Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Große lineare Gleichungssysteme für die B-Spline Interpolation auf Dünnen Gittern

Jens Drodofsky

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr. Dirk Pflüger
Betreuer/in:	M.Sc. Michael Rehme
Beginn am:	11. April 2017
Beendet am:	11. Oktober 2017
CR-Nummer:	G.1.1, G.1.3

Kurzfassung

Nutzt man in höheren Dimensionen mit einer regulären Diskretisierung benötigt man exponentiell steigende Laufzeit und Speicherplatz, dies wird durch den Begriff „Fluch der Dimensionalität“ bezeichnet. Dünne Gitter bieten eine Diskretisierungstechnik, welche erheblich weniger Gitterpunkte benötigen und trotzdem beinahe das gleiche Ergebnis erzielt. Somit können auch Lösungen berechnet werden, welche davor nicht möglich wären.

Dünngitter Interpolation auf dem Einheitsgebiet $[0, 1]^d$ wurde bereits intensiv untersucht, jedoch gibt es nur wenig Resultate für Gebiete, welche nicht durch die Tensorprodukt Struktur entstehen. Ein neuer Ansatz nutzt gewichtete B-Splines als Basisfunktion. Diese bieten die Möglichkeit Gebiet- und Randbedingungen anzugeben, so dass die Randbedingungen exakt eingehalten werden.

Ziel der Arbeit ist es das Lösen von linearen Gleichungssystem, welche während der Interpolation mit gewichteten B-Splines auf Dünnen Gittern auftreten, zu untersuchen. Dabei werden verschiedene Lösungsstrategien und Vorkonditionierungstechniken betrachten. Besondere Aufmerksamkeit wird dem BiCGSTAB-Verfahren gewidmet. Anschließend soll untersucht werden welche Methode die größten Erfolg zum Lösen beiträgt.

Vorwort

Als allererstes möchte ich mich bei Michael Rehme für die hervorragende Betreuung bedanken. Die letzten sechs Monate stand er mir durch sein Wissen und seine gute Erklärungen hilfreich zur Seite. Ich bin froh einen Betreuer wie Michael gehabt zu haben.

Als nächstes möchte ich bei meinen Kommilitonen bedanken, welche ich während des Bachelorstudiums kennengelernt habe. Vor allem möchte ich Nils Wenzler, Sandro Speth und Sarah Stieß erwähnen, welche stets zur Seite stehen, mich unterstützen und bei Problemen weiterhelfen. Ich möchte mir nicht vorstellen, wie es ohne sie gelaufen wäre.

Und zu guter Letzt danke ich meiner Familie für all das was sie mir ermöglichen. Ohne sie wäre all dies nicht möglich gewesen. Vielen Dank.

Auch wenn nicht immer alles so läuft wie geplant muss man mit dem Ergebnis leben. Wichtig ist nicht aufzugeben und immer weiter zu machen.

„Es geht immer immer weiter auch wenn du dein Ziel erreichst."

Inhaltsverzeichnis

1 Einleitung **9**

2 Dünne Gitter und B-Splines **11**
2.1 Eindimensionale Dünne Gitter 11
2.2 Mehrdimensionale Dünne Gitter 13
2.3 B-Splines 17
2.4 Gewichtete B-Splines 20

3 Interpolation **23**
3.1 Gleichungssystem 23
3.2 Besetztheit der Interpolationsmatrix 24
3.3 Löser 28

4 Vorkonditionierung **35**
4.1 Jacobi-Vorkonditionierung 36
4.2 Unvollständige LR-Zerlegung 36
4.3 Vorkonditionierung des BiCGSTAB-Verfahren 38

5 Vergleich **39**
5.1 Lösungsverfahren 39
5.2 Vorkonditionierung 41

6 Zusammenfassung und Ausblick **43**

Literaturverzeichnis **45**

1 Einleitung

Dünne Gitter werden unter anderem zur Quadratur, zur Lösung von partiellen Differentialgleichungen oder zur Interpolation verwendet. In dieser Arbeit werden wir Dünne Gitter für die Interpolation verwenden. Ohne Dünne Gitter wäre es nicht möglich höher dimensionale Probleme zu betrachten, da die Anzahl der Gitterpunkte bei einem vollen Gitter exponentiell mit der Dimension steigt. Ohne Beschränkung der Allgemeinheit betrachten wir nur Gebiete auf dem Einheitswürfel $[0, 1]^d$.

Auf diesen Dünnen Gittern verwenden wir die B-Splines als Basisfunktion. Diese sind stückweise definierte Polynome und haben schöne Eigenschaften. Jedoch gibt es bei den B-Splines das Problem mit dem Rand, da bei einem Dünnen Gitter dort keine Punkte mehr sind, welche der B-Spline benötigt. Deshalb wird der gewichtete B-Spline eingeführt um Gebiete besser zu betrachten.

Für die Dünngitter Interpolation entsteht dann ein lineares Gleichungssystem. Dieses muss dann entweder direkt oder iterativ gelöst werden. Es werden verschiedene Verfahren betrachtet, um solch ein Gleichungssystem zu lösen. Dabei liegt der Fokus auf dem iterativen BiCGSTAB-Verfahren. Im heutigen Zeitalter gibt es größtenteils nur noch Mehrkernprozessoren, damit wir diese auch vollständig für die Berechnung nutzen müssen die Lösungsverfahren alle parallelisiert sein. Dies wird im Verlaufe der Arbeit gemacht. Dazu wurde die Software Bibliothek `SG++`, welche von Dirk Pflüger in [Pfl10] entwickelt wurde, genutzt und erweitert.

Damit iterative Verfahren besser konvergieren gibt es die Möglichkeit der Vorkonditionierung. Diese beschleunigt die Konvergenz und somit auch die Laufzeit. Dabei werden zwei übliche Vorkonditionierungstechniken gezeigt und betrachtet.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Dünne Gitter und B-Splines: Hier werden werden die Grundlagen dieser Arbeit beschrieben.

Kapitel 3 – Interpolation: In diesem Kapitel wird auf die Interpolation und das Aufstellung der Interpolationsmatrix, sowie das Lösen des Gleichungssystems eingegangen.

Kapitel 4 – Vorkonditionierung beschreibt das einer äquivalenten Umformulierung des Gleichungssystems, so dass die neue Matrix bessere Eigenschaften hat. Vorkonditionierer werden vorallem bei iterativen Lösungsverfahren eingesetzt, um die Konvergenz zu beschleunigen.

Kapitel 5 – Vergleich der verschiedenen Verfahren, welche in den vorherigen Kapitel besprochen wurden.

Kapitel 6 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Dünne Gitter und B-Splines

Konventionelle Methoden, z. B. eine Diskretisierung mit einem vollen Gitter, zum Lösen von numerischen Problemen können in der Realität nur sehr eingeschränkt genutzt werden. Das größte Problem ist, dass die Anzahl von Punkten exponentiell mit der Anzahl von Dimensionen steigt. Dies macht es beinahe unmöglich fünfdimensionale Probleme zu betrachten. Dies wird in der Literatur mit dem „Fluch der Dimensionalität“ bezeichnet. Dünne Gitter sind eine Diskretisierungstechnik um verschiedene Arten von numerischen Problemen zu lösen. Diese haben den Vorteil den „Fluch der Dimensionalität“ zu überwinden und somit auch höher dimensionale Probleme zu betrachten. Das Resultat ist, obwohl viel weniger Gitterpunkte benötigt werden, ähnlich zu dem des vollen Gitters.

In Abschnitt 2.1 betrachten wir das Dünne Gitter zunächst für den eindimensionalen Fall. Dabei verwenden wir fürs erste die Standard-Hütchenfunktion als Basisfunktion. In Abschnitt 2.2 verallgemeinern wir das eindimensionale Gitter, um mehrere Dimensionen betrachten zu können. Für die Dünnen Gitter verwenden wir größtenteils die Notation aus [Gar13; Pfl10; Val14]. Dann betrachten wir in Abschnitt 2.3 die sogenannten B-Splines, welche eine verallgemeinern der Standard-Hütchenfunktion darstellen. Da diese mit Randbedingungen eher ungeeignet sind, betrachten wir in Abschnitt 2.4 die gewichteten B-Splines. Bei den B-Splines werden wir uns an die Notation aus [HH13; Höl03] halten.

2.1 Eindimensionale Dünne Gitter

Definition 2.1.1 (Eindimensionales Dünnes Gitter)
Sei $l \in \mathbb{N}$. Dann ist

$$G_l := \left\{ x_{l,j} := j \cdot h_l \mid j = 0, \ldots, 2^l \right\}, \tag{2.1}$$

das eindimensionale Dünne Gitter, wobei l das Diskretisierungslevel und $h_l := 2^{-l}$ die Schrittweite beschreibt.

Beim eindimensionalen Dünnen Gitter wird das Intervall $[0, 1]$ durch die Punkte $x_{l,j}$ unterteilt, welche alle in einem Abstand von h_l von einander entfernt sind. Betrachtet man das erste Level, dann hat das Dünne Gitter G_1 die Punkte $x_{1,j} \in \{0, \frac{1}{2}, 1\}$. Bei G_2 gibt es die Punkte $x_{2,j} \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$. Dies lässt sich daraus herleiten, dass die Schrittweite von G_1 halbiert wurde. Es wird also genau in der Mitte zwischen zwei bisherigen Punkten ein weiterer Punkt hinzugefügt. Das Dünne Gitter G_{l+1} besteht also aus allen Punkten von G_l plus diejenigen

die durch die Halbierung der Schrittweite hinzukommen. Somit erhält man eine hierarchische Anordnung und man kann, bezüglich der Mengeninklusion, eine aufsteigende Kette $G_1 \subset G_2 \subset G_3 \subset \cdots$ von Gittern bilden, wie in [Val14].

Nun wollen wir den zu G_l dazugehörigen Vektorraum V_l definieren. Dafür benötigen wir aber noch eine Basisfunktion ϕ. Zunächst nehmen wir dazu die Standard-Hütchenfunktion.

Definition 2.1.2 (Standard-Hütchenfunktion)
Die Standard-Hütchenfunktion $\phi : \mathbb{R} \to \mathbb{R}$ ist definiert wie in [BG04]:

$$\phi(x) := \begin{cases} 1 - |x| & \textit{falls } x \in [-1, 1], \\ 0 & \textit{sonst.} \end{cases} \tag{2.2}$$

Diese Definition von stückweise linearen Basisfunktionen kann genutzt werden um ein beliebige Funktion $\phi_{l,j}$, mit Stützstellen $[x_{l,j} - h_l, x_{l,j} + h_l] \cap [0, 1]$, zu generieren. Schoenberg [Höl03] führte durch Skalierung und Umwandlung folgende Variante ein:

$$\phi_{l,j}(x) := \phi\left(\frac{x - j \cdot h_l}{h_l}\right). \tag{2.3}$$

Die Funktion $\phi_{l,j}(x)$ hat ihr Maximum genau an dem Punkt $x_{l,j}$ und ist Null an allen anderen Punkten.

Definition 2.1.3 (Raum stückweise linearer stetiger Funktionen)
Sei $l \in \mathbb{N}$ das Level und $\phi_{l,j}$ wie in Gleichung (2.3) *definiert, dann ist*

$$V_l := \textit{span}\left\{\phi_{l,j} \mid j = 0, \ldots, 2^l\right\} \tag{2.4}$$

der Raum aller stückweise linearer stetiger Funktionen auf dem Intervall $[0, 1]$ mit Gitterweite h_l.

Da das Gitter sich mit steigendem Level mehr Punkte hierarchisch hinzu kommen, wollen wir dies auch für die Funktionen machen. Dafür definiert man den hierarchisch zunehmenden Unterraum W_l wie in [BG04; Gar13; Pfl10].

Definition 2.1.4 (Hierarchische Indexmenge)
Die hierarchische Indexmenge für das Level l ist

$$I_l := \left\{j \in \mathbb{N} \mid 1 \leq j \leq 2^l - 1, j \textit{ ungerade}\right\}. \tag{2.5}$$

Definition 2.1.5 (Hierarchischer Unterraum)
Sei $l \in \mathbb{N}$, dann ist

$$W_l := \textit{span}\left\{\phi_{l,j} \mid j \in I_l\right\} \tag{2.6}$$

der hierarchische Unterraum vom Level l.

Den Vektorraum V_q, mit einer Schrittweite h_q, kann man aus der direkten Summe aller hierarchischen Unterräume erhalten

$$V_q = \bigoplus_{l \leq q} W_l. \tag{2.7}$$

Jede Funktion $f \in V_q$ kann dargestellt werden als:

$$f(x) = \sum_{l \leq q} \sum_{j \in I_l} \alpha_{l,j} \cdot \phi_{l,j}(x), \tag{2.8}$$

wobei $\alpha_{l,j}$ die Koeffizienten, auch hierarchische Überschüsse genannt sind und genauso indiziert werden wie die dazugehörige Basisfunktion.

2.2 Mehrdimensionale Dünne Gitter

Nun wollen wir das eindimensionale Gitter verallgemeinern damit wir zum d-dimensionalen Dünnen Gitter kommen. Als erstes erweitern wir das Level auf einen Multi-Index $l = (l_1, \ldots, l_d)$. Somit muss auch die Schrittweite auf das erweiterte Level angepasst werden

$$h_l := (h_{l_1}, \ldots, h_{l_d}) = 2^{-l} := (2^{-l_1}, \ldots, 2^{-l_d}).$$

Dabei gilt weiterhin, dass die Punkte in einer Richtung k uniform sind.

Somit können wir das d-dimensionale volle Gitter definieren, welches sich aus dem kartesischen Produkt der Dimensionen zusammensetzt.

Definition 2.2.1 (d-dimensionale volle Gitter)
Das d-dimensionale volle Gitter vom Level $l \in \mathbb{N}^d$ und Schrittweite h_l ist

$$G_l := \bigtimes_{k=1}^{d} G_{l_k} = \{x_{l,j} := (x_{l_1,j_1}, \ldots, x_{l_d,j_d}) \mid \forall_{1 \leq k \leq d}\, x_{l_k,j_k} \in G_{l_k}\}. \tag{2.9}$$

Analog zum Eindimensionalen definieren wir nun die Hütchenfunktion. Diese muss nun für den d-dimensionalen Fall verallgemeinert werden. Dabei kann die Hütchenfunktion von Gleichung (2.3) nicht direkt verwendet werden. Deshalb nutzen wir den gebräuchlichen Tensorprodukt-Ansatz.

Definition 2.2.2 (d-dimensionale Hütchenfunktion)
Die d-dimensionale Hütchenfunktion vom Level $l \in \mathbb{N}^d$ und Index $j \in \mathbb{N}^d$ mit $j_k \in \{0, \ldots, 2^{l_k}\}$ ist das Produkt aus den jeweiligen Dimensionen:

$$\phi_{l,j}(x) := \prod_{k=1}^{d} \phi_{l_k,j_k}(x_k). \tag{2.10}$$

Analog zum Eindimensionalen erhält man den Vektorraum V_l von stückweisen d-linearen Funktionen auf den Punkten $x_{l,j}$.

Definition 2.2.3 (Raum stückweise *d*-lineare stetige Funktionen)
Sei $l \in \mathbb{N}$ das Level, dann ist

$$V_l := \textit{span}\{\phi_{l,j} \mid j_k = 0, \ldots, 2^{l_k}, k = 1, \ldots, d\} = \textit{span}\{\phi_{l,j} \mid 0 \leq j \leq 2^l\} \tag{2.11}$$

der Raum aller stückweise d-linearen, stetigen Funktionen auf dem Intervall $[0,1]^d$ mit Gitterweite h_l.

Definition 2.2.4 (Mehrdimensionale Indexmenge)
Die mehrdimensionale Indexmenge von Level $l = (l_1, \ldots, l_d)$ ist

$$J_l := \left\{ j \in \mathbb{N}^d \mid j_k \in I_{l_k}, 1 \leq k \leq d \right\}. \tag{2.12}$$

Definition 2.2.5 (Mehrdimensionaler hierarchischer Unterraum)
Sei $l \in \mathbb{N}$, dann ist

$$W_l := \textit{span}\left\{\phi_{l,j}(x) \mid j \in J_l\right\} \tag{2.13}$$

der hierarchische Unterraum mit Level l.

Ab sofort schauen wir ausschließlich Gitter mit gleichem Level in jeder Dimension an. Es gilt also $q := (n, \ldots, n)$ mit $n \in \mathbb{N}$. In Abbildung 2.1 sieht man, für den zweidimensionalen Fall, die hierarchischen Unterräume W_l mit den zugehörigen Basisfunktionen von V_4. Für das volle Gitter V_4 werden sowohl die schwarzen als auch die ausgegrauten Unterräume benötigt.

Analog zum Eindimensionalen kann der Raum V_q mit Schrittweite h_q dargestellt werden als,

$$V_q = \bigoplus_{|l|_\infty \leq q} W_l. \tag{2.14}$$

Jede Funktion $f \in V_q$ kann dargestellt werden durch

$$f(x) = \sum_{|l|_\infty \leq q} \sum_{j \in J_l} \alpha_{l,j} \cdot \phi_{l,j}(x). \tag{2.15}$$

Damit wir nun zum Dünnen Gitter kommen, nehmen wir die hierarchischen Unterräume, welche am meisten zur gesamten Lösung beitragen. Die Qualität eines hierarchischen Unterraums für die Lösung kann in [BG04] betrachtet werden und wollen uns damit nicht weiter beschäftigen.

Definition 2.2.6 (Dünnes Gitter)
Sei $l \in \mathbb{N}^d$ und q das maximale Level, dann ist

$$V_q^s := \bigoplus_{|l|_1 \leq q} W_l \tag{2.16}$$

das Dünne Gitter.

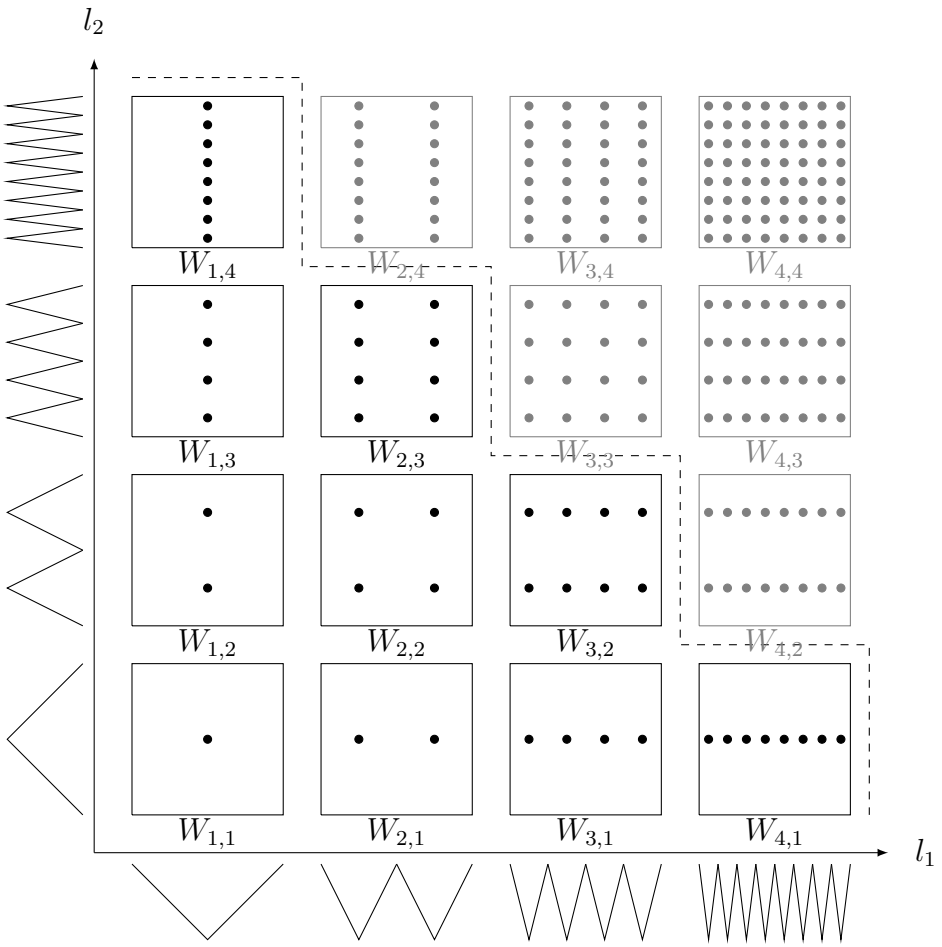

Abbildung 2.1: Hierarchische Unterräume W_l von V_4 für $d = 2$, dabei entspricht ein Quadrat einem Unterraum mit den dazugehörigen Punkten. In schwarz sind die hierarchischen Unterräume von $V^s_{4,0}$ und in grau die restlichen Unterräume von V_4, nach [BG04].

Eine Funktion $f \in V^s_q$ kann nun analog zu Gleichung (2.15) dargestellt werden,

$$f(x) = \sum_{|l|_1 \leq q} \sum_{j \in J_l} \alpha_{l,j} \cdot \phi_{l,j}(x). \tag{2.17}$$

Das Problem an der Definition 2.2.6 ist, dass $\mathcal{O}(d \cdot 2^{d+n})$ Punkte auf dem Rand liegen. Da wir uns nicht mit all den Randpunkten beschäftigen wollen, gibt es wie in [Pfl10] die Möglichkeit nur noch die inneren Gitterpunkte zu betrachten.

Definition 2.2.7 (Dünnes Gitter ohne Rand)
Sei $l \in \mathbb{N}^d$ und $q \in \mathbb{N}^d$ das maximale Level, dann ist

$$V^s_{q,0} := \bigoplus_{|l|_1 \leq q+d-1} W_l \tag{2.18}$$

das Dünne Gitter ohne Rand.

Diese Definition ist besonders nützlich für den Fall, dass auf dem Rand keine Freiheitsgrade sind, zum Beispiel bei Dirichlet-Randbedinungen. Somit finden keine Auswertungen für die Punkte auf dem Rand statt.

Analog zum Dünnen Gitter V_q^s kann eine Funktion $f \in V_{q,0}^s$ kann dargestellt werden durch

$$f(x) = \sum_{|l|_1 \leq q+d-1} \sum_{j \in J_l} \alpha_{l,j} \cdot \phi_{l,j}(x). \tag{2.19}$$

Abbildung 2.2 zeigt das Dünnes Gitter ohne Rand, welches man von Abbildung 2.1 aus den Unterräumen unterhalb der gestrichelten Linie erhält. Dabei kann man auch die vorherigen Level betrachten. Das Dünne Gitter von Level $q = 1$ besteht nur aus dem roten Punkt. Das Dünne Gitter von Level $q = 2$ setzt sich aus dem Dünnen Gitter von Level 1, dem roten Punkt, und den grünen Punkten zusammen. Das Dünne Gitter von Level $q = 3$ erhält man, indem man bei dem Dünnen Gitter von Level 2 die gelben Punkte hinzufügt. Das Dünne Gitter von Level $q = 4$ besteht aus allen Punkten.

Da später die Anzahl an Punkten eine wichtige Rolle für die Größe des Gleichungssystem spielt, betrachten wir nun folgendes Lemma.

Lemma 2.2.1
Die Anzahl der Punkte eines Dünnen Gitters ohne Rand mit Level q beträgt

$$n = \left|V_{q,0}^s\right| = \sum_{i=0}^{q-1} 2^i \cdot \binom{d-1+i}{d-1} \tag{2.20}$$

Beweis 2.2.1
Für den Beweis wird die Eigenschaft der hierarchischen Unterräume und die Gleichung (2.18) *genutzt, ausführlich in [BG04, S. 27].*

Schaut man sich das d-dimensionale volle Gitter G_q an, dann bemerkt man, dass es $\mathcal{O}(2^{qd})$ Punkte gibt. Somit ist die Anzahl der Funktionsauswertungen auch in der Komplexität $\mathcal{O}(h_q^{-d}) = \mathcal{O}(2^{qd})$. Wir haben also eine exponentielle Abhängigkeit von der Dimension. Dies wir in der Literatur mit dem „Fluch der Dimensionalität“ bezeichnet. Betrachtet man nun das Dünne Gitter $V_{q,0}^s$, dann wissen wir durch Lemma 2.2.1, dass es $\mathcal{O}(2^q \cdot q^{d-1}) = \mathcal{O}(h_q^{-1} \cdot |\log_2 h_q|^{d-1})$ Punkte gibt. Dies sind signifikant weniger Punkte, da hier die Dimension nur noch exponentiell zur logarithmischen Schrittweite ist. Die große Frage ist nun in wie weit sich die Fehler der Interpolation unterscheidet. Betrachtet man dazu die L_2 Norm und die Maximumsnorm, dann beträgt der Interpolationsfehler auf dem vollen Gitter $\mathcal{O}(2^{-2q})$ und für das Dünne Gitter lediglich geringfügig schlechter mit $\mathcal{O}(2^{-2q} \cdot q^{d-1})$. Einen Beweis dafür findet man in [BG04]. Dies stellt eine enorme Verbesserung dar, da erheblich weniger Punkte ausgewertet werden müssen und der Fehler beinahe derselbe ist. Dies vermindert den „Fluch der Dimensionalität“ und somit können nun auch höherdimensionale Probleme betrachtet werden. Jedoch ist zu beachten, dass der Fluch nicht vollständig verschwindet.

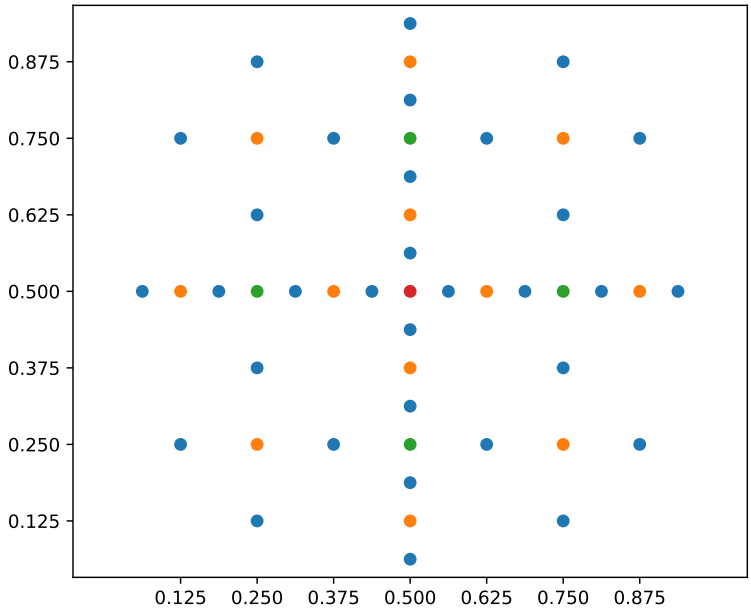

Abbildung 2.2: Zweidimensionales Dünnes Gitter mit Level 4

2.3 B-Splines

Ein Spline ist ein stückweise definiertes Polynom. Wir definieren den Splineraum durch B-Splines, genannt Basissplines, welche bemerkenswerte numerische Eigenschaften haben. Ein B-Spline vom Grad g hat $g+2$ Stützstellen und ist $(g-1)$-mal stetig differenzierbar. Er ist positiv in dem Intervall $(0, g+1)$ und verschwindet außerhalb. Außerdem ist er symmetrisch und monoton in den Intervallen $[0, (g+1)/2]$ und $[(n+1)/2, n+1]$.

Wir definieren den uniformen B-Spline wie in [HH13; Höl03].

Definition 2.3.1 (Uniformer B-Spline)
Der uniforme B-Spline $b^g : \mathbb{R} \to \mathbb{R}$ vom Grad g ist durch folgende Rekursionsformel definiert:

$$b^0(x) = \begin{cases} 1 & \textit{falls } 0 \leq x < 1 \\ 0 & \textit{sonst} \end{cases} \tag{2.21}$$

$$b^g(x) = \frac{x}{g} \cdot b^{g-1}(x) + \frac{g+1-x}{g} \cdot b^{g-1}(x-1) \tag{2.22}$$

In Abbildung 2.3 sind die uniformen B-Splines bis Grad $g = 5$ zu sehen. Dabei ist anzumerken, dass die bekannte Standard-Hütchenfunktion einem B-Spline von Grad $g = 1$ entspricht.

Damit die B-Splines für die Dünnen Gitter verwendet werden können, benötigt man eine hierarchische Version $\phi_{l,j}$. Dies erhält man, wie in [Pfl10] genutzt, wieder durch Skalierung und Umwandlung nach Schoenberg.

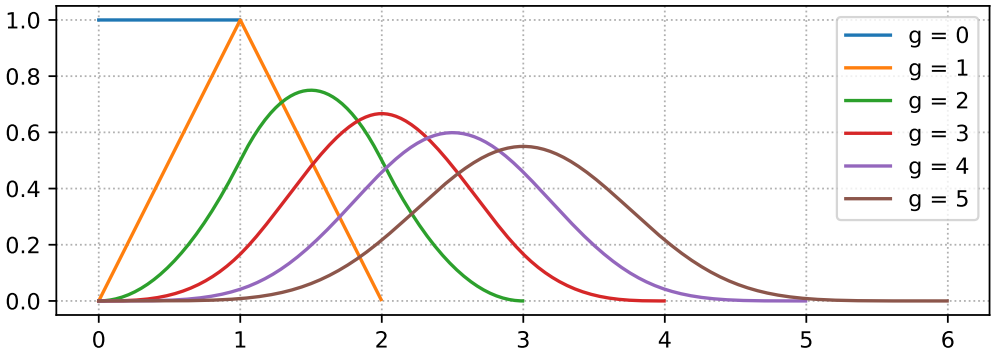

Abbildung 2.3: Uniforme B-Splines b^g von Grad $g = 0, \ldots, 5$

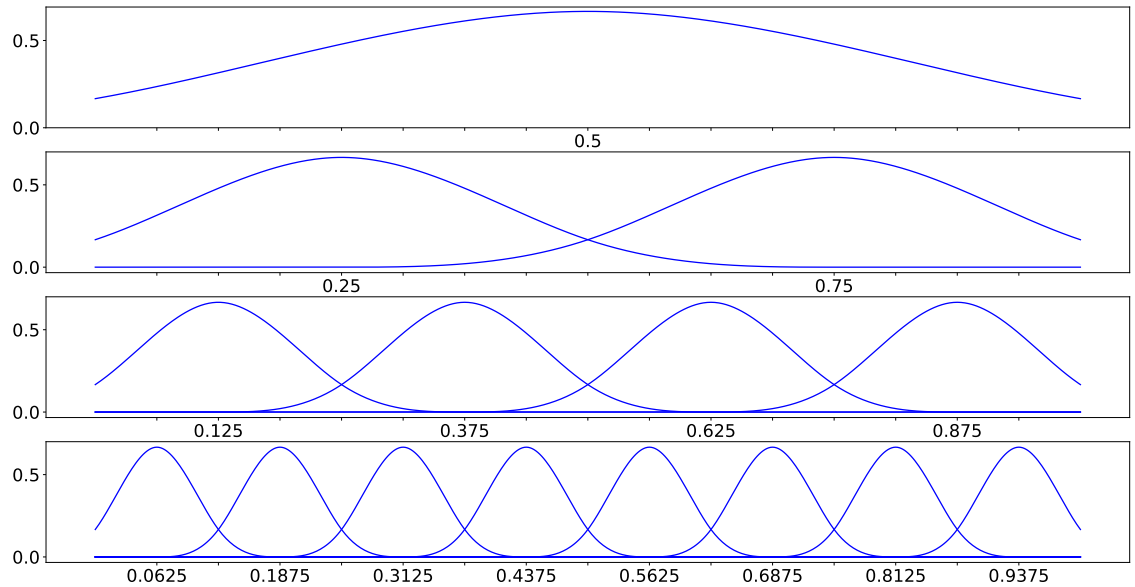

Abbildung 2.4: Hierarchische B-Splines b^3 von Level 1 (ganz oben) bis 4 (ganz unten)

Definition 2.3.2 (Hierarchischer, uniformer B-Spline)
Der hierarchische, uniforme B-Spline vom Grad g mit Level l und Index j ist definiert durch

$$\phi_{l,j}^g := b^g \left(\frac{x}{h_l} + \frac{g+1}{2} - j \right). \tag{2.23}$$

Wie bereits in [Val14] erwähnt wurde, liegt das Maximum von $\phi_{l,j}^g$ genau an dem Gitterpunkt $x_{l,j}$, wenn g ungerade ist. Bei einem geraden Grad g liegt das Maximum genau zwischen zwei Gitterpunkten, was unvorteilhaft sein kann. Deshalb beschränken wir uns auf den ersten Fall, dass g ungerade ist. Außerdem, falls nicht explizit erwähnt, wird immer von einem B-Spline von Grad $g = 3$ ausgegangen.

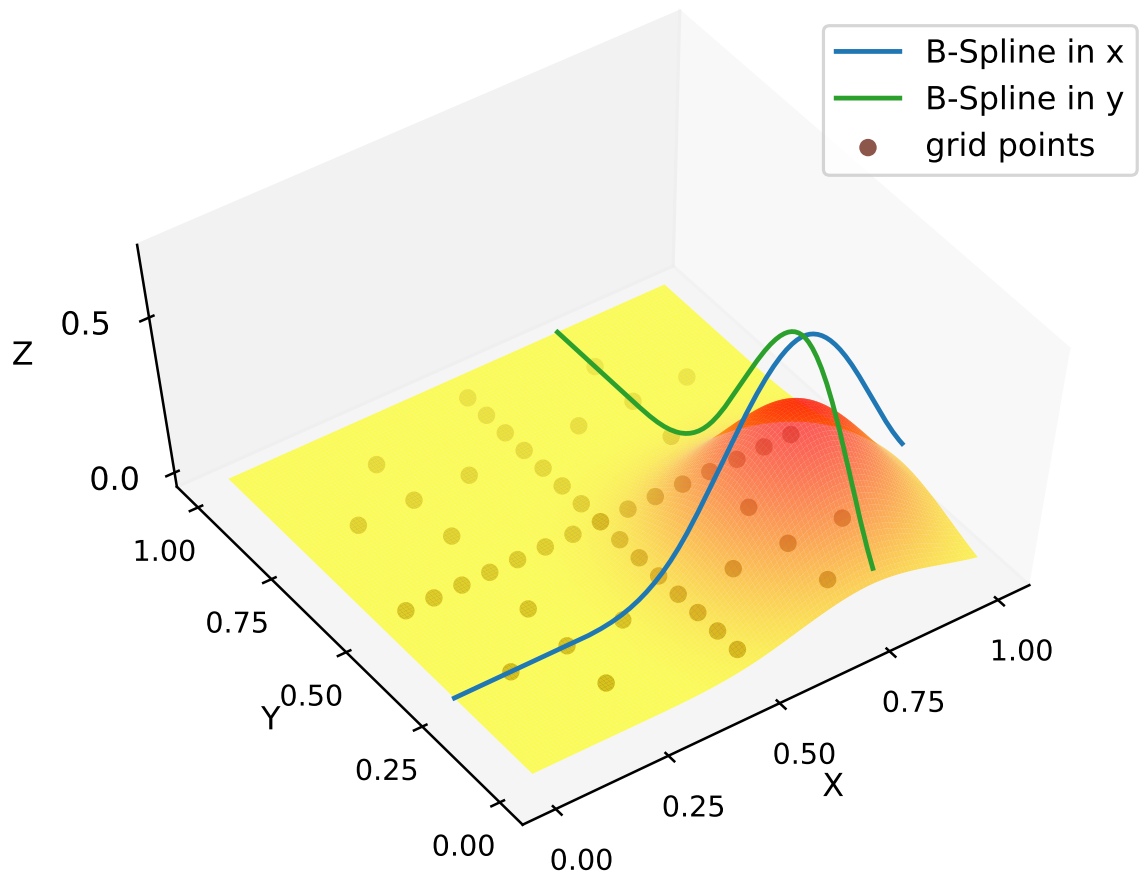

Abbildung 2.5: Bikubischer B-Spline auf einem Dünnen Gitter mit Level 4

Wir können nun die bisherigen Definitionen, welche für die Hütchenfunktionen gemacht wurden durch die B-Splines verallgemeinern. Dabei sei der hierarchische B-Spline-Raum analog zu Definition 2.1.5 definiert durch

$$W_l^g := \operatorname{span}\left\{\phi_{l,j}^g \mid j \in I_l\right\}. \tag{2.24}$$

Analog erhalten wir den Vektorraum V_q mit einer Schrittweite h_q aus der direkten Summe aller hierarchischen Unterräume

$$V_q = \bigoplus_{l \leq q} W_l^g. \tag{2.25}$$

Abbildung 2.4 zeigt die hierarchischen B-Splines bis Level 4.

Damit wir die B-Splines auch in höheren Dimensionen auf den Dünnen Gitter nutzen können, nutzen wir wieder den Tensorprodukt-Ansatz.

Definition 2.3.3 (d-dimensionale B-Splines)
Der d-dimensionale B-Spline mit Level $l \in \mathbb{N}$ und Index $j\mathbb{N}$ mit $j_k \in \{0, \ldots, 2^{l_k}\}$ ist das Produkt aus den jeweiligen Dimensionen:

$$\phi_{l,j}^g(x) := \prod_{k=1}^{d} \phi_{l_k,j_k}^g(x_k). \tag{2.26}$$

Ein großes Problem bei den B-Splines ist, dass es B-Splines am Rand gibt, welche nicht genug Gitterpunkte zur Verfügung haben. In [Pfl10] wurden deshalb sogenannte modifizierte B-Splines eingefügt.

2.4 Gewichtete B-Splines

Auf den ersten Blick sind B-Splines aufgrund der Tensorprodukt Struktur nicht wirklich geeignet um eine Funktion f auf einem Gebiet D mit Randbedingungen zu approximieren. Dies kann verbessert werden, indem man die B-Splines mit einer Gewichtsfunktion w multipliziert.

Definition 2.4.1 (Gewichtsfunktion)
Eine Gewichtsfunktion w der Ordnung $\gamma \in \mathbb{N}_0$ ist stetig auf $\overline{D}$ und erfüllt

$$w(x) \asymp dist(x, \Gamma)^y, x \in D,$$

für eine Teilmenge Γ von ∂D.

Die Gewichtsfunktion ist im Gebiet positiv, null auf dem Rand ∂D und negativ außerhalb. Das Gebiet kann somit implizit in der Gewichtsfunktion beschrieben werden

$$D = \{x \in \mathbb{R}^d \mid w(x) > 0\} \subseteq [0, 1]^d. \tag{2.27}$$

Definition 2.4.2 (Uniformer gewichteter B-Spline)
Sei w eine Gewichtsfunktion. Dann ist

$$B^g = wb^g \tag{2.28}$$

der gewichtete uniforme B-Spline mit Schrittweite h.

Um beliebige Gewichtsfunktionen zu konstruieren kann man die sogenannte R-Funktion Methode [HH13; Höl03], benannt nach Rvachev, genutzt werden. Diese Methode kann mehrere Gewichtsfunktionen, und somit auch Gebiete, auf boolesche Operationen zurückführen.

Definition 2.4.3 (R-Funktionen)
Sei das Gebiet D_v beschrieben mit der impliziten Form einer Gewichtsfunktion w_v. Dann beschreiben die Standard R-Funktionen

$$r_c(w_1) = -w_1 \tag{2.29}$$

$$r_\setminus(w_1, w_2) = w_1 - w_2 - \sqrt{w_1^2 + w_2^2} \tag{2.30}$$

$$r_\cup(w_1, w_2) = w_1 + w_2 + \sqrt{w_1^2 + w_2^2} \tag{2.31}$$

$$r_\cap(w_1, w_2) = w_1 + w_2 - \sqrt{w_1^2 + w_2^2} \tag{2.32}$$

die Gebiete $D_1^c, D_1 \setminus D_2, D_1 \cup D_2, D_1 \cap D_2$

Somit kann man beliebige Gebiete mit den üblichen Mengenoperationen verknüpfen.

3 Interpolation

In diesem Kapitel wollen wir uns mit der Interpolation und dem dazugehörigen Gleichungssystem beschäftigen. Bei der Interpolation ist es das Ziel eine Funktion f so gut wie möglich zu approximieren, d. h. $f(x) \approx u(x)$, indem

$$f(x_j) \approx u(x_j) := \sum_i \alpha_i \phi_i(x_j) \tag{3.1}$$

für eine Menge von Punkten $\{x_j\}_{1 \leq j \leq n}$ verlangt wird.

Im ersten Abschnitt wird kurz erwähnt, wie das Gleichungssystem zustande kommt. In Abschnitt 3.2 schauen an, wie viele Nicht-Null Einträge in der Interpolationmatrix existieren. Danach betrachten wir in Abschnitt 3.3 Verfahren, um das Gleichungssystem zu lösen.

3.1 Gleichungssystem

Wir formulieren Gleichung (3.1) als Gleichungssystem, d. h.

$$A\alpha = b \tag{3.2}$$

wobei die Matrix A definiert wird durch die Auswertungen der B-Splines an den jeweiligen Punkten

$$A = (\phi_j(x_i))_{i,j} = \begin{pmatrix} \phi_1(x_1) & \cdots & \phi_m(x_1) \\ \vdots & & \vdots \\ \phi_1(x_n) & \cdots & \phi_m(x_n) \end{pmatrix} \tag{3.3}$$

und b aus den Funktionsauswertungen an den Punkten x_i

$$b = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}.$$

Dabei wird das Gleichungssystem nach α aufgelöst und erhalten die Koeffizienten für die jeweiligen Basisfunktionen. Somit erhalten wir den eindeutigen Interpolaten u.

3.2 Besetztheit der Interpolationsmatrix

In diesem Abschnitt wollen wir kurz auf die Interpolationsmatrix für die B-Splines von Grad 3 eingehen. Dabei soll die Anzahl an Nicht-Null Einträgen der Matrix untersucht werden. Da die B-Splines mit zunehmenden Level weniger Trägerpunkte nutzen, ist anzunehmen, dass mit jedem neuen Level eine große Anzahl von Einträge, welche Null sind, hinzukommen.

Satz 3.2.1 (Eindimensionale Besetztheit der Interpolationsmatrix)
Sei l das Level des eindimensionalen Dünnen Gitters und n die Anzahl der Punkte. Dann gibt es im Eindimensionalen mit den B-Splines vom Grad $g = 3$

$$NNZ = n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot \left(2^{k-1} - 2 \right) \right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor \right) \tag{3.4}$$

Einträge ungleich Null in der Interpolationsmatrix.

Beweis 3.2.1
Für die Anzahl aller Punkte des Dünnen Gitters mit Level l nutzen wir Lemma 2.2.1. Da wir hier den eindimensionalen Fall betrachten gilt $d = 1$.

$$n = \sum_{i=0}^{l-1} 2^i \cdot \binom{d+i-1}{d-1} = \sum_{i=0}^{l-1} 2^i \cdot \binom{i}{0} = 2^l - 1$$

Der Beweis wird durch vollständige Induktion über das Level l durchgeführt.

Induktionsanfang:

$l = 1$:

$$NNZ = 1 + 0 + 0 = 1$$

Das Gitter hat genau einen Punkt und einen B-Spline, welcher sein Maximum an diesem Punkt hat. Somit gibt es einen Wert, welcher nicht Null ist.

$l = 2$:

$$NNZ = 3 + 0 + 4 = 7$$

Induktionsschritt:

$l \rightsquigarrow l+1$

$$
\begin{aligned}
NNZ &= n + \left(\sum_{k=2}^{l+1} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l+1} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) \\
&= n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) + \left(\left\lfloor \frac{n}{2^{l-1}} \right\rfloor \cdot (2^l-2)\right) + \left(2 \cdot \left\lfloor 3 \cdot \frac{n}{2^{l+1}} \right\rfloor\right) \\
&= n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) + \left(\left\lfloor \frac{2^{l+1}-1}{2^{l-1}} \right\rfloor \cdot (2^l-2)\right) + \\
&\quad + \left(2 \cdot \left\lfloor 3 \cdot \frac{2^{l+1}-1}{2^{l+1}} \right\rfloor\right) \\
&= n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) + \left(\left\lfloor \frac{2^{l+1}}{2^{l-1}} - \frac{1}{2^{l-1}} \right\rfloor \cdot (2^l-2)\right) + \\
&\quad + \left(2 \cdot \left\lfloor \frac{3 \cdot 2^{l+1}}{2^{l+1}} - \frac{3}{2^{l+1}} \right\rfloor\right) \\
&= n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) + \left(\left\lfloor 2^2 - \frac{1}{2^{l-1}} \right\rfloor \cdot (2^l-2)\right) + \\
&\quad + \left(2 \cdot \left\lfloor 3 - \frac{3}{2^{l+1}} \right\rfloor\right) \\
&= n + \left(\sum_{k=2}^{l} \left\lfloor \frac{n}{2^{k-2}} \right\rfloor \cdot (2^{k-1}-2)\right) + 2 \cdot \left(\sum_{k=2}^{l} \left\lfloor 3 \cdot \frac{n}{2^k} \right\rfloor\right) + \left(3 \cdot (2^l-2)\right) + (2 \cdot 2)
\end{aligned}
$$

Die B-Splines des vorherigen Levels bleiben weiterhin bestehen und erhalten die neuen Stützpunkte. Das einzige, was neu dazugekommen ist sind die B-Splines auf dem Level $l+1$. Durch die Verfeinerung sind 2^l neue Punkte entstanden und somit gibt es nun auch die gleiche Anzahl an B-Splines, davon liegen zwei auf dem Rand. Ein B-Spline auf dem höchsten Level hat drei Gitterpunkte, auf dem Rand nur zwei. Somit kommen $\left(3 \cdot (2^l-2)\right) + (2 \cdot 2)$ Nicht-Null Einträge auf dem Level $l+1$ hinzu.

□

Durch diese Formel kann man nun sehr leicht ausrechnen, wie viele Nicht-Null Einträge sich in der Matrix befinden. Diese gilt leider nur für den eindimensionalen Fall. Für höhere Dimensionen ist es zunehmend schwerer eine exakte Formel herauszufinden. Dies liegt vor allem an der Randproblematik, da es nun nicht nur in eine Richtung einen Rand gibt, sondern in mehrere. Deshalb wurde lediglich eine Abschätzung aufgestellt.

Bemerkung (Allgemeine Abschätzung der Besetztheit)
Sei q das maximale Level, d die Dimension, n die Anzahl der Punkte des Dünnes Gitters und $l \in \mathbb{N}^d$ das jeweilige Level des hierarchischen Unterraums, dann lässt sich die Anzahl der NNZ durch folgende Formel abschätzen:

$$NNZ \leq \sum_{|l|_1 \leq q+d-1} n \cdot \left(\prod_{1 \leq k \leq d} 2^{k-1} \cdot f(l_k) \right) \tag{3.5}$$

$$f(l_k) = \begin{cases} 1 & \textit{falls } l_k \in \{1, 2\}, \\ 3 \cdot 2^{-l_k+1} & \textit{sonst.} \end{cases} \tag{3.6}$$

Anmerkung
In der Formel gehen wir über alle hierarchischen Unterräume. Dabei wird geschaut, wie viele Punkte in einem Unterraum liegen. Dies wird durch folgende Formel beschrieben:

$$|W_l| = \prod_{1 \leq k \leq d} 2^{k-1}. \tag{3.7}$$

In [BG04] findet man die Formel in folgender Form:

$$|W_l| = 2^{|l-1|_1}$$

Diese lässt sich offensichtlich in 3.7 umwandeln.

Jeder Punkt hat einen Tensorprodukt B-Spline, wobei Randpunkte als normale Punkte behandelt werden. Um zu wissen wie viele Punkte von diesem B-Spline genutzt werden nutzen wir die Funktion f aus Gleichung (3.6)*. Diese ist ähnlich zum eindimensionalen Fall konzipiert und soll angeben, wie viel der Gitterpunkte genutzt werden. f gibt jedoch lediglich eine großzügige Abschätzung an. Dies wurde mit mehreren Dimensionen und Level überprüft und erfüllte die Gleichung* (3.5)*. Ein kleiner Ausschnitt davon ist Abbildung 3.2 zu sehen.*

Definition 3.2.1 (Matrixbesetztheit)
Die Matrixbesetztheit einer $m \times n$ Matrix beschreibt das Verhältnis von Nicht-Null Einträgen (NNZ) zu allen Einträgen und berechnet sich folgendermaßen

$$\textit{Matrixbesetztheit} := \frac{NNZ}{m \cdot n}. \tag{3.8}$$

Dieses Verhältnis kann in Abbildung 3.1 betrachtet werden. Dabei wurde für $d = 1$ die Formel, welche natürlich äquivalent zur experimentellen ist, verwendet und die Dimensionen $d = 2, 3, 5$ und 8 wurden durch `SG++` berechnet. In Abbildung 3.2 sieht man einen Vergleich der allgemeinen Abschätzungsformel. Dabei wurden die gestrichelten Linien mittels der Formel 3.5 aus der Bemerkung berechnet. Die durchgezogenen Linien entsprechen den tatsächlichen berechneten Werten. Es ist anzumerken, dass auch für $d = 10$ die Bibliothek `SG++` genutzt wurde, aber nur bis Level 5 berechnet wurde, aufgrund der zunehmenden Anzahl an Gitterpunkten und Matrixgröße.

Als Testbeispiel wurde die d-dimensionale Einheitsparabel verwendet der d-dimensionale Einheitskreis als Gewichtsfunktion.

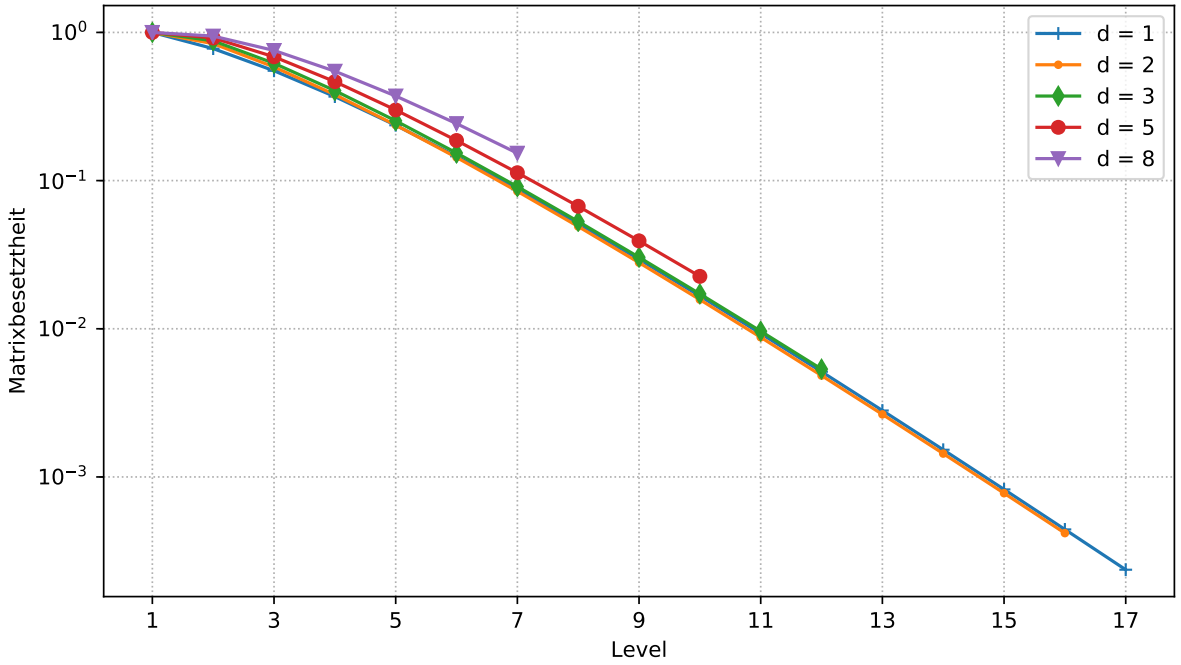

Abbildung 3.1: Besetztheit der Interpolationsmatrix

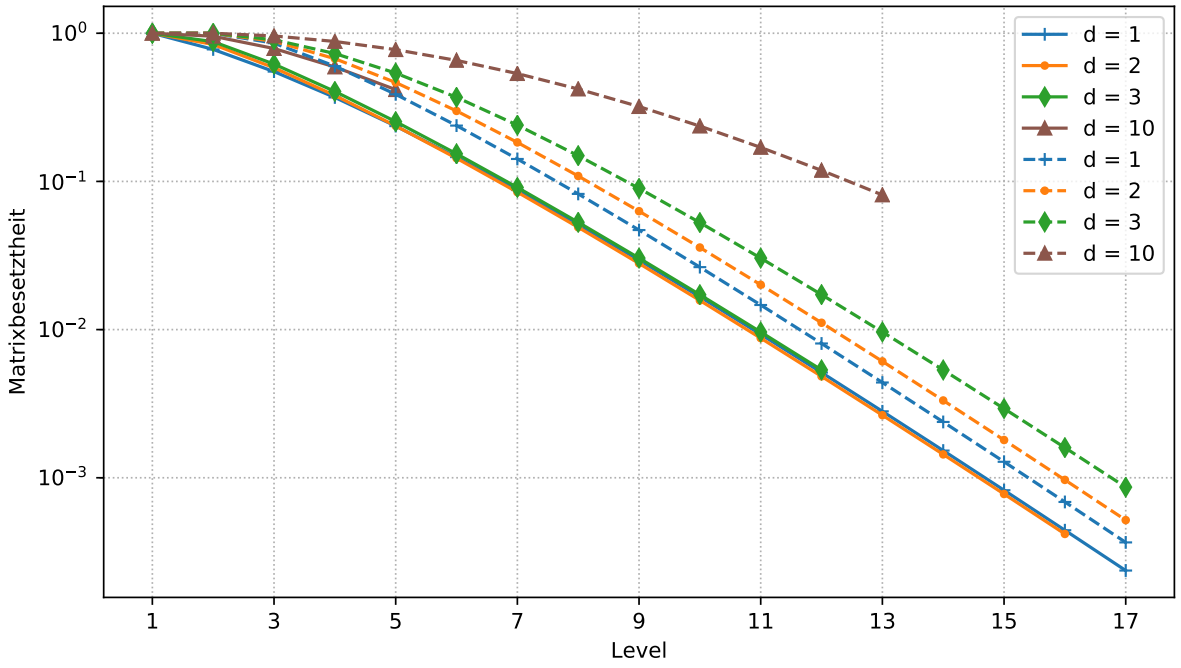

Abbildung 3.2: Besetztheit der Interpolationsmatrix, wobei die durchgezogenen Linien realen Ergebnissen entsprechen und die gestrichelten Linien durch die Abschätzungsformel berechnet wurden

3.3 Löser

Im folgenden werden verschiedene Methode zur Lösung des Gleichungssystems betrachtet. Um ein Gleichungssystem zu lösen gibt es zwei Kategorien von Lösungsmethoden, die direkte und die iterative Methode. Bei Verfahren der direkten Methode wird die Matrix A Schritt für Schritt vereinfacht, so dass nach einer endlichen Anzahl von Schritten die Lösung bekannt ist. Ein iteratives Verfahren tastet sich Schritt für Schritt an die Lösung heran. Die Matrix A wird dabei nicht verändert. Ziel ist es das Residuum $r = b - Ax$ zu minimieren. Der Vorteil von iterativen Verfahren ist, dass man nach einer gewissen Anzahl von Iterationen, obwohl die exakte Lösung noch nicht erreicht ist, aufhören kann und dennoch eine Annäherung an die Lösung erhält.

Für die nachfolgenden Berechnungen wurde ein Rechner mit acht Intel Xeon E7-8800 Prozessoren mit 18 Kernen, welche eine Taktgeschwindigkeit von 2,3 GHz haben, verwendet.

3.3.1 Gauß-Verfahren

Das Gauß-Verfahren ist wahrscheinlich die bekannteste Methode, um ein Gleichungssystem lösen kann. Es gehört zu der Kategorie der direkten Verfahren. Als erstes wird die Matrix in eine obere Dreiecksform umgeformt. Im i-ten Iterationsschritt wird ein Vielfaches der i-ten Zeile auf die nachfolgenden Zeilen drauf addiert, so dass in der Spalte i eine Null erzeugt wird.

Dies wird solange gemacht, bis die Matrix eine Zeilenstufenform hat. Somit ist die Matrix dann eine obere Dreiecksmatrix. Der nächste Schritt ist nun eine Diagonalmatrix zu erzeugen, dann kann man die Lösung direkt ablesen. Wie in [FH07] zu sehen kann den jeweiligen Eintrag x_i des Lösungsvektors x über folgende Gleichung berechnen:

$$x_i := \left(b_i - \sum_{k=i+1}^{n} a_{ik} x_k \right) / a_{ii} \quad \text{für } i = n, n-1, \ldots, 1. \tag{3.9}$$

Die Komplexität des Verfahren ist durch die Anzahl der Multiplikationen festgelegt und liegt bei $\mathcal{O}(n^3)$.

Das Gauß-Verfahren wurde in `SG++` implementiert und durch OpenMP parallelisiert, dies kann in Listing 3.1 betrachtet werden. Die Zeilen 3 bis 8 werden parallel durchlaufen. Die Änderungen in der Matrix W sind für alle Threads direkt sichtbar, da durch den Befehl `shared(W)` die Matrix `W` über alle Threads geteilt wird. Wichtig ist, dass jeder Thread eine eigene lokale Variable `multiplier`, `row` und `col` hat, so dass die Berechnungen korrekt durchgeführt werden und es zu keinen Lost Updates kommt. Dies wird durch die Anweisung `private(multiplier, row, col)` sichergestellt.

Listing 3.1 Paralleles Gauß-Verfahren

```
1  for (norm = 0; norm < n - 1; norm++) {
2   #pragma omp parallel for shared(W) private(multiplier,row,col)
3     for (row = norm + 1; row < n; row++) {
4        multiplier = W.get(row, norm) / W.get(norm, norm);
5        for (col = norm; col <= n; col++) {
6           W.set(row, col, W.get(row, col) - multiplier * W.get(norm, col));
7        }
8     }
9  }
10 // Backwards
11 for (size_t i = n; i > 0; i-- ) {
12  #pragma omp parallel for shared(W)
13    for (size_t j = 0; j < i -1; j++) {
14       W.set(j, n, W.get(j,n) - W.get(j, i-1) / W.get(i-1,i-1) * W.get(i-1,n) );
15    }
16    W.set(i-1, n, W.get(i-1, n) / W.get(i-1, i-1));
17 }
```

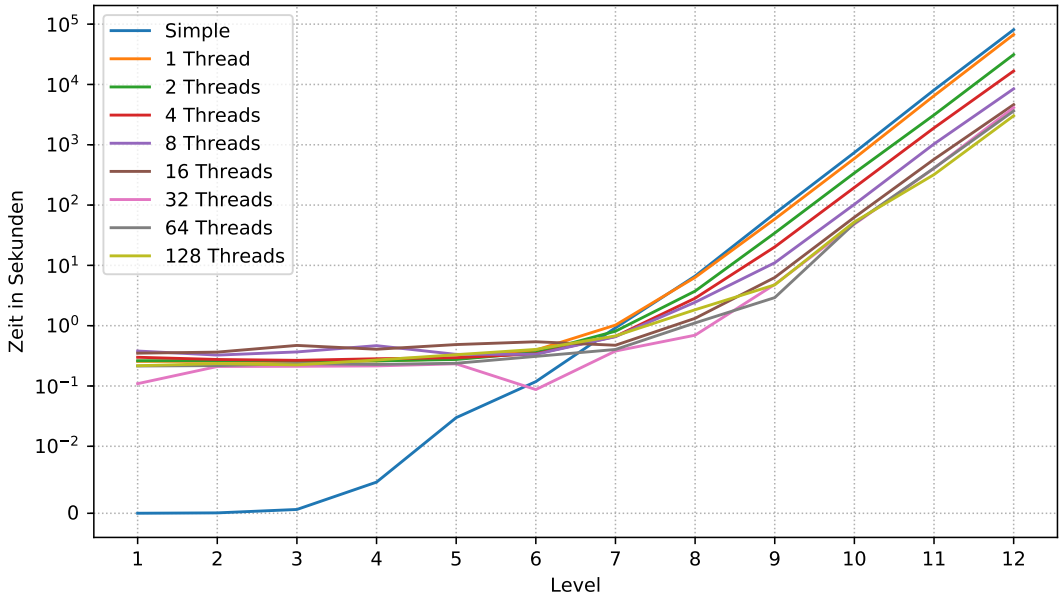

Abbildung 3.3: Laufzeit des Gauß-Verfahrens mit unterschiedlicher Anzahl an Threads für die Berechnung des zweidimensionalen Testfalls

In Abbildung 3.3 sieht man die Laufzeiten mit unterschiedlicher Anzahl von Threads, dabei bezeichnet Simple das bisherige implementierte Gauß-Verfahren, welches auch nur einen Thread nutzt. Man kann erkennen, dass umso mehr Threads genutzt werden, desto schneller läuft die Berechnung. Es ist auch zu sehen, dass diese Steigerung beschränkt ist. Zum Beispiel sieht man bei Level 12, dass 64 und 128 Threads beinahe dieselbe Laufzeit haben. Der Speedup ist durch den seriellen Anteil des Gauß-Verfahrens beschränkt.

Listing 3.2 Parallele LR-Zerlegung

```
1  for (k = 0; k < n - 1; k++) {
2     for(i = k + 1; i < n; i++) {
3        W.set(i, k, W.get(i,k) / W.get(k,k));
4     }
5   #pragma omp parallel for shared(W, k) private(i)
6     for (i = k + 1; i < n; i++) {
7        const double aik = W.get(i,k);
8        for (size_t j = k + 1; j < n; j++) {
9           W.set(i, j, W.get(i, j) - aik * W.get(k, j));
10       }
11    }
12 }
```

3.3.2 LR-Zerlegung

Bei der LR-Zerlegung wird die Matrix A in zwei Matrizen aufgeteilt, so dass $A = LR$. Eine untere Dreiecksmatrix L mit Diagonalelementen 1 und eine obere Dreiecksmatrix R. Dabei können die beiden Matrizen in einer einzigen Matrix gespeichert werden, die Diagonalelemente von L werden dabei nicht gespeichert. In der Literatur ist auch häufig von einer LU-Zerlegung die rede, diese Bezeichnung kommt aus dem Englischen und steht für lower und upper.

Um das Gleichungssystem $Ax = b$ zu lösen bzw. $LRx = b$ wird als erstes

$$Ly = b \tag{3.10}$$

und dann

$$Rx = y \tag{3.11}$$

gelöst.

Sowohl L als auch R sind nur Dreiecksmatrizen. Somit ist der Aufwand um das Gleichungssystem in 3.10 bzw. 3.11 zu lösen deutlich geringer und liegt bei $\mathcal{O}(n^2)$. Der größte Aufwand bei diesem Verfahren liegt bei der Erstellung der LR-Zerlegung. Der Aufwand dafür liegt bei $\mathcal{O}(n^3)$. Hierbei entspricht das Lösen von $Ly = b$ der Vorwärtssubstitution und $Rx = y$ der Rückwartssubstitution des Gauß-Verfahrens. Gleichung (3.11) kann analog zu Gleichung (3.9) gelöst werden.

Auch hier wurde das Verfahren weitestmöglich parallelisiert. Die parallele Erzeugung der Matrizen L und R ist in Listing 3.2 zu sehen. Die Matrizen L und R werden beide in der Matrix W gespeichert. Auch die Lösung der Gleichungen 3.10 und 3.11 wurde parallelisiert.

Listing 3.3 Parallele Matrix-Vektor Multiplikation

```
1  size_t i, j;
2
3  #pragma omp parallel shared (A, v, r) private(i,j)
4   {
5    #pragma omp for schedule(static)
6     for (i = 0; i < n; i++) {
7       r.set(i, 0.0);
8       for (j = 0; j < n; j++) {
9          r.set(i, r.get(i) + A.get(i,j) * v.get(j));
10      }
11    }
12  }
```

3.3.3 BiCGSTAB

Das BiCGSTAB-Verfahren ist ein iteratives Verfahren zum Lösen linearer Gleichungssysteme

$$Ax = b,$$

wobei A eine reguläre $n \times n$ Matrix ist.

Es wurde in [Vor92] vorgestellt und ist eine Verbesserung des BiCG-Verfahrens und des CG-S-Verfahrens (conjugate gradients-squared). Dabei soll das BiCG-Verfahren um die schnelle Konvergenz erweitert werden. Da das Verfahren verwandt mit dem BiCG-Verfahren ist und bessere Stabilitätseigenschaften hat, wurde es BiCGSTAB genannt. In [SB05] wurde angemerkt, dass obwohl das Verfahren an Stabilität gewonnen hat im Vergleich zum BiCG-Verfahren, ist es immer dann abbricht, wenn das zugrunde liegende BiCG-Verfahren kollabiert.

Das Verfahren ist in Algorithmus 3.1 abgebildet. Dabei bezeichnen griechische Buchstaben Skalare, lateinische Buchstaben Vektoren und Großbuchstaben stehen für Matrizen.

Es ist anzumerken, dass die Anzahl der maximalen Iterationen (`maxIt`), nicht allzu hoch festgelegt werden sollte. Ein Test hat gezeigt, dass wenn man das zweidimensionale Beispiel nicht beschränkt, dann bricht das Verfahren nach langer Zeit ab, da es keine Lösung findet.

In jedem Iterationsschritt werden zwei Matrix-Vektor Multiplikationen, vier Skalarprodukte und $12n + 5$ Gleichpunktoperationen berechnet. Aus diesem Grund ist der erste Ansatz, um dieses Verfahren zu parallelisieren, eine parallele Matrix-Vektor Multiplikation. Es wurde der Code von [app] genutzt, welcher in Listing 3.3 zu sehen ist. Die Matrix A und die Vektoren v und r wurden wieder über alle Threads geteilt. Damit keine Lost Updates entstehen, bekommt jeder Thread eine lokale Variable der Schleifenvariablen `i` und `j`.

Große lineare Gleichungssysteme neigen dazu sehr großen Speicher zu benötigen. Dann ist es nötig iterative Verfahren einzusetzen, da man die Matrix nicht abspeichern muss.

Algorithmus 3.1 BiCGSTAB

procedure BiCGSTAB(A, x, b, `maxIt`, ε, x_0)
  $r = b - Ax_0$;
  $\hat{r}_0 = r$;
  $\rho = \alpha = \omega = 1.0$;
  $v = p = 0$;
  for $i = 1, 2, \ldots,$ `maxIt` **do**
    $\rho_{old} = \rho$;
    $\rho = (\hat{r}_0, r)$;
    $\beta = (\rho/\rho_{old}) \cdot (\alpha/\omega)$;
    $p = r + \beta(p - \omega \cdot v)$;
    $v = Ap$;
    $\alpha = \rho/(\hat{r}_0, v)$;
    $s = r - \alpha v$;
    $t = As$;
    $\omega = (t, s)/(t, t)$;
    $x_0 = x_0 + \alpha p + \omega s$;
    if $x_i < \varepsilon$ **then return** x_0;
    end if
    $r = s - \omega t$;
  end for
  return x_0;
end procedure

Implizite Matrix

Für große lineare Gleichungssysteme ist es unmöglich die Matrix A abzuspeichern, da diese irgendwann nicht mehr in den Speicher passt. Beim BiCGSTAB-Verfahren benötigt man A lediglich an zwei Stellen pro Iterationsschritt. Daher ist es offensichtlich, dass man überlegt die Matrix nicht abzuspeichern und jeden Eintrag stets neu zu berechnen. Dies wird im folgenden als implizite Matrix bezeichnet. Die Hoffnung war, dass die Nutzung einer impliziten Matrix nicht wesentlich ins Gewicht der Laufzeit fällt. Leider hat es sich gezeigt, dass das Berechnen der Einträge deutlich zeitintensiver ist, als wenn man die komplette Matrix abspeichern kann. In Abbildung 3.4 wurde das BiCGSTAB-Verfahren in der zwei Dimensionen und mit verschiedenen Level ausgeführt. Dabei ist zu sehen, dass das explizite abspeichern der Matrix einen deutlichen Zeitgewinn mit sich bringt.

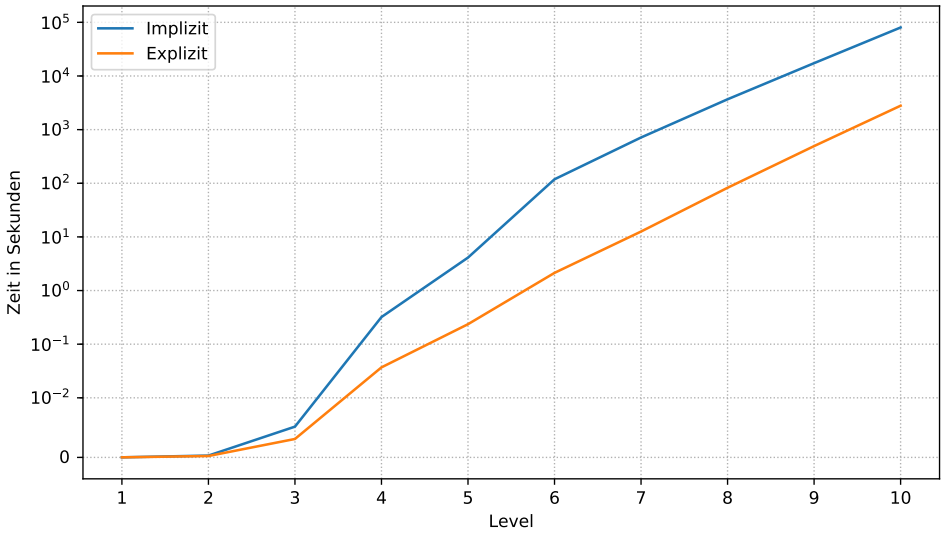

Abbildung 3.4: Vergleich des BiCGSTAB Verfahrens des zweidimensionalen Testfalls, mit impliziter- und expliziter Matrix

4 Vorkonditionierung

Bei der Vorkonditionierung geht es darum eine Matrix P zu konstruieren. Dabei gibt es zwei Kategorien, die Vorwärts- und Inversemethode. Bei der Vorwärtsmethode wird die ursprüngliche Matrix approximiert, d. h. $P \approx A$. Bei der Inversenmethode versucht man direkt die Inversematrix zu approximieren, also $P \approx A^{-1}$. P wird dann mit der Matrix A multipliziert. Wenn man die Matrix P von links an die Matrix A multipliziert, erhält man eine linke Vorkonditionierung. Wenn man P von rechts an A multipliziert erhält man eine rechte Vorkonditionierung. Außerdem gibt es noch eine gemische Vorkonditionierung, wo man die Matrix in zwei Matrizen P_1 und P_2 aufteilt und die eine von links und die andere von rechts an A multipliziert. Im folgenden betrachten wir die linke Vorkonditionierung. Bei der Vorwärtsmethode mit linker Vorkonditionierung wird dann nicht mehr das Gleichungssystem $Ax = b$ gelöst, sondern

$$PAx = Pb. \tag{4.1}$$

Die Vorkonditionierung kann sowohl für direkte Lösungsverfahren als auch für iterative Verfahren verwendet werden. Um zum Beispiel mit höherer Genauigkeit zu rechnen oder die Konvergenz zu beschleunigen.

Definition 4.0.1 (Kondition einer regulären Matrix)
Die Konditionszahl einer regulären Matrix $A \in \mathbb{R}^{n\times n}$ ist definiert als

$$\kappa(A) := \|A\| \cdot \left\|A^{-1}\right\|. \tag{4.2}$$

Damit man überprüfen kann, wie sinnvoll die Vorkonditionierung ist, betrachtet man die Kondition einer Matrix. Dabei soll durch die Vorkonditionierung die Konditionszahl der Matrix verringert werden. Die Konvergenz von iterative Verfahren zum Lösen von Gleichungssystem ist abhängig von der Kondition einer Matrix.

Verschiedene Vorkonditionierer werden in [Che05] und [Mei15] vorgestellt. Dabei werden hier zwei Vorkonditionierer betrachtet. In Abschnitt 4.1 betrachten wir die Jacobi-Vorkonditionerung, welche sehr einfach zu konstruieren ist. In Abschnitt 4.2 schauen wir uns eine sehr beliebte Vorkonditionierungsmethode an, die unvollständige LR-Zerlegung. Abschnitt 4.3 zeigt das BiCGSTAB-Verfahren mit Vorkonditionierung.

4.1 Jacobi-Vorkonditionierung

Der Jacobi Vorkonditionierer (auch Punkt-Vorkonditionierung genannt) ist die einfachste Methode der Vorkonditionierung. Dabei besteht die Vorkonditionierungsmatrix P lediglich aus den Diagonalelementen der Matrix A, also

$$P_{i,j} = \begin{cases} A_{i,j} & \text{falls } i = j, \\ 0 & \text{sonst.} \end{cases}$$

Es ist offensichtlich, dass die Inverse Matrix P^{-1} einfach zu berechnen ist, denn es muss lediglich der Kehrwert der Diagonaleinträge berechnet werden. Außerdem muss kein zusätzlicher Speicher genutzt werden, da die Matrix A bereits vorhanden ist und bei iterativen Verfahren nicht geändert wird.

Es ist zu beachten, dass diese Vorkonditionierung nur bei gewichteten B-Splines anzuwenden ist, da die Matrix bei ungewichteten B-Splines auf der Diagonalen überall der gleiche Wert steht. Somit würde nur jeder Eintrag der Matrix mit einer Konstanten multipliziert werden, welches jedoch keinen Mehrwert für die Kondition liefert.

4.2 Unvollständige LR-Zerlegung

Im folgenden soll nun ein Verfahren betrachtet werden, welches zum Aufstellen der Matrix etwas aufwändiger ist. Die unvollständige LR-Zerlegung gehört unter den Vorkonditionierern zu der Kategorie der Vorwärtsmethode. In der Literatur, wie in [Mei15], wird dieses Verfahren auch unvollständige LU-Zerlegung genannt, kurz ILU. Für die Aufstellung der Matrix kann die LR-Zerlegung größtenteils aus Abschnitt 3.3.2 verwendet werden. Der Unterschied ist, dass hier keine komplette Zerlegung gemacht wird, sondern das Besetzungsstruktur aus A verwendet wird. Einen Vorteil davon ist, da bei einer schwach besetzten Matrix die LR-Zerlegung sehr aufwändig ist, dies kann durch Vernachlässigung von mehreren Einträgen verbessert werden.

Definition 4.2.1
Zu einer gegebenen Matrix $A \in \mathbb{R}^{n \times n}$ bezeichnet

$$\mathcal{M}^A := \{(i,j) \mid a_{ij} \neq 0\} \tag{4.3}$$

die Besetzungsstruktur von A.

Analog zur LR-Zerlegung wird die Matrix A wieder in eine linke untere Dreiecksmatrix L und eine rechte obere Dreiecksmatrix R zerlegt werden. Dabei existiert die Zerlegung aus Gleichung (4.4) unten den folgenden Bedingungen:

1. $l_{ii} = 1$ für $i, \ldots, n$

Algorithmus 4.1 ILU-Zerlegung

```
procedure BERECHNEILU(A, ILU)
    for i = 0, ..., n − 1 do
        for j = 0, ..., n − 1 do
            ILU(i, j) = A(i, j);
        end for
    end for
    for i = 1, ..., n − 1 do
        for k = 0, ..., i − 1 do
            if A(i, k) ≠ 0 then
                ILU(i, k) = ILU(i, k)/ILU(k, k);
                for j = k + 1, ..., n − 1 do
                    if A(i, j) ≠ 0 then
                        ILU(i, j) = ILU(i, j) − ILU(i, k) · ILU(k, j);
                    end if
                end for
            end if
        end for
    end for
end procedure
```

2. $l_{ij} = u_{ij} = 0$, falls $(i, j) \notin \mathcal{M}^A$
3. $(LU)_{ij} = a_{ij}$, falls $(i, j) \in \mathcal{M}^A$

$$A = LR + F \tag{4.4}$$

Diese Zerlegung heißt unvollständig, da eine Restmatrix F übrig bleibt, so dass effizient erhalten bleibt. Sollte die Matrix A dicht besetzt sein, wie man gesehen hat ist dies hier nicht der Fall, dann würde man einen geeigneten Schwellenwert für die Einträge von A festlegen und erst dann die unvollständige LR-Zerlegung machen. Als Vorkonditionierungmatrix P verwenden wir die erhaltene Matrizen L und R, also $P = LR$. Damit die Vorkonditionierung für das Gleichungssystem funktioniert muss P, wie in Gleichung (4.1) zu sehen, noch invertiert werden. P^{-1} entspricht $R^{-1}L^{-1}$ und kann als Vorkonditionierung genutzt werden.

Um die unvollständige LR-Zerlegung zu erstellen wird folgender Algorithmus 4.1 genutzt, ähnlich zu [Che05; Mei15]. Dabei ist dieser Algorithmus für eine Matrix mit Nullindizierung, wie es in der Implementierung genutzt wird.

Algorithmus 4.2 BiCGSTAB (vorkonditioniert)

procedure VORKONDITIONIERTERBICGSTAB(A, x, b, maxIt, ε, x_0, P)
  $r = b - Ax_0$;
  $\hat{r}_0 = r$;
  $\rho = \alpha = \omega = 1.0$;
  $v = p = 0$;
  for $i = 1, 2, \ldots,$ maxIt **do**
    $\rho_{old} = \rho$;
    $\rho = (\hat{r}_0, r)$;
    $\beta = (\rho/\rho_{old}) \cdot (\alpha/\omega)$;
    $p = r + \beta(p - \omega \cdot v)$;
    Löse $Py = p$ nach y
    $v = Ay$;
    $\alpha = \rho/(\hat{r}_0, v)$;
    $s = r - \alpha v$;
    Löse $Pz = s$ nach z
    $t = Az$;
    $\omega = (P_1^{-1}t, P_1^{-1}s)/(P_1^{-1}t, P_1^{-1}t)$;
    $x_0 = x_0 + \alpha y + \omega z$;
    if $x_i < \varepsilon$ **then return** x_0;
    end if
    $r = s - \omega t$;
  end for
  return x_0;
end procedure

4.3 Vorkonditionierung des BiCGSTAB-Verfahren

Die vorkonditionierte Variante des BiCGSTAB-Verfahrens wird analog zu [Vor92] definiert. Die zwei vorgestellten Vorkonditionierungstechniken können mittels der Vorwärtsmethode gemacht werden, so dass $A \approx P = P_1 P_2$ gilt.

Wenn P_2 die Einheitsmatrix E ist, dann entspricht es der Vorkonditionierung von der linken Seite. Daraus folgt das vorkonditionierte BiCGSTAB-Verfahren, wie in Algorithmus 4.2 zu sehen.

5 Vergleich

In diesem Kapitel sollen nun die behandelten Verfahren und Strategien verglichen werden.

Wir beginnen im Abschnitt 5.1 mit den Lösungsverfahren. In Abschnitt 5.2 werden die verschiedenen Vorkonditionierer verglichen.

5.1 Lösungsverfahren

In diesem Abschnitt werden nun die in Abschnitt 3.3 betrachteten Lösungsverfahren verglichen. Dabei wird zuerst auf die serielle und danach die parallelisierte Varianten eingegangen.

Serielle Variante

Für den Vergleich betrachten wir wieder das zweidimensionale Problem mit verschiedenen Level. Damit alle Verfahren vergleichbar bleiben wird auch beim BiCGSTAB die Matrix explizit abgespeichert. Dabei ist in Abbildung 5.1 zu sehen, dass das Gauß-Verfahren und die LR-Zerlegung ungefähr gleich schnell sind. Das BiCGSTAB-Verfahren wurde hier mit einer Toleranzgrenze von 10^{-10}, sowie mit einer maximalen Anzahl von 10000 Iterationen beschränkt. Dabei ist zu bemerken, dass das BiCGSTAB-Verfahren ab Level 8 einen größeren Fehler zum vollen Gitter hat, als das Gauß-Verfahren und die LR-Zerlegung.

Parallelisierte Variante

Für die parallelisierte Variante betrachten wir nun ein fünfdimensionales Problem. Die Verfahren hatten jeweils mit 64 Threads zur Verfügung. Abbildung 5.3 zeigt die Laufzeit der Verfahren. Dabei ist zu beobachten, dass die Laufzeit ähnlich zum seriellen, zweidimensionalen Fall verläuft. Das Gauß-Verfahren verläuft wieder ähnlich zur LR-Zerlegung. Das BiCGSTAB-Verfahren ist weiterhin etwas schlechter.

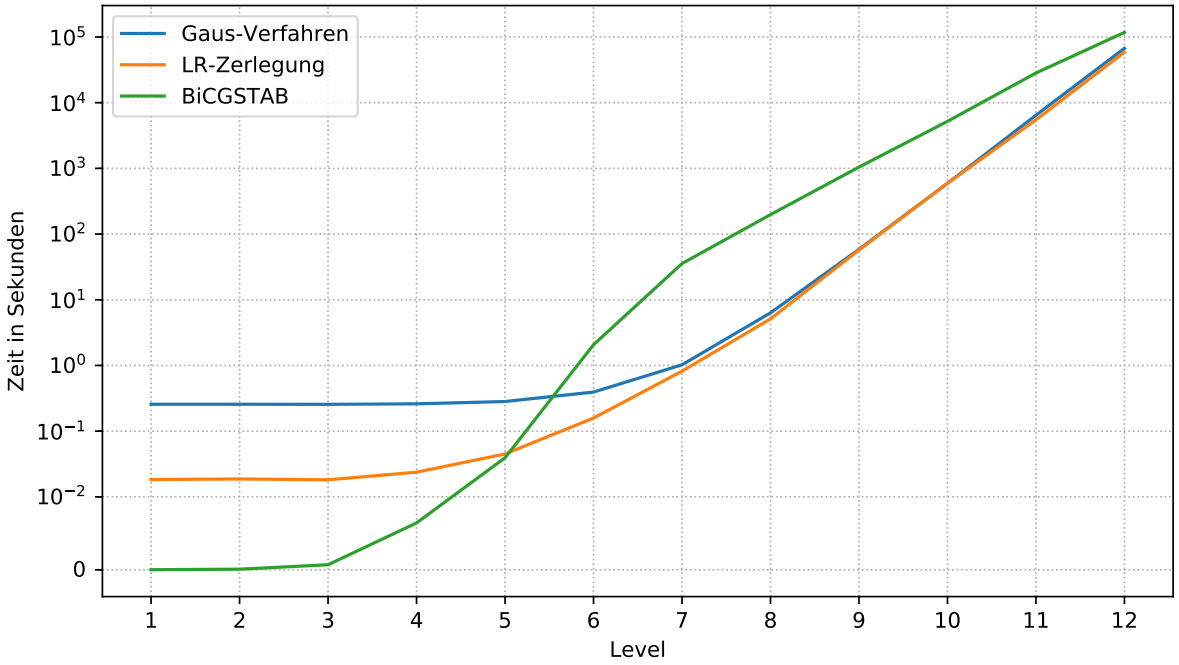

Abbildung 5.1: Vergleich der Laufzeit der seriellen Lösungsverfahren für den zweidimensionalen Testfall

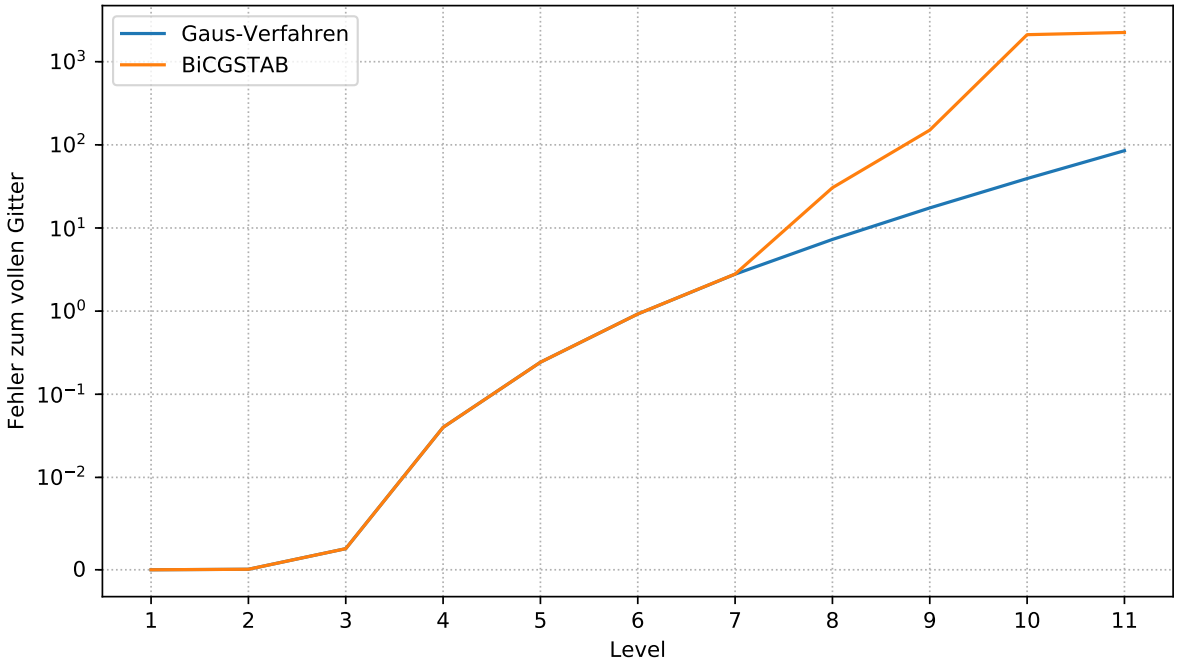

Abbildung 5.2: Vergleich des Fehlers zum vollen Gitter für den zweidimensionalen Testfall

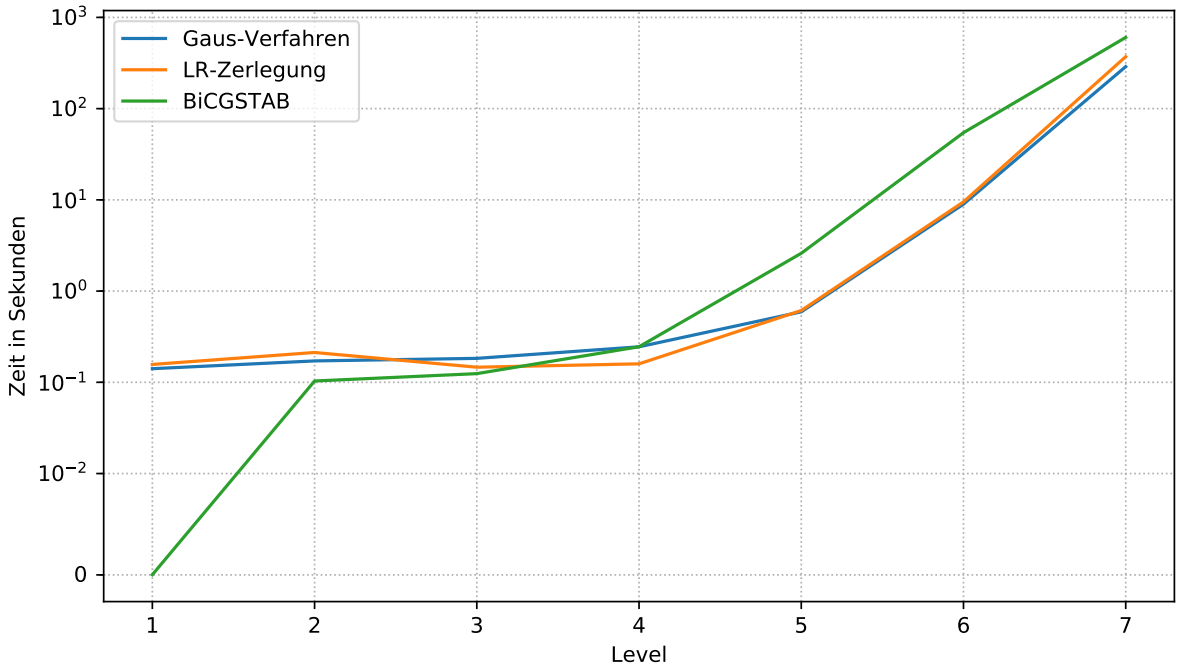

Abbildung 5.3: Vergleich der Laufzeit der parallelen Lösungsverfahren für den fünfdimensionalen Testfall

5.2 Vorkonditionierung

In diesem Abschnitt sollen die Vorkonditionierungen auf das BiCGSTAB-Verfahren angewandt und verglichen werden. Die Verfahren wurden seriell ausgeführt, da die Vorkonditionierung jeweils auf dem selben Verfahren angewandt wurde spielt die Parallelisierung keine Rolle für die Laufzeit.

Abbildung 5.4 zeigt die Konditionszahl der Vorkonditionierer für den zweidimensionalen Testfall.

In Abbildung 5.5 sieht man die Laufzeiten der Verfahren. Dabei wurde das Verfahren ohne Vorkonditionierung, mit der Jacobi-Vorkonditionierung und der unvollständigen LR-Zerlegung auf dem zweidimensionalen Testfall ausgeführt. Man kann erkennen, dass die Berechnung ohne Vorkonditionierung am längsten benötigt. Wie vorher angesprochen ist der Fehler zum vollen Gitter sogar deutlich höher. Sowohl mit der Jacobi-Vorkonditionierung, als auch mit der unvollständigen LR-Zerlegung ist der Fehler zum vollen Gitter derselbe, wie bei den direkten Lösungsverfahren. Wie erwartet ist das Lösen mit der unvollständige LR-Zerlegung am schnellsten. Die Jacobi-Vorkonditionierung ist nur etwas langsamer. Aber man muss auch beachten, dass die unvollständige LR-Zerlegung gemacht werden muss, diese beansprucht auch Zeit und somit ist die Gesamtzeit der Jacobi-Vorkonditionierung am schnellsten.

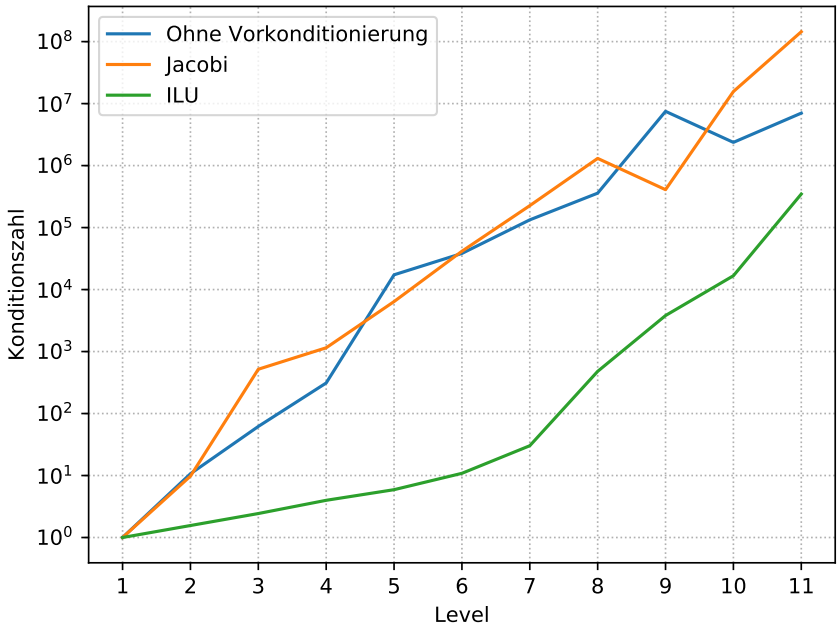

Abbildung 5.4: Vergleich der Konditionszahl für den zweidimensionalen Testfall

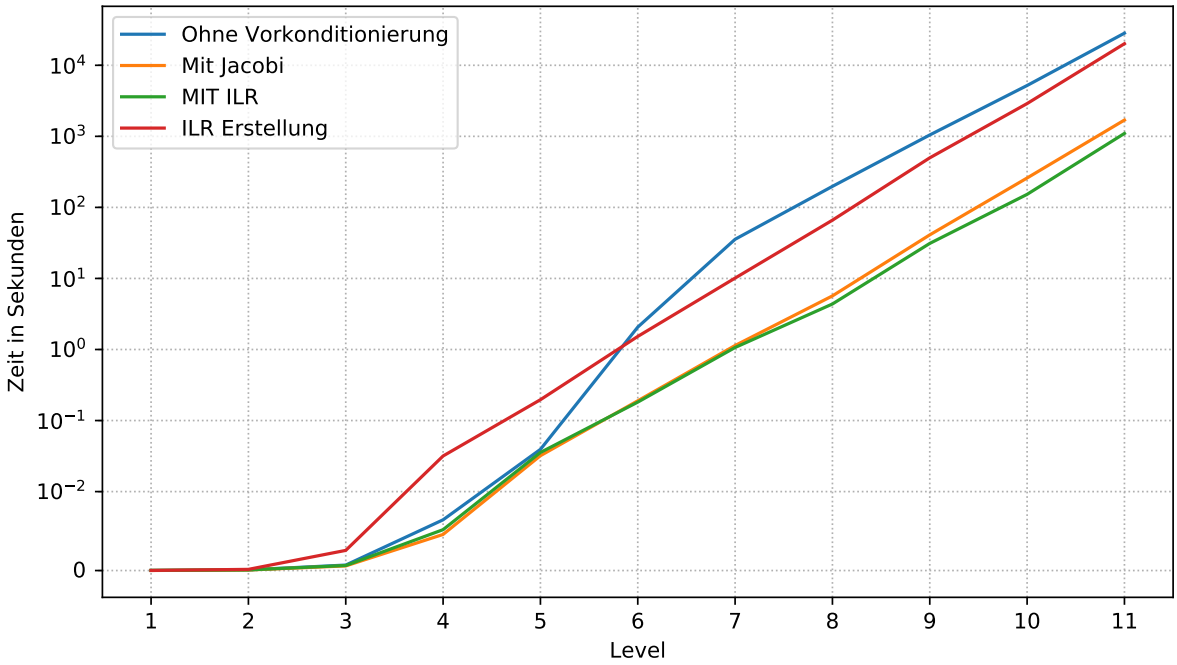

Abbildung 5.5: Vergleich der Vorkonditionierung für den zweidimensionalen Testfall

6 Zusammenfassung und Ausblick

Als erstes wurde die Interpolationsmatrix betrachtet. Dabei ist aufgefallen, dass man eine Formel angegeben kann, welche die Anzahl an Nicht-Null Einträgen in der Interpolationsmatrix berechnen kann. Somit kann man die Besetztheit der Matrix bestimmen und kann sich überlegen, einen Löser, welche für dünn besetzte Matrizen optimiert ist, zu verwenden. Es wurden das Gauß-Verfahren, die LR-Zerlegung und das BiCGSTAB-Verfahren in `SG++` implementiert und parallelisiert. Anschließend wurden vergleiche durchgeführt und bewertet. Da das BiCGSTAB-Verfahren nur sehr langsam lief, wurde die Methode der Vorkonditionierung betrachtet. Hier wurde die Jacobi- und die unvollständige LR-Vorkonditionierung implementiert. Es hat sich gezeigt, dass die unvollständige LR-Zerlegung schneller löst und weniger Iterationsschritte benötigt, jedoch ist der Aufwand die unvollständige LR-Zerlegung zu erzeugen deutlich höher als das Lösen mittels Jacobi-Vorkonditionierung.

Ausblick

Besetztheit der Interpolationsmatrix für allgemeine B-Splines

In Abschnitt 3.2 wurden Formel für die Besetztheit der Interpolationmatrix angegeben. Diese gelten leider nur für B-Splines von Grad $g = 3$. Die herleitung der Formel kam durch das betrachten des B-Splines und die Anzahl der Gitterpunkte zustande. Man kann diese Formel auch für einen beliebigen Grad erweitern.

Vorkonditionierer

In [Che05] gibt es noch weitere Vorkonditionierer, welche genutzt werden können. Auch die Betrachtung der Inversenvorkonditionierer kann untersucht werden. Vor allen kann aufgrund der Formel auf Vorkonditionierer für dünn besetzte Matrizen eingegangen werden.

Eine andere Möglichkeit der Vorkonditionierung ist die Matrix, welche durch die B-Splines von Grad 1 zustande kommt zu nutzen. Dabei sollte P deutlich leichter zu invertieren sein als A, da die Hütchenfunktionen sich nicht überlappen und man hätte wieder eine Vorkonditionierung mit $A \approx P$.

Effizientere parallele Matrix-Vektor Multiplikation

In [HLP95] wurde eine Methode zur effizienten parallelen Matrix-Vektor Multiplikation vorgestellt. Diese könnte untersucht werden in wie weit die Variante durch Kommunikation, schneller ist im Vergleich zu der vorgestellten Parallelen.

Literaturverzeichnis

[app] appentra. *A widely-used algebraic code: Parallel computation of matrix-vector product*. URL: https://www.appentra.com/parallel-computation-of-matrix-vector-product/ (zitiert auf S. 31).

[BG04] H.-J. Bungartz, M. Griebel. „Sparse grids“. In: *Acta Numerica* 13 (2004), S. 147–269. DOI: 10.1017/S0962492904000182 (zitiert auf S. 12, 14–16, 26).

[Che05] K. Chen. *Matrix preconditioning techniques and applications*. 1. publ. Cambridge [u.a.]: Cambridge University Press, 2005, XXIII, 568 Seiten. ISBN: 0-521-83828-2. URL: http://www.gbv.de/dms/goettingen/484962760.pdf (zitiert auf S. 35, 37, 43).

[FH07] R. W. Freund, R. H. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. Zehnte, neu bearbeitete Auflage. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, Online–Ressource (XI, 410 Seiten, 15 Abb, digital). ISBN: 978-3-540-45390-1. URL: http://dx.doi.org/10.1007/978-3-540-45390-1 (zitiert auf S. 28).

[Gar13] J. Garcke. „Sparse Grids in a Nutshell“. In: *Sparse Grids and Applications*. Hrsg. von J. Garcke, M. Griebel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 57–80. ISBN: 978-3-642-31703-3. DOI: 10.1007/978-3-642-31703-3_3. URL: https://doi.org/10.1007/978-3-642-31703-3_3 (zitiert auf S. 11, 12).

[GP16] J. Garcke, D. Pflüger. *Sparse Grids and Applications - Stuttgart 2014*. 1st ed. 2016. Cham: Springer, 2016, Online–Ressource (VIII, 336 p. 85 illus., 24 illus. in color, online resource). ISBN: 978-3-319-28262-6. URL: http://dx.doi.org/10.1007/978-3-319-28262-6.

[HH13] K. Höllig, J. Hörner. *Approximation and modeling with B-splines*. Philadelphia, Pa.: SIAM, Soc. for Industrial und Applied Mathematics, 2013, XIII, 214 Seiten. ISBN: 978-1-611972-94-8 (zitiert auf S. 11, 17, 20).

[HLP95] B. HENDRICKSON, R. LELAND, S. PLIMPTON. „AN EFFICIENT PARALLEL ALGORITHM FOR MATRIX-VECTOR MULTIPLICATION“. In: *International Journal of High Speed Computing* 07.01 (1995), S. 73–88. DOI: 10.1142/S0129053395000051. eprint: http://www.worldscientific.com/doi/pdf/10.1142/S0129053395000051. URL: http://www.worldscientific.com/doi/abs/10.1142/S0129053395000051 (zitiert auf S. 44).

[Höl03] K. Höllig. *Finite element methods with B-splines*. Philadelphia: Soc. for Industrial und Applied Mathematics, 2003, X, 145 Seiten. ISBN: 978-0-898716-99-3. URL: http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=2340445&custom_att_2=simple_viewer (zitiert auf S. 11, 12, 17, 20).

[HV16] S. Hougardy, J. Vygen. *Algorithmische Mathematik*. 1. Aufl. 2016. Berlin, Heidelberg: Springer Spektrum, 2016, Online–Ressource (XIII, 165 Seiten 29 Abb., 23 Abb. in Farbe, online resource). ISBN: 978-3-662-47014-5. URL: http://dx.doi.org/10.1007/978-3-662-47014-5.

[Mei15] A. Meister. *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren. Mit MATLAB®-Implementierungen von C. Vömel*. 5., überarb. Aufl. 2015. Wiesbaden: Springer Spektrum, 2015, Online–Ressource (XII, 276 Seiten 44 Abb, online resource). ISBN: 978-365-80720-0-1. URL: http://dx.doi.org/10.1007/978-3-658-07200-1 (zitiert auf S. 35–37).

[Pfl10] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010. ISBN: 9783868535556. URL: http://www5.in.tum.de/pub/pflueger10spatially.pdf (zitiert auf S. 9, 11, 12, 15, 17, 20).

[SB05] J. Stoer, R. Bulirsch. *Numerische Mathematik 2: Eine Einführung — unter Berücksichtigung von Vorlesungen von F.L. Bauer*. Fünfte Auflage. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, Online–Ressource (X, 394 Seiten 28 Abb, digital). ISBN: 978-3-540-26268-8. URL: http://dx.doi.org/10.1007/b137272 (zitiert auf S. 31).

[Val14] J. Valentin. *Hierarchische Optimierung mit Gradientenverfahren auf Dünngitterfunktionen*. 2014, Online–Ressource. DOI: 10.18419/opus-3462. URL: http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-97586 (zitiert auf S. 11, 12, 18).

[Vor92] 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. DOI: 10.1137/0913035. eprint: https://doi.org/10.1137/0913035. URL: https://doi.org/10.1137/0913035 (zitiert auf S. 31, 38).

Alle URLs wurden zuletzt am 08. 10. 2017 geprüft.

Erklärung

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

Ort, Datum, Unterschrift