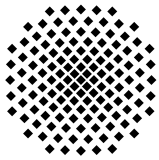
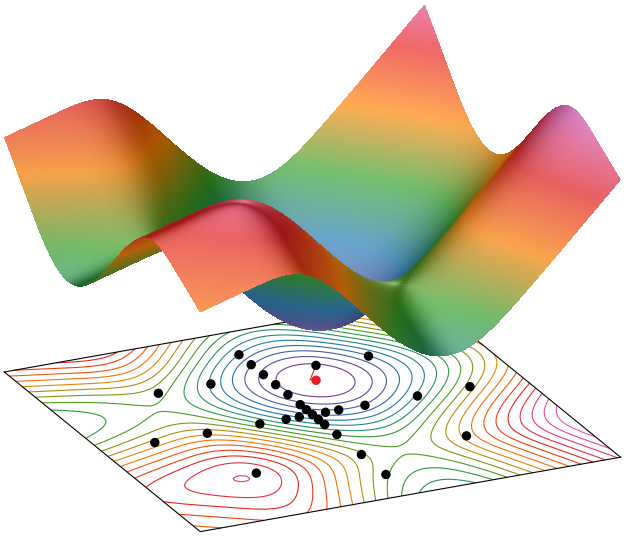
Institut für Parallele und Verteilte Systeme
Abteilung Simulation großer Systeme
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart



Masterarbeit Nr. 3629

Hierarchische Optimierung mit Gradientenverfahren auf Dünngitterfunktionen

Julian Valentin



Studiengang:	Mathematik
Prüfer:	Jun.-Prof. Dr. rer. nat. Dirk Pflüger
Betreuer:	Jun.-Prof. Dr. rer. nat. Dirk Pflüger
begonnen am:	17. März 2014
beendet am:	12. September 2014
CR-Klassifikation:	G.1.6

Lass bloß den Kopf nicht hängen! Ich wette, Einstein hat sich auch mehrmals verfärbt, bevor er dann seine Glühbirne erfunden hat.

(Homer zu Bart Simpson, nachdem sein schulisches Chemieexperiment explodiert ist und ihn grün gefärbt hat)

Vorwort

Zunächst einmal möchte ich bei meinem Prüfer und Betreuer Juniorprofessor Dr. Dirk Pflüger bedanken, der das interessante Thema bereitgestellt und mich in den vergangenen sechs Monaten ausgezeichnet betreut hat. Seine Vorlesung über Modellbildung und Simulation war sehr inspirierend für mich (zum Beispiel für die Anwendungsbeispiele in Kapitel 7). Auch bedanken möchte ich mich bei Professor Dr. Klaus Höllig, der schnell und bereitwillig die Zweitprüferschaft übernommen hat.

Des Weiteren will ich Daniel Hübner M.Sc. meinen Dank dafür aussprechen, dass er mir bei auftauchenden Problemen mit der Software CFS++ stets mit Rat und Tat zur Seite stand. Bei meinem Kommilitonen Florian Martin M.Sc. bedanke ich mich für die schöne gemeinsame Studienzeit der letzten fünf Jahre.

Von unschätzbarer Bedeutung für diese Arbeit ist jedoch die Unterstützung durch meine Familie gewesen. Sie gab mir die Kraft, diese Aufgabe erfolgreich meistern zu können.

Bietigheim-Bissingen, im September 2014
Julian Valentin

Inhaltsverzeichnis

Notationsverzeichnis

In dieser Arbeit wird als Dezimaltrennzeichen das Komma verwendet. Weil bei reellen Intervallen wie $[a, b]$ (und bei Mengen) normalerweise ebenfalls Kommas eingesetzt werden, wird ein Semikolon benutzt, wenn eine der Zahlen a, b bereits ein Komma enthält, also z. B. $[0, 1/4] = [0; 0{,}25]$.

Algorithmen werden in pythonähnlichem Pseudocode angegeben, bei dem Einrückungen anzeigen, zu welcher Kontrollstruktur die Codezeilen gehören. Um Platz zu sparen, wird bei Kontrollstrukturen (z. B. **`for`**-Schleifen), die nur eine Codezeile enthalten, die Kontrollstruktur mit der Codezeile kompakt in einer einzelnen Zeile wiedergegeben. Zuweisungen werden durch „$\leftarrow$“ symbolisiert.

In der folgenden Tabelle werden verschiedene in dieser Arbeit verwendete Symbole und Schreibweisen erklärt.

Notation	**Bedeutung**
$\mathbb{N}$, $\mathbb{N}_0$	$\{1, 2, 3, \dots\}$, $\{0, 1, 2, 3, \dots\}$
$A \subset B$	$A \subseteq B$ für Mengen A, B
$A \dot\cup B$	$A \cup B$ für disjunkte Mengen A, B
$V \leqslant W$	V ist Untervektorraum des Vektorraums W
$\bigtimes_{i=1}^{n} A_i$	kartesisches Produkt $A_1 \times \cdots \times A_n$ der Mengen $A_1, \dots, A_n$
$\bigoplus_{i=1}^{n} V_i$	direkte Summe $V_1 \oplus \cdots \oplus V_n$ der Vektorräume $V_1, \dots, V_n$
$\delta_{i,j}$	Kronecker-Delta für $i, j \in \mathbb{Z}$
$\operatorname{span}\{\vec v_1, \dots, \vec v_n\}$	linearer Aufspann von Vektoren $\vec v_1, \dots, \vec v_n \in V$
$c \cdot [a \pm b]$	$[c(a-b), c(a+b)]$ für $a, b, c \in \mathbb{R}$
$\lfloor x \rceil, \lceil x \rceil$	Gaußklammern für $x \in \mathbb{R}$
e^x	$\exp(x)$ für $x \in \mathbb{R}$
$\vec e$, $\vec e_i$	$(1, \dots, 1) \in \mathbb{R}^d$, $(0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^d$
$v_{\pi(1)} \leqslant \cdots \leqslant v_{\pi(n)}$	Sortierung von $\vec v \in \mathbb{R}^n$ mit Permutation π von $\{1, \dots, n\}$
$\lVert\vec v\rVert_1$, $\lVert\vec v\rVert_2$, $\lVert\vec v\rVert_\infty$	$\sum_{t=1}^{d} \lvert v_t\rvert$, $(\sum_{t=1}^{d} \lvert v_t\rvert^2)^{1/2}$, $\max_{t=1,\dots,d} \lvert v_t\rvert$ für $\vec v \in \mathbb{R}^d$
$\lVert f\rVert_{L^2}$	L^2-Norm von $f\colon [0,1]^d \to \mathbb{R}$
$\mathrm{D}^{\vec k}$	$\frac{\partial^{\lVert\vec k\rVert_1}}{\partial x_1^{k_1} \cdots \partial x_d^{k_d}}$ für $\vec k \in \mathbb{N}_0^d$
$\lvert f\rvert_{H^2_{0,\mathrm{mix}}}$	$\lVert \mathrm{D}^{2\vec e} f\rVert_{L^2}$ für $f\colon [0,1]^d \to \mathbb{R}$
$\mathcal{O}(f)$, $\Theta(f)$	Landau-Symbole für $f\colon \mathbb{N} \to \mathbb{R}$
$\operatorname{supp} f$	Träger von $f\colon [0,1]^d \to \mathbb{R}$
$\nabla f(\vec x)$, $H_f(\vec x)$	Gradient, Hesse-Matrix von $f\colon [0,1]^d \to \mathbb{R}$ in $\vec x \in [0,1]^d$
$\lambda_{\min}(A), \lambda_{\max}(A)$	kleinster/größter Eigenwert von $A \in \mathbb{R}^{n\times n}$ symm. pos. def.
$\kappa(A)$	Konditionszahl $\lambda_{\max}(A)/\lambda_{\min}(A)$ von $A \in \mathbb{R}^{n\times n}$ symm. pos. def.

1 Einleitung

Optimierung ist ein wichtiges Teilgebiet der numerischen Mathematik, das viele Anwendungen in Forschung und Industrie findet. Im Kern ist hierbei eine Zielfunktion in Abhängigkeit von mehreren unbekannten Variablen gegeben, wobei es das Ziel ist, die Variablen so zu wählen, dass die Zielfunktion möglichst klein (oder groß) wird. Weil wir an globalen Optima interessiert sind, spricht man auch von „globaler Optimierung“.

Anwendungen von globaler Optimierung

Eine Anwendung von Optimierung ist die Aufstellung von mathematischen Modellen für komplexe Vorgänge (Allgemeines zu Modellbildung siehe auch [7]). Ein Beispiel sind in [14] beschriebene biochemische Prozesse in biologischen Zellen. Mathematische Modelle für solche Prozesse hängen oft von einer großen Zahl (oft Dutzende oder Hunderte) von Parametern ab, deren Wahl das Verhalten des Modells in der Simulation stark beeinflusst. Die allermeisten Parameter können nicht experimentell bestimmt werden, da sich die Messung beispielsweise von Diffusionskonstanten als schwierig gestaltet. Erschwerend hinzu kommt noch, dass die Parameter oft sehr unterschiedliche Empfindlichkeiten haben und miteinander korrelieren. Globale Optimierung kann hier wie in [14] eine Möglichkeit sein, die Parameter so festzulegen, dass das Modell in der Simulation mit den experimentell ermittelten Daten übereinstimmt.

Eine andere Anwendungsmöglichkeit von globaler Optimierung befindet sich in der Modellierung physikalischer Vorgänge durch Finite Elemente. In [10] wird als Beispiel die Konzeption eines Raketentriebwerks genannt, das im Betrieb extremen Belastungen durch Druck und Temperatur ausgesetzt wird. Mit Finiten Elementen können diese Belastungen am Computer simuliert werden. Ziel ist es, die Triebwerksdüse so zu gestalten, dass sie den auftretenden Kräften standhält, aber gleichzeitig den Entwurfsvorgaben entspricht (etwa hinsichtlich des Raketengewichts und der -größe). Für diesen Zweck können geeignete Parameter wie Form und Dicke der Düse mithilfe von globaler Optimierung gefunden werden. Man spricht auch von einem „inversen Problem“.

Außer den beiden genannten gibt es natürlich noch unzählige weitere Anwendungen von globaler Optimierung. Sie kann zum Beispiel für Problemstellungen aus der Informatik, wie Verkehrssimulationen und Parameteroptimierung für Algorithmen (Beispiele siehe [29]), oder auch in der Medizin verwendet werden für Kernspinresonanz-Tomografie oder Gentherapie (Details siehe [10]). Ein paar einfache Beispiele zur Optimierung finden sich in dieser Arbeit in Abschnitt 7.2. Darunter sind unter anderem die Bestimmung der Länge einer Stimmgabel, damit diese einen bestimmten Ton erklingen lässt, und die Ermittlung von Parametern eines Motors, sodass das simulierte Verhalten so gut wie möglich mit beobachteten Werten übereinstimmt.

Optimierungsproblem

In dieser Arbeit befassen wir uns mit Optimierungsproblemen der folgenden Art. Es handelt sich dabei zwar stets um Minimierungsprobleme, allerdings können natürlich auch Maximierungsprobleme durch Umkehrung des Vorzeichens behandelt werden.

Definition 1.1 (allgemeines Optimierungsproblem)**.** Sei $f\colon [0,1]^d \to \mathbb{R}$ eine stetige Funktion („Zielfunktion"). Dann heißt die Aufgabe, $\vec{x}_{\text{opt}} = \arg\min_{\vec{x}\in[0,1]^d} f(\vec{x})$ zu finden, „allgemeines Optimierungsproblem". Es ist also $\vec{x}_{\text{opt}} \in [0,1]^d$ gesucht mit

$$f(\vec{x}_{\text{opt}}) = \min_{\vec{x}\in[0,1]^d} f(\vec{x}). \tag{1.1}$$

Zu unterscheiden vom Problem in Definition 1.1 ist die Suche nach lokalen Minima, bei denen der Funktionswert an der Minimalstelle nur minimal in einer Umgebung der Stelle ist. Gibt ein Algorithmus zur Lösung von (1.1) nur ein lokales, aber nicht globales Minimum zurück, dann sollte der gefundene Funktionswert dem minimalen Funktionswert idealerweise wenigstens recht nahe kommen. Eine noch allgemeinere Problemstellung ist die Optimierung mit m Nebenbedingungen. In diesem Fall wird das Minimum nicht im ganzen Gebiet gesucht, sondern nur an den Stellen $\vec{x} \in [0,1]^d$ mit $g_k(\vec{x}) \leqslant 0$ für $k = 1, \ldots, m$ und eine vektorwertige Funktion $\vec{g}\colon [0,1]^d \to \mathbb{R}^m$ mit $\vec{g}(\vec{x}) = (g_1(\vec{x}), \ldots, g_m(\vec{x}))$. Dieses Problem wurde von Ferenczi mit dünnen Gittern in [16] untersucht.

Eine Vereinfachung, die in Definition 1.1 bereits vorgenommen wurde, ist neben der Stetigkeit von f (zur Sicherung der Existenz eines Minimums) die Beschränktheit des Definitionsbereichs. Es wird also angenommen, dass für alle Parameter x_t sinnvolle untere und obere Schranken $x_t^{(\text{u})}, x_t^{(\text{o})} \in \mathbb{R}$ a priori bekannt sind, sodass $\vec{x}_{\text{opt}} \in \times_{t=1}^{d} [x_t^{(\text{u})}, x_t^{(\text{o})}]$. Mit einer simplen affinen Transformation kann man dann den Parameterbereich $\times_{t=1}^{d} [x_t^{(\text{u})}, x_t^{(\text{o})}]$ auf den in Definition 1.1 erforderlichen Einheitshyperwürfel $[0,1]^d$ transformieren und umgekehrt.

Aufgrund der Kompaktheit von $[0,1]^d$ existiert nach dem Extremwertsatz von Weierstraß die Stelle $\vec{x}_{\text{opt}} \in [0,1]^d$, die Suche nach ihr gestaltet sich jedoch alles andere als trivial. Naiverweise könnte man zum Beispiel f an allen Stellen aus dem „vollen Gitter" $\{0, h, 2h, \ldots, 1\}^d$ mit Schrittweite $h = 2^{-n}$ für ein $n \in \mathbb{N}$ auswerten und dann die Stelle mit dem kleinsten Funktionswert als $\vec{x}_{\text{opt}}$ wählen. Dies wären aber $(2^n + 1)^d$ Auswertungsstellen und würde bedeuten, dass man für eine Schrittweite von $h = 1/32$ pro Dimension bei $d = 10$ Dimensionen schon 33^{10} und damit über 1 Billiarde Funktionsauswertungen vornehmen müsste („Fluch der Dimensionalität"). Wir gehen jedoch davon aus, dass schon eine einzelne Auswertung von f „teuer" ist, also eine längere Zeit beanspruchen kann, und wollen daher eine möglichst hohe Genauigkeit bei möglichst wenig Auswertungen.

Dünne Gitter

Sogenannte „dünne Gitter" können helfen, dieses Problem zu lösen (siehe [6]). Anstatt f auf dem vollen Gitter auszuwerten, verwenden wir dünne Gitter, die gerade bei einer größeren Anzahl an Dimensionen erheblich weniger Punkte enthalten. Genauer enthält

ein reguläres dünnes Gitter nach [6] $\mathcal{O}(2^n n^{d-1})$ innere Punkte (bei „feinster“ Schrittweite $h = 2^{-n}$) und damit deutlich weniger als die $\mathcal{O}(2^{nd})$ inneren Punkte des vollen Gitters mit Schrittweite $h = 2^{-n}$. Gleichzeitig lässt sich zeigen (siehe [6]), dass sich der L^2-Fehler bei Interpolation auf dem dünnen Gitter mit $\mathcal{O}(2^{-2n} n^{d-1})$ nur um einen in n polynomiellen Faktor gegenüber dem L^2-Fehler $\mathcal{O}(2^{-2n})$ beim vollen Gitter verschlechtert. Es sind also keine zu großen Abstriche bei der Genauigkeit nötig.

Zusätzlich können dünne Gitter adaptiv erzeugt werden. Damit kann man z. B. um Stellen mit einem kleinen Funktionswert von f mehr Punkte erzeugen und so eine höhere Genauigkeit des Ergebnisses erzwingen.

Außerdem ist die Verwendung dünner Gitter sowohl speicher- als auch zeitsparend möglich. Mit der in C++ geschriebenen Dünngitter-Toolbox SG^{++} existiert bereits eine effiziente Software für dünne Gitter, die die für die dünnen Gitter nötigen Datenstrukturen und Algorithmen enthält.

Globale Optimierung mit Dünngitterfunktionen

Unter der Verwendung dünner Gitter lässt sich ein Algorithmus für die Lösung des Problems in Definition 1.1 herleiten. Eine Illustration der folgenden Schritte für eine beispielhafte Zielfunktion und ein recht grobes Gitter findet sich in Abb. 1.1. Dabei sollte man beachten, dass in der oberen Bildzeile von Abb. 1.1 die Konturen der Zielfunktion f und in der unteren Bildzeile die Konturen einer Interpolierenden $\widetilde{f}$ dargestellt sind.

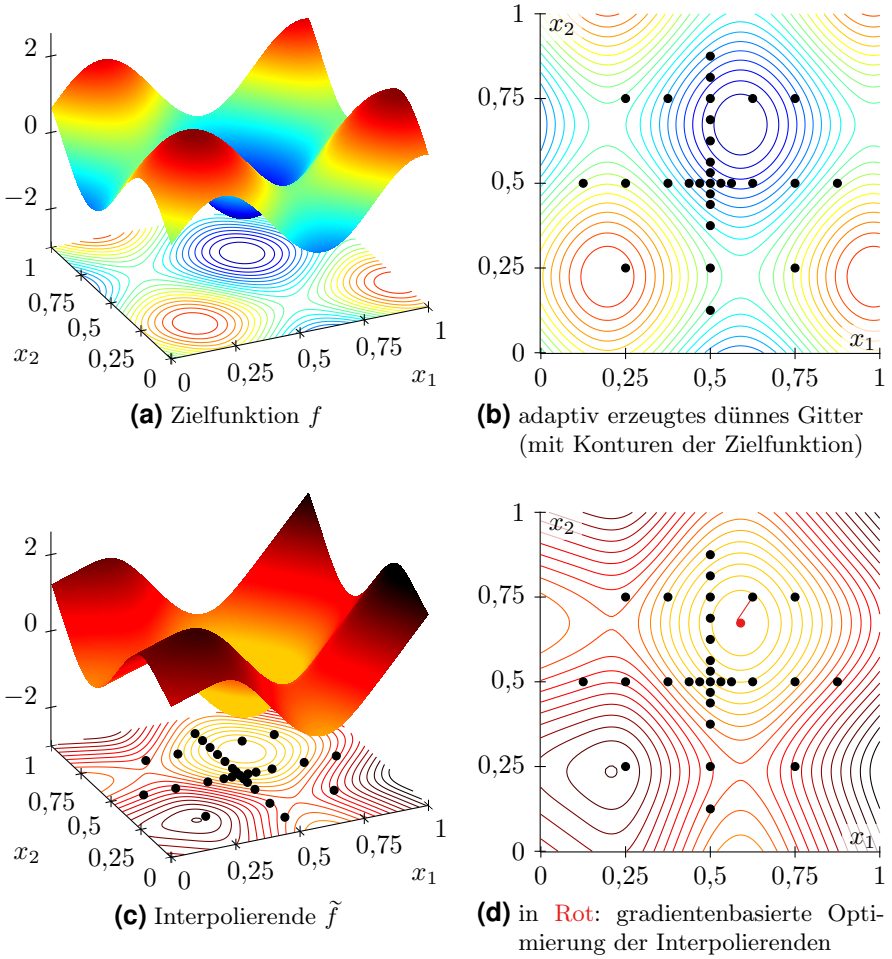
Zunächst einmal werten wir die Zielfunktion f (Abb. 1.1a) auf einem dünnen Gitter aus, das adaptiv durch die Funktionswerte in den Gitterpunkten generiert wurde (Abb. 1.1b). Anschließend interpolieren wir f an diesen Stellen durch eine Dünngitterfunktion $\widetilde{f}$ (Abb. 1.1c), also eine Linearkombination von Basisfunktionen, die auf dem dünnen Gitter definiert sind.

Die Grundidee des Verfahrens ist es nun, nicht die Zielfunktion f zu minimieren, sondern stattdessen die Interpolierende $\widetilde{f}$. Funktionsauswertungen von $\widetilde{f}$ sollten in der Praxis wesentlich „günstiger“ und damit schneller sein als solche von f. Weil die verwendeten Basisfunktionen bestimmte Glattheitseigenschaften erfüllen und die Ableitungen explizit ausgerechnet werden können, kann man gradientenbasierte Standardverfahren zur Minimierung der Dünngitterfunktion anwenden (Abb. 1.1d).

Diese Verfahren benötigen einen Startpunkt $\vec{x}_0$, der noch bestimmt werden muss. Dazu gibt es verschiedene Möglichkeiten: Man kann zum Beispiel wie in Abb. 1.1d dargestellt in dem Gitterpunkt starten, der den kleinsten Funktionswert von allen Gitterpunkten besitzt (diese Werte sind aufgrund der Interpolation ohnehin bereits bekannt). Eine weitere Möglichkeit wäre, die Interpolierende mithilfe globaler, gradientenfreier Verfahren zu minimieren und in dem so erhaltenen Minimum ein gradientenbasiertes Verfahren zu starten. Wir werden in Kapitel 6 eine Mischung beider Möglichkeiten anwenden.

Gliederung der Arbeit

Die weitere Arbeit gliedert sich wie folgt: Im nächsten Kapitel 2 werden dünne Gitter definiert und ein paar Eigenschaften genannt. Anschließend betrachten wir in Kapitel 3



(a) Zielfunktion f

(b) adaptiv erzeugtes dünnes Gitter (mit Konturen der Zielfunktion)

(c) Interpolierende $\widetilde{f}$

(d) in Rot: gradientenbasierte Optimierung der Interpolierenden

Abbildung 1.1: Optimierung der Funktion $f(x_1, x_2) := \sin(8x_1) + \sin(7x_2)$ mittels Gradientenverfahren auf Dünngitter-Interpolierenden ($N = 29$ Gitterpunkte)

Basisfunktionen, die man zur Interpolation verwenden kann. Kapitel 4 widmet sich der adaptiven Gittererzeugung und der Interpolation von Funktionswerten an den Punkten des erzeugten Gitters. Ein paar Methoden zur gradientenbasierten und -freien Optimierung werden in Kapitel 5 vorgestellt. Es schließt sich in Kapitel 6 die Beschreibung einer Implementierung des Verfahrens in C++ an. Eine Evaluierung des Verfahrens und der Implementierung in Kapitel 7 rundet die Arbeit ab.

2 Dünne Gitter

Dünne Gitter gibt es prinzipiell schon länger: Laut [18] wurden sie wohl vom russischen Mathematiker Smolyak entdeckt, der sie 1963 in [49] für die numerische Quadratur eingesetzt hat. Der Terminus „dünne Gitter“ (engl. *sparse grids*) wurde 1991 von Zenger in seiner Arbeit [58] geprägt, der dünne Gitter zur Lösung partieller Differentialgleichungen (PDEs) verwendete. In den 1990er-Jahren erfuhren dünne Gitter einen ungemeinen Popularitätsschub und wurden seitdem aufgrund verschiedener Vorzüge außer für die Lösung von PDEs durch Bungartz in [5] und für die numerische Integration durch Novak/Ritter in [38] und Gerstner/Griebel in [20] auch für Interpolation durch Klimke/Wohlmuth in [30] sowie für Data-Mining, also für die Gewinnung sinnvoller Informationen aus großen Datenmengen, benutzt (siehe die Arbeiten [19] von Garcke et al. und [40] von Pflüger).

Dünne Gitter wurden außerdem bereits für globale Optimierung verwendet, z. B. durch Ferenczi in [16] und Kinauer in [29]. Allerdings scheiterte die gradientenbasierte Optimierung von Dünngitterfunktionen an den verwendeten Basisfunktionen oder an der Umsetzung. Im Folgenden behandeln wir zunächst nur stetige, aber nicht stetig differenzierbare Basisfunktionen. Stetig differenzierbare und damit für die Optimierung geeignete Basisfunktionen werden in Kapitel 3 erklärt.

Wir folgen bei der Definition dünner Gitter im Wesentlichen den Darstellungen [40] von Pflüger und [18] von Garcke. Die Notationen in der Literatur unterscheiden sich vor allem bezüglich der Randbehandlung (siehe Abschnitt 2.2.3).

2.1 Eindimensionale hierarchische, stückweise lineare Basis

Definition 2.1 (eindimensionales Gitter)**.**

$$G_\ell := \{x_{\ell,j} := jh_\ell \mid j = 0, \ldots, 2^\ell\} \tag{2.1}$$

ist das eindimensionale Gitter mit Level $\ell \in \mathbb{N}_0$ und Schrittweite $h_\ell := 2^{-\ell}$.

Eindimensionale Gitter sind also spezielle äquidistante Unterteilungen von $[0, 1]$. Mit größer werdendem Level ℓ erhält man eine bzgl. der Mengeninklusion aufsteigende Kette $G_0 \subset G_1 \subset G_2 \subset \cdots$ von Gittern. Von G_ℓ gelangt man zu $G_{\ell+1}$, indem man jedes Intervall $[x_{\ell,j}, x_{\ell,j+1}]$ $(j = 0, \ldots, 2^\ell - 1)$ in der Mitte unterteilt und jeweils einen neuen Knoten einfügt. Auf G_ℓ kann man die Standard-Hütchenbasis definieren:

Definition 2.2 (Hütchenbasis)**.**

$$\varphi\colon \mathbb{R} \to \mathbb{R}, \quad \varphi(x) := \max(1 - |x|, 0) \tag{2.2}$$

ist die Standard-Hütchenfunktion. Allgemeine Hütchenfunktionen sind für $\ell \in \mathbb{N}_0$ und $i = 0, \ldots, 2^\ell$ definiert durch

$$\varphi_{\ell,i} \colon [0,1] \to \mathbb{R}, \quad \varphi_{\ell,i}(x) := \varphi(x/h_\ell - i). \tag{2.3}$$

Assoziiert man $\varphi_{\ell,i}$ mit dem Knoten $x_{\ell,i} \in G_\ell$, so ist $\varphi_{\ell,i}$ dadurch charakterisiert, dass $\varphi_{\ell,i}$ stetig ist, bzgl. G_ℓ stückweise linear ist sowie $\varphi_{\ell,i}(x_{\ell,j}) = \delta_{i,j}$ für alle $j = 0, \ldots, 2^\ell$ erfüllt ist (sogenannte „Lagrange-Eigenschaft“). Jedes der Hütchen hat also die „Höhe“ 1. Wichtig ist, dass $\varphi_{\ell,i}$ lokal getragen ist, genauer ist $\operatorname{supp} \varphi_{\ell,i} = [x_{\ell,i-1}, x_{\ell,i+1}] \cap [0,1]$.

Definition 2.3 (stückweise lineare, stetige Funktionen)**.**

$$V_\ell := \operatorname{span}\{\varphi_{\ell,i} \mid i = 0, \ldots, 2^\ell\} \tag{2.4}$$

ist der Raum aller (bzgl. G_ℓ) stückweise linearen, stetigen Funktionen.

Klar ist, dass die Funktionen $\varphi_{\ell,i}$, $i = 0, \ldots, 2^\ell$, aufgrund der Lagrange-Eigenschaft eine Basis von V_ℓ bilden, die „nodale Basis“. Indem man nur jede zweite Basisfunktion betrachtet, erhält man folgende

Definition 2.4 (hierarchischer Unterraum)**.**

$$W_\ell := \operatorname{span}\{\varphi_{\ell,i} \mid i \in I_\ell\} \quad \text{mit} \quad I_\ell := \begin{cases} \{i = 1, \ldots, 2^\ell - 1 \mid i \text{ ungerade}\}, & \text{falls } \ell \geqslant 1, \\ \{0, 1\}, & \text{falls } \ell = 0, \end{cases} \tag{2.5}$$

ist der hierarchische Unterraum mit Level $\ell \in \mathbb{N}_0$.

Eine elementare Eigenschaft der Basisfunktionen $\varphi_{\ell,i}$ eines hierarchischen Unterraums W_ℓ für $\ell \geqslant 1$ ist, dass der Schnitt von Trägern zweier Basisfunktionen aus W_ℓ entweder disjunkt ist oder nur einen Punkt enthält. Die hierarchische und die nodale Basis sind in Abb. 2.1 bis zu $\ell = 3$ dargestellt. Mit den hierarchischen Unterräumen kann man V_n für $n \in \mathbb{N}_0$ schreiben als direkte Summe

$$V_n = \bigoplus_{\ell=0}^{n} W_\ell, \tag{2.6}$$

was auch den Namen der Unterräume erklärt. Hier wurde ausgenutzt, dass die Basisfunktionen von W_{ℓ_1} und W_{ℓ_2} für $\ell_1 < \ell_2$ linear unabhängig sind, was daran liegt, dass $\varphi_{\ell_2,i_2}(x_{\ell_1,i_1}) = 0$ für alle $x_{\ell_1,i_1} \in G_{\ell_1}$ und $i_2 \in I_{\ell_2}$ gilt – an den Knoten eines Gitters G_{ℓ_1} verschwinden alle Basisfunktionen von hierarchischen Unterräumen höherer Levels ℓ_2 („Hierarchie-Eigenschaft“, siehe Abb. 2.1).

Wegen (2.6) kann man jede Funktion $u \in V_n$ als Linearkombination

$$u(x) = \sum_{\ell=0}^{n} \sum_{i \in I_\ell} \alpha_{\ell,i} \varphi_{\ell,i}(x) \tag{2.7}$$

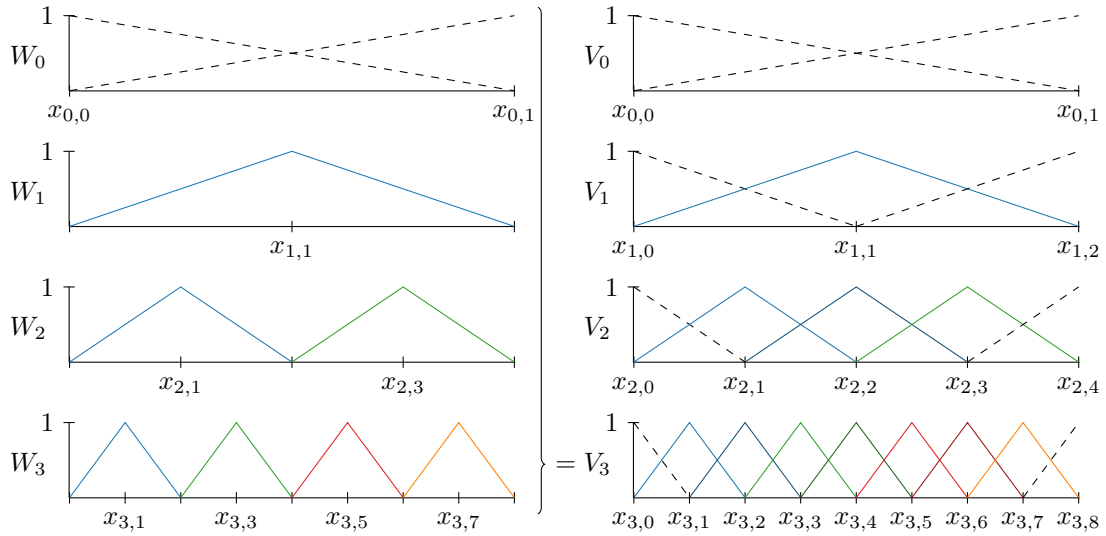


Abbildung 2.1: Hierarchische Basis von W_ℓ (links) und nodale Basis von V_ℓ (rechts). Die gestrichelten Funktionen sind für die Interpolation am Rand verantwortlich und werden in Abschnitt 2.2.3 weggelassen.

schreiben, wobei die Koeffizienten $\alpha_{\ell,i}$ „hierarchische Überschüsse“ genannt werden. Die Berechnung der hierarchischen Überschüsse, sodass $u(x)$ an den Gitterpunkten von G_n gegebene Daten interpoliert, gestaltet sich dank des folgenden Lemmas als sehr einfach.

Lemma 2.5 (hierarchische Überschüsse)**.** *Es gelten* $\alpha_{0,0} = u(0)$, $\alpha_{0,1} = u(1)$ *und*

$$\alpha_{\ell,i} = u(x_{\ell,i}) - \frac{u(x_{\ell,i+1}) + u(x_{\ell,i-1})}{2}, \quad \ell \geqslant 1,\, i \in I_\ell. \tag{2.8}$$

Beweis. Durch Einsetzen von $x = x_{\ell,i}$ mit $i \in I_\ell$ in (2.7) erhält man durch Lagrange- und Hierarchie-Eigenschaft

$$u(x_{\ell,i}) = \sum_{k=0}^{n} \sum_{j \in I_k} \alpha_{k,j}\varphi_{k,j}(x_{\ell,i}) = \alpha_{\ell,i} + \sum_{k=0}^{\ell-1} \sum_{j \in I_k} \alpha_{k,j}\varphi_{k,j}(x_{\ell,i}) = \alpha_{\ell,i} + u_{\ell-1}(x_{\ell,i}) \tag{2.9}$$

mit der Funktion $u_{\ell-1} \in V_{\ell-1}$, $u_{\ell-1}(x) := \sum_{k=0}^{\ell-1} \sum_{j \in I_k} \alpha_{k,j}\varphi_{k,j}(x)$, die $u(x)$ in $G_{\ell-1}$ interpoliert (wieder aufgrund der Hierarchie-Eigenschaft). Daher gilt

$$\begin{aligned} u_{\ell-1}(x_{\ell,i}) &= u_{\ell-1}\left(\frac{x_{\ell-1,(i-1)/2} + x_{\ell-1,(i+1)/2}}{2}\right) = \frac{u_{\ell-1}(x_{\ell-1,(i-1)/2}) + u_{\ell-1}(x_{\ell-1,(i+1)/2})}{2} \\ &= \frac{u(x_{\ell-1,(i-1)/2}) + u(x_{\ell-1,(i+1)/2})}{2}. \end{aligned} \tag{2.10}$$

Mit $x_{\ell-1,(i\pm1)/2} = x_{\ell,i\pm1}$ folgt (2.8) aus (2.9) und (2.10). $\alpha_{0,i} = u(i)$ folgt durch Einsetzen von i in (2.7) ($i = 0, 1$). □

2.2 Mehrdimensionale dünne Gitter

Definition 2.6 (d-dimensionales volles Gitter)**.**

$$G_{\vec{\ell}} := \bigtimes_{t=1}^{d} G_{\ell_t} = \{\vec{x}_{\vec{\ell},\vec{j}} := (x_{\ell_1,j_1}, \ldots, x_{\ell_d,j_d}) \mid \forall_{t=1,\ldots,d}\, x_{\ell_t,j_t} \in G_{\ell_t}\} \tag{2.11}$$

ist das d-dimensionale volle Gitter mit Level $\vec{\ell} = (\ell_1, \ldots, \ell_d) \in \mathbb{N}_0^d$ und Schrittweite $\vec{h}_{\vec{\ell}} := (2^{-\ell_1}, \ldots, 2^{-\ell_d})$.

Definition 2.7 (d-dimensionale Hütchenbasis)**.**

$$\varphi_{\vec{\ell},\vec{i}}\colon [0,1]^d \to \mathbb{R}, \quad \varphi_{\vec{\ell},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \varphi_{\ell_t,i_t}(x_t), \quad \vec{x} = (x_1, \ldots, x_d), \tag{2.12}$$

ist die d-dimensionale Hütchenfunktion mit Level $\vec{\ell} \in \mathbb{N}_0^d$ und Index $\vec{i} = (i_1, \ldots, i_d) \in \mathbb{N}_0^d$ mit $i_t \in \{0, \ldots, 2^{\ell_t}\}$ für alle $t = 1, \ldots, d$.

Diese Definitionen verallgemeinern das eindimensionale Gitter und die Hütchenfunktion über den üblichen Tensorprodukt-Ansatz auf mehrere Dimensionen. Solche Ansätze besitzen meist den Vorteil der Einfachheit bzgl. Analysis und Implementierung: So hat der Tensorprodukt-Ansatz neben der einfachen Auswertung der Funktion (und für die anderen Basisfunktionen aus Kapitel 3 auch der einfachen Auswertung der Ableitungen) den Vorteil, dass alle Eigenschaften aus dem 1D-Fall auf $\varphi_{\vec{\ell},\vec{i}}$ „vererbt“ werden. Die Hütchenfunktionen $\varphi_{\vec{\ell},\vec{i}}$ bilden zum Beispiel wieder die Basis des Raums aller stückweise d-linearen, stetigen Funktionen:

Definition 2.8 (stückweise d-lineare, stetige Funktionen)**.**

$$V_{\vec{\ell}} := \operatorname{span}\{\varphi_{\vec{\ell},\vec{i}} \mid \forall_{t=1,\ldots,d}\, i_t = 0, \ldots, 2^{\ell_t}\} \tag{2.13}$$

ist der Raum aller (bzgl. $G_{\vec{\ell}}$) stückweise d-linearen, stetigen Funktionen.

Zur Kehrseite des Ansatzes gehört, dass d-lineare Hütchenfunktionen hin zu einem Eckpunkt ihres Trägers wie $\mathcal{O}(x^d)$ $(x \to 0)$ stark abfallen. Das kann bei dünnen Gittern zu Oszillationen führen, sodass das Integral einer Dünngitter-Interpolierenden, die eine positive Funktion interpoliert, sogar negativ sein kann (konkretes Beispiel siehe [40]).

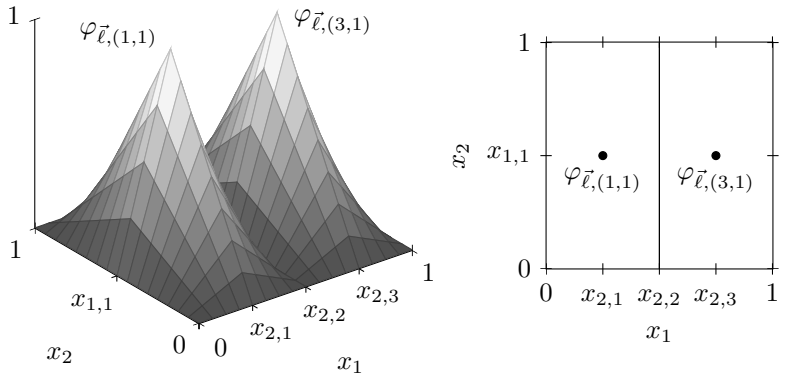
2.2.1 Hierarchische Unterräume

Wie in einer Dimension können wir auch in mehreren Dimensionen hierarchische Unterräume mit zugehörigen Indexmengen definieren.

Definition 2.9 (hierarchischer Unterraum)**.**

$$W_{\vec{\ell}} := \operatorname{span}\{\varphi_{\vec{\ell},\vec{i}} \mid \vec{i} \in I_{\vec{\ell}}\} \quad \text{mit} \quad I_{\vec{\ell}} := I_{\ell_1} \times \cdots \times I_{\ell_d} \tag{2.14}$$

ist der hierarchische Unterraum mit Level $\vec{\ell} \in \mathbb{N}_0^d$.



(a) 3D-Darstellung **(b)** Schema der Träger

Abbildung 2.2: Basisfunktionen $\varphi_{\vec{\ell},(1,1)}$ und $\varphi_{\vec{\ell},(3,1)}$ von $W_{\vec{\ell}}$ für $\vec{\ell} = (2,1)$

In Abb. 2.2 sind die beiden Basisfunktionen von $W_{\vec{\ell}}$ für $\ell = (2,1)$ dargestellt. In den folgenden Abbildungen werden wie in der schematischen Darstellung von Abb. 2.2b Basisfunktionen $\varphi_{\vec{\ell},\vec{i}}$ durch die Mittelpunkte $\vec{x}_{\vec{\ell},\vec{i}}$ ihrer Träger identifiziert. Mit den hierarchischen Unterräumen kann man den Raum $V_{\vec{n}}$ aller stückweise d-linearen, stetigen Funktionen schreiben als

$$V_{\vec{n}} = \bigoplus_{\ell_1=0}^{n_1} \cdots \bigoplus_{\ell_d=0}^{n_d} W_{\vec{\ell}}. \tag{2.15}$$

Nachfolgend betrachten wir nur noch Gitter mit denselben Levels in jeder Dimension, also $n\vec{e} := (n, \ldots, n)$ für $n \in \mathbb{N}_0$. Man erhält dann

$$V_{n\vec{e}} = \bigoplus_{\|\vec{\ell}\|_\infty \leqslant n} W_{\vec{\ell}}, \tag{2.16}$$

wobei die direkte Summe über alle Multiindizes $\vec{\ell} \in \mathbb{N}_0^d$ läuft, sodass jeder Eintrag höchstens n ist. In Abb. 2.3 sieht man $V_{3\vec{e}}$ im 2D-Fall aufgeteilt in die 16 hierarchischen Unterräume. Für jede Funktion $u \in V_{n\vec{e}}$ gilt analog wie im 1D-Fall mit geeigneten Koeffizienten, die wieder „hierarchische Überschüsse“ genannt werden, dass

$$u(\vec{x}) = \sum_{\|\vec{\ell}\|_\infty \leqslant n} \sum_{\vec{i} \in I_{\vec{\ell}}} \alpha_{\vec{\ell},\vec{i}} \varphi_{\vec{\ell},\vec{i}}(\vec{x}). \tag{2.17}$$

Die hierarchischen Überschüsse können Dimension für Dimension berechnet werden, indem man die Formel (2.8) sukzessiv für jede Dimension anwendet (siehe auch Abschnitt 4.2.1). Für $d = 2$ gilt z. B. für $\ell_1, \ell_2 \geqslant 1$ und $\vec{i} \in I_{\vec{\ell}}$

$$\begin{aligned} \alpha_{\vec{\ell},\vec{i}} = u(\vec{x}_{\vec{\ell},\vec{i}}) &- \frac{1}{2}\left(u(\vec{x}_{\vec{\ell},(i_1+1,i_2)}) + u(\vec{x}_{\vec{\ell},(i_1-1,i_2)}) + u(\vec{x}_{\vec{\ell},(i_1,i_2+1)}) + u(\vec{x}_{\vec{\ell},(i_1,i_2-1)})\right) \\ &+ \frac{1}{4}\left(u(\vec{x}_{\vec{\ell},(i_1+1,i_2+1)}) + u(\vec{x}_{\vec{\ell},(i_1-1,i_2+1)}) + u(\vec{x}_{\vec{\ell},(i_1+1,i_2-1)}) + u(\vec{x}_{\vec{\ell},(i_1-1,i_2-1)})\right). \end{aligned} \tag{2.18}$$

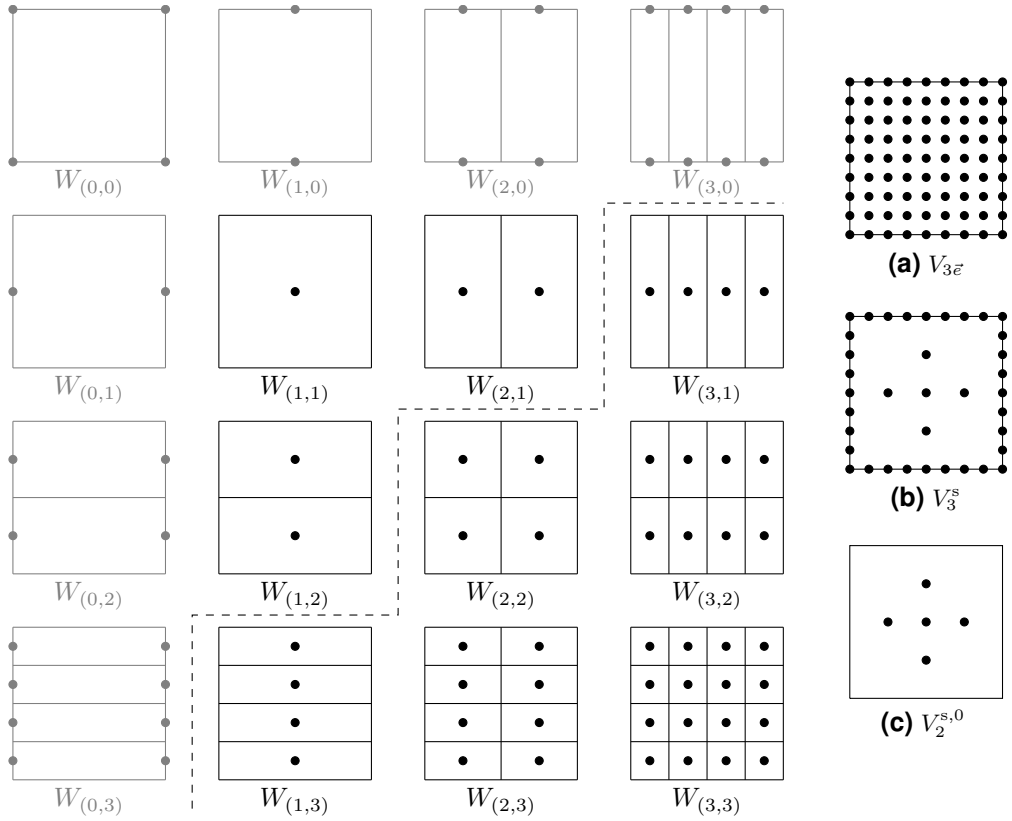


Abbildung 2.3: Gitterpunkte und Träger der Basisfunktionen (durchgezogene Linien) des vollen Gitters $V_{3\vec{e}}$ (alles) und des dünnen Gitters V_3^{s} (nur das oberhalb der gestrichelten Linie) in zwei Dimensionen. Die grauen hierarchischen Unterräume werden für $V_n^{\mathrm{s},0}$ weggelassen (siehe Abschnitt 2.2.3).

2.2.2 Dünne Gitter

Zur Darstellung einer Funktion $u_n \in V_{n\vec{e}}$ auf dem vollen Gitter benötigt man $(2^n - 1)^d$ viele innere Gitterpunkte. Wenn man also beispielsweise eine beliebige Funktion f interpolieren will, muss man $\mathcal{O}(2^{nd})$ Funktionsauswertungen durchführen, um nach [5] einen Interpolationsfehler von $\|u_n - f\|_{L^2} \in \mathcal{O}(2^{-2n})$ zu erhalten (wenn f hinreichend glatt ist und auf dem Rand verschwindet). Dieser gemeinhin als „Fluch der Dimensionalität“ (engl. *curse of dimensionality*) bezeichnete Sachverhalt drückt das Missverhältnis zwischen dem Interpolationsfehler und der mit d exponentiell steigenden Zahl an Gitterpunkten aus.

Die Idee der dünnen Gitter ist, in der direkten Summe (2.16) bestimmte Summanden wegzulassen, deren Einfluss bei der Interpolation in einem bestimmten Sinn „vernachlässigbar“ ist. Zur Motivation sei $H^2_{0,\mathrm{mix}}$ der Sobolev-Raum aller Funktionen auf $(0,1)^d$ mit verallgemeinerten Dirichlet-Nullrandwerten, deren gemischte Ableitungen

$$\mathrm{D}^{\vec{k}} := \frac{\partial^{\|\vec{k}\|_1}}{\partial x_1^{k_1} \cdots \partial x_d^{k_d}}, \quad k_1, \ldots, k_d \leqslant 2, \tag{2.19}$$

bis zum Koordinatengrad 2 (im Unterschied zum totalen Grad) L^2-integrierbar sind. Man kann zeigen (siehe z. B. [6, 18]), dass dann für $f \in H^2_{0,\text{mix}}$

$$\alpha_{\vec{\ell},\vec{i}} = \prod_{t=1}^{d} \left(-\frac{h_{\ell_t}}{2}\right) \cdot \int_{(0,1)^d} \varphi_{\vec{\ell},\vec{i}} \cdot \mathrm{D}^{2\vec{e}} f \,\mathrm{d}\vec{x} \tag{2.20}$$

für die hierarchischen Überschüsse gilt. Daraus folgt für $f(\vec{x}) = \sum_{\vec{\ell}\in\mathbb{N}_0^d} f_{\vec{\ell}}(\vec{x})$ mit den hierarchischen Anteilen

$$f_{\vec{\ell}}(\vec{x}) := \sum_{\vec{i}\in I_{\vec{\ell}}} \alpha_{\vec{\ell},\vec{i}} \varphi_{\vec{\ell},\vec{i}}(\vec{x}), \quad \text{dass} \quad \|f_{\vec{\ell}}\|_{L^2} \leqslant 3^{-d} 2^{-2\|\vec{\ell}\|_1} \cdot |f|_{H^2_{0,\text{mix}}}, \tag{2.21}$$

wenn man $\lim_{n\to\infty} V_{n\vec{e}}$ als dichten Teilraum von H^1 betrachtet (Beweis wieder siehe [6, 18]). Die hierarchischen Anteile fallen also bei steigendem Level $\vec{\ell}$ in der L^2-Norm ab wie $2^{-2\|\vec{\ell}\|_1}$, was die folgende Dünngitter-Definition motiviert. Alternativ kann man sie, wie in [6] geschehen, auch über die Lösung eines kontinuierlichen Rucksack-Problems herleiten.

Definition 2.10 (reguläres dünnes Gitter)**.**

$$V_n^{\text{s}} := \bigoplus_{\|\vec{\ell}\|_1 \leqslant n} W_{\vec{\ell}} \tag{2.22}$$

ist das reguläre dünne Gitter mit Level $n \in \mathbb{N}_0$.

Anschaulich gesprochen lässt man alle Basisfunktionen von $V_{n\vec{e}}$ weg, die einen sehr kleinen Träger besitzen (in Abb. 2.3 die unter der gestrichelten Linie). Jede Dünngitterfunktion $u \in V_n^{\text{s}}$ lässt sich analog zu (2.17) darstellen als

$$u(\vec{x}) = \sum_{\|\vec{\ell}\|_1 \leqslant n} \sum_{\vec{i}\in I_{\vec{\ell}}} \alpha_{\vec{\ell},\vec{i}} \varphi_{\vec{\ell},\vec{i}}(\vec{x}). \tag{2.23}$$

Für den L^2-Interpolationsfehler im Dünngitterraum gilt nach [6] für $H^2_{0,\text{mix}}$-Funktionen f, dass $\|u_n^{\text{s}} - f\|_{L^2} \in \mathcal{O}(2^{-2n} n^{d-1})$. Somit ist der Fehler nur um den vergleichsweise kleinen Faktor n^{d-1} schlechter als beim vollen Gitter. Die Anzahl der inneren Gitterpunkte ist dagegen nach [6] mit $\mathcal{O}(2^n n^{d-1})$ deutlich geringer als die Anzahl $\mathcal{O}(2^{nd})$ beim vollen Gitter.

2.2.3 Umgang mit dem Rand

Das Problem bei dem regulären dünnen Gitter aus Definition 2.10 ist, dass anteilsmäßig viele Punkte des Gitters auf dem Rand von $[0,1]^d$ liegen: Allein $\mathcal{O}(d2^{d+n})$ Punkte liegen auf den $d2^{d-1}$ Kanten von $[0,1]^d$, hinzu kommen viele weitere Punkte auf den Seitenflächen (k-dimensionale „Flächen“ für $k = 0, \ldots, d-1$). Bereits für $d = 2$ wird das deutlich (siehe Abb. 2.3b), aber speziell für höhere Dimensionalitäten liegt nur ein Bruchteil der Punkte im Inneren (beispielhafte Zahlen siehe [40]).

Das Problem wird etwas kleiner, wenn man W_0 und W_1 wie in [40] zusammenlegt. In diesem Fall erhält man auf dem Rand dieselbe Schrittweite wie auf den d „Hauptachsen“ (achsenparallele Geraden durch $0{,}5\vec{e}$). Auf diese Art konstruierte Gitter werden wir

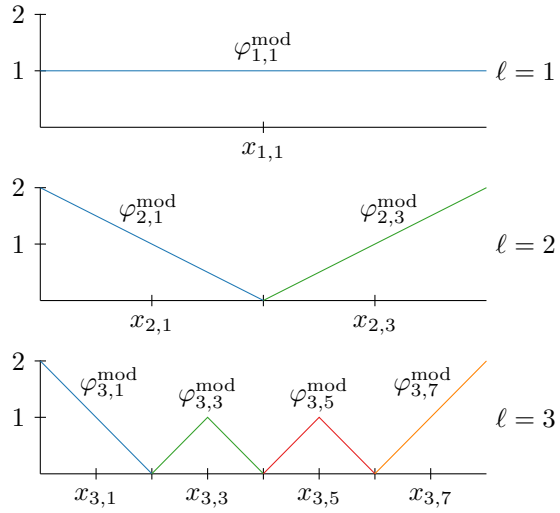


Abbildung 2.4: Modifizierte Basisfunktionen bis zum Level 3

in Abschnitt 2.3 „Boundary-Gitter“ nennen. Allerdings befinden sich dann immer noch relativ viele der Gitterpunkte auf dem Rand (siehe auch Abb. 2.6). Eine andere Lösung des Problems ist, wie in [40] nur die inneren Punkte für das dünne Gitter zu wählen:

Definition 2.11 (reguläres dünnes Gitter mit Nullrandwerten)**.**

$$V_n^{\text{s},0} := \bigoplus_{\|\vec{\ell}\|_1 \leqslant n+d-1} W_{\vec{\ell}}, \quad \forall_{t=1,\dots,d}\, \ell_t \geqslant 1, \tag{2.24}$$

ist das reguläre dünne Gitter mit Nullrandwerten und Level $n \in \mathbb{N}_0$.

Wir betrachten hier also nur die Basisfunktionen, die auf dem Rand von $[0,1]^d$ verschwinden. Man beachte die unterschiedlichen Summationsgrenzen für V_n^{s} und $V_n^{\text{s},0}$: Durch $\|\vec{\ell}\|_1 \leqslant n + d - 1$ wird sichergestellt, dass die feinste Gitterweite von V_n^{s} und $V_n^{\text{s},0}$ jeweils durch $h_n = 2^{-n}$ gegeben ist, indem eine weitere Diagonale in Abb. 2.3 hinzugenommen wird (in drei Dimensionen eine weitere „Diagonalebene“ usw.).

Analog wie vorhin kann man den Raum

$$V_{n\vec{e}}^0 := \operatorname{span}\{\varphi_{n\vec{e},\vec{i}} \mid \forall_{t=1,\dots,d}\, i_t = 1, \dots, 2^{\ell_t} - 1\} \tag{2.25}$$

definieren, der genau die stückweise d-linearen, stetigen Funktionen mit Nullrandwerten enthält. Laut [40] kann Definition 2.11 unter anderem für solche Situationen von Vorteil sein, in denen eine hohe Genauigkeit am Rand nicht erforderlich ist oder die Gitterpunkte adaptiv ausgewählt werden (wie im weiteren Verlauf dieser Arbeit).

Um Funktionen, die auf dem Rand nicht verschwinden, trotzdem einigermaßen akzeptabel interpolieren zu können, hat Pflüger in [40] modifizierte Basisfunktionen eingeführt. Bei diesen hat er für jeden Level $\ell > 1$ die beiden äußeren Basisfunktionen mit Index 1 bzw. $2^\ell - 1$ so verändert, dass die Funktionen vom inneren Gitterpunkt h_ℓ bzw. $1 - h_\ell$ zum Rand hin linear extrapolieren, also ihre Steigung beibehalten (siehe Abb. 2.4). Für den Level $\ell = 1$ wird die konstante Einsfunktion verwendet. Man erhält daher folgende

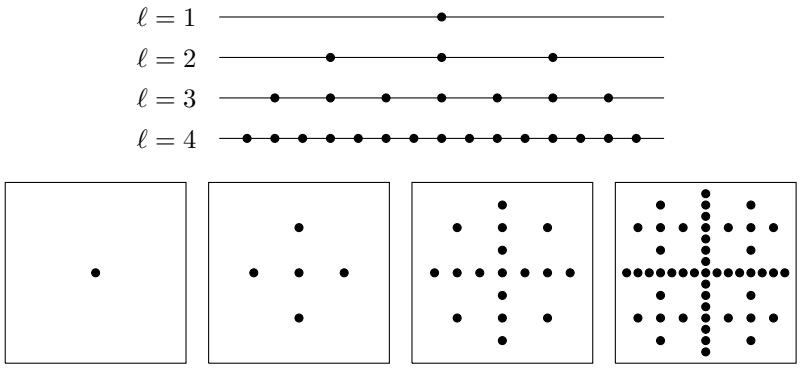


Abbildung 2.5: Noboundary-Gitter in einer und in zwei Dimensionen für $\ell = 1, 2, 3, 4$

Definition 2.12 (modifizierte Basisfunktionen)**.**

$$\varphi_{\ell,i}^{\text{mod}}(x) := \begin{cases} 1, & \text{falls } \ell = 1,\ i = 1, \\ 2 - x/h_\ell, & \text{falls } \ell > 1,\ i = 1,\ x \in [0, h_\ell], \\ 2 - (1-x)/h_\ell, & \text{falls } \ell > 1,\ i = 2^\ell - 1,\ x \in [1 - h_\ell, 1], \\ \varphi_{\ell,i}(x), & \text{sonst}, \end{cases} \tag{2.26}$$

ist die modifizierte Basisfunktion mit Level $\ell \in \mathbb{N}$ und Index $i \in I_\ell$.

Die mehrdimensionalen Basisfunktionen erhält man wie vorhin als Tensorprodukt. Mit ihnen kann man auch mit dem Gitter $V_n^{\text{s},0}$ auf dem Rand nicht verschwindende Funktionen interpolieren, sodass die Interpolierende einigermaßen sinnvoll auf den Rand fortsetzt ist.

2.3 Arten dünner Gitter

In Abschnitt 2.2.3 wurde bereits angedeutet, dass man durch eine andere Anordnung der Gitterpunkte im 1D-Gitter G_ℓ andere dünne Gitter erhält. Die hier vorgestellten Arten basieren teilweise auf numerischen Integrationsregeln, weil mit ihnen Integration auf dünnen Gittern durchgeführt werden kann (Details siehe [20, 38]).

2.3.1 Noboundary-Gitter

Das sogenannte Noboundary-Gitter ist hier nur der Vollständigkeit halber aufgezählt. Es ist das dünne Gitter $V_n^{\text{s},0}$ aus Abschnitt 2.2.3. Der Name ist wie in [16, 30] gewählt und leitet sich aufgrund der fehlenden Randpunkte her. Das Noboundary-Gitter mit Level $\ell \in \mathbb{N}$ ist in einer Dimension gegeben durch

$$G_\ell \backslash \{0, 1\} = \{j h_\ell \mid j = 1, \ldots, 2^\ell - 1\}. \tag{2.27}$$

In mehreren Dimensionen ist es definiert wie oben. Man erhält dann das bekannte dünne Gitter mit Nullrandwerten aus Definition 2.11. Für die ersten vier Levels und in bis zu zwei Dimensionen ist das Noboundary-Gitter in Abb. 2.5 dargestellt.

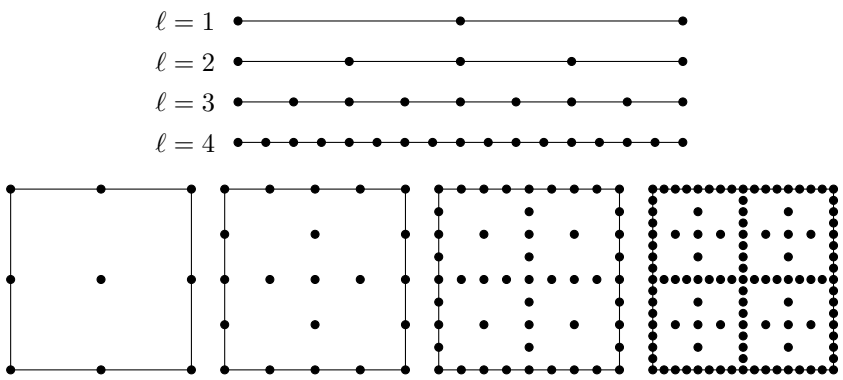


Abbildung 2.6: Boundary-Gitter in einer und in zwei Dimensionen für $\ell = 1, 2, 3, 4$

2.3.2 Boundary-Gitter

Randbehaftete Gitter heißen in [20] „Trapezregel-Gitter“ und in [16, 30, 47] verwirrenderweise „Clenshaw-Curtis-Gitter“. Wie man auf letzteren Namen kommt, ist ein wenig unklar, da die namensgebende Clenshaw-Curtis-Integrationsregel tschebyscheffverteilte Stützstellen verwendet, die im Gegensatz zu den folgenden Gitterpunkten alles andere als gleichmäßig verteilt sind. Das 1D-Boundary-Gitter mit Level $\ell \in \mathbb{N}$ ist jedenfalls definiert als

$$G_\ell = \{jh_\ell \mid j = 0, \ldots, 2^\ell\}. \tag{2.28}$$

In mehreren Dimensionen ist es ähnlich definiert wie das randlose Gitter in Definition 2.11. Für die ersten vier Levels und in bis zu zwei Dimensionen ist das Boundary-Gitter in Abb. 2.6 dargestellt. Im Unterschied zu V_n^{s} und zur oben genannten Literatur sind beim so definierten Boundary-Gitter die Schrittweiten von Rand und Hauptachsen gleich groß, weil beim Boundary-Gitter die Basisfunktionen von W_0 und W_1 im ersten Level zusammengefasst worden sind (in der genannten Literatur ist die Schrittweite auf dem Rand gröber, die 1D-Gitter sind dort $\{1/2\}$, $\{0, 1/2, 1\}$, $\{0, 1/4, 1/2, 3/4, 1\}$, $\ldots$). Weil die Implementierung später im SG^{++}-System erfolgt (siehe Kapitel 6) und SG^{++} das Boundary-Gitter wie hier definiert schon enthält, verwenden wir unsere Definition, bei der der Rand dieselbe Schrittweite wie die Hauptachsen besitzt.

2.3.3 Clenshaw-Curtis-Gitter

Wie eben schon erwähnt ist das Clenshaw-Curtis-Gitter hier ein Gitter mit nichtuniformen Stützstellen (im Gegensatz zu [16, 30, 47]). Der Name leitet sich von der Clenshaw-Curtis-Integrationsregel her, die von Clenshaw und Curtis in [8] zuerst beschrieben wurde. Diese Integrationsregel verwendet nach [20] entweder die Nullstellen oder die Extremstellen der Tschebyscheff-Polynome. Die Verwendung der Extremstellen hat den Vorteil, dass die Quadraturformeln geschachtelt sind. Das heißt, dass die Stützstellen der Formel eines bestimmten Grades eine Teilmenge der Stützstellen des nächsthöheren Grades sind. Genauer gilt sogar, dass von einem Grad zum nächsthöheren zwischen je zwei Stützstellen genau eine neue eingefügt wird.

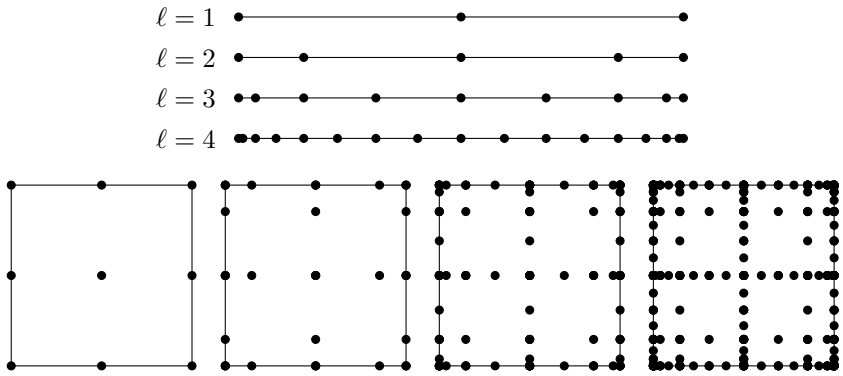


Abbildung 2.7: Clenshaw-Curtis-Gitter in einer und in zwei Dimensionen für $\ell = 1, 2, 3, 4$

Definition 2.13 (Tschebyscheff-Polynom)**.** Das n-te Tschebyscheff-Polynom ist für $n \in \mathbb{N}$ definiert durch

$$T_n \colon [-1, 1] \to \mathbb{R}, \quad T_n(x) := \cos(n \arccos(x)). \tag{2.29}$$

T_n besitzt $n + 1$ Extremstellen:

$$\cos\left(\frac{j\pi}{n}\right), \quad j = 0, \ldots, n. \tag{2.30}$$

Anschaulich bekommt man die Extremstellen von T_n, indem man wie in [48] einen Halbkreis-Bogen in n gleich große Stücke teilt und die Endpunkte jedes Stücks orthogonal auf den Durchmesser des Halbkreises projiziert. Aus den Extremstellen von T_{2^ℓ} erhält man nach einer affinen Transformation das 1D-Clenshaw-Curtis-Gitter

$$\left\{ \frac{1}{2}(\cos(\pi(1 - jh_\ell)) + 1) \;\middle|\; j = 0, \ldots, 2^\ell \right\}. \tag{2.31}$$

Das Argument von cos ist dabei speziell gewählt worden: Die Gitterpunkte werden auf diese Weise für steigendes j von links nach rechts angegeben, weil cos auf $[0, \pi]$ streng monoton fallend ist.

Mehrdimensionale Clenshaw-Curtis-Gitter definiert man analog wie das Boundary-Gitter. Für den ersten Level $\ell = 1$ ist das Clenshaw-Curtis- gleich dem Boundary-Gitter, für höhere Levels werden alle Punkte zum Rand hin etwas „verschoben“. Die Basisfunktionen sind ebenfalls Hütchen der Höhe 1, nur dieses Mal im Allgemeinen asymmetrisch (sodass sie die Lagrange-Eigenschaft erfüllen).

2.3.4 Weitere Arten

Es gibt noch einige weitere Arten von dünnen Gittern. Zu nennen ist etwa das Gauß-Legendre-Gitter, das auf der bekannten Gauß-Legendre-Quadratur fußt (Details siehe [20]). Sie verwendet die Nullstellen der Legendre-Polynome als Stützstellen. Das Problem ist, dass die Gauß-Legendre-Formeln im Allgemeinen nicht geschachtelt sind, weil die Nullstellen

der Legendre-Polynome für verschiedene Grade i. A. nicht ineinander enthalten sind – im Unterschied zu den Extremstellen der Tschebyscheff-Polynome. Man kann sich zwar wie Ferenczi in [16] das 1D-Gauß-Legendre-Gitter so definieren, dass die Gitter geschachtelt sind (definiere das 1D-Gitter mit Level ℓ rekursiv als das mit Level $\ell - 1$ zusammen mit den Nullstellen des Legendre-Polynoms vom Grad 2^ℓ). Das Resultat hat dann aber kaum noch etwas mit Gauß-Legendre-Quadratur zu tun.

Die erste Möglichkeit, um ein „richtiges“ Gauß-Legendre-Gitter zu erhalten, ist die Hinzufügung von zusätzlichen Stützstellen (Nullstellen der sogenannten Stieltjes-Polynome). Man erhält dann das in [20] abgebildete dünne Gitter. Der Nachteil dieser Methode ist, dass bei Erhöhung des 1D-Levels um eins nicht unbedingt in jedem „alten“ Intervall genau ein „neuer“ Gitterpunkt eingefügt wird (manchmal kommen mehrere neue Gitterpunkte in einem Intervall vor und manchmal keine), was die Implementierung von Dünngitter-Algorithmen deutlich erschwert.

Eine andere Möglichkeit ist das Gauß-Patterson-Gitter, bei dem eine bestimmte Folge von Polynomen konstruiert wird, die eine Orthogonalitätsbedingung erfüllen (Genaueres siehe [20]). Das Gitter enthält dann nicht nur die Nullstellen eines Legendre-Polynoms, sondern auch noch die der so konstruierten Polynome. Der Berechnung der letztgenannten Nullstellen ist jedoch kompliziert, weil dazu diese Polynome als Legendre- oder Tschebyscheff-Entwicklung geschrieben und nach Lösung eines linearen Gleichungssystems die Nullstellen durch ein Newton-Verfahren berechnet werden müssen. Alternativ kann man ein partielles inverses Eigenwertproblem lösen (siehe [20]).

Wir haben in dieser Arbeit auf eine Implementierung dieser Gitter aus zweierlei Gründen verzichtet: Erstens wollen wir die Algorithmen nicht unnötig kompliziert machen und zweitens möchten wir uns nicht zu lange mit der Gitterberechnung aufhalten.

3 Basisfunktionen

Wie in Kapitel 1 beschrieben, wollen wir gradientenbasierte Optimierungsverfahren auf Dünngitter-Interpolierende anwenden. Allerdings können Optimierungsverfahren wie Gradientenmethode und Newton-Verfahren bei Funktionen, die nicht stetig differenzierbar sind, im Allgemeinen nicht eingesetzt werden, weil Konvergenz nicht mehr garantiert werden kann (und natürlich, weil Gradienten oder Hesse-Matrizen eventuell erst gar nicht ausgerechnet werden können).

Daher müssen wir die aus dem letzten Kapitel bekannten stückweise linearen Basisfunktionen, die an „ihren“ Gitterpunkten nicht stetig differenzierbar sind, durch andere, „glattere“ Basisfunktionen ersetzen. Pflüger hat in [40] stückweise d-polynomiale Funktionen, B-Spline-Funktionen und Mexican-Hat-Funktionen untersucht. Die erstgenannten Funktionen sind mehr oder weniger glatte Versionen der Hütchen: Genauer hat die stückweise d-polynomiale Funktion $\varphi_{\ell,i}(x)$ den Träger $[x_{\ell,i-1}, x_{\ell,i+1}]$ und ist glatt in $x = x_{\ell,i}$, jedoch geht die Ableitung in $x = x_{\ell,i\pm1}$ unstetig in 0 über. Daher konzentrieren wir uns auf B-Spline- und Mexican-Hat-Basisfunktionen, wobei der Fokus auf B-Splines liegt.

3.1 B-Splines

B-Splines besitzen eine Reihe schöner Eigenschaften. Sie sind einfach zu handhaben, weil sie stückweise Polynome vom frei wählbaren Grad p sind. Sie sind an den Knoten $(p-1)$-mal stetig differenzierbar und sie haben einen kompakten Träger, der dazu noch je nach Grad p recht klein ist. Ein weiterer Vorteil ist, dass sie eine Verallgemeinerung der stückweise linearen, stetigen Basisfunktionen aus Kapitel 2 darstellen (diese erhält man für $p = 1$).

B-Splines werden für die verschiedensten Zwecke benutzt, zum Beispiel zur Modellierung (Stichwort „NURBS“), für Finite Elemente (eine gute Übersicht für diese Anwendung bietet [25]) oder zur Interpolation. In der früheren Arbeit [54] wurden sie beispielsweise zur Approximation unregelmäßig verteilter Daten verwendet. Bei der Anwendung für dünne Gitter wurden sie laut [40] in [39] für Regressionsprobleme eingeführt. In dieser Arbeit wollen wir die Tauglichkeit der B-Splines für die Optimierung von Dünngitter-Interpolierenden untersuchen.

Nachfolgend halten wir uns in etwa an die Darstellungen [26] von Höllig/Hörner und [40] von Pflüger. Wir betrachten der Kürzer halber nur das Noboundary-Gitter $V_n^{\mathrm{s},0}$ aus Definition 2.11 (dünnes Gitter mit Nullrandwerten) und das Clenshaw-Curtis-Gitter jeweils als Beispiel für uniforme bzw. nichtuniforme dünne Gitter. Die Betrachtung für das Boundary-Gitter ist ähnlich wie beim Noboundary-Fall. Zur Vereinfachung betrachten wir meist nur den Fall, dass p ungerade ist. In diesem Fall werden die B-Spline-Knoten mit den Gitterpunkten zusammenfallen, was die Situation speziell für nichtuniforme Gitter etwas erleichtert.

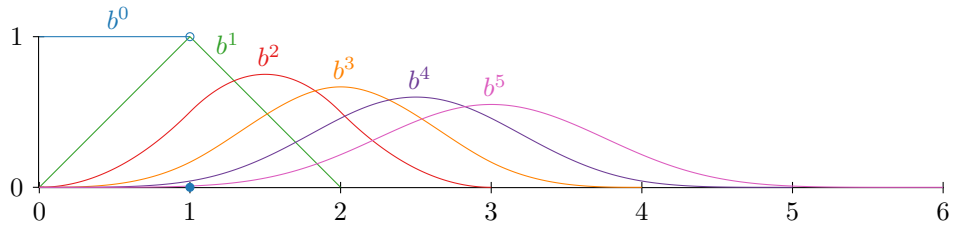


Abbildung 3.1: B-Splines b^p für $p = 0, 1, \ldots, 5$

3.1.1 Univariate, uniforme, hierarchische B-Splines

Definition 3.1 (uniformer B-Spline)**.** Der uniforme B-Spline $b^p \colon \mathbb{R} \to \mathbb{R}$ vom Grad $p \in \mathbb{N}_0$ ist durch die Cox-de-Boor-Rekursionsformel (siehe [3, 11]) definiert als

$$b^0(x) := \begin{cases} 1, & \text{falls } x \in [0,1), \\ 0, & \text{sonst,} \end{cases} \tag{3.1}$$

$$b^p(x) := \frac{1}{p}\left(x b^{p-1}(x) + (p+1-x) b^{p-1}(x-1)\right), \quad p \geqslant 1. \tag{3.2}$$

Die B-Splines mit den gängigsten Graden sind in Abb. 3.1 dargestellt. Der uniforme B-Spline $b^p(x)$ vom Grad p besitzt den Träger $[0, p+1]$. Auf jedem Intervall $[k, k+1)$, $k = 0, \ldots, p$, ist $b^p(x)$ ein nichtnegatives Polynom vom Grad p. Der B-Spline ist durch 1 nach oben beschränkt und wird im Punkt $\frac{p+1}{2}$ maximal. $b^p(x)$ ist an den „Knoten“ $x = 0, \ldots, p+1$ genau $(p-1)$-fach stetig differenzierbar, wobei für $x \in \mathbb{R}$

$$\frac{\mathrm{d}}{\mathrm{d}x} b^p(x) = b^{p-1}(x) - b^{p-1}(x-1) \tag{3.3}$$

gilt. Durch Integration erhält man die Formel

$$b^p(x) = \int_0^1 b^{p-1}(x-y)\,\mathrm{d}y, \tag{3.4}$$

die $b^p(x)$ als Ergebnis der Faltung von $b^{p-1}(x)$ mit $b^0(x)$ darstellt. Alternativ kann man B-Splines über (3.4) definieren und Definition 3.1 als Satz herleiten (wie in [25] geschehen).

Definition 3.2 (hierarchischer, uniformer B-Spline)**.** Die hierarchische, uniforme B-Spline-Basisfunktion mit Level $\ell \in \mathbb{N}$ und Index $i \in I_\ell$ ist definiert durch

$$\varphi^p_{\ell,i}(x) := b^p\left(\frac{x}{h_\ell} + \frac{p+1}{2} - i\right). \tag{3.5}$$

$\varphi^p_{\ell,i}(x)$ wurde so gewählt, dass das Maximum an der Stelle $x_{\ell,i}$ liegt und der Träger gleich $h_\ell \cdot [i \pm (p+1)/2]$ ist. Man beachte, dass für p ungerade die Knoten der B-Splines $\varphi^p_{\ell,i}(x)$ auf den Gitterpunkten liegen, genauer hat $\varphi^p_{\ell,i}(x)$ die Knoten

$$x_{\ell,i-(p+1)/2}, \quad \ldots, \quad x_{\ell,i}, \quad \ldots, \quad x_{\ell,i+(p+1)/2}. \tag{3.6}$$

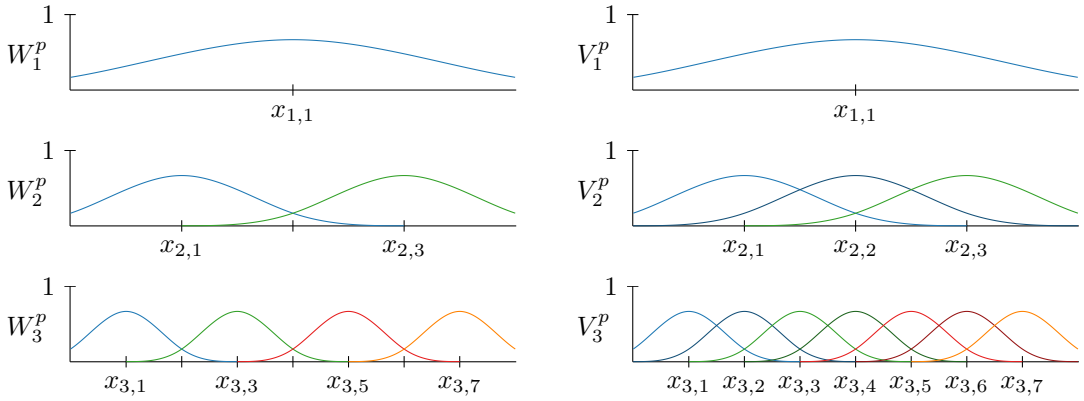


Abbildung 3.2: Hierarchische Basis von W_ℓ^p (links) und nodale Basis von V_ℓ^p (rechts) für $p = 3$

Für p gerade liegen die Knoten jeweils genau zwischen zwei Gitterpunkten. Die Knoten sind also in diesem Fall gegeben durch

$$x_{\ell,i-p/2} - \frac{h_\ell}{2},\ \ldots,\ x_{\ell,i} - \frac{h_\ell}{2},\ x_{\ell,i} + \frac{h_\ell}{2},\ \ldots,\ x_{\ell,i+p/2} + \frac{h_\ell}{2}. \tag{3.7}$$

Aus diesem Grund beschränken wir uns wie eingangs erwähnt auf den ungeraden Fall: Bei nichtuniformen Gittern wie dem Clenshaw-Curtis-Gitter kann man nichtuniforme B-Spline-Basisfunktionen konstruieren, indem man als B-Spline-Knoten einfach die Gitterpunkte wählt, was für geraden B-Spline-Grad nicht so leicht geht.

Wir können nun analog zum stückweise linearen Fall (Definition 2.4) für $\ell \in \mathbb{N}$ hierarchische B-Spline-Räume definieren durch

$$W_\ell^p := \operatorname{span}\{\varphi_{\ell,i}^p \mid i \in I_\ell\}. \tag{3.8}$$

Den Aufspann aller hierarchischen B-Splines eines Levels $\ell \in \mathbb{N}$ bezeichnen wir dagegen analog zu Definition 2.3 (hier jedoch ohne Randfunktionen) als

$$V_\ell^p := \operatorname{span}\{\varphi_{\ell,i}^p \mid i = 1, \ldots, 2^\ell - 1\}. \tag{3.9}$$

Die hierarchische und die nodale Basis sind in Abb. 3.2 dargestellt. Nun kann man sich die Frage stellen, welcher Raum von den W_ℓ^p aufgespannt wird, ob also hier, wie beim Fall stückweiser linearer Funktionen, ebenfalls $V_n^p = \bigoplus_{\ell=1}^n W_\ell^p$ gilt. Insbesondere sollten wir erst einmal klären, ob die $\varphi_{\ell,i}^p$ mit $i \in I_\ell$ überhaupt eine Basis von W_ℓ^p darstellen.

3.1.2 Nichtuniforme B-Splines

Um diese subtile Fragestellung beantworten und außerdem B-Splines auf nichtuniformen Gittern wie dem Clenshaw-Curtis-Gitter definieren zu können, ist es notwendig, die Definition der uniformen B-Splines zu nichtuniformen B-Splines zu verallgemeinern. Dabei halten wir uns größtenteils an Hölligs Notation in [26].

Definition 3.3 (nichtuniformer B-Spline)**.** Sei $\vec{\xi} = (\xi_0, \ldots, \xi_{m+p})$ für $m \geqslant 1$ eine Knotenfolge, d. h. eine endliche, monoton steigende Folge von reellen Zahlen. Dann sind die B-Splines $b^p_{k,\vec{\xi}}$ vom Grad p für $k = 0, \ldots, m-1$ definiert durch die Rekursion

$$b^0_{k,\vec{\xi}}(x) := \begin{cases} 1, & \text{falls } x \in [\xi_k, \xi_{k+1}), \\ 0, & \text{sonst,} \end{cases} \tag{3.10}$$

$$b^p_{k,\vec{\xi}} := \gamma^p_{k,\vec{\xi}} b^{p-1}_{k,\vec{\xi}} + (1 - \gamma^p_{k+1,\vec{\xi}}) b^{p-1}_{k+1,\vec{\xi}}, \quad \gamma^p_{k,\vec{\xi}}(x) := \frac{x - \xi_k}{\xi_{k+p} - \xi_k}, \quad p \geqslant 1. \tag{3.11}$$

Dabei werden Terme mit verschwindendem Nenner weggelassen.

Der letzte Satz der Definition ist für Knotenfolgen von Bedeutung, bei denen mehrere der Knoten zusammenfallen. Man spricht dann von „vielfachen Knoten“, also z. B. von doppelten oder dreifachen Knoten. Der uniforme B-Spline $b^p(x)$ ergibt sich als Spezialfall für $m := 1$, $k := 0$ und $\vec{\xi} = (\xi_0, \ldots, \xi_{p+1}) := (0, 1, \ldots, p+1)$. Die oben genannten Eigenschaften von uniformen B-Splines lassen sich bis auf die Position des Maximums sinngemäß auf den nichtuniformen Fall verallgemeinern. Die Ableitungsformel wird nach [26] zu

$$\frac{\mathrm{d}}{\mathrm{d}x} b^p_{k,\vec{\xi}}(x) = \alpha^p_{k,\vec{\xi}} b^{p-1}_{k,\vec{\xi}}(x) - \alpha^p_{k+1,\vec{\xi}} b^{p-1}_{k+1,\vec{\xi}}(x), \quad \alpha^p_{k,\vec{\xi}} := \frac{p}{\xi_{k+p} - \xi_k}. \tag{3.12}$$

Nun schauen wir den Raum an, der von allen nichtuniformen B-Splines einer Knotenfolge erzeugt wird. Die Funktionen dieses Raumes nennt man „Splines“. Der folgende Satz findet sich samt Beweis ebenfalls in [26].

Satz 3.4 (Spline-Raum)**.** *Sei $\vec{\xi} = (\xi_0, \ldots, \xi_{m+p})$ eine Knotenfolge ohne vielfache Knoten. Dann bilden die B-Splines $b^p_{0,\vec{\xi}}, \ldots, b^p_{m-1,\vec{\xi}}$ eine Basis des Spline-Raums*

$$S^p_{\vec{\xi}} := \operatorname{span}\{b^p_{k,\vec{\xi}} \mid k = 0, \ldots, m-1\}. \tag{3.13}$$

$S^p_{\vec{\xi}}$ besteht aus den auf $D := [\xi_p, \xi_m]$ stetigen Funktionen, die auf jedem Knotenintervall $[\xi_k, \xi_{k+1}]$ in D Polynome vom Grad $\leqslant p$ und an jedem Knoten ξ_k im Inneren von D mindestens $(p-1)$-fach stetig differenzierbar sind.

Um zu zeigen, dass $\{\varphi^p_{\ell,i} \mid i \in I_\ell\}$ eine Basis des uniformen, hierarchischen Raums W^p_ℓ aus Abb. 3.2 ist, betrachten wir den nodalen Raum V^p_ℓ zusammen mit den erzeugenden Funktionen $\varphi^p_{\ell,i}$. Wählt man

$$\xi_k := \left(k + 1 - \frac{p+1}{2}\right) h_\ell, \quad m := 2^\ell - 1, \tag{3.14}$$

so ist $V^p_\ell = S^p_{\vec{\xi}}$: Es gilt nämlich $\varphi^p_{\ell,i} = b^p_{i-1,\vec{\xi}}$ für $i = 1, \ldots, m$, da die Knoten von $b^p_{i-1,\vec{\xi}}$ uniform mit Abstand h_ℓ um den mittleren Knoten $\xi_{i+(p-1)/2} = ih_\ell$ liegen. Dabei ist zu beachten, dass man streng genommen hierfür jede Basisfunktion $\varphi^p_{\ell,i}$ auf $D = [\xi_p, \xi_m]$ einschränkt. Damit ist die Menge $\{\varphi^p_{\ell,i} \mid i = 1, \ldots, 2^\ell - 1\}$ der nodalen B-Spline-Funktionen eine Basis von V^p_ℓ und somit auch $\{\varphi^p_{\ell,i} \mid i \in I_\ell\}$ eine Basis von W^p_ℓ.

Jetzt betrachten wir die Summe $\sum_{\ell=1}^{n} W_\ell^p$ aller hierarchischen Räume der Levels $\ell \leqslant n$. Bei stückweisen linearen Funktionen ist die analoge Summe direkt, weil die Basisfunktionen eines Levels auf Knoten niedrigerer Levels verschwinden (Hierarchie-Eigenschaft). Für B-Splines ist die Situation komplizierter, weil das für $p \geqslant 2$ im Allgemeinen nicht gilt. Wir können also so nicht zeigen, dass die Summe direkt ist.

Für $p = 2$ kann man die Direktheit der Summe $\sum_{\ell=1}^{n} W_\ell^p$ wie folgt induktiv über $n \in \mathbb{N}$ beweisen: Für den Induktionsschritt $(n-1) \to n$ sei

$$f := \sum_{\ell=1}^{n} \sum_{i \in I_\ell} \alpha_{\ell,i} \varphi_{\ell,i}^p = 0 \tag{3.15}$$

eine Linearkombination der Nullfunktion. Dann ist f an jedem der „feinsten“ Gitterpunkte $x = x_{n,j}$ mit $j \in I_n$ eine Linearkombination von Funktionen, die in x glatt sind (alle $\varphi_{\ell,i}^p$ mit $\ell < n$), und einer einzigen Funktion $\varphi_{n,j}^p$, die in x nicht glatt ist, sondern nur $(p-1)$-fach stetig differenzierbar. Weil die Nullfunktion jedoch überall glatt ist, ist das nur möglich, wenn der Koeffizient $\alpha_{n,j}$ der nicht glatten Funktion verschwindet (dazu betrachte man die p-ten Ableitungen). Es gilt also $\alpha_{n,j} = 0$ für alle $j \in I_n$. Übrig bleibt

$$\sum_{\ell=1}^{n-1} \sum_{i \in I_\ell} \alpha_{\ell,i} \varphi_{\ell,i}^p = 0 \tag{3.16}$$

und induktiv verschwinden auch die restlichen Koeffizienten $\alpha_{\ell,i}$, $\ell < n$, $i \in I_\ell$. Somit ist die Summe $\sum_{\ell=1}^{n} W_\ell^p$ für $p = 2$ direkt.

Für $p \geqslant 3$ ist die Argumentation schwieriger, weil sich dann die Träger von „übernächsten“ Basisfunktionen $\varphi_{\ell,i\pm2}^p$ berühren. Wir gehen im Folgenden aus praktischen Gründen davon aus, dass die Summe $\sum_{\ell=1}^{n} W_\ell^p$ auch für diese Fälle direkt ist.

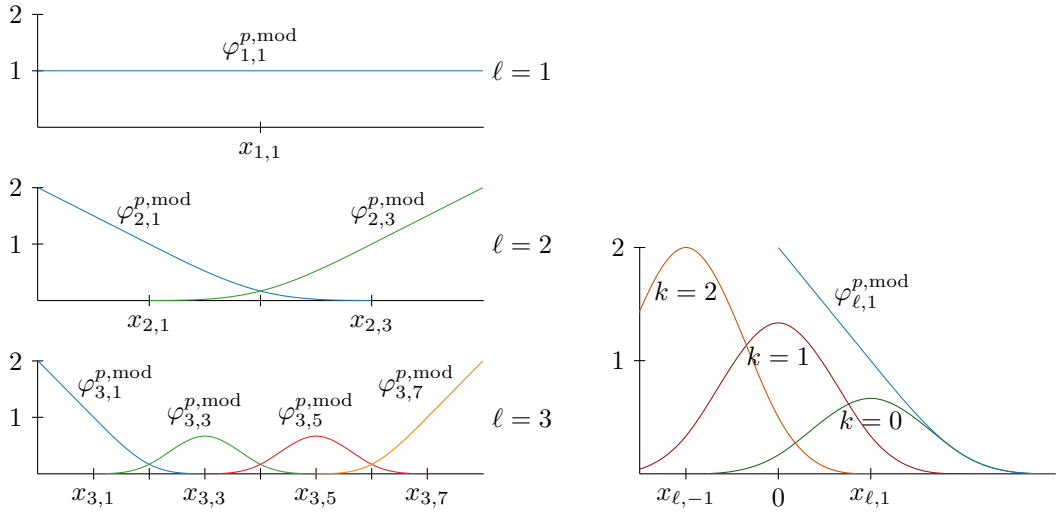
Nun bleibt die Frage, was $\bigoplus_{\ell=1}^{n} W_\ell^p$ ist. Weiter oben haben wir schon $W_n^p \leqslant S_{\vec{\xi}}^p = V_n^p$ festgestellt, wenn wir $\vec{\xi}$ und m so wählen wie in (3.14) mit $\ell = n$, also

$$\xi_k := \left(k + 1 - \frac{p+1}{2}\right) h_n, \quad m := 2^n - 1. \tag{3.17}$$

Schränkt man jede der Basisfunktionen $\varphi_{\ell,i}^p$ auf $D = [\xi_p, \xi_m]$ ein, so gilt $W_\ell^p \leqslant S_{\vec{\xi}}^p$ auch für $\ell < n$: Die entsprechenden Basisfunktionen $\varphi_{\ell,i}^p$, $i \in I_\ell$, niedrigerer Levels sind auf den Knotenintervallen von $\vec{\xi}$ in D Polynome vom Grad $\leqslant p$, die an den Knoten selbst mindestens $(p-1)$-fach stetig differenzierbar sind (hier geht ein, dass p ungerade ist). Daraus folgt nach Satz 3.4, dass $\varphi_{\ell,i}^p \in S_{\vec{\xi}}^p$ und somit $W_\ell^p \leqslant S_{\vec{\xi}}^p$ für $\ell = 1, \ldots, n$. Es gilt also $\bigoplus_{\ell=1}^{n} W_\ell^p \leqslant S_{\vec{\xi}}^p$. Dabei gilt sogar Gleichheit, weil

$$\dim\left(\bigoplus_{\ell=1}^{n} W_\ell^p\right) = \sum_{\ell=1}^{n} |I_\ell| = \sum_{\ell=1}^{n} 2^{\ell-1} = 2^n - 1 = m \tag{3.18}$$

und m nach Satz 3.4 die Dimension von $S_{\vec{\xi}}^p$ ist. Somit ist $\bigoplus_{\ell=1}^{n} W_\ell^p = S_{\vec{\xi}}^p = V_n^p$, wenn man jede der Basisfunktionen auf $D = [\xi_p, \xi_m]$ einschränkt.



(a) erste drei Levels $\ell = 1, 2, 3$ **(b)** Zustandekommen der Randfunktion

Abbildung 3.3: Modifizierte B-Spline-Basisfunktionen für $p = 3$

3.1.3 Modifizierte B-Splines

Die Basisfunktionen $\varphi_{\ell,i}^p(x)$ aus Definition 3.2 verschwinden zwar nicht alle auf dem Rand, dennoch werden Linearkombinationen von ihnen am Rand unnatürlich abfallen. Daher betrachtet Pflüger in [40] modifizierte Funktionen folgender Form (siehe Abb. 3.3a):

Definition 3.5 (modifizierte B-Splines)**.** Sei

$$\psi_\ell^p(x) := \sum_{k=0}^{\lceil (p+1)/2 \rceil} (k+1)\varphi_{\ell,1-k}^p(x). \tag{3.19}$$

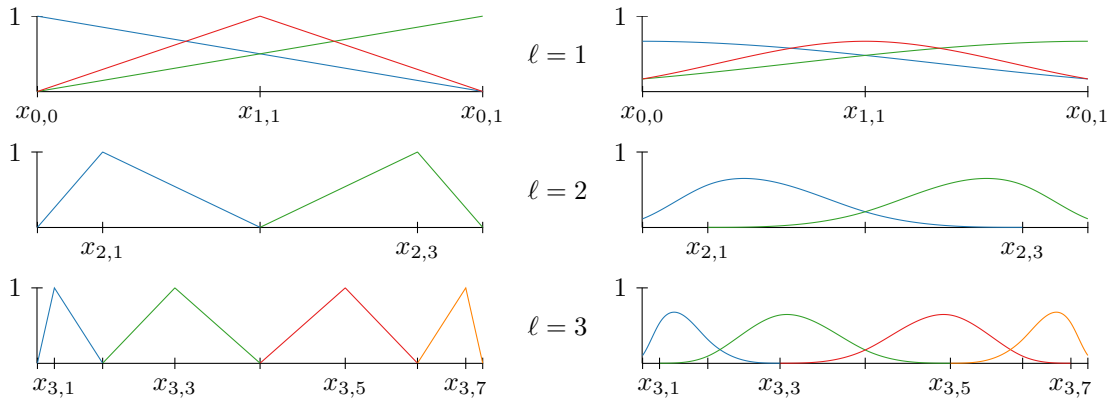
Dann ist

$$\varphi_{\ell,i}^{p,\mathrm{mod}}(x) := \begin{cases} 1, & \text{falls } \ell = 1,\ i = 1, \\ \psi_\ell^p(x), & \text{falls } \ell > 1,\ i = 1, \\ \psi_\ell^p(1-x), & \text{falls } \ell > 1,\ i = 2^\ell - 1, \\ \varphi_{\ell,i}^p(x), & \text{sonst}, \end{cases} \tag{3.20}$$

die modifizierte B-Spline-Basisfunktion mit Level $\ell \in \mathbb{N}$ und Index $i \in I_\ell$.

Die Rand-Basisfunktionen mit Index $i = 1$ bzw. $i = 2^\ell - 1$ entstehen also durch gewichtete Addition weiterer B-Splines, deren Maximalstelle nicht mehr in $(0, 1)$ liegt. Dabei wurden die Koeffizienten $(k+1)$ für die B-Splines $\varphi_{\ell,1-k}^p(x)$ so gewählt, dass $\psi_\ell^p(x)$ für $p \leqslant 4$ die ursprüngliche Basisfunktion $\varphi_{\ell,1}^p(x)$ linear zum Rand hin extrapoliert – ganz ähnlich wie bei den stückweise linearen Basisfunktionen. Das ergibt sich aus der Beziehung

$$x = \sum_{k \in \mathbb{Z}} \left(k + \frac{p+1}{2}\right) b^p(x-k), \quad x \in \mathbb{R}, \tag{3.21}$$



(a) $p = 1$ **(b)** $p = 3$

Abbildung 3.4: B-Splines-Basisfunktionen auf Clenshaw-Curtis-Gittern

die man mithilfe der sogenannten Marsden-Identität beweisen kann (Beweis für allgemeine Knotenfolgen siehe [26]). Daraus folgt

$$\psi_\ell^p(x) = 2 - \frac{x}{h_\ell}, \quad p \leqslant 4, x \in \left[0, \frac{5-p}{2} h_\ell\right] \tag{3.22}$$

(siehe Abb. 3.3b). Für $p > 4$ gilt das nicht mehr, weil dann relevante B-Splines in der Summe (3.19) fehlen. Die Abweichung von $2 - \frac{x}{h_\ell}$ ist aber in diesem Fall sehr klein.

3.1.4 B-Splines auf Clenshaw-Curtis-Gittern

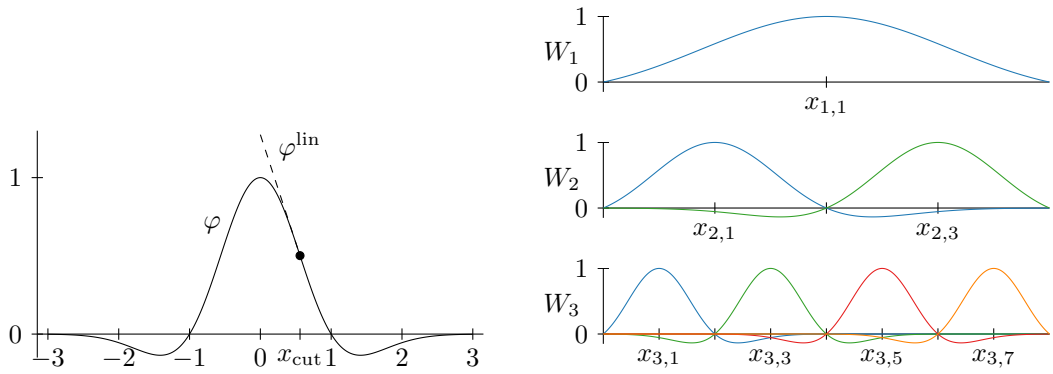
Auf nichtuniformen Gittern wie dem Clenshaw-Curtis-Gitter kann man für ungerades p passende B-Splines-Basisfunktionen durch nichtuniforme B-Splines (siehe Definition 3.3) definieren: Bezeichnet $x_{\ell,i}^{\mathrm{CC}}$ den i-ten Clenshaw-Curtis-Knoten des Levels ℓ, so erhält man die Basisfunktion $\varphi_{\ell,i}^{p,\mathrm{CC}}$ analog zu (3.6) durch Verwendung der Knotenfolge

$$(x_{\ell,i-(p+1)/2}^{\mathrm{CC}}, \ldots, x_{\ell,i}^{\mathrm{CC}}, \ldots, x_{\ell,i+(p+1)/2}^{\mathrm{CC}}). \tag{3.23}$$

Dies geht freilich nur für $\frac{p+1}{2} \leqslant i \leqslant 2^\ell - \frac{p+1}{2}$. Für die übrigen i gibt es mehrere Möglichkeiten. Ein Weg wäre z. B., für $i \in \mathbb{N}$ die fehlenden, zu weit links liegenden Gitterpunkte zu definieren als $x_{\ell,-i}^{\mathrm{CC}} := 0 = x_{\ell,0}^{\mathrm{CC}}$ (analog $x_{\ell,2^\ell+i}^{\mathrm{CC}} := 1 = x_{\ell,2^\ell}^{\mathrm{CC}}$). In diesem Fall wären 0 und 1 vielfache Knoten, was durch die Definition 3.3 von nichtuniformen B-Splines abgedeckt ist. Allerdings widerspricht diese Behandlung dem uniformen Fall, bei dem die ersten und letzten B-Splines eigentlich einen Träger haben, der über $[0, 1]$ hinaus geht.

Um das Clenshaw-Curtis-Gitter so wie im uniformen Fall auch außerhalb $[0, 1]$ logisch fortzusetzen, definieren wir stattdessen

$$x_{\ell,-i}^{\mathrm{CC}} := -i \cdot x_{\ell,1}^{\mathrm{CC}}, \quad x_{\ell,2^\ell+i}^{\mathrm{CC}} := 1 + i \cdot (1 - x_{\ell,2^\ell-1}^{\mathrm{CC}}), \quad i \in \mathbb{N}. \tag{3.24}$$



(a) Mexican-Hat-Wavelet φ mit Wendepunkt und affin-linearer Fortsetzung φ^{lin}

(b) hierarchische Mexican-Hat-Räume W_ℓ

Abbildung 3.5: Mexican-Hat-Basisfunktionen

Wir setzen also das Gitter mit der ersten und der letzten Schrittweite $x^{\text{CC}}_{\ell,1}$ bzw. $1 - x^{\text{CC}}_{\ell,2^\ell-1}$ links von 0 bzw. rechts von 1 fort. Die Schrittweiten sind aufgrund der Symmetrie identisch.

3.1.5 Multivariate B-Splines

B-Spline-Basisfunktionen auf mehrdimensionalen dünnen Gittern werden wie bei den stückweise linearen Funktionen ($p = 1$) durch einen Tensorprodukt-Ansatz konstruiert, d. h.

$$\varphi^p_{\vec{\ell},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \varphi^p_{\ell_t,i_t}(x_t), \quad \vec{x} = (x_1, \ldots, x_d), \tag{3.25}$$

für den Level $\vec{\ell} = (\ell_1, \ldots, \ell_d) \in \mathbb{N}^d$ und den Index $\vec{i} = (i_1, \ldots, i_d) \in I_{\vec{\ell}}$. Die Konstruktion für nichtuniforme Gitter erfolgt analog.

3.2 Mexican-Hat-Wavelets

Obwohl wir uns auf B-Splines konzentrieren wollen, betrachten wir noch eine weitere Art von Basisfunktionen, nämlich die sogenannten Mexican-Hat-Basisfunktionen, die bereits in [40] für finanzmathematische Fragestellungen untersucht wurden. Allerdings betrachten wir für Mexican-Hat-Wavelets aus mehreren Gründen nur uniforme Gitter. Zum einen ist die Verallgemeinerung auf nichtuniforme Gitter möglich (z. B. durch geeignete Parametertransformationen), aber nicht trivial. Zum anderen sind B-Splines wie auch schon in [40] bemerkt für die meisten Anwendungen besser geeignet als die Mexican-Hat-Wavelets.

3.2.1 Univariate, hierarchische Wavelets

Definition 3.6 (Mexican-Hat-Wavelet)**.** Das Mexican-Hat-Wavelet ist definiert durch

$$\varphi\colon \mathbb{R} \to \mathbb{R}, \quad \varphi(x) := \left(1 - x^2\right) \mathrm{e}^{-x^2}. \tag{3.26}$$

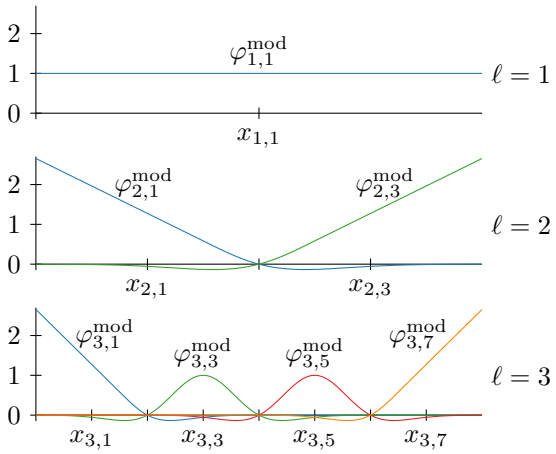


Abbildung 3.6: Modifizierte Mexican-Hat-Basisfunktionen bis zum Level 3

Durch affine Transformation erhält man die Mexican-Hat-Basisfunktion

$$\varphi_{\ell,i}(x) := \varphi(x/h_\ell - i) \tag{3.27}$$

mit Level $\ell \in \mathbb{N}$ und Index $i = 1, \ldots, 2^\ell - 1$.

Das Mexican-Hat-Wavelet, dessen Name sich von der sombreroähnlichen Kurvenform ableitet, ist laut [4] neben dem Haar-Wavelet eines der bekanntesten Wavelet-Funktionen, die in der Wavelet-Theorie verwendet werden. Die Definition (3.26) leitet sich nach [4] von der zweiten Ableitung $\left(x^2 - 1\right) \mathrm{e}^{-x^2/2}$ der gaußschen Standard-Glockenkurve $\mathrm{e}^{-x^2/2}$ her. In der Literatur unterscheiden sich die Definitionen des Mexican-Hat-Wavelets teilweise um das Vorzeichen oder den zusätzlichen Faktor im Exponenten. Wir wollen uns an die Notation von [40] halten, daher verwenden wir die in Abb. 3.5 illustrierte Definition (3.26).

Ein Problem wird anhand der Definition eminent: Die Basisfunktionen besitzen keinen lokalen Träger wie B-Splines, sondern ganz $[0, 1]$ als Träger. Dadurch wird die bei der Interpolation aufzustellende Matrix voll besetzt, was die schnelle und effiziente Lösung des zugehörigen linearen Gleichungssystems erschwert. Außerdem wird die Auswertung von Linearkombinationen von Mexican-Hat-Basisfunktionen deutlich verlangsamt. Bei der Implementierung wird daher das Wavelet $\varphi(x)$ für $|x| \geqslant 2$ abgeschnitten, womit jedoch die Glattheit verloren geht (siehe Abschnitt 6.2.3).

3.2.2 Multivariate und modifizierte Wavelets

Modifizierte Mexican-Hat-Basisfunktionen werden in [40] definiert, indem $\varphi_{\ell,1}(x)$ von $x = 0$ bis zu dem Rand abgewandten Wendepunkt x_{cut} nahe $x = h_\ell$ durch eine affin-lineare Funktion $\varphi^{\text{lin}}(x)$ so ersetzt wird, dass das Resultat zweifach stetig differenzierbar ist (analog mit $\varphi_{\ell,2^\ell-1}(x)$). Man bekommt für den Wendepunkt

$$x_{\text{cut}} = \frac{1}{2}\sqrt{7 - \sqrt{33}} \approx 0{,}560232 \tag{3.28}$$

und für die affin-lineare Funktion

$$\begin{aligned}\varphi^{\text{lin}}(x) &= \varphi'(x_{\text{cut}}) \cdot (x - x_{\text{cut}}) + \varphi(x_{\text{cut}}) \\ &= \left[\left(-\frac{\sqrt{33}+1}{4}\sqrt{7-\sqrt{33}}\right) x + (\sqrt{33}-4)\right] \cdot \mathrm{e}^{(\sqrt{33}-7)/4} \\ &\approx -1{,}380333x + 1{,}274615.\end{aligned} \tag{3.29}$$

Damit erhält man die folgenden, in Abb. 3.6 dargestellten modifizierten Basisfunktionen. Wie immer ergeben sich mehrdimensionale Basisfunktionen, egal, ob modifiziert oder nicht, als Tensorprodukt

$$\varphi_{\vec{\ell},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \varphi_{\ell_t,i_t}(x_t), \quad \vec{x} = (x_1, \ldots, x_d), \tag{3.30}$$

für den Level $\vec{\ell} = (\ell_1, \ldots, \ell_d) \in \mathbb{N}^d$ und den Index $\vec{i} = (i_1, \ldots, i_d) \in I_{\vec{\ell}}$.

Definition 3.7 (modifizierte Mexican-Hat-Basisfunktionen)**.**

$$\varphi_{\ell,i}^{p,\text{mod}}(x) := \begin{cases} 1, & \text{falls } \ell = 1,\ i = 1, \\ \varphi^{\text{lin}}(x/h_\ell - 1), & \text{falls } \ell > 1,\ i = 1,\ x \in [0, (1+x_{\text{cut}})h_\ell], \\ \varphi^{\text{lin}}((1-x)/h_\ell - 1), & \text{falls } \ell > 1,\ i = 2^\ell - 1,\ x \in [1-(1+x_{\text{cut}})h_\ell, 1], \\ \varphi_{\ell,i}(x), & \text{sonst}, \end{cases} \tag{3.31}$$

ist die modifizierte Mexican-Hat-Basisfunktion mit Level $\ell \in \mathbb{N}$ und Index $i \in I_\ell$.

4 Adaptive Gittererzeugung und Interpolation

Nachdem wir in den letzten beiden Kapiteln dünne Gitter und zugehörige Basisfunktionen definiert haben, möchten wir nun, wie in Kapitel 1 erklärt, eine gegebene Funktion $f\colon [0,1]^d \to \mathbb{R}$ global optimieren. Dazu erzeugen wir zunächst adaptiv ein dünnes Gitter: Die Feinheit des Gitters passt sich also mittels eines geeigneten Adaptivitätskriteriums an die Funktion an. Zum Beispiel kann man in der Nähe von Punkten mit sehr kleinen Funktionswerten weitere Punkte generieren, um so das evtl. dort vorhandene globale Minimum besser zu finden. Anschließend interpolieren wir f in diesen Gitterpunkten mittels der gewählten Basisfunktionen, um eine hinreichend glatte Interpolierende $\widetilde{f}$ zu erhalten. Auf $\widetilde{f}$ werden dann Optimierungsverfahren angewendet, die im nächsten Kapitel erklärt werden. Wir fokussieren uns bei der Untersuchung auf Noboundary-Gitter, ohne die Allgemeinheit einzuschränken. Die anderen Gitterarten kann man analog behandeln.

4.1 Adaptive Gittererzeugung

Zu Beginn betrachten wir die Verfeinerung eines einzelnen Gitterpunkts $\vec{x}_{\vec{\ell},\vec{j}} \in \mathbb{R}^d$ mit Level $\vec{\ell} \in \mathbb{N}^d$ und Index $\vec{j} \in I_{\vec{\ell}}$. Folgende Definition ist durch [16, 37] motiviert.

Definition 4.1 (Gitterpunkte-Nachbarn)**.** Seien $m \in \mathbb{N}$, $\vec{\ell}^* \in \mathbb{N}^d$ und $\vec{j}^* \in I_{\vec{\ell}^*}$. Dann heißt $\vec{x}_{\vec{\ell}^*,\vec{j}^*}$ „Nachbar m-ter Stufe“ von $\vec{x}_{\vec{\ell},\vec{j}}$, falls

$$(\ell_t^*, j_t^*) = \begin{cases} (\ell_s + m, 2^m j_s \pm 1), & \text{falls } t = s, \\ (\ell_t, j_t), & \text{sonst,} \end{cases} \qquad t = 1, \dots, d, \tag{4.1}$$

für ein $s \in \{1, \dots, d\}$. Nachbarn 1. Stufe heißen auch einfach „Nachbarn“.

In Abb. 4.1 sind beispielhaft Nachbarn 1. und 2. Stufe in zwei Dimensionen dargestellt. Von jeder Stufe gibt es in d Dimensionen stets genau $2d$ Nachbarn. Bei der adaptiven Gittererzeugung geht man von einem bestimmten Gitter als Grundlage aus (z. B. von einem regulären Gitter mit Level 3). Man erweitert das Gitter dann iterativ, indem man bestimmte Punkte „verfeinert“ und die feineren Punkte zum Gitter hinzufügt.

Definition 4.2 (Verfeinerung)**.** Die Berechnung aller Nachbarn (m-ter Stufe) eines Punktes $\vec{x}_{\vec{\ell},\vec{j}}$ heißt „Verfeinerung (m-ter Stufe)“ von $\vec{x}_{\vec{\ell},\vec{j}}$.

Für die Auswahl der Punkte, die verfeinert werden sollen, gibt es verschiedene Möglichkeiten, die im Folgenden vorgestellt werden. Die Auswahl hängt bei beiden vorgestellten Methoden von einem Adaptivitätsparameter $\alpha \in [0,1]$ ab. Für kleine α (oder gar $\alpha = 0$)

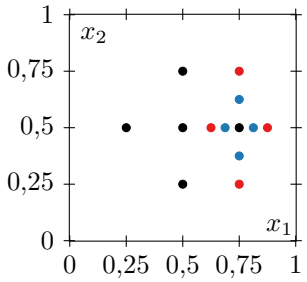


Abbildung 4.1: Reguläres randloses 2D-Gitter mit Level 2 und Nachbarn 1. und 2. Stufe von $(3/4, 1/2)$

hängt die Verfeinerung stark von der Zielfunktion ab, bei großen α (oder gar $\alpha = 1$) ist die Verfeinerung mehr oder weniger unabhängig von der Zielfunktion.

Wir betrachten ein Gitter als Menge $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ von Gitterpunkten $\vec{x}_i = \vec{x}_{\vec{\ell}_i, \vec{j}_i}$. Wird ein im Sinne der folgenden Definition verfeinerbarer Punkt verfeinert, dann werden die neuen Gitterpunkte als $\vec{x}_{n+1}$, $\vec{x}_{n+2}$ usw. in das Gitter eingefügt. Diese Definition ist besonders für die erste vorgestellte Verfeinerung mit hierarchischen Überschüssen relevant.

Definition 4.3 (verfeinerbar)**.** Ein Punkt $\vec{x}_i$ eines Gitters X heißt „(in m-ter Stufe) verfeinerbar“, falls sich mindestens ein Nachbar (m-ter Stufe) von $\vec{x}_i$ nicht bereits in X befindet.

4.1.1 Überschussbasierte Verfeinerung

Die überschussbasierte Verfeinerung basiert auf Formel (2.20). Sie besagt im Wesentlichen, dass genau die (linearen) hierarchischen Überschüsse betragsmäßig groß sind, auf deren zugehörigen Trägern $\mathrm{D}^{2\vec{e}} f$ betragsmäßig groß ist (wenn man davon ausgeht, dass der Träger so klein ist, dass $\mathrm{D}^{2\vec{e}} f$ dort nur ein Vorzeichen annimmt). Dies ist zum Beispiel bei starken Oszillationen von f der Fall, weswegen es zweckmäßig erscheint, die Punkte mit den betragsmäßig großen hierarchischen Überschüssen zu verfeinern, um f durch die stückweise lineare Interpolierende $\widetilde{f}$ möglichst genau darzustellen.

Ein auf diese Weise hergeleitetes Verfeinerungskriterium hat erst einmal nichts mit unserem Ziel der Minimierung zu tun, weil mögliche Minima von f gar nicht in das Kriterium eingehen. Jedoch hofft man natürlich, dass durch eine global genaue Interpolierende die Minima von f gut „getroffen“ werden. Dieses Verfeinerungskriterium ist also in dem Sinne vielseitig, dass die sich ergebende Interpolierende auch für andere Untersuchungen von f hilfreich sein kann.

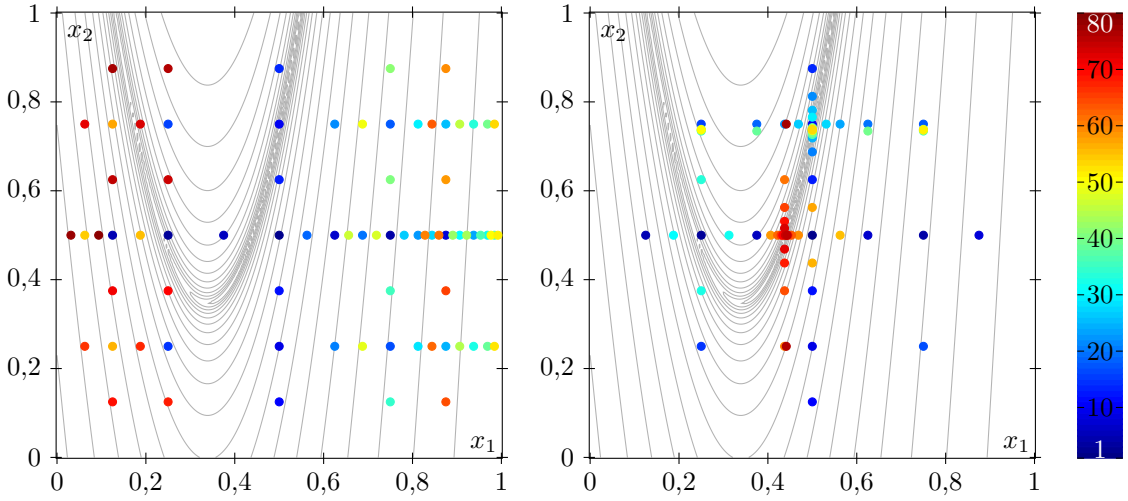
Eine Möglichkeit, wie man überschussbasiertes Verfeinern ausgehend von einem regulären dünnen Gitter mit Level 3 durchführen kann, ist als Pseudocode in Algorithmus 4.1 dargestellt. Neben der Zielfunktion f und der maximalen Größe N des zu erzeugenden Gitters wird auch ein Adaptivitätsparameter $\alpha \in [0, 1]$ benötigt. Er gibt den Anteil der in jedem Schritt zu verfeinernden Punkte an allen verfeinerbaren Punkten an: Für $\alpha = 0{,}1$ werden bspw. jedes Mal die 10 % der verfeinerbaren Punkte verfeinert, deren hierarchische Überschüsse betragsmäßig am größten sind. Sollte gegen Ende des Algorithmus die Größe des neuen Gitters die maximale Größe N überschreiten, so wird dieser Anteil iterativ halbiert, um das Maximum N möglichst gut „auszuschöpfen“.

Algorithmus 4.1: Adaptive Verfeinerung mit hierarchischen Überschüssen

Eingabe: Zielfunktion f, Adaptivitätsparameter $\alpha \in [0,1]$ und maximale Zahl $N \in \mathbb{N}$ an Gitterpunkten

Ausgabe: Gitter $X = \{\vec{x}_i\}_i$ und Funktionswerte $\vec{f} = (f_i)_i$ mit $f_i = f(\vec{x}_i)$

```
1  function (X, f⃗) = GenerateGridSurplus(f, α, N)
2      erzeuge reguläres Gitter X = {x⃗_i}_i mit Level 3
3      for all x⃗_i ∈ X do f_i ← f(x⃗_i)
4      (α_i)_i ← hierarchische Überschüsse für X, f⃗
5      β ← 1                                  ⇝ Faktor, falls zu viele Punkte erzeugt werden
6      for ever do
7          n ← Zahl der verfeinerbaren Punkte in X
8          n ← ⌈αβn⌉                          ⇝ Anzahl der zu verfeinernden Punkte
9          sortiere (α_i)_i, sodass |α_π(1)| ⩾ |α_π(2)| ⩾ ··· ⩾ |α_π(|X|)|   ⇝ π Permutation
10         verfeinere die ersten n verfeinerbaren Punkte in (x⃗_π(i))_i
11         X_neu ← {neue Punkte durch die Verfeinerung}\X
12         if |X| + |X_neu| > N then          ⇝ zu viele Punkte?
13             if n = 1 then return (X, f⃗)    ⇝ nur ein Punkt wurde verfeinert
14             else β ← β/2                   ⇝ probiere es nochmal mit halbem β
15         else
16             X ← X ∪̇ X_neu
17             for all x⃗_i ∈ X_neu do f_i ← f(x⃗_i)
18             (α_i)_i ← hierarchische Überschüsse für X, f⃗
```



(a) überschussbasiert ($\alpha = 0{,}1$, $N = 79$) **(b)** Ritter-Novak ($\alpha = 0{,}7$, $N = 77$)

Abbildung 4.2: Erzeugung eines adaptiven 2D-Gitters für die Rosenbrock-Funktion auf $[0,1]^2$ (siehe Anhang A). Die Farbe zeigt den Index i der Gitterpunkte $\vec{x}_i$: Blaue Punkte wurden zuerst in das Gitter eingefügt und rote Punkte zuletzt.

Algorithmus 4.2: Adaptive Verfeinerung nach Ritter und Novak

Eingabe: Zielfunktion f, Adaptivitätsparameter $\alpha \in [0,1]$ und maximale Zahl $N \in \mathbb{N}$ an Gitterpunkten
Ausgabe: Gitter $X = \{\vec{x}_i\}_i$ und Funktionswerte $\vec{f} = (f_i)_i$ mit $f_i = f(\vec{x}_i)$

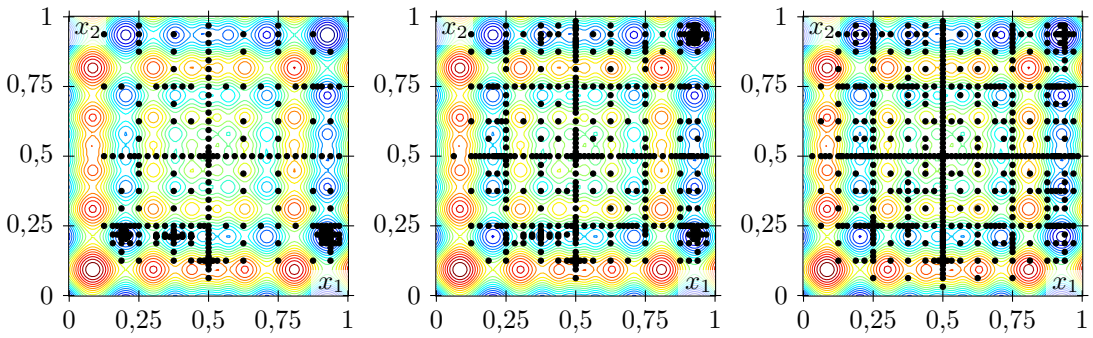
```
1  function (X, f⃗) = GenerateGridRitterNovak(f, α, N)
2      erzeuge reguläres Gitter X = {x⃗_i}_i mit Level 3
3      for all x⃗_i ∈ X do (f_i, d_i) ← (f(x⃗_i), 0)
4      for ever do
5          for all x⃗_i ∈ X do
6              r_i ← |{j | f_j ⩽ f_i}|                          ⇝ aktueller Rang von x⃗_i
7              β_i ← (‖ℓ⃗_i‖_1 + d_i + 1)^α · r_i^(1−α)          ⇝ Adaptivitätskriterium mit x⃗_i = x⃗_(ℓ⃗_i, j⃗_i)
8          i* ← arg min_i β_i
9          d_i* ← d_i* + 1
10         verfeinere x⃗_i*; existiert ein Nachbar bereits in X, dann berechne stattdessen den Nachbar 2. Stufe, existiert dieser ebenfalls, dann berechne den Nachbar 3. Stufe usw.
11         X_neu ← {neue Punkte durch die Verfeinerung}          ⇝ |X_neu| = 2d
12         if |X| + |X_neu| > N then return (X, f⃗)              ⇝ zu viele Punkte?
13         else
14             X ← X ∪̇ X_neu
15             for all x⃗_i ∈ X_neu do (f_i, d_i) ← (f(x⃗_i), 0)
```

Einen beispielhaften Verlauf für die auf $[0,1]^2$ skalierte, zweidimensionale Rosenbrock-Funktion sieht man in Abb. 4.2a. Dort zeigt die Farbe der Gitterpunkte die Reihenfolge ihrer Einfügung in das Gitter an. Der Knackpunkt des Algorithmus ist natürlich die Berechnung der hierarchischen Überschüsse. Naiv müsste man in jedem Durchlauf ein lineares Gleichungssystem (LGS) der Dimension $|X|$ lösen, was unter Umständen zeit- und speicheraufwendig ist. Später werden wir dieses Problem aber wesentlich besser angehen, sodass die Implementierung von Algorithmus 4.1 mit akzeptabler Zeit- und Speicherkomplexität möglich ist (siehe Abschnitt 4.2.1 und Abschnitt 6.2.4).

Anders sieht es aus, wenn man Adaptivität bzgl. der Zielbasis implementieren will: Wir verwenden B-Spline- oder Mexican-Hat-Basisfunktionen zur Interpolation. Da liegt es nahe, die hierarchischen Überschüsse in Algorithmus 4.1 gleich in diesen Basen zu berechnen. Natürlich ist das theoretisch möglich, aber nicht praxistauglich, da man hier nicht umhinkommt, ein LGS zu lösen, das im Allgemeinen nicht symmetrisch und voll besetzt ist (siehe Abschnitt 4.2.2). Daher wurde auf die Implementierung von Adaptivität bzgl. der Zielbasis verzichtet.

4.1.2 Ritter-Novak-Verfahren

Im Unterschied zur überschussbasierten Verfeinerung wollen wir jetzt ein Verfahren zur adaptiven Gittererzeugung betrachten, das einen deutlich größeren Wert auf die Funktionswerte selbst legt. Dieses Verfahren stammt von Novak und Ritter aus [37] (ursprünglich formuliert für sogenannte *hyperbolic cross points*, die Gitterpunkten von dünnen Gittern entsprechen) und wird durch Ferenczi in [16] ausführlich untersucht. Eine weitere Untersuchung findet sich in [29].



(a) $\alpha = 0{,}6$ **(b)** $\alpha = 0{,}8$ **(c)** $\alpha = 0{,}9$

Abbildung 4.3: Gittererzeugung nach Ritter/Novak für die 2D-Schwefel-Testfunktion auf $[0,1]^2$ (siehe Anhang A) mit Minimum bei $\vec{x}_{\text{opt}} \approx 0{,}92\vec{e}$

Das Verfahren ist in Algorithmus 4.2 dargestellt. Im Gegensatz zu Algorithmus 4.1 wird in jedem Schritt nur ein Punkt verfeinert und der Punkt kann später nochmals zur Verfeinerung ausgewählt werden. In diesem Fall (oder allgemeiner: falls bestimmte Nachbarn des ausgewählten Punktes bereits existieren) erhöht der Algorithmus so lange den Level neu einzufügender Punkte, bis diese eingefügt werden können. Auf diese Weise werden in jedem Schritt genau $2d$ neue Nachbarn (teilweise unterschiedlicher Stufen) erzeugt. Würde das Gitter die maximale Größe N durch die $2d$ neuen Punkte überschreiten, dann bricht der Algorithmus ab.

Um den in einer Runde zu verfeinernden Punkt auszuwählen, werden jedem Punkt $\vec{x}_i = \vec{x}_{\vec{\ell}_i,\vec{j}_i}$ des Gitters folgende Größen zugeordnet:

- $\|\vec{\ell}_i\|_1$ ist wie üblich die Summe der Levels von $\vec{x}_i$.
- d_i ist der „Grad“ von $\vec{x}_i$. Es handelt sich um die Anzahl, wie oft der Punkt bereits für eine Verfeinerung ausgewählt wurde (anfangs 0).
- r_i gibt den „Rang“ von $\vec{x}_i$ an: Sortiert man die $\vec{x}_i$ aufsteigend nach dem Funktionswert $f_i = f(\vec{x}_i)$ gemäß $f_{\pi(1)} \leqslant f_{\pi(2)} \leqslant \cdots \leqslant f_{\pi(|X|)}$, so ist $r_{\pi(i)} = i$ für alle i. Der Punkt $\vec{x}_i$ mit $r_i = 1$ ist also der mit dem kleinsten Funktionswert, der mit $r_i = 2$ ist der mit dem zweitkleinsten usw. Anders ausgedrückt gilt $r_i = |\{j \mid f_j \leqslant f_i\}|$.

Mit diesen Größen definieren wir wie Ritter/Novak in [37] die sogenannte Qualität des Gitterpunkts $\vec{x}_i$ durch

$$\beta_i := (\|\vec{\ell}_i\|_1 + d_i + 1)^\alpha \cdot r_i^{1-\alpha}. \tag{4.2}$$

Diese Formel enthält im Unterschied zu dem von Ritter und Novak vorgestellten Kriterium im ersten Faktor einen zusätzlichen Summanden 1. Der dient dazu, dass vermieden wird, dass sich die erste Klammer eventuell zu 0^0 ergibt, was für dünne Gitter mit Rand vorkommen kann. Algorithmus 4.2 wählt nun in jeder Runde den Gitterpunkt mit der „besten“, also kleinsten Qualität aus. Bei kleinem $\alpha \in [0,1]$ hat der erste Faktor kaum

Einfluss, sodass die Punkte mit kleinem Funktionswert (unter den Gitterpunkten) sehr häufig verfeinert werden. Der Nachteil ist dann natürlich, dass das Verfahren evtl. um ein nur lokales Minimum verfeinern könnte, welches kein globales Minimum ist. Bei großem $\alpha \in [0,1]$ verliert der zweite Faktor an Einfluss, sodass eher die Punkte verfeinert werden, die einen niedrigen Level besitzen und bisher kaum verfeinert wurden. Im Extremfall $\alpha = 1$ spielen der Rang und daher auch die Funktionswerte keine Rolle mehr, der Algorithmus betrachtet dann im Wesentlichen nur noch (partielle) reguläre dünne Gitter.

In Abb. 4.2b ist ein möglicher Verlauf für die skalierte Rosenbrock-Funktion in zwei Dimensionen dargestellt. Man kann deutlich sehen, dass in der Nähe des parabelförmigen „Tals“ und des Minimums bei (0,4; 0,4) öfters verfeinert wurde als anderswo, auch im Vergleich zur überschussbasierten Verfeinerung. Abb. 4.3 illustriert den Einfluss des Adaptivitätsparameters α auf das erzeugte Gitter für die Schwefel-Testfunktion mit $d = 2$ (siehe Anhang A). Dabei ist zu beachten, dass von den vielen lokalen Minima nur das Minimum oben rechts bei $0{,}92\vec{e}$ ein globales Minimum ist. Für $\alpha = 0{,}6$ wird in Abb. 4.3a dieses Minimum erst gar nicht gefunden, d. h. 0,6 ist hier als α ungeeignet. Andererseits ist $\alpha = 0{,}9$ (siehe Abb. 4.3c) etwas zu groß, weil nahe des Minimums nicht genügend oft verfeinert wurde. $\alpha = 0{,}8$ (Abb. 4.3b) scheint für diese Funktion am besten zu funktionieren. In dieser Arbeit (insb. in Kapitel 7) verwenden wir $\alpha = 0{,}85$ als Standard-Parameter, um Funktionen mit „schwierigen“ globalen Minima besser behandeln zu können.

4.2 Interpolation

Nun geht es um die Interpolation, genauer um die Aufgabe, zu gegebenen Gitterpunkten $\vec{x}_i = \vec{x}_{\vec{\ell}_i,\vec{j}_i} \in \mathbb{R}^d$ eines wie eben erzeugten Gitters und Funktionswerten $f_i = f(\vec{x}_i)$ (für $i = 1, \ldots, N$) eine Interpolierende

$$\widetilde{f}(\vec{x}) := \sum_{k=1}^{N} \alpha_k \varphi_k(\vec{x}), \quad \alpha_k \in \mathbb{R}, \tag{4.3}$$

zu finden mit $\varphi_k := \varphi_{\vec{\ell}_k,\vec{j}_k}$ und

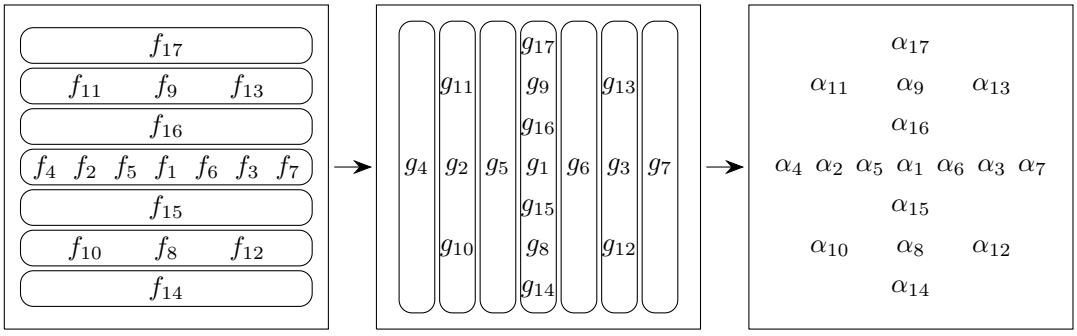
$$\widetilde{f}(\vec{x}_i) = f_i, \quad i = 1, \ldots, N. \tag{4.4}$$

Weil die α_k den hierarchischen Überschüssen aus (2.23) entsprechen, spricht man auch von „Hierarchisierung“. Entsprechend heißt der umgekehrte Vorgang, die Auswertung der Interpolierenden an einem beliebigen Punkt, „Dehierarchisierung“.

4.2.1 Hierarchisierung für stückweise lineare Basisfunktionen

Bei den stückweise linearen Basisfunktionen geschieht die Hierarchisierung in einer Dimension am einfachsten durch Anwendung von Lemma 2.5 (hierarchische 1D-Überschüsse). Die Berechnung kann je nach Implementierung der Gitterstruktur als Array oder als Baum iterativ für jeden Gitterpunkt bzw. rekursiv über alle Levels erfolgen.

Für mehrdimensionale Gitter wendet man das sogenannte unidirektionale Prinzip an, das in [2] eingeführt wurde (gute Erklärung siehe [40]). Dieses Prinzip entspricht allgemein



(a) zuerst in x_1-Richtung... **(b)** ... dann in x_2-Richtung **(c)** hierarchische Überschüsse

Abbildung 4.4: Unidirektionales Prinzip in zwei Dimensionen

gesagt der Anwendung eines eindimensionalen Operators (in diesem Fall die eindimensionale Hierarchisierung von eben) für alle eindimensionalen Teilgitter des Gitters in einer bestimmten Dimension. Dieser Vorgang wird für alle Dimensionen wiederholt.

Ein 2D-Beispiel mit einem regulären dünnen Gitter ist in Abb. 4.4 skizziert. Ausgehend von den Funktionswerten f_i in den Gitterpunkten $\vec{x}_i$ betrachtet man zeilenweise verschiedene eindimensionale dünne Teilgitter (Abb. 4.4a). In jeder Zeile berechnet man gemäß Lemma 2.5 Werte g_i (eindimensionale hierarchische Überschüsse). Dann wiederholt man das Verfahren für die zweite Dimension und die gerade berechneten Werte (Abb. 4.4b), um schließlich die hierarchischen Überschüsse zu erhalten (Abb. 4.4c). In zwei Dimensionen erfüllen diese die Formel (2.18).

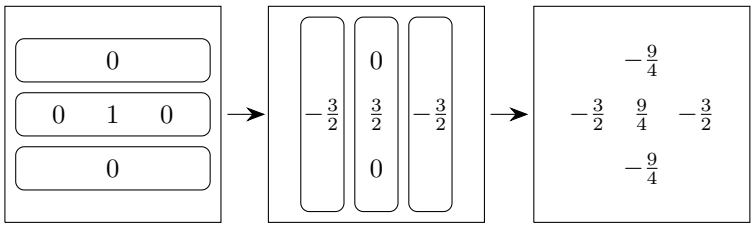
Das unidirektionale Prinzip berechnet die hierarchischen Überschüsse in $\mathcal{O}(d \cdot N)$ Zeit, also sehr schnell. Der Nachteil ist bei nichtregulären Gittern, dass jedes eindimensionale Teilgitter tatsächlich ein adaptiv verfeinertes dünnes 1D-Gitter sein muss. Wenn man das Gitter als Baum implementiert, dürfen also keine Vorfahren fehlen. In Abschnitt 6.1.3 werden wir noch einmal genauer darauf eingehen.

4.2.2 Hierarchisierung für B-Splines und Mexican-Hat-Wavelets

Leider verwenden wir keine stückweisen linearen Funktionen als Basisfunktionen für die Interpolierende, sondern B-Splines oder Mexican-Hat-Wavelets. Für diese Basisfunktionen lässt sich das unidirektionale Prinzip im Allgemeinen nicht anwenden. Dazu betrachten wir ein Beispiel (siehe Abb. 4.5): Gegeben sei das (randlose) reguläre dünne 2D-Gitter mit Level 2. Die $\vec{x}_i \in \mathbb{R}^2$ sind die Zeilen von X und die $f_i \in \mathbb{R}$ sind die Einträge von $\vec{f}$, wobei

$$X := \frac{1}{4}\begin{pmatrix} 2 & 1 & 3 & 2 & 2 \\ 2 & 2 & 2 & 1 & 3 \end{pmatrix}^{\mathrm{T}}, \quad \vec{f} := (1,0,0,0,0)^{\mathrm{T}}, \quad N := 5. \tag{4.5}$$

Es soll eine Interpolierende $\widetilde{f}(\vec{x})$ als Linearkombination von modifizierten B-Splines vom Grad 3 gefunden werden. Wir versuchen nun, das unidirektionale Prinzip mit den Bezeichnungen $\vec{f} \to \vec{g} \to \vec{\alpha}$ wie in Abb. 4.4 anzuwenden. Der Einfachheit halber bezeichnen wir die Punkte in der Mitte, links, rechts, unten, oben jeweils mit den Indizes $\mathrm{m}, \mathrm{l}, \mathrm{r}, \mathrm{u}, \mathrm{o}$. Be-



(a) $\vec{f} \to \vec{g}$ **(b)** $\vec{g} \to \vec{\alpha}$ **(c)** Resultat für $\vec{\alpha}$

Abbildung 4.5: Gegenbeispiel für die Nichtanwendbarkeit des unidirektionalen Prinzips bei modifizierten B-Splines vom Grad 3

trachtet man die Teilgitter in den Zeilen des dünnen Gitters (Abb. 4.5a), so stellt man wegen $f_\mathrm{u} = f_\mathrm{o} = 0$ direkt $g_\mathrm{u} = g_\mathrm{o} = 0$ fest. In der mittleren Zeile ergibt sich das LGS

$$\frac{1}{6}\begin{pmatrix} 6 & 1 & 1 \\ 6 & 6 & 0 \\ 6 & 0 & 6 \end{pmatrix}\begin{pmatrix} g_\mathrm{m} \\ g_\mathrm{l} \\ g_\mathrm{r} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \iff \begin{pmatrix} g_\mathrm{m} \\ g_\mathrm{l} \\ g_\mathrm{r} \end{pmatrix} = \frac{3}{2}\begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}. \tag{4.6}$$

Wendet man jetzt das unidirektionale Prinzip auf die Spalten des dünnen Gitters an (Abb. 4.5b), so folgt sofort $\alpha_\mathrm{l} = g_\mathrm{l} = -\frac{3}{2}$ und $\alpha_\mathrm{r} = g_\mathrm{r} = -\frac{3}{2}$, weil links und rechts nur die konstante Einsfunktion aktiv ist. In der mittleren Spalte ergibt sich ähnlich wie eben das LGS

$$\frac{1}{6}\begin{pmatrix} 6 & 1 & 1 \\ 6 & 6 & 0 \\ 6 & 0 & 6 \end{pmatrix}\begin{pmatrix} \alpha_\mathrm{m} \\ \alpha_\mathrm{u} \\ \alpha_\mathrm{o} \end{pmatrix} = \begin{pmatrix} 3/2 \\ 0 \\ 0 \end{pmatrix} \iff \begin{pmatrix} \alpha_\mathrm{m} \\ \alpha_\mathrm{u} \\ \alpha_\mathrm{o} \end{pmatrix} = \frac{9}{4}\begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}. \tag{4.7}$$

Damit bekommt man die Koeffizienten aus Abb. 4.5c heraus. Diese können aber wegen der Asymmetrie nicht gleich den richtigen hierarchischen Überschüssen sein. In der Tat sind die korrekten Koeffizienten gegeben durch $\vec{\alpha} = (\alpha_\mathrm{m}, \alpha_\mathrm{l}, \alpha_\mathrm{r}, \alpha_\mathrm{u}, \alpha_\mathrm{o})^\mathrm{T} = (2, -\frac{3}{2}, -\frac{3}{2}, -\frac{3}{2}, -\frac{3}{2})^\mathrm{T}$.

Das unidirektionale Prinzip ist also bei B-Splines (für den Grad $p > 1$) und analog auch bei Mexican-Hat-Wavelets im Allgemeinen nicht anwendbar. Daher bleibt für die Bestimmung der hierarchischen Überschüsse nur der Weg, das lineare Interpolations-Gleichungssystem aus (4.4), also

$$\widetilde{f}(\vec{x}_i) = \sum_{k=1}^{N} \alpha_k \varphi_k(\vec{x}_i) = f_i, \quad i = 1, \ldots, N, \tag{4.8}$$

direkt zu lösen. Hierbei tauchen mehrere Probleme auf: Das LGS ist i. A. nicht symmetrisch, was die Auswahl effizienter numerischer Löser einschränkt. Außerdem ist das LGS nur für B-Splines und niedrige Dimensionszahlen dünn besetzt. Mexican-Hat-Funktionen führen wegen des globalen Trägers (ohne „Abschneiden“) zu einer voll besetzten Matrix und für großes d enthalten die Träger der Basisfunktionen φ_k viel mehr Gitterpunkte $\vec{x}_i$, weswegen viele Einträge $\varphi_k(\vec{x}_i)$ der LGS-Matrix ungleich null sind. Bei der Evaluierung werden wir aber sehen, dass der entstehende zusätzliche Zeit-/Speicheraufwand verkraftbar ist.

5 Übersicht über Optimierungsverfahren

In diesem Kapitel stellen wir einige gängige gradientenbasierte und gradientenfreie Optimierungsverfahren vor. Die zu optimierende Funktion bezeichnen wir hier der Einfachheit halber mit $f\colon [0,1]^d \to \mathbb{R}$. Konkret kann es sich bei ihr um die Zielfunktion (bisher mit f benannt) oder um eine Dünngitter-Interpolierende (bisher mit $\widetilde{f}$ bezeichnet) handeln. Die Interpolierende aus B-Spline- oder Mexican-Hat-Basisfunktionen ist hinreichend glatt, um gradientenbasierte Optimierungsverfahren durchzuführen. Diese arbeiten lokal und bekommen einen Startpunkt $\vec{x}_0$, in dessen Nähe sie nach Minima suchen. Alle anderen Verfahren sind gradientenfrei und größtenteils global, das heißt, sie versuchen, globale Minima zu finden. Sie werden später sowohl zur Optimierung der Interpolierenden als auch zum Vergleich zur „direkten“ Optimierung der Zielfunktion eingesetzt (siehe Abschnitt 6.3).

5.1 Gradientenbasierte Optimierung

Alle in diesem Abschnitt vorgestellten Verfahren sind sogenannte Abstiegsverfahren. Sie gehen also von einem Startpunkt $\vec{x}_0 \in [0,1]^d$ aus und erzeugen dann eine Folge von Punkten $\vec{x}_k \in [0,1]^d$ mit immer kleiner werdenden Funktionswerten. Setzt man zu Beginn $k \leftarrow 1$, dann haben die Verfahren nach [51] allgemein folgende Form:

1. Wenn $\|\nabla f(\vec{x}_{k-1})\|_2$ klein ist, dann beende mit dem Ergebnis $\vec{x}_{k-1}$.
2. Wähle eine normierte Suchrichtung $\vec{s} \in \mathbb{R}^d$ mit $\nabla f(\vec{x}_{k-1}) \cdot \vec{s} < 0$ (Abstiegsrichtung).
3. Bestimme $\sigma > 0$ mit $f(\vec{x}_{k-1} + \sigma\vec{s})$ minimal.
4. Setze $\vec{x}_k \leftarrow \vec{x}_{k-1} + \sigma\vec{s}$, $k \leftarrow k+1$ und gehe zu 1.

Die nichttrivialen Schritte sind die Bestimmungen von $\vec{s}$ und σ. Für die folgenden Verfahren nutzen wir für die Berechnung von σ jeweils die Armijo-Schrittweitenregel aus [51], die in Algorithmus 5.1 dargestellt ist. Die Verfahren unterscheiden sich dann nur noch in der Wahl von $\vec{s}$. Es wurden auch andere Schrittweitenregeln ausprobiert, im Vergleich überzeugte jedoch die Armijo-Regel durch Einfachheit, Schnelligkeit und Robustheit.

Die Armijo-Regel prüft, ob mit $\sigma = \beta^\ell$ (für $\ell = 0, 1, \ldots$ und $\beta \in (0,1)$ konstant) die Funktionswert-Verbesserung $f(\vec{x}_{k-1}) - f(\vec{x}_k)$ des neuen Punkts $\vec{x}_k = \vec{x}_{k-1} + \sigma\vec{s}$ mindestens gleich $\gamma\sigma(-\nabla f(\vec{x}_{k-1}) \cdot \vec{s})$ ist ($\gamma \in (0,1)$ konstant). In [51] wird gezeigt, dass das unter einfachen Voraussetzungen für $\sigma > 0$ klein genug automatisch erfüllt ist. Die hier vorgestellte Version implementiert außerdem noch ein relatives Abbruchkriterium nach [43]: Ist die durch den Armijo-Schritt erzielte Verbesserung des Funktionswerts zu klein, dann wird durch die Ausgabe von `null` der Abbruch des Abstiegsverfahrens empfohlen.

Algorithmus 5.1: Minimierung entlang einer Halbgeraden mittels der Armijo-Regel

Eingabe: zu optimierende Funktion f, „alter“ Punkt $\vec{x}_{k-1} \in [0,1]^d$, normierte Suchrichtung $\vec{s} \in \mathbb{R}^d$, Toleranzen $\varepsilon_1, \varepsilon_2 > 0$ (Standard: 1E-8 bzw. 1E-18), Verkleinerungsfaktor $\beta \in (0,1)$ (Standard: 0,5), Akzeptanzfaktor $\gamma \in (0,1)$ (Standard: 1E-2) und maximale Schrittzahl $m \in \mathbb{N}$ (Standard: 100)

Ausgabe: „neuer“ Punkt $\vec{x}_k = \vec{x}_{k-1} + \sigma\vec{s}$ mit $\sigma \in (0,1)$ und $f(\vec{x}_k)$ „klein“; $\vec{x}_k =$ **`null`**, falls kein solcher Punkt gefunden werden konnte (auch für zu kleine Verbesserung $f(\vec{x}_{k-1}) - f(\vec{x}_k)$)

```
1 function x_k = LineSearch(f, x_{k-1}, s, ε1, ε2, β, γ, m)
2     q ← −∇f(x_{k-1}) · s                  ⇝ Skalarprodukt negierter Gradient und Suchrichtung
3     for ℓ = 1, ..., m do
4         σ ← β^{ℓ−1}
5         x_k ← x_{k-1} + σs                ⇝ berechne Punkt auf Halbgeraden
6         if (x_k ∈ [0,1]^d) ∧ (f(x_{k-1}) − f(x_k) ⩾ γσq) then   ⇝ Armijo-Bedingung
7             if |f(x_{k-1}) − f(x_k)| ⩾ ε1(|f(x_{k-1})| + |f(x_k)| + ε2) then return x_k
8             else return null   ⇝ Verbesserung zu klein, Abbruch des Verfahrens empfohlen
9     return null                ⇝ Fehler: hinreichend kleiner Funktionswert nicht gefunden
```

Algorithmus 5.2: Gradientenverfahren

Eingabe: zu optimierende Funktion f, Startpunkt $\vec{x}_0 \in [0,1]^d$, maximale Iterationszahl $N \in \mathbb{N}$ (Standard: 200) und Toleranz $\varepsilon > 0$ (Standard: 1E-8)

Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

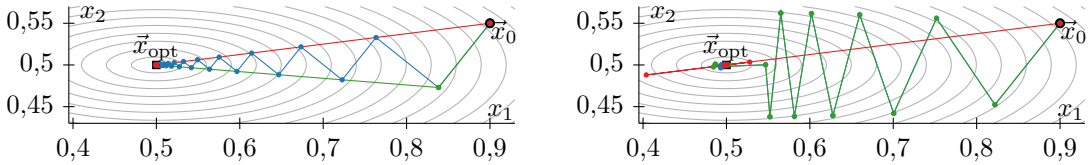
```
1 function x_opt = GradientMethod(f, x_0, N, ε)
2     for k = 1, ..., N do
3         if ‖∇f(x_{k-1})‖_2 < ε then return x_{k-1}   ⇝ Gradient hinreichend nahe bei 0⃗?
4         s ← −∇f(x_{k-1})/‖∇f(x_{k-1})‖_2             ⇝ normierte Suchrichtung
5         x_k ← LineSearch(f, x_{k-1}, s)
6         if x_k = null then return x_{k-1}            ⇝ Armijo-Regel fehlgeschlagen
7     return x_N
```

5.1.1 Gradientenverfahren

Das Gradientenverfahren (siehe Algorithmus 5.2) ist das denkbar einfachste Abstiegsverfahren. Als Suchrichtung $\vec{s}$ wird einfach der normierte negierte Gradient verwendet. Die Normierung erfolgt aufgrund der Beschränktheit des Definitionsbereichs $[0,1]^d$ – für große Gradienten müsste man sonst ein sehr kleines $\sigma > 0$ wählen, damit überhaupt erst $(\vec{x}_k + \sigma\vec{s}) \in [0,1]^d$ gilt.

In gewissen Fällen zeigt das Gradientenverfahren eine sehr langsame Konvergenz von $\vec{x}_k$ gegen $\vec{x}_{\text{opt}}$. Ist f zweifach stetig differenzierbar, die Hesse-Matrix $H_f(\vec{x}_{\text{opt}})$ im Minimum positiv definit (weil sie im Minimum notwendigerweise positiv semidefinit ist, reicht es, dass 0 kein Eigenwert ist) und ihre Konditionszahl

$$\kappa := \kappa(H_f(\vec{x}_{\text{opt}})) = \frac{\lambda_{\max}(H_f(\vec{x}_{\text{opt}}))}{\lambda_{\min}(H_f(\vec{x}_{\text{opt}}))} \tag{5.1}$$



(a) optimale Schrittweite **(b)** Armijo-Schrittweite

Abbildung 5.1: Anwendung von Gradienten-, NLCG- und Newton-Verfahren auf $f(x_1, x_2) := (x_1 - 0{,}5)^2 + \lambda(x_2 - 0{,}5)^2$ mit $\lambda := 10$. Die blaue Linie verläuft rechts anfangs genau unter der grünen, jedoch konvergiert NLCG nach 24 Schritten mit Abstand 1,2E-6 (dann erst 4,8E-4 beim Gradientenverfahren).

groß, dann verläuft das Gradientenverfahren im Zickzack und nähert sich nur langsam $\vec{x}_{\text{opt}}$ (siehe blaue Linie in Abb. 5.1a): Für streng konvexe, quadratische Funktionen gilt bei optimaler Schrittweite nämlich

$$f(\vec{x}_k) - f(\vec{x}_{\text{opt}}) \leqslant \underbrace{\left(\frac{\kappa - 1}{\kappa + 1}\right)^2}_{\approx 1} (f(\vec{x}_{k-1}) - f(\vec{x}_{\text{opt}})). \tag{5.2}$$

Ein Beweis findet sich z. B. in [51]. Die Abschätzung ist für die Funktion f aus Abb. 5.1 scharf (Rechnung ebenfalls siehe [51]), wodurch sich eine schlechte Konvergenzrate ergibt.

5.1.2 Nichtlineares CG-Verfahren (NLCG-Verfahren)

Eine Abhilfe ist das nichtlineare CG-Verfahren (NLCG-Verfahren), das dem CG-Verfahren zur Lösung symmetrischer, positiv-definiter linearer Gleichungssysteme $A\vec{x} = \vec{b}$ entspricht, wenn man die streng konvexe, quadratische Funktion $f(\vec{x}) := \frac{1}{2}\vec{x}^{\mathrm{T}} A\vec{x} - \vec{b}^{\mathrm{T}}\vec{x}$ minimiert (siehe [44]). Dabei korrespondiert das Residuum $\vec{r}_k := A\vec{x}_k - \vec{b}$ mit dem Gradienten $\nabla f(\vec{x})$. Der einzige wesentliche Unterschied ist, dass die Schrittweite σ beim linearen CG-Verfahren direkt berechnet werden kann, wogegen das NLCG-Verfahren die Schrittweite z. B. durch die Armijo-Regel annähern muss.

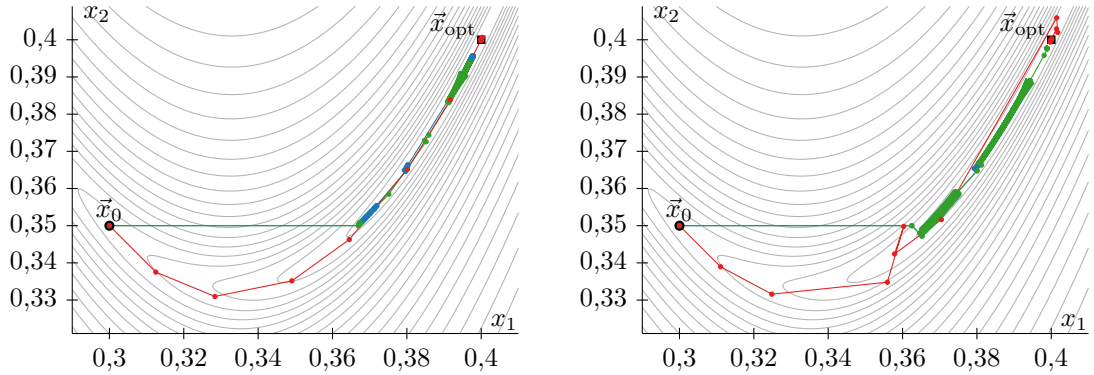
Der Vorteil des NLCG-Verfahrens ist, dass es bei exakter Rechnung und optimaler Schrittweite für streng konvexe, quadratische Funktionen nach höchstens d Iterationen die Minimalstelle genau findet. In Abb. 5.1a ist das bspw. bereits nach zwei Schritten der Fall (grüne Linie). Weil jede um $\vec{x}_{\text{opt}}$ dreifach stetig differenzierbare Funktion mit positiv definiter Hesse-Matrix nach dem Satz von Taylor in der Nähe von $\vec{x}_{\text{opt}}$ einer solchen Funktion ähnelt, findet das NLCG-Verfahren auch bei nichtquadratischen Funktionen das Minimum i. A. schneller als das Gradientenverfahren.

In Algorithmus 5.3 ist das NLCG-Verfahren in der Version von [44] dargestellt. Bei der Parameterwahl von β existieren verschiedene Varianten. Populärer ist die Wahl nach Polak-Ribière wie im Algorithmus, die alternative Fletcher-Reeves-Wahl ist im Kommentar angegeben. Ab und zu kann es sinnvoll sein, die Suchrichtung zurückzusetzen („Neustart“ des Verfahrens), weil die Konvergenz nach [42] sonst langsam sein kann. Im Unterschied zur Darstellung in [44] enthält Algorithmus 5.3 daher ein Neustart-Kriterium nach [42].

Algorithmus 5.3: Nichtlineares CG-Verfahren nach Polak-Ribière

Eingabe: zu optimierende Funktion f, Startpunkt $\vec{x}_0 \in [0,1]^d$, maximale Iterationszahl $N \in \mathbb{N}$ (Standard: 200), Toleranz $\varepsilon > 0$ (Standard: 1E-8) und Neustart-Schwelle $\alpha \in (0,1)$ (Standard: 0,1)
Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

```
1  function x⃗_opt = NonlinearConjugateGradient(f, x⃗_0, N, ε, α)
2      (d⃗_0, s⃗) ← (∇f(x⃗_0), −∇f(x⃗_0))                    ⇝ erster Gradient und Suchrichtung
3      for k = 1, ..., N do
4          if ‖d⃗_{k−1}‖_2 < ε then return x⃗_{k−1}          ⇝ Gradient hinreichend nahe bei 0⃗?
5          x⃗_k ← LineSearch(f, x⃗_{k−1}, s⃗/‖s⃗‖_2)
6          if x⃗_k = null then return x⃗_{k−1}               ⇝ Armijo-Regel fehlgeschlagen
7          d⃗_k ← ∇f(x⃗_k)
8          if |d⃗_k · d⃗_{k−1}|/‖d⃗_k‖_2^2 ⩾ α then β ← 0     ⇝ Neustart, Gradientenverfahren-Schritt
9          else β ← (d⃗_k · (d⃗_k − d⃗_{k−1}))/‖d⃗_{k−1}‖_2^2   ⇝ für Fletcher-Reeves: β ← ‖d⃗_k‖_2^2/‖d⃗_{k−1}‖_2^2
10         s⃗ ← βs⃗ − d⃗_k
11     return x⃗_N
```



(a) optimale Schrittweite **(b)** Armijo-Schrittweite

Abbildung 5.2: Vergleich von Gradienten-, NLCG- und Newton-Verfahren für die Rosenbrock-Funktion auf $[0,1]^2$ (siehe Anhang A). Die blaue Linie verläuft rechts anfangs wieder genau unter der grünen, allerdings liegt das NLCG-Verfahren nach 1000 Iterationen mit Abstand 2,3E-3 näher am Optimum als das Gradientenverfahren mit Abstand 1,1E-2.

5.1.3 Newton-Verfahren

Bei bestimmten Funktionen wie der Rosenbrock-Funktion in Abb. 5.2a konvergieren sowohl Gradienten- als auch NLCG-Verfahren langsam, weil die Hesse-Matrix $H_f(\vec{x}_{\text{opt}})$ mit einer Konditionszahl von $\kappa \approx 2508$ schlecht konditioniert ist. Ein Standard-Newton-Verfahren (rote Linie in Abb. 5.2a) kann deutlich besser abschneiden. Ein Nachteil ist aber, dass in jedem Schritt ein lineares Gleichungssystem (LGS) mit Matrix $H_f(\vec{x}_{k-1})$ berechnet werden muss – insbesondere muss $H_f(\vec{x}_{k-1})$ bekannt und leicht berechenbar sein. Im Wesentlichen darf die Hesse-Matrix hier also ebenfalls nicht zu schlecht konditioniert sein.

Algorithmus 5.4: Newton-Verfahren

Eingabe: zu optimierende Funktion f, Startpunkt $\vec{x}_0 \in [0,1]^d$, maximale Iterationszahl $N \in \mathbb{N}$ (Standard: 200), Toleranz $\varepsilon > 0$ (Standard: 1E-8), Faktoren $\alpha_1, \alpha_2 > 0$ (Standard: jeweils 1E-6) und Exponent $p > 0$ (Standard: 0,1)
Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

```
1 function x_opt = NewtonMethod(f, x_0, N, ε, α_1, α_2, p)
2     for k = 1, ..., N do
3         d_{k-1} ← ∇f(x_{k-1})
4         if ‖d_{k-1}‖_2 < ε then return x_{k-1}        ⇝ Gradient hinreichend nahe bei 0?
5         löse LGS H_f(x_{k-1}) · s = −d_{k-1}          ⇝ H_f Hesse-Matrix von f
6         if (Fehler beim Lösen) ∨ (−d_{k-1} · s < min(α_1, α_2‖d_{k-1}‖_2^p)‖d_{k-1}‖_2^2) then s ← −d_{k-1}
7         x_k ← LineSearch(f, x_{k-1}, s/‖s‖_2)
8         if x_k = null then return x_{k-1}             ⇝ Armijo-Regel fehlgeschlagen
9     return x_N
```

Algorithmus 5.4 zeigt das sogenannte globalisierte Newton-Verfahren aus [51]. Die Grundidee des Newton-Verfahrens ist die Taylor-Approximation zweiter Ordnung

$$f(\vec{x}_{k-1} + \vec{s}) \approx p(\vec{s}) := f(\vec{x}_{k-1}) + \vec{s} \cdot \nabla f(\vec{x}_{k-1}) + \frac{1}{2}\vec{s}^{\mathrm{T}} H_f(\vec{x}_{k-1})\vec{s}. \tag{5.3}$$

Durch Berechnung des kritischen Punkts von p erhält man die Suchrichtung $\vec{s}$ mit

$$\vec{0} = \nabla_{\vec{s}}\, p(\vec{s}) = \nabla f(\vec{x}_{k-1}) + H_f(\vec{x}_{k-1})\vec{s} \iff H_f(\vec{x}_{k-1})\vec{s} = -\nabla f(\vec{x}_{k-1}), \tag{5.4}$$

d. h. als Lösung eines LGS mit Matrix $H_f(\vec{x}_{k-1})$ und rechter Seite $-\nabla f(\vec{x}_{k-1})$. Algorithmus 5.4 prüft dann noch mit einer Winkelbedingung aus [51], ob $\vec{s}$ eine hinreichend gute Abstiegsrichtung ist – ist das nicht der Fall, wird der negierte Gradient als Suchrichtung verwendet. Weil die Approximation (5.3) für quadratische Funktionen exakt ist, konvergiert das Newton-Verfahren in Abb. 5.1a bei exakter Schrittweite schon in einem Schritt.

5.2 Gradientenfreie Optimierung

5.2.1 Nelder-Mead-Verfahren

Das Nelder-Mead-Verfahren (kurz NM-Verfahren) wurde zuerst von Nelder und Mead in [36] beschrieben. Algorithmus 5.5 beschreibt größtenteils die Version aus [17] (aufgrund Verschiedenheiten in der Literatur aber mit einer Abweichung bei der Bestimmung von $\vec{y}_4$). Es erzeugt anfangs einen vom Startpunkt $\vec{x}_0$ abhängenden Startsimplex mit Ecken $\vec{x}_0, \ldots, \vec{x}_d \in [0,1]^d$, wobei die Punkte immer aufsteigend nach ihrem Funktionswert sortiert sind. Anschließend wird in jeder Iteration jeweils eine der Operationen Reflexion, Expansion, (innere/äußere) Kontraktion und Schrumpfung auf die Ecken angewendet. Dabei kann es vorkommen, dass der Simplex am Anfang kurzzeitig über den Definitionsbereich $[0,1]^d$ hinausgeht. Daher setzen wir f zu $\widehat{f}\colon \mathbb{R}^d \to \mathbb{R}$ mit $\widehat{f}(\vec{x}) := +\infty$ für $\vec{x} \notin [0,1]^d$ fort, sodass $\widehat{f}(\vec{x}_i)$ stets wohldefiniert ist.

Algorithmus 5.5: Nelder-Mead-Verfahren

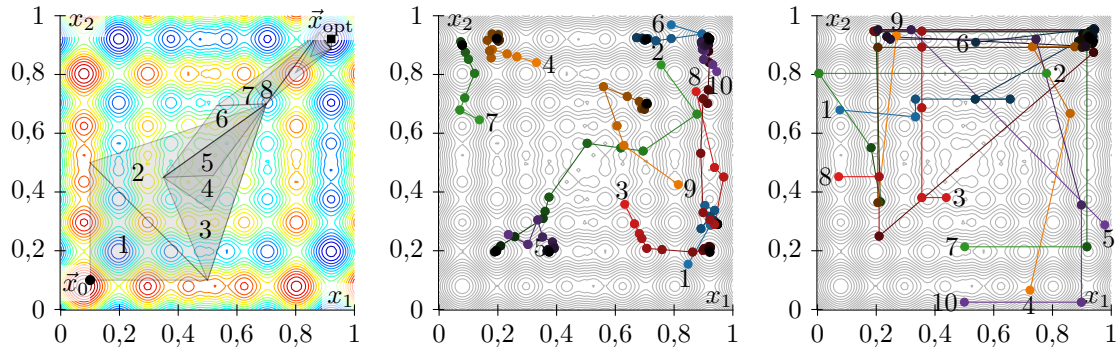
Eingabe: zu optimierende Funktion f, Startpunkt $\vec{x}_0 \in [0,1]^d$, maximale Auswertungszahl $N \in \mathbb{N}$ (Standard: 1000), Reflexionskoeffizient $\alpha > 0$ (Standard: 1), Expansionskoeffizient $\beta > 1$ (Standard: 2), Kontraktionskoeffizient $\gamma \in (0,1)$ (Standard: 0,5), Schrumpfkoeffizient $\delta \in (0,1)$ (Standard: 0,5) und Startsimplex-Kantenlänge $\ell_0 \in (0,1)$ (Standard: 0,4)
Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

```
1  function x_opt = NelderMead(f, x_0, N, α, β, γ, δ, ℓ_0)
2      for t = 1, ..., d do x_t ← x_0 + min(ℓ_0, 1 − x_{0,t}) e_t   ⇝ x_0 = (x_{0,t})_t, e_t t-ter Einheitsvektor
3      sortiere x_0, ..., x_d, sodass f(x_0) ⩽ ··· ⩽ f(x_d)
4      k ← d + 1                                                    ⇝ Anzahl der Auswertungen von f
5      while k + (d + 2) ⩽ N do
6          y_0 ← Σ_{t=0}^{d−1} x_t / d
7          (y_1, k) ← (y_0 + α(y_0 − x_d), k + 1)                   ⇝ reflektierter Punkt
8          if f̂(y_1) ∈ [f̂(x_0), f̂(x_{d−1})) then x_d ← y_1          ⇝ mit f̂ wie im Text erläutert
9          else if f̂(y_1) < f̂(x_0) then
10             (y_2, k) ← (y_0 + β(y_1 − y_0), k + 1)               ⇝ expandierter Punkt
11             if f̂(y_2) < f̂(y_1) then x_d ← y_2
12             else x_d ← y_1
13         else if f̂(y_1) < f̂(x_d) then                            ⇝ es gilt f̂(y_1) ∈ [f̂(x_{d−1}), f̂(x_d))
14             (y_3, k) ← (y_0 + γ(y_1 − y_0), k + 1)               ⇝ äußerer kontrahierter Punkt
15             if f̂(y_3) ⩽ f̂(y_1) then x_d ← y_3
16             else go to Zeile 21
17         else                                                     ⇝ es gilt f̂(y_1) ⩾ f̂(x_d)
18             (y_4, k) ← (y_0 − γ(y_0 − x_d), k + 1)               ⇝ innerer kontrahierter Punkt
19             if f̂(y_4) < f̂(x_d) then x_d ← y_4
20             else
21                 for t = 1, ..., d do (x_t, k) ← (x_0 + δ(x_t − x_0), k + 1)   ⇝ geschrumpfter Punkt
22         sortiere x_0, ..., x_d, sodass f̂(x_0) ⩽ ··· ⩽ f̂(x_d)
23     return x_0
```

Das Verfahren bricht in der in Algorithmus 5.5 vorgestellten Variante ab, wenn N Funktionsauswertungen erreicht werden. Natürlich gibt es auch (hier nicht implementierte) Kriterien, die durch Vergleich der Funktionswerte oder der Simplex-Kantenlängen einen vorzeitigen Abbruch herbeiführen. Das NM-Verfahren schneidet für kleine Dimensionen gut ab, obwohl in [17] erwähnt wird, dass es zweidimensionale Gegenbeispiele gibt, bei denen das Verfahren falsch konvergiert. Außerdem findet das NM-Verfahren meist nur lokale Minima. Ein möglicher Verlauf für die Schwefel-Testfunktion in zwei Dimensionen ist mit grau eingezeichneten Simplizes $\triangle\vec{x}_0\vec{x}_1\vec{x}_2$ in Abb. 5.3a dargestellt.

5.2.2 Zufällige Suche

Eine einfache Möglichkeit, Verfahren wie das NM-Verfahren zu globalisieren, besteht im Rahmen der „zufälligen Suche“ (engl. *random search*). Das Prinzip ist auf jedes lokale Optimierungsverfahren anwendbar und besteht in der m-maligen Anwendung des lokalen Verfahrens für zufällig ausgewählte Startpunkte, um m lokale Minimalstellen $\vec{x}_1, \ldots, \vec{x}_m$ zu erhalten. Der Punkt mit dem kleinsten Funktionswert „gewinnt“.



(a) NM-Verfahren mit nummerierten Simplizes ($N = 40$)

(b) zufällige Suche mit $m = 10$ Startpunkten ($N = 300$)

(c) DE-Verfahren mit Populationsgröße $m = 10$ ($N = 300$)

Abbildung 5.3: Visualisierung der gradientenfreien Verfahren für die Schwefel-Funktion auf $[0, 1]^2$ (siehe Anhang A)

Algorithmus 5.6: Zufällige Suche mit dem Nelder-Mead-Verfahren

Eingabe: zu optimierende Funktion f, maximale Auswertungszahl $N \in \mathbb{N}$ (Standard: 1000) und Anzahl der Nelder-Mead-Aufrufe $m \in \mathbb{N}$, $m \leqslant N$ (Standard: $\min(10d, 100)$)
Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

```
1 function x_opt = RandomSearch(f, N, m)
2     for k = 1, ..., m do
3         n ← ⌈N/(m − k + 1)⌉          ⇝ teile Auswertungszahl N gerecht auf
4         N ← N − n
5         wähle x_0 ∈ [0,1]^d gleichverteilt zufällig          ⇝ Startpunkt
6         x_k ← NelderMead(f, x_0, n)
7     k* ← arg min_{k=1,...,m} f(x_k)
8     return x_k*
```

Algorithmus 5.6 verwendet das NM-Verfahren zur lokalen Suche. Die Startpunkte werden gleichverteilt gewählt – bei Kenntnissen über f, die a priori vorhanden sind, könnte auch eine andere Verteilung gewählt werden. Die Standardwahl von m erfolgt wie in [56]. Wie die zufällige Suche bei der Schwefel-Testfunktion in zwei Dimensionen abschneidet, sieht man beispielhaft in Abb. 5.3b.

5.2.3 Differentialevolution

Das Verfahren der Differentialevolution (DE-Verfahren) ist ebenfalls ein globales Optimierungsverfahren und gehört zur Klasse der sogenannten evolutionären Algorithmen. Es wurde von Storn und Price in [50] formuliert und verwaltet eine „Population“ von m Punkten $\vec{x}_1, \ldots, \vec{x}_m \in [0, 1]^d$, die zu Anfang gleichverteilt zufällig gewählt werden. In jeder Iteration ergeben sich die m neuen Punkte aus den alten komponentenweise durch zufällige „Mutation“. Man kann von „Kreuzung“ der alten Punkte mit den jeweiligen mutierten Punkten sprechen. Die alten Punkte werden nur ersetzt, wenn der neue Funktionswert besser als der alte ist („Evolution“).

Algorithmus 5.7: Differentialevolution

Eingabe: zu optimierende Funktion f, maximale Auswertungszahl $N \in \mathbb{N}$ (Standard: 1000), Populationsgröße $m \in \mathbb{N}$, $m \leqslant N$ (Standard: $10d$), Kreuzungswahrscheinlichkeit $p \in [0,1]$ (Standard: 0,5), Skalierungsfaktor $\alpha \in [0,2]$ (Standard: 0,6), Wert $\beta \geqslant 0$, um den sich der Durchschnitt einer Generation verbessern sollte (Standard: 1E-6), Abstand $\gamma > 0$ für Abbruchbedingung (Standard: 1E-4) und Zahl $n \in \mathbb{N}$ von Generationen ohne Verbesserung bis Prüfung der Abbruchbedingung (Standard: 20)

Ausgabe: Minimalstelle $\vec{x}_{\text{opt}}$

function $\vec{x}_{\text{opt}}$ = DifferentialEvolution(f, N, m, p, α, β, γ, n)
  for $i = 1, \ldots, m$ **do** wähle $\vec{x}_i \in [0,1]^d$ gleichverteilt zufällig
  $(a_0, k_0) \leftarrow (0,0)$
  for $k = 1, \ldots, \lfloor N/m - 1 \rfloor$ **do** ⇝ *da genau m Funktionsauswertungen pro Runde*
    for $i = 1, \ldots, m$ **do**
      wähle $a, b, c \in \{1, \ldots, m\} \backslash \{i\}$ gleichverteilt zufällig und paarweise verschieden
      wähle $t_0 \in \{1, \ldots, d\}$ gleichverteilt zufällig
      for $t = 1, \ldots, d$ **do**
        wähle $z \in [0,1]$ gleichverteilt zufällig
        if $(z \leqslant p) \vee (t = t_0)$ **then** $y_t \leftarrow x_{a,t} + \alpha(x_{b,t} - x_{c,t})$ ⇝ *mit $\vec{x}_a = (x_{a,t})_t$ usw.*
        else $y_t \leftarrow x_{i,t}$
      if $\widehat{f}(\vec{y}) < f(\vec{x}_i)$ **then** $\vec{x}_i^* \leftarrow \vec{y}$ ⇝ *mit $\widehat{f}$ wie bei Nelder-Mead*
      else $\vec{x}_i^* \leftarrow \vec{x}_i$
    for $i = 1, \ldots, m$ **do** $\vec{x}_i \leftarrow \vec{x}_i^*$
    $a_k \leftarrow \sum_{i=1}^{m} f(\vec{x}_i)/m$ ⇝ *durchschnittlicher Funktionswert der Population*
    if $a_{k-1} - a_k \geqslant \beta$ **then** $k_0 \leftarrow k$ ⇝ *Verbesserung „akzeptabel“*
    else if $k - k_0 \geqslant n$ **then** ⇝ *letzte „akzeptable“ Verbesserung zu lange her*
      $i^* \leftarrow \arg\min_{i=1,\ldots,m} f(\vec{x}_i)$
      if $\max_{i=1,\ldots,m} \|\vec{x}_i - \vec{x}_{i*}\|_2 < \gamma$ **then break** ⇝ *Abstand zum besten Punkt klein*
  $i^* \leftarrow \arg\min_{i=1,\ldots,m} f(\vec{x}_i)$
  return $\vec{x}_{i*}$

In Algorithmus 5.7 ist das DE-Verfahren nach [50] dargestellt, wobei N wieder die maximale Auswertungszahl von f bezeichnet. Es werden höchstens $\lfloor N/m - 1 \rfloor$ Iterationen durchgeführt, weil f in jeder Iteration m-fach ausgewertet wird (an jedem mutierten Vektor). Zusätzlich kann vorzeitig abgebrochen werden, wenn ein bestimmtes Kriterium nach [59] erfüllt ist. Das Kriterium überprüft, ob sich der durchschnittliche Funktionswert der Population n Iterationen lang nicht entscheidend verbessert hat ($n \in \mathbb{N}$ konstant). Ist dies der Fall, dann wird geprüft, ob der Abstand aller Punkte der Population zum besten Punkt klein ist – trifft auch das zu, dann wird Konvergenz angenommen und abgebrochen. Einen exemplarischen Verlauf des DE-Verfahrens für die Schwefel-Testfunktion sieht man in Abb. 5.3c.

6 Implementierung

Als Nächstes geht es um die Implementierung der vorgestellten dünnen Gitter und Basisfunktionen sowie von Gittererzeugung, Interpolation und gradientenbasierten/-freien Optimierungsverfahren. Weil in der bereits vorhandenen SG^{++}-Toolbox die grundlegenden Datenstrukturen und Basisfunktionen schon implementiert waren, hat sich die Entwicklung eines neuen SG^{++}-Moduls namens `sg::opt` angeboten.

Der verwendete Testrechner ist ein Kubuntu-14.04-LTS-System (Linux-Kernel 3.13.0) mit Intel-i5-3450-Vierkernprozessor (3,1 GHz je Kern) und 8 GB Arbeitsspeicher. Als Compiler kam g++ 4.8.2 zum Einsatz, wobei der Code auch unter g++ 4.6 kompilierbar ist.

6.1 Die SG^{++}-Toolbox

6.1.1 Merkmale von SG^{++}

SG^{++} (auch SGpp) ist eine von Pflüger im Rahmen seiner Dissertation [40] mitentwickelte Software zum Umgang mit dünnen Gittern. Wie der Name nahelegt, wurde SG^{++} größtenteils in C++ programmiert. Allerdings finden sich auch Python- und Java-Schnittstellen für den High-Level-Gebrauch, sodass man schnell und bequem in SG^{++} einsteigen kann.

Die Modularität ist eines der wichtigsten Merkmale von SG^{++}. Datenstrukturen und Algorithmen wurden laut [40] soweit möglich getrennt, was die Wiederverwendbarkeit steigert. SG^{++} gliedert sich in verschiedene Module. Im wichtigsten Modul `sg::base` sind unter anderem die Datenstrukturen zur Erstellung von und Auswertung auf dünnen Gittern enthalten. Weitere anwendungsorientierte Module wie `sg::finance` und `sg::pde` behandeln z. B. Probleme aus der finanziellen Regression bzw. partieller Differentialgleichungen.

SG^{++} ist auf Geschwindigkeit hin optimiert worden, weswegen bspw. bei geschwindigkeitskritischen Programmteilen wie der Auswertung von Basisfunktionen explizit auf Vererbung verzichtet wurde. Sonst muss das Programm – z. B. bei jeder Auswertung einer Basisfunktion – bei jedem Aufruf des Programmteils zur Laufzeit in der sogenannten *vtable* (Tabelle virtueller Methoden) nachschlagen, welche der Unterklassen-Methoden verwendet werden soll. Wird der Programmteil sehr oft aufgerufen, dann wirkt sich das spürbar negativ auf die Laufzeit aus. Weitere Geschwindigkeitsvorteile werden nach [40] durch die Ausnutzung von Sprungvorhersagen (engl. *branch predictions*) und durch Parallelisierung erzielt.

Abbildung 6.1: Logo von SG^{++}, Quelle: [40]

6.1.2 Gestaltung und Aufbau

Die wichtigsten Bestandteile des Basismoduls `sg::base` von SG^{++} sind wie folgt:

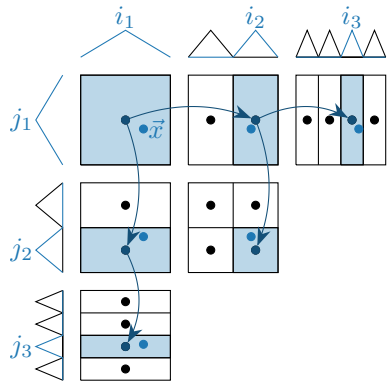
- **Gitterverwaltung**: Ein einzelner Gitterpunkt $\vec{x}_{\vec{\ell},\vec{j}}$ wird als Paar von Level und Index in einem `HashGridIndex` gespeichert. Komplette Gitter $X = \{\vec{x}_i\}_i$ werden in Hashtabellen der Klasse `HashGridStorage` festgehalten.
- **Gitterarten**: Je nachdem, welche Gitterart und Basisfunktionen man verwenden will, erzeugt man Instanzen von `Grid`-Unterklassen. Es stehen u. a. `LinearGrid` und `ModLinearGrid` für stückweise lineare Basisfunktionen auf Noboundary-Gittern zur Verfügung (ohne bzw. mit Modifikation der Basisfunktionen).
- **Gittererzeugung**: Darum kümmern sich `GridGenerator`-Unterklassen. Mit ihnen kann man z. B. reguläre dünne Gitter erzeugen oder ein bestehendes Gitter mittels eines `RefinementFunctor`s (Verfeinerungsfunktor) verfeinern.
- **Basisfunktionen**: Jede der Kombinationen von Gittertyp und Basisfunktionsart besitzt eine eigene Klasse, die mindestens die Auswertung in einer Dimension enthält. Wie oben beschrieben ist das Fehlen einer gemeinsamen Oberklasse gewollt.
- **Operationen**: Für Operationen wie Hierarchisierung und Basisfunktionsauswertung gibt es z. B. die abstrakten Klassen `OperationHierarchisation` bzw. `OperationEval`, die durch Klassen in Abhängigkeit von den Basisfunktionen abgeleitet werden.

6.1.3 Algorithmen zur Hierarchisierung und Auswertung

Die bedeutendsten in SG^{++} enthaltenen Operationen auf dünnen Gittern sind die Hierarchisierung (Bestimmung der Koeffizienten einer interpolierenden Linearkombination der Basisfunktionen) und die Auswertung (Auswertung der Linearkombination in einem beliebigen Punkt $\vec{x} \in [0,1]^d$). Die Hierarchisierung für stückweise lineare Basisfunktionen geschieht mithilfe des bereits in Abschnitt 4.2 beschriebenen unidirektionalen Prinzips. Dazu gibt es in SG^{++} eine Template-Klasse `sweep`, die dieses Prinzip generisch implementiert.

Nun wollen wir uns der Auswertung zuwenden: Gegeben ist eine Linearkombination $\widetilde{f}(\vec{x}) := \sum_{k=1}^{N} \alpha_k \varphi_k(\vec{x})$ mit $\alpha_k \in \mathbb{R}$ und $\varphi_k := \varphi_{\vec{\ell}_k,\vec{j}_k}$. Gesucht ist für ein $\vec{x} \in [0,1]^d$ der Wert $\widetilde{f}(\vec{x})$. Naiverweise würde man alle Werte $\varphi_k(\vec{x}) = \prod_{t=1}^{d} \varphi_{\ell_{k,t},j_{k,t}}(x_t)$ ausrechnen, was aber Nd Auswertungen von 1D-Basisfunktionen entspricht. Es gibt nach [40, 41] allerdings eine bessere Möglichkeit: Man sucht rekursiv für jede Dimension die Basisfunktionen, deren Träger $\vec{x}$ enthalten, und klammert 1D-Basisauswertungen aus. Ein Beispiel für stückweise lineare Basisfunktionen findet sich in Abb. 6.2. Durch diese Aufspaltung kommt man mit deutlich weniger 1D-Auswertungen aus.

Die entsprechende Routine heißt in SG^{++} `GetAffectedBasisFunctions` (kurz `GABF`) und ist in Algorithmus 6.1 beschrieben. Zur Vereinfachung der Notation sei im Folgenden $\alpha_{\vec{\ell},\vec{j}}$ der Koeffizient α_i mit $\vec{x}_i = \vec{x}_{\vec{\ell},\vec{j}}$. Außerdem bezeichnen die Indizes $(\mathrm{rn}(t))$ und $(\mathrm{ln}(t))$ den rechten bzw. linken direkten Nachbar in Dimension t (also $j_t^* := j_t \pm 2$) sowie $(\mathrm{rc}(t))$ und $(\mathrm{lc}(t))$ das rechte bzw. linke direkte Kind in Dimension t (also $(\ell_t^*, j_t^*) := (\ell_t + 1, 2j_t \pm 1)$).



$$\begin{aligned}\widetilde{f}(\vec{x}) = \varphi_{1,i_1}(x_1) \cdot \big[\varphi_{1,j_1}(x_2) + \varphi_{2,j_2}(x_2) + \varphi_{3,j_3}(x_2)\big] \\ + \varphi_{2,i_2}(x_1) \cdot \big[\varphi_{1,j_1}(x_2) + \varphi_{2,j_2}(x_2)\big] \\ + \varphi_{3,i_3}(x_1) \cdot \big[\varphi_{1,j_1}(x_2)\big]\end{aligned}$$

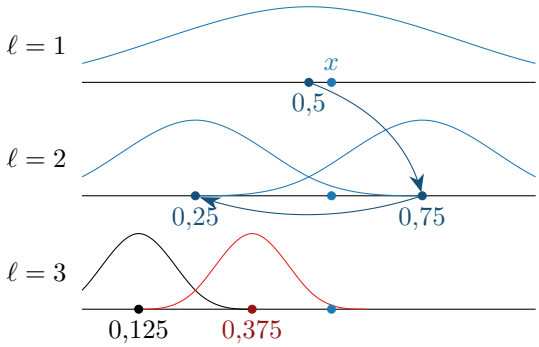
Abbildung 6.2: Auswertung in $\vec{x}$ auf einem regulären dünnen 2D-Gitter mit Level 3 für stückweise lineare Basisfunktionen (Abbildung nach [41])

Algorithmus 6.1: Approximative Auswertung von Linearkombinationen auf dünnen Gittern, Zeilen 5, 6, 9, 10 nicht für stückweise lineare Basisfunktionen

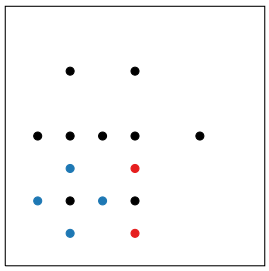
Eingabe: Gitter $X = \{\vec{x}_i\}_i$, Koeffizienten $\vec{\alpha} = (\alpha_i)_i$, Auswertungspunkt $\vec{x} \in [0,1]^d$, aktuelle Dimension $t \in \{1, \ldots, d\}$ (anfangs 1), Level und Index $(\vec{\ell}, \vec{j})$ des aktuellen Punkts (für randlose Gitter anfangs $(\vec{e}, \vec{e})$) und aktuelles Produkt b von 1D-Auswertungen (anfangs 1)
Ausgabe: $a \approx \widetilde{f}(\vec{x}) = \sum_{k=1}^{N} \alpha_k \varphi_k(\vec{x})$ (für stückweise lineare Funktionen sogar $a = \widetilde{f}(\vec{x})$)

```
1  function a = GetAffectedBasisFunctions(X, α⃗, x⃗, t, ℓ⃗, j⃗, b)
2      if x_{ℓ⃗,j⃗} ∉ X then return 0                    ⇝ nichts tun, falls Gitterpunkt nicht vorhanden
3      if t = d then
4          a ← α_{ℓ⃗,j⃗} · (b · φ_{ℓ_d,j_d}(x_d))        ⇝ letzte Dimension: Summanden zu Ergebnis addieren
5          if x⃗^{(rn(d))}_{ℓ⃗,j⃗} ∈ X then a ← a + α^{(rn(d))}_{ℓ⃗,j⃗} · (b · φ^{(rn(d))}_{ℓ_d,j_d}(x_d))
6          if x⃗^{(ln(d))}_{ℓ⃗,j⃗} ∈ X then a ← a + α^{(ln(d))}_{ℓ⃗,j⃗} · (b · φ^{(ln(d))}_{ℓ_d,j_d}(x_d))
7      else
8          a ← GABF(X, α⃗, x⃗, t + 1, ℓ⃗, j⃗, b · φ_{ℓ_t,j_t}(x_t))          ⇝ nächste Dimension
9          if x⃗^{(rn(t))}_{ℓ⃗,j⃗} ∈ X then a ← a + GABF(X, α⃗, x⃗, t + 1, ℓ⃗, j⃗^{(rn(t))}, b · φ^{(rn(t))}_{ℓ_t,j_t}(x_t))
10         if x⃗^{(ln(t))}_{ℓ⃗,j⃗} ∈ X then a ← a + GABF(X, α⃗, x⃗, t + 1, ℓ⃗, j⃗^{(ln(t))}, b · φ^{(ln(t))}_{ℓ_t,j_t}(x_t))
11     if x_t > j_t h_{ℓ_t} then a ← a + GABF(X, α⃗, x⃗, t, ℓ⃗^{(rc(t))}, j⃗^{(rc(t))}, b)    ⇝ nächster Level
12     else a ← a + GABF(X, α⃗, x⃗, t, ℓ⃗^{(lc(t))}, j⃗^{(lc(t))}, b)
13     return a
```

Die dunkelgrünen Zeilen 5, 6, 9, 10 können bei stückweise linearen Funktionen ignoriert werden. Die Routine ist rekursiv in der Dimension (Zeile 8) und im Level (Zeilen 11, 12). Bei Basisfunktionen, die einen „breiteren“ Träger haben (B-Splines, Wavelets), kommt eine modifizierte Version des Algorithmus zum Einsatz, die zusätzlich die Zeilen 5, 6, 9, 10 enthält. Diese Zeilen betrachten bei der Rekursion in der Dimension auch linke/rechte direkte Nachbarn der Knoten. Damit können Linearkombinationen mit Basisfunktionen $\varphi_{\ell,j}$, die $\operatorname{supp} \varphi_{\ell,j} \subset h_\ell \cdot [j \pm 3]$ erfüllen, auf regulären dünnen Gittern exakt berechnet werden. Zu diesen Basisfunktionen gehören B-Splines vom Grad $\leqslant 5$ und auch Mexican-



(a) Beispiel dafür, dass `GABF` nicht exakt arbeitet (unmodifizierte B-Splines vom Grad 3 auf nichtregulärem 1D-Gitter, $x = 0{,}55$)



(b) notwendige hierarchische Vorfahren bei Einfügung der blauen Knoten (nach [40])

Abbildung 6.3: Probleme der in SG^{++} implementierten Algorithmen

Hat-Basisfunktionen, wenn diese geeignet abgeschnitten werden (siehe Abschnitt 6.2.3). B-Splines vom Grad > 5 besitzen einen zu breiten Träger, sodass `GABF` nicht exakt arbeitet, weil auch übernächste Nachbarn betrachtet werden müssten.

Für nichtreguläre Gitter arbeitet Algorithmus 6.1 allerdings meistens nicht exakt, weil „falsch“ zum nächsten Level abgestiegen wird. Für ein einfaches Gegenbeispiel reicht $d = 1$, siehe Abb. 6.3a. Das 1D-Gitter $\{0{,}125; 0{,}25; 0{,}375; 0{,}5; 0{,}75\}$ entsteht durch zwei Verfeinerungen (zunächst von 0,5 und dann von 0,25) aus dem „trivialen“ Gitter $\{0{,}5\}$. `GABF` „besucht“ auf dem in Abb. 6.3a mit Pfeilen markierten Weg nur die blauen Basisfunktionen $\varphi_{1,1}$, $\varphi_{2,1}$ und $\varphi_{2,3}$. Die rote Basisfunktion $\varphi_{3,3}$ wird nicht betrachtet, obwohl $\varphi_{3,3}(x) \neq 0$ gilt. Speziell in höheren Dimensionen wird der so entstandene Fehler bemerkbar.

Ein weiteres Problem, das nicht nur die Auswertung, sondern auch die Hierarchisierung betrifft, ist die Voraussetzung der Algorithmen, dass jeder Gitterpunkt in X in jeder Dimension t alle hierarchischen Vorfahren besitzen muss. Das heißt für randlose Gitter

$$\left[x_{\vec{\ell},\vec{j}} \in X,\ t \in \{1,\dots,d\},\ \ell_t > 1\right] \implies x_{\vec{\ell},\vec{j}}^{(\mathrm{p}(t))} \in X \tag{6.1}$$

mit $x_{\vec{\ell},\vec{j}}^{(\mathrm{p}(t))}$ dem direkten Vorfahren in Dimension t (also $(\ell_t^*, j_t^*) := (\ell_t - 1, 2\lfloor j_t/4 \rfloor + 1)$). SG^{++} besitzt in der für die Verfeinerung zuständigen Klasse `AbstractRefinement` eine automatische rekursive Erzeugung der (indirekten) hierarchischen Vorfahren, wenn ein Gitterpunkt zum Gitter hinzugefügt wird, damit Eigenschaft (6.1) erfüllt bleibt.

Das Problem dabei ist, dass bei einer Einfügung eines neuen Gitterpunkts für die d direkten Vorfahren auch deren (indirekte) Vorfahren erzeugt werden müssen. Die Zahl der für eine einzelne Gitterpunkt-Einfügung tatsächlich einzufügenden Knoten steigt sehr schnell mit der Dimensionalität d. Beispielsweise führt die Ritter-Novak-Verfeinerung aus Abschnitt 4.1.2, mit $\alpha = 0{,}8$ angewandt auf die Rosenbrock-Testfunktion (siehe Anhang A), für $d = 2, 3, 4, 5, 10$ jeweils zu $1128, 223, 61, 33, 16$ Iterationen, bis $N = 10000$ Punkte erzeugt worden sind. Eigentlich werden in jeder Iteration (ohne Vorfahren) nur $2d$ Punkte eingefügt. Damit wurden für $d = 2$ nur 45 %, für $d = 3$ nur 13 % und für $d \geqslant 4$ weniger als 5 % der möglichen Zahl an Verfeinerungen ausgenutzt, was nur schwer akzeptabel scheint.

Die Problematik wird dadurch noch verschärft, dass einige der zusätzlichen hierarchischen Vorfahren an Stellen liegen, an denen sie keinen Nutzen bringen. Manche der Vorfahren liegen auf den Hauptachsen (Achsenparallelen durch $0{,}5\vec{e}$), obwohl der eigentlich einzufügende Punkt vielleicht eher am Rand von $[0,1]^d$ liegt. Die Gittererzeugung nach Ritter-Novak, bei der besonders Stellen mit kleinem Funktionswert oft verfeinert werden, kann mit diesen Vorfahren also wenig anfangen. Außerdem steigt die Genauigkeit der Interpolierenden an diesen für die Optimierung eher uninteressanten Stellen, anstatt anderswo höher zu sein.

6.2 `sg::opt` als neues SG^{++}-Modul

`sg::opt` ist ein neues Modul für SG^{++}, das die in dieser Arbeit behandelten Themen wie Gittererzeugung, Hierarchisierung und Optimierung implementiert. Es wurde darauf geachtet, dass `sg::opt` auch ohne zusätzliche externe Bibliotheken kompilierbar ist. Gleichwohl kann mit dem Vorhandensein unterstützter Software zum Lösen von linearen Gleichungssystemen (LGS) die Hierarchisierung erheblich beschleunigt werden (siehe Abschnitt 6.2.5).

`sg::opt` unterstützt teilweise Parallelisierung mittels OpenMP [12]. Diese kommt bei der Gittererzeugung, der Aufstellung des Interpolations-LGS (je nach Löser auch bei der Lösung des LGS) sowie bei den gradientenfreien Optimierungsverfahren zum Einsatz und ermöglicht eine deutlich schnellere Ausführung.

6.2.1 Aufbau des Moduls

`sg::opt` beinhaltet folgende Bestandteile:

- **`function`**: Klassen für Zielfunktion, Gradient und Hesse-Matrix, `function::test` enthält alle Testfunktionen aus Anhang A
- **`gridgen`**: adaptive Gittererzeugung mit linearen Überschüssen und nach Ritter-Novak
- **`sle`**: Anbindung an verschiedene Bibliotheken zur Lösung linearer Gleichungssysteme, `sle::system::Hierarchisation` ermöglicht Hierarchisierung
- **`optimizer`**: Methoden zur Optimierung von Funktionen
- **`tools`**: verschiedene Hilfsklassen

In den folgenden Abschnitten werden wir auf die wichtigsten Punkte näher eingehen.

6.2.2 Ziel- und Testfunktionen

Zielfunktionen müssen von der abstrakten Klasse `function::Objective` abgeleitet werden. Es gibt zwar auch `ObjectiveGradient` und `ObjectiveHessian`, aber diese Klassen werden nur von den gradientenbasierten Optimierungsroutinen benötigt (die auch Funktionen vom Typ `function::Objective` optimieren, egal, ob es sich um die eigentliche Zielfunktion oder eine Interpolierende handelt).

Tabelle 6.1: Unterstützte Kombinationen von Gittertypen (Noboundary, Noboundary mit modifizierter Basis, Boundary, Clenshaw-Curtis) und Basisfunktionen

Basis \ Gitter	NB	mod.	B	CC
stückweise linear	✓	✓	✓	✓
B-Splines	✓	✓	✓	✓
Mexican-Hat	✓	✓	✓	✗

Die in `function::test` implementierten Testfunktionen leiten sich alle von der abstrakten Klasse `Test` ab, die wie in [29] eine automatische, pseudozufällige Verschiebung der Funktion implementiert, also

$$f\colon [0,1]^d \to \mathbb{R} \quad \rightsquigarrow \quad f^*\colon [0,1]^d \to \mathbb{R},\ f^*(\vec{x}) := f(\vec{x} + \vec{d}) \tag{6.2}$$

mit d_t normalverteilt mit Erwartungswert 0 und Standardabweichung $\sigma > 0$, $t = 1, \ldots, d$ (Standard: $\sigma = 0{,}01$). Bei der Evaluierung werden wir den Durchschnitt mehrerer Durchläufe mit unterschiedlicher Verschiebung berechnen, um den Ergebnissen mehr Aussagekraft zu verleihen (Genaueres siehe Anhang B.1). Zu beachten ist, dass bei manchen Testfunktionen die Menge der erlaubten Verschiebungen $\vec{d}$ eingeschränkt wurde, weil sonst Polstellen in oder Minimalstellen außerhalb $[0,1]^d$ fallen würden.

6.2.3 Basisfunktionen und Gitterarten

Das Basismodul `sg::base` implementiert bereits modifizierte B-Splines und modifizierte Mexican-Hat-Basisfunktionen auf Noboundary-Gittern. Im Zuge dieser Arbeit wurden weitere, in Tabelle 6.1 dargestellte Kombinationen von Gittertypen und Basisfunktionen in `sg::base` implementiert. Es wurden auch lineare Basisfunktionen betrachtet, weil diese für die überschussbasierte Verfeinerung notwendig sind. Für Clenshaw-Curtis-Gitter wurde die Gitterpunkt-Klasse `sg::base::HashGridIndex` so verändert, dass Gitterpunkt-Koordinaten zwischengespeichert werden. Damit können Gittererzeugung und Hierarchisierung beschleunigt werden, weil nicht bei jedem Aufruf cos-Werte berechnet werden müssen.

Die in `sg::opt` standardmäßige Implementierung der Auswertung durch `OperationEval` verwendet den weiter oben in Abschnitt 6.1.3 erläuterten `GetAffectedBasisFunctions`-Algorithmus (`GABF`). Die Pferdefüße von `GABF` bei der Interpolation mit glatten Funktionen sind die teilweise fehlerbehaftete Auswertung und die Notwendigkeit hierarchischer Vorfahren. Deswegen ist es besser, Gitter zuzulassen, die nicht mehr notwendigerweise die Vorfahren-Eigenschaft (6.1) erfüllen, und bei der Auswertung von Linearkombinationen statt `GABF` zu verwenden (wie `OperationEval`) naiv alle Summanden auszuwerten. Die entsprechende Operation wurde `OperationNaiveEval` genannt und muss statt `OperationEval` genutzt werden, wenn mit `sg::opt` gearbeitet wird.

Zusätzlich zur Auswertung der Basisfunktionen wurden (außer für die stückweise linearen Funktionen) auch die für die gradientenbasierten Optimierungsverfahren notwendigen ersten und zweiten Ableitungen implementiert (Operationen `OperationNaiveEvalGradient` und `OperationNaiveEvalHessian`). Die in `sg::base` ebenfalls fehlende Hierarchisierung für B-Splines und Wavelets findet sich separat im `sg::opt::sle`-Bereich.

Bei den Mexican-Hat-Basisfunktionen werden wie schon in `sg::base` und in [40] abgeschnittene Versionen betrachtet, genauer $\varphi_{\ell,i}(x) := 0$ für $|x - ih_\ell| \geqslant 2h_\ell$. Damit haben diese Basisfunktionen denselben Träger wie B-Splines vom Grad 3. Das hat zwei Vorteile: Zum einen wird dadurch `GABF` für diese Basisfunktionen anwendbar, was allerdings nicht so wichtig ist, da wir wegen der fehlenden hierarchischen Vorfahren auf diesen Algorithmus verzichten. Zum anderen wäre ohne Abschneiden die Matrix des Gleichungssystems bei der Interpolation voll besetzt, was den Lösungsaufwand stark vergrößern würde. Mit Abschneiden verschwinden je nach Dimensionalität d viele der Einträge. Natürlich wird durch das Abschneiden ein Fehler eingeführt, der absolut durch 0,055 (Wert von $\varphi_{\ell,i}$ in $(i \pm 2)h_\ell$) nach oben beschränkt ist. Außerdem ist die Interpolierende dann nicht mehr glatt, ja nicht einmal mehr stetig. Wie stark sich das Abschneiden auf die gradientenbasierten Optimierungsverfahren auswirkt, werden wir in Abschnitt 7.1.6 untersuchen.

6.2.4 Adaptive Gittererzeugung

Die beiden Methoden der adaptiven Gittererzeugung mit linearen Überschüssen oder nach Ritter-Novak wurden in `sg::opt::gridgen` implementiert. Das Adaptivitätskriterium der Ritter-Novak-Verfeinerung enthält (siehe (4.2)) zwei Exponentiationen, die in jeder Runde für alle momentan existierenden Gitterpunkte berechnet werden müssen. Man kommt insgesamt auf die nicht unbedeutende Zahl von $\Theta(N^2/d + N)$ notwendigen Exponentiationen. Die Verwendung approximativer Exponentiationen wie `fastPow` von [1] statt `std::pow` bringt einen erheblichen Geschwindigkeitsvorteil: Die Gittererzeugung benötigt so nur noch ca. ein Drittel (für $d = 2$) oder die Hälfte (für $d = 10$) der Zeit. Allerdings geht dieser Vorteil mit einem schlechteren Ergebnis der Optimierung bei „schwierigen“ Zielfunktionen (z. B. Easom-Testfunktion) einher, obwohl der relative Fehler von `fastPow` der bei einem typischen Ablauf der Gittererzeugung auftretenden Exponentiationen selten 5 % übersteigt. Wir verwenden daher standardmäßig `std::pow`, lassen aber `fastPow` im Programm als Option zu. Außerdem wird bei der Ritter-Novak-Verfeinerung sichergestellt, dass ein bestimmter Maximallevel durch die erzeugten Gitterpunkte nicht überschritten wird (um sinnlose Verfeinerungen und Ganzzahlüberläufe zu vermeiden).

Bei der überschussbasierten Verfeinerung haben wir in Abschnitt 4.1.1 offen gelassen, wie die linearen Überschüsse berechnet werden. Eigentlich müsste in jeder Runde ein lineares Gleichungssystem (LGS) gelöst werden, das aber mit jeder Iteration größer wird. Zur Lösung des Problems erzeugen wir bei der überschussbasierten Verfeinerung im Gegensatz zur Verfeinerung nach Ritter und Novak auch die (eigentlich unnötigen) hierarchischen Vorfahren mit. Dadurch ist das LGS nämlich in unterer Dreiecksform, weil für die stückweise linearen Basisfunktionen $\varphi_{\vec{\ell}*,\vec{j}*}$ neu eingefügter Punkte $\vec{x}_{\vec{\ell}*,\vec{j}*}$ gilt, dass $\varphi_{\vec{\ell}*,\vec{j}*}(\vec{x}_{\vec{\ell},\vec{j}}) = 0$ für alle davor eingefügten Gitterpunkte $\vec{x}_{\vec{\ell},\vec{j}}$. Andernfalls wäre nämlich $\varphi_{\vec{\ell}*,\vec{j}*}(\vec{x}_{\vec{\ell},\vec{j}}) > 0$ und damit müsste $\vec{x}_{\vec{\ell}*,\vec{j}*}$ ein (indirekter) hierarchischer Vorfahre von $\vec{x}_{\vec{\ell},\vec{j}}$ sein. Also müsste $\vec{x}_{\vec{\ell}*,\vec{j}*}$ bei Einfügung von $\vec{x}_{\vec{\ell},\vec{j}}$ bereits eingefügt worden sein, ein Widerspruch ($\vec{x}_{\vec{\ell}*,\vec{j}*}$ könnte nicht neu eingefügt werden). Die untere Dreiecksform macht das LGS durch Vorwärtseinsetzen sehr einfach lösbar. Hinzu kommt, dass sich bestehende Zeilen der LGS-Matrix im weiteren Verlauf der Gittererzeugung nicht ändern, sodass nur die Überschüsse neu eingefügter Punkte in linearer Zeit durch Einsetzen der alten Koeffizienten berechnet werden müssen.

Tabelle 6.2: Unterstützte Löser linearer Gleichungssysteme

Klasse	Bibliothek	Methode	direkt/ iterativ	dünn/voll besetzt	parallelisierbar
`Armadillo`	Armadillo, [46]	LR-Zerlegung	direkt	voll	ja
`BiCGStab`	SG^{++}	BiCGStab	iterativ	voll	nein
`Eigen`	Eigen, [22]	QR-Householder-Zerl.	direkt	voll	nein
`Gmmpp`	Gmm++, [45]	GMRES	iterativ	dünn	nein
`UMFPACK`	UMFPACK, [13]	LR-Zerlegung	direkt	dünn	ja

6.2.5 Hierarchisierung

Die Hierarchisierung findet im `sg::opt::sle`-Namensraum statt. Er teilt sich in den `system`-Namensraum, der mit `System` eine abstrakte LGS-Klasse bereitstellt, und den `solver`-Namensraum, der verschiedene Klassen zum Lösen von LGS enthält. Die Klassen kapseln größtenteils Aufrufe externer Bibliotheken und sind in Tabelle 6.2 aufgelistet. Bei der Auswahl der Bibliotheken wurde darauf geachtet, dass sie frei verfügbar und einfach erhältlich sind (alle Bibliotheken lassen sich etwa aus den Ubuntu-Paketquellen herunterladen).

Die unterstützten LGS-Löser unterteilen sich in verschiedene Gruppen. Die direkten Löser sind für die bei der Dünngitter-Interpolation auftretenden LGS meist deutlich schneller als iterative Löser, benötigen aber wesentlich mehr Arbeitsspeicher. Löser, die mit dünn besetzten Matrizen arbeiten, sind nur für Gleichungssysteme sinnvoll, die selbst auch dünn besetzt sind. In diesem Fall ist natürlich der benötigte Speicher ebenfalls viel geringer wie bei Lösern, die mit voll besetzten Matrizen arbeiten. Bei der Hierarchisierung treten (je nach N und d) LGS mit unterschiedlichen Größen und Besetzungsstrukturen auf. Daher gibt es den `Auto`-Löser, der empirisch anhand Größe des LGS, geschätztem Anteil der Nicht-Null-Einträge und installierten Bibliotheken selbst einen Löser wählt. Ist kein Löser installiert, so kommt das intern implementierte BiCGStab-Verfahren aus [55] zum Einsatz.

6.2.6 Optimierung

Die implementierten Optimierungsverfahren sind die in Kapitel 5 vorgestellten Verfahren, also Gradienten-, CG- und Newton-Verfahren (gradientenbasierte Methoden) sowie Nelder-Mead-Verfahren, zufällige Suche und Differentialevolution (gradientenfreie Methoden). Die Verfahren benötigen jeweils eine zu optimierende Funktion der oben erwähnten abstrakten Klasse `function::Objective`. Die gradientenbasierten Verfahren benötigen zusätzlich noch den Gradienten mit `function::ObjectiveGradient` oder im Fall vom Newton-Verfahren Gradient und Hesse-Matrix durch `function::ObjectiveHessian`.

6.3 Arbeitsablauf zur globalen Optimierung

6.3.1 Optimierung auf glatter Dünngitter-Interpolierenden

Jetzt können wir beschreiben, wie wir bei der Optimierung von Zielfunktionen vorgehen wollen. Bevor man die Optimierung startet, muss man ihre Parameter wählen, weswegen wir das als „0. Schritt“ bezeichnen.

0. **Optimierungsparameter**: Wähle Zielfunktion f, Gittertyp, Art der Basisfunktionen, Adaptivität α und maximale Auswertungszahl $N \in \mathbb{N}$.

Ist die Wahl getroffen, so verfahren wir zur globalen Optimierung von f mit glatten Basisfunktionen wie folgt:

1. **Gittererzeugung**: Erzeuge adaptiv mithilfe eines der beiden vorgestellten Verfahren ein dünnes Gitter $X = \{\vec{x}_1, \dots, \vec{x}_n\} \subset \mathbb{R}^d$ (mit $\vec{x}_i := \vec{x}_{\vec{\ell}_i, \vec{j}_i}$) und erhalte $f_i := f(\vec{x}_i)$.
2. **Hierarchisierung**: Löse das LGS $\sum_{k=1}^{n} \alpha_k \varphi_k(\vec{x}_i) = f_i$ $(i = 1, \dots, n,\ \varphi_k := \varphi_{\vec{\ell}_k, \vec{j}_k})$ und erhalte die Interpolierende $\widetilde{f}\colon [0,1]^d \to \mathbb{R}$, $\widetilde{f}(\vec{x}) := \sum_{k=1}^{n} \alpha_k \varphi_k(\vec{x})$.
3. **Optimierung der Interpolierenden**: Bestimme $\vec{y}_0 := \vec{x}_{i*}$, $i^* := \arg\min_{i=1,\dots,n} f_i$. Berechne Minimalstellen $\vec{y}_1, \vec{y}_2, \vec{y}_3$ von $\widetilde{f}$ durch die drei gradientenbasierten Optimierungsverfahren auf $\widetilde{f}$ mit Startpunkt $\vec{y}_0$. Berechne nun Minimalstellen $\vec{y}_4, \vec{y}_5, \vec{y}_6$ von $\widetilde{f}$ durch die gradientenfreien Verfahren auf $\widetilde{f}$. Bestimme $\vec{y}_7 := \vec{y}_{j*}$ mittels $j^* := \arg\min_{j=0,\dots,6} f(\vec{y}_j)$. Berechne nochmals Minimalstellen $\vec{y}_8, \vec{y}_9, \vec{y}_{10}$ von $\widetilde{f}$ durch die gradientenbasierten Verfahren auf $\widetilde{f}$ mit Startpunkt $\vec{y}_7$. Wähle nun $\vec{x}^*_{\text{opt}} := \vec{y}_{k*}$, $k^* := \arg\min_{k=7,\dots,10} f(\vec{y}_k)$.

Der letzte, dritte Schritt erfordert zusätzlich zu den bereits bei der Gittererzeugung erfolgten Auswertungen von f noch einmal 10 Auswertungen. Es werden somit eigentlich $\leqslant N + 10$ Auswertungen getätigt. Die zusätzlichen Auswertungen sind aber konstant viele und können daher im Vergleich zu N vom Aufwand her vernachlässigt werden.

Die zweite Runde der gradientenbasierten Verfahren zur Bestimmung von $\vec{y}_8, \vec{y}_9, \vec{y}_{10}$ kann man auch weglassen. Sie dient nur der Sicherheit, falls die gradientenfreien Verfahren zwar eine Stelle mit kleinem Funktionswert gefunden haben, aber nicht vollständig zu einem lokalen Minimum konvergiert sind. In den allermeisten Fällen werden die Optimierungsverfahren der zweiten Runde nach sehr wenigen Schritten abbrechen.

Dadurch, dass $\vec{y}_0$ bei der Berechnung von $\vec{y}_7$ mit einbezogen wird, gilt auf jeden Fall $f(\vec{x}^*_{\text{opt}}) \leqslant f_i$ für alle $i = 1, \dots, n$. $\vec{x}^*_{\text{opt}}$ ist damit der Punkt, der von allen Punkten, an denen f im Verlauf des Programms ausgewertet wurde (auch während der Gittererzeugung), den kleinsten Funktionswert besitzt.

6.3.2 Vergleich mit bestehenden Verfahren

Im nächsten Kapitel werden wir die Leistung des obigen Algorithmus mit bestehenden Optimierungsalgorithmen vergleichen. Dazu führen wir zwei Vergleiche durch: Erstens betrachten wir auch die stückweise lineare Interpolierende auf dem dünnen Gitter X und optimieren diese. Genauer berechnen wir in Schritt 2 darüber hinaus noch die Interpolierende, wenn φ_k die stückweise linearen Basisfunktionen sind, optimieren diese mit den gradientenfreien Verfahren (Nelder-Mead lassen wir dabei in $\vec{y}_0$ starten) und wählen wieder den nach dem Zielfunktionswert „besten“ Punkt $\vec{x}'_{\text{opt}}$.

Zweitens wenden wir die gradientenfreien Verfahren auch direkt auf die Zielfunktion f an, um die von den drei resultierenden Punkten „beste“ Minimalstelle $\vec{x}''_{\text{opt}}$ zu erhalten. Damit auch hier maximal N Funktionsauswertungen von f ausgeführt werden, darf jedes der gradientenfreien Verfahren höchstens $\lceil N/3 \rceil$ Auswertungen vornehmen.

7 Evaluierung

Wir stellen uns jetzt die Frage, wie gut sich das eben in Abschnitt 6.3 vorgestellte Verfahren zur globalen Optimierung eignet. Dazu werden wir die Methode auf analytische Beispiele, bei denen man die Zielfunktion explizit aufschreiben kann, und auf reale Beispiele, bei denen das Aussehen der Zielfunktion weitgehend unbekannt ist, anwenden. Wir werden die verschiedenen Optimierungsparameter variieren und das Verfahren auch mit der Optimierung mit stückweise linearen Basisfunktionen und der gradientenfreien, direkten Optimierung der Zielfunktion vergleichen. Soweit nicht anders angegeben, wird als Standard ein Noboundary-Gitter mit modifizierten B-Splines vom Grad $p = 3$ und die Ritter-Novak-Gittererzeugung mit Adaptivität $\alpha = 0{,}85$ verwendet.

7.1 Analytische Beispiele

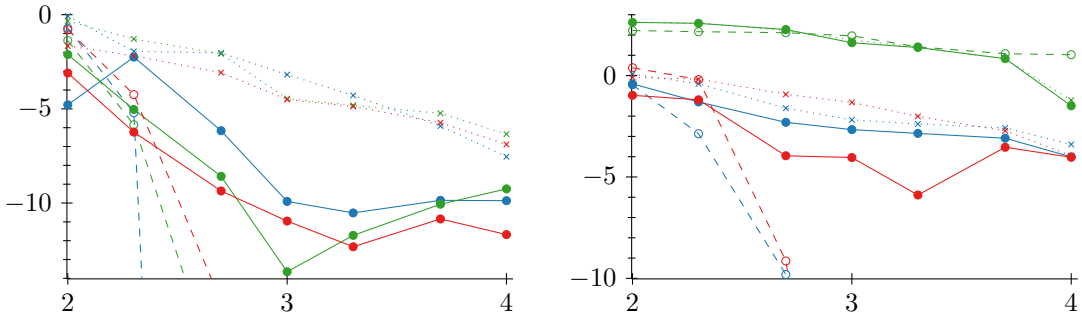
7.1.1 Testfunktionen

In Anhang A werden die Definitionen der analytischen Testfunktionen $f(\vec{x})$ jeweils zusammen mit dem Definitionsbereich $\times_{t=1}^{d}[x_t^{(\mathrm{u})}, x_t^{(\mathrm{o})}]$, der globalen Minimalstelle $\vec{x}_{\mathrm{opt}}$ (manchmal gibt es mehrere) und dem minimalen Funktionswert f_{opt} wiedergegeben. Die Auswahl der Testfunktionen ist eine Mischung zwischen denen, die in [16, 35, 57] angegeben sind. Die Literatur weicht zum Teil von den in Anhang A angegebenen Definitionen ab. Ein paar der Definitionsbereiche der Testfunktionen wurden verändert, sodass sich das Minimum nicht mehr genau in der Mitte befindet (um das Finden des Minimums nicht ganz so leicht zu machen).

Die meisten der Testfunktionen sind zweidimensional. Es sind natürlich auch Testfunktionen in höheren Dimensionen dabei (entweder fest, z. B. $d = 6$, oder allgemein, also $d \in \mathbb{N}$ beliebig). Zu den Testfunktionen, die sich für $d = 2$ ausrechnen lassen, gibt es in Anhang A zusätzlich zwei Plots. Die Plots beziehen sich je nach Testfunktion auf $f(\vec{x})$ oder $\ln(f(\vec{x}) + c)$. Allen Plots ist gemeinsam, dass der Definitionsbereich auf $[0, 1]^d$ skaliert ist und dass die Positionen der globalen Minima durch Kreise markiert sind.

7.1.2 Ergebnisse in zwei Dimensionen

Dargestellt ist in den folgenden Plots der Fehlerlogarithmus $\log_{10}(f(\vec{x}^*_{\mathrm{opt}}) - f_{\mathrm{opt}})$ in Abhängigkeit vom Logarithmus $\log_{10} N$ der Auswertungszahl. Es gibt drei Linien für jede Zielfunktion: Die durchgezogenen, gepunkteten und gestrichelten Linien entsprechen der Optimierung der glatten Interpolierenden, der stückweise linearen Interpolierenden bzw. der Zielfunktion (in der Notation von Abschnitt 6.3 also $\vec{x}^*_{\mathrm{opt}}$, $\vec{x}'_{\mathrm{opt}}$, $\vec{x}''_{\mathrm{opt}}$). In Anhang B.1 können die zugehörigen Daten nachgeschlagen werden.



(a) Br, Hi und Mi **(b)** Ack, Egg und Ro

Abbildung 7.1: Fehlerentwicklung für $d = 2$

In Abb. 7.1 sieht man sechs der Testfunktionen für $d = 2$. Links in Abb. 7.1a sind die Plots für drei vermeintlich „leichte“ Zielfunktionen abgebildet, weil diese Funktionen kleine zweite Ableitungen besitzen. Für solche Funktionen funktioniert die Optimierung der glatten Interpolierenden erwartungsgemäß sehr gut, weil die Funktionen gut durch B-Spline-Linearkombinationen approximiert werden können (auch wenn die Qualität der Ergebnisse auf dem hohen Niveau für größere N etwas nachlässt). Wenn man nur die stückweise lineare Interpolierende optimiert, benötigt man ein Vielfaches an Auswertungen, um dieselbe Qualität zu erreichen. Allerdings schlagen sich die gradientenfreien Verfahren angewendet auf die Zielfunktion noch besser.

Rechts in Abb. 7.1b geht es um drei wesentlich „schwierigere“ Zielfunktionen, die viele lokale Minima haben, große zweite Ableitungen besitzen oder deren Minimum „in einem dünnen Tal versteckt“ ist. Bei der Eggholder-Funktion kommt noch dazu, dass das Minimum auf dem Rand von $[0, 1]^2$ liegt, sodass sich auch die gradientenfreien Verfahren schwer tun. Man kann aber erkennen, dass die Optimierung der glatten Interpolierenden oft einen Vorsprung gegenüber der Optimierung der stückweise linearen Interpolierenden hat.

7.1.3 Ergebnisse in höheren Dimensionen

In Abb. 7.2 ist die Entwicklung des Fehlers für drei Testfunktionen in $d = 3, 4, 6, 10$ Dimensionen sichtbar. Während für kleines d eine relativ schnelle Konvergenz zu beobachten ist, ist die Konvergenzgeschwindigkeit in hohen Dimensionen deutlich langsamer (man beachte die unterschiedlichen Achsenskalierungen). Die Spitzen der durchgezogenen grünen Linie deuten an, dass die Methode für die Rastrigin-Funktion nicht komplett robust ist, sondern ab und zu in einem der vielen lokalen Minima „stecken bleibt“.

Ein Hindernis für die Dünngitter-Optimierung ist bei den ausgewählten Funktionen, dass das globale Optimum auf einer Quaderdiagonalen von $[0, 1]^d$ liegt, nämlich auf der Ursprungsdiagonalen durch $\vec{e}$. Weil die Optimalstellen zusätzlich nicht gerade neben dem Mittelpunkt $0{,}5\vec{e}$ von $[0, 1]^d$ liegen, erfordert es eine größere Zahl an Verfeinerungen, bis Punkte in ihrer Nähe erzeugt wurden. Genauer braucht man $(n - 1)d$ sukzessive Verfeinerungen, um $\vec{x}_{\vec{\ell},\vec{j}} = h_n\vec{e}$ zu erzeugen (mit $(\vec{\ell}, \vec{j}) = (n\vec{e}, \vec{e})$). Für $d = 10$ müsste ein Punkt mit seinen Nachfahren also schon 20 Mal verfeinert werden, nur um in allen Dimensionen

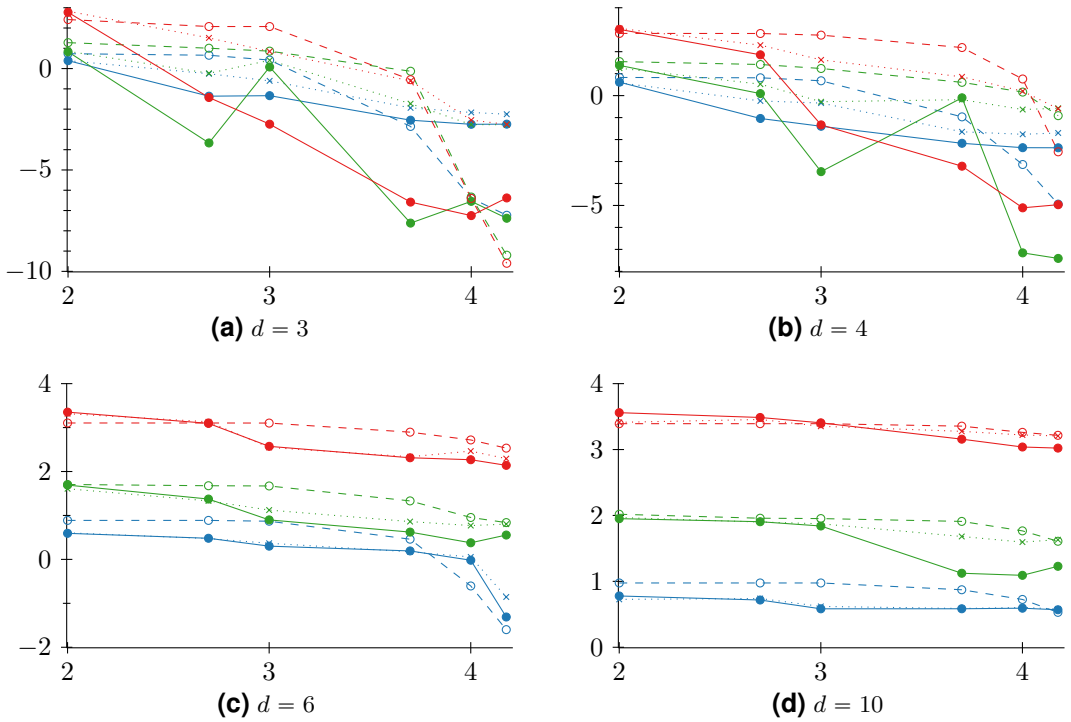


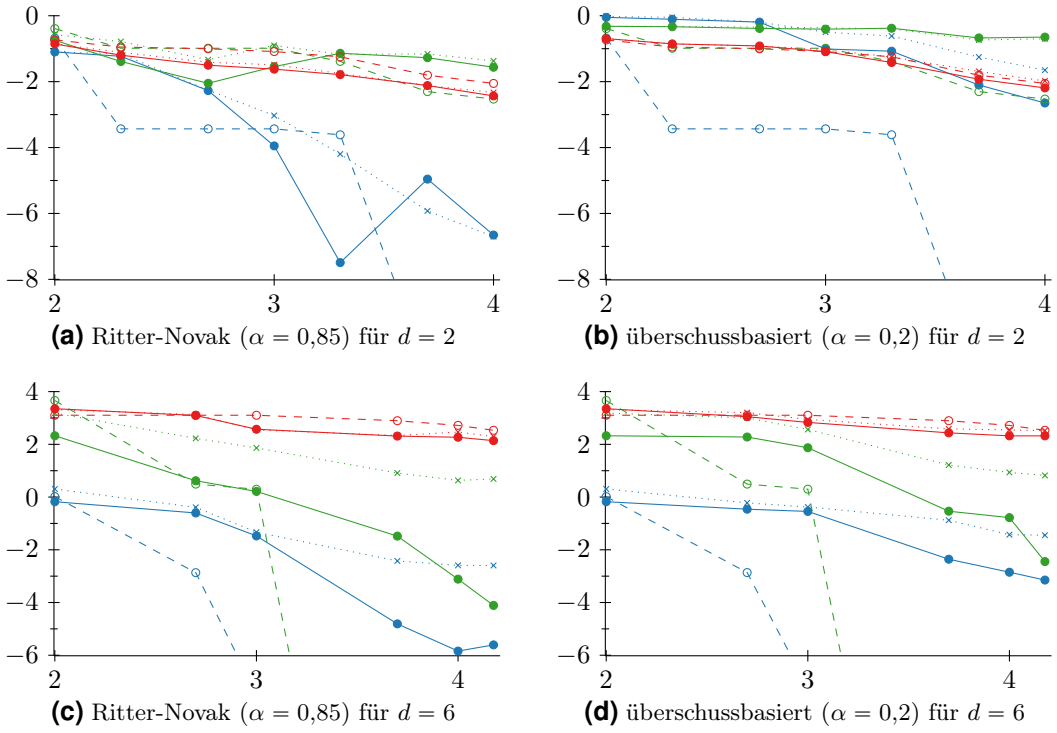
Abbildung 7.2: Fehlerentwicklung für $d = 3, 4, 6, 10$ für Ack, Ra und Sch

Level 3 zu erreichen. Diese Einschränkung liegt in der Natur der dünnen Gitter und ist schwierig zu überwinden (Stichwort „Fluch der Dimensionalität“). Ackley- und Rastrigin-Testfunktion gehören übrigens zu den bereits erwähnten Funktionen, die im Vergleich zur gängigen Literatur extra verschoben wurden, damit es für die dünnen Gitter nicht so leicht ist – in der Literatur liegt das Optimum jeweils bei $0{,}5\vec{e}$. Es bleibt jedenfalls festzuhalten, dass die Methode der Optimierung der glatten Interpolierenden für die dargestellten Funktionen unter den drei vorgestellten Verfahren (Optimierung der stückweise linearen Interpolierenden oder der Zielfunktion) oftmals am besten abschneidet.

7.1.4 Art der Gittererzeugung

Wir vergleichen nun in Abb. 7.3 die Ergebnisse der Ritter-Novak-Gittererzeugung mit der bisher nach [40] bei dünnen Gittern üblichen überschussbasierten Verfeinerung. In Abb. 7.3a und 7.3b kann man z. B. anhand der Beale-Funktion feststellen, dass die Ritter-Novak-Verfeinerung meist substanziell besser arbeitet, weil sie konkret auf potenzielle Minima eingeht, was man schon anhand der gepunkteten Linie (stückweise lineare Interpolierende) erkennt. Die überschussbasierte Verfeinerung ist eher daran interessiert, eine global gute (stückweise lineare) Interpolierende zu konstruieren.

Der Vorteil der Ritter-Novak-Verfeinerung wird in höheren Dimensionen noch deutlicher (Abb. 7.3c und 7.3d). Hier leidet die überschussbasierte Verfeinerung besonders unter den aus algorithmischen Gründen mit zu erzeugenden hierarchischen Vorfahren. Überra-



(a) Ritter-Novak ($\alpha = 0{,}85$) für $d = 2$

(b) überschussbasiert ($\alpha = 0{,}2$) für $d = 2$

(c) Ritter-Novak ($\alpha = 0{,}85$) für $d = 6$

(d) überschussbasiert ($\alpha = 0{,}2$) für $d = 6$

Abbildung 7.3: Fehlerentwicklung für verschiedene Arten der Gittererzeugung und Be, Gr, Ml (Abb. a, b) bzw. H6, Ro, Sch (Abb. c, d)

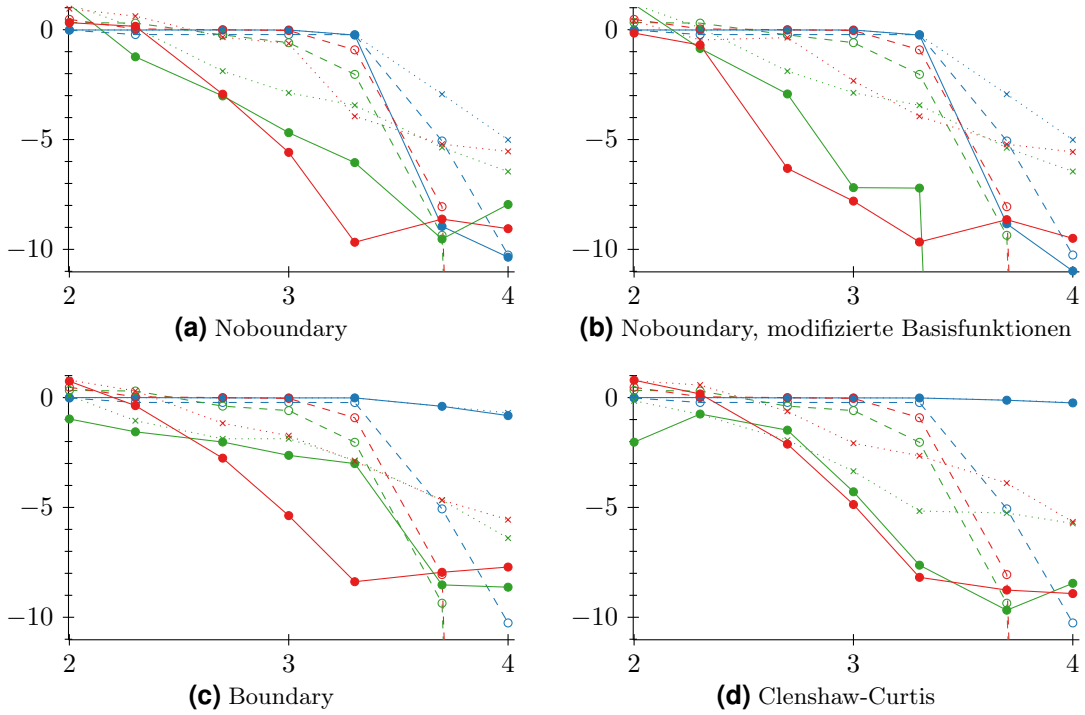
schend ist, dass zusätzlich dazu der durch die Verwendung glatter gegenüber stückweise linearer Interpolierenden entstehende Nutzeffekt (Abstand der durchgezogenen zu den gepunkteten Linien) bei der Ritter-Novak-Verfeinerung wesentlich größer ausfällt als bei der überschussbasierten Verfeinerung.

7.1.5 Verschiedene Gitterarten

In Abb. 7.4 sieht man die Entwicklung des Fehlers für die vier verschiedenen Gitterarten, jeweils für drei zweidimensionale Testfunktionen. Zunächst fällt auf, dass sich die Easom-Testfunktion (blau) erheblich besser durch randlose Gitter optimieren lässt. Das ist aber nicht weiter verwunderlich, da diese Funktion im Wesentlichen eine umgedrehte Glockenkurve darstellt, deren Spitze sehr klein ist und sich in der Mitte von $[0, 1]^2$ befindet. Am Rand verschwindet die Funktion beinahe, sodass zusätzliche Gitterpunkte auf dem Rand so gut wie wirkungslos sind.

Auch bei den anderen Funktionen arbeiten die randbehafteten Gitter (Abb. 7.4c, 7.4d) schlechter als die randlosen, selbst bei der Rastrigin-Funktion, deren Minimum eher in Richtung Rand liegt. Das liegt an der in Abschnitt 2.2.3 erläuterten Problematik, dass der Rand von (randbehafteten) dünnen Gittern anteilsmäßig übermäßig viele Punkte stellt.

Vergleicht man Noboundary ohne und mit modifizierten Basisfunktionen (Abb. 7.4a vs. 7.4b), so stellt man fest, dass die Modifikation der Basisfunktionen teilweise einen deut-



(a) Noboundary **(b)** Noboundary, modifizierte Basisfunktionen

(c) Boundary **(d)** Clenshaw-Curtis

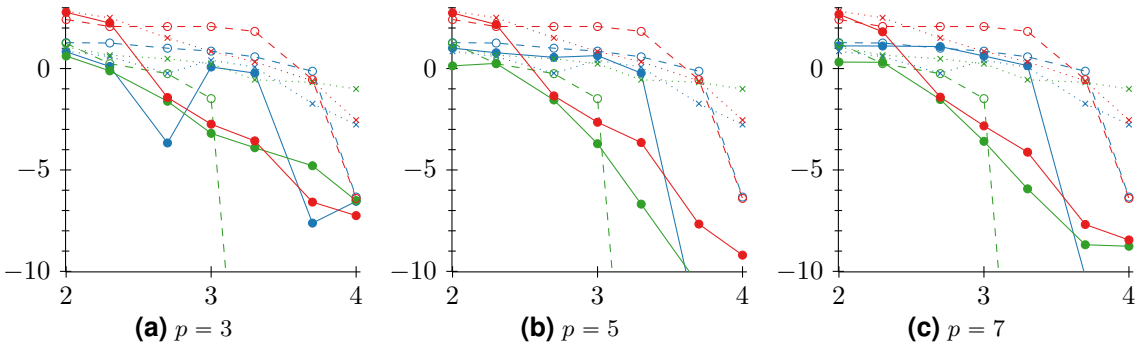
Abbildung 7.4: Fehlerentwicklung für verschiedene Gitter, $d = 2$ und Ea, Hö, Ra

lichen Vorteil bietet. Ohne Modifikation fallen die Interpolierenden am Rand in Richtung 0 ab (sind aber nicht wie im stückweise linearen Fall gleich 0), was schlecht ist, wenn die Funktionen wie hier dort nicht verschwinden. Auch der Vorsprung der glatten Interpolierenden gegenüber den stückweise linearen Interpolierenden wird durch die Modifikation größer, sodass B-Splines offensichtlich ganz besonders davon profitieren.

Dass die Hölder-Tisch-Funktion (grün) besser durch Clenshaw-Curtis-Gitter als durch Boundary-Gitter optimiert werden kann, liegt übrigens daran, dass deren Minima nahe den Ecken von $[0, 1]^2$ liegen und Clenshaw-Curtis-Gitter bei der Verfeinerung den Rand „begünstigen“. Insgesamt kann man sagen, dass Noboundary-Gitter mit modifizierten Basisfunktionen die beste Qualität liefern, was auch für andere Funktionen und andere Dimensionalitäten d gilt.

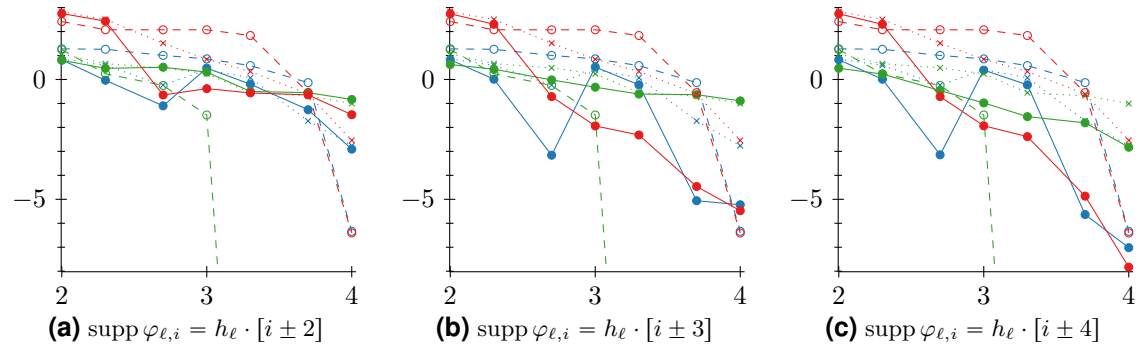
7.1.6 Wahl der Basisfunktionen

Wir betrachten jetzt in Abb. 7.5 die Abhängigkeit der Fehlerentwicklung vom Grad p der verwendeten B-Splines. Für die Rastrigin-Testfunktion (blau) lassen sich durch höheren Grad bei moderatem N keine Verbesserungen erzielen, während Rosenbrock- und Schwefel-Funktion (grün bzw. rot) im Vergleich zu $p = 3$ besser abschneiden. Das liegt daran, dass die letztgenannten, „glatteren“ Funktionen im Vergleich zur erstgenannten weniger Oszillationen aufweisen. Linearkombinationen von B-Splines höherer Grade sind öfters stetig differenzierbar, weswegen diese „glattere“ Funktionen besser approximieren können.



(a) $p = 3$ **(b)** $p = 5$ **(c)** $p = 7$

Abbildung 7.5: Fehlerentwicklung für verschiedene B-Spline-Grade, $d = 3$ und Ra, Ro, Sch



(a) $\operatorname{supp} \varphi_{\ell,i} = h_\ell \cdot [i \pm 2]$ **(b)** $\operatorname{supp} \varphi_{\ell,i} = h_\ell \cdot [i \pm 3]$ **(c)** $\operatorname{supp} \varphi_{\ell,i} = h_\ell \cdot [i \pm 4]$

Abbildung 7.6: Fehlerentwicklung für Mexican-Hat-Basisfunktionen mit verschiedenen Abschneidestellen, $d = 3$ und Ra, Ro, Sch

Nun schauen wir uns in Abb. 7.6 beispielhafte Ergebnisse für Mexican-Hat-Basisfunktionen an. Die verwendeten Testfunktionen sind identisch mit denen aus Abb. 7.5, um einen Vergleich mit B-Splines zu ermöglichen. In Abschnitt 6.2.3 wurde erwähnt, dass wir bei der Implementierung die Mexican-Hat-Basisfunktionen nach zwei Intervallen „abschneiden“, damit das Gleichungssystem bei der Hierarchisierung mehr Nullen enthält. Wie man im Vergleich von Abb. 7.5a mit Abb. 7.6a aber sieht, hat das schwerwiegende Auswirkungen auf das Ergebnis der gradientenbasierten Optimierung – die Mexican-Hat-Basisfunktionen schneiden viel schlechter ab als ihre B-Spline-Pendants. Der Grund liegt natürlich in der fehlenden Stetigkeit der sich ergebenden Interpolierenden.

Schneidet man erst nach drei oder vier Intervallen ab (Abb. 7.6b bzw. 7.6c), so ist dieser Effekt wesentlich schwächer und die Mexican-Hat-Basisfunktionen werden deutlich besser. Allerdings ist als Preis ein stark gestiegener Aufwand zu bezahlen: Benötigt ein kompletter Programmdurchlauf für $N = 10000$ beim Abschneiden nach zwei Intervallen noch durchschnittlich 8,3 s und 101 MB Speicher, so sind es bei drei Intervallen bereits 23,5 s und 1,8 GB und bei vier Intervallen sogar 26,3 s und 2,2 GB. Das ist im Vergleich zu B-Splines unverhältnismäßig viel (siehe nächster Abschnitt), weswegen wir die Mexican-Hat-Basisfunktionen trotzdem nach zwei Intervallen abschneiden.

Tabelle 7.1: Durchschnittlicher Zeit- und Speicherbedarf für $d = 2$

N	100	200	500	1000	2000	5000	10000
Gittererzeugung	65 ms	66 ms	76 ms	98 ms	176 ms	704 ms	2,6 s
Hierarchisierung	3 ms	5 ms	12 ms	54 ms	178 ms	790 ms	2,8 s
Optimierung	74 ms	67 ms	95 ms	152 ms	261 ms	504 ms	833 ms
Gesamtzeit	142 ms	138 ms	184 ms	304 ms	615 ms	2,0 s	6,2 s
Speicherbedarf	20 MB	20 MB	23 MB	34 MB	57 MB	63 MB	130 MB

Tabelle 7.2: Durchschnittlicher Zeit- und Speicherbedarf für $d = 6$

N	100	500	1000	5000	10000	15000
Gittererzeugung	13 ms	67 ms	83 ms	285 ms	917 ms	2,0 s
Hierarchisierung	24 ms	14 ms	48 ms	1,8 s	11,6 s	36,1 s
Optimierung	81 ms	131 ms	599 ms	2,8 s	5,3 s	9,8 s
Gesamtzeit	117 ms	212 ms	730 ms	4,9 s	17,8 s	47,9 s
Speicherbedarf	21 MB	22 MB	37 MB	420 MB	1,6 GB	3,5 GB

7.1.7 Zeit- und Speicherbedarf

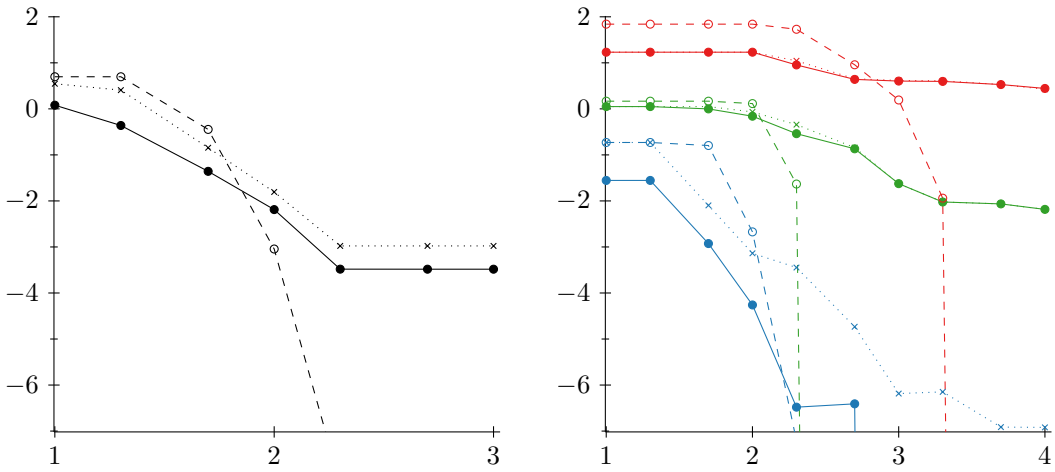
Tabelle 7.1 und Tabelle 7.2 zeigen den Zeit- und Speicheraufwand, den das Programm bei der Optimierung mithilfe der glatten Interpolierenden betreibt (Durchschnitte über alle Testfunktionen der jeweiligen Dimension, Spezifikationen des Rechners siehe Anfang von Kapitel 6). Die aufgeführte Gesamtzeit unterteilt sich in die drei in Abschnitt 6.3.1 erklärten Schritte „Gittererzeugung“, „Hierarchisierung“ und „Optimierung“.

Für $d = 2$ und kleine N sind Gittererzeugung und Optimierung zeitlich dominant, während der Aufwand der Hierarchisierung erwartungsgemäß für größere N schnell wächst. Die Gesamtzeit hält sich aber wie der benötigte Speicher in Grenzen.

Bei höheren Dimensionalitäten wie $d = 6$ geht die Gittererzeugung schneller vonstatten, weil pro Iteration mehr Gitterpunkte erzeugt werden und die Obergrenze N der Gitterpunkte-Zahl schneller erreicht wird. Hierarchisierung und Optimierung brauchen dagegen deutlich länger. Bei der Hierarchisierung liegt das daran, dass das zu lösende lineare Gleichungssystem eine vollere Besetzungsstruktur besitzt. Dies schlägt sich auch im viel größeren Speicherbedarf nieder. Die längere Laufzeit der Optimierung lässt sich mit der gestiegenen Zahl an 1D-Basisauswertungen begründen.

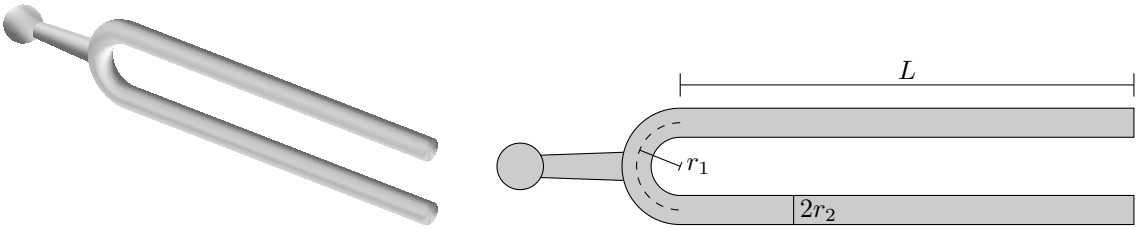
7.2 Reale Beispiele

Nun wollen wir uns „realen“ Zielfunktionen zuwenden, also Zielfunktionen, die konkret aus Anwendungen folgen. Aus Platzgründen werden wir nicht wie eben für jedes Beispiel Kombinationen von verschiedenen Optimierungsparametern ausprobieren, sondern nur die Anwendbarkeit der Standard-Parameter (Noboundary-Gitter, modifizierte B-Splines vom Grad 3 und Ritter-Novak-Gittererzeugung mit $\alpha = 0{,}85$) testen. Abb. 7.7 zeigt vorab analog zum vorherigen Abschnitt die Entwicklung des Fehlers der vier folgenden Anwendungsbeispiele. Wir werden bei jedem Beispiel noch einmal gesondert darauf eingehen.



(a) Stimmgabel **(b)** Motor, Weltbevölkerung, Lotka-Volterra

Abbildung 7.7: Fehlerentwicklung für die vier „realen“ Beispiele



(a) 3D-Modell **(b)** Seitenriss mit ausgewählten Parametern

Abbildung 7.8: Stimmgabel-Modell aus [9]

7.2.1 Stimmgabel ($d = 1$)

Als erstes Beispiel verwenden wir das bei der Finite-Elemente-Software (FE-Software) COMSOL mitgelieferte Beispiel-Modell [9] einer Stimmgabel. Obwohl es sich recht banal anhört, werden laut [9] „mikroskopisch kleine Stimmgabeln auch bei Quarzuhren und anderen elektronischen Geräten“ genutzt.

Die Stimmgabel ist in Abb. 7.8 abgebildet und insgesamt ca. 12 cm lang und 2 cm breit. Der Radius des runden Teils, der die beiden zylinderförmigen Zinken verbindet, und der Radius der Zinken selbst betragen $r_1 = 7{,}5\,\text{mm}$ bzw. $r_2 = 2{,}5\,\text{mm}$. Alle anderen Parameter wie Form und Material seien konstant, z. B. besteht die Stimmgabel aus dem normierten Stahl „AISI 4340“. Nur die genaue Länge L der Zinken ist variabel und soll so gewählt werden, dass die Stimmgabel auf den Standard-Kammerton a’ (440 Hz) gestimmt ist. Die eindimensionale Zielfunktion f ordnet einer Länge L den Betrag der Differenz zwischen der zugehörigen ersten Eigenfrequenz ν (Frequenz der Grundschwingung) und 440 Hz zu.

Bei der Optimierung kommuniziert das C++-Hauptprogramm mit einem in MATLAB geschriebenen kleinen Server, der auf einem Universitätsrechner läuft und zur Simulation COMSOL aufruft. Eine Auswertung dauert ca. acht Sekunden, wobei der Hauptanteil

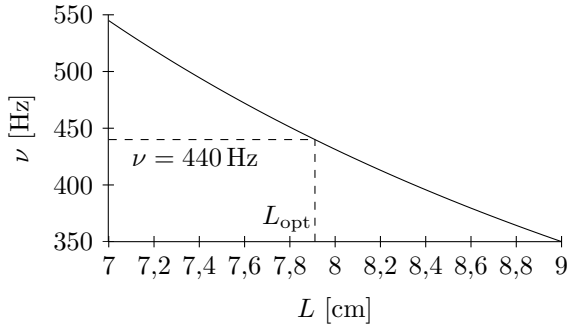


Abbildung 7.9: Haupt-Eigenfrequenz der Stimmgabel in Abhängigkeit von der Länge

der Simulationsteil ist (der Mehraufwand durch die Kommunikation ist vergleichsweise klein). Dies ist der Grund, warum das Modell möglichst einfach gehalten wurde – bei mehr Parametern oder einem anspruchsvolleren Modell hätte die noch längere Simulationsdauer das Experimentieren z. B. mit verschiedenen Optimierungsparametern deutlich erschwert.

Was zur Optimierung noch fehlt, ist ein sinnvoller Parameterbereich für L. Einen solchen Bereich könnte man z. B. experimentell herleiten. Wir verwenden stattdessen wie [9] die theoretische Abschätzung

$$\nu \approx \frac{1{,}875^2}{2\pi L_1^2}\sqrt{\frac{EI}{\rho A}} \tag{7.1}$$

aus [23]. Dabei ist L_1 die Gesamtlänge eines Zinkens der Stimmgabel, E und ρ der Elastizitätsmodul bzw. die Dichte (beides Materialkonstanten), I das Flächenträgheitsmoment und A die Querschnittsfläche der Stimmgabel, also

$$L_1 = L + \frac{\pi}{2}r_1, \quad E = 205\,\mathrm{GPa}, \quad \rho = 7850\,\frac{\mathrm{kg}}{\mathrm{m}^3}, \quad I = \frac{\pi}{4}r_2^4, \quad A = \pi r_2^2 \tag{7.2}$$

(Materialkonstanten und kreisförmiger Querschnitt). Setzt man die Werte in (7.1) ein und löst mit $\nu = 440\,\mathrm{Hz}$ nach L auf, so bekommt man $L \approx 7{,}83\,\mathrm{cm}$ heraus. Laut [9] ist die wahre Länge etwas größer, weil der untere Teil der Zinken an der „Basis“ nicht so flexibel ist und daher nicht so gut mitschwingen kann. Dies motiviert die Wahl des Parameterbereiches

$$L \in [7\,\mathrm{cm}, 9\,\mathrm{cm}]. \tag{7.3}$$

In Abb. 7.9 sieht man die durch Simulation erhaltene Eigenfrequenz ν in Abhängigkeit von L (der Betrag von der Differenz zu 440 Hz ist die Zielfunktion). Wenn man schon vor der Optimierung weiß, dass der Zusammenhang monoton ist, dann wäre eine einfache Bisektion wesentlich effizienter. Allerdings wollen wir das in dieser Arbeit entwickelte Optimierungsverfahren testen und „ignorieren“ deswegen Abb. 7.9. Die optimale Länge ist

$$L_{\mathrm{opt}} = 7{,}9107\,\mathrm{cm} \tag{7.4}$$

und damit tatsächlich wie bereits erwähnt etwas größer als obige Schätzung.

Mit den Standard-Optimierungsparametern erhält man den in Abb. 7.7a dargestellten Plot. Bereits mit $N = 20$ Gitterpunkten weicht die Eigenfrequenz ν um weniger als 1 Hz von der Zielfrequenz 440 Hz ab. Für diese geringe Anzahl an Gitterpunkten ist die Optimierung der glatten Interpolierenden sowohl besser als die Optimierung der stückweise linearen Interpolierenden als auch besser als die gradientenfreie Optimierung der Zielfunktion.

Für größere N gibt es zwei systembedingte Tatsachen, die den Vergleich etwas verfälschen: Zum einen wurde das unbekannte, „tatsächliche Optimum“, zu dem die Fehler der Optimierungen bestimmt werden, selbst mit dem Nelder-Mead-Verfahren ermittelt, womit der Fehler der gradientenfreien Optimierung (gestrichelte Linie) irgendwann auf 0 springt. Zum anderen arbeitet COMSOL nicht beliebig genau und kann die Frequenz für eine bestimmte Länge nicht exakt angeben, sondern nur in winzig kleinen Sprüngen. Das ist der Grund, warum die dünngitterbasierten Optimierungen (durchgezogene und gepunktete Linie) ab $N = 200$ keine Verbesserungen mehr erzielen. Die Länge, die man durch die Dünngitteroptimierung bei $N = 200$ erhält, unterscheidet sich aber nur noch im Nanometerbereich von L_{opt}. Dadurch weicht die Frequenz der Stimmgabel nur um einen Bruchteil eines Millihertz von 440 Hz ab.

7.2.2 Gleichstrom-Motor ($d = 2$)

Wir betrachten einen einfachen Gleichstrom-Motor nach [34], der sich beim Anlegen einer Gleichspannung nach einer Einschwingphase gleichmäßig dreht. Im Folgenden seien θ die Winkelposition in rad (Bogenmaß) und $\omega = \dot{\theta}$ die Winkelgeschwindigkeit in rad/s. Wie in [32] hergeleitet wird, kann der idealisierte Motor unter Vernachlässigung von Störungen durch das lineare Zustandsraummodell

$$\begin{aligned} \dot{\theta}(t) &= \omega(t), & \theta(0) &= \theta_0, \\ \dot{\omega}(t) &= -\frac{1}{\tau}\omega(t) + \frac{k}{\tau}U(t), & \omega(0) &= \omega_0, \end{aligned} \tag{7.5}$$

mit Zustand und Ausgang $(\theta, \omega)^{\mathrm{T}}$ und Eingang U modelliert werden, wobei U die angelegte Spannung in Volt und τ und k vom Motor abhängige Konstanten bezeichnen („Anlaufzeitkonstante“ τ in Sekunden und „Drehzahlkonstante“ $k/(2\pi)$, wobei rad/(V s) die Einheit k ist). Für konstante Spannungen $U(t) \equiv U_0$ kann man die Lösung dieses Systems gewöhnlicher Differentialgleichungen (ODE-System) leicht direkt berechnen, indem man zunächst die zweite Gleichung löst und danach die erste. Man erhält dann

$$\begin{aligned} \theta(t) &= -c_2\tau \mathrm{e}^{-t/\tau} + kU_0 t + c_1, \\ \omega(t) &= c_2 \mathrm{e}^{-t/\tau} + kU_0, \end{aligned} \qquad c_1 := \theta_0 + c_2\tau, \quad c_2 := \omega_0 - kU_0. \tag{7.6}$$

Die Motorkonstanten τ und k stellen also jeweils die Einschwingzeit ($\tau \ln 2$ ist die Zeit, in der sich der Einschwinganteil von ω halbiert) und die Verstärkung des Systems (kU_0 ist die maximale Winkelgeschwindigkeit) dar. Diese Konstanten sind vom Motor abhängig, jedoch a priori nicht bekannt und müssen daher aus Messdaten des Motors experimentell bestimmt werden.

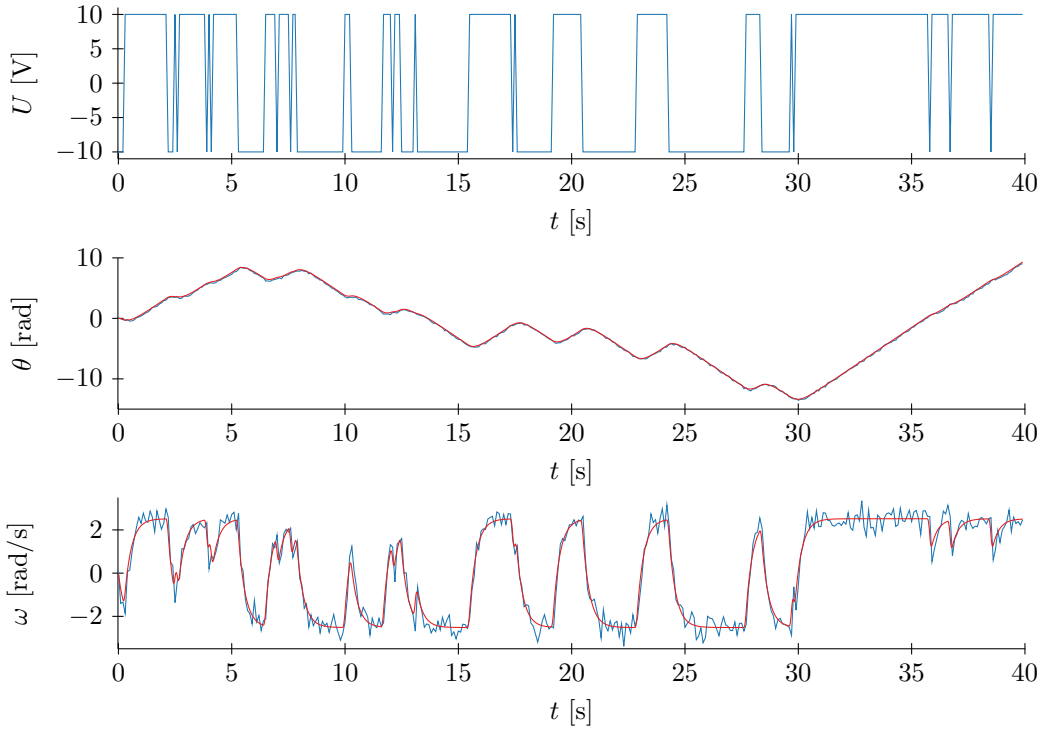


Abbildung 7.10: Eingangsspannung U, Winkelposition θ und Winkelgeschwindigkeit ω der Testdaten aus [34] und optimiertes Modell

In der System Identification Toolbox von MATLAB R2014a befinden sich Testdaten eines solchen Motors (siehe [34], Datei `iddemos/data/dcmotordata.mat`). Die mit 10 Hz abgetasteten Daten sind in Abb. 7.10 als blaue Linien dargestellt. Die angelegte Spannung U ist abschnittsweise konstant, weswegen θ und ω sich abschnittsweise nach (7.6) richten, doch es wären auch kompliziertere Spannungsverläufe möglich. Die Anfangswerte sind bekannt als $\theta_0 = 0\,\mathrm{rad}$, $\omega_0 = 0\,\mathrm{rad/s}$. Wir wollen nun die beiden Parameter τ und k so bestimmen, dass die Lösung $(\theta, \omega)^{\mathrm{T}}$ von (7.5) möglichst „gut“ mit den Messdaten $(\theta_k, \omega_k)^{\mathrm{T}}$ übereinstimmt ($k = 1, \ldots, m$). Als Maß für die „Gutheit“ wählen wir die ℓ^2-Norm

$$\left(\sum_{k=1}^{m} \Big((\theta(t_k) - \theta_k)^2 + (\omega(t_k) - \omega_k)^2 \Big) \right)^{1/2}, \tag{7.7}$$

wobei m die Anzahl der Samples und t_k die Sample-Zeitpunkte angeben. Man könnte auch bei Bedarf den Fehler von θ oder ω stärker gewichten oder nur einen der Fehler betrachten.

Vernünftige Parameterbereiche für τ und k bekommt man aus den Daten wie folgt: Den τ-Bereich kann man herleiten, indem man die Einschwingzeit der Daten nach einem der Spannungswechsel anschaut. Die „Halbwertszeit“ $\tau \ln 2$ des Einschwinganteils beträgt ca. 0,2 s, was zu $\tau \approx 0{,}289\,\mathrm{s}$ führen würde. Daher setzen wir den τ-Bereich z. B. auf

$$\tau \in [0{,}1\,\mathrm{s}; 0{,}5\,\mathrm{s}]. \tag{7.8}$$

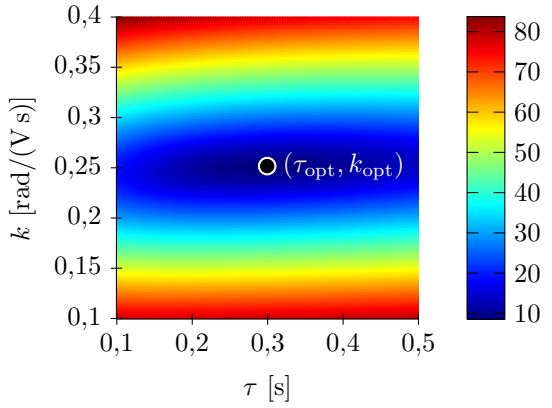


Abbildung 7.11: Zielfunktion des Gleichstrom-Motors (ohne Skalierung auf $[0,1]^2$)

Für den k-Bereich bemerkt man, dass ω_k Werte von maximal ca. 2,5 rad/s annimmt. Die maximal mögliche Winkelgeschwindigkeit kU_0 von $\omega(t)$ liegt also höchstwahrscheinlich bspw. zwischen 1 rad/s und 4 rad/s. Dies führt mit $U_0 = 10\,\text{V}$ zur Annahme von

$$k \in \left[0{,}1\,\frac{\text{rad}}{\text{V s}}; 0{,}4\,\frac{\text{rad}}{\text{V s}}\right]. \tag{7.9}$$

Die Zielfunktion ist in Abb. 7.11 abgebildet und die optimale Parameterkombination ist

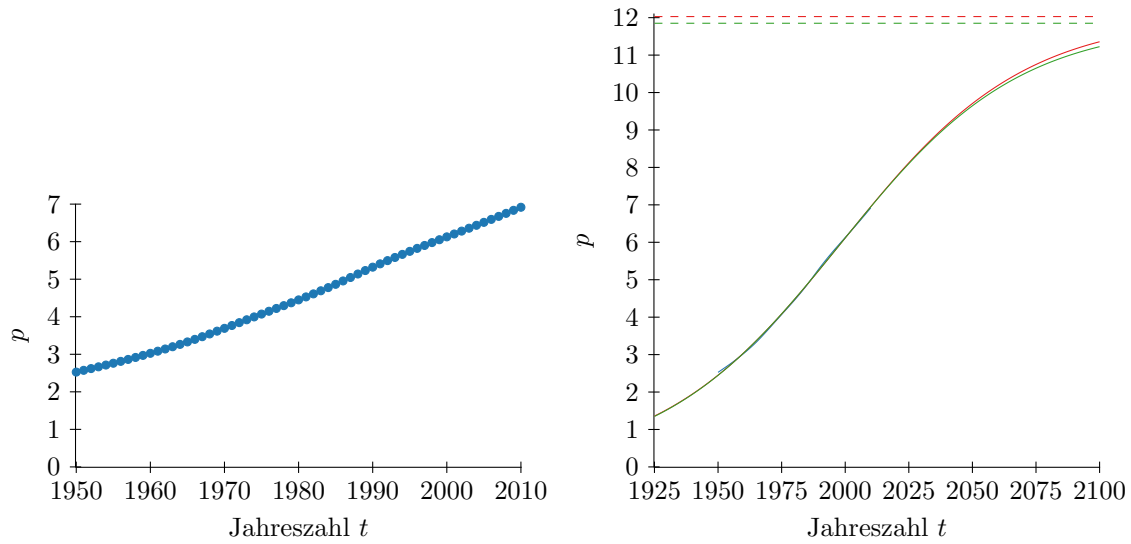
$$\tau_{\text{opt}} = 0{,}299\,351\,\text{s}, \quad k_{\text{opt}} = 0{,}251\,672\,\frac{\text{rad}}{\text{V s}}. \tag{7.10}$$

Die Winkelpositionen und -geschwindigkeiten, die man bei Simulation des Modells mit diesen Parametern erhält, sind in Abb. 7.10 als rote Linien dargestellt.

In Abb. 7.7b sieht man in Blau die Fehlerentwicklung bei Anwendung von Dünngitteroptimierung auf dieses Beispiel. Wir haben bei der Wahl der Parameterbereiche ein glückliches Händchen bewiesen: Das Optimum liegt sehr nahe der Mitte der Bereiche – weil dünne Gitter „hauptachsenlastig“ sind (Hauptachsen sind Achsenparallelen durch $0{,}5\vec{e}$), wird dort mit nur wenigen Verfeinerungen eine hohe Genauigkeit erzielt. Zudem ist die Zielfunktion wie in Abb. 7.11 zu sehen leicht zu optimieren, da es nur ein lokales Minimum gibt und die Funktion einer einfachen quadratischen Funktion stark ähnelt. Erfreulich ist auch, dass die Optimierung mit B-Splines einen klaren Vorteil gegenüber der stückweise linearen Optimierung darstellt. Ab $N = 1000$ hat die Optimierung der glatten Interpolierenden das Minimum mit obigen zugehörigen Parametern $(\tau_{\text{opt}}, k_{\text{opt}})$ sogar exakt gefunden.

7.2.3 Weltbevölkerung ($d = 3$)

Die Anzahl der Menschen auf der Welt wächst immer weiter, momentan leben ca. 7,2 Milliarden Menschen auf der Erde. Ab dem 19. Jahrhundert stieg das Bevölkerungswachstum stark an. Natürlich können aufgrund begrenzter Ressourcen nicht beliebig viele Menschen auf dem Globus leben, die Zahl ist also nach oben beschränkt. Es stellt sich nun die Frage: Wie wird sich die Bevölkerungszahl in Zukunft entwickeln?



(a) Datenpunkte aus [52]

(b) UN-Daten, Optimum durch Interpolation und tatsächliches Optimum mit Asymptoten

Abbildung 7.12: Entwicklung der Weltbevölkerungszahl $p(t)$ in Mrd.

In Abb. 7.12a sieht man Schätzungen dieser Zahl von den Vereinten Nationen (UN) aus [52] für die Jahre 1950 – 2010. Bemerkenswert ist, dass die Daten selbst tatsächlich auch nur Schätzungen sind, weil laut Angaben der UN ca. die Hälfte der über 200 Länder und Gebiete keine hinreichend genauen Bevölkerungsstatistiken erheben oder weitergeben.

Als einfaches kontinuierliches Modell für die Bevölkerungszahl $p(t)$ in Mrd. (t Jahreszahl) verwenden wir logistisches Wachstum (Herleitung siehe [7]): $p(t)$ wächst für kleine Bevölkerungszahlen exponentiell und erreicht wegen beschränkter Ressourcen eine Sättigung für große Populationen. $p(t)$ erfüllt damit die Differentialgleichung (DGL)

$$\dot{p}(t) = (a - bp(t)) \cdot p(t), \quad p(t_0) = p_0, \quad a, b > 0, \tag{7.11}$$

mit Lösung

$$p(t) = \frac{a}{b + (a/p_0 - b) \cdot \mathrm{e}^{-a(t-t_0)}}. \tag{7.12}$$

Wir setzen $t_0 := 1950$ und suchen Werte für die Parameter a, b und p_0, was zu einem 3D-Problem führt. p_0 ist unbekannt, um dem Modell mehr Flexibilität zu verleihen.

Für die Herleitung sinnvoller Parameterbereiche betrachten wir das asymptotische Verhalten und die Wendestelle t^* von $p(t)$ durch

$$\lim_{t\to\infty} p(t) = \frac{a}{b}, \tag{7.13}$$

$$\ddot{p}(t^*) = 0 \iff t^* = t_0 + \frac{1}{a}\ln\left(\frac{a}{bp_0} - 1\right) \iff b = \frac{a}{p_0(1 + \mathrm{e}^{a(t^*-t_0)})}. \tag{7.14}$$

(7.14) für b in (7.13) eingesetzt ergibt für den Sättigungswert $p_\infty > 0$

$$\lim_{t\to\infty} p(t) = p_\infty \iff a = \frac{1}{t^* - t_0} \ln\left(\frac{p_\infty}{p_0} - 1\right). \tag{7.15}$$

Sinnvollerweise setzen wir z. B. $p_0 \in [2, 3]$ (weil laut UN 1950 ca. 2,5 Mrd. Menschen auf der Erde waren) und schätzen aufgrund der Daten, dass $t^* = 1990$. Außerdem sei bspw. $p_\infty \in [8, 30]$ (sehr konservative Wahl). Dann erhält man aus (7.15) als kleinst- und größtmöglichen Wert für a ca. 0,013 bzw. ca. 0,066 und mit diesen Werten aus (7.14) als kleinst- und größtmöglichen Wert für b ca. 0,0016 bzw. ca. 0,0022. Mit einer sicherheitshalben Vergrößerung der Parameterbereiche motiviert dies die Wahl von

$$a \in [0{,}005; 0{,}1], \quad b \in [0{,}0005; 0{,}01], \quad p_0 \in [2, 3]. \tag{7.16}$$

Als Maß für die Approximationsgüte verwenden wir wie eben die ℓ^2-Norm

$$\left(\sum_{k=1}^{m} (p(t_k) - p_k)^2\right)^{1/2}, \tag{7.17}$$

wobei (t_k, p_k) die UN-Daten und $p(t_k)$ die Lösung (7.12) zu den Zeitpunkten $t_k = 1949 + k$, $k = 1, \ldots, m$, darstellt ($m = 61$).

Die optimalen Parameter sind durch

$$a_{\text{opt}} = 0{,}027\,931\,765, \quad b_{\text{opt}} = 0{,}002\,322\,043, \quad p_{0,\text{opt}} = 2{,}450\,997 \tag{7.18}$$

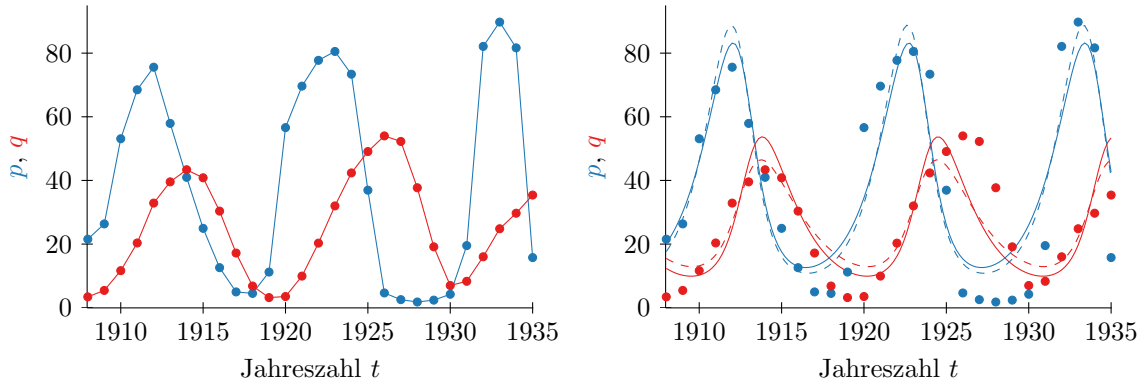
gegeben. Die entsprechende Lösung ist zusammen mit den UN-Schätzungen in Abb. 7.12b rot eingezeichnet. Nach diesem ziemlich naiven Modell würde sich die Bevölkerungszahl langfristig bei $a_{\text{opt}}/b_{\text{opt}} \approx 12{,}03$ Milliarden einpendeln.

Wenn man sich in Abb. 7.7b anschaut, wie gut sich die Dünngitteroptimierung schlägt, so stellt man fest, dass ab $N = 2000$ mit 10^{-2} ein akzeptabler Fehler im Vergleich zum Optimum erreicht wird. Mehr Gitterpunkte bringen keine weiteren Verbesserungen mehr. Die direkte, gradientenfreie Optimierung der Zielfunktion ist wie oben erwähnt systembedingt wieder im Vorteil, weil durch sie das Optimum der Zielfunktion festgelegt wurde. Was negativ auffällt, ist, dass die Optimierung der glatten Interpolierenden hier kaum etwas gegenüber der Optimierung der stückweise linearen Interpolierenden bringt. Das könnte darauf hindeuten, dass die Zielfunktion nicht sonderlich glatt ist.

Durch die Dünngitteroptimierung erhält man bei $N = 10000$ die Parameter

$$a^*_{\text{opt}} = 0{,}028\,227\,741, \quad b^*_{\text{opt}} = 0{,}002\,381\,943, \quad p^*_{0,\text{opt}} = 2{,}445\,516. \tag{7.19}$$

Mit diesen Parametern erhält man die in Abb. 7.12b grün dargestellte Kurve. Wie man sieht (oder eben nicht), unterscheidet sich diese Kurve im Daten-Zeitraum 1950 – 2010 kaum von der, die zum tatsächlichen Minimum gehört. Allerdings ändert sich das langfristige Verhalten stärker, da sich die Bevölkerungszahl mit diesen Parametern asymptotisch nur noch $a^*_{\text{opt}}/b^*_{\text{opt}} \approx 11{,}85$ Milliarden annähert. Das zeigt, dass das Modell bzgl. kleiner Parameteränderungen empfindlich reagiert und instabil ist.



(a) Datenpunkte aus [33] (via [24])

(b) Vergleich Datenpunkte und LV-Modell (durchgezogen: Optimum durch Interpolation, gestrichelt: tatsächliches Optimum)

Abbildung 7.13: Pelzzahlen von Kanadischen Luchsen und Schneeschuhhasen, jeweils in Tsd.

7.2.4 Räuber-Beute-Modell ($d = 6$)

Räuber-Beute-Modelle gehören zu den sogenannten Zwei-Spezies-Modellen der Populationsdynamik. Es geht jetzt also um die Größen zweier Populationen von „Räubern“ und „Beutetieren“, die zueinander in Konkurrenz treten (Herleitung siehe [7]). Es muss sich aber keineswegs um Tiere handeln, ein ähnliches Phänomen ist auch als Schweinezyklus in der Wirtschaft (Angebot – Nachfrage) oder bei Studierendenzahlen (Studierende – freie Arbeitsplätze) bekannt.

Als Datengrundlage verwenden wir die in der Literatur (z. B. [21, 24]) oft verwendeten Daten von MacLulich aus [33] (genaue Werte von [24]). Die kanadische Hudson's Bay Company hat zwischen 1845 und 1935 die Anzahlen der jährlich neu gehandelten Pelze von Kanadischen Luchsen und Schneeschuhhasen festgehalten. MacLulich hat diese Daten aufbereitet und von den Pelzzahlen auf die Populationen geschlossen. Die Genauigkeit der Daten wird zwar angezweifelt (Genaueres dazu siehe [15, 21]), was ihrer Popularität jedoch keineswegs geschadet hat. Wir beschränken uns hier auf die Jahre 1908 bis 1935, die zugehörigen Daten sind in Abb. 7.13a dargestellt.

Wir benutzen zur Modellierung des Zusammenhangs zwischen Räuber und Beute das einfache Lotka-Volterra-Modell (LV-Modell, Herleitung siehe [7]). Die Populationen $p(t)$ und $q(t)$ von Beute (Schneeschuhhase) bzw. Räuber (Kandadischer Luchs) erfüllen darin das System gewöhnlicher Differentialgleichungen

$$\begin{aligned} \dot{p}(t) &= p(t) \cdot (\alpha - \beta q(t)), \quad & p(t_0) &= p_0, \\ \dot{q}(t) &= -q(t) \cdot (\gamma - \delta p(t)), \quad & q(t_0) &= q_0, \end{aligned} \tag{7.20}$$

mit unbekannten Parametern $\alpha, \beta, \gamma, \delta > 0$, die den Einfluss der Größen auf sie selbst oder auf die jeweils andere festlegen. Wie bei der Weltbevölkerung sei $t_0 := 1908$ fix, aber $p_0, q_0 > 0$ unbekannt. Wir erhalten also ein sechsdimensionales Problem.

Sinnvolle Parameterbereiche kann man, ähnlich wie bei den vorangegangenen Beispielen, auf analytischem oder numerischem Wege begründen. Wir beschränken uns auf die bloße Angabe:

$$\begin{gathered}\alpha \in [0{,}2; 2], \quad \beta \in [0{,}005; 0{,}05], \quad \gamma \in [0{,}1; 1], \quad \delta \in [0{,}005; 0{,}05], \\ p_0 \in [10, 35], \quad q_0 \in [1, 20].\end{gathered} \tag{7.21}$$

Wiederum benutzen wir die Wurzel der Fehlerquadratsumme ähnlich wie in (7.17) als Fehlermaß des Modells. Das optimale Modell mit den Parametern

$$\begin{gathered}\begin{pmatrix}\alpha_{\text{opt}} & \beta_{\text{opt}} \\ \gamma_{\text{opt}} & \delta_{\text{opt}}\end{pmatrix} = \begin{pmatrix}1{,}006\,943 & 0{,}038\,425 \\ 0{,}386\,874 & 0{,}010\,439\end{pmatrix}, \\ p_{0,\text{opt}} = 17{,}622\,499, \quad q_{0,\text{opt}} = 14{,}600\,384\end{gathered} \tag{7.22}$$

ist in Abb. 7.13b dargestellt (gestrichelte Linien). Es ist eine qualitative Übereinstimmung mit den Daten zu erkennen, wenngleich die absoluten Abweichungen zu groß sind. Das liegt wohl daran, dass das verwendete Modell nicht so gut geeignet ist, die Daten zu erklären.

In Abbildung 7.7b zeigt die rote Kurve den Verlauf des Fehlers bei der Optimierung mit dünnen Gittern. Der Vergleich zwischen Optimierung der Interpolierenden und gradientenfreier Optimierung der Zielfunktion hinkt analog zu den vorherigen Beispielen, aber leider bringt auch hier die Optimierung mittels B-Splines kaum etwas gegenüber der Optimierung der stückweise linearen Interpolierenden. Zusätzlich zur Instabilität des Modells spielt hier noch die höhere Dimensionalität mit hinein.

Mit $N = 10000$ bekommt man durch Optimierung der glatten Interpolierenden die Parameter

$$\begin{gathered}\begin{pmatrix}\alpha^*_{\text{opt}} & \beta^*_{\text{opt}} \\ \gamma^*_{\text{opt}} & \delta^*_{\text{opt}}\end{pmatrix} = \begin{pmatrix}0{,}696\,355 & 0{,}026\,869 \\ 0{,}564\,245 & 0{,}015\,078\end{pmatrix}, \\ p^*_{0,\text{opt}} = 20{,}220\,649, \quad q^*_{0,\text{opt}} = 12{,}356\,606,\end{gathered} \tag{7.23}$$

die sich deutlich von den optimalen Parametern unterscheiden. Entsprechend unterscheiden sich auch die durchgezogenen Lösungskurven in Abb. 7.13b von den optimalen Kurven. Allerdings stimmen auch diese Linien qualitativ mit den Daten einigermaßen überein (richtige Frequenz der Höhen und Tiefen).

8 Interpolation von Elastizitätstensoren in der Materialoptimierung

Zum Abschluss der Arbeit wollen wir uns mit der Materialoptimierung einer weiteren Anwendung von dünngitterbasierter Interpolation widmen. Beim Teilgebiet der Formoptimierung geht es, vereinfacht gesagt, um die Bestimmung der Form eines Bauteils oder eines anderen zu bauenden Objekts, sodass bestimmte physikalische Größen optimal sind. Ein Beispiel kann eine Brücke mit zwei Pfeilern und einer im Querschnitt parabelförmigen Durchfahrt sein, wenn die Parameter der Parabel so gewählt werden sollen, dass die Belastung der Brücke unter ihrer eigenen Gewichtskraft minimal sein soll. Meist sind Nebenbedingungen gegeben, sodass die entstehenden Optimierungsprobleme Definition 1.1 unseres allgemeinen Optimierungsproblems nicht erfüllen und wir somit die in dieser Arbeit entwickelte Methode zur Optimierung mit dünnen Gittern nicht anwenden können. Allerdings kann die in Abschnitt 4.2 beschriebene dünngitterbasierte Interpolation dabei helfen, andere Optimierungsverfahren wesentlich zu beschleunigen.

In diesem Kapitel richten wir uns bei Problembeschreibung und Notation nach der Arbeit [27] von Hübner. Wir werden die von ihm bei der Formoptimierung eingesetzten Interpolationsverfahren durch die dünngitterbasierte B-Spline-Interpolation ersetzen und ihre Eignung anhand eines Vergleichs der Resultate überprüfen.

8.1 Formoptimierung mit Zwei-Skalen-Ansatz

8.1.1 Homogenisierung als Lösungsansatz

Das Problem der Formoptimierung ist, so wie es oben dargestellt wurde, sehr allgemein gehalten. Im Allgemeinen ist nämlich die Topologie der optimalen Form unbekannt, also nicht nur die Position des Randes, sondern zum Beispiel auch die Anzahl der „Löcher“ (topologisches Geschlecht). Dies macht die allgemeine Formoptimierung zu einem sehr komplexen und schwer lösbaren Problem.

Die klassischen Lösungsansätze zur Formoptimierung schränken daher nach [27] die erlaubten Topologien stark ein, sodass die im Verlauf der Optimierung betrachteten Formen alle zueinander homöomorph sind. Allerdings ist ohne Vorabwissen, welche Topologie die optimale Form besitzt, die so erhaltene Lösung im Allgemeinen deutlich schlechter als die Optimallösung. Zusätzlich erfordert selbst diese starke Vereinfachung anspruchsvolle Lösungsalgorithmen, die die bei der Finite-Elemente-Approximation verwendeten Diskretisierungen des Gebiets adaptiv verändern können (Boundary-Movement-Verfahren).

Einen vollkommen anderen Ansatz stellt die sogenannte Homogenisierung dar. Hier ist das komplette, der Form zur Verfügung stehende Gebiet potentiell mit Material gefüllt. Nur die Dichte des Materials ist von Punkt zu Punkt verschieden. Ist sie in einem Punkt

also 0 %, so befindet sich dort kein Material, ist sie dagegen 100 %, so befindet sich in diesem Punkt viel Material. Das Besondere ist, dass auch Dichten dazwischen erlaubt sind, wodurch Optimierungsalgorithmen wesentlich mehr Freiheit gegeben ist. Auch wenn eine Dichte von zum Beispiel 50 % eventuell wenig Rückschlüsse auf die zu bauende Form zulässt (ein Bauteil besteht an einem Punkt meistens entweder aus Material oder aus Luft), so lässt sich aus dem Gesamtbild des Resultats oft die Topologie der optimalen Form ableiten. Nach [27] kann man die Topologie des Ergebnisses auch als Startpunkt für andere Verfahren der Formoptimierung verwenden.

Wir wollen nun die optimale Dichteverteilung bestimmen – optimal in dem Sinne, dass das sogenannte Compliance-Funktional

$$J := \int_\Omega \vec{F} \cdot \vec{u}(\vec{x}) \, \mathrm{d}\vec{x} \tag{8.1}$$

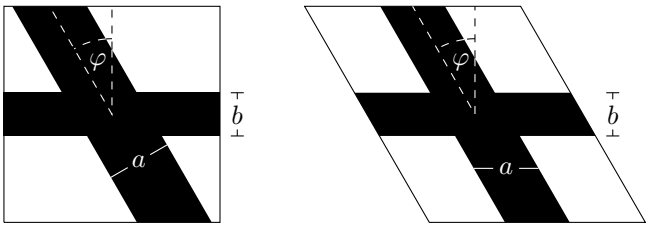
minimal wird, also die „Nachgiebigkeit des Körpers Ω unter Einwirkung einer Kraft $\vec{F}$“ (siehe [27]) im Rahmen der linearen Elastizität. Dabei ist $\vec{u}(\vec{x})$ die Verschiebung des Körpers im Punkt $\vec{x} \in \Omega$. Sie hängt von der Materialverteilung ab und kann mittels Finiter Elemente bestimmt werden (Herleitung siehe [25] und [27]). Im Folgenden arbeiten wir in $d = 2$ Dimensionen – man kann aber alle Ansätze auf den 3D-Fall erweitern.

8.1.2 Mikrozellen im Zwei-Skalen-Ansatz

Damit das durch die Homogenisierung entstehende Problem durch Computer lösbar wird, ist eine Diskretisierung des Materials hilfreich. Der sogenannte Zwei-Skalen-Ansatz stellt das Material laut [27] als „Verbandsstruktur“ dar, die aus „periodischen Mikrozellen“ besteht. Für den prinzipiellen Aufbau einer Mikrozelle gibt es wieder verschiedene Möglichkeiten. Wie in [27] betrachten wir zunächst quadratische Mikrozellen mit Seitenlänge 1, in denen sich zwei Balken, deren Mittellinien sich im Mittelpunkt der Mikrozelle schneiden, aus Material befinden (siehe Abb. 8.1a). Die beiden Balken können unterschiedliche Dicken $a, b \in [0, 1]$ besitzen. Der Balken mit der Dicke b liegt stets horizontal, während der andere mit der Dicke a um den Winkel $\varphi \in (-\pi/2, \pi/2)$ zur Senkrechten gedreht ist. Somit ist jede Mikrozelle durch drei Parameter $x_1, x_2 \in [0, 1]$, $x_3 \in (0, 1)$ definiert. Man erhält die Dicken und den Winkel aus den Parametern durch

$$a = x_2, \qquad b = x_1, \qquad \varphi = \pi \left(x_3 - \frac{1}{2} \right). \tag{8.2}$$

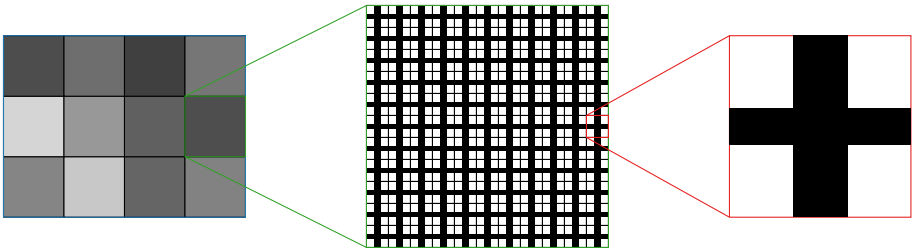
Weil das Material der Mikrozellen allerdings periodisch sein soll, müsste man für größere Drehungen ($a > 0$, $|\varphi| \geqslant \pi/4$ ist hinreichend) zusätzliche Balken platzieren, sodass die Mikrozellen nahtlos ineinander übergehen, wenn man sie nebeneinander anordnet. Daher ersetzen wir die Drehungen durch Scherungen in horizontaler Richtung (siehe Abb. 8.1b). Die Mikrozellen können jetzt also allgemein horizontale Parallelogramme sein, die aus dem Einheitsquadrat durch Scherung mit dem Scherwinkel $\varphi \in (-\pi/2, \pi/2)$ hervorgehen. a und b geben jetzt die horizontale bzw. vertikale Breite des vertikalen bzw. horizontalen Balkens an. Wäre a die „orthogonale Breite“ des vertikalen Balkens (Höhe auf die lange Seite), so würden Unstetigkeiten in den sich ergebenden Elastizitätstensoren entstehen (siehe [27]).



(a) erste Version mit gedrehtem Balken

(b) finale Version mit Scherung

Abbildung 8.1: Aufbau der Mikrozellen mit Parametern $a = 0{,}3$, $b = 0{,}2$, $\varphi = \pi/6$



(a) gesamtes Gebiet mit Dichteverteilung

(b) Makrozelle (periodische Mikrozellen)

(c) einzelne Mikrozelle

Abbildung 8.2: Zwei-Skalen-Ansatz in der Homogenisierung, vereinfacht dargestellt (Parameter $a = 0{,}3$, $b = 0{,}2$, $\varphi = 0$)

8.1.3 Kopplung von Mikro- und Makroproblem

Das komplette Gebiet, in dem Material verteilt werden kann, besteht im Zwei-Skalen-Ansatz aus einer $(N_1 \times N_2)$-Matrix von Makrozellen (siehe Abb. 8.2a). Jede Makrozelle setzt sich aus periodisch angeordneten, identischen und unendlich kleinen Mikrozellen zusammen (siehe Abb. 8.2b und 8.2c) – unendlich klein im Sinne eines Grenzwerts gegen 0. Sind nun die drei Mikrozellen-Parameter einer solchen Makrozelle bekannt, so kann man mithilfe Finiter Elemente (FE) den sogenannten Elastizitätstensor dieser Makrozelle ermitteln („Mikroproblem“). Beim Elastizitätstensor handelt es sich um einen symmetrischen Tensor in $\mathbb{R}^{2\times2\times2\times2}$, sodass dieser durch eine symmetrische (3×3)-Matrix

$$E = \begin{pmatrix} E_{1,1} & E_{1,2} & E_{1,3} \\ E_{1,2} & E_{2,2} & E_{2,3} \\ E_{1,3} & E_{2,3} & E_{3,3} \end{pmatrix} \tag{8.3}$$

oder einen $\mathbb{R}^6$-Vektor darstellbar ist. Wenn man die Elastizitätstensoren aller Makrozellen kennt, dann kann man wiederum mit Finiten Elementen die oben erwähnte Verschiebung $\vec{u}(\vec{x})$ des Körpers berechnen („Makroproblem“) und damit das Compliance-Funktional (8.1).

Das Ziel ist, die Mikrozellen-Parameter (drei pro Makrozelle) so zu wählen, dass das Compliance-Funktional minimal wird. Man erhält also ein Optimierungsproblem mit $3N_1N_2$ Variablen. Als Nebenbedingung wird gefordert, dass der volumenmäßige Anteil des

Materials am Gesamtgebiet einen bestimmten Wert nicht überschreitet. Bei der Lösung dieses Optimierungsproblems mit Nebenbedingung muss ein Algorithmus immer wieder das Compliance-Funktional in Abhängigkeit der $3N_1N_2$ Variablen auswerten, also ein Makroproblem lösen. Jede solche Auswertung erfordert allerdings die Lösung von N_1N_2 Mikroproblemen (eins je Makrozelle), was recht zeitaufwendig ist, weil jedes Mal ein FE-Problem zu lösen ist. Nach [27] sagt man daher auch, Makro- und Mikroproblem seien gekoppelt. Erschwerend hinzu kommt noch, dass die verwendeten Optimierungsalgorithmen meist gradientenbasiert arbeiten und deswegen nicht nur den Elastizitätstensor, sondern auch seine partiellen Ableitungen erfordern, was den Rechenaufwand weiter erhöht.

8.2 Bestimmung der Elastizitätstensoren

Zur Lösung dieser Problematik berechnet [27] in einer sogenannten „Offline-Phase“ vor der eigentlichen Formoptimierung Lösungen des Mikroproblems für verschiedene Parameterkombinationen. Während der Optimierung in der „Online-Phase“ wird dann ein Interpolationsverfahren verwendet, um die Elastizitätstensoren für allgemeine Parameter bestimmen zu können. Das entspricht einer Entkopplung von Mikro- und Makroproblem. Die in [27] eingesetzten Verfahren zur Interpolation umfassen die sogenannte $\mathcal{C}^1$-Interpolation und die Dünngitterinterpolation mit unmodifizierten, stückweise linearen Basisfunktionen. Wir werden in diesem Abschnitt zusätzlich weitere, auf B-Splines basierende Interpolationsarten betrachten.

8.2.1 Materialkataloge

Die zu interpolierenden Daten waren vorgegeben (identisch mit denen von [27]) und wurden in verschiedenen sogenannten „Materialkatalogen“ bereitgestellt. Dabei handelt es sich zum einen um einen Katalog auf dem vollen 3D-Gitter

$$\left\{\frac{1}{64},\dots,\frac{64}{64}\right\}\times\left\{\frac{1}{64},\dots,\frac{64}{64}\right\}\times\left\{\frac{1}{64},\dots,\frac{63}{64}\right\} \tag{8.4}$$

(x_3 darf nicht eins werden, siehe Abschnitt 8.1.2) und zum anderen um Kataloge auf den regulären dünnen 3D-Gittern mit Level 3, 4, 5, 6, 7. Die Genauigkeit der Dünngitter-Daten hängt vom Level ab, sodass beispielsweise der Level-7-Katalog nicht nur mehr, sondern auch genauere Werte als der Level-6-Katalog enthält (siehe auch Abschnitt 8.3.3). Wir weisen darauf hin, dass sich gegen Ende der Arbeit die Berechnung der Materialkataloge aufgrund eines Programmfehlers signifikant geändert hat. Bei zukünftigen Arbeiten werden sich die genauen Werte daher ändern – die hier vorgestellten Methoden werden aber übertragbar sein. Auf den Vergleich der im Folgenden vorgestellten Methoden hat dies keinen Einfluss.

8.2.2 $\mathcal{C}^1$-Interpolation

Die „$\mathcal{C}^1$-Interpolation“ stammt aus [31] und entspricht einer stückweise kubischen Interpolation auf einem regelmäßigen dreidimensionalen Gitter. Sei $f\colon [0,1]^3 \to \mathbb{R}$ die zu interpolierende Funktion (bei uns ist f ein Eintrag des Elastizitätstensors (8.3) in Abhängigkeit

der drei Mikrozellen-Parameter). Wir unterteilen den Definitionsbereich $[0,1]^3$ regelmäßig in kleine, gleich große Würfel und setzen für die Interpolierende $\widetilde{f}$ ein abschnittsweises Polynom vom Koordinatengrad 3 an, also

$$\widetilde{f}(\vec{x}) = \sum_{i_1=0}^{3} \sum_{i_2=0}^{3} \sum_{i_3=0}^{3} a_{i_1,i_2,i_3} x_1^{i_1} x_2^{i_2} x_3^{i_3} \tag{8.5}$$

mit je Teilwürfel im Allgemeinen unterschiedlichen Koeffizienten a_{i_1,i_2,i_3}. Die 64 Koeffizienten pro Teilwürfel sind eindeutig festgelegt, wenn $\widetilde{f}$ und die partiellen Ableitungen folgende acht Funktionen an allen acht Teilwürfel-Eckpunkten interpolieren:

$$f, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial^2 f}{\partial x_1 \partial x_2}, \frac{\partial^2 f}{\partial x_1 \partial x_3}, \frac{\partial^2 f}{\partial x_2 \partial x_3}, \frac{\partial^3 f}{\partial x_1 \partial x_2 \partial x_3}, \tag{8.6}$$

also f selbst, die ersten partiellen Ableitungen, die zweiten gemischten partiellen Ableitungen und eine dritte gemischte partielle Ableitung. Weil f und die ersten Ableitungen darunter sind, ist die Interpolierende stetig differenzierbar (daher „$\mathcal{C}^1$-Interpolation“). Laut [27] erhöht die Wahl der übrigen Ableitungen in (8.6) die Glattheit der Interpolierenden. Außerdem ist das entstehende lineare Gleichungssystem (LGS) nach [31] regulär. Würde man die doppelten partiellen Ableitungen statt den gemischten verwenden, so wäre das LGS singulär (siehe [31]). Beachtet werden muss, dass die exakten partiellen Ableitungen der Einträge des Elastizitätstensors nicht bekannt sind. Daher muss man die Ableitungen numerisch zum Beispiel mithilfe zentraler Differenzenquotienten approximieren.

Ein großer Nachteil der $\mathcal{C}^1$-Interpolation ist, dass die zu interpolierenden Daten auf einem regelmäßigen und damit vollen 3D-Gitter vorliegen müssen. Damit dauert die Berechnung eines Materialkatalogs relativ lange. Außerdem führt die große Anzahl an Koeffizienten zu einem größeren Speicherverbrauch: Zum Beispiel enthält das obige Gitter (8.4) $63 \cdot 63 \cdot 62 = 246\,078$ Teilwürfel, daher sind wegen der 64 Koeffizienten pro Teilwürfel knapp 16 Millionen Werte für jeden der sechs Einträge des Elastizitätstensors notwendig.

8.2.3 Dünngitter-Interpolation

Eine Möglichkeit, diesen Problemen aus dem Weg zu gehen, ist die Verwendung von dünngitterbasierter Interpolation. In [27] wurde die Interpolation mit unmodifizierten, stückweise linearen Basisfunktionen betrachtet (siehe auch Abschnitt 4.2.1). Weil die verwendeten Optimierungsverfahren gradientenbasiert arbeiten, muss der Gradient der Interpolierenden angenähert werden (z. B. mit zentralen Differenzenquotienten wie eben). Allerdings sind die Verfahren nicht darauf ausgelegt, stückweise lineare Interpolierende zu optimieren, weil die Gradienten unstetig sind. Außerdem verschwinden die Interpolierenden auf dem Rand von $[0,1]^3$, weswegen der Optimierungsbereich in [27] künstlich eingeschränkt wurde, was aber die Qualität der Resultate merklich verringert (siehe auch Abschnitt 8.4).

Es liegt nahe, stattdessen die in Abschnitt 4.2.2 diskutierte Dünngitter-B-Spline-Interpolation zu verwenden. Vorteile sind neben der höheren Glattheit und der einfachen Berechnung des exakten Gradienten auch die größere Genauigkeit der Interpolierenden. Dadurch sollte die Optimierung nicht nur schneller konvergieren, sondern auch genauer.

8.2.4 Vollgitter-Interpolation mit B-Splines

Um neben der $\mathcal{C}^1$-Interpolation einen weiteren Anhaltspunkt zu haben, wie gut sich die dünngitterbasierte B-Spline-Interpolation schlägt (der optimale Zielfunktionswert ist ja unbekannt), wurde eine weitere Vollgitter-Interpolationsmethode implementiert. Dazu wird zunächst eine Linearkombination aller zweidimensionalen B-Splines vom Grad p mit Level $n := 6$ betrachtet, also

$$\widetilde{f}(\vec{x}) := \sum_{i_1=1}^{2^n} \sum_{i_2=1}^{2^n} \sum_{i_3=1}^{2^n-1} \alpha_{n\vec{e},\vec{i}} b^p_{n\vec{e},\vec{i}}(\vec{x}), \quad \vec{i} = (i_1, i_2, i_3), \tag{8.7}$$

sodass $\widetilde{f}$ die Zielfunktion f an den Punkten des vollen Gitters (8.4) interpoliert. Das Problem ist, dass diese B-Splines nach Satz 3.4 nur eine Basis des Spline-Raums auf

$$h_n \cdot ([m, 2^n - m + 1] \times [m, 2^n - m + 1] \times [m, 2^n - m]), \qquad m := \frac{p+1}{2}, \tag{8.8}$$

darstellen (punktweise Skalierung des Kreuzprodukts). Außerhalb dieses Bereichs in $[0,1]^3$ fehlen relevante B-Splines, weswegen die Linearkombination $\widetilde{f}$ zum Rand hin unnatürlich abfällt. Um auf dem kompletten Gebiet $[0,1]^3$ eine einigermaßen sinnvolle Interpolierende zu erhalten, wurden per simpler koordinatenweiser linearer Extrapolation zusätzliche Daten generiert, sodass die fehlenden B-Splines dazukommen. Genauer gesagt wurden jeweils $m+1$ Datenpunkte am Anfang und am Ende jeder Koordinatenrichtung erzeugt, womit man die erweiterte Interpolierende

$$\widetilde{f}(\vec{x}) := \sum_{i_1=-m}^{2^n+m+1} \sum_{i_2=-m}^{2^n+m+1} \sum_{i_3=-m}^{2^n+m} \alpha_{n\vec{e},\vec{i}} b^p_{n\vec{e},\vec{i}}(\vec{x}) \tag{8.9}$$

erhält. Die entsprechenden B-Splines sind eine Basis des Spline-Raums auf

$$h_n \cdot ([-1, 2^n + 2] \times [-1, 2^n + 2] \times [-1, 2^n + 1]), \tag{8.10}$$

was den Einheitswürfel nun mit einschließt. Für $n = 6$ und $p = 3$ lautet das erweiterte Gitter beispielsweise

$$\left\{-\frac{2}{64}, \ldots, \frac{67}{64}\right\} \times \left\{-\frac{2}{64}, \ldots, \frac{67}{64}\right\} \times \left\{-\frac{2}{64}, \ldots, \frac{66}{64}\right\}. \tag{8.11}$$

8.3 Implementierung

8.3.1 CFS++

Die Implementierung erfolgt mithilfe der seit 2000 an den Universitäten Erlangen-Nürnberg und Klagenfurt entwickelten Finite-Elemente-Software CFS++ (*Coupled Field Simulation in C++*, [28]). CFS++ ist speziell auf die Anwendung bei gekoppelten Feldproblemen ausgerichtet, die nach [28] partielle Differentialgleichungen aus Elektromagnetik, Mechanik, Akustik und Fluiddynamik umfassen können. Allerdings enthält CFS++ auch Routinen zur

Abbildung 8.3: Logo von CFS++, Quelle: [53]

Formoptimierung per Homogenisierung mit Zwei-Skalen-Ansatz. So können sowohl Mikro- als auch Makroproblem (siehe Abschnitt 8.1.3) durch CFS++ gelöst werden. Lösungen des Mikroproblems werden bei Verwendung von Interpolation nicht benötigt, allerdings kam CFS++ bei der Erzeugung der Materialkataloge in der Offline-Phase zum Einsatz. Die verwendete CFS++-Revisionsnummer ist 13325.

Die oben erklärten Interpolationsarten sind im Einzelnen wie folgt implementiert: Die Vollgitter-Interpolationen, also die $\mathcal{C}^1$- und die B-Spline-Vollgitter-Interpolation, befinden sich direkt in CFS++, wobei die Koeffizienten jeweils im Voraus extern berechnet wurden. Die dünngitterbasierten Interpolationen rufen dagegen SG^{++}-Funktionen auf (aus `sg::base` und `sg::opt`, siehe Kapitel 6). Der Bequemlichkeit halber wurde die Hierarchisierung bei der B-Spline-Interpolation im Voraus extern durchgeführt, aber natürlich kann auch SG^{++} diese Aufgabe übernehmen. Weil die Tensoreinträge $E_{1,1}$, $E_{1,2}$, $E_{2,2}$ und $E_{3,3}$ nicht negativ werden dürfen, wurden die entsprechenden Interpolierenden bei Dünn- und Vollgitter-Interpolation durch ihre positiven Anteile ersetzt (also $\max\{\cdot, 0\}$).

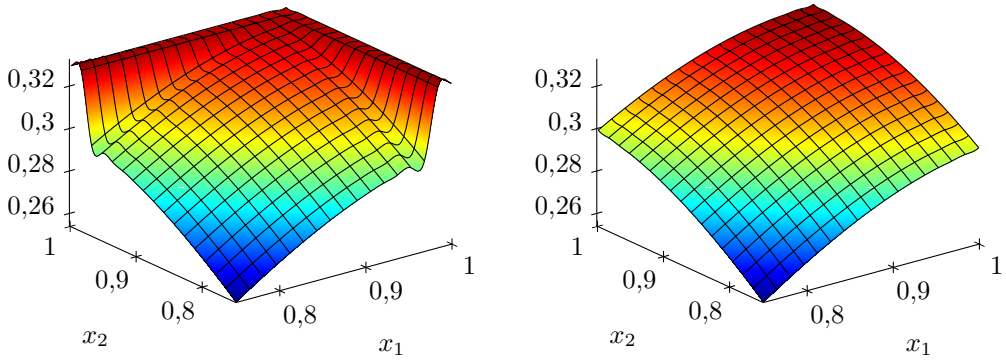
8.3.2 Optimierung

CFS++ bietet Schnittstellen zu verschiedenen Optimierungsbibliotheken. Wir verwenden hier wie in [27] den kommerziellen Optimierer SNOPT. Die Optimierung verläuft dabei wie folgt: Im Verlauf der Optimierung minimiert SNOPT das Compliance-Funktional $J = J(\vec{y})$ aus (8.1) in Abhängigkeit der $3N_1N_2$ Mikrozellen-Parameter

$$\vec{y} = (a^{(1)}, b^{(1)}, \varphi^{(1)}, \ldots, a^{(N_1N_2)}, b^{(N_1N_2)}, \varphi^{(N_1N_2)}) \tag{8.12}$$

unter Einhaltung einer Nebenbedingung (Materialanteil). Um $J(\vec{y})$ für ein gegebenes $\vec{y}$ zu berechnen, muss der Elastizitätstensor $E^{(i)}$ jeder Makrozelle $i = 1, \ldots, N_1N_2$ in Abhängigkeit von $a^{(i)}, b^{(i)}, \varphi^{(i)}$ bestimmt werden. Dazu müsste man eigentlich N_1N_2 Mikroprobleme lösen. Wir verwenden stattdessen unter anderem dünngitterbasierte Interpolation, was zu einem deutlich geringeren Zeitaufwand führt. Im Gegensatz zu den vorherigen Kapiteln wird also die zu optimierende Zielfunktion nicht interpoliert – die Interpolierenden werden „indirekt“ bei der Auswertung der Zielfunktion verwendet. SNOPT gibt dann den Wert des Compliance-Funktionals für die Mikrozellen-Parameter $\vec{y}$, die SNOPT als optimal erachtet, zurück, wobei die dabei benötigten Elastizitätstensoren nicht exakt berechnet, sondern interpoliert wurden.

Pro „Iteration“ berechnet SNOPT $J(\vec{y})$ teilweise mehrfach mit verschiedenen $\vec{y}$, in den letzten Iterationen (vor dem erfolgreichen Abbruch) bis zu zehn Mal. Wir wollen hier nicht weiter auf die genauere Funktionsweise von SNOPT eingehen, weil eine Behandlung der Interna von SNOPT den Rahmen dieses Kapitels sprengen würde.



(a) Original-Daten mit Unstetigkeiten und daraus resultierenden Oszillationen

(b) modifizierte Daten ohne durch Extrapolation entfernte Unstetigkeiten

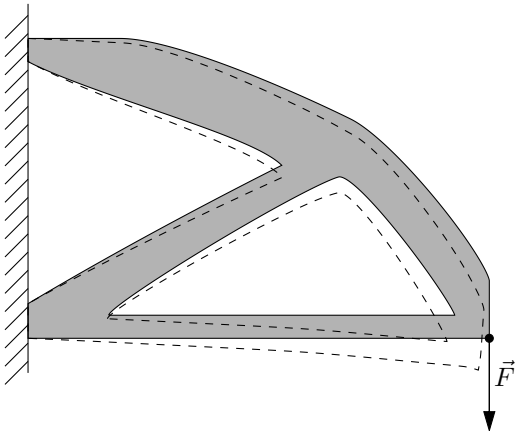
Abbildung 8.4: Ausschnitt der Vollgitter-B-Spline-Interpolation (Flächen, Gitter) der Vollgitter-Daten (Gitter-Schnittpunkte) von $E_{1,2}$ ($x_1, x_2 \in [0{,}75; 1]$, $x_3 = 0{,}5$)

8.3.3 Unstetigkeiten in den Daten

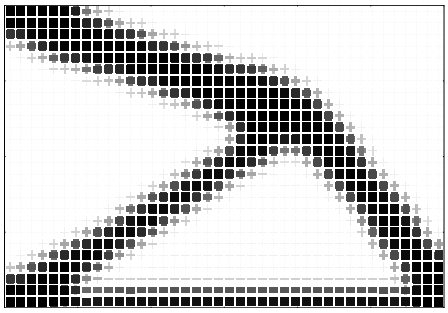
Bevor wir die oben beschriebenen Interpolationsarten auf die Bestimmung der Elastizitätstensoren anwenden können, muss noch eine subtile Problematik geklärt werden. Wie in [27] beschrieben, enthalten die Einträge der Elastizitätstensoren eine Unstetigkeit bei den Mikrozellen-Kreuzdicken $a = 1$ oder $b = 1$. Dies liegt nach [27] darin begründet, dass zum Beispiel für $b = 1$ die Mikrozellen und damit auch die zugehörige Makrozelle als periodischer Verbund vollständig mit Material ausgefüllt sind, unabhängig vom Wert von a (oder φ). Damit ist der Elastizitätstensor auf der Geraden $b = 1$ konstant, allerdings nicht in einer Umgebung der Geraden, wodurch man Unstetigkeiten erhält.

Das Problem wird dadurch noch größer, dass die Unstetigkeiten bedingt durch die Berechnung der Elastizitätstensoren als Lösungen eines diskretisierten Finite-Elemente-Problems (Mikroproblem) schon auf den Geraden $a = 1 - h'$ und $b = 1 - h'$ für ein $h' > 0$ auftreten (siehe [27]). Die Mikrozellen wurden nämlich bei der Berechnung der Materialkataloge mit der Schrittweite $h' = h_n := 2^{-n}$ diskretisiert, die zum jeweiligen Katalog-Level n korrespondiert. Somit erklärt sich auch die bereits erwähnte Tatsache, dass Kataloge höherer Levels nicht nur mehr, sondern auch genauere Werte als die niedrigerer Levels enthalten. Als zusätzlicher Nebeneffekt tritt der obige Vollmaterial-Fall schon bei $a = 1 - h'$ oder $b = 1 - h'$ ein (siehe Abb. 8.4a). Damit gehören in einem Katalog mit Level n (egal, ob volles oder dünnes Gitter) alle Werte mit Parametern $a \in \{1 - h_n, 1\}$ oder $b \in \{1 - h_n, 1\}$ zum „konstanten Plateau“, das die Unstetigkeit verursacht.

Es ist leicht verständlich, dass diese Unstetigkeiten erheblich bei der Interpolation stören. In [27] wurde daher der Optimierungsbereich stärker eingeschränkt (sodass $a, b \leqslant 0{,}9$), was die Qualität der Ergebnisse entsprechend schmälert, wie wir später sehen werden. Wir wollen einen anderen Weg gehen und setzen Extrapolation ein, um die Unstetigkeiten direkt aus den Daten zu eliminieren. Im Detail haben wir die letzten zwei Datenzeilen und -spalten, die $a \in \{1 - h_n, 1\}$ oder $b \in \{1 - h_n, 1\}$ für $n = 6$ entsprechen, aus den ursprünglichen Vollgitter-Daten für das Gitter (8.4) entfernt, sodass nur die Daten für das



(a) Schema eines Kragträgers zusammen mit äußerer Kraft und tatsächlicher Verschiebung



(b) Beispiel-Resultat der Formoptimierung (ohne Scherwinkeloptimierung)

Abbildung 8.5: Kragträger (Form links basiert auf Optimierungsergebnis rechts, gestrichelte Verschiebung links wurde dabei durch CFS++ ermittelt)

eingeschränkte Gitter

$$\left\{\frac{1}{64},\ldots,\frac{62}{64}\right\}\times\left\{\frac{1}{64},\ldots,\frac{62}{64}\right\}\times\left\{\frac{1}{64},\ldots,\frac{63}{64}\right\} \tag{8.13}$$

übrig blieben. Anschließend wurde die in Abschnitt 8.2.4 beschriebene Erweiterung des Gitters durch Extrapolation durchgeführt, nur mit der Maßgabe, dass am Ende der ersten zwei Koordinatenrichtungen nicht nur $m+1$, sondern $m+3$ zusätzliche Werte extrapoliert werden. So erhält man für $p=3$ auf dem Gitter (8.11) Werte, die die Unstetigkeiten nicht mehr enthalten. Auch wenn die verwendete Extrapolationsart relativ simpel ist, so scheint die neue Interpolierende nahe dem Rand von $[0,1]^3$ eine recht gute Qualität zu besitzen (siehe Abb. 8.4b), sodass jetzt keine größere Bereichseinschränkung mehr nötig ist.

8.4 Ergebnisse

Wir verwenden als Testbeispiel das „Beispiel A“ aus [27]. Dabei handelt es sich um einen Kragträger in zwei Dimensionen, der an der linken Seite befestigt ist und an dessen rechtem Ende eine Kraft nach unten wirkt (Schema siehe Abb. 8.5a). Wie in [27] wird das rechteckige Gebiet in $40\cdot 26$ Makrozellen diskretisiert. Das sich ergebende Optimierungsproblem mit im Allgemeinen $3\cdot 40\cdot 26 = 3120$ Variablen wird durch CFS++ gelöst und das Ergebnis in Plots wie Abb. 8.5b ausgegeben. Jedes Kästchen, das für eine Makrozelle steht, enthält ein Balkenkreuz, dessen Parameter die Parameter der zugehörigen periodischen Mikrozellen-Struktur angeben. Zur Vereinfachung sind die Kästchen rechteckig und ungeschert – der vertikale Balken wird stattdessen um den Winkel φ gedreht (wie in Abb. 8.1a).

Wir behandeln zunächst wie in [27] den Fall der ausgeschalteten Scherwinkeloptimierung, bei dem $\varphi = 0$ fixiert ist und nur in zwei Dimensionen interpoliert wird. Wie wir sehen werden, erhöht sich mit Scherwinkeloptimierung die Komplexität des Problems

spürbar. Zur Vergleichbarkeit mit [27] betrachten wir ebenfalls die zwei unterschiedlichen sogenannten Materialanteile 35 % und 50 % (maximal gefüllter Volumenanteil des Gebiets). Außerdem wird bei den Dünngitter-Interpolationen standardmäßig ein reguläres Gitter mit Level 7 verwendet. Die genauen Resultate der Formoptimierungen zusammen mit Zeit- und Speicherbedarf stehen in Anhang B.2.

Leider lassen sich unsere Ergebnisse nicht ohne Weiteres mit denen von [27] vergleichen, da in der Zwischenzeit ein Fehler im Code von CFS++ gefunden wurde, der die Zielfunktionswerte drastisch kleiner werden lässt. Wir rechnen daher zum Vergleich die Interpolationsarten aus [27] auch mit den Original-Daten für das Optimierungsgebiet $[0{,}02; 0{,}9]^2 \times [0{,}15; 0{,}85]$ von [27] (bzw. $[0{,}02; 0{,}9]^2$ ohne Scherwinkeloptimierung). Die übrigen vorgestellten Methoden (wie auch die in [27] behandelten Methoden) kommen mit den um die Unstetigkeiten bereinigten Daten auf dem erweiterten Gebiet $[0{,}02; 0{,}98]^2 \times [0{,}15; 0{,}85]$ (ohne Scherwinkeloptimierung $[0{,}02; 0{,}98]^2$) zum Einsatz.

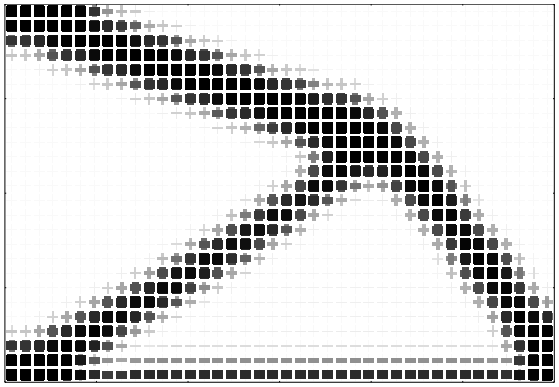
Sehr interessant wäre zum Vergleich noch das Ergebnis, das man erhalten würde, wenn man die Elastizitätstensoren bei der Optimierung nicht interpoliert, sondern direkt als Lösung der Mikroprobleme mittels Finiter Elemente berechnet. Leider ist das bei dem vorliegenden Beispiel impraktikabel, weil die Berechnung eines Elastizitätstensors mehrere Sekunden benötigt. Weil in jeder Iteration des Optimierers das Compliance-Funktional mindestens ein Mal ausgewertet wird, bräuchte man wohl ca. eine Stunde für eine Iteration, was bei den notwendigen Iterationszahlen zu wochen- oder monatelangen Berechnungen führen würde (und das nur für eine einzelne Gebiet-Materialanteil-Kombination).

8.4.1 Ohne Scherwinkeloptimierung

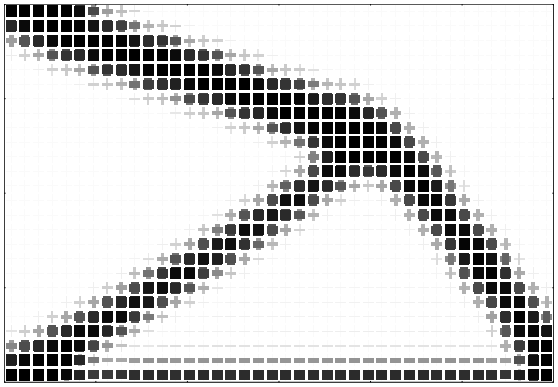
Setzt man $\varphi = 0$ fest und optimiert nur die Balkendicken, so erhält man die in Abb. 8.6 dargestellten Materialverteilungen. Auf den ersten Blick unterscheiden sich die Verteilungen der stückweise linearen Interpolation auf dem Original-Gebiet (linke Bildspalte) kaum von denen der B-Spline-Dünngitterinterpolation auf dem erweiterten Gebiet (rechte Spalte). Allerdings werden mit der B-Spline-Interpolation etwas bessere Zielfunktionswerte erreicht, bei 35 % Materialanteil z. B. 65,17 gegenüber 67,39 (siehe Anhang B.2.1). Das heißt, dass die durch B-Splines optimierte Form bei Belastung um ca. 3 % weniger nachgibt.

Allein die Elimination der Unstetigkeiten und die dadurch möglich gewordene Erweiterung des Optimierungsbereichs verbessern bereits den Wert des Compliance-Funktionals von 67,39 auf 66,28 (bei unmodifizierten stückweise linearen Basisfunktionen, 35 % Materialanteil). Eine weitere Verbesserung auf 65,29 wird durch die Verwendung modifizierter stückweise linearer Basisfunktionen erzielt. Allerdings braucht die dünngitterbasierte B-Spline-Interpolation nur gut halb so lange und liefert mit 65,17 ein besseres Ergebnis zurück. Das Ergebnis ist erstaunlicherweise sogar besser als die Resultate der Vollgitter-Interpolationen, auch wenn diese noch einmal etwas schneller konvergieren.

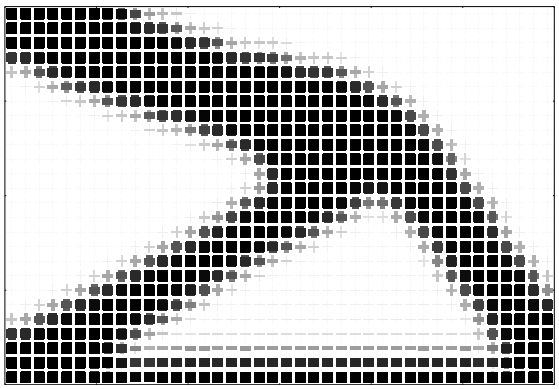
Analoge Aussagen gelten auch für 50 % Materialanteil, wobei hier die ermittelten Zielfunktionswerte deutlich kleiner sind, weil die Festigkeit der Struktur mit mehr „verbautem“ Material zunimmt. Dass die Optimierung mit unmodifizierten stückweise linearen Basisfunktionen auf dem erweiterten Gebiet wesentlich länger braucht, liegt daran, dass diese Basisfunktionen nahe dem Rand von $[0, 1]^2$ stark abfallen, was den Optimierer bei der Arbeit erheblich stört.



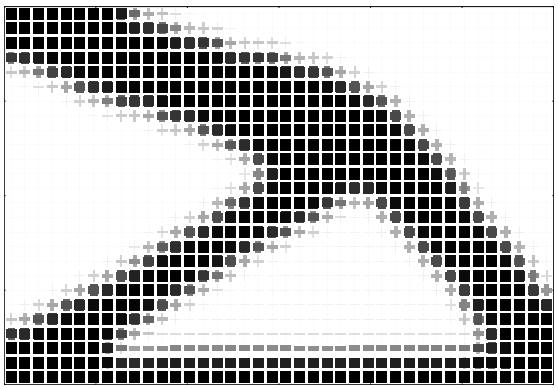
(a) Bereich $[0{,}02; 0{,}9]^2$, Materialanteil 35 %, unmodifizierte stückweise lineare Basis



(b) Bereich $[0{,}02; 0{,}98]^2$, Materialanteil 35 %, modifizierte B-Splines, $p = 3$



(c) Bereich $[0{,}02; 0{,}9]^2$, Materialanteil 50 %, unmodifizierte stückweise lineare Basis

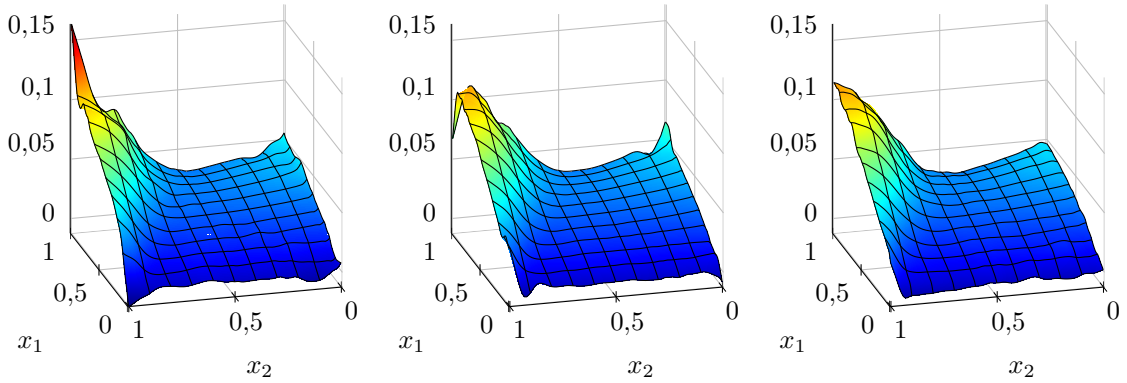


(d) Bereich $[0{,}02; 0{,}98]^2$, Materialanteil 50 %, modifizierte B-Splines, $p = 3$

Abbildung 8.6: Ergebnisse der Formoptimierung mithilfe dünngitterbasierter Interpolation, ohne Scherwinkeloptimierung

8.4.2 Mit Scherwinkeloptimierung

Wenn auch der Scherwinkel optimiert werden soll, dann erhöht sich die Komplexität des Problems im Vergleich zum Fall „$\varphi = 0$“ drastisch: Die $\mathcal{C}^1$-Interpolation braucht dann beispielsweise rund 15 Mal so lange und benötigt 18 Mal so viel Speicher (siehe Anhang B.2.2). Zunächst einmal liegt dies an der deutlich erhöhten Anzahl an benötigten Iterationen. Aber auch eine einzelne Iteration braucht vier bis fünf Mal länger als ohne Scherwinkeloptimierung. Dafür gibt es mehrere Gründe: Erstens müssen im Fall „ohne Scherwinkeloptimierung“ nur vier Tensoreinträge interpoliert werden (weil $E_{1,3} = E_{2,3} = 0$ für $\varphi = 0$) und jetzt alle sechs. Zweitens werden je Iteration alle Elastizitätstensoren und sämtliche partielle Ableitungen jeweils $(N_1 N_2)$-fach ausgewertet – das führt zu $4N_1N_2$ Auswertungen (Elastizitätstensor und drei partielle Ableitungen), während vorher nur $3N_1N_2$ Auswertungen nötig waren (nur zwei partielle Ableitungen). Drittens führt die Abhängigkeit der Basisfunktionen von nun drei Variablen dazu, dass 50 % mehr 1D-Auswertungen notwendig



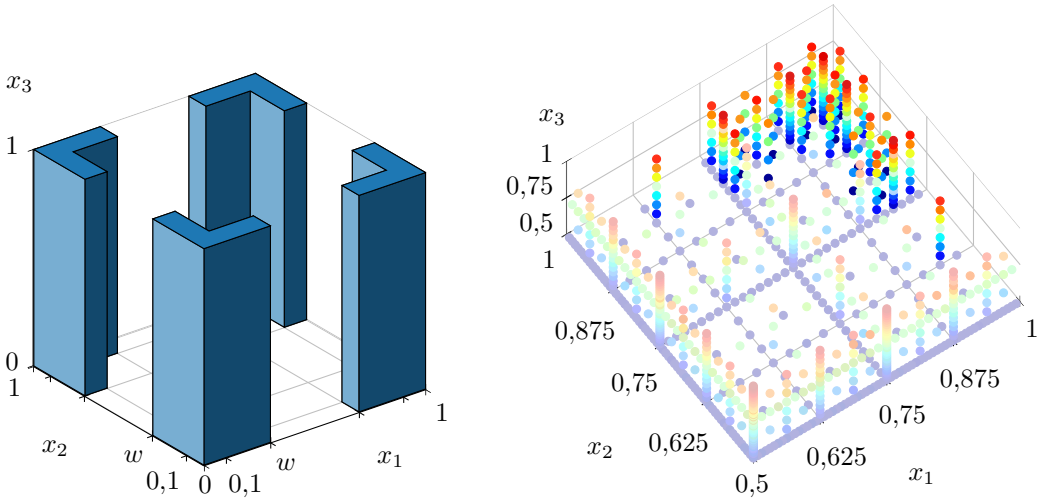
(a) reguläres Gitter mit Level 8 **(b)** reguläres Gitter mit Level 9 **(c)** verfeinertes Gitter

Abbildung 8.7: Oszillationen bei der dünngitterbasierten B-Spline-Interpolation von $E_{1,2}$ ($x_3 = 0{,}15$, modifizierte B-Splines vom Grad $p = 3$)

sind. Viertens hängt das gesamte Optimierungsproblem nun von $3N_1N_2$ statt wie vorher von $2N_1N_2$ Variablen ab, was den Optimierungsaufwand nochmals vergrößert.

Außerdem hat der Optimierer jetzt mehr Probleme mit den trivariaten Interpolierenden, sodass viele der Durchläufe auf regulären dünnen Gittern auch mit B-Splines ohne vernünftiges Ergebnis abbrechen. Ein x_1-x_2-Plot der zugehörigen Interpolierenden wie in Abb. 8.7 zeigt, dass die Tensoreinträge bei festem x_3 in den Ecken von $[0,1]^2$ teilweise größere Oszillationen besitzen, die auch bei Verwendung von Materialkatalogen höherer Levels (Erzeugung zusätzlicher Punkte mittels Vollgitter-Interpolation) nicht so schnell weggehen. Selbst für reguläre dünne Gitter mit Level 9 sind die Oszillationen zu groß (siehe Abb. 8.7b) und vorzeitige Abbrüche möglich. Für die Oszillationen an den Ecken gibt es hauptsächlich zwei Gründe: Zum einen besitzen reguläre dünne Gitter an den Ecken weniger Gitterpunkte, sodass hier die Genauigkeit der Interpolierenden nicht so groß ist. Zum anderen werden modifizierte B-Splines verwendet, um sinnvolle Randwerte zu erhalten. Diese führen aber wegen der zum Rand hin linearen Extrapolation leichter zu Oszillationen.

Zur Lösung des Problems wurde ausgehend vom regulären dünnen Gitter mit Level 7 per Vollgitter-Interpolation ein verfeinertes Gitter wie folgt konstruiert: Sei $A(w) \subset [0,1]^3$ für $w > 0$ die Menge der vier in Abb. 8.8a dargestellten „L-förmigen Türme“ mit Breite w und Dicke 0,1. Definiere nun $(w_1, \ldots, w_4) := (0{,}3; 0{,}3; 0{,}2; 0{,}1)$ und verfeinere (im Sinne von Definition 4.2) nacheinander für $k = 1, \ldots, 4$ alle Punkte $\vec{x}_{\vec{\ell},\vec{i}}$, die in $A(w_k)$ liegen und $\|\vec{\ell}\|_\infty \leqslant 4$ erfüllen. Die Verfeinerungsbereiche wurden so gewählt, dass einerseits nahe den Ecken von $(x_1, x_2) \in [0,1]^2$ neue Punkte erzeugt werden, andererseits aber nicht zu viele, um die Optimierungsdauer nicht unnötig zu verlängern. Durch die Bedingung „$\|\vec{\ell}\|_\infty \leqslant 4$“ werden nur Punkte erzeugt, deren Levels ℓ_t nicht größer als 5 sind, damit nicht zu viele neue Punkte erzeugt werden (das obige Verfeinerungskriterium wird ja zum Beispiel mit w_2 auch auf die mit w_1 erzeugten Punkte angewendet). Das auf diese Art erzeugte Gitter ist in Abb. 8.8b dargestellt und besitzt mit 4439 Punkten deutlich weniger Gitterpunkte als das reguläre Gitter mit nächsthöherem Level 8 (7423 Punkte) oder gar Level 9 (18943 Punkte). Gleichzeitig werden die unerwünschten Oszillationen an den Ecken deutlich geringer oder



(a) L-förmige Verfeinerungsbereiche

(b) Ausgangsgitter (hell) und neue Punkte, Farbe entspricht x_3-Koordinate (Oktant-Ausschnitt)

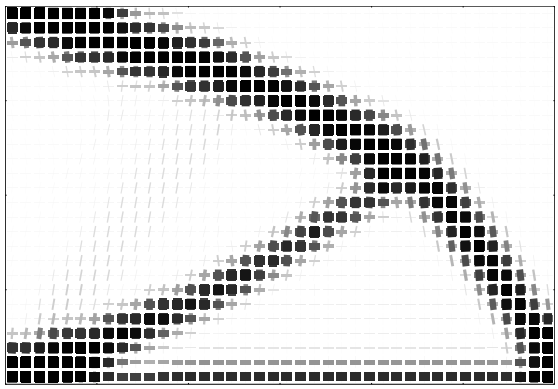
Abbildung 8.8: Verfeinerung des regulären 3D-Gitters mit Level 7 zur Erhöhung der Genauigkeit am Rand

verschwinden ganz (siehe Abb. 8.7c). Als Folge dessen bricht der Optimierer SNOPT bei diesem verfeinerten Gitter nicht mehr ab.

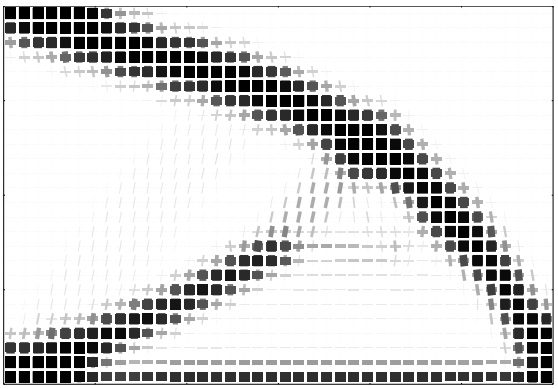
Betrachtet man die Ergebnisse der dünngitterbasierten Interpolation in Anhang B.2.2, so stellt man zunächst fest, dass sich mit Scherwinkeloptimierung der Wert des Compliance-Funktionals im Vergleich zu ohne um ca. 4 % senken lässt. Das ist nicht besonders viel, was aber am verwendeten Mikrozellen-Modell liegt. Ein mit mehr Parametern ausgestattetes Modell könnte womöglich größere Verbesserungen bringen.

Des Weiteren fällt bei Anblick der Ergebnisse auf, dass nur die Hälfte der Durchläufe auf den regulären Gittern mit Level 7, 8 und 9 ein vernünftiges Ergebnis hervorgebracht hat und die wenigen erfolgreichen Versuche meist sehr viel Zeit in Anspruch genommen haben. Verwendet man jedoch das obige verfeinerte Gitter mit modifizierten B-Splines, dann konvergiert der Optimierungsalgorithmus und liefert ein akzeptables Ergebnis in annehmbarer Zeit zurück (für beide Materialanteile). Die Anzahl der benötigten Iterationen ist sogar vergleichbar mit den Vollgitter-Interpolationsmethoden, auch wenn eine einzelne Iteration bei den dünnen Gittern wesentlich länger benötigt – bei den Vollgitter-Methoden ist die Anzahl der notwendigen Basisauswertungen unabhängig von der Gittergröße, während bei der Dünngitter-Interpolation alle Basisfunktionen ausgewertet werden. Obige Konstruktion des verfeinerten Gitters wurde übrigens intuitiv-visuell hergeleitet. Man könnte auch Gitter, die auf andere Weise konstruiert wurden, verwenden, solange sie genügend Punkte in den Ecken besitzen. Man ist also keineswegs auf obige Konstruktion beschränkt.

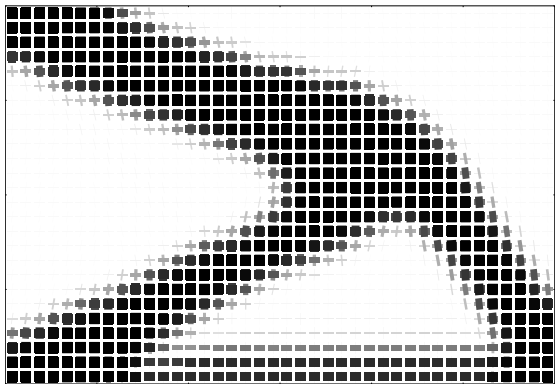
In Abb. 8.9 sieht man die sich ergebenden Materialverteilungen bei unterschiedlichen Optimierungsbereichen und Materialanteilen. Dass Abb. 8.9b etwas unorthodox aussieht, liegt wohl schlicht daran, dass 35 % Materialanteil zu gering sind, um eine vernünftige Lastverteilung zu ermöglichen. Man kann in Abb. 8.9a und 8.9b links der Querbalken in



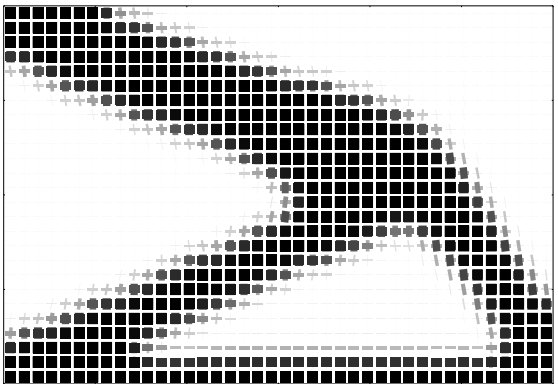
(a) Bereich $[0{,}02; 0{,}9]^2 \times [0{,}15; 0{,}85]$, Materialanteil 35 %, unmodifizierte stückweise lineare Basis, reguläres Gitter mit Level 7



(b) Bereich $[0{,}02; 0{,}98]^2 \times [0{,}15; 0{,}85]$, Materialanteil 35 %, modifizierte B-Splines, $p = 3$, verfeinertes Gitter



(c) Bereich $[0{,}02; 0{,}9]^2 \times [0{,}15; 0{,}85]$, Materialanteil 50 %, unmodifizierte stückweise lineare Basis, reguläres Gitter mit Level 7



(d) Bereich $[0{,}02; 0{,}98]^2 \times [0{,}15; 0{,}85]$, Materialanteil 50 %, modifizierte B-Splines, $p = 3$, verfeinertes Gitter

Abbildung 8.9: Ergebnisse der Formoptimierung mithilfe dünngitterbasierter Interpolation, mit Scherwinkeloptimierung

hellem Grau erkennen, dass der Optimierer gerne noch einen zweiten Balken „einbauen“ würde, um die Stabilität zu erhöhen. Die These wird auch dadurch gestützt, dass deutlich weniger der Optimierungsdurchläufe in Tabelle B.16 ein Ergebnis zustande bringen als bei 50 % Materialanteil in Tabelle B.18. Ähnliche Ergebnisse wie Abb. 8.9b bekommt man auch mit Vollgitter-Interpolation oder dünngitterbasiert mit stückweise linearen Basisfunktionen, sodass Einflüsse von Implementierungsfehlern oder Basisfunktionen weitgehend ausgeschlossen werden können.

Insgesamt kann man sagen, dass sowohl ohne als auch mit Scherwinkeloptimierung dünngitterbasierte B-Spline-Interpolation bei Verwendung geeigneter Gitter akzeptable Resultate liefert. Dabei benötigt die Dünngitter-Interpolation nur einen Bruchteil der Gitterpunkte-Anzahl der Vollgitter-Interpolationsmethoden. Speziell für ein Mikrozellen-Modell mit mehr Parametern wäre die Dünngitter-Interpolation deutlich praktikabler.

9 Fazit

In dieser Arbeit haben wir eine neue, dünngitterbasierte, gradientenfreie Methode zur Optimierung von hinreichend glatten Funktionen $f\colon [0,1]^d \to \mathbb{R}$ entwickelt. Nach einer Einführung über dünne Gitter haben wir die stückweise linearen Basisfunktionen durch glattere Funktionen wie B-Spline- und Mexican-Hat-Funktionen ersetzt. Anschließend haben wir Verfahren zur adaptiven Gittererzeugung und die Probleme bei der Hierarchisierung beschrieben. Nach einem Überblick über verschiedene Optimierungsverfahren haben wir mit der Implementierung namens `sg::opt` die Funktions- und Leistungsfähigkeit der Methode für analytische sowie reale Beispiele getestet. Schließlich haben wir die Anwendbarkeit dünngitterbasierter B-Spline-Interpolation bei einem weiteren Teilgebiet der Optimierung, der Formoptimierung, überprüft.

Die in dieser Arbeit hergeleitete Optimierungsmethode kann teilweise durchaus mit etablierten gradientenfreien Verfahren mithalten. Weil die Methode mit Interpolation durch glattere Basisfunktionen arbeitet, sollte die zu optimierende Zielfunktion eine gewisse Glattheitsvoraussetzung erfüllen. Stetigkeit ist unabkömmlich, besser ist aber stetige Differenzierbarkeit – je glatter, desto besser. In Kapitel 7 haben wir gesehen, dass einige zusätzliche Eigenschaften der Zielfunktion die effiziente Lösung durch unsere Optimierungsmethode begünstigen. Dazu gehören das Vorhandensein nur weniger lokaler Minima und kleine zweite Ableitungen (darüber hinaus eine gut konditionierte Hesse-Matrix in der Minimalstelle). Bei „schwierigeren“ Testfunktionen, die diese Merkmale nicht besitzen, oder einer größeren Zahl N von Auswertungspunkten schneiden die in Abschnitt 5.2 beschriebenen gradientenfreien Optimierungsverfahren meistens besser ab. Das gilt auch für geringe Dimensionalitäten d. Die in der Arbeit entwickelte dünngitterbasierte Methode spielt ihre Vorteile eher bei mittelgroßem d und N aus. Insgesamt lässt sich festhalten: Auch wenn die Qualität der Ergebnisse unserer Methode mitunter variiert, so gab es bei kaum einer Testfunktion Ausreißer, die eine fehlende Konvergenz angezeigt hätten (im Vergleich zu den gradientenfreien Verfahren). Das spricht für eine gute Robustheit des entwickelten Verfahrens.

Die B-Spline-Basisfunktionen waren bei den Testfunktionen ihren stückweise linearen Pendants zumeist deutlich überlegen. Wir haben im letzten Kapitel 8 gesehen, dass die B-Spline-Funktionen auch bei anderen gradientenbasierten Optimierungsverfahren in der Formoptimierung entscheidende Vorteile gegenüber den stückweise linearen Funktionen bringen. Die dünngitterbasierte B-Spline-Interpolation braucht sich bei diesem Anwendungsbeispiel im Vergleich auch nicht vor anderen Vollgitter-Interpolationsmethoden zu verstecken, die wesentlich mehr Daten zur Verfügung haben.

A Testfunktionen

Ackley-Funktion (Ack, $d \in \mathbb{N}$)

$$f(\vec{x}) := 20 + \mathrm{e} - 20\exp\left(-\frac{\|\vec{x}\|_2}{5\sqrt{d}}\right) - \exp\left(\frac{1}{d}\sum_{t=1}^{d}\cos(2\pi x_t)\right), \quad \vec{x} \in [-1,9]^d, \\ \vec{x}_{\mathrm{opt}} = \vec{0}, \quad f_{\mathrm{opt}} = 0 \tag{A.1}$$

Beale-Funktion (Be, $d = 2$)

$$f(\vec{x}) := (1{,}5 - x_1(1-x_2))^2 + (2{,}25 - x_1(1-x_2^2))^2 + (2{,}625 - x_1(1-x_2^3))^2, \\ \vec{x} \in [-5,5]^2, \quad \vec{x}_{\mathrm{opt}} = (3; 0{,}5), \quad f_{\mathrm{opt}} = 0 \tag{A.2}$$

Branin-Funktion (Br, $d = 2$)

$$f(\vec{x}) := \left(x_2 - 5{,}1x_1^2/(4\pi^2) + 5x_1/\pi - 6\right)^2 + 10\left(1 - 1/(8\pi)\right)\cos x_1 + 10, \\ \vec{x} \in [-5,10] \times [0,15], \quad \vec{x}_{\mathrm{opt}} \in \{(-\pi; 12{,}275), (\pi; 2{,}275), (9{,}42478; 2{,}475)\}, \\ f_{\mathrm{opt}} = 0{,}397887 \tag{A.3}$$

Easom-Funktion (Ea, $d = 2$)

$$f(\vec{x}) := -\cos x_1 \cos x_2 \exp(-(x_1-\pi)^2 - (x_2-\pi)^2), \quad \vec{x} \in [-100,100]^2, \\ \vec{x}_{\mathrm{opt}} = (\pi, \pi), \quad f_{\mathrm{opt}} = -1 \tag{A.4}$$

Eggholder-Funktion (Egg, $d = 2$)

$$f(\vec{x}) := -(x_2 + 47)\sin\left(\sqrt{|x_1/2 + x_2 + 47|}\right) - x_1 \sin\left(\sqrt{|x_1 - x_2 - 47|}\right), \\ \vec{x} \in [-512,512]^2, \quad \vec{x}_{\mathrm{opt}} = (512; 404{,}2319), \quad f_{\mathrm{opt}} = -959{,}6407 \tag{A.5}$$

Goldstein-Price-Funktion (GP, $d = 2$)

$$f(\vec{x}) := (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot \\ \cdot (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)), \\ \vec{x} \in [-2,2]^2, \quad \vec{x}_{\mathrm{opt}} = (0, -1), \quad f_{\mathrm{opt}} = 3 \tag{A.6}$$

Griewank-Funktion (Gr, $d \in \mathbb{N}$)

$$f(\vec{x}) := 1 + \frac{\|\vec{x}\|_2^2}{4000} - \prod_{t=1}^{d}\cos\left(\frac{x_t}{\sqrt{t}}\right), \quad \vec{x} \in [-600,600]^d, \quad \vec{x}_{\mathrm{opt}} = \vec{0}, \quad f_{\mathrm{opt}} = 0 \tag{A.7}$$

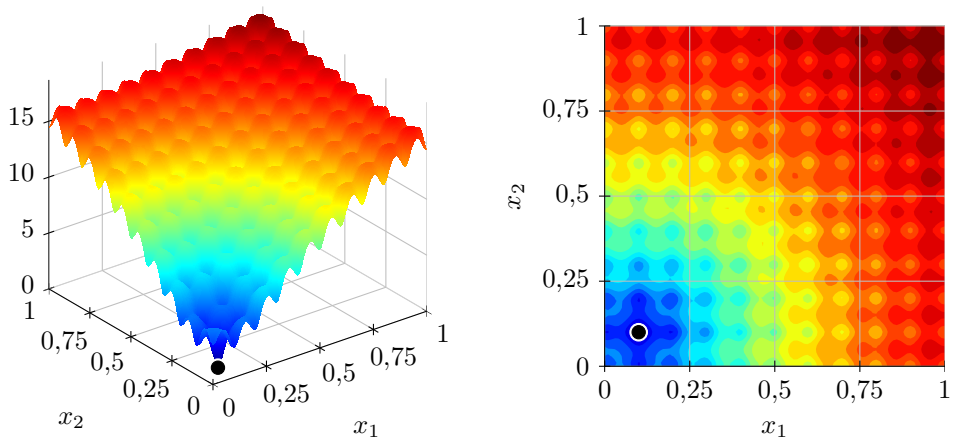


Abbildung A.1: Ackley-Funktion $f(\vec{x})$ in 2D

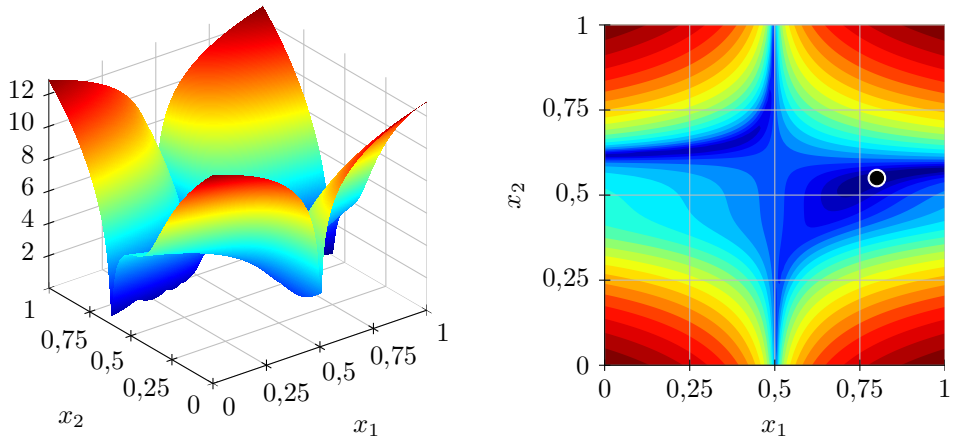


Abbildung A.2: $\ln(f(\vec{x}) + 1)$ mit Beale-Funktion $f(\vec{x})$

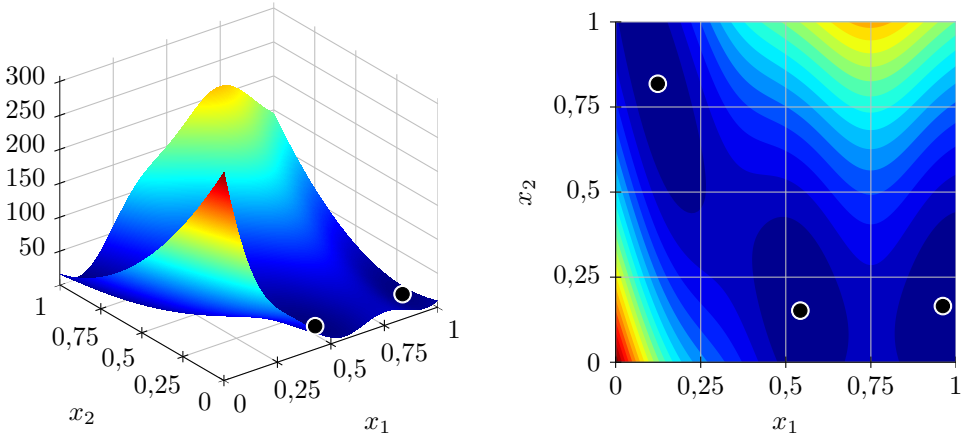


Abbildung A.3: Branin-Funktion $f(\vec{x})$

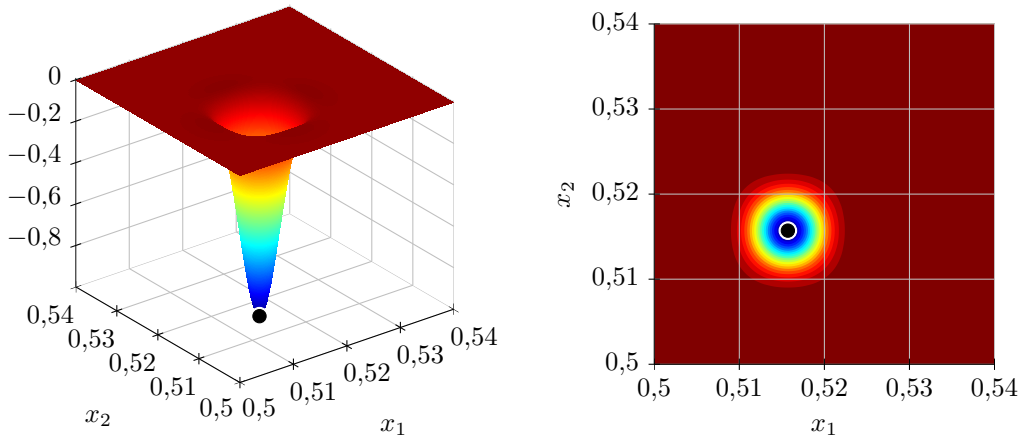


Abbildung A.4: Ausschnitt der Easom-Funktion $f(\vec{x})$

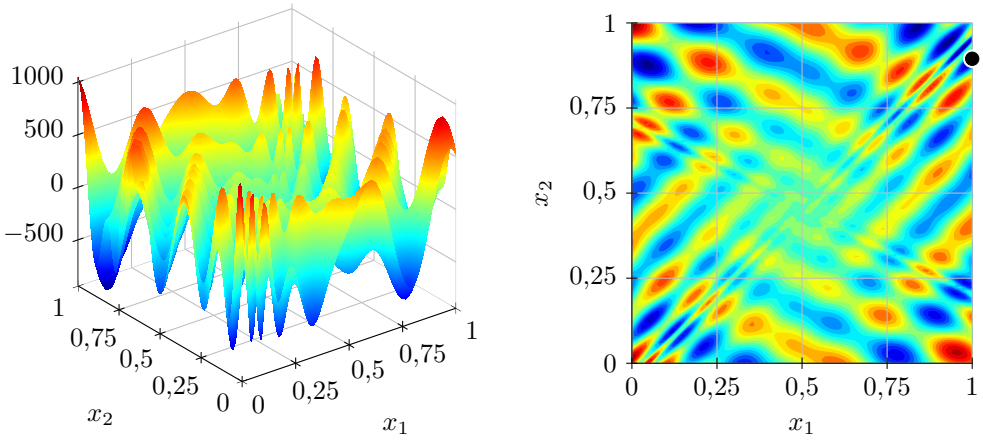


Abbildung A.5: Eggholder-Funktion $f(\vec{x})$

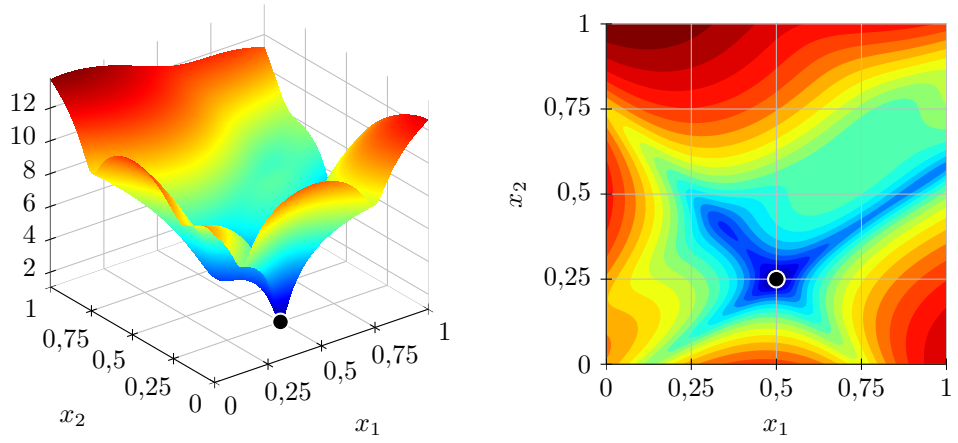


Abbildung A.6: $\ln f(\vec{x})$ mit Goldstein-Price-Funktion $f(\vec{x})$

Hartman3-Funktion (H3, $d = 3$)

$$
\begin{gathered}
f(\vec{x}) := -\sum_{i=1}^{4} a_i \exp\left(-\sum_{t=1}^{3} b_{i,t}(x_t - c_{i,t})^2\right), \\
\vec{a} = \begin{pmatrix} 1 \\ 1{,}2 \\ 3 \\ 3{,}2 \end{pmatrix}, \quad B := \begin{pmatrix} 3 & 10 & 30 \\ 0{,}1 & 10 & 35 \\ 3 & 10 & 30 \\ 0{,}1 & 10 & 35 \end{pmatrix}, \quad C := \begin{pmatrix} 0{,}3689 & 0{,}1170 & 0{,}2673 \\ 0{,}4699 & 0{,}4387 & 0{,}7470 \\ 0{,}1091 & 0{,}8732 & 0{,}5547 \\ 0{,}0382 & 0{,}5743 & 0{,}8828 \end{pmatrix}, \\
\vec{x} \in [0,1]^3, \quad \vec{x}_{\text{opt}} = (0{,}114614; 0{,}555649; 0{,}852547), \quad f_{\text{opt}} = -3{,}862785
\end{gathered}
\tag{A.8}
$$

Hartman6-Funktion (H6, $d = 6$)

$$
\begin{gathered}
f(\vec{x}) := -\sum_{i=1}^{4} a_i \exp\left(-\sum_{t=1}^{6} b_{i,t}(x_t - c_{i,t})^2\right), \\
\vec{a} = \begin{pmatrix} 1 \\ 1{,}2 \\ 3 \\ 3{,}2 \end{pmatrix}, \quad B := \begin{pmatrix} 10 & 3 & 17 & 3{,}5 & 1{,}7 & 8 \\ 0{,}05 & 10 & 17 & 0{,}1 & 8 & 14 \\ 3 & 3{,}5 & 1{,}7 & 10 & 17 & 8 \\ 17 & 8 & 0{,}05 & 10 & 0{,}1 & 14 \end{pmatrix}, \\
C := \begin{pmatrix} 0{,}1312 & 0{,}1696 & 0{,}5569 & 0{,}0124 & 0{,}8283 & 0{,}5886 \\ 0{,}2329 & 0{,}4135 & 0{,}8307 & 0{,}3736 & 0{,}1004 & 0{,}9991 \\ 0{,}2348 & 0{,}1451 & 0{,}3522 & 0{,}2883 & 0{,}3047 & 0{,}6650 \\ 0{,}4047 & 0{,}8828 & 0{,}8732 & 0{,}5743 & 0{,}1091 & 0{,}0381 \end{pmatrix}, \quad \vec{x} \in [0,1]^6, \\
\vec{x}_{\text{opt}} = (0{,}20169; 0{,}150011; 0{,}476874; 0{,}275332; 0{,}311652; 0{,}6573), \\
f_{\text{opt}} = -3{,}322368
\end{gathered}
\tag{A.9}
$$

Himmelblau-Funktion (Hi, $d = 2$)

$$
\begin{gathered}
f(\vec{x}) := (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \quad \vec{x} \in [-5,5]^2, \\
\vec{x}_{\text{opt}} \in \{(3,2), (-2{,}8051; 3{,}1313), (-3{,}7793; -3{,}2832), (3{,}5844; -1{,}8481)\}, \\
f_{\text{opt}} = 0
\end{gathered}
\tag{A.10}
$$

Hölder-Tisch-Funktion (Hö, $d = 2$)

$$
\begin{gathered}
f(\vec{x}) := -\left|\sin x_1 \cos x_2 \exp\left(\left|1 - \frac{\|\vec{x}\|_2}{\pi}\right|\right)\right|, \quad \vec{x} \in [-10,10]^2, \\
\vec{x}_{\text{opt}} \in \{(8{,}0550; \pm 9{,}6646), (-8{,}0550; \pm 9{,}6646)\}, \quad f_{\text{opt}} = -19{,}2085
\end{gathered}
\tag{A.11}
$$

Michalewicz-Funktion (Mi, $d = 2$)

$$
\begin{gathered}
f(\vec{x}) := -\sin x_1 \sin^{20}\left(x_1^2/\pi\right) - \sin x_2 \sin^{20}\left(2x_2^2/\pi\right), \quad \vec{x} \in [0,5]^2, \\
\vec{x}_{\text{opt}} = (2{,}2029055; \pi/2), \quad f_{\text{opt}} = -1{,}8013
\end{gathered}
\tag{A.12}
$$

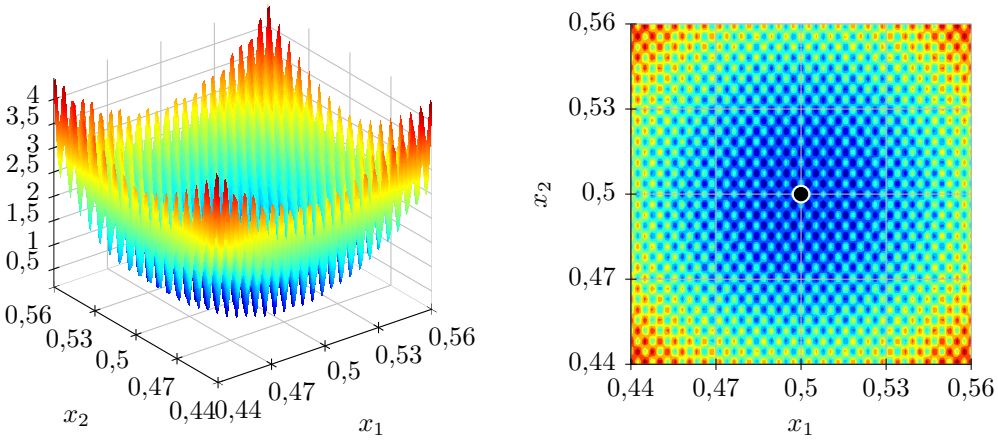


Abbildung A.7: Ausschnitt der Griewank-Funktion $f(\vec{x})$ in 2D

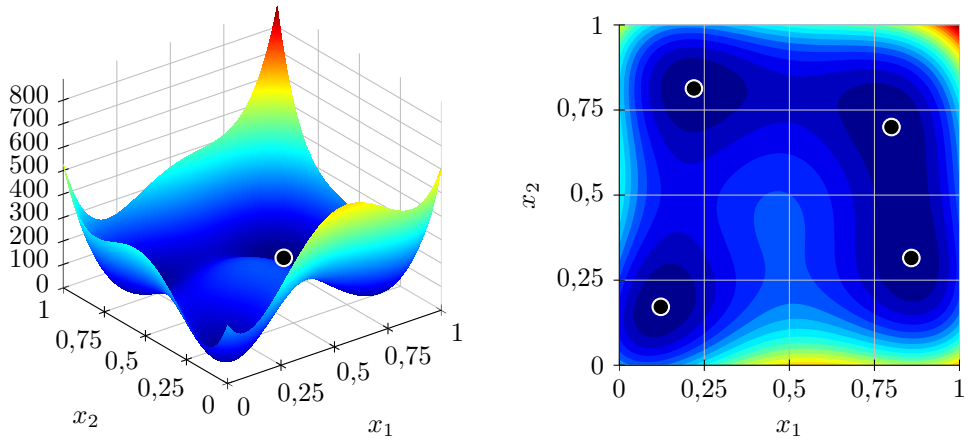


Abbildung A.8: Himmelblau-Funktion $f(\vec{x})$

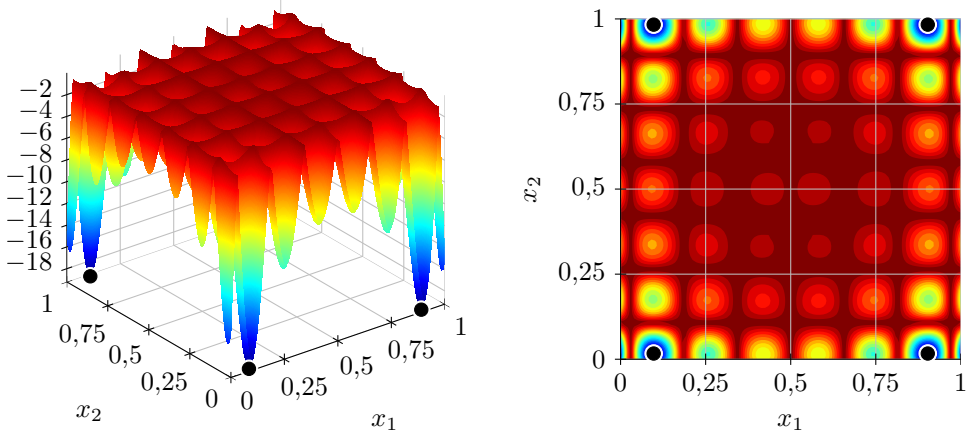


Abbildung A.9: Hölder-Tisch-Funktion $f(\vec{x})$

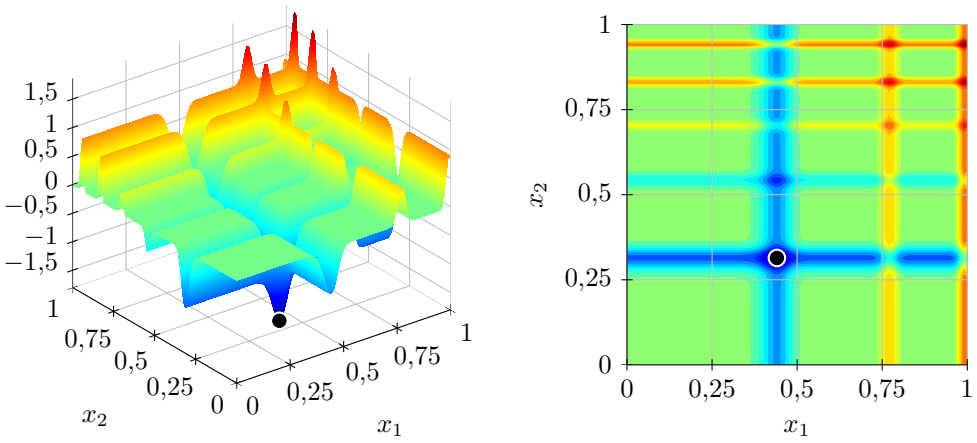


Abbildung A.10: Michalewicz-Funktion $f(\vec{x})$

Mladineo-Funktion (Ml, $d = 2$)

$$f(\vec{x}) := 1 + \frac{1}{2}\|\vec{x}\|_2^2 - \cos(10\ln(2x_1))\cos(10\ln(3x_2)), \quad \vec{x} \in [0{,}01; 1]^2,$$
$$\vec{x}_{\text{opt}} = (0{,}0115; 0{,}0144), \quad f_{\text{opt}} = 0{,}000170 \tag{A.13}$$

Rastrigin-Funktion (Ra, $d \in \mathbb{N}$)

$$f(\vec{x}) := 10d + \|\vec{x}\|_2^2 - 10\sum_{t=1}^{d}\cos(2\pi x_t), \quad \vec{x} \in [-2, 8]^d, \quad \vec{x}_{\text{opt}} = \vec{0}, \quad f_{\text{opt}} = 0 \tag{A.14}$$

Rosenbrock-Funktion (Ro, $d \in \mathbb{N}$)

$$f(\vec{x}) := \sum_{t=1}^{d-1}\left(100(x_{t+1} - x_t^2)^2 + (1 - x_t)^2\right), \quad \vec{x} \in [-5, 10]^d, \quad \vec{x}_{\text{opt}} = \vec{e}, \quad f_{\text{opt}} = 0 \tag{A.15}$$

SHCB-Funktion (SHCB, $d = 2$)

$$f(\vec{x}) := x_1^2\left(4 - 2{,}1x_1^2 + x_1^4/3\right) + x_1x_2 + 4x_2^2\left(x_2^2 - 1\right), \quad \vec{x} \in [-5, 5]^2,$$
$$\vec{x}_{\text{opt}} \in \{(0{,}0898; -0{,}7127), (-0{,}0898; 0{,}7127)\}, \quad f_{\text{opt}} = -1{,}031628 \tag{A.16}$$

Schwefel-Funktion (Sch, $d \in \mathbb{N}$)

$$f(\vec{x}) := -\sum_{t=1}^{d} x_t \sin\sqrt{|x_t|}, \quad \vec{x} \in [-500, 500]^d,$$
$$\vec{x}_{\text{opt}} = (420{,}9687; \ldots; 420{,}9687), \quad f_{\text{opt}} = -418{,}9829d \tag{A.17}$$

Sphere-Funktion (Sph, $d \in \mathbb{N}$)

$$f(\vec{x}) := \|\vec{x}\|_2^2, \quad \vec{x} \in [-1, 9]^d, \quad \vec{x}_{\text{opt}} = \vec{0}, \quad f_{\text{opt}} = 0 \tag{A.18}$$

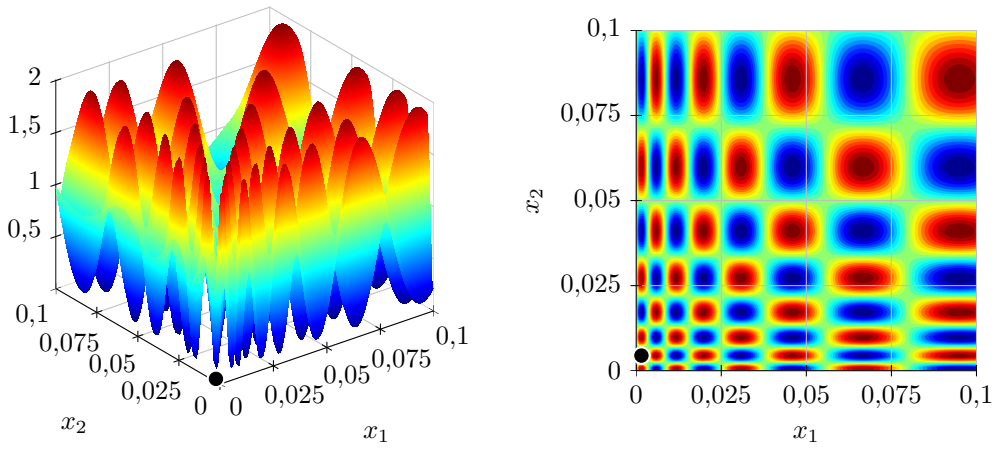


Abbildung A.11: Ausschnitt der Mladineo-Funktion $f(\vec{x})$

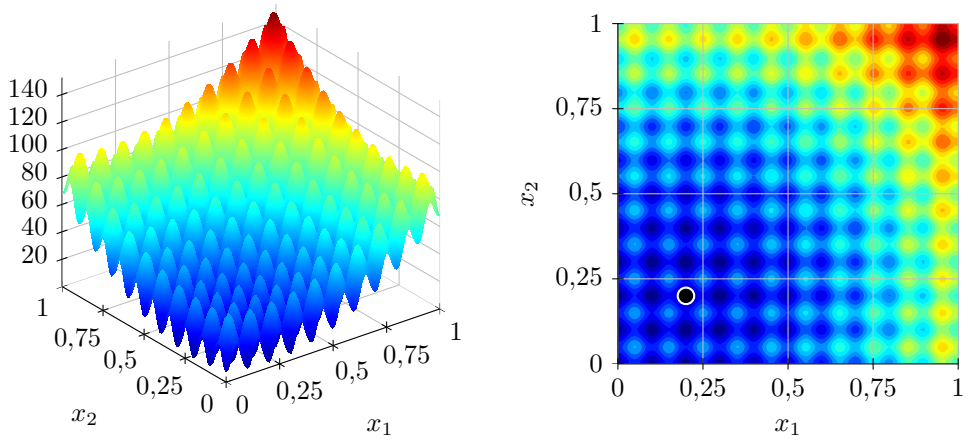


Abbildung A.12: Rastrigin-Funktion $f(\vec{x})$ in 2D

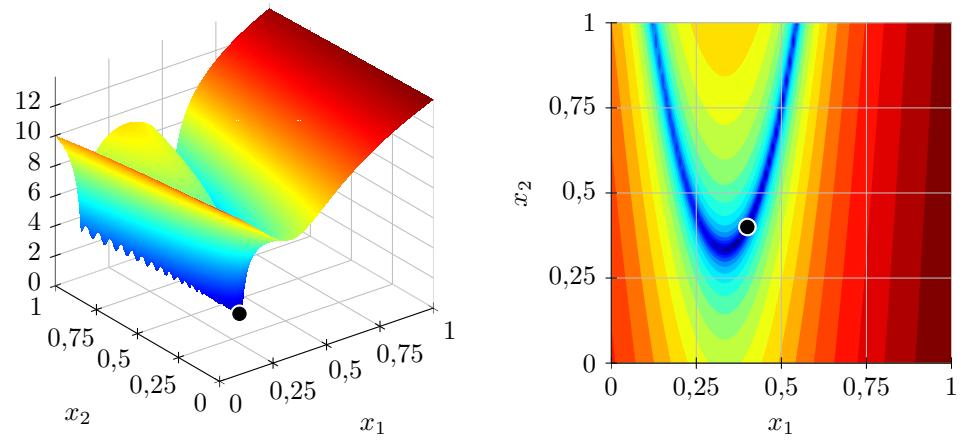


Abbildung A.13: $\ln(f(\vec{x}) + 1)$ mit Rosenbrock-Funktion $f(\vec{x})$ in 2D

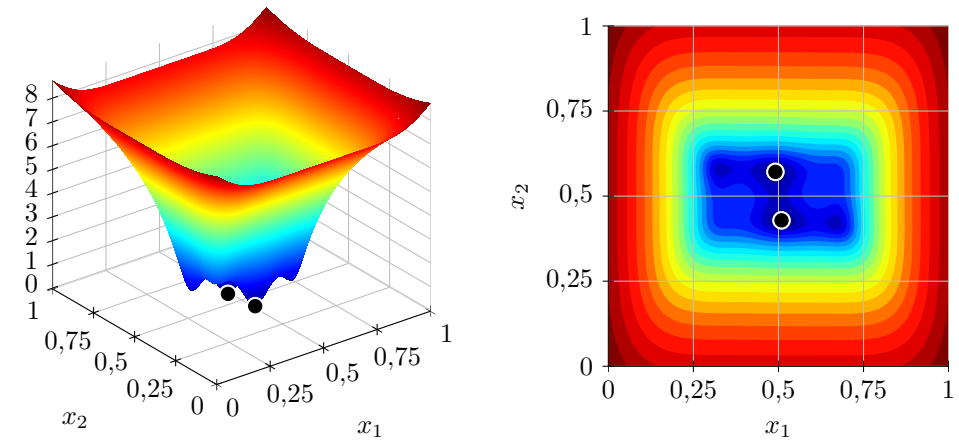


Abbildung A.14: $\ln(f(\vec{x}) + 2)$ mit SHCB-Funktion $f(\vec{x})$

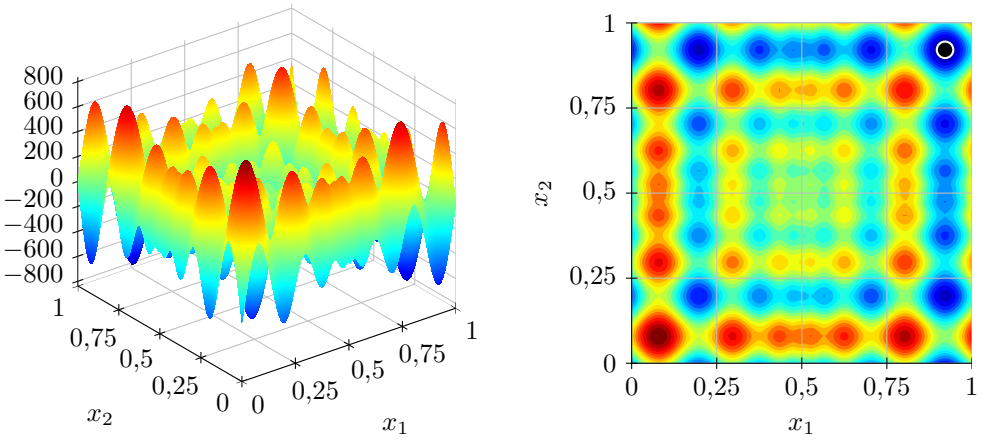


Abbildung A.15: Schwefel-Funktion $f(\vec{x})$ in 2D

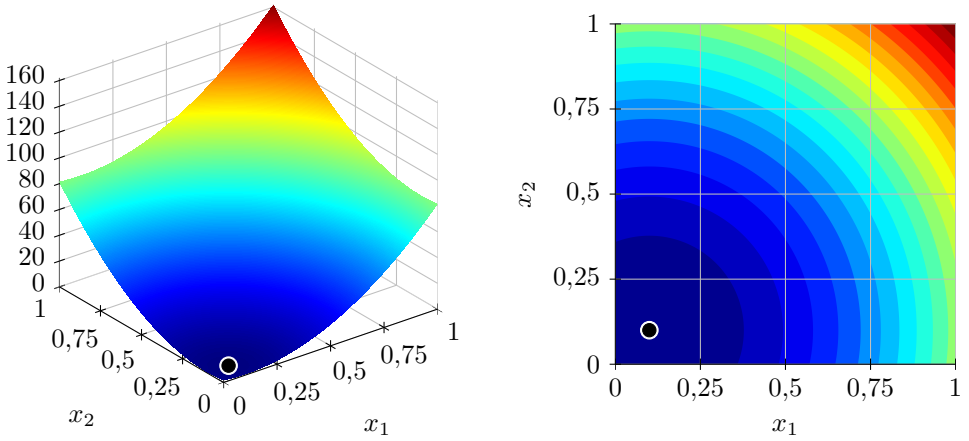


Abbildung A.16: Sphere-Funktion $f(\vec{x})$ in 2D

B Tabellen

B.1 Daten zu Kapitel 7

In diesem Abschnitt befinden sich die Daten, die den Abbildungen aus den Abschnitten 7.1.2, 7.1.3 und 7.1.4 zugrunde liegen. Enthalten sind die Zahlen für die Dimensionalitäten $d = 2, 3, 4, 6, 10$ und die jeweils anwendbaren Testfunktionen – einmal für die Ritter-Novak- und noch einmal für die überschussbasierte Verfeinerung.

Die folgenden Tabellen enthalten je Zelle drei Werte, die übereinander stehen: Oben steht in Grau das Optimum, das aus der Optimierung der stückweise d-linearen Interpolierenden folgt. In der Mitte folgt in Schwarz das Optimum, das man durch Optimierung der (hinreichend glatten) B-Spline-/Mexican-Hat-Interpolierenden mittels gradientenbasierter Verfahren erhält. Schließlich wird unten wieder in Grau das Optimum der direkten Anwendung der gradientenfreien Optimierungsverfahren auf die Zielfunktion angegeben.

Jeder der Werte entspricht dem Abstand der resultierenden Funktionswerte der jeweiligen Verfahren zum optimalen Funktionswert f_{opt} und ist daher nichtnegativ. Alle Werte sind Durchschnittswerte von fünf Durchläufen, wobei die Zielfunktion jedes Mal pseudozufällig in jeder Koordinate normalverteilt mit Mittelwert 0 und Standardabweichung 0,01 verschoben wurde. Zur besseren Vergleichbarkeit sind die Verschiebungen für die verschiedenen f, N und Arten der Gittererzeugung jedoch gleich.

Die Anzahl der signifikanten Stellen der angegebenen Durchschnittswerte μ richtet sich nach der Standardabweichung σ der erhaltenen Ergebnisse: Ist $\sigma \in [10^n, 10^{n+1})$ für ein $n \in \mathbb{Z}$, so wird μ bis zur Stelle mit dem Wert 10^{n-1} angegeben (überflüssige Stellen werden weggerundet). Beispielsweise bedeutet die Angabe von 0,00123, dass der Durchschnitt im Bereich $[0{,}001225; 0{,}001235)$ liegt und die Standardabweichung zwischen 10^{-4} und 10^{-3}.

B.1.1 Ergebnisse mit der Ritter-Novak-Verfeinerung

Tabelle B.1: Ergebnisse für die Ritter-Novak-Verfeinerung ($\alpha = 0{,}85$) und $d = 2$

f \ N	100	200	500	1000	2000	5000	10000
Ack	1,17 0,39 0,35	0,4 0,05 0,001 36	0,0245 0,005 1,56E-10	0,0066 0,0022 4,4E-16	0,0041 0,0014 4,4E-16	0,0026 0,0008 4,4E-16	0,0004 9,8E-5 4,4E-16
Be	0,08 0,08 0,21	0,0698 0,059 0,000 37	0,0055 0,0054 0,000 37	0,000 94 0,000 11 0,000 37	6,4E-5 3,2E-8 0,000 24	1,19E-6 1,1E-5 1,8E-11	2E-7 2,2E-7 7E-23
Br	0,85 1,6E-5 0,2	0,0114 0,0056 6,2E-6	0,0094 7E-7 0	0,000 66 1,2E-10 0	5,2E-5 3E-11 0	1,2E-6 1,4E-10 0	2,9E-8 1,3E-10 0
Ea	0,97 0,97 0,953	0,968 0,968 0,6	0,968 0,968 0,6	0,968 0,968 0,55	0,57 0,57 0,43	0,0012 1,1E-9 1,1E-7	7,8E-6 2,5E-11 0
Egg	430 420 167	401 376 146	166 187 130	58 42 91	24 24 26	6,7 6,8 12	0,061 0,032 11
GP	0,043 0,041 16	0,032 0,014 12	0,013 0,005 8	0,019 1,1E-5 1,8	0,0036 3,6E-11 0,0074	4,9E-5 4,7E-5 8E-11	9,7E-7 1,6E-5 0
Gr	0,27 0,18 0,41	0,16 0,041 0,1	0,0463 0,0089 0,1	0,13 0,029 0,1	0,068 0,073 0,041	0,069 0,053 0,005	0,043 0,027 0,003
Hi	0,46 0,0077 0,042	0,051 9E-6 1,44E-6	0,008 2,6E-9 1,9E-20	3,5E-5 2,2E-14 3,9E-30	1,55E-5 1,9E-12 3,9E-30	5,8E-6 8,6E-11 3,9E-30	4,6E-7 5,6E-10 3,9E-30
Hö	14,468 13,94 1,56	0,8 0,19 1,56	0,012 0,0009 1,09	0,0008 4,1E-7 0,42	0,0007 3,6E-8 0,0037	7,1E-6 1,1E-10 1E-9	6,4E-7 0 0
Mi	0,022 0,0008 0,16	0,0065 6E-7 6E-5	0,0008 4,4E-10 5,8E-16	3,1E-5 1,1E-11 4,4E-16	1,3E-5 4,8E-13 4,4E-16	1,8E-6 1,4E-11 4,4E-16	1,29E-7 2,1E-12 2,220 45E-16
Ml	0,139 0,139 0,184	0,0776 0,063 0,112	0,039 0,0315 0,1	0,032 0,024 0,082	0,0177 0,0163 0,057	0,0076 0,0076 0,0157	0,0047 0,0037 0,0088
Ra	2,56 0,8 2,9	0,34 0,000 16 1,09	0,23 4,9E-7 0,994 959	0,0047 1,6E-8 0,81	0,000 11 1,8E-10 0,12	6E-6 2,2E-9 1,5E-10	2,7E-6 3,1E-10 0
Ro	0,83 0,106 2,43	0,67 0,064 0,63	0,12 0,000 111 7,2E-10	0,048 9E-5 2,7E-30	0,0097 1,3E-6 2,7E-30	0,0019 0,000 29 2,7E-30	0,000 11 9E-5 2,7E-30
SHCB	0,015 2,7E-7 0,0092	0,0014 3,8E-9 4,7E-7	0,000 35 3,5E-11 0	3,3E-5 1E-13 0	1,7E-5 1,5E-13 0	1,8E-7 1,6E-7 0	8,4E-8 5,9E-8 0
Sch	76 24 4,7	10,7 0,035 7,7E-5	1,2 2,1E-7 4,5E-14	0,052 3E-10 4,5E-14	0,0011 1,4E-7 4,5E-14	0,000 13 7E-8 4,5E-14	1,7E-5 2E-8 0
Sph	0,037 0,0017 0,0065	0,0079 8E-5 2,24E-7	5E-5 3,5E-10 2,7E-21	6,4E-6 4,4E-15 4,9E-33	4E-6 1,3E-9 4,9E-33	3,1E-7 3,2E-9 4,9E-33	1,7E-8 3,3E-9 4,9E-33

Tabelle B.2: Ergebnisse für die Ritter-Novak-Verfeinerung ($\alpha = 0{,}85$) und $d = 3$

f \ N	100	500	1000	5000	10000	15000
Ack	2,46 2,46 5,4	0,55 0,043 4,5	0,25 0,046 2,63	0,012 0,0028 0,0014	0,0069 0,0018 4,1E-7	0,0056 0,0018 5,8E-8
Gr	0,55 0,55 0,84	0,22 0,19 0,15	0,12 0,097 0,15	0,129 0,134 0,078	0,11 0,105 0,0134	0,11 0,101 0,0133
H3	0,116 0,056 0,052	0,0011 1,8E-5 0	0,000 45 1,8E-7 0	6,1E-6 0 0	4,7E-7 0 0	3E-7 1E-9 0
Ra	7,1 6,7 19	0,59 0,000 21 10,1	2,6 1,2 7,2	0,019 2,4E-8 0,74	0,0017 2,8E-7 4,6E-7	0,0017 4,2E-8 6,3E-10
Ro	12 4,2 18	3,05 0,024 0,56	1,7 0,000 64 0,033	0,21 1,6E-5 9,6E-29	0,099 3,1E-7 9,6E-29	0,036 3,2E-6 9,6E-29
Sch	670 600 261	33 0,037 118,438	7 0,0018 118,438	0,25 2,6E-7 0,28	0,0029 5,6E-8 4E-7	0,0018 4,2E-7 2,5E-10
Sph	0,191 0,191 2,7	0,017 0,000 15 2,8E-11	0,007 1,5E-7 6E-24	2,6E-5 5,7E-16 1,2E-32	1,2E-5 2,5E-11 1,2E-32	7,8E-6 9E-9 1,2E-32

Tabelle B.3: Ergebnisse für die Ritter-Novak-Verfeinerung ($\alpha = 0{,}85$) und $d = 4$

f \ N	100	500	1000	5000	10000	15000
Ack	3,84 4,1 6,7	0,572 0,091 6,47	0,46 0,04 4,7	0,0225 0,0068 0,108	0,017 0,0043 0,000 73	0,02 0,0042 1,15E-5
Gr	0,72 0,57 1,29	0,4 0,38 0,16	0,39 0,38 0,11	0,24 0,165 0,11	0,27 0,27 0,077	0,19 0,158 0,022
Ra	17,8 23,7 35,4	3,3 1,2 26,1	0,53 0,000 35 17,2	0,65 0,8 4,02	0,23 7E-8 1,45	0,27 3,9E-8 0,122
Ro	206 11,8 650	8,5 0,8 1,65	4,5 0,051 0,013	1,37 7,4E-5 1,5E-28	0,7 1,2E-5 1,5E-28	0,64 1,2E-6 1,5E-28
Sch	1150 1040 680	200 72 670	42 0,047 570	7,2 0,000 61 154	1,6 8E-6 5,7	0,25 1,1E-5 0,0028
Sph	3,84 1,44 6,9	0,0376 0,0018 6,8E-7	0,0251 0,0008 8E-16	0,000 19 1,4E-8 2E-32	7,5E-5 6,3E-15 2E-32	0,000 27 3,2E-15 2E-32

Tabelle B.4: Ergebnisse für die Ritter-Novak-Verfeinerung ($\alpha = 0{,}85$) und $d = 6$

f \ N	100	500	1000	5000	10000	15000
Ack	4,02 3,92 7,7	3 3 7,7	2,3 2 7,4	1,5 1,6 2,88	1,1 1 0,248	0,14 0,049 0,025
Gr	1,1 0,68 1,18	0,84 0,82 0,27	0,61 0,58 0,075	0,71 0,51 0,075	0,63 0,63 0,075	0,54 0,64 0,075
H6	2,021 0,666 1,04	0,41 0,25 0,0014	0,047 0,034 3,5E-8	0,0038 1,5E-5 0	0,0026 1,4E-6 0	0,0025 2,5E-6 0
Ra	40,5 49,1 51	21,4 24 47,5	13,3 7,9 46,9	7,3 4,2 21,4	5,9 2,4 9	6,3 3,6 6,88
Ro	1530 211 4600	169 4,2 3,1	73 1,6 2	8,2 0,033 3,8E-27	4,3 0,0008 2,9E-27	4,9 7,8E-5 2,9E-27
Sch	2070 2246 1272	1360 1250 1272	350 370 1267	217 210 790	292 190 520	200 140 341
Sph	2,04 2,58 24	0,23 0,23 0,0054	0,066 0,024 2,6E-6	0,0058 0,005 3E-31	0,0057 4,6E-7 3E-31	0,0057 3,4E-7 3E-31

Tabelle B.5: Ergebnisse für die Ritter-Novak-Verfeinerung ($\alpha = 0{,}85$) und $d = 10$

f \ N	100	500	1000	5000	10000	15000
Ack	5,3 6 9,5	5,5 5,2 9,5	4,17 3,83 9,5	3,88 3,84 7,46	4,01 3,92 5,31	3,81 3,71 3,39
Gr	1,308 0,149 1,308	1,052 0,25 1,086	0,948 0,53 0,298	0,7 0,67 0,067	0,78 0,6 0,067	0,68 0,7 0,067
Ra	92,9 89,4 104,2	79,6 80,4 90,8	74,7 69,2 89,67	48 13,3 81,2	39,4 12,3 58	42,8 16,9 40,6
Ro	1860 510 9800	1680 304 1550	346 145 81	228 0,137 16	226 3,1 5	217 2,6 2,2
Sch	2590 3630 2470	2860 3070 2470	2250 2545 2470	1889 1440 2260	1660 1090 1820	1610 1050 1640
Sph	4,23 2,21 60	4,2 3,1 4,2	0,375 0,375 0,015	0,042 0,042 6E-16	0,023 0,023 2,4E-30	0,022 0,022 2,4E-30

B.1.2 Ergebnisse mit der überschussbasierten Verfeinerung

Tabelle B.6: Ergebnisse für die Überschuss-Verfeinerung ($\alpha = 0{,}2$) und $d = 2$

f \ N	100	200	500	1000	2000	5000	10000
Ack	1,54 0,9 0,35	1,03 1,1 0,001 36	0,26 0,06 1,41E-10	0,174 0,0212 4,4E-16	0,053 0,0097 4,4E-16	0,037 0,0078 4,4E-16	0,029 0,0044 4,4E-16
Be	0,95 0,89 0,21	0,89 0,78 0,000 37	0,64 0,64 0,000 37	0,32 0,098 0,000 37	0,24 0,084 0,000 24	0,056 0,0079 1,8E-11	0,022 0,0022 7E-23
Br	0,067 0,000 61 0,2	0,018 3,9E-7 3,6E-6	0,0042 2,6E-10 0	0,001 34 4E-11 0	0,0003 8,3E-11 0	2,9E-5 2,6E-10 0	1,81E-5 1,4E-10 0
Ea	1 1 0,953	0,983 0,973 0,6	0,7 0,65 0,6	0,21 0,041 0,55	0,03 0,000 27 0,43	0,0021 1,1E-8 1,1E-7	3,2E-5 6E-11 0
Egg	101 81 167	68 66 146	57 26 130	44 44 91	12,3 15,7 26	7 7,7 12	3,6 3,75 11
GP	0,65 0,65 16	0,65 0,65 12	0,65 0,65 8	0,65 0,026 1,8	0,19 0,000 53 0,0074	0,029 8,4E-6 8E-11	0,042 4,3E-7 0
Gr	0,48 0,48 0,41	0,48 0,46 0,1	0,45 0,41 0,1	0,43 0,39 0,1	0,4 0,42 0,041	0,184 0,21 0,005	0,2 0,22 0,003
Hi	1,02 0,000 46 0,042	0,093 2E-6 1,25E-6	0,025 3,2E-10 1,5E-20	0,0092 5,6E-13 3,9E-30	0,0025 4,2E-11 3,9E-30	0,000 32 2,6E-11 3,9E-30	6,6E-5 1,5E-11 3,9E-30
Hö	3,32 4,7 1,56	2,1 1,31 1,56	0,28 0,33 1,09	0,29 0,000 34 0,42	0,032 6,8E-5 0,0037	0,019 4,1E-5 1E-9	0,0025 1E-8 0
Mi	0,37 0,22 0,16	0,2 0,16 6E-5	0,0012 9E-10 5,8E-16	2,5E-5 3,3E-11 4,4E-16	1,25E-5 1,4E-11 4,4E-16	3,8E-6 9E-11 4,4E-16	4,9E-7 1,3E-10 2,220 45E-16
Ml	0,205 0,202 0,184	0,139 0,14 0,112	0,121 0,121 0,1	0,094 0,081 0,082	0,056 0,0381 0,057	0,02 0,0118 0,0157	0,0104 0,0064 0,0088
Ra	2,24 1,1 2,9	0,87 0,21 1,09	0,19 0,01 0,994 959	0,15 0,01 0,81	0,0022 9,3E-11 0,12	0,000 48 6E-11 1,5E-10	0,000 22 2E-10 0
Ro	5,3 0,47 2,43	1,2 0,189 0,61	0,54 0,0013 5E-10	0,42 0,000 141 2,7E-30	0,077 1,8E-5 2,7E-30	0,02 0,000 16 2,7E-30	0,0121 0,000 14 2,7E-30
SHCB	0,81 0,095 0,0092	0,06 0,031 4,7E-7	0,06 0,000 83 0	0,037 0,000 39 0	0,028 0,000 37 0	0,001 23 2,9E-6 0	0,000 36 9E-8 0
Sch	280 180 4,7	190 100 7,7E-5	0,87 6,1E-7 4,5E-14	0,17 1,17E-8 4,5E-14	0,055 1,32E-9 4,5E-14	0,0057 3,8E-9 4,5E-14	0,001 28 5,3E-9 0
Sph	0,013 4,7E-5 0,0065	0,0083 7,2E-7 2,24E-7	0,0014 1,5E-16 2,7E-21	0,000 14 5,3E-17 4,9E-33	7,1E-5 2,4E-9 4,9E-33	1,2E-5 7E-10 4,9E-33	5,4E-6 3,6E-9 4,9E-33

Tabelle B.7: Ergebnisse für die Überschuss-Verfeinerung ($\alpha = 0{,}2$) und $d = 3$

f \ N	100	500	1000	5000	10000	15000
Ack	3,94 3,98 5,4	3,36 3,24 4,5	1,88 1,89 2,63	0,29 0,09 0,0014	0,18 0,0143 4,1E-7	0,132 0,0182 5,8E-8
Gr	0,9 0,77 0,84	0,8 0,64 0,15	0,76 0,64 0,15	0,59 0,47 0,078	0,51 0,36 0,0134	0,5 0,46 0,0133
H3	0,198 0,167 0,055	0,0082 0,0054 0	0,0054 5,4E-6 0	0,000 29 5E-8 0	0,000 12 9,4E-9 0	9,7E-5 0 0
Ra	11,6 13,1 19	1,2 1 10,1	0,79 0,0042 7,2	0,0102 3,7E-10 0,74	0,0021 2,7E-10 4,6E-7	0,000 48 3,4E-10 6,3E-10
Ro	140 5,89 18	54 2 0,56	9,4 0,19 0,033	0,51 0,000 37 9,6E-29	0,115 0,000 109 9,6E-29	0,041 0,000 12 9,6E-29
Sch	720 440 261	87 71 118,438	6,1 0,005 118,438	0,14 1,3E-9 0,28	0,019 2,4E-9 4E-7	0,0123 3,1E-9 2,5E-10
Sph	0,59 0,0102 2,7	0,0064 2,8E-9 2,8E-11	0,0022 2E-13 6E-24	9,3E-5 1,7E-9 1,2E-32	2,7E-5 3,3E-9 1,2E-32	1,26E-5 6E-9 1,2E-32

Tabelle B.8: Ergebnisse für die Überschuss-Verfeinerung ($\alpha = 0{,}2$) und $d = 4$

f \ N	100	500	1000	5000	10000	15000
Ack	3,97 3,804 6,7	2,77 3,66 6,47	3,76 4,01 4,7	3,53 3,39 0,108	1,6 2,2 0,000 73	0,3 1,95 1,15E-5
Gr	1,29 0,66 1,29	1,01 0,62 0,15	0,95 0,55 0,11	0,88 0,87 0,11	0,71 0,71 0,077	0,69 0,58 0,022
Ra	19,3 32,4 35,4	10,7 5,1 26,1	3,9 0,63 17,2	0,29 1,33E-7 4,02	0,6 1,1E-9 1,45	0,6 3,2E-10 0,122
Ro	650 71 650	670 19,7 1,64	120 2 0,0077	2,45 0,001 15 1,5E-28	1,07 0,000 14 1,5E-28	0,73 0,000 22 1,5E-28
Sch	1150 790 680	430 500 670	6,5 7,5E-5 570	0,45 4,3E-6 154	0,21 2,8E-5 5,7	0,0254 4,6E-6 0,0027
Sph	1,53 0,95 6,9	0,029 7,6E-6 6,8E-7	0,0061 3E-9 3E-16	0,000 274 1E-16 2E-32	9,3E-5 1E-16 2E-32	2,02E-5 6,9E-9 2E-32

Tabelle B.9: Ergebnisse für die Überschuss-Verfeinerung ($\alpha = 0{,}2$) und $d = 6$

f \ N	100	500	1000	5000	10000	15000
Ack	4,02 3,92 7,7	5,27 5,52 7,7	4,15 4,82 7,4	4,09 4,59 2,88	4,09 4,42 0,248	3,79 4,13 0,025
Gr	1,1 0,68 1,18	1,13 0,53 0,27	1,14 0,59 0,075	1,044 0,67 0,075	0,91 0,68 0,075	0,96 0,63 0,075
H6	2,021 0,666 1,04	0,6 0,347 0,0014	0,419 0,285 6E-8	0,132 0,0044 0	0,037 0,001 41 0	0,036 0,000 712 0
Ra	40,5 49,1 51	30,1 37 47,5	13,8 10,9 46,9	6,1 4,6 21,4	6,6 3,2 9	6,9 2,19 6,88
Ro	1530 211 4600	1010 190 3,1	370 74 2	16,5 0,29 3,8E-27	8,6 0,166 2,9E-27	6,59 0,0036 2,9E-27
Sch	2070 2246 1272	1590 1120 1272	890 680 1267	380 272 790	350 210 520	330 210 341
Sph	2,04 2,58 24	0,044 0,0019 0,0054	0,0294 8,6E-5 2,6E-6	0,0018 9E-13 3E-31	0,001 24 4,2E-16 3E-31	0,000 338 5,3E-16 3E-31

Tabelle B.10: Ergebnisse für die Überschuss-Verfeinerung ($\alpha = 0{,}2$) und $d = 10$

f \ N	100	500	1000	5000	10000	15000
Ack	5,3 6 9,5	5,39 6,02 9,5	5,44 5,18 9,5	6,4 5,86 7,46	4,98 6 5,31	4,88 5,7 3,39
Gr	1,308 0,149 1,308	1,308 0,28 1,086	1,308 0,13 0,298	1,223 0,23 0,067	1,048 0,35 0,067	1,038 0,31 0,067
Ra	92,9 89,4 104,2	92,9 92,6 90,8	74 86,7 89,67	42,8 15 81,2	40,3 20 58	48,4 20 40,6
Ro	1860 510 9800	2300 446 1550	2100 429 80	233 8,7 16	202 7,1 5	135 1 2,2
Sch	2590 3630 2470	3000 2090 2470	2400 1890 2470	1710 1080 2260	1710 1130 1820	1630 1010 1640
Sph	4,23 2,21 60	4,64 2,17 4,2	0,81 0,15 0,015	0,046 0,000 13 6E-16	0,0151 7E-9 2,4E-30	0,0119 5E-10 2,4E-30

B.2 Daten zu Kapitel 8

Dieser Abschnitt enthält die Ergebnisse, die man bei der Formoptimierung durch Homogenisierung mittels Interpolation von Elastizitätstensoren erhält (siehe Kapitel 8). Die Tabellen teilen sich grob auf in den Fall ohne Scherwinkeloptimierung, bei dem der Scherwinkel $\varphi = 0$ ist und nur 2D-Interpolierende betrachtet werden, und den Fall mit Scherwinkeloptimierung, bei dem der Scherwinkel variabel ist und mit 3D-Interpolierenden gearbeitet wird. Die Konfiguration des verwendeten Testrechners ist am Anfang von Kapitel 6 beschrieben.

In der ersten Spalte stehen „VG“ und „DG“ für „volles Gitter“ bzw. „dünnes Gitter“. Reguläre Gitter, beispielsweise mit Level 7, werden durch „reg. 7“ angezeigt, während „verf.“ für das in Abschnitt 8.4.2 beschriebene verfeinerte Gitter steht. Die dritte Spalte zeigt an, ob der Optimierungsalgorithmus erfolgreich konvergiert hat oder ob aufgrund von numerischen Schwierigkeiten vorzeitig abgebrochen werden musste. Die weiteren Spalten enthalten den zurückgegebenen Wert des Compliance-Funktionals (8.1), die Anzahl der benötigten Iterationen, die in Anspruch genommene Zeit und den maximal verbrauchten Speicher.

Bei manchen Zeilen ist die Interpolierende so ungenau, dass der Optimierer kein vernünftiges Ergebnis erhalten hat. Offensichtlich aufgrund physikalisch unsinniger Tensoreinträge, die der Optimierer durch die Interpolation bekommt, erhält der Optimierer in solchen Fällen sehr kleine Zielfunktionswerte wie $-10^6, -10^7, -10^8$ usw., obwohl das Compliance-Funktional sinnvollerweise nichtnegativ sein sollte. In diesem Fall rechnet der Optimierer sehr lange, ohne wieder in den für Zielfunktionswerte „normalen Bereich“ zu kommen (im Gegenteil, der Zielfunktionswert wird mit jedem Schritt noch kleiner). Daher fehlen in diesem Fall die Werte in den letzten vier Spalten.

B.2.1 Ohne Scherwinkeloptimierung

Tabelle B.11: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}9]^2$ und 35 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	67,25	229	134 s	45 MB
DG, reg. 7	linear	✗	67,39	391	275 s	43 MB

Tabelle B.12: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}98]^2$ und 35 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	65,26	180	108 s	44 MB
VG	B-Splines, $p = 3$	✓	65,28	189	85 s	41 MB
VG	B-Splines, $p = 5$	✓	65,30	234	104 s	43 MB
DG, reg. 7	linear	✗	66,28	2586	23 min	58 MB
DG, reg. 7	mod. linear	✗	65,29	302	236 s	45 MB
DG, reg. 7	mod. B-Splines, $p = 3$	✓	65,17	170	127 s	43 MB
DG, reg. 7	mod. B-Splines, $p = 5$	✓	65,04	164	130 s	44 MB

Tabelle B.13: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}9]^2$ und 50 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	44,08	191	136 s	48 MB
DG, reg. 7	linear	✗	44,16	410	276 s	47 MB

Tabelle B.14: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}98]^2$ und 50 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	42,85	170	128 s	47 MB
VG	B-Splines, $p = 3$	✓	42,83	311	163 s	46 MB
VG	B-Splines, $p = 5$	✓	42,87	432	243 s	47 MB
DG, reg. 7	linear	✗	43,59	3364	38 min	62 MB
DG, reg. 7	mod. linear	✗	43,00	704	546 s	51 MB
DG, reg. 7	mod. B-Splines, $p = 3$	✓	42,80	377	299 s	47 MB
DG, reg. 7	mod. B-Splines, $p = 5$	✓	42,82	204	153 s	47 MB

B.2.2 Mit Scherwinkeloptimierung

Tabelle B.15: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}9]^2 \times [0{,}15; 0{,}85]$ und 35 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	63,41	904	25 min	894 MB
DG, reg. 7	linear	✗	64,99	9193	7 h	96 MB

Tabelle B.16: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}98]^2 \times [0{,}15; 0{,}85]$ und 35 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	61,90	1715	42 min	894 MB
VG	B-Splines, $p = 3$	✓	62,54	1507	36 min	88 MB
VG	B-Splines, $p = 5$	✓	62,55	1273	32 min	94 MB
DG, reg. 7	linear	✗	63,58	6357	6 h	84 MB
DG, reg. 7	mod. linear	✗				
DG, reg. 7	mod. B-Splines, $p = 3$	✗				
DG, reg. 7	mod. B-Splines, $p = 5$	✗				
DG, reg. 8	linear	✗	63,37	7541	8 h	90 MB
DG, reg. 8	mod. linear	✗	61,97	4969	4 h	79 MB
DG, reg. 8	mod. B-Splines, $p = 3$	✗				
DG, reg. 8	mod. B-Splines, $p = 5$	✗				
DG, reg. 9	linear	✗	63,04	5587	6 h	89 MB
DG, reg. 9	mod. linear	✗				
DG, reg. 9	mod. B-Splines, $p = 3$	✗				
DG, reg. 9	mod. B-Splines, $p = 5$	✗				
DG, verf.	linear	✗				
DG, verf.	mod. linear	✗				
DG, verf.	mod. B-Splines, $p = 3$	✓	62,63	1451	89 min	59 MB
DG, verf.	mod. B-Splines, $p = 5$	✓	62,86	1223	100 min	58 MB

Tabelle B.17: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}9]^2 \times [0{,}15; 0{,}85]$ und 50 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	42,86	1487	36 min	894 MB
DG, reg. 7	linear	✗	43,37	26962	13 h	166 MB

Tabelle B.18: Ergebnisse für das Optimierungsgebiet $[0{,}02; 0{,}98]^2 \times [0{,}15; 0{,}85]$ und 50 % Materialanteil

Gitter	Methode/Basis	Konv.	Zielfkt.	Iter.	Zeit	Speicher
VG	$\mathcal{C}^1$	✓	41,86	1307	27 min	894 MB
VG	B-Splines, $p = 3$	✓	41,87	1230	29 min	86 MB
VG	B-Splines, $p = 5$	✓	41,91	1289	30 min	92 MB
DG, reg. 7	linear	✗	42,87	7401	7 h	88 MB
DG, reg. 7	mod. linear	✗	42,45	9515	6 h	95 MB
DG, reg. 7	mod. B-Splines, $p = 3$	✗				
DG, reg. 7	mod. B-Splines, $p = 5$	✗				
DG, reg. 8	linear	✗	42,45	8440	9 h	95 MB
DG, reg. 8	mod. linear	✗	41,95	4203	163 min	75 MB
DG, reg. 8	mod. B-Splines, $p = 3$	✓	41,26	1483	139 min	67 MB
DG, reg. 8	mod. B-Splines, $p = 5$	✗				
DG, reg. 9	linear	✗	42,22	9074	8 h	102 MB
DG, reg. 9	mod. linear	✗	42,61	9229	7 h	102 MB
DG, reg. 9	mod. B-Splines, $p = 3$	✓	42,06	1559	16 h	72 MB
DG, reg. 9	mod. B-Splines, $p = 5$	✗				
DG, verf.	linear	✗				
DG, verf.	mod. linear	✗				
DG, verf.	mod. B-Splines, $p = 3$	✓	42,14	1287	84 min	59 MB
DG, verf.	mod. B-Splines, $p = 5$	✓	42,10	1650	130 min	61 MB

Literaturverzeichnis

[1] M. Ankerl: *Optimized Approximative pow() in C / C++*. 25. Jan. 2012. URL: http://martin.ankerl.com/2012/01/25/optimized-approximative-pow-in-c-and-cpp/ (besucht am 08. 07. 2014).

[2] R. Balder: *Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern*. Diss. TU München, Institut für Informatik, 1994.

[3] C. de Boor: *On Calculating with B-Splines*. In: Journal of Approximation Theory 6.1 (1972), S. 50–62. ISSN: 0021-9045. DOI: 10.1016/0021-9045(72)90080-9.

[4] R. Brinks: *On the convergence of derivatives of B-splines to derivatives of the Gaussian function*. In: Computational & Applied Mathematics 27 (2008), S. 79–92. ISSN: 1807-0302.

[5] H.-J. Bungartz: *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*. Diss. TU München, Institut für Informatik, 1992.

[6] H.-J. Bungartz und M. Griebel: *Sparse grids*. In: Acta Numerica 13 (2004), S. 147–269. ISSN: 1474-0508. DOI: 10.1017/S0962492904000182.

[7] H.-J. Bungartz u. a.: *Modellbildung und Simulation. Eine anwendungsorientierte Einführung*. 2. Aufl. Berlin, Heidelberg: Springer, 2013. DOI: 10.1007/978-3-642-37656-6.

[8] C. W. Clenshaw und A. R. Curtis: *A method for numerical integration on an automatic computer*. In: Numerische Mathematik 2 (1 1960), S. 197–205. ISSN: 0029-599X. DOI: 10.1007/BF01386223.

[9] COMSOL Inc.: *Tuning Fork*. Dokumentation mit Beispiel-Modell. COMSOL Multiphysics 4.3b. 2013. URL: http://www.comsol.com/model/tuning-fork-computing-the-eigenfrequency-and-eigenmode-8499 (besucht am 23. 05. 2014).

[10] G. F. Corliss und R. B. Kearfott: *Rigorous Global Search: Industrial Applications*. In: Developments in Reliable Computing. Hrsg. von T. Csendes. Boston: Kluwer Academic Publishers, 1999. DOI: 10.1007/978-94-017-1247-7_1.

[11] M. G. Cox: *The Numerical Evaluation of B-Splines*. In: IMA Journal of Applied Mathematics 10.2 (1972), S. 134–149. ISSN: 0272-4960. DOI: 10.1093/imamat/10.2.134.

[12] L. Dagum und R. Menon: *OpenMP: An Industry Standard API for Shared-Memory Programming*. In: IEEE Computational Science & Engineering 5 (1 1998), S. 46–55. ISSN: 1070-9924. DOI: 10.1109/99.660313.

[13] T. A. Davis: *Algorithm 832: UMFPACK V4.3—An Unsymmetric-pattern Multifrontal Method*. In: ACM Transactions on Mathematical Software 30.2 (2004), S. 196–199. ISSN: 0098-3500. DOI: 10.1145/992200.992206.

[14] M. M. Donahue, G. T. Buzzard und A. E. Rundell: *Robust Parameter Identification with Adaptive Sparse Grid-Based Optimization for Nonlinear Systems Biology Models*. In: Proceedings of the 2009 American Control Conference. St. Louis: IEEE, 2009, S. 5055–5060. DOI: 10.1109/ACC.2009.5160512.

[15] C. Elton und M. Nicholson: *The Ten-Year Cycle in Numbers of the Lynx in Canada*. In: Journal of Animal Ecology 11.2 (1942), S. 215–244.

[16] I. Ferenczi: *Globale Optimierung unter Nebenbedingungen mit dünnen Gittern*. Diplomarbeit. TU München, Zentrum Mathematik, 2005.

[17] F. Gao und L. Han: *Implementing the Nelder-Mead simplex algorithm with adaptive parameters*. In: Computational Optimization and Applications 51 (1 2012), S. 256–277. ISSN: 0926-6003. DOI: 10.1007/s10589-010-9329-3.

[18] J. Garcke: *Sparse Grids in a Nutshell*. In: Sparse Grids and Applications. Hrsg. von J. Garcke und M. Griebel. Bd. 88. Lecture Notes in Computational Science and Engineering. Berlin Heidelberg: Springer, 2013, S. 57–80. DOI: 10.1007/978-3-642-31703-3_3.

[19] J. Garcke, M. Griebel und M. Thess: *Data Mining with Sparse Grids*. In: Computing 67 (3 2001), S. 225–253. ISSN: 0010-485X. DOI: 10.1007/s006070170007.

[20] T. Gerstner und M. Griebel: *Numerical integration using sparse grids*. In: Numerical Algorithms 18 (3–4 1998), S. 209–232. ISSN: 1017-1398. DOI: 10.1023/A:1019129717644.

[21] M. E. Gilpin: *Do Hares Eat Lynx?* In: The American Naturalist 107.957 (1973), S. 727–730. ISSN: 0003-0147.

[22] G. Guennebaud, B. Jacob und andere: *Eigen v3*. 2010. URL: http://eigen.tuxfamily.org/ (besucht am 10. 07. 2014).

[23] S. M. Han, H. Benaroya und T. Wei: *Dynamics of Transversely Vibrating Beams Using Four Engineering Theories*. In: Journal of Sound and Vibration 225.5 (1999), S. 935–988. ISSN: 0022-460X. DOI: 10.1006/jsvi.1999.2257.

[24] S. He: *Computational Modelling with MATLAB. Model Parameter Estimation*. Folien zur Vorlesung. University of Birmingham, 2013. URL: http://www.cs.bham.ac.uk/~szh/teaching/matlabmodeling/Lecture19_body_ModelParameterEstimation.pdf (besucht am 03. 06. 2014).

[25] K. Höllig: *Finite Element Methods with B-Splines*. Philadelphia: SIAM, 2003. DOI: 10.1137/1.9780898717532.

[26] K. Höllig und J. Hörner: *Approximation and Modeling with B-Splines*. Philadelphia: SIAM, 2013.

[27] D. Hübner: *Mehrdimensionale Parametrisierung der Mikrozellen in der Zwei-Skalen-Optimierung*. Masterarbeit. Universität Erlangen-Nürnberg, Department Mathematik, 2014.

[28] M. Kaltenbacher: *Advanced Simulation Tool for the Design of Sensors and Actuators*. In: *Procedia Engineering*. Bd. 5: Eurosensor XXIV Conference. Hrsg. von B. Jakoby und M. J. Vellekoop. Linz: Elsevier, 2010, S. 597–600. DOI: 10.1016/j.proeng.2010.09.180.

[29] S. Kinauer: *Hierarchische Optimierung mit Dünnen Gittern*. Projektarbeit. TU München, Fakultät für Informatik, 2013.

[30] A. Klimke und B. Wohlmuth: *Algorithm 847: spinterp. Piecewise Multilinear Hierarchical Sparse Grid Interpolation in MATLAB*. In: ACM Transactions on Mathematical Software 31.4 (2005), S. 561–579. ISSN: 0098-3500. DOI: 10.1145/1114268.1114275.

[31] F. Lekien und J. E. Marsden: *Tricubic interpolation in three dimensions*. In: International Journal for Numerical Methods in Engineering 63.3 (2005), S. 455–471. ISSN: 1097-0207. DOI: 10.1002/nme.1296.

[32] L. Ljung: *System Identification. Theory for the User*. 2. Aufl. Upper Saddle River, New Jersey: Prentice Hall, 1999.

[33] D. A. MacLulich: *Sunspots and abundance of animals*. In: Journal of the Royal Astronomical Society of Canada 30 (1936), S. 233–246. ISSN: 0035-872X.

[34] The MathWorks, Inc.: *Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation*. Dokumentation mit Beispiel-Daten. MATLAB R2014a, System Identification Toolbox. 2014. URL: http://www.mathworks.de/de/help/ident/examples/represent-nonlinear-dynamics-using-matlab-file-for-grey-box-estimation.html (besucht am 23. 05. 2014).

[35] M. Molga und C. Smutnicki: *Test functions for optimization needs*. 2005. URL: www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf (besucht am 23. 04. 2014).

[36] J. A. Nelder und R. Mead: *A simplex method for function minimization*. In: The Computer Journal 7 (4 1965), S. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308.

[37] E. Novak und K. Ritter: *Global Optimization Using Hyperbolic Cross Points*. In: State of the Art in Global Optimization. Computational Methods and Applications. Hrsg. von C. A. Floudas und P. M. Pardalos. Bd. 7. Nonconvex Optimization and Its Applications. USA: Springer, 1996, S. 19–33. DOI: 10.1007/978-1-4613-3437-8_2.

[38] E. Novak und K. Ritter: *High dimensional integration of smooth functions over cubes*. In: Numerische Mathematik 75 (1 1996), S. 79–97. ISSN: 0029-599X. DOI: 10.1007/s002110050231.

[39] D. Pandey: *Regression with Spatially Adaptive Sparse Grids in Financial Applications*. Masterarbeit. TU München, Fakultät für Informatik, 2008.

[40] D. Pflüger: *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, 2010.

[41] D. Pflüger, J. Blank und A. Heinecke: *Dokumentation von SG^{++}*. 2009. URL: http://www5.in.tum.de/SGpp/releases/index.html (besucht am 05. 07. 2014).

[42] M. J. D. Powell: *Restart procedures for the conjugate gradient method*. In: Mathematical Programming 12 (1 1977), S. 241–254. ISSN: 0025-5610. DOI: 10.1007/BF01593790.

[43] W. H. Press u. a.: *Numerical Recipes. The Art of Scientific Computing*. 3. Aufl. New York: Cambridge University Press, 2007.

[44] R. Reinhardt, A. Hoffmann und T. Gerlach: *Nichtlineare Optimierung. Theorie, Numerik und Experimente*. Berlin Heidelberg: Springer, 2013. DOI: 10.1007/978-3-8274-2949-0.

[45] Y. Renard und J. Pommier: *Gmm++*. 2014. URL: http://download.gna.org/getfem/html/homepage/gmm/index.html (besucht am 10. 07. 2014).

[46] C. Sanderson: *Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*. Techn. Ber. Brisbane: NICTA, Sep. 2010. URL: http://arma.sourceforge.net/armadillo_nicta_2010.pdf (besucht am 30. 03. 2014).

[47] A. Schreiber: *Die Methode von Smolyak bei der multivariaten Interpolation*. Diss. Universität Göttingen, Fakultät für Mathematik und Informatik, 2000.

[48] H. R. Schwarz und N. Köckler: *Numerische Mathematik*. 7. Aufl. Wiesbaden: Vieweg+Teubner, 2009. DOI: 10.1007/978-3-8348-9282-9.

[49] S. A. Smolyak: *Quadrature and interpolation formulas for tensor products of certain classes of functions*. In: Soviet Mathematics - Doklady 4 (1963), S. 240–243. ISSN: 0197-6788.

[50] R. Storn und K. Price: *Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. In: Journal of Global Optimization 11 (4 1997), S. 341–359. ISSN: 0925-5001. DOI: 10.1023/A:1008202821328.

[51] M. Ulbrich und S. Ulbrich: *Nichtlineare Optimierung*. Mathematik Kompakt. Basel: Springer, 2012. DOI: 10.1007/978-3-0346-0654-7.

[52] United Nations, Department of Economic and Social Affairs, Population Division: *World Population Prospects. The 2012 Revision*. DVD Edition. 2013. URL: http://esa.un.org/wpp/Excel-Data/population.htm.

[53] Universitäten Erlangen-Nürnberg und Klagenfurt, Hrsg.: *CFS++ – Developer's Manual*. Entwicklerhandbuch. 8. Apr. 2014.

[54] J. Valentin: *Spline-Approximation unregelmäßig verteilter Daten*. Bachelorarbeit. Universität Stuttgart, Fachbereich Mathematik, 2012.

[55] H. A. van der Vorst: *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*. In: SIAM Journal on Scientific and Statistical Computing 13.2 (1992), S. 631–644. ISSN: 0196-5204. DOI: 10.1137/0913035.

[56] Wolfram Research: *Numerical Nonlinear Global Optimization*. Dokumentation. Mathematica 9. 2014. URL: http://reference.wolfram.com/mathematica/tutorial/ConstrainedOptimizationGlobalNumerical.html (besucht am 29. 05. 2014).

[57] X.-S. Yang: *Appendix A: Test Problems in Optimization*. In: Engineering Optimization. An Introduction with Metaheuristic Applications. Hoboken, New Jersey: John Wiley & Sons, 2010, S. 261–266. DOI: 10.1002/9780470640425.app1.

[58] C. Zenger: *Sparse Grids*. In: Parallel Algorithms for Partial Differential Equations: Proceedings of the Sixth GAMM-Seminar. Hrsg. von W. Hackbusch. Bd. 31. Notes on Numerical Fluid Mechanics. Braunschweig: Vieweg, 1991, S. 241–251.

[59] K. Zielinski, D. Peters und R. Laur: *Stopping Criteria for Single-Objective Optimization*. In: Proceedings of the 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems. Singapur, 2005.

Schriftliche Versicherung

Hiermit versichere ich,

1. dass ich die vorliegende Arbeit selbstständig verfasst habe,
2. dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
3. dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
4. dass ich diese Arbeit bisher weder teilweise noch vollständig veröffentlicht habe und
5. dass das elektronische Exemplar der Arbeit mit den anderen Exemplaren übereinstimmt.

Bietigheim-Bissingen, den 12. September 2014

Julian Valentin