Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 81

Interaktive Visualisierung von hochdimensionalen Funktionen

René Schwarz

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dirk Pflüger
Betreuer/in:	M. Sc. Fabian Franzelin
Beginn am:	17. Juni 2013
Beendet am:	17. Dezember 2013
CR-Nummer:	D.2.2, G.4

Kurzfassung

Der Fluch der Dimensionalität stellt bei der numerischen Lösung komplexer und damit zumeist hochdimensionaler Abläufe ein großes Hindernis dar. Dünne Gitter bieten eine Möglichkeit, um diesem Problem entgegen zu wirken, die Visualisierung der Ergebnisse bedurfte bisher jedoch einiges an Mehraufwand.

In dieser Bachelorarbeit werden verschiedene Möglichkeiten untersucht, mit denen sich hochdimensionale Funktionen interaktiv darstellen lassen. Gesucht sind dabei Darstellungen, welche die Untersuchung von Parameterabhängigkeiten, Unstetigkeiten und der Koeffizienten Dünner Gitter weitestgehend vereinfachen, um einen gutes Verhältnis zwischen Entwicklung, Analyse und Anpassung der Gitter zu erreichen. Auf den Erkenntnissen aufbauend wird anschließend ein Programm mit einer Grafischen Benutzeroberfläche als Feature zu SG^{++} entwickelt, welches mit wenig Aufwand aussagekräftige Diagramme erstellt und Werkzeuge bietet, um in Echtzeit die Daten soweit aufzubereiten, dass Auffälligkeiten einfach erkennbar sind. Zum Schluss wird der Funktionsumfang des Tools anhand von Beispielen aus der Forschung und analytischen Mathematik demonstriert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1 Einleitung

Mit steigender Rechenleistung moderner Computer in den letzten Jahrzehnten ist auch der Anspruch an die Beschreibung komplexer Abläufe gewachsen. Bei der Bewertung des Börsenkurses oder der Modellierung von Proteinen, aber auch in der Physik, Geologie oder Medizin entstehen zunehmend komplexe Modelle mit vielen unbekannte Parameter. Die exponentielle Zunahme des Rechenaufwands mit der damit verbundenen wachsenden Dimensionalität stellt bei der numerischen Lösung solcher Probleme jedoch ein großes Hindernis dar.

Eine Möglichkeit dieses Größenproblem zu lösen ist die Verwendung von Dünnen Gittern, welche wir im nächsten Kapitel betrachten werden. Besonders in den letzten Jahren wurde hier viel Aufwand betrieben Methoden zu entwickeln, mit denen auch hochdimensionale Probleme effizient gelöst werden können. Unter anderem entstand dabei eine räumlich adaptive Methode, zu deren Nutzung eine Toolbox namens SG^{++} zusammengestellt wurde [Pfl10] und auf die diese Arbeit aufbaut.

Neben der Berechenbarkeit stellt aber auch die Visualisierung großer Datensätze mit vielen Dimensionen ein erhebliches Problem dar. Im Bereich des Data Mining wurden deshalb in den vergangenen Jahrzehnten verschiedene Ansätze zumeist aus der Statistik aufgegriffen und auf ihren Nutzen hin getestet. Im 3. Kapitel werden wir einige dieser Darstellungen vorstellen und ihre Anwendbarkeit auf Dünne Gitter prüfen.

Ziel dieser Arbeit war es die vielversprechendsten Darstellungen aufzugreifen und für Dünne Gitter einfach nutzbar zu machen. Einfach nutzbar machen heißt heutzutage, dass keine lange Einarbeitungszeit und so wenig Klicks wie möglich nötig sind. Um diesen Anspruch gerechtzuwerden haben wir ein auf Qt aufbauendes grafisch gestütztes Visualisierungs-Tool entwickelt, mit dem wir mit SG^{++} erstellte Gitter in ansprechenden Grafiken erstellen und damit interaktiv arbeiten können. Auf den Aufbau des Tools und den Funktionsumfang gehen wir in Kapitel 4 ein. Außerdem gehen wir in diesem Kapitel noch auf die Echtzeit-Lauffähigkeit, die ein notwendiges Kriterium für ein angenehmes interaktives Arbeiten ist, ein.

In Kapitel 5 zeigen wir anhand einiger Klassifikations- und analytischer Beispiele mit verschiedenen Dimensionalitäten auf, wie sich Dünne Gitter mit dem Tool darstellen lassen, und wie die einzelnen Darstellungen bei der Analyse der Gitter nützlich sein können. Am Ende fassen wir die Resultate dieser Arbeit noch einmal zusammen und geben einen kurzen Ausblick auf mögliche anknüpfende Projekte.

2 Dünne Gitter mit SG^{++}

Die Idee ein dünn besetztes Gitter für mathematische Berechnungen zu verwenden, hatte erstmals der russischen Mathematiker Smolyak [Smo63], als er 1963 Gitter für die numerische Integration benutzte. In den 1990ern, als neue Verfahren zur Lösung komplexer Probleme mit mehr als drei bis vier Dimensionen gesucht wurden, hat man Smolyaks Ansatz wieder aufgegriffen und zunächst für die Lösung von partiellen Differenzialgleichungen entwickelt, später folgte dann auch die Anwendung auf Integralgleichungen, Interpolationen und Approximationen [BG04] [Gar11].

Dieses Kapitel stellt in Anlehnung an [Pfl10] die wesentlichen Grundlagen der Dünnen Gitter vor, erklärt am Beispiel der Interpolation. Wir werden zu Beginn die hierarchische Basis nur im eindimensionalen Fall betrachten, bevor wir den d dimensionalen Raum abdecken. Danach befassen wir uns mit der Randbetrachtung und der Adaptivität von Dünnen Gittern, da sie für die spätere Visualisierung wichtig sind. Für eine ausführlichere Erklärung sei hier auf [BG04] verwiesen. Zum Schluss dieses Kapitels folgt eine kurze Übersicht über das SG^{++} Projekt, welches alle nötigen Werkzeuge bereitstellt, um Dünne Gitter verwenden zu können.

2.1 Vorüberlegungen

Wir werden im Folgenden wie auch SG^{++} mit einem normalisierten Merkmalsraum $\Omega = [0,1]^d$ arbeiten, nicht nur um die Konsistenz zu wahren, sondern auch, weil dadurch die Betrachtung erleichtert wird. Dies grenzt die Verwendbarkeit des Tools in keinster Weise ein, da abweichende Definitionsbereiche später ohne großen Aufwand umgerechnet werden können. Außerdem setzen wir die Randwerte zunächst auf null, um die Einführung nicht unnötig zu erschweren. Auf die Berücksichtigung des Randes und den sich damit ergebenden Schwierigkeiten werden wir im Anschluss eingehen. Schlussendlich ist es für die betrachtete Interpolation als Beispiel noch wichtig zu definieren, dass wir stets davon ausgehen, dass sich Werte der Ausgangsfunktion an beliebigen Punkten berechnen lassen oder ausreichend vorliegen.

2.2 Hierarchische Basis

Bereits der antike griechische Mathematiker Archimedes hat bei seinen Überlegungen zur Volumen- und Flächenberechnung herausgefunden, dass sich die Fläche einer Parabel

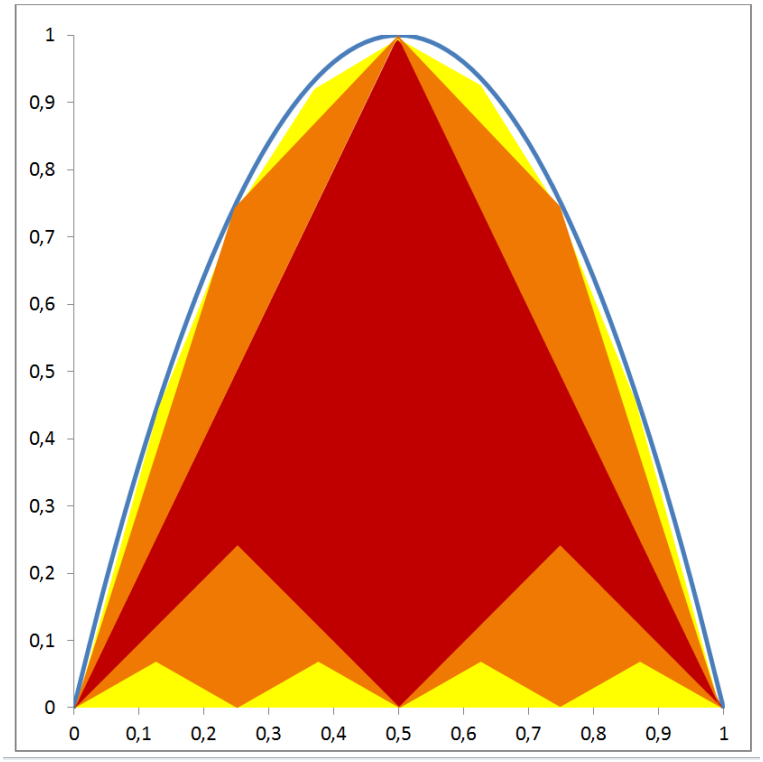

Abbildung 2.1: Quadratur der Parabel nach Archimedes

mithilfe von sich aufsummierenden Dreiecken wie in Abbildung 2.1 gut approximieren lässt [Pflo5].

Diese „Quadratur der Parabel“ läuft wie folgt ab: Ausgehend von einem Basisdreieck mit dem Maximum der Parabel als Spitze und den Randpunkten als Basispunkte werden so lange um den Faktor vier kleiner werdende Dreiecke mit ihrer Basis auf die Seitenflächen der davor eingefügten Dreiecke gesetzt, bis die Parabelfläche ausreichend genau approximiert ist.

Diesen hierarchischen Ansatz können wir nun für unser Verfahren problemlos dahin gehend erweitern, dass er allgemein gültig für jede zu approximierende Funktion wird. Anstelle der aufsummierenden Dreiecke verwenden wir als Basisfunktion die Standard-Hutfunktion,

$$\varphi(x) = \begin{cases} 1 - |x|, & \text{falls } x \in [-1, 1] \\ 0, & \text{sonst} \end{cases} \tag{2.1}$$

welche wir mittels Stauchung und Translation in Abhängigkeit vom Level l und Index i mit $0 < i < 2^l$ definieren als

$$\varphi_{l,i}(x) = \varphi(x \cdot 2^l - i). \tag{2.2}$$

Aus dieser Menge an sich ergebenden Funktionen für jedes Level l benötigen wir für nun für unsere Approximation nur die Hutfunktionen mit ungeradem Index i, welche jeweils im Gitterpunkt $x_{l,i} = 2^{-l} \cdot i$ lokalisiert sind, da an den Stellen mit geraden Indizes bereits eine Hutfunktion niedrigeren Levels lokalisiert ist. Je nach gewähltem maximalen Level $n \in \mathbb{R}$ erhalten wir unterschiedlich viele Gitterpunkte mit einer Maschenweite $h_n = 2^{-l}$.

Unsere hierarchische Basis lässt sich somit mit dem Index-Set

$$I_l := \{i \in \mathbb{N} \wedge 1 \leq i \leq 2^l - 1 \wedge i \text{ ungerade}\} \tag{2.3}$$

mithilfe der hierarchischen Teilräumen W_l,

$$W_l := \text{span } \{\varphi_{l,i}(x) : i \in I_l\} \tag{2.4}$$

für die Level $l \leq n$ als Summe der Teilräume definieren zu dem Raum V_n,

$$V_n := \bigoplus_{l \leq n} W_l. \tag{2.5}$$

Der Funktionswert $u(x)$ für einen Gitterpunkt $x_{l,i}$ ergibt sich damit aus der Summe der mit einem hierarchischen Überschuss $\alpha_{l,i}$ gewichteten Basisfunktionen,

$$u(x) = \sum_{l \leq n, i \in I_l} \alpha_{l,i} \varphi_{l,i}(x). \tag{2.6}$$

2.3 Hierarchische Basisfunktionen im d dimensionalen Raum

Nachdem wir den eindimensionalen Fall kennengelernt haben können wir nun einfach für den mehrdimensionalen Raum unsere vorherige Basis und alle hinzu gehörigen Notationen erweitern. Dazu werden l und i als Multi-Indizes mit $\vec{l} = (l_1, \ldots, l_d)$ und $\vec{i} = (i_1, \ldots, i_d) \in \mathbb{N}^d$ definiert und damit das Tensor-Produkt der Basisfunktionen gebildet,

$$\varphi_{\vec{l},\vec{i}}(\vec{x}) = \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j). \tag{2.7}$$

Für das erweiterte Index-Set $I_{\vec{l}}$,

$$I_{\vec{l}} := \{\vec{i} : 1 \leq i_j \leq 2^{l_j} - 1 \wedge i_j \text{ ungerade} \wedge 1 \leq j \leq d\} \tag{2.8}$$

sind die Teilräume $W_{\vec{l}}$ dann definiert als

$$W_{\vec{l}} := \text{ span } \{\varphi_{\vec{l},\vec{i}}(\vec{x}) : \vec{i} \in I_{\vec{l}}\}. \tag{2.9}$$

Der Raum V_n für ein Level n eines vollbesetzten Gitters mit einer Maschenweite h_n und insgesamt $(2^n - 1)^d$ Gitterpunkten ergibt sich wieder aus der Summe der Teilräume $W_{\vec{l}}$,

$$V_n := \sum_{l_1=1}^{n} \cdots \sum_{l_d=1}^{n} W(l_1, \ldots l_d) = \bigoplus_{|\vec{l}|_\infty \leq n} W_{\vec{l}}. \tag{2.10}$$

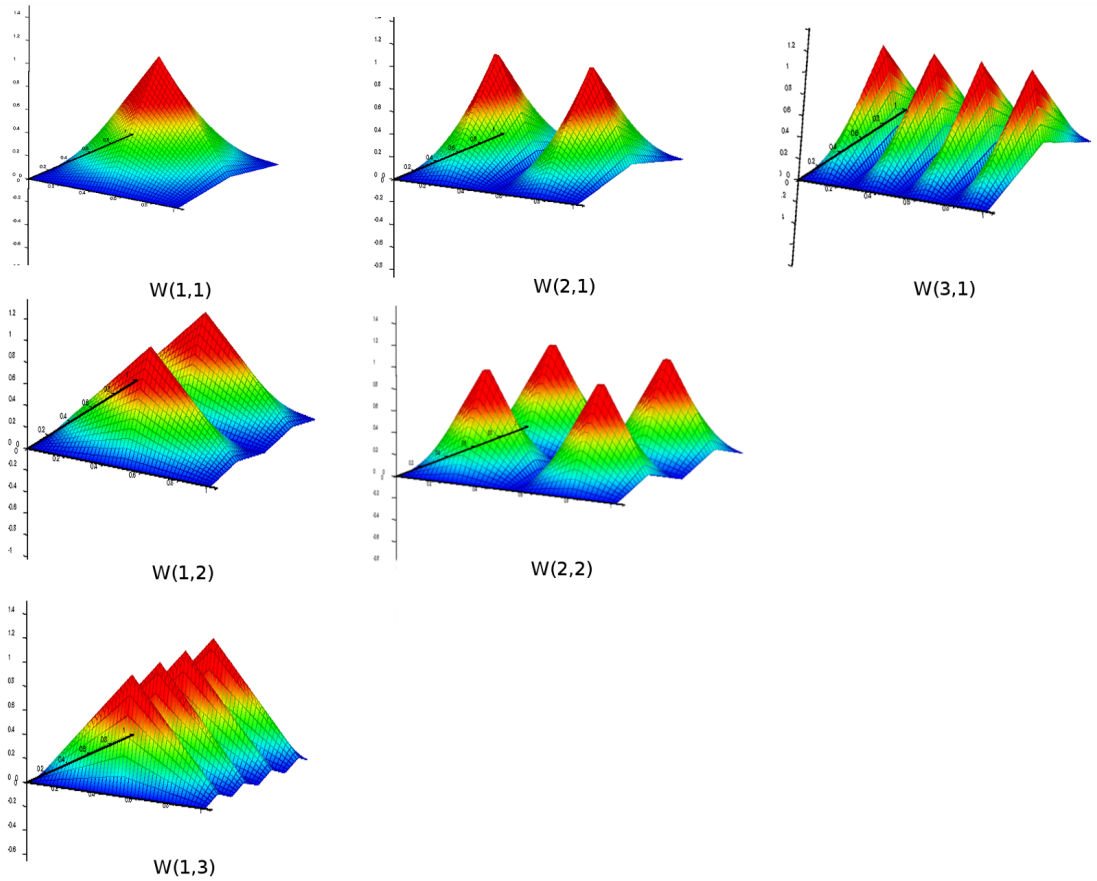

Abbildung 2.2: Dünn Gitter Basisfunktionen für d = 2 und Level l = 3 [Pfl10]

Mit diesem Gitter ist es nun möglich, eine Funktion beliebiger Dimensionalität d mithilfe stückweise d-lineare Basisfunktionen für n Level zu interpolieren. Für andere Anwendungen wurden neben den Hutfunktionen aber noch weitere Basisfunktionen in SG^{++} implementiert, von denen hauptsächlich die d-polynomialen Basisfunktionen, welche höher geordnete Interpolationen erlauben, und für den Finanz Sektor,für den spezielle Anforderungen durch das Gitter erfüllt sein müssen, außerdem noch Wavelet-Basis und B-Spline Basisfunktionen verwendet werden. Für eine nähere Beschreibung dieser Basisfunktionen sei auf [Pfl10] verwiesen, da für diese Ausarbeitung die Betrachtung der linearen Hutfunktionen ausreichend ist.

2.4 Dünne Gitter

Alle bisherigen Überlegungen zielten auf ein normales, vollbesetztes Gitter mit einer Größe von $|V_n| = (2^n - 1)^d$ Gitterpunkten ab. Damit wird klar, dass auch Gitter nicht von der

d	$\lvert V_n^{0B(1)}\rvert$	$\lvert V_n^{(1)}\rvert$	$\lvert V_n\rvert$
1	33	31	31
2	385	129	961
3	3,809	351	29,791
4	35,585	769	923,521
5	324,033	1,471	28,629,151
6	2,912,257	2,561	887,503,681
7	25,986,369	4,159	27,512,614,111

Tabelle 2.1: Vergleich der Anzahl an Gitterpunkten verschiedener Dünner Gitter und einem Vollen Gitter ohne Randbetrachtung für verschiedene Dimensionen und Level n = 5 [Pfl10]. Die Verbesserung der Dünnen Gitter zum Vollbesetzten ist selbst bei schlecht gewählter Randapproximation erheblich.

extremen Wachstumsrate bei steigender Dimensionalität verschont bleiben, Richard Bellman bezeichnete diesen Zusammenhang treffend als „Fluch der Dimensionalität“ [Bel57]. Beispielsweise müssen bereits bei einer Verfeinerung mit Level n = 5 und einer Dimensionalität d = 5 mehr als 28 Millionen Gitterpunkte für ein volles Gitter berechnet werden. Mit Hinblick auf die Performance und die praktische Nutzbarkeit von Gittern für hochdimensionale Funktionen ist daher unumgänglich, in einem nächsten Schritt zu versuchen die Anzahl der Gitterpunkte weitestgehend zu reduzieren und dabei eine hohe Approximations-Genauigkeit zu erhalten. Eigentlich handelt es sich bei dieser Aufgabe um ein klassisches „Rucksackproblem“, dass bekanntlich NP hart ist. Bei unseren Gittern lässt sich aber der Umstand zeigen, dass der Beitrag der Teilräume $W_{\vec{l}}$ zur gesuchten Funktion mit steigender Level-Summe $|\vec{l}|_1$ abnimmt [BG04]. Es bietet sich daher an, die Summe der Teilräume wie in Abbildung 2.3 veranschaulicht ab einen bestimmten Wert abzuschneiden, um so ein dünner besetztes Gitter zu erhalten, dass ähnlich genaue Resultate wie das volle Gitter liefert.

Diesen Grenzwert für die verwendeten Teilräume definieren wir für die Summe der Levels über den Dimensionen als

$$|\vec{l}|_1 \leq n + d - 1. \tag{2.11}$$

Damit lässt sich jetzt unser Dünngitterraum V_n^1 definieren,

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}. \tag{2.12}$$

Die Anzahl der Gitterpunkte kann mit diesem Ansatz auf $(2^n \cdot n^{d-1})$ Punkte reduziert werden, ohne dabei einen allzu großen Verlust an Genauigkeit gegenüber eines vollbesetzten Gitters hinnehmen zu müssen. Für den Beweis sei hierzu auf [BG04] verwiesen.

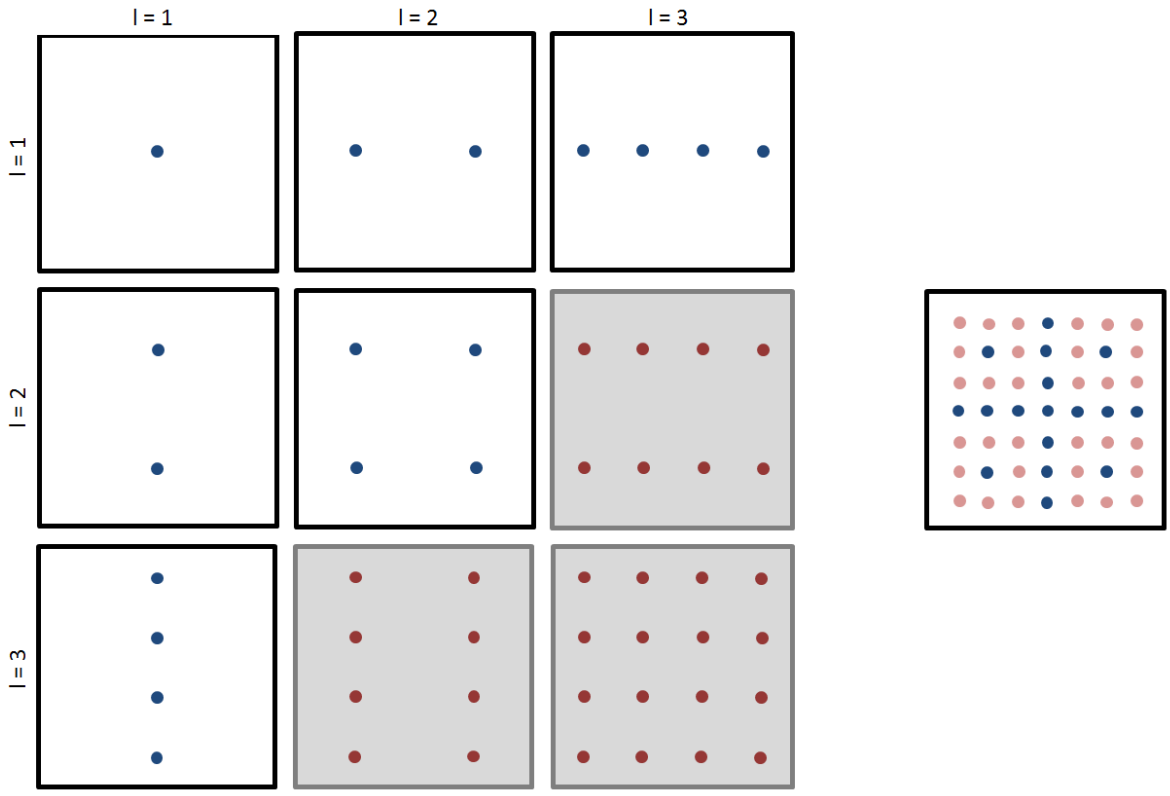

Abbildung 2.3: Teilräume für Dimension d = 2 und Level l = 3. Die ausgegrauten Räume werden für das Dünne Gitter eingespart. Ganz rechts ist der Unterschied zwischen einem vollen und einem dünnem Gitter gut zu erkennen [Pfl10]

2.5 Randbetrachtung

Nachdem wir bisher den Rand stets null gesetzt hatten, wollen wir nun versuchen auch diesen zu interpolieren, ohne dabei den Vorteil eines dünnen Gitters, mit wenigen Einträgen aus zu kommen, zu verlieren. Im eindimensionalen Fall bietet es sich an, unter die Funktion zunächst ein Trapez mit den Randpunkten als obere Eckpunkte zu setzen, auf dem dann die hierarchische Basis aufgesetzt wird. Das Trapez können wir mithilfe zweier zusätzlicher Funktionen,welche auf die Randpunkte zentriert werden, für ein neues Level $l = 0$ ersetzen. Für den eindimensionalen Fall sind das

$$\varphi_{0,0}(x) := \begin{cases} 1 - x & \text{falls } x \in [0,1] \\ 0 & \text{sonst} \end{cases} \tag{2.13}$$

und

$$\varphi_{0,1}(x) := \begin{cases} x & \text{falls } x \in [0,1] \\ 0 & \text{sonst} \end{cases} \tag{2.14}$$

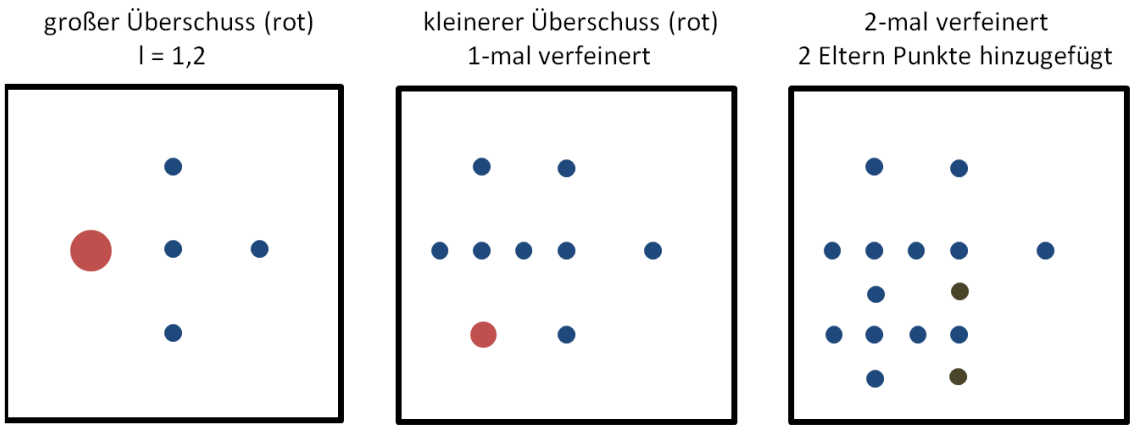

Abbildung 2.4: Gitter, bei dem sowohl am Rand als auch innerhalb verfeinert wurde [Pfl10]

Der Vorteil hierbei ist, dass sich an den innen liegenden Gitterpunkten nichts ändert. Für den mehrdimensionalen Fall bedeutet das Hinzufügen von Basisfunktionen aber einen erheblichen Zuwachs an Gitterpunkten, womit sich wieder das ursprüngliche Problem der Unbrauchbarkeit der Gitter für hohe Dimensionen ergibt. Tabelle 2.1 verdeutlicht dies anschaulich.

Um diesem Problem entgegen zu wirken gibt es Möglichkeiten die Randpunkte zu reduzieren. Zum einen, indem man die Basisfunktionen von Level 0 auf Level 1 legt und dadurch für jeden inneren Punkt genau einen Punkt zu jedem Rand in jeder Dimension erhält. Zum anderen, dass man anstelle von Randpunkten bestehende Basisfunktionen zum Rand hin extrapoliert. Für eine genauere Beschreibung sei auf [Pfl10] verwiesen.

2.6 Adaptivität

Unser Dünnes Gitter ist jetzt in der Lage Funktionen mit hoher Dimensionalität mit einem beliebigen Grad an Verfeinerung zu approximieren, sowohl mit als auch ohne Berücksichtigung der Randpunkte. Das Gitter wird dabei für jedes Level immer gleichmäßig verfeinert. Wenn aber nun beispielsweise eine Funktion vorliegt, die mit einem Level l schon ausreichend gut approximiert ist und nur in einem bestimmten Bereich noch ein größerer Fehler vorliegt, so wäre es unvorteilhaft, für diesen einen Abschnitt das gesamte Gitter um ein Level zu erweitern.

Eine Besonderheit, die die Verwendung von hierarchischen Basen zur Approximation mit sich bringt, ist die Möglichkeit eine Funktion nicht nur gleichmäßig zu verfeinern, sondern auch einzelne Gitterpunkte anhand lokaler Fehlerabschätzungen zur spezifischen Verfeinerung einfügen zu können. Als Fehlerindikator können wir einfach die Überschüsse der Gitterpunkte nutzen.

Eine Adaption kann üblicherweise bei dem Gitterpunkt mit dem größten Überschuss begonnen werden. Für diesen Gitterpunkt werden entweder alle oder nur eine bestimmte Auswahl an Kindern berechnet und dem Gitter hinzugefügt. Elternpunkte, die für die Kinder benötigt werden und noch nicht teil des Gitters sind, sollten auch dem Gitter hinzugefügt werden, um die Baumstruktur, welche notwendig für die Anwendung rekursiver Algorithmen ist, aufrecht zu erhalten. Anschließend wird dieser Schritt so oft für den jeweils neuen größten Überschuss wiederholt, bis der gewählte Bereich ausreichend genau approximiert ist. Abbildung 2.4 veranschaulicht die Adaption eines Gitters in einem einfachen zweidimensionalen Fall.

2.7 Das Tool SG^{++}

SG^{++} ist ein Open Source Werkzeug, das dazu entwickelt wurde, den Einsatz räumlich adaptiver Dünner Gitter möglichst einfach und Effizient sowohl in C++, als auch in Python, Java und Matlab zu ermöglichen, ohne dass umfangreiche Programmierungen von Klassen und Algorithmen nötig sind (`http://www5.in.tum.de/SGpp/releases/index.html`). Die Toolbox bietet dabei unter anderem folgende Funktionalitäten:

- Verschiedene Basisfunktionen mit und ohne Randbetrachtung
- Multi-Core Unterstützung via OpenMP
- einfache Adaptive Verfeinerung der Gitter
- gute Erweiterbarkeit

Auf der Website finden sich umfangreiche Hinweise, wie das Tool genutzt werden kann und welche Anforderungen zur Ausführung erfüllt sein müssen. Innerhalb dieser Ausarbeitung wurde das Tool ausschließlich in der C++ Variante verwendet, über den Export der Alpha- und Grid-Daten lassen sich aber alle mittels SG^{++} erzeugten Gitter sprachen übergreifend zugänglich machen.

3 Visualisierung von hochdimensionalen Funktionen

Nachdem wir im vorherigen Kapitel die Grundlagen zur Berechnung hochdimensionaler Funktionen mit Dünnen Gittern besprochen haben, stellt dieses Kapitel die bekanntesten Methoden zur Visualisierung solcher Daten vor. Dabei werden neben den typischerweise verwendeten Streudiagrammen, Funktions- und Image-Graphen (oder „Heatmap“) auch neuere Ansätze wie Parallelkoordinaten,Bildsymbole und Projektionstechniken untersucht, welche zumeist aus dem Bereich Data-Mining stammen.

Am Ende dieses Kapitel fassen wir die Erkenntnisse über die verschiedenen Darstellungen zusammen und prüfen, welche der Darstellungen sich auf Funktionen anwenden lassen und wie Dünne Gitter dabei helfen können weitere Darstellungen für das im nächsten Schritt zu erstellende Tool in Betracht zu ziehen.

3.1 Exploration hochdimensionaler Daten

Hochdimensionale Daten konfrontieren den Betrachter oft mit dem Problem, dass schon bei drei Dimensionen Zusammenhänge zwischen den Daten von subjektiven Eindrücken beeinflusst werden oder sogar gänzlich unerkennbar bleiben. Bereits in den 70ern haben sich deshalb Analysten damit beschäftigt, Methoden zu entwickeln, die solche Datensätze aufbereiten um objektive Ergebnisse gewinnen zu können. John W. Tukey, der als Vorreiter dieser Methoden gilt, prägte dafür die Bezeichnung „Explorative Datenanalyse“ [Tuk77]. Darunter verstand er nicht nur Visualisierungstechniken, sondern die Anwendung verschiedenster Rechentechniken, Analysen und einfachen Visualisierungen der Ergebnisse. Erst in den 80ern rückten die Visualisierungen dank der wachsenden Leistungsstärke von Computern in den Fokus der Analysten. Der Computer soll dabei nicht den Analysten ersetzen, sondern vielmehr die Daten so aufbereiten, dass das Potential des geschulten menschlichen Auges optimal ausgenutzt werden kann, um sämtliche Auffälligkeiten in den Daten aufdecken zu können. Im Folgenden werden wir einige vielversprechende Explorations-Methodiken betrachten, die in den letzten 40 Jahren entwickelt wurden. Für die Analyse multivariater Daten existieren auch noch eine Reihe weiterer Darstellungen, die eher auf die grafische Umsetzung großer Datensätze und weniger auf eine hohe Dimensionalität ausgerichtet sind. Deshalb verzichten wir hier auf eine ausführlichere Betrachtung und verweisen auf [Deg06] [Cha06] [BFC98] [Sch08], die sich ausführlich mit der Visualisierung hochdimensionaler Daten beschäftigen.

3.2 Streudiagramme

Das Streudiagramm ist die wohl bekannteste und einfachste Form zur Darstellung zweidimensionale Datensätze. Bei Funktionen spricht man auch vom Funktionsgraphen. An ihm lassen sich direkt alle Zusammenhänge und Ungewöhnlichkeiten von zwei Merkmalen erkennen. Möchte wir den Datenpunkten zusätzliche Eigenschaften zuweisen, so kann dies durch Einfärbung oder variierender Punktdarstellung geschehen. So kann zum Beispiel ein eingefärbter Kreis, dessen Radius und Farbe abhängig von zwei Eigenschaften ist, anstelle eines Punktes verwendet werden, um das zweidimensionale Diagramm auf vier Merkmale auszuweiten. Daneben können Punktmengen beispielsweise auch mittels Ellipsen gruppiert werden, was uns dabei hilft die Daten schneller zuordnen zu können. Beide Techniken hängen jedoch stark davon ab, wie wir die Eigenschaften gewichten lassen, weshalb sich subjektiven Ergebnisse nicht vermeiden lassen.

Natürlich gibt es neben den Streudiagrammen noch eine Vielzahl anderer zweidimensionaler Darstellungsvarianten wie dem Box-Plot, dem Balkendiagramm oder dem Q-Plot, doch für hochdimensionale Funktionen sind diese eher ungeeignet.

3.2.1 3D Diagramme

3D Streudiagramme stellen die nächsthöhere Ebene dar um mehr als zwei Merkmale in einem Diagramm darzustellen. Die Anordnung der Merkmale und der Blickwinkel auf das Diagramm haben aber einen starken Einfluss darauf, wie wir die Daten und ihre Zusammengehörigkeiten wahrnehmen oder ob Überlagerungen wie in Abbildungen 3.1 die Sicht auf einen Teil der Punktmenge versperren. Deshalb sind 3D Diagramme oft nur bedingt hilfreich und eher zur Präsentation geeignet. Dank leistungsfähiger Grafikkarten ist es aber auch möglich dynamische Diagramme zu erstellen, die sich in Echtzeit beliebig rotieren und verändern lassen. So ist es uns möglich auch aus dreidimensionalen Streudiagrammen hilfreiche Erkenntnisse zu gewinnen. Eine bewiesen vollständige Möglichkeit die Daten in einem 3D Diagramm unter Berücksichtigung aller Winkel und Kombinationen begutachten zu können ist die kleine Reise 3.5.2.

Eine größere Ausnutzung des dreidimensionalen Raumes kann mit „Worlds within worlds“ erreicht werden, eine hierarchische Technik, bei der ein dreidimensionaler Raum in mehrere Subräume unterteilt wird, um so mehrere Dimensionen auf einmal in einer 3D Ansicht anzeigen zu können [BF93]. In Abbildung 3.2 etwa ist eine Funktion $f(x_1, x_2, x_3, x_4, x_5)$ zu sehen, bei der die Variablen x_1 und x_2 sowie die Funktionswerte in einem normalen 3D-Funktionsgraphen abgebildet werden. Für die übrigen drei Variablen, welche bei der Berechnung des Graphen konstant gehalten werden, wird nun ein weiterer Graph mit drei Achsen als „äußere Welt“ erstellt, in den der Funktionsgraph als „innere Welt“ eingebettet wird. Mit der Position, in der die beiden Welten zueinander stehen, können dann die Konstanten beeinflusst werden. Die Erzeugung solcher Welten in Welten erfordert leider einen großen Rechenaufwand und führt bei mehreren Dimensionen zu sehr komplexen Darstellungen.

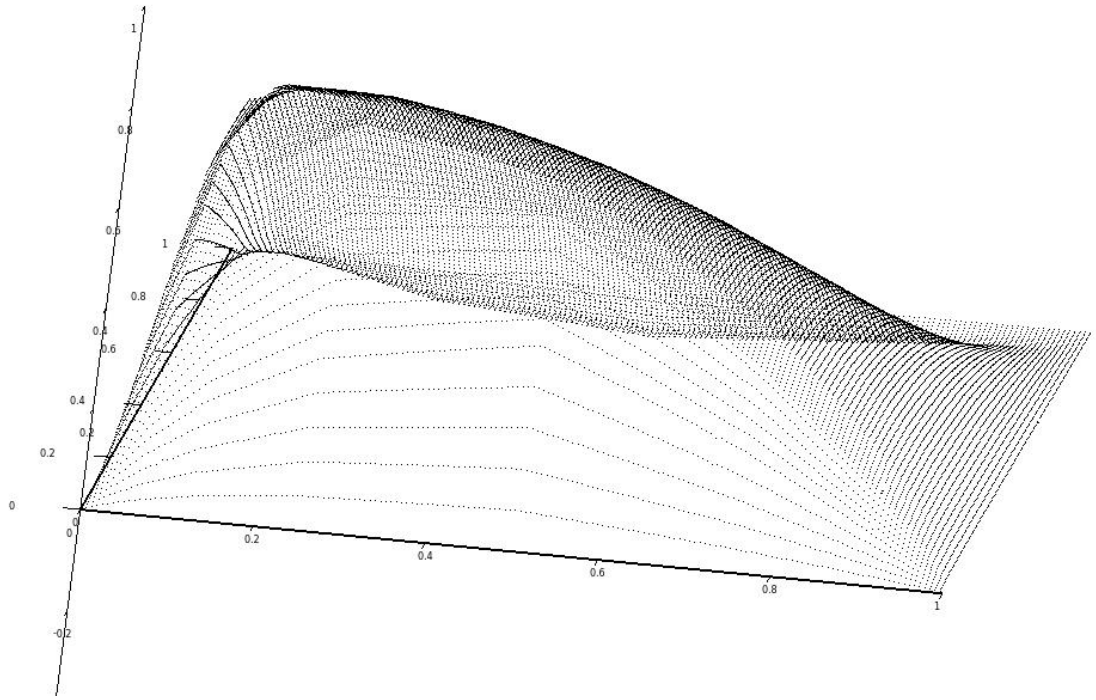

Abbildung 3.1: Dreidimensionaler Funktionsgraph als Punktmenge dargestellt. Die Überlagerung der Punkte aus dem gewählten Blickwinkel ist deutlich erkennbar.

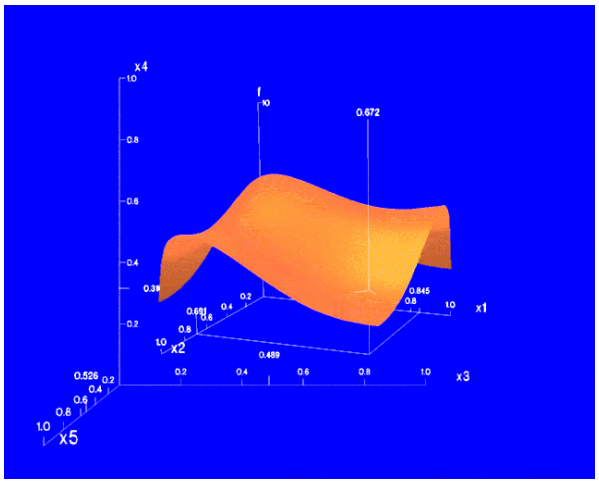

Abbildung 3.2: n-Vision einer fünfdimensionalen Funktion [BF93]

3.2.2 Höhenliniendiagramme

Gerade bei Funktionsgraphen, bei denen ein Merkmal immer von einem oder mehren Merkmalen abhängt, bietet es sich an, ein Kontur- beziehungsweise Höhenliniendiagramm zu erstellen. Hierbei werden äquidistant liegende Punkte vordefinierter Höhen zu einer Linie verbunden und wahlweise für ein 2D Diagramm auf eine Ebene projiziert. Anstelle der Höhenlinien können aber auch Farbabstufungen verwendet werden. Dieser eingefärbte Graph, den wir als Image-Plot oder Heatmap bezeichnen, ist gut zur Datenanalyse geeignet, da unser Sehen auf die Farbwahrnehmung ausgerichtet ist und wir deshalb Veränderungen In den Verläufen besonders schnell identifizieren können.

3.2.3 Streudiagramm-Matrix

Ab einer Dimensionalität größer drei reicht das normale Streudiagramm jedoch nicht mehr aus und es können lediglich Schnittebenen dargestellt werden. Deshalb hat man die Streudiagramm-Matrix eingeführt, in der alle möglichen Merkmals-Paarungen als Schnittebene in einem Hyperwürfel dargestellt werden. Bei einer Dimensionalität von sechs ergibt das eine 6 X 6 Matrix. Theoretisch könnte man auch drei statt zwei Merkmale visualisieren, allerdings ist die Wahrscheinlichkeit groß, dass viele Auffälligkeiten aufgrund der Komplexität der Darstellung verloren gingen. Da im unteren und oberen Dreieck die selben Paarungen entstehen, verwenden wir üblicherweise nur die Felder im unteren Dreieck als Streudiagramme und nutzen das obere Dreieck wie in Abbildung 3.3 um die Merkmale in einer weiteren Darstellung aufzubereiten. Auch die Paarungen auf der Diagonalen können wir aufgrund ihrer trivialen Daten einsparen und sie stattdessen beispielsweise für textuelle Beschreibungen verwenden. Doch auch mit der Streudiagramm-Matrix lassen sich Funktionen mit $d > 10$ nur bedingt darstellen, da die einzelnen Diagramme schnell sehr klein werden und die Matrix insgesamt unübersichtlich erscheint. Außerdem lassen sich mit den Streudiagrammen immer nur jeweils zwei Merkmale mit einander vergleichen und alle anderen als konstant zu betrachten.

Streudiagramme sind bei niedriger Dimensionalität gut dazu geeignet, sich einen Überblick über die Daten zu verschaffen und Auffälligkeiten zwischen 2 - 3 Dimensionen zu analysieren. Farben und Punktgrößen können hier unterstützend wirken. Streudiagramm-Matrizen erlauben zudem eine Ausweitung des Verwendungsbereiches auf etwa zehn Dimensionen, darüber hinaus findet sich an den Diagrammen aber nur wenig praktischer Nutzen.

3.3 Profil-Diagramme

Aufgrund der eingeschränkten Dimensionalität der Streudiagramm Matrizen musste eine andere Lösung für hohe Dimensionalitäten gefunden werden. Dabei entstand die Idee, statt immer nur zwei Merkmale in einer Darstellung zu präsentieren, alle Merkmale auf einmal in einem einzigen zweidimensionalen Diagramm als „Profile“ darzustellen. Eine sehr einfache Form der Profil Diagramme sind beispielsweise die von D. Daetz vorgeschlagenen Polygon

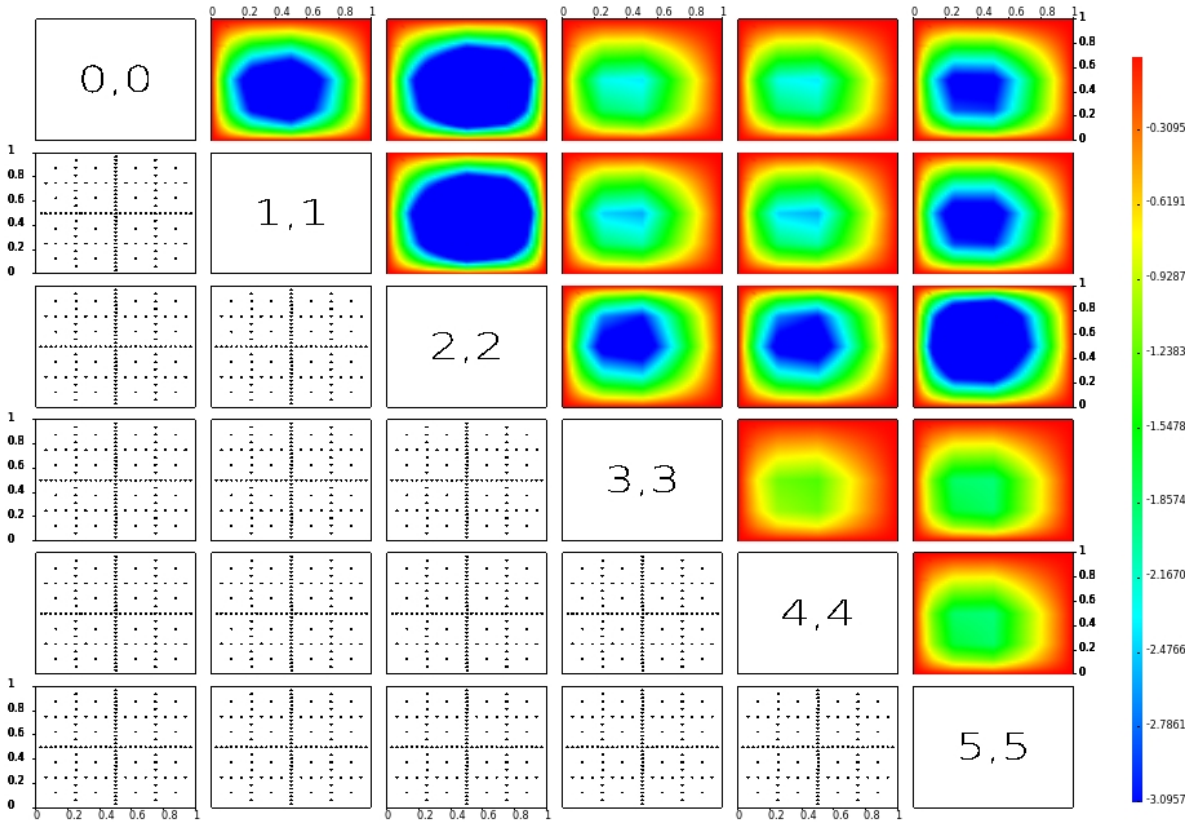

Abbildung 3.3: Beispiel einer Streudiagramm-Matrix mit Dimensionalität d = 6. Als 2. Darstellung wurde eine Heatmap Variante für das obere Dreieck verwendet

Profile, in der Literatur auch als „Kreisförmige Parallelkoordinaten“ bezeichnet. Zunächst wird dafür ein Kreis gezeichnet, dessen Radius das Maximum oder den Durchschnitt der möglichen Werte darstellt. Anschließend werden in den Kreis d Achsen vom Mittelpunkt zum Rand als Radien verlaufend eingezeichnet. Auf jedem dieser Strahlen tragen wir dann die Werte eines Datenpunktes für jedes Merkmal vom Kreismittelpunkt aus beginnend als Punkt ab und verbinden sie anschließend zu einem Polygon. Anhand des Polygons wie in Abbildung 3.4 können wir nun leicht das Profil eines Datenpunktes über alle Dimensionen ablesen. Für mehrere Datenpunkte können wir entweder jedes Mal ein neues Polygon Diagramm erstellen, oder aber wir tragen alle Datenpunkte in ein Diagramm ein um so leicht Unterschiede zwischen den einzelnen Polygonen erkennbar zu machen. Problematisch wird es jedoch bei großen Datensätzen, bei denen durch Überlagerungen der einzelnen Polygone das Diagramm unübersichtlich wird und einzelne Datensätze in der Masse verloren gehen. Diesem Nachteil unterliegen alle Vertreter der Profildiagramme, auch die weitaus bekannteren Andrews-Kurven (3.3.1) und die Parallele Koordinaten (3.3.2).

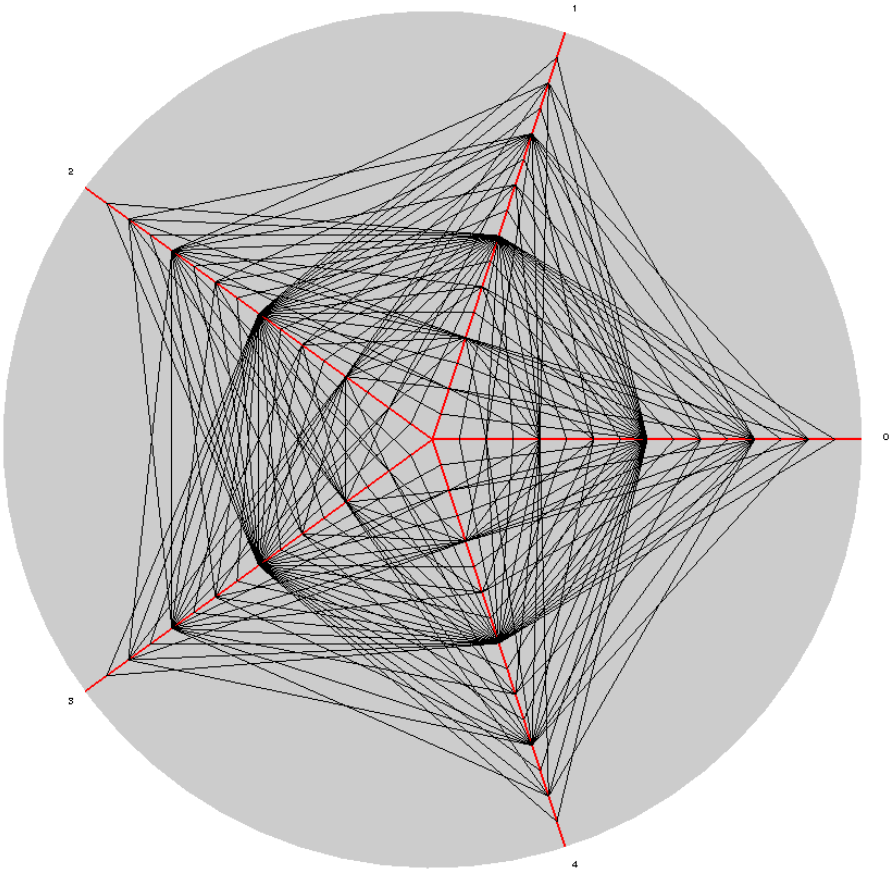

Abbildung 3.4: Polygon Profil nach Daetz

3.3.1 Andrews-Kurven

Andrews verfolgte die Überlegung, hochdimensionale Daten nicht länger direkt in ein zwei- oder dreidimensionales Diagramm einzutragen, sondern mithilfe trigonometrischer Funktionen jeden Datenpunkt in einen mathematisch einfach analysierbare zweidimensionale Kurve zu überführen [And72]. Dazu entwickelte er für einen k dimensionalen Datenpunkt $x_k = (x_1, \ldots, x_k)$ die Funktion

$$f_x(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + \ldots, \tag{3.1}$$

welche anschließend im Bereich $-\pi < t < \pi$ gezeichnet wird, was ausreicht, da sich die Funktion außerhalb dieses Bereichs aufgrund ihres trigonometrischen Charakters wiederholt. Eine Kurve bildet damit alle Merkmale eines Punktes ab, gewichtet nach der Reihenfolge, in der sie verwendet wurden. Ähnliche Datenpunkte erzeugen ähnliche Kurven, die Distanzen

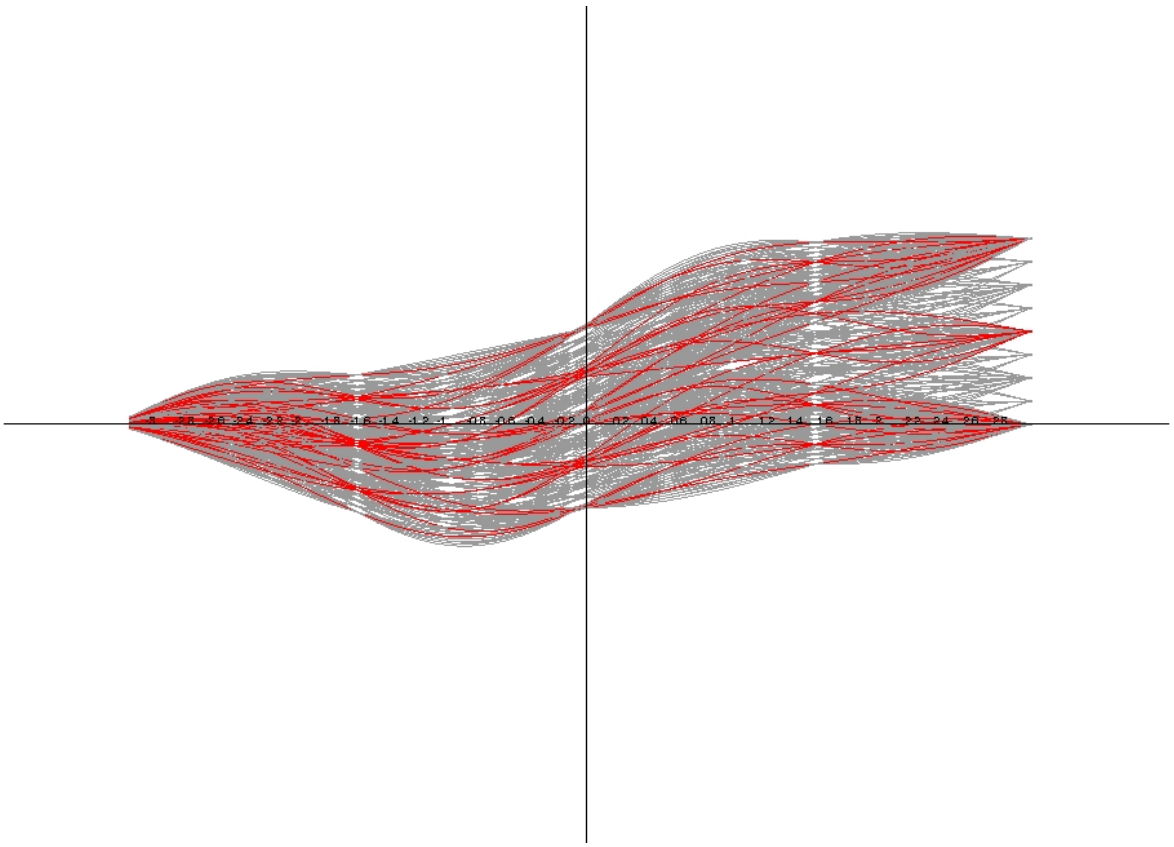

Abbildung 3.5: Andrews-Kurven eines fünfdimensionalen Datensatzes der Größe n 2000. Zur besseren Visualisierung einer kleinen Gruppe wurde Brushing (siehe 4.3.4) verwendet.

zwischen den Kurven sind proportional zum euklidischen Abstand der Datenpunkte. Damit lassen sich zusammenhängende Daten und Ausreißer an Andrews Kurven schnell erkennen. Gerade aber die Abhängigkeit der Kurven von der Reihenfolge der Merkmale und der damit verbunden unterschiedlichen Gewichtung bedeutet einen Nachteil bei der Betrachtung und man ist gezwungen verschiedene Sortierungen auszuprobieren. Und wie bereits erwähnt verliert die Darstellung mit steigender Kurvenzahl an Übersichtlichkeit, mal abgesehen von Gruppierungen ähnlichen Daten, solange der Datensatz einen gewisse Größe nicht überschreitet. Vermutlich wegen ihres mathematischen Charakters, aber auch aufgrund der unterschiedlichen Gewichtung werden Andrews-Kurven zwar oft zitiert, finden praktisch aber nur wenig Anwendung.

3.3.2 Parallele Koordinaten

Die Parallele Koordinaten Darstellung (PCP - parallel coordinates plot), welche erstmals von Maurice d'Ocagne 1885 beschrieben und seit 1978 von Alfred Inselberg [ID90] weiterentwi-

ckelt wurde, ist ähnlich den Andrews Kurven ein Versuch hochdimensionale Daten in einem zweidimensionalen Graphen zu visualisieren. Für eine umfassende exemplarische Analyse mit PCP sei hier auf [Weg90] verwiesen. Zur Erstellung eines Diagramms zeichnen wir die Dimensionen zunächst als parallele Linien zu einer der beiden Achsen, je nach dem ob das Diagramm horizontal oder vertikal ausgerichtet ist, im gleichen Abstand zu einander ein. Anschließend tragen wir jeden Datenpunkt als Linie ab, die durch alle Dimensions-Achsen verläuft. Der Linienverlauf bietet so einen direkten Aufschluss über die Merkmale eines Datenpunktes, zudem können wir die Streuung des Datensatzes über jede Dimension sofort ablesen, ähnliche Datenpunkte sind anhand gleich verlaufender Linien und negative Zusammenhänge an kreuzenden Linien erkennbar. Außerdem lässt sich das Diagramm leicht um eine oder mehrere Dimensionen erweitern, indem einfach weitere Achsen am Ende hinzugefügt werden. Bei der Analyse mithilfe der Parallelen Koordinaten hat sich gezeigt, dass sich die Reihenfolge der Dimensionen nicht unerheblich auf das Ergebnis auswirken kann. Die Skalierung für alle Achsen sollte zwar einheitlich gewählt sein, aber auch unterschiedliche Skalen können interessante Aufschlüsse ergeben.

Im Gegensatz zu den Andrews Kurven haben wir bei den Parallelen Koordinaten nicht das Problem der absteigend gewichteten Dimensionen, weshalb die Grenze für darstellbare Dimensionen rein abhängig von der Größe der Anzeige ist und deshalb theoretisch beliebig hoch sein kann. Die Größe der Datensätze wirkt sich zwar auch hier wieder auf die Übersichtlichkeit aus, muss aber nicht zwangsläufig zu schlechten Diagrammen führen. Gerade Gruppierungen lassen sich mithilfe einiger Techniken, etwa Selektieren,Ausblenden von Linienzügen oder Brushing [HLD02], gut erkennbar machen. In [FWR99] wird zudem beschrieben, wie mithilfe von hierarchischem Clustering auch Datensätze > 100.000 ansprechend visualisiert werden können. Parallele Koordinaten stellen somit vielleicht den besten Ansatz dar um einfach aber wirkungsvoll hochdimensionale Daten auf zusammenhänge, Anomalien und mutlivariates Verhalten hin zu analysieren.

3.4 Bildsymbole

1973 schlug Hermann Chernoff vor, die Charakteristiken eines menschlichen Gesichts - etwa Augenposition, Nasengröße und Haaransatz 3.7- zur Darstellung von hochdimensionalen Daten zu verwenden [Che73]. Die Chernoff-Gesichter bieten so die Möglichkeit insgesamt 18 Dimensionen darzustellen. Weil ein symmetrisches Gesicht zu redundanten Darstellungen führt und praktisch eine Gesichtshälfte eingespart werden könnte, schlugen Bernhard Flury und Hans Riedwyl vor, asymmetrische Gesichter zu verwenden und so Chernoffs Ansatz auf 36 Dimensionen zu erweitern [FR81]. Diese unkonventionelle Visualisierungsmethode, die später noch um einige andere Varianten erweitert wurde, bietet dank unserer Fähigkeit kleinste Unterschiede in Gesichtszügen zu erkennen ein großes Potential für die Darstellung hochdimensionaler Daten.

Bisher haben die Chernoff Gesichter aber nur wenig praktische Verwendung gefunden. Denn zum einen ist die Zuordnung und Interpretation von Merkmalen höchst subjektiv, sodass bereits die Darstellung stark variiert, je nach dem welche Gewichtung wir den Merkmalen,

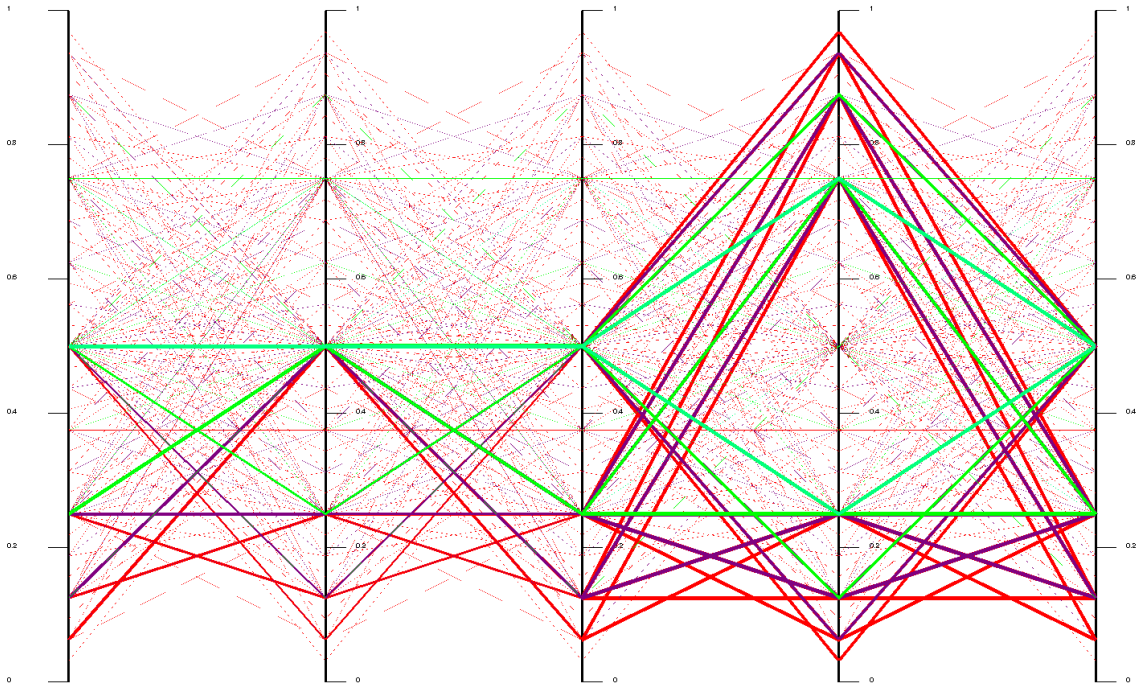

Abbildung 3.6: Parallelkoordinaten eines Datensatzes der Größe $p > 2000$ mit Dimension $d = 5$ und eingefärbten Levels $l = 4$. Die Dicke der Linien visualisiert die absolute Größe der Überschüsse (siehe 4.3.4)

aber auch den einzelnen Gesichtszügen zuschreiben. Besonders kulturelle Unterschiede können hier zu vollständig unterschiedlichen Ergebnissen führen. Zum anderen fällt auch die Erstellung von Gesichts-Diagrammen recht komplex aus, weil jedes Merkmal an einen Gesichtszug angepasst werden muss, und bedarf entweder viel Handarbeit bei der Anfertigung von Zeichnungen, oder aufwendiger grafisch unterstützter Modellierungswerkzeuge. Auch andere Ansätze leiden an diesen Schwierigkeiten, sodass Bildsymbole in Theorie zwar viel Potential zugesprochen bekommen, aber praktisch wohl nie wirklich Anwendung finden werden.

3.5 Projektionstechniken

Bisher haben wir hauptsächlich Methoden betrachtet, die versuchen den gesamten Datensatz direkt darzustellen. Wie in Abbildung 3.5 zu sehen ist kommen diese Darstellungen jedoch bei großen Datensätzen trotz unterstützender Techniken schnell an ihre Grenzen, wodurch eigentlich wichtige Auffälligkeiten in der Masse verloren gehen. Es stellt sich also die Frage, ob es überhaupt hilfreich ist alle Merkmale eines Datensatzes ungefiltert zu visualisieren

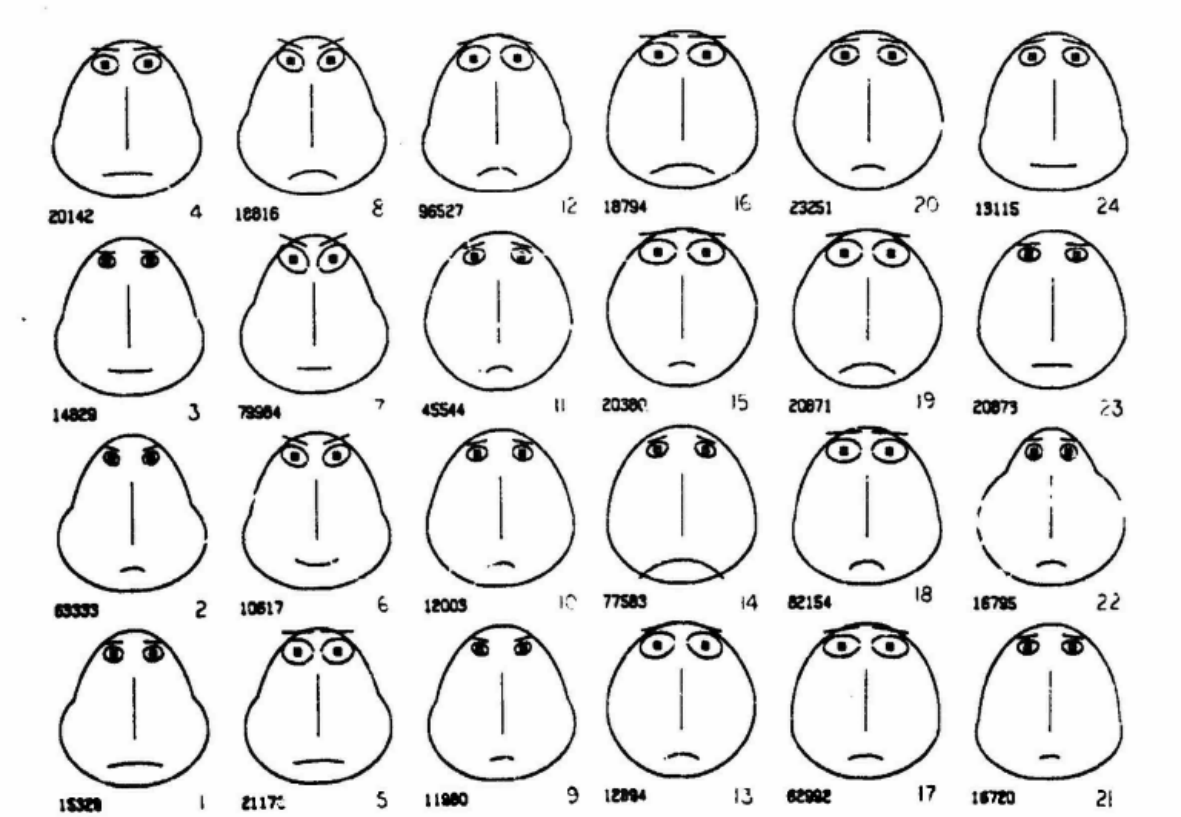

Abbildung 3.7: Originaldarstellung der Chernoff-Gesichter [Che73]

oder ob es nicht stattdessen besser wäre die Daten auf einige wenige Dimensionen zu vereinfachen. Dies kann wie bereits bei den beschriebenen Streudiagramm-Matrizen einfach mithilfe von Schnittebenen im Hyperwürfel geschehen, bei denen bis auf zwei Merkmale alle anderen konstant gehalten werden. Aufgrund der Vielzahl an Kombinations-Möglichkeiten bei hoher Dimensionalität ist das jedoch nur wenig sinnvoll, zudem bleibt die Größe des Datensatzes, also die Anzahl der Punkte erhalten.

3.5.1 Projection Pursuit

Anstatt einfach eine Schnittebene zu erstellen ist es ratsamer statistische Verfahren dazu zu verwenden den Datensatz zu filtern und in zweidimensionale Darstellungen zu projizieren. Diesen Ansatz veröffentlichten 1974 erstmals J.W. Tukey und J.H. Friedman als „Exploratory Projection Pursuit“ [FT74], für den sie einen linearen Mapping Algorithmus entwickelten, welcher anhand eines Index der Interessantheit die Daten auf Streuung und Gruppierung hin analysiert. Anschließend wird das Ergebnis in einem für uns einfach lesbaren ein- oder zweidimensionalen Graphen projiziert.

Das Problem bei diesem Verfahren ist, dass wir entscheiden müssen, auf welche Eigenschaft wir uns bei der Projektion konzentrieren wollen was umfangreiche Kenntnisse im Umgang mit großen Datensätzen voraussetzt. Zwar können wir anschließend sehr schnell die gesuchten Muster erkennen, jedoch können dabei unbekannte Eigenschaften des Datensatzes unbeachtet bleiben. Somit wird uns eine Projektion nicht davor bewahren können den Datensatz noch einmal separat und ungefiltert zu betrachten.

3.5.2 Kleine und Große Reise

Anstelle statischer Darstellungen gibt es auch die Idee alle möglichen Betrachtungsweisen auf einen Datensatz der Reihe nach anzuzeigen. Dazu wird eine Abfolge aller Blickwinkel auf das Diagramm durchlaufen, bei der zusätzlich auch alle Merkmalskombinationen für die Achsen berücksichtigt werden müssen. Als Diagramm wird entweder ein dreidimensionales Streudiagramm verwendet, was Stützle 1984 als „Kleine Reise“ bezeichnet hat, oder aber für die „Große Reise“ [Asi85] eine zweidimensionale Streudiagramm-Matrix mit einer beliebigen Größe erstellt. Für eine erfolgreiche Analyse muss es möglich sein, den Durchlauf zu unterbrechen und beliebig oft vor- und zurück zu gehen, um eventuell erkannte Besonderheiten genauer betrachten zu können. Für einige wenige Dimensionen können damit Daten vollständig in ansprechender Weise untersucht werden. Allerdings führen hochdimensionale Daten zu einer so großen Anzahl an Möglichkeiten, dass ein Durchlauf nur unter enormen Zeitaufwand oder gar nicht mehr umsetzbar ist.

3.6 Ergebnis

Wir haben nun einige Techniken kennengelernt, mit denen sich höher dimensionale Daten gut visualisieren lassen. Die Ansätze gehen dabei zum Teil ganz verschiedene Wege und beschränken sich entweder auf eine sehr geringe Dimensionalität der Darstellung ($d \leq 3$ Dimensionen) und konzentrieren sich auf die Lesbarkeit großer Datenmengen, oder aber sie versuchen möglichst viele Dimensionen darzustellen, jedoch mit der Einschränkung nur für kleinere Datensätze übersichtlich zu sein. Schwierig wird es jedoch bei einer Dimensionalität $d > 50$: Keiner der genannten Techniken kann hier vollständig zufriedenstellende Ergebnisse liefern, ohne die Daten in Teilgruppen aufzuspalten oder Verfahren zur Reduzierung der Größe anzuwenden.

Im Hinblick auf die Visualisierung von Funktionen, also Werten, die von einem oder mehreren Merkmalen abhängig sind, stellen Schnittebenen im Hyperwürfel als 2D-Heatmap oder als 3D-Funktionsgraphen,unterstützt durch Höhenlinien oder Farbverläufe, die einzig sinnvolle Darstellungsvariante dar [Sco09], wenn nicht noch zusätzliche Informationen zur Beschreibung des Funktionsverlaufs vorliegen, die stattdessen für weitere Darstellungen genutzt werden können.

3.6.1 Anwendbarkeit auf Dünne Gitter

Durch die Gitterpunkte und den zugehörigen Überschüssen unserer Dünnen Gitter liegen uns neben den normalen Funktionswerten genügend Zusatzinformationen vor, so dass wir die interpolierte Funktion neben den Schnittebenen auch noch mithilfe anderer Darstellungen analysieren können.

Für die Gitterpunkte selbst ist es naheliegend, Streudiagramme in allen genannten Variationen, also 2D/3D einzeln oder als Matrix, zu verwenden und die Überschüsse für die Darstellung der Punkte heranziehen, einfach aus dem Grund, dass wir an die Arbeit mit Streudiagrammen gewöhnt sind und für einige wenige Dimensionen die Matrizen sehr viele Informationen bereit stellen.

Ein weiterer Vorteil unserer Dünnen Gitter ist, dass wir stets versuchen, die Anzahl der Punkte klein zu halten, was uns bei allen Darstellungsvarianten zugutekommt. Gerade deshalb eignen sich auch die Profildiagramme in allen Variationen besonders gut für unsere Dünnen Gitter, da wir mit ihnen auch hochdimensionale Funktionen vernünftig anzeigen können. Außerdem bieten gerade die Profildiagramme mit ihren zahlreichen Einstellmöglichkeiten viele Hilfsmittel um alle in unseren Gittern enthalten Informationen einzeln untersuchen zu können. So können wir etwa die Entwicklung der Überschüsse über die Level hinweg untersuchen oder aber die verschiedenen Dimensionen einzeln und zueinander auf ihre Abhängigkeiten hin prüfen.

Vermutlich auch interessant aber leider nur schwer umsetzbar wären die Chernoff Gesichter, deren praktischer Nutzung nicht leicht abschätzbar ist, vermutlich aber nicht im Verhältnis zum Aufwand für die Umsetzung steht. Die Projektionstechniken stellen sich als weniger interessant für Dünne Gitter dar, weil oftmals bereits Klassifikationen der Erstellung der Gitter zugrunde liegen, aber auch, weil uns bei den Gittern, deren Größe ohnehin bereits versucht wurde auf ein Minimum zu begrenzen, jeder Punkt interessiert und eine Filterung daher unerwünscht wäre.

4 Entwicklung eines Visualisierungswerkzeugs für SG^{++}

Bei der Arbeit mit SG^{++} ist es wichtig, Veränderungen am Gitter und den Koeffizienten ohne großen Aufwand visuell auf ihren Nutzen hin überprüfen zu können, etwa wenn Adaptionen zur Verbesserung der Interpolation oder Zeitreihenanalysen durchgeführt werden. Da SG^{++} für hochdimensionale Anwendungen entwickelt wurde, können bei den Gitterpunkten oftmals nur Visualisierungen der Koeffizienten Aufschluss über ihren Beitrag zur Funktion geben. Im vorherigen Kapitel haben wir einige Visualisierungen kennengelernt, mit denen wir Daten darstellen können. Typischerweise sind Werkzeuge zur Erzeugung solcher Diagramme sehr allgemein gehalten um eine möglichst große Bandbreite an Anwendungsgebieten abdecken zu können. Besonders die Bereitstellung der Daten und das Anpassen der Darstellung kann daher sehr zeitaufwendig sein. Eine interaktive Arbeit ist zumeist nicht möglich, weshalb oftmals für jede Änderung der Darstellung eine neue Grafik erstellt werden muss. Gitter spezifische Darstellungswünsche, wie dem Hervorheben eines Gitterlevels, müssen aufwendig per Hand programmiert werden, was den Fokus der Arbeit von der eigentlichen Analysetätigkeit ablenkt.

4.1 Zielsetzung

Um ein interaktives Visualisieren schnell, unkompliziert und ohne aufwendige Programmierarbeit zu ermöglichen, haben wir im Rahmen dieser Arbeit ein eigenes Werkzeug entwickelt, dass speziell für den Einsatz bei Dünnen Gittern mit SG^{++} gedacht ist. Wichtig war, dass der Zugriff auf die Datenstruktur möglichst allgemein gehalten bleiben soll, um alle Gittertypen von SG^{++} zu unterstützen, und dass wir ohne die Verwendung von lizenzierten Bibliotheken auskommen um ein freies Programm zu erhalten. Das Tool selbst sollte vor allem bieten:

- eine einfache grafische Oberfläche (GUI), welche ohne große Einarbeitung verstanden und genutzt werden kann
- die Verwendung des Tools durch direkte Einbindung in den Code, oder
- den Import von Gitter- und Koeffizienten-Informationen aus Dateien während der Laufzeit
- die Bereitstellung möglichst vieler verschiedener Darstellungen
- Unterstützung hochdimensionaler Funktionen

- ein interaktives Arbeiten mit den Darstellungen, besonders im Hinblick auf Besonderheiten von Dünnen Gittern
- stabile Performance auch bei hoher Dimensionalität und Level-Tiefe
- leichte Erweiterbarkeit in Hinblick auf neue Darstellungstypen

4.2 Die Entwicklungsumgebung

Die erste Frage, die sich bei der Entwicklung einer GUI stellt, ist, welche Programmiersprache man verwenden möchte. SG^{++} unterstützt sowohl die Sprache Python, als auch Java und C++, womit sich ein großer Raum an Möglichkeiten bietet. Auch die Verwendung von OpenGL zum Zeichnen aufwendiger Darstellungen kann mit allen drei Sprachen umgesetzt werden. Bei Java wird dazu auf das Open-Source-Project „Java OpenGL“ (JOGL) zurückgegriffen, bei Python kann „PyOpenGL“ verwendet werden. Für die grafische Benutzeroberfläche stehen für die jeweiligen Sprachen verschiedene Toolkits bereit.

4.2.1 Java

Für Java stehen drei Bibliotheken für die Erstellung einer GUI bereit. Zum einen sind dass das Abstract Windowing Toolkit (AWT) und das darauf aufbauende Swing, welche beide Bestandteil der Java Foundation Classes (JFC) sind, und das von IBM für die Entwicklungsumgebung Eclipse entwickelte Standard Widget Toolkit. JOGL ist zu allen drei Ansätzen kompatibel. Während Swing auf leichtgewichtige, von der Plattform losgelöste Komponenten setzt, verwenden AWT und SWT die nativen Komponenten des Betriebssystems, um so eine angenäherte Optik mit anderen Programmen des Systems zu erreichen. SWT erfreut sich heute großer Beliebtheit, kann jedoch zu Performance-Problemen führen und wird standardmäßig nicht unterstützt. Swing zeichnet sich besonders durch seine Geschwindigkeit aus und ist fester Bestandteil der Java Runtime Environment, ist aber eigentlich nur als programmierte Oberfläche gedacht, weshalb WYSIWYG Ansätze nur von Drittanbietern zur Verfügung stehen. Trotzdem stellt sich Swing aufgrund seiner Schnelligkeit und seines Funktionsumfangs als die vermutlich beste Wahl für die Erstellung anspruchsvoller Oberflächen mit Java dar. Siehe dazu auch [Ull].

Schlussendlich haben wir uns gegen die Nutzung von Java entschieden, hauptsächlich, weil die Performance von Java Programmen mit der von C++ und Python nicht mithalten kann, aber auch weil die Wahl der Sprache am Ende eine subjektive Entscheidung bleibt.

4.2.2 Python

Python ist eine sehr leicht zu erlernende und gut lesbare Sprache, welche sich immer größerer Beliebtheit erfreut. Eine GUI kann mithilfe des Tools Tk, welche in der Standard Python Bibliothek enthalten ist, und dem Modul Tkinter entwickelt werden, jedoch ist der

Funktionsumfang begrenzt. So muss bereits für die Verwendung von PyOpenGL zusätzlich ein Modul eingebunden werden. Bessere Alternativen stellen aus der C++ Umgebung übernommene Toolkits wie wxPython und PyQt dar, welche durch Bindings auch für Python nutzbar sind. Diese werden wir aber im nächsten Abschnitt betrachten. Für eine ausführlichere Beschreibung sei auf [PK08] verwiesen.

4.2.3 C++

C++ ist die vermutlich umfangreichste Sprache und bietet unter anderem auch eine große Auswahl an Toolkits zur GUI Erstellung. Die wichtigsten plattformunabhängigen Umgebungen stellen Gtk, wxWidgets und Qt dar. Alle drei erfreuen sich großer Beliebtheit und bieten einen großen Funktionsumfang. Während aber Gtk und wxWidgets mehr und mehr aus Groß-Projekten verschwinden, wird der Einsatzbereich von Qt immer größer und stellt heute das zugrunde liegende Framework für die Unix basierte Desktop Umgebung KDE und bald auch für Unity dar. Das ist nicht zuletzt auch der Vielzahl an verfügbaren Einführungen in Qt zu verdanken, welche den Einstieg in die ohnehin einfach zu erlernende Umgebung ermöglichen. Für die Verwendung von OpenGL bietet Qt zudem eine Vielzahl vereinfachter Klassen an, sodass die doch recht umfangreichen OpenGL Befehls-Konstrukte weitestgehend vermieden werden können.

Deswegen und auch weil mit der einfachen Einbindung von Python die beiden wichtigsten Sprachen, welche bei der Arbeit mit SG^{++} verwendet werden, abgedeckt sind, viel unsere Wahl für die Entwicklung unseres Werkzeugs auf Qt (`http://qt-project.org/`).

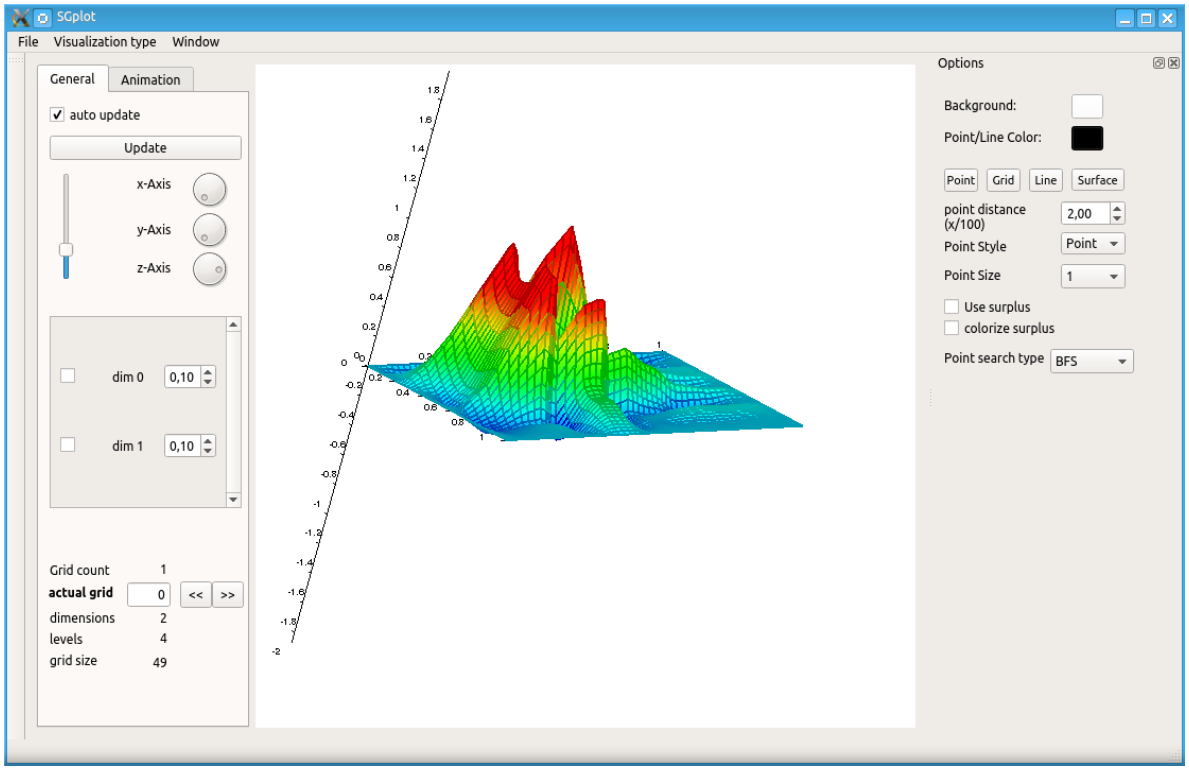

Abbildung 4.1: SGplot - Standard Ansicht nach dem Start. Für die Grafik wurde ein Moons-Datensatz (siehe Kapitel 5) gewählt und als 3D Heatmap dargestellt.

4.3 SGplot

Mit SGplot (Sparse Grid plot) haben wir ein Programm erstellt, dass ein oder mehrere Dünne Gitter und ihre dazugehörigen Koeffizienten in verschiedenen Darstellungen präsentieren kann, sowohl statisch als auch animiert. Mit Qt als Entwicklungsumgebung ist das Tool Betriebssystem unabhängig verwendbar, vorausgesetzt die benötigten Bibliotheken von SG^{++} sind für die jeweilige Plattform verfügbar. Die Abbildung 4.1 zeigt die grafische Oberfläche des Programms nach dem Start. Im folgenden werden wir uns näher mit dem Aufbau von SGplot, den verschiedenen Darstellungstypen und ihren Einstellmöglichkeiten, sowie den Import- und Export-Möglichkeiten und die Verwendungsmöglichkeiten des Programms beschäftigen. Zur Darstellung haben wir einige Beispiel-Datensätze verwendet, auf die wir aber im nächsten Kapitel noch näher eingehen werden.

4.3.1 Aufbau

Wir haben versucht das Programm so aufzubauen, dass es möglichst einfach verstanden und weiter entwickelt werden kann. Die wichtigsten Klassen sind:

1. SGinterface
2. SGplotMain
3. GLWidget
4. Drawer

SGInterface

Das „Interface“ dient als Schnittstelle zu den Gitter Daten und den Koeffizienten. Liegt eine Menge von Gittern (Set) vor, wird für jedes Gitter ein eigenes Interface erstellt, sodass nur der Zeiger auf das Objekt geändert werden muss um Iterationen über ein Set möglichst einfach unterstützen zu können. Innerhalb dieser Klasse haben wir zwei Ansätze untersucht, mit denen wir die Funktionswerte berechnen können (siehe dazu Performance 4.3.7): Punkt für Punkt und mit Hilfe einer Matrix. Außerdem haben wir den Algorithmus 4.1 implementiert, mit dem wir die Gitterpunkte dem Level nach sortieren um so die Punkte sowohl der Reihe nach als auch Level für Level durchlaufen zu können.

Algorithmus 4.1 Sortierung der Gitterpunkte nach Level

```
function getNextSortedPoint(pos)
    if FirstCall then                                  // nur beim ersten Aufruf sortieren
        pointList[Number of GridPoints]
        LevelLists[Number of Grid Levels]
        for all p ∈ GridPoints do
            levelSum ← levelSum of Point p from GridStorage
            if BoundaryType then
                LevelLists[levelSum − d + 1] ← pushBack(p)
            else
                LevelLists[levelSum] ← pushBack(p)
            end if
        end for
        for all l ∈ Grid − Levels do
            pointList[p] ← add all points fromList[l]
        end for
        FirstCall ← false ;
    end if
    return pointList[pos]
end function
```

SGplotMain

Dies ist die Hauptklasse des Programms. Sie enthält alle Programm-Teile und regelt die Ansteuerung des Tools durch die Benutzeroberfläche. Die Kommunikation zwischen den

Listing 4.1 Auszug aus den erstellten Verbindungen im Konstruktor von SGplotMain. Zu beachten ist, dass immer nur der Typ einer Variablen angegeben wird

```
connect(ui->actionDockScatter, SIGNAL (triggered()), ui->dockScatter, SLOT (show()));
connect(ui->dSC_dimAll, SIGNAL (toggled (bool)), ui->dSC_dimensions, SLOT
    (setDisabled(bool)));
onnect(ui->action3D,SIGNAL (triggered()), ui->DrawArea, SLOT(set3Dim()));
```

Algorithmus 4.2 die Zeichenroutine des GLWidgets

procedure PAINTGL
  GLCLEARCOLOR(bgRed, bgGreen, bgBlue, alpha)
  GLCLEARDEPTH(1.0)
  GLCLEAR($GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT$)
  GLMATRIXMODE($GL\_PROJECTION$)
  GLLOADIDENTITY
  drawer $\leftarrow$ XYDRAWER($SGinterface, DrawSettings$)
  GLORTHO($left, right, bottom, top, near, far$)
  GLMATRIXMODE($GL\_MODELVIEW$)
  DRAWERDRAWXYPLOT
  DRAWAXIS
end procedure

GUI-Elementen und den Objekten haben wir mit QT's SIGNAL/SLOT Konzept umgesetzt (`http://qt-project.org/doc/qt-4.8/signalsandslots.html`), welches im Vergleich zu Callbacks typsicher und einfach anzuwenden ist, auch wenn sie zu einem minimalen Geschwindigkeitsverlust führen, der in der Praxis aber nicht bemerkbar ist. Zur Erstellung werden die von Qt eingeführten C++ Schlüsselwörter *signal, slot, emit* verwendet. Um einen Slot auf ein Signal reagieren lassen zu können müssen wir sie zuvor einmalig verbinden.

Daneben können Slots auch als normale Funktionen verwendet werden, um beispielsweise Code generierte Aufrufe nicht extra über Signals auslösen zu müssen.

GLWidget

In diesem Objekt der GUI, welches von QGLWidget abgeleitet wurde, können mithilfe von OpenGL Grafiken erstellt werden. Der Hauptaugenmerk liegt hier auf der `paintGL()` Prozedur 4.2. Jedes mal wenn ein Update ausgelöst wird , wird mit `paintGL()` die aktuelle Darstellung gelöscht und ein neuer Frame gezeichnet. Je nach Darstellungstyp wird dazu erst die Perspektive auf die Szene eingestellt und anschließend ein *Drawer* aufgerufen, der das Erstellen des Diagramms übernimmt, bevor zum Schluss noch zusätzliche Inhalte wie Achsen, Beschriftungen oder Farb-Skalen gezeichnet werden.

Drawer

ist die Parent-Klasse, von der alle weiteren Zeichenroutinen abgeleitet werden. Die Idee hinter der Vererbung war es, alle Gemeinsamkeiten unter den Zeichnern bereits im Drawer zu implementieren. Jede Unterklasse bietet dann die Möglichkeit verschiedene Zeichenroutinen der selben Darstellungsmethode unter zu bringen, beispielsweise die Methoden Parallele Koordinaten, Andrews Kurven und Polygon-Diagramm in der Unterklasse „Profil-Diagramme“. Wir haben uns innerhalb dieser Arbeit für die folgenden fünf Drawer-Unterklassen entschieden:

1. GraphDrawer - ein- oder mehrfarbiger (Heatmap) Funktionsgraph der Funktionswerte im 2 und 3 dimensionalen Raum
2. GridDrawer - Visualisierung der Gitterpunkte im 2 und 3 dimensionalen Raum
3. ChartDrawer - innerhalb dieser Klasse können Methoden zur Erstellung von Balken oder Kreis Diagrammen erstellt werden
4. SCPDrawer - Erstellung einer Scatterplotmatrix
5. PCPDrawer - verschiedene Methoden zur Erstellung von Profil Diagrammen

4.3.2 Bedienkonzept

Wir haben versucht die Oberfläche von SGplot so zu gestalten, dass es sich weitestgehend intuitiv bedienen lässt. Neben den in Abbildung 4.2 gezeigten Einstellungsfeldern, über die alle Analyse Techniken und Einstellungsmöglichkeiten für die Grafiken erreichbar sind, gibt es noch die Programm-Leiste mit den Einträgen **File, Visualization Type, Window**. Über File wird typischerweise der Datei-Manager und die Speicherfunktion bereit gestellt, über Visualization Type kann die Darstellungsvariante gewählt werden und über Window lassen sich die einzelnen Optionsfelder ein- und ausblenden. Außerdem bietet SGplot einige Tastenkürzel um das arbeiten zu beschleunigen (siehe 7). Mithilfe der Maus und gedrückter linker Taste kann zudem der 3D Graph frei rotiert werden.

4.3.3 Darstellungen

Unser Anspruch war es möglichst interaktiv mit hochdimensionalen Dünnen Gittern arbeiten zu können und Methoden bereit zu stellen mit denen sich Gitter spezifische Eigenschaften untersuchen lassen. In Kapitel 3 haben wir bereits einige Darstellungstypen betrachtet. Im Hinblick auf Dünne Gitter haben wir uns bei SGplot wie in Abschnitt 3.6 erläutert für die folgenden Darstellungen entschieden:

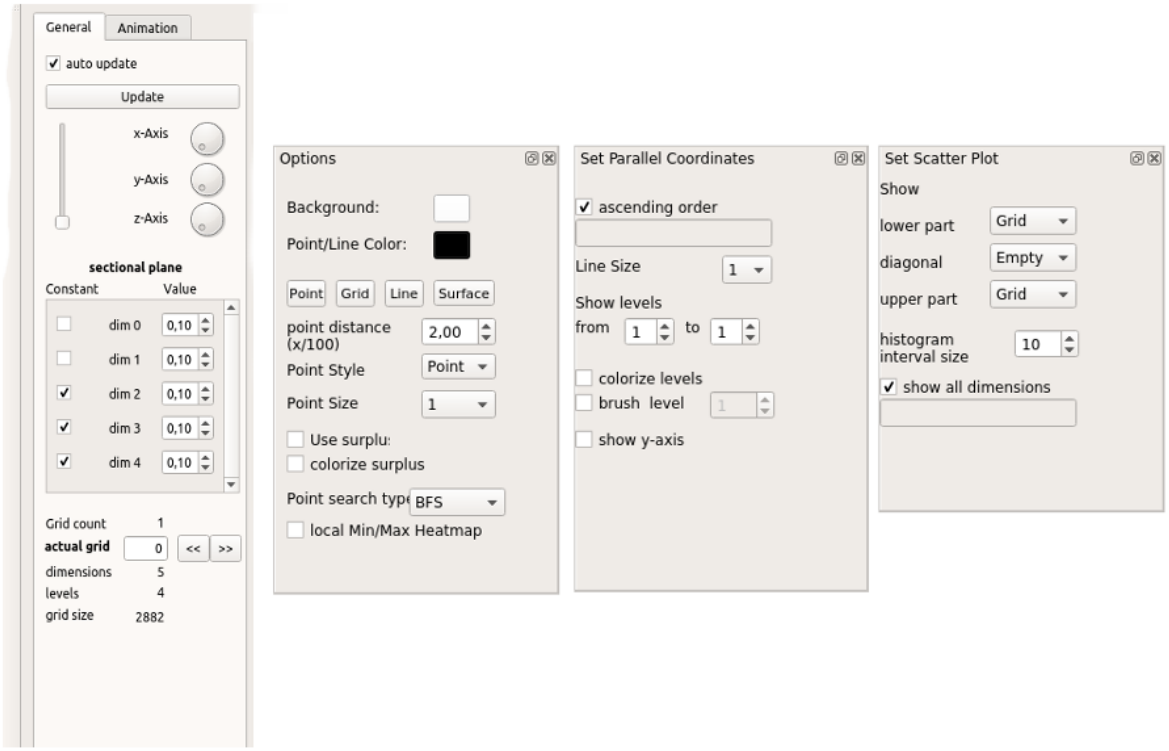

Abbildung 4.2: Bedienfelder von SGplot, links zur Einstellung der Perspektive, Schnittpunktwahl und Animation, rechts Einstellmöglichkeiten für die verschiedenen Darstellungen wie Farbe, Überschuss-Verwendung, Dimensions- und Level-Wahl

2D / 3D-Funktions-Graph

Einfache Funktionsgraphen gehören zu den Basisdarstellungen und sollten keinesfalls fehlen. Sie helfen uns dabei einen ersten Eindruck von der approximierten Funktion zu bekommen. In SGplot kann dazu neben der Anzahl der Stützstellen auch zwischen den vier verschiedenen Funktionswert-Darstellungen **Punkte, Linien, Gitter, Fläche** gewählt werden. Für die 3D Darstellungen stehen zusätzlich noch Achsrotation und Zoomen bereit. So kann ein Maximum an Interaktivität mit einem Funktionsplot erreicht werden.

2D / 3D Heatmap

Als Erweiterung zum normalen Funktionsgraphen bietet SGplot zusätzlich noch die Möglichkeit die berechneten Punkte anhand ihres Funktionswertes einfärben zu lassen. Um eine Vergleichbarkeit bei mehreren möglichen Dimensions-Paarungen gewährleisten zu können, kann die Farbe über dem Werte-Minimum und -Maximum aller Dimensionen interpoliert

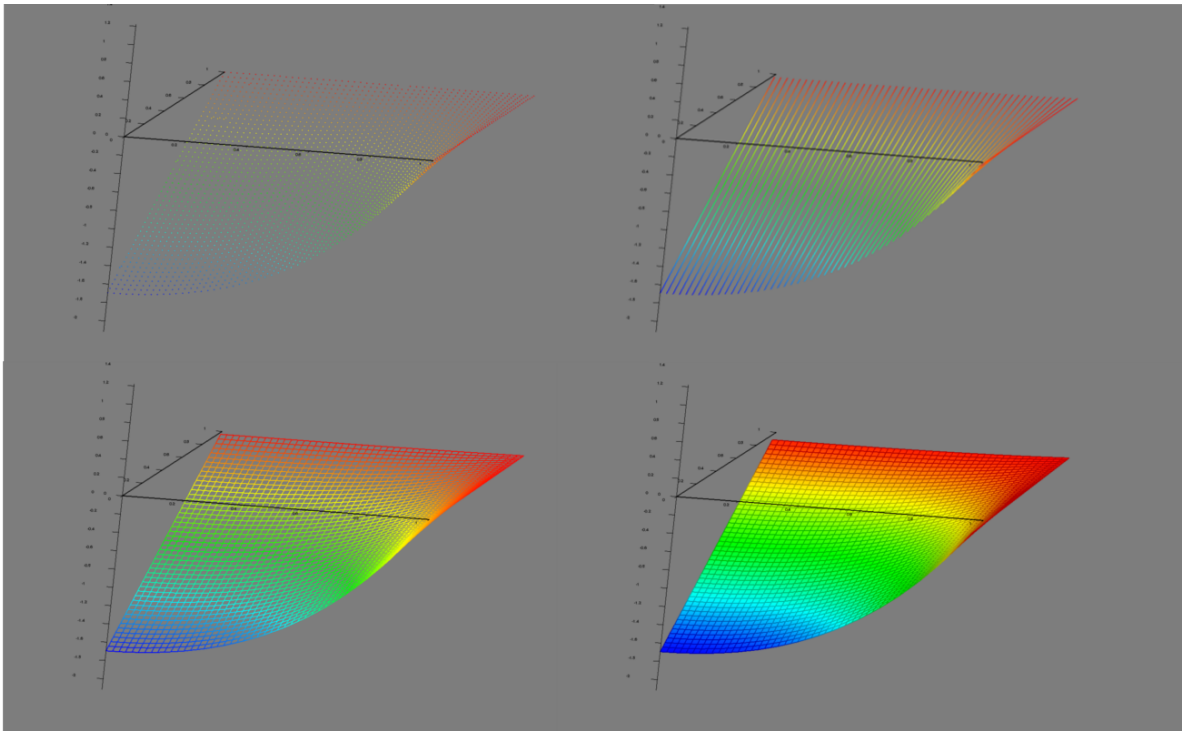

Abbildung 4.3: Funktionsgraph 3D in allen 4 Funktionswert-Darstellungsvarianten

werden, eine lokale Einfärbung ist aber auch möglich. Für die 2D Darstellung wird die dreidimensionale Heatmap auf die xy-Ebene projiziert.

2D / 3D Gitterpunkt Darstellung

Für die Visualisierung der Gitterpunkte stellt SGplot ein zwei- und dreidimensionales Streudiagramm bereit. Im 2D Diagramm können die dazugehörigen Überschüsse als absolute Skalierung der Punkte visualisiert werden. Anstelle der Punkte können aber auch Kreise, Rechtecke oder Kreuze gezeichnet werden. Bei allen vier Punktarten können außerdem die Überschüsse zur Farb-Interpolation genutzt werden, wodurch negative (blau) und positive (rot) Überschüsse unterschieden werden können. Bei der dreidimensionalen Darstellungen können entweder die Gitterpunkte in 3D dargestellt werden. Anstelle der dritten Dimension für die Gitterpunkte können aber auch die Überschüsse als Balkendiagramm visualisiert werden. Durch negativ und positiv verlaufende Balken sind so auch Überschüsse kleiner Null darstellbar.

Algorithmus 4.3 Grundschema für das Zeichnen der Funktionsgraphen

```
procedure drawHeatmap(r)                                  // Anzahl Stützstellen r
    Values ← sgi->getValuesByMatrix(r);
    glColor3f(r, g, b)
    for x = 0; x ≤ r do
        glBegin(PointType)
        for y = 0; y ≤ r do
            if Heatmap then
                color_rgb ← calculate rgb color
                glColor3f(r, g, b)
            end if
            if 3D then
                glVertex3f(x, y, Values[x ∗ r + y])
            else if Heatmap then
                glVertex3f(x, y, 0)
            end if
        end for
    end for
end procedure
```

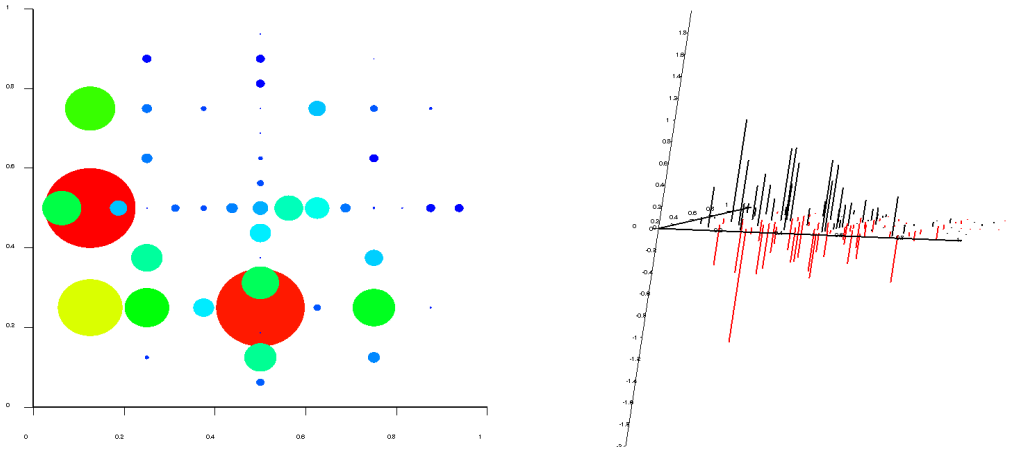

Abbildung 4.4: Streudiagramm in 2D und 3D mit Überschüssen. Links sind die Überschüsse betragsmäßig als eingefärbte Kreise dargestellt, rechts sieht man die Überschüsse in ihrer nativen Form, negative Überschüsse sind als rote Balken dargestellt

Scatter-Plot-Matrix

Handelt es sich um ein mehrdimensionales Gitter sind Matrix Darstellungen das am meist verbreitete Mittel um sich auf einem Blick einen Eindruck über alle Dimensionen zu verschaffen, solange die Anzahl der Dimensionen kleiner 10 bleibt. Um den gesamten Platz einer Matrix effizient zu nutzen, bietet SGplot die Möglichkeit für jeden der drei Bereiche „unteres Dreieck, oberes Dreieck, Diagonale“ verschiedene Darstellungen zu verwenden. Zunächst stehen dazu das 2D-Gitter, die 2D-Heatmap, ein Histogramm der Funktionswerte und für die Diagonale zusätzlich noch eine textuelle Beschreibung zur Verfügung. Das Tool kann aber noch um beliebige andere Darstellungen erweitert werden. Bei der Darstellung der Matrix haben wir uns an den Ergebnissen der Sprache „R“ orientiert, bei der eine im Wechsel versetzte Beschriftung gewährleistet, dass die Matrix leicht ablesbar bleibt und nicht überladen wirkt.

Profil Diagramme

Um auch viele Dimensionen überschaubar darstellen zu können, haben wir in SGplot die Profil Diagramme **Parallele Koordinaten, Andrews Kurven** und das **Polygon Diagramm** implementiert. Die drei Darstellungen basieren dabei auf dem selbem Basis-Algorithmus mit allen zu Verfügung stehenden Analyse Techniken und unterscheiden sich nur in ihrem Erscheinungsbild. Während bei den Parallelen Koordinaten und dem Polygon Diagramm die Gitterpunkte einfach als Linien entweder geradlinig oder im Kreis dargestellt werden, werden sie für die Andrews Kurven vorher in eine trigonometrische Funktion umgewandelt und nur diese wird anschließend im Bereich von $-\pi$ bis π gezeichnet.

4.3.4 Analyse-Techniken

Neben allgemeinen Einstellmöglichkeiten wie der Hintergrundfarbe, Punktgröße und -farbe haben wir zusätzlich noch einige Methoden entwickelt, die die Analyse mit der jeweiligen Darstellung verbessern sollen. Außerdem bieten wir dem Benutzer die Möglichkeit, sich den Datensatz animieren zu lassen. Im Folgenden werden wir die verschiedenen Techniken beschreiben und ihren Nutzen für die Analyse bei Dünnen Gittern ersichtlich machen.

Schnittebenen-Konfiguration

Schnittebenen stellen eine sinnvolle Möglichkeit dar um Funktionsgraphen zu zeichnen, welche von mehr als zwei Merkmalen abhängig sind. Alle nicht verwendeten Merkmale werden konstant gehalten. Um dennoch eine Betrachtung des gesamten Datensatzes gewährleisten zu können, haben wir einen dynamischen Schnittebenen-Konfigurator erstellt, mit dem ausgewählt werden kann, welche Merkmale betrachtet und welchen konstanten Wert für die übrigen Dimensionen als Schnittpunkt angenommen werden sollen. Im Zusammenspiel mit dem Betrachtungswinkel auf die Szene, welchen wir über die Maus oder die Einstellräder für

Algorithmus 4.4 Grundschema für das Zeichnen der Profildiagramme, das Streudiagramm wird auf ähnliche Weise erstellt

```
  procedure drawProfiles
Require: minLevel : mi , maxLevel : ma
    dim = Number of chosen dimensions
    lineDistance = 1/(dim − 1)
    if checkMouse then
        checkLineSelected(sgi, lineDistance, mi, ma)
        checkMouse ← false
    end if
    drawSelectedLines(sgi, dim, lineDistance, mi, ma)
    vec[dim]
    size ← GridSize
    glLineWidth(lineSize)
    for all p ∈ GridPoints do
        getNext_BFS_PointNumber(p)
        levelSum ← getGridCoord(vec, p)
        if mi <= levelSum <= ma then
            if NotSelected then
                if surplus then
                    interpolateSurplus(p)
                else if LevelAppearsFirstTime then
                    chooseColor(levelSum)
                end if
                if isPCP then
                    drawParallelCoordinates(dim, lineDistance, vec)
                else if isPolygon then
                    drawPolygonPlot(dim, vec)
                else
                    drawAndrewsPlot(dim, vec)
                end if
            end if
        end if
    end for
  end procedure
```

die Achsdrehung einstellen, kann so der Datensatz gemäß der kleinen Reise 3.5.2 vollständig betrachtet werden. Aber auch für die Scatter-Plot-Matrix kann die Wahl der Konstanten nützlich sein, die Festlegung der verwendeten Merkmale wird hier ignoriert.

Überschuss-Mapping

Von besonderer Bedeutung bei Gittern ist die Analyse der Überschüsse, denn erst sie geben Auskunft über die Qualität der Approximation. Bei einigen tausend Punkten bieten jedoch Zahlen nur eine unzureichende Anschaulichkeit. Deshalb haben wir dem Anwender die Möglichkeit gegeben, sich die Überschüsse auf verschiedene Weise grafisch darstellen zu lassen:

1. Als Balkendiagramm im 3D Gitterpunkte Diagramm, woran sich die positiven und negativen Überschüsse direkt erkennen lassen. Der Vorteil dieser Darstellung ist, dass die Überschüsse weder normalisiert noch betragsmäßig angepasst werden müssen, dafür sinkt die Übersichtlichkeit mit der Zahl der Gitterpunkte.
2. Als Punktgröße in einem Gitterpunkte Diagramm. Als Punktdarstellung kann ein Punkt, Kreis, Rechteck oder Kreuz gewählt werden, deren Größe und wenn gewünscht auch Farbe abhängig von der Größe des Überschusses ist. Diese Darstellung ist besonders gut für die Scatter-Matrix geeignet, da so schnell die Verteilung der Überschüsse erkennbar ist. Allerdings zieht die Überlagerung der Punkte aus den verschiedenen Ebenen eine gewisse Unübersichtlichkeit mit sich, hier kann die Darstellung der interpolierten Punkte als 3D Gitterdiagramm helfen.
3. Als Linienstärke und Farbe in den Profil Diagrammen, um die Bedeutung der einzelnen Linien im Profil Diagramm einschätzen zu können. Zusammen mit der *Level- und Dimensionen-Auswahl* können auffällige Gitterpunkte oder ganze Level-Charakteristiken schnell identifiziert werden.

Für die Interpolation der Farbe wird wie bei der Heatmap die statische Funktion rgb_color$(alphaMin, alphaMax, size, r, g, b)$ der Toolbox verwendet. Für die Größe der Punkte und Linien bietet SGplot vier Interpolations-Ansätze, ein Multiplikator b, der sich über die GUI einstellen lässt, bietet die Möglichkeit Einfluss auf die Größe aller Punkte zu nehmen:

1. Linear: $size = max \cdot |s| / alphaMax$ liefert eine gut differenzierte Darstellung aller Größen, allerdings können die Überschüsse nur absolut verwendet werden
2. Quadratisch: $size = max \cdot s^2$ hebt besonders größere Überschüsse hervor, auch wieder ohne zwischen positiver und negativer Werte unterscheiden zu können
3. Exponentiell: $size = 0.5 \cdot max \cdot \exp(s)$ gewichtet besonders die positiven ($\exp(s)$) oder negativen ($\exp(-s)$) Überschüsse
4. Logarithmisch: $size = max \cdot |\log(s)|$ gewichtet besonders kleine und sehr große Überschüsse

Level- und Dimensionen-Auswahl

Für eine erfolgreiche Analyse mit Profil Diagrammen ist es notwendig, dass sich die Reihenfolge der Dimensionen beliebig ändern lässt. In SGplot haben wir diese Grundanforderung noch dahingehend erweitert, dass wir nicht nur die Reihenfolge frei wählen können, sondern auch beliebig viele Dimensionen angeben können um so sowohl Teilmengen als auch Mehrfachnennungen zu unterstützen. Die selbe Technik stellen wir zudem auch für die Scatter-Matrix bereit, da mit der Dimensionswahl auch bei hoher Dimensionalität sowohl die Übersichtlichkeit, als auch die Echtzeit-Lauffälligkeit erhalten werden kann.

Außerdem bieten wir in SGplot die Möglichkeit nur Gitterpunkte eines einzelnen oder einer bestimmten Menge an Level darstellen zu lassen, was besonders wichtig bei der Arbeit mit Dünnen Gittern ist, um etwa schnell untersuchen zu können, wie sich die Überschüsse mit steigendem Level verhalten. Die Auswahl dafür geschieht über die Level-Summe und lässt sich nach unten und oben beschränken.

Level-Einfärbung

Zusätzlich zur Level-Auswahl lassen sich die einzelnen Levels auch unterschiedlich einfärben, um eine optimale Unterscheidung zu ermöglichen. Allerdings wird bei einer zu großen Anzahl an Gitterpunkten die Einfärbung unkenntlich, das gleiche gilt für eine hohe Level-Tiefe.

Brushing

Sollen trotz schwieriger Erkennbarkeit möglichst große Mengen von Gitterpunkten in den Profil Diagrammen dargestellt werden, bietet es sich an statt einer Einfärbung aller Linien nur eine bestimmte Menge „heraus zu kämmen“. Dieses sogenannte Brushing haben wir zur Hervorhebung eines Levels umgesetzt, Gitterpunkte des ausgewählte Levels werden rot alle anderen grau gezeichnet. Gerade, wenn wir versuchen Gitterpunkte niedrigen Levels in einem sehr dicht besetzten Gitter zu untersuchen, ohne dabei einen Hauptteil der Punkte ausblenden zu müssen, kann sich diese Methode als sehr nützlich erweisen.

Animation

Für die Arbeit mit Schnittebenen haben wir bereits den Schnittebenen Konfigurator vorgestellt, mit dem sich die Schnittebene beliebig anpassen lässt. Das Ergebnis ist dann ein statisches Bild für jede Konfiguration. Um das Verhalten der Funktion für eine konstant gesetzte Dimension zu untersuchen müsste per Hand der Schnittpunkt Schritt für Schritt verändert werden. Das selbe Problem entsteht bei einem Set von Gittern. Deshalb haben wir uns überlegt, dass es nützlich sein kann, wenn wir die Schnittpunktwahl und die Set Iteration in einer Animation automatisieren. SGplot enthält deshalb ein Tool mit dem sich

beide Animationen erzeugen lassen, die Geschwindigkeit und Dimension können frei gewählt werden. Je nach Rechenleistung und Größe des Gitters können damit flüssig laufende Animationen präsentiert werden. Zudem bietet das Tool die Möglichkeit, die einzelnen Momentaufnahmen der Animation als Bildserie zu speichern, was besonders bei großen Gittern, bei denen Echtzeit-Berechnungen nicht mehr möglich sind, nützlich ist.

4.3.5 Import / Export

Bevor SGplot etwas darstellen kann, muss zunächst ein Datensatz geladen werden. Dazu stellt SGplot vier Möglichkeiten bereit:

1. Import eines einzelnen Gitters. Dazu wird über den Datei-Manager eine .grid Datei, welche zuvor mit SG^{++} erstellt wurde, ausgewählt. Das Gitter und die dazugehörigen Überschüsse, welche in einem gleichnamigen .alpha.arff stehen, werden anschließend in den Speicher geladen und über ein neues SGinterface (4.3.1) zur Verfügung gestellt.
2. Import mehrerer Gitter. Ähnlich wie bei einem einzelnen Gitter wird über den Verzeichnis-Manager ein Ordner ausgewählt, aus dem dann automatisch alle enthaltenen Gitter geladen und als Liste von SGinterfaces zur Verfügung gestellt werden.
3. Laden über config Datei. Die im JSON Format erstellte config Datei, welche neben den Pfadangaben auch Dimensionsbezeichnungen, Scatter-Matrix-Einträge und Parameter Namen enthalten kann, wird auch wieder über den Datei-Manager ausgewählt. Anschließend wird die Datei mithilfe der „boost.propertyTree library“ (`http://www.boost.org/doc/libs/1_41_0/doc/html/property_tree.html`) eingelesen und anschließend wie bei der 2. Variante verfahren. Wichtig hierbei ist, dass das festgelegte Format (siehe Anhang) eingehalten wird.
4. Laden einer DataMatrix. Soll ein Datensatz anstelle eines Gitters visualisiert werden, stellt SGplot auch dafür einige Darstellungen bereit. Der Import der Matrix geschieht über eine im Datei-Manager ausgewählte .arff Datei, deren Einträge in eine DataMatrix geladen werden.

SGplot ist aber nicht nur als abgeschlossenes Programm entwickelt worden. Neben der ausführbaren Anwendung haben wir auch eine Bibliothek erstellt, die es erlaubt das Tool zusätzlich in einem Code einzubetten um so den Umweg über den Export / Import von Gitter Daten vermeiden zu können. Bisher ist dieses Feature aber nur für C++ umgesetzt.

4.3.6 Programm-Start

Wir haben SGplot so entwickelt, dass es sich als normales Programm in eine grafische Desktop Umgebung einfügt. Neben dem direkten Aufruf über das Icon haben wir auch einige Programm-Optionen bereit gestellt, über die sich das Werkzeug konfigurieren lässt. So kann bereits vor dem Start die gewünschte Darstellung gewählt und eingestellt und ein

Datensatz geladen werden. Damit ist es dem Benutzer möglich, schnell und effizient die gewünschten Grafiken zu erzeugen.

Für die Erzeugung der Programm Optionen haben wir die „boost boot options library“ (`http://www.boost.org/doc/libs/1_54_0/doc/html/program_options.html`) verwendet. Mit dieser frei nutzbaren Bibliothek können wir ein Befehlsreihenfolge-unabhängiges, einfaches Analysieren der Kommandozeile umsetzen, ohne einen eigenen Interpreter entwickeln zu müssen. Die aktuell umgesetzten Befehle für SGplot befinden sich im Anhang dieser Ausarbeitung.

4.3.7 Performance

Für ein interaktives Arbeiten ist es notwendig, dass unser Tool schnell auf Eingaben reagiert und Berechnungen nach Möglichkeit in Echtzeit ablaufen. Das ist mit unserem Anspruch hochdimensionale Funktionen zu unterstützen jedoch nicht immer vereinbar. Wir haben deshalb versucht, die Anzahl an Berechnungen weitestgehend gering zu halten und haben untersucht, wo sich ein Geschwindigkeitsverlust einstellt und ob das Problem lösbar ist.

Funktionswert-Berechnung

Für die Darstellung eines Funktionsgraphen müssen wir für eine Menge x an Wertepaaren den Funktionswert mit SG^{++} berechnen und anschließend jeden einzelnen Punkt richtig eingefärbt zeichnen. Flächen- und Linien-Darstellungen werden aus den Punkten mit OpenGL automatisch erzeugt. Die Anzahl der Stützstellen können wir in SGplot über die SpinBox „resolution“ wählen. Unser erster Ansatz war es jeden Funktionswert einzeln mit dem Aufruf `OperationEval->eval(alpha, vector)` zu ermitteln. Für jede Stelle muss dazu ein DataVector, welcher die xy-Koordinaten und die Schnittpunkte der Dimensionen enthält, erstellt werden. Das ganze ist in der Funktion `getValueByConstants(x,y)` im SGinterface umgesetzt.

In Tabelle 4.1 haben wir für verschiedene Auflösungen die Zeit für ein fünf dimensionales Gitter auf einem normalen PC mit einer durchschnittlichen Rechenleistung gemessen. Selbst bei unter 1000 Stützstellenzahl stellen sich Verzögerungen von etwa 150ms bei der Erzeugung des Bildes ein, was beim Rotieren im dreidimensionalen Raum sehr störend auffällt.

Als nächstes haben wir die Berechnung mithilfe einer Matrix in `calcMatrix(resolution)` umgesetzt. Hier müssen wir nur einmal eine DataMatrix erstellen, mit den Wertepaaren befüllen und mit SG^{++} einen Ergebnis-Vektor berechnen. Diese Methode bringt uns zwar einen Zeitgewinn von 50% führt aber bei hohen Auflösungen nach wie vor zu Verzögerungen. Deshalb haben wir uns dazu entschieden das Ergebnis einmal zu berechnen und dann zu speichern, was im Hinblick auf heutige Speichergrößen problemlos möglich ist. Damit erreichen wir auch bei sehr hohen Auflösungen ruckelfreie 3D Manipulationen, einzig die erste Berechnung benötigt einen kurzen Moment ehe die Grafik angezeigt wird.

Auflösung	Stützstellen	Einzeln[ms]	Matrix[ms]	Matrix-Speichern[ms]
10	121	30	20	20
25	676	150	85	20
50	2,601	650	300	20
100	10,201	2,450	1,170	20

Tabelle 4.1: Performance Analyse für unterschiedliche Berechnungsverfahren, durchgeführt mit einem Intel Core i5 Prozessor

Wir können also festhalten, dass die Zeit zur Funktionsdarstellung einzig von der Berechnung mit SG^{++} abhängt und das eigentliche Zeichnen mit OpenGL nur einen unwesentlichen Einfluss hat. Das bedeutet aber auch, dass wir keinen Einfluss auf die Performance haben, wenn es um die Berechnung der Werte geht, was besonders für der Wahl der aktiven Dimensionen und der Änderung der Konstanten negative Auswirkungen auf die Rechenzeit hat. Eine Vorberechnung aller Werte ist aufgrund der großen Anzahl an Möglichkeiten leider nicht umsetzbar, da wir bereits bei nur 5 Dimensionen

$$\binom{n}{k} \cdot 10^3 = \binom{5}{2} \cdot 1.000 = 10.000$$

Möglichkeiten für alle Dimensions und Parameter-Kombinationen (0 bis 1 in 0,1 Schritten) und damit bei 700 Stützstellen einen Zeitaufwand von

$10.000 \cdot 150ms = 1.500.000ms \approx 416$ Stunden erhalten.

Scatter-Plot-Matrix

Eine Scatter Matrix besteht aus einer Serie an Darstellungen für alle paarweise möglichen Dimensions-Kombinationen, typischerweise in zwei verschiedenen Darstellungen. Bei einem fünf dimensionalen Gitter ergibt das 20 zu erstellende Grafiken. Mit der oben beschriebenen Matrix Rechenmethode und einer Auflösung von 50 x 50 führt das zu einem Zeitaufwand von $20x300ms = 6$ Sekunden. Das ist zwar noch akzeptabel, doch im Hinblick auf den linearen Anstieg der Rechenzeit mit steigender Anzahl an Gitterpunkten wird deutlich, dass mit dieser Darstellungsmethode schnell Zeiten von mehreren Minuten pro Aktualisierung erreicht werden (siehe Tabelle 4.2) . Die einzige Möglichkeit zur Reduzierung der Rechenzeit bestünde darin, die Berechnung der Funktionswerte sämtlicher Paarungen zu parallelisieren, und eventuell in ein Array zu speichern, um nur noch bei Schnittpunkt-Änderungen neu berechnen zu müssen. Das Array dürfte aber bei vielen Dimensionen eine erhebliche Größe erreichen. Die Umsetzung war innerhalb dieser Bachelorarbeit aber nicht möglich, weshalb wir uns auf die Dimensionsauswahl zur Reduzierung der Anzahl an Paarungen für die Matrix bei einer Funktionswert-abhängigen Darstellung beschränken müssen.

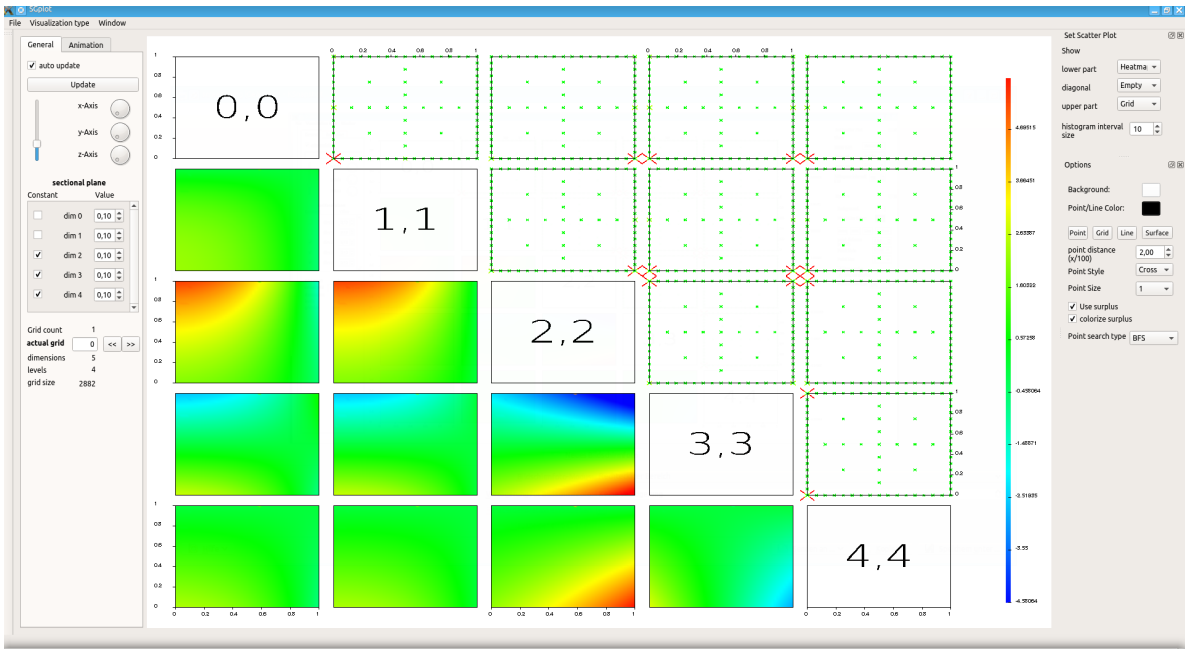

Abbildung 4.5: Performance SCP - 5 dimensionale Scatter Matrix mit Heatmap in Surface Darstellung und Auflösung von 50, sowie Streudiagramm mit farbigen Überschüssen als Kreuze

Anders sieht es bei den Streudiagrammen für die Gitterpunkte mit Überschuss-Darstellung aus. In einem ersten Ansatz haben wir für jedes Feld der Matrix eine neue Zeichenroutine aufgerufen, wodurch jedes Mal sämtliche Gitterpunkte neu betrachtet werden mussten. Das führte jedoch bei 20 Matrixfeldern bereits zu einem Zeitaufwand von $20 * 7 = 140ms$ und bei unserem 12 dimensionalen Beispielgitter zu Zeiten von über 15 Sekunden (siehe Tabelle 4.2). Für eine wesentlich effizientere Gitterpunktbetrachtung haben wir uns deshalb entschieden, für die Gitterpunkte nicht mehr eine Feld für Feld Schleife zu verwenden, sondern jeden Punkt einmal zu Betrachten und anschließend in jedes Feld ein zu zeichnen. Damit konnten wir den Zeitaufwand erheblich verbessern, für die 20 Paarungen beispielsweise nur noch $70ms$ und für das 12D Gitter nur noch $2400ms$ statt über $14000ms$.

Dimensionen	Gitterpunkte	Dauer[ms]	Darstellung 1	Darstellung 2
2	49	20	Heatmap	Streudiagramm1
3	138	15	Streudiagramm2	Streudiagramm2
3	138	50 (100)	Heatmap	Streudiagramm1
5	2,882	70 (150)	Streudiagramm2	Streudiagramm2
5	2,882	300 (2,400)	Streudiagramm1	Streudiagramm1
5	2,882	3,200 (4,400)	Heatmap	Streudiagramm1
12	25,089	2,400 (14,000)	Streudiagramm2	Streudiagramm 2
12	25,089	14,000 (140,000)	Streudiagramm1	Streudiagramm 1
12	25,089	99,000 (170,000)	Heatmap	Streudiagramm1

Tabelle 4.2: Performance Analyse für die Scatter Matrix Darstellung, durchgeführt mit einem Intel Core i5 Prozessor. Die Zeiten in Klammern wurden mit der ersten Gitterpunktdarstellung erreicht. Für die Heatmap haben wir die Surface Darstellung mit einer Auflösung von 50 gewählt, beim Streudiagramm1 wurden die Punkte mit Überschüsse als eingefärbte Kreise gezeichnet, bei Streudiagramm2 als Kreuze

5 Anwendungsbeispiele

In diesem Kapitel wollen wir nun anhand einiger Beispiele aufzeigen, wie sich SGplot in der Praxis anwenden lässt, in wie weit sich die umgesetzten Techniken als nützlich erweisen und wo die Grenzen des Programms liegen. Dazu haben wir die vier folgenden Szenarien ausgesucht, um ein möglichst breites Spektrum an Anwendungsfällen abdecken zu können.

5.1 Two Moons

Bei diesem 200 Datenpunkte umfassenden Datensatz geht es darum zwei verschränkte Halbmonde von einander so zu trennen, dass jedem Mond die Hälfte der Punkte zugeordnet werden kann. Die Schwierigkeit hierbei liegt in der nicht linear verlaufenden Trennung. Im Rahmen einer Masterarbeit [Fra11] wurden Algorithmen zur Klassifikation von Datensätzen erstellt und getestet. Für die Two Moons entstanden dabei für jeden Mond ein Set aus 51 Dünnen Gittern mit 49 Punkten bei Level 4 und 129 Punkten bei Level 5 für unterschiedliche Regularisierungsparameter λ. Mithilfe der mit SGplot erstellten Animation (siehe Bildserie 5.2) können wir die Auswirkung des steigenden λ auf die Weichzeichnung der Funktion sehr schön beobachten, die Artefakte, welche den Basisfunktionen geschuldet sind (leichte Unregelmäßigkeiten im blauen Bereich), sind dank der Einfärbung schnell erkennbar. Aber auch eine Animation der zweidimensionalen Scatter Matrix bietet reichlich Aufschluss über die sich ständig verändernden Überschüsse und die sich damit ändernde Funktion, besonders wenn wir uns sowohl eine 2D-Heatmap mit überlagerten Gitterpunkten oder aber die Überschüsse als farbige Kreise zeichnen lassen (Abbildung 5.1). Die Abbildung zeigt aber auch auf, dass die Darstellungen links unten und oben rechts gespiegelt an einer gedachten Linie von unten links nach oben rechts zu einander stehen, weshalb Zusammenhänge nicht immer sofort erkennbar sind. Abhilfe kann aber die Überlagerung der Punkte und der Heatmap, wie im linken Teil dargestellt, schaffen. Die verschiedenen Überschuss-Interpolationen in Abbildung 5.3 helfen zudem dabei die Überschüsse in mehrere Bereiche zu unterteilen und lösen so ansatzweise das Problem der Unterscheidung positiver und negativer Überschüsse. Die Andrews Kurven und das Polygondiagramm sind aufgrund der geringen Dimensionalität nicht anwendbar.

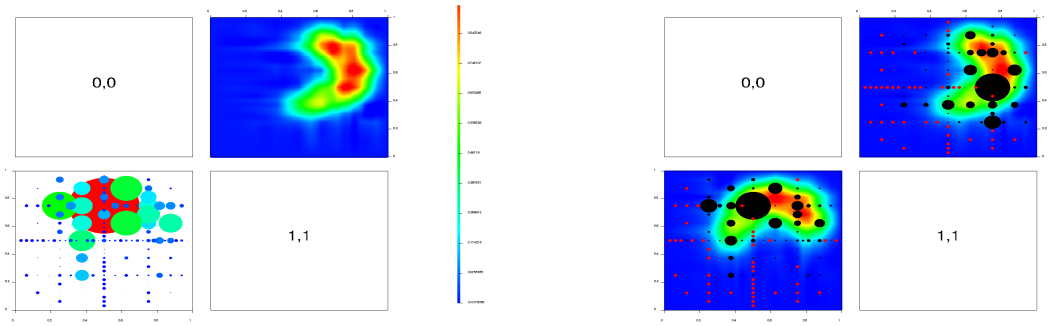

Abbildung 5.1: 2-dimensionale Streudiagramm Matrix, links mit einer 2D-Heatmap in Flächendarstellung und einem Streudiagramm der Gitterpunkte mit absoluten Überschüssen, repräsentiert als eingefärbte Kreise. Rechts die selbe Darstellung, allerdings mit überlagerten Gitterpunkten über die Heatmap

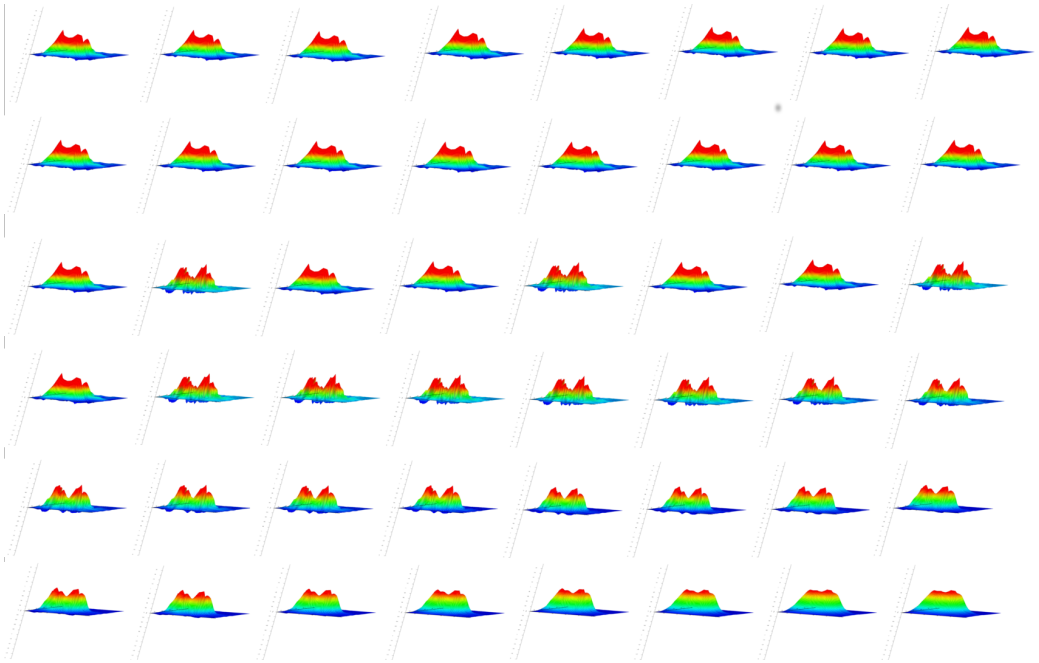

Abbildung 5.2: Bildserie der Animation über alle 51 Gitter. Die verschieden ausgeprägte Funktionsnachbildung und die wachsende Weichzeichnung sind gut zu erkennen.

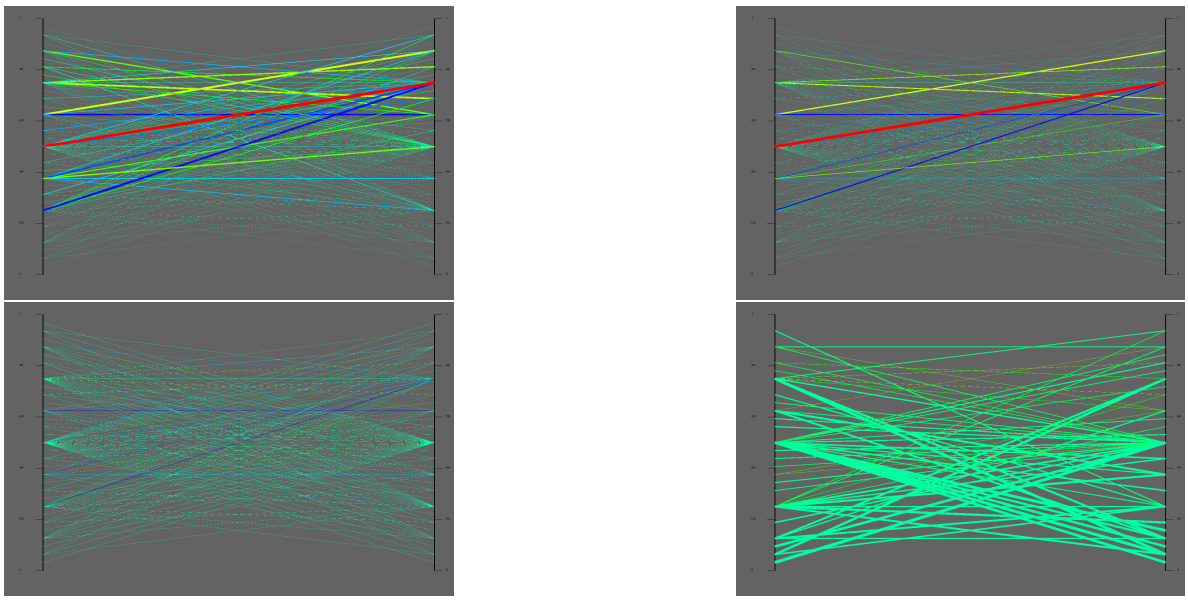

Abbildung 5.3: Parallel Koordinaten eines Moons Gitter mit linearer, quadratischer, exponentieller und logarithmischer Interpolationen

5.2 Oil Flow

Auch dieser Datensatz wurde zur Validierung von Klassifikations-Algorithmen verwendet[Fra11]. Hier handelt es sich um ein Gemisch aus Wasser, Öl und Gas in einer Pipeline, das es gilt vollständig von einander zu trennen. Speziell für die Validierung von SGplot wurde mithilfe dieses Datensatzes ein Gitter mit 12 Dimensionen und über 25.000 Gitterpunkten erstellt, um zu überprüfen, wie stabil sich das Programm auch bei mehreren Dimensionen und einer sehr großen Anzahl an Punkten verhält (siehe Abschnitt 4.3.7 Performance). Abbildung 5.4 beweist, dass die Scatter Matrix bei mehr als 10 Dimensionen aufgrund der Vielzahl an Darstellungen nur noch schwer interpretierbar ist. Nichts desto trotz lassen sich die Zusammenhänge zwischen Funktion und Gitterpunkten noch einiger Maßen gut erkennen. Detailanalysen müssen aber eine Reduzierung der dargestellten Dimensionen vorausgehen. Anders sieht es da bei den Profildiagrammen aus: die Abbildungen in 5.5 zeigen sehr schön wie mithilfe der eingefärbten Linien die verschiedenen Level deutlich hervorgehoben werden können und wie sich die Gitterpunkte über die Dimensionen verteilen. Bereiche mit größeren Überschüssen können schnell identifiziert werden, bei dem hier verwendetem Gitter beispielsweise liegen auch in höherem Level noch deutlich heraus stechende Überschüsse zwischen den Dimensionen 6 -10 vor.

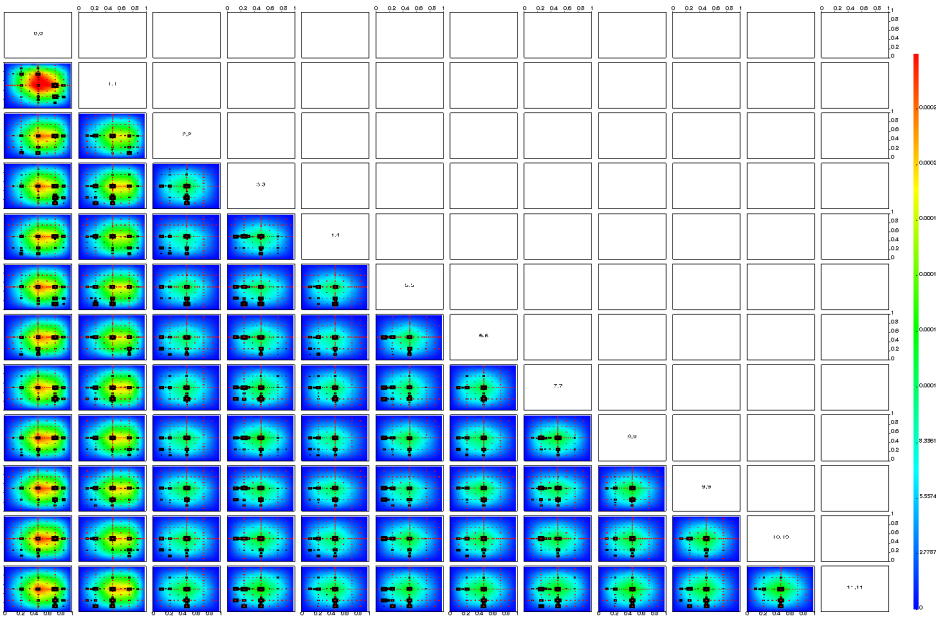

Abbildung 5.4: 12-dimensionale Scatter-Matrix, bei der durch die Einsparung einer 2. Darstellungsvariante die Übersichtlichkeit erhöht wurde. Die Größe der einzelnen Darstellungen macht eine genauere Betrachtung jedoch schwierig.

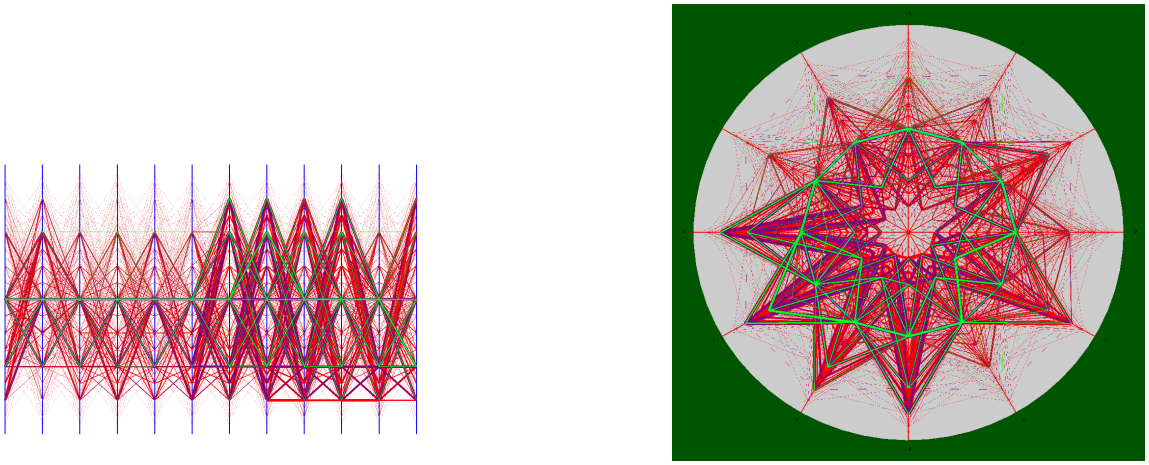

Abbildung 5.5: Parallele Koordinaten und Polygondiagramm eines 12-dimensionalen Gitters mit 25.089 Gitterpunkten, bei denen die Level zur besseren Unterscheidung unterschiedlich eingefärbt wurden

5.3 Peridynamik

Die Peridynamik ist eine partikelbasierte Methode zur Simulation von Rissentwicklung. Dieses Verfahren wurde auf ein physikalisches Experiment angewandt, bei dem eine Stahlkugel auf eine Platte aus Quarz-Glas geschossen wird. Ziel ist die Quantifizierung des Schadens über die Zeit t in Abhängigkeit unsicherer Modellparameter mit adaptiven Dünnen Gittern. Die dabei für jeden Zeitpunkt der Simulation entstehenden Gitter eignen sich gut, um zu zeigen, wie SGplot die Effektivität der Adaption von Gitterpunkten unterstützen kann. Mithilfe der Animation etwa sind die Unterschiede zwischen den verschieden verfeinerten Gittern schnell erfassbar, egal ob Scatter Matrix oder Profildiagrammen. Für die Abbildung 5.6 haben wir uns für das Gitter zum Zeitpunkt $t = 700$ entschieden, da hier die Entwicklung der Überschüsse über die Level sehr fassettenreich ist. Abbildungen 5.7 und 5.8 zeigen die Entwicklung des Gitters über die Zeit $t = 0$ - $t = 1200$, zuerst als Andrews Kurven und dann als Matrix. Bei der Matrix Darstellung können wir gut erkennen, dass mehrdimensionale Gitter zu einer Überlagerung der Punkte führen, wodurch sich die einzelnen Gitterpunkte nicht so leicht zuordnen lassen. Diesem Nachteil unterliegen die Profildiagramme nicht, bei den Andrews Kurven etwa kann jede Linie zugeordnet werden, bei sehr dichten Linien-Scharen hilft die Levelauswahl bei der Identifizierung. Besonders in der Scatter Matrix helfen die Farben dabei die positiven von den negativen Überschüssen zu unterscheiden.

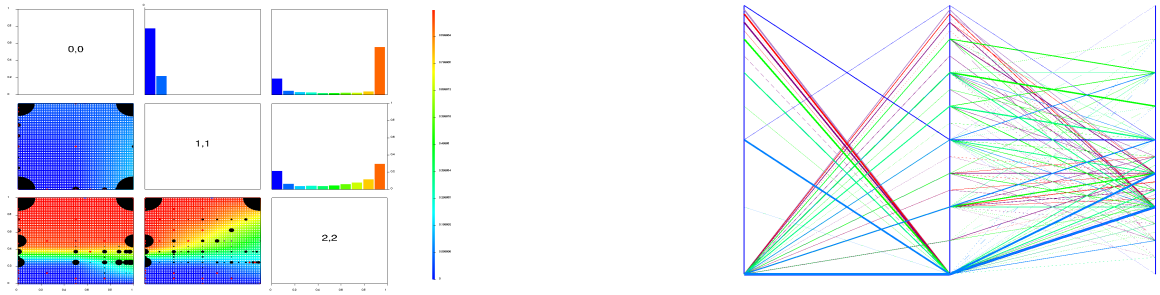

Abbildung 5.6: Gitter mit 3 Dimensionen, 6 Level und 159 Punkten, links als Scatter-Matrix mit Heatmap/Grid und Histogramm, rechts als Parallele Koordinaten. Während die Scatter Matrix einen guten Überblick über die Punkte bietet, visualisieren die Parallelen Koordinaten den überwiegenden Abfall der Überschüsse mit steigendem Level. Sehr schön zu erkennen sind auch die auffällig hohen Überschüsse selbst bei hohen Level zwischen der 1. und 2. Dimension im oberen Bereich

5.4 Analytische Funktion

Als Beispiel für ein direkt in den Code implementiertes Gitter haben wir eine analytische Funktion mit fünf Variablen gewählt,

$$f(x_0, x_1, x_2, x_3, x_4) = 16.0 \cdot (x_0 - 0.3) \cdot (x_1 - 1.0) \cdot \exp(x_1) \cdot (x_2 - 1) \cdot (x_3 - 0.5) \cdot (x_4 - 1.0);$$

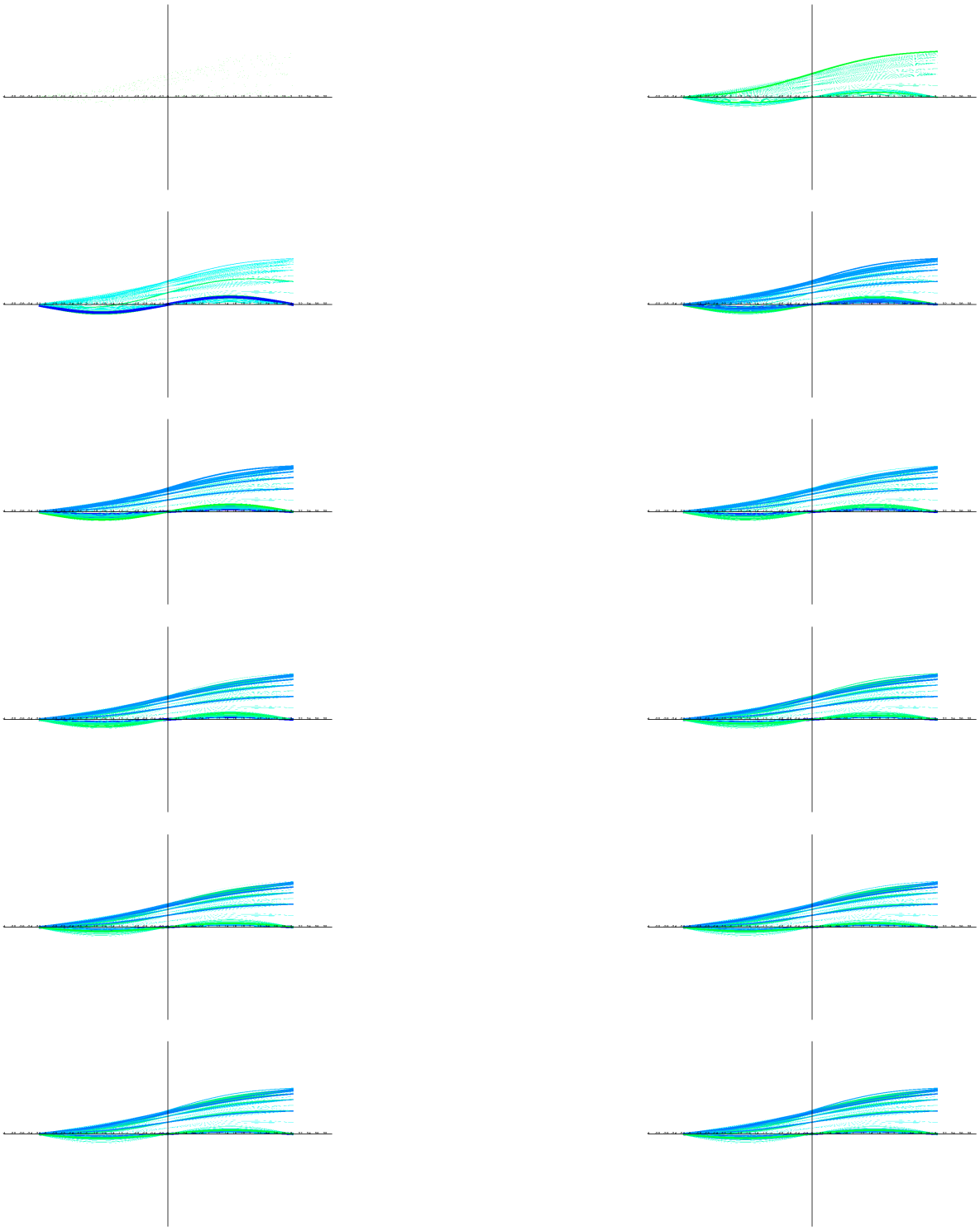

Abbildung 5.7: Animation der Peridynamik über die Zeit t als Andres Kurven

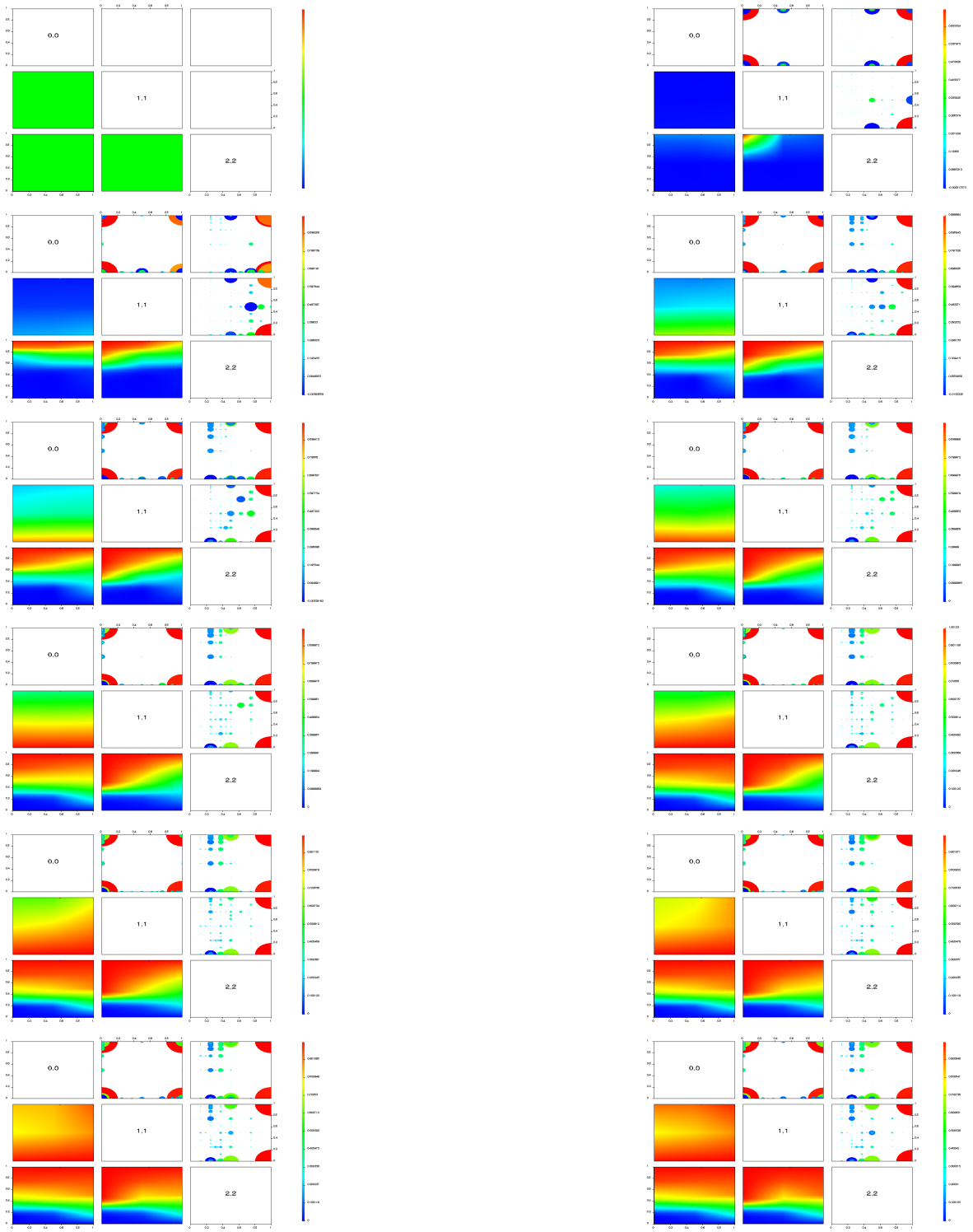

Abbildung 5.8: Animation der Peridynamik über die Zeit t als Scatter Matrix mit Heatmap und Streudiagramm. In den Streudiagrammen überlagern sich zum Teil mehrere Punkte, trotzdem sind die verschiedenen adaptierten Gitter gut voneinander zu unterscheiden

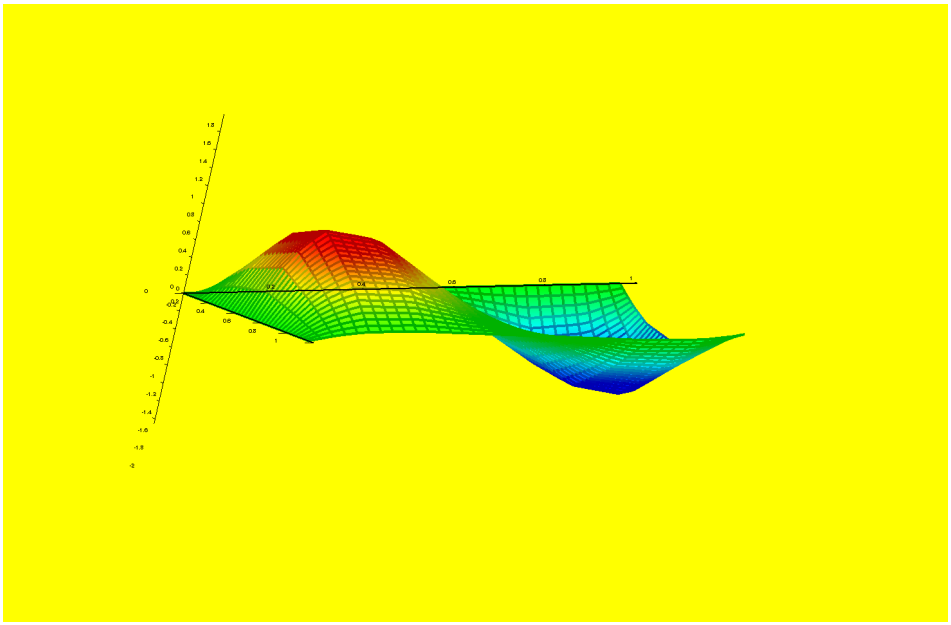

Abbildung 5.9: 12-dimensionale Scatter-Matrix, bei der durch die Einsparung einer 2. Darstellungsvariante die Übersichtlichkeit erhöht wurde. Die Größe der einzelnen Darstellungen macht eine genauere Betrachtung jedoch schwierig.

die einzelnen Komponenten sind dabei so gewählt worden, dass wir eine möglichst visuell interessante Funktion erhalten (Abbildung 5.9). Bei dem Dünnen Gitter haben wir uns für ein LinearGrid mit 5 Level entschieden. Die Erstellung des Gitters erfolgte in der `main()` Methode des Programms. Abbildung 5.10 zeigt das Gitter als fünfdimensionale Scatter-Matrix.

5.5 Hohe Dimensionalität

Zuletzt haben wir noch mithilfe des 12D Oil-Flow Gitters getestet, wie sich die Darstellungen bei einer Dimensionalität von ca. 50 verhalten. Dazu haben wir mithilfe des Dimensionsauswahl-Werkzeugs vier verschiedene Anordnungen der Dimensionen aneinandergereiht und uns das Ergebnis anzeigen lassen (Abbildungen 5.12 und 5.11). Wie erwartet liefert die Scatter Matrix lediglich ein buntes Bild auf dem nicht sonderlich viel zu erkennen ist, zudem hat die Erstellung der Grafik mehrere Minuten gedauert. Die Parallelen Koordinaten hingegen benötigten nur wenige Sekunden, liefern zwar auch eine überfüllte Grafik, trotzdem lassen sich aber Zusammenhänge und Muster erkennen, auf die eine genauere Analyse aufbauen kann. Mehr als 50 Dimensionen erscheinen aber auch mit den Parallelen Koordinaten nicht mehr darstellbar sein.

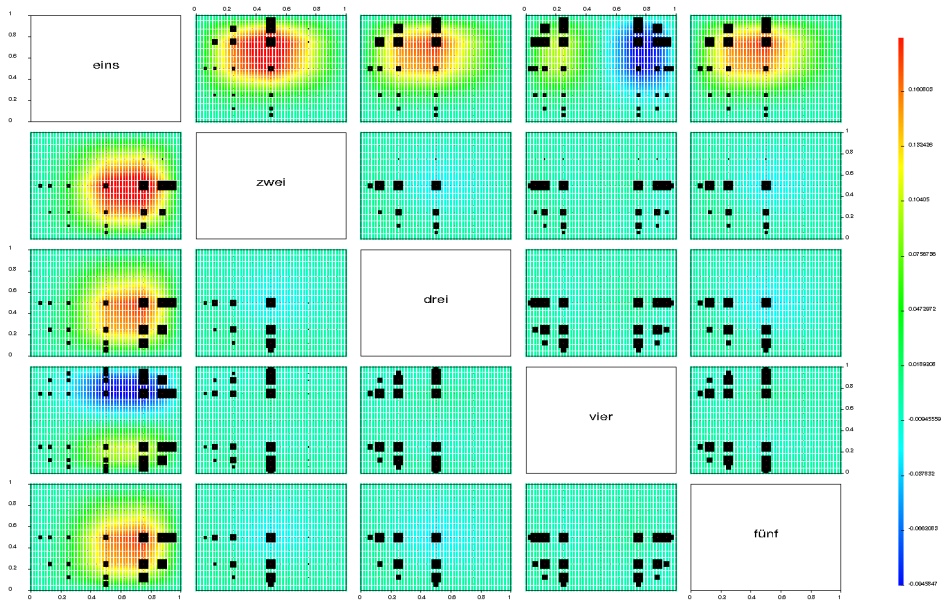

Abbildung 5.10: 5-dimensionale Scatter-Matrix eines direkt im Code erstellten Gitters. Auf die Diagonale wurde ein Beispieltext aus einer config Datei dargestellt

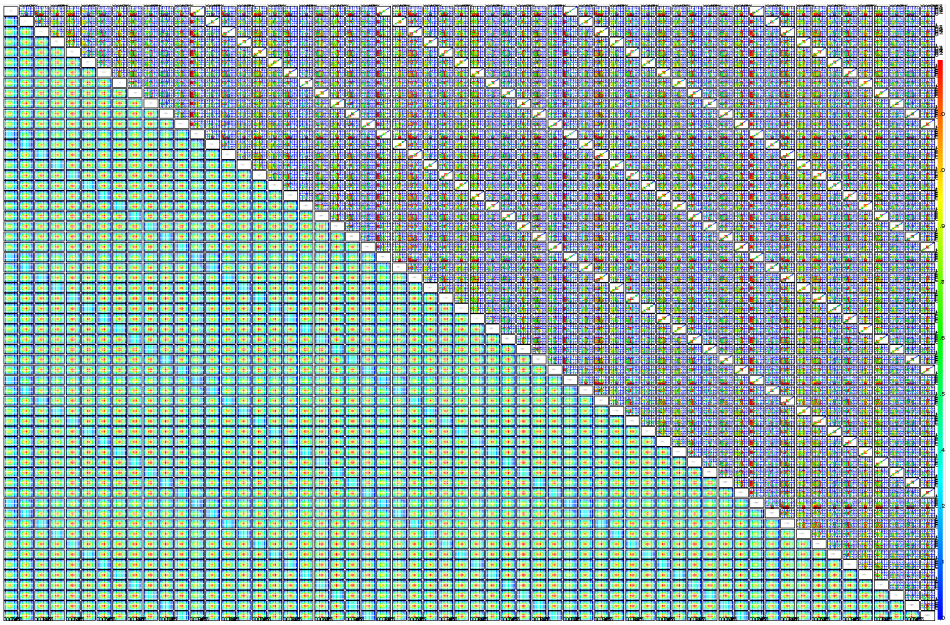

Abbildung 5.11: 48-dimensionale Scatter-Matrix, mit einer Streudiagramm Darstellung im oberen und einer Heatmap im unteren Bereich.

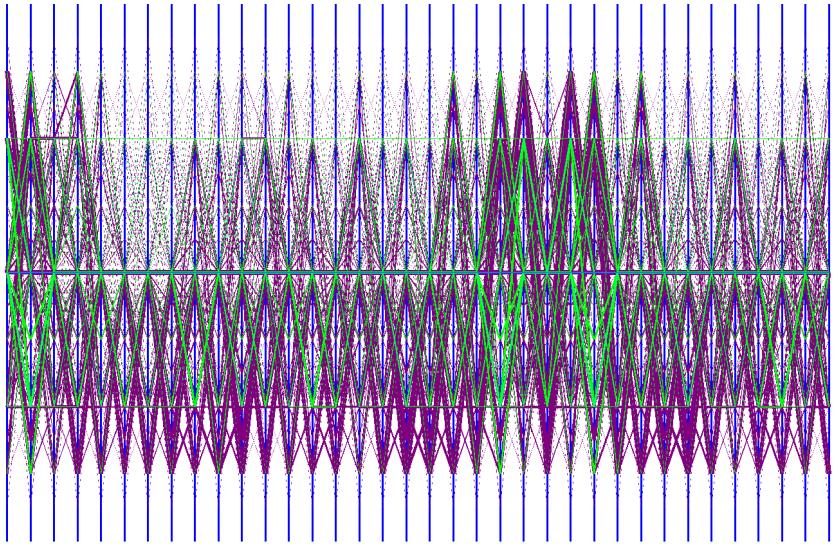

Abbildung 5.12: 48-dimensionale Parallele Koordinaten in denen trotz des geringen Abstands zwischen den Dimensionen ein Muster der Überschüsse erkennbar ist.

6 Zusammenfassung und Ausblick

Zusammenfassung

In dieser Arbeit haben wir untersucht, wie sich hochdimensionale Funktionen darstellen lassen und wie uns Dünne Gitter dabei unterstützen können. Dazu haben wir uns nach einer kurzen Einführung zu Dünnen Gittern die bedeutendsten Methoden angeschaut, mit denen sich Datensätze visualisieren lassen und haben untersucht in wie weit sie für unsere Zwecke hilfreich sein können. Als interessanteste Variante haben sich dabei neben den bewährten Streudiagramm Matrizen die Profil Kurven hervorgehoben.

Im nächsten Schritt haben wir das grafisch gestützte Visualisierungswerkzeug SGplot entwickelt, mit dem wir die meisten der zuvor vorgestellten Diagramme erzeugen und interaktiv anpassen können. Unser Hauptaugenmerk lag dabei auf der Entwicklung spezieller Analyse-Techniken zur Unterstützung von SG^{++} und den Dünnen Gittern, deren Gitterpunkte wesentlich zur Betrachtung beitragen können, aber vor allem auch auf die Echtzeit-Lauffähigkeit des Programms. Hierzu haben wir Zeit-Untersuchungen durchgeführt, um die Möglichkeiten und Grenzen von SGplot auf zu zeigen.

Im letzten Schritt haben wir verschiedene Beispiele angeführt, mit denen wir die Einsatzmöglichkeiten von SGplot demonstriert und eine Reihe von unterschiedlichen Grafiken erzeugt haben. Hier zeigten sich die Stärken und Schwächen der einzelnen Darstellungen und die Erkenntnis, dass erst verschiedene Darstellungen zusammen einen vollständigen Überblick über die Daten ermöglichen. Auch können wir festhalten, dass nicht immer alle Darstellungen anwendbar sind, so eignen sich die Andrews Kurven und das Polygon Diagramm erst ab einer Dimensionalität von drei, die Streudiagramm Matrix hingegen ist nur für bis zu zehn Dimensionen sinnvoll.

Ausblick

An diese Arbeit ließe sich an mehreren Stellen anknüpfen. Zum einen gilt es zu untersuchen, wie sich die Berechnung mehrerer Schnittebenen für die Streudiagramm Matrix beschleunigen lässt um auch bei hoher Dimensionalität längere Darstellungsverzögerungen vermeiden zu können. Daneben böte es sich an Bildsymbole wie die Chernoff-Gesichter noch einmal genauer zu betrachten und in SGplot zu implementieren, eventuell finden sich aber auch noch andere hier nicht erwähnte Darstellungen, die implementiert werden könnten.

Besonders im Hinblick auf sehr hohe Dimensionsgrößen $d > 50$ konnten bisher keine ausreichend zufrieden stellende Diagramme gefunden werden, die ohne eine Reduzierung der Dimensionsanzahl auskommen.

7 Anhang

Programm Optionen

Um das Programm auch über die Konsole steuern zu können, bietet SGplot mehrere Befehle, die beim Aufruf der Anwendung übergeben werden können. Eine Übersicht der Befehle kann mit der Hilfe Funktion angezeigt werden:

Die Eingabe von `./SGplot -help` erzeugt folgende Ausgabe:

```
Allowed options:
  --help                        produce help message

  -I [ --input-grid ] arg       [input-grid] arg : path of an input grid
  -S [ --grid-set ] arg         a folder where a grid set is located
  -P [ --plot-type ] arg        Visualization Type:
                                1 =  Function Graph 3D
                                2 =  Heatmap 3D
                                3 =  Scatter plot
                                4 =  Scatter plot matrix
                                5 =  Parallel coordinates
                                6 =  Andrew curves

  --pstyle arg                  set the point style. Possibilities: point |
                                grid | line | surface
  -a [ --surplus_on ] arg       use the surpluses for point and line diagrams
  -c [ --surplus_colorize ] arg colorize the surpluses for point and line
                                diagrams
                                works only when surplus_on is also set
```

Shortcuts

Zur schnelleren Steuerung von SGplot werden folgende Shortcuts unterstützt:

Shortcut	Funktion
Pfeiltaste hoch/runter	drehen um die x-Achse
Pfeiltaste links/rechts	drehen um die y-Achse
+/-	Zoom für 3D Plot
STRG + d	Wechsel der Ansicht zwischen 2D und 3D
STRG + s	Speichert die aktuelle Grafik
STRG + 1|2|3|4|5|6	Wechsel zwischen den verschiedenen Darstellungen

Config File

Wenn ein Config File geladen werden soll, muss diese Datei als JSON File im folgenden Format gespeichert sein:

Listing 7.1 Beispiel Config-File im JSON Format

```
{
    "Grid": {
      "dimNames": ["a", "b", "c" , "d" , "e" ],
      "matrixEntries": ["eins","zwei","drei","vier","fnf"]
    },
    "Set": {
      "filenames": ["a.grid", "b.grid", "c.grid"],
      "paramValues": [1, 2, 3],
      "paramName": "Time"
    }
}
```

Literaturverzeichnis

[And72] D. F. Andrews. Plots of high-dimensional data. *Biometrics*, S. 125–136, 1972. (Zitiert auf Seite 22)

[Asi85] D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM Journal on Scientific and Statistical Computing*, 6(1):128–143, 1985. (Zitiert auf Seite 27)

[Bel57] R. Bellman. *Dynamic Programming*. Princton University Press, 1957. (Zitiert auf Seite 13)

[BF93] C. Beshers, S. Feiner. AutoVisual: Rule-based design of interactive multivariate visualizations. *IEEE Computer Graphics and Applications*, S. 41–49, 1993. (Zitiert auf den Seiten 4, 18 und 19)

[BFC98] C. Brunsdon, A. Fotheringham, M. Charlton. An investigation of methods for visualising highly multivariate datasets. *Case Studies of Visualization in the Social Sciences, Joint Information Systems Committee/ESRC*, S. 55–80, 1998. (Zitiert auf Seite 17)

[BG04] H.-J. Bungartz, M. Griebel. Sparse Grids. *Acta Numerica*, (13):1–123, 2004. (Zitiert auf den Seiten 9 und 13)

[Cha06] W. W.-Y. Chan. A survey on multivariate data visualization. 2006. (Zitiert auf Seite 17)

[Che73] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973. (Zitiert auf den Seiten 4, 24 und 26)

[Deg06] H. Degen. Statistische Methoden zur visuellen Exploration mehrdimensionaler Daten. In *Analytische Informationssysteme*, S. 305–326. Springer, 2006. (Zitiert auf Seite 17)

[FR81] B. Flury, H. Riedwyl. Graphical representation of multivariate data by means of asymmetrical faces. *Journal of the American Statistical Association*, 76(376):757–765, 1981. (Zitiert auf Seite 24)

[Fra11] F. Franzelin. *Classification with Estimated Densities on Sparse Grids*. Diplomarbeit, Technische Universit{ä}t M{ü}nchen, 2011. (Zitiert auf den Seiten 49 und 51)

[FT74] J. H. Friedman, J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *Computers, IEEE Transactions on*, 100(9):881–890, 1974. (Zitiert auf Seite 26)

[FWR99] Y.-H. Fua, M. O. Ward, E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *Proceedings of the conference on Visualization'99: celebrating ten years*, S. 43–50. IEEE Computer Society Press, 1999. (Zitiert auf Seite 24)

[Gar11] J. Garcke. Sparse Grid Tutorial, 2011. (Zitiert auf Seite 9)

[HLD02] H. Hauser, F. Ledermann, H. Doleisch. Angular brushing of extended parallel coordinates. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, S. 127–130. IEEE, 2002. (Zitiert auf Seite 24)

[ID90] A. Inselberg, B. Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proceedings of the First IEEE Conference on Visualization*, S. 361–378. 1990. doi:10.1109/VISUAL.1990.146402. (Zitiert auf Seite 23)

[Pfl05] D. Pflüger. Data Mining mit Dünnen Gittern. 2005. (Zitiert auf Seite 10)

[Pfl10] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Dissertation, Technische Universit{ä}t M{ü}nchen, 2010. (Zitiert auf den Seiten 4, 7, 9, 12, 13, 14 und 15)

[PK08] J. E. Peter Kaiser. *Python - Das umfassende Handbuch*. Galileo Computing, 2008. (Zitiert auf Seite 31)

[Sch08] R. Schnell. *Graphisch gestützte Datenanalyse*. 2008. (Zitiert auf Seite 17)

[Sco09] D. W. Scott. *Multivariate density estimation: theory, practice, and visualization*, Band 383. Wiley. com, 2009. (Zitiert auf Seite 27)

[Smo63] S. Smolyak. Quadrature and interpolation formulas for Tensor Products of Certain Classes of Functions. *Dokl. Akad. Nauk SSSR, Russian, Engl. Transl.: Soviet Math. Dokl.*, S. 1042–1043, 1963. (Zitiert auf Seite 9)

[Tuk77] J. W. Tukey. Exploratory Data Analysis. 1977. (Zitiert auf Seite 17)

[Ull] C. Ullenboom. *Java 7 – Mehr als eine Insel*. Galileo Computing. (Zitiert auf Seite 30)

[Weg90] E. J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990. (Zitiert auf Seite 24)

Alle URLs wurden zuletzt am 15. 12. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift