen wird eine Genauigkeit von 10^{-10} gefordert und es werden fünf Summanden pro Inverse berechnet.

In Tabelle 6.1 sind die Fehler der einzelnen Rechenschritte des Algorithmus 4.2 für die Originaldaten, bezeichnet mit A, zu finden. Es ist deutlich zu erkennen, dass bei stärkerer Gewichtung des Glättungsterms die Fehlerwerte wesentlich schneller abnehmen und diese deutlich geringer sind. Wie später näher erläutert wird, ist in diesen Fällen eine Verwendung als Vorkonditionierer möglich und die Bestimmung von nur fünf Summanden für die Inverse ausreichend. Ein Nachteil bei der Verwendung der Originaldaten ist die lange Dauer der Bestimmung der

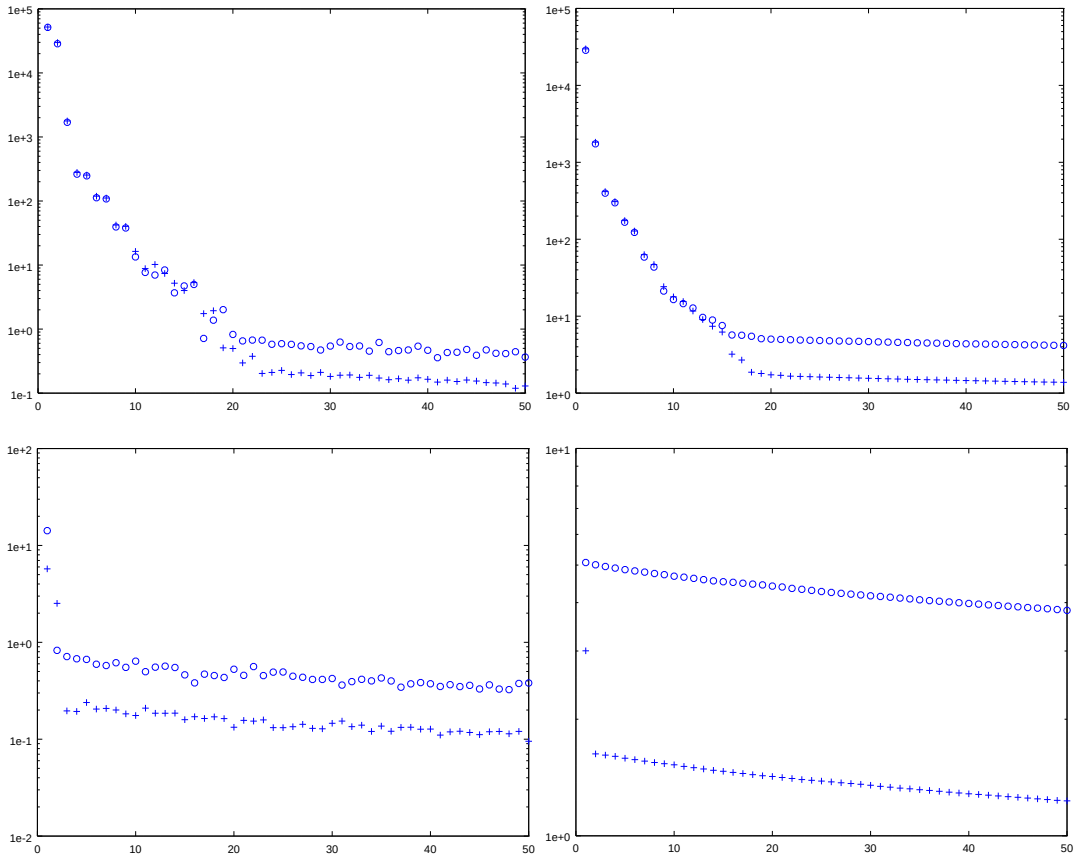

Abbildung 6.3.: Abfallen der Singulär- und Fehlerwerte bei einer Anwendung von Algorithmus 3.1 bei $N2$-Daten

Die oberen beiden Abbildungen zeigen die ersten 50 *Singulärwerte (links) sowie die entsprechenden Fehlerwerte* $\|A - \sum_i^k X_i\|_F$ *(rechts) bzgl. Algorithmus 3.1 für* $\lambda = 1$. *Die unteren beiden Abbildungen zeigen die jeweiligen Werte für* $\lambda = 0.0001$. *Aufgrund der Rechenungenauigkeit kann es zu kleinen Sprüngen bei den Werten kommen. Die Daten entsprechen den folgenden Punkten:* $+$: `N2_1000A5D` *und* $\circ$: `N2_100005D`.

Daten: $NH_3 1000A$ (orig.)						
$\text{err}_k = \|A\sum_i^k X_i - I\|_F$						
λ	1	0.1	0.01	0.001	0.0001	0.00001
err_1	22.3347	21.8059	21.5704	27.7811	37.4053	37.2248
err_2	18.7701	17.9761	20.4448	27.2714	37.2718	37.0945
err_3	16.85	15.9937	17.0914	26.8584	37.1978	36.9074
err_4	16.3183	15.4420	16.7612	26.6023	37.1199	36.8128
err_5	9.7468	9.4408	15.1741	26.2569	37.0515	36.7443

Tabelle 6.1.: Fehlerwerte der approximativen Inversen für Originaldaten

Approximation. Diese benötigt in dem vorliegenden Fall ca. neun Stunden, was nicht effizient ist und eine Verwendung — zum Beispiel als Vorkonditionierer — unmöglich macht. Dies liegt daran, dass die Originaldaten zuviele Summanden haben und somit die Berechnung sehr aufwendig ist.

In Tabelle 6.2 sind die Ergebnisse für die komprimierten Daten, mit $\tilde{A}$ bezeichnet, zu sehen. Dabei wurden für die Komprimierung jeweils die 20 führenden „Singulärwerte“ und -„vektoren“ der zugehörigen Originaldaten verwendet. Diese wurden iterativ mit dem Algorithmus 3.1 berechnet. Es ist auffällig, dass bei stärkerer Gewichtung des Glättungsterms die Fehlerwerte den Werten aus Tabelle 6.1 der Originaldaten ähneln. Im zweiten Teil der Tabelle werden weiterhin die Werte der Norm $\|AX-I\|_F$ bezüglich der Originaldaten A und der berechneten Inversen X für die komprimierten Daten $\tilde{A}$ aufgelistet. Daraus folgt, dass die Inverse bzgl. der komprimierten Daten auch für die Originaldaten in bestimmten Fällen gute Ergebnisse liefert und somit einen adäquaten Ersatz bietet. Wenn die Gewichtung des Glättungsterms abnimmt, kann die Komprimierung zur Bestimmung der Inversen nicht mehr verwendet werden, da die Norm mit jedem weiteren Summanden zunimmt. Dies wird sogar verstärkt, wenn anstelle von 20 Summanden bei der Komprimierung mehr Summanden eingesetzt werden. Ein sehr großer Vorteil bei der Verwendung der Komprimierung ist die Dauer der Berechnung. Diese beträgt in dem betrachteten Fall weniger als eine Minute und dadurch kann diese Variante als Vorkonditionierer für das GMRES-Verfahren in Betracht gezogen werden.

In Tabelle 6.3 sind die Ergebnisse bzgl. der Verwendung der approximativen Inversen als Vorkonditionierer für das GMRES-Verfahren zu finden. Das GMRES-

Daten: $NH_3 1000A$ (kompr.)						
$\text{err}_k = \|\tilde{A} \sum_i^k X_i - I\|_F$						
λ	1	0.1	0.01	0.001	0.0001	0.00001
err_1	22.3325	21.6988	19.2876	24.0960	34.9536	35.0642
err_2	18.7671	17.8227	15.2653	23.4988	34.6132	34.7257
err_3	16.8471	15.8270	13.3920	23.0777	34.4084	34.4639
err_4	16.3154	15.2766	12.8033	22.8455	34.1249	34.2545
err_5	9.7237	8.8167	10.8876	22.4092	33.9647	34.0670
$\text{err}_k = \|A \sum_i^k X_i - I\|_F$						
err_1	22.3346	21.7987	22.1378	31.3139	41.4542	40.7624
err_2	18.7696	17.9416	18.8557	32.0401	41.7309	40.8764
err_3	16.8487	15.9360	17.1739	30.4360	41.7634	41.2060
err_4	16.3174	15.3496	16.4694	30.2507	42.1578	41.2857
err_5	9.7452	9.5257	16.5710	30.3910	42.5084	41.5635

Tabelle 6.2.: Fehlerwerte der approximativen Inversen für komprimierte Daten

Verfahren wurde wie folgt initialisiert: Die Rechentoleranz ist auf 10^{-5} gesetzt, dies ist laut [24] ausreichend. Die Anzahl der Restarts beträgt 5 und die Anzahl der maximalen Iterationen wird beliebig auf 500 festgelegt. Es werden dieselben Daten verwendet und das zum zugehörigen Smoothing Spline entsprechende LGS gelöst. Der Fehler wird durch $\|b - Ax\|_2$ berechnet. An dieser Stelle wird deutlich, wie gut die approximierte Kronecker-Inverse bei stärkerer Gewichtung des Glättungsterms funktioniert. Stellenweise erzielt die approximierte Inverse der Komprimierung sogar bessere Ergebnisse. Dies veranschaulicht eine sinnvolle Verwendung der Komprimierung. Durch sie ist überhaupt erst eine effiziente Nutzung der Kronecker-Inversen möglich, da die Bestimmung der Approximation dann schnell genug ist, um sie als Vorkonditionierer einsetzen zu können.

6.3. SSOR-Vorkonditionierer

An dieser Stelle werden die Resultate für den SSOR-Vorkonditionierer aus Kapitel 5 bzgl. des CG-Verfahren vorgestellt. Die Daten sind aus dem `NH3_1000A4D`-Datensatz und das zu lösende LGS gehört zum entsprechenden Smoothing Spline. Die Rechengenauigkeit für das CG-Verfahren beträgt 10^{-5}, dies ist laut [24]

Daten: $NH_3 1000A$				
λ	kein Vork.		Skal.	
	err	Iter.	err	Iter.
1	3.2138e-04	291	3.1981e-04	274
0.1	2.9499e-04	171	3.0374e-04	158
0.01	2.9471e-04	124	3.1105e-04	107
0.001	3.0449e-04	120	3.1659e-04	90
0.0001	3.2231e-04	198	3.2119e-04	196
0.00001	3.2163e-04	260	3.2167e-04	192
λ	approx. Inv. (or.)		approx. Inv. (komp.)	
	err	Iter.	err	Iter.
1	1.8469e-04	17	3.2211e-04	16
0.1	3.0721e-04	12	1.9150e-04	12
0.01	2.8309e-04	21	2.1341e-04	20
0.001	3.0657e-04	60	3.0409e-04	52
0.0001	3.2317e-04	155	3.2016e-04	176
0.00001	3.2263e-04	340	3.5692e-03	2500

Tabelle 6.3.: Ergebnisse der approximativen Inversen als Vorkonditionierer für das GMRES-Verfahren

für die Bestimmung des Splines ausreichend. Der Fehler wird durch $\|b - Ax\|_2$ berechnet. In Tabelle 6.4 sind die erzielten Ergebnisse abgebildet. Mit dem SSOR-Vorkonditionierer werden zwar in jedem Fall wesentlich weniger Iterationen benötigt, aber die Dauer der Rechenzeit ist viel zu groß, so dass die Verwendung dieses Vorkonditionierers trotz der effizienten Umsetzung aus Kapitel 5 momentan nicht sinnvoll scheint. Aufgrund der zunehmenden Zeilenanzahl verschlechtert sich in höheren Dimensionen sogar die Dauer der Berechnung. Die Ergebnisse sind allerdings abhängig von der Implementierung, welche sicherlich noch effizienter umgesetzt werden kann.

Eventuell kann das LGS des Vorkonditionierers auf eine andere Art als durch Vorwärtselimination und Rücksubstitution effizienter gelöst werden. Interessant ist allerdings, dass die Verwendung der Komprimierung bei entsprechenden Werten des Glättungsparameters wesentlich bessere Resultate erzielt, obwohl diese im Vergleich zur Skalierung zeitlich gesehen auch nicht effizient sind. Wenn der Approximationsterm allerdings zu stark gewichtet wird, dann divergiert das Verfahren bei der Verwendung der Komprimierung als Grundlage für den Vor-

Daten: $NH_3 1000A$						
λ	kein Vork.			Skal.		
	err	Iter.	Zeit [sec]	err	Iter.	Zeit [sec]
1	9.6421e-06	571	9.58	9.8738e-06	493	8.42
0.1	8.5935e-06	334	5.57	9.4932e-06	267	4.62
0.01	9.9815e-06	233	3.89	9.3146e-06	183	3.18
0.001	8.6204e-06	242	4.04	9.4633e-06	201	3.52
0.0001	9.8463e-06	578	9.67	8.9370e-06	394	6.74
0.00001	9.7799e-06	1468	24.71	9.8461e-06	773	13.02
λ	$SSOR_{orig}$			$SSOR_{kompr}$		
	err	Iter.	Zeit [sec]	err	Iter.	Zeit [sec]
1	9.2708e-06	170	750.62	8.8598e-06	171	77.88
0.1	9.1177e-06	81	360.43	7.8006e-06	85	39.8
0.01	9.0213e-06	50	223.25	6.8796e-06	58	27.62
0.001	7.1826e-06	51	239.26	9.3409e-06	86	40.98
0.0001	8.5591e-06	93	415.43	X	X	X
0.00001	9.9258e-06	234	1069.4	X	X	X

Tabelle 6.4.: Ergebnisse des SSOR-Vorkonditionierer mit originalen und komprimierten Daten für CG-Verfahren

konditionierer. Insofern kann der SSOR-Vorkonditionierer zwar theoretisch mit der Kronecker-Struktur umgesetzt werden, bietet aber praktisch noch keinen wirklichen Vorteil oder eine Anwendungsmöglichkeit.

7

Zusammenfassung & Fazit

In dieser Arbeit wurden mehrere Methoden vorgestellt, um Tensorprodukt Smoothing Splines auf gestreuten Daten effizient bearbeiten zu können. Diese Untersuchungen sind sinnvoll, da die Bestimmung eines solchen Splines mit zunehmenden Dimensionen sehr speicherintensiv wird. Ein wesentlicher Baustein bei der Umsetzung der genannten Splines sind Kronecker-Produkte, welche eine zentrale Rolle in der vorliegenden Arbeit einnehmen. Die vorgestellten Verfahren werden zwar teilweise für Tensorprodukt Smoothing Splines spezialisiert, aber sie sind ebenfalls für reine Summen von Kronecker-Produkten anwendbar, welche in einem anderen Zusammenhang — zum Beispiel bei „Stochastic Automata Networks“ — auftreten können.

Nachdem der Leser mit den Grundlagen vertraut gemacht wurde, beschäftigte sich diese Arbeit mit einem Algorithmus, um die auftretenden Summen von Kronecker-Produkten effizient zu komprimieren. Für diesen Algorithmus wurde ein Zusammenhang zwischen Kronecker-Produkten und Multilinearformen aufgezeigt und daraufhin von der zugehörigen Summe von Multilinearformen eine Approximation in 1-Formen bestimmt. Diese Annäherung lässt sich zur Komprimierung der ursprünglichen Summe verwenden. Der angegebene Algorithmus kann im Zusammenhang mit Tensoren unter dem Begriff „Higher Order Power Method“ wiedergefunden werden, so dass der Zusammenhang zwischen Multilinearformen und Tensoren bei dem betrachteten Problem untersucht und die bekannten Verfahren für Tensoren mit dem angegebenen Verfahren aus dieser Arbeit verglichen wurden. Das Verfahren für Multilinearformen ist bereits be-

kannt, nur die Anwendung auf Tensorprodukt Smoothing Splines sowie deren Umsetzung sind neu. Es wurde außerdem hinterfragt, wann eine solche Komprimierung bei dem untersuchten Problem noch effizient ist und welche Informationen sie darüber hinaus bieten kann. Weiterhin wurden Ergebnisse für die Verwendung der Methode bei dem vorliegenden Problem präsentiert. Diese Resultate können in dem Fall überzeugen, wenn der Glättungsterm des Smoothing Splines stärker gewichtet wird. Dadurch lässt sich die Komprimierung gewinnbringend einsetzen und es wurden direkt mögliche Anwendungsfälle aufgezeigt.

Diese Arbeit stellte weiterhin eine Möglichkeit vor, um eine Inverse einer Summe von Kronecker-Produkten zu approximieren. Die Approximation entspricht ebenfalls einer Summe von Kronecker-Produkten und ist aufgrund dessen bei dem untersuchten Problem verwendbar. Es wurde ein Algorithmus aufgestellt, welcher diese Inverse bestimmt, und die Methode wurde für die vorliegende Tensorprodukt Spline Problematik spezialisiert. Die Idee der Methode basiert auf einer Idee des ALS-Konzepts. Abschließend wurde das Verhalten des Verfahrens im Bezug auf Konvergenz und Abklingrate untersucht. Die Arbeit stellte des Weiteren Ergebnisse für die Approximation der Inversen in der behandelten Problemstellung vor und zeigte mit Hilfe von numerischen Tests auf, dass die Komprimierung an dieser Stelle gewinnbringend eingesetzt werden kann. Darüber hinaus wurde eine Verwendung der Inversen als Vorkonditionierer des GMRES-Verfahrens präsentiert. Die Resultate konnten überzeugen, wenn der Glättungsterm stärker als der Approximationsterm gewichtet wird. Insbesondere die Verwendung der Komprimierung ermöglicht dieser Methode einen gewinnbringenden Einsatz in der Praxis. Damit ist die Verwendung ein wichtiger Anwendungsfall für die vorgestellte Komprimierung.

Als letzten Punkt präsentierte die Arbeit zwei mögliche Vorkonditionierer für das CG-Verfahren, wobei der SSOR-Vorkonditionierer genauer untersucht und der Vorkonditionier auf Basis des Prokrustes-Problems nur theoretisch aufgestellt wurde. Es wurde veranschaulicht, wie der SSOR-Vorkonditionerer ohne Verletzung der Kronecker-Struktur angewendet werden kann. Weiterhin wurde eine Struktur des Approximationsterms der Smoothing Splines aufgezeigt, welche für den SSOR-Vorkonditionierer vorteilhaft einsetzbar ist. In praktischen Tests zeigte sich allerdings, dass dieser Vorkonditionierer bisher nicht effizient genug arbeitet, um unter reellen Bedingungen Anwendung finden zu können. Die Tests weisen aber darauf hin, dass die Komprimierung in bestimmten Fällen bei

der Umsetzung von Nutzen sein kann und es durch sie vielleicht eine Möglichkeit gibt, den Vorkonditionierer profitabel umzusetzen.

Abschließend lässt sich sagen, dass durch die vorgestellten Verfahren und Ideen die Bestimmung von Tensorprodukt Smoothing Splines auf gestreuten Daten wesentlich verbessert wurden und diese Verfahren auch unabhängig von den Splines für Summen von Kronecker-Produkten anwendbar sind. Es wurden mehrere Möglichkeiten für potentielle Vorkonditionierer präsentiert, wobei diese nur im Fall der approximierten Kronecker-Inversen praktisch verwendet werden sollten. In den anderen Fällen kann dies eventuell durch weitere Forschung erreicht werden. Die Bestimmung des Glättungsparameters konnte aus Zeitgründen nicht untersucht werden, aber auch in diesem Fall bieten die vorgestellten Verfahren Verbesserungsmöglichkeiten — zum Beispiel ließe sich die Kronecker-Inverse für das Verfahren der „Generalized Cross Validation“ einsetzen.

8 Ausblick

Zum Abschluss dieser Arbeit sollen mögliche Ideen für die weiterführende Forschung vorgestellt werden. Ein zentraler Punkt, welcher bisher nicht ausreichend bearbeitet wurde, ist die effiziente Umsetzung der Bestimmung des Splines bei einer stärkeren Gewichtung des Approximationsterms. In diesem Fall können die vorgestellten Verfahren zwar noch verwendet werden, sind aber nicht mehr von praktischer Relevanz, so dass an dieser Stelle Verbesserungsbedarf besteht. Dies wirkt sich ebenfalls auf die Bestimmung des Glättungsparameters aus, welche bisher nicht effizient implementiert werden konnte. Es wurde lediglich ein heuristischer Schwellwert für den Glättungsparameter vorgestellt, welcher eine Grenze zwischen der Gewichtung des Approximations- und Glättungsterms angibt. Dies wäre ebenfalls ein weiterer Forschungsansatz, den es sich lohnt genauer zu untersuchen. Dafür sollte überprüft werden, inwieweit die Approximation der Inversen bei der „Generalized Cross Validation“ zur Bestimmung des Glättungsparameters eingesetzt werden kann.

Beim SSOR-Vorkonditionierer wäre es sinnvoll nachzuforschen, ob dieser durch bessere Ausnutzung der Strukturen und eine effizientere Implementierung praktisch rentabel gestaltet werden kann. Des Weiteren sollte untersucht werden, ob mit der Verwendung der Kronecker-Inverse und der Komprimierung bessere Rechenzeiten erzielt werden können. Für die praktische Umsetzung wäre eine Weiterentwicklung der Vorkonditionierer für das CG-Verfahren relevant, wobei die bisherigen Ergebnisse durchaus aufgegriffen werden sollten. Dieser Punkt bietet großes Verbesserungspotential für die Bestimmung des Tensorprodukt

Smoothing Splines. Bei der Implementierung sollte im nächsten Schritt versucht werden, die Möglichkeit zur Parallelisierung gewinnbringend zu programmieren. Es gibt sicherlich noch weitere Forschungsansätze und aus den bisher genannten werden sich wahrscheinlich auch neue Problemstellung ergeben, so dass dieses Thema noch ein gewisses Potenzial für die Zukunft bietet.

A Implementierung

Der folgende Abschnitt wird zu Beginn kurz die verwendeten Datenstrukturen erklären. Weiterhin wird die Implementierung der vorgestellten Verfahren erläutert und es wird aufgezeigt, wie diese aufzurufen sind. Der Quellcode ist unter dem Link

`https://subversion.forwiss.uni-passau.de/publications/Dissertation-Lamping`

zu finden. Die Ordnerstruktur ist wie folgt aufgebaut: Die Verfahren werden nach den Kapiteln geordnet und die Ordner sind nach diesen benannt. In den Ordnern für die Komprimierung und die Approximation der Inversen befinden sich neben dem Quellcode auch die berechneten Komprimierungen und Approximationen. Es wird in den Unterabschnitten nur der zum Aufrufen relevante Quellcode erklärt. Auf die Erklärung der Hilfsprogramme wird aus Übersichtsgründen verzichtet. Für weitere Informationen zu den Hilfsprogrammen ist jeweils zu Beginn des entsprechenden `m-files` in den Kommentaren eine kurze Beschreibung der Funktion sowie der Ein- und Ausgabe zu finden.

Wie bereits erwähnt, werden `for`-Schleifen-lastige Programme durch effiziente `oct-files` realisiert. Diese werden mittels der Programmiersprache `C` implementiert. Eine kurze Übersicht zu `oct-files` ist in [12] oder [10] zu finden. Zu jedem `oct-file` existiert ein zugehöriges `m-file`, in welchem Kommentare zur Vorgehensweise zu finden sind. Weiterhin ist zu sagen, dass die `oct-files` mit dem Befehl

```
mkoctfile DATEI.cc
```

für die entsprechende C-Datei kompiliert werden können. Für diesen Befehl muss unter dem Betriebssystem Ubuntu das Paket `liboctave-dev` installiert sein. Es existiert die Datei `makefile`, mit welcher zumindest unter dem Betriebssystem Ubuntu alle `oct-files` in einem Ordner auf einmal kompiliert werden können. Dieses Skript wird mittels `make` aufgerufen. Zur Implementierung werden bekannte Datenstrukturen wie Matrizen, Vektoren und Skalare aus `Octave` benutzt. Allerdings kommen auch zwei weniger bekannte Strukturen zum Einsatz. Dabei handelt es sich um `cell-arrays` und `structs`. Ein `cell-array` entspricht einem mehrdimensionalen Array, in welchem bezüglich Art und Dimension unterschiedliche Variablen gespeichert werden können. Der Umgang mit einem `cell-array` soll durch folgendes Beispiel verdeutlicht werden:

```
c = cell(1,2); % Initialisierung
c = {"a string", [1,2;3,4]}; % Befuellen des cell-arrays
c{1,1}  % Aufrufen der Eintraege
    ans = "a string"
```

Es ist zu beachten, dass mit Matrixvariablen eines `cell-arrays` nur mittels geschweiften Klammern `c{1,2}(1,2)=2` gearbeitet werden kann. Mehr Informationen zu `cell-arrays` sind zum Beispiel in der `Octave`-Dokumentation [12] oder in [9] zu finden. Im Wesentlichen entspricht eine `struct` einem `cell-array`, allerdings werden anstelle von Indizes direkte Bezeichnungen für die einzelnen Variablen verwendet:

```
x.a = 1;
x.b = [1,2;3,4];
x.c = "string";
```

Weitere Information sind ebenfalls in der `Octave`-Dokumentation [12] aufgeführt oder in [11] zu finden.

A.1. Tensorprodukt Smoothing Splines

Die Programme in diesem Abschnitt stammen noch aus der vom Autor erstellten Diplomarbeit [24] und sollen der Vollständigkeit halber vorgestellt werden. Die Routinen bzgl. der univariaten Splines und ihrer Komponenten stammen aus [41].

`TensorProductSmoothSpline:`

Funktion: Die Prozedur bereitet die Daten auf und berechnet damit die univariaten Komponenten eines Smoothing Splines. Anschließend werden diese zum Beispiel an die Funktion `SmoothSplineCoeffCG` übergeben, damit die Koeffizienten des zugehörigen Smoothing Splines bestimmt werden.
Eingabe: *[X,Y,Grad,NKnots,lambda,tol,MinMax]*, dabei entsprechen *X, Y* den Punkten der zu approximierenden Funktion, *Grad* dem Grad des Splines, *NKnots* der Anzahl der inneren Knoten, *lambda* dem Glättungsparameter, *tol* der Fehlertoleranz und *MinMax* entspricht einer Matrix mit den Grenzen der Daten *X*.
Ausgabe: *[Modell]*, zurückgegeben wird eine `struct` *Modell* mit den Koeffizienten des Splines, den Knotenvektoren, dem Splinegrad und der Anzahl der inneren Knoten.

`SmoothSplineCoeffCG:`

Funktion: Die Prozedur berechnet die Koeffzienten des Smoothing Splines mittels dem CG-Verfahren.
Eingabe: *[Vander,Gram,lambda,Ny,nKnots,tol,nX,dimX]*, der Input *Vander* und *Gram* sind `cell-arrays` mit den entprechenden univariaten Matrizen des Smoothing Splines, *lambda* bezeichnet den Glättungsparameter, *Ny* die rechte Seite des LGS, *nKnots* einen Vektor mit der Anzahl der inneren Knoten, *tol* die Fehlertoleranz für das CG-Verfahren, *nX* die Anzahl der Stützstellen und *dimX* die Dimensionsanzahl.
Ausgabe: *[coeff,counter]*, *coeff* bezeichnet die Koeffzienten des Smoothing Splines und *counter* die Anzahl der benötigten Iterationen für die Bestimmung dieser.

`SmoothSplineCoeffPCG:`

Funktion: Die Prozedur berechnet die Koeffzienten des Smoothing Splines mit Hilfe des vorkonditionierten CG-Verfahrens. Die gewünschten Vorkonditionierer müssen jeweils per Kommentar eingeschaltet werden und die benötigten Daten werden direkt geladen.
Eingabe: Analog zu `SmoothSplineCoeffCG`.
Ausgabe: Analog zu `SmoothSplineCoeffCG`.

`UmicoreSmoothSpline*D:`

Funktion: Die Funktion gibt es als 4D- oder 5D-Variante, dafür ist `*` durch 4 oder 5 zu ersetzen. Sie lädt die Umicore-Datensätze, bereitet diese auf und übergibt sie an die Prozedur `TensorProductSmoothSpline`. Sie berechnet mit dem zurückgegebenen Modell des Smoothing Splines die zugehörigen Fehlerwerte.
Eingabe: *[TrainDatei,Grad,AnzKnots,lambda,tol,TestDatei]*, dei Eingaben *TrainDatei* und *TestDatei* enthalten Umicore-Datensätze und *Grad, AnzKnots, lambda, tol* entsprechen den Eingaben der Prozedur `TensorProductSmoothSpline`.
Ausgabe: *[Modell, RelErr]*, zurückgegeben wird eine `struct`-Variable *Modell* mit den Koeffizienten des Splines, den Knotenvektoren, dem Splinegrad und der Anzahl der inneren Knoten sowie eine `struct`-Variable *RelErr* mit den Fehlern für die Trainings- und Testdaten.

`SmoothSplKronSumUmicore*D:`

Funktion: Die Prozedur existiert als 4D- und 5D-Variante, dafür ist `*` durch 4 oder 5 zu ersetzen. Sie berechnet die einzelnen Matrizen der Summe von Kronecker-Produkten aus den Umicore-Daten für den zugehörigen Smoothing Spline.
Eingabe: Es ist keine Eingabe erforderlich. Die benötigten Daten werden intern geladen.
Ausgabe: *[KronSumMat]*, zurückgegeben wird ein `cell-array` mit den Matrizen der Summe von Kronecker-Produkten.

A.2. Iterative Verfahren zum Lösen von LGS

Es sollen die implementierten Prozeduren für die iterativen Verfahren vorgestellt werden. Diese existieren in zwei Varianten. Zum einen gibt es allgemeine Prozeduren für Summen von Kronecker-Produkten und zum anderen speziell auf Tensorprodukt Smoothing Splines zugeschnittene. Es werden nur die allgemeinen Funktionen vorgestellt, bei den speziellen ändert sich nur die Einga-

be. Anstelle eines `cell-arrays` werden zwei `cell-arrays` mit den univariaten Komponenten des Smoothing Splines übergeben.

`kronGMRES:`

Funktion: Die Funktion berechnet iterativ die Lösung x des LGS $Ax = b$ mittels GMRES-Verfahren. Sie ist speziell für Summen von Kronecker-Produkten implementiert.
Eingabe: *[A,b,max_restrt,tol,max_iter]*, A entspricht einem `cell-array` mit den Matrizen der Summe von Kronecker-Produkten, b der rechten Seite des LGS, *max_restrt* der maximalen Anzahl an Neustarts, *tol* der Fehlertoleranz und *max_iter* der maximal erlaubten Anzahl an Iterationen.
Ausgabe: *[x,err,iter_ges]*, x ist der Lösungsvektor, *err* der Fehler der Lösung und *iter_ges* die benötigte Anzahl an Iterationen.

`kronCG:`

Funktion: Die Funktion berechnet iterativ die Lösung x des LGS $Ax = b$ mittels des CG-Verfahrens. Sie ist speziell für Summen von Kronecker-Produkten implementiert.
Eingabe: *[A,b,tol,max_iter]*, A entspricht einem `cell-array` mit den Matrizen der Summe von Kronecker-Produkten, b der rechten Seite des LGS, *tol* der Fehlertoleranz und *max_iter* der maximal erlaubten Anzahl an Iterationen.
Ausgabe: *[x,err,iter]*, analog zu `kronGMRES`.

`kronPrecondCG:`

Funktion: Die Funktion berechnet iterativ die Lösung x des LGS $Ax = b$ mittels vorkonditioniertem CG-Verfahren. Sie ist speziell für Summen von Kronecker-Produkten ausgelegt.
Eingabe: *[A,b,C,tol,max_iter]*, A entspricht einem `cell-array` mit den Matrizen der Summe von Kronecker-Produkten, b der rechten Seite des LGS, C einem `cell-array` mit den Matrizen der Summe von Kronecker-Produkten zur Vorkonditionierung, *tol* der Fehlertoleranz und *max_iter* der maximal erlaubten Anzahl an Iterationen.
Ausgabe: *[x,err,iter]*, analog zu `kronGMRES`.

A.3. Komprimierung

Es werden die zwei wichtigsten Funktionen für die Komprimierung vorgestellt. Sie minimieren mittels ALS die Norm $\|A - X\|_F$. Zu Testzwecken wurden die Verfahren auch als pivotisierende Variante implementiert. Dabei werden immer alle komponentenweisen „Least-squares“ Probleme berechnet und nur das mit der größten Verbesserung ausgewählt. Die Beschreibung der Prozeduren ist analog zum zyklischen Fall. Weiterhin wird noch zwischen Prozeduren unterschieden, die aufgrund der besseren Lesbarkeit auf `oct-files` verzichten und jenen, die `oct-files` verwenden.

`powerMethodMLFkronCyclic:`

Funktion: Die Prozedur bestimmt eine Annäherung an die übergebene Summe von Kronecker-Produkten durch ein Kronecker-Produkt der gleichen Dimension bezüglich der Frobeniusnorm. Sie entspricht der Umsetzung des Algorithmus 3.1.
Eingabe: *[A,Y,tol]*, A ist ein `cell-array` mit den Matrizen einer Summe von Kronecker-Produkten (entspricht MLF), Y ist ein `cell-array` mit Initialisierung der Approximation und *tol* die Toleranz für das Abbruchkriterium.
Ausgabe: *[sigma,X]*, *sigma* ist das σ aus Algorithmus 3.1 und X entspricht den $v_1, \ldots, v_p$ aus diesem.

`kronCompressCyclic:`

Funktion: Die Prozedur bestimmt eine Annäherung an eine Summe von Kronecker-Produkten durch eine Summe von selbigen mit weniger Summanden. Erreicht wird dies durch die mehrfache Anwendung der Prozedur `powerMethodMLFkronCyclic`.
Eingabe: *[A,ranges,tol,s]*, A ist ein `cell-array` mit den Matrizen einer Summe von Kronecker-Produkten (entspricht MLF), *ranges* die Dimension der einzelnen Faktoren der Kronecker-Produkte, s ist die maximale Anzahl von Summanden, welche die Approximation besitzen darf, und *tol* die Toleranz für das Abbruchkriterium.

Ausgabe: *[SIGMA,X]*, *SIGMA* entspricht den σ's bei mehrfacher Anwendung des Algorithmus 3.1 und X ist ein `cell-array` mit den zugehörigen Vektoren bzw. Matrizen.

A.4. Approximation der Kronecker-Inversen

Es wird die Anwendung der Implementierungen der Verfahren aus Kapitel 4 vorgestellt:

`kronInvIter1:`

Funktion: Die Funktion berechnet ein Kronecker-Produkt X als eine approximative Inverse an A mittels Minimierung der Norm $\|AX-I\|_F^2$ durch ein ALS-Verfahren. Die Prozedur ist eine Implementierung des Algorithmus 4.1.
Eingabe: *[A,X,tol,max_iter,n]*, A entspricht einem `cell-array` mit den Matrizen einer Summe von Kronecker-Produkten, X einem `cell-array` mit der Initialisierung der Kronecker-Inverse, *tol* der Fehlertoleranz, die Variable *max_iter* der maximalen Anzahl an Iterationen und n ist ein Vektor mit den Dimensionen der Kronecker-Faktoren.
Ausgabe: *[X,nrm,iter]*, X entspricht einem `cell-array` mit der approximierten Kronecker-Inverse, *nrm* dem Wert der Norm und *iter* der Anzahl von benötigten Iterationen.

`kronInvIterk:`

Funktion: Die Prozedur bestimmt zusätzliche Summanden für die approximative Kronecker-Inverse X an A, welche mittels `kronInvIter1` berechnet wurde. Die Funktion ist eine Implementierung des zweiten Teils des Algorithmus 4.2.
Eingabe: *[A,X,tol,max_iter,n]*, analog zu `kronInvIter1`, wobei X dem Rückgabewert aus `kronInvIter1` entspricht.
Ausgabe: *[X,nrm,iter]*, analog zu `kronInvIter1`.

kronInverse:

Funktion: Die Prozedur berechnet mittels den Funktionen `kronInvIter1` und `kronInvIterk` eine approximative Kronecker-Inverse $X_1 + \cdots + X_s$ an A durch mehrfache Minimierung der Norm $\|AX - I\|_F^2$ mit dem ALS-Verfahren.
Eingabe: *[A,tol,max_iter,max_sum]*, A entspricht einem `cell-array` mit den Matrizen einer Summe von Kronecker-Produkten, *tol* der Fehlertoleranz, *max_iter* der maximalen Anzahl an Iterationen und *max_sum* ist die maximale Summandenanzahl der Approximation.
Ausgabe: *[X,nrm]*, X entspricht einem `cell-array` mit der approximierten Kronecker-Inversen und *nrm* dem Wert der Norm.

A.5. Vorkonditionierer

Es wird die Anwendung bzw. Bestimmung der beiden Vorkonditionierer aus Kapitel 5 erklärt, wobei der Prokrustes-Vorkonditionierer für das CG-Verfahren nur testweise implementiert wurde und noch nicht ausgereift ist.

applySSORPrec:

Funktion: Die Prozedur wendet den SSOR-Vorkonditionierer auf die eingegebenen Daten an, indem das LGS $C^{-1}h = r$ mit $C^{-1} = (L+D)D^{-}1(D+L)^t$ gelöst wird. Dieses wird mittels Vorwärtselimination und Rücksubstitution umgesetzt. Die Funktion ist explizit für Tensorprodukt Smoothing Splines implementiert.
Eingabe: *[r,cellVander,cellGram,lambda,diagVec,nDims]*, r entspricht der rechten Seite des LGS, *cellVander, cellGram* und *lambda* sind die Daten des Smoothing Splines, wobei *cellVander* und *cellGram* `cell-arrays` mit univariaten Komponenten sind, *diagVec* ist die Diagonale der Matrix des Smoothing Splines und *nDims* ist ein Vektor mit den Dimensionen der Kronecker-Faktoren.
Ausgabe: *[h]*, entspricht der Lösung des LGS.

`applySSORPrecKompr:`

Funktion: Die Prozedur berechnet das Gleiche wie `applySSORPrec`, ist aber allgemein für beliebige Summen von Kronecker-Produkten ausgelegt.
Eingabe: *[r,cellMat,diagVec,nDims]*, r entspricht der rechten Seite des LGS, *cellMat* ist ein `cell-array` mit den Matrizen der Summe von Kronecker-Produkten, *diagVec* entspricht der Diagonalen der Matrix C^{-1} und *nDims* ist ein Vektor mit den Dimensionen der Kronecker-Faktoren.
Ausgabe: *[h]*, entspricht der Lösung des LGS.

`calculKronProcPrecCGKron:`

Funktion: Berechnet die Kronecker-Produkte X und Y des Prokrustes-Vorkonditionierers $\|YAX - I\|_F + \omega\|X - Y^t\|_F$ für das CG-Verfahren aus Kapitel 5.2.
Eingabe: *[A,omega,tol,max_iter]*, A entspricht einem `cell-array` mit den Matrizen einer Summe von Kronecker-Produkten, *omega* dem Gewicht des Strafterms $\|X - Y^t\|_F$, *tol* der Fehlertoleranz und *max_iter* beschreibt die maximal erlaubte Anzahl an Iterationen.
Ausgabe: *[Y,X,nrm,iter]*, X und Y entsprechen `cell-arrays` mit Faktoren der Kronecker-Produkte, *nrm* der Summe der beiden Normen und der Ausgabewert *iter* beschreibt die Anzahl an benötigten Iterationen.

Symbolverzeichnis

Symbole	**Beschreibungen**	**Seiten**
$(U_t)_{t\in T_p}$	Basisbaum der hierarchischen Tucker-Zerlegung	88
(X_i, y_i)	Stützstellen und Funktionswerte eines Interpolationsproblems	27
$(k_t)_{t\in T_p}$	Rang-Verteilung bzgl. des Dimensionsbaums T_p	88
$A^\dagger$	Moore-Penrose-Inverse einer Matrix	84
B_t	Transfertensor der hierarchischen Tucker-Zerlegung	89
$G^\mu(T)$	Multivariate Gram-Matrix	36
$G^{\mu_j}_{\kappa_j}(T_j)$	Univariate Gram-Matrix in der j-ten Dimension	37
$N^{\mu_j}(X_j\|T_j)$	Univariate Vandermonde-Matrix eines Splines in der j-ten Dimension	33
$N^\mu(X\|T)$	Multivariate Vandermonde-Matrix für einen Tensorprodukt Spline	33
$N^{\mu_j}_{\kappa_j}(\cdot\|T_j)$	Univariater B-Spline in der j-ten Dimension	30
$N^\mu_\kappa(\cdot\|T)$	Tensorprodukt B-Spline	31
T_j	Univariate Knotenfolge in der j-ten Dimension	29
T_p	Dimensionsbaum der hierarchischen Tucker-Zerlegung	86
$T_{\mu\nu}$	Tensorprodukt Knotenfolge (auch T)	29

$U\Sigma V^t$	Singulärwert-Zerlegung	23
U_t	Basismatrix der hierarchischen Tucker-Zerlegung	88
$W(T_p)$	Wurzel eines Dimensionsbaums der hierarchischen Tucker-Zerlegung	86
$X^\flat$	Multilinearform	49
$\Gamma_{\gamma\eta}$	Menge von Multiindizes	50
α	Multiindex	28
$\chi_{[a,b]}$	Charakteristische Funktion auf $[a, b]$	30
$\circ_H$	Hadamard-Produkt	83
$\circ_j$	Tensor-Matrix Multiplikation in der j-ten Dimension	61
$\circ$	Multilineare Multiplikation eines Tensors	61
$\dot\cup$	Disjunkte Vereinigung	87
$\kappa(A)$	Konditionszahl einer Matrix A	42
$\|A\|_F$	Frobenius-Norm einer Matrix $A \in \mathbb{R}^{m\times n}$	21
$\|H(s_\mu)\|_F^2$	Differentialoperator für Tensorprodukt Smoothing Spline, H bezeichnet die Hesse-Matrix	35
$\langle \mathcal{X}, \mathcal{Y}\rangle$	Inneres Produkt zweier Tensoren	55
$\lceil x \rceil$	Aufrundungsfunktion	57
$\mathbb{N}^p$	p-dimensionale Menge der natürlichen Zahlen	28
$\mathbb{S}^n$	n-dimensionale Einheitssphäre	73
$\mathbb{S}_\mu(T)$	Splineraum der Tensorprodukt Splines vom Grad μ zu der Knotenfolge T	34
$\mathbb{T}^{\gamma\eta}$	$\gamma\eta$-dimensionaler Torus	69
$\mathcal{I}(T_p)$	Innere Knoten des Dimensionsbaums T_p	87

$\mathcal{L}(T_p)$	Blätter des Dimensionsbaums T_p	87
$\mathcal{M}_j(\mathcal{X})$	Matrizication eines Tensors in die j-te Dimension	59
$\mathcal{M}_t^H(\mathcal{X})$	Hierarchische Matrizication eines Tensors	88
$\mathcal{R}(A)$	Operator zur Umstrukturierung einer Matrix $A \in \mathbb{R}^{m\times n}$	21
$\mathcal{V}$	Rang-1 Tensor	53
$\mathcal{X}_H$	Hierarchische Tucker-Zerlegung eines Tensors $\mathcal{X}$	89
$\mathcal{X}$	Tensor	50
$\odot$	Khatri-Rao-Produkt	37
$\otimes_t$	Tensorielles Produkt	52
$\otimes$	Kronecker-Produkt	18
$\psi(\alpha)$	Abbildung eines Multindexes auf einen Spalten-/Zeilenindex	58
$\text{rang}_H((k_t)_{t\in T_p})$	Menge aller Tensoren mit maximalem hierarchischen Rang $(k_t)_{t\in T_p}$	88
$\rho(i,j)$	Abbildung von Zeilen-/Spaltenindizes auf einen Multiindex	57
σ_j	Singulärwert einer Matrix bzw. eines Tensors	23
$\text{rang}_j(\mathcal{X})$	j-Rang eines Tensors	85
$\text{tr}(Q)$	Spur einer Matrix $Q \in \mathbb{R}^{n\times n}$	20
$\varphi(t_\kappa)$	Vielfachheit eines Tensorprodukt-Knotens	34
k_t	Hierarchischer Rang eines Tensors	88
$s_\mu(\cdot)$	Tensorprodukt Spline	31
$vec(A)$	*vec*-Operator bzgl. einer Matrix $A \in \mathbb{R}^{m\times n}$	19
$vec^{-1}(a)$	Inverser *vec*-Operator bzgl. eines Vektors a	19

Literatur

[1] Bellman, R.E.: *Adaptive Control Processes: A Guided Tour*. 'Rand Corporation. Research studies. Princeton: Princeton University Press, 1961.

[2] Benzi, M.: *Preconditioning Techniques for Large Linear Systems: A Survey*. Journal of Computational Physics 182(2) (2002), S. 418–477. DOI: 10.1006/jcph.2002.7176.

[3] Beylkin, G.; Mohlenkamp, M.J.: *Numerical operator calculus in higher dimensions*. Proceedings of the Nat. Acad. Sci. USA 99(16) (2002), S. 10246–10251. DOI: 10.1073/pnas.112329799.

[4] Bojanczyk, A.; Lutoborski, A.: *The Procrustes Problem for Orthogonal Kronecker Products*. SIAM Journal on Scientific Computing 25(1) (2003), S. 148–163. DOI: 10.1137/S1064827501396464.

[5] de Boor, C.: *A practical guide to splines; rev. ed.* Applied mathematical sciences. Berlin: Springer, 2001.

[6] de Boor, C.: *Efficient Computer Manipulation of Tensor Products*. ACM Transactions on Mathematical Software 5(2) (1979), S. 173–182. DOI: 10.1145/355826.355831.

[7] Burkard, Rainer E.; Zimmermann, Uwe: *Einführung in die Mathematische Optimierung*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-28673-5.

[8] Carroll, J.D.; Chang, J.J.: *Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckhart-Young" Decomposition*. Psychometrika 35(3) (1970).

[9] Eaton, J.W.: *Cell arrays in Octave*. 2011 (zuletzt besucht April 2015). URL: http://www.gnu.org/software/octave/doc/interpreter/Cell-Arrays.html#Cell-Arrays.

[10] Eaton, J.W.: *Oct-files in Octave*. 2011 (zuletzt besucht April 2015). URL: www.gnu.org/software/octave/doc/interpreter/Oct_002dFiles.html.

[11] Eaton, J.W.: *Structures in Octave*. 2011 (zuletzt besucht April 2015). URL: www.gnu.org/software/octave/doc/interpreter/Structures.html#Structures.

[12] Eaton, J.W. u. a.: *GNU Octave*. Boston: Free Software Foundation, 2011.

[13] Farin, G.: *Curves and Surfaces for CAGD: A Practical Guide*. San Diego: Academic Press, 2002.

[14] Fischer, G.: *Lineare Algebra*. Wiesbaden: Springer Fachmedien, 2014. DOI: 10.1007/978-3-658-03945-5.

[15] Golub, G. H.; van Loan, C. F.: *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[16] Grasedyck, L.: *Hierarchical singular value decomposition of tensors*. SIAM Journal on Matrix Analysis and Applications 31(4) (2010), S. 2029–2054. DOI: 10.1137/090764189.

[17] Grasedyck, L.; Hackbusch, W.: *An introduction to hierarchical (H-)rank and TT-rank of tensors with examples*. Computational methods in applied mathematics 11(3) (2011), S. 291–304. DOI: 10.2478/cmam-2011-0016.

[18] Hackbusch, W.: *Tensor spaces and numerical tensor calculus*. New York: Springer, 2012. DOI: 10.1007/978-3-642-28027-6.

[19] Harshman, R.A.: *Foundations of the PARAFAC procedure: models and conditions for an "explanatory" multimodal factor analysis*. UCLA Working Papers in Phonetics 16(10) (1970), S. 1–84.

[20] Higham, N.J.: *The symmetric Procrustes problem*. BIT Numerical Mathematics 28(1) (1988), S. 133–143. DOI: 10.1007/BF01934701.

[21] Kanzow, C.: *Numerik linearer Gleichungssysteme und iterative Verfahren*. Heidelberg: Springer, 2005. DOI: 10.1007/b138019.

[22] Kolda, T.G.: *Orthogonal Tensor Decompositions*. SIAM Journal on Matrix Analysis and Applications 23(1) (2001), S. 243–255. DOI: 10.1137/S0895479800368354.

[23] Kolda, T.G.; Bader, B.W.: *Tensor decompositions and applications*. SIAM review 51(3) (2009), S. 455–500. DOI: 10.1137/07070111X.

[24] Lamping, F.: *Wirklich multivariate Smoothing Splines und eine Anwendung in der Chemie*. Diplomarbeit. Gießen, 2011.

[25] Lamping, F.; Peña, J.M.; Sauer, T.: *Spline Approximation, Kronecker Products and Multilinear Forms*. Numerical Linear Algebra with Applications, accepted for publication (2015).

[26] Langville, A. N.; Stewart, W. J.: *A Kronecker product approximate preconditioner for SANs*. Numerical Linear Algebra with Applications 11(8-9) (2004), S. 723–752. DOI: 10.1002/nla.344.

[27] Langville, A.N.; Stewart, W.J.: *Testing the Nearest Kronecker Product Preconditioner on Markov Chains and Stochastic Automata Networks*. INFORMS Journal on Computing 16(3) (2004), S. 300–315. DOI: 10.1287/ijoc.1030.0041.

[28] Langville, A.N.; Stewart, W.J.: *The Kronecker product and stochastic automata networks*. Journal of computational and applied Mathematics 167(2) (2004), S. 429–447. DOI: 10.1016/j.cam.2003.10.010.

[29] de Lathauwer, L.; de Moor, B.; Vandewalle, J.: *A multilinear singular value decomposition*. SIAM Journal on Matrix Analysis and Applications 21(4) (2000), S. 1253–1278. DOI: 10.1137/S0895479896305696.

[30] de Lathauwer, L.; de Moor, B.; Vandewalle, J.: *On the best rank-1 and rank-(R_1,...,R_n) approximation of higher-order tensors*. SIAM Journal on Matrix Analysis and Applications 21(4) (2000), S. 1324–1342. DOI: 10.1137/S0895479898346995.

[31] Leibovici, D.; Sabatier, R.: *A singular value decomposition of a k-way array for a principal component analysis of multiway data, PTA-k*. Linear Algebra and its Applications 269(1-3) (1998), S. 307–329. DOI: 10.1016/S0024-3795(97)81516-9.

[32] van Loan, C.F.; Pitsianis, N.: *Approximation with Kronecker Products*. Linear Algebra for Large Scale and Real- Time Applications 232 (1993), S. 293–314. DOI: 10.1007/978-94-015-8196-7_17.

[33] van Loan, C.F.: *The ubiquitous Kronecker product*. Journal of Computational and Applied Mathematics 123(1-2) (2000), S. 85–100. DOI: 10.1016/S0377-0427(00)00393-9.

[34] Mallat, S.: *A wavelet tour of signal processing*. 2. ed., Academic Press, 1999.

[35] Meister, A.: *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren*. Wiesbaden: Vieweg+Teuber Verlag, 2015. DOI: 10.1007/978-3-658-07200-1.

[36] Mohlenkamp, M.J.: *Musings on multilinear fitting*. Linear Algebra and its Applications 438(2) (2013), S. 834–852. DOI: 10.1016/j.laa.2011.04.019.

[37] Nagy, J. G.; Kilmer, M. E.: *Kronecker product approximation for preconditioning in three-dimensional imaging applications.*. IEEE Transactions on Image Processing 15(3) (2006), S. 604–613. DOI: 10.1109/TIP.2005.863112.

[38] Oseledets, I.V.: *Tensor-train decomposition*. SIAM Journal on Scientific Computing 33(5) (2011), S. 2295–2317. DOI: 10.1137/090752286.

[39] Sauer, T.: *Numerische Mathematik II*. Vorlesungsskript. Justus-Liebig Universität Gießen, 2000.

[40] Sauer, T.: *Persönlicher Austausch mit Herrn Prof. Dr. Sauer*. (2011).

[41] Sauer, T.: *Splinekurven und -flächen in Theorie und Anwendung*. Vorlesungsskript. Justus-Liebig Universität Gießen, 2008.

[42] Schönemann, P. H.: *A generalized solution of the orthogonal procrustes problem*. Psychometrika 31(1) (1966), S. 1–10. DOI: 10.1007/BF02289451.

[43] Steeb, W.H.: *Kronecker product of matrices and applications*. Mannheim, Wien, Zürich: BI-Wiss.-Ver., 1991.

[44] Votsmeier, M. u. a.: *Simulation of automotive NH3 oxidation catalysts based on pre-computed rate data from mechanistic surface kinetics*. Catalysis Today 151(3-4) (2010), S. 271–277. DOI: 10.1016/j.cattod.2010.01.018.

[45] Wallner, N.: *Vorkonditionierer für Matrizen in Kronecker-Form*. Examensarbeit. Justus-Liebig Universität Gießen, 2011.

Erklärung

Ich erkläre: Ich habe die vorgelegte Dissertation selbständig und ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt, die ich in der Dissertation angegeben habe.

Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, und alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht.

Bei den von mir durchgeführten und in der Dissertation erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis, wie sie in der „Satzung der Justus-Liebig-Universität Gießen zur Sicherung guter wissenschaftlicher Praxis“ niedergelegt sind, eingehalten.

Gießen, 04. Mai 2015

Frank Lamping