

Technische Universität München

ZENTRUM MATHEMATIK

**Globale Optimierung unter
Nebenbedingungen mit dünnen Gittern**

Diplomarbeit
von
Izabella Ferenczi

Themensteller: Prof. Dr. Rudi Zagst, Dr. Ralf Werner
Betreuer: Dr. Ralf Werner
Abgabetermin: 01.10.2005

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbstständig und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe.

München, den 30. September 2005

Izabella Ferenczi

Danksagung

Bei Herrn Prof. Dr. Rudi Zagst möchte ich mich herzlich für die Ermöglichung dieser Diplomarbeit bedanken.

Für die hervorragende Betreuung danke ich Herrn Dr. Ralf Werner. Ohne seine Unterstützung und Ratschläge wäre diese Diplomarbeit nicht möglich gewesen.

Darüberhinaus möchte ich Katrin Schöttle, Franz Fuchs und Michaela Tempes für ihre Unterstützung danken.

Inhaltsverzeichnis

1 Globale Optimierung	1
1.1 Einleitung	1
1.2 Grundlagen	3
2 Globale Optimierung mittels Hyperbolic Cross Points	4
2.1 Einleitung	4
2.2 Der HCP-Algorithmus	4
2.3 Der Algorithmus von Horn	13
3 Dünne Gitter	16
3.1 Einleitung	16
3.2 Hierarchische Teilraumzerlegung	18
3.3 Die Kombinationsmethode	28
3.4 Lösung von Optimierungsproblemen auf dünnen Gittern	29
3.5 Erzeugung dünner Gitter	30
3.6 Äquidistante Gitter im eindimensionalen Fall	31
3.6.1 Das Maximumgitter	31
3.6.2 Das Noboundary-Gitter	33
3.6.3 Das Clenshaw-Curtis-Gitter	35
3.7 Gitter mit variablen Schrittweiten	37
3.7.1 Das Gauß-Legendre-Gitter	37
3.7.2 Das Tschebyscheff-Gitter	40
3.7.3 Das kubische Gauß-Legendre-Gitter	42
3.8 Dünne Gitter und der HCP-Algorithmus	45
3.9 Konvergenz der Optimierung auf dünnen Gittern	46
3.10 Numerischer Vergleich der Gittertypen	48
4 Globale Optimierung mit dünnen Gittern	52
4.1 Einleitung	52
4.2 Optimierung auf einem gegebenen Gitter	52
4.3 Dünne Gitter als Bäume	57
4.3.1 Bäume	57
4.3.2 Eindimensionale dünne Gitter als Bäume	58
4.3.3 Mehrdimensionale Gitter	61
4.4 Iterative Gittererzeugung	63
4.5 Adaptive Optimierung auf dünnen Gittern	69
4.6 Numerischer Vergleich der Gittertypen	73

5 Cluster Methoden und Cluster Analyse	75
5.1 Motivation	75
5.2 Clusteranalyse	75
5.2.1 Fuzzy C-Means	77
5.2.2 Eine subtraktive Clustermethode: <code>cluster_estimation</code>	79
5.3 Numerische Tests und Beispiele	81
5.3.1 FCM-Algorithmus	81
5.3.2 Vergleich von <code>subclust</code> und <code>cluster_estimation</code>	83
5.3.3 Fazit	85
6 Lokale Optimierung	86
6.1 Einleitung	86
6.2 Der Algorithmus von Nelder und Mead	86
6.2.1 Konvergenz des Nelder-Mead Algorithmus	89
6.2.2 Das Gegenbeispiel von McKinnon	91
6.2.3 Numerische Tests	92
7 Nichtlineare Optimierung unter Nebenbedingungen	94
7.1 Einleitung	94
7.2 Optimalitätsbedingungen	94
7.3 Das klassische Strafverfahren	95
7.4 Exakte Penalty-Verfahren	98
7.5 Das Barrieverfahren	99
7.6 Multiplikator-Methoden	100
7.7 Penalty-Barrier-Multiplier Methoden (PBM)	102
7.8 Beispiele	105
8 Implementierung in Matlab und numerische Ergebnisse	109
8.1 Übersicht über die implementierten Routinen	109
8.1.1 HCP-Algorithmus	109
8.1.2 Rekursive dünne Gitter-Berechnung	110
8.1.3 Globale Optimierung auf dünnen Gittern	111
8.2 Numerische Ergebnisse	114
8.2.1 Testfunktionen	115
8.2.2 Numerische Ergebnisse	124
A Abkürzungs- und Symbolverzeichniss	127
A.1 Allgemein gebräuchliche Abkürzungen	127
A.2 In dieser Diplomarbeit verwendete Abkürzungen	128
Abbildungsverzeichnis	129
Literaturverzeichnis	131

Kapitel 1

Globale Optimierung

1.1 Einleitung

Optimierung findet in vielen unterschiedlichen Bereichen der Industrie und Wissenschaft Anwendung. In der Wirtschaft geht es häufig um die Gewinnmaximierung, bzw. um die Maximierung von Kapitalerträgen, z. B. bei der *Portfoliooptimierung*. In vielen technischen Zweigen muss eine Parameteroptimierung durchgeführt werden, um die Ergebnisse eines parameterabhängigen Algorithmus an reale Messwerte anzupassen. Solche Probleme gehören in den Bereich der *nichtlinearen Regressionsrechnung*. Von zunehmender Wichtigkeit sind auch Optimierungsprobleme im Bereich *data mining*. Data mining bezeichnet die Klassifikation von Daten auf Basis von Lernmustern, z. B. in der Krebsprognose, siehe auch [GG01].

In dieser Diplomarbeit beschäftigen wir uns mit nichtlinearen Optimierungsproblemen. Die Schwierigkeit besteht hier darin, dass oftmals viele lokale und globale Minima existieren. Die bislang gängigen numerischen Verfahren zur nichtlinearen Optimierung garantieren aber nur die Konvergenz zu einem lokalen Minimum. In den letzten Jahren gab es auch viele unterschiedliche moderne Ansätze zur globalen Optimierung. Vor allem evolutionäre Algorithmen erfreuen sich zunehmender Beliebtheit. Traditionellere Ansätze sind stochastische Verfahren, Clustermethoden, Intervallmethoden und verallgemeinerte Abstiegsverfahren, siehe auch [TZ89].

Behandelt man Optimierungsprobleme aus der Praxis, hat man häufig das Problem, dass man keine analytischen Ableitungen zur Verfügung hat und es auch sehr schwierig ist die Ableitung numerisch zu approximieren. Hinzu kommt, dass eine Funktionsauswertung oftmals viel Rechenzeit benötigt. Dies kann z. B. der Fall sein, wenn bei der Auswertung der Zielfunktion mehrere Integrale berechnet werden müssen, oder z. B. ein Experiment durchgeführt werden muss.

In der vorliegenden Arbeit wird deshalb ein ableitungsfreies, globales Optimierungsverfahren entwickelt, welches nur mit Hilfe von Funktionswerten versucht das globale Minimum einer nichtlinearen Zielfunktion zu finden. Das bedeutet insbesondere, dass man keine Ableitung der Zielfunktion benötigt und die Ableitung auch nicht numerisch angenähert werden muss. Ein weiteres Ziel ist, dass das Verfahren generell wenig Funktionsauswertungen benötigt um in den Einzugsbereich des globalen Minimums zu gelangen. Eine höhere Genauigkeit der Lösung wird dann mit Hilfe eines geeigneten lokalen Verfahrens erreicht.

Im Anschluss an die Einleitung wird in diesem Kapitel die allgemeine Problemstellung formuliert und es werden einige grundlegende Annahmen, die für die restliche Arbeit von Bedeutung sind, getroffen.

Die Motivation für den in dieser Arbeit entwickelten Algorithmus zur globalen Optimierung beruht auf der Veröffentlichung *Global Optimization Using Hyperbolic Cross Points* von Novak und Ritter aus dem Jahr 1996, siehe [NR96a], und wird im zweiten Kapitel eingehend präsentiert.

Anschließend wird das Vorgehen von Novak und Ritter verallgemeinert, wozu im dritten Kapitel eine Einführung der Theorie der *dünnen Gitter* notwendig ist. Dazu gehört auch ein Verfahren zur Erzeugung dünner Gitter, sowie die Definitionen der verschiedenen, implementierten Gittertypen.

Im vierten Kapitel erfolgt die Einführung einer alternativen Datenstruktur für dünne Gitter, aus der anschließend der eigentliche Algorithmus zur globalen Optimierung resultiert. Dieser ist dem Algorithmus von Novak und Ritter sehr ähnlich, stützt sich aber stärker auf die Theorie dünner Gitter und ermöglicht insbesondere die Wahl verschiedener Gittertypen als Grundlage. Hier erweist sich in den meisten Fällen das Noboundary-Gitter als am besten geeignet.

Da der im vierten Kapitel beschriebene Algorithmus adaptiv arbeitet und dadurch mehr Punkte in den Gebieten erzeugt, in denen das globale Minimum vermutet wird, schließt sich im fünften Kapitel eine Clusteranalyse der Gitterpunkte an. Wir gehen vor allem auf *Fuzzy-C-Means Clustering*, kurz *FCM*, und auf eine subtraktive Clustermethode ein. In den Minima der so identifizierten Cluster kann man anschließend ein lokales Verfahren starten.

Als lokales Verfahren wählen wir den *Algorithmus von Nelder und Mead* aus dem Jahr 1965 und stellen ihn im sechsten Kapitel eingehend vor. Dabei gehen wir auch auf die Beispiele von McKinnon aus dem Jahr 1998 ein, die zeigen, dass der Algorithmus von Nelder und Mead auch bei relativ einfachen, zweidimensionalen Problemen zu einem nichtstationären Punkt konvergieren kann. Trotz dieser Schwäche arbeitet der Algorithmus für die meisten Probleme zuverlässig.

Damit die bislang beschriebenen Algorithmen auch nichtlineare Ungleichungsnebenbedingungen berücksichtigen können, müssen sie im siebten Kapitel dementsprechend ergänzt werden. Dies geschieht beim ersten Teil des globalen Algorithmus durch die Einführung einer *exakten Penaltyfunktion*. Der zweite Teil, die lokale Suche, muss dagegen anders behandelt werden. Hier haben wir uns für den *PBM-Algorithmus* entschieden und optimieren statt der Zielfunktion die erweiterte Lagrangefunktion. Hierzu wird zunächst die Theorie zu Penalty-, Barriere- und Multiplikatormethoden eingeführt und anschließend die Funktionsweise des PBM-Algorithmus erläutert.

Im letzten Kapitel werden die implementierten Algorithmen eingehend getestet und die benötigten Rechenzeiten angegeben.

1.2 Grundlagen

Zunächst wollen wir ein allgemeines nichtlineares Optimierungsproblem formulieren:

Definition 1.1 Nichtlineares Minimierungsproblem

Sei $S = [a, b] \subset \mathbb{R}^d$ und seien $f : S \rightarrow \mathbb{R}$ und $g : S \rightarrow \mathbb{R}^m$ stetige Funktionen. Dann kann ein nichtlineares Optimierungsproblem mit Ungleichungsnebenbedingungen geschrieben werden als:

$$\begin{aligned} \min_{x \in S} \quad & f(x) \\ \text{s.t. } & g(x) \leq 0. \end{aligned} \tag{GP}$$

S ist eine Obermenge des zulässigen Bereiches Z ,

$$Z = \{x \in S : g(x) \leq 0\}.$$

Ein Punkt $x^* \in Z$ heißt globaler Minimalpunkt von (GP), wenn gilt:

$$f(x^*) \leq f(x) \quad \forall x \in Z.$$

Ist diese Bedingung nur für zulässige Punkte aus einer Umgebung U von x^* erfüllt, dann ist x^* ein lokaler Minimalpunkt von (GP).

Durch die Einschränkung auf stetige Funktionen auf einem abgeschlossenen und beschränkten, also kompakten, Gebiet, ist die Existenz einer Lösung von (GP), für $Z \neq \emptyset$, nach dem Satz von Weierstraß sichergestellt, siehe [Br01].

Affine Transformation

Die Menge $S = [a, b] \subset \mathbb{R}^d$ ist ein d -dimensionaler Quader. Ab hier betrachten wir nur noch den Quader $[-0.5, 0.5]^d$, dies stellt keine Einschränkung dar, da $[a, b]$ durch eine affine Transformation T problemlos in $[-0.5, 0.5]^d$ überführt werden kann:

$$T : [a, b] \longrightarrow [-0.5, 0.5]^d \text{ mit } T(x) = \frac{1}{b-a}x - 0.5 - \frac{a}{b-a}; \tag{1.1}$$

$$T^{-1} : [-0.5, 0.5]^d \longrightarrow [a, b] \text{ mit } T^{-1}(x) = (b-a)x + 0.5(b+a). \tag{1.2}$$

Da wir im weiteren Verlauf immer wieder Transformationen dieser Art benötigen werden, bezeichnen wir diese kurz mit $T_{[a,b] \rightarrow [u,v]}$, wobei $[a, b]$ die Definitionsmenge und $[u, v]$ die Wertemenge bezeichnet. Dann gilt:

$$T_{[a,b] \rightarrow [u,v]}(x) = \frac{v-u}{b-a}x + u - \frac{v-u}{b-a}a.$$

Zur Auswertung der Zielfunktion muss $x \in [-0.5, 0.5]^d$ durch die Anwendung von T^{-1} auf $[a, b]$ transformiert werden.

Kapitel 2

Globale Optimierung mittels Hyperbolic Cross Points

2.1 Einleitung

Das im Folgenden vorgestellte Verfahren zur globalen Optimierung basiert auf dem Artikel *Global Optimization Using Hyperbolic Cross Points* von Erich Novak und Klaus Ritter aus dem Jahr 1996, siehe [NR96a].

Das Charakteristische an diesem Verfahren ist, dass als Information über die zu minimierende Funktion lediglich Funktionswerte an speziell ausgewählten Punkten herangezogen werden, es handelt sich damit um eine *direct search* Methode. Ziel des Verfahrens ist das globale Minimum mit möglichst wenig Funktionsauswertungen zu finden, da diese in der Praxis oftmals sehr teuer sind.

2.2 Der HCP-Algorithmus

Der Algorithmus geht iterativ vor und benutzt jeweils die Information über die bislang berechneten Punkte, um neue Punkte zu bestimmen. Die betrachteten Punkte sind so genannte Hyperbolic Cross Points:

Definition 2.1 *Hyperbolic Cross Points [HCP]*

Ein Punkt $x \in [-0.5, 0.5]^d$ heißt *Hyperbolic Cross Point*, oder kurz *HCP*, wenn die Koordinaten x_i des Punktes x folgende Form haben:

$$x_i = \pm \sum_{j=1}^k a_j 2^{-j}, \quad \text{mit } a_j \in \{0, 1\}.$$

Für $x_i \neq 0$ muss $a_k = 1$ sein.

Man nennt Koordinaten dieser Form auch *dyadische Koordinaten*.

Definition 2.2 *Level eines HCP*

Sei x ein Hyperbolic Cross Point. Dann ist die Zahl k (aus Definition 2.1) der Level der jeweiligen Koordinate. Der Level von x ist die Summe der Level der Koordinaten:

$$\text{Level}(x) = \sum_{i=1}^d \text{Level}(x_i).$$

Beispiel 2.3 Berechnung des Level über die Summe der Level der Koordinaten:

1. $\text{Level}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \text{Level}(0) + \text{Level}(0) = 0.$
2. $\text{Level}\left(\begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix}\right) = \text{Level}(0.25) + \text{Level}(0.5) = 2 + 1 = 3.$
3. $\text{Level}\left(\begin{bmatrix} 0.375 \\ 0.25 \\ 0 \end{bmatrix}\right) = \text{Level}(0.375) + \text{Level}(0.25) + \text{Level}(0) = 5.$

Mit Hilfe dieser Vorbemerkungen lässt sich ein erster, nichtadaptiver Algorithmus formulieren:

Algorithmus 2.4 nichtadaptiver HCP-Algorithmus

1. Bestimme $X = \{x \in [-0.5, 0.5]^d : x \text{ ist HCP mit } \text{Level}(x) \leq k\};$
2. Berechne $F = f(X);$
3. Bestimme x^* mit $f(x^*) = \min(F).$

Für die Werte $k = 6, 7$ und 8 bestimmt Algorithmus 2.4 folgende Punkte:

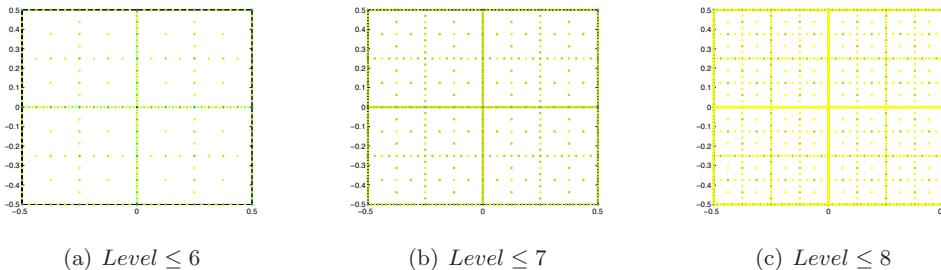


Abbildung 2.1: Hyperbolic Cross Points im \mathbb{R}^2 .

Die Farbe verdeutlicht den Level der Punkte: je heller die Farbe, desto höher der Level.

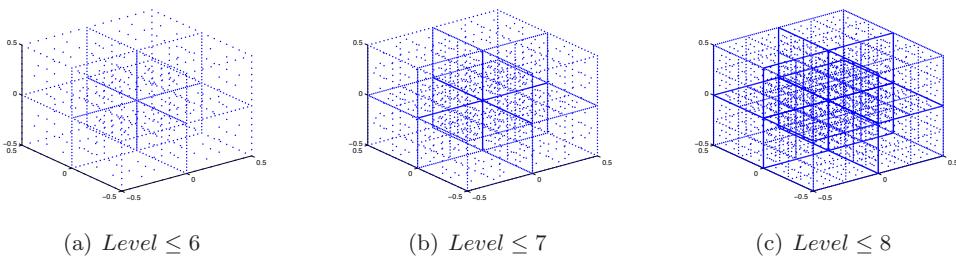


Abbildung 2.2: Hyperbolic Cross Points im \mathbb{R}^3 .

Mit einigen zusätzlichen Definitionen kann man Algorithmus 2.4 so erweitern, dass er adaptiv abläuft. Der erste Begriff, der benötigt wird, ist der des *Nachbarn* eines HCP x .

Definition 2.5 Nachbar der Stufe m eines HCP

Ein Punkt $y \in [-0.5, 0.5]^d$ heißt *Nachbar der Stufe m eines HCP x* , wenn gilt:

$$\begin{aligned} y_i &= x_i \pm 2^{-\text{level}(x_i)-m} && \text{für ein } i \in \{1, \dots, d\} \\ y_j &= x_j && \text{für alle } j \neq i. \end{aligned}$$

Ein Nachbar der Stufe m eines Punktes mit Level k hat Level $k + m$.

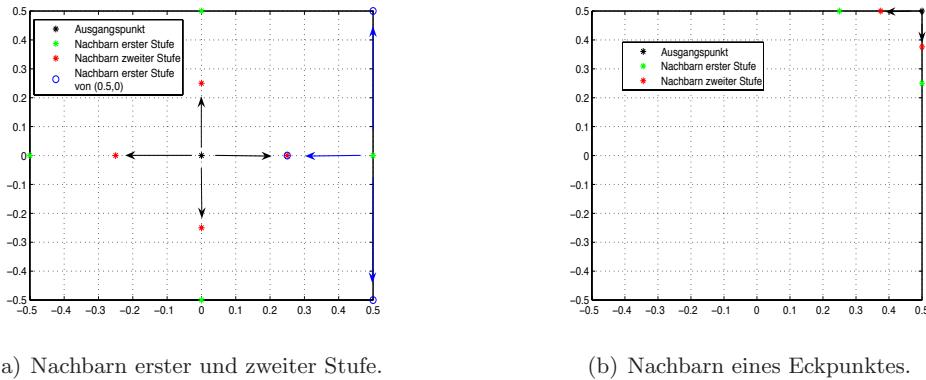
Beispiel 2.6 Beispiele für Nachbarn sind:

1. Die Nachbarn der Stufe 1 von $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ sind die Punkte
 $\begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}, \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -0.5 \end{bmatrix}.$
2. Die Nachbarn der Stufe 2 von $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ sind die Punkte
 $\begin{bmatrix} 0.25 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0.25 \end{bmatrix}, \begin{bmatrix} -0.25 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -0.25 \end{bmatrix}.$
3. Die Nachbarn der Stufe 1 von $\begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$ sind die Punkte
 $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 0 \end{bmatrix}.$
4. Die Nachbarn der Stufe 1 des Eckpunkts $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ sind
 $\begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix}.$
5. Die Nachbarn der Stufe 2 von $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ sind
 $\begin{bmatrix} 0.375 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 0.375 \end{bmatrix}.$

Definition 2.7 Rang eines HCP

Sei $X = \{x_0, x_1, \dots, x_n\}$ die Menge der betrachteten Punkte und $F = \{f(x_0), f(x_1), \dots, f(x_n)\}$ die Menge der zugehörigen Funktionswerte. Dann ist der Rang eines Punktes x aus der Menge X definiert als:

$$\text{Rang}(x) = \#\{y \in F : y \leq f(x)\}.$$

Abbildung 2.3: Beispiele für Nachbarn im \mathbb{R}^2 .**Definition 2.8 Grad eines HCP**

In jeder Iteration des weiter unten beschriebenen Algorithmus 2.10 werden alle bis dahin berechneten Punkte bewertet und der „beste“ Punkt wird ausgewählt. Jedem Punkt $x \in X$ wird die Zahl $Grad(x)$ zugeordnet, die angibt wie oft er schon als bester Punkt ausgewählt wurde.

Definition 2.9 Bewertungsfunktion

Die Funktion $\beta_\alpha : \mathbb{R}^d \rightarrow \mathbb{R}$ mit

$$\beta_\alpha(x) = (Level(x) + Grad(x))^\alpha Rang(x)^{1-\alpha} \text{ mit } \alpha \in [0, 1]$$

heißt Bewertungsfunktion, siehe [NR96a]. Punkte mit einem niedrigen Wert der Bewertungsfunktion werden bevorzugt.

Insgesamt kann mit den bisher eingeführten Begriffen folgender Algorithmus beschrieben werden:

Algorithmus 2.10 HCP-Algorithmus

1. Input: $k \in \mathbb{N}$, der maximale Level aller Punkte,
Adaptivitätsparameter: $\alpha \in [0, 1]$;
2. Startpunkt ist $\tilde{x} = 0$ mit $Level(\tilde{x}) = Grad(\tilde{x}) = 0$,
 $Rang(\tilde{x}) = 1$, $X = \{\tilde{x}\}$, $F = \{f(\tilde{x})\}$;
3. $Grad(\tilde{x}) = Grad(\tilde{x}) + 1$;
4. Berechne alle Nachbarn X_{neu} von \tilde{x} der Stufe $Grad(\tilde{x})$;
5. Für alle Punkte $x \in X_{neu}$ gilt: $Level(x) = Level(\tilde{x}) + Grad(x)$;
6. Setze $X = X \cup \{X_{neu}\}$, $F = F \cup \{f(X_{neu})\}$;
7. Bestimme $Rang(x)$ für alle $x \in X$;
8. Bewerte alle $x \in X$ mit der Bewertungsfunktion $\beta_\alpha(x)$,
siehe Definition 2.9;

9. Wähle einen Punkt \tilde{x} mit $\tilde{x} = \arg \min_{x \in X} (\beta_\alpha(x))$;
10. (a) Falls $\max_{x \in X} \text{level}(x) < k$:
gehe zu 3;
(b) Falls der maximale Level erreicht ist:
Lösung ist ein x_{\min} mit $x_{\min} = \arg \min_{x \in X} (f(x))$.

Der Parameter $\alpha \in [0, 1]$ steuert die Adaptivität des Verfahrens: Für $\alpha = 1$ fällt der Rang nicht ins Gewicht und man erhält ein nichtadaptives Verfahren, welches die Punkte gemäß ihrem Level und Grad auswählt und in jeder Iteration alle Nachbarn erzeugt, so dass ein gleichmäßiges Gitter entsteht. Je kleiner α ist, desto stärker fällt der Rang ins Gewicht, desto „adaptiver“ wird das Verfahren.

Abbildungen 2.4 und 2.5 zeigen den Einfluss verschiedener Werte von α anhand zweier Testbeispiele. Die verwendeten Testfunktionen werden im 8. Kapitel genau vorgestellt. Bei dem ersten Beispiel handelt es sich um die Funktion $BR : [a, b] \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, bei dem zweiten um die Funktion $Hn3 : [a, b] \subset \mathbb{R}^3 \rightarrow \mathbb{R}$, siehe auch [TZ89]. Im zweidimensionalen Fall, siehe Abbildung 2.4, sieht man die Höhenlinien der Zielfunktion BR und die Punkte, die der HCP-Algorithmus für dieses Problem verwendet. Diese Zielfunktion hat drei globale Minima (in den dunkelblauen Gebieten).

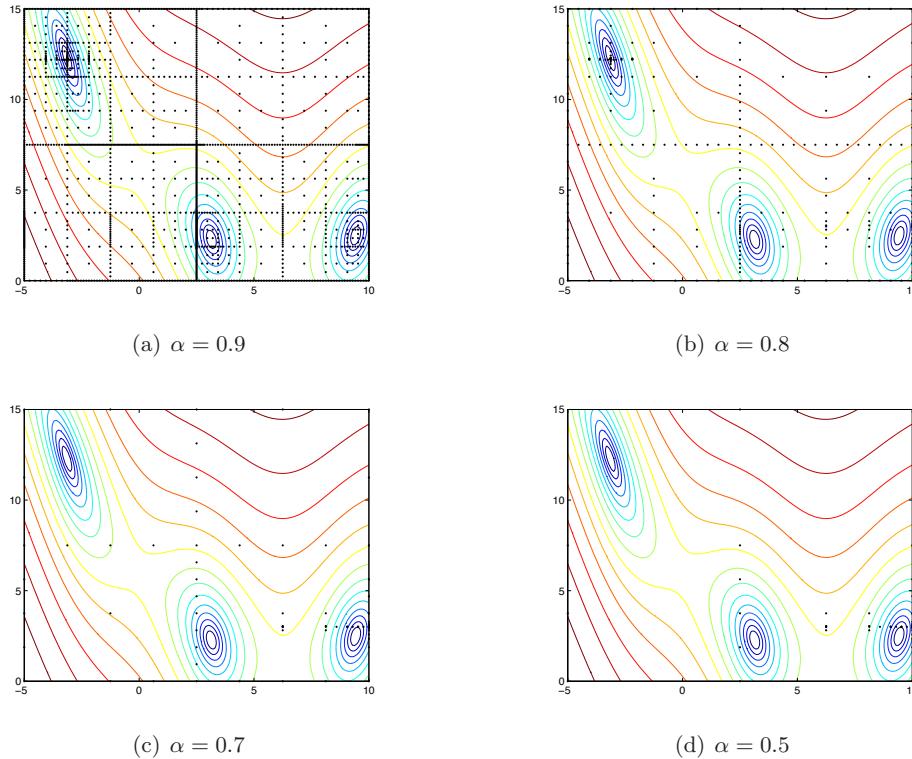


Abbildung 2.4: HCP-Algorithmus für verschiedene α mit $k = 15$.

Zur Darstellung einer Funktion $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ wurde in Abbildung 2.5 die Methode gewählt, den verwendeten Punkten eine Farbe zuzuordnen, die dem Funktionswert

entspricht. Wieder steht blau für niedrige und rot für hohe Funktionswerte.

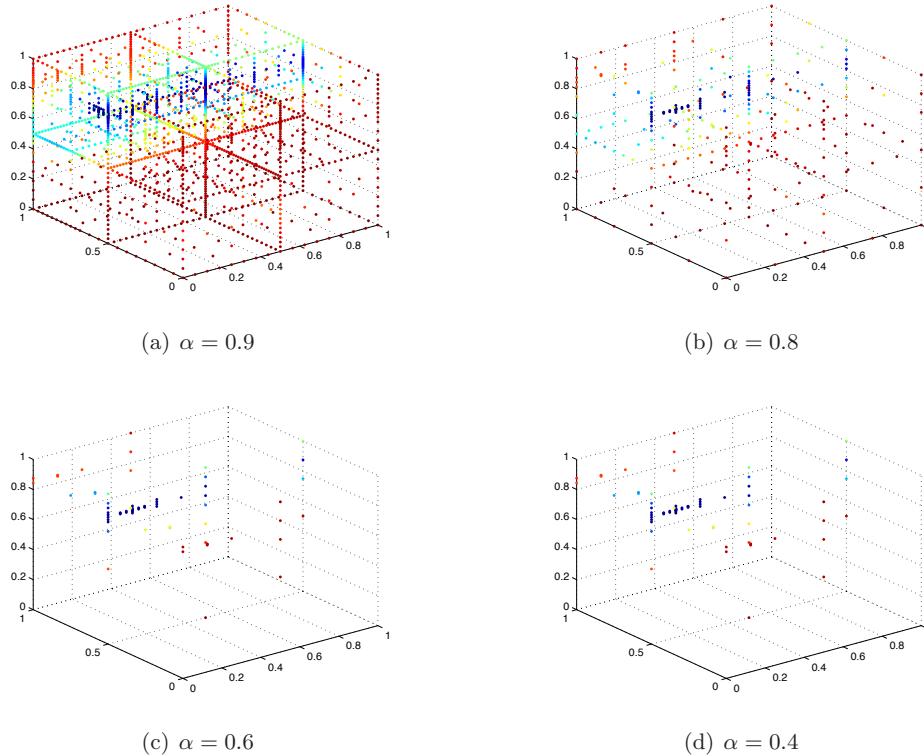


Abbildung 2.5: HCP-Algorithmus für verschiedene α mit $k = 12$.

Die Bewertungsfunktion

$$\beta_\alpha(x) = (\text{Level}(x) + \text{Grad}(x))^\alpha \text{Rang}(x)^{1-\alpha} \text{ mit } \alpha \in [0, 1]$$

bevorzugt für größere α Punkte mit kleinem Level und Grad. Dies stellt sicher, dass der Algorithmus global ausgerichtet ist. Ein niedriger Grad bedeutet, dass der Punkt noch nicht oft als bester Punkt ausgewählt wurde, also wird in Gebieten gesucht, in denen noch wenige Punkte bestimmt wurden. Ein niedriger Level bedeutet, dass der jeweilige Punkt kein allzu naher Nachbar eines anderen Punktes ist. Je kleiner α ist, desto mehr rücken diese beiden Kriterien in den Hintergrund und der Rang der Punkte gewinnt an Bedeutung.

Man kann natürlich auch andere Bewertungsfunktionen einsetzen, z. B.:

$$\eta_\alpha(x) = \left(\left(\frac{\text{Level}(x)}{M\text{Level} + 1} \right)^3 + \left(\frac{\text{Grad}(x)}{M\text{Grad} + 1} \right)^2 \right)^\alpha \frac{\text{Rang}(x)^{1-\alpha}}{M\text{Rang}}.$$

Dabei bezeichnet $M\text{Level}$ den Mittelwert aller Level , usw. Das Verfahren arbeitet mit η_α bei gleichem α etwas weniger adaptiv als mit β_α . Das bedeutet, dass mit dieser Bewertungsfunktion mehr Gitterpunkte mit kleinem Level bestimmt werden, bevor ein Punkt mit hohem Level erzeugt wird. Dies verdeutlicht auch Abbildung 2.6. Die nichtadaptiven Bewertungsfunktionen β_1 und η_1 liefern ein gleichmäßiges

Gitter.

Für $\alpha = 0.75$ und $\alpha = 0.9$ zeigt Abbildung 2.6, welche Punkte mit den zwei Bewertungsfunktionen für die gleiche Testfunktion bestimmt werden.

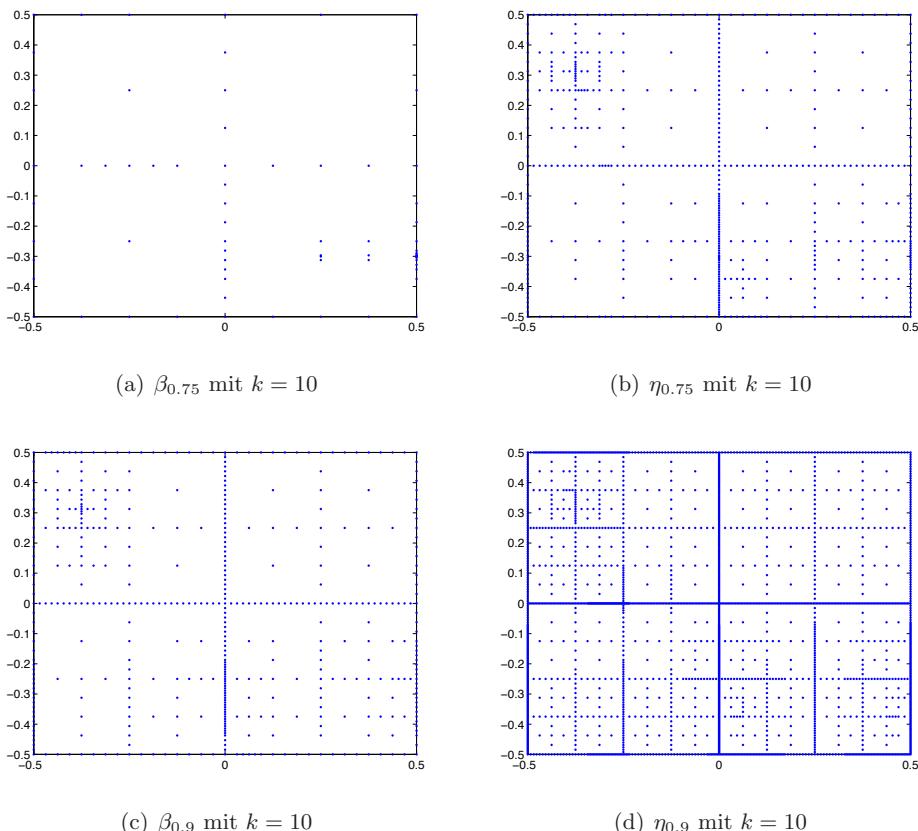


Abbildung 2.6: Vergleich der beiden Bewertungsfunktionen β und η .

Bisher wurde als Abbruchkriterium der maximale Level aller Punkte vorgeschlagen. Diese Zahl beschränkt die maximale Anzahl der verwendeten Punkte und bestimmt die Feinheit des Gitters an dem die Funktion ausgewertet wird. Ein volles Gitter mit Schrittweite 2^{-k} in jede Koordinatenrichtung hat $O(2^{kd})$ Punkte, wohingegen ein Gitter, wie es der vorliegende Algorithmus verwendet, Punkte in der Größenordnung $O(2^k k^{d-1})$ hat, siehe [Bu92].

Abbildung 2.7 verdeutlicht den Anstieg der Anzahl der Gitterpunkte für die Dimensionen 1 bis 20 für verschiedene k . Die y-Achse wurde zur besseren Darstellung logarithmiert.

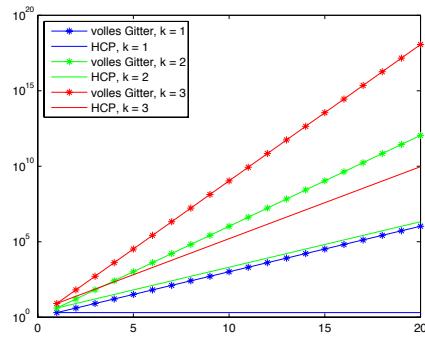


Abbildung 2.7: Anzahl der Gitterpunkte eines vollen Gitters im Vergleich zu den Punkten des HCP-Algorithmus.

Abbildung 2.8 stellt für $d = 2$ und $k = \{2, 3, 4\}$ dar, welche Punkte ein volles Gitter der Schrittweite 2^{-k} mit den Punkten des HCP-Algorithmus mit maximalem Level k gemeinsam hat.

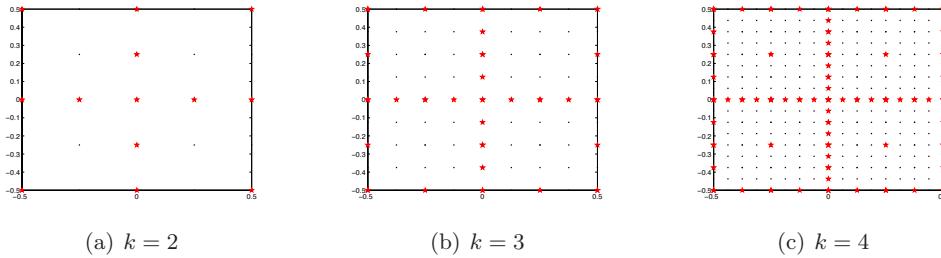


Abbildung 2.8: Volles Gitter mit Schrittweiten 2^{-k} (blauen Punkte) und Punkte des HCP-Algorithmus (rote Sterne) mit $\alpha = 1$.

Darüber hinaus ist es sinnvoll, einen weiteren Parameter $\varepsilon \in]0, 1[$ einzuführen, mit dem man überprüfen kann, ob die neuen Punkte in einer ε -Umgebung um den alten Punkt liegen. Somit stellt ε eine Toleranzgrenze für die Konvergenz dar. Wird die Toleranzgrenze erreicht, geht man davon aus, ein globales Minimum gefunden zu haben und kann nach Novak und Ritter das Verfahren abbrechen. Will man stattdessen die Suche fortsetzen, um die Wahrscheinlichkeit, nur um lokale Minima zu verfeinern, zu verkleinern, kann man alternativ anhand bestimmter Kriterien, wie z. B. dem Grad oder Rang oder auch zufällig, einen anderen Punkt bestimmen und das Verfahren zwingen, um diesen Punkt neue Nachbarn zu bestimmen. Wir ändern an dieser Stelle das Verfahren so, dass die Nachbarn des Punktes mit dem kleinsten Grad bestimmt werden. Dies stellt sicher, dass in einem Gebiet gesucht wird, in dem noch nicht viele Punkte bestimmt wurden. Algorithmus 2.10 wird also nach Schritt 4 um den folgenden Block ergänzt:

4.1. Falls $\|X_{neu} - \hat{x}\| < \varepsilon$:

1. Wähle $\hat{x} \in X$ mit $\hat{x} = \arg \min_{x \in X} (\text{Grad}(x))$;
2. Bestimme die Nachbarn X_{neu} der Stufe $\text{Grad}(\hat{x})$ von \hat{x} ;
3. Gehe zu 4.1.;

4.2. Gehe zu 5. im HCP-Algorithmus.

Will man für $\alpha = 1$ ein gleichmäßiges Gitter erhalten, muss man diesen Block weglassen, da sonst (je nach ε) ein ungleichmäßiges Gitter entsteht. Dieses Verhalten wird in Abbildung 2.9 für die Parameter $k = 7$, $\alpha = 1$ und $\epsilon = 0.01, \epsilon = 0.02$ sowie $\epsilon = 0.03$ dargestellt.

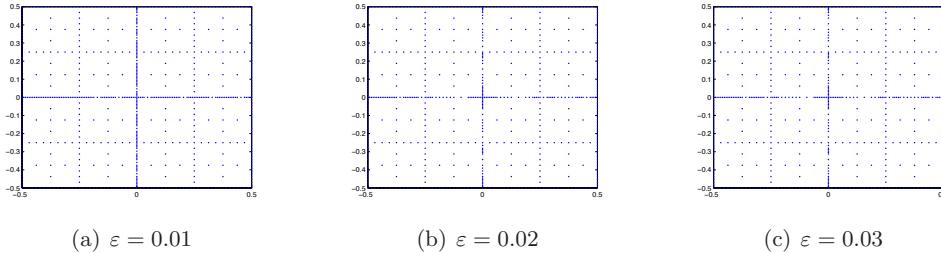


Abbildung 2.9: Gitter des HCP-Algorithmus für verschiedene ε mit $\alpha = 1$ und $k = 7$.

Man sieht, dass die Punkte nicht so gleichmäßig verteilt sind, wie man es für $\alpha = 1$ erwarten würde. Der Grund ist, dass der Algorithmus jetzt schon früher Punkte mit einem Level, der größer ist als k , berechnet und daher abbricht, bevor er alle Punkte mit kleinerem Level berechnet hat.

Das Verfahren hängt sehr stark von den gewählten Parametern und der Bewertungsfunktion ab. Natürlich kann ein erfahrener Anwender, der über gewisse Informationen über die zu minimierende Funktion verfügt, die Parameter angemessen wählen und somit den Algorithmus auf seine Bedürfnisse ausrichten. Novak und Ritter denken aber auch an den unerfahrenen Anwender und verfolgen das Ziel, den Algorithmus so allgemein wie möglich zu halten. Daher schlagen sie bestimmte Parameter vor, die sich in vielen Fällen als günstig erwiesen haben:

d	k	ε	α
2	14	0.02	0.50
3	7	0.05	0.40
4	5	0.08	0.20
5	4	0.15	0.15
≥ 6	3	0.30	0.10

Tabelle 2.1: Parameter des HCP-Algorithmus

Bei schwierigen Problemen, z. B. wenn die Funktion sehr viele lokale Minima besitzt, empfiehlt es sich für k und α größere Werte zu wählen. Ein größeres k erlaubt mehr Punkte im Gitter und vergrößert dadurch die Chance, das globale Minimum zu finden. Ein größeres α verhindert ebenfalls die Konvergenz zu einem lokalen Minimum.

2.3 Der Algorithmus von Horn

Wir werden den HCP-Algorithmus in den folgenden Kapiteln für dünne Gitter verallgemeinern und darauf aufbauend ein Verfahren zur globalen Optimierung entwickeln. Einen anderen Ansatz verfolgt Horn in [Ho05]. Ausgehend von Novak und Ritters HCP-Algorithmus entwickelt er ein Verfahren zur globalen Optimierung welches mit Lipschitzkonstanten arbeitet, bzw. diese approximiert wenn sie unbekannt sind. Es werden also zwei Fälle unterschieden:

1. Die Lipschitzkonstante der zu minimierenden Funktion f ist bekannt.
2. Die Lipschitzkonstante ist unbekannt.

Er gibt für beide Fälle adaptiv arbeitende Algorithmen an, die einen ähnlichen Aufbau wie der HCP-Algorithmus besitzen und kann für eine bestimmte Funktionsklasse zeigen, dass zum einen Adaptivität entscheidend zur Verbesserung des Verfahrens beiträgt und zum anderen Randomisierung keine Verbesserung bringt. Die Funktionen f aus dieser Funktionenklasse \mathcal{F} müssen folgende Bedingungen erfüllen:

1. $f \in \mathcal{C}^2([0, 1]^d, \mathbb{R})$.
2. f hat endlich viele globale Minima x^* .
3. Die Hessematrix $(\nabla^2 f)(x^*)$ ist positiv definit für alle x^* .

Als Kriterium für die Güte eines Verfahrens \mathcal{A} verwendet Horn die Fehlerzahl e_n , die den *worst case* Fehler des Algorithmus \mathcal{A} angewendet auf alle Funktionen f aus einer Funktionenklasse, mit maximal n Funktionsauswertungen, angibt. Für konvexe, symmetrische Funktionen gilt in diesem *worst case setting*, dass Adaptivität ein Verfahren nicht verbessern kann. Für die Klasse \mathcal{F} gilt dies allerdings nicht.

Wir beschreiben nun kurz den Algorithmus für den Fall, dass die Lipschitzkonstante bekannt ist. Betrachtet wird der Quader $[0, 1]^d$, Startpunkt ist der Mittelpunkt $(0.5, \dots, 0.5)$. Als Information über die betrachteten Punkte werden nur die Funktionswerte benötigt. In den folgenden Schritten werden mit Hilfe der Lipschitzkonstante Gebiete, in denen das globale Minimum nicht liegen kann, ausgeschlossen und in den restlichen Gebieten neue Punkte hinzugenommen.

Algorithmus 2.11 Algorithmus von Horn für Lipschitzkonstante L

1. Setze $X_1 = \{(0.5, \dots, 0.5)\}$ und bestimme $F_1 = f(X_1)$,
sei $x^* = X_1$ und $f^* = F_1$,
setze $k = 1$;
2. $k = k + 1$;
3. Setze $X_k = \{\}$ und $F_k = \{\}$;
4. Bestimme die Punkte Y_k , siehe Bemerkung 2.12.
5. $i = 1$;
6. Falls $f_i \in F_{k-1} \leq f^* - L2^{-1}3^{-j+2}$:
 - (a) $X_k = X_k \cup x_i$ und $F_k = F_k \cup f_i$;
 - (b) Bestimme $f(x_i + y)$ für alle $y \in Y_k$;
 - (c) $X_k = X_k \cup (x_i + y)$ und $F_k = F_k \cup f(x_i + y)$;
 - (d) Falls $f(x_i + y) < f^*$:
 $f^* = f(x_i + y)$ und $x^* = (x_i + y)$;
7. Falls $i < |F_{k-1}|$: $i = i + 1$, gehe 6.,
sonst: gehe zu 2.;
8. Minimalpunkt ist x^* mit Minimallösung f^* .

Bemerkung 2.12 Die Punkte in Y_k haben folgende Gestalt:

$$Y_k = \left\{ \sum_{j=1}^d b_j E_j : b_j \in \{-3^{-k+1}, 0, 3^{-k+1}\} \right\} \setminus \{0\}.$$

Wobei E_j den j-ten Einheitsvektor bezeichnet. Für den Fall $d = 2$ veranschaulicht Abbildung 2.10 das Vorgehen. Die Menge Y_k besteht für alle k aus $3^d - 1$ Punkten.

Für den Fall, dass die Lipschitzkonstante unbekannt ist, kann man Algorithmus 2.11 mit einer monoton wachsenden Folge von Lipschitzkonstanten $L_1 < \dots < L_j < L_{j+1} < \dots$ anwenden. Nach einer bestimmten Anzahl von Schritten wählt man die nächste Lipschitzkonstante für die Abfrage in Schritt 6 und iteriert so nach und nach durch die gesamte Folge (L_j). Dies zeigt auch Abbildung 2.11.

Insgesamt ist der Algorithmus von Horn eine *Pattern Search* Methode. In jedem Schritt werden um den aktuell betrachteten Punkt neue Punkte, die ein bestimmtes Muster bilden, betrachtet.

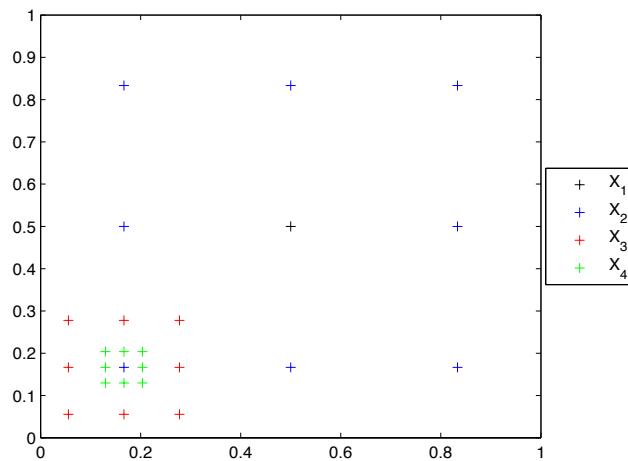
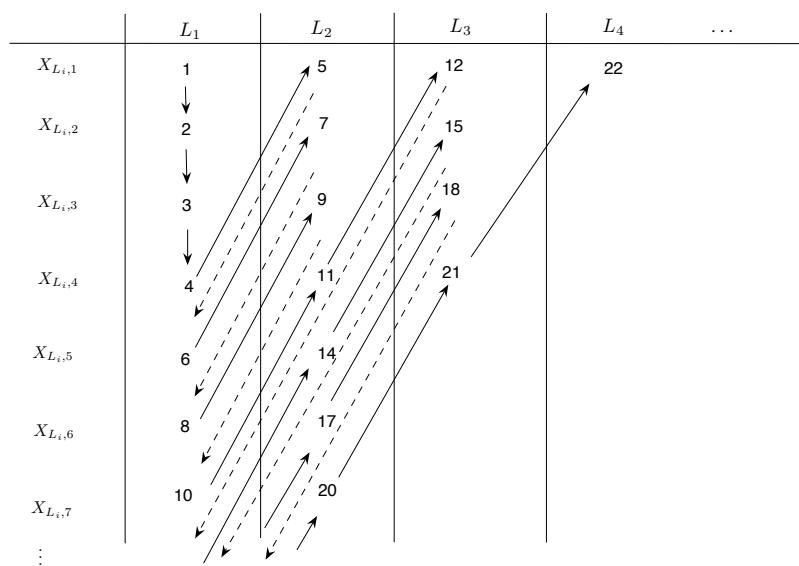
Abbildung 2.10: Das *Pattern* der Punkte $x + y$ für $k = 1, \dots, 4$ im \mathbb{R}^2 .

Abbildung 2.11: Algorithmus 2.11 für unbekannte Lipschitzkonstante.

Kapitel 3

Dünne Gitter

3.1 Einleitung

Dem im zweiten Kapitel beschriebenen HCP-Algorithmus 2.10 dient ein spezielles Gitter als Grundlage, das sich wesentlich von einem gewöhnlichen (vollen) Gitter unterscheidet. Wie Abbildung 2.8 zeigt, verwendet dieses Gitter nur einen Teil der Punkte eines vollen Gitters. Dadurch entsteht ein so genanntes *dünnnes Gitter*.

Im Folgenden wird erklärt, wie ein dünnes Gitter definiert ist und was die Vorteile von dünnen Gittern gegenüber vollen Gittern sind. Daneben wird die klassische Methode dünne Gitter zu erzeugen erläutert. Anschließend wird eine Auswahl verschiedener Typen dünner Gitter vorgestellt.

Der Name „dünnes Gitter“ (*sparse grid*) kommt von dünn besetzten Matrizen und beschreibt das wesentliche Merkmal dieser Gitter: Es werden nicht alle, sondern nur bestimmte Punkte eines vollen Gitters verwendet.

Bei der numerischen Behandlung (partieller) Differentialgleichungen oder von Quadratur-Problemen ist die Wahl einer geeigneten Diskretisierung von zentraler Bedeutung. Man steht vor dem Problem, die Balance zwischen der bestmöglichen Approximation der Lösung und möglichst geringem Rechenaufwand zu finden. Vor allem bei höherdimensionalen Problemen führt eine zu feine Diskretisierung zu einer Explosion der Rechenzeit. Dieses Phänomen wird auch als „*curse of dimension*“ bezeichnet, siehe [NR97].

Eine Möglichkeit, dieses Problem einzudämmen, stellt die adaptive Verfeinerung dar. Die Grundidee hierbei ist, dort wo die Funktion keine Unregelmäßigkeiten aufweist (z. B. Gebiete, in denen die Norm des Gradienten $\|\nabla f\|$ klein ist) ein grobes Gitter zu wählen und an Stellen mit starken Veränderungen eine hohe Gitterauflösung zu verwenden. Dadurch erreicht man, dass die Funktion nur an den Stellen häufig ausgewertet wird, an denen viel Information benötigt wird.

Smolyak-Quadratur

Die Methode der dünnen Gitter geht zurück auf Smolyak (1963), siehe [Sm63], wurde seit 1990 durch Zenger, siehe [GSZ90] und [Ze90], Bungartz, siehe [Bu92], Griebel [BG04] und anderen entscheidend weiterentwickelt und wird seitdem auf zahlreichen Gebieten, vor allem bei der numerischen Lösung partieller Differentialgleichungen, eingesetzt.

Ursprünglich löste Smolyak mit Hilfe dünner Gitter Quadratur- und Kubaturprobleme. Er entwickelte Kubaturformeln

$$\hat{I}_{d,n}(f) = \sum_{i=1}^n a_i f(x_i)$$

zur Approximation mehrdimensionaler Integrale,

$$I_d(f) = \int_{[0,1]^d} f(x) dx,$$

indem er zuerst Quadraturformeln zur Approximation eindimensionaler Probleme bestimmte und diese dann mit Hilfe des Tensorproduktes geschickt kombinierte, siehe z. B. [NR96b].

Partielle Differentialgleichungen

Zenger, Bungartz und Griebel nutzten dünne Gitter zur Lösung spezieller partieller Differentialgleichungen. In seiner Dissertation, siehe [Bu92], beschäftigt sich Bungartz mit der Lösung der Poissonsleichung

$$\sum_{i=1}^n \frac{\partial^2}{\partial x_i^2} \zeta(x) = f(x)$$

mit Dirichlet'schen Nebenbedingungen, d. h. die gesuchte Funktion ζ muss am Rand des betrachteten Gebietes Ω gleich einer Funktion $h : \partial\Omega \rightarrow \mathbb{R}$ sein, also $\zeta|_{\partial\Omega} = h$.

Data mining

Auch Themen wie z. B. *data mining*, also das Erkennen von Mustern, Abhängigkeiten und Trends in großen Datenmengen, werden mit Hilfe dünner Gitter untersucht, siehe auch [GG01].

3.2 Hierarchische Teilraumzerlegung

Wir betrachten im Folgenden den Raum $\mathcal{C}^2([-0.5, 0.5]^d, \mathbb{R})$ und geben eine Zerlegung dieses Funktionenraumes in eine direkte Summe aus endlich dimensionalen, hierarchisch geordneten Teilräumen an. Die hier vorgestellte Theorie folgt im Wesentlichen der Darstellung von Bungartz ([Bu92] und [BG04]) und Achatz ([Ac03]).

Wir werden die Theorie zunächst für den eindimensionalen Fall $d = 1$ entwickeln und dann für $d > 1$ erweitern. Das Vorgehen ist in beiden Fällen das gleiche: Zunächst werden die Approximationsräume von $\mathcal{C}^2([-0.5, 0.5]^d, \mathbb{R})$ als Linearkombination gewisser Basisfunktionen (der so genannten *nodalen Basis*, bzw. *Knotenbasis*) dargestellt und dann als Linearkombination so genannter *hierarchischer* Basisfunktionen.

Eindimensionaler Fall

Zunächst benötigen wir die Definition eines eindimensionalen Gitters mit äquidistanter Schrittweite.

Definition 3.1 eindimensionales äquidistantes Gitter

$$G_l := \left\{ x_{i,l} = i2^{-l} : i = \pm\{0, 1, 2, 3, \dots, 2^{l-1}\} \right\} \subset [-0.5, 0.5]$$

mit $l \in \mathbb{N}$ bildet das äquidistante, eindimensionale Gitter mit **Level** l , d. h. mit der Schrittweite $h_l = 2^{-l}$.

Das Gitter G_0 mit der Schrittweite $h_0 = 1$ ist definiert als

$$G_0 := \{x_{i,-1} = i2^{-1} : i = \pm 1\} = \{-0.5, 0.5\}.$$

Abbildung 3.1 stellt die G_l für $l = 0, \dots, 5$ dar.

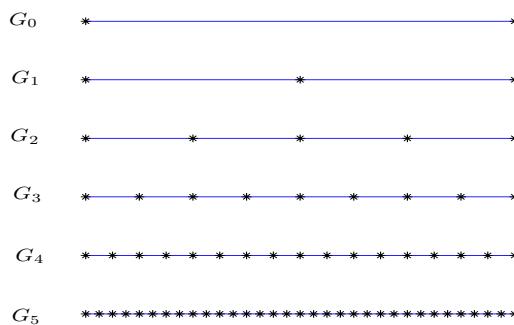


Abbildung 3.1: G_l mit $l = 0, 1, \dots, 5$.

Bemerkung 3.2

1. Die Schreibweise $i = \pm\{0, 1, 2, 3, \dots, 2^{l-1}\}$ ist äquivalent zu $i = \{-2^{l-1}, -2^{l-1} + 1, \dots, 2^{l-1} - 1, 2^{l-1}\}$.

2. Das Gitter G_l , mit $l \geq 0$, besteht aus $|G_l| = 2^l + 1$ Punkten.
3. Nach Konstruktion gilt $G_l \subset G_{l+1}$ für alle $l \in \mathbb{N}_0$.
4. Im Zusammenhang mit dem Level l des Gitters G_l ist auch die Bezeichnung *feineres*, bzw. *gröberes* Gitter gebräuchlich: Ein Gitter G_i heißt *feiner* als G_k , wenn gilt $G_k \subset G_i$ (d. h. $k < i$). Umgekehrt ist dann G_k *gröber* als G_i .

Definition 3.3 Indexmengen I_l

In Zusammenhang mit den eindimensionalen äquidistanten Gittern G_l , $l \in \mathbb{N}_0$, werden wie folgt Mengen $I_l \subset \mathbb{Z}$ definiert:

$$\begin{aligned} I_0 &:= \{-1, 1\}, \\ I_l &:= \{0, \pm 1, \pm 2, \pm 3, \dots, \pm 2^{l-1}\}, \text{ für } l > 0. \end{aligned}$$

Zur Einführung der Basisfunktionen in den Approximationsräumen von $\mathcal{C}^2([-0.5, 0.5]^d, \mathbb{R})$ benötigen wir eine Hilfsfunktion $\phi : \mathbb{R} \rightarrow [0, 1]$ mit

$$\phi(x) = \begin{cases} 1 - |x| & \text{für } x \in [-1, 1], \\ 0 & \text{sonst.} \end{cases} \quad (3.1)$$

Mit der Hilfsfunktion ϕ können wir die sog. Hutfunktionen (auch Finite-Elemente-Basisfunktionen genannt) definieren.

Definition 3.4 Hutfunktionen

Sei $l \in \mathbb{N}_0$, $x_{i,l}$ und h_l wie in Definition 3.1. Dann kann man Funktionen $\phi_{i,l} : [-0.5, 0.5] \rightarrow [0, 1]$, mit $i \in I_l$, die sog. Hutfunktionen, definieren als:

$$\phi_{i,l}(x) := \begin{cases} \phi\left(\frac{x-x_{i,l}}{h_l}\right) & \text{für } x \in [x_{i,l} - h_l, x_{i,l} + h_l] \cap [-0.5, 0.5], \\ 0 & \text{sonst.} \end{cases} \quad (3.2)$$

Für $l = 0$ gilt:

$$\begin{aligned} \phi_{-1,0}(x) &= -x + 0.5 \text{ für } x \in [-0.5, 0.5] \\ \phi_{1,0}(x) &= x + 0.5 \text{ für } x \in [-0.5, 0.5]. \end{aligned}$$

Für ein festes $l \in \mathbb{N}_0$ bilden die $\phi_{i,l}$ die *nodale Basis* (bzw. *Knotenbasis*) des Raumes der stückweise linearen Funktionen auf G_l ,

$$\mathcal{Q}_l := \text{span}\{\phi_{i,l} : i \in I_l\}^1.$$

Abbildung 3.2 zeigt die nodale Basis des Raumes \mathcal{Q}_2 .

Seien nun die Mengen J_l für $l \in \mathbb{N}_0$ wie folgt definiert:

$$J_0 = I_0 = \{-1, 1\}, \quad (3.3)$$

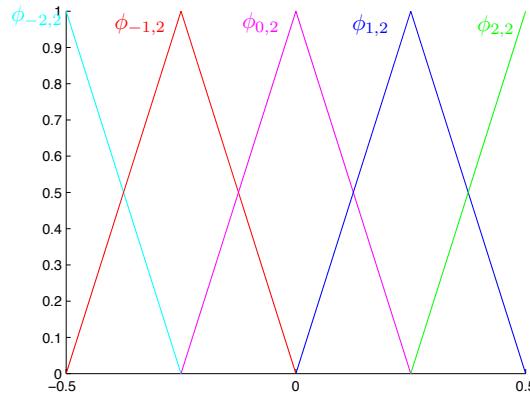
$$J_1 = \{0\}, \quad (3.4)$$

$$J_l = \{i \in I_l \setminus \{0\}, i \text{ ungerade}\} \text{ für } l > 1. \quad (3.5)$$

Mit Hilfe der J_l können wir für $l \in \mathbb{N}_0$ Räume \mathcal{T}_l definieren als

$$\mathcal{T}_l := \text{span}\{\phi_{i,l} : i \in J_l\}.$$

¹D. h. der Raum \mathcal{Q}_l wird von allen Funktionen $\{\phi_{i,l} : i \in I_l\}$ aufgespannt.

Abbildung 3.2: $\phi_{i,l}$ für $l = 2$ und $i = \{0, \pm 1, \pm 2\}$.

Beispiel 3.5 Die ersten drei Unterräume sind:

1. $\mathcal{T}_0 = \text{span}\{\phi_{j,0} \mid j \in J_0\} = \text{span}\{\phi_{-1,0}, \phi_{1,0}\} = \mathcal{Q}_0$.
2. $\mathcal{T}_1 = \text{span}\{\phi_{j,1} \mid j \in J_1\} = \text{span}\{\phi_{0,1}\}$.
3. $\mathcal{T}_2 = \text{span}\{\phi_{j,2} \mid j \in J_2\} = \text{span}\{\phi_{-1,2}, \phi_{1,2}\}$.

Es gilt:

$$\mathcal{Q}_0 = \mathcal{T}_0$$

und für $l > 0$

$$\mathcal{Q}_l = \mathcal{Q}_{l-1} \oplus \mathcal{T}_l.$$

Damit gilt:

$$\mathcal{Q}_l = \bigoplus_{0 \leq k \leq l} \mathcal{T}_k. \quad (3.6)$$

Gleichung (3.6) kann durch Induktion und Nachrechnen bewiesen werden. In Beispiel 3.6 werden die Basiswechselmatrizen für $l = 1$ und $l = 2$ angegeben.

Beispiel 3.6 1. Für $l = 1$ ist zu zeigen: $f \in \mathcal{Q}_1 \Leftrightarrow f \in \mathcal{T}_0 \oplus \mathcal{T}_1$.

Bezeichne $\hat{B}_1 = \{\phi_{-1,1}, \phi_{0,1}, \phi_{1,1}\}$ die nodale Basis von \mathcal{Q}_1 und $\tilde{B}_1 = \{\phi_{-1,0}, \phi_{0,1}, \phi_{1,0}\}$ die Basis von $\mathcal{T}_0 \oplus \mathcal{T}_1$.

„ \Rightarrow “: Sei $f \in \mathcal{Q}_1$. Dann kann man mit

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

die Basisvektoren in \tilde{B}_1 wie folgt als Linearkombination der Basisvektoren in \hat{B}_1 darstellen: $(\phi_{-1,0}, \phi_{0,1}, \phi_{1,0}) = (\phi_{-1,1}, \phi_{0,1}, \phi_{1,1})A$. Somit gilt $f \in \mathcal{T}_0 \oplus \mathcal{T}_1$.

„ \Leftarrow “: Sei $f \in \mathcal{T}_0 \oplus \mathcal{T}_1$. Mit

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & -0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

kann man die Basisvektoren in \hat{B}_1 wie folgt als Linearkombination der Basisvektoren in \tilde{B}_1 darstellen: $(\phi_{-1,1}, \phi_{0,1}, \phi_{1,1}) = (\phi_{-1,0}, \phi_{0,1}, \phi_{1,0}) A^{-1}$. Somit gilt $f \in \mathcal{Q}_1$.

2. Für $l = 2$ ist zu zeigen: $f \in \mathcal{Q}_2 \Leftrightarrow f \in \mathcal{T}_0 \oplus \mathcal{T}_1 \oplus \mathcal{T}_2$. Bezeichne wieder $\hat{B}_2 = \{\phi_{-2,2}, \phi_{-1,2}, \phi_{0,2}, \phi_{1,2}, \phi_{2,2}\}$ die nodale Basis von \mathcal{Q}_2 und $\tilde{B}_2 = \{\phi_{-1,0}, \phi_{-1,2}, \phi_{0,1}, \phi_{1,2}, \phi_{1,0}\}$ die Basis von $\mathcal{T}_0 \oplus \mathcal{T}_1 \oplus \mathcal{T}_2$.

„ \Rightarrow “: Sei $f \in \mathcal{Q}_2$. Analog zu 1. kann man mit der Matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0.75 & 1 & 0.5 & 0 & 0.25 \\ 0.5 & 0 & 1 & 0 & 0.5 \\ 0.25 & 0 & 0.5 & 1 & 0.75 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

die Basisvektoren in \hat{B}_2 wie folgt als Linearkombination der Basisvektoren in \tilde{B}_2 darstellen:

$$(\phi_{-1,0}, \phi_{-1,2}, \phi_{0,1}, \phi_{1,2}, \phi_{1,0}) = (\phi_{-2,2}, \phi_{-1,2}, \phi_{0,2}, \phi_{1,2}, \phi_{2,2}) A.$$

Somit gilt $f \in \mathcal{T}_0 \oplus \mathcal{T}_1 \oplus \mathcal{T}_2$.

„ \Leftarrow “: Sei $f \in \mathcal{T}_0 \oplus \mathcal{T}_1 \oplus \mathcal{T}_2$. Mit

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 \\ -0.5 & 0 & 1 & 0 & -0.5 \\ 0 & 0 & -0.5 & 1 & -0.5 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

lassen sich die Basisvektoren in \hat{B}_2 wie folgt als Linearkombination der Basisvektoren in \tilde{B}_2 darstellen:

$$(\phi_{-2,2}, \phi_{-1,2}, \phi_{0,2}, \phi_{1,2}, \phi_{2,2}) = (\phi_{-1,0}, \phi_{-1,2}, \phi_{0,1}, \phi_{1,2}, \phi_{1,0}) A^{-1}.$$

Somit gilt $f \in \mathcal{Q}_2$.

Abbildung 3.4 zeigt die Basisfunktionen in \hat{B}_l und \tilde{B}_l für $l = \{0, 1, 2, 3\}$.

Definition 3.7 hierarchische Basis von \mathcal{Q}_l

Für ein festes $l \in \mathbb{N}_0$ bildet die Menge

$$B_l = \{\phi_{j,k} : j \in J_k, 0 \leq k \leq l\}$$

die so genannte hierarchische Basis von \mathcal{Q}_l .

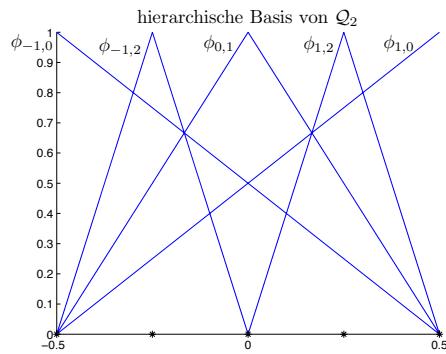
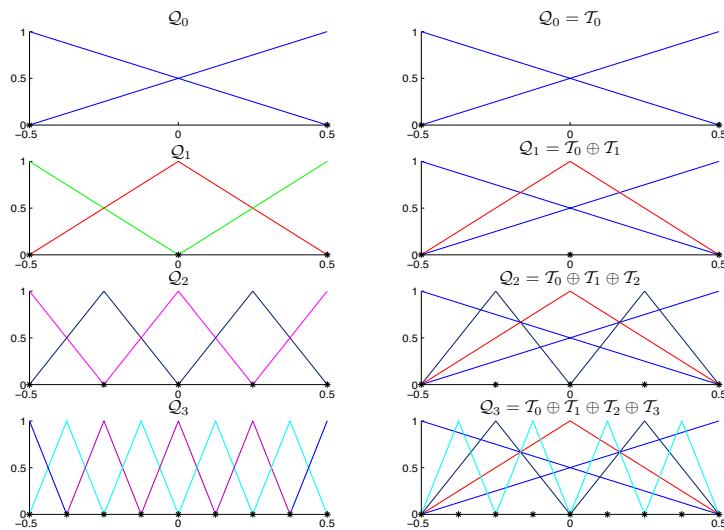
Abbildung 3.3: $\phi_{j,l}$ mit $l \in \{0, 1, 2\}$ und $j = J_l$.Abbildung 3.4: Nodale und hierarchische Basis von Q_l mit $l = \{0, 1, 2, 3\}$.

Abbildung 3.3 zeigt die hierarchische Basisfunktionen für den Raum \mathcal{Q}_2 und Abbildung 3.4 vergleicht die nodale mit der hierarchischen Basis der Räume \mathcal{Q}_l , für $l = 0, 1$ und 2 .

Bemerkung 3.8 Der Vorteil der hierarchischen Basisfunktionen gegenüber der Knotenbasis besteht darin, dass bei der Erhöhung der Dimension des Approximationsraumes \mathcal{Q}_l die Basisfunktionen nicht komplett ausgetauscht werden müssen, sondern lediglich neue Basisfunktionen hinzugenommen werden.

Bevor wir zum mehrdimensionalen Fall übergehen, überlegen wir noch kurz, in wiefern die Räume \mathcal{Q}_l tatsächlich Approximationsräume des unendlich dimensionalen Funktionenraums $\mathcal{C}^2([-0.5, 0.5], \mathbb{R})$ sind.

Satz 3.9 Für die Räume \mathcal{Q}_l , mit $l \in \mathbb{N}_0$ gilt:

1. $\mathcal{Q}_0 \subset \dots \subset \mathcal{Q}_l \subset \mathcal{Q}_{l+1} \subset \dots \subset \mathcal{C}^2([-0.5, 0.5], \mathbb{R})$.
2. Für alle $f \in \mathcal{C}^2([-0.5, 0.5], \mathbb{R})$ existiert ein $f_l \in \mathcal{Q}_l$ mit $\|f_l - f\|_{\mathcal{C}^2} \rightarrow 0$ für $l \rightarrow \infty$. ($\|f\|_{\mathcal{C}^2} = \|f\|_{L^2} + \|f'\|_{L^2} + \|f''\|_{L^2}$.)

Die erste Aussage wird sofort durch den Aufbau der \mathcal{Q}_l als direkte Summe der Unterräume \mathcal{T}_k klar, siehe (3.6). Für den Beweis der zweiten Aussage siehe z. B. [Bu92].

Mehrdimensionaler Fall

Um die im vorherigen Abschnitt eingeführte hierarchische Basiszerlegung des Raumes der stückweise linearen Funktionen für beliebige Dimensionen zu verallgemeinern, müssen wir zunächst den Begriff des *Tensorproduktes* einführen.

Definition 3.10 Tensorprodukt von Vektorräumen

Seien V und W zwei Vektorräume auf \mathbb{R} . Sei $B_V = \{\nu_i : i \in I_V\}$ eine Basis von V und $B_W = \{\omega_j : j \in I_W\}$ eine Basis von W und I_V, I_W seien zugehörige Indexmengen. Dann ist das Tensorprodukt $V \otimes W$ ein Vektorraum mit der Basis $B_{V \otimes W} = \{\nu_i \otimes \omega_j : i \in I_V \text{ und } j \in I_W\}$. Es gilt: $\dim(V \otimes W) = \dim(V) \cdot \dim(W)$.

Die geltenden Rechenregeln und weitere Details können einem Standardwerk der Linearen Algebra entnommen werden, z. B. [Fi00].

Seien $I \in \mathbb{Z}^d$ und $L \in \mathbb{N}_0^d$ Vektoren mit $I = (i_1, \dots, i_d)$ und $L = (l_1, \dots, l_d)$. Dann entsteht der Raum \mathcal{Q}_L als Tensorprodukt der Räume \mathcal{Q}_{l_j} :

$$\mathcal{Q}_L = \bigotimes_{j=1}^d \mathcal{Q}_{l_j}.$$

Mit den Rechenregeln für Tensorprodukte gilt:

$$\mathcal{Q}_L = \bigotimes_{j=1}^d \mathcal{Q}_{l_j} = \bigotimes_{j=1}^d \left(\bigoplus_{0 \leq k \leq l_j} \mathcal{T}_k \right) = \bigoplus_{(i_1, \dots, i_d)=0}^{(l_1, \dots, l_d)} \underbrace{\left(\bigotimes_{j=1}^d \mathcal{T}_{i_j} \right)}_{\mathcal{T}_K} = \bigoplus_{0 \leq K \leq L} \mathcal{T}_K.$$

Zur Vereinfachung der Notation wird für die Räume \mathcal{Q}_L mit $L = (n, n, \dots, n)$ auch die Schreibweise

$$\mathcal{Q}_n = \bigoplus_{\|K\|_\infty \leq n} T_K$$

verwendet.

Die zugehörigen Basisfunktionen $\phi_{I,L} : [-0.5, 0.5]^d \rightarrow \mathbb{R}$ sind definiert als:

$$\phi_{I,L}(x) = \prod_{k=1}^d \phi_{i_k, l_k}(x_k).$$

Notation: Die Funktionen $\phi_{I,L}$ lassen sich schreiben als:

$$\phi_{I,L} = \phi_{i_1, l_1} \otimes \phi_{i_2, l_2} \otimes \dots \otimes \phi_{i_d, l_d}.$$

Beispiel 3.11 Betrachten wir zweidimensionale Probleme. Zum Beispiel entsteht der Raum \mathcal{Q}_L mit $L = (2, 3)$ als Tensorprodukt der Räume \mathcal{Q}_2 und \mathcal{Q}_3 . Da der Raum \mathcal{Q}_2 5-dimensional und \mathcal{Q}_3 9-dimensional ist, hat $\mathcal{Q}_{(2,3)}$ die Dimension $5 \cdot 9 = 45$. Die hierarchische Basis $B_{(2,3)}$ lässt sich schreiben als:

$$B_{(2,3)} = \{\phi_{i,k} \otimes \phi_{j,l} : i \in J_k, 0 \leq k \leq 2, j \in J_l, 0 \leq l \leq 3\}.$$

Eine Basisfunktion aus $B_{(2,3)}$ ist z. B.

$$\phi_{(0,-1),(1,2)} = (\phi_{0,1} \otimes \phi_{-1,2})(x_1, x_2) = \phi_{0,1}(x_1) \phi_{-1,2}(x_2)$$

mit dem Träger $[-0.5, 0.5] \times [-0.5, 0]$ siehe auch Abbildung 3.5.

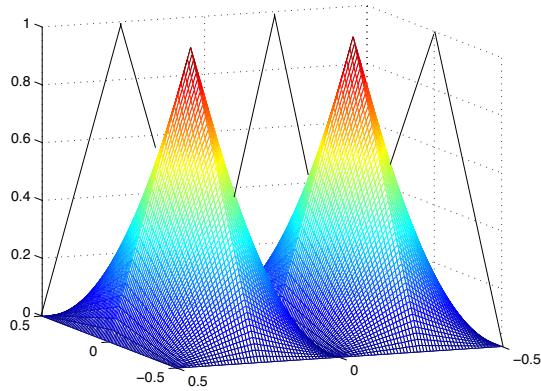


Abbildung 3.5: $\phi_{(0,-1),(1,2)}$ und $\phi_{(0,1),(1,2)}$.

Bemerkung 3.12 Für $d = 2$ heißen die Funktionen aus dem Raum \mathcal{Q}_L auch stückweise *bilinear*. Eine d -lineare Funktion ist in jeder Komponente linear.

Da die nodalen Basisfunktionen der Räume \mathcal{Q}_L an einem Punkt innerhalb ihres Trägers den Wert 1 annehmen und dann zum Rand hin linear abfallen, sagt man auch, dass die Funktionen $f \in \mathcal{Q}_L$ auf einem Gitter G_L , bestehend aus genau diesen Punkten, leben.

Definition 3.13 *d-dimensionales Gitter mit Level L*

Sei $L = (l_1, \dots, l_d) \in \mathbb{N}_0^d$, dann bezeichnet $G_L = G_{l_1, \dots, l_d}$ ein Gitter auf $[-0.5, 0.5]^d$ mit Schrittweite $h_j = 2^{-l_j}$ in die x_j -Richtung. Das Gitter G_L entsteht als kartesisches Produkt der eindimensionalen äquidistanten Gitter G_{l_j} :

$$G_{l_1, \dots, l_d} = G_{l_1} \times G_{l_2} \times \dots \times G_{l_d}$$

und ist dadurch in jede Koordinatenrichtung äquidistant. Der Vektor $L = (l_1, \dots, l_d)$ bezeichnet den Level des Gitters. Ein Gitter G_{l_1, \dots, l_d} ist **feiner** als G_{k_1, \dots, k_d} , wenn gilt:

$$K \leq L \iff k_j \leq l_j \quad \forall j \quad \text{und} \quad \sum_{j=1}^d k_j < \sum_{j=1}^d l_j. \quad (3.7)$$

Die Menge

$$B_L = \{\phi_{I,K} \mid I \in I_K : 0 \leq K \leq L\}$$

bildet die hierarchische Basis von \mathcal{Q}_L .

Für das Beispiel $\mathcal{Q}_{3,3}$ stellt Abbildung 3.6 die Basisfunktionen der Teilräume $\mathcal{T}_{1,1}, \mathcal{T}_{1,2}, \dots$ graphisch dar.

Die Basisfunktionen der Teilräume $\mathcal{T}_{0,0}, \mathcal{T}_{0,1}, \dots$ wurden hier nicht dargestellt.

In Abbildung 3.7 sind die Träger der Basisfunktionen aller Teilräume von $\mathcal{Q}_{3,3}$ dargestellt.

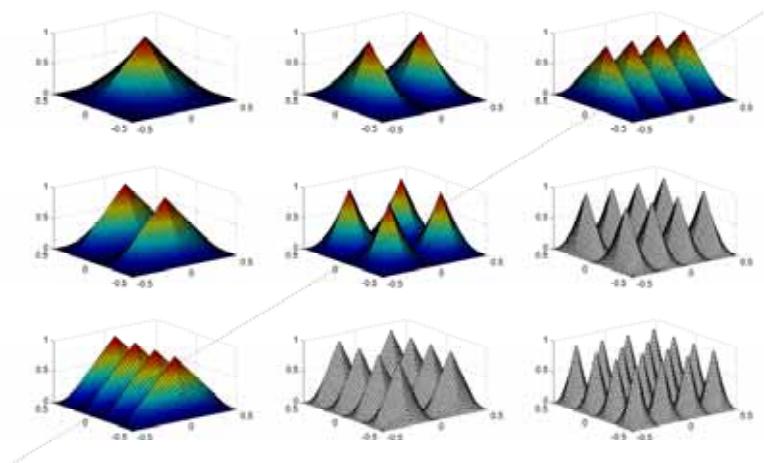
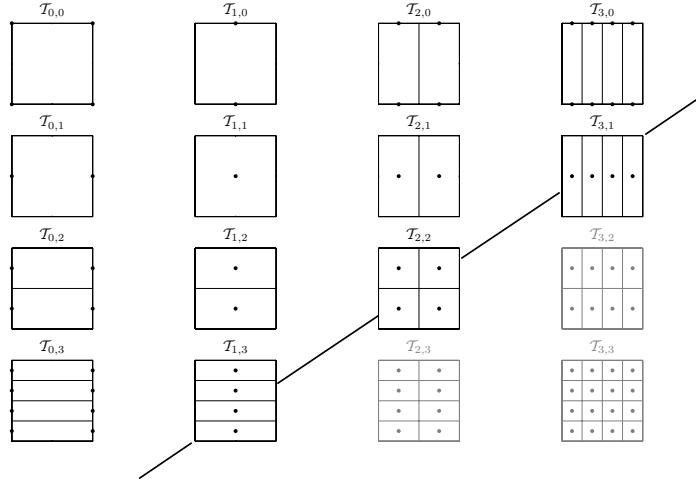


Abbildung 3.6: Basisfunktionen auf den Teilräumen $\mathcal{T}_{i,j}$ mit i und $j \in \{1, 2, 3\}$.

In [Ze90] zeigt Zenger den Zusammenhang zwischen der Größe des Trägers einer hierarchischen Basisfunktion und deren Bedeutung für die Approximation einer

Abbildung 3.7: Träger der hierarchischen Basisfunktion von $\mathcal{Q}_{3,3}$.

Funktion $f \in \mathcal{C}^2([-0.5, 0.5], \mathbb{R})$. Bungartz erweitert diese Betrachtungen in [Bu92] für Funktionen $f \in \mathcal{C}^2([-0.5, 0.5]^d, \mathbb{R})$ für $d \geq 1$. Er zeigt, dass für glatte Funktionen der Anteil einer hierarchischen Basisfunktion an der Approximation durch die Größe des Trägers nach oben beschränkt ist. Das bedeutet, dass Basisfunktionen mit einem kleinen Träger einen relativ kleinen Beitrag zur Verbesserung der Approximationsgüte leisten. Aufgrund dieser Betrachtungen werden die Basisfunktionen mit kleinem Träger bei der Bildung dünner Gitter nicht berücksichtigt. Damit wird der Raum Q_n^s (d.h. \mathcal{Q}_L mit $L = (n, n, \dots, n)$) über die Gleichung

$$Q_n^s = \bigoplus_{\|K\|_1 \leq n+1} \mathcal{T}_K \quad (3.8)$$

erklärt und ist ein Teilraum von \mathcal{Q}_n .

Graphisch bedeutet dies, dass die Teilräume unterhalb der in Abbildung 3.6 und 3.7 angedeuteten Diagonalen nicht berücksichtigt werden.

Definition 3.14 *dünnes Gitter mit Level n*

Die Gitter G_n^s , die zu den Approximationsräumen Q_n^s gehören, heißen *dünne Gitter mit Level n*. Das dünne Gitter G_n^s entsteht wie folgt als kartesisches Produkt der eindimensionalen Gitter G_{l_j} :

$$G_n^s = \bigcup_{(l_1 + \dots + l_d) \leq n+1} (G_{l_1} \times G_{l_2} \times \dots \times G_{l_d}).$$

Beispiele für dünne Gitter im zwei- und dreidimensionalen Raum sieht man in Abbildung 3.8. Wegen der starken Konzentration der Gitterpunkte auf den Rand des Quaders $[-0.5, 0.5]^d$ nennen wir das so entstandene dünne Gitter auch **Boundary-Gitter** und bezeichnen das d -dimensionale Boundary-Gitter mit Level l mit BG_l^d .

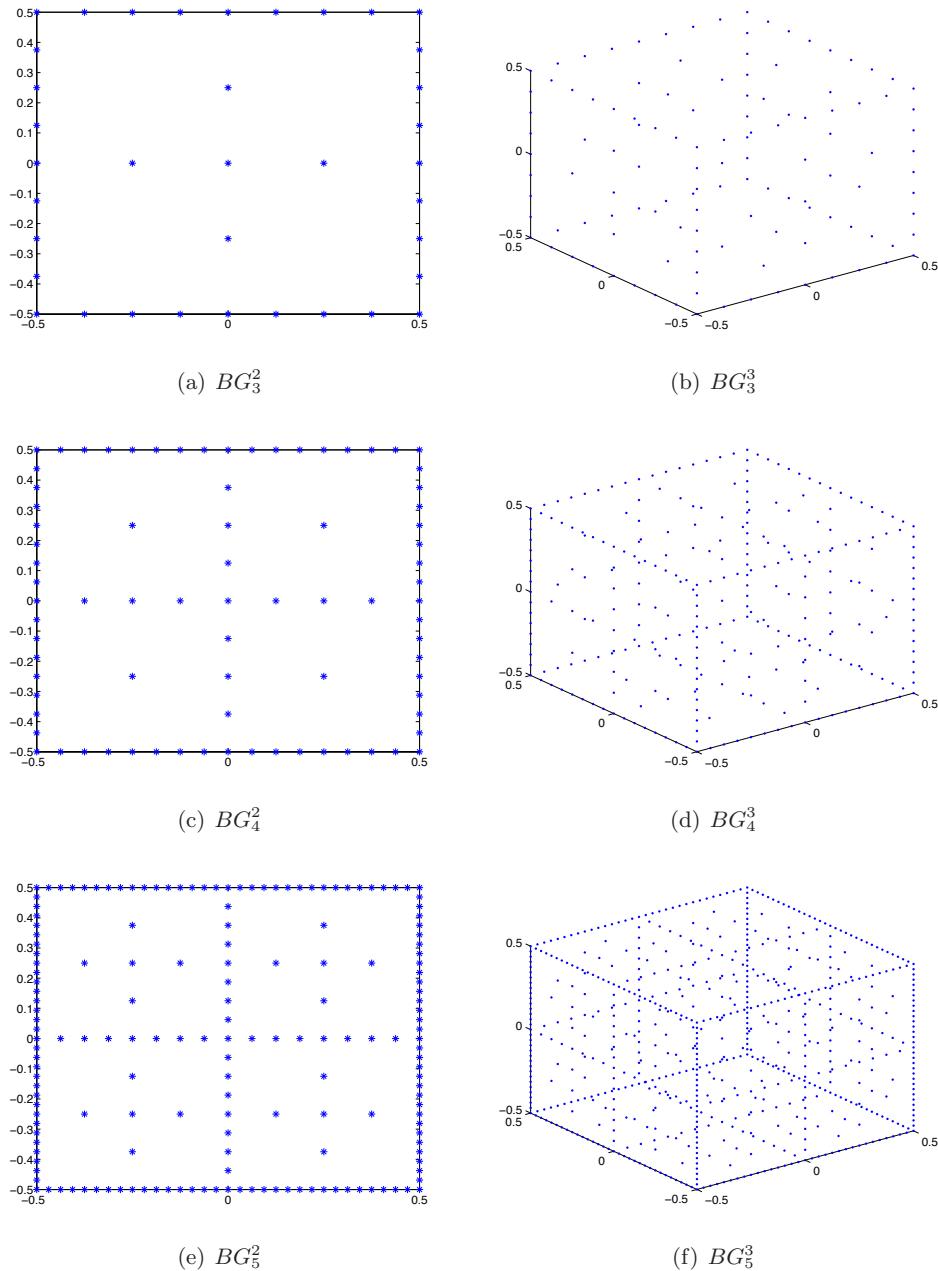


Abbildung 3.8: Zwei- und dreidimensionales Boundary-Gitter mit Level 3 bis 5.

3.3 Die Kombinationsmethode

Zenger, Griebel und Schneider entwickeln in [GSZ90] die Kombinationsmethode zur Lösung partieller Differentialgleichungen auf dünnen Gittern. Dabei ist die Idee, dass das zu lösende Problem zunächst auf einer Folge von rechtwinkligen, in jeder Koordinatenrichtung äquidistanten, Gittern (vgl. Definition 3.13) gelöst wird und eine Linearkombination dieser Lösungen die Lösung des Ausgangsproblems ergibt. Bei großen Datenmengen bietet es sich hier an, mit einem Rechnernetzwerk zu arbeiten und die unterschiedlichen Gitterlösungen parallel zu bestimmen.

Definition 3.15 Kombinationsmethode

Sei $L = (l_1, \dots, l_d) \in \mathbb{N}_0^d$, und seien G_L d-dimensionale Gitter gemäß Definition 3.13. Bezeichne f_L die Lösung eines linearen Problems, z. B. eine lineare partielle Differentialgleichung, ein Quadraturproblem etc., auf dem Gitter G_L . Dann ist die Lösung des Problems mit der Kombinationsmethode gegeben als:

$$f_n^c = \sum_{\|L\|_1=n} f_L - \sum_{\|L\|_1=n-1} f_L. \quad (3.9)$$

Abbildung 3.9 stellt die Kombinationsmethode für $d = 2$ und $n = 4$ dar.

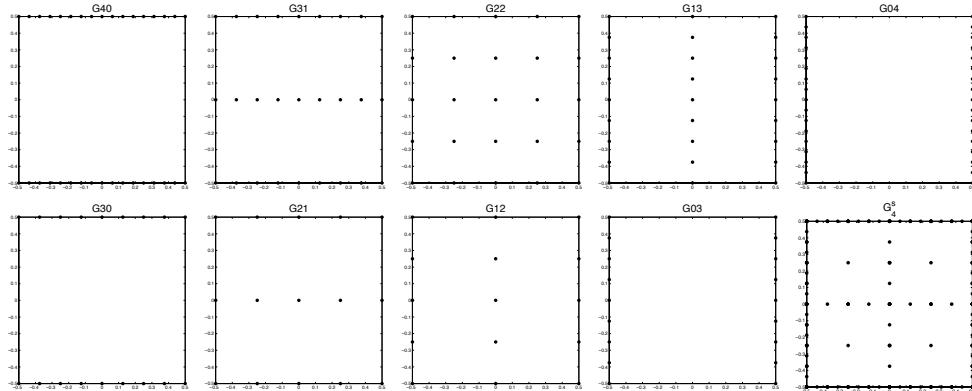


Abbildung 3.9: Kombinationsmethode für $n = 4$: $G_{4,0} + G_{3,1} + G_{2,2} + G_{1,3} + G_{0,4} - G_{3,0} - G_{2,1} - G_{1,2} - G_{0,3} = G_4^s$.

Beispiel 3.16 zweidimensionales Quadraturproblem

Zu lösen ist das Quadraturproblem $I_2(f) = \int_{[-0.5, 0.5]^2} f(x) dx$. Laut Definition 3.15 gilt

$$f_n^c = \sum_{k+l=n} f_{k,l} - \sum_{k+l=n-1} f_{k,l}.$$

Mit der Trapezregel ist die Lösung $f_{k,l}$ des Quadraturproblems auf dem Gitter $G_{k,l}$ gegeben als

$$f_{k,l} = \sum_{i \in I_k} \sum_{j \in I_l} c_{k,l} g_i f(x_{i,k}, x_{j,l}) h_k h_l.$$

Das Produkt $h_k h_l$ ist die Grundfläche, die mit dem Funktionswert des Stützknotens $(x_{i,k}, x_{j,l})$ multipliziert wird. Die Gewichte c_k und g_l steuern den Beitrag der Fläche zur Approximation der Lösung und haben für die Trapezregel den Wert 1 im Inneren, $\frac{1}{2}$ am Rand und $\frac{1}{4}$ in den Ecken, siehe auch [Zik04].

Bemerkung 3.17

1. Die Vollgitterlösung auf einem Gitter mit Schrittweite $h = 2^{-n}$ in jede Koordinatenrichtung wird üblicherweise mit f_n bezeichnet.
2. Die Lösung auf einem dünnen Gitter G_n^s (siehe Definition 3.9) bezeichnet man mit f_n^s .

3.4 Lösung von Optimierungsproblemen auf dünnen Gittern

Da die dünnen Gitter in der vorliegenden Arbeit zur Lösung eines Optimierungsproblems, einer nichtlinearen Problemstellung, genutzt werden sollen, ist hier die Kombinationsmethode nicht der geeignete Ansatz.

Betrachten wir das Ausgangsproblem (GP) ohne Nebenbedingungen:

$$\min_{x \in S} f(x).$$

Die Lösung auf einem vollen Gitter mit Schrittweite $h = 2^{-n}$, $G_{n,n,\dots,n}$, ist definiert als

$$f_n = \min(f(G_{n,n,\dots,n})).$$

Analog kann man die Lösung auf einem dünnen Gitter G_n^s erklären als das Minimum aller Funktionswerte von f ausgewertet an den Gitterpunkten von G_n^s :

$$f_n^s = \min(f(G_n^s)).$$

Da dünne Gitter hierarchisch aufgebaut sind gilt:

$$f_n^s = \min(f(G_n^s)) = \min(f(\bigcup_{l \leq n} G_l^s)).$$

Mit Induktion kann man zeigen, dass gilt:

$$G_n^s = G_{n-1}^s \cup \left(\bigcup_{(l_1+\dots+l_d)=n} G_{l_1} \times G_{l_2} \times \dots \times G_{l_d} \right).$$

Wie in Bemerkung 3.2 erwähnt, gilt für die eindimensionalen Gitter G_i und G_k :

$$G_i \subset G_k, \text{ für } i < k.$$

Das kartesische Produkt $G_i \times G_k$ kombiniert alle Elemente aus G_i mit allen Elementen aus G_k , das bedeutet, dass im Produkt $G_i \times G_k$ alle $G_j \times G_m$ mit $j < i$ und $m < k$ enthalten sind. Wir erhalten somit als Lösung eines Optimierungsproblems:

$$f_n^s = \min(f(G_n^s \setminus G_{n-1}^s)) = \min\left(f\left(\bigcup_{(l_1+\dots+l_d)=n} G_{l_1} \times G_{l_2} \times \dots \times G_{l_d}\right)\right).$$

Da es sich bei der Optimierung um ein nichtlineares Problem handelt, kann man hier keine auf parallelen Rechnern arbeitende Methode implementieren. Man muss

stattdessen zunächst das gewünschte Gitter erzeugen, dann Punkte, die mehrmals vorkommen, aus der Menge der Gitterpunkte entfernen und schließlich die Zielfunktion an allen Gitterpunkten auswerten. Im nächsten Abschnitt dieses Kapitels wird eine Methode vorgestellt, mit der man dünne Gitter rekursiv erzeugen kann. Anschließend wird in Kapitel 4 ein Algorithmus entwickelt, der iterativ ein dünnes Gitter erzeugt und somit Adaptivität ermöglicht. Das bedeutet, das Gitter wird dort verfeinert, wo die Zielfunktion kleine Werte annimmt, also das Minimum vermutet wird.

3.5 Erzeugung dünner Gitter

Durch die Konstruktion der dünnen Gitter ist klar, dass die eindimensionalen Gitter G_l , siehe Definition 3.1, grundlegend für die Erzeugung beliebig dimensionaler dünner Gitter sind. Je nachdem, wie man diese wählt, erhält man unterschiedliche Gitter mit unterschiedlichen Vor- und Nachteilen.

Für welche eindimensionalen Gitter man sich auch entscheidet, der Grundalgorithmus zur Erzeugung dünner Gitter folgt immer dem gleichen, rekursiven Schema:

Algorithmus 3.18 sparsegrid

1. Input: $l \in \mathbb{N}$, der Level des dünnen Gitters,
 $d \in \mathbb{N}$, die Dimension des dünnen Gitters;
2. Falls $d = 1$:

$$Y = G_l;$$
3. Sonst

$$Y = \{\},$$

$$\text{für } k = 0:n$$

$$Y = Y \cup (G_k \times \text{sparsegrid}(l - k, d - 1));$$
4. Y ist das d -dimensionale dünne Gitter mit Level l .

Wählt man für die G_l die eindimensionalen Gitter aus Definition 3.13, erhält man mehrdimensionale Gitter, die auf dem Rand eine kleinere Schrittweite haben als im Inneren, siehe auch Abbildung 3.8.

Wenn man bei der Optimierung einer nichtlinearen Funktion davon ausgeht, dass das Minimum nicht auf dem Rand des Quaders S liegt, können diese Gitter nachteilig sein.

Wie im ersten Kapitel schon erwähnt, ist unser Ziel in der globalen Optimierung nicht das Auffinden des exakten globalen Minimums, sondern die Bestimmung eines Punktes, der sich im Konvergenzbereich des globalen Minimums befindet, so dass ein lokaler Algorithmus dieses findet. Sollte sich das globale Minimum auf dem Rand von S befinden, kann man davon ausgehen, dass das lokale Verfahren (siehe hierzu auch Kapitel 6) dieses mit einem Startpunkt, der im Inneren von S im Konvergenzbereich liegt, finden wird. Daher kann man in den meisten Fällen auf eine Auswertung der Zielfunktion auf dem Rand von S verzichten.

Aus diesen Beweggründen werden wir im folgenden Unterkapitel verschiedene Gittertypen einführen.

3.6 Äquidistante Gitter im eindimensionalen Fall

Das bislang vorgestellte dünne Gitter hat zwei wesentliche Eigenschaften:

1. Die zugrunde liegenden eindimensionalen Gitter sind äquidistant.
2. Das Gitter G_l mit Level l enthält alle Punkte der Gitter mit einem Level $\leq l$.

Im ersten Teil dieses Unterkapitels werden wir Gitter vorstellen, die beide Eigenschaften aufweisen. Danach konstruieren wir Gitter, die die Äquidistanz im eindimensionalen Fall aufgeben.

3.6.1 Das Maximumgitter

Definition 3.19 Eindimensionales Maximumgitter

Ein eindimensionales Maximumgitter mit Level $l \in \mathbb{N}_0$ besteht aus der Menge:

$$M_l := \left\{ x_{i,l} = i2^{-(l+1)} : i \in I_{l+1} \right\}.$$

Bemerkung 3.20 Für die in Definition 3.1 erklärten Boundary-Gitter gilt: $M_l = BG_{l+1}$.

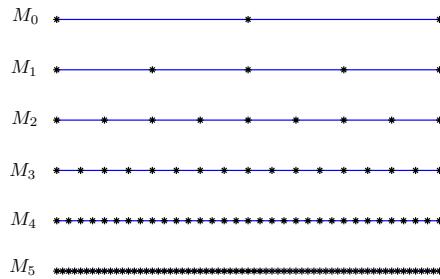


Abbildung 3.10: M_l mit Level 0 bis 5.

Definition 3.21 d -dimensionales Maximumgitter

Das d -dimensionale Maximumgitter mit Level $n \in \mathbb{N}_0$ und Dimension $d > 1$ besteht aus der Menge

$$M_n^d := \bigcup_{l_1 + \dots + l_d = n} M_{l_1} \times M_{l_2} \times \dots \times M_{l_d}.$$

Abbildung 3.11 zeigt zwei- und dreidimensionale Maximumgitter mit verschiedenen Level.

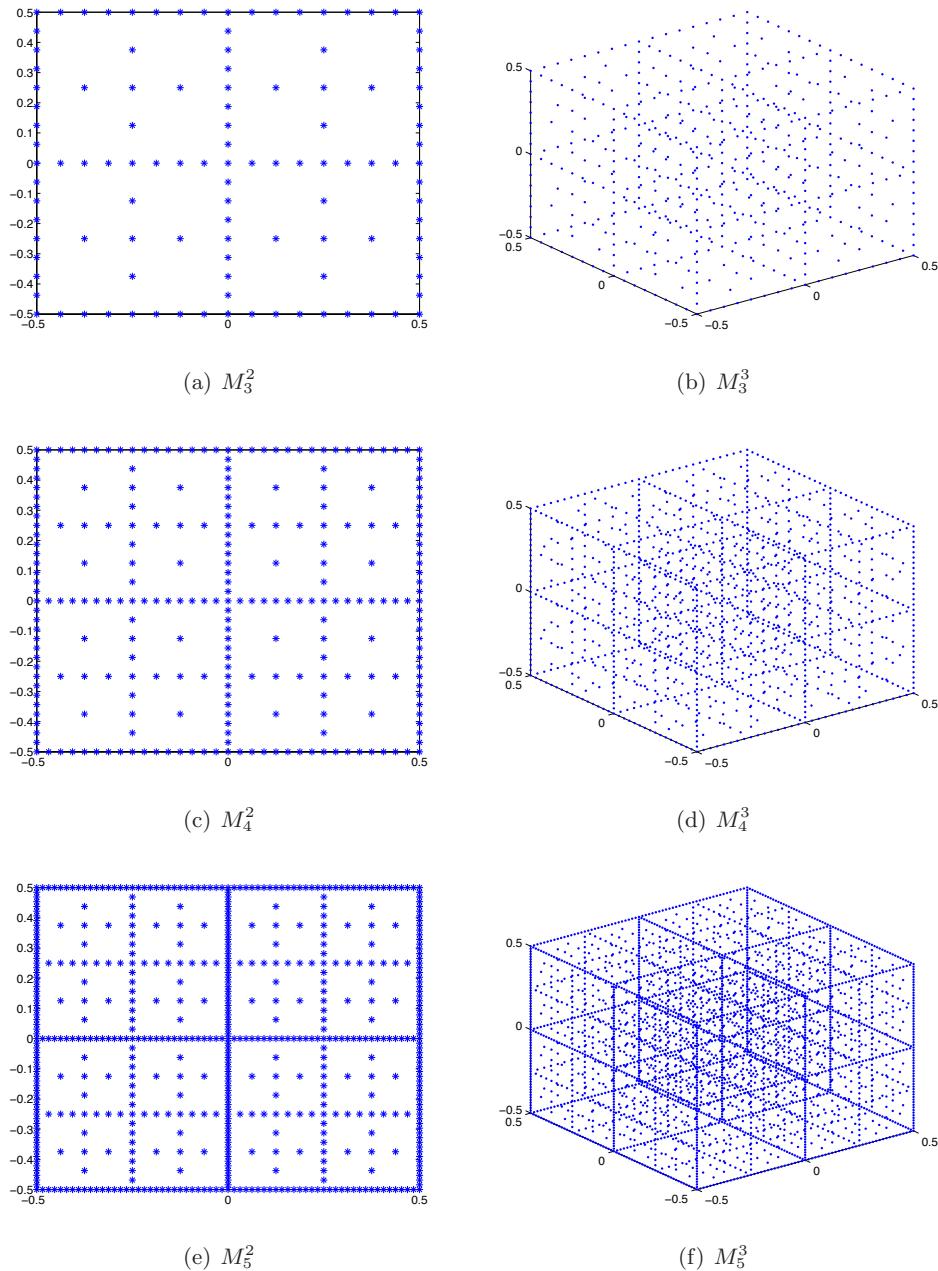


Abbildung 3.11: Zwei- und dreidimensionale Maximumgitter mit Level 3 bis 5.

3.6.2 Das Noboundary-Gitter

Wie schon erwähnt, kann man unter Umständen ganz darauf verzichten, die betrachtete Zielfunktion auf dem Rand von S auszuwerten, daher geben wir auch ein dünnes Gitter ohne Randpunkte an, das Noboundary-Gitter.

Definition 3.22 Eindimensionales Noboundary-Gitter

Ein eindimensionales Noboundary-Gitter mit Level $l \in \mathbb{N}_0$ besteht aus der Menge

$$NB_l := \left\{ x_{i,l} = i2^{-(l+1)} : i = \pm\{0, 1, 2, 3, \dots, 2^l - 1\} \right\}.$$

Dies entspricht einem eindimensionalen Maximumgitter M_l ohne Randpunkte.

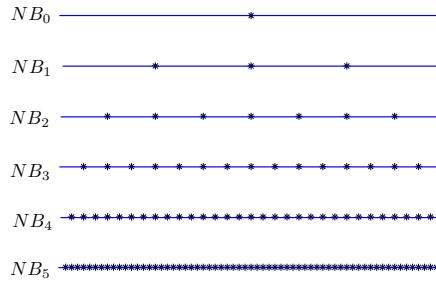


Abbildung 3.12: NB_l mit Level 0 bis 5.

Definition 3.23 d -dimensionales Noboundary-Gitter

Das d -dimensionale Noboundary-Gitter mit Level $n \in \mathbb{N}_0$ und Dimension $d > 1$ besteht aus der Menge

$$NB_n^d := \bigcup_{l_1 + \dots + l_d = n} NB_{l_1} \times NB_{l_2} \times \dots \times NB_{l_d}.$$

Abbildung 3.13 zeigt zwei- und dreidimensionale Noboundary-Gitter mit verschiedenen Level.

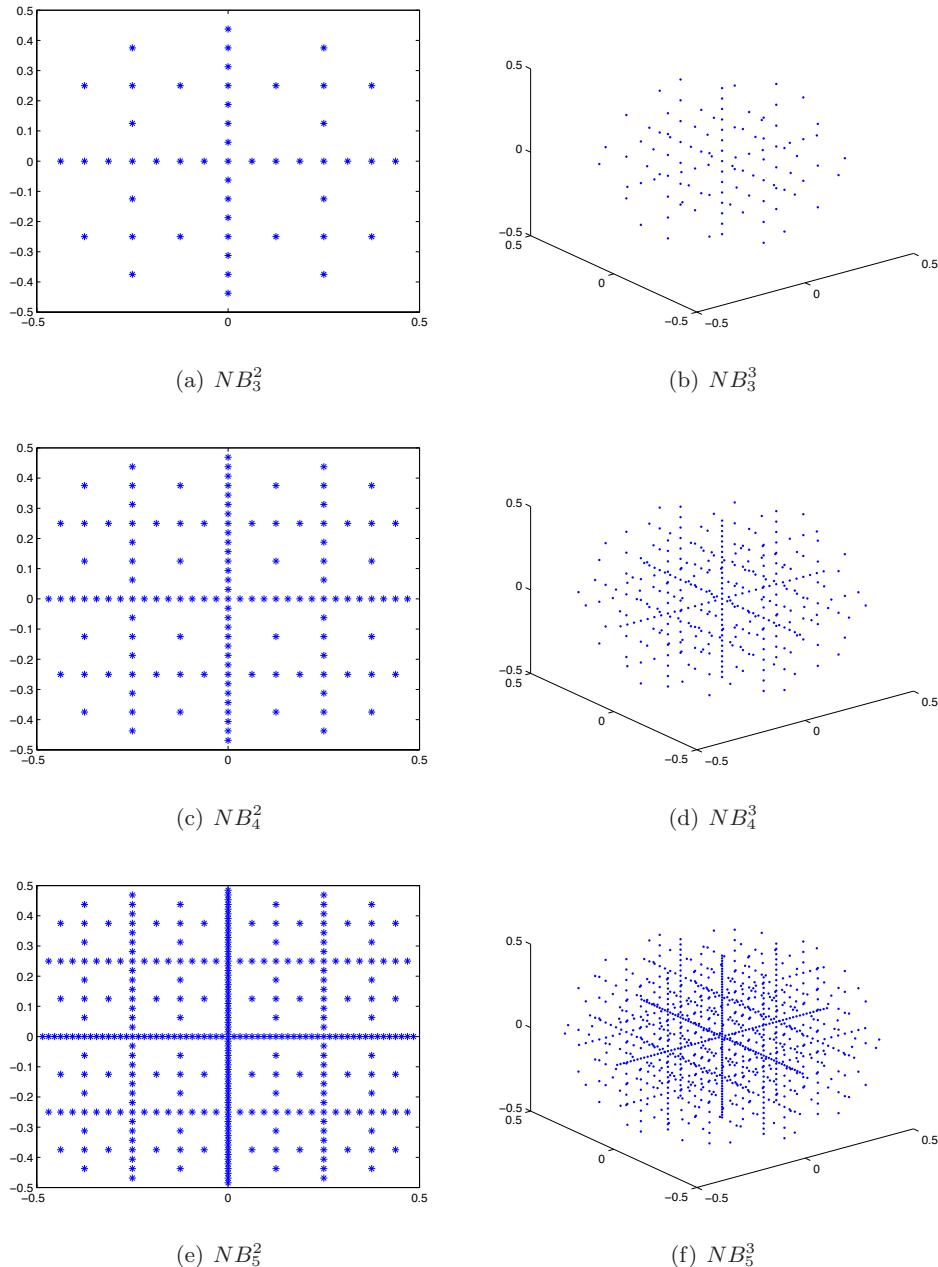


Abbildung 3.13: Zwei- und dreidimensionale Noboundary-Gitter mit Level 3 bis 5.

3.6.3 Das Clenshaw-Curtis-Gitter

Die zweite Alternative zum Maximumgitter ist das Clenshaw-Curtis Gitter: Hierbei ist die Überlegung, dass man auf die Randpunkte nicht ganz verzichtet, die Punkte auf dem Rand aber mit einer größeren Schrittweite berechnet werden, als die, bei denen mindestens eine Koordinate Null ist. Die restlichen Punkte im Inneren werden mit einer noch größeren Schrittweite als die auf dem Rand berechnet. Dieser Unterschied zum Maximumgitter wird allerdings erst im mehrdimensionalen Fall deutlich.

Definition 3.24 Eindimensionales Clenshaw-Curtis-Gitter

Das eindimensionale Clenshaw-Curtis-Gitter mit Level $l \in \mathbb{N}$ besteht aus der Menge

$$CC_l := \left\{ x_{i,l} = i2^{-l} : i \in I_l \right\}.$$

Für $l = 0$ gilt: $CC_0 = \{0\}$.

Die eindimensionalen Clenshaw-Curtis-Gitter mit Level $l > 0$ entsprechen den eindimensionalen Maximumgittern mit Level $l - 1$, es gilt also: $CC_l = M_{l-1}$.

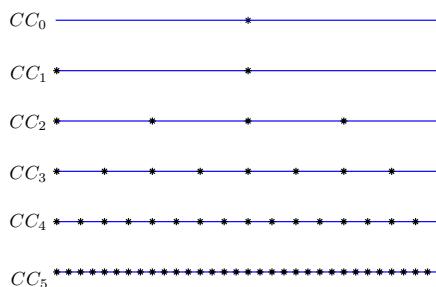


Abbildung 3.14: CC_l mit Level 0 bis 5.

Definition 3.25 d-dimensionales Clenshaw-Curtis-Gitter

Das d -dimensionale Clenshaw-Curtis-Gitter mit Level $n \in \mathbb{N}_0$ und Dimension $d > 1$ besteht aus der Menge

$$CC_n^d := \bigcup_{l_1 + \dots + l_d = n} CC_{l_1} \times CC_{l_2} \times \dots \times CC_{l_d}.$$

Abbildung 3.15 stellt die zwei- und dreidimensionalen Clenshaw-Curtis-Gitter für verschiedene Level dar.

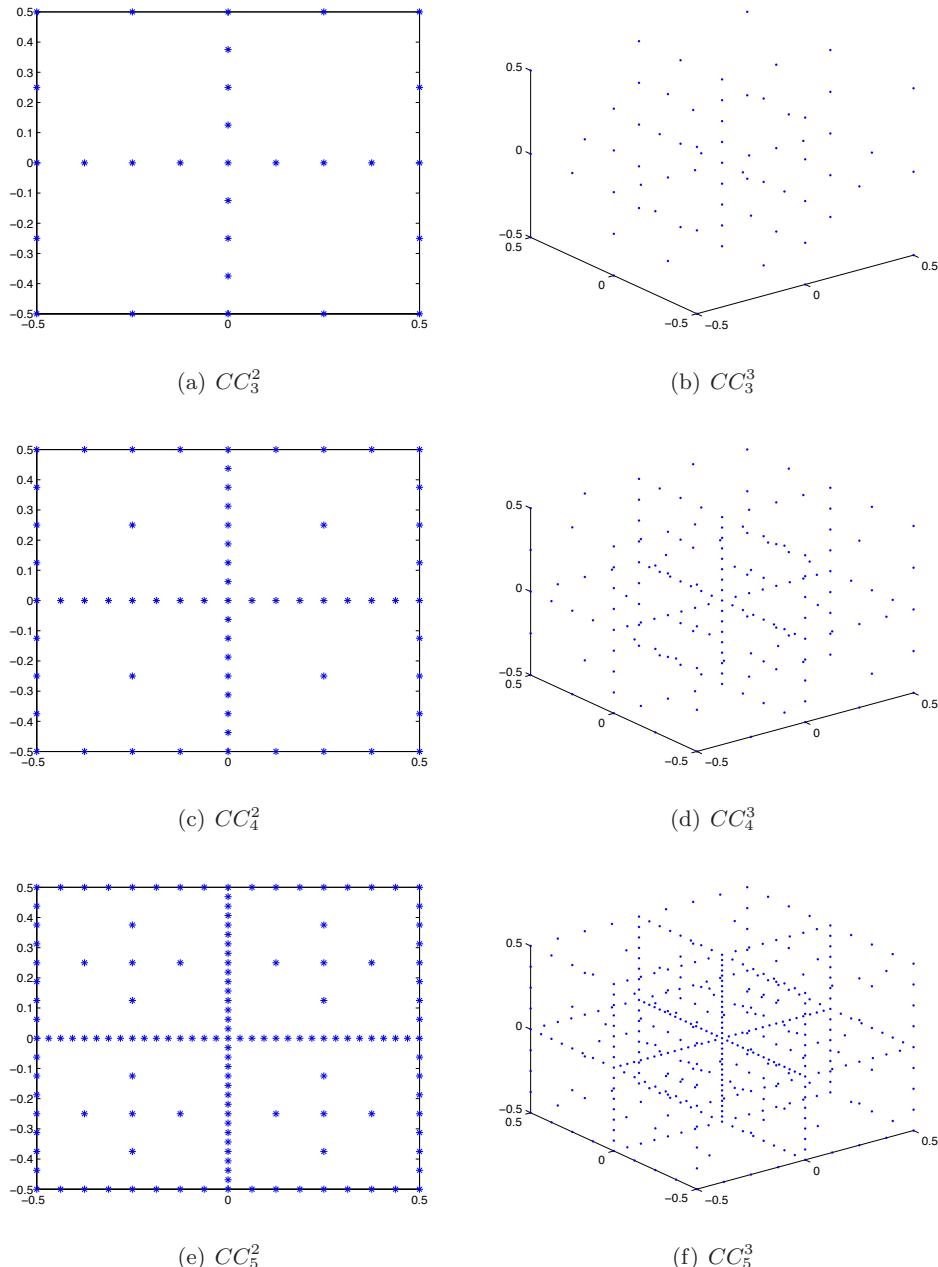


Abbildung 3.15: Zwei- und dreidimensionale Clenshaw-Curtis-Gitter mit Level 3 bis 5.

3.7 Gitter mit variablen Schrittweiten

Nachdem die Idee dünner Gitter, wie in der Einleitung zu diesem Kapitel erwähnt, aus dem Gebiet der Quadratur stammt, ist es naheliegend auch mit anderen, in der Quadratur üblichen, Stützknoten zu arbeiten. Neben den bislang verwendeten äquidistant verteilten Knoten mit Schrittweiten der Form $h_l = 2^{-l}$, sind vor allem die *Gaußpunkte* und die Extrema der *Tschebyscheff-Polynome* gebräuchlich. In den folgenden Unterkapiteln werden wir diese Stützstellen vorstellen und mit Hilfe der so entstehenden eindimensionalen Gitter, wie in Algorithmus 3.18 beschrieben, beliebig dimensionale dünne Gitter konstruieren.

3.7.1 Das Gauß-Legendre-Gitter

Wählt man die aus der Quadratur bekannten Gaußpunkte als Stützstellen für die stückweise linearen hierarchischen Basispolynome aus Kapitel 3.2, erhält man, im Eindimensionalen, Gitter mit variablen Schrittweiten.

Definition 3.26 Gaußpunkte

Die n Nullstellen des n -ten Legendrepolynoms $P_n(x) = \frac{1}{2^n n!} \frac{\partial^n (x^2 - 1)^n}{\partial x^n}$ heißen *Gaußpunkte*. Die Rekursionsformel zur Berechnung der Legendrepolynome lautet:

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_{n+1}(x) &= \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x) \text{ für } n \geq 1. \end{aligned}$$

Es gilt: Alle n Nullstellen von $P_n(x)$ sind reell und einfach und liegen im Intervall $(-1, 1)$.

Nachdem wir auch in Zukunft den Quader $[-0.5, 0.5]^d$ betrachten werden, müssen wir die Gaußpunkte durch die Transformation $T_{[-1,1] \rightarrow [-0.5,0.5]}$ mit $T_{[-1,1] \rightarrow [-0.5,0.5]}(x) = 0.5x$ auf $[-0.5, 0.5]$ transformieren.

Definition 3.27 Eindimensionales Gauß-Legendre-Gitter

Das eindimensionale Gauß-Legendre-Gitter mit Level $l \in \mathbb{N}_0$ besteht aus der Menge

$$GL_l := \left\{ \frac{1}{2}x : P_{2^l}(x) = 0 \right\} \cup GL_{l-1}.$$

Es gilt $GL_0 = \{0\}$.

Abbildung 3.16 zeigt eindimensionale Gauß-Legendre-Gitter:

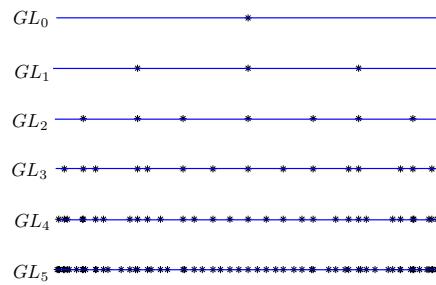


Abbildung 3.16: Eindimensionale Gauß-Legendre-Gitter mit Level 0 bis 5.

Definition 3.28 *d-dimensionales Gauß-Legendre-Gitter*

Das d -dimensionale Gauß-Legendre-Gitter mit Level $n \geq 0$ besteht aus der Menge

$$GL_n^d := \bigcup_{l_1 + \dots + l_d = n} GL_{l_1} \times GL_{l_2} \times \dots \times GL_{l_d}.$$

Abbildung 3.17 zeigt zwei- und dreidimensionale Gauß-Legendre-Gitter mit verschiedenen Level.

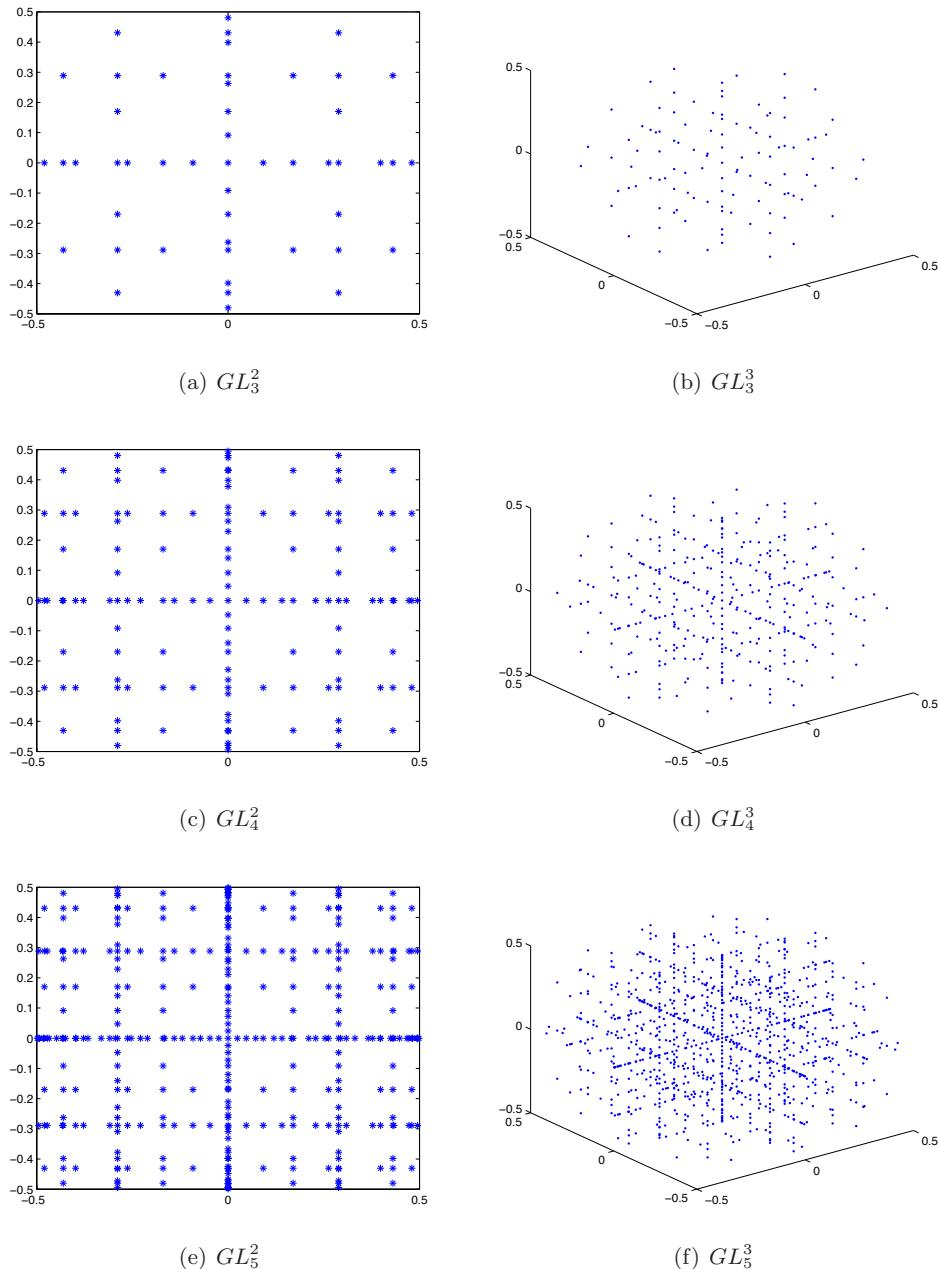


Abbildung 3.17: Zwei- und dreidimensionale Gauß-Legendre-Gitter mit Level 3 bis 5.

3.7.2 Das Tschebyscheff-Gitter

Verwendet man statt den Gaußpunkten die Extrema der Tschebyscheff Polynome erhält man ebenfalls ein Gitter mit nicht äquidistanten Gitterpunkten im eindimensionalen Fall.

Definition 3.29 Extrema der Tschebyscheff Polynome

Die Extrema $x_{\nu,n}$ der Tschebyscheff Polynome $T_n(x) = \cos(n \arccos(x))$ können für $x \in [-1, 1]$ und $n > 0$ mit

$$x_{\nu,n} = \cos\left(\frac{\nu\pi}{n}\right), \text{ mit } \nu = \{0, 1, 2, \dots, n\}$$

berechnet werden.

Damit wir uns weiterhin auf $[-0.5, 0.5]$ bewegen, müssen wir auch hier die Transformation $T_{[-1,1] \rightarrow [-0.5,0.5]}$ auf die $x_{\nu,n}$ anwenden.

Definition 3.30 Eindimensionales Tschebyscheff-Gitter

Das eindimensionale Tschebyscheff-Gitter mit Level $l \in \mathbb{N}_0$ besteht aus der Menge

$$TG_l := \left\{ x_{\nu,2^l} = 0.5 \cos\left(\frac{\nu\pi}{2^l}\right) : \nu = \{0, 1, 2, \dots, 2^l\} \right\}.$$

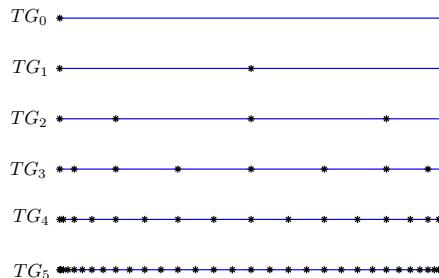


Abbildung 3.18: Eindimensionalen Tschebyscheff-Gitter mit Level 0 bis 5.

Definition 3.31 d-dimensionales Tschebyscheff-Gitter

Das d -dimensionale Tschebyscheff-Gitter mit Level n besteht aus der Menge

$$TG_n^d := \bigcup_{l_1+\dots+l_d=n} TG_{l_1} \times TG_{l_2} \times \dots \times TG_{l_d}.$$

Abbildung 3.19 zeigt zwei- und dreidimensionale Tschebyscheff-Gitter mit verschiedenen Level.

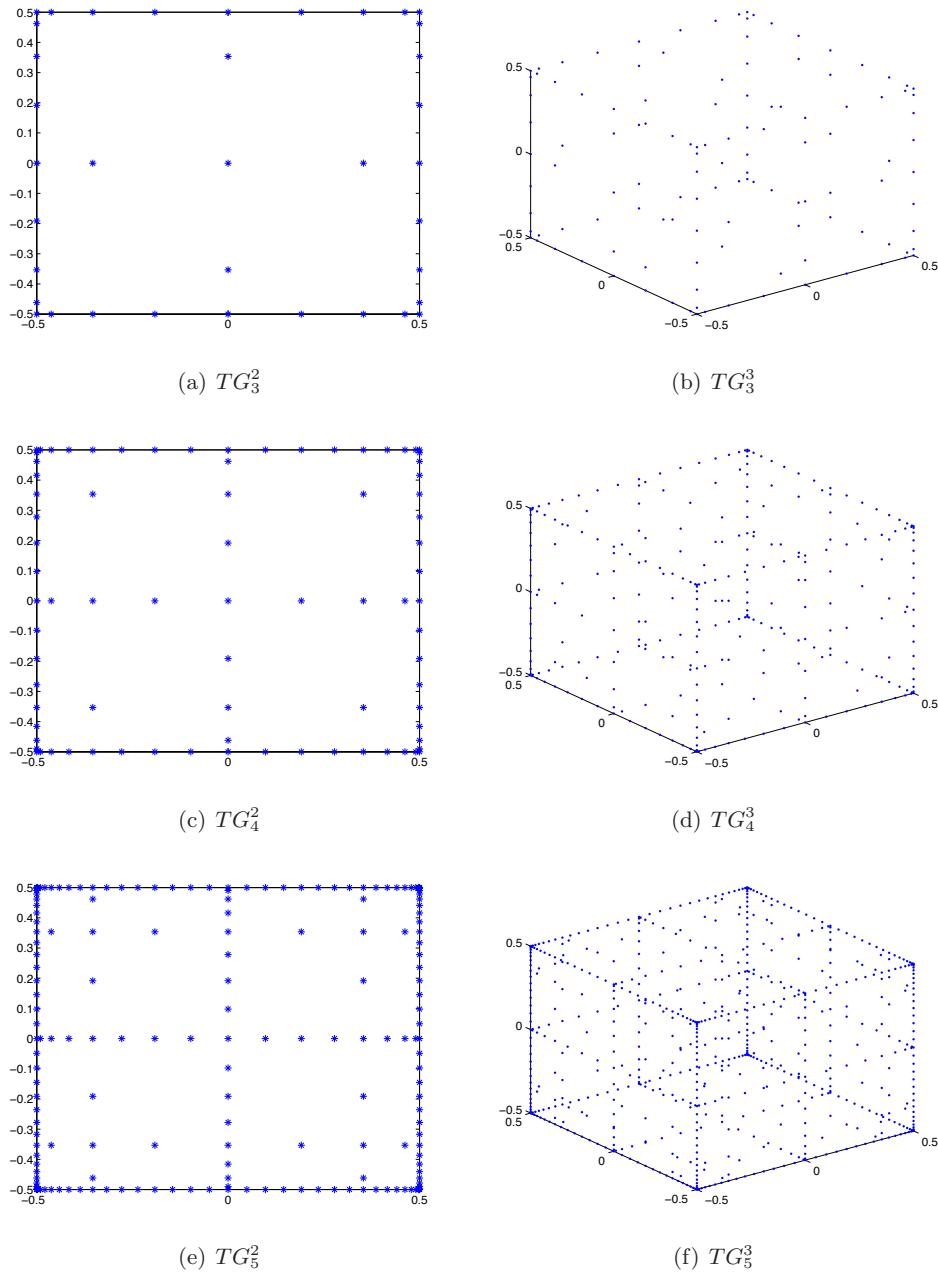


Abbildung 3.19: Zwei- und dreidimensionale Tschebyscheff-Gitter mit Level 3 bis 5.

3.7.3 Das kubische Gauß-Legendre-Gitter

Bei der numerischen Berechnung der Gaußpunkte muss eine Nullstellensuche durchgeführt werden. Diese wird in Matlab mit Hilfe der Berechnung der Eigenwerte der Begleitmatrix realisiert. Leider kommt es bei der Berechnung der Nullstellen des Legendrepolynoms mit einem Grad > 45 , d. h. ab einem Level von $l = 6$ im eindimensionalen Fall, zu numerischen Instabilitäten mit dem Ergebnis, dass komplexe Nullstellen ausgegeben werden. Daher werden wir einen neuen Gittertyp angeben, der ebenfalls auf den Gaußpunkten beruht, mit dem Unterschied, dass man von den Nullstellen des zweiten Legendrepolynoms,

$$P_2(x) = \frac{1}{8} \frac{\partial^2(x^2 - 1)^2}{\partial x^2},$$

ausgeht und diese bei der Erhöhung des Levels in die entstandenen Teilintervalle transformiert. Wir nennen das so entstandene Gitter kubisches Gauß-Legendre-Gitter, weil hier im eindimensionalen die Anzahl der Gitterpunkte ungefähr wie 3^{l+1} anwächst. Bei den bislang vorgestellten Gittertypen wächst die Anzahl der Gitterpunkte im eindimensionalen ungefähr wie 2^{l+1} .

Definition 3.32 Eindimensionales kubisches Gauß-Legendre-Gitter

Seien x_1 und x_2 die Nullstellen des 2-ten Legendrepolynoms. Sei des Weiteren $u = T_{[-1,1] \rightarrow [-0.5,0.5]}(x_1)$ und $v = T_{[-1,1] \rightarrow [-0.5,0.5]}(x_2)$. Dann ist das eindimensionale kubische Gauß-Legendre-Gitter mit Level 0 definiert als

$$\text{cubGL}_0 := \{u, v\}.$$

Man erhält nun die Punkte des eindimensionalen kubischen Gauß-Legendre-Gitters mit Level $l > 0$, indem man die Punkte $\{u, v\}$ in die 3^l Teilintervalle des Gitters cubGL_{l-1} zusammen mit $\{\pm 0.5\}$ transformiert. Zusammen mit den Punkten des Gitters cubGL_{l-1} bilden die so entstandenen Transformierten das kubische Gauß-Legendre-Gitter mit Level l .

Die eindimensionalen kubischen Gauß-Legendre-Gitter sind in Abbildung 3.20 dargestellt.

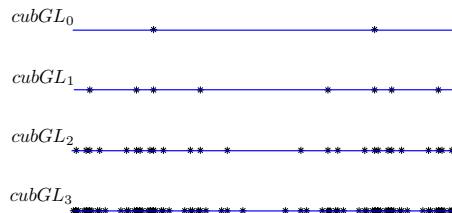


Abbildung 3.20: Eindimensionale kubische Gauß-Legendre-Gitter mit Level 0 bis 5.

Beispiel 3.33 Das kubische Gauß-Legendre-Gitter mit Level $l = 1$ entsteht wie folgt:

1. Es ist laut Definition 3.32 $cubGL_0 = \{u, v\}$, also $cubGL_0 \approx \{\pm 0.2887\}$.
2. Die entstandenen Teilintervalle sind: $t_1 = [-0.5, u]$, $t_2 = [u, v]$ und $t_3 = [v, 0.5]$.
3. Bezeichne nun $u_1 = T_{[-0.5, 0.5] \rightarrow t_1}(u)$, $v_1 = T_{[-0.5, 0.5] \rightarrow t_1}(v) \dots$
4. Dann ist das eindimensionale kubische Gauß-Legendre-Gitter mit Level 1 definiert als:

$$cubGL_1 = \{u_i, v_i; i = 1, 2, 3\} \cup \underbrace{\{u, v\}}_{cubGL_0}.$$

Definition 3.34 *d-dimensionales kubisches Gauß-Legendre-Gitter*

Das d -dimensionale kubische Gauß-Legendre-Gitter mit Level n besteht aus der Menge

$$cubGL_n^d := \bigcup_{l_1 + \dots + l_d = n} cubGL_{l_1} \times cubGL_{l_2} \times \dots \times cubGL_{l_d}.$$

Abbildung 3.21 stellt zwei- und dreidimensionale kubische Gauß-Legendre-Gitter mit verschiedenen Level dar.

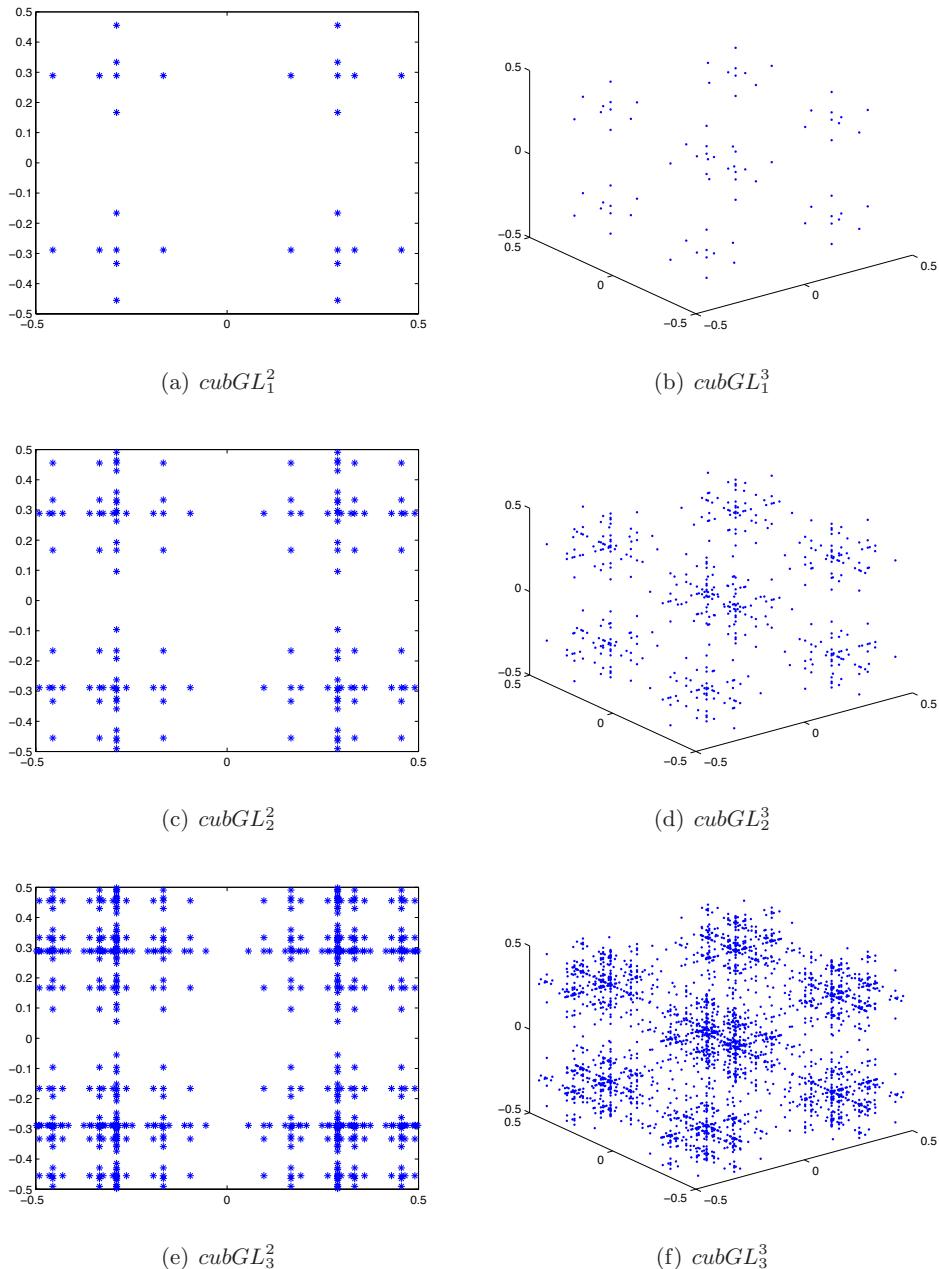


Abbildung 3.21: Zwei- und dreidimensionale kub. Gauß-Legendre-Gitter mit Level 1 bis 3.

3.8 Dünne Gitter und der HCP-Algorithmus

Die Motivation, dünne Gitter in der Optimierung zu nutzen, entstand durch den im 2. Kapitel vorgestellten HCP-Algorithmus 2.10. Nun bleibt zu beweisen, dass das durch den nicht adaptiven HCP-Algorithmus 2.10 (mit $\alpha = 1$) entstandene Gitter tatsächlich ein dünnes Gitter darstellt, nämlich das Clenshaw-Curtis-Gitter.

Satz 3.35 *Sei $X_k \subset [-0.5, 0.5]^d$ die Menge der vom HCP-Algorithmus 2.10 (mit den Parametern $\alpha = 1$, $\varepsilon = 0$, $k \geq 0$) bei der Lösung eines globalen Minimierungsproblems berechneten Punkte. Dann gilt: $X_k = CC_k^d$.*

Beweis:

Der nicht adaptive HCP-Algorithmus erzeugt sukzessive alle hyperbolic cross points mit einem Level $\leq k$. Das sind im eindimensionalen Fall alle $x \in [-0.5, 0.5]$ der Form

$$x = \pm \sum_{j=1}^m a_j 2^{-j} \text{ mit } a_j \in \{0, 1\}, m \leq k.$$

Die HCP bilden im eindimensionalen Fall ein äquidistantes Gitter auf $[-0.5, 0.5]$. Es gilt

$$\begin{aligned} X_0 &= \{0\}, \\ X_1 &= \{-0.5, 0, 0.5\}, \\ X_2 &= \{-0.5, -0.25, 0, 0.25, 0.5\} \dots \end{aligned}$$

Man kann die Menge der HCP mit Level $\leq k$ für $k > 0$ auch schreiben als:

$$X_k = \left\{ x_{i,k} = i 2^{-k}, i = \pm \{0, 1, 2, 3, \dots, 2^{k-1}\} \right\}.$$

Dies entspricht gerade dem eindimensionalen Clenshaw-Curtis-Gitter mit Level k , CC_k , siehe auch Definition 3.24.

Das mehrdimensionale Clenshaw-Curtis-Gitter mit Level k entsteht als kartesisches Produkt von eindimensionalen Clenshaw-Curtis-Gittern, deren Level sich in der Summe (über die Dimension) zu k ergeben. Das bedeutet, dass alle Punkte im Clenshaw-Curtis-Gitter Koordinaten der Form

$$x_{j,m_j} = i 2^{-m_j} \text{ mit } m_j \in \{1, \dots, k\} \text{ und } i \in \pm \{0, 1, 2, \dots, 2^{m_j-1}\}$$

mit

$$\sum_{j=1}^d m_j \leq k$$

besitzen. Das sind genau die HCP mit Level $\leq k$. \square

3.9 Konvergenz der Optimierung auf dünnen Gittern

Nun kann man mit Hilfe der rekursiven Methode einen einfachen kurzen Algorithmus zur Optimierung auf dünnen Gittern angeben. Der Nachteil dieses Verfahrens ist, dass die rekursive Methode für große Dimensionen und Level sehr lange Rechenzeiten benötigt, siehe auch die Tabelle der Rechenzeiten im nächsten Unterabschnitt. Daher eignet sich dieses Verfahren in der Praxis nur für Probleme mit $d \leq 10$ und $l \leq 8$, das sind für das Boundary-Gitter ca. 10^7 Punkte.

Algorithmus 3.36 Optimierung auf einem gegebenen Gitter

1. Erzeuge mit Algorithmus 3.18 ein d -dimensionales dünnes Gitter G_n^d mit Level n ;
($G_n^d = BG_n^d, M_n^d, NB_n^d, CC_n^d, GL_n^d, TG_n^d$ oder $cubGL_n^d$)
2. Berechne $F = f(G_n^d)$ und bestimme x^* mit $f^* = f(x^*) = \min_{y \in F}(F)$.

Im Folgenden zeigen wir, dass Algorithmus 3.36, mit dem Noboundary-Gitter mit Level n als Grundlage, gegen die globale Lösung von (GP) (ohne Ungleichungsnebenbedingungen) konvergiert. Dazu müssen wir zeigen, dass das Noboundary-Gitter für $n \rightarrow \infty$ in $[-0.5, 0.5]^d$ dicht liegt.

Satz 3.37 Für alle $x \in [-0.5, 0.5]^d$ gibt es eine Folge $(x_n)_{n \in \mathbb{N}} \subset \mathbb{R}^d$ mit $x_n \in NB_n^d$ und $\|x_n - x\|_\infty \rightarrow 0$ für $n \rightarrow \infty$.

Beweis:

Sei zunächst $d = 1$:

Das eindimensionale Noboundary-Gitter mit Level l ist laut Definition 3.22 ein auf $[-0.5, 0.5]$ äquidistantes Gitter mit Schrittweite $h = 2^{-(l+1)}$ ohne Randpunkte. Das heißt:

$$\text{dist}(x, NB_l^1) \leq 2^{-(l+1)} \rightarrow 0 \text{ für } l \rightarrow \infty.$$

Für $d > 1$ gilt:

Das d -dimensionale Noboundary-Gitter mit Level ld entsteht gerade als kartesisches Produkt der eindimensionalen Noboundary-Gitter mit Level l :

$$NB_{ld}^d = NB_l^1 \times NB_l^1 \times \dots \times NB_l^1.$$

Daher gilt für ein $x \in [-0.5, 0.5]^d$:

$$\text{dist}(x, NB_{ld}^d) \leq 2^{-(l+1)} \rightarrow 0 \text{ für } l \rightarrow \infty.$$

□

Bemerkung 3.38 Für die anderen Gittertypen gilt:

1. Da das Maximumgitter mit Level $n \in \mathbb{N}_0$ alle Punkte des Noboundary-Gitter mit Level n enthält, gilt die obige Abschätzung auch für das Maximumgitter.
2. Das Clenshaw-Curtis-Gitter mit Level $n_l = dl + 1$ (mit $l \in \mathbb{N}_0$) enthält alle Punkte des Noboundary-Gitter mit Level $n_l - d$ und liegt somit ebenfalls dicht in $[-0.5, 0.5]^d$.

3. Das Boundary-Gitter mit Level $n_l \geq d$ enthält alle Punkte des Noboundary-Gitters mit Level $n_l - d$ und liegt daher dicht in $[-0.5, 0.5]^d$.
4. Die Nullstellen der Legendrepolynome vom Grad n liegen für $n \rightarrow \infty$ dicht in $[-1, 1]$, siehe [Ch68]. Daher gilt, dass das d -dimensionale Gauß-Legendre-Gitter mit Level l für $d > 0$ und $l \rightarrow \infty$ ebenfalls dicht in $[-0.5, 0.5]^d$ ist.
5. Für ein eindimensionales Tschebyscheff-Gitter mit Level l gilt für ein beliebiges $x \in [-0.5, 0.5]$:

$$\text{dist}(x, TG_l) \leq \frac{1}{4} \cos\left(\frac{(2^{l-1}-1)\pi}{2^l}\right).$$

Für $l \rightarrow \infty$ gilt: $\frac{(2^{l-1}-1)\pi}{2^l} = (\frac{1}{2} - \frac{1}{2^l})\pi \rightarrow \frac{1}{2}\pi$. Mit $\cos(\pi/2) = 0$ gilt: $\text{dist}(x, TG_l) \rightarrow 0$ für $l \rightarrow \infty$. Das d -dimensionale Tschebyscheff-Gitter mit Level l ist als kartesisches Produkt der eindimensionalen Gitter für $l \rightarrow \infty$ ebenfalls dicht in $[-0.5, 0.5]^d$.

6. Sei $cubGL_0 = \{u, v\}$, dann gilt für das eindimensionale kubische Gauß-Legendre-Gitter mit Level l für ein beliebiges $x \in [-0.5, 0.5]$: $\text{dist}(x, cubGL_l) \leq 2^l v^{l+1}$. Diese Folge entsteht, wie man leicht nachrechnen kann, durch die sukzessive affine Transformation von $\{u, v\} = \{-v, v\}$ in das entstehende, um den Ursprung symmetrische, Teilintervall. Es gilt

$$\lim_{l \rightarrow \infty} (2^l v^{l+1}) = \lim_{l \rightarrow \infty} ((2v)^l v) = 0,$$

da $2v \approx 2 \cdot 0.2887 \approx 0.5774 < 1$. Somit ist das eindimensionale kubische Gauß-Legendre-Gitter mit Level $l \rightarrow \infty$ dicht in $[-0.5, 0.5]$. Damit sind auch die d -dimensionalen kubischen Gauß-Legendre-Gitter dicht in $[-0.5, 0.5]^d$, da diese als kartesisches Produkt der eindimensionalen Gitter entstehen.

Satz 3.39 Sei G_l^d ein beliebiges d -dimensionales dünnes Gitter mit Level l . Sei des Weiteren

$$x_l = \arg \min_{x \in G_l^d} f(x).$$

Dann gilt: Jeder Häufungspunkt \tilde{x} der Folge $(x_l)_l$ ist eine globale Lösung von (GP).

Beweis:

Bezeichne im Folgenden wieder f^* das globale Minimum der zu minimierenden Zielfunktion f und x^* die zugehörige Minimallösung. Betrachten wir die gegen \tilde{x} konvergierende Teilfolge $(y_l)_l$ von $(x_l)_l$. Dann gilt $f(y_{l+1}) \leq f(y_l)$ wegen $G_l^d \subset G_{l+1}^d$. Somit kann man zwei Fälle unterscheiden:

1. $f(y_l) \rightarrow f^*$, somit $\tilde{x} = x^*$.
2. $f(y_l) > f^*$ für alle l .

Im ersten Fall gilt die Behauptung. Im zweiten Fall gibt es wegen der Dichtheit des Gitters einen Punkt $\hat{x} \in G_l^d$, der sich zwischen \tilde{x} und der globalen Lösung x^* befindet. Wegen der Stetigkeit von f gilt hier: $f(\tilde{x}) > f(\hat{x}) > f(x^*)$. Dies steht im Widerspruch zur Definition der Folge (x_l) . Insgesamt gilt: $y_l \rightarrow \tilde{x} = x^*$.

□

3.10 Numerischer Vergleich der Gittertypen

Die folgenden Tabellen zeigen, wie viele Punkte ein Gitter eines bestimmten Typs mit Level l und Dimension d hat und wie lange die Erzeugung des Gitters mit Algorithmus 3.18 dauert. Alle Berechnungen wurden auf einem Intel Pentium M Prozessor mit 1.60GHz durchgeführt. Die Rechenzeiten sind in Sekunden angegeben und bezeichnen die tatsächlich abgelaufene Rechenzeit (realtime), also inklusive der Zeit die für die Speicherverwaltung, Funktionsauswertung etc. notwendig ist. Ein „-“ bedeutet, dass der Arbeitsspeicher (512MB RAM) zu klein war, um die entsprechende Berechnung durchzuführen und der Rechner daher auslagern musste. In so einem Fall müsste man die CPU-Zeit messen. Allerdings sind für die meisten Anwender die realen Rechenzeiten interessanter, da man wissen will, wie lange es wirklich dauert, etwas zu berechnen. Natürlich hängt dies auch von der Stärke des Rechners ab. Hier sieht man, wo die Grenzen der rekursiven Implementierung liegen. Somit eignet sich die rekursive Erzeugung dünner Gitter nur bei Problemen mit Dimension ≤ 10 und Level ≤ 8 . Dies ist, neben der Adaptivität, eine weitere Motivation für die Entwicklung einer alternativen, iterativen, Methode, dünne Gitter zu erzeugen. Bei der iterativen Erzeugung dünner Gitter werden wir uns auf 4 Typen beschränken. Wegen der starken Konzentration des Boundary-Gitters BG_l^d auf den Rand von $[-0.5, 0.5]^d$ werden wir dieses Gitter nicht weiter betrachten. Wie bereits erwähnt, treten bei der Suche nach den Nullstellen der Legendrepolynome höheren Grades numerische Instabilitäten auf, daher werden wir das Gauß-Legendre-Gitter ebenfalls außer Acht lassen. Wegen der starken Konzentration der Punkte des Tschebyscheff-Gitters auf die Ecken werden wir diesen Gittertyp ebenfalls nicht weiter verfolgen. Daher werden wir uns auf das Noboundary-, Maximum-, Clenshaw-Curtis- und kubische Gauß-Legendre-Gitter beschränken.

Gitter

Boundary-Gitter BG_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
0	2	4	8	16	32	64	128	256	0.000	0.015
1	3	8	20	48	112	256	576	1280	0.000	0.015
2	5	17	50	136	352	880	2144	5120	0.000	0.015
3	9	37	123	368	1032	2768	7184	18176	0.000	0.015
4	17	81	297	961	2882	8204	22472	59744	0.000	0.015
5	33	177	705	2441	7763	23288	66908	185888	0.000	0.015
6	65	385	1649	6065	20333	63953	191858	554768	0.000	0.015
7	129	833	3809	14801	52073	171053	533963	-	0.000	0.016
8	257	1793	8705	35585	130913	447713	1450193	-	0.000	0.016

49

Gitter

Maximugitter M_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
0	3	9	27	81	243	729	2187	6561	0.000	0.015
1	5	21	81	297	1053	3645	12393	41553	0.000	0.015
2	9	49	225	945	3753	14337	53217	193185	0.000	0.015
3	17	113	593	2769	12033	49761	198369	768609	0.000	0.015
4	33	49	111	7681	36033	159489	676161	-	0.000	0.015
5	65	257	1505	20481	102785	483201	-	-	0.000	0.017
6	129	577	3713	52993	282625	1403137	-	-	0.000	0.019
7	257	2817	1281	8961	133889	754945	-	-	0.000	0.022
8	-	-	-	-	-	-	-	-	0.000	0.047

Gitter

Noboundary-Gitter NB_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
0	1	1	1	1	1	1	1	1	0.000	0.015
1	3	5	7	9	11	13	15	17	0.000	0.015
2	7	17	31	49	71	97	127	161	0.000	0.015
3	15	49	111	209	351	545	799	1121	0.000	0.015
4	31	129	351	769	1471	2561	4159	6401	0.000	0.016
5	63	321	1023	2561	5503	10625	18943	31745	0.000	0.016
6	127	769	2815	7937	18943	40193	78079	141569	0.000	0.016
7	255	1793	7423	23297	61183	141569	297727	580865	0.000	0.094
8	511	4097	18943	65537	187903	471041	1066495	-	0.000	0.031

Gitter

Kapitel 3 Dimension Gitter

Gitter Clenshaw-Curtis-Gitter CC_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
02	0	1	1	1	1	1	1	1	1	0.000
	1	3	5	7	9	11	13	15	17	0.000
	2	5	13	25	41	61	85	113	145	0.000
	3	9	29	69	137	241	389	589	849	0.000
	4	17	65	177	401	801	1457	2465	3937	0.000
	5	33	145	441	1105	2433	4865	9017	15713	0.000
	6	65	321	1073	2929	6993	15121	30241	56737	0.000
	7	129	705	2561	7537	19313	44689	95441	190881	0.000
	8	257	1537	6017	18945	51713	127105	287745	609025	0.000

Gitter Gauß-Legendre-Gitter GL_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
02	0	1	1	1	1	1	1	1	1	0.000
	1	3	5	7	9	11	13	15	17	0.000
	2	7	17	31	49	71	97	127	161	0.000
	3	15	49	111	209	351	545	799	1121	0.000
	4	31	129	351	769	1471	2561	4159	6401	0.000
	5	63	321	1023	2561	5503	10625	18943	31745	0.016
	6	127	769	2815	7937	18943	40193	78079	141569	0.313
	7	255	1946	7701	23762	61897	142594	299125	-	0.328
	8	511	4482	20186	68437	193565	480876	-	-	0.843

Gitter Tschebyscheff-Gitter TG_l^d

Gitter	Anzahl der Punkte								Rechenzeit	
	$l \setminus d$	1	2	3	4	5	6	7	8	
02	0	2	4	8	16	32	64	128	256	0.000
	1	3	8	20	48	112	256	576	1280	0.000
	2	5	17	50	136	352	880	2144	5120	0.000
	3	9	37	123	368	1032	2763	7184	18176	0.000
	4	17	81	297	961	2882	8204	22472	59744	0.000
	5	33	177	705	2441	7763	23288	66908	185888	0.000
	6	65	385	1649	6065	20333	63953	191858	554768	0.000
	7	129	833	3809	14801	52073	171053	533963	1601888	0.000
	8	257	1793	8705	35585	130913	447713	1450193	-	0.000

Gitter	$l \setminus d$	Anzahl der Punkte								Rechenzeit							
		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
0	0	2	4	8	16	32	64	128	256	0.000	0.015	0.016	0.016	0.016	0.016	0.016	0.016
1	8	28	80	208	512	1216	2816	6400	0.000	0.015	0.016	0.016	0.031	0.062	0.125	-	-
2	26	136	512	1648	4832	13312	35072	89344	0.000	0.015	0.016	0.063	0.094	0.250	0.547	1.562	-
3	80	568	2672	10288	35072	110080	325376	918784	0.000	0.015	0.032	0.219	0.609	2.031	6.8130	23.093	-
4	242	2188	12392	55648	216512	763264	-	-	0.000	0.015	0.110	0.796	4.047	18.063	-	-	
5	728	8120	53216	273376	1196288	-	-	-	0.015	0.047	0.688	6.407	25.656	-	-	-	
6	2186	28432	216512	1253152	-	-	-	-	0.031	0.203	2.922	24.375	-	-	-	-	
7	6560	98416	846368	-	-	-	-	-	0.062	0.891	14.859	-	-	-	-	-	
8	19682	334612	3208328	-	-	-	-	-	0.188	3.755	69.369	-	-	-	-	-	

kubisches Gauß-Legendre-Gitter $cubGL_l^d$

Kapitel 4

Globale Optimierung mit dünnen Gittern

4.1 Einleitung

In diesem Kapitel wollen wir die im dritten Kapitel kennengelernten dünnen Gitter zur Optimierung nutzen. Wie wir im letzten Teil gesehen haben, stößt die rekursive Implementierung bei hochdimensionalen Problemen schnell an ihre Grenzen. Daher können wir mit dieser Methode nur sehr eingeschränkt arbeiten.

Der bislang formulierte Algorithmus zur Optimierung auf dünnen Gittern, Algorithmus 3.36, ist nur geeignet, wenn darüber hinaus die Funktionsauswertungen relativ wenig kosten. Ist dagegen eine Funktionsauswertung sehr teuer, will man adaptiv vorgehen und keine Punkte in Gebieten mit großen Funktionswerten betrachten. Für niedrig dimensionale Probleme werden wir, aufbauend auf Algorithmus 3.36, ein Verfahren angeben, welches dies leistet. Um auch hochdimensionale Probleme lösen zu können, werden wir anschließend, in Anlehnung an den HCP-Algorithmus, einen Algorithmus angeben, der iterativ ein dünnes Gitter erzeugt. Darüberhinaus wird durch die iterative Erzeugung Adaptivität ermöglicht.

4.2 Optimierung auf einem gegebenen Gitter

Will man ein niedrig dimensionales Optimierungsproblem lösen, bei dem des Weiteren die Funktionsauswertungen wenig Rechenzeit erfordern, bietet sich Algorithmus 3.36 an. Ist dagegen eine Funktionsauswertung relativ teuer, will man es vermeiden an allen Punkten eines dünnen Gitters auszuwerten. Mit Algorithmus 3.18 als Grundlage kann man ein Verfahren formulieren, welches in jedem Schritt adaptiv die Punkte bestimmt, an denen die Zielfunktion ausgewertet wird. Die Hauptidee ist, im ersten Schritt mit dem `sparsegrid`-Algorithmus 3.18 ein dünnes Gitter mit Level n (G_n^d) zu erzeugen und dieses in disjunkte Mengen \tilde{G}_k zu zerlegen, wobei die \tilde{G}_k , $k = \{0, \dots, n\}$, alle Punkte beinhalten, die zum dünnen Gitter mit Level k , aber nicht zum dünnen Gitter mit Level $k - 1$ gehören. Nun werden sukzessive die Punkte aus \tilde{G}_k bewertet, ohne dabei die Zielfunktion auszuwerten. Man verwendet lediglich die Information über die bereits ausgewerteten Punkte, den Abstand der zu bewertenden Punkte zu diesen und den Level der zu bewertenden Punkte. Nach der Bewertung wird die Zielfunktion nur an den besten Punkten ausgewertet.

Algorithmus 4.1 adaptive_sparsegrid

1. Input: $d \in \mathbb{N}$, die Dimension des Problems,
 $n \in \mathbb{N}$, der Level des zugrundeliegenden dünnen Gitters,
 $\alpha \in [0, 1]$, der Adaptivitätsparameter,
 $\text{maxPoints} \in \mathbb{N}$, die maximale Anzahl der Funktionsauswertungen;
2. Erzeuge mit Algorithmus 3.18 ein d - dimensionales dünnes Gitter G_n^d
mit Level n , (BG_n^d , M_n^d , NB_n^d , CC_n^d , oder $cubGL_n^d$);
3. Zerlege $G_n^d = \bigcup_{k=0}^n \tilde{G}_k$, mit \tilde{G}_k disjunkt: $\tilde{G}_k = G_k^d \setminus G_{k-1}^d$;
4. Setze $k = 1$, $X_Z = \{\}$;
5. $X_B = \{\tilde{G}_0\}$, $F_B = \{f(X_B)\}$, bestimme für alle $x \in X_B$ den $\text{Rang}(x)$;
6. $X_Z = \{\tilde{G}_k\} \cup \{X_Z \setminus X_B\}$;
7. Bestimme für alle $x \in X_Z$ den $\text{Level}(x)$;
8. Bewerte alle $x \in X_Z$ mit einer Bewertungsfunktion $\beta_\alpha(x)$,
(vgl. Bemerkung 4.2);
9. Sortiere alle $x \in X_Z$ nach ihrem Wert $\beta_\alpha(x)$;
10. Bestimme, wie viele Punkte maximal ausgewählt werden sollen:
 $A = \min\{|X_Z|, \text{maxPoints} - |X_B|, \text{round}(\alpha |X_Z| + 1)\}$;
11. $X_{Z_A} = \{x_1, \dots, x_A\} \subset X_Z$,
 $X_B = X_B \cup X_{Z_A}$,
 $F_B = F_B \cup \{f(X_{Z_A})\}$;
12. Falls $|X_B| < \text{maxPoints}$:
falls $\frac{|X_B|}{|X_Z|} \geq \alpha$: setze $X_Z = X_Z \setminus X_{Z_A}$ und gehe zu 7,
sonst: setze $k = k + 1$, gehe zu 6.;
13. Wähle $x^* \in X_B$ mit $x^* = \arg \min_{y \in F_B} (F_B)$.

Bemerkung 4.2

1. Die Adaptivität des Verfahrens wird wieder über den Parameter $\alpha \in [0, 1]$ gesteuert. Für $\alpha = 0$ erhält man ein rein adaptives Verfahren. Daneben steuert natürlich auch die obere Schranke für die Anzahl der Funktionsauswertungen, maxPoints , die Adaptivität des Verfahrens. Lässt man viele Funktionsauswertungen zu, werden natürlich mehr Gitterpunkte verwendet als bei wenigen Funktionsauswertungen, wenn man sich mehr auf die besser bewerteten Punkte konzentriert.
2. Die Bewertungsfunktion β_α wird so gewählt, dass zum einen Punkte bevorzugt werden, die nahe an den bisher bestimmten Punkten mit kleinen Funktionswerten sind. Zum anderen werden auch Punkte ausgewählt, die weit von den

bisher bestimmten Punkten entfernt sind. Daneben werden Punkte mit kleinem Level bevorzugt. Damit wird versucht, die Balance zwischen globaler und lokaler Suche zu finden. Ein mögliches Beispiel ist:

$$\beta_\alpha(x_i) = \alpha \left(\frac{1 - \|x_i - x_{B_i}\|}{\sum_{x_i \in X_Z} (1 - \|x_i - x_{B_i}\|)} + \frac{\text{Level}(x_i)}{n} \right) + (1 - \alpha) \frac{\text{Rang}(x_{B_i})}{|X_B|}$$

für alle $x_i \in X_Z$. Hier bezeichnet x_{B_i} den Punkt aus der Menge X_B , der sich am nächsten zu x_i befindet, also:

$$x_{B_i} = \arg \min_{x_k \in X_B} \|x_i - x_k\|.$$

Abbildung 4.1 veranschaulicht die Wirkung von α an Hand des Standardtestbeispiels BR (siehe Kapitel 8), die drei globalen Minima liegen in den blau eingekreisten Gebieten. Bei allen Beispielen wurde zunächst ein Noboundary-Gitter mit Level $n = 7$ erzeugt. Als Abbruchkriterium wirkt $\text{maxPoints} = 400$. Man sieht, dass die Punkte für größere α gleichmäßiger verteilt sind und für kleinere α dichter um die globalen Minima liegen.

Abbildung 4.2 zeigt das gleiche Beispiel mit den anderen Gittertypen als Grundlage für $\alpha = 0.5$. Das kubische Gauß-Legendre-Gitter wurde mit Level $n = 4$ erzeugt. Man sieht, dass mit dem allgemeinen dünnen Gitter, dem Maximum-Gitter und dem Clenshaw-Curtis-Gitter sehr viele Punkte am Rand des betrachteten Gebietes erzeugt werden. Das Ergebnis mit dem Gauß-Legendre-Gitter als Grundlage sieht für dieses Beispiel sehr gut aus, das Gitter ist aber für ein Beispiel, indem das globale Minimum in der Mitte des betrachteten Gebietes liegt (z. B. Testbeispiel C), nicht so gut geeignet, siehe Abbildung 4.3. Hier erzielt das Noboundary-Gitter ein wesentlich besseres Ergebnis.

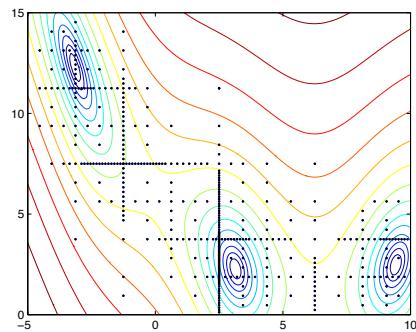
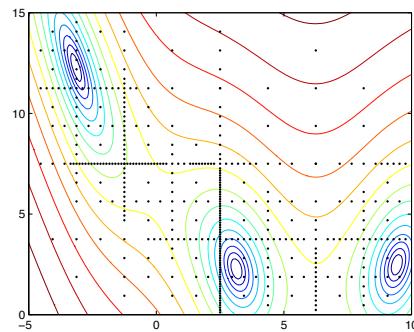
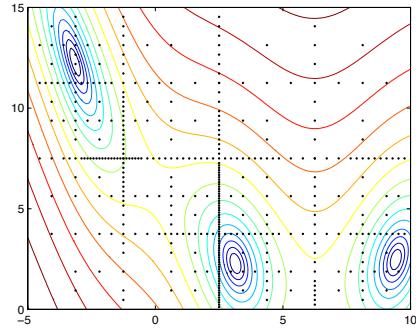
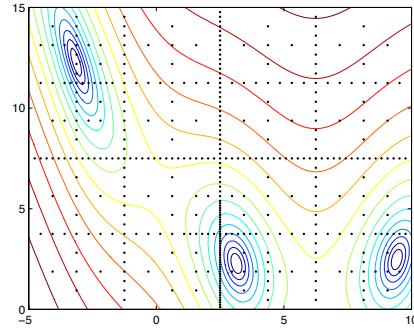
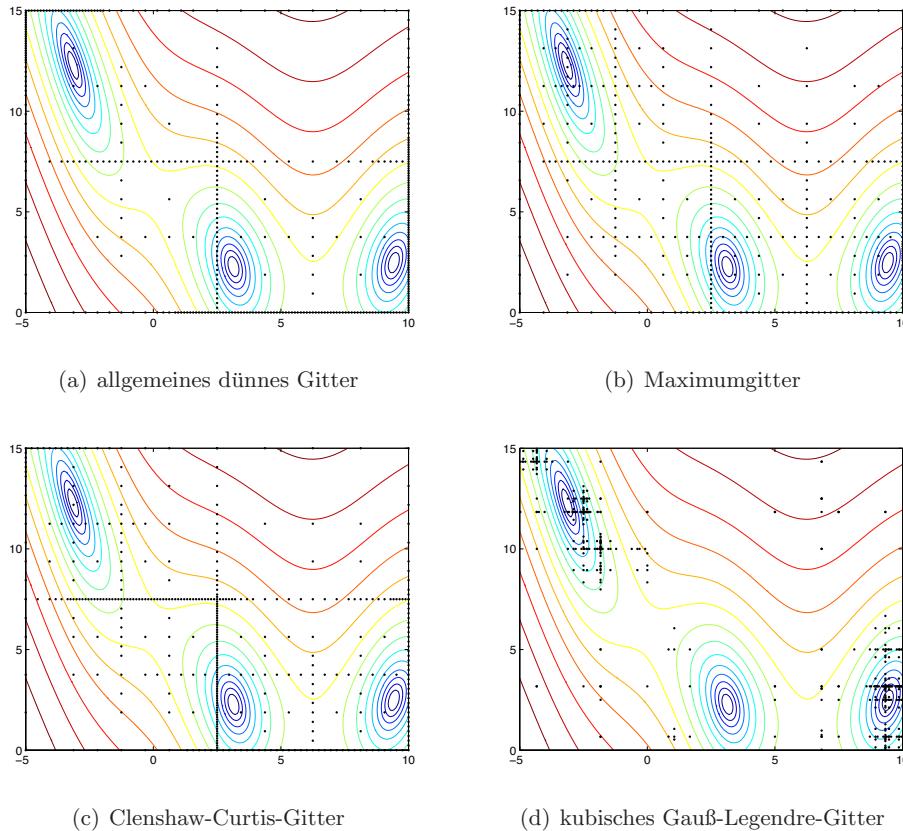
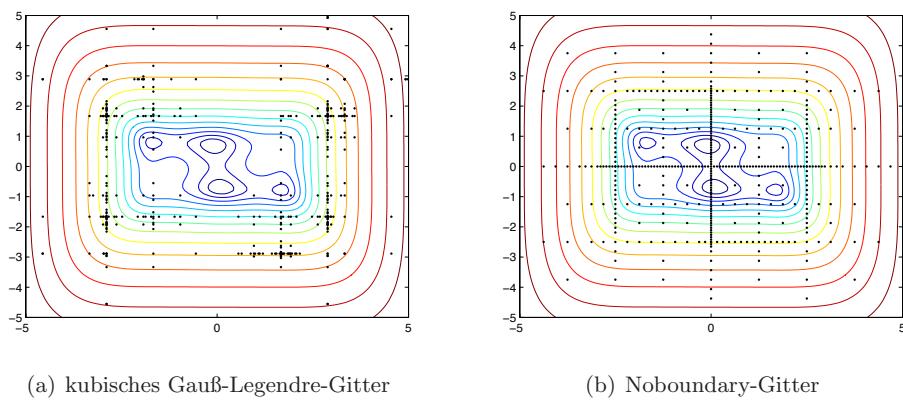
(a) $\alpha = 0.3$ (b) $\alpha = 0.5$ (c) $\alpha = 0.7$ (d) $\alpha = 0.9$

Abbildung 4.1: `adaptive_sparsegrid` mit dem Noboundary-Gitter für verschiedene α .

Abbildung 4.2: `adaptive_sparsegrid` mit den anderen Gittertypen und $\alpha = 0.5$.Abbildung 4.3: `adaptive_sparsegrid` für Testbeispiel C mit $\alpha = 0.5$.

4.3 Dünne Gitter als Bäume

Zur Darstellung dünner Gitter ist eine spezielle Datenstruktur, die Baumstruktur, besonders gut geeignet. Am einfachsten können Binäräbäume zur Abbildung des eindimensionalen Noboundary-Gitters herangezogen werden. Um die anderen Gittertypen darzustellen, müssen die Bäume an diese angepasst werden. Zum Beispiel werden die Randpunkte durch die Hinzunahme spezieller Knoten, die oberhalb der Wurzel geschrieben werden, einbezogen. Für das kubische Gauß-Legendre-Gitter sind Binäräbäume ungeeignet, da im Eindimensionalen jeder Knoten drei, statt zwei, Kinder hat. Mehrdimensionale Gitter können als ein System von Bäumen aufgefasst werden.

Der Vorteil dieser Interpretation dünner Gitter ist die eindeutige Identifikation eines Gitterpunkts mit einem Knoten eines speziellen Baumes. Mit Hilfe dieser Eins-zu-Eins Beziehung kann man leicht die Punkte bestimmen, die im nächst feineren Gitter Nachbarn eines betrachteten Gitterpunktes sind. Diese sind genau die Kinder des entsprechenden Knotens. Wie im HCP-Algorithmus können alle Punkte bewertet und in jedem Schritt die „Nachbarn“ des besten Punktes bestimmt werden. Adaptivität bedeutet nun, dass der Baum in den Gebieten wächst, in denen es die Funktionswerte erforderlich machen, siehe auch [Ze90] und [Bu92].

4.3.1 Bäume

Da es sich bei Bäumen um spezielle Graphen handelt, müssen wir zunächst einige Begriffe aus der Graphentheorie einführen.

Definition 4.3 Graph

Ein Graph \mathcal{G} ist ein geordnetes Paar (V, E) aus einer Menge V von Knoten und einer Menge E von Kanten. Auf E ist eine Abbildung, die Inzidenzfunktion, erklärt, die jedem Element von E eindeutig ein geordnetes oder ungeordnetes Paar von Elementen aus V zuordnet. Ist jedem Element von E ein ungeordnetes Paar zugeordnet, dann wird \mathcal{G} ein ungerichteter Graph genannt. Andernfalls spricht man von einem gerichteten Graphen.

Definition 4.4 Knotengrad

Sei $v \in V$ ein Knoten im Graph \mathcal{G} , dann bezeichnet man die Anzahl der mit v indizierten Kanten als den Knotengrad dieses Knotens.

Definition 4.5 Baum

Ein ungerichteter zusammenhängender Graph, in dem kein Kreis existiert, wird Baum genannt. Jeder Baum mit der Knotenzahl N hat genau $N-1$ Kanten.

Bemerkung 4.6 Zusammenhängend bedeutet, dass zwischen zwei beliebigen Knoten in \mathcal{G} ein Weg, d. h. eine Verbindung aus Kanten (eine Kantenfolge) besteht. Eine geschlossene Kantenfolge (d. h. der Anfangsknoten ist zugleich der Endknoten) wird Kreis genannt.

Definition 4.7 Wurzelbaum

Ein Baum mit einem ausgezeichneten Knoten wird Wurzelbaum genannt, der ausgezeichnete Knoten heißt Wurzel. Bei der graphischen Darstellung wird die Wurzel oben angeordnet, und die Wege werden von der Wurzel weggerichtet betrachtet. Wurzelbäume dienen zur Darstellung hierarchischer Strukturen.

Definition 4.8 regulärer binärer Baum

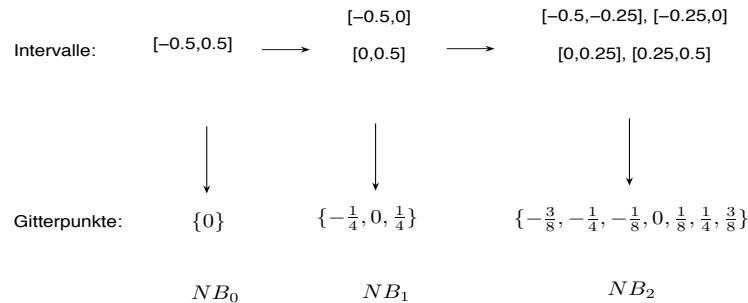
Hat ein Baum genau einen Knoten vom Grad 2 und sonst nur Knoten vom Grad 1 oder 3, dann wird er regulärer binärer Baum genannt. Das **Niveau** eines Knotens ist sein Abstand von der Wurzel. Das maximale auftretende Niveau wird **Höhe** des Baumes genannt.

Bemerkung 4.9 Im nachfolgenden Unterkapitel werden wir zwar von einem regulären Binärbaum ausgehen, diesen aber immer wieder verändern, damit er dem jeweiligen dünnen Gitter entspricht. Daher wird die Definition für das Niveau eines Knotens und die Höhe des Baumes dem jeweiligen Fall angepasst werden. Zur Abgrenzung werden wir ab hier statt des Niveaus den Begriff der *Tiefe* eines Knotens und statt der Höhe die *Tiefe* eines Baumes einführen.

4.3.2 Eindimensionale dünne Gitter als Bäume

Noboundary-Gitter

Im eindimensionalen Fall entsteht ein Noboundary-Gitter, indem man, ausgehend von $[-0.5, 0.5]$, in jedem Schritt die Mittelpunkte aller Teilintervalle bestimmt und diese zur Menge der Gitterpunkte hinzunimmt:



Dieser Vorgang kann als ein Baum der Tiefe n (n ist das Level des dünnen Gitters) dargestellt werden. Die Knoten des Baumes entsprechen den Gitterpunkten des eindimensionalen Noboundary-Gitters. Bezeichnet man in einem solchen Binärbaum den linken Ast mit einer [0] und den rechten Ast mit einer [1], kann man jeden Knoten eindeutig über die entsprechende Abfolge von Nullen und Einsen identifizieren, siehe Abbildung 4.4.

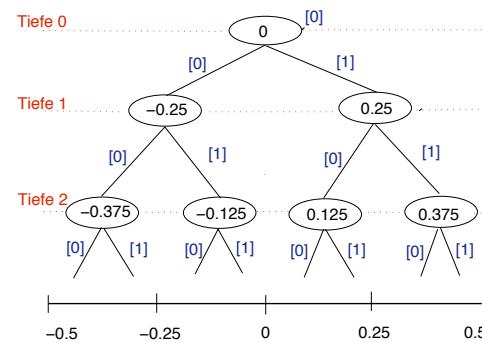


Abbildung 4.4: Baum für das eindimensionale Noboundary-Gitter.

Maximum-Gitter

Wenn man die Randpunkte mit einbezieht, wie beim Maximum-Gitter, muss man den Baum um zwei Knoten für die Randpunkte ergänzen. Diese stehen oberhalb der eigentlichen Wurzel und erhalten, wie auch der Knoten 0, die Tiefe 0, siehe Abbildung 4.5.

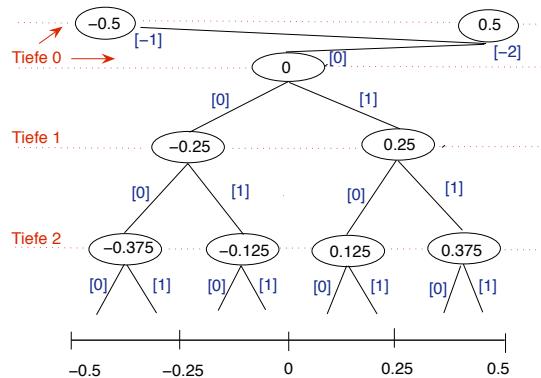


Abbildung 4.5: Baum für das eindimensionale Maximum-Gitter.

Clenshaw-Curtis-Gitter

Zur Interpretation des eindimensionales Clenshaw-Curtis-Gitters verwendet man den gleichen Baum wie für das Maximumgitter. Lediglich die Tiefen der Knoten ändern sich: Der Knoten 0 erhält die Tiefe 0, die Punkte ± 0.5 die Tiefe 1, die erste Generation (± 0.25) Tiefe 2 usw. Hier entspricht also die Tiefe eines Knotens genau dem Level des zugehörigen Punktes im Sinne von Definition 2.2, siehe Abbildung 4.6.

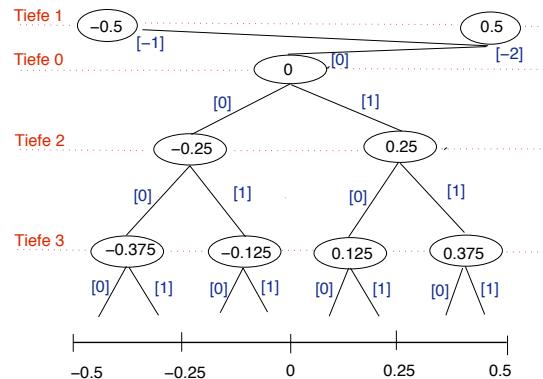


Abbildung 4.6: Baum für das eindimensionale Clenshaw-Curtis-Gitter.

Kubisches Gauß-Legendre-Gitter

Die Gitterpunkte des kubischen Gauß-Legendre-Gitters entstehen durch Transformation der beiden Ausgangspunkte, den Nullstellen des Legendrepolynoms 2. Grades, in die entstehenden Teilintervalle. Das bedeutet, man kann sie eindeutig identifizieren, wenn man weiß, in welchem Teilintervall sie sich befinden und ob es sich um die Transformation des ersten oder zweiten Punktes handelt. Dies verdeutlicht auch Abbildung 4.7. Somit benötigen wir eine aus den Ziffern 0,1 und 2 bestehende Folge, um jeden Punkt eindeutig zu bezeichnen. Dieses Vorgehen kann als Baum ohne Wurzel dargestellt werden, bei dem jeder Knoten 3 Kinder hat, siehe Abbildung 4.7.

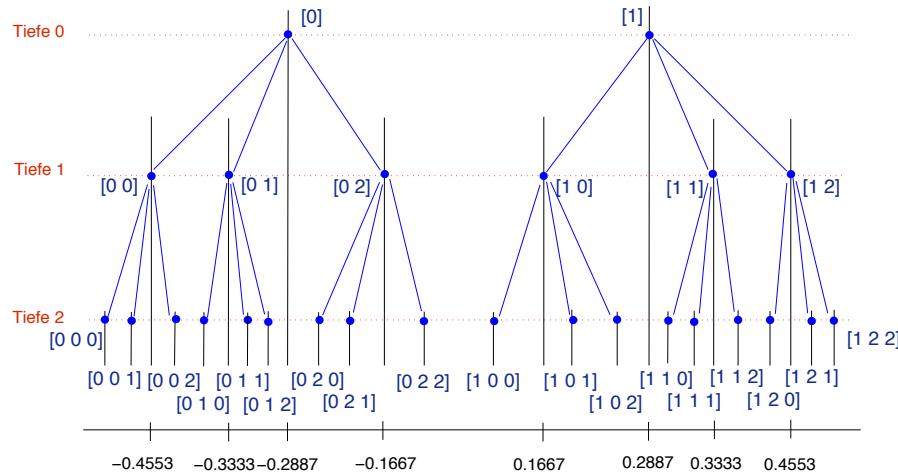


Abbildung 4.7: Das eindimensionale Legendre-Gitter.

4.3.3 Mehrdimensionale Gitter

Im zweidimensionalen Fall stellt jeder Knoten im Baum eine Linie im dünnen Gitter dar, siehe auch Abbildung 4.9. Für den Knoten 0 sind das z. B. alle Punkte der Form $(0, x)$, wobei x dem eindimensionalen dünnen Gitter (also einem Baum der entsprechenden Tiefe) entspricht. Die Tiefen müssen sich so zusammensetzen, dass sie in der Summe den gewünschten Level liefern, siehe Abbildung 4.8, vgl. dazu die Definitionen der einzelnen Gittertypen im dritten Kapitel.

Wie im eindimensionalen Fall kann man auch im zweidimensionalen Fall auf jeden Punkt im Gitter über eine Zeichenfolge zugreifen. Diese setzt sich nun aus zwei Komponenten zusammen, eine für die 1. und eine für die 2. Koordinate. Z. B. steht im zweidimensionalen Noboundary Gitter die Zeichenfolge $[0,0; 0,1,0]$ für den Punkt $(-0.25, -0.125)$. Für höher dimensionale Gitter kann dieses Vorgehen sukzessive weitergeführt werden.

Abbildung 4.9 stellt dar, wie jeder Punkt eines zweidimensionalen Maximumgitters als Knoten zweier Bäume interpretiert werden kann. Jeder Knoten im Baum (der

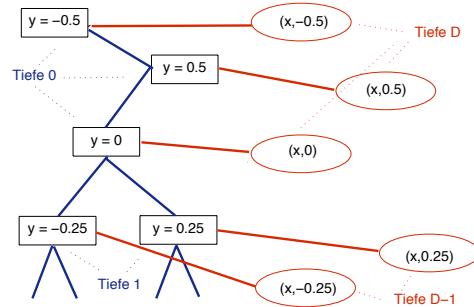


Abbildung 4.8: Binärbaum für das zweidimensionale Maximum-Gitter.

dem eindimensionalen Gitter entspricht) stellt eine Linie im mehrdimensionalen Gitter dar.

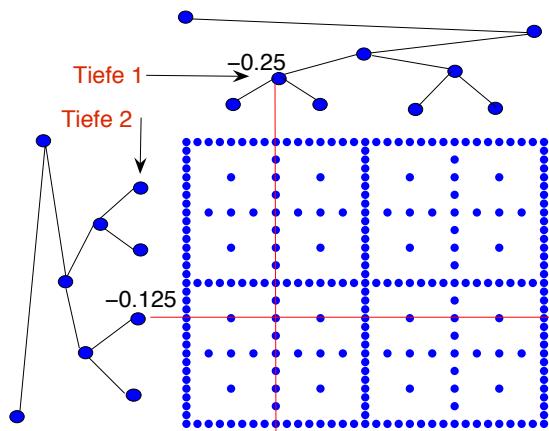


Abbildung 4.9: Zweidimensionales Maximumgitter mit Level 4.

4.4 Iterative Gittererzeugung

In Anlehnung an den HCP-Algorithmus aus dem zweiten Kapitel kann mit Hilfe der Baumstruktur ein Algorithmus beschrieben werden, nach dem iterativ ein dünnes Gitter erzeugt und zugleich ein globales Optimierungsproblem gelöst wird. In einem weiteren Schritt wird, analog zum HCP-Algorithmus, Adaptivität eingeführt.

Wesentlicher Bestandteil des HCP-Algorithmus ist, dass in jedem Schritt die Nachbarn des aktuell besten Punktes berechnet werden. Nachdem wir in Zukunft mit der Baumstruktur arbeiten wollen, müssen wir zunächst einige notwendige Definitionen formulieren. Im Folgenden bezeichnet G_n^d ein d -dimensionales dünnes Gitter mit Level n ($G_n^d = NB_n^d, M_n^d, CC_n^d$ oder $cubGL_n^d$).

Definition 4.10 Level eines Punktes im dünnen Gitter G_n^d

Der Level eines Punktes in einem dünnen Gitter entspricht dem Level des gröbsten Gitters, in dem er sich befindet. Übertragen auf die Baumstruktur entspricht der Level eines Punktes genau der Tiefe des zugehörigen Knotens (im entsprechenden eindimensionalen Baum).

Definition 4.11 Position eines Punktes $x \in G_n^d$

Die Position $Pos(x)$, eines Punktes x in einem dünnen Gitter, entspricht der Ziffernfolge, die jedem Knoten des entsprechenden Baumes zugeordnet ist. Im mehrdimensionalen Fall besteht die Position aus separaten Ziffernfolgen für alle Koordinaten. Die Länge der Ziffernfolge minus 1 ist gleich der Tiefe des Knotens.

Definition 4.12 Nachbarn eines Punktes im dünnen Gitter G_n^d

Die Nachbarn der Stufe m eines Punktes im dünnen Gitter G_n^d sind genau die Punkte, die sich im dünnen Gitter G_{n+m}^d , mit Level $n+m$, am nächsten zum betrachteten Punkt befinden (siehe auch Bemerkung 4.19).

Die aus dem zweiten Kapitel bekannten Begriffe *Grad* und *Rang* eines Punktes bleiben wie in Definition 2.8 und 2.7 festgelegt. Zusammen mit den oben definierten Begriffen der Nachbarn und Level auf der Grundlage der in Kapitel 4.3 eingeführten Bäume, kann man einen Algorithmus formulieren, der iterativ ein dünnes Gitter erzeugt. Die neuen Gitterpunkte werden als Nachbarn der schon bestehenden Gitterpunkte bestimmt. Dies geschieht mit Hilfe der Positionsvektoren der Ausgangspunkte und wird nach der Formulierung des Algorithmus genau erklärt.

Algorithmus 4.13 iterative_sparsagrid

1. Input: $d \in \mathbb{N}$, die Dimension des Problems,
 $\text{maxPoints} \in \mathbb{N}$, die maximale Anzahl der Funktionsauswertungen;
2. Bestimme die Startpunkte $X_0 = G_0^d$ und deren Positionen
 $Pos(x) \quad \forall x \in X_0$;
3. Es gilt: $Level(x) = Grad(x) = 0 \quad \forall x \in X_0$;
4. Setze $X = \{X_0\}$, $k = 0$ und wähle irgendein $\hat{x} \in X$;
5. Falls $|X| < \text{maxPoints}$:
 - (a) $Grad(\hat{x}) = Grad(\hat{x}) + 1$;

(b) $k = k + 1$, gehe zu 6.;

Falls $|X| \geq \text{maxPoints}$: gehe zu 10.;

6. Bestimme, ausgehend von $\text{Pos}(\hat{x})$, die Position der Nachbarn der Stufe 1 von \hat{x} , P_{neu} und mit diesen die Nachbarn $X_{neu} = \text{PosToReal}(P_{neu})$;

7. $X = X \cup X_{neu}$;

8. $\text{Grad}(x) = 0$, $\text{Level}(x) = \text{Level}(\hat{x}) + 1 \quad \forall x \in X_{neu}$;

9. Bewerte alle $x \in X$ mit

$$\beta_1(x) = \text{Level}(x) + \text{Grad}(x)^1,$$

und wähle $\hat{x} = \arg \min_{x \in X} (\beta_1(x))$,

Gehe zu 5.;

10. Die Lösung ist $\tilde{x} = \arg \min_{x \in X} (f(x))$.

Im 6-ten Schritt des Algorithmus 4.13 werden zunächst die Positionen der Nachbarpunkte bestimmt. Hierbei muss Folgendes beachtet werden:

1. Im Noboundary-Gitter unterscheidet sich die Position der Nachbarpunkte von \hat{x} nur in einer Koordinate von $\text{Pos}(\hat{x})$. In jeder Koordinatenrichtung entsteht die neue Position der beiden Nachbarn der Stufe 1 durch das Anhängen einer 1 und einer 0 an die bestehende Ziffernfolge. Die beiden Nachbarn x_{neu_1} und x_{neu_2} von \hat{x} , haben in der i-ten Koordinatenrichtung folgende Positionen:
 $\text{Pos}(x_{neu_1})_i = [\text{Pos}(\hat{x})_i, 0]$, $\text{Pos}(x_{neu_1})_j = [\text{Pos}(\hat{x})_j] \quad \forall j \neq i$
 $\text{Pos}(x_{neu_2})_i = [\text{Pos}(\hat{x})_i, 1]$, $\text{Pos}(x_{neu_2})_j = [\text{Pos}(\hat{x})_j] \quad \forall j \neq i$.
Dies verdeutlicht auch Beispiel 4.14 und Abbildung 4.10.

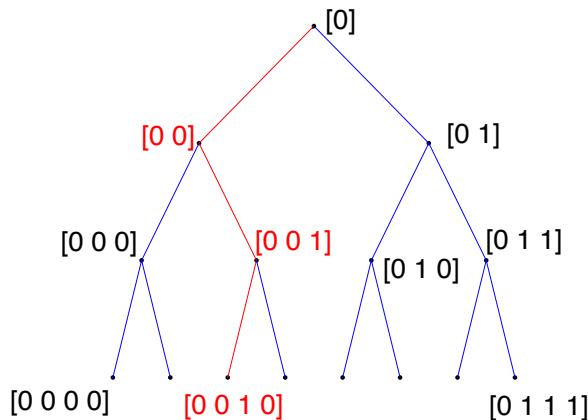


Abbildung 4.10: Gitterpunkte und ihre Positionen im Binärbaum.

¹Dies entspricht der Bewertungsfunktion des HCP-Algorithmus 2.10 mit $\alpha = 1$

2. Bei der Erzeugung des Maximumgitters muss man beachten, dass ein Randpunkt nicht in alle Richtungen Nachbarn haben kann. Es kann bei dem Punkt mit Position [-1] nur die 0 und bei einem Punkt mit Position [-2] nur die 1 angehängt werden.
3. Beim Clenshaw-Curtis-Gitter muss beachtet werden, dass ein innerer Punkt Nachbarn im Inneren und auf dem Rand von $[-0.5, 0.5]^d$ haben kann, während ein Randpunkt nur Nachbarn im Inneren und in den Ecken hat. Der Grund hierfür ist die spezielle Struktur des Clenshaw-Curtis-Gitters, dass die Punkte auf dem Rand mit einer größeren Schrittweite verteilt sind als die Punkte, bei denen mindestens eine Koordinate Null ist.
4. Bei der Erzeugung des kubischen Gauß-Legendre-Gitters muss berücksichtigt werden, dass ein Gitterpunkt nicht nur zwei, sondern drei Nachbarn in jeder Koordinatenrichtung hat, siehe auch Abbildung 4.11. Daher muss an den Positionsvektor des Ausgangspunktes \hat{x} in jeder Koordinate einmal eine Null, einmal eine Eins und einmal eine Zwei angehängt werden, während die restlichen Koordinaten gleich bleiben, siehe hierfür auch Abbildung 4.7. Also haben die drei Nachbarn der Stufe 1 in der i-ten Koordinatenrichtung folgende Positionen:

$$\begin{aligned} \text{Pos}(x_{neu1})_i &= [\text{Pos}(\hat{x})_i, 0], \text{Pos}(x_{neu1})_j = [\text{Pos}(\hat{x})_j] \quad \forall j \neq i, \\ \text{Pos}(x_{neu2})_i &= [\text{Pos}(\hat{x})_i, 1], \text{Pos}(x_{neu2})_j = [\text{Pos}(\hat{x})_j] \quad \forall j \neq i, \\ \text{Pos}(x_{neu3})_i &= [\text{Pos}(\hat{x})_i, 2], \text{Pos}(x_{neu3})_j = [\text{Pos}(\hat{x})_j] \quad \forall j \neq i. \end{aligned}$$

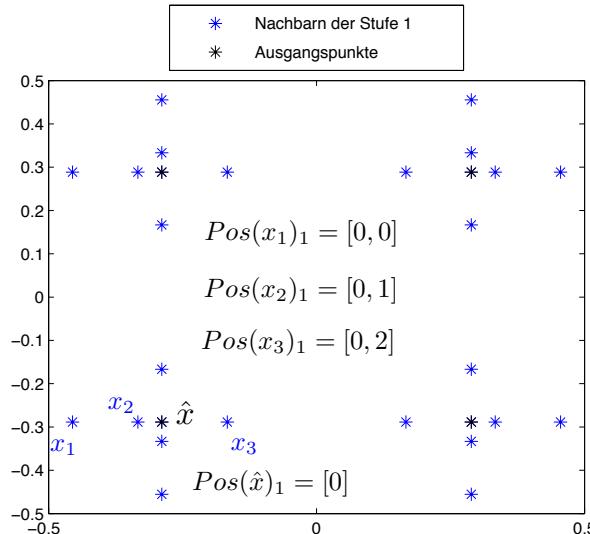


Abbildung 4.11: Die Nachbarn der Punkte im kubischen Gauß-Legendre-Gitter für $d = 2$.

Beispiel 4.14 Zur Berechnung der Nachbarn in Schritt 5. von Algorithmus 4.13 betrachten wir folgendes zweidimensionales Beispiel für das Noboundary-Gitter:

Gestartet wird mit $x = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, x hat die Position $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$,

die Nachbarn von x haben die Positionen

$$\begin{bmatrix} 00 \\ 0 \end{bmatrix}, \begin{bmatrix} 01 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 00 \end{bmatrix}, \begin{bmatrix} 0 \\ 01 \end{bmatrix},$$

das sind die Punkte

$$\begin{pmatrix} -0.25 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.25 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.25 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.25 \end{pmatrix}.$$

Sei nun $\begin{pmatrix} -0.25 \\ 0 \end{pmatrix}$ der neue beste Punkt, die Nachbarn haben die Positionen

$$\begin{bmatrix} 000 \\ 0 \end{bmatrix}, \begin{bmatrix} 001 \\ 0 \end{bmatrix}, \begin{bmatrix} 00 \\ 00 \end{bmatrix}, \begin{bmatrix} 00 \\ 01 \end{bmatrix},$$

das sind die Punkte

$$\begin{pmatrix} -0.375 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.125 \\ 0 \end{pmatrix}, \begin{pmatrix} -0.25 \\ -0.25 \end{pmatrix}, \begin{pmatrix} -0.25 \\ 0.25 \end{pmatrix} \dots$$

Bemerkung 4.15 Die Funktion $\text{PosToReal}: \mathbb{Z}^{d \times l} \rightarrow \mathbb{R}^d$ weist einer Position den zugehörigen Knoten im Baum zu. Auch hier muss nach Gittertyp unterschieden werden.

1. Für das Noboundary-, Maximum, Clenshaw-Curtis- und das Boundary-Gitter arbeitet PosToReal wie folgt:

Algorithmus 4.16 $u = \text{PosToReal}(v)$

1. Falls $\text{length}(v) = 1$:
 - (a) Falls $v = -1 : u = -0.5$;
 - (b) Falls $v = -2 : u = 0.5$;
 - (c) Falls $v = 0 : u = 0$;
 2. sonst
 - (a) Setze $u_1 = -0.5$ und $u_2 = 0$;
 - (b) Für $k = 2 : \text{length}(v)$:
 - i. Falls $v(k) = 1$:

$$u_3 = u_2 + 0.5 |u_2 - u_1|,$$

$$u_1 = u_2,$$

$$u_2 = u_3;$$
 - ii. Falls $v(k) = 0$:

$$u_3 = u_2 - 0.5 |u_2 - u_1|,$$

$$u_1 = u_2,$$

$$u_2 = u_3;$$
- $u = u_2$.

Siehe auch Beispiel 4.18.

2. Für das kubische Gauß-Legendre-Gitter mit $cubGL_0 = \{x, y\}$ gilt:

Algorithmus 4.17 $u = \text{PosToReal}(v)$

1. Falls $v(1) = 0$:
 - (a) $u = x$;
 - (b) $P = 'x'$;
2. Falls $v(1) = 1$:
 - (a) $u = y$;
 - (b) $P = 'y'$;
3. Setze $\text{bounds} = [-0.5, 0.5]$, $k = 2$;
4. Falls $k < \text{length}(v)$: gehe zu 5.;
5. Falls $P = 'x'$:
 - (a) Falls $v(k) = 0$:
 $\text{bounds} = [\text{bounds}(1), x];$
 - (b) Falls $v(k) = 1$:
 $\text{bounds} = [\text{bounds}(1), x],$
 $P = 'y'$;
 - (c) Sonst:
 $\text{bounds} = [x, y];$
6. Falls $P = 'y'$:
 - (a) Falls $v(k) = 0$:
 $\text{bounds} = [x, y];$
 - (b) Falls $v(k) = 1$:
 $\text{bounds} = [y, \text{bounds}(2)],$
 $P = 'x'$;
 - (c) Sonst:
 $\text{bounds} = [y, \text{bounds}(2)];$
7. $x = T_{[-0.5, 0.5] \rightarrow \text{bounds}}(x),$
 $y = T_{[-0.5, 0.5] \rightarrow \text{bounds}}(y);$
8. Falls $P = 'x'$:
 $u = x,$
 Sonst:
 $u = y;$
9. Setze $k = k + 1$, gehe zu 4.

Beispiel 4.18 Sei $v = [0, 0, 1, 1]$ im Noboundary-Gitter. Dann bedeutet $v(2) = 0$, dass der gesuchte Punkt in $] -0.5, 0 [$ liegt, $v(3) = 1$ verkleinert das Intervall auf $] -0.25, 0 [$, $v(4) = 1$ bezeichnet den rechten Sohn von -0.125 , also -0.0625 .

Bemerkung 4.19 Übertragen auf die Baumstruktur bedeutet Definition 4.12: Die Nachbarn der Stufe 1 eines Punktes sind die Kinder des entsprechenden Knotens im zugehörigen Baum. Die Nachbarn der Stufe m eines Gitterpunktes erhält man, indem man im entsprechenden Baum, ausgehend vom linken Kind $(m - 1)$ -mal dem rechten Kind und ausgehend vom rechten Kind $(m - 1)$ -mal dem linken Kind

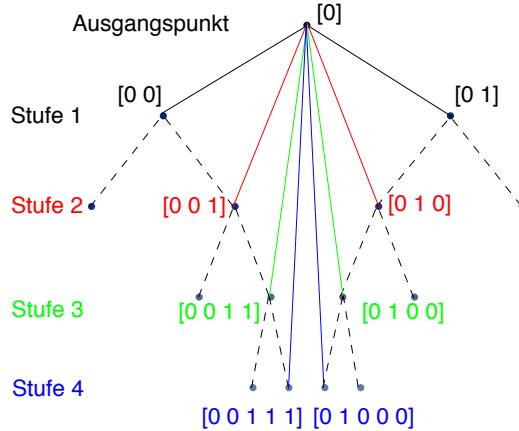


Abbildung 4.12: Nachbarn verschiedener Stufen im Noboundary-Gitter.

folgt. Für das Noboundary-Gitter und die verwandten Gittertypen sind dann auf der linken Seite der rechte Knoten und auf der rechten Seite der linke Knoten die Nachbarn der Stufe m , siehe auch Abbildung 4.12.

Für das kubische Gauß-Legendre-Gitter bilden im letzten Schritt die drei nächstgelegenen Knoten die Nachbarn der Stufe m . Hier muss man unterscheiden ob es sich beim Ausgangspunkt um die Transformation des linken oder rechten Punktes handelt (zur Erinnerung: $cubGL_0 = \{u, v\}$). Im Fall, dass es sich beim Ausgangspunkt um den linken Punkt (u) handelt, bilden die zwei Knoten auf der linken und der Knoten auf der rechten Seite die Nachbarn der Stufe m , siehe auch Abbildung 4.13. Im anderen Fall bilden die zwei Knoten auf der rechten Seite und der Knoten auf der linken Seite die Nachbarn der Stufe m .

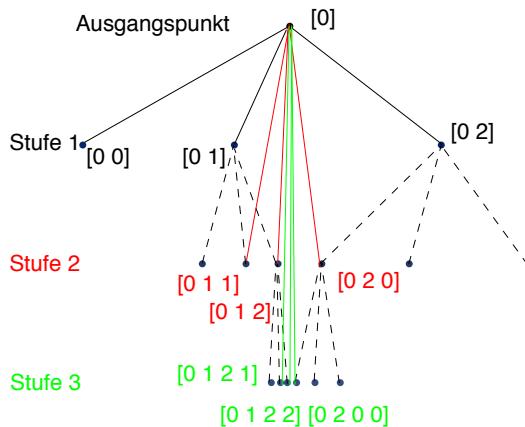


Abbildung 4.13: Nachbarn verschiedener Stufen im kub. Gauß-Legendre-Gitter.

Bislang haben wir einen Algorithmus entwickelt, der Schritt für Schritt ein dünnes Gitter erzeugt und damit ein ähnliches Ergebnis liefert wie der nichtadaptive HCP-Algorithmus. Wir haben darüberhinaus die Möglichkeit aus verschiedenen Gittertypen zu wählen und ggf. z. B. auf eine Auswertung am Rand des gegebenen Quaders zu verzichten. Als nächstes wollen wir auch die Adaptivität wie beim HCP-Algorithmus umsetzen. Für unseren Fall bedeutet dies: Der Baum wächst nur dort in die Tiefe, wo es die Funktionswerte erforderlich machen.

4.5 Adaptive Optimierung auf dünnen Gittern

Ausgehend von Algorithmus 4.13 kann man, analog zum HCP-Algorithmus, Adaptivität mit der Einführung einer parameterabhängigen Bewertungsfunktion β_α umsetzen, siehe hierzu Definition 2.9. Wir müssen Algorithmus 4.13 lediglich in Schritt 8. und 9. wie folgt ergänzen:

8. $Grad(x) = 0, \quad Level(x) = Level(\hat{x}) + 1 \quad \forall x \in X_{neu},$
bestimme $Rang(x) \quad \forall x \in X;$

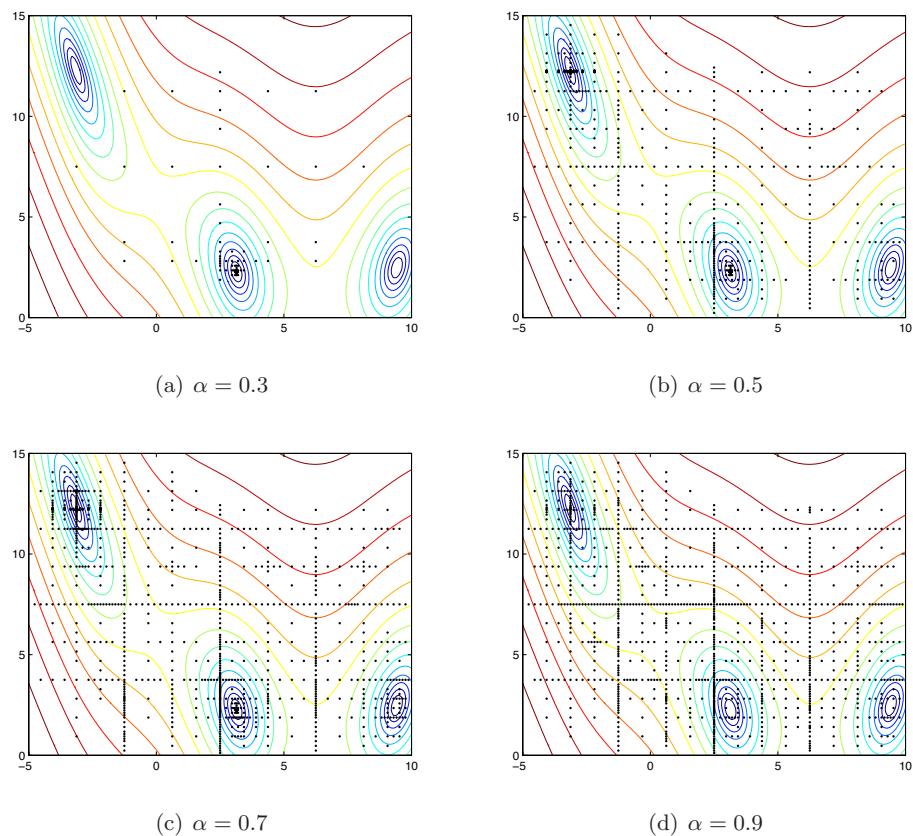
9. Bewerte alle $x \in X$ mit

$$\beta_\alpha(x) = (Level(x) + Grad(x))^\alpha Rang(x)^{1-\alpha},$$

wähle $\hat{x} = \arg \min_{x \in X} (\beta_\alpha(x)),$
gehe zu 5.;

Man muss bei der Bestimmung der Nachbarn eines Punktes, wie im HCP-Algorithmus, seinen Grad berücksichtigen und falls dieser größer als 0 ist, nähere Nachbarn, also Nachbarn der Stufe $Grad(\hat{x})$ bestimmen, siehe Bemerkung 4.19. Des Weiteren kann es vorkommen, dass ein Punkt zwar Grad 0 hat, im ersten Schritt aber trotzdem keine neuen Nachbarn berechnet werden, da diese schon als Nachbarn anderer Punkte bestimmt wurden. In diesem Fall muss der Grad des Ausgangspunktes entsprechend erhöht werden und in einem neuen Durchlauf näher liegende Nachbarn bestimmt werden. Wenn man so vorgeht, ändert man allerdings die Reihenfolge der Punkterzeugung. Es kann nämlich passieren, dass Punkte aus einem feineren Gitter hinzugefügt werden, obwohl noch nicht alle Punkte eines größeren Gitters vorhanden sind.

Der Grad der Adaptivität hängt von dem Parameter $\alpha \in [0, 1]$ ab, für $\alpha = 1$ erhält man den nichtadaptiven Algorithmus 4.13. Die folgenden Abbildungen zeigen, welche Punkte der adaptive Algorithmus 4.13 für verschiedene α für das Noboundary-Gitter ausgewählt hat, siehe Abbildung 4.14. Zugrundegelegt wurde wieder das zweidimensionale Testbeispiel BR, siehe Kapitel 8. Als Abbruchkriterium wirkte bei allen Beispielen die maximale Anzahl der Funktionsauswertungen, diese wurde hier auf 1000 gesetzt. Abbildung 4.15 zeigt die berechneten Gitterpunkte für die anderen Gittertypen, hier wurde $\alpha = 0.3$ und $\alpha = 0.7$ gesetzt. Schließlich zeigt Abbildung 4.16 die berechneten Punkte für das dreidimensionale Testproblem $Hn3$ für verschiedene α und das Noboundary-Gitter, sowie das kubische Gauß-Legendre-Gitter. Hier entspricht die Farbe der einzelnen Punkte dem Funktionswert des jeweiligen Punktes (blau steht für niedrige Werte, rot für hohe).

Abbildung 4.14: `iterative_sparsagrid` für das Noboundary-Gitter.

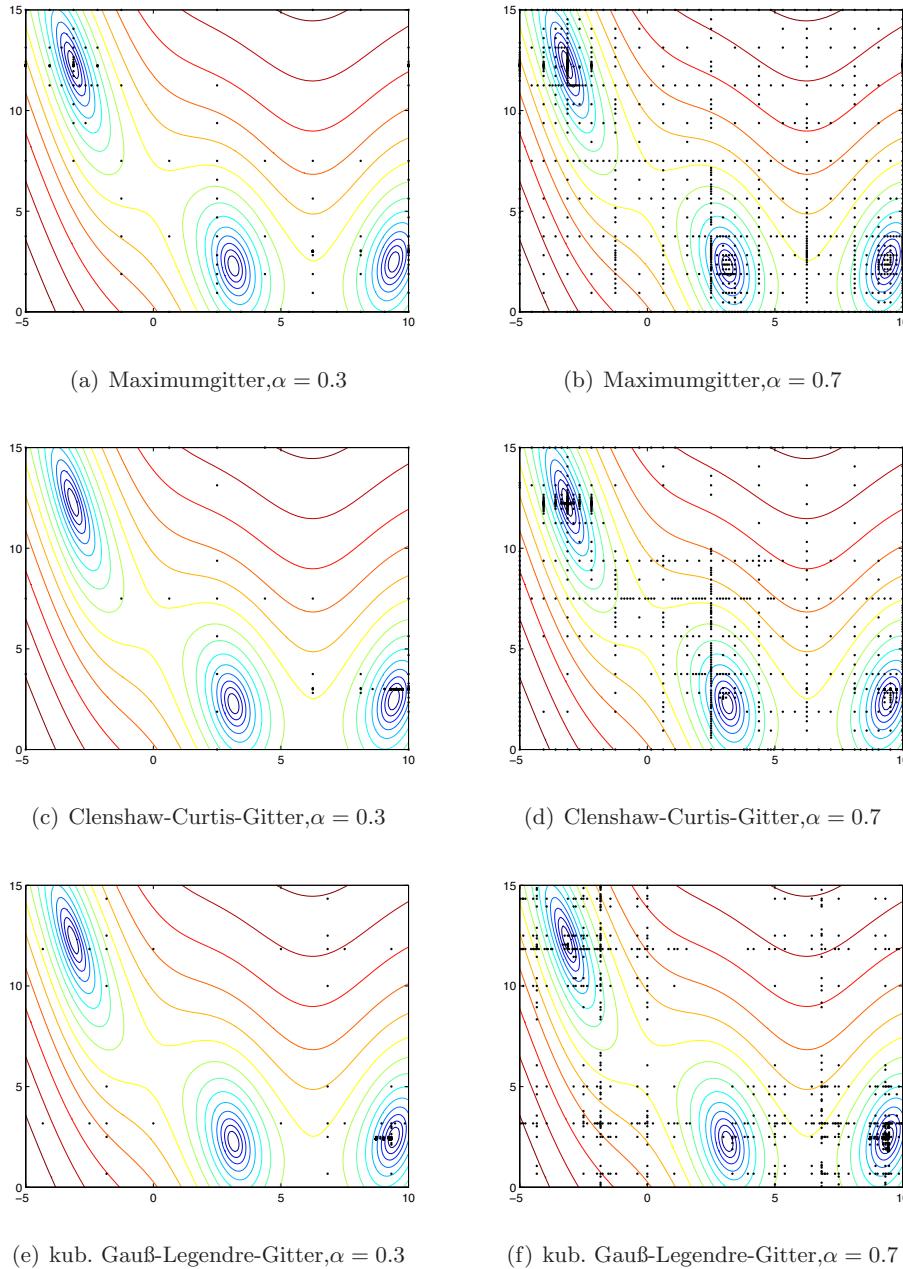
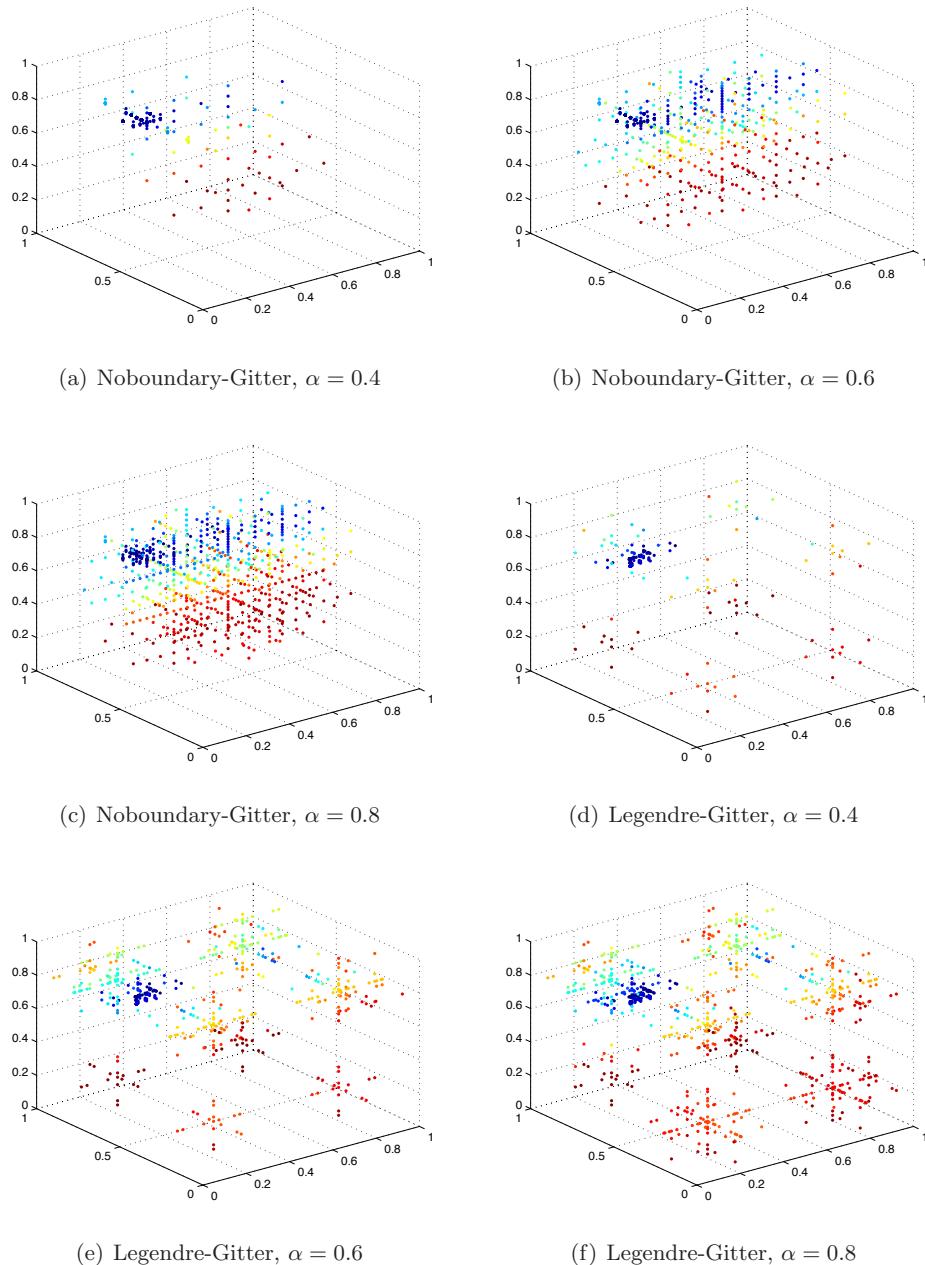


Abbildung 4.15: `iterative_sparsegrid` für die restlichen Gittertypen mit $\alpha = 0.3$ und $\alpha = 0.7$.

Abbildung 4.16: `iterative_sparsegrid` mit verschiedenen α .

4.6 Numerischer Vergleich der Gittertypen

Abschließend soll die Güte des `iterative_sparsesgrid`-Algorithmus, abhängig von den vier verschiedenen Gittertypen, bestimmt werden. Hierzu wird das Verfahren anhand einer Auswahl an Standardtestproblemen, siehe Kapitel 8, getestet. Der folgenden Tabelle kann man die Rechenzeiten und die erreichte Lösung entnehmen. In der letzten Spalte ist das globale Minimum f^* der Zielfunktion angegeben. In der letzten Zeile sieht man die Summe der Rechenzeiten und die Summe der Abweichungen $|f^* - f_{opt}|$ für alle Probleme bis auf das Testproblem von Schäffler (Abk. Sch), für welches das Verfahren nicht konvergiert. Alle Berechnungen wurden auf einem Intel Pentium M Prozessor mit 1.60GHz durchgeführt. Die Rechenzeiten sind in Sekunden angegeben und bezeichnen die tatsächlich abgelaufene Rechenzeit (realtime). Ein „-“ bedeutet, dass der Arbeitsspeicher (512MB RAM) zu klein war und große Datenmengen ausgelagert werden mussten. Der Algorithmus wurde jedesmal mit dem Adaptivitätsparameter $\alpha = 0.7$ und 1000 Funktionsauswertungen als Abbruchkriterium gestartet. Man sieht, dass der Algorithmus mit dem Noboundary-Gitter als Grundlage am besten abschneidet. Der Algorithmus ist hier nicht nur schneller, sondern erzielt auch bessere Lösungen als mit den anderen Gittern als Grundlage. Mit dem Clenshaw-Curtis-Gitter werden für die meisten Probleme gute Ergebnisse erzielt, allerdings ist es etwas langsamer und die Ergebnisse sind im Durchschnitt etwas schlechter als mit dem Noboundary-Gitter. Das gleiche gilt auch für das Maximumgitter, hier dauert die Berechnung allerdings in allen Fällen länger. Das kubische Gauß-Legendre-Gitter ist für die meisten Probleme zu speziell und konzentriert sich von vornherein zu sehr auf bestimmte Bereiche, verfeinert also auch im nichtadaptivem Fall um Gitterpunkte, an denen nicht notwendigerweise das globale Minimum vermutet wird. Daher ist es eher ungeeignet. Bei hohen Dimensionen hat man darüber hinaus das Problem, dass die Rechenzeit schon bei der Erzeugung des Startgitters explodiert. Auf Grund dieser Untersuchungen kommen wir zu dem Ergebnis, dass sich für die meisten Probleme das Noboundary-Gitter am besten eignet. Zum einen werden keine Punkte am Rand des betrachteten Gebietes bestimmt, wodurch Funktionsauswertungen eingespart werden, zum anderen findet im nichtadaptivem Fall keine starke Konzentration der Gitterpunkte auf bestimmte Gebiete statt, somit wird die globale Suche in der Vordergrund gestellt.

d		maximum		noboundary		clenshaw		legendre		f^*
		Zeit	f_{opt}	Zeit	f_{opt}	Zeit	f_{opt}	Zeit	f_{opt}	
2	B	2.250	0.300	1.703	0.011	1.829	0.000	0.984	0.014	0.000
2	BR	2.391	0.399	1.625	0.398	1.641	0.399	1.328	0.398	0.398
2	C	2.265	-1.000	1.563	-1.031	1.718	-0.996	1.047	-0.853	-1.031
2	G2	7.797	0.000	1.797	0.000	2.031	0.000	1.531	0.000	0.000
2	GP	6.203	3.000	1.625	3.000	1.718	3.000	1.063	8.482	3.000
2	H	2.390	0.000	1.781	0.000	1.578	0.000	2.985	0.104	0.000
2	M	2.453	0.009	1.485	0.009	2.000	0.009	1.063	0.009	0.000
2	R	1.969	-2.000	1.406	-2.000	1.563	-2.000	0.938	-1.926	-2.000
2	Ro	2.203	0.018	1.610	0.009	1.797	0.018	0.984	0.184	0.000
3	Hn3	2.672	-3.861	1.704	-3.861	1.984	-3.861	1.094	-3.828	-3.862
6	Hn6	2.563	-2.491	2.110	-3.394	2.156	-3.087	0.985	-2.403	-3.394
10	G10	2.672	0.852	2.265	0.004	2.250	0.024	0.437	0.005	0.000
50	Sch	7.641	6.022	5.985	6.022	5.922	6.022	-	-	0.000
		37.828	2.1130	20.674	0.0340	22.265	0.3930	14.439	7.075	

Tabelle 4.1: Ergebnisse des `iterative_grid`-Algorithmus für die verschiedenen Gitter.

Kapitel 5

Cluster Methoden und Cluster Analyse

5.1 Motivation

Nachdem der `iterative_sparsegrid` Algorithmus bei der Suche nach dem globalen Minimum adaptiv vorgeht, kann man bei geeigneter Wahl des Adaptivitätsparameters α erwarten, dass in Gebieten mit niedrigen Funktionswerten mehr Punkte erzeugt werden als in Gebieten mit hohen Funktionswerten. Gebiete, in denen Punkte dichter zusammenliegen als in der Umgebung, nennt man *Cluster*, siehe Definition 5.1. Da man vermutet, dass sich das globale Minimum in einem dieser Cluster befindet, will man in jedem Cluster ein lokales Verfahren starten, welches die Lösung mit einer höheren Genauigkeit bestimmen kann. In diesem Kapitel werden wir uns mit Methoden beschäftigen, die dazu dienen, solche Cluster, also Gruppen die in irgendeiner Weise zusammengehören, in einer Datenmenge zu erkennen und die Clustermittelpunkte, siehe Definition 5.3, zu berechnen.

5.2 Clusteranalyse

Die Clusteranalyse findet in vielen Gebieten der Wissenschaft Anwendung. Sie wird vor allem zur automatischen Klassifikation, in der Bildverarbeitung und bei der Mustererkennung eingesetzt. In [Hi05] wurden Methoden der Clusteranalyse zur Klassifizierung von Hedge Fonds verwendet. Je nach Anwendungsgebiet fasst man die zu untersuchenden Objekte nach unterschiedlichen Kriterien zusammen. Es gibt jedoch zwei allgemeine Eigenschaften, die die resultierenden Cluster erfüllen sollen:

1. Homogenität innerhalb der Cluster.
2. Heterogenität zwischen den Clustern.

Definition 5.1 *Cluster*

Sei $X = \{x_j : j = \{1, \dots, n\}\} \subset \mathbb{R}^d$ die Menge der zu untersuchenden Datenpunkte. Dann heißen die Mengen $C_i \subset X, i = \{1, \dots, K\}$ Cluster, falls gilt:

1. Die C_i sind disjunkt, d. h. $C_k \cap C_l = \emptyset$ für alle $k \neq l, k, l \in \{1, \dots, K\}$.
2. $X = \bigcup_{i=1}^K C_i$.

Bemerkung 5.2

1. Ein Homogenitätskriterien kann z. B. der Abstand zweier Punkte $x, y \in X$, $dist(x, y) : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow [0, \infty[$ sein, z. B. bzgl. der euklidschen Norm,

$$dist(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}.$$

2. In der Optimierung bietet es sich an, bei der Entscheidung, ob sich zwei Punkte ähnlich sind, auch deren Funktionswerte einzubeziehen. Man kann daher die Datenpunkte $(x, f(x)) \in X \times \mathbb{R}$ analysieren und so den Abstand der Funktionswerte berücksichtigen.

Das Ergebnis einer Clustermethode sind entweder die gefundenen Cluster $C_i, i = \{1, \dots, K\}$ oder die Clustermittelpunkte, auch *Centroide* genannt (engl. für Schwerpunkt).

Definition 5.3 Centroid

Das Centroid c^* eines Clusters C ist der Punkt, dessen quadrierter Abstand zu allen Elementen aus C in der Summe minimal ist, siehe auch [Hi05].

$$c^* = \arg \min_{c \in \mathbb{R}^d} \sum_{x_i \in C} \|x_i - c\|_2^2.$$

Bemerkung 5.4 Im Allgemeinen ist das Centroid c^* eines Clusters C nicht in C . Will man sicherstellen, dass ein Punkt ausgewählt wird der in C enthalten ist, muss man folgendes Minimierungsproblem lösen um den *Medoid* eines Clusters zu bestimmen:

$$x^* = \arg \min_{x \in C} \sum_{x_i \in C} \|x_i - x\|_2^2.$$

In der Literatur ist es üblich, die Clusteranalyse-Methoden in hierarchische und nicht-hierarchische Methoden (sog. Partitionierungsverfahren) aufzuteilen. Daneben gibt es modellbasierte Methoden und Fuzzy Clustering, vgl. [Hi05].

Wir werden zwei Clustering Methoden betrachten, um die in Algorithmus 4.13 erzeugten Punkte zu untersuchen. Wenn man weiß, in wie viele Cluster man einen Datensatz zerlegen will, bietet sich der *Fuzzy C-Means* Algorithmus (kurz FCM-Algorithmus) an. Dieser von Bezdek, siehe [Be73], entwickelte Algorithmus ist die bekannteste und einfachste Fuzzy Clustering Methode. Als Ergebnis erhält man K Cluster mit den zugehörigen Centroiden. Verfügt man dagegen nicht über die Information, wie viele Cluster in der Datenmenge enthalten sind, kann man eine subtraktive Cluster Methode anwenden, die die Clustermittelpunkte ermittelt.

Beide Algorithmen sind in der kommerziellen Fuzzy Logic Toolbox von Matlab enthalten. Daneben gibt es ein freeware Paket, welches ebenfalls den Fuzzy C-Means Algorithmus enthält, siehe [BAF05]. Die subtraktive Cluster Methode wurde implementiert und ist im zur Diplomarbeit gehörenden Programmpaket enthalten.

5.2.1 Fuzzy C-Means

Der FCM-Algorithmus, siehe Algorithmus 5.7, beruht auf der Minimierung des sog. C-Means Funktionals, siehe [KK98]. Es handelt sich um einen Fuzzy Algorithmus, weil ein Datenpunkt nicht 100%-ig zu einem Cluster gehört, sondern sein so genannter Zugehörigkeitsgrad bestimmt wird. Der FCM-Algorithmus arbeitet mit der euklidischen Norm und versucht ungefähr gleich große Cluster zu bestimmen, siehe auch [KK97].

Definition 5.5 C-Means Funktional

Das Funktional $\mathcal{J} : \mathbb{R}^{d \times n} \times \mathbb{R}^{K \times n} \times \mathbb{R}^{d \times K} \rightarrow \mathbb{R}$

$$\mathcal{J}(X, U, C) = \sum_{i=1}^K \sum_{j=1}^n u_{ij}^m \|c_i - x_j\|_2^2$$

heißt C-Means Funktional. Zu den Bezeichnungen:

- $X = \{x_1, \dots, x_n\}$ bezeichnet die Menge der Datenpunkte.
- $K \geq 2$ ist die Anzahl der zu bestimmenden Cluster.
- $m \in (1, \infty)$ ist der Fuzzy-Index, gewöhnlich wählt man $m = 2$.
- $C = (c_1, \dots, c_K)$ ist die Menge der Clustercentroide.
- $U = (u_{ij})_{\substack{1 \leq i \leq K \\ 1 \leq j \leq n}}$ ist die Zugehörigkeitsmatrix. Die Zahl $u_{ij} \in [0, 1]$ steht für den Grad der Zugehörigkeit des Punktes x_j zum Cluster c_i .

Man kann die Bestimmung der Menge der Clustercentroide C als nichtlineares Minimierungsproblem formulieren:

$$\min_{U, C} \mathcal{J}(X, U, C) \quad (5.1)$$

$$s.t. \quad \sum_{i=1}^c u_{ij} = 1 \text{ für alle } j \in \{1, \dots, n\} \quad (5.2)$$

$$\sum_{j=1}^n u_{ij} \geq \varepsilon \text{ für alle } i \in \{1, \dots, K\}, \varepsilon > 0. \quad (5.3)$$

Bemerkung 5.6 Interpretation von (5.2) und (5.3):

1. Die erste Nebenbedingung stellt sicher, dass sich die Gewichte u_{ij} für jeden Datenpunkt x_j zu eins aufaddieren.
2. Die zweite Nebenbedingung stellt sicher, dass kein Cluster leer bleibt.

Die zu (5.1)-(5.3) gehörende Lagrangefunktion (für eine feste Menge X) lautet:

$$\bar{\mathcal{J}}(X, U, C, \lambda_1, \dots, \lambda_{n+K}) = \mathcal{J}(X, U, C) + \sum_{j=1}^n \lambda_j \left(\sum_{i=1}^K u_{ij} - 1 \right) + \sum_{i=n+1}^{n+K} \lambda_i \left(\sum_{j=1}^n u_{ij} - \varepsilon \right). \quad (5.4)$$

Differenziert man (5.4) nach den u_{ij} erhält man für $\varepsilon \rightarrow 0$ als Optimalitätsbedingung für die u_{ij} :

$$u_{ij} = \left(\sum_{k=1}^K \left(\frac{dist(c_i, x_j)^2}{dist(c_k, x_j)^2} \right)^{\frac{1}{m-1}} \right)^{-1}. \quad (5.5)$$

Differenziert man (5.4) nach den c_i erhält man als Optimalitätsbedingung an die c_i :

$$c_i = \frac{\sum_{j=1}^n u_{ij}^m x_j}{\sum_{j=1}^n u_{ij}^m}. \quad (5.6)$$

Insgesamt kann der FCM-Algorithmus wie folgt beschrieben werden:

Algorithmus 5.7 FCM-Algorithmus

1. Wähle eine Startlösung C_0 , und bestimme die Zugehörigkeitsmatrix U_0 mit Hilfe von (5.5), setze $i = 0$;
2. Setze $i = i + 1$;
3. Bestimme die neuen Clustercenter C_i nach (5.6) mit U_{i-1} ;
4. Bestimme die neue Zugehörigkeitsmatrix U_i nach (5.5) mit C_i ;
5. Falls $\|C_i - C_{i-1}\| > \delta$: gehe zu 2., sonst: die Lösung ist C_i .

Die Anzahl K der zu bestimmenden Cluster und die Menge der Startcluster C_0 kann man z. B. mit der im nächsten Abschnitt beschriebenen subtraktiven Clustermethode bestimmen.

Bemerkung 5.8 Wie in [Be74] diskutiert, erhält man, je nachdem wie man den Parameter $m > 1$ in Definition 5.5 wählt, entweder eine klare Einteilung in K Cluster (für $m \rightarrow 1$ gilt entweder $u_{ij} \rightarrow 1$ oder $u_{ij} \rightarrow 0$) oder eine verschwommene Einteilung (für $m \rightarrow \infty$ erhält man $u_{ij} \rightarrow \frac{1}{K}$). Gewöhnlich wählt man z. B. $m = 2$.

Abbildung 5.1 zeigt die Ergebnisse des FCM-Algorithmus, implementiert in der free-ware Fuzzy Logic Toolbox.

Konvergenz des FCM-Algorithmus

Wie in [AM96] gezeigt, kann der FCM-Algorithmus als eine Art Gradientenverfahren interpretiert werden. Im FCM-Algorithmus ist Schritt 3. eine Minimierung nach U (mit C fest) und Schritt 4. eine Minimierung nach C (mit U fest). Allerdings kann man keine Aussage darüber treffen, ob diese Minimierung zu einem Sattelpunkt oder zu einem (lokalen) Extremum konvergiert, siehe [Be80]. Höppner und Klawonn haben in [HK03] eine hinreichende Bedingung für die Konvergenz des FCM-Algorithmus zu einem (lokalen) Extremum angegeben.

5.2.2 Eine subtraktive Clustermethode: cluster_estimation

Die Güte des FCM-Algorithmus hängt stark davon ab, ob man den Parameter K und die Startwerte C_0 geeignet wählt. Meistens kann man diese Werte nicht angeben, sondern muss sie bestimmen. Hierfür gibt es andere Clustermethoden, z. B. subtraktive Methoden. Die bekannteste ist das Verfahren von S. L. Chiu, siehe [Ch94], welches ebenfalls in der kommerziellen Fuzzy Logic Toolbox (hier heißt die Methode `subclust`) von Matlab enthalten ist. Da man nicht davon ausgehen kann, dass jeder Nutzer diese Toolbox besitzt, wurde diese Methode unter dem Namen `cluster_estimation` implementiert, siehe Algorithmus 5.9. Ein Vergleich der beiden Routinen schließt sich im nächsten Kapitel an.

In jeder Iteration wird das Potential jedes Datenpunktes ein Clustermittelpunkt zu sein bestimmt. Man wählt den Punkt mit dem größten Potential aus und schließt ihn, zusammen mit den Punkten aus seiner Umgebung, von der weiteren Betrachtung aus. Dies wird so lange wiederholt, bis alle Datenpunkte einem Cluster zugeordnet sind. Wie im Zusammenhang mit dem FCM-Algorithmus schon erwähnt, werden wir die lokale Suche nicht in den Clustermittelpunkten, sondern in den Minima der entsprechenden Cluster starten.

Algorithmus 5.9 $C = \text{cluster_estimation}$

1. Input: Datenmenge $X \in \mathbb{R}^{d \times n}$;
2. Wähle die Parameter $r_a, \bar{\varepsilon}$, und $\underline{\varepsilon}$;
3. $\alpha = \frac{4}{r_a^2}, \beta = \frac{4}{r_b^2}, r_b = 1.5r_a, k = 0, C = \emptyset$;
4. Setze $k = k + 1$;
5. Bestimme das Potential P_i aller Punkte $x_i \in X$ ein Clustermittelpunkt zu sein:

$$P_i = \sum_{j=1}^n \exp(-\alpha \|x_i - x_j\|^2), \text{ für } i = \{1, \dots, n\};$$

6. Wähle $x_k^* = x_{j^*}$ mit $j^* = \arg \max_i (P_i)$,
Setze $P_k^* = P_{j^*}$;
7. Setze $P_i = P_i - P_k^* \exp(-\beta \|x_i - x_k^*\|^2)$, für $i = \{1, \dots, n\}$;
 - (a) Falls $P_k^* > \bar{\varepsilon} P_1^*$:
Setze $C = C \cup \{x_k^*\}$ und gehe zu 4.;
 - (b) Falls $P_k^* < \underline{\varepsilon} P_1^*$:
Setze $C = C \cup \{x_k^*\}$ und beende den Algorithmus;
 - (c) Sonst:
 - i. Sei δ der kürzeste Abstand zwischen x_k^* und allen Clustermittelpunkten aus C ;
 - ii. Falls $\frac{\delta}{r_a} + \frac{P_k^*}{P_1^*} \geq 1$:
Setze $C = C \cup \{x_k^*\}$ und gehe zu 4.;

iii. Sonst:

Lehne x_k^* ab, setze $P_k^* = 0$ und gehe zu 4.

Bemerkung 5.10 Zur Wahl der Parameter im Algorithmus 5.9:

1. Der Parameter $r_a \in [0, 1]$ bestimmt den Radius eines Clusters, d. h. wurde ein Clustermittelpunkt gefunden, werden alle Datenpunkte innerhalb dieses Radius von der weiteren Betrachtung ausgeschlossen. Gewöhnlich liegt r_a in $[0.2, 0.6]$, ein Standardwert ist $r_a = 0.4$.¹
2. Die Parameter $\bar{\varepsilon}$ und $\underline{\varepsilon}$ geben Schranken für die Akzeptanz, bzw. Ablehnung eines Datenpunktes als Clustermittelpunkt an. Standardwerte sind: $\bar{\varepsilon} = 0.5$ und $\underline{\varepsilon} = 0.15$.

¹Man betrachtet im Allgemeinen die auf den Bereich $[0, 1]^d$ (bzw. $[-0.5, 0.5]^d$) transformierten Datenpunkte X .

5.3 Numerische Tests und Beispiele

Bei allen Beispielen wurde zunächst der `iterative_sparsegrid` Algorithmus mit $\alpha = 0.7$ auf ein zwei- und dreidimensionales Testproblem angewendet. Zum einen soll mit den folgenden Tests untersucht werden, ob es sich positiv auswirkt, statt der Datenpunkte X die Datenmengen (X, F) zu clustern. Zum anderen soll überprüft werden, in wie weit sich die Ergebnisse der selbst implementierten Methode `cluster_estimation` von den Ergebnissen der Matlab-internen Methode `subclust` unterscheiden. Wie bereits erwähnt interessieren uns weniger die ermittelten Clustermittelpunkte, als die Minima der gefundenen Cluster.

5.3.1 FCM-Algorithmus

Die folgenden Tabellen geben die Laufzeiten der kommerziellen Implementation und der freeware Implementation des FCM-Algorithmus an. Alle Berechnungen wurden auf einem Intel Pentium M Prozessor mit 1.60GHz durchgeführt. Die Rechenzeiten sind in Sekunden angegeben und bezeichnen die tatsächlich abgelaufene Rechenzeit (realtime). In den Tabellen bezeichnet d die Dimension des Problems und n die Anzahl der untersuchten Datenpunkte.

$d \setminus n$	500	1000	2000	3000	4000
2	0.172	0.547	1.766	5.297	10.485
3	0.375	1.172	4.187	8.109	14.359
4	0.610	2.406	11.422	22.844	44.563

Tabelle 5.1: Laufzeiten des freeware FCM-Algorithmus.

$d \setminus n$	500	1000	2000	3000	4000
2	0.110	0.469	0.484	0.344	0.516
3	0.203	0.625	0.500	0.500	0.891
4	0.390	0.422	0.343	0.531	0.515

Tabelle 5.2: Laufzeiten des kommerziellen FCM-Algorithmus.

In Abbildung 5.1 zeigen die grünen Kreise die ermittelten Clustercenter und die roten Kreise die Minima der Cluster.

Überraschenderweise zeigt Abbildung 5.1, dass mit der Menge X ein besseres Ergebnis erzielt wird als mit (X, F) . Dies ist überraschend, da man annehmen würde, dass in den Gebieten in denen der `iterative_sparsegrid` Algorithmus mehr Punkte bestimmt hat, auch die Funktionswerte einheitlicher sind, und es daher einfacher wäre diese Cluster zu identifizieren. Man sieht des Weiteren, dass es für die Güte des FCM-Algorithmus wichtig ist, die richtige Anzahl der Clustercenter anzugeben. Die Laufzeitentabellen zeigen, dass die kommerzielle Implementierung erheblich schneller ist als die freeware-Routine. Nun wollen wir überprüfen ob die Ergebnisse der subtraktiven Clustermethode ähnlich gut sind wie die des FCM-Algorithmus und in wie weit sich die selbstimplementierte Methode von der kommerziellen unterscheidet.

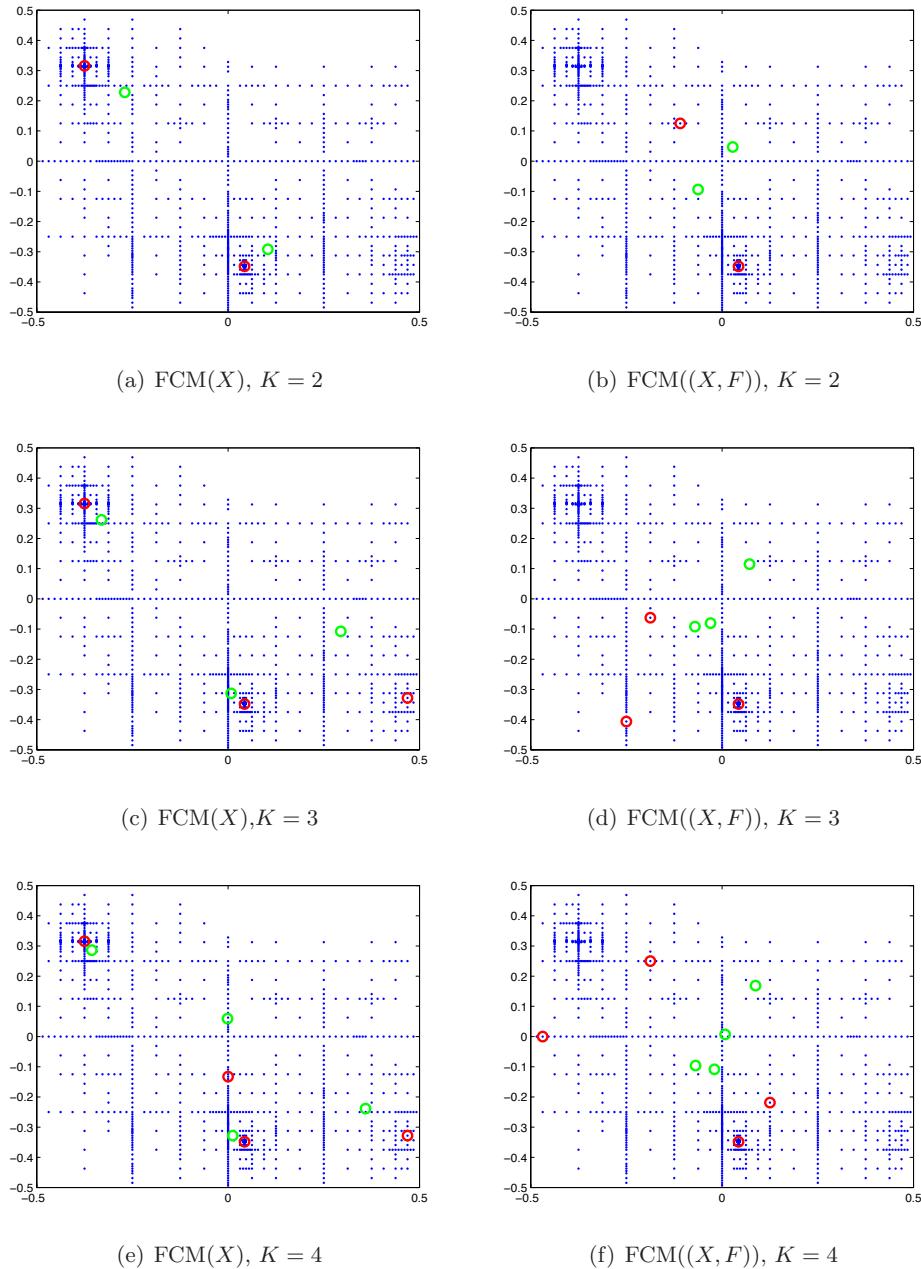


Abbildung 5.1: FCM-Algorithmus angewendet auf 1000 Datenpunkte.

5.3.2 Vergleich von `subclust` und `cluster_estimation`

Die folgenden Abbildungen zeigen die Ergebnisse der selbstimplementierten Methode `cluster_estimation` (rot) und der Matlab Methode `subclust` (grün). Diesmal wurden die Ergebnisse von Algorithmus 4.13 für die verschiedenen Gittertypen untersucht. Es findet wieder ein Vergleich der Menge X mit der Menge (X, F) (mit auf $[-0.5, 0.5]$ transformiertem F) statt, um zu entscheiden, welche Datenmenge besser geeignet ist. Hier sehen wir nicht die Clustermittelpunkte, sondern die Minima der Cluster.

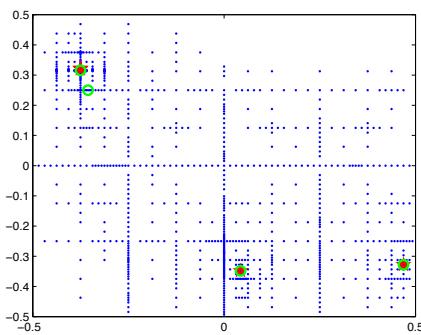
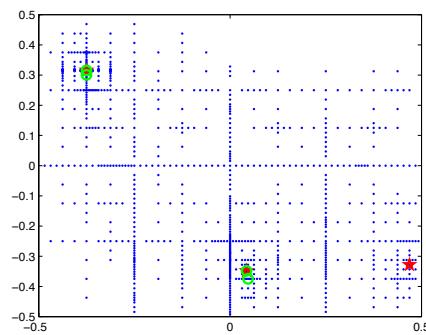
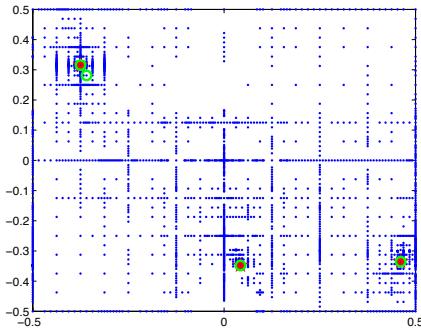
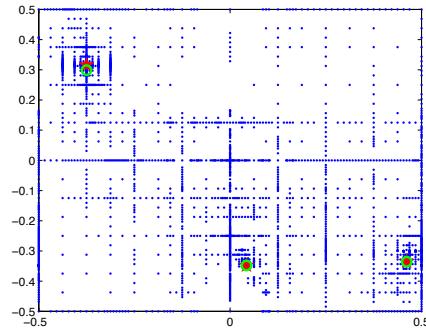
(a) Noboundary-Gitter, (X) , 1000 Pkt.(b) Noboundary-Gitter, (X, F) , 1000 Pkt.(c) Clenshaw-Curtis-Gitter, (X) , 3000 Pkt.(d) Clenshaw-Curtis-Gitter, (X, F) , 3000 Pkt.

Abbildung 5.2: Vergleich von `cluster_estimation` (rot) und `subclust` (grün) im 2-dimensionalen.

Mit der subtraktiven Clustermethode erzielt man für beide Datenmengen vergleichbare Ergebnisse. Es ist also für das Ergebnis fast unerheblich, ob man die Menge X oder (X, F) clustert. Hier kann man die Güte über den Parameter r_a , siehe Bemerkung 5.10, steuern, d.h für kleinere r_a erkennt das Verfahren mehr Cluster als für größere. Die vorliegenden Abbildungen wurden mit $r_a \in [0.3, 0.5]$ erzeugt. Daneben sieht man, dass die Ergebnisse von `cluster_estimation` in den meisten Fällen mit den Ergebnissen von `subclust` übereinstimmen. Wie Abbildung 5.2 (a) zeigt, sind die Ergebnisse bei geeigneter Parameterwahl fast identisch zu den Ergebnissen des FCM-Algorithmus. Die Laufzeitentabelle zeigt, dass die Laufzeiten des selbstimplementierten Methode ähnlich zu den Laufzeiten der kommerziellen Methode sind.

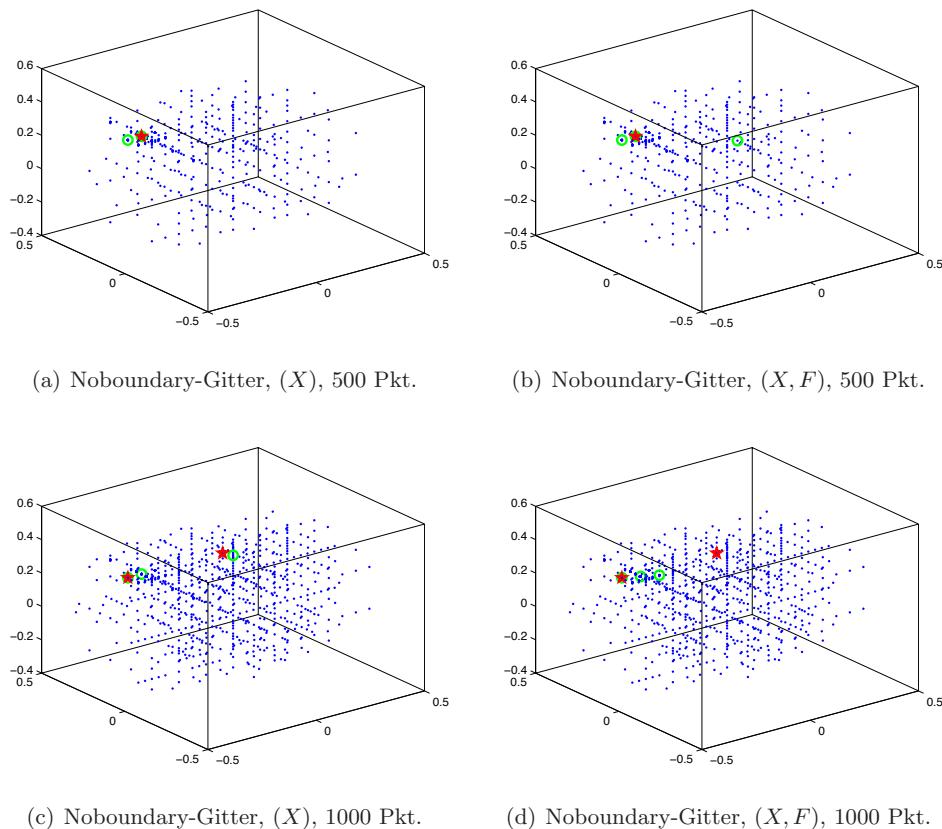


Abbildung 5.3: Vergleich von `cluster_estimation` (rot) und `subclust` (grün) im 3-dimensionalen.

Für höhere Dimensionen und mehr Punkte arbeitet `subclust` etwas schneller.

$d \setminus n$	500		1000		2000		3000		4000	
	ce	sc								
2	0.172	0.688	0.547	0.844	1.515	1.734	3.719	3.485	6.562	5.859
3	0.172	0.625	0.437	0.938	1.593	1.890	3.406	3.360	5.891	5.922
4	0.218	0.641	0.516	0.969	2.000	1.984	4.422	3.656	7.843	7.258

Tabelle 5.3: Vergleich der Laufzeiten von `cluster_estimation` und `subclust`.

5.3.3 Fazit

Wie man den Abbildungen entnehmen kann, sind die Ergebnisse von `cluster_estimation` durchaus mit den Ergebnissen des FCM-Algorithmus zu vergleichen. Daneben ist `cluster_estimation` schneller als die freeware Version des FCM-Algorithmus. Daher werden wir zur Analyse der bestimmten Gitterpunkte die selbstimplementierte subtraktive Clustermethode verwenden. Nachdem die Analyse der Datenmenge (X, F) gegenüber X nicht die erhoffte Verbesserung bringt, werden wir, aus Laufzeitgründen, die Clusteranalyse auf X anwenden.

Kapitel 6

Lokale Optimierung

6.1 Einleitung

Bislang haben wir eine Methode kennengelernt, die einen gegebenen Quader adaptiv absucht und so, zusammen mit der Clusteranalyse, Gebiete bestimmt in denen das globale Minimum vermutet wird. Die Minima der Cluster eignen sich somit als Startpunkte für ein lokales Optimierungsverfahren. Zur lokalen Suche wollen wir eine Methode anwenden, die ebenfalls ableitungsfrei arbeitet und relativ wenig Punkte benötigt, um die Lösung zu bestimmen. Hier bietet sich eines der beliebtesten Verfahren, der Algorithmus von Nelder und Mead, siehe [NM65], an. Der Grund für die Beliebtheit dieses Verfahrens ist, dass es für viele Probleme aus der Praxis sehr gut funktioniert und gegenüber Störungen in den Funktionswerten sehr robust ist. Diese Methode ist sehr weit verbreitet und unter anderem auch in Matlab implementiert (`fminsearch`). Da sie weder eine Einschränkung auf einen Quader, noch nichtlineare Ungleichungsnebenbedingungen berücksichtigen kann, können wir die Methode allerdings nicht direkt anwenden. Wir werden mit ihr, statt der Zielfunktion f , die erweiterte Lagrangefunktion minimieren und somit Nebenbedingungen miteinbeziehen. Dies wird im nächsten Kapitel genau erläutert. Im Folgenden werden wir die Funktionsweise des Nelder-Mead Algorithmus genau erklären und Vorteile sowie Nachteile des Verfahrens aufzeigen.

6.2 Der Algorithmus von Nelder und Mead

Das Verfahren lässt sich geometrisch wie folgt beschreiben: Ein Simplex, siehe auch Definition 6.1, sucht sich seinen Weg durch einen Teil des \mathbb{R}^d in Richtung eines lokalen Minimums. Dieser Vorgang wird auch durch den alternativen Namen „Downhill Simplex“ verdeutlicht. Bei seinem Weg zum gesuchten Minimum durchläuft das Startsimplex verschiedene Deformationen, um eine Folge von Punkten zu finden, an denen der Zielfunktionswert immer kleiner wird. McKinnon hat Beispiele gefunden, bei denen das Verfahren zu einem nicht stationären Punkt konvergiert, siehe [McK98]. Diese werden wir später kurz ansprechen.

Definition 6.1 Simplex

Im \mathbb{R}^d ist ein Simplex Δ die konvexe Hülle von $d + 1$ Punkten, die nicht in einer Hyperebene liegen.

Bemerkung 6.2 Die Bedingung, dass die $d+1$ Punkte, die ein Simplex aufspannen, nicht in einer Hyperebene liegen, stellt sicher, dass das Simplex nicht degeneriert ist, also ein endliches, d -dimensionales Volumen besitzt.

Algorithmus 6.3 Nelder-Mead

1. Input:

Reflektionsparameter $\rho > 0$,
 Expansionsparameter $\chi > \max\{\rho, 1\}$,
 Kontraktionsparameter $0 < \gamma < 1$,
 Schrumpfparameter $0 < \sigma < 1$,
 Gegeben: ein nichtdegeneriertes Simplex Δ_1 , setze $k = 0$;

2. Setze $k = k + 1$;

3. Ordne die $x \in \Delta_k$ so, dass gilt:

$$f(x_1) \leq f(x_2) \leq \dots \leq f(x_{d+1}),$$

x_1 ist der beste Punkt; x_{d+1} der schlechteste Punkt;

4. Reflektion:

$$x_r = x + \rho(x - x_{d+1}) = (1 + \rho)x + \rho x_{d+1}$$

mit:

$$x = \frac{1}{d} \sum_{i=1}^d x_i,$$

bestimme $f_r = f(x_r)$,

falls $f_1 \leq f_r < f_d$:

$\Delta_{k+1} = \{x_r\} \cup \{\Delta_k \setminus \{x_{d+1}\}\}$, gehe zu 2.;

5. Falls $f_r < f_1$: Expansion:

$$x_e = x + \chi(x_r) - x = x + \rho\chi(x - x_{d+1}) = (1 + \rho\chi)x - \rho\chi x_{d+1}$$

und $f_e = f(x_e)$;

6. Falls $f_e < f_r$:

$\Delta_{k+1} = \{x_e\} \cup \{\Delta_k \setminus \{x_{d+1}\}\}$, gehe zu 2.;

7. Falls $f_e \geq f_r$:

$\Delta_{k+1} = \{x_r\} \cup \{\Delta_k \setminus \{x_{d+1}\}\}$, gehe zu 2.;

8. Falls $f_r \geq f_d$: Kontraktion:

(a) Falls $f_d \leq f_r < f_{d+1}$: Äußere Kontraktion:

$$x_c = x + \gamma(x_r - x) = x + \gamma\rho(x - x_{d+1}) = (1 + \rho\gamma)x - \rho\gamma x_{d+1},$$

berechne $f_c = f(x_c)$,

falls $f_c \leq f_r$:

$\Delta_{k+1} = \{x_c\} \cup \{\Delta_k \setminus \{x_{d+1}\}\}$, gehe zu 2.;

(b) Falls $f_r \geq f_{d+1}$: Innere Kontraktion:

$$x_{cc} = x - \gamma(x - x_{d+1}) = (1 - \gamma)x + \gamma x_{d+1},$$

berechne $f_{cc} = f(x_{cc})$,

falls $f_{cc} < f_{d+1}$:

$\Delta_{k+1} = \{x_{cc}\} \cup \{\Delta_k \setminus \{x_{d+1}\}\}$, gehe zu 2.;

9. Schrumpfung:

Berechne

$$v_i = x_1 + \sigma(x_i - x_1); \text{ für } i = 2, \dots, d+1,$$

bestimme $f(v_i)$ für $i = 2, \dots, d+1$,

$\Delta_{k+1} = \{x_1, v_2, \dots, v_{d+1}\}$, gehe zu 2.

Standardwerte für die Parameter sind:

$$\rho = 1, \chi = 2, \gamma = \frac{1}{2}, \sigma = \frac{1}{2}.$$

Der Einfluss der einzelnen Schritte auf das Startsimplex werden durch Abbildung 6.1 bis 6.3 veranschaulicht.

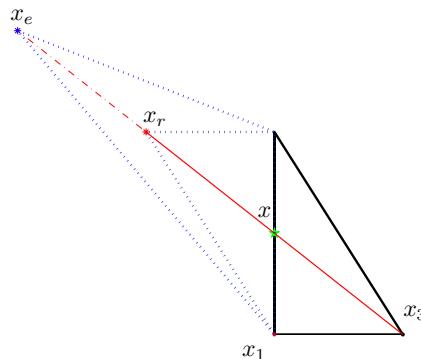


Abbildung 6.1: Reflektion und Expansion.

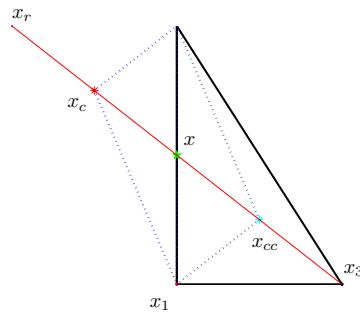


Abbildung 6.2: Äußere und innere Kontraktion.

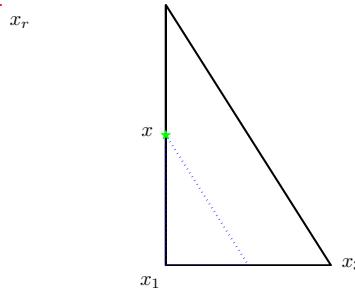


Abbildung 6.3: Schrumpfung.

Nach einigen Kontraktionsschritten muss man immer wieder überprüfen, ob das Simplex nicht degeneriert ist. Dies kann z. B. anhand der Konditionszahl κ des Simplex geschehen:

Definition 6.4 Konditionszahl κ eines Simplex

Sei $K := [x_1 - x_{d+1}, \dots, x_d - x_{d+1}] \in \mathbb{R}^{d \times d}$, dann kann das Volumen des Simplex $\Delta = \{x_1, \dots, x_{d+1}\}$ berechnet werden als

$$\text{vol}(\Delta) = |\det(K)|.$$

Die Konditionszahl $\kappa(\Delta)$ ist dann definiert als die Konditionszahl der Matrix K :

$$\kappa(\Delta) = \kappa(K) = \|K\| \|K^{-1}\|.$$

In dem Fall einer sehr großen Konditionszahl erfolgt ein Neustart im besten Punkt x_1 .

6.2.1 Konvergenz des Nelder-Mead Algorithmus

Die folgenden allgemeinen Aussagen über das Verhalten des Nelder-Mead Algorithmus sind dem Artikel „Convergence Properties of the Nelder-Mead Simplex Algorithm in Low Dimensions“ von Lagarias et al., siehe [LRWW98], entnommen.

Lemma 6.5 Falls das Startsimplex Δ_1 nicht degeneriert ist, sind auch alle im Laufe des Nelder-Mead Algorithmus entstehenden Simplizes nicht degeneriert. Ist dagegen Δ_1 degeneriert, so sind es auch alle weiteren.

Der Beweis dieses Lemmas erfolgt mit Hilfe der Multiplikationsregel für Determinanten. Leider kann keine obere Schranke für die Konditionszahl $\kappa(\Delta_k)$ angegeben werden.

Lemma 6.6 Affine Invarianz

Der Nelder-Mead Algorithmus verhält sich invariant unter affiner Transformation. Das bedeutet, dass der Algorithmus für eine Zielfunktion f und ein Startsimplex Δ_1 die gleiche Schrittfolge mit den gleichen Funktionswerten an den Ecken erzeugt, wie für die transformierte Funktion g , mit $g(x) := f(h(x)) = f(Ax + b)$, und dem Startsimplex $\hat{\Delta}_1 = h^{-1}(\Delta_1)$. Die Funktion $h(x) = Ax + b$ bezeichnet eine lineare Abbildung mit nicht singulärer Matrix A und der Umkehrfunktion $h^{-1}(x) = A^{-1}x - A^{-1}b$.

Der Beweis von Lemma 6.6 erfolgt durch Nachrechnen. Mit Hilfe dieses Lemmas kann die Betrachtung allgemeiner strikt konvexer quadratischer Funktionen auf die einfache Funktion $f(x) = \|x\|_2^2$ reduziert werden.

Das folgende Lemma ist eine Zusammenfassung mehrerer allgemeiner Aussagen über den Nelder-Mead Algorithmus.

Lemma 6.7 *Sei $f : \mathbb{R}^d \rightarrow \mathbb{R}$ eine nach unten beschränkte Funktion und starte der Nelder-Mead Algorithmus mit einem nicht degenerierten Startsimplex. Dann gilt:*

1. *Die Folge der besten Funktionswerte $(f_1^{(k)})_k$ ist konvergent.*
2. *Nach jedem Reflektions-, Expansions- oder Kontraktionsschritt gilt: $f_i^{(k)} \leq f_i^{(k-1)}$ für $1 \leq i \leq d+1$ mit $<$ für mindestens ein i .*
3. *Wird der Schrumpfungsschritt nur endlich oft durchgeführt, dann gilt:*
 - (a) *Jede Folge $(f_i^{(k)})_k$ konvergiert für $1 \leq i \leq d+1$ gegen den Wert \bar{f}_i .*
 - (b) *$\bar{f}_i \leq f_i^{(k)}$ für alle $1 \leq i \leq d+1$ und für alle k .*
 - (c) *$\bar{f}_1 \leq \dots \leq \bar{f}_{d+1}$.*
4. *Werden ab einem Schritt nur noch Schrumpfungsschritte durchgeführt, zieht sich das Simplex auf einen Punkt zusammen.*

In der Praxis treten Schrumpfungsschritte extrem selten auf. So kann man bei einer Auswahl gewöhnlicher Testfunktionen nur 33mal, bei insgesamt 2.9 Millionen Schritten, Schrumpfung beobachten, siehe [LRWW98]. Für strikt konvexe Funktionen kann man sogar zeigen, dass nie Schrumpfungen auftreten. Das liegt daran, dass bei strikter Konvexität entweder die äußere oder die innere Kontraktion erfolgreich ist. Deshalb gehen die folgenden Aussagen davon aus, dass keine Schrumpfung auftritt.

Lemma 6.8 Unterbrochene Konvergenz

Falls ein Index j existiert mit $\bar{f}_j < \bar{f}_{j+1}$, so bleiben die ersten j Ecken des Simplex ab einer Iteration K konstant.

Definition 6.9 Änderungsindex k^*

Für die k -te Iteration entspricht der Änderungsindex k^ dem kleinsten Index der Punkte die im k -ten Schritt neu in das Simplex Δ_k aufgenommen wurden, also*

$$k^* = \min_i (x_i^{(k)} \neq x_i^{(k+1)}).$$

Mit Hilfe des Änderungsindex können wir folgendes Lemma formulieren:

Lemma 6.10 *Unter der Voraussetzung von Lemma 6.8 gilt $\bar{f}_1 = \dots = \bar{f}_d$, falls der Änderungsindex unendlich oft 1 ist.*

Für strikt konvexe Funktionen kann man, wenn man die Wahl der Parameter etwas einschränkt, zeigen, dass der schlechteste Punkt und der zweit schlechteste Punkt übereinstimmen.

Lemma 6.11 Gelte zusätzlich, zu den anderen Bedingungen an die Parameter, $\rho\gamma < 1$. Dann gilt für strikt konvexe Funktionen f :

1. $\bar{f}_d = \bar{f}_{d+1}$.
2. Es gibt unendlich viele Iterationen mit $x_d^{(k+1)} \neq x_d^{(k)}$.

Für den Beweis siehe [LRWW98]. Im zweidimensionalen Fall $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ kann man folgendes Lemma zeigen, welches schließlich die Überleitung zu dem eingangs erwähnten Beispiel von McKinnon bildet.

Lemma 6.12 Sei $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ strikt konvex und seien $\rho = 1$ und $\gamma = 0.5$. Dann gilt:

$$\bar{f}_1 = \bar{f}_2 = \bar{f}_3.$$

Zum Beweis unterscheidet man die beiden Fälle, dass entweder x_1 ab einer gewissen Iteration konstant bleibt und dann alle weiteren Ecken gegen x_1 konvergieren oder dass x_1 unendlich oft wechselt, hier greift Lemma 6.10. Das Beispiel von McKinnon greift genau den ersten Fall auf, nämlich dass x_1 konstant bleibt und sich das Simplex durch eine Folge von inneren Kontraktionsschritten immer weiter zu x_1 zusammenzieht. In den konkreten Beispielen handelt es sich bei dem Punkt x_1 jedoch nicht um einen stationären Punkt.

6.2.2 Das Gegenbeispiel von McKinnon

Für das Gegenbeispiel von McKinnon führt der Nelder-Mead Algorithmus eine Folge von inneren Kontraktionsschritten durch, bei denen der zweit-schlechteste Punkt zum schlechtesten wird, siehe [McK98]. Dies passiert für eine bestimmte Klasse von zweidimensionalen, konvexen Funktionen wenn sich eine Ecke des Startsimplex im Ursprung befindet, siehe Abbildung 6.4.

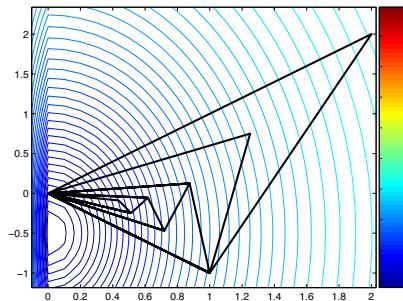


Abbildung 6.4: Folge von inneren Kontraktionsschritten.

In Abbildung 6.4 sieht man auch die Höhenlinien der zugrunde liegenden Funktion, die folgende Form hat:

$$f(x, y) = \begin{cases} \theta\phi |x|^\tau + y + y^2, & x \leq 0 \\ \theta x^\tau + y + y^2, & x > 0. \end{cases} \quad (6.1)$$

Es gibt mehrere geeignete Parameterfamilien, z. B. $\tau = 1, \theta = 15, \phi = 10$ oder $\tau = 2, \theta = 6, \phi = 60$ (siehe Abb. 6.4).

Bemerkung 6.13 In der Praxis kann man dieses Verhalten nur bis zu einer gewissen Anzahl von Iterationen beobachten. Durch Rundungsfehler werden die Daten so stark gestört, dass (z. B. in der selbstimplementierten Routine schon nach 43 Iterationen) ein Reflektionsschritt erfolgreich ist und somit das Simplex doch zum Minimum der Zielfunktion konvergiert.

6.2.3 Numerische Tests

Beispiel 6.14 Betrachten wir die sehr einfache Testfunktion

$$f(x) = \|x\|_2^2, \quad x \in \mathbb{R}^d. \quad (6.2)$$

Für d von 1 bis 100 zeigt Abbildung 6.5 welche Lösung der Algorithmus von Nelder und Mead erzielt. Als Abbruchkriterium wirkt die maximale Anzahl von Funktionsauswertungen. Als Startwert wurde jeweils der d -dimensionale Vektor $\frac{1}{\sqrt{d}}(1, \dots, 1)$ gewählt. Auf der y-Achse sieht man die Norm der bestimmten Minimallösung x_{opt} . Zur besseren Darstellung wurde die y-Achse logarithmiert.

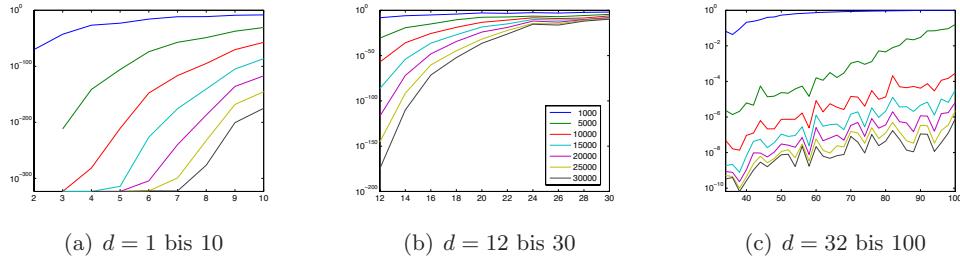


Abbildung 6.5: $\|x_{opt}\|_2$ in Abhängigkeit von d .

Abschließend folgt noch ein Vergleich des selbstimplementierten Nelder-Mead Algorithmus, `nelder_mead`, mit der Matlab-internen Routine `fminsearch`. Wieder wurde die Testfunktion (6.2), diesmal mit dem Einsvektor als Startpunkt, minimiert. Für beide Routinen wurde als Abbruchkriterium zum einen die maximale Zahl der Funktionsauswertungen ($= 1000d$) und die Toleranzgrenze $\varepsilon = 10^{-4}$ gewählt. Verglichen werden die Rechenzeit (realtime in Sekunden), die erreichte Lösung und die Anzahl der Funktionsauswertungen.

<i>d</i>	nelder_mead			fminsearch		
	# Pkte.	$f(x^*)$	Zeit	# Pkte.	$f(x^*)$	Zeit
5	191	5.744e-005	0.047	298	5.012e-005	0.047
10	484	7.614e-005	0.063	1228	1.223e-004	0.157
15	1147	3.107e-004	0.156	3551	6.509e-004	0.407
20	1955	4.360e-004	0.172	12614	3.220e-004	1.016
25	2357	3.421e-004	0.250	21612	7.523e-004	1.906
30	3457	1.978e-004	0.422	30000	2.417e-002	2.859
35	4255	3.181e-004	0.438	35000	7.436e-002	3.359
40	4171	1.580e-004	0.719	40000	3.594e-002	4.328
45	6744	3.425e-004	1.204	45000	3.500e-002	5.969
50	8263	4.032e-004	1.906	50000	7.201e-002	8.171
55	8843	3.521e-004	2.047	55000	1.355e-002	11.297
60	10958	2.862e-004	2.828	60000	1.329e-003	13.656

Tabelle 6.1: Vergleich von `nelder_mead` und `fminsearch`.

Auffallend ist, dass `fminsearch` viel mehr Funktionsauswertungen (ab $d = 30$ wird die Grenze der maximalen Funktionsauswertungen erreicht) benötigt als `nelder_mead` und dabei keine signifikant bessere Lösung findet. Dieser Unterschied ist für schwierig auszuwertende Funktionen bedeutend, da hier der eigentliche Vorteil des Nelder-Mead Algorithmus, nämlich mit relativ wenigen Funktionsauswertungen eine Lösung zu finden, wieder zunichte gemacht wird.

Kapitel 7

Nichtlineare Optimierung unter Nebenbedingungen

7.1 Einleitung

In der Einleitung dieser Diplomarbeit wurde das allgemeine Optimierungsproblem (GP) mit Ungleichungsnebenbedingungen formuliert. Das bislang entwickelte Verfahren zur globalen Optimierung auf dünnen Gittern konnte lediglich so genannte Box-Constraints ($x \in S = [a, b]$) berücksichtigen. Das lokale Verfahren, der Nelder-Mead Algorithmus, berücksichtigt keinerlei Nebenbedingungen. Daher werden wir im vorliegenden Kapitel Methoden vorstellen mit deren Hilfe man Nebenbedingungen in die bislang betrachteten Verfahren aufnehmen kann. Wir haben uns hier für die Klasse der *Penalty- und Barrieremethoden* entschieden und werden den global agierenden Teil des Verfahrens anders behandeln als den lokalen Teil. Der *iterative_sparsegrid* Algorithmus kann durch die Einführung einer speziellen *exakten Penaltyfunktion* leicht modifiziert werden und so problemlos mit Nebenbedingungen umgehen. Der Nelder-Mead Algorithmus wird innerhalb einer *PBM-Methode* auf die *erweiterte Lagrangepunkt* angewendet. Zunächst werden einige theoretische Grundlagen aufgeführt und dann die verschiedenen Verfahren vorgestellt.

7.2 Optimalitätsbedingungen

Betrachten wir das Problem (GP)

$$\begin{aligned} & \min_{x \in S} f(x) \\ & \text{s.t. } g(x) \leq 0. \end{aligned}$$

Der zulässige Bereich besteht aus der Menge Z mit

$$Z = \{x \in S \subset \mathbb{R}^d : g_i(x) \leq 0 \ \forall i \in \{1, \dots, m\}\}.$$

Ein Punkt x heißt zulässig, wenn $x \in Z$ ist. Die Menge S beschreibt hier wieder einen vorgegebenen Quader im \mathbb{R}^d , auf dem optimiert werden soll. Im Folgenden wird diese Bedingung in weitere Nebenbedingungen der Form $g(x) \leq 0$ transformiert.

Definition 7.1 Lagrangefunktion

Die Funktion $L : \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}$ mit

$$L(x, \lambda) = f(x) + \lambda^T g(x)$$

heißt *Lagrangefunktion für (GP)*. Die $\lambda_i \in \mathbb{R}_+, i \in \{1, \dots, m\}$ heißen *Lagrange-Multiplikatoren*.

Ein Punkt (x^*, λ^*) heißt *Sattelpunkt von L*, wenn gilt:

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*) \quad \forall x \in \mathbb{R}^d, \lambda \in \mathbb{R}_+^m.$$

Mit Hilfe der Lagrangefunktion kann man die globale Kuhn-Tucker-Bedingung formulieren:

Definition 7.2 Globale Kuhn-Tucker-Bedingung

Ein Punkt $x^* \in \mathbb{R}^d$ genügt der globalen Kuhn-Tucker-Bedingung, wenn ein $\lambda^* \in \mathbb{R}_+^m$ existiert, so dass (x^*, λ^*) ein Sattelpunkt von L ist.

Bemerkung 7.3 Hinreichende Optimalitätsbedingung:

Gentigt ein Punkt $x^* \in \mathbb{R}^d$ der globalen Kuhn-Tucker-Bedingung, so ist x^* ein Minimalpunkt von (GP).

Sind die Funktionen f und g differenzierbar, so lassen sich lokale Optimalitätsbedingungen ableiten:

Definition 7.4 Karush-Kuhn-Tucker-Bedingungen:

Ein Punkt $x^* \in \mathbb{R}^d$ genügt den lokalen Karush-Kuhn-Tucker Bedingungen (ist ein KKT-Punkt), falls Lagrange-Multiplikatoren $\lambda \in \mathbb{R}_+^m$ existieren mit

1. $g(x^*) \leq 0$ *(Zulässigkeit)*
2. $\lambda^T g(x^*) = 0$ *(Komplementaritätsbedingung)*
3. $\nabla f(x^*) + \lambda^T \nabla g(x^*) = 0$ *(Stationarität)*

Ein KKT-Punkt ist ein stationärer Punkt von (GP).

7.3 Das klassische Strafverfahren

Mit Methoden, die auf dem Penalty-Ansatz aufbauen, hat man Verfahren zur Hand, die Minimierungsprobleme mit Nebenbedingungen in eine Folge von Minimierungsproblemen ohne Nebenbedingungen transformieren. Eingehend wurde dieses Vorgehen z. B. von Fiacco und McCormick studiert, siehe [FM68]. Die zentrale Idee ist, Punkte, die außerhalb des zulässigen Gebietes liegen, durch Addition eines Straftersms auf den Zielfunktionswert zu bestrafen, siehe auch [Ti03], [Ul04] und [Sp93]. Statt des ursprünglichen Problems (GP) wird eine Folge von unbeschränkten Optimierungsproblemen der Art

$$\min_{x \in \mathbb{R}^d} P_\rho(x) \tag{7.1}$$

gelöst.

Hierbei ist die **Penaltyfunktion** P_ρ definiert als:

$$P_\rho(x) := f(x) + \rho \text{Pen}(x), \text{ mit } \rho > 0. \quad (7.2)$$

Der Wert $\text{Pen}(x)$ ist der **Penaltyterm**, der den Wert der Zielfunktion für zulässige Punkte nicht verändert und für Punkte außerhalb von Z mit einem positiven Wert bestraft. Das Gewicht der Strafterme wird durch den **Penaltyparameter** ρ sukzessive erhöht.

Definition 7.5 Eine Abbildung $P_\rho : \mathbb{R}^d \rightarrow \mathbb{R}$ ist eine Penaltyfunktion für (GP), wenn gilt:

$$P_\rho(x) = f(x) + \rho \text{Pen}(x) = f(x) + \rho \sum_{i=1}^m \vartheta(g_i(x))$$

mit

1. $\vartheta(y) = 0$ für $y \leq 0$.
2. $\vartheta(y) > 0$ für $y > 0$.

Beispiel 7.6 Häufig werden quadratische Penaltyfunktionen verwendet:

$$P_\rho(x) = f(x) + \rho \frac{1}{2} \sum_{i=1}^m \max\{0, g_i(x)\}^2.$$

Man nennt dann das resultierende Penaltyverfahren auch quadratisches Penaltyverfahren.

Bemerkung 7.7 Der in Beispiel 7.6 vorgestellte Strafterm ist stetig differenzierbar und hat den Wert 0 für zulässige Punkte, sowie einen positiven Wert für unzulässige Punkte. Allerdings bedeutet die Glattheit von $\text{Pen}(x)$, dass der Strafterm beim Verlassen des zulässigen Bereichs nur sehr langsam wächst, daher sind in der Regel die Minima x_ρ von $P_\rho(x)$ nicht zulässig, siehe auch [Ul04].

Algorithmus 7.8 Penalty-Methode

1. Input: Toleranzgrenze ε und Vergrößerungsfaktor $\gamma > 1$, Startpunkt $x^{(0)}$, Penaltyparameter $\rho_0 > 0$, setze $k = 0$;
2. $k = k + 1$;
3. Bestimme mit $x^{(k-1)}$ als Startpunkt die Minimallösung $x^{(k)}$ von

$$\arg \min_{x \in \mathbb{R}^d} P_{\rho_k}(x) = \arg \min_{x \in \mathbb{R}^d} (f(x) + \rho_k \text{Pen}(x));$$

4. (a) Falls $g(x^{(k)}) \leq \varepsilon$:
Lösung ist $x^{(k+1)}$;
- (b) Falls $g(x^{(k)}) > \varepsilon$:
 $\rho_{k+1} = \gamma \rho_k$,
gehe zu 2.

Die Penalty-Methode bestraft Punkte, die außerhalb des Zulässigkeitsbereiches liegen, und „zwingt“ den Optimierer, (von außen) in Richtung des Zulässigkeitsbereiches zu gehen. Zu beachten ist, dass die Minimierung in Schritt 3. für große ρ_k im Allgemeinen schlecht konditioniert ist. Man erhält also insgesamt nur eine (unzulässige) Näherungslösung für (GP). Die Minimierung in Schritt 3. kann z. B. mit dem Algorithmus von Nelder und Mead geschehen.

Satz 7.9 Globale Konvergenz

Seien $f : \mathbb{R}^d \rightarrow \mathbb{R}$ und $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$ stetig und sei der zulässige Bereich Z nicht leer. Es gelte $\rho_k \rightarrow \infty$ für $k \rightarrow \infty$. Sei des Weiteren $(x^{(k)})_k$ die von Algorithmus 7.8 (ohne Abbruchkriterium) erzeugte unendliche Folge und sei P_ρ wie in Beispiel 7.6. Dann gilt:

1. Die Folge $(P_{\rho_k}(x^{(k)}))_k$ wächst monoton.
2. Die Folge $(\text{Pen}(x^k))_k$ fällt monoton.
3. Die Folge $(f(x^{(k)}))_k$ wächst monoton.
4. Es gilt $\lim_{k \rightarrow \infty} (\max\{0, g(x^{(k)})\}) = 0$.
5. Jeder Häufungspunkt von $(x^{(k)})_k$ ist eine globale Lösung von (GP).

Beweis: (aus [Ul04])

zu 1.: Mit $\rho_k < \rho_{k+1}$, $\text{Pen}(x) \geq 0$ und $x^{(k)}$ Minimalpunkt von $P_{\rho_k}(x)$ gilt:

$$\begin{aligned} P_{\rho_k}(x^{(k)}) &\leq P_{\rho_k}(x^{(k+1)}) = f(x^{(k+1)}) + \rho_k \text{Pen}(x^{(k+1)}) \\ &\leq f(x^{(k+1)}) + \rho_{k+1} \text{Pen}(x^{(k+1)}) = P_{\rho_{k+1}}(x^{(k+1)}). \end{aligned}$$

zu 2.: Addition der Ungleichungen $P_{\rho_k}(x^{(k)}) \leq P_{\rho_k}(x^{(k+1)})$ und $P_{\rho_{k+1}}(x^{(k+1)}) \leq P_{\rho_{k+1}}(x^{(k)})$ ergibt

$$\rho_k \text{Pen}(x^{(k)}) + \rho_{k+1} \text{Pen}(x^{(k+1)}) \leq \rho_k \text{Pen}(x^{(k+1)}) + \rho_{k+1} \text{Pen}(x^{(k)})$$

dies ist gleichbedeutend zu

$$(\rho_k - \rho_{k+1})(\text{Pen}(x^{(k)}) - \text{Pen}(x^{(k+1)})) \leq 0.$$

Mit $\rho_k < \rho_{k+1}$ gilt $\text{Pen}(x^{(k+1)}) \leq \text{Pen}(x^{(k)})$.

zu 3.: Mit 2.) gilt:

$$\begin{aligned} 0 &\leq P_{\rho_k}(x^{(k+1)}) - P_{\rho_k}(x^{(k)}) = \\ f(x^{(k+1)}) - f(x^{(k)}) + \rho_k &(\text{Pen}(x^{(k+1)}) - \text{Pen}(x^{(k)})) \leq \\ f(x^{(k+1)}) - f(x^{(k)}). \end{aligned}$$

zu 4.: Wegen $Z \neq \emptyset$ existiert ein $x \in Z$ und es gilt:

$$P_{\rho_k}(x^{(k)}) \leq P_{\rho_k}(x) = f(x).$$

Mit 3. gilt:

$$f(x) \geq P_{\rho_k}(x^{(k)}) = f(x^{(k)}) + \rho_k \text{Pen}(x^{(k)}) \geq f(x_0) + \rho_k \text{Pen}(x^{(k)}).$$

Wegen $\rho_k \rightarrow \infty$ gilt daher $\text{Pen}(x^{(k)}) \rightarrow 0$.

zu 5.: Sei \bar{x} ein Häufungspunkt von $(x^{(k)})$ und sein $(x^{(k)})_{k \in K}$ eine Teilfolge mit $(x^{(k)})_{k \in K} \rightarrow \bar{x}$.

1. Zulässigkeit: Nach 4.) gilt $\text{Pen}(x^{(k)}) \rightarrow \text{Pen}(\bar{x}) = 0$, also $\bar{x} \in Z$.

2. Optimalität: Wir erhalten für jedes $x \in Z$:

$$f(\bar{x}) = \lim_{k \in K \rightarrow \infty} f(x^{(k)}) \leq \lim_{k \in K \rightarrow \infty} P_{\rho_k}(x^{(k)}) \leq \lim_{k \in K \rightarrow \infty} P_{\rho_k}(x) = f(x).$$

□

Approximation der Lagrange-Multiplikatoren

Sei $x^{(k)}$ das Minimum von (7.2) mit quadratischer Penaltyfunktion, dann ist $x^{(k)}$ ein stationärer Punkt von P_{ρ_k} :

$$\nabla P_{\rho_k}(x^{(k)}) = \nabla f(x^{(k)}) + \lambda_k^T \nabla g(x^{(k)}) = 0$$

mit

$$\lambda_k = \rho_k(\max\{0, g(x^{(k)})\}).$$

Satz 7.10 Seien f, g stetig differenzierbar. Betrachten wir die von Algorithmus 7.8 erzeugte Folge $(x^{(k)})$. Dann gilt mit $\lambda_k = \rho_k(\max\{0, g(x^{(k)})\})$: Ist $(\bar{x}, \bar{\lambda})$ ein Häufungspunkt von $(x^{(k)}, \lambda_k)$, dann ist \bar{x} ein globales Minimum von (GP) und erfüllt mit $\bar{\lambda}$ die KKT-Bedingungen.

Für den Beweis siehe z. B. [Ul04].

7.4 Exakte Penalty-Verfahren

Mit geeigneten, nichtglatten Straffunctionen kann man bereits für endliche Penaltyparameter ρ_k eine Lösung von (GP) erhalten.

Definition 7.11 exakte Penaltyfunktion

Sei x^* eine lokale Lösung von (GP). Eine Penaltyfunktion $P : \mathbb{R}^d \rightarrow \mathbb{R}$ heißt exakt im Punkt x^* , wenn x^* ein lokales Minimum von P ist.

Unter geeigneten Voraussetzungen, z. B. wenn die Funktionen f und g konvex und stetig differenzierbar sind, ist folgende Penalty-Funktion für bestimmte ρ exakt:

$$P_{\ell_1, \rho}(x) = f(x) + \rho(\|\max\{0, g(x)\}\|_1).$$

Satz 7.12 Seien $f, g_i : \mathbb{R}^d \rightarrow \mathbb{R}$ konvex und stetig differenzierbar. Ist x^* ein KKT-Punkt mit Lagrange-Multiplikator λ^* , dann ist x^* ein globales Minimum von (GP) und für alle

$$\rho \geq \max\{\lambda_i^*\}$$

ist x^* ein globales Minimum von $P_{\ell_1, \rho}$.

Ein Beispiel für eine (unstetige) exakte Penaltyfunktion ist:

Beispiel 7.13 $P : \mathbb{R}^d \rightarrow \mathbb{R}$, $P(x) = f(x) + \text{Pen}(x)$ mit

$$\text{Pen}(x) = \begin{cases} 0 & \forall x \in Z \\ \infty & \forall x \notin Z. \end{cases}$$

In Matlab hat man den großen Vorteil, dass man diese Penaltyfunktion direkt implementieren kann, da Matlab mit „Inf“ wie mit einer Zahl arbeiten kann. Daher werden wir diese Penaltyfunktion verwenden um `iterative_sparsesgrid` Algorithmus aus dem 4. Kapitel auch für Optimierungsprobleme der Form (GP) anwenden zu können.

7.5 Das Barrieverfahren

Das Barrieverfahren geht ähnlich vor wie das Strafverfahren, hier werden Punkte bestraft, die sich (von innen) den Grenzen des Zulässigkeitsbereichs nähern. Das bedeutet, dass man, ausgehend von einem inneren Punkt, im Inneren des Zulässigkeitsbereichs bleibt. Daher nennt man Barrieverfahren auch innere Penaltyverfahren.

Das Ausgangsproblem (GP) wird umgewandelt in ein Problem der folgenden Form:

$$\min_{x \in \mathbb{R}^d} B_\mu(x), \text{ mit } \mu > 0, \quad (7.3)$$

wobei die **Barrierefunktion** B_μ definiert ist als:

$$B_\mu(x) := f(x) + \mu \text{Bar}(x); \quad (7.4)$$

Der Term $\text{Bar}(x)$ ist wie bei der Penaltyfunktion der Barriereterm und μ , mit $\mu \rightarrow 0$, bezeichnet den Barriereparameter.

Definition 7.14 Barrierefunktionen für (GP) sind Abbildungen $B_\mu : \mathbb{R}^d \rightarrow \mathbb{R}$,

$$B_\mu(x) = f(x) + \mu \sum_{i=1}^m \vartheta(g_i(x))$$

mit den folgenden Eigenschaften für $\vartheta : \mathbb{R} \rightarrow \mathbb{R}$:

1. ϑ ist auf \mathbb{R}^- stetig.
2. $\vartheta(y) \rightarrow \infty$ für $y \rightarrow 0$.

Beispiel 7.15 Typische Barriereterme sind z. B.

1. $\text{Bar}(x) = -\sum_{i=1}^m \frac{1}{g_i(x)}$.
2. $\text{Bar}(x) = -\sum_{i=1}^m \ln(\min\{0, -g_i(x)\})$.
3. $\text{Bar}(x) = -\sum_{i=1}^m \ln(-g_i(x))$.

Das Barrieverfahren läuft analog zum Penaltyverfahren ab, mit dem Unterschied, dass als Startpunkt ein innerer Punkt benötigt wird. Der Barriereparameter μ wird nach jeder Iteration verkleinert.

Algorithmus 7.16 Barriere-Methode

1. Input: Toleranzgrenze ε und Verkleinerungsfaktor $\gamma \in]0, 1[$, Startpunkt $x^{(0)}$, Barriereparameter $\mu_0 > 0$, setze $k = 0$;

2. $k = k + 1$;
3. Bestimme mit $x^{(k-1)}$ als Startpunkt die Minimallösung $x^{(k)}$ von

$$\arg \min_{x \in \mathbb{R}^d} B_{\mu_k}(x) = \arg \min_{x \in \mathbb{R}^d} (f(x) + \mu_k \text{Bar}(x));$$

4. (a) Falls $g(x^{(k)}) \leq \varepsilon$:
Lösung ist $x^{(k+1)}$;
- (b) Falls $g(x^{(k)}) > \varepsilon$:
 $\mu_{k+1} = \gamma \mu_k$,
Gehe zu 2.

Für Barrieremethoden kann der folgende Satz bewiesen werden:

Satz 7.17 Seien $f : \mathbb{R}^d \rightarrow \mathbb{R}$ und $g_i : \mathbb{R}^d \rightarrow \mathbb{R}$ für $i = 1, \dots, m$ konvex, und sei (μ_k) eine positive Folge mit $\lim_{k \rightarrow \infty} \mu_k = 0$. Sei $(x^{(k)})$ die Folge der durch die Barriermethode (ohne Abbruchkriterium) ermittelten optimalen Lösungen von (7.4). Dann gilt:

1. Die Funktion B_{μ_k} sind konvex.
2. Die Folge $(x^{(k)})$ ist beschränkt.
3. Für alle $k \in \mathbb{N}$ mit $\lim_{k \rightarrow \infty} f(x^{(k)}) = f^*$ gilt
$$0 \leq f(x^{(k)}) - f^* \leq \mu_k m.$$
4. Jeder Häufungspunkt der Folge $(x^{(k)})$ ist ein globales Minimum von (GP).
5. Falls (μ_k) eine monoton fallende Folge und $(x^{(k)})$ konvergent ist, dann gilt:

$$\lim_{k \rightarrow \infty} B_{\mu_k} = f^*.$$

Für den Beweis siehe [Ab01].

7.6 Multiplikator-Methoden

Eine in der Praxis häufig verwendete Methode stellt eine Mischform der Penalty- und Multiplikator-Methode dar. Ursprünglich wurde die Lagrange Multiplikatormethode für Minimierungsprobleme unter Gleichungsnebenbedingungen entwickelt, sie lässt sich jedoch z. B. mit der Methode von Rockafellar, siehe [Sp93] oder [Ro93], auch auf ungleichungsrestriktierte Probleme übertragen. Das Optimierungsproblem (GP) ist durch Einführung vorzeichenbeschränkter Schlupfvariablen äquivalent zu

$$\min_{x \in Z} f(x) \tag{GP^*}$$

$$Z = \{y = [x, z] : x \in S \subset \mathbb{R}^d, z \in \mathbb{R}_+^m; h_i(y) = g_i(x) + z_i = 0, i = \{1, \dots, m\}\}.$$

Dann kann man die erweiterte Lagrangefunktion für (GP*) definieren als:

Definition 7.18 Erweiterte Lagrangefunktion

Die Abbildung $L_A : \mathbb{R}^{d+m} \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}$ mit $y \in \mathbb{R}^{d+m} = [x, z]$, $x \in \mathbb{R}^d$ und $z \in \mathbb{R}_+^m$ mit:

$$L_A(y, \mu, \rho) = f(x) + \sum_{i=1}^m \mu_i h_i(y) + \rho \sum_{i=1}^m h_i(y)^2$$

heißt erweiterte Lagrangefunktion, (engl. augmented Lagrangian).

Bemerkung 7.19 Die ersten zwei Teile von $L_A(y, \mu, \rho)$ entsprechen der Lagrangefunktion von (GP^*) , während der letzte Teil von $L_A(y, \mu, \rho)$ einen quadratischen Penaltyterm mit Penaltyparamter ρ darstellt. Daher auch der Name erweiterte Lagrangefunktion.

Minimierung von $L_A(y, \mu, \rho)$ nach z ist äquivalent zu:

$$\min_{z \in \mathbb{R}_+^m} L_A(x, z, \mu, \rho) = f(x) + \sum_{i=1}^m \min_{z_i \geq 0} (\mu_i [g_i(x) + z_i] + \rho [g_i(x) + z_i]^2).$$

Das heißt man minimiert eine quadratische Funktion mit Vorzeichenbeschränkung. Man hat also zwei Möglichkeiten, entweder entspricht das Minimum dem stationären Punkt der quadratischen Funktion, oder dem Nullpunkt, siehe auch Abbildung 7.1.

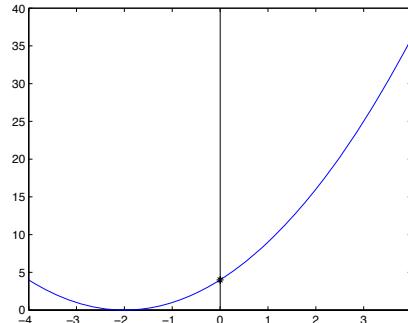


Abbildung 7.1: Falls die Minimallösung einer quadratischen Funktion negativ ist, ist der Ursprung die zulässige Minimallösung.

Differentiation nach z_i liefert die Optimalitätsbedingung:

$$\mu_i + 2\rho[g_i(x) + z_i] = 0, \text{ für } z_i > 0.$$

Damit ist für $z \geq 0$ die Optimallösung gegeben durch $z_i^* = \max\{0, -[g_i(x) + \frac{\mu_i}{2\rho}]\}$, $i = \{1, \dots, m\}$. Es ergibt sich für

$$g_i(x) + z_i = \begin{cases} -\frac{\mu_i}{2\rho}, & \text{falls } -[g_i(x) + \frac{\mu_i}{2\rho}] > 0, \text{ d.h. } z_i > 0 \\ g_i(x), & \text{falls } -[g_i(x) + \frac{\mu_i}{2\rho}] \leq 0, \text{ d.h. } z_i = 0 \end{cases}$$

und für die erweiterte Lagrangefunktion (auch erweiterte Lagrangefunktion von Rockafellar genannt):

$$L_A(x, \mu, \rho) = f(x) + \sum_{i=1}^m \begin{cases} \frac{\mu_i^2}{4\rho}, & \text{falls } -[g_i(x) + \frac{\mu_i}{2\rho}] > 0 \\ \mu_i g_i(x) + \rho g_i(x)^2, & \text{falls } -[g_i(x) + \frac{\mu_i}{2\rho}] \leq 0. \end{cases} \quad (7.5)$$

Mit Hilfe der erweiterten Lagrangefunktion (7.5) kann man das Problem (GP) z. B. mit der Multiplikator-Methode von Hestenes, siehe z. B. [Sp93], lösen.

Algorithmus 7.20 Multiplikator-Methode von Hestenes

1. Input: $x^{(0)} \in \mathbb{R}^d$, $\mu^{(0)} \in \mathbb{R}^m$, $\rho^{(0)} > 0$,
Vergrößerungsfaktor $\gamma > 1$, setze $k = 0$;

2. $k = k + 1$;

3. Bestimme die Minimallösung $x^{(k)}$ von

$$\arg \min_{x \in \mathbb{R}^d} L_A(x, \mu^{(k-1)}, \rho^{(k-1)})$$

mit $x^{(k-1)}$ als Startwert,
falls die Minimierung versagt:
setze $\rho^{(k)} = \beta \rho^{(k-1)}$,
 $x^{(k)} = x^{(k-1)}$,
gehe zu 2.;

4. Setze $\rho^{(k)} = \rho^{(k-1)}$,
 $\mu_i^{(k)} = \mu_i^{(k-1)} - 2\rho^{(k)}(g_i(x^{(k)}) - \max\{0, -[g_i(x^{(k)}) + \frac{\mu_i^{(k-1)}}{2\rho^{(k)}}]\})$;

5. Falls $x^{(k)} \in Z$ und $\nabla_x L(x^{(k)}, \mu^{(k)}) = 0$:
Lösung ist $x^{(k)}$,
sonst: gehe zu 2.

Die Minimierung in Schritt 3. von Algorithmus 7.20 kann mit dem Algorithmus von Nelder-Mead durchgeführt werden. Sind $f, g \in \mathcal{C}^2$ und gilt in x^* die strikte Komplementarität (siehe Definition 7.4), so ist $L_A(x^*, \mu^*) \in \mathcal{C}^2$ und man könnte auch ein auf Ableitungen beruhendes Minimierungsverfahren anwenden.

7.7 Penalty-Barrier-Multiplier Methoden (PBM)

Penalty-Barrier-Multiplier Methoden sind eine Kombination aus Strafverfahren, Barrierverfahren und Multiplikatorverfahren. Begründet wurde dieses Vorgehen von Polyak, siehe [Po92]. Ben-Tal und Zibulevsky, siehe [BZ95], haben die PBM-Methode für konvexe und semi-definite Probleme formuliert. Kocvara und Stingl, siehe [KS05], verallgemeinerten die PBM-Methode und entwickelten die zugehörige Software PENNON, siehe auch [KS03].

Wir wollen den PBM-Algorithmus anwenden, um mit dem Nelder-Mead Algorithmus ein Optimierungsprobleme der Form (GP) lösen zu können. Dies geschieht durch eine „Ummantelung“ mit einem PBM-Algorithmus. Wie bislang beschrieben,

wird statt dem ursprünglichen Problem eine Folge von unrestringierten Optimierungsproblemen gelöst. Dabei wird in jeder Iteration die erweiterte Lagrangefunktion minimiert. Durch einen geeigneten Update der Lagrangenmultiplikatoren und Strafparameter erreicht man, dass sich die lokalen Minima der erweiterten Lagrangefunktion immer besser den lokalen Minima von (GP) angleichen. Der Namensbestandteil „Barrier“ kommt daher, dass eine zusammengesetzte Penaltyfunktion verwendet wird, die aus einem an quadratische Penaltyfunktionen erinnernden und einem an logarithmische Barrierefunktionen erinnernden Teil besteht.

In jeder Iteration wird ein Problem der Form

$$\min_{x \in \mathbb{R}^d} F_k(x, u_k, p_k) \text{ für } u_k, p_k \in \mathbb{R}^m$$

gelöst. Wie bei den Penaltyverfahren muss die Bedingung $x \in S$ in weitere Ungleichungsnebenbedingungen umgeformt werden. Die erweiterte Lagrangefunktion $F_k(x, u_k, p_k)$ ist hierbei definiert als:

$$F_k(x, u_k, p_k) := f(x) + \sum_{i=1}^m u_{k_i} p_{k_i} \varphi\left(\frac{g_i(x)}{p_{k_i}}\right), \quad (7.6)$$

mit der Notation:

- Die $u_k \in \mathbb{R}^m$ bezeichnen die Lagrangenmultiplikatoren.
- Die $p_k \in \mathbb{R}^m$ sind Strafparameter.
- Die Funktion $\varphi : \mathbb{R} \rightarrow \mathbb{R}$ muss folgende Voraussetzungen erfüllen:
 1. φ ist strikt konvex und monoton steigend.
 2. $\varphi \in C^2$.
 3. Der Definitionsbereich ist $D(\varphi) = (-\infty, b)$ mit $0 < b \leq \infty$.
 4. $\varphi(0) = 0$.
 5. $\varphi'(0) = 1$.
 6. $\lim_{t \rightarrow b^-} (\varphi'(t)) = \infty$.
 7. $\lim_{t \rightarrow -\infty} (\varphi'(t)) = 0$.
 8. $\varphi''(t) \geq \frac{1}{M}$, $\forall t \in [0, b]$, für ein $M > 0$.

Dann gilt, dass die Nebenbedingung $g_i(x) \leq 0$ gleichbedeutend ist zu:

$$p_i \varphi\left(\frac{g_i(x)}{p_i}\right) \leq 0, \text{ für } p_i > 0, i = \{1, \dots, m\}.$$

Beispiel 7.21 Geeignete φ sind z. B.:

1. Quadratisch - Logarithmische Penalty/Barrier Funktion:

$$\varphi_r(t) = \begin{cases} c_1 \frac{1}{2}t^2 + c_2 t + c_3 & t \geq r \\ c_4 \log(c_5 t) + c_6 & t < r. \end{cases}$$

Für die Parameter $r = -1/3$, $b = \infty$ und den Konstanten

$$c_1 = 1, c_2 = 1, c_3 = 0, c_4 = \frac{32}{27}, c_5 = 1, c_6 = -\frac{7}{6}$$

zeigt Abbildung 7.2 das zugehörige φ .

2. Quadratisch - Reziproke Penalty/Barrier Funktion:

$$\varphi_r(t) = \begin{cases} c_1 \frac{1}{2}t^2 + c_2 t + c_3 & t \geq r \\ \frac{c_4}{(c_5 - t)} + c_6 & t < r. \end{cases}$$

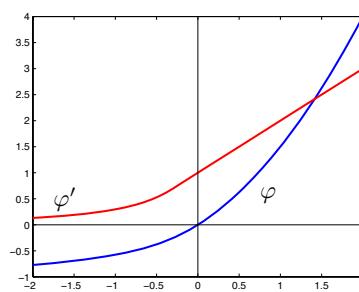


Abbildung 7.2: Beispiel für φ .

Um den PBM-Algorithmus zu formulieren benötigen wir noch eine monoton steigende Penalty-Update-Funktion $\tau : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$. Die Funktion $\tau(t)$ kann entweder konstant oder linear gewählt werden, Beispiele werden in Kapitel 8.1.3 angegeben.

Algorithmus 7.22 PBM-Algorithmus

1. Input:
Startwerte $u^{(0)}$ und $p^{(0)}$ mit $u^{(0)}, p^{(0)} \in \mathbb{R}_+^m$ und $x^{(0)} \in Z$,
Penalty-Update-Funktion τ ,
Maximale Zahl von Iterationen: k_{max} ,
setze $k = 0$;
2. $k = k + 1$,
Bestimme die Minimallösung $x^{(k)}$ von
$$\arg \min_{x \in \mathbb{R}^d} F(x, u^{(k-1)}, p^{(k-1)})$$
mit Startpunkt $x^{(k-1)}$;
3. $u_i^{(k)} = u_i^{(k-1)} \varphi'(\frac{g_i(x^{(k)})}{p_i^{(k-1)}})$ für $i = 1, \dots, m$;
4. $p_i^{(k)} = \tau^{(k)} p_i^{(k-1)}$ für $i = 1, \dots, m$;
5. Falls $k < k_{max}$:
Gehe zu 2.

Bemerkung 7.23 Die Minimierung in Schritt 2. des PBM-Algorithmus erfolgt mit dem Nelder-Mead Algorithmus.

Für konvexe Optimierungsprobleme lässt sich ein Beweis des folgenden Satzes angeben, siehe [BZ95].

Satz 7.24 Seien f, g konvex und die Funktion τ_k sublinear für alle k , d. h. $\tau_k(t)$ ist für alle $t > 0$ monoton steigend und $\tau_k(t) \leq ct$ für ein $c > 0$. Seien $(x^{(k)})$ und $(u^{(k)})$ die von Algorithmus 7.22 für $k \rightarrow \infty$ erzeugten Folgen. Dann gilt:

1. $\max\{0, g_i(x^{(k)})\} \rightarrow 0$ für $i = 1, \dots, m$.
2. $f(x^{(k)}) \rightarrow f^*$. (f^* bezeichnet die Lösung von (GP)).
3. $u_i^{(k)} g_i(x^{(k)}) \rightarrow 0$ für $i = 1, \dots, m$.
4. Die Folgen $(x^{(k)})$ und $(u^{(k)})$ sind beschränkt. Jeder Häufungspunkt von $(x^{(k)})$ ist eine globale Minimallösung von (GP).

7.8 Beispiele

Die folgenden Abbildungen zeigen, wie sich die erweiterte Lagrangefunktion in den Iterationen des PBM-Algorithmus verändert und sich dem Ausgangsproblem immer besser anpasst. Als letztes Bild sieht man jeweils die Höhenlinien der eigentlichen Zielfunktion und den zugehörigen Nebenbedingungen. Die betrachteten Testprobleme sind in Kapitel 8 beschrieben.

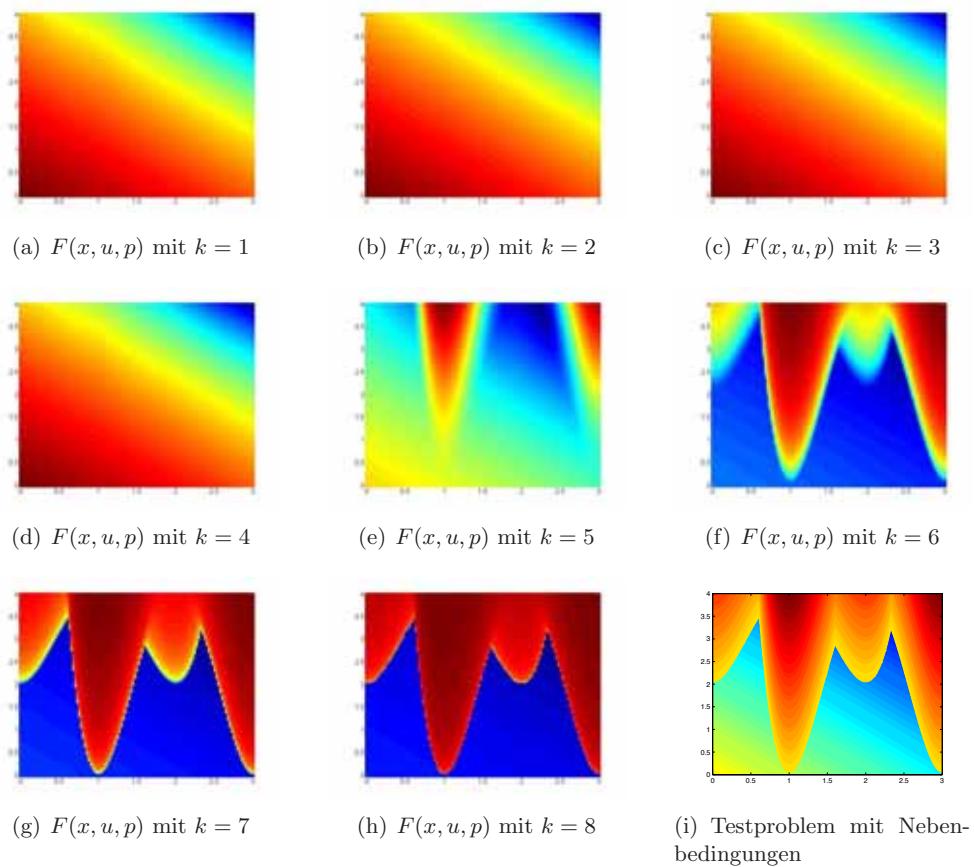


Abbildung 7.3: erweiterte Lagrange-funktion und Zielfunktion mit Nebenbedingungen für Testbeispiel 18.

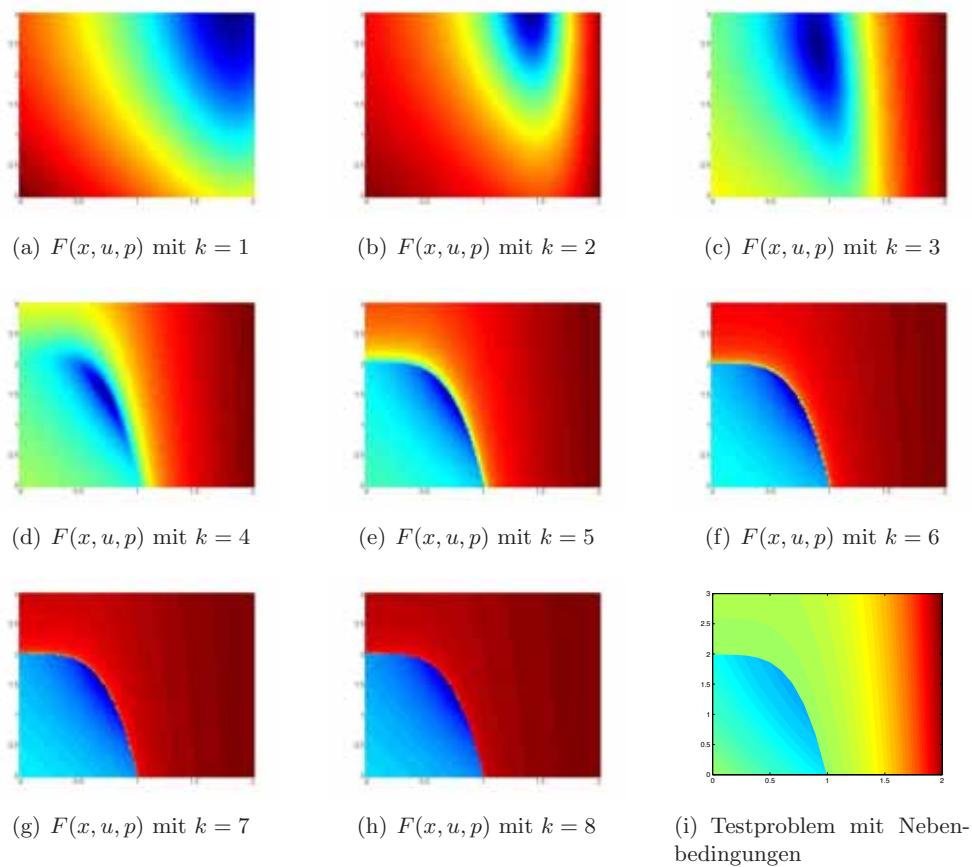


Abbildung 7.4: Erweiterte Lagrangefunktion und Zielfunktion mit Nebenbedingungen für Testbeispiel 19.

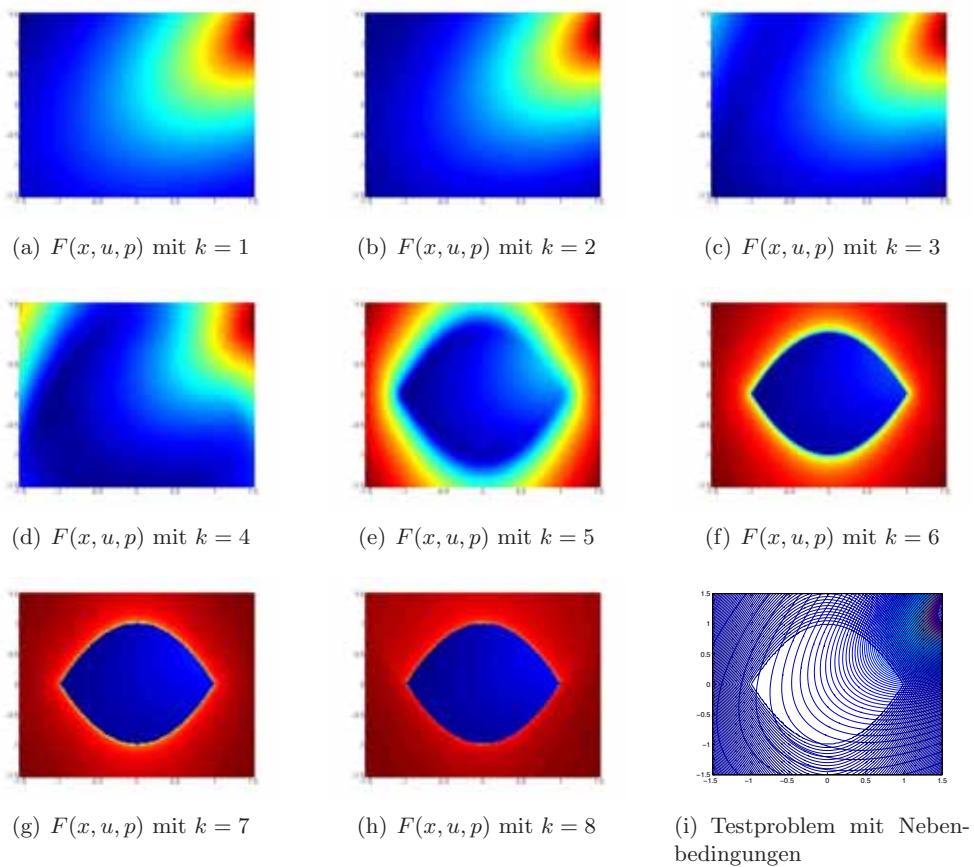


Abbildung 7.5: Erweiterte Lagrangefunktion und Zielfunktion mit Nebenbedingungen für Testbeispiel 20.

Kapitel 8

Implementierung in Matlab und numerische Ergebnisse

8.1 Übersicht über die implementierten Routinen

Im vorliegenden Kapitel werden die implementierten Routinen mit benötigtem In- und Output der Hauptfunktionen, einer kurzen Erläuterung der Funktionsweise, Auflistung der Unterfunktionen und Anmerkungen zu Besonderheiten vorgestellt. Alle Routinen befinden sich, zusammen mit den implementierten Testfunktionen, auf der beigefügten CD. Anschließend werden die betrachteten Testfunktionen vorgestellt und eine Auswertung durchgeführt.

8.1.1 HCP-Algorithmus

Die Hauptroutine ist $[x_{opt}, f_{opt}] = \text{globmin_hcp}(f, a, b)$ und löst das globale Optimierungsproblem

$$\begin{aligned} \min \quad & f \\ \text{s.t. } & a \leq x \leq b \end{aligned}$$

mit Algorithmus 2.10.

Die Ausgabe x_{opt} bezeichnet die gefundene Minimallösung mit $f_{opt} = f(x_{opt})$. Durch den Aufruf $[x_{opt}, f_{opt}] = \text{globmin_hcp}(f, a, b, \alpha, k, \varepsilon)$ können die Verfahrensparameter

- $\alpha \in [0, 1]$ steuert die Adaptivität ($\alpha = 1$ bedeutet, dass das verfahren nicht adaptiv vorgeht);
- $k \in \mathbb{N}$ beschränkt den maximalen Level aller betrachteten HCP und wirkt somit als Abbruchkriterium;
- $\varepsilon \geq 0$ gibt den minimalen Abstand der Punkte an;

Werden diese nicht angegeben, arbeitet das Verfahren mit den Standardwerten, siehe Tabelle 2.1.

Die verwendeten Unterfunktionen sind:

1. `initHCP_parameters` initialisiert die Parameter, falls diese nicht beim Aufruf der Funktion `globmin_hcp` angegeben wurden.

2. Die Funktion `initHCP_info` initialisiert die Datenstruktur `info`, die alle notwendigen Informationen für das Verfahren enthält:

- (a) `info.x` die berechneten HCP.
- (b) `info.y` die HCP transformiert in S .
- (c) `info.f` die Funktionswerte der transformierten Punkte.
- (d) `info.level` den Level aller HCP.
- (e) `info.degree` den Grad aller HCP.
- (f) `info.rank` den Rang aller HCP.
- (g) `info.g` die Werte der Bewertungsfunktion.

In jeder Iteration werden alle neuen Punkte und die zugehörigen Informationen an die Struktur `info` angehängt. Die Informationen Rang und Wert der Bewertungsfunktion müssen in jeder Iteration neu bestimmt werden.

- 3. Die Routine `getInformation` berechnet in jeder Iteration die Nachbarn des besten Punktes (mit der subroutine `getNeighbours`, siehe auch Definition 2.5) und bestimmt die zugehörige Informationsstruktur.
- 4. Schließlich benötigt man noch die Funktion `transform`, die die HCP aus dem Gebiet $[-0.5, 0.5]^d$ in den Quader $S = [a, b]$ transformiert, siehe (1.2).

8.1.2 Rekursive dünne Gitter-Berechnung

Die Hauptroutine ist $X = \text{sparsegrid}(l, d, \text{gridtype})$, die die bestimmten Gitterpunkte (in $[-0.5, 0.5]^d$) des d -dimensionalen dünnen Gitters mit Level l und Typ `gridtype` als Spalten der Matrix X zurück gibt. Der Aufbau dieser Funktion ist wie in Algorithmus 3.18 beschrieben. Die eindimensionalen Gitter werden mit der Funktion `onedim_gridpoints(l, gridtype)` in den entsprechenden Subroutinen bestimmt. Als Typen sind möglich:

1. 'boundary'
2. 'maximum'
3. 'noboundary'
4. 'clenshaw'
5. 'legendre'
6. 'chebyshev'
7. 'legendre_cubic'

Zur Berechnung der Nullstellen der Legendrepolygone benötigt man die Subroutine $y = \text{legendrepolynomials}(n)$ die die Rekursionsformel (siehe Definition 3.26) und die Matlab Funktion `roots` verwendet. Schließlich benötigt man noch die Funktion $u = \text{tensorprod}(v, w)$, die das kartesische Produkt der Vektoren (bzw. Matrizen) v und w bestimmt.

8.1.3 Globale Optimierung auf dünnen Gittern

1. Globale Optimierung auf einem gegebenen Gitter

Die Hauptroutine ist `[xopt, fopt] = globmin_adaptgrid(f, a, b, l, alpha, mP, gridtype)` und bestimmt die globale Lösung der Problems

$$\min_{x \in [a,b]} f.$$

Die Routine arbeitet wie in Algorithmus 4.1 beschrieben. Die Verfahrensparameter sind

- $l \in \mathbb{N}$ bildet eine Schranke für den maximalen Level aller betrachteten Punkte.
- $\alpha \in [0, 1]$ bestimmt den Grad der Adaptivität des Verfahrens. Für $\alpha = 1$ arbeitet das Verfahren nichtadaptiv.
- mP gibt die maximale Anzahl an Funktionsauswertungen an und ist somit ein Abbruchkriterium.
- `gridtype` kann hier sein:
 - 'boundary'
 - 'maximum'
 - 'noboundary'
 - 'clenshaw'
 - 'legendre_cubic'

2. Globale Optimierung mit iterativer Gittererzeugung und Berücksichtigung von Nebenbedingungen

Die Hauptfunktion

`[xopt, fopt] = globmin_itergrid_con(f, A, B, Aeq, Beq, a, b, C, Ceq, alpha, mP, gridtype)`

bestimmt die globale Lösung des Problems

$$\begin{aligned} & \min f \\ \text{s. t. } & Ax \leq B \\ & Aeqx = Beq \\ & C(x) \leq 0 \\ & Ceq(x) = 0 \\ & a \leq x \leq b. \end{aligned}$$

Nachdem keine Gleichungsnebenbedingungen berücksichtigt werden können, werden diese ignoriert.

Falls keine Nebenbedingungen existieren, lautet der Aufruf

`[xopt, fopt] = globmin_itergrid_con(f, [], [], [], [], a, b, [], [], alpha, mP, gridtype).`

Falls nur einzelne Nebenbedingungen nicht besetzt sind, schreibt man $A = []$ etc.

Zur Wirkung der Verfahrensparameter:

- $\alpha \in [0, 1]$ bestimmt den Grad der Adaptivität des Verfahrens (siehe auch HCP-Algorithmus).
- $mP \in \mathbb{N}$ beschränkt die maximale Anzahl der Funktionsauswertungen und bildet somit das Abbruchkriterium.
- Als Gittertypen kann man 'maximum', 'noboundary', 'clenshaw' oder 'legendre' zugrunde legen, wobei hier 'legendre' eigentlich 'legendre_cubic' liefert.

Die Routine `globmin_itergrid_con` besteht aus mehreren Teilen:



Abbildung 8.1: Schematischer Aufbau von `globmin_itergrid_con`.

Iterative, adaptive Gittererzeugung

Der Ablauf des ersten Teils ist wie in Algorithmus 4.13 beschrieben. Mit Hilfe der folgenden Unterfunktionen wird ein Punkt, der sich in der Nähe der globalen Lösung befindet, gesucht.

1. `getStartingPoints`, zur Bestimmung der Startpunkte für das jeweilige Gitter und der zugehörigen Positionsangaben, siehe Algorithmus 4.13.
2. `getNewPoints`, zur Bestimmung der Nachbarn des besten Punktes zusammen mit der Position. Dies geschieht mit Hilfe der Funktionen `posToReal`, bzw. `posToReal_legendre` für das Legendre Gitter, siehe Bemerkung 4.15.
3. `keepPositions`, die die Positionsmatrizen der neuen Punkte an die bestehenden Positionsangaben anfügt. Es wird wieder mit einer Struktur `info`, wie im HCP-Algorithmus beschrieben, gearbeitet.
4. `penaltymethod`, zur Bestimmung des Straftermes $Pen(x)$ für die exakte Penaltyfunktion, der zur eigentlichen Zielfunktion addiert wird (0 falls $x \in Z$ und ∞ falls $x \notin Z$), siehe Beispiel 7.13.

Clusteranalyse

Nach Ablauf des ersten Teils des Verfahrens wird die Clusteranalyse zur Bestimmung geeigneter Startpunkte für das anschließende lokale Verfahren durchgeführt. Dies geschieht mit der Methode $[C, MC] = \text{cluster_estimation}(X, F, r_a)$, wie in Algorithmus 5.9 beschrieben.

- Die Menge X besteht aus den zulässigen, im ersten Teil des Verfahrens bestimmten, Punkten. Hier werden die normalisierten Punkte ($X \in [-0.5, 0.5]^d$ verwendet).
- Die Menge F besteht aus den Funktionswerten an den Punkten in X .
- Der Parameter r_a bestimmt den Durchmesser der Cluster. Der Standardwert ist 0.4, siehe auch Bemerkung 5.10.

Es werden nicht nur die ermittelten Clustermittelpunkte C ausgegeben, sondern auch die Minima der zu C gehörenden Cluster bestimmt und in MC ausgegeben. Die lokale Optimierung benutzt die Punkte in MC als Startpunkte. Falls viele Startpunkte bestimmt wurden, lohnt es sich evtl. zu überprüfen wie dicht diese beieinander liegen. Wir betrachten nur Startpunkte die mehr als 0.01 voneinander entfernt sind (die Daten sind auf $[-0.5, 0.5]^d$ normalisiert).

Lokale Suche und PBM-Algorithmus

Zur lokalen Suche wurde der Algorithmus von Nelder und Mead implementiert, siehe Algorithmus 6.3. Der Aufruf erfolgt mit $[x_{opt}, f_{opt}] = \text{nelder_mead}(f, x_0)$. Es wird eine unbeschränkte Minimierung der Zielfunktion f mit Startpunkt x_0 (die Minima der Cluster) durchgeführt. Zusätzlich können über den Aufruf $\text{nelder_mead}(f, x_0, sw, mP)$ noch zwei weitere inputs angegeben werden: Die Zahl sw gibt die Schrittweite zur Bestimmung des Startsimplex an, voreingestellt ist die 1. Wir wählen einen Bruchteil des minimalen Abstandes zwischen oberer und unterer Grenze. Daneben kann als zusätzliches Abbruchkriterium die maximale Anzahl der Funktionsauswertungen mP angegeben werden, hier ist $mP = 500d$ die Voreinstellung. Weitere, interne, Verfahrensparameter sind:

- Die Toleranzgrenze, $\varepsilon > 0$, gibt an, ab welcher Verbesserungsrate der Funktionswerte, bzw. Abstand der Simplexecken, das Verfahren abgebrochen wird. Wir setzen $\varepsilon = 10^{-5}$.
- Die Konditionszahl der Matrix K , siehe Definition 6.4, wird mit der Matlab Funktion `cond` berechnet. Es erfolgt ein Neustart im aktuell besten Punkt, falls die Zahl $\frac{1}{\kappa} \leq \varepsilon$ ist.

Bemerkung 8.1 Weder bei der globalen, noch bei der lokalen Suche lässt sich eine Auswertung der Zielfunktion im unzulässigen Bereich vermeiden. Daher muss der Anwender sicherstellen, dass die Auswertung hier keinen Fehler wirft. Dies kann bei Zielfunktionen aus der Praxis oftmals der Fall sein, da man häufig außerhalb des zulässigen Bereichs keine Stetigkeit, etc. voraussetzen kann. Daneben kann natürlich auch die Auswertung der Nebenbedingung einen Fehler werfen, auch hier muss der Anwender eine Fehlerabfrage einbauen. Diese kann in Matlab z. B. mit

einem `try...catch...end`-Block realisiert werden. Innerhalb des `iterative_grid` Algorithmus könnte man im Fall eines Fehlers einfach den Wert `Inf` zurückgeben, allerdings ist dies für den lokalen Teil nicht geeignet. Wenn z. B. alle Ecken des Simplex unzulässig sind, kann es passieren, dass sich das Simplex immer weiter zusammenzieht und sich somit nicht mehr aus dem unzulässigen Bereich hinausbewegen könnte. Hier muss im Fall eines Fehlers ein sehr hoher Funktionswert zurückgegeben werden, z. B. das Maximum aller bislang bestimmten Funktionswerte. Im Fall eines Fehlers bei der Auswertung der Nebenbedingungen kann der Wert z. B. auf 1 gesetzt werden.

Für jeden, mit der Clusteranalyse bestimmten, Startpunkt wird der Algorithmus von Nelder und Mead auf die erweiterte Lagrangefunktion angewendet. Dies geschieht innerhalb der Methode

$$[x_{opt}, f_{opt}] = \text{pbm}(f, x^{(k)}, \tau^{(k)}, u^{(k)}, p^{(k)}, LB, UB, A, B, Aeq, Beq, C, Ceq).$$

Mit der subroutine `augLagrange` wird die erweiterte Lagrangefunktion als „anonyme“ Funktion bestimmt und als Input an den Nelder-Mead Algorithmus weitergegeben. Die Methode wird 20 mal hintereinander aufgerufen. Der Startpunkt ist dann jeweils die Lösung $x^{(k)}$ aus der letzten Iteration. Der Update der u und p erfolgt wie in Algorithmus 7.22 beschrieben. Nach Ben-Tal und Zibulevsky, siehe [BZ95], muss man, um $x^{(k)}$ als Startpunkt verwenden zu können, sicherstellen, dass sich die erweiterte Lagrangefunktion $F(x, u^{(k)}, p^{(k)})$ nicht zu stark von $F(x, u^{(k-1)}, p^{(k-1)})$ unterscheidet. Um dies zu gewährleisten muss die folgende Abfrage beim Update der $u^{(k)}$ eingefügt werden:

Sei $\bar{u}_i^{(k)} = u_i^{(k-1)} \varphi'(\frac{g_i(x^{(k)})}{p_i^{(k-1)}})$ und $0 < \delta < 1$, z. B. $\delta = 0.3$, dann gilt:

$$u_i^{(k)} = \begin{cases} \delta u_i^{(k-1)}, & \text{falls } \frac{\bar{u}_i^{(k)}}{u_i^{(k-1)}} < \delta \\ \bar{u}_i^{(k)}, & \text{falls } \delta \leq \frac{\bar{u}_i^{(k)}}{u_i^{(k-1)}} \leq \frac{1}{\delta} \\ \frac{1}{\delta} u_i^{(k-1)}, & \text{falls } \frac{\bar{u}_i^{(k)}}{u_i^{(k-1)}} > \frac{1}{\delta}. \end{cases}$$

Als Startwert wird $u_i^{(0)} = 0.01$ ($\forall i$) empfohlen, für $p_i^{(0)}$ kann man z. B. 1 wählen. Neben der Penaltyfunktion φ , die wie in Beispiel 7.21 beschrieben implementiert wurde, benötigt man noch eine Penalty-Update-Funktion τ . Hier kann man z. B.

$$\tau_1^{(k)}(t) = \delta^{(k)} \tau_0 t \text{ oder } \tau_2^{(k)}(t) = \delta^{(k)} \tau_0, \quad \tau_0 > 0$$

verwenden. Wir haben uns für $\tau_2^{(k)}$ mit $\tau_0 = 2$ entschieden.

8.2 Numerische Ergebnisse

Um die betrachteten und implementierten Verfahren mit anderen Methoden vergleichen zu können und eine Aussage über die Güte treffen zu können, wurden sie auf eine Auswahl bekannter Testfunktionen angewendet, siehe [TZ89] und [FP90]. Wir werden die Testbeispiele zunächst vorstellen und dann eine Tabelle mit Rechenzeiten, Anzahl der Funktionsauswertungen und bestimmter Lösung angeben.

8.2.1 Testfunktionen

1. B, Beale, $d = 2$

$$f_B(x) = (1.5 - x_1 x_2)^2 + (2.25 - x_1(1 - x_2^2)^2 + (2.625 - x_1(1 - x_2^3)))^2, \quad -5 \leq x \leq 5.$$

Die optimale Lösung ist $f^* = 0$ bei $x^* = (3.2226, 0.4654)$.

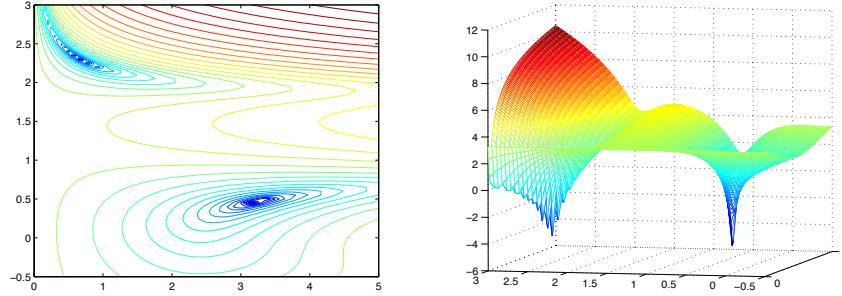


Abbildung 8.2: Höhenlinien der Testfunktion f_B .

2. BR, Branin, $d = 2$

$$f_{BR}(x) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos(x_1) + 10$$

$$-5 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 15.$$

Mit drei globalen Minima $f^* = 0.3979$ bei $x^* = (-3.142, 12.275), (3.142, 2.275)$ und $x^* = (9.425, 2.425)$.

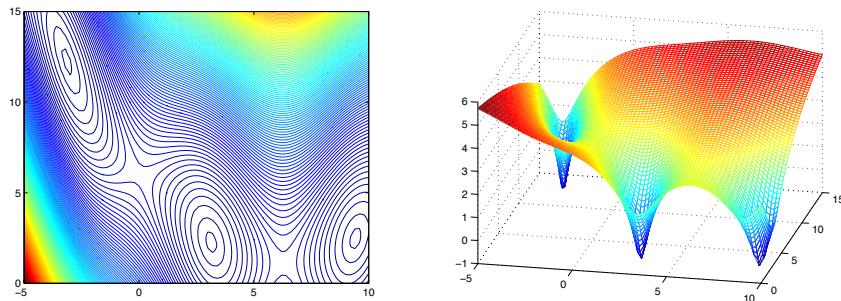


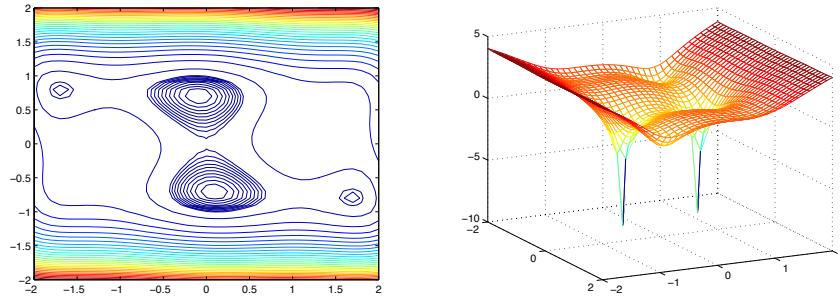
Abbildung 8.3: Höhenlinien der Testfunktion f_{BR} .

3. C, Six-Hump-Camel-Back Funktion, $d = 2$

$$f_C(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4, \quad -5 \leq x_i \leq 5$$

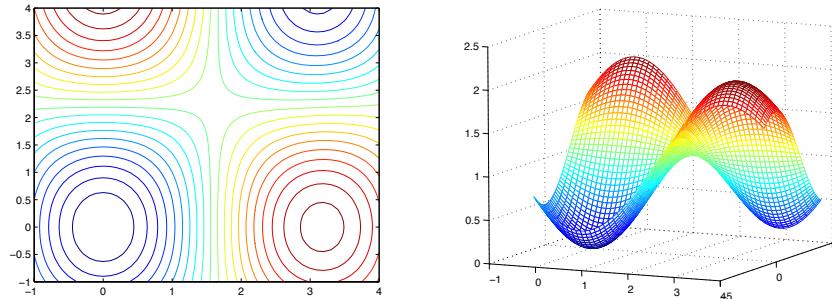
hat zwei globale Minima $f^* = -1.0316$ bei $x^* = (0.08983, -0.7126)$ und $(-0.08983, 0.7126)$.

4. G2, G10, vereinfachte Griewank Funktionen, $d = 2$ und $d = 10$

Abbildung 8.4: Höhenlinien der Testfunktion f_C .

$$f_G = 1 + \frac{1}{d} \sum_{i=1}^d x_i^2 - \prod_{i=1}^d \cos(x_i) \quad -1 \leq x \leq 4.$$

Die optimale Lösung ist $f^* = 0$ bei $x^* = (0, \dots, 0)$.

Abbildung 8.5: Höhenlinien der Testfunktion f_{G2} .

5. GP, Goldstein - Price, $d = 2$

$$f_{GP}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

für

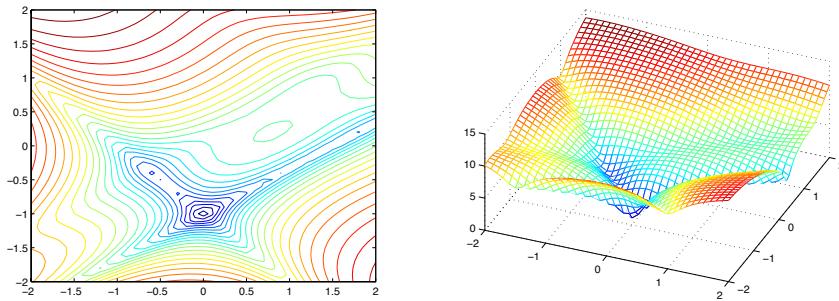
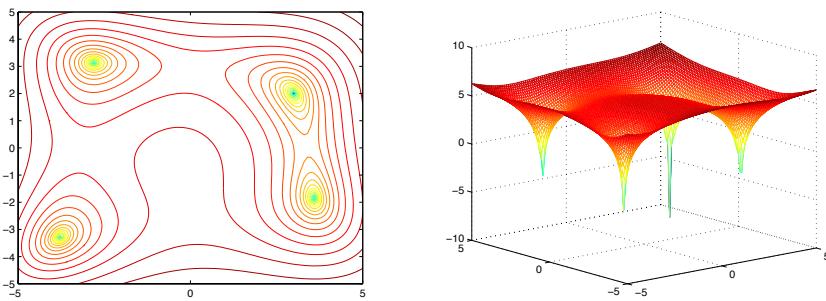
$$-2 \leq x_i \leq 2.$$

Das globale Minimum $f^* = 3$ ist bei $x^* = (0, -1)$.

6. H, Himmelblau, $d = 2$

$$f_H = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \quad -5 \leq x \leq 5;$$

Die Funktion f_H besitzt bei $x^* = (3, 2)$ ein globales Minimum mit $f^* = 0$;

Abbildung 8.6: Höhenlinien der Testfunktion f_{GP} .Abbildung 8.7: Höhenlinien der Testfunktion f_H .

7. Hn3, Hn6, Hartman, $d = 3$ und $d = 6$

$$f_{Hn}(x) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^d a_{ij}(x_j - b_{ij})^2\right]$$

für $0 \leq x_i \leq 1$, $i = \{1, \dots, n\}$. Die Parameter für $d = 3$ a , c und b sind:

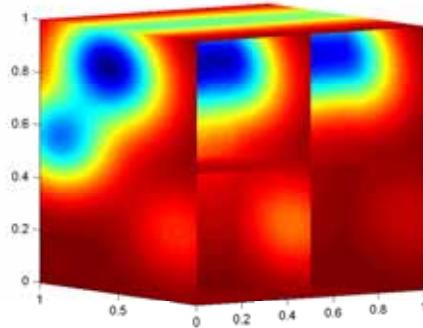
$$\begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \quad \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0382 & 0.5743 & 0.8828 \end{pmatrix}$$

für $d = 6$, sind a , c und b wie folgt:

$$\begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}$$

$$\begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.2883 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.5743 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

Hn3 hat ein globales Minimum $f^* = -3.8628$ bei $x^* = (0.114, 0.556, 0.852)$ und

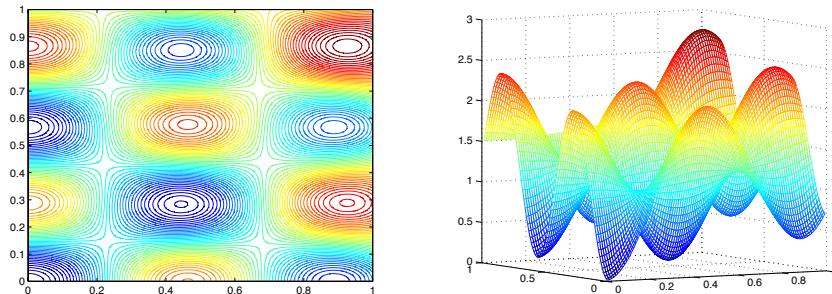
Abbildung 8.8: Die Funktion f_{Hn3} .

Hn6 hat ein globales Minimum $f^* = -3.3937$ bei $x^* = (0.196, 0.151, 0.500, 0.2812, 0.312, 0.656)$.

8. M, Mladineo, d beliebig

$$f_M(x) = 1 + \frac{1}{d} \sum_{i=1}^d x_i^2 - \prod_{i=1}^d \cos(10 \log((i+1)x_i)) \quad 0 \leq x \leq 1.$$

Für $d = 2$ hat f_M ein globales Minimum $f^* = 0$ bei $x^* = (0, 0)$.

Abbildung 8.9: Höhenlinien der Testfunktion f_M .

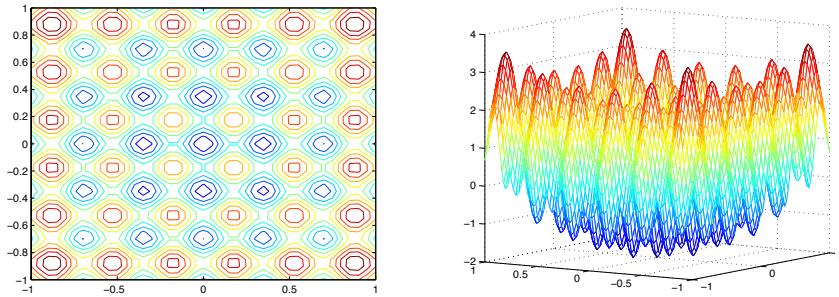
9. R, Rastrigin, $d = 2$

$$f_R(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$$

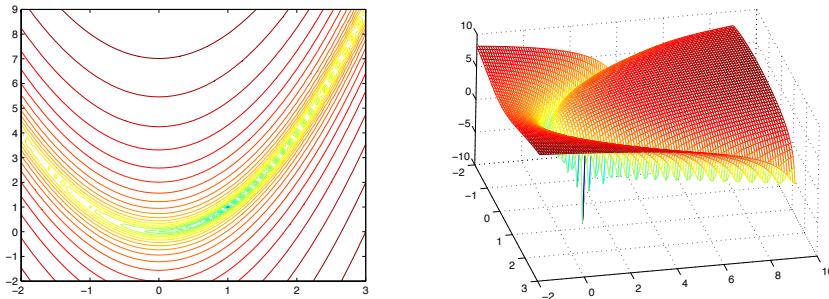
für $-1 \leq x \leq 1$. Das globale Minimum $f^* = -2$ ist im Punkt $x^* = (0, 0)$.

10. RO, Rosenbrock, $d = 2$

$$f_{RO} = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \quad -2 \leq x_1 \leq 3, -2 \leq x_2 \leq 9.$$

Abbildung 8.10: Höhenlinien der Testfunktion f_R .

Die Funktion f_{RO} besitzt bei $x^* = (1, 1)$ ein globales Minimum mit $f^* = 0$.

Abbildung 8.11: Höhenlinien der Testfunktion f_{RO} .

11. Sch, Schäffler, $d = 50$

$$f_{Sch}(x) = 1 + 6x_1^2 - \cos(12x_1) + 590 \sum_{i=2}^{50} (x_i - x_{i-1})^2, \quad -1.05 \leq x \leq 2.95.$$

Die Funktion f_{Sch} hat ein globales Minimum $f^* = 0$ im Ursprung $x^* = (0, \dots, 0)$. Dieses Problem erweist sich als zu schwierig, keines der Verfahren konvergiert hier zum globalen Minimum.

12. Testproblem

$$\begin{aligned} \min \quad & f(x) = c^T x - 0.5x^T Q x \\ \text{s.t.} \quad & 20x_1 + 12x_2 + 11x_3 + 7x_4 + 4x_5 - 40 \leq 0 \\ & [0; 0; 0; 0; 0] \leq x \leq [1; 1; 1; 1; 1] \end{aligned}$$

mit $c = [42; 44; 45; 47; 47.5]$ und $Q = 100I$; I bezeichnet hier die Einheitsmatrix. Dieses Testproblem hat ein globales Minimum bei $x^* = [1; 1; 0; 1; 0]$ mit $f(x^*) = -17$.

13. Testproblem

$$\min f(x, y) = c^T x - 0.5x^T Q x + d^T y$$

$$\begin{aligned} \text{s.t. } 6x_1 + 3x_2 + 2x_4 + x_5 - 6.5 &\leq 0 \\ 10x_1 + 10x_3 + y - 20 &\leq 0 \end{aligned}$$

$$[0; 0; 0; 0; 0; 0] \leq x \leq [1; 1; 1; 1; 1; 20]$$

mit $c = [-10.5; -7.5; -3.5; -2.5; -1.5]$, $Q = I$ und $d = -10$;

Das globale Minimum ist $[x^*, y^*] = [0; 1; 0; 1; 1; 20]$ mit $f([x^*, y^*]) = -213$.

14. Testproblem

$$\min f(x, y) = c^T x - 0.5x^T Q x + d^T y;$$

$$\begin{aligned} \text{s.t. } 2x_1 + 2x_2 + y_6 + y_7 - 10 &\leq 0 \\ 2x_1 + 2x_3 + y_6 + y_8 - 10 &\leq 0 \\ 2x_2 + 2x_3 + y_7 + y_8 - 10 &\leq 0 \\ -8x_1 + y_6 &\leq 0 \\ -8x_2 + y_7 &\leq 0 \\ -8x_3 + y_8 &\leq 0 \\ -2x_4 - y_1 + y_6 &\leq 0 \\ -2y_2 - y_3 + y_7 &\leq 0 \\ -2y_4 - y_5 + y_8 &\leq 0 \end{aligned}$$

$$[0; 0; 0; 0; 0; 0; 0; 0; 0; 0] \leq [x, y] \leq [1; 1; 1; 1; 1; 1; 1; 5; 5; 5; 1]$$

mit $c = [5; 5; 5; 5]$, $Q = 10I$ und $d = [-1; -1; -1; -1; -1; -1; -1; -1]$.

Das globale Minimum ist $[x^*, y^*] = [1; 1; 1; 1; 1; 1; 1; 3; 3; 3; 1]$ mit $f^* = -15$.

Mit den hier angegebenen Nebenbedingungen ist es sehr schwierig zulässige Punkte zu finden, daher betrachten wir das vereinfachte Problem mit den Nebenbedingungen 4,5 und 6. Dann ist die globale Lösung $[x^*, y^*] = [1; 1; 1; 0; 1; 1; 1; 1; 1; 5; 5; 5; 1]$ mit $f^* = -21$.

15. Testproblem

$$\min f(x, y) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5;$$

$$s.t. Ax \leq b$$

$$[0; 0; 0; 0; 0] \leq [x, y] \leq [6; 6; 6; 6; 6]$$

mit

$$A = \begin{pmatrix} 1 & 2 & 8 & 1 & 3 & 5 \\ -8 & -4 & -2 & 2 & 4 & -1 \\ 2 & 0.5 & 0.2 & -3 & -1 & -4 \\ 0.2 & 2 & 0.1 & -4 & 2 & 2 \\ -0.1 & -0.5 & 2 & 5 & -5 & 3 \end{pmatrix}$$

$$\mathbf{b} = [16; -1; 24; 12; 3].$$

Das globale Minimum ist $[x^*, y^*] = [0; 6; 0; 1; 1; 0]$ mit $f([x^*, y^*]) = -11$.

Auch bei diesem Problem findet man sehr schwer zulässige Punkte, daher vernachlässigen wir die 1. Nebenbedingung. Dann ist die globale Lösung $[x^*, y^*] = [0; 6; 6; 4.8; 6; 0]$ mit $f([x^*, y^*]) = -44.4$.

16. Testproblem

$$\min f(\mathbf{x}) = -25(x_1 - 2)^2 - (x_2 - 2)^2 - (x_3 - 1)^2 - (x_4 - 4)^2 - (x_5 - 1)^2 - (x_6 - 4)^2$$

$$\begin{aligned} \text{s.t. } & -(x_3 - 3)^2 - x_4 + 4 \leq 0 \\ & -(x_5 - 3)^2 - x_6 + 4 \leq 0 \\ & x_1 - 3x_2 - 2 \leq 0 \\ & -x_1 + x_2 - 2 \leq 0 \\ & x_1 + x_2 - 6 \leq 0 \\ & -x_1 - x_2 + 2 \leq 0 \end{aligned}$$

$$[0, 0, 1, 0, 1, 0] \leq \mathbf{x} \leq [6, 5, 5, 6, 5, 10]$$

Die Lösung ist $\mathbf{x}^* = [5, 1, 5, 0, 5, 10]$ mit $f(\mathbf{x}^*) = -310$.

17. Testproblem

$$\min f(\mathbf{x}) = -2x_1 + x_2 - x_3$$

$$\begin{aligned} \text{s.t. } & x_1 + x_2 + x_3 \leq 0 \\ & 3x_2 + x_3 - 6 \leq 0 \\ & -\mathbf{x}^T B^T B \mathbf{x} + 2\mathbf{r}^T B \mathbf{x} - \|\mathbf{r}\|^2 + 0.25\|\mathbf{b} - \mathbf{v}\|^2 \leq 0 \end{aligned}$$

$$[0; 0; 0] \leq \mathbf{x} \leq [3; 3; 3]$$

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ -2 & 1 & -1 \end{pmatrix}$$

$$\mathbf{b} = [3, 0, -4], \mathbf{v} = [0, -1, -6], \mathbf{r} = [1.5, -0.5, -5].$$

Die globale Lösung ist $\mathbf{x}^* = [3, 0, 1]$ mit $f(\mathbf{x}^*) = -7$.

18. Testproblem

$$\min f(x, y) = -x - y$$

$$\begin{aligned} \text{s.t. } & y - 2x^4 + 8x^3 - 8x^2 - 2 \leq 0 \\ & y - 4x^4 + 32x^3 - 88x^2 + 96x - 36 \leq 0 \end{aligned}$$

$$[0, 0] \leq [x, y] \leq [3, 4]$$

Die globale Lösung ist $[x^*, y^*] = [2.3295, 3.1783]$ mit $f([x^*, y^*]) = -5.5079$.

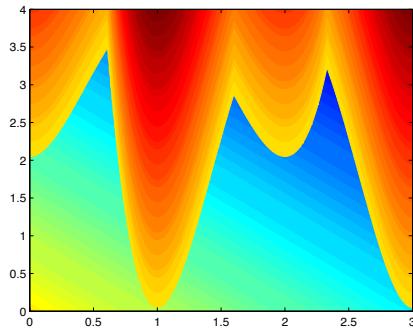


Abbildung 8.12: Testproblem 18.

19. Testproblem

$$\min f(x, y) = -12x - 7y + y^2$$

$$\text{s.t. } 2x^4 - 2 + y \leq 0$$

$$[0, 0] \leq [x, y] \leq [2, 3]$$

Die eigentliche Nebenbedingung $-2x^4 + 2 - y = 0$ wurde hier in eine Ungleichung umgewandelt. Die Lösung lautet: $[x^*, y^*] = [0.71751, 1.470]$ mit $f([x^*, y^*]) = -16.73889$.

20. Testproblem

$$\begin{aligned} \min f(x, y) &= 100((x + y - 3.5)^2 + 4(y - x + 0.5)^2)^{-1} \\ \text{s. t. } & -(1 + y - x^2) \leq 0 \\ & -(1 - y - x^2) \leq 0 \\ & -1.5 \leq (x, y) \leq 1.5. \end{aligned}$$

Das globale Minimum ist $x^* = (-1, 0)$ mit $f^* = 3.4188$.

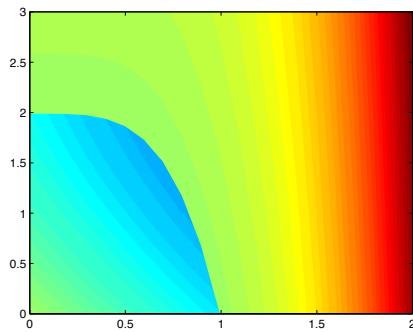


Abbildung 8.13: Testproblem 19.

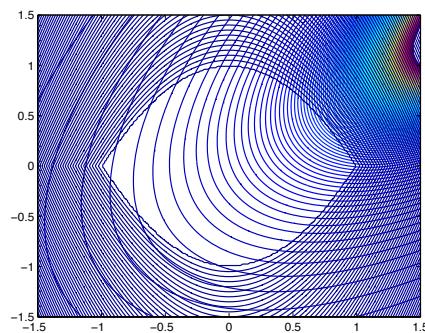


Abbildung 8.14: Testproblem 20.

8.2.2 Numerische Ergebnisse

Zunächst werden die Ergebnisse des HCP-Algorithmus, angewendet auf die Testprobleme ohne Nebenbedingungen, präsentiert:

Problem	# Pkte.	Rechenzeit	x_{opt}	f_{opt}	f^*
B	185	0.203	(3.2031; 0.4686)	0.0027	0
BR	115	0.125	(9.5513; 2.9297)	0.5843	0.3979
C	175	0.797	(0.0781;-0.7031)	-1.0305	-1.0316
G2	207	0.203	(0.0156; 0.0156)	1.85e-4	0
GP	126	0.109	(0.0000;-1.0000)	3.0000	3
H	146	0.422	(3.5938;-1.8750)	0.0136	0
M	178	0.5	(0.0100; 0.0100)	0.0085	0
R	133	0.141	(0.0000; 0.0000)	-2.0000	-2
RO	288	0.344	(0.7500; 0.5781)	0.0869	0
Hn3	161	0.453	(0.1250; 0.5630; 0.8440)	-3.8534	-3.8628
Hn6	43	0.001	(0.5000;1.0000;1.0000;... 0.5000;0.0000;0.0000)	-2.2799	-3.3937
G10	79	0.015	(0.25;0.25;1.50;...;1.50)	0.8399	0
Sch	401	0.297	(0.95;...;0.95)	6.0215	0

Tabelle 8.1: Ergebnisse des HCP-Algorithmus.

Um den HCP-Algorithmus mit dem ähnlich arbeitenden `iterative_grid`-Algorithmus verglichen zu können wurde dieser mit dem Parameter $\alpha = 0.7$, dem Noboundary-Gitter als Grundlage und max. 1000 Funktionsauswertungen auf die gleichen Testprobleme angewendet.

Problem	# Pkte.	Rechenzeit	x_{opt}	f_{opt}	f^*
B	1002	1.453	(3.1250; 0.4492)	0.0110	0
B ($\alpha = 0.5$)	1001	1.984	(3.2341;0.4681)	2.8046e-4	0
BR	1001	1.562	(3.1445; 2.2727)	0.3979	0.3979
C	1000	1.500	(0.0899;-0.7129)	-1.0316	-1.0316
G2	1000	1.515	(0.0000; 0.0000)	5.937e-11	0
GP	1000	1.390	(0.0000;-1.0000)	3	3
H	1000	1.590	(3.0000; 2.0000)	1.3315e-18	0
M	1003	1.453	(0.0100; 0.0100)	0.0085	0
R	1003	1.328	(0.0000; 0.0000)	-2	-2
RO	1001	1.360	(1.0938; 1.1958)	0.0088	0
Hn3	1000	1.578	(0.1250; 0.5625; 0.8503)	-3.8605	-3.8628
Hn6	1001	1.828	(0.1963; 0.1512; 0.5000;... 0.2813; 0.3125; 0.6563)	-3.3937	-3.3937
G10	1001	2.031	(0.0156;-0.0625;...;-0.0625)	0.0039	0
Sch	1099	6.453	(0.95;...;0.95)	6.0215	0

Tabelle 8.2: Ergebnisse von `iterative_grid`

Daneben erfolgt ein Vergleich mit Algorithmus 4.1, der Optimierung auf einem gegebenen dünnen Gitter. Der Adaptivitätsparameter ist in den meisten Fällen $\alpha = 0.7$ und das zugrundeliegende dünne Gitter ist ein Noboundary-Gitter mit Level 7.

Problem	# Pkte.	Rechenzeit	x_{opt}	f_{opt}	f^*
B	200	0.25	(2.5000; 0.3125)	0.6894	0
BR	200	0.203	(-3.1250; 12.1875)	0.4015	0.3979
C	200	0.281	(0.0000; -0.6250)	-0.9521	-1.0316
G2	200	0.266	(-0.0625; 0.2500)	0.0178	0
GP	200	0.281	(0.0000; -1.0000)	3.0000	3
H	300	0.375	(3.1250; 1.8750)	0.5396	0
M ($\alpha = 0.2$)	600	2.062	(0.0313; 0.0313)	0.0813	0
R	200	0.234	(0.0000; 0.0000)	-2.0000	-2
RO	400	0.640	(0.8906; 0.7500)	0.1987	0
Hn3	300	1.031	(0.2500; 0.5000; 0.8438)	-3.7440	-3.8628
Hn6	1200	12.047	(0.25; 0.25; 0.50; ... 0.25; 0.25; 0.50)	-2.5044	-3.3937
G10	1000	1.171	(0.25; 0.25; 0.25; 1.5; ...; 1.5)	0.0813	0

Tabelle 8.3: Ergebnisse von `adaptive_sparsegrid`

Abschließend wurde `globmin_itergrid_con`, also die Funktion `iterative_grid` zusammen mit Clusteranalyse und lokaler Optimierung, auf alle Testprobleme angewendet. Wir geben zunächst die mit `iterative_grid` erzielte Lösung an und dann die Lösung mit dem gesamten Algorithmus, sowie wie viele Punkte zusätzlich verwendet wurden.

iterative_grid

Problem	#Pkt.	iterative_grid				PBM & lokale Optimierung				Zeit
		x_{opt}	f_{opt}	#Pkt.	x_{opt}	f_{opt}	#Pkt.	x_{opt}	f_{opt}	
B	200	(3.125,0.449)	0.0110	1200	(3.223,0.4655)	1.5e-17	0			1.032
BR	200	(3.145, 2.273)	0.3979	1200	(3.142, 2.275)	0.3979	0			1.015
C	200	(-0.089;0,703)	-1.0309	1200	(-0.089,0,713)	-1.0316	-1.0316			1.032
G2	200	(-0,0;-0,2)10 ⁻⁴	5.9e-11	1200	(0,1,0,1)10 ⁻⁷	0	0			1.062
GP	200	(-0.085,-1.016)	4.908	1200	(0,-1)	3	3			0.860
H	200	(3.000,1.999)	1.3e-8	1200	(2.999,1.999)	3.2e-19	0			0.828
M	500	(0,0,0,9)10 ⁻⁶	5.5e-11	3000	(0,2,0,1)10 ⁻⁸	0	0			2.297
R	500	(-0,0,-0,8)10 ⁻⁵	-1.999	3000	(0,4,0,9)10 ⁻⁹	-2	-2			1.797
RO	200	(0.500,0.249)	0.250	1200	(1.000,1.000)	4.3e-19	0			1.641
Hn3	300	(0.125,0.563,0.850)	-3.860	4200	(0.115,0.555,0.852)	-3.8628	-3.8628			3.532
Hn6	600	(0.196,0.151,0.499,0.281,0.312,0.656)	-3.3937	4200	(0.197,0.151,0.486,0.279,0.313,0.655)	-3.3960	-3.3960			3.047
G10	1000	(0,0,0,0,0,0,0,0,0,0)	0		keine Verbesserung	0	0			1.000
12	500	(0.999,0.999,0.000,0.999,0.000)	-16.499		keine Verbesserung	-17	-17			11.359
13	1000	(0.500,0.500,0.500,0.900,10.000)	-114.999	9000	(0.000,0.992,0.000,0.991,0.989,19.999)	-212.847	-213			10.343
14	1000	(0.968,0.968,0.968,0.031,0.875,0.875,...)	-19.051		keine Verbesserung	-21	2.813			
15	2000	0.875,0.875,0.875,4.843,4.843,4.843,0.750 (0.093,3.000,2.625,1.500,3.000,2.250)	-20.395	12000	(0.000,5.956,5.972,4.803,5.999,0.005) (4.999,1.000,4.964,0.380,4.877,1.413)	-44.317	-44.4			15.391
16	1000	(4.500,1.250,4.999,0.000,4.999,9.999)	-240.779	8000	(3.000,0.000,1.000)	-276.403	-276.403			8.891
17	300	(1.500,0,463,0,750)	-3.286	1800	(2.329,3.178)	-7.000	-7.000			2.047
18	200	(2.437,2.615)	-5.052	1600	(0.717,1.470)	-5.508	-5.508			1.828
19	200	(0.707,1.500)	-16.735	1600	(-0.999,0.000)	-16.738	-16.738			1.797
20	200	(-0.999,0.000)	3.418		keine Verbesserung	3.418	3.418			4.593

Anhang A

Abkürzungs- und Symbolverzeichniss

A.1 Allgemein gebräuchliche Abkürzungen

\mathbb{N}	Menge der natürlichen Zahlen ohne Null.
\mathbb{N}_0	Menge der natürlichen Zahlen mit Null.
\mathbb{Z}	Menge der ganzen Zahlen.
\mathbb{R}	Menge der reellen Zahlen.
\mathbb{R}_+	Menge der positiven reellen Zahlen.
\mathbb{R}^d	d -dimensionaler reeller Raum.
$\mathcal{C}^2(D, W)$	Menge aller zweimal stetig differenzierbaren Funktionen $f : D \longrightarrow W$.
$\ \cdot \ _1$	1-Norm, bzw. Betragssummennorm: $\ x\ _1 = \sum_{i=1}^d x_i $ für $x \in \mathbb{R}^d$.
$\ \cdot \ _2$	2-Norm, bzw. Euklid'sche Norm: $\ x\ _2 = \sqrt{\sum_{i=1}^d x_i^2}$ für $x \in \mathbb{R}^d$.
$\ \cdot \ _\infty$	∞ -Norm, bzw. Maximumnorm: $\ x\ _\infty = \max_{i \in \{1, \dots, d\}} (x_i)$ für $x \in \mathbb{R}^d$.
$dist(x, y)$	Abstand von x zu y für $x, y \in \mathbb{R}^d$ bzgl. einer Norm.
$x \times y$	Kartesisches Produkt der Vektoren x und y z. B. für $x, y \in \mathbb{R}^d$: $x \times y = \{(x_i, y_j) \in \mathbb{R}^2 \mid i \in \{1, \dots, d\}, j \in \{1, \dots, d\}\}$.
$U \otimes V$	Tensorprodukt zweier Vektorräume U und V .

A.2 In dieser Diplomarbeit verwendete Abkürzungen

f	Bezeichnet die zu minimierende Zielfunktion $f : S = [a, b] \subset \mathbb{R}^d \rightarrow \mathbb{R}$; d ist die Dimension des Problems.
g	Bezeichnet die nichtlinearen Ungleichungsnebenbedingungen: $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$.
x^*, f^*	f und g werden als stetig vorausgesetzt. Die Minimallösung x^* eines Optimierungsproblems mit Funktionswert $f^* = f(x^*)$.
$T_{[a,b] \rightarrow [u,v]}$	Affine Transformation, die den Quader $[a, b]$ auf den Quader $[u, v]$ abbildet: $T_{[a,b] \rightarrow [u,v]}(x) = \frac{v-u}{b-a}x + u - \frac{v-u}{b-a}a$.
X	Die Menge $X = \{x_1, \dots, x_n\}$ besteht aus allen, im Laufe eines Algorithmus, betrachteten Punkten x_i mit $x_i \in \mathbb{R}^d$.
F	Die Menge $F = \{f(x_1), \dots, (x_n)\}$ besteht aus den Zielfunktionswerten an den Punkten in X .
α	Die Variable $\alpha \in [0, 1]$ steuert den Grad der Adaptivität eines Verfahrens. $\alpha = 0$ bedeutet, dass das Verfahren rein adaptiv arbeitet.
β	$\beta : \mathbb{R}^d \rightarrow \mathbb{R}$ ist eine Bewertungsfunktion.
\mathcal{Q}	Der Raum der stückweise d -linearen Funktionen.
\mathcal{T}	Ein Teilraum von \mathcal{Q} .
ϕ	Hutfunktion $\phi : \mathbb{R} \rightarrow [0, 1]$.
$\phi_{i,l}$	Basisfunktionen von \mathcal{Q} und \mathcal{T} .
G_L	$G_L = G_{l_1, \dots, l_d}$ ein rechtwinkliges Gitter mit Schrittweite $h_j = 2^{-l_j}$ in x_j - Richtung.
BG_l^d	d -dimensionales Boundary-Gitter mit Level l .
M_l^d	d -dimensionales Maximumgitter mit Level l .
NB_l^d	d -dimensionales Noboundary-Gitter mit Level l .
CC_l^d	d -dimensionales Clenshaw-Curtis-Gitter mit Level l .
GL_l^d	d -dimensionales Gauß-Legendre-Gitter mit Level l .
TG_l^d	d -dimensionales Tschebyscheff-Gitter mit Level l .
$cubGL_l^d$	d -dimensionales kubisches Gauß-Legendre-Gitter mit Level l .
f_n^s	Die Lösung eines Problems auf einem dünnen Gitter mit Level n .
f_n^c	Die Lösung eines Problems mit der Kombinationsmethode.
C	Die Menge der Clustermittelpunkte, bzw. -centroide in einer Menge X .
L	$L : \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}$ Lagrangefunktion.
L_A	Erweiterte Lagrangefunktion.
P_ρ	Penaltyfunktion, ρ ist der Penaltyparameter und Pen der Penaltyterm.
B_μ	Barrierefunktion, μ ist der Barriereparameter und Bar der Barriereterm.
φ	Ist eine Mischform des Penalty- und Barriereterms in der PBM-Methode.
u, p	Parameter innerhalb der PBM-Methode, u steht für die Lagrange-Multiplikatoren, p steht für die Strafparameter.

Abbildungsverzeichnis

2.1	HCP im zweidimensionalen Fall.	5
2.2	HCP im dreidimensionalen Fall.	5
2.3	Beispiele für Nachbarn im \mathbb{R}^2 .	7
2.4	Algorithmus 2.10 für Testproblem BR.	8
2.5	Algorithmus 2.10 für Testproblem Hn3.	9
2.6	Vergleich der beiden Bewertungsfunktionen β und η .	10
2.7	Volles Gitter vs. HCP-Algorithmus.	11
2.8	Vgl. volles Gitter mit HCP-Algorithmus.	11
2.9	Auswirkung von ε .	12
2.10	Das <i>Pattern</i> der Punkte $x + y$ für $k = 1, \dots, 4$ im \mathbb{R}^2 .	15
2.11	Algorithmus 2.11 für unbekannte Lipschitzkonstante.	15
3.1	Eindimensionales äquidistantes Gitter.	18
3.2	Nodale Basis von \mathcal{Q}_2 .	20
3.3	Hierarchische Basis von \mathcal{Q}_2 .	22
3.4	Vergleich von nodaler und hierarchischer Basis.	22
3.5	Beispiel für eine Basisfunktion für $d = 2$.	24
3.6	Basisfunktionen der Teilräume $\mathcal{T}_{i,j}$.	25
3.7	Träger der Basisfunktionen von $\mathcal{T}_{i,j}$.	26
3.8	Zwei- und dreidimensionales Boundary-Gitter.	27
3.9	Kombinationsmethode	28
3.10	Eindimensionales Maximumgitter.	31
3.11	Zwei- und dreidimensionale Maximumgitter.	32
3.12	Eindimensionales Noboundary-Gitter.	33
3.13	Zwei- und dreidimensionale Noboundary-Gitter.	34
3.14	Eindimensionales Clenshaw-Curtis-Gitter.	35
3.15	Zwei- und dreidimensionale Clenshaw-Curtis-Gitter.	36
3.16	Eindimensionales Gauß-Legendre-Gitter.	38
3.17	Zwei- und dreidimensionale Gauß-Legendre-Gitter.	39
3.18	Eindimensionales Tschebyscheff-Gitter.	40
3.19	Zwei- und dreidimensionale Tschebyscheff-Gitter.	41
3.20	Eindimensionales kubischen Gauß-Legendre-Gitter.	42
3.21	Zwei- und dreidimensionale kubische Gauß-Legendre-Gitter.	44
4.1	<code>adaptive_sparsegrid</code> I.	55
4.2	<code>adaptive_sparsegrid</code> II.	56
4.3	<code>adaptive_sparsegrid</code> III.	56
4.4	Baum für das eindimensionale Noboundary-Gitter.	59
4.5	Baum für das eindimensionale Maximum-Gitter.	59

4.6	Baum für das eindimensionale Clenshaw-Curtis-Gitter.	60
4.7	Das eindimensionale Legendre-Gitter.	61
4.8	Binärbaum für das zweidimensionale Maximum-Gitter.	62
4.9	Zweidimensionales Maximumgitter mit Level 4.	62
4.10	Gitterpunkte und ihre Positionen im Binärbaum.	64
4.11	Nachbarn im kub. Gauß-Legendre-Gitter.	65
4.12	Nachbarn der Stufe M.	68
4.13	Nachbarn der Stufe M.	68
4.14	<code>iterative_sparsegrid</code> für das Noboundary-Gitter.	70
4.15	<code>iterative_sparsegrid</code> für $d = 2$.	71
4.16	<code>iterative_sparsegrid</code> für $d = 3$.	72
5.1	FCM-Algorithmus angewendet auf 1000 Datenpunkte.	82
5.2	Vergleich von <code>cluster_estimation</code> und <code>subclust I</code>	83
5.3	Vergleich von <code>cluster_estimation</code> und <code>subclust II</code>	84
6.1	Reflektion und Expansion.	88
6.2	Äußere und innere Kontraktion.	88
6.3	Schrumpfung.	89
6.4	Folge von inneren Kontraktionsschritten.	91
6.5	$\ x_{opt}\ _2$ in Abhängigkeit von d .	92
7.1	Schlupfvariable z_i^* .	101
7.2	Beispiel für φ .	104
7.3	Erweiterte Lagrangefunktion für Testbeispiel 18.	106
7.4	Erweiterte Lagrangefunktion für Testbeispiel 19.	107
7.5	Erweiterte Lagrangefunktion für Testbeispiel 20.	108
8.1	Schematischer Aufbau von <code>globmin_itergrid_con</code> .	112
8.2	Höhenlinien der Testfunktion f_B .	115
8.3	Höhenlinien der Testfunktion f_{BR} .	115
8.4	Höhenlinien der Testfunktion f_C .	116
8.5	Höhenlinien der Testfunktion f_{G2} .	116
8.6	Höhenlinien der Testfunktion f_{GP} .	117
8.7	Höhenlinien der Testfunktion f_H .	117
8.8	Die Funktion f_{Hn3} .	118
8.9	Höhenlinien der Testfunktion f_M .	118
8.10	Höhenlinien der Testfunktion f_R .	119
8.11	Höhenlinien der Testfunktion f_{RO} .	119
8.12	Testproblem 18.	122
8.13	Testproblem 19.	123
8.14	Testproblem 20.	123

Literaturverzeichnis

- [Ab01] L. Abbe.
A logarithmic barrier approach and its regularization applied to convex semi-infinite programming problems.
Dissertation, Universität Trier, 2001.
- [AM96] A.J. Abrantes und J.S. Marques.
A class of constrained clustering algorithms for object boundary extraction.
IEEE Trans. Image Processing, 5:1507-1521, 1996.
- [Ac03] S. Achatz.
Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten.
Dissertation, Technische Universität München, Institut für Informatik, 2003.
- [BAF05] B. Balasko, J. Abonyi und B. Feil.
Fuzzy Clustering and Data Analysis Toolbox.
Department of Process Engineering, University of Veszprem, 2005.
www.fmt.vein.hu/softcomp/fclusttoolbox/
- [BZ95] A. Ben-Tal und M. Zibulevsky.
Penalty/Barrier Multiplier Methods for convex Programming Problems.
Faculty of Industrial Engineering and Management Technion - Israel Institute of Technology, Haifa, 1995.
- [Be73] J.C. Bezdek.
Fuzzy mathematics in pattern classification.
Dissertation, Cornell University Ithaca, NY, 1973.
- [Be74] J.C. Bezdek.
Cluster Validity with fuzzy sets.
J. Cybernetics, 3(3):58-73, 1974.
- [Be80] J.C. Bezdek.
A convergence theorem for the fuzzy ISODATA clustering algorithm.
IEEE Trans. Pattern Anal. Machine Intell., 2:1-8, 1980.
- [Bu92] H.-J. Bungartz.
Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung.

- Dissertation, Technische Universität München, Institut für Informatik, 1992.
- [Bu98] H.-J. Bungartz.
Finite Elements of Higher Order on Sparse Grids.
Habilitationsschrift, Technische Universität München, Institut für Informatik, 1998.
- [BG04] H.-J. Bungartz und M. Griebel.
Sparse Grids,
Acta Numerica, 13:1-123, 2004.
- [Br01] I.N. Bronstein, K.A. Semendjajew, G. Musiol, H. Mühlig.
Taschenbuch der Mathematik.
Verlag Harri Deutsch, 2001.
- [Ch68] T.S. Chihara.
Orthogonal polynomials whose zeros are dense in intervals.
J. Math. Anal. Appl., 24:362-371, 1968.
- [Ch94] S.L. Chiu.
Fuzzy Model Identification Based on Cluster Estimation.
Journal of intelligent and fuzzy systems, 2:267-278, 1994.
- [CNR99] R. Cools, E. Novak und K. Ritter.
Smolyak's construction of cubature formulas of arbitrary trigonometric degree.
Computing, 62:147 - 162, 1999.
- [Di00] S. Dirnstorfer.
Numerical quadrature on sparse grids.
Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2000.
- [FM68] A.V. Fiacco und G.P. McCormick.
Nonlinear Programming Sequential Unconstrained Minimization Techniques.
John Wiley and Sons Inc., New York, 1968.
- [Fi00] G. Fischer.
Lineare Algebra.
Vieweg, 2000.
- [FP90] C.A. Floudas und P.M. Pardalos.
A Collection of Testproblems for Constrained Global Optimization Algorithms.
Lecture Notes in Computer Science, No 455, Springer, 1990.
- [Ga04] J. Garcke.
Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern.
Dissertation, Rheinische Friedrichs-Wilhelms-Universität, Bonn, 2004.

- [GG01] J. Garcke und M. Griebel.
Data Mining With Sparse Grids Using Simplicial Basis Functions.
 Institut für Angewandte Mathematik, Rheinische Friedrichs-Wilhelms-Universität, Bonn, 2001.
- [GSZ90] M. Griebel, M. Schneider und C. Zenger.
A Combination Technique For The Solution Of Sparse Grid Problems.
 Technischer Bericht, SFB-Bericht 342/19/90, Technische Universität München, Institut für Informatik, 1990.
- [He03] M. Hegland.
Adaptive Sparse Grids.
 Anziam Journal 44(E):C335-C353, 2003.
- [Hi05] M. Hirmer.
Style Classification of Hedge-Funds by Cluster Analysis.
 Diplomarbeit, Technische Universität München, Zentrum Mathematik, 2005.
- [HK03] F. Höppner und F. Klawonn.
A Contribution To Convergence Theory Of Fuzzy-c-Means and Derivatives.
 IEEE Transactions on Fuzzy Systems, 11(5):682-694, 2003.
- [Ho05] M. Horn.
Optimal algorithms for global optimization in case of unknown Lipschitz constant.
 Mathematisches Institut, Universität Jena, 2005.
- [Ho90] R. Horst und H. Tuy.
Global Optimization.
 Springer, Berlin, 1990.
- [KK98] F. Klawonn und A. Keller
Fuzzy Clustering with Evolutionary Algorithms.
 Universität Braunschweig, Fakultät für Informatik, 1997.
- [KK97] F. Klawonn und R. Kruse
Constructing a Fuzzy Controller from Data.
 Fachhochschule Ostfriesland, Emden und Deutsches Zentrum für Luft- und Raumfahrt, Braunschweig, 1998.
- [Kl03] A. Klimke.
Piecewise Multilinear Sparse Grid Interpolation in MATLAB.
 Technischer Report, Universität Stuttgart, Institut für Angewandte Analysis und Numerische Simulation, 2003.
- [KS03] M. Kocvara und M. Stingl.
PENNON A Code for Convex Nonlinear and Semidefinite Programming.
 Optimization Methods and Software, 18(3):317-333, 2003.

- [KS05] M. Kocvara und M. Stingl.
On the solution of large-scale SDP problems by the modified barrier method using iterative solvers.
 Research Report 304, Institute of Applied Mathematics, University of Erlangen, 2005.
- [LRWW98] J.C. Lagarias, J.A. Reeds, M.H. Wright und P.E. Wright.
Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions.
 SIAM J. Optimization, 9(1):112 - 147, 1998.
- [McK98] K. I. M. McKinnon.
Convergence of the Nelder-Mead Simplex Method to a Nonstationary Point.
 SIAM J. Optimization, 9(1):148 - 158, 1998.
- [Me93] F. Mengedoth.
Visualisierung der Kombinationsmethode.
 Diplomarbeit, Technische Universität München, Institut für Informatik, 1993.
- [NM65] J.A. Nelder und R. Mead.
A Simplex Method for Function Minimization.
 Computer Journal, 7(4):308-313, 1965.
- [NR96a] E. Novak und K. Ritter.
Global Optimization Using Hyperbolic Cross Points.
 in *State of the Art in global Optimization*, C.A. Floudas, P.M. Pardalos (eds.), pp. 19-33.
 Kluwer Academic Publishers, Dordrecht, 1996.
- [NR96b] E. Novak und K. Ritter.
High dimensional integration of smooth functions over cubes.
 Numer. Math., 75:79-97, 1996.
- [NR97] E. Novak und K. Ritter.
The Curse of Dimension and an Universal Method for Numerical Integration.
 in *Multivariate Approximation and Splines*, G.Nürnberger, J.W. Schmidt, G. Walz (eds.), pp. 177 - 188, 1997.
- [Po92] R. Polyak.
Modified Barrier Functions: Theory and Methods.
 Math. Programming, 54:177-222, 1992.
- [Ro93] R.T. Rockafellar.
Lagrange Multipliers and Optimality.
 Siam Review, 35(2):183 - 238, 1993.
- [Sm63] S. Smolyak.
Quadrature and interpolation formulas for tensor products of certain classes of functions.
 Soviet Mathematics, Doklady, 4:240 - 243, 1963.

- [Sp93] P. Spellucci.
Numerische Verfahren der nichtlinearen Optimierung.
Birkhäuser, Basel, 1993.
- [Ti03] G. Tinhofer.
Nichtlineare Optimierung.
Skriptum zur Vorlesung, Technische Universität München, 2003.
- [TZ89] A. Törn und A. Zilinskas.
Global Optimization.
Lecture Notes in Computer Science, 350, Springer, 1989.
- [Ul04] S. Ulbrich
Nichtlineare Optimierung.
Skriptum zur Vorlesung, Technische Universität München, 2004.
- [Ze90] C. Zenger.
Sparse Grids.
Technischer Bericht, SFB-Bericht 342/18/90, Technische Universität München, Institut für Informatik, 1990.
- [Zi96] M. Zibulevsky.
Penalty/Barrier Multiplier Methods for Large-Scale Nonlinear and Semidefinite Programming.
Phd Thesis, Technion - Israel Institute of Technology, 1996.
- [Zik04] D. Zikic.
Mehrdimensionale Numerische Integration mittels hierarchischer Basen.
Präsentation JobSis 2004, Technische Universität München, Institut für Informatik, 2004.