Diplomarbeit

Interpolation von vektorwertigen Funktionen mit adaptiven dünnen Gittern

Angefertigt am
Institut für Numerische Simulation

Vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

November 2007

Von
Astrid Fischer
Aus
Darmstadt

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung

Heutzutage ist die Anwendung numerischer Simulationen aus vielen Bereichen der Wissenschaften und Forschung nicht mehr weg zu denken. Im Gleichschritt mit der Entwicklung immer leistungsfähiger Rechner sind numerische Verfahren ein wichtiges Instrument geworden, um komplexe Abläufe zu simulieren. Ein wichtiger Vorteil der numerischen Simulationen ist dabei, dass sie es erlauben, finanziell aufwändige Experimente durch wiederholbare, in der Erstellung meist kostengünstigere Rechnungen zu ersetzen. Die Einsatzgebiete des wissenschaftlichen Rechnens sind dabei weit gefächert und reichen von der Nachbildung technischer und physikalischer bis hin zur Nachbildung biologischer oder ökonomischer Prozesse mittels mathematischer Modelle. In vielen Fällen ersetzen sie nicht nur Experimente sondern ermöglichen erst Erkenntnisse zu gewinnen, da oftmals reale Experimente nicht durchführbar sind oder mit zu großen negativen Auswirkungen verbunden wären. Dies gilt zum Beispiel bei der Klimamodellierung, bei ökonomischen Simulationsmodellen oder bei der Erforschung neuer nuklearer Technologien. Obwohl uns leistungsfähigere Rechner zur Verfügung stehen, erhalten wir mittels numerischer Vefahren nur eine näherungsweise Lösung. Vorgegebene Funktionen werden dabei vereinfacht und die Lösung von schwer lösbaren Gleichungen approximativ bestimmt. Die Funktionsapproximation ist dazu eine wichtige Grundlage. Funktionen können mit geringem Speicheraufwand gespeichert und mit wenig Rechenaufwand weiter genutzt werden. Eine zur Funktionsapproximation oft verwendete Methode ist die Interpolation. Bei der Interpolation wird eine Funktion an den gegebenen Datenpunkten, auch Stützstellen genannt, exakt wiedergegeben und dazwischen näherungsweise bestimmt. Die Methode der Interpolation wurde schon weit vor der Entwicklung der ersten Computer angewendet. So wurden zum Beispiel im 15. bzw. 16. Jahrhundert trigonometrische Tabellen und Logarithmentafeln aufgestellt und dazu genutzt, die nicht in den Tabellen und Tafeln vorkommenden Funktionswerte mittels Interpolation zu bestimmen. Diese Tabellen und Tafeln werden sogar noch bis heute analog verwendet. Selbst wenn wir den Computer bemühen uns den Sinus an einer Stelle x zu berechnen, steht im Hintergrund eine Tabelle, mit Hilfe dieser dann der gesuchte Funktionswert interpoliert wird.

1.2 Problemstellungen und Lösungsansätze

Wir werden in dieser Arbeit Funktionen mittels Interpolation approximieren. Je nachdem welche Ansatzfunktion man wählt, erhält man andere Varianten. Die Ansatzfunktion ist die Funktion, mit der die zwischen den Stützstellen liegenden Werte berechnet werden. Bei der Polynominterpolation nimmt man im eindimensionalen Fall als Ansatzfunktion ein Polynom n-ten Grades bei gegebenen $n+1$ Stützstellen. Geht man dabei stückweise vor, d.h. man hat $k \cdot n+1$ Stützstellen und legt nacheinander k Polynome des n-ten Grades durch die Stützstellen, so kann

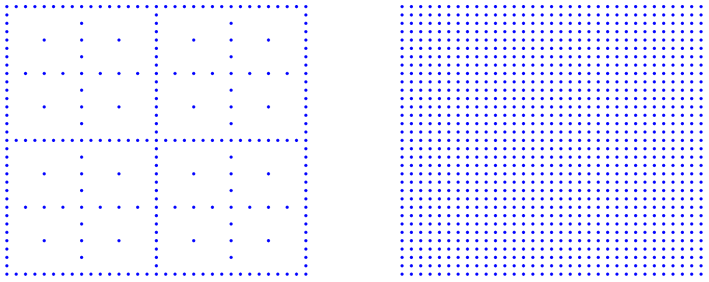


Abb. 1.1: Ein reguläres dünnes Gitter (links) und ein Produktgitter (rechts)

man eine Funktion an beliebig vielen Stützstellen mit Polynomen stückweise interpolieren. Die stückweise Polynominterpolation hat den Vorteil gegenüber der normalen Polynominterpolation, bei der man ein Polynom für das gesamte Interpolationsgebiet verwendet, dass man die Funktion genauer und mit weniger Rechenaufwand interpolieren kann. In der Praxis haben sich Polynome bis maximal des fünften Grades als sinnvoll erwiesen. Wir werden stückweise lineare Funktionen zur Interpolation verwenden.
Im mehrdimensionalen Fall gehen wir grundsätzlich ebenso vor, nur wählen wir Punkte eines beliebigen Gitters auf dem Interpolationsgebiet als Stützstellen aus. Eine Verwendung von Produktgitter-Verfahren (Abbildung 1.1) scheitert bei der Interpolation von hochdimensionalen Funktionen. Wie gezeigt wird, braucht dieses Verfahren zum Level n $O((2^n)^d)$ Punkte, um eine d-dimensionale Funktion mit einer Genauigkeit von $O(h^2)$ in der L_2-Norm zu interpolieren, wobei h die kleinste Schrittweite des Gitters ist und das Level den kleinsten vorkommenden Abstand zwischen zwei Gitterpunkten bestimmt. Der Aufwand des Verfahrens steigt exponentiell mit der Dimension, aber die Genauigkeit nicht. Man spricht dabei vom Fluch der Dimension.
Zur effizienten Interpolation benötigen wir also ein Gitter, das einen geringeren Speicheraufwand hat, aber dennoch eine ähnliche Genauigkeit aufweist wie das Produktgitter-Verfahren. Ein reguläres dünnes Gitter (Abbildung 1.1) erfüllt diese Voraussetzungen. Es basiert auf der stückweise linearen hierarchischen Basis in einer Raumdimension, wobei diese durch den Tensorproduktansatz auf mehrere Dimensionen erweitert wird. Für eine Genauigkeit von $O(h^2(log(h^{-1}))^{d-1})$ braucht das reguläre Dünngitter-Verfahren nur $O(2^n n^{d-1})$ Punkte. Das Verhältnis von Aufwand zu Genauigkeit bei der Funktionsinterpolation wird also durch die dünnen Gitter wesentlich verbessert. Aus diesem Grund eignen sich reguläre Dünngitter-Verfahren eher zur Interpolation von hochdimensionale Funktionen als Produktgitter-Verfahren.
Für die Interpolation singulärer Funktionen sind jedoch auch die regulären Dünngitter-Verfahren nur bedingt geeignet, weil das Verhältnis von Aufwand zu Genauigkeit in diesem Fall unverhältnismäßig hoch ist. Die Idee ist nun dieses Verhältnis durch die Verwendung von Adaptivität noch zu verbessern, so dass singuläre Funktionen angemessen interpoliert werden können. Adaptivität in Bezug auf unsere Problemstellung bedeutet, dass die zur Interpolation benötigten Punkte eines Gitters durch die Koeffizienten der linearen Ansatzfunktionen, die aus der zu interpolierenden Funktion berechnet werden, bestimmt werden. Dadurch kann die Anzahl der benutzten Gitterpunkte bei gleichbleibender Genauigkeit weit geringer ausfallen als bei einem nicht adaptiven Gitter. Wir werden im Verlauf der Arbeit sehen, dass der Aufwand

speziell bei der Interpolation singulärer Funktionen auf adaptiven dünnen Gittern erheblich vermindert werden kann.
Ein weiterer Aspekt dieser Arbeit ist die Betrachtung der Interpolation von vektorwertigen Funktionen auf dünnen Gittern. Auf Grund des oben genannten Arguments können auch hier adaptive Dünngitter-Verfahren gewinnbringend eingesetzt werden. Wir werden zeigen, dass durch eine Aufteilung einer vektorwertige Funktion $F : [0,1]^d \to \mathbb{R}^m$ mit $m \in \mathbb{N}$ in m Teilfunktionen $F_j : [0,1]^d \to \mathbb{R}$ mit $j = 1, \ldots, m$ die Interpolation sinnvoll durchgeführt werden kann. Wir können nämlich so für jede Teilfunktion, unabhängig von den anderen Teilfunktionen, einen adaptiven Dünngitter-Interpolanten bestimmen und ermöglichen dadurch eine effiziente Interpolation vektorwertiger Funktionen.
Bei einigen Diskretisierungs- und Lösungsproblemen müssen Operationen mit Interpolanten durchgeführt werden. In [Sch98] wurden schon die einfachen Operatoren wie Addition und Multiplikation angewendet auf Dünngitter-Interpolanten behandelt. Wir werden diese Dünngitter-Algebra aus [Sch98] kurz zusammenfassen und mit der Hintereinanderschaltung zweier Funktionen erweitern. Betrachten wir dazu eine Funktion F, die äquivalent zur Hintereinanderschaltung von f und g, $f \circ g$, ist. Wir werden dazu zeigen, dass der Interpolationsfehler der Funktion $f \circ g$, wenn f und g für sich interpoliert werden, durch die Interpolationsfehler von f und g additiv abgeschätzt werden kann.
Dabei kann interessanterweise die Idee der Hintereinanderschaltung zur Verbesserung der Interpolation einer Funktion benutzt werden. Wir können nämlich die Genauigkeit der Interpolation von $F = f \circ g$ durch eine geschickte Wahl von f und g derart beeinflussen, so dass wir durch eine Hintereinanderschaltung der Interpolanten einen geringeren Fehler erhalten als bei einer direkten Interpolation von F. Wir werden dazu ein neues Verfahren zur Funktionsinterpolation durch Hintereinanderschaltung konstruieren und zeigen, dass dieses Verfahren basierend auf adaptiven dünnen Gittern Funktionen mit Polstellen besser interpoliert als ein adaptives Dünngitter-Interpolationsverfahren.

1.3 Eigene Beiträge

Zusammenfassend sind die eigenen Beiträge dieser Arbeit:

- die Umsetzung der Interpolation von vektorwertigen Funktionen auf adaptiven dünnen Gittern und die Betrachtung der dafür geeigneten Datenstrukturen

- die Erweiterung der Dünngitter-Algebra aus [Sch98] durch die Operation der Hintereinanderschaltung mit Beweis der Abschätzung des Interpolationsfehlers zweier hintereinandergeschalteter Dünngitter-Interpolanten durch die Interpolationsfehler der beteiligten Interpolanten

- die Konstruktion eines Verfahrens zur Funktionsinterpolation durch Hintereinanderschaltung auf adaptiven dünnen Gittern

- die numerische Untersuchung der Konvergenzrate und des Interpolationsfehlers an Hand der ausgewählten Testfälle

1.4 Gliederung der Arbeit

Wir werden im Kapitel 2 zeigen, wie ein Gitter aufgebaut ist, und die nodale Basis, die hierarchische Basis und die Hutfunktion, auf der die Ansatzfunktionen aufbauen, vorstellen. Desweiteren werden wir zwei Wege aufzeigen, wie die Adaptivität bei der Funktionsinterpolation auf dünnen Gittern durchgeführt werden kann. Am Ende des Kapitels sprechen wir noch über die Berechnung der Anzahl der Gitterpunkte eines regulären dünnen Gitters und eines Produktgitters.
Danach im Kapitel 3 werden wir ausführlich den Interpolationsfehler und die Konvergenzrate von Interpolationsverfahren besprechen. Als erstes werden wir den Interpolationsfehler bei der Funktionsinterpolation auf einem regulären dünnen Gitter und einem Produktgitter herleiten und die Konvergenzrate definieren. Danach stellen wir die Möglichkeiten vor, die uns zur Verfügung stehen, die Konvergenzrate und den Interpolationsfehler in der Praxis zu berechnen, und diskutieren ihre Qualität.
Im Kapitel 4 werden wir über die Operationen, die mit Dünngitter-Interpolanten durchgeführt werden können, sprechen. Besonders werden wir dabei auf den Gesamtfehler eingehen, der bei einer solchen Operation gemacht wird. Wir werden zeigen, dass bei jeder vorgestellten Operation der Gesamtfehler durch eine Kombination der Interpolationsfehler der beteiligten Funktionen beschrieben werden kann. Wir betrachten hier speziell die Hintereinanderschaltung und beweisen ausführlich die zugehörige Fehlerabschätzung.
Nachdem wir theoretische Kenntnisse über die Funktionsinterpolation auf adaptiven und regulären dünnen Gittern erworben haben, werden wir im Kapitel 5 die Algorithmen vorstellen, die wir für eine solche Interpolation benötigen. Das Kapitel ist so unterteilt, dass zuerst die Algorithmen für ein reguläres Dünngitter-Verfahren und dann darauf aufbauend die noch fehlenden Algorithmen vorgestellt werden, um daraus ein adaptives Verfahren zu konstruieren. Danach wird die Interpolation von einfachen Funktionen $f : [0,1]^d \to \mathbb{R}$ auf vektorwertige Funktionen $F : [0,1]^d \to \mathbb{R}^m$ erweitert und dann ein Algorithmus vorgestellt, wie die Interpolation einer Hintereinanderschaltung von Funktionen durchgeführt werden kann. Zum Abschluss werden wir noch unser Verfahren zur Funktionsinterpolation durch Hintereinanderschaltung beschreiben und die dafür benötigten Routinen vorstellen.
Das Kapitel 6 wird sich dann damit beschäftigen, wie wir die Algorithmen sinnvoll und effizient umsetzen können und welche Form unsere Datenstruktur haben muss, wozu wir die für uns geeigneteste Form, eine Hashtabelle, vorstellen.
Abschliessend werden wir im Kapitel 7 die Approximationsgüte unserer Dünngitter-Verfahren an einfachen Funktionen besprechen und aufzeigen, dass das adaptive Verfahren sinnvoll eingesetzt werden kann. Danach werden wir noch die Interpolationsverbesserung des adaptiven Verfahrens, die wir durch den Lookahead-Algorithmus erhalten, anhand der Interpolation von vektorwertigen Funktionen diskutieren. Desweiteren werden wir die Qualität unserer Fehlerabschätzung der Hintereinanderschaltung zweier Dünngitter-Interpolanten an Beispielen untersuchen und schließlich an drei verschiedenen Typen von Funktionen sehen, dass unser neues Verfahren zur Interpolation Funktionen mit Polstellen besser interpoliert als das adaptive Dünngitter-Verfahren.
Im letzten Kapitel werden die Erkenntnisse kurz zusammengefasst und ein Ausblick auf mögliche Erweiterungen und Verbesserungen gegeben.

Danksagung

An dieser Stelle möchte ich allen danken, die zur Entstehung dieser Arbeit beigetragen haben. Insbesondere gilt mein Dank Prof. Dr. Michael Griebel für die Überlassung des Themas, die zahlreichen hilfreichen Ideen und die hervoragenden Arbeitsbedingungen am Institut. Ich bedanke mich bei Prof. Dr. Rolf Krause für die Übernahme des Zweitgutachtens. Desweiteren danke ich Dr. habil. Thomas Gerstner für die gute Betreuung und die vielen konstruktiven Vorschläge. Außerdem gilt mein besonderer Dank meiner Familie für die moralische Unterstützung.

2 Grundlagen

2.1 Gitter

In der Numerik müssen Räume und Geometrien diskretisiert werden, da ein Computer nur endliche Zahlen und Daten verarbeiten kann. Zur Diskretisierung können Gitter verwendet werden. Ein Gitter auf einem Gebiet ist eine Menge von diskreten Punkten, die innerhalb oder auf dem Rand des Gebietes liegen. Die Punkte können dabei im Prinzip eine beliebige Anordnung haben. Zum Beispiel kann das Gebiet $[0,1]^2$, wie in Abbildung 2.1 zu sehen ist, auf verschiedene Art und Weise diskretisiert werden.

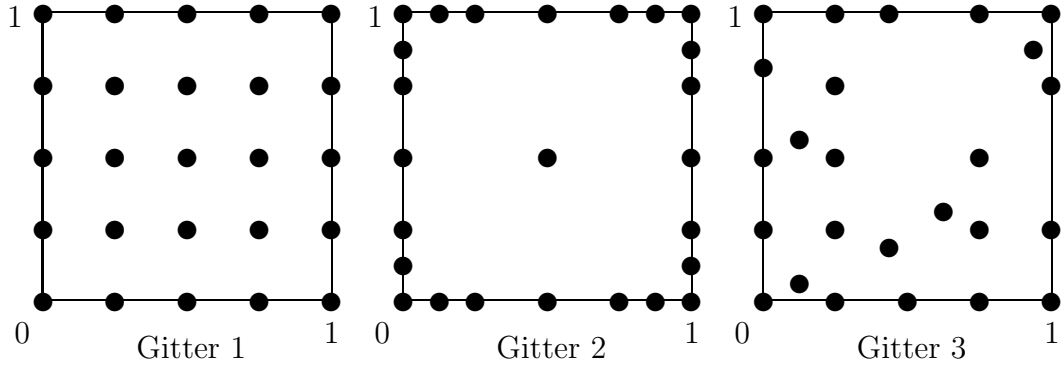


Abb. 2.1: Drei unterschiedliche Gitter auf dem Gebiet $[0,1]^2$ mit jeweils 25 Gitterpunkten

In der Regel wird aber eine geordnete Verteilung der Gitterpunkte bevorzugt, da es sich leichter auf solchen Gittern rechnen lässt und die Gitterpunkte einfacher zu identifizieren sind. Das Gitter 1 in Abbildung 2.1 ist ein sogenanntes volles Gitter oder Produktgitter, welches wir in diesem Kapitel noch genauer besprechen und definieren werden.
Fangen wir dazu mit der Definition der Familie der d-dimensionalen Gitter an und führen einige grundlegende Begriffe ein. Sei $\bar{\Omega} = [0,1]^d$ der d-dimensionale Einheitswürfel. Ein Multi-Index $\mathbf{m}$ besteht aus d Indizes $\mathbf{m} = (m_1, \ldots, m_d) \in \mathbb{N}_0^d$, wobei m_j der Index in die j-te Dimensionsrichtung ist. Seien die zum Multi-Index gehörigen Semi-Normen wie folgt definiert:

$$|\mathbf{m}|_1 = \sum_{i=1}^{d} m_i \quad \text{und} \quad |\mathbf{m}|_\infty = \max_{1 \le i \le d} m_i \tag{2.1}$$

Arithmetische Operationen wie Addition, Subtraktion, Skalarmultiplikation oder Vergleiche werden auf Multi-Indizes stets komponenetenweise ausgeführt. Mit Hilfe dieser Multi-Indizes können wir jetzt die Familie der d-dimensionalen Gitter wie folgt definieren.

Definition 2.1 *Die Familie der d-dimensionalen Gitter sei definiert durch*

$$\{\bar{\Omega}_\mathbf{l}\}_{\mathbf{l} \in \mathbb{N}_0^d}, \tag{2.2}$$

wobei $\mathbf{l} = (l_1, \ldots, l_d)$ *und* $l_i, i = 1, \ldots, d$ *für die i-te Dimensionsrichtung einen Level, d.h. die Gröbe des Gitters in dieser Dimension, angibt.*

Je größer das Level, desto kleiner ist der Abstand zwischen zwei Gitterpunkten. Dieser Zusammenhang führt uns zur Definition der Schrittweite.

Definition 2.2 *Sei* $\mathbf{l} = (l_1, \ldots, l_d)$ *das Level des Gitters* $\bar{\Omega}_\mathbf{l}$*, dann ist die Schrittweite des Gitters definiert durch:*

$$\mathbf{h_l} = (h_{l_1}, \ldots, h_{l_d}) = (2^{-l_1}, \ldots, 2^{-l_d}) = 2^{-|\mathbf{l}|}, \tag{2.3}$$

wobei $l_k \neq l_j$ *für* $k \neq j$ *gelten kann.*

Man könnte die Schrittweite durch $h_{l_k} = n^{-l_k}$ mit $n \geq 2$ und $k \in \{1, \ldots, d\}$ auch anders definieren. Die in der Definition angegebene Variante ist jedoch die gängiste Variante, die wir auch im Rahmen dieser Arbeit verwenden werden. Nach der Definition des Abstands zwischen zwei Gitterpunkten, der Schrittweite, muss noch der Ort, also der Gitterpunkt selbst definiert werden.

Definition 2.3 *Ein Gitterpunkt zum Gitter* $\bar{\Omega}_\mathbf{l}$ *gehörig sei wie folgt definiert:*

$$x_{\mathbf{l},\mathbf{i}} = (x_{l_1,i_1}, \ldots, x_{l_d,i_d}) = \mathbf{i} \cdot \mathbf{h_l} \tag{2.4}$$

mit $x_{l_j,i_j} = i_j \cdot h_{l_j}$ *mit* $i_j \in \{1, \ldots, 2^{l_j} - 1\}$ *für* $h_{l_j} < 1$ *und* $i_j \in \{0, 1\}$ *für* $h_{l_j} = 1$*. Hierbei steht* $\mathbf{i}$ *für den Ort des Gitterpunktes auf einem gegebenen Level* $\mathbf{l}$*.*

In dieser Arbeit werden wir Gitter benutzen, um darauf Funktionen zu interpolieren. Der einfachste Ansatz dies zu tun ist folgender. Man approximiert die betrachtete Funktion zwischen zwei Gitterpunkten mittels Polynomen ersten Grades. Prinzipiell könnte man auch Polynome höheren Grades verwenden, aber diese sind komplizierter zu handhaben. Im nächsten Abschnitt werden wir uns deswegen mit Polynomen ersten Grades bzw. lineare Funktionen beschäftigen und sehen, wie wir diese Polynome für unsere zu interpolierenden Funktionen jeweils berechnen können. Dies führt uns zunächst zum Begriff der hierarchischen Basis.

2.2 Hierarchische Basis

Verbindet man die Polynome ersten Grades, die eine Funktione interpolieren sollen, so erhält man, sofern man das Interpoaltionsproblem in der Dimension d betrachtet, eine stückweise d-lineare Funktion. Der Grundbaustein der stückweise d-linearen Funktionen ist die Hutfunktion. Mit Hilfe dieser Hutfunktion kann erst der Raum der stückweise d-linearen Funktionen definiert werden.

Definition 2.4 *Die Hutfunktion ist definiert als:*

$$\phi(x) = \begin{cases} 1 - |x|\,, & \text{für } x \in [-1, 1] \\ 0, & \text{sonst} \end{cases} \tag{2.5}$$

Zum j-ten Eintrag eines Gitterpunktes $x_{\mathbf{l},\mathbf{i}}$, also x_{l_j,i_j} kann mittels Dilatation und Translation eine zum Punkt passende Hutfunktion erzeugt werden.

Definition 2.5 *Sei $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ und $j \in \{1, \ldots, d\}$, dann ist mit*

$$\phi_{l_j,i_j}(x_j) = \phi\left(\frac{x_j - i_j \cdot h_{l_j}}{h_{l_j}}\right). \tag{2.6}$$

die lineare Hutfunktion zum Gitterpunkt $\mathbf{x}$ definiert.

Da wir Funktionen im d-dimensionalen Raum betrachten wollen, brauchen wir noch ein d-dimensionales Pendant zur Gleichung 2.6.

Definition 2.6 *Sei $x_{\mathbf{l},\mathbf{i}} = (x_{l_1}, \ldots, x_{l_d})$ ein Gitterpunkt des Gitters $\bar{\Omega}_\mathbf{l}$ und $\mathbf{x} = (x_1, \ldots, x_d)$ mit $\mathbf{x} \in \mathbb{R}^d$ dann heißt*

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

die d-lineare Hutfunktion zum Gitterpunkt $x_{\mathbf{l},\mathbf{i}}$.

Somit haben wir nun zu jedem Gitterpunkt des Gitters $\bar{\Omega}_\mathbf{l}$ eine d-lineare Funktion konstruiert und erhalten daraus den Raum dieser Funktionen wie folgt.

Definition 2.7 *Der Raum der stückweise d-linearen Funktionen zum Level $\mathbf{l} \in \mathbb{N}_0^d$ ist definiert über:*

$$V_\mathbf{l} = \operatorname{span}\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) : \mathbf{i} \in I_\mathbf{l}\}, \tag{2.8}$$

wobei $I_\mathbf{l} = \{\mathbf{i} \in \mathbb{N}_0^d : \ i_j = 1, \ldots, 2^{l_j} - 1$ für $l_j > 0$ und $i_j = 0, 1$ für $l_j = 0, \ 1 \leq j \leq d\}$ die Indexmenge des Ortes $\mathbf{i}$ zum Level $\mathbf{l}$ beschreibt.

Definition 2.8 *Die Menge der Funktionen $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ stellen eine Basis des Raumes $V_\mathbf{l}$ dar:*

$$\mathcal{B}_n = \{\phi_{\mathbf{l},\mathbf{i}} : \mathbf{i} \in I_\mathbf{l}\}. \tag{2.9}$$

$\mathcal{B}_n$ wird als nodale Basis bezeichnet.

Nimmt man nur jene Funktionen in $V_\mathbf{l}$, die ungerade Indizes $\mathbf{i}$ besitzen, so erhalten wir die Differenzräume $W_\mathbf{l}$ aufbauend auf den Räumen $V_\mathbf{l}$.

Definition 2.9 *Der Differenzraum $W_\mathbf{l}$ ist definiert durch*

$$W_\mathbf{l} = V_\mathbf{l} \setminus \bigoplus_{j=1}^{d} V_{\mathbf{l}-\mathbf{e}_j} = \operatorname{span}\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \in V_\mathbf{l} : \mathbf{i} \in I_\mathbf{l}^u\}, \tag{2.10}$$

wobei $\mathbf{e}_j$ der d-dimensionale Einheitsvektor als Multi-Index aufzufassen ist und $I_\mathbf{l}^u = \{\mathbf{i} \in \mathbb{N}_0^d : \ i_j = 1, \ldots, 2^{l_j} - 1$ mit i_j ungerade für $l_j > 0$ und $i_j = 0, 1$ für $l_j = 0$, $1 \leq j \leq d\}$ ist. Außerdem müssen wir $V_{\mathbf{l}-\mathbf{e}_j} = \emptyset$ setzen, falls $l_j = 0$ gilt, da sonst die Definition lückenhaft wäre.

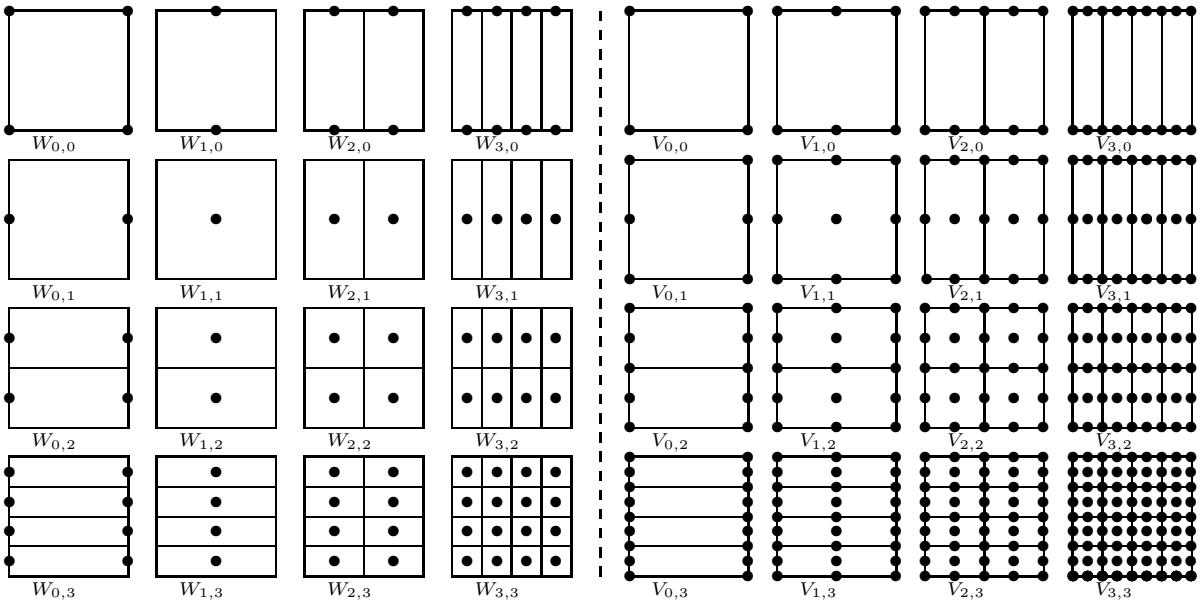


Abb. 2.2: Träger der Basisfunktionen der Differenzräume $W_{i,j}$ und der Räume $V_{i,j}$ mit $i, j = 0 \ldots 3$ in zwei Raumdimensionen

Den Raum $V_{\mathbf{l}}$ können wir nun durch Differenzräume $W_{\mathbf{k}}$ ausdrücken

$$V_{\mathbf{l}} = \bigoplus_{\mathbf{k} \leq \mathbf{l}} W_{\mathbf{k}}. \tag{2.11}$$

Somit erhalten wir eine weitere Basis zum Raum $V_{\mathbf{l}}$.

Definition 2.10 *Die hierarchische Basis $\mathcal{B}_h$ des Raumes $V_{\mathbf{l}}$ ist definiert durch:*

$$\mathcal{B}_h = \{\phi_{\mathbf{k},\mathbf{i}} : \mathbf{i} \in I^u_{\mathbf{k}}, \mathbf{k} \leq \mathbf{l}\}. \tag{2.12}$$

Betrachten wir einmal grafisch die Definitionen und Konzepte. In Abbildungen 2.2 sehen wir die Träger der Basisfunktionen, d.h. die zu den Basen zugehörigen Gitterpunkte, in zwei Dimensionen zu den Leveln (0,0) bis (3,3). Es ist gut zu erkennen, dass z.B. der Raum $W_{2,1}$ aus dem Raum $V_{2,1}$ ohne den Raum $V_{1,1}$ oder der Raum $W_{3,3}$ aus dem Raum $V_{3,3}$ ohne die Räume $V_{3,2}$ und $V_{2,3}$ entsteht (siehe Gleichung (2.10)). Addiert man die Räume $W_{0,0}$, $W_{0,1}$, $W_{0,2}$, $W_{1,0}$, $W_{1,1}$, $W_{1,2}$, $W_{2,0}$, $W_{2,1}$ und $W_{2,2}$, so erhält man den Raum $V_{2,2}$ wie aus Gleichung (2.11) folgt.
In Abbildung 2.3 sind die Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}$ und ihre Gitterpunkte $x_{\mathbf{l},\mathbf{i}}$ der Differenzräume $W_{\mathbf{l}}$ und der Räume $V_{\mathbf{l}}$ in einer Dimension für $\mathbf{l} \in \{0, 1, 2, 3\}$ zu sehen. Für W_2 sind z.B. die Basisfunktionen $\phi_{2,1}$ und $\phi_{2,3}$ zu den Gitterpunkten $x_{2,1}$ und $x_{2,3}$ abgebildet. Für V_2 kommt noch die Basisfunktion $\phi_{2,2}$ dazu. Umso mehr Basisfunktionen wir nehmen, desto besser wird unser Intervall von 0 bis 1 abgedeckt. Also umso höher das Level, desto besser die Diskretisierung des Intervalls $[0, 1]$. Mittels dieser Überlegung und den vorangegangenen Definitionen werden wir nun den Limes von $V_{\mathbf{l}}$ für $\mathbf{l}$ gegen ∞ bilden und damit den Raum der stückweise d-linearen Funktionen definieren.

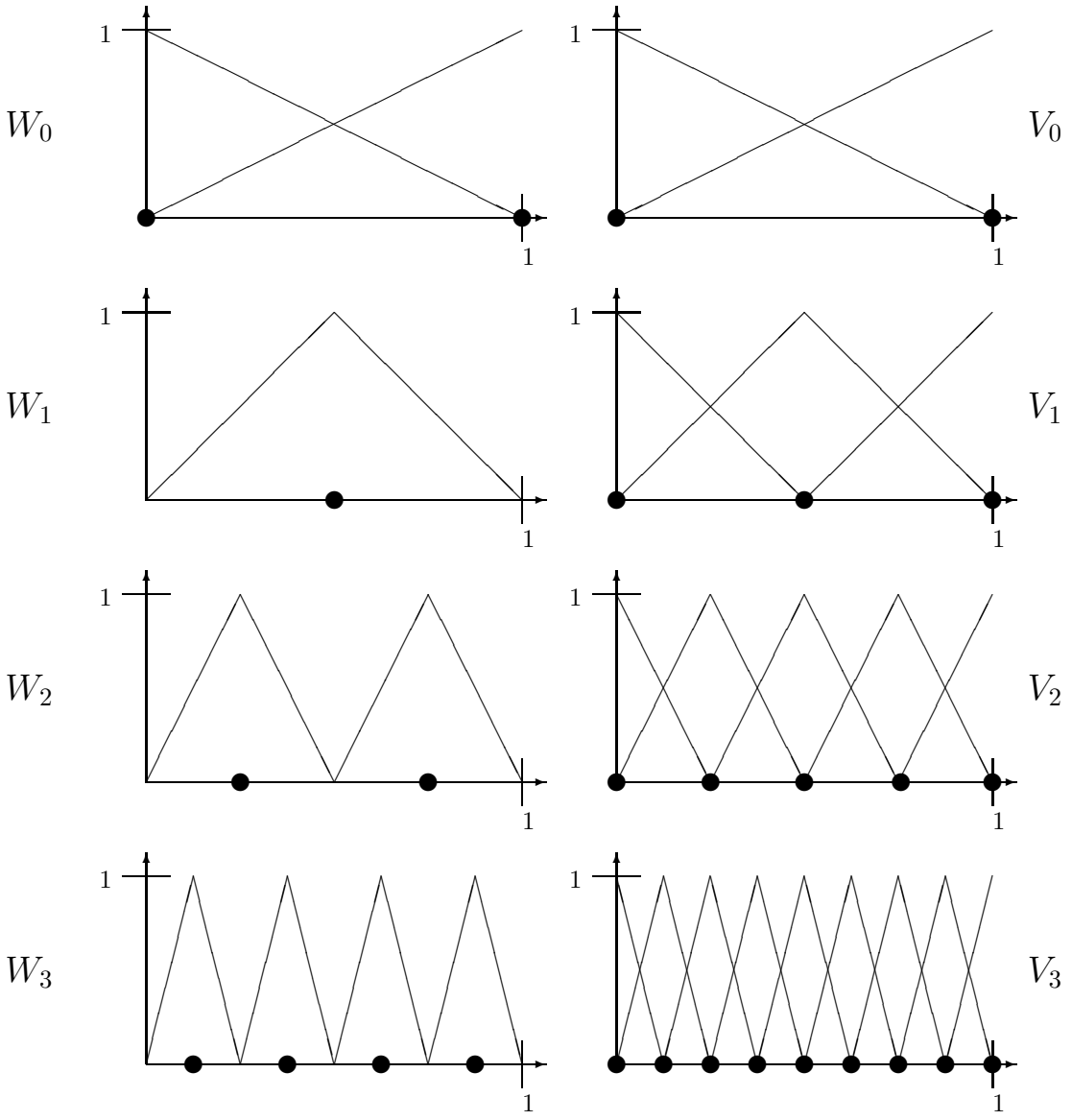


Abb. 2.3: Basisfunktionen der Differenzräume W_l und der Räume V_l in einer Raumdimension für $l \in \{0, 1, 2, 3\}$

Definition 2.11 *Sei $V_\mathbf{l}$ der Raum der stückweise d-linearen Funktionen des Levels $\mathbf{l}$ und $W_\mathbf{l}$ die zugehörigen Differenzräume mit $\mathbf{l} \in \mathbb{N}_0^d$. Sei $\mathbf{0} = (0, \ldots, 0)$ als Multi-Index zu verstehen. Dann definiert*

$$V = \lim_{\mathbf{l}\to\infty} V_\mathbf{l} = \bigoplus_{\mathbf{l}\geq\mathbf{0}} W_\mathbf{l} \tag{2.13}$$

den Raum der stückweise d-linearen Funktionen.

Bildet man die Vervollständigung von V, $\overline{V}$, so ist diese bezüglich der H^1-Norm äquivalent zum Sobolev-Raum $H_0^1\left(\overline{\Omega}\right)$ (siehe Definition 9.3). Auf Grund dieser Äquivalenz kann $u \in H_0^1\left(\overline{\Omega}\right)$ mit Basisfunktionen des Raumes $\overline{V}$ geschrieben werden. Da V entweder durch die nodale oder hierarchische Basis erzeugt werden kann, können wir zudem die Diffenrenzräume anstatt der Räume $V_{\mathbf{l}}$ benutzen, um eine Funktion zu interpolieren. Damit kann dann jegliche Funktion $u \in H_0^1\left(\overline{\Omega}\right)$ mittels

$$u = \sum_{\mathbf{l} \geq \mathbf{0}} u_{\mathbf{l}} = \sum_{\mathbf{l} \geq \mathbf{0}} \sum_{\mathbf{i} \in \mathbf{I}_{\mathbf{l}}^{\mathbf{u}}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}} \tag{2.14}$$

dargestellt werden, wobei $u_{\mathbf{l}} \in W_{\mathbf{l}}$ ist. Die Koeffizienten der Basisfunktionen $u_{\mathbf{l},\mathbf{i}}$ werden hierarchische Überschüsse genannt. Vergleiche zu diesem Abschnitt [BuGri04], [Bun92], [Feuer05], [Gri97] und [Sch98].

2.3 Spezielle reguläre Gitter

In diesem Abschnitt werden nun unterschiedliche Gittertypen definiert. Beginnend mit der Definition des Produktgitters gehen wir über zu der Definition verschiedener dünner Gitter, mit denen wir im Rahmen dieser Arbeit arbeiten werden.

Definition 2.12 *Sei n das maximale Level, d die Dimension, z_0 die Anzahl der $l_k = 0$ von $\mathbf{l}$ und z_1 die Anzahl der $l_k = -1$ von $\mathbf{l}$ und mit $\mathbb{N}_{0,-1} = \mathbb{N} \cup \{0, -1\}$ sind die natürlichen Zahlen mit 0 und -1 gemeint. Mit 0-Rand werden wir die Menge der (l_j, i_j) mit $x_{l_j,i_j} = 0$ und mit 1-Rand werden wir die Menge der (l_j, i_j) mit $x_{l_j,i_j} = 1$ bezeichnen, dann können wir die Gitter wie folgt definieren:*

1. *Produktgitter (PG):*
$$G^{n,d,P} = \bigoplus_{|\mathbf{l}|_\infty \leq n} W_{\mathbf{l}} \text{ mit } \mathbf{l} \in \mathbb{N}_0^d \tag{2.15}$$

2. *Produktgitter mit konstantem 0-Rand (PGK):*
$$G_k^{n,d,P} = \bigoplus_{|\mathbf{l}|_\infty \leq n} W_{\mathbf{l}} \text{ mit } \mathbf{l} \in \mathbb{N}_{0,-1}^d \tag{2.16}$$

3. *reguläres dünnes Gitter (DG):*
$$G^{n,d,D} = \bigoplus_{|\mathbf{l}|_1 - d + 1 + z_0 \leq n} W_{\mathbf{l}} \text{ mit } \mathbf{l} \in \mathbb{N}_0^d \tag{2.17}$$

4. *reguläres dünnes Gitter mit konstantem 0-Rand (DGK'):*
$$G_{k'}^{n,d,D} = \bigoplus_{|\mathbf{l}|_1 - d + 1 + z_0 + 2 \cdot z_1 \leq n} W_{\mathbf{l}} \text{ mit } \mathbf{l} \in \mathbb{N}_{0,-1}^d \tag{2.18}$$

5. *reguläres dünnes Gitter mit konstantem 0-Rand ab -1 gezählt (DGK):*
$$G_k^{n,d,D} = \bigoplus_{|\mathbf{l}|_1 + d - 1 \leq n} W_{\mathbf{l}} \text{ mit } \mathbf{l} \in \mathbb{N}_{0,-1}^d \tag{2.19}$$

Wie man erkennen kann, sind die Gitter endliche Teilräume des Raumes der stückweise d-linearen Funktionen V. Das heißt, wir können die hierarchische Basis $\mathcal{B}_h$ verwenden um Funktionen auf den Gittern zu interpolieren. Zur grafischen Veranschaulichung der Gitter siehe Abbildung (2.4). Diese Abbildung zeigt, dass das DG, DGK und PG zum Level 5 zur Dimension 2.
In Abbildung (2.5) kann man die Unterschiede der Gitter DG, DGK und PG in Bezug auf die benötigten Differenzräume noch besser erkennen, da die Punkte des Gitters jeweils durch ihre Differenzräume gekennzeichnet sind. Im weiteren Verlauf der Arbeit werden wir nur noch die Gitter DG, DGK und PG betrachten.

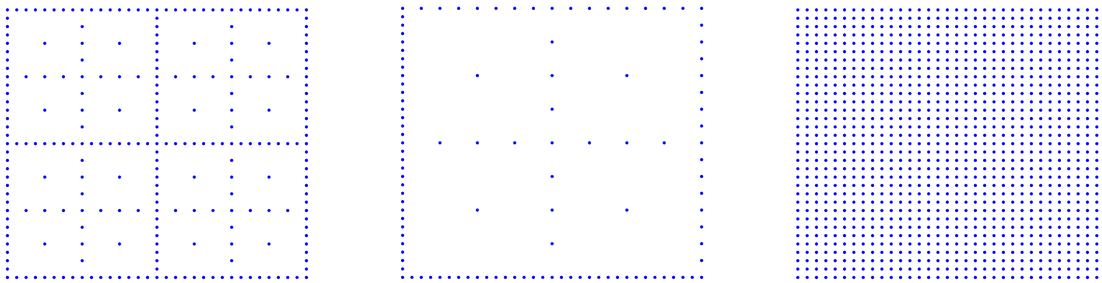


Abb. 2.4: Diese Abbildung zeigt drei verschiedene Gitter zum Level 5 der Dimension 2, wobei das linke Gitter ein DG, das Mittlere ein DGK und das Rechte ein PG ist. Beachte, dass man das linke Gitter auch erhält, wenn mann das DGK' nimmt und dass rechte Gitter, wenn man das PGK nimmt.

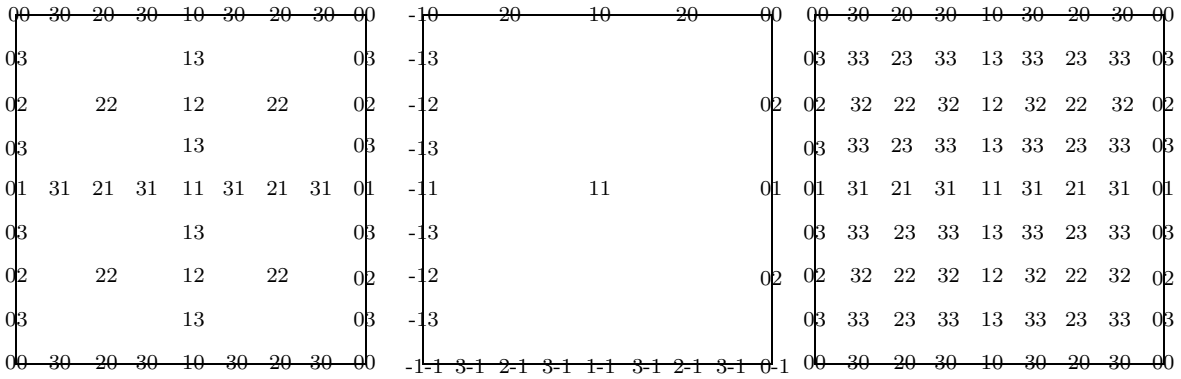


Abb. 2.5: Zu sehen sind drei Gitter mit d=2 und n=3. Die Gitterpunkte sind jeweils durch ihre zugehörigen Differenzräume gekennzeichent. So steht zum Beispiel das Zahlenpaar 22 für einen Punkt des Differenzraumes W_{22}.

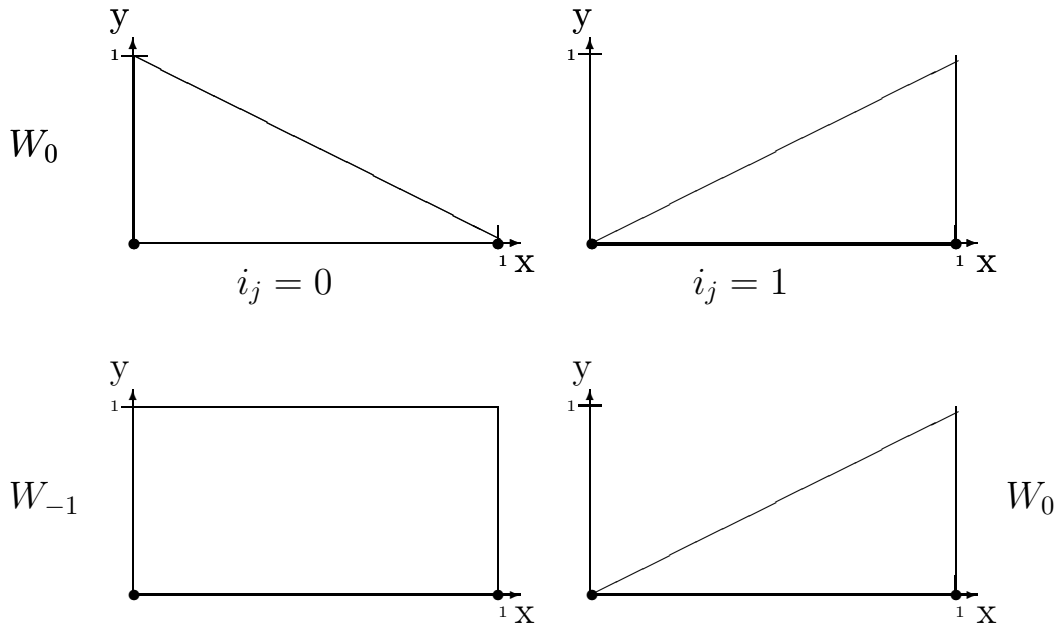


Abb. 2.6: Basisfunktionen des Randes für Gitter mit 0-Rand (oben) und für Gitter mit konstantem 0-Rand (unten).

Im Falle des PGK, des DGK' und des DGK müssen wir die Randpunkte etwas anders als in den Abschnitten 2.1 und 2.2 definieren, da wir an einigen Rändern keine lineare Basisfunktion benutzen, sondern die konstante Basisfunktion verwenden.
Wenn der Levelindex $l_j = 0$ ist, dann sei die Schrittweite $h_{l_j} = 1$ und der Gitterpunkt $x_{l_j,i_j} = 1$ in die j-te Dimensionsrichtung mit $i_j = 1$. Wenn $l_j = -1$ ist, dann sei $h_{l_j} = 1$ und $x_{l_j,i_j} = 0$ mit $i_j = 1$. Außerdem sei $\phi_{l_j,i_j}(x_j) = 1$ für $l_j = -1$. Vergleiche dazu Gleichung (2.6).
Das heißt, der 0-Rand wird durch einen konstanten 0-Rand ersetzt (siehe Abb. 2.6). Dies können wir machen, da die so erzeugte Basis äquivalent zur hierarchischen Basis ist.
Es sei noch darauf hingewiesen, dass es noch andere Varianten von dünnen Gittern gibt. Wählt man andere Normen, z.B. die Energienorm zur Berechnung der Differenzräume, so erhält man Varianten von dünnen Gittern, siehe [BuGri04] und [BuGri97].

2.4 Adaptive Gitter

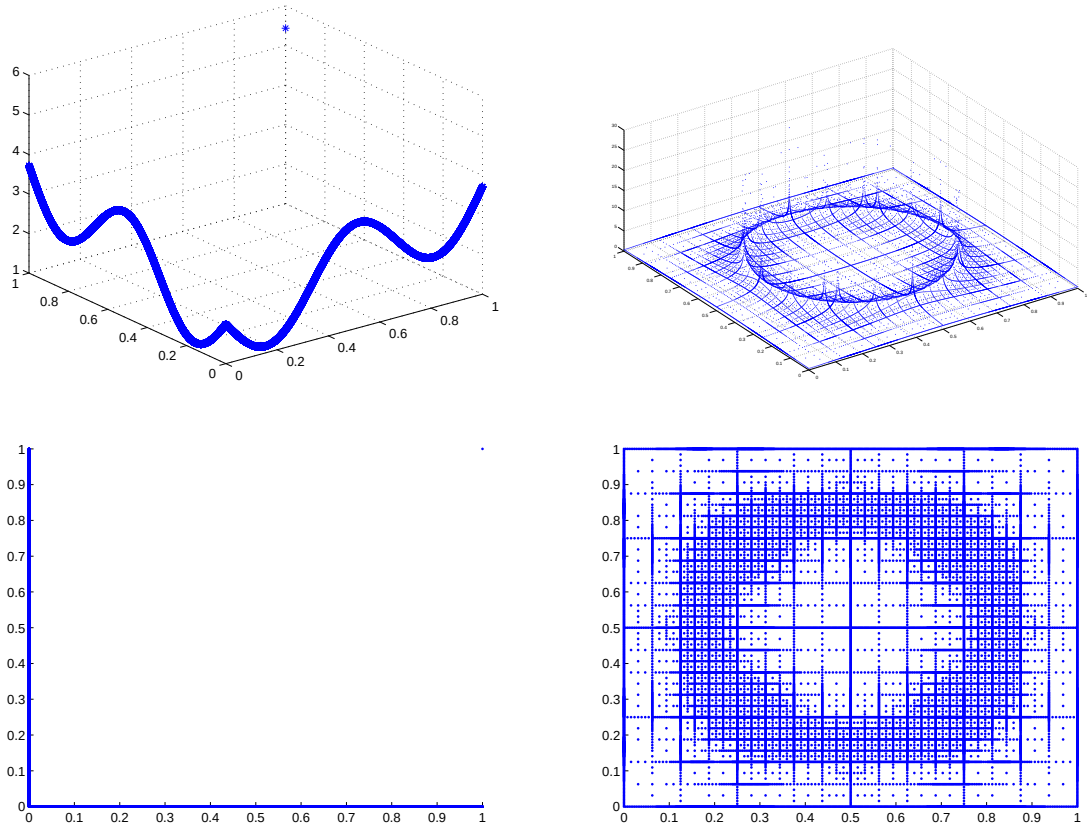


Abb. 2.7: Hier sind zwei adaptive dünne Gitter mit den Funktionswerten geplottet (oben) und zwei adaptive dünne Gitter ohne Funktionswerte geplottet (unten) zu sehen, die durch die Interpolation zweier Funktionen entstanden sind.

Nichtadaptive Gitter sind dadurch gekennzeichnet, dass sich ihre Struktur niemals ändert. Ihr Aussehen bleibt immer gleich. Adaptive Gitter sind Gitter deren Struktur je nach Problemstellung im Laufe des Lösungsprozesses konstruiert wird. Abbildung 2.7 zeigt zwei durch Interpolation von Funktionen entstandene adaptive dünne Gitter. Der Zusammenhang zwischen nichtadaptiven Gittern und adaptiven Gittern besteht darin, dass adaptiven Gittern immer ein nichtadaptives Gitter zu Grunde liegt.
Zur Beschreibung adaptiver Gitter benötigen wir grundlegende Konzepte der Graphentheorie. Zentral dabei sind die Begriffe des Vaters und Sohns aus dem Bereich der Bäume. In Abbildung 2.8 wird ein solcher Baum grafisch dargestellt. Wichtig für uns ist dabei im speziellen der Binärbaum.

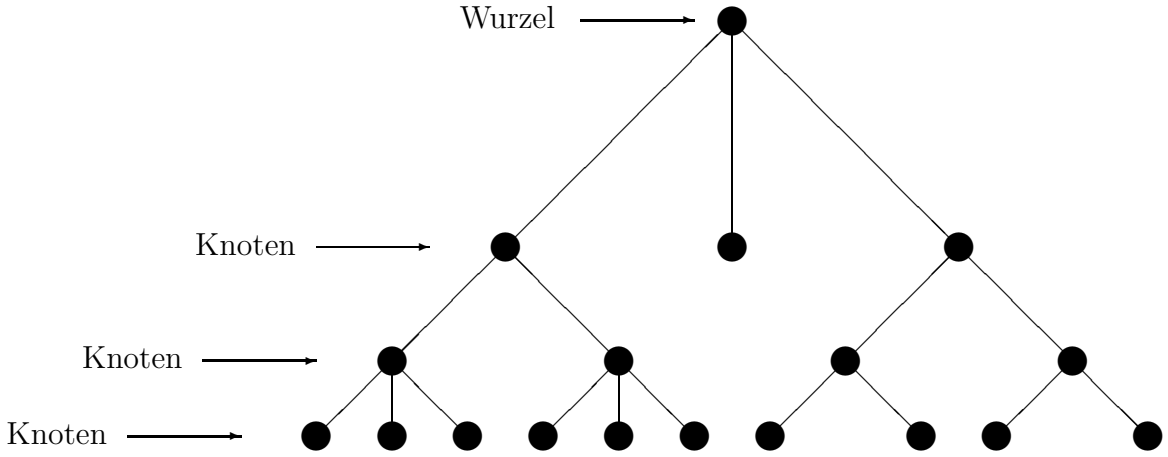


Abb. 2.8: Ein Baum.

Definition 2.13 *Ein Baum ist ein Binärbaum, wenn jeder Knoten des Baumes maximal zwei Nachfolger hat. Der Grundknoten wird Wurzel und die Knoten, die keine Nachfolger haben, Blätter genannt. Der Vorfahre eines Knoten heißt Vater und der Nachfolger Sohn.*

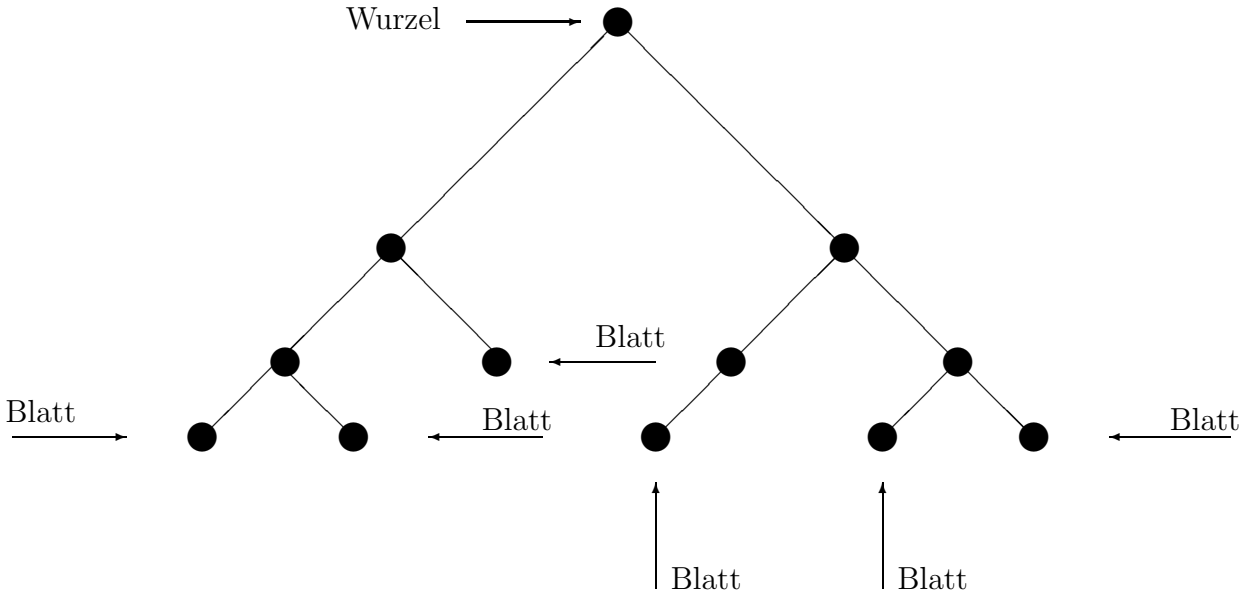


Abb. 2.9: Ein Binärbaum.

Die Verbindung zwischen einem Binärbaum und einem Gitter besteht darin, dass ein Gitter mit der Schrittweite 2^{-n}, wobei $n \in \mathbb{N}_0$ ist, mittels eines Binärbaums äquivalent dargestellt werden kann. Dies wird augenscheinlich, wenn wir den eindimensionalen Fall betrachten. Wie in Abbildung 2.4 zu sehen, kann jeder Knoten als ein Punkt eines Gitters interpretiert werden. Der dort gezeigte Baum zeigt ein DGK des Levels 3. Wir verwenden Binärbäume dazu, um

die Abhängigkeiten zwischen Punkten unseres adaptiven dünnen Gitters mittels der Begriffe Vater und Sohn beschreiben zu können. Mit Hilfe dieses Konstrukts wird es einfacher adaptive dünne Gitter zu beschreiben, wie im weiteren Verlauf der Arbeit deutlich wird.

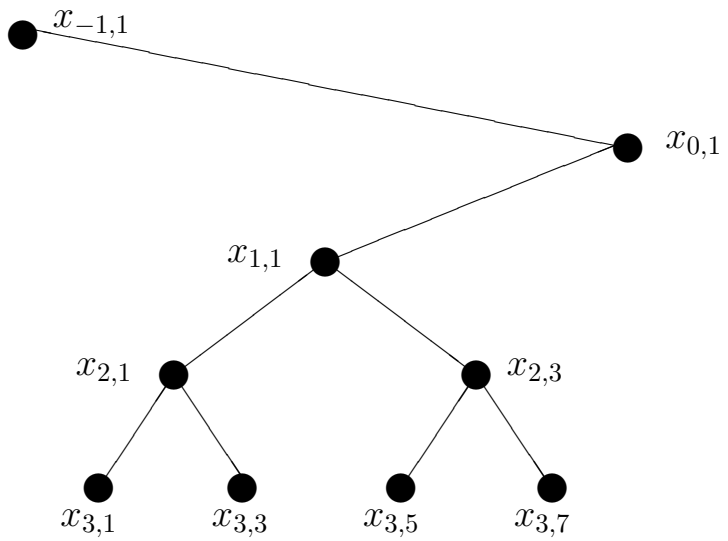


Abb. 2.10: Ein zum Gitter $G_k^{3,1,D}$ äqivalenter Binärbaum. Dabei seien mit $x_{\mathbf{l},\mathbf{i}}$ die Gitterpunkte bezeichnet.

Ein adaptives dünnes Gitter zur Funktionsinterpolation kann wie folgt mit zur Hilfenahme der Binärbaumbegriffe auf zwei Wegen erstellt werden. Auf dem ersten Weg bestimmt man die zur Interpolation benötigten Gitterpunkte nacheinander, d.h. man durchläuft den Baum eines nichtadaptiven dünnen Gitters von der Wurzel beginnend und entscheidet anhand eines Fehlerindikators, ob der betrachtete Knoten relevant ist. Falls dies der Fall ist, so geht man von diesem Knoten (Vater) zu seinen Söhnen und wiederholt die Prozedur. Falls der Knoten nicht relevant ist, wird der Knoten und alle seine Nachfolger ohne weitere Prüfung nicht für das adaptive dünne Gitter verwendet. Beim zweiten Weg stellen wir das komplette nichtadaptive dünne Gitter auf und sortieren alle nichtrelevanten Knoten aus. Bei dieser Variante können Löcher in der Baumstruktur entstehen, was bei der ersten Methode nicht der Fall ist. Dieses Problem kann man aber beheben, indem man nachträglich die entstandenen Löcher wieder mit Knoten füllt. Offensichtlich benötigt die erste Methode in den meisten Fällen weniger Speicher als die zweite Methode, da im zweiten Fall Speicherplatz für ein komplettes nichtadaptives dünnes Gitter reserviert werden muss. Ein Schwerpunkt dieser Arbeit ist es den Speicheraufwand zu minimieren, daher ist hier die erste Methode sinnvoller und deswegen wurde auch diese Methode in den Algorithmen in Kapitel 5 umgesetzt.

2.5 Punktanzahl der Gitter

Nachdem wir den Begriff der hierarchischen Basis eingeführt und Gitter definiert haben, wollen wir uns nun mit der Anzahl der Punkte eines Gitters beschäftigen, um exakte Aussagen über den Speicheraufwand eines adaptiven dünnen Gitters bei der Funktionsinterpolation machen zu können. Sei d die Dimension und l das Level, dann gilt:

- die Anzahl der Punkte eines Produktgitters ist gegeben durch:

$$P_{l,d} = (2^l + 1)^d \tag{2.20}$$

- die Anzahl der Punkte eines regulären dünnen Gitters ergibt sich:

$$\begin{aligned} D_{l,d} &= 2 \cdot D_{l-1,d} + 3 \cdot D_{l,d-1} - 4 \cdot D_{l-1,d-1} \\ &\overset{oder}{=} 3 \cdot D_{l,d-1} + \sum_{j=1}^{l-1} 2^j \cdot D_{l-j,d-1} \\ &\overset{oder}{=} 2 \cdot D_{l,d-1} + \sum_{j=0}^{l-1} 2^j \cdot D_{l-j,d-1} \end{aligned} \tag{2.21}$$

  mit $\quad D_{n,1} = 2^n + 1 \ \forall n \in \mathbb{N}_0$ und $D_{1,t} = 3^t \ \forall t \in \mathbb{N}$

- die Anzahl der Punkte eines dünnen Gitters mit konstantem 0-Rand ab -1 gezählt ist:

$$\begin{aligned} D^k_{l,d} &= D^k_{l,d-1} + D^k_{l-1,d-1} + \sum_{j=-1}^{l-2} 2^{j+1} \cdot D^k_{l-1-j,d-1} \\ &\overset{oder}{=} 2 \cdot D^k_{l-1,d} + D^k_{l,d-1} - D^k_{l-1,d-1} - D^k_{l-2,d-1} \end{aligned} \tag{2.22}$$

  mit $\quad D^k_{-1,t} = 1 \qquad \forall t \in \mathbb{N}$
  $\qquad\quad D^k_{0,t} = t + 1 \qquad \forall t \in \mathbb{N}$
  $\qquad\quad D^k_{n,1} = 2^n + 1 \quad \forall n \in \mathbb{N}_0$

Tabelle 2.5 zeigt die großen Unterschiede zwischen einem Produktgitter und einem dünnen Gitter bezüglich der benötigten Punkte. Für das Level 20 zur Dimension 4 braucht man für ein Produktgitter fast 10^{15}mal soviele Punkte wie für ein dünnes Gitter. Das DG und das DGK unterscheiden sich nur wenig, wobei das DGK weniger Punkte benötigt. Das heißt, dünne Gitter benötigen weniger Speicherplatz als Produktgitter. Deswegen kann man mit Dünngitter-Interpolationsverfahren auch in größeren Dimensionen und mit höheren Leveln arbeiten als mit Produktgitter-Interpolationsverfahren. Wie wir im nächsten Kapitel sehen werden, kann man dabei mit Dünngitter-Interpolationsverfahren unter bestimmten Voraussetzungen eine ähnliche Genauigkeit des Interpolanten wie mit einem Produktgitter-Interpolationsverfahren erreichen.

d	l	$P_{l,d}$	$D_{l,d}$	$D^k_{l,d}$
2	0	4.000000e+00	4.000000e+00	3.000000e+00
2	5	1.089000e+03	2.570000e+02	2.410000e+02
2	10	1.050625e+06	1.331300e+04	1.280100e+04
2	15	1.073807e+09	5.898250e+05	5.734410e+05
2	20	1.099514e+12	2.411725e+07	2.359296e+07
3	0	8.000000e+00	8.000000e+00	4.000000e+00
3	5	3.593700e+04	1.505000e+03	1.337000e+03
3	10	1.076891e+09	1.146890e+05	1.067530e+05
3	15	3.518759e+13	6.619137e+06	6.283265e+06
3	20	1.152925e+18	3.323986e+08	3.190292e+08
4	0	1.600000e+01	1.600000e+01	5.000000e+00
4	5	1.185921e+06	7.681000e+03	6.525000e+03
4	10	1.103813e+12	8.089610e+05	7.322890e+05
4	15	1.153062e+18	5.865472e+07	5.452186e+07
4	20	1.208930e+24	3.528458e+09	3.329360e+09

Tabelle 2.1: Anzahl der Punkte der betrachteten Gitter PG , DG und DGK für $d = 2 \dots 4$ mit $l = 0, 5, 10, 15, 20$.

3 Berechnung des Interpolationsfehlers und der Konvergenzrate

Die Gitter des vorigen Kapitels wollen wir nun zur Interpolation von Funktionen nutzen. Deswegen betrachten wir zuerst welchen Fehler wir bei der Interpolation mit stückweise linearen Funktionen auf dem Produktgitter und welchen Fehler wir bei der Interpolation auf einem regulären dünnen Gitter erwarten können, wenn die zu interpolierende Funktion eine bestimmte Glattheit aufweist. Dann werden wir den Aufwand der Interpolation im Verhältnis zur Genauigkeit des Interpolanten untersuchen und den Begriff der Konvergenzrate einführen.
Und zum Schluss wollen wir einige Möglichkeiten zeigen, mit denen sich der Interpolationsfehler und die Konvergenzrate in der Praxis approximativ bestimmen lassen und ihre Approximationsqualität diskutieren.

3.1 Theorie

3.1.1 Berechnung des Interpolationsfehler

Starten wir mit der Abschätzung des Interpolationsfehlers. Wir werden ihn einmal in der L_2-Norm und einmal in der L_∞-Norm abschätzen. Bevor wir aber die wesentlichen Sätze und Lemmata mit den zugehörigen Beweisen zur Abschätzung des Interpolationsfehlers besprechen können, benötigen wir noch zwei Definitionen.

Definition 3.1 *Sei $p \in \{2, \infty\}$ und $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$ ein Multi-Index. Dann definieren wir den Raum $X^p(\Omega)$ wie folgt:*

$$X^p(\Omega) := \{u \in L^p(\Omega) : \ D^{\boldsymbol{\alpha}} u \in L^p(\Omega) \ \text{ existiert } \forall \, |\boldsymbol{\alpha}|_\infty \leq 2\}$$
$$\text{mit} \quad D^{\boldsymbol{\alpha}} u := \frac{\partial^{|\boldsymbol{\alpha}|_1} u}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}} \quad \text{und} \quad \bar{\Omega} := [0,1]^d.$$

$X_0^p(\Omega) \subset X^p(\Omega)$ ist derjenige Unterraum, dessen Funktionen auf dem Rand des Gebiets $\bar{\Omega}$ verschwinden, also $X_0^p(\Omega) := \{u \in X^p(\Omega) \ : \ u|_{\partial\bar{\Omega}} = 0\}$.

Desweiteren betrachten wir die zum Raum $X_0^p(\Omega)$, $p \in \{2, \infty\}$ gehörigen Seminormen:

Definition 3.2 *Die zum Raum $X_0^p(\Omega)$, $p \in \{2, \infty\}$ gehörigen Seminormen sind wie folgt definiert.*

$$\begin{aligned} |u|_\infty &:= \left|D^{\mathbf{2}} u\right|_\infty && \text{mit} \quad u \in X_0^\infty(\Omega) \\ |u|_2 &:= \left|D^{\mathbf{2}} u\right|_2 = \left(\int_\Omega \left|D^{\mathbf{2}} u(\mathbf{x})\right|^2 \, dx\right)^{\frac{1}{2}} && \text{mit} \quad u \in X_0^2(\Omega) \end{aligned} \tag{3.1}$$

Bemerkung *$X^p(\Omega)$ ist eine Teilmenge des Sobolev-Raumes $W^{2,p}(\Omega)$ (siehe Definition 9.3) und wird im Fall $p = 2$ mit $H^2_{mix}(\Omega)$ bezeichnet.*

Produktgitter

Mit Hilfe der vorigen Definitionen 3.1 und 3.2 können wir nun zur Abschätzung der Interpolationsfehler übergehen. Wir starten mit der Abschätzung des Interpolationsfehlers auf einem Produktgitter.

Satz 3.1 *Sei $u \in X_0^p(\Omega),\ p \in \{2, \infty\}$, dann gelten für den Interpolationsfehler auf einem Produktgitter, gemessen in der L_∞-Norm und in der L_2-Norm, folgende Abschätzungen:*

$$\begin{aligned} |u - \tilde{u}_{n,P}|_\infty &\leq \frac{d}{6^d} \cdot 2^{-2n} \cdot |u|_\infty = O\left(h_n^2\right) \\ |u - \tilde{u}_{n,P}|_2 &\leq \frac{d}{9^d} \cdot 2^{-2n} \cdot |u|_2 = O\left(h_n^2\right), \end{aligned} \tag{3.2}$$

wobei $\tilde{u}_{n,P}$ der Interpolant zu u auf einem Produktgitter mit maximalem Level n ist.

Zum Beweis dieses Satzes benötigen wir noch folgende Lemmata.

Lemma 3.1 *Für jede Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ aus Gleichung (2.7) gelten folgende Gleichungen:*

$$\begin{aligned} |\phi_{\mathbf{l},\mathbf{i}}|_\infty &= 1, \\ |\phi_{\mathbf{l},\mathbf{i}}|_1 &= 2^{-|\mathbf{l}|_1}, \\ |\phi_{\mathbf{l},\mathbf{i}}|_2 &= \left(\frac{2}{3}\right)^{\frac{d}{2}} \cdot 2^{-\frac{|\mathbf{l}|_1}{2}}. \end{aligned} \tag{3.3}$$

Beweis: Die Aussage für die Maximumsnorm folgt direkt aus der Definition der Hutfunktion ϕ (Definition 2.4). Für die L_2-Norm gilt:

$$\begin{aligned} |\phi_{\mathbf{l},\mathbf{i}}|_2 &= \left(\int_\Omega \left(\phi_{\mathbf{l},\mathbf{i}}\right)^2 \, dx\right)^{\frac{1}{2}} \\ &= \left(\int_\Omega \left(\prod_{j=1}^d \phi_{l_j,i_j}(x_j)\right)^2 dx\right)^{\frac{1}{2}} && \text{durch Verwendung von (2.7)} \\ &= \left(\int_\Omega \prod_{j=1}^d \left(\phi_{l_j,i_j}(x_j)\right)^2 \, dx\right)^{\frac{1}{2}} \\ &= \prod_{j=1}^d \left(\int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \left(\phi\left(2^{l_j} x_j - i_j\right)\right)^2 \, dx_j\right)^{\frac{1}{2}} && \text{durch Verwendung von (2.6)} \\ &= \prod_{j=1}^d \left(\int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \left(1 - \left|2^{l_j} x_j - i_j\right|\right)^2 dx_j\right)^{\frac{1}{2}} && \text{durch Verwendung von (2.5)} \\ &= \prod_{j=1}^d \left(\frac{2}{3 \cdot 2^{l_j}}\right)^{\frac{1}{2}} = \left(\frac{2}{3}\right)^{\frac{d}{2}} \cdot 2^{\frac{-\sum_{j=1}^d l_j}{2}} = \left(\frac{2}{3}\right)^{\frac{d}{2}} \cdot 2^{\frac{-|\mathbf{l}|_1}{2}} \end{aligned}$$

Dabei sind $\frac{i_j}{2^{l_j}}$ der Mittelpunkt, $\frac{i_j-1}{2^{l_j}}$ und $\frac{i_j+1}{2^{l_j}}$ die Randpunkte des Trägers von ϕ_{l_j,i_j}. Außerdem ergibt sich für die 1-Norm dann:

$$\begin{aligned}
|\phi_{\mathbf{l},\mathbf{i}}|_1 &= \prod_{j=1}^{d} \int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \phi\left(2^{l_j} x_j - i_j\right)\, dx_j \qquad \text{durch Verwendung von (2.6)} \\
&= \prod_{j=1}^{d} \int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} 1 - \left|2^{l_j} x_j - i_j\right|\, dx_j \qquad \text{durch Verwendung von (2.5)} \\
&= \prod_{j=1}^{d} 2^{-l_j} = 2^{-|\mathbf{l}|_1}
\end{aligned}$$

q.e.d.

Als nächstes führen wir eine Integraldarstellung des hierarchischen Überschusses ein, durch welche der Betrag des hierarchischen Überschusses leichter abgeschätzt werden kann.

Lemma 3.2 (Integraldarstellung der hierarchischen Überschüsse in d Dimensionen) *Sei $u \in X_0^p(\Omega)$, $p \in \{2, \infty\}$ und $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ wie in Gleichung (2.7). Dann kann der hierarchische Überschuss durch die Integraldarstellung*

$$u_{\mathbf{l},\mathbf{i}} = \frac{2^{-|\mathbf{l}|_1}}{2^d} \cdot \int_\Omega \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \frac{\partial^{2d} u(\mathbf{x})}{\partial x_1^2 \ldots x_d^2}\, d\mathbf{x} = \frac{2^{-|\mathbf{l}|_1}}{2^d} \cdot \int_\Omega \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot D^{\mathbf{2}} u(\mathbf{x})\, d\mathbf{x}. \tag{3.4}$$

berechnet werden.

Beweis:

$$\begin{aligned}
&\frac{2^{-|\mathbf{l}|_1}}{2^d} \cdot \int_\Omega \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot D^{\mathbf{2}} u(\mathbf{x})\, d\mathbf{x} = \prod_{j=1}^{d} \frac{1}{2^{l_j+1}} \int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \phi_{l_j,i_j}(x_j) \frac{\partial^2 u(x_j)}{\partial x_j^2}\, dx_j \\
&= \prod_{j=1}^{d} \frac{1}{2^{l_j+1}} \int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \phi\left(2^{l_j} x_j - i_j\right) \frac{\partial^2 u(x_j)}{\partial x_j^2}\, dx_j \\
&= \prod_{j=1}^{d} \left(\frac{1}{2^{l_j+1}} \left[\underbrace{\phi\left(2^{l_j} x_j - i_j\right) \frac{\partial u(x_j)}{\partial x_j^2} \Bigg|_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}}}_{=0} - \int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \frac{\partial \phi\left(2^{l_j} x_j - i_j\right)}{\partial x_j} \frac{\partial u(x_j)}{\partial x_j}\, dx_j \right] \right) \\
&= \prod_{j=1}^{d} \left(\frac{1}{2^{l_j+1}} \left[\int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j}{2^{l_j}}} 2^{l_j} \frac{\partial u(x_j)}{\partial x_j}\, dx_j - \int_{\frac{i_j}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} 2^{l_j} \frac{\partial u(x_j)}{\partial x_j}\, dx_j \right] \right) \\
&= \prod_{j=1}^{d} \left(\frac{1}{2} \left[\int_{\frac{i_j-1}{2^{l_j}}}^{\frac{i_j}{2^{l_j}}} \frac{\partial u(x_j)}{\partial x_j}\, dx_j - \int_{\frac{i_j}{2^{l_j}}}^{\frac{i_j+1}{2^{l_j}}} \frac{\partial u(x_j)}{\partial x_j}\, dx_j \right] \right) \\
&= \prod_{j=1}^{d} \left[u\left(\frac{i_j}{2^{l_j}}\right) - \frac{u\left(\frac{i_j-1}{2^{l_j}}\right) + u\left(\frac{i_j+1}{2^{l_j}}\right)}{2} \right] = \prod_{j=1}^{d} u_{l_j,i_j} = u_{\mathbf{l},\mathbf{i}}
\end{aligned}$$

q.e.d.

Lemma 3.3 *Sei $u \in X_0^p(\Omega)$, $p \in \{2, \infty\}$ bezüglich der Gleichung (2.14) gegeben, dann gelten für die hierarchischen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ folgende Abschätzungen:*

$$\begin{aligned} |u_{\mathbf{l},\mathbf{i}}| &\leq 2^{-d} \cdot 2^{-2|\mathbf{l}|_1} \cdot |u|_\infty, \\ |u_{\mathbf{l},\mathbf{i}}| &\leq 2^{-d} \cdot \left(\frac{2}{3}\right)^{\frac{d}{2}} \cdot 2^{-\frac{3}{2}|\mathbf{l}|_1} \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_2, \end{aligned} \tag{3.5}$$

wobei $\chi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ die charakteristische Funktion des Trägers der Funktion $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ ist.

Beweis:

$$\begin{aligned} |u_{\mathbf{l},\mathbf{i}}| &= \left| \int_\Omega 2^{-|\mathbf{l}|_1 - d} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot D^{\mathbf{2}} u(\mathbf{x})\, d\mathbf{x} \right| \leq 2^{-|\mathbf{l}|_1 - d} \cdot \phi_{\mathbf{l},\mathbf{i}} \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_\infty \\ &= 2^{-|\mathbf{l}|_1 - d} \cdot |\phi_{\mathbf{l},\mathbf{i}}|_1 \cdot |\chi_{\mathbf{l},\mathbf{i}} \cdot u|_\infty \overset{\text{mit (3.1)}}{\leq} 2^{-2|\mathbf{l}|_1 - d} \cdot |u|_\infty \\ |u_{\mathbf{l},\mathbf{i}}| &= \left| \int_\Omega 2^{-|\mathbf{l}|_1 - d} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot D^{\mathbf{2}} u(\mathbf{x})\, d\mathbf{x} \right| \leq \left|2^{-|\mathbf{l}|_1 - d} \cdot \phi_{\mathbf{l},\mathbf{i}}\right|_2 \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_2 \\ &= 2^{-|\mathbf{l}|_1 - d} \cdot |\phi_{\mathbf{l},\mathbf{i}}|_2 \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_2 \overset{\text{mit (3.1)}}{=} 2^{-|\mathbf{l}|_1 - d} \cdot \left(\frac{2}{3}\right)^{\frac{d}{2}} \cdot 2^{-\frac{|\mathbf{l}|_1}{2}} \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_2 \\ &= 2^{-\frac{3}{2}|\mathbf{l}|_1 - \frac{1}{2}d} \cdot 3^{-\frac{1}{2}d} \cdot \left|\chi_{\mathbf{l},\mathbf{i}} \cdot D^{\mathbf{2}} u\right|_2 \end{aligned}$$

q.e.d.

Lemma 3.4 *Sei $u \in X_0^p(\Omega)$, $p \in \{2, \infty\}$ bezüglich der Gleichung (2.14) gegeben, dann gelten für ihre Komponenten $u_{\mathbf{l}} \in W_{\mathbf{l}}$ folgende Abschätzungen:*

$$\begin{aligned} |u_{\mathbf{l}}|_\infty &\leq 2^{-d} \cdot 2^{-2 \cdot |\mathbf{l}|_1} \cdot |u|_\infty, \\ |u_{\mathbf{l}}|_2 &\leq 3^{-d} \cdot 2^{-2 \cdot |\mathbf{l}|1} \cdot |u|_2. \end{aligned} \tag{3.6}$$

Beweis: Zuerst verifizieren wir die Aussage für die Maximumsnorm und dananch für die L_2-Norm. Mit Lemmata 3.1 und 3.3 erhalten wir:

$$\begin{aligned} |u_{\mathbf{l}}|_\infty &= \left| \sum_{i \in I_{\mathbf{l}}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}} \right|_\infty = \sum_{i \in I_{\mathbf{l}}} |u_{\mathbf{l},\mathbf{i}}| \cdot |\phi_{\mathbf{l},\mathbf{i}}|_\infty \\ &\leq \sum_{i \in I_{\mathbf{l}}} 2^{-d} \cdot 2^{-2 \cdot |\mathbf{l}|_1} \cdot |\chi_{\mathbf{l},\mathbf{i}} \cdot u|_\infty = 2^{-d} \cdot 2^{-2 \cdot |\mathbf{l}|_1} \cdot |u|_\infty \end{aligned}$$

Im Falle der L_2-Norm betrachten wir das Quadrat der Norm und erhalten dann mit Lemmata 3.1 und 3.3:

$$\begin{aligned}
|u_{\mathbf{l}}|_2^2 &= \left|\sum_{i\in I_{\mathbf{l}}} u_{\mathbf{l},\mathbf{i}}\cdot\phi_{\mathbf{l},\mathbf{i}}\right|_2^2 = \sum_{i\in I_{\mathbf{l}}} |u_{\mathbf{l},\mathbf{i}}|^2\cdot|\phi_{\mathbf{l},\mathbf{i}}|_2^2 \\
&\le \sum_{i\in I_{\mathbf{l}}} 2^{-2\cdot d}\cdot 2^{-3\cdot|\mathbf{l}|_1}\cdot\left(\frac{2}{3}\right)^d\cdot|\chi_{\mathbf{l},\mathbf{i}}\cdot u|_2^2\cdot|\phi_{\mathbf{l},\mathbf{i}}|_2^2 \\
&= 2^{-2\cdot d}\cdot 2^{-3\cdot|\mathbf{l}|_1}\cdot\left(\frac{2}{3}\right)^d\cdot|u|_2^2\cdot\left(\frac{2}{3}\right)^d\cdot 2^{-|\mathbf{l}|_1} \\
&= 3^{-2\cdot d}\cdot 2^{-4\cdot|\mathbf{l}|_1}\cdot|u|_2^2 \qquad \Rightarrow \text{Behauptung}
\end{aligned}$$

q.e.d.

Mit Lemmata 3.1 bis 3.4 können wir nun den Satz 3.1 beweisen.
Beweis Satz 3.1: Für die L_∞-Norm erhalten wir:

$$\begin{aligned}
|u-\tilde{u}_{n,P}|_\infty &= \left|\sum_{|\mathbf{l}|_\infty=1}^{\infty} u_{\mathbf{l}} - \sum_{|\mathbf{l}|_\infty=1}^{n} u_{\mathbf{l}}\right|_\infty \\
&= \left|\sum_{|\mathbf{l}|_\infty>n} u_{\mathbf{l}}\right|_\infty \le \sum_{|\mathbf{l}|_\infty>n} |u_{\mathbf{l}}|_\infty \\
&\overset{\text{mit (3.6)}}{\le} \sum_{|\mathbf{l}|_\infty>n} 2^{-d}\cdot 2^{-2\cdot|\mathbf{l}|_1}\cdot|u|_\infty \\
&= 2^{-d}\cdot|u|_\infty \sum_{|\mathbf{l}|_\infty>n} 2^{-2\cdot|\mathbf{l}|_1} \\
&= 2^{-d}\cdot|u|_\infty\Big[\sum_{|\mathbf{l}|_\infty=1}^{\infty} 4^{-|\mathbf{l}|_1} - \sum_{|\mathbf{l}|_\infty=1}^{n} 4^{-|\mathbf{l}|_1}\Big] \\
&= 2^{-d}\cdot|u|_\infty\Big[\sum_{l_1=1}^{\infty}\cdots\sum_{l_d=1}^{\infty} 4^{-l_1}\cdots 4^{-l_d} - \sum_{l_1=1}^{n}\cdots\sum_{l_d=1}^{n} 4^{-l_1}\cdots 4^{-l_d}\Big] \\
&= 2^{-d}\cdot|u|_\infty\Big[\big(\sum_{l_1=1}^{\infty} 4^{-l_1}\big)^d - \big(\sum_{l_1=1}^{n} 4^{-l_1}\big)^d\Big]
\end{aligned}$$

Mit der geometrischen Reihe erhält man dann:

$$\begin{aligned}
&= 2^{-d}\cdot|u|_\infty\Big[\big(\frac{4^{-1}}{1-4^{-1}}\big)^d - \big(\frac{4^{-1}\cdot(1-4^{-n})}{1-4^{-1}}\big)^d\Big] \\
&= 2^{-d}\cdot|u|_\infty\Big[3^{-d}\cdot(1-(1-4^{-n})^d)\Big] \\
&\le 2^{-d}\cdot|u|_\infty\cdot 3^{-d}\cdot(1-(1-d\cdot 4^{-n})) \\
&= 6^{-d}\cdot 4^{-n}\cdot d\cdot|u|_\infty
\end{aligned}$$

Für die L_2-Norm erhalten wir analog:

$$\begin{aligned} |u - \tilde{u}_{n,P}|_2 &\leq \sum_{|\mathbf{l}|_\infty > n} |u_{\mathbf{l}}|_2 \overset{\text{mit (3.6)}}{=} 3^{-d} \cdot |u|_2 \sum_{|\mathbf{l}|_\infty > n} 2^{-2\cdot|\mathbf{l}|_1} \\ &\leq 3^{-d} \cdot |u|_2 \cdot 3^{-d} \cdot 4^{-n} \cdot d \\ &= 9^{-d} \cdot 4^{-n} \cdot d \cdot |u|_2 \end{aligned}$$

q.e.d.

Dünne Gitter

Satz 3.2 *Sei $u \in X_0^p(\Omega)$, $p \in \{2, \infty\}$, dann gelten für den Interpolationsfehler auf dünnen Gittern, gemessen in der L_∞-Norm und in der L_2-Norm, folgende obere Schranken:*

$$\begin{aligned} |u - \tilde{u}_{n,D}|_\infty &\leq \frac{4}{3} \cdot 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \cdot A(d,n) = O\left(h_n^2 \cdot n^{d-1}\right) \\ |u - \tilde{u}_{n,D}|_2 &\leq \frac{4}{3} \cdot 12^{-d} \cdot 2^{-2n} \cdot |u|_2 \cdot A(d,n) = O\left(h_n^2 \cdot n^{d-1}\right), \end{aligned} \tag{3.7}$$

wobei $\tilde{u}_{n,D}$ der Interpolant zu u auf einem dünnen Gitter mit maximalem Level n ist und

$$A(d,n) := \sum_{k=0}^{d-1} \binom{n+d-1}{k} = \frac{n^{d-1}}{(d-1)!} + O(n^{d-2}). \tag{3.8}$$

Beweis: Für die L_∞-Norm erhalten wir:

$$\begin{aligned} |u - \tilde{u}_{n,D}|_\infty &= \left| \sum_{|\mathbf{l}|_1=1}^{\infty} u_{\mathbf{l}} - \sum_{|\mathbf{l}|_1=1}^{n+d-1} u_{\mathbf{l}} \right|_\infty = \left| \sum_{|\mathbf{l}|_1 > n+d-1} u_{\mathbf{l}} \right|_\infty \leq \sum_{|\mathbf{l}|_\infty > n+d-1} |u_{\mathbf{l}}|_\infty \\ &\overset{\text{mit (3.6)}}{\leq} \sum_{|\mathbf{l}|_1 > n+d-1} 2^{-d} \cdot 2^{-2\cdot|\mathbf{l}|_1} \cdot |u|_\infty \\ &= 2^{-d} \cdot |u|_\infty \sum_{|\mathbf{l}|_1 > n+d-1} 2^{-2\cdot|\mathbf{l}|_1} \\ &= 2^{-d} \cdot |u|_\infty \sum_{i=n+d}^{\infty} 2^{-2i} \cdot \sum_{|\mathbf{l}|_1 = i} 1 \\ &= 2^{-d} \cdot |u|_\infty \sum_{i=n+d}^{\infty} 2^{-2i} \cdot \binom{i-1}{d-1} \; (*) \\ &= 2^{-d} \cdot |u|_\infty \sum_{i=0}^{\infty} 2^{-2(i+n+d)} \cdot \binom{i+n+d-1}{d-1} \\ &= 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \sum_{i=0}^{\infty} 2^{-2i} \cdot \binom{i+n+d-1}{d-1} \end{aligned}$$

Nun zeigen wir, dass $\forall\ |x| < 1$

$$\sum_{i=0}^{\infty} x^i \cdot \binom{i+n+d-1}{d-1} = \sum_{k=0}^{d-1} \binom{n+d-1}{k} \cdot \left(\frac{x}{1-x}\right)^{d-1-k} \cdot \frac{1}{1-x} \tag{3.9}$$

gilt, um dann den Beweis zu Ende führen zu können.

$$\begin{aligned}
\sum_{i=0}^{\infty} x^i \cdot \binom{i+n+d-1}{d-1} &= \sum_{i=0}^{\infty} x^i \cdot \frac{(i+n+d-1)!}{(d-1)!(i+n)!} \\
&= \sum_{i=0}^{\infty} x^i \cdot \frac{1}{(d-1)!} \prod_{j=0}^{d-2} (i+n+d-1-j) \\
&= \frac{1}{(d-1)!} \sum_{i=0}^{\infty} x^{-n} \cdot x^{i+n} \prod_{j=0}^{d-2} (i+n+d-1-j) \\
&= \frac{x^{-n}}{(d-1)!} \sum_{i=0}^{\infty} \left(x^{i+n+d-1}\right)^{(d-1)} \quad (**) \\
&= \frac{x^{-n}}{(d-1)!} \left(\sum_{i=0}^{\infty} x^{i+n+d-1}\right)^{(d-1)} \quad (***) \\
&= \frac{x^{-n}}{(d-1)!} \left(x^{n+d-1} \lim_{i\to\infty} \frac{1-x^{i+1}}{1-x}\right)^{(d-1)} \\
&\overset{\text{da } |x|<1}{=} \frac{x^{-n}}{(d-1)!} \left(\frac{x^{n+d-1}}{1-x}\right)^{(d-1)} \\
&\overset{\text{Produktregel}}{=} \frac{x^{-n}}{(d-1)!} \sum_{k=0}^{d-1} \binom{d-1}{k} \cdot \left(x^{n+d-1}\right)^{(k)} \cdot (1-x)^{(k-d+1)} \\
&= x^{-n} \sum_{k=0}^{d-1} \frac{1}{k!} \cdot \prod_{j=0}^{k-1} (n+d-1-j) \cdot \frac{x^{n+d-1}}{(1-x)^{d-k}} \\
&= \sum_{k=0}^{d-1} \binom{n+d-1}{k} \cdot \left(\frac{x}{1-x}\right)^{d-1-k} \cdot \frac{1}{1-x}
\end{aligned}$$

$(*)$ Es gibt d natürliche Zahlen, um die Summe i zu bilden, also $\binom{i-1}{d-1}$ Möglichkeiten.
$(**)$ Mit $(x^n)^{(j)}$ ist die j-te Ableitung von x^n gemeint, also $(x^n)^{(j)} = x^{n-j} \cdot \prod_{k=0}^{j-1}(n-k)$.
$(***)$ Die j-te Ableitung einer Summe ist gleich die Summe der j-ten Ableitung der Summanden.

Mit Hilfe von (3.9) können wir nun für die L_∞-Norm den Beweis fortführen:

$$\begin{aligned}
|u - \tilde{u}_{n,D}|_\infty &= 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \sum_{i=0}^{\infty} 2^{-2i} \cdot \binom{i+n+d-1}{d-1} \\
&\overset{(3.9)}{=} 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \sum_{k=0}^{d-1} \binom{n+d-1}{k} \cdot \left(\frac{2^{-2}}{1-2^{-2}}\right)^{d-1-k} \cdot \frac{1}{1-2^{-2}} \\
&= 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \sum_{k=0}^{d-1} \binom{n+d-1}{k} \cdot \left(\frac{1}{3}\right)^{d-1-k} \cdot \frac{4}{3} \\
&\leq 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \cdot \frac{4}{3} \sum_{k=0}^{d-1} \binom{n+d-1}{k} \\
&\overset{(3.8)}{=} \frac{4}{3} \cdot 8^{-d} \cdot 2^{-2n} \cdot |u|_\infty \cdot A(d,n)
\end{aligned}$$

Für die L_2-Norm ergibt sich fast analog, bis auf einen kleinen Unterschied in den Konstanten, der aber auf die Ungleichung (3.6) zurückzuführen ist, folgende obere Grenzen:

$$\begin{aligned}
|u - \tilde{u}_{n,D}|_2 &\leq \sum_{|\mathbf{l}|_2 > n+d-1} |u_{\mathbf{l}}|_2 \\
&= 3^{-d} \cdot |u|_2 \sum_{|\mathbf{l}|_1 > n+d-1} 2^{-2 \cdot |\mathbf{l}|_1} \\
&= \frac{4}{3} \cdot 12^{-d} \cdot 2^{-2n} \cdot |u|_2 \cdot A(d,n)
\end{aligned}$$

q.e.d.

Vergleich der beiden Gitter

Der Interpolationsfehler eines Dünngitter-Interpolationsverfahrens kann maximal so gut wie der eines Produktgitter-Interpolationsverfahrens werden, nie besser. Das liegt daran, dass das dünne Gitter nur ein Teil des zugehörigen Produktgitters ist. In der Tabelle 3.1 sind die Ergebnisse aus Abschnitt 3.1.1 zusammengefasst. Man kann der Tabelle 3.1 entnehmen, dass die Interpolation auf einem dünnen Gitter erheblich weniger Punkte benötigt als die Interpolation auf einem Produktgitter. Außerdem ist der Interpolationsfehler des Dünngitter-Verfahrens nur um den Faktor n^{d-1} schlechter als der Interpolationsfehler eines Produktgitter-Verfahrens. Das reguläre Dünngitter-Verfahren kann also noch bei hochdimensionalen Funktionen sinnvoll eingestzt werden. Das Produktgitter-Verfahren hingegen erliegt bei hochdimensionalen Funktionen dem Fluch der Dimension, da die Anzahl der Punkte mit dem Level exponentiell steigt. Für das adaptive Dünngitter-Interpolationsverfahren erwarten wir die gleichen Interpolationsfehlerschranken wie für das reguläre Dünngitter-Interpolationsverfahren, weil das adaptive Verfahren nur die Punkte berücksichtigt, die für die Interpolation der betrachten Funktion nötig

	$G^{n,d,P}$	$G^{n,d,D}$	$G^{n,d,P}$	$G^{n,d,D}$
$\|u-\tilde{u}^{n,\cdot}\|_\infty$	$\frac{d}{6^d}\cdot 2^{-2n}\cdot \|u\|_\infty$	$\frac{2}{8^d}\cdot 2^{-2n}\cdot \|u\|_\infty \cdot A(d,n)$	$O(h_n^2)$	$O(h_n^2\cdot n^{d-1})$
$\|u-\tilde{u}^{n,\cdot}\|_2$	$\frac{d}{9^d}\cdot 2^{-2n}\cdot \|u\|_2$	$\frac{2}{12^d}\cdot 2^{-2n}\cdot \|u\|_2 \cdot A(d,n)$	$O(h_n^2)$	$O(h_n^2\cdot n^{d-1})$
Punktanzahl	$2^{n\cdot d}$	-	$O((2^n)^d)$	$O(2^n n^{d-1})$

Tabelle 3.1: Obere Schranke des Interpolationsfehlers einer d-dimensionalen Funktion $u \in X_0^p$ mit $p \in \{2,\infty\}$ interpoliert auf einem dünnen Gitter und einem Produktgitter zum maximalen Level n und die Anzahl an Gitterpunkten der Gitter

sind. Für das Verhältnis der Anzahl der Punkte zum Interpolationsfehler erwarten wir mit dem adaptiven Verfahren ein besseres Ergebnis als mit dem nicht adaptiven Verfahren. Im schlechtesten Fall aber kann das adaptive dünne Gitter nach der Interpolation ein reguläres dünnes Gitter sein. Dann entspricht das Verhältnis der Anzahl der Punkte zum Interpolationsfehler mit dem adaptiven Dünngitter-Verfahren dem des regulären Dünngitter-Verfahrens. Wir werden später an Beispielfunktionen in Kapitel 7 unsere Erwartungen bestätigt sehen.

3.1.2 Berechnung der Konvergenzrate

Um später Aussagen über die Interpolationsgüte machen zu können, benötigen wir neben dem Interpolationsfehler noch ein Maß, das das Verhältnis zwischen Aufwand und der Genauigkeit der Interpolation beschreibt. Bei der Interpolation auf nicht adaptiven Gittern ist die Konvergenzordnung ein solches Maß.

Definition 3.3 *Das Verfahren hat eine Konvergenz der Ordnung p mit $p > 0$, wenn*

$$\|u - u_n\| = O(h_n^p) \quad \forall\ n \geq N \in \mathbb{N}. \tag{3.10}$$

gilt, wobei u die zu interpolierende Funktion und u_n die interpolierte Funktion zur Schrittweite h_n des Gitters ist. Für $p = 2$ spricht man von quadratischer Konvergenz und für $p = 1$ von linearer Konvergenz.

Leider sagt die Konvergenzordnung nur bei nicht adaptiven Gittern etwas über das Verhältnis von Aufwand zu Genauigkeit aus, da der Aufwand an die Schrittweite gekoppelt ist. Bei adaptiven Gittern kann man die Schrittweite nicht als Maß für den Aufwand verwenden, sondern muss sich etwas anderes überlegen. Die Anzahl der Gitterpunkte ist eine gute Größe, um den Aufwand der Interpolation zu beschreiben, aber auch ein ordentliches Maß, um ein adaptives und ein nichtadaptives Gitter miteinander vergleichen zu können.

Definition 3.4 *Das Verfahren hat eine Konvergenzrate $p > 0$ in Abhängigkeit von der Dimension d, wenn*

$$\|u - u_n\| = O(A^{-p}) \quad \forall\ n \geq N \in \mathbb{N} \tag{3.11}$$

gilt. Dabei ist u die zu interpolierende Funktion und u_n die interpolierte Funktion zur Schrittweite h_n des Gitters. A ist die Anzahl an benötigten Punkten.

Die Konvergenzrate bietet uns eine Möglichkeit ein reguläres Dünngitter-Verfahren mit einem adaptiven Dünngitter-Verfahren in Bezug auf das Verhältnis von Aufwand zu Genauigkeit zu vergleichen.
Zum Beispiel erwarten wir für eine zweidimensionale Funktion mit Singularitäten unterschiedliche Konvergenzraten für ein adaptives und ein nicht adaptives Verfahren und für eine zweidimensionale glatte Funktion dieselbe asymptotische Rate (Tabelle 3.2).

Glattheit der Funktion	PG	APG	DG	ADG
glatt	1	1	2	2
singulär	$\frac{1}{2}$	1	$\frac{1}{2}$	2

Tabelle 3.2: Die theoretisch zu erwartende Konvergenzrate eines Produktgitter-Verfahrens (PG), eines adaptiven Produktgitter-Verfahrens (APG), eines Dünngitter-Verfahrens (DG) und eines adaptiven Dünngitter-Verfahrens (ADG) bei der Interpolation einer glatten zweidimensionalen Funktion und einer singulären zweidimensionalen Funktion

3.2 Praxis

3.2.1 Berechnung des Interpolationsfehlers

Der Interpolationsfehler gibt uns ein Maß für die Genauigkeit einer Funktionsinterpolation. Wenn man die Interpolante nur in einem Punkt mit der Originalfunktion vergleicht, kann man keine gute Aussage über die Qualität der Interpolation machen. Deswegen macht es Sinn den Fehler global zu betrachten. Wir werden dazu die L_2-Norm und die L_∞-Norm verwenden.

Definition 3.5 *Der absolute L_2-Interpolationsfehler ist definiert durch*

$$|u - \tilde{u}|_2 := \sqrt{\int_\Omega (u(x) - \tilde{u}(x))^2 \, dx} \tag{3.12}$$

und der relative L_2-Interpolationsfehler durch

$$\frac{|u - \tilde{u}|_2}{|u|_2} := \sqrt{\frac{\int_\Omega (u(x) - \tilde{u}(x))^2 \, dx}{\int_\Omega (u(x))^2 \, dx}}, \tag{3.13}$$

wobei $u \in \mathcal{L}_2^m(\Omega)$ mit $\bar{\Omega} = [0,1]^d$, $m \in \mathbb{N}$ und $\tilde{u}$ der Interpolant zu u ist.
Der absolute L_∞-Interpolationsfehler wird wie folgt definiert:

$$|u - \tilde{u}|_\infty := \max_{x \in \bar{\Omega}} |u(x) - \tilde{u}(x)| \tag{3.14}$$

und der relative L_∞-Interpolationsfehler wie:

$$\frac{|u-\tilde{u}|_\infty}{|u|_\infty} := \frac{\max_{x\in\bar{\Omega}} |u(x)-\tilde{u}(x)|}{\max_{x\in\bar{\Omega}} |u(x)|}. \tag{3.15}$$

Das erste Problem, welches bei der Berechnung des Interpolationsfehlers auftritt, ist, dass das Integral in der L_2-Norm schwer zu berechnen ist und das Maximum des gesamten Gebiets schwer zu finden sein kann. Deswegen werden wir zu den Normen diskrete Pendants verwenden, die uns die Berechnung erheblich vereinfachen und die Möglichkeit geben auch zu nicht integrierbaren Funktionen den L_2-Interpolationsfehler berechnen zu können.

Definition 3.6 *Sei $\bar{\Omega} = [0,1]^d$, dann ist mit $\bar{\Omega}_n = \{x^1, \dots, x^n\}$ eine n-diskrete Menge von $\bar{\Omega}$ definiert, wobei $x^i \in \bar{\Omega}$ für $i = 1, \dots n$ und $n \in \mathbb{N}$ gilt.*

Außerdem können Funktionswerte der zu interpolierenden Funktion Null sein, weswegen wir nur den absoluten diskreten Interpolationsfehler betrachten werden.

Definition 3.7 *Sei $u : \bar{\Omega} \to \mathbb{R}^m$ mit $u = (u_1, \dots, m)$ und $u_i : \bar{\Omega} \to \mathbb{R}$ für $i = 1, \dots, m$, wobei $m \in \mathbb{N}$. Die Interpolante zu u sei gegeben durch $\tilde{u} = (\tilde{u_1}, \dots, \tilde{u_m})$. Dann definieren wir den diskreten L_∞-Interpolationsfehler durch*

$$|u-\tilde{u}|_\infty^{dis} := \max_{x\in\bar{\Omega}_n,\ i=1\dots m} |u_i(x)-\tilde{u_i}(x)| \tag{3.16}$$

und den diskreten L_2-Interpolationsfehler wie folgt:

$$|u-\tilde{u}|_2^{dis} := \sqrt{\frac{1}{n\cdot m}\sum_{x\in\bar{\Omega}_n}\sum_{i=1}^{m} |u_i(x)-\tilde{u_i}(x)|^2}. \tag{3.17}$$

Dabei seien $\bar{\Omega}$ und $\bar{\Omega}_n$ wie in Definition 3.6 gegeben und $n \in \mathbb{N}$.

Der L_∞-Fehler gibt uns das globale Maximum auf dem diskreten Gebiet $\bar{\Omega}_n$, was durchaus das globale Maximum auf dem kontinuierlichen Gebiet $\bar{\Omega}$ sein kann, aber zumindest eine Näherung daran ist. Da sich der kontinuierliche L_2-Interpolationsfehler auch mit Hilfe einer Summe über alle Punkte des Gebiets $\bar{\Omega}$ berechnen lässt

$$|u-\tilde{u}|_2 := \sqrt{\int_\Omega (u(x)-\tilde{u}(x))^2\,dx} = \sqrt{\frac{1}{k}\sum_{i=0}^{k} |u(x_k)-\tilde{u}(x_k)|^2} \quad \text{mit } x_k \in \Omega,$$

kann durch abschneiden der Summe eine Näherung erreicht werden, also durch Ersetzung des Gebiets $\bar{\Omega}$ durch $\bar{\Omega}_n$.
Als nächstes beschäftigen wir uns mit der Wahl der Menge $\bar{\Omega}_n$. Durch sie können wir die Genauigkeit beeinflussen. Das Ziel bei der Berechnung des Interpolationsfehlers ist ein möglichst gutes Verhältnis von Zeit zu Genauigkeit zu erhalten. Wir betrachten dazu drei verschiedene Möglichkeiten die Menge $\bar{\Omega}_n$zu wählen.

1. **Fehlerberechnung mit zufälligen Punkten (FZ)**
   Sei $\bar{\Omega}_n$ bei dieser Variante eine Menge mit zufällig gewählten Punkten. Wir erhalten, solange dasselbe n benutzt wird, vergleichbare Ergebnisse bei unterschiedlichen Leveln des Gitters und bei unterschiedlichen Gittertypen. Diese Variante wird als unabhängige Methode bezeichnet, da die zufällig gewählten Punkte nicht von der Wahl des Gitters oder des Levels abhängen. Bei singulären Funktionen kann es vorkommen, dass nicht jeder Punkt eingesetzt werden kann, da zu manchen Punkten der Funktionswert nicht definiert ist.

2. **Fehlerberechnung mit Dünngitter-Punkten (FD)**
   Bei dieser Methode merken wir uns das höchste Level $maxl$ unseres Gitters und setzen $\bar{\Omega}_n$ mit den neuen Gitterpunkten aus dem dünnen Gitter mit dem Level $maxl + k$ (mit $k > 0$ und $k \in \mathbb{N}$) gleich. n gibt nun die Anzahl der neuen Gitterpunkte an. Die neuen Gitterpunkte sind die Anzahl der Punkte des Gitters mit Level l ohne die Punkte des Levels $l - 1$. Das heißt formal ausgedrückt:

   $$\left|G_{neu}^{m,d,D}\right| = \left|G^{m,d,D}\right| - \left|G^{m-1,d,D}\right|$$

   mit $m = l + d - 1$.
   Man kann aber auch die neuen Gitterpunkte zweifach auswählen, d.h. man nimmt die Punkte des Levels l ohne die Punkte des Levels $l-2$. Das zweifache Vorgehen erhöht zwar die Anzahl der Punkte der Menge $\bar{\Omega}_n$, kann aber auch die Genauigkeit des berechneten Fehlers zum wirklichen Fehler erhöhen.

3. **Fehlerberechnung mit Produktgitter-Punkten (FP)**
   Für diese Methode erzeugen wir ein Produktgitter und setzen dieses mit der Menge $\bar{\Omega}_n$ gleich. Das Produktgitter sollte, wenn möglich, die Punkte des Gitters, welches wir zur Interpolation benutzen, nicht treffen, da die Stützstellen bei der Interpolation exakt wiedergegeben werden und nur alle anderen Stellen des Interpolanten etwas über den Fehler aussagen können.
   Hätten wir zur Interpolation ein Produktgitter mit Schrittweite 2^{-l} mit $l \in \mathbb{N}$, dann würde ein Produktgitter mit Schrittweite 3^{-L} mit $L \in \mathbb{N}$ zur Berechnung des Interpolationsfehlers unsere Bedingung gut erfüllen. Außerdem sollte das maximale Level des Produktgitters zur Berechnung des Interpolationsfehlers so groß wie möglich gewählt werden, um ein genaues Ergebnis zu erhalten. Je größer die Anzahl der Punkte der Menge $\bar{\Omega}$, desto näher liegt der berechnete Interpolationsfehler am tatsächlichen Interpolationsfehler. Diese Methode ist auch eine unabhängige Methode. Aber die Punkte der Menge $\bar{\Omega}_n$ können so gewählt werden, dass die Funktionswerte im Gegensatz zur FZ-Methode dazu existieren.

Numerische Berechnung des "realen" L_2-Fehlers (L2)

Damit wir die Genauigkeit der Methoden FZ, FD und FP beurteilen können, benötigen wir einen Vergleichswert, der ungefähr dem kontinuierlichen L_2-Interpolationsfehler entspricht. Die L2-Methode liefert uns diesen gewünschten Wert.
Sei nun o.B.d.A. $\tilde{u}$ der Interpolant zu u auf einem dünnen Gitter mit maximalen Level l_{max},

Dimension d und $m = l_{max} + d - 1$. Sei z_0 die Anzahl der $l_k = 0$ von $\mathbf{l}$, z_0^1 die Anzahl der $l_{1,k} = 0$ von $\mathbf{l_1}$ und z_0^2 die Anzahl der $l_{2,k} = 0$ von $\mathbf{l_2}$. Dann ergibt sich der Integralterm unter der Wurzel des L_2-Interpolationsfehlers aus (3.12) zu

$$
\begin{aligned}
&\int_\Omega \left(u(x) - \tilde{u}(x)\right)^2 dx \\
&= \int_\Omega \left(u(x) - \sum_{|\mathbf{l}|_1 - z_0 \le m} \sum_{\mathbf{i} \in \mathbf{I_l}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(x) \right)^2 dx \\
&= \underbrace{\int_\Omega \left(u(x)\right)^2 - 2u(x) \sum_{|\mathbf{l}|_1 - z_0 \le m} \sum_{\mathbf{i} \in \mathbf{I_l}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(x) + \left(\sum_{|\mathbf{l}|_1 - z_0 \le m} \sum_{\mathbf{i} \in \mathbf{I_l}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(x) \right)^2 dx}_{(*)} \\
&= \int_\Omega \left(u(x)\right)^2 - 2u(x) \sum_{|\mathbf{l}|_1 - z_0 \le m} \sum_{\mathbf{i} \in \mathbf{I_l}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(x) + \\
&\quad + \sum_{|\mathbf{l_1}|_1 - z_0^1 \le m} \sum_{\mathbf{i_1} \in \mathbf{I_{l_1}}} \sum_{|\mathbf{l_2}|_1 - z_0^2 \le m} \sum_{\mathbf{i_2} \in \mathbf{I_{l_2}}} u_{\mathbf{l_1},\mathbf{i_1}} \cdot \phi_{\mathbf{l_1},\mathbf{i_1}}(x) \cdot u_{\mathbf{l_2},\mathbf{i_2}} \cdot \phi_{\mathbf{l_2},\mathbf{i_2}}(x) dx.
\end{aligned}
$$

Berechnen wir den L_2-Interpolationsfehler mit der Formel aus den letzten beiden Zeilen der vorigen Herleitung, erhalten wir einen Wert, der dem tatsächlichen Fehler sehr nahe kommt. Diese Methode ist sehr aufwendig und zudem muss die zu interpolierende Funktion quadratisch integrierbar sein. Es taucht sogar ein weiteres Problem auf, das Problem der Auslöschung. Die Summe (*) kann, wenn wir sie mit dem Computer berechnen, kleiner als Null werden, obwohl eigentlich ein Wert großer oder gleich Null erwartet wird. Das liegt daran, dass der Computer nur bis zur Maschinengenauigkeit genau rechnen kann. Das heißt, dass eine Summe kleiner Werte wichtig sein kann, aber bei der Berechnung nicht berücksichtigt wird, da ein kleiner Wert durch Addition zu einem großen Wert den großen Wert mit weniger als Maschinengenauigkeit verändert. Durch sinnvolles Umordnen der Summen der Integrale könnte dieses Problem etwas behoben werden, was in der Praxis aber hohe Kosten verursachen würde. Dieses Problem kann schon bei relativ kleinem Level auftreten. Außerdem können nur Rechnungen bis zu maximal drei Dimensionen, ohne dem Fluch der Dimensionen zum Opfer zu fallen, durchgeführt werden. Sie sind einfach zu zeitintensiv.

3.2.2 Berechnung der Konvergenzrate

Es gibt mehrere Methoden die Konvergenzrate des Interpolationsverfahrens in der Praxis zu berechnen. Wir werden uns drei verschiedene Möglichkeiten anschauen:

- Konvergenzgraphenmethode
- Steigungsmethode
- Quotientenmethode (siehe auch in [Sch98] Kapitel 7)

Bei einem nicht adaptiven Gitterverfahren werden wir die Interpolante mit der Originalfunktion pro Level l bis zum maximalen Level des Gitters und bei einem adaptiven Verfahren pro ϵ

bis zur ϵ-Schranke des adaptiven Gitters vergleichen. Mit ϵ bezeichnen wir den Wert anhand dessen sortiert wird. Das heißt, wenn der hierarchische Überschuss kleiner ist als ϵ, wird der dazugehörige Knoten nicht weiter verfeinert und das adaptive Gitter an dieser Stelle nicht erweitert. Desweiteren bezeichne P_β mit $\beta \in \{l, \epsilon\}$ die Anzahl der Punkte des jeweiligen Gitters. Somit ist eine Vergleichsmöglichkeit für das nicht adaptive Gitter mit dem adaptiven Gitter gegeben.
Bei den einzelnen Methoden sind mit u die zu interpolierende Funktion, mit u_l die auf einem nicht adaptiven Gitter zum Level l interpolierte Funktion, mit u_ϵ die auf einem adaptiven Gitter zur Adaptionsschranke ϵ interpolierte Funktion und mit $\hat{u}_\beta^\alpha = \|u - u_\beta\|_{L_\alpha}$ der zugehörige diskrete L_α-Interpolationsfehler mit $\alpha \in \{2, \infty\}$ und $\beta \in \{l, \epsilon\}$ gemeint.

Konvergenzgraphenmethode

In der Praxis sind Konvergenzgraphen sehr hilfreich, mit Hilfe derer man sofort erkennen kann, ob eine hohe oder eine niedrige Konvergenzrate vorliegt. Ein Konvergenzgraph ist ein Plot, bei dem der Interpolationsfehler gegen die Anzahl der Punkte in einem Loglog-Plot geplottet wird. Der Betrag der Steigung der Geraden, die man durch diese Punkte legen kann, ist die gesuchte Konvergenzrate. Um die Gerade zu errechnen, kann das Kriterium der kleinsten Quadrate verwendet werden.

Steigungsmethode

Leider gibt uns die Gerade, die man durch die Punkte des Konvergenzgraphen legen kann, nur einen Wert, die durchschnittliche Konvergenzrate. Der Verlauf der Konvergenz bleibt außen vor. Wenn wir aber die Steigung zwei aufeinander folgender Punkte berechnen und das für alle Punkte unseres Graphen tun, erhalten wir eine Liste von Konvergenzraten. Anhand dieser ist es leicht eine Tendenz und das Konvergenzverhalten abzulesen. Die Konvergenzrate q_α^β zum zugehörigen L_α-Interpolationsfehler mit $\alpha \in \{2, \infty\}$ und $\beta \in \{l, \epsilon\}$ errechnet man dann wie folgt:

$$q_\alpha^l = \frac{\ln(\hat{u}_{l-1}^\alpha) - \ln(\hat{u}_l^\alpha)}{\ln(P_l) - \ln(P_{l-1})} \text{ für das nicht adaptive und}$$

$$q_\alpha^\epsilon = \frac{\ln(\hat{u}_{\epsilon-\epsilon_{\text{Schritt}}}^\alpha) - \ln(\hat{u}_\epsilon^\alpha)}{\ln(P_\epsilon) - \ln(P_{\epsilon-\epsilon_{\text{Schritt}}})} \text{ für das adaptive Gitter.}$$

Als nächstes zeigen wir, wie diese Methode aus der Definition der Konvergenzrate (Def. 3.4) zustande kommt.
Dazu betrachten wir o.B.d.A. q_2^l. Nach Definition 3.4 muss $q_2^l = p$ gelten, d.h.:

$$\hat{u}_l^2 = c \cdot \left(\frac{1}{P_l}\right)^{q_2^l} \quad \forall\, l > L \in \mathbb{N}$$

Also muss für $l+1$ das c mit dem c für l übereinstimmen. Daraus erhalten wir:

$$\hat{u}_l^2 \cdot (P_l)^p = \hat{u}_{l+1}^2 \cdot (P_{l+1})^p$$

Nimmt man nun den Logarithmus davon

$$\begin{aligned} 0 = &\ \ln(\hat{u}_l^2 \cdot (P_l)^p) - \ln(\hat{u}_{l+1}^2 \cdot (P_{l+1})^p) \\ = &\ \ln(\hat{u}_l^2) + p \cdot \ln(P_l) - \ln(\hat{u}_{l+1}^2) - p \cdot \ln(P_{l+1}) \end{aligned}$$

so führt dies zu

$$\begin{aligned} p = &\ \frac{\ln(\hat{u}_l^2) - \ln(\hat{u}_{l+1}^2)}{\ln(P_{l+1}) - \ln(P_l)} \\ = &\ q_2^{l+1}. \end{aligned}$$

Die Herleitung zeigt, dass über die Steigung des Konvergenzgraphen oder über die Steigung zweier Punkte des Konvergenzgraphen die Konvergenzrate des Verfahrens gut berechnet werden kann. Da wir aber nicht immer einen sprunglosen Konvergenzgraphen haben, könnten auch negative Konvergenzraten auftreten, was die Interpretation der Konvergenz schwieriger macht.

Quotientenmethode

Wie wir in der Theorie schon gesehen haben, ist die Konvergenzrate ein Maß für Aufwand zu Genauigkeit eines Verfahrens. Deswegen kann die Quotientenmethode auch zur Interpretation der Konvergenzrate benutzt werden. Auch wenn sie nicht ganz genau die Konvergenzrate berechnet, so liegt sie in einigen Fällen ziemlich nahe an der exakten Konvergenzrate. Sie ist aber nur mit Bedacht anzuwenden, da sie nur für eine Konvergenzrate von ungefähr eins und für den Spezialfall $q_P = 4$ und $q_{L_\alpha} = 2$ oder $q_P = 2$ und $q_{L_\alpha} = 4$ funktioniert.
Betrachten wir die letzte Gleichung der vorigen Herleitung, so können wir daraus die Quotientenmethode konstruieren:

$$p = \frac{\ln(\hat{u}_l^2) - \ln(\hat{u}_{l+1}^2)}{\ln(P_{l+1}) - \ln(P_l)} = \frac{\ln(\frac{\hat{u}_l^2}{\hat{u}_{l+1}^2})}{\ln(\frac{P_{l+1}}{P_l})} \approx \frac{\left(\frac{\hat{u}_l^2}{\hat{u}_{l+1}^2}\right)}{\left(\frac{P_{l+1}}{P_l}\right)} \quad \text{, falls } \frac{\ln(\frac{\hat{u}_l^2}{\hat{u}_{l+1}^2})}{\ln(\frac{P_{l+1}}{P_l})} \approx 1$$

Wir berechnen also mit der Quotientenmethode einerseits den Fehlerquotienten

$$q_{L_\alpha}^l = \frac{\|u - u_{l-1}\|_{L_\alpha}}{\|u - u_l\|_{L_\alpha}} \text{ für das nicht adaptive Gitter und}$$

$$q_{L_\alpha}^\epsilon = \frac{\|u - u_{\epsilon - \epsilon\text{Schritt}}\|_\alpha}{\|u - u_\epsilon\|_\alpha} \text{ für das adaptive Gitter,}$$

und andererseits den Punktquotienten

$$q_P^l = \frac{P_l}{P_{l-1}} \text{ für das nicht adaptive Gitter und}$$

$$q_P^\epsilon = \frac{P_\epsilon}{P_{\epsilon - \epsilon_{\text{Schritt}}}} \text{ für das adaptive Gitter.}$$

Der Quotient aus Fehlerquotient und Punktquotient $\hat{q}_\alpha^\beta = \frac{q_{L_\alpha}^\beta}{q_P^\beta}$ definiert uns dann die approximative Konvergenzrate $\hat{q}_\alpha^\beta$.[1]

[1] Die Notation ist an [Sch98] angelehnt.

Die Quotientenmethode kann uns also sagen, dass, wenn die Anzahl der Punkte ver-q_P^β-facht wird, der Fehler ge-$q_{L_\alpha}^\beta$-telt wird.

3.3 Bewertung der Methoden

Wir werden jetzt unsere Methoden zur Berechnung des Interpolationsfehlers und der Konvergenzrate an Beispielen untersuchen, diskutieren und dann darüber entscheiden, welche der untersuchten Methoden die sinnvollsten Methoden zur Berechnung des Interpolationsfehlers und der Konvergenzrate sind. Diese Methoden werden wir dann in Kapitel 7 zur Bewertung der Interpolationsverfahren verwenden.

3.3.1 Fehlerberechnungsmethoden

Wir wollen in diesem Abschnitt den Fehler betrachten, den wir zusätzlich zum Interpolationsfehler mit den Fehlerberechnungsmethoden machen. Den errechneten "realen" L_2-Interpolationsfehler (L2-Methode) nehmen wir als Vergleichsmaß zu den anderen Fehlerberechnungsmethoden, wie die Fehlerberechnungsmethode mit Zufallspunkten (FZ-Methode), die Fehlerberechnungsmethode mit Dünngitter-Punkten (FD-Methode) und die Fehlerberechnungsmethode mit Produktgitter-Punkten (FP-Methode). Wir werden uns dazu zwei Beispiele ansehen, eine glatte zweidimensionale Funktion und eine zweidimensionale Funktion mit Polstelle, die wir auf einem regulären dünnen Gitter interpolieren.

Beispiel 1:
Sei in diesem Beispiel $f(x,y) = x \cdot (1-x) \cdot y \cdot (1-y)$ mit $(x,y) \in [0,1]^2$. Wir interpolieren diese glatte zweidimensionale Funktion auf einem regulären dünnen Gitter mit Level 0 bis 9. In der Abbildung 3.1 sind drei verschiedene Graphen zu sehen. Der erste Graph zeigt die Differenz zwischen dem L_2-Interpolationsfehler mit der L2-Methode berechnet und dem L_2-Interpolationsfehler mit den übrigen Fehlerberechnungsmethoden berechnet geplottet gegen das Level in einem Loglogplot. Dabei wurde für jede Fehlerberechnungsmethode eine ähnliche Anzahl an Punkten verwendet, für die FZ-Methode 60000 Punkte, für die FD-Methode 69632 Punkte (nur Punkte aus Level 13) und für die FP-Methode 59536 Punkte (Schrittweite 3^{-5}). In der rechten Graphik ist der L_2-Interpolationsfehler in einer logarithmischen Skala gegen die Zeit geplottet. Die zu den Graphiken benötigten Daten sind in den Tabellen 3.3 und 3.4 zu finden.
Betrachten wir zuerst die FD-Methode. Wir sehen im ersten Konvergenzgraph der Abbildung 3.1, dass die Genauigkeit der FD-Methode weit von den anderen beiden Methoden, FP-Methode und FZ-Methode, abweicht, obwohl für jede Fehlerberechnungsmethode ungefähr die gleiche Punktanzahl zur Berechnung des Interpolationsfehlers benutzt wurde. In der Tabelle 3.3 ist das noch besser zu sehen. Mit der FZ-Methode und der FP-Methode errechnen wir zur Funktion f fast denselben L_2-Interpolationsfehler, mit der FD-Methode immer einen besseren Interpolationsfehler als mit den anderen Methoden. Vor allem ist der L_2-Interpolationsfehler, berechnet mit der FD-, FZ- und der FP-Methode, immer besser als der L_2-Interpolationsfehler, den wir mit der L2-Methode erhalten.
Betrachten wir nun die Zeit die zur Berechnung des Interpolationsfehlers gebraucht wurde. In

Abbildung 3.1 sehen wir, dass die L2-Methode die meiste Zeit benötigt um den Interpolationsfehler zu berechnen. Für diese Beispielfunktion braucht die L2-Methode immer fünfmal so lang wie ein Level zuvor. Zum Level 9 benötigt die L2-Methode ganze 2 Minuten, um den Interpolationsfehler zu berechnen, obwohl wir hier nur eine zweidimensionale Funktion betrachten. Aus dem zeitlichen Gesichtspunkt wird klar, warum wir die L2-Methode nur als Vergleichswert zum Testen der anderen Methoden benutzen können. Abschließend können wir zusammenfassen, dass wir mit der FP-Methode und der FZ-Methode die besten Ergebnisse für dieses Beispiel erhalten.

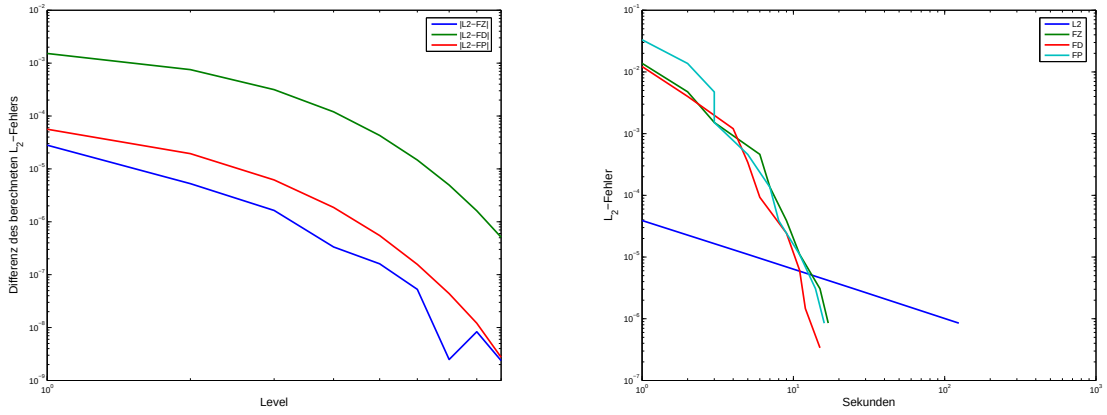


Abb. 3.1: Graphiken zum L_2-Interpolationsfehler der Funktion f. Die linke Graphik zeigt die Differenz zwischen dem L_2-Interpolationsfehler mit der L2-Methode berechnet und dem L_2-Interpolationsfehler mit den Fehlerberechnungsmethoden berechnet geplottet gegen das Level. Die rechte Graphik zeigt den L_2-Interpolationsfehler berechnet mit den einzelnen Fehlerberechnungsmethoden geplottet gegen die Zeit in Sekunden gemessen, die zur Berechnung nötig war.

Level	Fehler$_{L2}$	Fehler$_{FZ}$	Fehler$_{FD}$	Fehler$_{FP}$
0	3.333333e-02	3.339056e-02	3.117174e-02	3.319672e-02
1	1.374053e-02	1.376860e-02	1.222150e-02	1.368435e-02
2	4.759288e-03	4.764551e-03	4.007306e-03	4.739891e-03
3	1.518477e-03	1.516841e-03	1.203314e-03	1.512303e-03
4	4.615054e-04	4.611712e-04	3.415574e-04	4.596354e-04
5	1.358099e-04	1.356493e-04	9.291705e-05	1.352616e-04
6	3.905508e-05	3.900235e-05	2.435002e-05	3.889873e-05
7	1.103848e-05	1.103599e-05	6.093177e-06	1.099461e-05
8	3.078135e-06	3.069793e-06	1.463337e-06	3.066051e-06
9	8.485734e-07	8.509440e-07	3.378097e-07	8.458304e-07

Tabelle 3.3: Die L_2-Interpolationsfehler zur Funktion f

Level	Zeit_{L2}	Zeit_{FZ}	Zeit_{FD}	Zeit_{FP}
0	0	0	0	1
1	0	1	1	2
2	0	2	2	3
3	0	3	4	3
4	0	6	5	5
5	0	7	6	7
6	1	9	9	8
7	5	11	11	11
8	25	15	12	14
9	124	17	15	16

Tabelle 3.4: Die benötigten Sekunden für die Berechnung der L_2-Interpolationsfehler

Beispiel 2:
Im zweiten Beispiel betrachten wir eine singuläre Funktion, um die Ergebnisstabilität der Methoden zu testen. Sei $g(x,y) = \left|x - y - \frac{1}{3}\right|^{-\frac{1}{3}}$ mit $(x,y) \in [0,1]^2$. Wir interpolieren die Funktion g auf einem regulären dünnen Gitter mit Level 0 bis 10.
Wir wenden jede Methode zur Berechnung des Interpolationsfehlers mit unterschiedlichen Anzahlen an Punkten an. Die FD-Methode wenden wir einmal mit Dünngitter-Punkten des Levels 11, einmal mit Punkten der Level 11 und 12, einmal mit Punkten des Levels 12, einmal mit Punkten der Level 12 und 13 und einmal mit Dünngitter-Punkten des Levels 15 an. Die FZ-Methode wenden wir einmal mit 10^4 zufälligen Punkten, einmal mit 10^5 zufälligen Punkten und einmal mit $3.9 \cdot 10^5$ zufälligen Punkten an. Für die FP-Methode wählen wir abschließend einmal die Schrittweite 5^{-2}, einmal 5^{-3} und einmal 5^{-4}.
In der Abbildung 3.2 sind vier verschiedene Konvergenzgraphen zu sehen. Dabei ist in jeder Graphik der L_2-Interpolationsfehler gegen die Anzahl der Punkte des Gitters in einem Loglogplot geplottet. Der Konvergenzgraph links oben zeigt die besten Ergebnisse der Fehlerberechnungsmethoden im Vergleich zur L2-Methode. Dabei wurde für jede Fehlerberechnungsmethode eine ähnliche Anzahl an Punkten verwendet, für die FZ-Methode 390000 Punkte, für die FD-Methode 311296 Punkte (nur Punkte aus Level 15) und für die FP-Methode 391876 Punkte (Schrittweite 5^{-4}). In der oberen rechten Graphik ist nur die FZ-Methode im Vergleich zur L2-Methode mit unterschiedlichen Anzahlen von Punkten verwendet worden. Die untere linke Graphik zeigt die FP-Methode im Vergleich zur L2-Methode und die letzte Graphik die FD-Methode im Vergleich zur L2-Methode. Die dazu gehörigen Daten sind in den Tabellen 3.5 bis 3.8 zu finden.
Selbst bei Funktionen mit Singularitäten sind die Fehlerberechnungsmethoden eine gute Alternative zur L2-Methode, wie wir dem Konvergenzgraphen links oben der Abbildung 3.2 entnehmen können. Wir erkennen, dass der Interpolationsfehler, der mit der FZ-Methode berechnet wurde, das genauste Ergebnis ist. Etwas schlechter fällt das Ergebnis der FP-Methode aus und das schlechteste Ergebnis erhalten wir mit der FD-Methode.
In den Graphiken rechts oben, links unten und rechts unten der Abbildung 3.2 erkennt man, dass sich die L_2-Interpolationsfehler, berechnet mit allen drei Methoden, wenn man mehr Punkte benutzt, dem tatsächlichen L_2-Interpolationsfehler annähern. Dabei passt sich der mit der FP-Methode berechnete Interpolationsfehler dem Verlauf des wirklichen Interpolationsfehlers

am Besten an. Die FP-Methode ist aber nicht so genau wie die FZ-Methode. Die Ergebnisse der FZ-Methode hingegen oszillieren um den wahren Interpolationsfehler. Die mit der FD-Methode berechneten Interpolationsfehler entfernen sich mit steigendem Level immer mehr vom mit der L2-Methode berechneten Interpolationsfehler.
Betrachten wir desweiteren die Tabellen 3.5 bis 3.8, so sehen wir, dass das Ergebnis der FD-Methode und der FP-Methode im Vergleich zum Ergebnis der L2-Methode immer kleiner als das Ergebnis der L2-Methode ist und somit eine untere Schranke für den Interpolationsfehler darstellt. Die Ergebnisse der FZ-Methode sind mal größer und mal kleiner als die Ergebnisse der L2-Methode.
So kann man zusammenfassend sagen, dass die FZ-Methode die genaueste Approximation des Interpolationsfehlers liefert, aber die FP-Methode die ergebnisstabilste Methode ist.

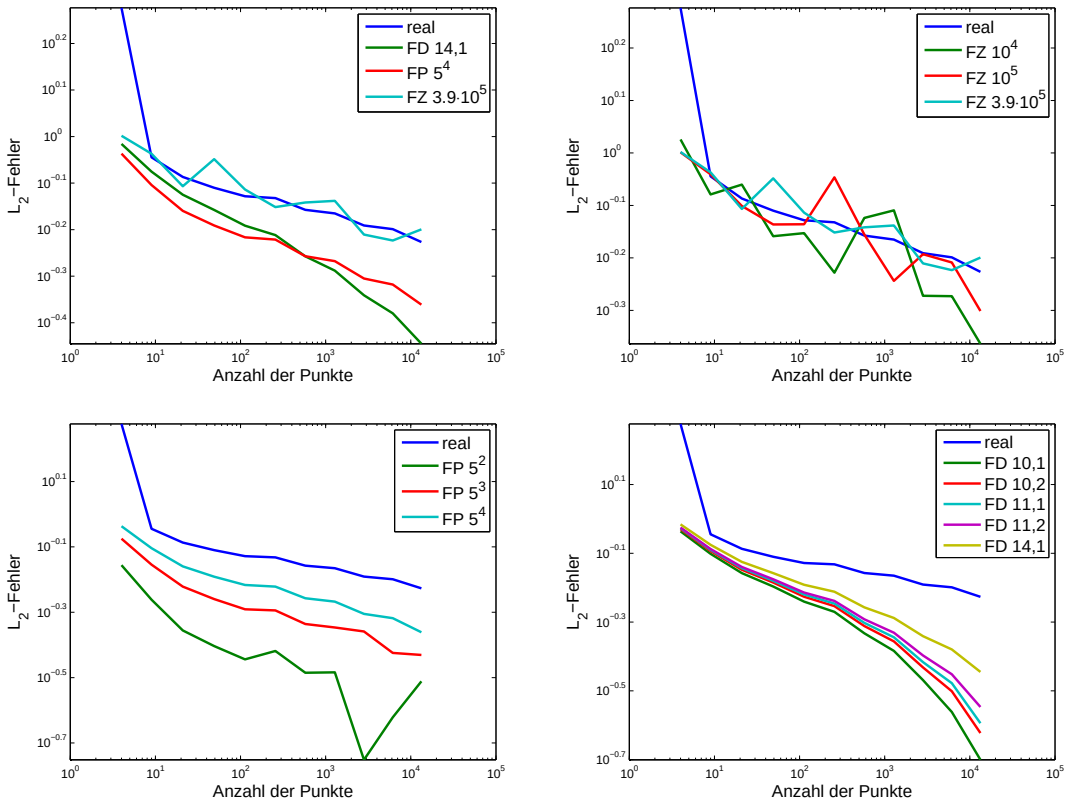


Abb. 3.2: In dieser Abbildung sind vier Konvergenzgraphiken zum L_2-Interpolationsfehler der Funktion $g(x,y) = \left|x - y - \frac{1}{3}\right|^{-\frac{1}{3}}$ mit $(x,y) \in [0,1]^2$. Dabei zeigt der Graph links oben die besten Ergebnisse der einzelnen Fehlerberechnungsmethoden und die Graphik rechts oben die Ergebnisse der FZ-Methode im Vergleich zu den Ergebnissen der L2-Methode. Im Graph links unten sind die Ergebnisse der FP-Methode im Vergleich zu denen der L2-Methode geplottet und rechts unten die Ergebnisse der FD-Methode im Vergleich zu denen der L2-Methode.

Level	Punkte	L2
0	4	1.890041e+00
1	9	9.024434e-01
2	21	8.187551e-01
3	49	7.762658e-01
4	113	7.442482e-01
5	257	7.375874e-01
6	577	6.959761e-01
7	1281	6.837339e-01
8	2817	6.441952e-01
9	6145	6.323529e-01
10	13313	5.933650e-01

Tabelle 3.5: Der L_2-Interpolationsfehler zur Funktion g mit der L2-Methode ausgerechnet

Level	Punkte	FD(10,1)	FD(10,2)	FD(11,1)	FD(11,2)	FD(14,1)
0	4	9.201202e-01	9.335698e-01	9.398081e-01	9.438582e-01	9.642626e-01
1	9	7.914269e-01	8.062586e-01	8.131178e-01	8.177901e-01	8.405427e-01
2	21	6.957022e-01	7.117018e-01	7.190790e-01	7.245661e-01	7.498112e-01
3	49	6.358766e-01	6.530552e-01	6.609540e-01	6.674791e-01	6.955393e-01
4	113	5.748014e-01	5.939840e-01	6.027658e-01	6.109024e-01	6.433950e-01
5	257	5.365023e-01	5.577453e-01	5.674293e-01	5.772442e-01	6.142936e-01
6	577	4.646582e-01	4.879948e-01	4.985579e-01	5.105443e-01	5.528998e-01
7	1281	4.133304e-01	4.403275e-01	4.524282e-01	4.666757e-01	5.151953e-01
8	2817	3.397794e-01	3.698316e-01	3.831078e-01	4.007693e-01	4.559996e-01
9	6145	2.744093e-01	3.153826e-01	3.328572e-01	3.532154e-01	4.170507e-01
10	13313	1.992698e-01	2.382708e-01	2.545034e-01	2.838572e-01	3.587474e-01

Tabelle 3.6: Der L_2-Interpolationsfehler zur Funktion g mit der FD-Methode ausgerechnet, dabei bedeutet FD(a,1): [berechnet mit neuen Dünngitter-Punkten des Levels a+1] und FD(a,2): [berechnet mit neuen Dünngitter-Punkten der Level a+1 und a+2]

Level	Punkte	FZ(10^4)	FZ(10^5)	FZ($3.9 \cdot 10^5$)
0	4	1.061289e+00	1.002620e+00	1.004106e+00
1	9	8.335300e-01	9.105048e-01	9.184229e-01
2	21	8.704649e-01	7.926169e-01	7.820494e-01
3	49	6.938123e-01	7.306970e-01	8.943534e-01
4	113	7.033328e-01	7.313239e-01	7.696108e-01
5	257	5.913868e-01	8.985860e-01	7.053762e-01
6	577	7.518427e-01	6.988286e-01	7.216521e-01
7	1281	7.771744e-01	5.704841e-01	7.274656e-01
8	2817	5.343233e-01	6.414305e-01	6.158415e-01
9	6145	5.334808e-01	6.182966e-01	5.979436e-01
10	13313	4.328846e-01	4.998002e-01	6.318286e-01

Tabelle 3.8: Der L_2-Interpolationsfehler zur Funktion g mit der FZ-Methode ausgerechnet, dabei bedeutet FZ(n): [berechnet mit n zufälligen Punkten]

Level	Punkte	FP($\frac{1}{5^2}$)	FP($\frac{1}{5^3}$)	FP($\frac{1}{5^4}$)
0	4	6.985177e-01	8.417098e-01	9.188987e-01
1	9	5.475069e-01	7.012765e-01	7.869544e-01
2	21	4.409998e-01	6.004281e-01	6.923400e-01
3	49	3.950206e-01	5.499599e-01	6.440203e-01
4	113	3.595834e-01	5.119662e-01	6.073428e-01
5	257	3.815128e-01	5.076136e-01	6.007077e-01
6	577	3.272587e-01	4.612801e-01	5.532661e-01
7	1281	3.282248e-01	4.503169e-01	5.400022e-01
8	2817	1.772759e-01	4.378904e-01	4.954450e-01
9	6145	2.390142e-01	3.764405e-01	4.807806e-01
10	13313	3.081682e-01	3.709977e-01	4.353523e-01

Tabelle 3.7: Der L_2-Interpolationsfehler zur Funktion g mit der FP-Methode ausgerechnet, dabei bedeutet FP(h): [berechnet mit einem Produktgitter mit Schrittweite h]

Fazit

Wir haben anhand zweier Beispiele gesehen, wie gut der L_2-Interpolationsfehler bei der Interpolation einer glatten und einer singulären Funktion durch die Fehlerberechnungsmethoden approximiert wird. Auf Grund der gefundenen Resultate gelangen wir zu der Auffassung, dass der L_2-Interpolationsfehler durch die Fehlerberechnungsmethoden, wie die FD-Methode, die FP-Methode und die FZ-Methode, gut approximiert werden kann. Dabei ist die Methode mit Produktgitter-Punkten (FP-Methode) zu bevorzugen, da sie genauer als die FD-Methode ist und stabilere Ergebnisse liefert als die FZ-Methode. Die Methode mit Produktgitter-Punkten liefert sogar ordentliche Resultate bei singulären Funktionen. Zudem kommt hinzu, dass der mit der FP-Methode berechnete Interpolationsfehler immer kleiner als der tatsächliche Interpolationsfehler ist. Die FP-Methode liefert also eine untere Schranke für den Interpolationsfehler. Die Fehlerberechnungsmethode mit zufälligen Punkten (FZ-Methode) ist fast genauso gut, aber der mit ihr berechnete Interpolationsfehler oszilliert in einigen Fällen sehr stark um den tatsächlichen Interpolationsfehler herum. Dadurch können wir mit der FZ-Methode keine Aussage über den tatsächlichen Interpolationsfehler machen, da wir nicht wissen, ob der tatsächliche Interpolationsfehler kleiner oder größer als der berechnete Interpolationsfehler ist. Da die Punkte zur Berechnung des Interpolationsfehlers bei der FZ-Methode zufällig ausgesucht werden, kann es vorkommen, dass der Funktionswert zu ihnen nicht bestimmt werden kann. Zum Beispiel kann ein Nenner Null werden. Dieses Problem lässt sich durch vorherige Auswahl beheben, indem nur Punkte des Gebiets $\bar{\Omega}$ zugelassen werden, die eingesetzt in die Funktion einen Funktionswert liefern. Das ist aber relativ aufwendig. Mit der FP-Methode können wir solche Punkte umgehen, indem wir eine Schrittweite des Produktgitters wählen, so dass diese Punkte nicht erzeugt werden. Bei der FZ-Methode kommt außerdem noch hinzu, dass bei jeder Anwendung immer eine unterschiedliche Menge an Punkten eingesetzt wird, so dass die FZ-Methode für dieselbe Funktion und denselben Interpolanten natürlich nie das gleiche Ergebnis liefern kann. Bei der Bewertung der Punktanzahl des adaptiven Gitters gegenüber dem nicht adaptiven Gitters ist das aber von großem Interesse. Man kann nur am gleichen Interpola-

tionsfehler die zur Interpolation wirklich benötigten Punkte erkennen, da bei der Interpolation auf einem adaptiven Gitter mit der Senkung der ϵ-Schranke der dazu berechnete Interpolationsfehler gegen den Interpolationsfehler des nicht adaptiven Gitterverfahrens konvergiert.
Die FD-Methode ist in der Genauigkeit schlechter als die FP-Methode und die FZ-Methode, da der mit der FD-Methode berechnete Interpolationsfehler den größten Abstand zum tatsächlichen Interpolationsfehler aufweist. Der mit der FD-Methode berechnete Interpoolationsfehler ist aber genau wie der mit der FP-Methode berechnete Interpolationsfehler eine untere Schranke für den tatsächlichen Interpolationsfehler. Zudem liegt dieser Methode die Struktur eines dünnen Gitters zugrunde. Das kann in besonderen Fällen von Vorteil sein, da die von der Methode eingesetzten Punkte immer zwischen den Stützstellen des Dünngitter-Interpolanten liegen und somit jeder Fehler, der bei der Interpolation gemacht wird, berücksichtigt wird.
Der Vorteil der Gitterpunktmethoden liegt darin, dass der tatsächliche Interpolationsfehler nicht besser sein kann als die mit den Gitterpunktmethoden (FP-Methode und FD-Methode) errechneten Interpolationsfehler. Deswegen eignen sich diese Methoden besser zur Bewertung der Interpolation auf adaptiven Gittern im Gegensatz zur FZ-Methode. Die FZ-Methode ist zwar die genaueste Methode im Vergleich zur FP-Methode und FD-Methode, aber sie berechnet kein eindeutiges Ergebnis.
Da wir im Vergleich zu regulären Dünngitter-Interpolationen aber adaptive Dünngitter-Interpolationen und ihre zugehörigen Interpolanten beurteilen möchten, ist die sinnvollste Wahl zur Berechnung des Interpolationsfehlers die FP-Methode. Wenn die FP-Methode keine ordentlichen Ergebnisse liefern kann, so werden wir die FD-Methode zur Berechnung des Interpolationsfehlers verwenden.

3.3.2 Methoden zur Berechnung der Konvergenzrate

Zum Abschluss betrachten wir ein Beispiel, an dem wir die gerade besprochenen Methoden zur Berechnung der Konvergenzrate erproben und veranschaulichen können. Sei dazu $f(x, y) = x^2 + y^2$ mit $(x, y) \in [0, 1]^2$ eine glatte Funktion (siehe Abb. 3.3). Auf Grund der Glattheit erwarten wir gute Konvergenzraten, d.h. beim regulären Dünngitter-Interpolationsverfahren und beim adaptiven Dünngitter-Interpolationsverfahren müssten sie beim Wert zwei liegen.
Zuerst berechnen wir den L_2-Interpolationsfehler und den L_∞-Interpolationsfehler für das reguläre Dünngitter-Verfahren, sowie für das adaptive Dünngitter-Verfahren. Die Ergebnisse sind in der Tabelle 3.9 zusammengefasst.
Den Konvergenzgraphen zur Funktion f erhalten wir, wenn wir den Interpolationsfehler gegen die Anzahl der benötigten Punkte mit einer logarithmischen Skala plotten (siehe Abb. 3.4). Wie man erkennen kann, liegt der Betrag der Steigung der Geraden bei zwei.
Um genaue Ergebnisse zu erhalten benutzen wir die Software Matlab mit seinen eingebauten Funktionen *polyfit(log(x),log(y),n)* und *robustfit(log(x),log(y))*, die die Steigung der durch die Punkte gelegten Geraden berechnen. Die Variable x repräsentiert dabei den Punktanzahlvektor und die Variable y den Interpolationsfehlervektor. Da wir eine Gerade anpassen wollen, müssen wir die Variable n=1 wählen. Die Ergebnisse für dieses Beispiel sind in Tabelle 3.10 aufgeführt. Wie wir der Tabelle 3.10 entnehmen können, liefert die Matlabfunktion *robustfit* stabilere Ergebnisse im Gegensatz zur Matlabfunktion *polyfit*. Im Kapitel 7 werden wir die Matlabfunktion *robustfit* deswegen noch des öfteren bei der Berechnung der Konvergenzrate verwenden.

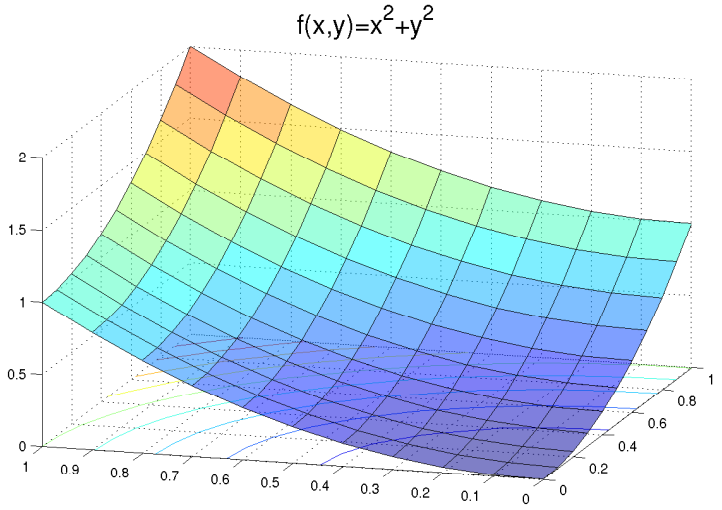


Abb. 3.3: Funktionsplot zur Beispielfunktion f

P_{DG}	L_2^{DG}-Fehler	L_∞^{DG}-Fehler	P_{ADG}	L_2^{ADG}-Fehler	L_∞^{ADG}-Fehler
1	7.806492e-01	1.982262e+00			
3	3.480529e-01	4.999901e-01			
6	8.701323e-02	1.249975e-01	6	8.701323e-02	1.249975e-01
12	2.175331e-02	3.124938e-02	10	2.175331e-02	3.124938e-02
25	5.438327e-03	7.812346e-03	18	5.438327e-03	7.812346e-03
53	1.359582e-03	1.953086e-03	34	1.359582e-03	1.953086e-03
113	3.398954e-04	4.882716e-04	66	3.398954e-04	4.882716e-04
241	8.497385e-05	1.220679e-04	130	8.497385e-05	1.220679e-04
513	2.124346e-05	3.051698e-05	258	2.124346e-05	3.051698e-05
1089	5.310866e-06	7.629244e-06	514	5.310866e-06	7.629244e-06
2305	1.327716e-06	1.907311e-06	1026	1.327716e-06	1.907311e-06
4865	3.319291e-07	4.768277e-07	2050	3.319291e-07	4.768277e-07

Tabelle 3.9: Die Tabelle zeigt die L_2-Interpolationsfehler, die L_∞-Interpolationsfehler und die dazu benötigte Anzahl an Gitterpunkten (P) für das reguläre Dünngitter-Verfahren (DG) und für das adaptive Dünngitter-Verfahren (ADG).

Die Tabelle 3.10 zeigt zudem, dass die Konvergenzrate des adaptiven Dünngitter-Verfahres höher als die Konvergenzrate des regulären Verfahrens ist. Zudem liegen die Konvergenzraten, wie erwartet, ungefähr bei zwei. Wollen wir aber ins Detail des Verlaufs der Konvergenzrate gehen, müssen wir uns noch die Werte, berechnet mit der Quotientenmethode als auch mit der Steigungsmethode, anschauen.
In der Tabelle 3.11 sind die Konvergenzraten aufgelistet, die mit der Steigungsmethode errechnet wurden. Wie wir in der Tabelle 3.11 sehen, ist die Tendenz der Konvergenzrate beim adaptiven Dünngitter-Verfahren abnehmend und beim regulären Dünngitter-Verfahren zuneh-

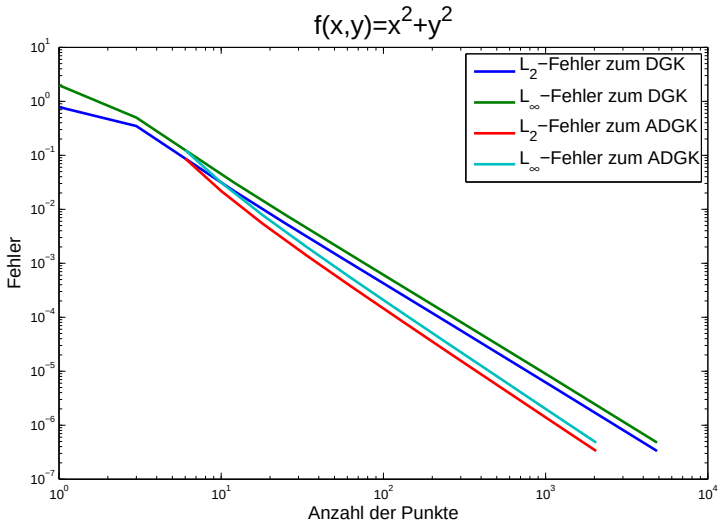


Abb. 3.4: Konvergenzgraph zur Beispielfunktion f

Matlab-Funktion	DG_{L_2}	DG_{L_∞}	ADG_{L_2}	ADG_{L_∞}
robustfit	1.84	1.84	2.09	2.09
polyfit	1.80	1.83	2.11	2.11
polyfit ohne die ersten zwei Ergebnisse der Tabelle 3.9	1.85	1.85		
robustfit ohne die ersten zwei Ergebnisse der Tabelle 3.9	1.84	1.84		

Tabelle 3.10: Konvergenzraten zu den regulären Dünngitter-Verfahren (DG) und zum adaptiven Dünngitter-Verfahren (ADG) berechnet mit den Matlabfunktionen *polyfit* und *robustfit* über den L_2-Interpolationsfehler und den L_∞-Interpolationsfehler

mend. Das heißt, dass die Punkte, die die Interpolation wesentlich beeinflussen, beim adaptiven Dünngitter-Interpolationsverfahren bei großen ϵ-Schranken und beim regulären Dünngitter-Interpolationsverfahren erst bei hohen Leveln gefunden werden.
In den Tabellen 3.12 und 3.13 sind die mit der Quotientenmethode gewonnenen Ergebnisse aufgelistet. Der Tabelle 3.13 können wir entnehmen, dass der Interpolationsfehler zu beiden Verfahren geviertelt wird, wenn die Punktanzahl verdoppelt wird. Die errechneten Konvergenzraten der Quotientenmethode weichen leicht von den Konvergenzraten der Steigungsmethode ab, wobei die mit der Steigungsmethode ermittelten Konvergenzraten mehr mit der mit der Matlabfunktion *robustfit* ermittelten Konvergenzrate übereinstimmen als die mit der Quotientenmethode ermittelten Konvergenzraten. Die Tendenz des Verlaufs der Konvergenzraten ist aber bei beiden Methoden gleich gut zu erkennen.

DG_{L_2}	DG_{L_∞}	ADG_{L_2}	ADG_{L_∞}
7.352652e-01	1.253769e+00		
2.000000e+00	2.000000e+00		
2.000000e+00	2.000000e+00	2.713831e+00	2.713831e+00
1.888764e+00	1.888764e+00	2.358499e+00	2.358499e+00
1.844909e+00	1.844909e+00	2.179746e+00	2.179747e+00
1.831069e+00	1.831068e+00	2.090015e+00	2.090014e+00
1.830311e+00	1.830311e+00	2.045045e+00	2.045045e+00
1.834988e+00	1.834987e+00	2.022532e+00	2.022532e+00
1.841666e+00	1.841666e+00	2.011268e+00	2.011269e+00
1.848835e+00	1.848834e+00	2.005635e+00	2.005635e+00
1.855850e+00	1.855851e+00	2.002817e+00	2.002818e+00

Tabelle 3.11: Die Tabelle zeigt Ergebnisse der Steigungsmethodee für das adaptive Dünngitter-Verfahren (ADG) und für das reguläre Dünngitter-Verfahren.

q_P DG	q_{L_2} DG	q_2 DG	q_{L_∞} DG	q_∞ DG
3.00	2.24	0.75	3.96	1.32
2.00	4.00	2.00	4.00	2.00
2.00	4.00	2.00	4.00	2.00
2.08	4.00	1.92	4.00	1.92
2.12	4.00	1.89	4.00	1.89
2.13	4.00	1.88	4.00	1.88
2.13	4.00	1.88	4.00	1.88
2.13	4.00	1.88	4.00	1.88
2.12	4.00	1.88	4.00	1.88
2.12	4.00	1.89	4.00	1.89
2.11	4.00	1.90	4.00	1.90

Tabelle 3.12: Die Tabelle zeigt Ergebnisse der Quotientenmethode für das reguläre Dünngitter-Verfahren.

q_P ADG	q_{L_2} ADG	q_2 ADG	q_{L_∞} ADG	q_∞ ADG
1.67	4.00	2.40	4.00	2.40
1.80	4.00	2.22	4.00	2.22
1.89	4.00	2.12	4.00	2.12
1.94	4.00	2.06	4.00	2.06
1.97	4.00	2.03	4.00	2.03
1.98	4.00	2.02	4.00	2.02
1.99	4.00	2.01	4.00	2.01
2.00	4.00	2.00	4.00	2.00
2.00	4.00	2.00	4.00	2.00

Tabelle 3.13: Die Tabelle zeigt Ergebnisse der Quotientenmethode für das adaptive Dünngitter-Verfahren.

Fazit

Wie erwartet liegt die Konvergenzrate des adaptiven Dünngitter-Interpolationsverfahren und des regulären Dünngitter-Interpolationsverfahren für die Beispielfunktion $f(x, y) = x^2 + y^2$ bei zwei. Dabei haben wir gesehen, dass mit Hilfe der Matlabfunktion *robustfit* die Konvergenzrate eines Verfahrens gut berechnet werden kann.
Möchten wir den Verlauf der Konvergenzrate studieren, eignet sich die Steigungsmethode eher als die Quotientenmethode, da die Quotientenmethode nur näherungsweise die Konvergenzrate berechnen kann.
Wollen wir aber mehr über das Verhältnis der Anzahl der Punkte zum Interpolationsfehler des Interpolationsverfahrens wissen, so können der Punktquotient und der Interpolationsfehlerquotient der Quotientenmethode hilfreich sein.
Mit Hilfe des Konvergenzgraphen können wir uns schnell ein Bild vom Konvergenzverhalten und von der Konvergenzrate machen.
Wir können abschließend leider nicht feststellen, ob eine der Methoden zur Berechnung der Konvergenzrate zu bevorzugen ist, da alle Methoden ihre Vor- und Nachteile haben. Jede der drei vorgestellten Methoden kann in unterschiedlichen Bereichen der Bewertung der Konvergenz sinnvoll eingestzt werden. Leider werden wir in Kapitel 7 noch Funktionen kennenlernen, bei deren Interpolation die Konvergenzrate des benutzten Verfahrens entweder nicht zu berechnen oder nicht interpretierbar ist.

4 Dünngitter-Algebra

Bei numerischen Verfahren kann es beim Lösungsprozess vorkommen, dass zwei Funktionen, die auf adaptiven dünnen Gittern interpoliert worden sind, miteinander durch einfache Operatoren, wie Addition oder Multiplikation, verknüpft werden müssen.
Betrachten wir dazu zwei Funktionen u_1 und u_2 und ihre zugehörigen adaptiven Dünngitter-Interpolanten $\hat{u}_1^{\epsilon_1}$ und $\hat{u}_2^{\epsilon_2}$, wobei ϵ_1 und ϵ_2 die benötigten ϵ-Schranken der zu den Interpolanten gehörigen adaptiven dünnen Gitter sind. Es stellt sich die Frage, inwieweit der durch die Operation neugewonnene adaptive Dünngiter-Interpolant $\hat{u}_3^{\epsilon_3}$ von den anderen beiden adaptiven Dünngitter-Interpolanten $\hat{u}_1^{\epsilon_1}$ und $\hat{u}_2^{\epsilon_2}$ abhängt, wie sich dies auf den gemachten Fehler zur Funktion u_3 auswirkt und wie die zugehörige ϵ-Schranke ϵ_3 aussieht. Der Umstand, dass wir durch die Vereinigung der adaptiven Dünngitter-Interpolanten wieder einen adaptiven Dünngitter-Interpolanten erhalten, liegt am Unterraumschema, auf dem ein adaptives dünnes Gitter aufgebaut ist. Brauchen wir eine Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}$ zur Interpolation der Funktion, so sind alle Väter zu dieser Basisfunktion auch in der Darstellung enthalten, d.h. alle Basisfunktionen $\phi_{\mathbf{k},\mathbf{j}}$ mit $\mathbf{k} \leq \mathbf{l}$, für die $supp(\phi_{\mathbf{l},\mathbf{i}}) \subset supp(\phi_{\mathbf{k},\mathbf{j}})$ gilt.
Deswegen wenden wir uns in diesem Kapitel einer Dünngitter-Algebra zu, die sich mit Operatoren auf adaptiven Dünngitter-Interpolanten beschäftigt. Speziell betrachten wir dabei die Abschätzung des Interpolationsfehlers des Dünngitter-Interpolantens $\hat{u}_3^{\epsilon_3}$ und die Abhängigkeit dieses Interpolationsfehlers von den Interpolationsfehlern der Dünngitter-Interpolanten $\hat{u}_1^{\epsilon_1}$ und $\hat{u}_2^{\epsilon_2}$ für unterschiedliche Operatoren. Dabei erweitern wir die Dünngitter-Algebra aus [Sch98] mit dem Operator der Hintereinanderschaltung. Zuerst fassen wir im ersten Abschnitt die einfachen Operatoren aus [Sch98], wie skalare Multiplikation, Addition, Subtraktion, Multiplikation und Division, kurz zusammen. Im zweiten Abschnitt befassen wir uns dann mit der Hintereinanderschaltung zweier Dünngitter-Interpolanten.

4.1 Skalare Multiplikation, Addition, Subtraktion, Multiplikation und Division

Sei $\bar{\Omega} = [0,1]^d$, $u_i \in X^p(\Omega)$ und $\mu_i \geq 0$ mit $i \in \{1,2,3\}$ und $p \in \{2,\infty\}$. Dabei seien $\hat{u}_i^{\epsilon_i}$ die adaptiven Dünngitter-Interpolanten von u_i mit $i \in \{1,2,3\}$. Mit $\|\cdot\|$ sei die Norm bezeichnet, mit der der Interpolationsfehler berechnet wird. Hier sind das die L_2-Norm oder L_∞-Norm.
Zuerst betrachten wir die skalare Multiplikation.

Satz 4.1 *Sei $u_2 = c \cdot u_1$ mit $c \in \mathbb{R}$ und es gelte*

$$\|u_1 - \hat{u}_1^{\epsilon_1}\| < \mu_1.$$

Dann folgt daraus:

$$\|u_2 - \hat{u}_2^{\epsilon_2}\| < \mu_2 = |c| \cdot \mu_1 \quad \textit{mit } \epsilon_2 = c \cdot \epsilon_1$$

Beweis:

$$\begin{aligned}\|u_2 - \hat{u}_2^{\epsilon_2}\| &= \|c \cdot u_1 - \hat{u}_2^{c\cdot\epsilon_1}\| \\ &= \|c \cdot u_1 - c \cdot \hat{u}_1^{\epsilon_1}\| \\ &= |c| \cdot \|u_1 - \hat{u}_1^{\epsilon_1}\| \\ &\leq |c| \cdot \mu_1 = \mu_2\end{aligned}$$

Dass $\hat{u}_2^{c\cdot\epsilon_1} = c \cdot \hat{u}_1^{\epsilon_1}$ gilt, liegt daran, dass die hierarchischen Überschüsse von u_2 das c-fache der hierarchische Überschüsse von u_1 sind, also $(u_2)_{\mathbf{l},\mathbf{i}} = c \cdot (u_1)_{\mathbf{l},\mathbf{i}}$. Eine ausführlichere Erklärung findet sich in [Sch98].

q.e.d.

Als nächstes wenden wir uns der Addition bzw. der Subtraktion zu.

Satz 4.2 *Es gelte*

$$\|u_1 - \hat{u}_1^{\epsilon_1}\| < \mu_1 \quad \wedge \quad \|u_2 - \hat{u}_2^{\epsilon_2}\| < \mu_2,$$

dann folgt für $u_3 = u_1 \pm u_2$ die Abschätzung

$$\|u_3 - \hat{u}_3^{\epsilon_3}\| < \mu_3$$

mit $\mu_3 = \mu_1 + \mu_2$. Dabei ist $\hat{u}_3^{\epsilon_3} = \hat{u}_1^{\epsilon_1} \pm \hat{u}_2^{\epsilon_2}$ und $\epsilon_3 = \epsilon_1 + \epsilon_2$.

Beweis:

$$\begin{aligned}\|u_3 - \hat{u}_3^{\epsilon_3}\| &= \|u_1 \pm u_2 - \hat{u}_1^{\epsilon_1} \pm \hat{u}_2^{\epsilon_2}\| \\ &= \|(u_1 - \hat{u}_1^{\epsilon_1}) \pm (u_2 - \hat{u}_2^{\epsilon_2})\| \\ &\leq \|u_1 - \hat{u}_1^{\epsilon_1}\| \pm \|u_2 - \hat{u}_2^{\epsilon_2}\| \\ &\leq \mu_1 + \mu_2\end{aligned}$$

Zudem erhalten wir:

$$\begin{aligned}|(u_3)_{\mathbf{l},\mathbf{i}}| &= |(u_1)_{\mathbf{l},\mathbf{i}} \pm (u_2)_{\mathbf{l},\mathbf{i}}| \\ &\leq |(u_1)_{\mathbf{l},\mathbf{i}}| + |(u_2)_{\mathbf{l},\mathbf{i}}| \\ &\leq \epsilon_1 + \epsilon_2\end{aligned}$$

q.e.d.

Zum Schluss kommen wir zur Multiplikation. Den Fall der Division brauchen wir nicht zu behandeln, da für zwei Funktionen $f, g \in X^p\{\Omega\}$ mit $g \neq 0$ $\frac{f}{g} = f \cdot g^{-1}$ gilt.

Satz 4.3 *Es gelte*

$$\|u_1 - \hat{u}_1^{\epsilon_1}\| < \mu_1 \quad \wedge \quad \|u_2 - \hat{u}_2^{\epsilon_2}\| < \mu_2,$$

dann folgt für $u_3 = u_1 \cdot u_2$ die Abschätzung

$$\|u_3 - \hat{u}_3^{\epsilon_3}\| < \mu_3$$

mit

$$\mu_3 = \min\{\|u_1\| \cdot \mu_2 + \|\hat{u}_2^{\epsilon_2}\| \cdot \mu_1, \|u_2\| \cdot \mu_1 + \|\hat{u}_1^{\epsilon_1}\| \cdot \mu_2\}. \tag{4.1}$$

Eine schwächere Abschätzung liefert

$$\mu_3 = \max\{\|u_2\|, \|\hat{u}_2^{\epsilon_2}\|\} \cdot \mu_1 + \max\{\|u_1\|, \|\hat{u}_1^{\epsilon_1}\|\} \cdot \mu_2. \tag{4.2}$$

Dabei ist $\hat{u}_3^{\epsilon_3} = \hat{u}_1^{\epsilon_1} \cdot \hat{u}_2^{\epsilon_2}$ *auf der Indexmenge*

$$\mathcal{M}_3 = \mathcal{M}_1 \cup \mathcal{M}_2.$$

Wobei $\mathcal{M}_i = \mathcal{M}(u_i, \epsilon_i)$ $i \in \{1,2,3\}$ *mit*
$\mathcal{M}(u,\epsilon) = \{(\mathbf{l},\mathbf{i}) : (|\mathbf{l}|_1 = -d) \vee (|u_{\mathbf{l},\mathbf{i}}| \geq \epsilon \wedge \exists \min 1$ *Vater in* $\mathcal{M}(u,\epsilon))$ *zu* $(\mathbf{l},\mathbf{i})\}$ *ist.*

Beweis:
Wir betrachten zuerst die Abschätzung (4.1). Wir werden zweimal die Null addieren, um den gewünschten Ausdruck zu erhalten.

- $u_1 \cdot \hat{u}_2^{\epsilon_2} - u_1 \cdot \hat{u}_2^{\epsilon_2} = 0$
- $u_2 \cdot \hat{u}_1^{\epsilon_1} - u_2 \cdot \hat{u}_1^{\epsilon_1} = 0$

Also

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &= \|u_1 \cdot u_2 - \hat{u}_1^{\epsilon_1} \cdot \hat{u}_2^{\epsilon_2}\| \\
&= \|u_1 \cdot u_2 - u_1 \cdot \hat{u}_2^{\epsilon_2} + u_1 \cdot \hat{u}_2^{\epsilon_2} - \hat{u}_1^{\epsilon_1} \cdot \hat{u}_2^{\epsilon_2}\| \\
&= \|u_1 \cdot (u_2 - \hat{u}_2^{\epsilon_2}) + \hat{u}_2^{\epsilon_2} \cdot (u_1 - \hat{u}_1^{\epsilon_1})\| \\
&\leq \|u_1\| \cdot \|u_2 - \hat{u}_2^{\epsilon_2}\| + \|\hat{u}_2^{\epsilon_2}\| \cdot \|u_1 - \hat{u}_1^{\epsilon_1}\| \\
&\leq \|u_1\| \cdot \mu_2 + \|\hat{u}_2^{\epsilon_2}\| \cdot \mu_1
\end{aligned}$$

Analog erhalten wir mit der zweiten Gleichung

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &= \|u_1 \cdot u_2 - u_2 \cdot \hat{u}_1^{\epsilon_1} + u_2 \cdot \hat{u}_1^{\epsilon_1} - \hat{u}_1^{\epsilon_1} \cdot \hat{u}_2^{\epsilon_2}\| \\
&\leq \|u_2\| \cdot \|u_1 - \hat{u}_1^{\epsilon_1}\| + \|\hat{u}_1^{\epsilon_1}\| \cdot \|u_2 - \hat{u}_2^{\epsilon_2}\| \\
&\leq \|u_2\| \cdot \mu_1 + \|\hat{u}_1^{\epsilon_1}\| \cdot \mu_2
\end{aligned}$$

Aus beiden Abschätzung folgt also:

$$\|u_3 - \hat{u}_3^{\epsilon_3}\| \leq \min\{\|u_1\| \cdot \mu_2 + \|\hat{u}_2^{\epsilon_2}\| \cdot \mu_1, \|u_2\| \cdot \mu_1 + \|\hat{u}_1^{\epsilon_1}\| \cdot \mu_2\}$$

Addiert man nun beide Abschätzungen und teilt das Ganze durch zwei, so bekommen wir schließlich die schwächere Abschätzung (4.2).

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &\leq \frac{1}{2} \cdot (\|u_1\| \cdot \mu_2 + \|\hat{u}_2^{\epsilon_2}\| \cdot \mu_1 + \|u_2\| \cdot \mu_1 + \|\hat{u}_1^{\epsilon_1}\| \cdot \mu_2) \\
&= \frac{1}{2} \cdot [(\|u_1\| + \|\hat{u}_1^{\epsilon_1}\|) \cdot \mu_2 + (\|\hat{u}_2^{\epsilon_2}\| + \|u_2\|) \cdot \mu_1] \\
&\leq \max\{\|u_2\|, \|\hat{u}_2^{\epsilon_2}\|\} \cdot \mu_1 + \max\{\|u_1\|, \|\hat{u}_1^{\epsilon_1}\|\} \cdot \mu_2
\end{aligned}$$

q.e.d.

Eine ausführlichere Diskussion der Multiplikation ist in [Sch98] zu finden.

4.2 Hintereinanderschaltung

Satz 4.4 *Sei $u_1, u_2, u_3 \in H_0^1(\bar{\Omega})$ und es gelte*

$$\|u_1 - \hat{u}_1^{\epsilon_1}\| < \mu_1 \quad \wedge \quad \|u_2 - \hat{u}_2^{\epsilon_2}\| < \mu_2,$$

dann folgt für $u_3 = u_1(u_2)$ die Abschätzung

$$\|u_3 - \hat{u}_3^{\epsilon_3}\| < \mu_3$$

mit

$$\mu_3 = \mu_1 + \sum_{k=-1}^{l_{u_1}} \max_{|\mathbf{l}|_1+d-1=k, \mathbf{i}\in I_\mathbf{l}} \left\{ \left| u_{\mathbf{l},\mathbf{i}}^1 \right| \cdot c_\mathbf{l} \right\} \cdot \sum_{j=1}^{d} 2^{d-j} \binom{d}{j} \mu_2^j, \tag{4.3}$$

wobei l_{u_i} das maximale Level von $\hat{u}_i^{\epsilon_i}$ $i = 1, 2$ und $l_{u_1} \leq l_{u_2}$ gilt. Dabei sind
$I_\mathbf{l} = \{\mathbf{i} : \ i_j = 1, \ldots, 2^{l_j} - 1$ mit i_j ungerade für $l_j > 0$ und $i_j = 1$ für $l_j \leq 0, \ 1 \leq j \leq d\}$,
$u_{\mathbf{l},\mathbf{i}}^1$ die hierarchischen Überschüsse zu $\hat{u}_1^{\epsilon_1}$ und $c_\mathbf{l} = \prod_{k=1}^d c_{l_k}$ mit $c_{l_k} = \begin{cases} 1 & \text{wenn } l_k = -1 \\ 2^{l_k} & \text{sonst} \end{cases}$.

Beweis:
Zuerst betrachten wir u_3 und $\hat{u}_3^{\epsilon_3}$ in der Summenschreibweise. Dabei werden wir sehen, dass die Abschätzung aus einem inneren und einem äußeren Teil besteht, d.h. dass wir die Abschätzung in zwei Abschätzungen aufteilen können, einmal in den vom Interpolationsfehler zu u_1 und einmal in den vom Interpolationsfehler zu u_2 abhängigen Teil.

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &= \left\| \sum_{|\mathbf{l}|_1=-1}^{\infty} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 \phi_{\mathbf{l},\mathbf{i}}(u_2) - \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right\| \\
&\leq \underbrace{\left\| \sum_{|\mathbf{l}|_1=l_{u_1}+1}^{\infty} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 \phi_{\mathbf{l},\mathbf{i}}(u_2) \right\|}_{\text{der Fehler, der von } u_1 \text{ abhängt}} + \underbrace{\left\| \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 \phi_{\mathbf{l},\mathbf{i}}(u_2) - \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right\|}_{\text{der Fehler, der von } u_2 \text{ abhängt}} \\
&< \mu_1 + \left\| \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 [\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})] \right\|
\end{aligned}$$

Im weiteren Verlauf werden wir uns nur noch den inneren Teil anschauen, da wir mit dem äußeren Teil fertig sind. Außerdem muss pro Level $\mathbf{l}$ nur ein Unterraum $W_\mathbf{l}$ (Definition 2.9) betrachtet werden, da ein Punkt nicht in zwei Unterräumen desselben Levels liegen kann, d.h. wir haben nur noch $l_{u_1} + 2$ Summanden.

$$\left\| \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u_{\mathbf{l},\mathbf{i}}^1 [\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})] \right\| \leq \left\| \sum_{k=-1}^{l_{u_1}} \max_{k=|\mathbf{l}|_1, \mathbf{i}\in I_\mathbf{l}} \{ u_{\mathbf{l},\mathbf{i}}^1 [\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})] \} \right\|$$

Betrachten wir nun zuerst die Abschätzung in einer Dimension.
Liegen $u_2(x)$ und $\hat{u}_2^{\epsilon_2}(x)$ im Träger von der Basisfunktion $\phi_{l,i}$, so kann man die Differenz zwischen $u_2(x)$ und $\hat{u}_2^{\epsilon_2}(x)$ wie folgt abschätzen:
falls $l \geq 0$ ist:

$$\begin{aligned}\|\phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2})\| &= \left\|1 - \left|2^l u_2 - i\right| - 1 + \left|2^l \hat{u}_2^{\epsilon_2} - i\right|\right\| \\ &\leq 2^l \|u_2 - \hat{u}_2^{\epsilon_2}\| \\ &< 2^l \mu_2\end{aligned}$$

falls $l = -1$ ist:

$$\|\phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2})\| = \|1 - 1\| = 0$$

Für den Fall, dass $u_2(x)$ und $\hat{u}_2^{\epsilon_2}(x)$ nicht im Träger von der Basisfunktion $\phi_{l,i}$ liegen, wird die Differenz zwischen $u_2(x)$ und $\hat{u}_2^{\epsilon_2}(x)$ null.

Bleibt nur noch der Fall zu betrachten, in dem $u_2(x)$, bzw. $\hat{u}_2^{\epsilon_2}(x)$ im und $\hat{u}_2^{\epsilon_2}(x)$, bzw. $u_2(x)$ außerhalb des Trägers von $\phi_{l,i}$ liegt. Dies trifft nur für $l > 0$ zu, da für $l \leq 0$ der Träger von $\phi_{l,i}$ gleich $\bar{\Omega}$ ist.
Sei o.B.d.A. $\phi_{l,i}(\hat{u}_2^{\epsilon_2}(x)) = 0$, dann ist $\hat{u}_2^{\epsilon_2}(x)$ entweder $< \frac{i-1}{2^l}$ oder $> \frac{i+1}{2^l}$ und die Abschätzung kann wie folgt durchgeführt werden:

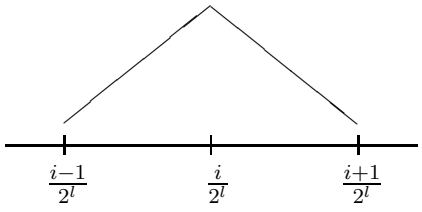


Abb. 4.1: Basisfunktion $\phi_{l,i}$

$$\begin{aligned}\|\phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2})\| &= \|\phi_{l,i}(u_2)\| \\ &= \left\|1 - \left|2^l u_2 - i\right| \right\| \\ &= \left\|2^l u_2 - i - 1 \cdot sign(2^l u_2 - i)\right\| \\ &= 2^l \left\|u_2 - \frac{i + 1 \cdot sign(2^l u_2 - i)}{2^l}\right\| \\ &< 2^l \|u_2 - \hat{u}_2^{\epsilon_2}\| \\ &< 2^l \mu_2\end{aligned}$$

Wenn wir alle betrachteten Fälle zusammen nehmen, ergibt das für unsere Abschätzung:

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &< \mu_1 + \left\| \sum_{l=-1}^{l_{u_1}} \sum_{i \in I_l} u_{l,i}^1 [\phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2})] \right\| \\
&\leq \mu_1 + \left\| \sum_{l=-1}^{l_{u_1}} \max_{i \in I_l} \{ u_{l,i}^1 [\phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2})] \} \right\| \\
&\leq \mu_1 + \sum_{l=-1}^{l_{u_1}} \max_{i \in I_l} \{ \left| u_{l,i}^1 \right| \, \| \phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2}) \| \} \\
&= \mu_1 + \sum_{l=0}^{l_{u_1}} \max_{i \in I_l} \{ \left| u_{l,i}^1 \right| \, \| \phi_{l,i}(u_2) - \phi_{l,i}(\hat{u}_2^{\epsilon_2}) \| \} \\
&< \mu_1 + \sum_{l=0}^{l_{u_1}} \max_{i \in I_l} \{ \left| u_{l,i}^1 \right| \} \cdot 2^l \mu_2 \\
&\leq \mu_1 + \sum_{l=-1}^{l_{u_1}} c_l \max_{i \in I_l} \{ \left| u_{l,i}^1 \right| \} \mu_2
\end{aligned}$$

Damit ist die Abschätzung in einer Dimension bewiesen. Im Mehrdimensionalen wird die Abschätzung komplexer, da $\phi_{\mathbf{l},\mathbf{i}}$ durch $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j)$ definiert wird. Wir werden vier Fälle betrachten und dann diese zusammenführen, um die endgültige Abschätzung zu erlangen.
Sei im weiteren Verlauf des Beweises

$$\begin{aligned}
&\mathbf{x} = (x_1, \ldots, x_d), \\
&u_2(\mathbf{x}) = (u_{2,1}(x_1), \ldots, u_{2,d}(x_d)), \\
&\hat{u}_2^{\epsilon_2}(\mathbf{x}) = (\hat{u}_{2,1}^{\epsilon_2}(x_1), \ldots, \hat{u}_{2,d}^{\epsilon_2}(x_d)), \\
&\hat{u}_{2,n}^{\epsilon_2}(x_n) = u_{2,n}(x_n) + \xi_n(x_n) \text{ mit } \xi_n(x_n) \in \mathbb{R}, \\
&\xi = \max_{n=1 \ldots d} |\xi_n| \text{ und } \xi < \mu_2.
\end{aligned}$$

Fall 1: In diesem Fall liegen $u_2(\mathbf{x})$ und $\hat{u}_2^{\epsilon_2}(\mathbf{x})$ im Träger von $\phi_{\mathbf{l},\mathbf{i}}$ und $l_n \geq 1,\ n = 1 \ldots d$
Wir definieren uns folgende Mengen

$$\Delta(\mathbf{j}, n, d) := \{(j_1, \ldots, j_n) \in \mathbb{N} \text{ mit } j_n \leq d \text{ und } j_1 < j_2 < \cdots < j_n\} \text{ und}$$

$$\bigtriangledown(\mathbf{j}, n, d) := \{s : s \in \{1, \ldots, d\} \ \wedge \ s \neq j_t \text{ für } t = 1, \ldots, n\},$$

damit der Beweis übersichtlicher wird. Dann ist

$$
\begin{aligned}
&\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| \\
&= \left\| \prod_{n=1}^{d} \left(1 - \left|2^{l_n} u_{2,n} - i_n\right|\right) - \prod_{n=1}^{d} \left(1 - \left|2^{l_n} u_{2,n} + 2^{l_n}\xi_n - i_n\right|\right) \right\| \\
&= \left\| \sum_{n=1}^{d} \Delta(\mathbf{j},n,d)(-1)^n \prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} - i_{j_t}\right| - \sum_{n=1}^{d} \Delta(\mathbf{j},n,d)(-1)^n \prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right| \right\| \\
&= \left\| \sum_{n=1}^{d} \Delta(\mathbf{j},n,d)(-1)^n \left[\prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} - i_{j_t}\right| - \prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right| \right] \right\| \\
&\le \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \left\| \prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} - i_{j_t}\right| - \prod_{t=1}^{n} \left|2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right| \right\| \\
&= \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \left\| \left| \prod_{t=1}^{n} 2^{l_{j_t}} u_{2,j_t} - i_{j_t} \right| - \left| \prod_{t=1}^{n} 2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t} \right| \right\| \\
&\le \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \left\| \prod_{t=1}^{n} (2^{l_{j_t}} u_{2,j_t} - i_{j_t}) - \prod_{t=1}^{n} (2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}) \right\| \\
&= \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \\
&\qquad \left\| \sum_{m=0}^{n} \Delta(\mathbf{k},m,n) \left[\prod_{t=1}^{m} 2^{l_{j_{k_t}}} u_{2,j_{k_t}} \prod_{s\in\bigtriangledown(\mathbf{k},m,n)} (-i_{j_s}) - \prod_{t=1}^{m} \left(2^{l_{j_{k_t}}} u_{2,j_{k_t}} + 2^{l_{j_{k_t}}}\xi_{j_{k_t}}\right) \prod_{s\in\bigtriangledown(\mathbf{k},m,n)} (-i_{j_s}) \right] \right\| \\
&\le \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \sum_{m=0}^{n} \Delta(\mathbf{k},m,n) \\
&\qquad \left\| \prod_{t=1}^{m} 2^{l_{j_{k_t}}} u_{2,j_{k_t}} \prod_{s\in\bigtriangledown(\mathbf{k},m,n)} (-i_{j_s}) - \sum_{r=0}^{m} \Delta(\mathbf{v},r,m) \prod_{t=1}^{r} 2^{l_{j_{k_{v_t}}}} u_{2,j_{k_{v_t}}} \prod_{p\in\bigtriangledown(\mathbf{v},r,m)} 2^{l_{j_{k_p}}} \xi_{j_{k_p}} \prod_{s\in\bigtriangledown(\mathbf{k},m,n)} (-i_{j_s}) \right\| \\
&\le \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \sum_{m=1}^{n} \Delta(\mathbf{k},m,n) \sum_{r=0}^{m-1} \Delta(\mathbf{v},r,m) \\
&\qquad \left\| \prod_{t=1}^{r} 2^{l_{j_{k_{v_t}}}} u_{2,j_{k_{v_t}}} \prod_{p\in\bigtriangledown(\mathbf{v},r,m)} 2^{l_{j_{k_p}}} \xi_{j_{k_p}} \prod_{s\in(\bigtriangledown(\mathbf{k},m,n)\cap\bigtriangledown(\mathbf{k_v},r,n))} i_{j_s} \right\| \qquad (*) \\
&\le \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \sum_{m=1}^{n} \Delta(\mathbf{k},m,n) \sum_{r=0}^{m-1} \Delta(\mathbf{v},r,m)\ \xi^{m-r} \prod_{t=1}^{r} 2^{l_{j_{k_{v_t}}}} \prod_{p\in\bigtriangledown(\mathbf{v},r,m)} 2^{l_{j_{k_p}}} \prod_{s\in(\bigtriangledown(\mathbf{k},m,n)\cap\bigtriangledown(\mathbf{k_v},r,n))} (2^{l_{j_s}} - 1) \\
&= \sum_{n=1}^{d} \Delta(\mathbf{j},n,d) \sum_{m=1}^{n} \Delta(\mathbf{k},m,n) \prod_{t=1}^{m} 2^{l_{j_{k_t}}} \prod_{s\in\bigtriangledown(\mathbf{k},m,n)} (2^{l_{j_s}} - 1) \sum_{r=0}^{m-1} \Delta(\mathbf{v},r,m)\ \xi^{m-r}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{n=1}^{d} \Delta(\mathbf{j}, n, d) \sum_{m=1}^{n} \Delta(\mathbf{k}, m, n) \prod_{t=1}^{m} 2^{l_{j_{k_t}}} \prod_{s \in \triangledown(\mathbf{k},m,n)} (2^{l_{j_s}} - 1) \sum_{r=0}^{m-1} \binom{m}{r} \xi^{m-r} \\
&= \sum_{m=1}^{d} \Delta(\mathbf{k}, m, n) \prod_{t=1}^{m} 2^{l_{k_t}} \prod_{s \in \triangledown(\mathbf{k},m,d)} 2^{l_s} \sum_{r=0}^{m-1} \binom{m}{r} \xi^{m-r} \\
&= \prod_{s=1}^{d} 2^{l_s} \left[\sum_{m=1}^{d} \Delta(\mathbf{k}, m, n) \sum_{r=0}^{m-1} \binom{m}{r} \xi^{m-r} \right] = \prod_{s=1}^{d} 2^{l_s} \left[\sum_{m=1}^{d} \binom{d}{m} \sum_{r=0}^{m-1} \binom{m}{r} \xi^{m-r} \right] \\
&= \prod_{s=1}^{d} 2^{l_s} \left[\sum_{m=1}^{d} \binom{d}{m} \cdot \left(\sum_{r=0}^{m} \binom{m}{r} \xi^{m-r} - 1 \right) \right] = \prod_{s=1}^{d} 2^{l_s} \left[\sum_{m=1}^{d} \binom{d}{m} (\xi+1)^m - \sum_{m=1}^{d} \binom{d}{m} \right] \\
&= \prod_{s=1}^{d} 2^{l_s} \left[\sum_{m=1}^{d} \binom{d}{m} (\xi+1)^m - \binom{d}{0} - \sum_{m=1}^{d} \binom{d}{m} + \binom{d}{0} \right] \\
&= \prod_{s=1}^{d} 2^{l_s} \left[(\xi+2)^d - 2^d \right] = \prod_{s=1}^{d} 2^{l_s} \left[\sum_{k=0}^{d} \binom{d}{k} \xi^k 2^{d-k} - 2^d \right] = \prod_{s=1}^{d} 2^{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \xi^k 2^{d-k} \right]
\end{aligned}
$$

$(*)$: Es gilt für $n = 1 \dots d$ und $l_n \geq 1$, dass $\|\xi_n\| \leq \xi$, $\|u_{2,n}\| \leq 1$ und $\|i_n\| \leq 2^{l_n} - 1$ ist.

Fall 2: Ist $l_n < 1,\ n \in \{1, \dots, d\}$, dann liegen $u_{2,n}(x_n)$ und $\hat{u}_{2,n}^{\epsilon_2}(x_n)$ immer im Träger von ϕ_{l_n,i_n}. Wir betrachten dazu die Fälle $l_n = -1$ und $l_n = 0$ getrennt voneinader.

- Ist $l_n = -1$, so ist $\phi_{l_n,i_n}(u_{2,n}) = \phi_{l_n,i_n}(\hat{u}_{2,n}^{\epsilon_2}) = 1$.
  Das heißt, wir müssen nach dem Umsortieren nur den Fall
  $\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) = \prod_{n=1}^{d-t} \left(1 - \left|2^{l_n} u_{2,n} - i_n\right|\right) - \prod_{n=1}^{d-t} \left(1 - \left|2^{l_n} u_{2,n} + 2^{l_n} \xi_n - i_n\right|\right)$ betrachten, wobei t für die Anzahl der $l_n = -1,\ n = 1 \dots d$ steht.
  Sei nun t-mal das Level -1 vertreten und $d - t$-mal das Level ≥ 1:

$$
\begin{aligned}
&\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| \\
&= \left\| \prod_{n=1}^{d-t} \left(1 - \left|2^{l_n} u_{2,n} - i_n\right|\right) - \prod_{n=1}^{d-t} \left(1 - \left|2^{l_n} u_{2,n} + 2^{l_n} \xi_n - i_n\right|\right) \right\| \\
&\leq \prod_{s=1}^{d-t} 2^{l_s} \left[\sum_{k=1}^{d-t} \binom{d-t}{k} \xi^k 2^{d-t-k} \right]
\end{aligned}
$$

  Die obere grenze ist aber immer $\leq \prod_{s=1}^{d-t} 2^{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \xi^k 2^{d-k} \right]$. Wenn wir einen Koeffizientenvergleich durchführen sehen wir, dass für diese Aussage

  a) $\binom{d}{k} \geq \binom{d-t}{k}$
  b) $2^{d-k} \geq 2^{d-t-k}$

gelten muss. Das ist aber leicht zu beweisen:

$$a) \qquad \binom{d}{k} = \frac{d!}{(d-k)!k!}$$
$$= \frac{d\cdots(d-t+1)(d-t)!}{(d-k)\cdots(d-k-t+1)(d-t-k)!k!}$$
$$= \underbrace{\frac{d\cdots(d-t+1)}{(d-k)\cdots(d-k-t+1)}}_{\geq 1}\binom{d-t}{k} > \binom{d-t}{k}$$

$$b) \qquad \frac{2^{d-k}}{2^{d-k-t}} = 2^t > 1$$

- Ist $l_n = 0$, so ist $\phi_{l_n,i_n}(\hat{u}_{2,n}^{\epsilon_2}) = \left|2^{l_n}\hat{u}_{2,n}^{\epsilon_2}\right| = \left|\hat{u}_{2,n}^{\epsilon_2}\right|$ und $\phi_{l_n,i_n}(u_{2,n}) = \left|2^{l_n}u_{2,n}\right| = |u_{2,n}|$. Wir teilen die Basisfunktionen auf in Basisfunktionen zum Level $l_n = 0$ und $l_n > 0$ und sortieren diese um, so erhalten wir Q Basisfunktionen zum Level 0 und $d-Q$ Basisfunktionen zum Level > 0.

$$\begin{aligned}
&\|\phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) - \phi_{\mathbf{l},\mathbf{i}}(u_2)\| \\
&= \Bigg\| \prod_{n=1}^{d-Q}\left(1 - \left|2^{l_n}u_{2,n} + 2^{l_n}\xi_n - i_n\right|\right) \prod_{n=d-Q+1}^{d}|u_{2,n}+\xi_n| \\
&\quad - \prod_{n=1}^{d-Q}\left(1 - \left|2^{l_n}u_{2,n} - i_n\right|\right) \prod_{n=d-Q+1}^{d}|u_{2,n}| \Bigg\| \\
&= \Bigg\| \left(1 + \sum_{n=1}^{d-Q}\Delta(\mathbf{j},n,d-Q)(-1)^n \prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right|\right) \prod_{n=d-Q+1}^{d}|u_{2,n}+\xi_n| \\
&\quad - \left(1 + \sum_{n=1}^{d-Q}\Delta(\mathbf{j},n,d-Q)(-1)^n \prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} - i_{j_t}\right|\right) \prod_{n=d-Q+1}^{d}|u_{2,n}| \Bigg\| \\
&= \Bigg\| \left(\prod_{n=d-Q+1}^{d}|u_{2,n}+\xi_n| - \prod_{n=d-Q+1}^{d}|u_{2,n}|\right) + \prod_{n=d-Q+1}^{d}|u_{2,n}| \left[\sum_{n=1}^{d-Q}\Delta(\mathbf{j},n,d-Q)(-1)^n \right. \\
&\quad \left. \cdot \left(\prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right| - \prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} - i_{j_t}\right|\right)\right] \\
&\quad + \left(\prod_{n=d-Q+1}^{d}|u_{2,n}+\xi_n| - \prod_{n=d-Q+1}^{d}|u_{2,n}|\right) \\
&\quad \cdot \left[\sum_{n=1}^{d-Q}\Delta(\mathbf{j},n,d-Q)(-1)^n \left(\prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} + 2^{l_{j_t}}\xi_{j_t} - i_{j_t}\right| - \prod_{t=1}^{n}\left|2^{l_{j_t}}u_{2,j_t} - i_{j_t}\right|\right)\right] \Bigg\|
\end{aligned}$$

$$
\begin{aligned}
&\leq \underbrace{\left\| \prod_{n=d-Q+1}^{d} |u_{2,n}+\xi_n| - \prod_{n=d-Q+1}^{d} |u_{2,n}| \right\|}_{\blacksquare} \\
&\quad + \underbrace{\left\| \prod_{n=d-Q+1}^{d} |u_{2,n}| \right\|}_{\leq 1} \\
&\quad \cdot \underbrace{\left\| \sum_{n=1}^{d-Q} \Delta(\mathbf{j},n,d-Q)(-1)^n \left(\prod_{t=1}^{n} \left| 2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}} \xi_{j_t} - i_{j_t} \right| - \prod_{t=1}^{n} \left| 2^{l_{j_t}} u_{2,j_t} - i_{j_t} \right| \right) \right\|}_{\prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{k=1}^{d-Q} \binom{d-Q}{k} \xi^k 2^{d-Q-k} \right]} \\
&\quad + \left\| \prod_{n=d-Q+1}^{d} |u_{2,n}+\xi_n| - \prod_{n=d-Q+1}^{d} |u_{2,n}| \right\| \\
&\quad \cdot \left\| \sum_{n=1}^{d-Q} \Delta(\mathbf{j},n,d-Q)(-1)^n \left(\prod_{t=1}^{n} \left| 2^{l_{j_t}} u_{2,j_t} + 2^{l_{j_t}} \xi_{j_t} - i_{j_t} \right| - \prod_{t=1}^{n} \left| 2^{l_{j_t}} u_{2,j_t} - i_{j_t} \right| \right) \right\|
\end{aligned}
$$

$$
\begin{aligned}
\blacksquare &= \left\| \prod_{n=1}^{Q} |u_{2,n+d-Q}+\xi_{n+d-Q}| - \prod_{n=1}^{Q} |u_{2,n+d-Q}| \right\| \\
&= \left\| \left| \sum_{n=0}^{Q} \Delta(\mathbf{j},n,Q) \prod_{t=1} u_{2,j_t+d-Q} \prod_{s\in\triangledown(\mathbf{j},n,Q)} \xi_{s+d-Q} \right| - \left| \prod_{n=1}^{Q} u_{2,n+d-Q} \right| \right\| \\
&\leq \left\| \sum_{n=0}^{Q-1} \Delta(\mathbf{j},n,Q) \prod_{t=1} u_{2,j_t+d-Q} \prod_{s\in\triangledown(\mathbf{j},n,Q)} \xi_{s+d-Q} \right\| \\
&\leq \sum_{n=0}^{Q-1} \Delta(\mathbf{j},n,Q) \underbrace{\left\| \prod_{t=1} u_{2,j_t+d-Q} \right\|}_{\leq 1} \cdot \left\| \prod_{s\in\triangledown(\mathbf{j},n,Q)} \xi_{s+d-Q} \right\| \\
&\leq \sum_{n=0}^{Q-1} \Delta(\mathbf{j},n,Q) \xi^{Q-n} \\
&= \sum_{n=0}^{Q-1} \binom{Q}{n} \xi^{Q-n} \\
&= \sum_{n=1}^{Q} \binom{Q}{n} \xi^{n}
\end{aligned}
$$

Mit ■ erhalten wir:

$$\begin{aligned}
&\|\phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) - \phi_{\mathbf{l},\mathbf{i}}(u_2)\| \\
&\leq \sum_{n=1}^{Q} \binom{Q}{n} \xi^n + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{k=1}^{d-Q} \binom{d-Q}{k} \xi^k 2^{d-Q-k} \right] \\
&\quad + \left[\sum_{t=1}^{Q} \binom{Q}{t} \xi^t \right] \cdot \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{k=1}^{d-Q} \binom{d-Q}{k} \xi^k 2^{d-Q-k} \right] \\
&= \sum_{n=1}^{Q} \binom{Q}{n} \xi^n + (\xi+1)^Q \cdot \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{k=1}^{d-Q} \binom{d-Q}{k} \xi^k 2^{d-Q-k} \right] \\
&= \sum_{n=1}^{Q} \binom{Q}{n} \xi^n + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{t=0}^{Q} \sum_{k=1}^{d-Q} \binom{Q}{t} \binom{d-Q}{k} \xi^{k+t} 2^{d-Q-k} \right]
\end{aligned}$$

Wir werden nun zeigen, dass

$$\sum_{n=1}^{Q} \binom{Q}{n} \xi^n + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{t=0}^{Q} \sum_{k=1}^{d-Q} \binom{Q}{t} \binom{d-Q}{k} \xi^{k+t} 2^{d-Q-k} \right] \leq \prod_{n=1}^{d-Q} 2^{l_n} \sum_{k=1}^{d} \binom{d}{k} \xi^k 2^{d-k}$$

gilt. Dafür betrachten wir im Folgenden die Koeffizienten zu ξ_k für $k = 1, \ldots, d$.
Der Koeffizient zu ξ_k ist

$$\bigstar = \binom{Q}{k} + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=1}^{k-1} \binom{d-Q}{s} \binom{Q}{k-s} 2^{d-Q-s} + \binom{d-Q}{k} 2^{d-Q-k} \right]$$

und das erwartete Ergebnis

$$\clubsuit = \prod_{n=1}^{d-Q} 2^{l_n} \binom{d}{k} 2^{d-k}.$$

Es müssen vier verschiedene Fällen betrachtet werden:

Fall A $d - Q \geq t$ und $Q \geq t$

Fall B $d - Q < t$ und $Q < t$

Fall C $d - Q \geq t$ und $Q < t$

Fall D $d - Q < t$ und $Q \geq t$

Bei den Abschätzungen werden wir das dritte Additionstheorem für verallgemeinerte Binomialkoeffizienten (Satz 9.1) verwenden.

Fall A:

$$\begin{aligned}\bigstar &\leq \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=0}^{k} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &\leq \prod_{n=1}^{d-Q} 2^{l_n} 2^{d-Q} \left[\sum_{s=0}^{k} \binom{d-Q}{s}\binom{Q}{k-s}\right] \\ &= \prod_{n=1}^{d-Q} 2^{l_n} 2^{d-Q} \binom{d}{k} \leq \clubsuit\end{aligned}$$

Fall B:

$$\begin{aligned}\bigstar &= \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=1}^{k-1} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &= \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=k-Q}^{d-Q} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &\leq \prod_{n=1}^{d-Q} 2^{l_n} 2^{d-t} \left[\sum_{s=k-Q}^{d-Q} \binom{d-Q}{s}\binom{Q}{k-s}\right] = \clubsuit\end{aligned}$$

Fall C:

$$\begin{aligned}\bigstar &= \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=k-Q}^{k-1} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s} + \binom{d-Q}{k} 2^{d-Q-k}\right] \\ &= \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=k-Q}^{k} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &\leq \prod_{n=1}^{d-Q} 2^{l_n} 2^{d-t} \left[\sum_{s=k-Q}^{k} \binom{d-Q}{s}\binom{Q}{k-s}\right] = \clubsuit\end{aligned}$$

Fall D:

$$\begin{aligned}\bigstar &= \binom{Q}{k} + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=1}^{d-Q} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &\leq \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{s=0}^{d-Q} \binom{d-Q}{s}\binom{Q}{k-s} 2^{d-Q-s}\right] \\ &\leq \prod_{n=1}^{d-Q} 2^{l_n} 2^{d-Q} \binom{d}{k} \leq \clubsuit\end{aligned}$$

Damit ist bewiesen, dass

$$\begin{aligned}
&\|\phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) - \phi_{\mathbf{l},\mathbf{i}}(u_2)\| \\
&\leq \sum_{n=1}^{Q} \binom{Q}{n} \xi^n + \prod_{n=1}^{d-Q} 2^{l_n} \left[\sum_{t=0}^{Q} \sum_{k=1}^{d-Q} \binom{Q}{t} \binom{d-Q}{k} \xi^{k+t} 2^{d-Q-k} \right] \\
&\leq \prod_{n=1}^{d-Q} 2^{l_n} \sum_{k=1}^{d} \binom{d}{k} \xi^k 2^{d-k}
\end{aligned}$$

Fall 3: Wenn $u_{2,n}(x_n)$ und $\hat{u}_{2,n}^{\epsilon_2}(x_n) = u_{2,n}(x_n) + \xi_n(x_n)$ mit $|\xi_n(x_n)| < \mu_2$ für mindestens ein $n \in \{1, \ldots, d\}$ nicht im Träger von ϕ_{l_n,i_n} liegen, gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| = 0.$$

In diesem Fall wird der Fehler der von $u_1^{\epsilon_2}$, der an der Stelle $\mathbf{x}$ gemacht werden könnte, nicht an die Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}$ von $u_2^{\epsilon_2}$ weitergegeben.

Fall 4: Wenn entweder $u_{2,n}(x_n)$ oder $\hat{u}_{2,n}^{\epsilon_2}(x_n) = u_{2,n}(x_n) + \xi_n(x_n)$ mit $|\xi_n(x_n)| < \mu_2$ und $l_n \geq 1$ außerhalb des Trägers von ϕ_{l_n,i_n} liegt, ist $2^{l_n} u_{2,n}(x_n)$ oder $2^{l_n} \hat{u}_{2,n}^{\epsilon_2}(x_n) > i_n + 1$ bzw. $< i_n - 1$. Sei nun o.B.d.A. $u_{2,n}(x_n)$ im Träger und $\hat{u}_{2,n}^{\epsilon_2}(x_n)$ außerhalb des Trägers von ϕ_{l_n,i_n}:

$$\begin{aligned}
\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| &= \|\phi_{\mathbf{l},\mathbf{i}}(u_2)\| \\
&= \left\| \prod_{k=1}^{d} \phi_{l_k,i_k}(u_{2,k}) \right\| \\
&\leq \left\| 1 - \left| 2^{l_n} u_{2,n} - i_n \right| \right\|, \text{ da } 0 \leq \phi_{l_k,i_k} \leq 1 \\
&= \left\| 2^{l_n} u_{2,n} \underbrace{-i_n - 1 \cdot sign(2^{l_n} u_{2,n} - i_n)}_{-(i_n \pm 1)} \right\| \\
&< \left\| 2^{l_n} u_{2,n} - 2^{l_n} u_{2,n} - 2^{l_n} \xi_n \right\| \\
&= 2^{l_n} \|\xi_n\| < 2^{l_n} \mu_2
\end{aligned}$$

Das gleiche gilt für den anderen Fall, d.h. $u_{2,n}(x_n)$ liegt außerhalb und $\hat{u}_{2,n}^{\epsilon_2}(x_n)$ innerhalb des Trägers von ϕ_{l_n,i_n}.

Zusammenführung:

Fall 1 $l_n \geq 1$ für $n = 1 \ldots d$, $u_2(\mathbf{x})$ und $\hat{u}_2^{\epsilon_2}(\mathbf{x})$ liegen im Träger von $\phi_{\mathbf{l},\mathbf{i}}$, dann gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| < \prod_{s=1}^{d} 2^{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \mu_2^k 2^{d-k} \right]$$

Fall 2 $l_n = -1$ für t verschiedene n mit $n \in \{1, \ldots, d\}$, sonst $l_n > 0$, $u_2(\mathbf{x})$ und $\hat{u}_2^{\epsilon_2}(\mathbf{x})$ liegen im Träger von $\phi_{\mathbf{l},\mathbf{i}}$, dann gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| < \prod_{s=1}^{d-t} 2^{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \mu_2^k 2^{d-k} \right]$$

$l_n = 0$ für Q verschiedene n mit $n \in \{1, \ldots, d\}$, sonst $l_n > 0$, $u_2(\mathbf{x})$ und $\hat{u}_2^{\epsilon_2}(\mathbf{x})$ liegen im Träger von $\phi_{\mathbf{l},\mathbf{i}}$, dann gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| < \prod_{s=1}^{d-Q} 2^{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \mu_2^k 2^{d-k} \right]$$

Fall 3 $l_n \geq 1$ für $n = 1 \ldots d$, $u_2(\mathbf{x})$ und $\hat{u}_2^{\epsilon_2}(\mathbf{x})$ liegen nicht im Träger von $\phi_{\mathbf{l},\mathbf{i}}$, dann gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| = 0$$

Fall 4 Entweder $u_{2,n}(x_n)$ oder $\hat{u}_{2,n}^{\epsilon_2}(x_n)$ mit $l_n \geq 1$ liegt außerhalb des Trägers von ϕ_{l_n,i_n} für mindestens ein n, dann gilt:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| < 2^{l_n} \mu_2$$

Die Ergebnisse von **Fall 2** bis **Fall 4** sind Teilmengen des Ergebnisses von **Fall 1**. Da aber für den Fall $l_n = -1$ $2^{l_n} = \frac{1}{2}$ gilt, ersetzen wir in der Abschätzung von **Fall 1** 2^{l_s} durch c_{l_s} mit $c_{l_s} = \begin{cases} 1 & \text{wenn } l_s = -1 \\ 2^{l_s} & \text{sonst} \end{cases}$. Somit sind dann alle Fälle abgedeckt und $\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\|$ kann für alle Fälle abgeschätzt werden durch:

$$\|\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})\| < \prod_{s=1}^{d} c_{l_s} \left[\sum_{k=1}^{d} \binom{d}{k} \mu_2^k 2^{d-k} \right] = c_{\mathbf{l}} \left[\sum_{k=1}^{d} \binom{d}{k} \mu_2^k 2^{d-k} \right].$$

Das ergibt dann für die Interpolationsfehlerabschätzung von u_3:

$$\begin{aligned}
\|u_3 - \hat{u}_3^{\epsilon_3}\| &= \left\| \sum_{|\mathbf{l}|_1=-1}^{\infty} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(u_2) - \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right\| \\
&\leq \left\| \sum_{|\mathbf{l}|_1=l_{u_1}+1}^{\infty} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(u_2) \right\| + \left\| \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(u_2) - \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right\| \\
&< \mu_1 + \left\| \sum_{|\mathbf{l}|_1=-1}^{l_{u_1}} \sum_{\mathbf{i}\in I_\mathbf{l}} u^1_{\mathbf{l},\mathbf{i}} \left[\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right] \right\| \\
&\leq \mu_1 + \left\| \sum_{k=-1}^{l_{u_1}} \max_{k=|\mathbf{l}|_1, \mathbf{i}\in I_\mathbf{l}} \left\{ u^1_{\mathbf{l},\mathbf{i}} [\phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2})] \right\} \right\| \\
&\leq \mu_1 + \sum_{k=-1}^{l_{u_1}} \max_{k=|\mathbf{l}|_1, \mathbf{i}\in I_\mathbf{l}} \left\{ \left| u^1_{\mathbf{l},\mathbf{i}} \right| \left\| \phi_{\mathbf{l},\mathbf{i}}(u_2) - \phi_{\mathbf{l},\mathbf{i}}(\hat{u}_2^{\epsilon_2}) \right\| \right\} \\
&< \mu_1 + \sum_{k=-1}^{l_{u_1}} \max_{k=|\mathbf{l}|_1, \mathbf{i}\in I_\mathbf{l}} \left\{ \left| u^1_{\mathbf{l},\mathbf{i}} \right| c_\mathbf{l} \left[\sum_{k=1}^{d} 2^{d-k} \binom{d}{k} \mu_2^k \right] \right\} \\
&= \mu_1 + \sum_{k=-1}^{l_{u_1}} \max_{k=|\mathbf{l}|_1, \mathbf{i}\in I_\mathbf{l}} \left\{ \left| u^1_{\mathbf{l},\mathbf{i}} \right| c_\mathbf{l} \right\} \left[\sum_{k=1}^{d} 2^{d-k} \binom{d}{k} \mu_2^k \right]
\end{aligned}$$

q.e.d.

Damit haben wir gezeigt, dass der Interpolationsfehler für die Hintereinanderschaltung zweier Interpolanten von den Interpolationsfehlern der Interpolanten additiv abhängt.

5 Algorithmen

Bis hierhin haben wir uns hauptsächlich mit der Theorie dünner Gitter beschäftigt. Dabei wurde theoretisch erläutert wie, ausgehend von der Idee der hierarchischen Basis und des dazugehörigen Differenzraumschemas, dünne Gitter erzeugt werden können. Danach wurden Konvergenzraten und Interpolationsfehler diskutiert, die wir bei einer Funktionsinterpolation auf dünnen Gittern erhalten. Schließlich wurden die Operatoren, die wir auf dünnen Gittern benutzen können, vorgestellt. In diesem und dem nachfolgenden Kapitel besprechen wir nun die Implementierung eines dünnes Gitters zur Funktionsinterpolation. Schwerpunkt dieses Kapitels ist dabei die Besprechung grundlegender Algorithmen, die wir zur effizienten Umsetzung benötigen.

Zuerst wird der Tröpfchenalgorithmus vorgestellt, mit dem u.a. die Punkte eines dünnen Gitters berechnet werden können. Danach wird ein Algortihmus erläutert mit dem man die Dünngitter-Basiskoeffizienten berechnen kann. Mit Hilfe dieser beiden Algorithmen wird dann ein Algorithmus zur Konstruktion eines adaptiven Dünngitter-Interpolanten entwickelt, welcher erstmal zur Interpolation von Funktionen der Form $f : \bar{\Omega} \to \mathbb{R}$ mit $\bar{\Omega} = [0, 1]^d$ verwendet werden kann. Dieser Algorithmus wird dann nachfolgend zur Interpolation vektorwertiger Funktionen verallgemeinert und dann auf die Interpolation hintereinandergeschalteter Funktionen erweitert. Abschließend besprechen wir einen ganz neuen Ansatz zur Funktionsinterpolation durch Hintereinanderschaltung von Funktionen, mit dem es möglich ist, Funktionen mit Polstellen signifikant besser zu interpolieren als mit einem einfachen adaptiven Dünngitter-Interpolationsverfahren.

Nachfolgend sei, wenn nicht anders angegeben, mit der Bezeichnung *dünnes Gitter* ein dünnes Gitter mit konstantem Rand (DGK) gemeint. Weiter gibt die Variable d die Dimension eines Gitters, bzw. der zu approximierenden Funktion f an. Sei $L_{\max}$ das maximale Level des Gitters. Desweiteren sei der Levelmultiindex $\mathbf{l} = (l_1, \ldots, l_d)$, der Ortsmutliindex $\mathbf{i} = (i_1, \ldots, i_d)$, der Punkt $x_{\mathbf{li}} = (x_{l_1 i_1}, \ldots, x_{l_d i_d})$ und der hierarchische Überschuss $u_{\mathbf{li}}$ wie in Kapitel 2 definiert. Mit $f_{\mathbf{li}}$ sei der Funktionswert der Funktion f an der Stelle $x_{\mathbf{li}}$ gemeint.

5.1 Tröpfchenalgorithmus

Um ein dünnes Gitter aufzubauen und dann dadurch die Vorteile des Differenzraumschemas nutzen zu können, benötigen wir einen Algorithmus der auf direktem Wege effizient alle Indizes und Punkte eines dünnen Gitters erstellen kann. Aufbauend auf dem allgemeinen Tröpfchenalgorithmus wird in diesem Abschnitt ein Tröpfchenalgorithmus zur Erstellung eines DGKs vorgestellt. Wir müssen nämlich alle möglichen Kombinationen von $(\mathbf{l}, \mathbf{i})$ mit $\mathbf{l} = (l_1, \ldots, l_d)$ und $\mathbf{i} = (i_1, \ldots, i_d)$ eines dünnen Gitters erstellen.
Der allgemeine Tröpchenalgorithmus erlaubt uns eine beliebige Menge von Multiindizes folgend einer gegebenen Bedingung zu erstellen. Angenommen wir suchen eine Menge von Vektoren

M_k, in der alle Vektoren der Menge die Bedingung B_k und eine gegebene Abbruchbedingung A_k erfüllen, dann sieht der dazugehörige allgemeine Tröpfchenalgorithmus wie folgt aus:

Sei $\mathbf{k} = (k_1, \ldots, k_n)$ ein Vektor mit n Einträgen und gesucht eine Menge $M_k = \{\mathbf{k} : \mathbf{k} \text{ erfüllt Bedingung } B_k \text{ und Abbruchbedingung } A_k\}$. Der Algorithmus 5.1 bestimmt diese Menge M_k, indem eine **while** -Schleife mit einer **if** -Abfrage kombiniert wird.

Algo. 5.1 *allgemeiner Tröpfchenalgorithmus:*

```
for i = 1 to n do
    k_i = k_Start
end
i = 1
while k erfüllt A_k do
        if k erfüllt nicht Bedingung B_k
          then
                k_i = k_Start
                i = i + 1
           else
                speichere k in M_k
                i = 1
        fi
        k_i = k_i + 1
end
```

Zur Erstellungen eines dünnen Gitters brauchen wir den Tröpfchenalgorithmus an zwei Stellen. Einmal bei der Konstruktion des Levelindex **l** und einmal für den Ortsindex **i**. Durch Kombination beider Indizes wird ein Punkt eines dünnen Gitters eindeutig bestimmt. Da der Ortsindex vom Levelindex abhängt, besteht der Tröpfchenalgorithmus zur Erstellung eines dünnen Gitters aus zwei ineinandergeschachtelten Tröpfchenalgorithmen. Der äußere Tröpfchenalgorithmus im Algorithmus 5.2 bestimmt alle möglichen Levelkombinationen der l_1 bis l_d bis zum maximalen Level $L_{\max}$ über die Bedingung, dass $\sum_{t=1}^{d} l_t \leq L_{\max} - d + 1$ gilt. Genügt der Levelindex **l** dieser Bedingung, wird im inneren Tröpfchenalgorithmus die möglichen Ortsindizes erzeugt. Da zu jedem Level mehrere Punkte existieren, wird die Eindeutigkeit erst durch den Ortsindex **i** sichergestellt.

Mit dem Algorithmus 5.2 können wir nun ein reguläres dünnes Gitter aufstellen, mit dem wir nun eine beliebige Funktion $f : \bar{\Omega} \to \mathbb{R}$ mit $\bar{\Omega} = [0, 1]^d$ interpolieren können.

Algo. 5.2 *Tröpfchenalgorithmus eines DGKs:*

```
for j = 1 to d do
    l_j = −1
end
j = 1
while l_d ≤ L_max do
    if Σ_{t=1}^{d} l_t > L_max − d + 1
      then
          l_j = −1
          j = j + 1
       else
          for t = 1 to d do
              i_t = 1
          end
          t = 1
          while (i_d ≤ 2^{l_d−1} ∧ l_d > 0) ∨ (i_d ≤ 1 ∧ l_d ≤ 0) do
              if (i_t ≤ 2^{l_t−1} ∧ l_t > 0) ∨ (i_t ≤ 1 ∧ l_t ≤ 0)
                then
                    i_t = 1
                    t = t + 1
                 else
                    speichere den Punkt(l, i)
                    t = 1
              fi
              i_t = i_t + 2
          end
          j = 0
    fi
    l_j = l_j + 1
end
```

5.2 Berechnung der Dünngitter-Basiskoeffizienten

Zur Interpolation einer Funktion f mit einem dünnen Gitter (regulär und adaptiv) benötigen wir die Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}$ (Definitionen 2.4, 2.5 und 2.6) und deren Basiskoeffizienten. Durch die Basisfunktionen wird die Funktion f stückweise linear interpoliert. Die Basiskoeffizienten der Basisfunktionen müssen aber erst über die Gleichungen 5.1 und 5.2 berechnet werden. Da wir mit der hierachischen Basis arbeiten, werden die Basiskoeffizienten hierarchische Überschüsse (HÜ) genannt. Die Erzeugung der hierarchischen Überschüsse wird als Hierarchisierung bezeichnet. Auf Grund der Tensorproduktdarstellung unserer Basisfunktionen berechnet sich der HÜ aus $d-1$ Übergangsüberschüssen (ÜÜ). Für jede Dimensionsrichtung muss ein Basiskoeffizient bzw. ÜÜ bestimmt werden, wobei der ÜÜ der Richtung k aus dem ÜÜ der Richtung $k-1$ bestimmt wird. Der HÜ ist also der letzte berechnete ÜÜ.
Sei $u_{\mathbf{li}}$ der HÜ an der Stelle $x_{\mathbf{li}}$ und $f_{\mathbf{li}}$ der Funktionswert an der Stelle $x_{\mathbf{li}}$. Sei :$\mathrm{T}_{\mathbf{li}}$: der Tröpf-

chenalgorithmus, der alle (**li**) mit $l_j = 0$ für mindestens ein j bestimmt. Dann lässt sich eine beliebige Funktion f auf einem DGK wie folgt hierarchisieren:

Algo. 5.3 *Berechnung der HÜ:*

setze $u_{\mathbf{li}} = f_{\mathbf{li}} \; \forall$ **li**
for $t = 1$ **to** d **do**
  begin : $\mathrm{T}_{\mathbf{li}}$:
    for $k = 1$ **to** d **do**
      $l_k^{\text{links}} = l_k$
      $l_k^{\text{rechts}} = l_k$
      $i_k^{\text{links}} = i_k$
      $i_k^{\text{rechts}} = i_k$
    end
    $l_t^{\text{links}} = -1, \; i_t^{\text{links}} = 1$
    $l_t^{\text{rechts}} = 0, \; i_t^{\text{rechts}} = 1$
    $links = f_{\mathbf{l}^{\text{links}}\mathbf{i}^{\text{links}}}$
    $rechts = f_{\mathbf{l}^{\text{rechts}}\mathbf{i}^{\text{rechts}}}$
    $H(u_{\mathbf{li}}, links, rechts, t)$
  end
end

dazu benötigte Hierarchisierungsroutine:

proc $H(u_{\mathbf{li}}, links, rechts, t)$
  if $l_t = -1$
    then
      $u_{\mathbf{li}} = u_{\mathbf{li}}$
  elsif $l = 0$
    then
      $c = u_{\mathbf{li}}$
      $u_{\mathbf{li}} = u_{\mathbf{li}} - links$
      for $k = 1$ **to** d **do**
        $l_k^{\text{links}} = l_k$
        $i_k^{\text{links}} = i_k$
      end
      $l_t^{\text{links}} = l_t + 1$
      $i_t^{\text{links}} = 2 \cdot i_t - 1$
      $H(u_{\mathbf{l}^{\text{links}}\mathbf{i}^{\text{links}}}, links, c, t)$
    else
      $c = u_{\mathbf{li}}$
      $u_{\mathbf{li}} = u_{\mathbf{li}} - \frac{links+rechts}{2}$
      for $k = 1$ **to** d **do**
        $l_k^{\text{links}} = l_k$
        $l_k^{\text{rechts}} = l_k$
        $i_k^{\text{links}} = i_k$
        $i_k^{\text{rechts}} = i_k$
      end
      $l_t^{\text{links}} = l_t + 1$
      $i_t^{\text{links}} = 2 \cdot i_t - 1$
      $l_t^{\text{rechts}} = l_t + 1$
      $i_t^{\text{rechts}} = 2 \cdot i_t + 1$
      $H(u_{\mathbf{l}^{\text{links}}\mathbf{i}^{\text{links}}}, links, c, t)$
      $H(u_{\mathbf{l}^{\text{rechts}}\mathbf{i}^{\text{rechts}}}, c, rechts, t)$
  fi
end

5.3 Algorithmen zur Konstruktion eines adaptiven Dünngitter-Interpolanten

Nachdem wir jetzt gesehen haben, wie wir die Basiskoeffizienten auf einem regulären dünnen Gitter berechnen können, gehen wir jetzt einen Schritt weiter und zeigen wie wir dies für ein adaptives dünnes Gitter tun können.
Adaptive Gitter unterscheiden sich von nicht adaptiven Gittern darin, dass in Ersterem nur die Punkte des Gitters benutzt werden, die zur Approximation wirklich nötig sind, um eine

vorgegebene Genauigkeit bei der Interpolation einer Originalfunktion zu erreichen. Aus diesem Grund muss die Berechnung der Basiskoeffizienten ganz anders angegangen werden als bei einem regulären DGK.

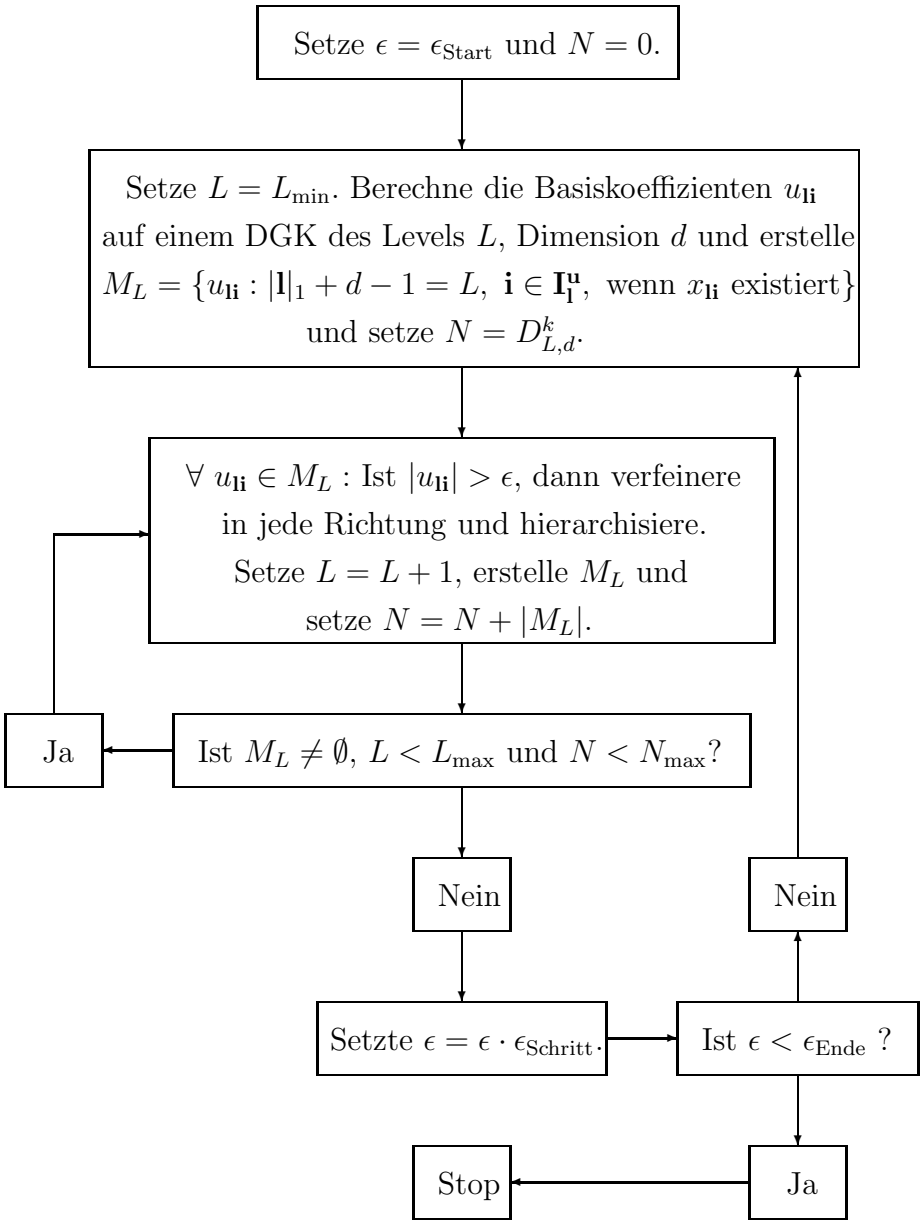


Abb. 5.1: Das Diagramm zeigt den Algorithmus zur Konstruktion eines adaptiven DGK-Interpolanten. Dabei sei $L, L_{\min}, L_{\max} \in \mathbb{N}_{0,-1}$, $d, N_{\max} \in \mathbb{N}$, $N \in \mathbb{N}_0$, $\mathbf{l} \in \mathbb{N}_{0,-1}^d$ und $\epsilon, \epsilon_{\text{Schritt}}, \epsilon_{\text{Start}}, \epsilon_{\text{Ende}} \in \mathbb{R}$

Als erstes betrachten wir deswegen das Diagramm in Abbildung 5.1. Dieses Diagramm zeigt,

wie der Interpolant einer Funktion mit einem adaptiven DGK-Verfahren bestimmt werden kann.
Noch ein paar Anmerkungen zum Diagramm (Abb. 5.1): Mit N können wir die Punktanzahl des Gitters überwachen. Wir starten mit der Gesamtzahl eines DGKs zum Level L, $D^k_{L,d}$, und erhöhen die Anzahl der Punkte um die hinzugewonnenen Punkte immer dann, wenn das Level erhöht wird. Wenn die Basiskoeffizienten kleiner sind als das vorgegebene ϵ_{Ende}, wird die Routine abgebrochen. Denn dann beschreibt der Interpolant die Originalfunktion genau genug. $L_{\max}$ und $N_{\max}$ sind zwei Abbruchbedingungen, die in dem Fall weiterhelfen, wenn die Basiskoeffizienten bei steigendem Level nicht kleiner werden, also die Menge M_L nicht leer wird. Ohne diese Abbruchbedingungen würde es zwangsläufig zu Speicherproblemen kommen, die mit einer fest vorgegebenen maximalen Punktanzahl $N_{\max}$ oder mit einem fest vorgebenen maximalen Level $L_{\max}$ verhindert werden können.

Der im obigen Diagramm beschriebene Algorithmus, der hier den Interpolanten für den Fall eines ADGK berechnet, kann leicht abgeändert werden, so dass Interpolanten basierend auf anderen Gittern analog berechnet werden können. Wenn man zum Beispiel die Bedingung $|\mathbf{l}|_1 + d - 1 = L$ für das DG in $|\mathbf{l}|_1 - d + 1 + z_0 = L$ umändert, dann erhält man ein ADG-Verfahren zur Konstruktion eines Interpolanten (s. Abschnitt 2.3 für die Definition von z_0).
In den folgenden Unterabschnitten gehen wir auf den bisher noch nicht erläuterten Punkt der Hierarchisierung im Algorithmus ein und nehmen diesen näher unter die Lupe.

5.3.1 Bestimmung der direkten Nachbarn eines Punktes eines dünnen Gitters

Die Berechnung der Basiskoeffizienten eines adaptiven dünnen Gitters ist komplizierter als bei einem regulären dünnen Gitter, da die HÜ einzeln hintereinander und nicht gleichzeitig bestimmt werden.
Sei X ein beliebiger Punkt, aber kein Randpunkt, an dem der HÜ berechnet werden soll, seien X^k_l der linke direkte Nachbar und X^k_r der rechte direkte Nachbar in Dimensionsrichtung k für $k = 1, \ldots, d$. Sei F der Funktionswert zu X, F_l der Funktionswert zu X_l und F_r der Funktionswert zu X_r. Und schließlich seien U^k, U^k_l und U^k_r die ÜÜ in die Dimensionsrichtung k für $k = 1, \ldots, d$ an den Punkten X, X_l und X_r, dabei sind U^d, U^d_l und U^d_r die endgültigen hierarchischen Überschüsse. Dann kann man den HÜ an der Stelle X wie folgt berechnen:

$$U^1 = F - \frac{F_l + F_r}{2} \tag{5.1}$$

$$U^k = U^{k-1} - \frac{U^{k-1}_l + U^{k-1}_r}{2} \quad \text{für } k = 1, \ldots, d \tag{5.2}$$

Falls X ein Randpunkt ist, wird er zwar in der gleichen Reihenfolge berechnet, aber mit leicht abgeänderten Formeln (s. Algorithmus 5.5).
Da in jede Dimensionsrichtung ein Übergangsüberschuss bestimmt werden muss, muss jeder direkte Nachbar und jeder zum direkten Nachbar zugehörige Übergangsüberschuss existieren und wenn nicht, errechnet werden. Deswegen müssen wir zuerst die direkten Nachbarn des Punktes, an dem der HÜ berechnet werden soll, in jede Dimensionsrichtung suchen. Die direkten Nachbarn eines Punktes des Levels k liegen auf Leveln $k_{Nachbar} < k$, was die Bestimmung

etwas erschwert.

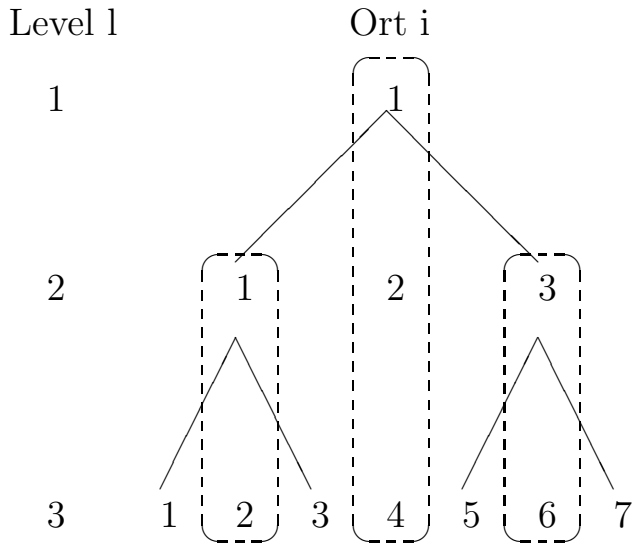


Abb. 5.2: Beschreibung der 7 inneren Punkte eines $G^{3,1,D}$ durch die Kombination vom Level l und dem Ort i

Diese Problem wird an folgendem Beispiel deutlich. Betrachten wir dazu die inneren Punkte eines regulären dünnen Gitters in einer Dimension und zum dritten Level. Wir erinnern uns, dass das $G^{3,1,D}$ sieben innere Punkte und zwei Randpunkte besitzt. In Abbildung 5.2 sieht man, dass man die drei gestrichelt umrandeten inneren Punkte in unterschiedlicher Weise schreiben kann. Wir werden hier die Punkte mittel des Levelindex und des Ortsindex $(\mathbf{l}, \mathbf{i})$ beschreiben. Also ist mit (1,1), (2,2) und (3,4) derselbe Punkt gemeint, das gleiche gilt für (2,1) und (3,2), sowie für (2,3) und (3,6). Sei nun (3,5) der Punkt zu dem wir den linken und rechten Nachbarn suchen. Der linke Nachbar ist (3,4) und der rechte Nachbar ist (3,6). Da wir die Punkte nur unter ihrem niedrigsten Level abspeichern, um sie nicht mehrmals speichern zu müssen, brauchen wir einen Rechenweg um einen Punkt der mittels höherer Level beschrieben wird, auf eine Schreibweise zum niedrigsten Level herunterzubrechen. Hier im Beispiel also vom Punkt (3,4) im hohem Level zu (1,1) im niedrigsten Level und vom Punkt (3,6) zu (2,3). Wir erhalten also, wenn wir den Ortsindex i^{alt}, den Ortsindex des höheren Levels, in ein Produkt von $2^{Exponent}$ und $Rest$ aufteilen, eine Möglichkeit den von uns gesuchten Ortsindex i^{neu}, den Ortsindex des niedrigsten Levels, daraus zu errechnen.

$\mathbf{i}^{alt}$	$\mathbf{l}^{alt}$	$\mathbf{i}^{neu}$	$\mathbf{l}^{neu}$
$4 = 2^2 \cdot 1$	3	1	3-2=1
$6 = 2^1 \cdot 3$	3	3	3-1=1

Aus den Überlgungen zur Berechnung des Ortsindex und des Levelindex können wir einen Algorithmus herleiten, mit dem wir den direkten Nachbar eines beliebigen Gitterpunktes be-

stimmen können.
Für unseren Algorithmus zur Bestimmung der direkten Nachbarn (Algorithmus 5.4) sei l der Level unseres Punktes und i der dazugehörende Ort. In Richtung s seien der linke Nachbar über (l^l, i^l) und der rechte Nachbar über (l^r, i^r) definiert.

Algo. 5.4 *Bestimmung der direkten Nachbarn*

```
for k = 1 to d do
    if s = k
      then
            if i_k = 1
              then
                    i_k^l = 1
                    l_k^l = −1
               else
                    i_k^l = rest (i_k − 1)
                    l_k^l = l_k − expo (i_k − 1)
            fi
            if (i_k = 2^{l_k} − 1) ∨ (l_k = 0)
              then
                    i_k^r = 1
                    l_k^r = 0
               else
                    i^r = rest (i_k + 1)
                    l^r = l_k − expo (i_k + 1)
            fi
       else
            i_k^l = i_k
            i_k^r = i_k
            l_k^l = l_k
            l_k^r = l_k
    fi
end
```

Dazu benötigte Routinen:

```
proc rest(i)
  while 0 ≡ i mod 2 do
        i = i/2
  end
  return i
end

proc expo(i)
  t = 0
  while 0 ≡ i mod 2 do
        i = i/2
        t = t + 1
  end
  return t
end
```

Es bleiben noch die Routinen rest(i) und expo(i) des Algorithmus zu erklären. Die Routine rest(i) teilt i in Faktoren auf, in ein Produkt von Zweien und in die übrigen (restlichen) Faktoren. Die restlichen Faktoren sind das Ergebnis der Routine rest(i). Das heißt, dass die Routine rest(i) das gesuchte neue i berechnet. Die Routine expo(i) teilt i in dieselben Faktoren auf, aber als Ergebnis erhalten wir die Anzahl des Faktors Zwei in i, also den Exponenten von Zwei. Mit diesem Exponenten können wir leicht das neue l errechnen, indem wir vom alten l den Exponenten abziehen.

5.3.2 Berechnung des Dünngitter-Basiskoeffizienten an einem beliebigen Punkt

Algo. 5.5 *Berechnung des HÜ am Punkt* $x_{\mathbf{l}\mathbf{i}}$*:*

proc $HP(x_{\mathbf{l}\mathbf{i}})$
  for t **to** d **do**
    $N(\mathbf{l}, \mathbf{i}, t)$
    if $x_{\mathbf{l}^l\mathbf{i}^l}$ nicht existent
      then
        $HP(x_{\mathbf{l}^l\mathbf{i}^l})$
    fi
    if $x_{\mathbf{l}^r\mathbf{i}^r}$ nicht existent
      then
        $HP(x_{\mathbf{l}^r\mathbf{i}^r})$
    fi
    if $t = 1$
      then
        if $l_1 < 0$
          then
            $u^1_{\mathbf{l}\mathbf{i}} = f_{\mathbf{l}\mathbf{i}}$
        elsif $l_1 = 0$
          then
            $u^1_{\mathbf{l}\mathbf{i}} = f_{\mathbf{l}\mathbf{i}} - f_{\mathbf{l}^l\mathbf{i}^l}$
          else
            $u^1_{\mathbf{l}\mathbf{i}} = f_{\mathbf{l}\mathbf{i}} - \frac{f_{\mathbf{l}^l\mathbf{i}^l} + f_{\mathbf{l}^r\mathbf{i}^r}}{2}$
        fi
      else
        if $l_t < 0$
          then
            $u^t_{\mathbf{l}\mathbf{i}} = u^{t-1}_{\mathbf{l}\mathbf{i}}$
        elsif $l_t = 0$
          then
            $u^t_{\mathbf{l}\mathbf{i}} = u^{t-1}_{\mathbf{l}\mathbf{i}} - u^{t-1}_{\mathbf{l}^l\mathbf{i}^l}$
          else
            $u^t_{\mathbf{l}\mathbf{i}} = u^{t-1}_{\mathbf{l}\mathbf{i}} - \frac{u^{t-1}_{\mathbf{l}^l\mathbf{i}^l} + u^{t-1}_{\mathbf{l}^r\mathbf{i}^r}}{2}$
        fi
    fi
  end
end

Mit Hilfe des im vorherigen Abschnitt vorgestellten Algorithmus 5.4 können wir die direkten Nachbarn eines beliebigen Gitterpunktes bestimmen. Somit kommen wir dem Ziel ein adaptives Dünngitter-Interpolationsverfahren zu konstruieren schon etwas näher. Wir müssen uns jetzt nur noch eine Routine überlegen, die die Hierarchisierung an einem beliebigen Punkt durchführt. Den Algorithmus zur Bestimmung der Nachbarn in Richtung s werden wir mit $N(\mathbf{l}, \mathbf{i}, s)$ bezeichnen und den linken Nachbar mit $x_{\mathbf{l}^l\mathbf{i}^l}$ und den rechten mit $x_{\mathbf{l}^r\mathbf{i}^r}$.

Seien $u_{\mathbf{li}}^t,\ t = 1 \ldots d$, die ÜÜ zum Punkt $x_{\mathbf{li}}$, wobei $u_{\mathbf{li}}^d$ der endgültige HÜ ist, und $f_{\mathbf{li}}$ der Funktionswert an der Stelle $x_{\mathbf{li}}$. Dann kann man an diesem Punkt eines ADGKs die Basiskoeffizienten, wie in Algorithmus 5.5 zu sehen, bestimmen.
Zuerst werden die HÜ an den direkten Nachbarpunkten und dann wird jeder ÜÜ an der Stelle $x_{\mathbf{li}}$ berechnet.

5.3.3 Verbesserung der Berechnung der Dünngitter-Basiskoeffizienten durch den Lookaheadalgorithmus

Solange die Basiskoeffizienten mit steigendem Level abfallend sind, reichen die Algorithmen, die wir bis jetzt besprochen haben, vollkommen aus zur Funktionsinterpolation auf adaptiven dünnen Gittern. Wenn dies aber nicht der Fall ist oder hierarchische Überschüsse vom Wert Null vorkommen, kann das adaptive Dünngitter-Verfahren nicht alle notwendigen HÜ bestimmen und der adaptive Dünngitter-Interpolant wird nicht so genau wie der reguläre Dünngitter-Interpolant. Wenn ein HÜ vom Wert Null vorkommt, wird das Gitter an dieser Stelle nicht verfeinert und somit kann der HÜ an der verfeinerten Stelle nicht berechnet werden, obwohl dieser HÜ eventuell wesentlich für den Interpolanten ist.
Die Sinusfunktion ist ein gutes Beispiel für diesen Fall. Sei $u(x) = \sin(2\pi x)$ mit $x \in \bar{\Omega} = [0, 1]$, dann ist u an den Stellen 0, $\frac{1}{2}$ und 1 gleich 0. Zu diesen Punkten werden keine Söhne ezeugt und so das adaptive dünn Gitter nicht erweitert. Das heißt also, dass kein Punkt des zweiten Levels erzeugt wird, obwohl die Funktion nicht konstant Null ist sondern komplexer. Also müssen wir einen Algorithmus finden, um diesem Problem aus dem Weg gehen zu können.

Algo. 5.6 *einfacher Lookahead eines ADGKs*

```
proc la1(x)
  HP(x)
  wert = u
  for k to d do
      Söhne(x, k)
      if ∄ u^{l(k)} then HP(x^{l(k)}) fi
      if ∄ u^{r(k)} then HP(x^{r(k)}) fi
      wert = wert + u^{l(k)} + u^{r(k)}
  end
  if wert < ε
    then
          for k to d do
              lösche x^{l(k)}
              lösche x^{r(k)}
          end
  fi
end
```

Berechnung der Söhne in die kte Richtung

```
proc Söhne(x, k)
  for t to d do
      if t = k
        then
              l_k^{lr} = l_k + 1
              i_k^l = 2 · i_k − 1
              i_k^r = 2 · i_k + 1
         else
              l_t^{lr} = l_t
              i_t^l = i_t
              i_t^r = i_t
      fi
  end
  return x^{l(k)} ∧ x^{r(k)}.
end
```

Eine Lösung bietet der Lookaheadalgorithmus (Algorithmus 5.6). In diesem Algorithmus betrachtet man nicht nur den HÜ $u_{\mathbf{li}}$, sondern zusätzlich die HÜ der Söhne. Das heisst, man addiert den HÜ am Punkt $x_{\mathbf{li}}$ zu den HÜen an den Söhnen des Punktes, vergleicht diesen Wert mit dem ϵ aus dem Algorithmus 5.1 und entscheidet dann, ob die Söhne ins Git-

ter aufgenommen werden oder nicht. Diese Idee kann man erweitern, indem man nicht nur die Söhne, sondern auch die Söhne der Söhne, also die Enkel, betrachtet. Schaue dazu den Algorithmus 5.7 an. Wir werden beide Varianten behandeln, wobei Schwierigkeiten bei der Implementieren dieser Algorithmen auftauchen, die in Kapitel 6 näher beschrieben werden.

Algo. 5.7 *zweifacher Lookahead eines ADGKs*

```
proc la2(x)
  HP(x)
  wert = u
  for k to d do
      Söhne(x, k)
      if ∄ u^l(k) then HP(x^l(k)) fi
      if ∄ u^r(k) then HP(x^r(k)) fi
      wert = wert + u^l(k) + u^r(k)
      for t to d do
          Söhne(x^l(k), t)
          if ∄ u^l(k)l(t) then HP(x^l(k)l(t)) fi
          if ∄ u^l(k)r(t) then HP(x^l(k)r(t)) fi
          wert = wert + u^l(k)l(t) + u^l(k)r(t)
          Söhne(x^r(k), t)
          if ∄ u^r(k)l(t) then HP(x^r(k)l(t)) fi
          if ∄ u^r(k)r(t) then HP(x^r(k)r(t)) fi
          wert = wert + u^r(k)l(t) + u^r(k)r(t)
      end
  end
  if wert < ϵ
    then
          for k to d do
              lösche x^l(k)
              lösche x^r(k)
              for t to d do
                  lösche x^l(k)l(t)
                  lösche x^r(k)l(t)
                  lösche x^l(k)r(t)
                  lösche x^r(k)r(t)
              end
          end
  fi
end
```

Folgend seien noch ein paar Anmerkungen zu den Algorithmen 5.6 und 5.7 gemacht. Die Variable ϵ ist wie in 5.1 definiert und x bezeichnet den zu überprüfenden Gitterpunkt, d.h ob an dieser Stelle verfeinert werden soll oder nicht. Aus Übersichtsgründen ist der Index **li** weggelassen worden. Mit $x^{l(k)}$ bezeichnen wir den linken Sohn von x in Richtung k und mit $x^{r(k)}$ den rechten Sohn von x in Richtung k. Desweiteren ist $x^{l(k)l(t)}$ der linke Sohn von $x^{l(k)}$ in Richtung

t, etc. Die dazugehörigen hierarchischen Überschüsse sind analog definiert. Zum Beispiel ist der HÜ an der Stelle $x^{l(k)l(t)}$ gleich $u^{l(k)l(t)}$. Für die Funktion $HP(x)$ betrachte den Algorithmus 5.5.
Durch diese Lookaheadalgorithmen haben wir die Menge der Funktionen, die wir interpolieren können, erweitert. Speziell die trigonometrischen Funktionen können nun besser interpoliert werden.

5.4 Interpolation von vektorwertigen Funktionen

Bis jetzt haben wir nur Algorithmen zur Interpolationen von Funktionen $f : [0,1]^d \to \mathbb{R}$ mit $d \in \mathbb{N}$ betrachtet. Befassen wir uns an Stelle dieser mit Funktionen $v : \mathbb{R}^d \to \mathbb{R}^m$ mit $m \in \mathbb{N}$. Nun haben wir nicht nur einen Funktionswert, sondern m Funktionswerte und m Teilfunktionen von v, also $v = (v_1, \ldots, v_m)$ zu interpolieren. Der Schritt von der Interpolation einfacher zu der Interpolation vektorwertiger Funktionen ist leicht vollzogen, da man nur eine zusätzliche Schleife in die bestehenden Algorithmen zur Funktionsinterpolation einbauen muss. Diese zusätzliche Schleife kann in den Algorithmen wie folgt auftreten:

(1) ineinander:
for $i = 1$ **to** d **do**
  for $j = 1$ **to** m **do**
  end
end

(2) ineinander:
for $j = 1$ **to** m **do**
  for $i = 1$ **to** d **do**
  end
end

(3) hintereinander:
for $i = 1$ **to** d **do**
end
for $j = 1$ **to** m **do**
end

Zur Berechnung des Funktionswerts wird zum Beispiel die letzte Variante benutzt, da ein und dasselbe $x = (x_1, \cdots, x_d)$ in m Teilfunktionen eingesetzt wird.

5.5 Interpolation von hintereinandergeschalteten Funktionen

Nun gehen wir noch einen Schritt weiter und betrachten die Interpolation von hintereinandergeschalteten Funktionen. Sei $G : [0,1]^d \to [0,1]^n$ und $F : [0,1]^n \to \mathbb{R}^m$ mit $d, n, m \in \mathbb{N}$, dann ist die Funktion $F(G) : [0,1]^d \to \mathbb{R}^m$ die Hintereinanderschaltung von F und G. Prinzipiell könnten wir diese Funktion $F(G)$ wie gewohnt interpolieren. Hier wollen wir aber erst die Funktion G interpolieren und dann mit dem Interpolanten von G die Funktion $F(G)$. Sei mit n das maximale Level bezeichnet, die Dimension d und die Abbruchbedingung ϵ_{Ende} fest gewählt, dann interpolieren wir $F(G)$ wie in Algorithmus 5.8 beschrieben.

Algo. 5.8 *Interpolation von F(G)*

interpoliere G auf einem adaptiven dünnen Gitter$(n,d)(1)$ mit ϵ_{Ende}
  $\Rightarrow \quad \tilde{G}$
interpoliere nun F mit $\tilde{G}$ auf einem adaptiven dünnen Gitter$(n,d)(2)$
  mit ϵ_{Ende}, dabei beachte:
    benutze für $x_{\mathbf{li}} \;\; \tilde{G}(x_{\mathbf{li}})$, d.h.
    benutze $F(\tilde{G}(x_{\mathbf{li}}))$ anstatt $F(x_{\mathbf{li}})$ an der Stelle $x_{\mathbf{li}}$
      $\Rightarrow \quad \tilde{F}(\tilde{G})$

Das maximale Level n und das ϵ_{Ende} des ersten Gitters müssen nicht unbedingt mit dem Level und der Abbruchbedingung des zweiten Gitters übereinstimmen. Es ist aber sinnvoll jeweils das gleiche Level und die gleiche Abbruchbedingung zu wählen. Wählt man nämlich für die erste Interpolation ein grobes Gitter, so entsteht mit bei der interpolation von G schon ein großer Fehler, der später mittels eines feinmaschigen Gitters bei der zweiten Interpolation nicht mehr ausgeglichen werden kann.

5.6 Funktionsinterpolation durch Hintereinanderschaltung von Funktionen

In Kapitel 4 haben wir gesehen, dass der Interpolationsfehler einer hintereinandergeschalteten Funktion $F = f(g)$ additiv durch die Fehler von f und g beschrieben werden kann. Die Idee, die daraus resultiert, ist, dass eine Funktion F unter Umständen besser durch zwei einzelne Funktionen f und g interpoliert werden kann. Die Frage ist nun, wie die Funktionen f und g aussehen müssen, damit F optimal interpoliert wird?
Betrachten wir dazu zuerst das ganze Problem in einer Dimension und fangen mit der Betrachtung des Fehlerfunktionals $\mathcal{F} = \|F(x) - f(g(x))\|^2_{L_2}$ an. Um ein Minimum von $\mathcal{F}$ zu erhalten, müssen wir die Ableitung $\mathcal{F}'$ berechnen und gleich Null setzen.
Seien die Funktionen f und g wie folgt gegeben:

$$f(y) = \sum_l \sum_i f_{li}\phi_{li}(y),\ g(x) = \sum_k \sum_j g_{kj}\phi_{kj}(x)$$

und $H = \{f_{l,i}, g_{k,j} : l, k \in \mathbb{N}_{0,-1},\ i \in I^u_l \text{ und } j \in I^u_k\}$ die Menge der hierarchischen Überschüsse zu f und g. I^u_l sei definiert über

$$I^u_{\mathbf{l}} = \{\mathbf{i} \in \mathbb{N}^d_0 :\ i_j = 1, \ldots, 2^{l_j} - 1 \text{ mit } i_j \text{ ungerade für } l_j > 0 \text{ und } i_j = 1 \text{ für } l_j < 0, 1 \leq j \leq d\}$$

und I^u_k analog dazu. Dann ist die Ableitung von $\mathcal{F}$ in Richtung H gegeben als:

$$\frac{\partial \mathcal{F}}{\partial H} = \frac{\partial}{\partial H} \int |F(x) - f(g(x))|^2 \, dx$$

Schauen wir uns zuerst den Teil an, der in Richtung $f_{l,i}$ abgeleitet wird:

$$\begin{aligned}\frac{\partial \mathcal{F}}{\partial f_{l,i}} &= \int \frac{\partial}{\partial f_{l,i}} |F(x) - f(g(x))|^2 \, dx = \int \frac{\partial}{\partial f_{l,i}} \left([F(x)]^2 - 2F(x)f(g(x)) + [f(g(x))]^2\right) dx \\ &= \int -2F(x)\phi_{l,i}(g(x)) + 2f(g(x))\phi_{l,i}(g(x))dx = \int 2\phi_{l,i}(g(x)) \left[f(g(x)) - F(x)\right] dx\end{aligned}$$

Leiten wir nun in Richtung $g_{k,j}$ ab:

$$\begin{aligned}\frac{\partial \mathcal{F}}{\partial g_{k,j}} &= \int \frac{\partial}{\partial g_{k,j}} |F(x) - f(g(x))|^2 \, dx = \int \frac{\partial}{\partial g_{k,j}} \left([F(x)]^2 - 2F(x)f(g(x)) + [f(g(x))]^2\right) dx \\ &= \int \frac{\partial}{\partial g_{k,j}} \left[f(g(x))f(g(x)) - 2F(x)f(g(x))\right] dx\end{aligned}$$

Da $f(g(x)) = \sum_l \sum_i f_{l,i}\phi_{l,i}\left(\sum_k \sum_j g_{k,j}\phi_{k,j}(x)\right)$ gilt, ist die Ableitung in einer Dimension in Richtung $g_{k,j}$ schon hochgradig nichtlinear. Daher macht es wenig Sinn f und g über die Ableitung des Funktionals zu bestimmen. Wir müssen uns also eine andere Vorgehensweise überlegen, um die Funktionen f und g optimal zu wählen.

Eine Idee wäre es $\tilde{f}$ und $\tilde{g}$ abwechselnd über eine Iterationsvorschrift zu bestimmen, aber auf Grund des Ergebnis der Fehlerfunktionalableitung können wir diesen Weg nicht weiterverfolgen.
Eine praktikable Idee ist es, sich erstmal nur auf die Funktion g zu beschränken und diese über eine bestimmte Vorschrift zu bestimmen. Die Funktion f ergibt sich dann aus dem Zusammenhang von F und g. Die Entscheidung erst g zu wählen und dann f zu bestimmen und nicht umgekehrt liegt dem Umstand zu Grunde, dass der Interpolationsfehler der inneren Funktion g dominanter als der Interpolationsfehler der äußeren Funktion f in den Gesamtfehler eingeht und zudem diese Variante einfacher umzusetzen ist. Es zeigt sich, dass diese Vorgehensweise bei Funktionen mit Polstellen sehr gute Ergebnisse liefert.
Betrachten wir diese Idee in d Dimensionen. Sei $F : \bar{\Omega} \to \mathbb{R}^m$. Damit $f(g) = F$ gilt, muss $g : \bar{\Omega} \to \bar{\Omega}'$ und $f : \bar{\Omega}' \to \mathbb{R}^m$ gelten. In unserem Fall werden wir das Interpolationsproblem noch etwas vereinfachen, indem wir $g : \bar{\Omega} \to \bar{\Omega}$ und $f : \bar{\Omega} \to \mathbb{R}^m$ mit $\bar{\Omega} = [0,1]^d$ wählen. Außerdem muss g streng monoton steigend sein und für jeden Eckpunkt des Raumes $\bar{\Omega}$ gilt $\mathbf{x} = g(\mathbf{x})$, da sonst F nicht aus $f(g)$ eindeutig rekonstruiert werden kann.

Als nächstes werden wir diese Idee in einem Algorithmus festhalten. Wir berechnen dazu zuerst alle Funktionswerte zu F auf einem Produktgitter mit Maschenweite 2^{-L} und benennen diese Menge mit F_L. Dann berechnen wir nach einer bestimmten Vorschrift aus der Menge F_L die Menge der Funktionswerte von g auf demselben Produktgitter und interpolieren mit einem ADGK-Verfahren bis zum Level L, so dass wir den Interpolanten g_L erhalten. Anschließend berechnen wir mit Hilfe der Funktion F und dem Interpolanten g_L den adaptiven Dünngitter-Interpolanten f_L. Wir erhöhen dabei das Level L solange, bis der Interpolationsfehler zwischen $f_L(g_L)$ und F kleiner als ϵ ist oder das Level einen maximalen Wert MAX erreicht hat. Das ganze Verfahren ist im Algorithmus 5.9 zusammengefasst.

Algo. 5.9 *Funktionsinterpolation mit $f(g) = F$*

for $L = MINL$ **to** MAX **do**
  sample F_L auf einem Produktgitter mit Maschenweite 2^{-L}
  berechne g_L aus F_L
  berechne f_L aus g_L und F
  bestimme den $FEHLER$ von $f_L(g_L)$ zu F
  if $FEHLER < \epsilon$
    then
      stoppe
  fi
end

Die Teile "berechne g_L aus F_L" (T1) und "berechne f_L aus g_L und F"(T2) des Verfahrens werden wir noch gesondert betrachten, da der Algorithmus 5.9 sonst unübersichtlich würde.

Aber befassen wir uns zuerst mit Teil T2. Die Berechnung von f_L ist unabhängig davon wie g_L gewählt wird, da man g_L und F nur auswertet. Sei $\mathbf{x} = (x_1, \ldots, x_d)$ und $\mathbf{y} = (y_1, \ldots, y_d)$ mit $\mathbf{x}, \mathbf{y} \in \bar{\Omega}$, dann ist der Algorithmus zur Berechnung von f_L gegeben durch:

Algo. 5.10 *berechne f_L aus g_L und F*
interpoliere f auf einem ADGK mit maximalem Level L, dabei:
berechne $\mathbf{y}$ mit Hilfe eines Nullstellenverfahrens aus $\mathbf{x} = g_L(\mathbf{y})$
setze $f(\mathbf{x}) = F(\mathbf{y})$
erhalte f_L

Gehen wir nun zum Teil T2 über. Dabei müssen wir uns Gedanken machen, wie wir g bestimmen können und welche Eigenschaften g haben sollte, bzw. wie wir die Eigenschaften der Funktion F auf f und g aufteilen, so dass die Approximation bei diesem Verfahren optimal wird.
Es macht Sinn eine der beiden Funktionen f und g linear zu wählen, so dass die Interpolation dieser linearen Funktion mit wenig Aufwand betrieben werden kann. Betrachten wir also die Steigung zu F. Die Steigung von F kann in den Grad der Steigung, den Betrag der Ableitung von F, und die Richtung der Steigung, d.h., ob F fallend oder steigend ist, aufgeteilt werden. Da g in unserem Fall streng monoton steigend sein soll, kann g nur durch den Grad der Steigung charakterisiert werden. Wenn die Funktion g also mit dem Betrag der Ableitung von F gleichgesetzt wird, dann beschreibt g die Verteilung der Gitterpunkte durch die Funktion F in $\bar{\Omega}$. Ist g so gewählt wird f in den meisten Fällen zu einer linearen Funktion. Da g und f genauer interpoliert werden können als F, weil g und f nur Teile von F beschreiben, erwarten wir, dass F durch diese Wahl genauer oder mindestens genauso gut bei gleichem Aufwand beschrieben wird. Wie wir im übernächsten Kapitel sehen werden, werden speziell Funktionen mit Polstellen mit diesem Verfahren besser als mit einem einfachen adaptiven Dünngitter-Verfahren interpoliert. Mehr zur Bewertung des Verfahrens zur Funktionsinterpolation durch Hintereinanderschaltung findet sich in Kapitel 7.
Fassen wir abschließend die Methode im Algorithmus 5.11 zusammen. In diesem Algorithmus werden mit den Variablen F die zu approximiernde Funktion und $F_i = F(x_i)$ die Funktionswerte von F an den Stellen $x_i = i \cdot 2^{-L}$ für $i = 0 \ldots n$ mit $n = 2^L$ bezeichnet. ϵ ist eine Zahl, die zwischen 0 und 1 liegt und garantiert, dass die berechnete Funktion g streng monoton steigend wird. Dann kann der Algorithmus zur Berechnung der Funktion g aus F wie folgt geschrieben werden:

Algo. 5.11 *berechne g_L aus F_L*
$a_0 = |F_0| + \epsilon$
for $i = 1$ **to** n **do**
  $a_i = |F_i - F_{i-1}| + \epsilon$
end
for $i = 1$ **to** n **do**
  $a_i = a_i + a_{i-1}$
end
for $i = 0$ **to** n **do**
  $g_i = \frac{a_i - a_0}{a_n - a_0}$
end
interpoliere g auf einem ADGK mit maximalem Level L, daraus erhalte g_L

Fazit

Wir haben in diesem Kapitel die einzelnen Algorithmen betrachtet, die wir zur Konstruktion eines adaptiven Dünngitter-Interpolationsverfahren benötigen. Dabei sind wir besonders auf die Berechnung der Basiskoeffizienten an beliebigen Stellen des dünnen Gitters eingegangen. Eines der Probleme, das wir bei dieser Berechnung antrafen, war, dass die Basiskoeffizienten mit steigendem Level nicht abfallend sein konnten. Dieses Problem konnten wir teilweise mit dem Lookahead-Algorithmus beheben. Ein anderes Problem war die Bestimmung der direkten Nachbarn eines beliebigen Gitterpunktes, wozu wir uns eine geeignete Routine überlegt haben. Außerdem haben wir ausgeführt, wie wir vektorwertige Funktionen auf adaptiven dünnen Gittern und wie wir mit Hilfe des Interpolanten zur Funktion G die Hintereinanderschaltung $F(G)$ interpolieren können. Zum Schluss haben wir noch die Hintereinanderschaltung von zwei Interpolanten betrachtet und aus den Erkenntnisse dieser Betrachtung ein neues Verfahren zur Funktionsinterpolation konstruiert.
Jetzt brauchen wir uns nur noch über eine sinnvolle Datenstruktur und die effiziente Umsetzung dieser Algorithmen Gedanken zu machen.

6 Datenstruktur und Implementierung

Im vorigen Kapitel haben wir Algorithmen, die wir für die Konstruktion eines adaptiven Dünngitter-Interpolanten benötigen, und das neue Verfahren zur Funktionsinterpolation durch Hintereinanderschaltung vorgestellt. Jetzt wollen wir über die Implementierung dieser Algorithmen sprechen und die dafür benötigten Datenstrukturen diskutieren. Dabei beginnen wir mit der Besprechung des grunglegenden Problems des Abspeicherns der Daten. Es gibt mehrere Möglichkeiten die Gitterpunkte abzuspeichern, aber nur wenige Datenstrukturen erfüllen folgende gewünschte Eigenschaften:

- einfacher Zugriff auf beliebige Daten des dünnen Gitters, am besten mit Kosten der Ordnung $O(1)$ für einen Zugriff
- einfache Handhabbarkeit der Adaptivität
- geringer Speicheraufwand für einen einzelnen Gitterpunkt
- der Zugriff auf hierarchische Väter und Söhne muss einfach und schnell sein

Datenstrukturen, die diesen Anforderungen einigermaßen entsprechen und damit einen schnellen Zugriff auf die Gitterpunkte ermöglichen sind Hashtabellen.

6.1 Hashtabelle

Eine Hashtabelle ist vereinfacht ausgedrückt ein großer Array $A[\cdot]$, dessen grundlegende Eigenschaft es ist direkt auf beliebige Felder in A zuzugreifen.
Wenn wir Gitterpunkte in A abspeichern möchten, benötigen wir einen "Schlüssel", der jedem einzelnen Punkt eindeutig einen Platz in A zuweist. Dieser "Schlüssel" heißt Hashfunktion und berechnet aus einem gegebenen key den Platz in der Hashtabelle. In unserem Fall erweist sich $(\mathbf{l}, \mathbf{i})$ als key und die Hashfunktion

$$hfkt(\mathbf{l}, \mathbf{i}) = \sum_{j=1}^{d} (2^{l_j} + i_j) \cdot prim(j) \cdot prim(43 + (d-2) \cdot 10 - j)$$

als sinnvoll, wobei $prim(j)$ die j-te Primzahl ausgibt. Leider ist die Eindeutigkeit der Platzzuweisung im Speicher in der Praxis nicht immer gegeben und wir stehen vor dem Problem der Kollisionsauflösung. In diesem Fall gibt es zwei Lösungsansätze:

1. interne Kollisionsauflösung, d.h. Sprung zum nächsten freien Platz
2. externe Kollisionsauflösung, d.h. Abspeicherung in einer angehängten Liste

Wir werden die interne Kollisionsauflösung benutzen, da diese Methode einfacher zu implementieren ist. Wir müssen bei dieser Variante keine zusätzlichen Datenstrukturen erstellen.
Nun stellt sich die Frage, wie viele Informationen bei der Durchführung der Interpolation abgespeichert werden müssen. Wichtig ist für uns dabei, so wenig Speicherplatz wie nur möglich zu benutzen.

6.2 Datenstruktur und Implementierung bei einfachen Funktionen

Als Erstes betrachten wir die benötigten Datenstrukturen für das reguläre und adaptive Dünngitter-Interpolationsverfahren für einfache Funktionen. Mit einfachen Funktionen sind alle Funktionen $f : [0,1]^d \to \mathbb{R}$ gemeint, wobei d die Dimension angibt.
Um einen Punkt eindeutig bestimmen zu können, brauchen wir seinen Multi-Index $\mathbf{i}$, der die Lage des Gitterpunkts auf einem gegebenen festen Level $\mathbf{l}$ beschreibt. Zum Abspeichern beider Angaben brauchen wir schon einmal $2 \cdot d$ Arrayelemente. Außerdem müssen noch der Funktionswert und der hierarchische Überschuss zum Punkt abgespeichert werden, so dass insgesamt $2 \cdot d + 2$ Arrayelemente benötigt werden. Mit dieser Anzahl an Arrayelementen können einfache Funktionen nur auf einem regulären dünnen Gitter interpoliert werden, aber nicht auf einem adaptiven dünnen Gitter. Im adaptiven Fall kommmen die hierarchischen Übergangsüberschüsse noch dazu, wie dem Algorithmus 5.5 zu entnehmen ist. Demnach brauchen wir für den adaptiven Fall $3 \cdot d + 1$ Arrayelemente pro Gitterpunkt.
Während der Interpolation einer Funktion müssen mehrere Operationen auf der Hashtabelle durchgeführt werden. Folgende Routinen werden dafür benötigt:

- Eintragung eines Punktes in die Hashtabelle A
- Löschung eines Punktes aus der Hashtabelle A
- Existenzüberprüfung eines Punktes in der Hashtabelle A
- Eintragung des HÜ oder der ÜÜ zu einem gespeicherten Punkt
- Eintragung des Funktionswerts zu einem gespeicherten Punkt
- Auslesen der HÜ oder der ÜÜ zu einem gespeicherten Punkt
- Auslesen des Funktionswerts zu einem gespeicherten Punkt

Im Prinzip brauchen wir also für n Gitterpunkte im adaptiven Fall eine Hashtabelle mindestens der Arraylänge $n * (3 \cdot d + 1)$, weil an dem für einen Gitterpunkt mittels der Hashfunktion berechneten Platz die $3 \cdot d + 1$ nächsten Arrayelemente zur Speicherung der Multi-Indizes, des Funktionswerts, des HÜ und der ÜÜ benutzt werden. Wobei die Länge der Hashtabelle in der Praxis grer gewählt werden sollte, um die Notwendigkeit der internen Kollisionsauflösung gering zu halten.
Im adaptiven Fall müssen wir nicht nur die schon bearbeiteten Punkte in der Hashtabelle speichern, sondern auch die noch zu bearbeitenden Punkte temporär in einem Array oder Liste ablegen. Denn wenn wir uns beim hierarchisieren dafür entscheiden, dass das Gitter an der Stelle $x_{\mathbf{li}}$ verfeinert werden muss, müssen wir alle Söhne des Punktes $x_{\mathbf{li}}$ betrachten. Wir

können die einzelnen Söhne aber nur nacheinander darauf überprüfen, ob das Gitter an diesen Stellen verfeinert werden muss oder nicht. Deswegen müssen die noch zu bearbeiteten Punkte irgendwo zwischengespeichert werden. Die Fragen, die sich nun stellen, sind, wie wir diese Punkte zwischenspeichern und welche Informationen pro Punkt gespeichert werden müssen. Da es uns reicht die zeitliche Speicherreihenfolge der Punkte und die Menge an Punkten zu kennen, haben wir zwei Möglichkeiten:

1. Entweder man erstellt ein Array, in dem alles nacheinander abgespeichert wird und in dem direkt auf einen Eintrag zugegriffen werden kann,
2. oder man erstellt eine Liste, die man von Anfang bis zum gewünschten Eintrag durchlaufen muss, um an den Eintrag zu gelangen.

Wir werden die erste Möglichkeit wählen und ein Array, welches wir mit N bezeichnen, benutzen. Für jeden Punkt müssen wir das Paar der Multi-Indizes (**li**) und den hierarchischen Überschuss in dem Array N abspeichern. Der Funktionswert und die Übergangüberschüsse sind hier nicht nötig, da sie bei der Überprüfung des Punktes nicht gebraucht werden. Also brauchen wir pro Punkt $2 \cdot d + 1$ Arrayelemente. Als nächstes müssen wir uns Gedanken darüber machen, wie wir dieses Array N zur Zwischenspeicherung geschickt nutzen und welche Routinen wir dafür benötigen. Wie in Abbildung 5.1 zu sehen, starten wir die Interpolation mit einem Startgitter des Levels L. Dann werden nur die Punkte dieses Levels in das Array N eingetragen. Sobald der erste Punkt aus N bearbeitet wurde, wird er aus N gelöscht und alle übrigen Punkte um einen Punkt im Array nach vorne verschoben. Wenn hingegen dazu aber neue Punkte dazukommen, werden diese hinter dem letzten Punkt gespeichert. Wenn das Array N leer wird, bricht die Hierarchisierung ab. Wir benötigen also drei Routinen für die Zwischenabspeicherung.

1. Eintragung eines Punktes in den Array N
2. Löschung eines Punktes aus dem Array N
3. Auslesung der Daten eines Punktes des Arrays N

Damit haben wir nun vollständig, bis auf den Lookahead, die benötigten Datenstrukturen und zugehörigen Routinen für das reguläre und adaptive Dünngitter-Interpolationsverfahren für einfache Funktionen beschrieben.

6.3 Datenstruktur und Implementierung bei vektorwertigen Funktionen

Wenden wir uns nun dem Fall der Interpolation vektorwertiger Funktionen zu. Wenn vektorwertige Funktionen interpoliert werden sollen, erhalten wir neue Abspeicherungsprobleme. Sei $f : [0,1]^d \to \mathbb{R}^m$ eine vektorwertige Funktion, $f = (f_1, \ldots, f_m)$, dann kommen pro Punkt $m-1$ Funktionswerte, $(d-1) \cdot (m-1)$ hierarchische Ubergangsüberschüsse und $m-1$ hierarchische Überschüsse dazu. Es gibt drei sinnvolle Möglichkeiten alle Daten abzuspeichern:

1. Alles wird in einer Hashtabelle gespeichert, dabei werden pro Gitterpunkt m Funktionswerte, hierarchische Überschüsse etc. gespeichert.

2. Alles wird in einer Hashtabelle gespeichert, aber jeder Punkt wird m-mal jeweils mit einem der m Funktionswerte gespeichert.

3. Jeder Punkt wird in m Hashtabellen gespeichert, wobei in der j-ten Hashtabelle der Punkt mit dem j-ten Funktionswert gespeichert wird.

Interpolieren wir auf regulären dünnen Gittern, ist die erste Methode die beste Wahl. Mit der zweiten und dritten Methode würden wir nur unnötig Speicherplatz verbrauchen, da zu jedem Punkt jeder der m Funktionswerte gegeben ist.
Bei der Interpolation auf adaptiven Gittern ist das anders. Da wir jede Teilfunktion für sich interpolieren, kann es vorkommen, dass zu einem beliebigen Punkt bei der j-ten Teilfunktion kein Eintrag in die Hashtabelle vorzunehmen ist. Bei der ersten Methode würden wir in diesem Fall $d+1$ Nullen abspeichern und zudem hätten wir keinen unabhängigen Zugriff auf die eizelnen Teilfunktionen. Das heißt wir könnten die Existenz eines Punktes der Teilfunktion j nicht feststellen, wenn der Punkt schon für die Teilfunktion $k \neq j$ gespeichert wurde.
Bei der zweiten und dritten Methode ist diese Unabhängigkeit der Teilfunktionen gegeben und daher kommen für uns nur diese beiden Methoden in Frage. Betrachten wir zuerst die dritte Möglichkeit. Wir verwenden in dieser Arbeit die einfachste Form einer Hashtabelle, d.h. die Hashtabelle bekommt zu Anfang eine Größe zugewiesen und wird sich während der Interpolation nicht verkleinern oder vergrößern. Somit würden wir im letzten Fall m Hashtabellen der gleichen Größe initialisieren, obwohl vielleicht in einigen nur wenige Punkte gespeichert werden müssten und damit Speicherplatz unnötig verschwenden. Zudem kommt noch hinzu, dass die Umsetzung der Interpolation, wenn wir auf m Hashtabellen zugreifen, erheblich schwieriger wird, als wenn wir nur auf eine zugreifen würden. Bei der zweiten Variante haben wir die Möglichkeit, die unterschiedliche Anzahl an benötigten Punkten zur Interplation der einzelnen Teilfunktionen mit einzukalkulieren. Wir können also eine etwas kleinere Hashtabelle initialisieren als für ein reguläres dünnes Gitter nötig wären. Pro Punkt einer Teilfunktion brauchen wir im Vergleich zu den einfachen Funktionen lediglich ein Arrayelement mehr zu speichern, weil auf Grund der Eindeutigkeit die Nummer der Teilfunktion mitgespeichert werden muss. Wir speichern also $3 \cdot d + 2$ Arrayelemente pro Punkt einer Teilfunktion. Anschaulich ist das in folgendem Diagramm dargestellt:

$$\begin{array}{rcl} [l_1, \ldots, l_d, i_1, \ldots, i_d, u_1^1, \ldots, u_d^1, f_1, 1] & \sim & f_1(x_{\mathbf{li}}) \\ \vdots & & \vdots \\ [l_1, \ldots, l_d, i_1, \ldots, i_d, u_1^m, \ldots, u_d^m, f_m, m] & \sim & f_m(x_{\mathbf{li}}) \end{array}$$

Dabei entspricht $[\cdot]$ einem Eintrag in der Hashtabelle A.
Da im Fall der vektorwertigen Funktionen fr jede Teilfunktion an der Stelle x_{li} bei der Verwendung der Hashfunktion $hfkt(l,i)$ der selbe Platz in der Hashtabelle zugewiesen wrde, mssen wir die Hashfunktion erweitern. Dies ist leicht getan durch:

$$hfkt(m, \mathbf{l}, \mathbf{i}) = (m+1) \cdot \sum_{j=1}^{d} (2^{l_j} + i_j) \cdot prim(j) \cdot prim(43 + (d-2) \cdot 10 - j),$$

wobei m die Nummer der Teilfunktion angibt.
Bei der Zwischenspeicherung im Array N müssen wir zusätzlich die Nummer der Teilfunktion

mit abspeichern, also brauchen wir zum Abspeichern pro Punkt $2 \cdot d + 2$ Arrayelemente in N bei der Interpolation von vektorwertigen Funktionen.

6.4 Datenstruktur und Implementierung bei Hintereinanderschaltung von Funktionen

Bei der Hintereinanderschaltung von zwei Funktionen, wie in Algorithmus 5.8 beschrieben, brauchen wir keine zwei Aufbewahrungsarrays zur Zwischenspeicherung, sondern können das Array N zwei mal nacheinander benutzen. Das liegt daran, dass die Funktionen G und $F(\hat{G})$ hintereinander interpoliert werden, wobei $\hat{G}$ der Interpolant zu G ist. Da aber auf den Interpolanten $\hat{G}$ bei der Interpolation von $F(\hat{G})$ zugegriffen werden muss, müssten wir wieder, wenn wir nur eine Hashtabelle benützten, ein zusätzliches Arrayelement pro Punkt zur Unterscheidung der Funktionen einfügen. Eine einfachere Möglichkeit in diesem Fall ist es, für jede Interpolation eine Hashtabelle zu benutzen. Seien $G : [0,1]^d \to [0,1]^d$ und $F : [0,1]^d \to \mathbb{R}^m$ mit $d, m \in \mathbb{N}$. Dann brauchen wir $3 \cdot d + 2$ Arrayelemente pro Punkt für die Hashtabelle der Funktion G und $3 \cdot d + 2$ Arrayelemente pro Punkt für die Hashtabelle der Funktion $F(\hat{G})$. Wir erhalten also eine Hashtabelle B für G und eine Hashtabelle A für $F(\hat{G})$.

6.5 Implementierung bei Lookahead

Beim Lookahead-Algorithmus 5.6 betrachten wir zur Überprüfung eines Punktes nicht nur den HÜ an dem Punkt, sondern auch die HÜ an den Söhnen, beim zweifachen Lookahead sogar die HÜ an den Enkeln.

Wenn wir einen Gitterpunkt x erzeugen, so speichern wir ihn in der Hashtabelle ab, falls er dort noch nicht existiert. Beim Lookahead werden zustzlich noch die Shne und Enkel in der Hashtabelle abgespeichert. Im Aufbewahrungsarray N hingegen speichern wir nur die Shne. Wenn aber an einem Gitterpunkt x nach der berprfung das Gitter nicht verfeinert wird, mssten eigentlich die Shne bzw.Shne und Enkel wieder aus der Hashtabelle und die Shne aus dem Aufbewahrungsarray gelscht werden. Die Shne aus dem Aufbewahrungs Array N zu lschen ist einfach, da die letzten $2 \cdot d$ Punkte im Array genau die Shne sind.

Aber wenn wir uns die Shne bzw. Shne und Enkel nicht zuvor zustzlich zwischengspeichern, mssen wir den Levelindex und der Ortsindex der Shne bzw. der Shne und Enkel zum Lschen erst wieder neu erzeugen.

Zur Umsetzung des Lookaheadalgorithmen haben wir zwei Möglichkeiten:

1. Entweder wir merken uns die gerade erzeugten Söhne bzw. Enkel in einem zusätzlichen Array und können so leichter, wenn am Vater das Gitter nicht erweitert wird, das Löschen der Söhne und Enkel vornehmen

2. oder wir belassen alle erzeugten Söhne und Enkel in der Hashtabelle A.

Beide Wege haben ihre Vor- und Nachteile. Die erste Variante scheint zeitaufwendiger als die zweite Variante zu sein. Bei der zweiten Varianten kann es vorkommen, dass an Punkten interpoliert wird, die nicht nötig für die Interpolation sind. Experimentell hat sich ergeben, dass die erste Methode in den meisten Fällen sogar weniger Zeit braucht als die zweite Methode und

dass mit der ersten Methode am Ende wesentlich weniger Punkte zur Interpolation benötigt werden als mit der zweiten Methode. Der einzige Nachteil der ersten Methode liegt darin, dass zur Zeit der Berechnung der Interpolante immer wieder Punkte im zusätzlichen Array zusätzlich zu der Hashtabelle A und dem Aufbewahrungsarray N gespeichert werden müssen. Aber auf Grund der Zeit und der Anzahl der Punkte haben wir uns für die erste Variante entschieden.

7 Numerische Resultate

Wir wollen nun unser adaptives Dünngitter-Interpolationsverfahren auf einfache, vektorwertige und hintereinadergeschaltete Funktionen anwenden und die Stärken und Schwächen des Verfahrens diskutieren. Es interessiert uns dabei insbesondere das Abschneiden des adaptiven Dünngitter-Verfahrens (ADGV) im Vergleich zum regulären Dünngitter-Verfahren (DGV), wobei wir als Grundgitter der Verfahren das dünne Gitter mit konstantem Rand (DGK) verwendet haben. Dabei wird ein besonderes Augenmerk auf Funktionen mit verschiedenen Glattheiten und Singularitäten gelegt. Danach wenden wir uns dem Verfahren zur Funktionsinterpolation durch Hintereinanderschaltung von Funktionen mit adaptiven dünnen Gittern zu und besprechen die Vor- und Nachteile des neuen Ansatzes.

7.1 Einfache Funktionen

Mit einfachen Funktionen bezeichnen wir Funktionen der Gestalt $f : \bar{\Omega} \to \mathbb{R}$ mit $\bar{\Omega} = [0, 1]^d$. Wir werden uns nun verschiedene d-dimensionale Funktionen anschauen, wobei sich die Funktionen hauptsächlich in ihrer Glattheit unterscheiden. Wir unterteilen Funktionen in Bezug auf ihre Glattheit wie folgt:

1. glatte Funktionen, d.h. Funktionen ohne Singularität,
2. Funktionen mit Ebenensingularität,
   - die Singularität ist parallel zu den Achsen
   - die Singularität ist schräg zu den Achsen
3. Funktionen mit Punktsingulariät,
   - die Singularität liegt in einer Ecke von $\bar{\Omega}$
   - die Singularität liegt auf einem Rand von $\bar{\Omega}$
   - die Singularität liegt im Inneren von $\bar{\Omega}$
4. und Funktionen mit Kreissingularitäten.

Wobei jede der oben aufgeführten Singularitäten auf verschiedene Art und Weise auftreten kann:

1. Singularität als Polstelle
2. Singularität als Sprung
3. Singularität als Knick

Wir beschränken uns bei der Betrachtung der Funktionen auf die Dimensionen $d \in 2, \ldots, 5$, da diese eingeschränkte Betrachtung vollkommen ausreicht um allgemeingültige Ergebnisse festzustellen und zu erläutern.

Wir werden nun anhand von fünf Beispielen sehen, wie gut die zwei Interpolationsverfahren, DGV und ADGV, bei Funktionen mit Singularitäten funktionieren. Um qualitative Aussagen machen zu können betrachten wir die Konvergenzrate, die Genauigkeit der Interpolanten und den dafür benötigten maschinellen Aufwand. Die Ergebnisse werden in Konvergenzgraphen, einer Konvergenztabelle und einer Punktetabelle gezeigt bzw. aufgelistet. Es folgt eine kurze Beschreibung dieser drei Darstellungen.

- Im Konvergenzgraph sind die Anzahl der benötigten Punkte auf der x-Achse und der dazugehörige errechnete L_2-Interpolationsfehler oder der L_∞-Interpolationsfehler auf der y-Achse gegeneinader geplottet, wobei beide Achsen logarithmisch skaliert sind. Zu jeder im Beispiel angegebenen Funktion werden jeweils zwei Konvergenzkurven (Fehlerkurven) im Konvergenzgraph gezeichnet, einen für den Interpolanten auf dem regulären dünnen Gitter und einen für den Interpolanten auf dem adaptiven dünnen Gitter.

- In der Konvergenztabelle stehen die Konvergenzraten p der einzelnen Interpolationsverfahren mit der benutzten Fehlernorm zu den im Beispiel interpolierten Funktionen. In der Konvergenztabelle steht in der Spalte mit der Überschrift DG_{L_2} zum Beispiel die Konvergenzrate des L_2-Fehlergraphen für das reguläre Dünngitter-Verfahren.

- In den Punktetabellen ist aufgeführt, wieviele Punkte das DGV benötigt den dort angegebenen L_2- und L_∞-Interpolationsfehler zu den Funktionen im Beispiel zu erreichen. Weiter steht in der Tabelle in der zugehörigen Spalte oder Zeile die Anzahl der Punkte die ein ADGV benötigte um den selben Interpolationsfehler zu erreichen. Wenn sich die Anzahl der Punkte im Falle des adaptiven Dünngitter-Verfahrens für den L_2-Fehler von der für den L_∞-Fehler unterscheiden, kennzeichnen wir dies durch $P(L_2)$ für den L_2-Fehler und $P(L_\infty)$ für den L_∞-Fehler. In dem Fall, indem unterschiedliche maximale Level zur Interpolation der einzelnen Funktionen benutzt werden, sind diese mit angegeben. Ansonsten ist das maximale Level in der Beispielbeschreibung mit angegeben.

Bei der Verwendung des ADGV haben wir eine ϵ-Schrittweite von 4^{-1} gewählt, haben den Algorithmus mit einem Startwert von $\epsilon = 1$ begonnen und die ϵ-Schranke auf 4^{-21} gesetzt. Wir werden bei jedem Beispiel die benötigte ϵ-Schranke angeben, also die minimale ϵ-Schranke, sobald sie von 4^{-21} abweicht.

Beispiel PEK: In unserem ersten Beispiel betrachten wir eine d-dimensionale Fläche, die an einer $d-1$-dimensionalen Ebene, die parallel zu den Achsen verläuft, geknickt wird:

$$f(x_1, \ldots, x_d) = \begin{cases} 0 & , \text{falls } x_1 <= 0.4 \\ (x_1 - 0.4) \cdot \frac{5}{3} & , \text{sonst.} \end{cases}$$

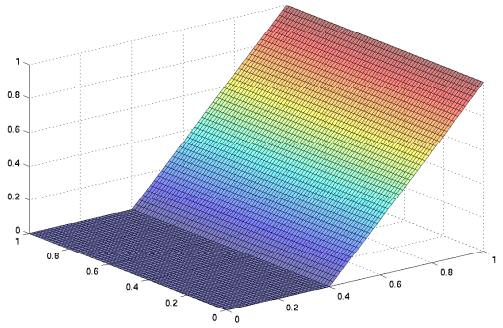


Abb. 7.1: Funktionsplot der Beispielfunktion PEK in 2 Dimensionen

Wir betrachten also eine Funktion mit einer Ebenensingularität, die parallel zu den Achsen verläuft. Wir beschränken uns wie vorhin erläutert auf die Fälle $d = 2, \ldots, 5$. Es stellt sich heraus, dass das Startlevel des adaptiven Dünngitter-Verfahrens auf 1 gesetzt werden muss, weil einige zum Aufbau des Gitters benötigten hierarchischen Überschüsse vom Algorithmus bei noch geringerem Startlevel nicht berechnet werden. Weiter wurde mit steigender Funktionsdimension ein niedrigeres maximales Level gewählt, da eine Berechnung des Interpolationsfehlers in annehmbarer Zeit sonst nicht realisierbar gewesen wäre. Daher wird die ϵ-Schranke mit steigender Dimension immer größer und damit ist automatisch der Interpolationsfehler bei dieser Funktion (PEK) in höheren Dimensionen größer als in kleineren Dimensionen.

Dimension	2	3	4	5
ϵ-Schranke	4^{-8}	4^{-7}	4^{-6}	4^{-5}
Schrittweite von FP	$\frac{1}{445}$	$\frac{1}{60}$	$\frac{1}{20}$	$\frac{1}{10}$

Tabelle 7.1: Tabelle zeigt jeweils die implementierten Funktionsdimensionen, die zur Funktionsinterpolation gewählte ϵ-Schranke und die zur Fehlerberechnung gewählte Schrittweite der Fehlerberechnungsmethode mit Produktgitterpunkten (FP).

Da wir die Interpolationsfehler der Funktion in verschiedenen Dimensionen vergleichen wollen, müssen wir für jede Fehlerberechnung ungefähr die gleiche Punktanzahl verwenden. Diese Bedingung führt uns zu den Schrittweiten, die in der vorherigen Tabelle 7.1 aufgelistet sind. Für nähere Erläuterungen zur Fehlerberechnungsmethode mit Produktgitterpunkten (FP) siehe Abschnitt 3.
Gehen wir nun zur Auswertung dieses Beispiels über. Fangen wir mit der in Abbildung 7.2 gezeigten Konvergenzgraphen an. Wie in der Abbildung zu erkennen, erhalten wir für das ADGV in diesem Beispiel eine außergewöhnlich hohe Konvergenzrate, die darauf zurückzuführen ist, dass unsere Beispielfunktion einen konstanten Teil und einen linearen Teil besitzt und zudem nur von x_1 abhängt. Auf Grund dieser sehr einfachen Struktur benötigt das ADGV sehr wenig Punkte (siehe Tabelle 7.2), um die Funktion zu interpolieren. Das DGV baut unabhängig von der zu interpolierenden Funktion ein komplettes dünnes Gitter auf. So kommt es, dass das ADGV bei der Interpolation der Beispielfunktion für $d = 2$ gerade einmal ein dreitausendstel

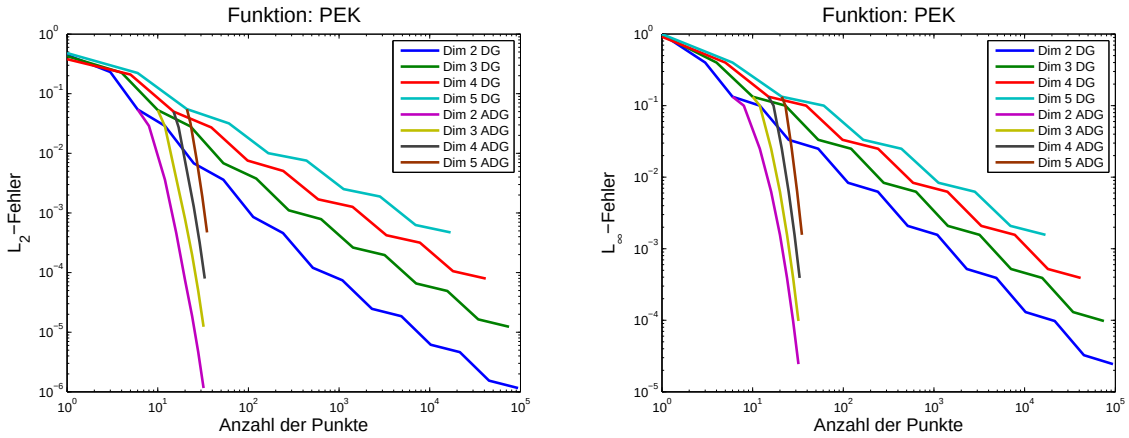


Abb. 7.2: L_2- und L_∞-Fehler zum Beispiel PEK

der Punkte des DGVs braucht, um den gleichen Interpolationsfehler zu erhalten. Für $d = 5$ werden immerhin nur noch ein fünfhundertstel der Punkte des DGVs benötigt.
Betrachten wir den Verlauf der Fehlerkurven in Abbildung 7.2, so sehen wir dass der L_∞-Fehler den gleichen guten Verlauf wie der L_2-Fehler aufweist. Dies zeigt, dass die Dünngitter-Verfahren diesen Typ von Funktion gut im Griff haben und speziell das adaptive Dünngitter-Verfahren für die Funktion PEK sinnvoll eingesetzt werden kann. In der Abbildung 7.2 kann man außerdem erkennen, dass umso genauer der Interpolationsfehler wird, desto mehr weichen die Fehlerkurven des DGVs (d.h. die Kurven für die Funktion in verschiedenen Dimensionen) voneinander ab. Beim ADGV tritt dieses Verhalten nicht auf. Dort bleibt der Abstand zwischen den Fehlerkurven bestehen. Das heißt also, dass die Punktanzahl mit dem ADGV gegenüber dem DGV nicht nur mit steigender Genauigkeit verbessert wird, sondern auch mit steigendener Dimension.

d	l	L_2	L_∞	$\mathrm{P_{DG}}$	$\mathrm{P_{ADG}}$
2	14	1.156039e-06	2.441406e-05	94209	32
3	12	1.229862e-05	9.765625e-05	75009	32
4	10	7.922576e-05	3.906250e-04	41425	33
5	8	4.711115e-04	1.562500e-03	17002	35

Tabelle 7.2: Minimaler Interpolationsfehler zum höchsten Level l in Dimension d für das Beispiel PEK mit FP errechnet

Beispiel G: Gegenstand der Betrachtung in diesem Beispiel ist eine vollkommen glatte Funktion, die aus einem Sinusteil und einem Exponentialteil besteht. Die Funktion hat folgendes Aussehen:

$$f(x_1, \ldots, x_d) = \sum_{i=1}^{d} exp(x_i) - sin(3 \cdot \pi \cdot x_i).$$

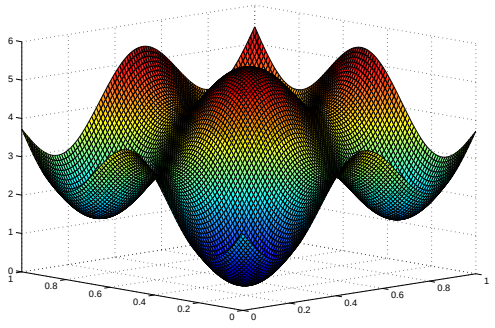


Abb. 7.3: Funktionsplot der Beispielfunktion G für 2 Dimensionen

Auch hier haben wir die Funktion in den Dimensionen zwei bis fünf untersucht und haben auf Grund des gleichen Arguments, wie in **Beispiel PEK**, die Schrittweiten zur Fehlerberechnung wie folgt gewählt:

Dimension	2	3	4	5
ϵ-Schranke	4^{-17}	4^{-14}	4^{-11}	4^{-10}
Schrittweite von FP	$\frac{1}{445}$	$\frac{1}{59}$	$\frac{1}{20}$	$\frac{1}{10}$

Tabelle 7.3: Tabelle zeigt jeweils die implementierten Funktionsdimensionen, die zur Funktionsinterpolation gewählte ϵ-Schranke und die zur Fehlerberechnung gewählte Schrittweite der FP.

Die ϵ-Schranken erklären sich auch hier mit dem niedrigeren maximalen Level in höheren Dimensionen. Betrachten wir zuerst die Konvergenztabelle 7.4. Beide Verfahren weisen eine Konvergenzrate von ca. zwei auf, wobei das ADGV eine leicht höhere Konvergenrate aufweist. Sehr gut zu erkennen ist auch, dass das DGV mit steigender Dimension langsamer und das ADGV schneller konvergiert, wie im Konvergenzgraph, Abbildung 7.4, zu sehen ist. Dies entspricht den vorherigen Beobachtungen, die wir beim **Beispiel PEK** machen konnten. Der Abstand der Fehlerkurven des DGVs wird mit steigender Genauigkeit immer größer und der Abstand der Fehlerkurven des ADGVs immer geringer. Also bleibt die Konvergenzrate, wenn man auf einem adaptiven dünnen Gitter interpoliert und die Dimension der Funktion erhöht, zumindest im Vergleich zur niedrigsten Dimension erhalten oder wird eventuell sogar besser. Beim regulären Dünngitter-Verfahren trifft uns an dieser Stelle der Fluch der Dimension. Mit der Dimension der zu interpolierenden Funktion steigt auch die Anzahl der Gitterpunkte, die ein reguläres dünnes Gitter besitzt, aber der Interpolationsfehler wird nicht in gleichem Maß kleiner. So kann die Konvergenzrate des DGV für Funktionen in höheren Dimensionen gar nicht besser sein als die in einer niedrigeren Dimension, wenn der Funktionstyp über die einnzelnen Dimensionen vergleichbar bleibt. Vergleicht man in Bezug auf die Konvergenzrate zum Beispiel eine zweidimensionale Exponentialfunktion mit einer vierdimensionalen Sprungfunktion, so macht das natürlich keinen Sinn.

Wenden wir uns nun der Punktetabelle 7.5 zu. Braucht das ADGV in zwei Dimension ein drittel der Punkte des DGVs, um den gleichen Interpolationsfehler, wie das DGV zu erreichen, so braucht es bei drei Dimensionen ein sechstel, bei vier Dimesionen ein zehntel und in fünf Dimensionen nur noch ein dreizehntel der Punkte des DGVs. Das heißt, je höher die Dimension, desto mehr unwichtige Punkte, das sind alle die Punkte, die die Genauigkeit nicht verbessern, fallen bei der Interpolation an.
Außerdem ist gut zu erkennen, auch in Abbildung 7.4, dass bei glatten Funktionen kaum ein Unterschied zwischen dem L_2- und dem L_∞-Fehler zu bemerken ist. Das bedeutet, dass die Funktion an jeder Stelle gleich gut interpoliert wird. Es gibt keine Stellen an der sehr viel mehr Arbeit als an anderen Stellen investiert werden muss. Bei singulären Funktionen ist dies etwas anders, wie wir in den verbleibenden Beispielen sehen werden.

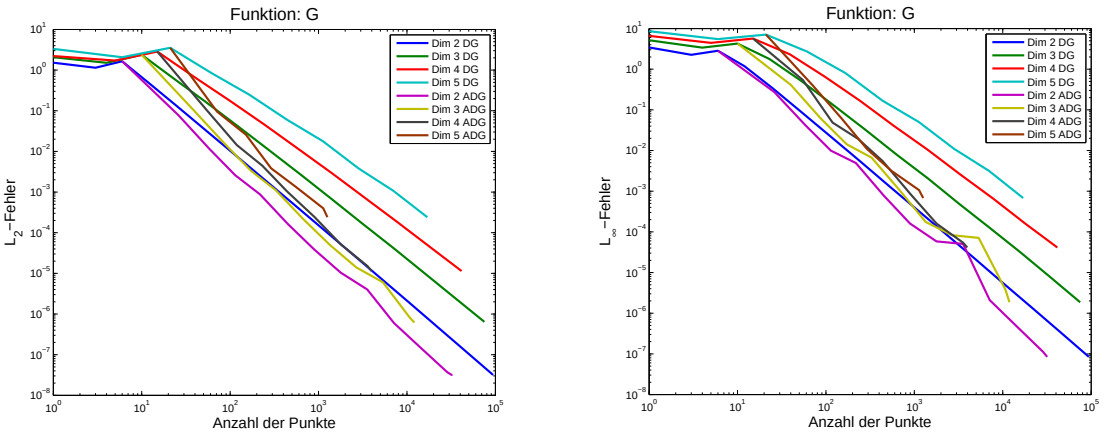


Abb. 7.4: L_2- und L_∞-Fehler zum Beispiel G

Dim.	DG_{L_2}	ADG_{L_2}	DG_{L_∞}	ADG_{L_∞}
2	1.8413	2.0210	1.8479	2.0888
3	1.6983	2.0709	1.7217	2.0369
4	1.5697	2.0949	1.5838	2.1495
5	1.4241	2.2501	1.4200	2.2418

Tabelle 7.4: Konvergenzrate für das Beispiel G

d	l	L_2	L_∞	P_{DG}	$\mathrm{P}_{\mathrm{ADG}}^{L_2}$	$\mathrm{P}_{\mathrm{ADG}}^{L_\infty}$
2	14	3.173845e-08	8.455687e-08	94209	32762	31806
3	12	6.436699e-07	1.859173e-06	75009	12226	11932
4	10	1.144043e-05	4.122800e-05	41425	3967	3967
5	8	2.401656e-04	6.792474e-04	17002	1261	1261

Tabelle 7.5: Minimaler Interpolationsfehler zum höchsten Level l in Dimension d für das Beispiel G

Beispiel SE: Beschäftigen wir uns nun mit drei verschiedenen Singularitäten an einer zu den Achsen schrägen Ebene:

SEK: Funktion mit Knick an der schrägen Ebene

$$f(x_1,\ldots,x_d) = \begin{cases} 0 & \text{, falls } x_1 - \sum_{i=2}^{d} x_i - \frac{1}{4} \leq 0 \\ (x_1 - \sum_{i=2}^{d} x_i - \frac{1}{4}) \cdot \frac{4}{3} & \text{, sonst.} \end{cases}$$

SES: Funktion mit Sprung der schrägen Ebene

$$f(x_1,\ldots,x_d) = \begin{cases} 0 & \text{, falls } x_1 - \sum_{i=2}^{d} x_i - \frac{1}{4} = 0 \\ 1 & \text{, sonst.} \end{cases}$$

SEP: Funktion mit Polstelle an der schrägen Ebene

$$f(x_1,\ldots,x_d) = \left| x_1 - \sum_{i=2}^{d} x_i - \frac{1}{3} \right|^{-\frac{1}{3}}$$

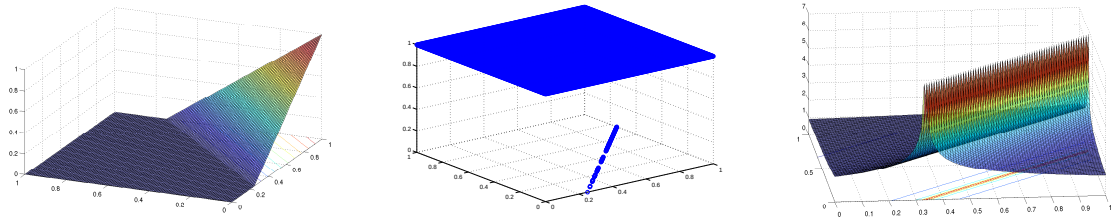


Abb. 7.5: Funktionen SEK, SES und SEP in 2 Dimensionen

Die Schrittweite bei der Fehlerberechnung der Funktion SES unterscheidet sich von den Schrittweiten der anderen Funktionen, da wir hier darauf achten müssen, dass die Sprungstelle auch von den eingesetzten Punkten getroffen wird, was mit der von uns üblich gewählten Schrittweite $\frac{1}{445}$ nicht der Fall ist.

Funktion	SEK	SES	SEP
ϵ-Schranke	4^{-5}	4^{-3}	4^{-10}
Schrittweite von FP	$\frac{1}{445}$	$\frac{1}{444}$	$\frac{1}{445}$
Startlevel	-1	4	3
Hierarchisieren	mit Lookahead	ohne Lookahead	ohne Lookahead

Tabelle 7.6: Die zu den zu interpolierenden Funktionen gewählte ϵ-Schranke, Schrittweite von FP, das Startlevel und Hierarchisierungsweg (mit Lookahead oder ohne Lookahead, siehe Kapitel 5).

Bei diesem Beispiel wollen wir Funktionen mit unterschiedlichen Singularitäten, wie Knick, Sprung und Polstelle, vergleichen, wobei die Singularität an einer zu den Achsen schrägen Ebene auftritt. Speziell mit solchen Funktionen hat ein ein reguläres Dünngitter-Verfahren große Schwierigkeiten auf Grund der Tensorproduktdarstellung der Basisfunktionen. Unser besonderes Augenmerk liegt darauf, wie gut die unterschiedlichen Singularitäten im Allgemeinen von den Dünngitter-Verfahren approximiert werden, wie die Interpolation der Funktionen mit Singularitäten im Vergleich zur Interpolation der glatten Funktionen ausfallen und wie gut das adaptive Dünngitter-Verfahren im Vergleich zum regulären Dünngitter-Verfahren abschneidet. Beginnen wir mit dem Vergleich der Interpolation der Funktionen mit den verschiedenen Singularitäten mit Hilfe der Konvergenzgraphen in Abbildung 7.6. Man sieht, dass die Funktion

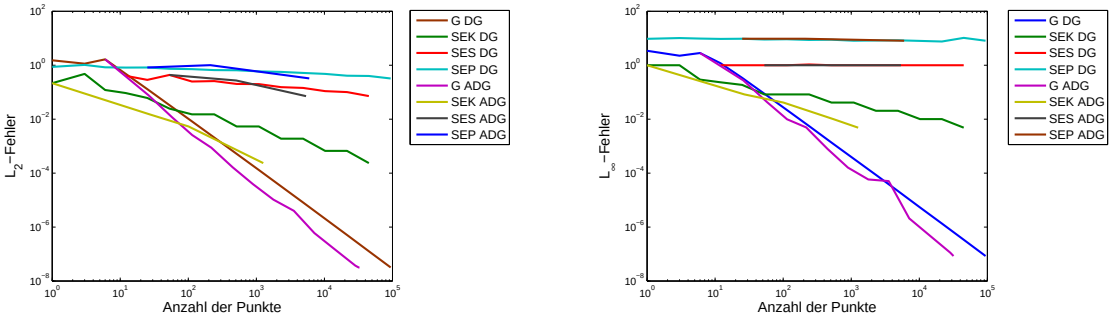


Abb. 7.6: L_2- und L_∞-Fehler zum Beispiel SE

SEK am genauesten interpoliert wird, am schlechtesten die Funktion mit Polstelle abschneidet und die Fehlerkurve der Sprungfunktion SES genau zwischen den der Funktionen SEK und SEP liegt. Wie sieht das Abschneiden nun aber bezüglich der Konvergenzrate aus? In Tabelle 7.7 sind die Ergebnisse zur Konvergenzrate zu sehen. Die Konvergenzrate beider Interpola-

Fkt.	DG_{L_2}	ADG_{L_2}	DG_{L_∞}	ADG_{L_∞}	h
SEK	0.6694	0.9487	0.4627	0.7361	2^{-14}
SES	0.1973	0.3935	-	-	2^{-14}
SEP	0.0905	0.1344	0.0235	0.0291	2^{-14}

Tabelle 7.7: Konvergenzrate für das Beispiel SE

tionsverfahren ist in diesem Beispiel für jede der betrachten Funktionen niedrig. Wobei die Dünngitter-Verfahren bei der Funktion SEK eine höhere Konvergenzrate aufweisen als bei den Funktionen SES und SEP.

Die gleiche Rangfolge tritt auch bei der Anzahl an benötigten Punkten (Tabelle 7.9) beim adaptiven Dünngitter-Verfahren auf, d.h. wir brauchen am wenigsten Punkte für die Funktion SEK und am meisten für die Funktion SEP.

Betrachten wir nun alle bis jetzt bearbeiteten Funktionen, d.h. inklusive Funktionen G (Tabelle 7.5) und PEK (Tabelle 7.2). Wir sehen, dass die Funktion G am genaueseten interpoliert wird. Etwas schlechter sieht es schon bei der Funktion PEK aus und noch ungenauer werden die Funktionen SEK, SES und SEP interpoliert. So können wir die Funktionen in Bezug auf

den Interpolationsfehler, der bei der Interpolation dieser Funktionen gemacht wird, sortieren (1=gut,...,5=schlecht):

Genauigkeit	1	2	3	4	5
Funktion	G	PEK	SEK	SES	SEP

Tabelle 7.8: Funktionen geordnet nach dem Interpolationsfehler (ordinal), der bei der Interpolation dieser Funktion gemacht wird (1=gut,...,5=schlecht).

Also Funktionen mit Singularitäten an den Achsen parallelen Ebenen werden genauer interpoliert als Funktionen mit Singularitäten auf Ebenen, die schräg zu den Achsen liegen. Am besten aber schneiden die glatten Funktionen bei der Interpolation ab. Jeder bisher untersuchte Funktionstyp wird mit dem adaptiven Dünngitter-Verfahren besser interpoliert als mit dem regulären Dünngitter-Verfahren, denn es braucht weniger Punkte um den gleichen Interpolationsfehler, wie das reguläre Dünngitter-Verfahren zu erreichen.

Fkt.	SEK	SES	SEP
P_{DGK}	45057	94209	94209
$P_{ADGK}^{L_2}$	1265	5385	48371
$P_{ADGK}^{L_\infty}$	1265	5385	27815
L_2	2.364303e-04	7.154228e-02	3.241375e-01
L_∞	4.840508e-03	1.000000e+00	8.147634e+00

Tabelle 7.9: Minimaler Interpolationsfehler zum höchsten Level $l = 14$ in 2 Dimensionen für das Beispiel SE und die dafür benötigten Punkte

Beispiel P: In unserem vierten Beispiel befassen wir uns mit Funktionen mit Punktsingularitäten, genauer gesagt mit Funktionen mit Polstellen. Wir werden sehen, dass die Lage der Polstelle Einfluss darauf hat, ob und inwiefern unsere benutzten Dünngitter-Verfahren Schwierigkeiten bei der Interpolation haben. Wir betrachten dazu drei Funktionen mit verschiedenen Lagen der Polstellen:

IPP: Funktion mit Polstelle in einem inneren Punkt

$$f(x_1, x_2) = (\left|x_1 - \frac{1}{3}\right| + \left|x_2 - \frac{1}{3}\right|)^{-\frac{1}{3}}$$

RPP: Funktion mit Polstelle in einem Randpunkt

$$f(x_1, x_2) = (\left|x_1 - \frac{1}{3}\right| + |x_2|)^{-\frac{1}{3}}$$

EPP: Funktion mit Polstelle in einem Eckpunkt

$$f(x_1, x_2) = (x_1^2 + x_2^2 + 10^{-10})^{-\frac{1}{3}}$$

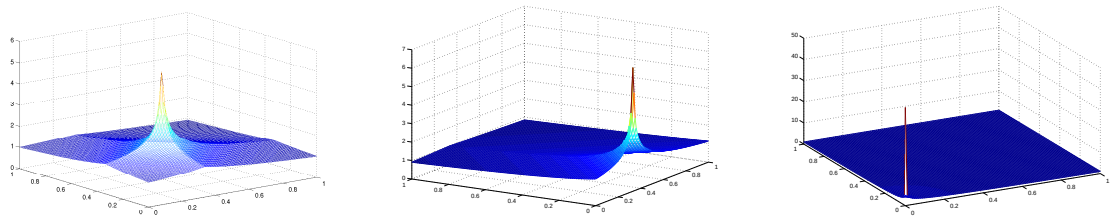


Abb. 7.7: Funktionen IPP (links), RPP (mitte) und EPP (rechts)

Benutzt man zur Fehlerberechnung nach der Interpolation der Funktion EPP die Fehlerberechnungsmethode mit zufälligen Punkten (FZ) so erhalten wir in Bezug auf die Genauigkeit des Interpolationsfehlers gute Ergebnisse. Haben wir aber zuvor zur Interpolation das adaptive Dünngitter-Verfahren verwendet, dann oszilliert die Fehlerkurve berechnet mit der FZ-Methode. Daher können wir nicht feststellen, bei wievielen Punkten der Interpolant, der mit dem adaptiven Dünngitter-Verfahren erstellt wird, sich nicht mehr nennenswert verändert. Verwenden wir zur Fehlerberechnung die FP-Methode, die wir in den vorherigen Beispielen benutzt haben, erhalten wir auch keine verwertbaren Ergebnisse, weil mit der Schrittweite $(445)^{-1}$ ab Level 9 die interpolierte Polstelle nicht mehr getroffen wird und nichts mehr über die Genauigkeit des Interpolanten an dieser Stelle ausgesagt werden kann. Erst bei einer Schrittweite von 10^{-6} des Produktgitters der FP-Methode würden wir wieder genaue Ergebnisse erhalten, das würde aber zuviel Rechenzeit kosten. Nur mit der Fehlerberechnungsmethode mit Dünngitterpunkten (FD) ist der Interpolationsfehler zur Funktion EPP gut zu approximieren. Die FD-Methode bei der Punkte des Levels 15 benutzt werden, benötigt weniger Punkte, genauer gesagt braucht sie 102400 Punkte, als die FP-Methode mit Schrittweite $\frac{1}{445}$ und hat zudem noch eine kleinere Schrittweite. Dadurch wird die Polstelle berücksichtigt und der Interpolationsfehler wird genauer approximiert. Deswegen werden wir in diesem Beispiel den Interpolationsfehler mit der FD-Methode mit Punkten des Levels 15 berechnen.
Um den L_2-, bzw. L_∞-Fehler, den wir bei der Interpolation der Funktionen IPP, RPP und EPP mit dem regulären Dünngitter-Verfahren machen, zu erreichen, haben wir folgende ϵ-Schranken für das adaptive Dünngitter-Verfahren gewählt:

Funktion	IPP	RPP	EPP
ϵ-Schranke$_{L_2}$	4^{-10}	4^{-8}	4^{-5}
ϵ-Schranke$_{L_\infty}$	4^{-3}	4^{-2}	4^{-1}

Tabelle 7.10: Zur Interpolation gewählte ϵ-Schranke.

In der Abb. 7.8 sehen wir, dass der minimalste L_2-Interpolationsfehler zur Funktion EPP mit dem adaptiven Dünngitter-Verfahren schon nach wenigen Interpolationsschritten bei 338 Punkten (Tabelle 7.12) erreicht ist, wobei der Interpolant schon bei einer geringeren Anzahl an Punkten nahe am Optimum liegt. Das heißt, der Interpolant startet schon mit einer guten Approximation der Funktion und die Hinzunahme von weiteren Punkten verbessert diese nicht nennenswert. Deswegen wurde keine Konvergenzrate dazu in Tabelle 7.11 angegeben. Beim zugehörigen L_∞-Fehler sehen wir ein ähnliches Verhalten. Betrachten wir die Konverganzrate zu

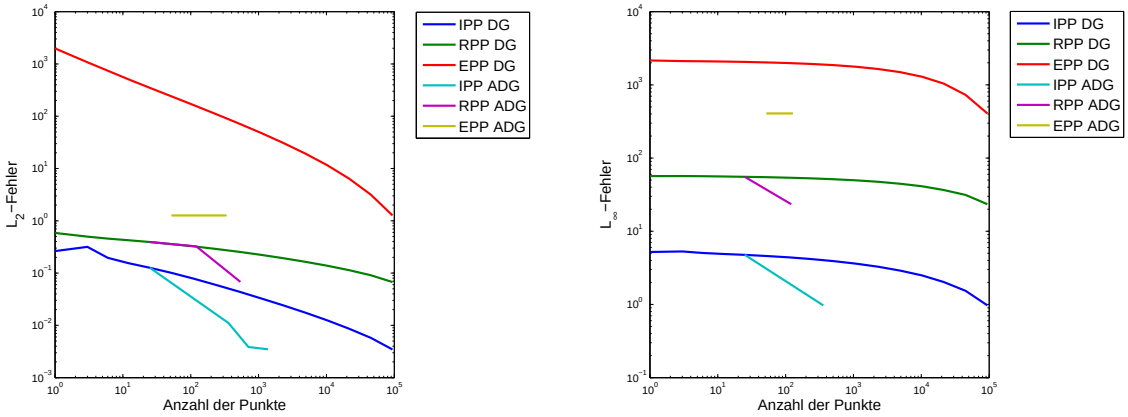


Abb. 7.8: L_2- und L_∞-Fehler zum Beispiel P

Dim.	DG_{L_2}	ADG_{L_2}	DG_{L_∞}	ADG_{L_∞}
IPP	0.3985	0.9402	0.0926	0.5952
RPP	0.1717	0.5576	0.0386	0.5470
EPP	0.5412	-	0.0558	-

Tabelle 7.11: Konvergenzrate für das Beispiel P

den regulären Dünngitter-Verfahren, so erwarten wir eigentlich die schlechteste Komvergenzrate bei der Interpolation der Funktion EPP, weil die Polstelle im Eckpunkt liegt, d.h. zwei Ränder bei der Polstelle aufeinandertreffen. Die berechnete Konvergenzrate für diesen Fall ist entgegen dieser Erwartung besser als die Konvergenzraten bei der Interpolation der Funktionen IPP und RPP. Das liegt daran, dass die Funktion EPP nicht wirklich eine Funktion mit Polstelle in einer Ecke ist. Da bei der Interpolation die Eckpunkte der Funktion ausgewertet werden, hätten wir hier eine Stelle gehabt die nicht hätte ausgewertet werden können und so wäre der Interpolant nicht bestimmt worden. Deswegen haben wir diese Stelle etwas vom Eckpunkt weg verschoben, so dass sie ausgewertet werden kann. Jetzt ist der Funktionswert an dieser Stelle zwar noch sehr groß, aber es existiert im eigentlichen Sinne keine Polstelle mehr. Auf Grund dieser Tatsache hat das Dünngitter-Verfahren für die Funktion EPP eine bessere Konvergenzrate als das Dünngitter-Verfahren angewandt auf die anderen beiden Funktionen. Aber eines kann man der Konvergenztabelle doch ablesen, dass das adaptive Dünngitter-Verfahren viel besser, also mit sehr wenigen Punkten, eine Funktion mit Punktpolstelle approximiert als ein reguläres Dünngitter-Verfahren, siehe dazu die Tabelle 7.12.
Zum Ende dieses Beispiels sollten wir noch über die Genauigkeit, mit der unsere Funktionen IPP, RPP und EPP interpoliert werden, sprechen. Wie den Konvergenzgraphen in Abbildung 7.8 zu entnehmen, ist die Lage der Polstelle sehr entscheidend, damit die Diffenrenz zwischen dem Interpolanten und der Originalfunktion nur gering ausfällt. Sobald die Polstelle im Inneren des Interpolationsgebiets liegt haben wir gute Chancen ein ordentliches Ergebnis in Bezug auf den Interpolationsfehler zu bekommen im Vergleich zu den Fällen, in denen die Polstelle am Rand liegt. Logisch erscheint auch das Ergebnis für die Funktion RPP, welches, wenn man sich

die Interpolationgenauigkeit anschaut, genau zwischen den der Extremfälle IPP und EPP liegt.

Fkt.	L_2	L_∞	P_{DG}	$\mathrm{P}_{\mathrm{ADG}}^{L_2}$	$\mathrm{P}_{\mathrm{ADG}}^{L_\infty}$
IPP	3.457025e-03	9.714231e-01	94209	23671	360
RPP	6.724407e-02	2.340623e+01	94209	8104	121
EPP	1.264887e+00	4.071497e+02	94209	338	52

Tabelle 7.12: Minimaler Interpolationsfehler zum höchsten Level 14 in 2 Dimensionen für das Beispiel P

Beispiel POL: Zum Abschluss der einfachen Funktionen betrachten wir die Interpolation von Funktionen mit einer vorgegebenen Singularität, hier eine Polstelle. Der Träger einer Singularität kann verschiedene Formen annehmen. Wir werden hier fünf zweiddimensionale Funktionen mit unterschiedlichen Trägern der Polstelle, G, IPP, KP, PEP und SEP vergleichen. Dabei haben wir darauf geachtet, dass die Polstelle im Innern des Gitters liegt.

KP Funktion mit Polstelle auf einer Kreislinie

$$f(x_1,\ldots,x_d) = \left|\sum_{i=1}^{d}(\frac{1}{2}-x_i)^2 - \frac{1}{9}\right|^{-\frac{1}{3}} - 1$$

PEP Funktion mit Polstelle auf der zu den Achsen parallelen Ebene

$$f(x_1,\ldots,x_d) = \left|x_1 - \frac{1}{3}\right|^{-\frac{1}{3}}$$

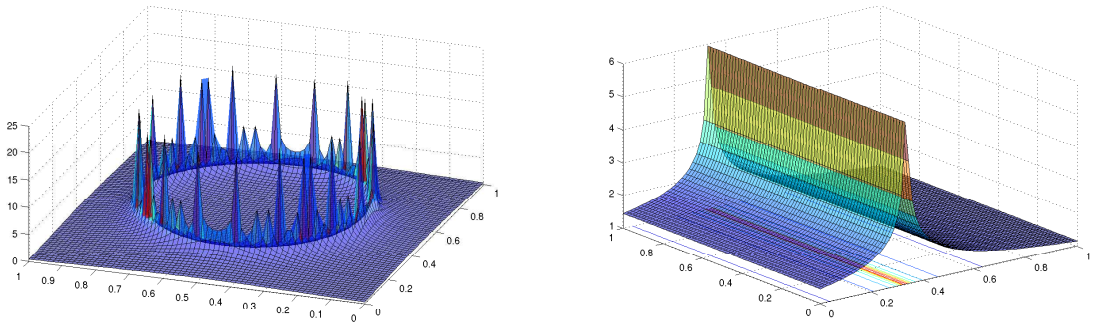


Abb. 7.9: Funktionen KP und PEP in 2 Dimensionen

Alle Interpolantionsfehler wurden mit der Fehlerberechnungsmethode FP mit der Schrittweite $h = 445^{-1}$) berechnet. Die ϵ-Schranken, die benötigt werden, um die Interpolationsfehler bei der Anwendung des DGVs auch mit dem ADGV zu erreichen, sind in der nachstehenden Tabelle aufgelistet:

Funktion	G	PEP	IPP	SEP	KP
ϵ-Schranke$_{L_2}$	4^{-16}	4^{-13}	4^{-8}	4^{-10}	4^{-6}
ϵ-Schranke$_{L_\infty}$	4^{-13}	4^{-6}	4^{-5}	4^{-8}	4^{-5}

Tabelle 7.13: Die zu den zu interpolierenden Funktionen gewählte ϵ-Schranke.

Betrachtet man die Tabellen 7.14 und 7.15, so sieht man, dass bei Polstellen die Genauigkeit eng mit der Konvergenz verknüpft ist. Je genauer die Funktion interpoliert wird, das gilt für beide Dünngitter-Verfahren, desto höher ist auch die Konvergenzrate.
Da die Wahrscheinlichkeit eine parallel zu einer der Achsen verlaufenden Polstelle zu treffen höher ist, als die Wahrscheinlichkeit eine Polstelle in einem Punkt zu treffen, ist es klar, warum die Genauigkeit der Interpolation der Funktion PEP höher ausfällt als die der Interpolation der Funktion IPP. Warum es aber schwerer ist eine Polstelle auf einer Geraden schräg zu den Achsen liegend zu approximieren als eine Polstelle in einem Punkt, liegt an dem Tensorprodukt auf dem die dünnen Gitter aufgebaut sind.
Da beim Kreis die Punkte nicht wie Gitterpunkte nebeneinander liegen, werden noch mehr Fehler bei der Interpolation der Funktion KP gemacht als bei der Funktion SEP. Die Tabellen 7.14, 7.15 und die Abbildung 7.10 zeigen diese Ergebnisse.

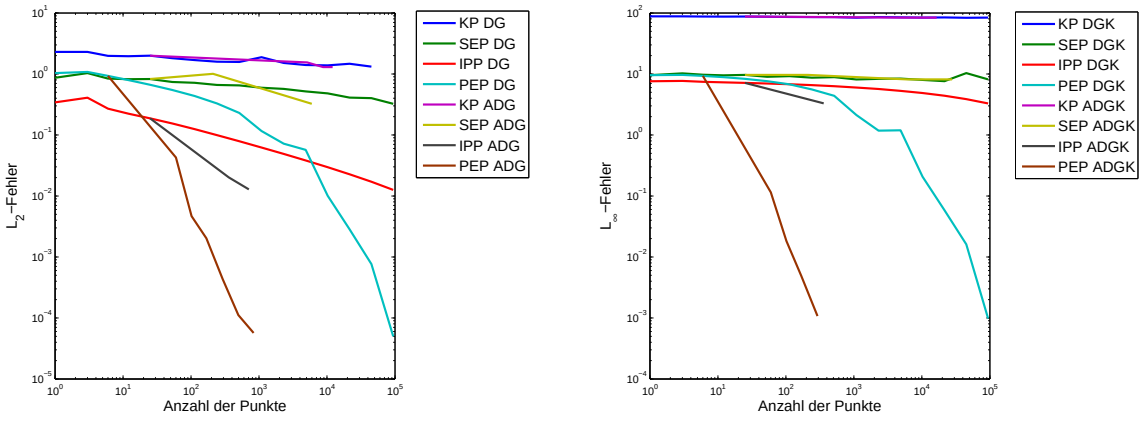


Abb. 7.10: L_2- und L_∞-Fehler zum Beispiel POL

Fkt.	DG$_{L_2}$	ADG$_{L_2}$	DG$_{L_\infty}$	ADG$_{L_\infty}$
G	1.8413	2.0210	1.8479	2.0888
PEP	0.6561	1.9733	0.5714	2.2087
IPP	0.3175	0.4369	0.0617	0.2092
SEP	0.0905	0.1344	0.0235	0.0291
KP	0.0498	0.0662	0.0047	0.0065

Tabelle 7.14: Konvergenzrate für das Beispiel POL

Fkt.	L_2	L_∞	P_{DG}	$P_{ADG}^{L_2}$	$P_{ADG}^{L_\infty}$
G	3.173845e-08	8.455687e-08	94209	32762	31806
PEP	4.989575e-05	9.809810e-04	94209	12362	496
IPP	1.256021e-02	3.291156e+00	94209	8427	1375
SEP	3.241375e-01	8.147634e+00	94209	48371	27815
KP	1.299674e+00	8.394102e+01	94209	22341	16489

Tabelle 7.15: Minimaler Interpolationsfehler zum höchsten Level 14 in 2 Dimensionen für das Beispiel POL

Fazit

Tragen wir nun alle Ergebnisse und Erkenntnisse zusammen. Mit dem adaptiven Dünngitter-Verfahren können wir in allen Beispielen die Konvergenzrate im Vergleich zum regulären Dünngitter-Verfahren erhöhen und damit schneller und mit weniger benötigten Punkten die Genauigkeit des regulären Dünngitter-Verfahrens erreichen. Dabei kann die Punktanzahl bei der Interpolation von Funktionen mit Singularitäten durch den Einsatz des adaptiven Dünngitter-Verfahrens erheblich verbessert werden, da der Interpolationsfehler hauptsächlich durch die Singularitäten beeinflusst wird und so relativ große hierarchische Überschüsse ausreichen, um die Funktion so genau wie mit einem regulären Dünngitter-Interpolationsverfahren zu beschreiben.
Desweiteren haben wir noch die Funktionen mit unterschiedlichen Glattheitsgraden, Singularitäten und unterschiedlichen Lagen der Singularitäten diskutiert und kamen zu dem Ergebnis, dass glatte Funktionen am genauesten interpoliert werden (siehe Abbildung 7.6). Betrachten wir nur die Art der Singularität, so wird eine Funktion mit Knick genauer interpoliert als eine mit Sprung und die widerum genauer als eine Funktion mit Polstelle (siehe Tabelle 7.9). Außerdem entscheidet noch die Lage der Singularität über die zu erwartende Genauigkeit der Interpolation. Die in den Endpunkten des Interpolationsgebiets liegende Singularität wird äußerst ungenau von den Dünngitter-Verfahren interpoliert. Liegt die Singularität nur am Rand des Interpolationsgebiets wird die Interpolation schon genauer. Am besten schneiden die Dünngitter-Verfahren ab, wenn die Singularität im Innern liegt (siehe Tabelle 7.12).
Die glatten Funktionen werden also am genauesten interpoliert, danach folgen die Funktionen mit Singularitäten. Der Träger einer Singularität kann mehrere Formen annehmen. Entspricht der Träger der Singularität einer parallelen Ebene zu den Achsen wird diese Funktion genauer interpoliert, als wenn der Träger nur einem Punkt entspricht. Hat die Funktion eine schräge Ebene als Singularität, so ist die zugehörige Interpolation noch ungenauer als die mit einer Punktsingularität. Nimmt der Träger der Singularität die Form eines Kreises oder kompliziertere Gebilde an, so wird das Ergebnis der Interpolation einer Funktion mit einer solchen Singularität noch ungenauer im Vergleich zu den Funktion vorheriger Singularitäten (siehe Tabelle 7.15).
Zum Ablschuss dieses Abschnitts sind die Ergebnisse noch einmal in einer Tabelle optisch festgehalten (nach Genauigkeit sortiert):

Genauigkeit	1	2	3	4
Dimension der Funktion	2	3	4	5
Glattheit der Funktion	glatt	singulär		
Träger der Singularität	parallele Ebene	Punkt	schräge Ebene	Kreis
Art der Singularität	Knick	Sprung	Polstelle	
Lage der Singularität	Innen	Rand	Endpunkt	

Tabelle 7.16: Funktionen, durch ihre Eigenschaften unterteilt, geordnet nach dem Interpolationsfehler (ordinal), der bei der Interpolation dieses Funktionstyps gemacht wird (1=gut,. . . ,4=schlecht)

7.2 Vektorwertige Funktionen

Wir erweitern unsere Betrachtungen jetzt, indem wir von den einfachen Funktionen zu den vektorwertigen Funktionen übergehen. Das heißt, wir lassen nun den Bildraum $\mathbb{R}^m$ mit $m \in \mathbb{N}$ zu und beschränken uns nicht weiter auf den Raum $\mathbb{R}$. Man kann eine vektorwertige Funktion f als einen Vektor von m verschiedenen einfachen Funktionen f_j, $j = 1 \dots m$, ansehen, wenn $f : \bar{\Omega} \to \mathbb{R}^m$ und $f_j : \bar{\Omega} \to \mathbb{R}$ gilt. Diese Aufteilung in Teilfunktionen ist bei der Interpolation einer Funktion ein großer Vorteil, da jede Teilfunktion f_j, $j = 1 \dots m$, für sich interpoliert werden kann. Wir werden dies in den ausgewählten Beispielen sehen. Außerdem werden wir die Funktionsinterpolation mit der Lookahead-Methode (siehe Abschnitt 5.3.3) genauer unter die Lupe nehmen. Wir zeigen anhand der Beispiele, dass man durch die Lookahead-Methode einige Funktionen, die mit dem normalen adaptiven Dünngitter-Verfahren nicht oder nur schwer interpoliert werden können, besser interpolieren kann.

Beispiel MFK(1,3): Unser erstes Beispiel, eine Schraubenkurve, dient dazu, den Vorteil, den wir bei der Interpolation trigonometrischer Funktionen mit der Lookahead-Methode erhalten zu veranschaulichen.

SK Schraubenkurve mit Höhe $h = 1$ und Radius $R = 1$ auf einem $[0, 1]$-Gitter

$$f(x_1) = \begin{pmatrix} cos(4 \cdot \pi \cdot x_1) \\ sin(4 \cdot \pi \cdot x_1) \\ 4 \cdot \pi x_1 \end{pmatrix}$$

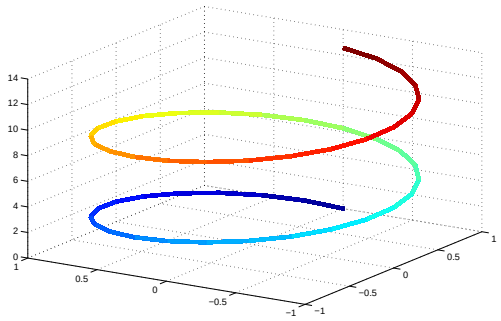


Abb. 7.11: Mannigfaltigkeit SK

Wir berechnen den Fehler, den wir bei der Interpolation der Funktion SK machen, mit der Fehlerberechnungsmethode mit Produktgitterpunkten (FP) und der Schrittweite $h = \frac{1}{200000}$. Die minimalste Schranke, die das ADGV benötigt, um den Interpolationsfehler des DGVs zu erreichen, liegt für den L_2-Fehler bei $\epsilon = 4^{-18}$ und für den L_∞-Fehler bei $\epsilon = 4^{-15}$. Dabei ist das Startlevel des adaptiven Gitters abhängig von der Hierarchisierungsmethode, d.h. ob mit der Lookahead-Methode oder ohne die Lookahead-Methode hierarchisiert wird. Wenn wir mit der Lookahead-Methode interpolieren, starten wir beim Level 2 und ohne die Loookahead-Methode erst beim Level 3, da die Interpolation ohne die Lookahead-Methode erst dann greift. Je kleiner das Startlevel ist, desto größer ist die Chance, dass wir für die Interpolation wenig Punkte benötigen. Benutzen wir beim adaptiven Dünngitter-Interpolationsverfahren den

Fkt.	DG_{L_2}	ADG_{L_2}	DG_{L_∞}	ADG_{L_∞}
SK	2.0020	2.0061	2.0033	1.9820

Tabelle 7.17: Konvergenzrate für das Beispiel MFK(1,3) interpoliert mit der Lookahead-Methode

Lookahead-Algorithmus, so erhalten wir für die Funktion SK eine gleichmäßig fallende Fehlerkurve, wie man den Konvergenzgraphen in Abbildung 7.12 entnehmen kann. Interpolieren wir hingegen ohne den Lookahead-Algorithmus, so ist die Fehlerkurve zur Funktion SK zu Anfang konstant und fällt am Ende sehr steil ab. Siehe dazu die Konvergenzgraphen in Abbildung 7.13. Bei einer fast konstanten Funktion, wie es die Fehlerkurve bei der Interpolation ohne den Lookahead-Algorithmus ist, kann keine vernünftige globale Steigung berechnet werden, also auch keine Konvergenzrate. Im Fall der Interpolation mit dem Lookahead-Algorithmus kann die globale Steigung der Fehlerkurve hingegen gut berechnet werden und damit auch die Konvergenzrate. Da die Funktion SK aus drei glatten einfachen Funktionen besteht, war die gute Konvergenzrate von zwei (siehe Tabelle 7.17) zu erwarten und spiegelt die guten Ergebnisse wieder, die wir auch bei den glatten einfachen Funktionen hatten.

Da hier jede Teilfunktion für sich interpoliert wird, können bei jeder einzelnen Teilfunktion Punkte im Vergleich zur Interpolation mit dem DGV eingespart werden. In diesem Beispiel können insbesondere in der letzten Teilfunktion $SK_3(x_1) = 4 \cdot \pi x_1$ Punkte eingespart werden, da sie nur linear ist. Da wir in unserem ADGV für jede Teilfunktion dasselbe Startlevel ver-

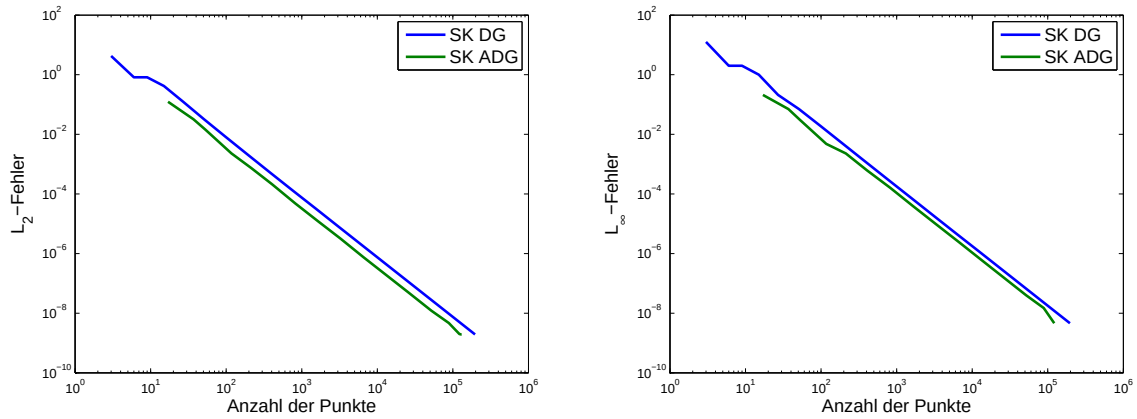


Abb. 7.12: L_2- und L_∞-Fehler zum Beispiel MFK(1,3) mit dem Lookahead-Algorithmus beim ADGV

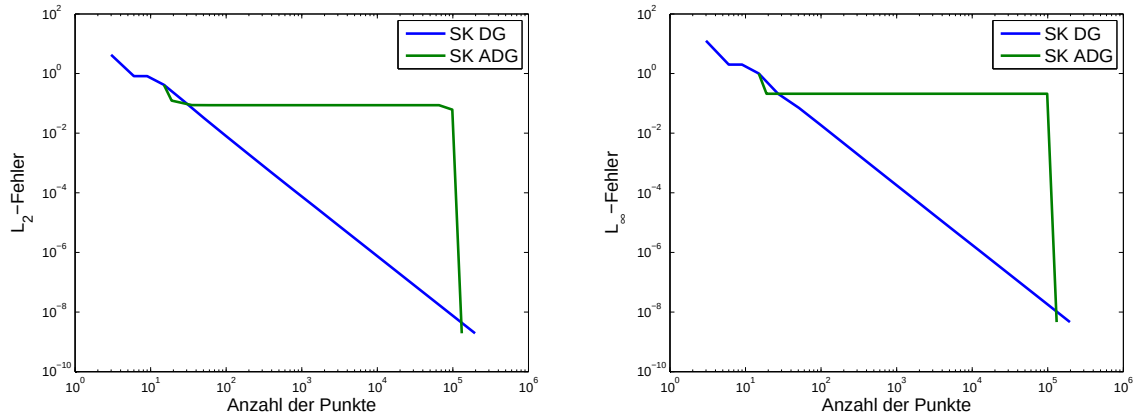


Abb. 7.13: L_2- und L_∞-Fehler zum Beispiel MFK(1,3) ohne den Lookahead-Algorithmus beim ADGV

wenden, könnten wir die Anzahl der Gitterpunkte noch verbessern, indem wir das Startlevel für jede Teilfunkton unabhängig vom Startlevel der anderen Teilfunktionen wählten.

Fkt.	L_2	L_∞	P_{DG}	$\mathrm{P}_{\mathrm{ADG}}^{L_2}$	$\mathrm{P}_{\mathrm{ADG}}^{L_\infty}$
SK	1.937794e-09	4.595749e-09	196611	130933	121749

Tabelle 7.18: Minimaler Interpolationsfehler zum höchsten Level 16 für das Beispiel MFK(1,3) mit der Lookahead-Methode

Schauen wir uns nach der eindimensionalen vektorwertigen Funktion im nächsten Beispiel noch drei zweidimensionale vektorwertige Funktionen an, die Parametrisierungen zu bekannten Mannigfaltigkeiten sind.

Beispiel MFK(2,3): Wir haben für dieses Beispiel drei verschiedene Mannigfaltigkeiten genommen, den Torus, den Sattel und die Sanduhr.

T Torus mit äußerem Radius $R = 10$ und innerem Radius $r = 3$ auf einem $[0, 1] \times [0, 1]$-Gitter (in Abbildung 7.14 links):

$$f(x_1, x_2) = \begin{pmatrix} 10 \cdot cos(2 \cdot \pi \cdot x_1) + 2 \cdot cos(2 \cdot \pi \cdot x_1) \cdot cos(2 \cdot \pi \cdot x_2) \\ 10 \cdot sin(2 \cdot \pi \cdot x_1) + 2 \cdot sin(2 \cdot \pi \cdot x_1) \cdot cos(2 \cdot \pi \cdot x_2) \\ 2 \cdot sin(2 \cdot \pi \cdot x_2) \end{pmatrix} \tag{7.1}$$

S Sattel (in Abbildung 7.14 mitte):

$$f(x_1, x_2) = \begin{pmatrix} 1 - 2 \cdot x_1 \\ 1 - 2 \cdot x_2 \\ (1 - 2 \cdot x_1)^2 - (1 - 2 \cdot x_2)^2 \end{pmatrix} \tag{7.2}$$

SU Sanduhr (in Abbildung 7.14 rechts):

$$f(x_1, x_2) = \begin{pmatrix} cos(2 \cdot \pi \cdot x_1) \cdot cos(2 \cdot \pi x_2) \\ cos(2 \cdot \pi \cdot x_1) \cdot sin(2 \cdot \pi x_2) \\ 2 \cdot \pi x_1 \end{pmatrix} \tag{7.3}$$

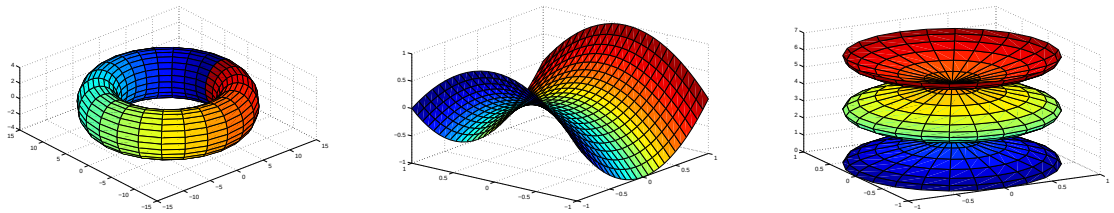


Abb. 7.14: Mannigfaltigkeiten T, S und SU

Zur Interpolation der Funktionen T und SU auf adaptiven dünnen Gittern ist die Hierarchisierung mit dem Lookahead-Algorithmus sinnvoll, da sonst die ϵ-Schranke sehr klein gewählt werden müsste, um die gleiche Genauigkeit wie mit einem regulären Dünngitter-Verfahren zu erreichen. Außerdem müssen wir, wenn wir ohne Lookahead die Funktionen T und SU interpolieren, mit größerem Level starten, als wenn wir mit Lookahead arbeiten. Die Chance mit wenig Gitterpunkten die Funktionen T und SU zu interpolieren ist im Fall des ADGVs mit Lookahead größer als beim ADGV ohne Lookahead. Die ϵ-Schranken und die Startlevel zur Interpolation der Funktionen T, S und SU mit einem adaptiven Dünngitter-Verfahren sehen wie in Tabelle 7.19 aufgelistet aus:

Funktion	S	SU	T
Startlevel ohne Lookahead	1	3	4
Startlevel mit Lookahead	-	1	1
ϵ-Schranke ohne Lookahead	4^{-11}	4^{-30}	4^{-28}
ϵ-Schranke mit Lookahead	-	4^{-10}	4^{-12}

Tabelle 7.19: Zur Interpolation der Funktionen S, SU und T gewählte ϵ-Schranke und das gewählte Startlevel

Betrachten wir zunächst die Interpolationsfehler und die dazu benötigten Dünngitter-Punkte. In Tabelle 7.20 sind die mit der Fehlerberechnungsmethode mit Produktgitter-Punkten mit Schrittweite $\frac{1}{445}$ (FP($\frac{1}{445}$)) berechneten L_2-Fehler und L_∞-Fehler zu den Funktionen T, S und SU aufgelistet und die verwendete Gitterpunktanzahl, einmal für die Interpolation mit Lookahead und einmal für die Interpolation ohne Lookahead.

				ohne Lookahead		mit Lookahead	
Fkt.	L_2	L_∞	P_{DG}	$P_{ADG}^{L_2}$	$P_{ADG}^{L_\infty}$	$P_{ADG}^{L_2}$	$P_{ADG}^{L_\infty}$
T	5.404105e-05	1.874469e-04	64515	37517	37517	37552	32288
S	1.457479e-08	5.960434e-08	64515	8206	8206	-	-
SU	2.704870e-05	9.374837e-05	64515	32782	32782	28847	22143

Tabelle 7.20: Minimaler Interpolationsfehler zum höchsten Level 12 für das Beispiel MFK(2,3)

Da die Funktion S aus zwei linearen Teilfunktionen und einer quadratischen Teilfunktion besteht und somit die hierarchischen Überschüsse abfallend sind, ist es unnötig die Hierarchisierung mit Lookahead durchzuführen.
Das Bild der Funktion S ist eine Teilmenge des $\mathbb{R}^3$, also kann das Bild in drei Dimensionsrichtungen unterteilt werden. Jede Dimensionsrichtung wird für sich bearbeitet, deswegen erhalten wir 3 verschiedene adaptive Dünngitter-Interpolanten, einen in x-Richtung, einen in y-Richtung und einen in z-Richtung. Diese sind in der Abbildung 7.15 zu sehen. Da in Dimensionsrichtung x und y lineare Funktionen zu interpolieren sind, braucht unser adaptives Dünngitter-Verfahren nur jeweils 3 Punkte dafür. In Dimensionsrichtung z haben wir ein Polynom zweiten Grades, bei welchem der Rand sehr stark verfeinert werden muss, um die Funktion gut zu interpolieren. Dafür benötigen wir 8200 Punkte. Hier hat sich unsere Wahl für den einseitig konstanten Rand schon bezahlt gemacht, da wir in zwei Dimensionsrichtungen fast gar keine Punkte brauchen.
Betrachten wir die Funktionsdarstellung der Sanduhr (Gleichung 7.3), so sehen wir, dass in z-Richtung, also die dritte Teilfunktion von SU, die Funktion linear ist. Somit können wir erwarten, dass wir wenig Punkte zur Interpolation der dritten Teilfunktion benötigen. Abbildung 7.16 bestätigt diese Erwartung. Es sind gerade mal drei Punkte, die das ADGV benötigt um diese Teilfunktion exakt zu interpolieren. Besonders interessant zu untersuchen ist die Funktion T, weil $T_3(x_1, x_2) = 2 \cdot sin(2 \cdot \pi \cdot x_2)$ nur von einer Variablen abhängt und nicht linear ist. Wir erwarten hier, dass in x_2-Richtung viele Punkte gebraucht werden, aber in x_1-Richtung fast gar keine. Betrachten wir dazu abschließend die Abbildung 7.17, so sehen wir, dass, um die dritte Teilfunktion darstellen zu können, nur zwei Sinuskurven nötig waren und keine gesamte

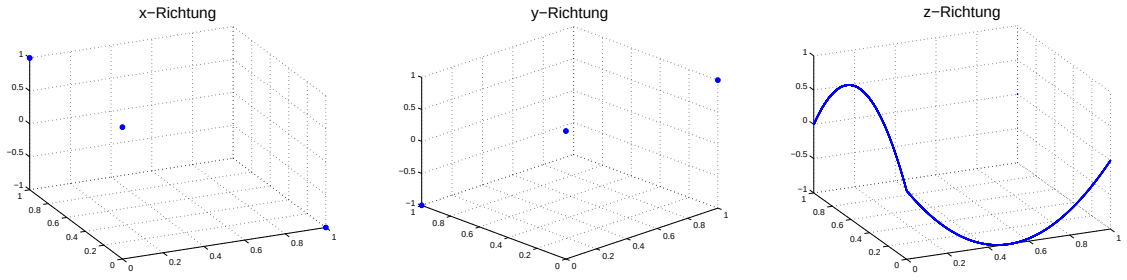


Abb. 7.15: Die drei verschiedenen adaptiven dünnen Gitter der Funktion S in die Dimensionsrichtung x,y und z bei $\epsilon = 4^{-11}$ mit den Funktionswerten geplottet.

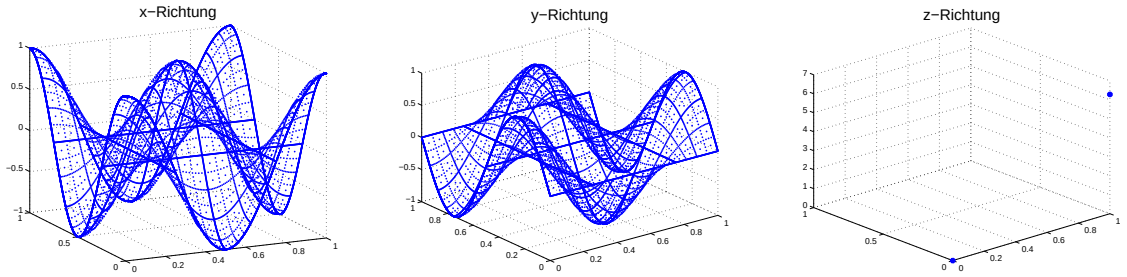


Abb. 7.16: Die drei verschiedenen adaptiven dünnen Gitter mit Lookahead der Funktion SU in Richtung x,y und z bei $\epsilon = 4^{-10}$ mit den Funktionswerten geplottet.

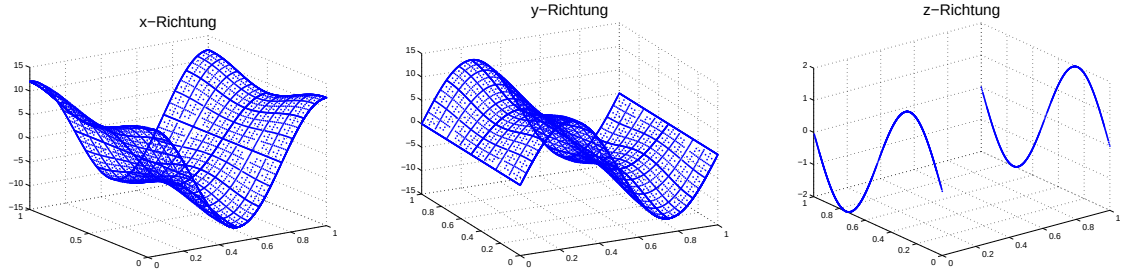


Abb. 7.17: Die drei verschiedenen adaptiven dünnen Gitter mit Lookahead der Funktion T in Richtung x,y und z bei $\epsilon = 4^{-12}$ mit den Funktionswerten geplottet.

Sinusebene. Also wird die Punktanzahl, die nötig ist um die Funktion mit einem ADGV genauso gut wie mit einem DGV zu interpolieren, auch durch die Anzahl an abhängigen Variablen beeinflusst und nicht nur durch den Typ der zu interpolierenden Funktion.
Zum Schluss unserer Betrachtung des **Beispiels MFK(2,3)** werden wir noch über die Konvergenzrate (Tabelle 7.21) und die Konvergenzgraphen (Abbildung 7.18 und Abbildung 7.19) sprechen. In Tabelle 7.21 ist zu sehen, dass das Verfahren zur Interpolation der Funktion T eine ähnliche Konvergenzrate wie das Verfahren zur Interpolation der Funktion SU aufweist. Dieses

Ergebnis wird durch den Konvergenzgraph (Abbildung 7.18) bestätigt. Die ähnliche Konvergenzrate bei der Interpolation der Funktionen T und SU ist darauf zurückzuführen, dass deren Teilfunktionen ähnlich aufgebaut sind. Die erste und zweite Teilfunktion bestehen aus zusammengesetzten Sinus- und Cosinus-Funktionen. Außerdem hängt die letzte Teilfunktion von SU und T von derselben Variable ab. Sie unterscheiden sich nur dadurch, dass die letzte Teilfunktion bei SU linear ist und T eine Sinus-Funktion. Das Ergebnis, dass das Interpolationsverfahren für die Funktion S die beste Konvergenzrate und die höchste Genauigkeit aufweist, ist leicht zu erklären. Die Funktion S besteht aus drei einfach zu interpolierenden Teilfunktionen, wobei die ersten zwei Teilfunktionen sogar linear und nur von einer Variablen abhängen und die dritte Teilfunktion nur quadratisch ist.

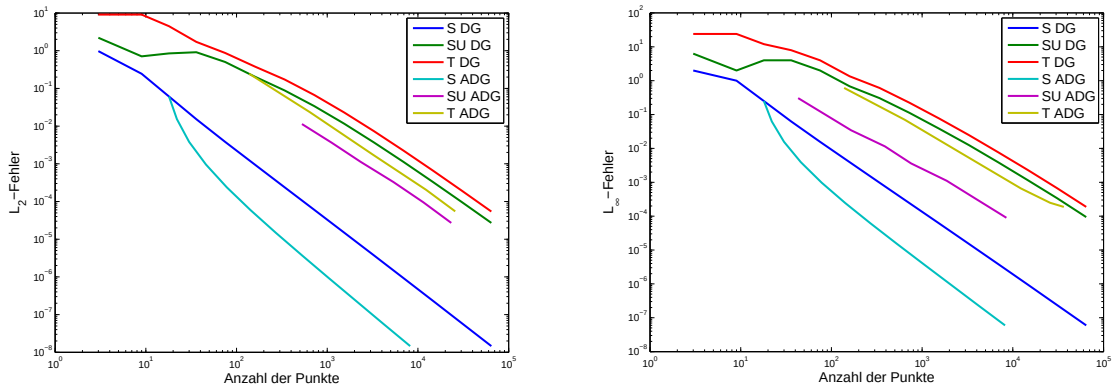


Abb. 7.18: L_2- und L_∞-Fehler zum Beispiel MFK(2,3), wobei zur Interpolation von SU und T ein ADGV mit Lookahead benutzt wurde

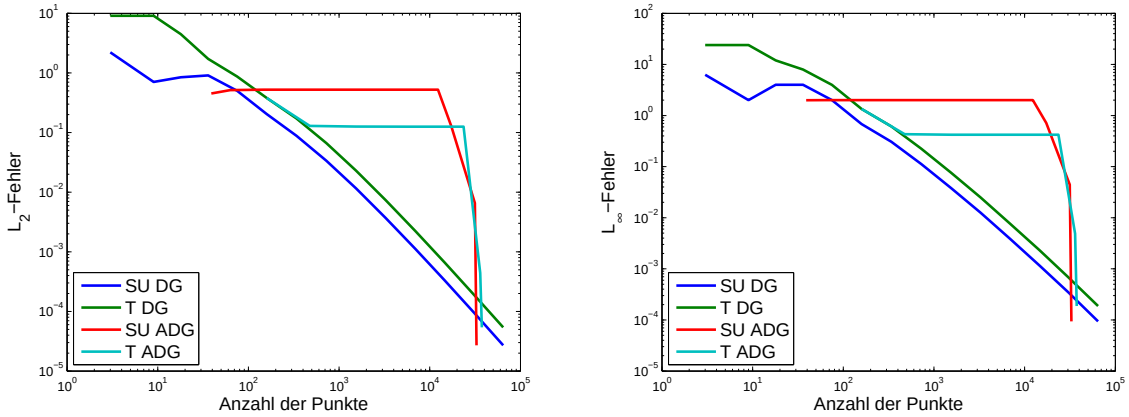


Abb. 7.19: L_2- und L_∞-Fehler zu den Funktionen SU und T, wobei zur Interpolation von SU und T ein ADGV ohne Lookahead benutzt wurde

Fkt.	DGK_{L_2}	$ADGK_{L_2}$	DGK_{L_∞}	$ADGK_{L_\infty}$
T	1.3157	1.5816	1.3759	1.5254
S	1.8458	2.3029	1.8522	2.1083
SU	1.1645	1.5353	1.2912	1.5182

Tabelle 7.21: Konvergenzrate für das Beispiel MFK(2,3), wobei zur Interpolation von S und T ein ADGV mit Lookahead benutzt wurde

Fazit

Wir haben uns in diesem Abschnit vier vektorwertige Funktionen angeschaut und analysiert. Dabei war zu beobachten, dass die Vereinfachung der vektorwertigen Funktionen auf m einfache Funktionen positive Effekte auf die benötigte Gitterpunktanzahl hat, da jede Teilfunktion für sich interpoliert wird.
Zwei weitere Möglichkeiten das ADGK-Verfahren für vektorwertige Funktionen zu optimieren gibt es noch. Einerseits wird in unserem Verfahren das Startlevel für alle Teilfunktionen gleichgesetzt und andererseits ist die ϵ-Schranke für alle Teilfunktionen identisch. Wenn man aber im Interpolationsverfahren nun jeder Teilfunktion das eigene Startlevel und die eigene ϵ-Schranke zukommen lässt, kann man die benötigte Gitterpunktanzahl noch verringern.
Außerdem haben wir gesehen, wie wir durch die Hierarchisierung mit dem Lookaheadalgorithmus Funktionen besser interpolieren können, und dass dadurch die Möglichkeit geschaffen wurde die Konvergenzrate zu diskutieren. Speziell die trigonometrischen Funktionen können so entscheidend qualitativ besser interpoliert werden.

7.3 Hintereinanderschaltung von Funktionen

In Kapitel 4 haben wir über die Hintereinanderschaltung von zwei Funktionen und den Interpolationsfehler, der bei der Interpolation solcher Funktionen entsteht, diskutiert. Jetzt wollen wir uns drei Varianten anschauen die Interpolation der Hintereinanderschaltung von Funktionen durchzuführen. Folgend sei mit " $\widehat{\ }$ " der Interpolant der zu interpolierenden Funktion gekennzeichnet. Seien also drei Funktionen g, f und F gegeben, so dass $f \circ g = F$ gilt, dann können wir als nächstes drei Varianten betrachten, $f \circ g$ zu interpolieren:

1. $\hat{F}_1 = \widehat{f \circ g}$
2. $\hat{F}_2 = \widehat{f \circ \hat{g}}$
3. $\hat{F}_3 = \hat{f} \circ \hat{g}$

Wir erhalten drei verschiedene Interpolanten $\hat{F}_1$, $\hat{F}_2$ und $\hat{F}_3$ der Funktion F , deren Genauigkeit wir in diesem Abschnitt unter die Lupe nehmen wollen.

Als erstes stellen wir dazu alle drei Fälle mittels der Basisfunktionen dar:

- $\hat{F}_1(x) = \sum_l \sum_i F_{li}\phi_{li}(x)$
- $\hat{F}_2(x) = \sum_l \sum_i (f\hat{g})_{li}\phi_{li}(x)$
- $\hat{F}_1(x) = \sum_l \sum_i f_{li}\phi_{li}(\sum_k \sum_j g_{kj}\phi_{kj}(x))$,

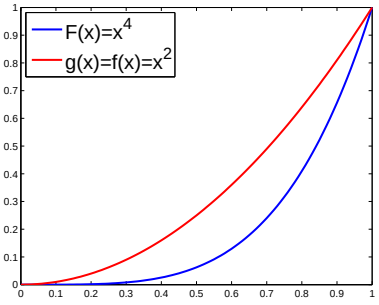
wobei f_{li} als der Basiskoeffizient in Abhängigkeit von f zu ϕ_{li}, F_{li} als der Basiskoeffizient in Abhängigkeit von F zu ϕ_{li}, g_{kj} als der Basiskoeffizient in Abhängigkeit von g zu ϕ_{kj} und $(f\hat{g})_{li}$ als der Basiskoeffizient in Abhängigkeit von $f \circ \hat{g}$ zu ϕ_{li} zu verstehen ist.
Wir werden an mehreren Beispielfunktionen den Interpolationsfehler für alle drei Varianten betrachten und diskutieren.

Beispiel 1:
Wir fangen mit einem ganz einfachen Fall an. Wir wählen für F, f und g eindimensionale glatte Funktionen mit $x, y \in [0, 1]$.

F: $F(x) = x^4$

f: $f(y) = y^2$

g: $g(x) = x^2$



Die Funktionen haben wir jeweils zum maximalen Level 10 und zum Startlevel 1 mit dem adaptiven Dünngitter-Interpolationsverfahren (ADG-Verfahren) interpoliert. Dabei haben wir

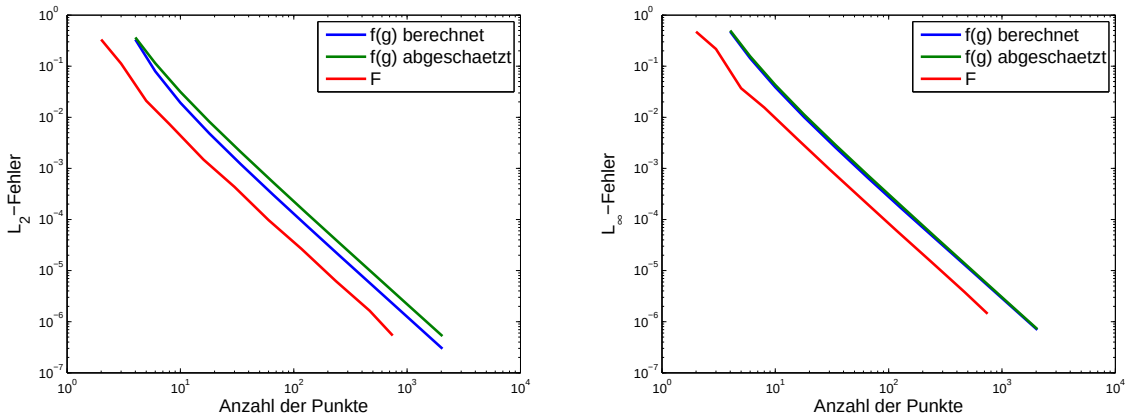


Abb. 7.20: Geplottet ist jeweils die Interpolationsfehler zum Interpolanten $\hat{F}_3$, einmal mit der FP-Methode berechnet und einmal nach Satz 4.4 abgeschätzt, und der interpolationsfehler zum Interpolanten $\hat{F}_1$ gegen die benötigte Anzahl an Punkten in einer logarithmischen Skala.

die ϵ-Schranke beginnend bei eins nach jeder Interpolation immer mit 4^{-1} multipliziert bis die endgültige Schranke von 4^{-20} erreicht war.
Beginnen wir mit dem Vergleich der Interpolationsfehler e_1 und e_3 zu den Interpolanten $\hat{F}_1$ und $\hat{F}_3$. In Abbildung 7.20 sieht man, dass die Abschätzung des Interpolationsfehlers mit der Gleichung 4.3 aus Satz 4.4, also der Abschätzung des Interpolationsfehlers von $f \circ g$ durch die Summe des Interpolationsfehlers von f und des Interpolationsfehlers von g, eine gute obere Schranke für den berechneten Interpolationsfehler e_3 ist.
Die Fehlerkurven der Interpolationsfehler e_1 und e_3 weisen die gleiche Steigung auf, also haben beide Methoden die gleiche Konvergenzrate (vergleiche Abschnitt 3.2.2), aber der Aufwand, um den Interpolanten $\hat{F}_1$ zu berechnen ist geringer, als der Aufwand, um den Interpolanten $\hat{F}_3$ zu berechnen (siehe Abbildung 7.20).
Wenn wir f und g zu denselben Bedingungen interpolieren, d.h. dieselbe ϵ-Schranke und dasselbe Maximallevel zur Interpolation benutzen, dann macht die Betrachtung des Interpolanten $\hat{F}_2$ keinen Sinn, da in diesem Fall $\hat{F}_2 = \hat{F}_1$ gilt. Machen wir aber die genannten Bedingungen variabel, also benutzen unterschiedliche ϵ-Schranken oder unterschiedliche Maximallevel zur Interpolation von f und g, so können wir darüber diskutieren, ob und wie eine solche Änderung den Interpolationsfehler beeinflusst. Auf Grund dieser Überlegungen vergleichen wir nun die Interpolanten $\hat{F}_2$ und $\hat{F}_3$. Dazu haben wir einmal die Funktion g bis zum maximalen Level 6 und die Funktion f bis zum maxiamlen Level 14 interpoliert, und das andere Mal g bis zum Level 14 und f bis zum Level 6. Ansonsten sind wir genauso vorgegangen wie zuvor.
Die Abschätzung des Interpolationsfehlers e_3 ist auch bei dieser Vorgehensweise eine gute obere Schranke des berechneten Interpolationsfehlers e_3, wie in den Abbildungen 7.21 und 7.22 zu sehen ist. Außerdem ist in den beiden Abbildungen auch zu erkennen, dass die zweite Methode den Interpolanten von F zu bestimmen, wenn F in zwei identische Funktionen f und g aufgeteilt wird, die schlechtere im Vergleich zur dritten Methode ist.

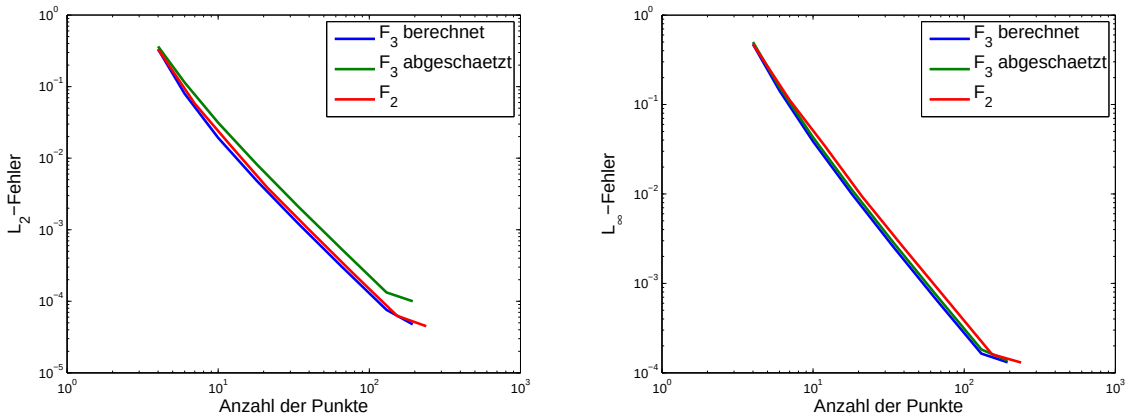


Abb. 7.21: Geplottet ist jeweils der Interpolationsfehler zum Interpolanten $\hat{F}_3$, einmal mit der FP-Methode berechnet und einmal nach Satz 4.4 abgeschätzt, und der Interpolationsfehler zum Interpolanten $\hat{F}_2$ gegen die benötigte Anzahl an Punkten in einer logarithmischen Skala. Dabei ist f bis zum Level 14 interpoliert worden und g bis zum Level 6.

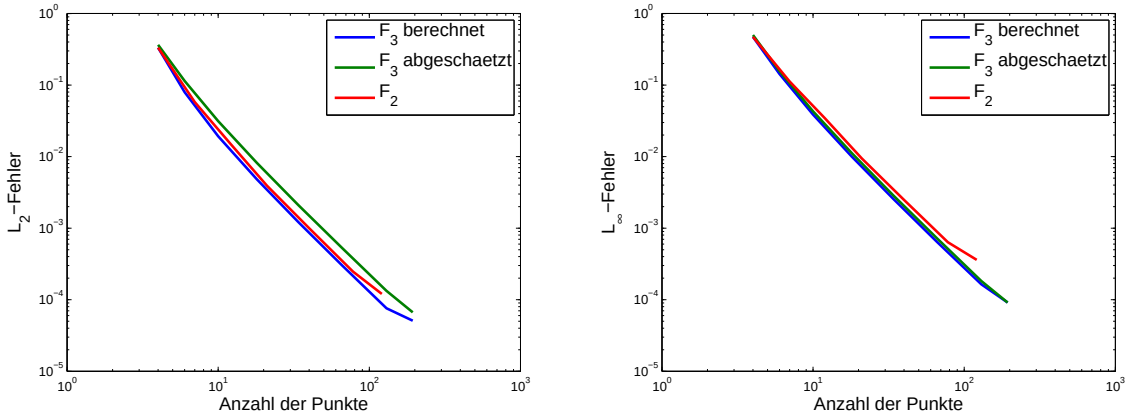


Abb. 7.22: Geplottet ist jeweils der Interpolationsfehler zum Interpolanten $\hat{F}_3$, einmal mit der FP-Methode berechnet und einmal nach Satz 4.4 abgeschätzt, und der Interpolationsfehler zum Interpolanten $\hat{F}_2$ gegen die benötigte Anzahl an Punkten in einer logarithmischen Skala. Dabei ist f bis zum Level 6 interpoliert worden und g bis zum Level 14.

Das Spannendste aber ist die Betrachtung der dritten Methode, also die Bestimmung von F_3, wenn wir unterschiedliche Level für f und g benutzen. Auf Grund der Tatsache, dass die Interpolationsfehler zu nah bei dieser Methodefür unterschiedliche Level aneinander liegen und so in einem Graphen nicht vernünftig darstellbar sind, müssen wir die Ergebnisse tabellarisch betrachten. Wir bezeichnen in Tabelle 7.22 mit $e_3(l_f, l_g)$ den L_2-Interpolationsfehler, wobei f auf einem Gitter mit maximalen Level l_f und g auf einem maximalen Level l_g interpoliert wor-

ϵ	$P_3(10,10)$	$e_3(10,10)$	$P_3(6,14)$	$e_3(6,14)$	$P_3(14,6)$	$e_3(14,6)$
1.000000e+00	4	3.333167e-01	4	3.333167e-01	4	3.333167e-01
6.250000e-02	6	7.952600e-02	6	7.952600e-02	6	7.952600e-02
1.562500e-02	10	1.928768e-02	10	1.928768e-02	10	1.928768e-02
3.906250e-03	18	4.831089e-03	18	4.831089e-03	18	4.831089e-03
9.765625e-04	34	1.210723e-03	34	1.210723e-03	34	1.210723e-03
2.441406e-04	66	3.030908e-04	66	3.030908e-04	66	3.030908e-04
6.103516e-05	130	7.586154e-05	130	7.586154e-05	130	7.586154e-05
1.525879e-05	258	1.897791e-05	194	5.087809e-05	194	4.760017e-05
3.814697e-06	514	4.747257e-06	322	4.550719e-05	322	4.165545e-05
9.536743e-07	1026	1.187239e-06	578	4.428355e-05	578	4.030212e-05
2.384186e-07	2050	2.969250e-07	1090	4.398615e-05	1090	3.997394e-05
5.960464e-08			2114	4.391225e-05	2114	3.989244e-05

Tabelle 7.22: $P_3(l_f, l_g)$ steht für die benötigte Punktanzahl und $e_3(l_f, l_g)$ für den L_2-Fehler für den Interpolanten F_3, wobei f auf einem Gitter mit maximalen Level l_f und g auf einem maximalen Level l_g interpoliert worden ist.

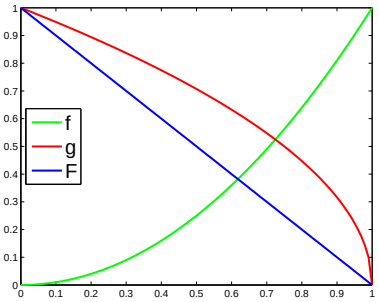
den ist. Die gleiche Notation gebrauchen wir für die benötigte Anzahl an Punkten $P_3(l_f, l_g)$. Da f und g identische Funktionen sind, lässt sich hier eine Aussage über den Einfluss des Interpolationsfehlers der inneren und der äußeren Funktion zum Gesamtfehler machen. In der Tabelle 7.22 ist zu sehen, dass der Interpolationsfehler $e_3(6,14)$ größer ist als der Interpolationsfehler $e_3(14,6)$. Das heißt, dass der Interpolationsfehler der äußeren Funktion einen größeren Einfluss auf den Gesamtfehler hat als der Interpolationsfehler der inneren Funktion.
Außerdem erhalten wir eine höhere Genauigkeit, wenn die Funktionen f und g ungefähr bis zu demselben maximalen Level interpoliert werden. Betrachten wir die Interpolationsfehler in der Tabelle 7.22 bis zur Punktanzahl von 130, sind keine Unterschiede zu erkennen, da bis hierhin nur maximal Punkte des Levels 6 vorkommen. Danach wird der Interpolationsfehler $e_3(10,10)$ kleiner als die Interpolationsfehler $e_3(6,14)$ und $e_3(14,6)$, da hier beide Funktionen f und g mit Punkten bis zum Level 10 interpoliert werden. Bei den Interpolationsfehlern $e_3(14,6)$ und $e_3(6,14)$ wird nur eine der beiden Funktionen mit Punkten höherer Level als 6 interpoliert. Es ist also sinnvoll bei der Interpolation der inneren und der äußeren Funktion dieselben Interpolationsbedingungen, wie das maximale Level und die ϵ-Schranke, festzulegen, um den optimalen Interpolationsfehler mit der dritten Methode zu erreichen.

Beispiel 2: Betrachten wir noch weitere Funktionen:

F: $F(x) = |x-1|$

f: $f(y) = x^2$

g: $g(x) = \sqrt{|x-1|}$



Wir haben hier bewusst eine lineare Funktion für F und nichtlineare Funktionen für g und f gewählt. Für F braucht das Interpolationsverfahren genau zwei Punkte, um die Gerade optimal zu beschreiben.
Wenn wir die Funktionen f und g bis zum Level 10 interpolieren, erhalten wir für beide Interpolanten einen Interpolationsfehler der größer als Null ist. Damit kann der Gesamtfehler, wenn wir f und g hintereinanderschalten, auf Grund der Ineinandersetzung nicht Null werden. Der Fehler der durch die Interpolation der inneren Funktion an einer beliebigen Stelle gemacht wird, kann nur vom Interpolanten der äußeren Funktion leicht verbessert werden, aber nicht verhindert werden. Im **Beispiel 1** konnten wir aber auf Grund der Wahl der inneren und äußeren Funktion einen ähnlichen Interpolationsfehler mit der dritten Methode wie mit der ersten Methode erreichen, wenn wir zur Interpolation dasselbe maximale Level gebrauchten. Nur der Aufwand der Interpolation der Hintereinanderschaltung war größer als der Aufwand der Interpolation der Originalfunktion.
Daraus können wir folgern, dass die Genauigkeit der Interpolation der Hintereinanderschaltung durch die Wahl der Funktionen f und g stark beeinflusst wird. Zur Bestätigung unserer Aussage betrachten wir als nächstes die Abbildung. 7.23.

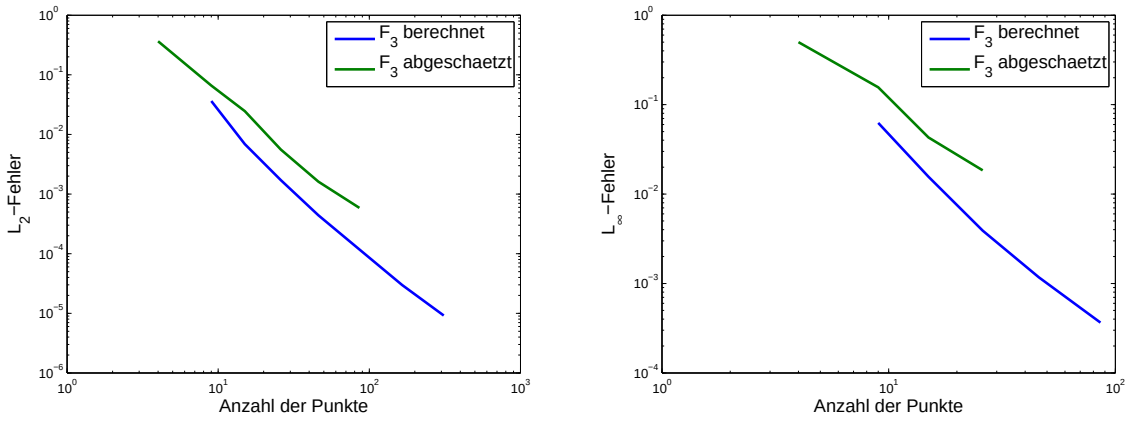


Abb. 7.23: Geplottet ist jeweils der Interpolationsfehler zum Interpolanten $\hat{F}_3$, einmal mit der FP-Methode berechnet und einmal nach Satz 4.4 abgeschätzt gegen die benötigte Anzahl an Punkten in einer logarithmischen Skala. Dabei sind f und g bis zum Level 10 interpoliert worden.

Die Fehlerkurve für den ersten Interpolanten $\hat{F}_1$ kann leider nicht graphisch dargestellt werden, da sie konstant Null ist. Der abgeschätzte Interpolationsfehler ist auch hier eine obere Schranke, aber die Schranke ist nicht so genau wie im **ersten Beispiel**.
Zur Vollständigkeit sind der minimalste L_2-Interpolationsfehler und der minimalste L_∞-Interpolationsfehler zu den Funktionen f, g, F und $f(g)$ mit der benötigten Gitterpunktanzahl in der Tabelle 7.23 vermerkt. Wenn wir den Fehler betrachten, sehen wir, dass f genauer interpoliert wird als g (Tabelle 7.23). Der Gesamtfehler liegt zwischen den Interpolationsfehlern von f und g. Also wird in diesem Beispiel der Interpolationsfehler, der von $\hat{g}$ gemacht wird, nicht durch $\hat{f}$ verstärkt, sondern ein wenig gemindert.

Funktion	Anzahl der Punkte	L_2-Fehler	L_∞-Fehler
f	1025	1.741076e-07	2.384180e-07
g	182	1.771574e-04	7.742136e-03
f(g)	1059	5.719007e-06	2.434082e-04
F	2	0.000000e+00	0.000000e+00

Tabelle 7.23: Hier sind die Interpolationsfehler zu den Funktionen f, g und die Interpolationsfehler der Interpolanten $\hat{F}_1$ (F), $\hat{F}_3$ (f(g)) zu sehen, berechnet mit der FP-Methode

Beispiel 3: Zum Schluss betrachten wir noch eine mehrdimensionale Funktion, die wir durch zwei hintereinandergeschaltete Funktionen beschreiben wollen.
Wir werden zwei verschiedene Hintereinanderschaltungen betrachten und vergleichen. Dabei gilt $g_1(x,y) : [0,1]^2 \to [0,1]$, $f_1(x) : [0,1] \to \mathbb{R}$ und $g_2(x,y) : [0,1]^2 \to [0,1]^2$, $f_2(x,y) : [0,1]^2 \to \mathbb{R}$, wenn wir maximal auf einem Gitter des Levels 8 interpolieren, so dass dann die Hintereinanderschaltung der Funktionen f_j und g_j, mit j=1,2, eine Funktion von $[0,1]^2$ nach $\mathbb{R}$ ist.

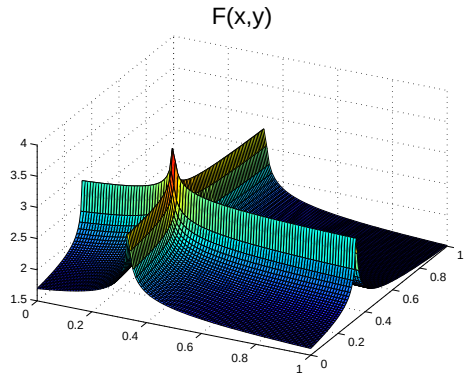
F: $F(x,y) = \sqrt{\left|x-\frac{1}{3}\right|^{-\frac{1}{3}} + \left|y-\frac{1}{3}\right|^{-\frac{1}{3}}}$

f1: $f_1(x) = \sqrt{20 \cdot x}$

g1: $g_1(x,y) = \frac{\left|x-\frac{1}{3}\right|^{-\frac{1}{3}} + \left|y-\frac{1}{3}\right|^{-\frac{1}{3}}}{20}$

f2: $f_2(x,y) = \sqrt{10 \cdot (x+y)}$

g2: $g_2(x,y) = \begin{pmatrix} \frac{\left|x-\frac{1}{3}\right|^{-\frac{1}{3}}}{10} \\ \frac{\left|y-\frac{1}{3}\right|^{-\frac{1}{3}}}{10} \end{pmatrix}$



In diesem Beispiel lassen wir den Interpolanten $\hat{F}_2$ erstmal komplett außer Betracht und beschäftigen uns nur mit der Interpolation über die Interpolanten $\hat{F}_1$ und $\hat{F}_3$.
Wir interpolieren F, f_1, f_2 , g_1 und g_2 jeweils auf einem adaptiven dünnen Gitter des Levels 8 und berechnen danach für jede Funktion den Interpolationsfehler mit der FP-Methode mit einer Schrittweite von 225^{-1}.
In der ersten Tabelle 7.24 sind die berechneten Interpolationsfehler und die dafür benötigte Gitterpunktanzahl zu jeder Funktion aufgelistet. Vergleicht man die letzten drei Zeilen, so sehen wir, dass die Hintereinanderschaltung von f_1 und g_1 fast genauso gut wie die Originalfunktion interpoliert wird. Das heißt, sie unterscheiden sich nur sehr gering in der Genauigkeit und in der dafür benötigten Anzahl an Punkten. Die andere Hintereinanderschaltung $f_2(g_2)$ wird ziemlich schlecht interpoliert. Der Interpolationsfehler und die benötigte Punktanzahl sind wesentlich größer als bei der Interpolation von F. Das kommt daher, dass f_1 und g_1 mit einer höheren Genauigkeit interpoliert werden als f_2 und g_2.

Funktion	Punkte$_{L_2}$	L_2-Fehler	Punkte$_{L_\infty}$	L_∞-Fehler
f_1	257	2.480741e-04	32	2.682511e-03
f_2	1089	1.836352e-02	301	2.012903e-01
g_1	99	6.164707e+02	31	1.310635e+04
g_2	37	8.698903e+02	37	1.310635e+04
$f_1(g_1)$	398	3.368084e+01	398	5.078701e+02
$f_2(g_2)$	687	4.788309e+01	17	5.120000e+02
F	297	3.367917e+01	195	5.075020e+02

Tabelle 7.24: Hier sind die Interpolationsfehler zu den Funktionen f_j, g_j, $(f_j(g_j))$ für j=1,2 und F zu sehen, berechnet mit der FP-Methode.

Betrachten wir als nächstes die Genauigkeit der Interpolanten zu den Funktionen $f_1(g_1)$ und $f_2(g_2)$, einmal mit der FP-Methode berechnet und einmal mit Satz 4.4 abgeschätzt (siehe Tabelle 7.25). Die Abschätzung des Interpolationsfehlers der Hintereinanderschaltung durch die Interpolationsfehler der inneren und der äußeren Funktion ist in beiden Fällen zwar eine obere Schranke, aber die Schranke ist ziemlich weit entfernt vom berechneten Interpolationsfehler. Da die Genauigkeit schon bei der Interpolation der Einzelfunktionen f_j und g_j für j=1,2 sehr niedrig ist (Tabelle 7.24), also der Interpolationsfehler sehr groß ist, wirkt sich das auf die Abschätzung so aus, dass die Abschätzung des Interpolationsfehlers der Funktion $f_j(g_j)$ sehr ungenau wird. Im Umkehrschluss bedeutet das, dass die Abschätzung des Gesamtfehlers für den Interpolanten $\hat{F}_3$ umso genauer wird, desto genauer die Einzelfunktionen interpoliert werden.

Vorgehensweise	L_2-Fehler	L_∞-Feheler
berechnet $(f_1(g_1))$	3.368084e+01	5.078701e+02
abgeschätzt $(f_1(g_1))$	4.411105e+04	9.378139e+05
berechnet $(f_2(g_2))$	4.788309e+01	5.120000e+02
abgeschätzt $(f_2(g_2))$	3.837482e+07	8.692598e+09

Tabelle 7.25: In dieser Abbildung sind die mit der FP-Methode berechneten Interpolationsfehler und die durch Satz 4.4 abgeschätzten Interpolationsfehler zu den Hintereinanderschaltungen zu sehen.

Zuletzt betrachten wir die Interpolation über $\hat{F}_2$ im Vergleich zu der Interpolation über $\hat{F}_3$ am Beispiel der Funktionen g_2 und f_2. Die Interpolation von $f_1(g_1)$ ist vielleicht zweckdienlicher als die Interpolation von $f_2(g_2)$, da diese Hintereinanderschaltung in der vorherigen Betrachtung schon bessere Ergebnisse geliefert hat. Die Hintereinanderschaltung von f_1 und g_1 über $\widehat{f_1(\hat{g_1})}$ ist aber nicht einfach umzusetzen, da die hierarchischen Überschüsse und Gitterpunkte mit der Dimension der äußeren Funktion gespeichert werden müssen. Die Dimension von f_1 ist hier eins und bei der Originalfunktion F zwei. Aus diesem Grund haben wir den Fall, dass die innere Funktion eine größere Dimension als die äußere Funktion hat, nicht implementiert und können so nun nur die Hintereinanderschaltung $f_2(g_2)$ interpolieren und diskutieren.
In der Tabelle 7.26 sind die Ergebnisse zu sehen. Aus diesen Ergebnissen können wir folgern,dass es besser ist die innere Funktion genauer zu interpolieren. Im **Beispiel 1** war das

Level	Funktion	Punkte$_{L_2}$	L_2-Fehler	Punkte$_{L_\infty}$	L_∞-Fehler
(4)	f_2	53	1.251058e-01	26	4.029341e-01
(8)	f_2	1089	1.836352e-02	301	2.012903e-01
(4)	g_2	19	8.699244e+02	19	1.310686e+04
(8)	g_2	37	8.698903e+02	37	1.310635e+04
(4,8)	$f_2(g_2)$ über $\hat{F}_2$	252	3.376819e+01	175	5.093146e+02
(4,8)	$f_2(g_2)$ über $\hat{F}_3$	254	4.788152e+01	17	5.120000e+02
(8,4)	$f_2(g_2)$ über $\hat{F}_2$	463	3.376959e+01	463	5.093895e+02
(8,4)	$f_2(g_2)$ über $\hat{F}_3$	528	4.794558e+01	17	5.120000e+02

Tabelle 7.26: Hier sind die Interpolationsfehler zu den Funktionen f_2, g_2, $(f_2(g_2))$ zu sehen, berechnet mit der FP-Methode.

genaue Gegenteil der Fall. Wir können also daraus schließen, dass beste und genaueste Interpolationsergebnis erhalten wir, wenn wir beide Funktionen, die Innere und die Äußere, zum selben Level und zur selben ϵ-Schranke interpolieren.
Da die hierarchischen Überschüsse des Interpolanten $\hat{F}_2$ näher an den hierarchischen Überschüssen von $\hat{F}_1$ liegen als die hierarchischen Überschüsse von $\hat{F}_3$, ist es klar, warum die Interpolationsfehler, wenn wir unterschiedliche Level für die Interpolation von f_1 und g_1 benutzen, der Interpolation von $f_2(g_2)$ über $\hat{F}_2$ kleiner sind als die Interpolationsfehler von $f_2(g_2)$ über $\hat{F}_3$. Das gilt nämlich, weil bei gleichem Level für die Interpolation von f_2 und g_2 die Interpolation von $f_2(g_2)$ viel schlechter abgeschnitten hat als die Interpolation von F (Tabelle 7.24). Aus diesem Grund muss das auch der Fall bei unterschiedlichen Leveln sein.

Fazit

Wir haben über drei verschiedene Arten gesprochen eine Interpolation von zwei hintereinandergeschalteten Funktionen durchzuführen. Erstens kann man $f \circ g$ direkt interpolieren ($\hat{F}_1$), zweitens zuerst g und dann $f \circ \hat{g}$ interpolieren ($\hat{F}_2$) und drittens f und g für sich interpolieren und dann die Hintereinanderschaltung bilden ($\hat{F}_3$).
In unseren Beispielen hat immer der Interpolant $\hat{F}_1$ am besten abgeschnitten. Aber im letzten Beispiel war zu sehen, dass der Interpolationsfehler des Interpolanten $\hat{F}_3$ dem Interpolationsfehler des Interpolanten $\hat{F}_1$ schon ziemlich nahe kommt. Es besteht also eventuell die Möglichkeit, dass man mit dieser Methode sogar eine Funktion besser interpolieren kann. Um einen Vergleich des Interpolanten $\hat{F}_1$ zum Interpolanten $\hat{F}_3$ zu haben, wenn wir die innnere Funktion auf einem anderen Level als die äußere Funktion interpolieren, ist der Interpolant $\hat{F}_2$ eine gute Möglichkeit. Der Interpolationsfehler von $\hat{F}_2$ ist auch in unseren Beispielen immer besser als der Interpolationsfehler von $\hat{F}_3$.
Also in den meisten Fällen ist die Interpolationsmethode über $\hat{F}_1$ die sinnvollste Methode, aber es scheint auch Funktionen zu geben, die besser mittels einer Hintereinanderschaltung von Funktionen interpoliert werden können. Wenn man mittels $\hat{F}_3$ interpoliert, sollten die innere Funktion und die äußere Funktion mit demselben Level und derselben ϵ-Schranke interpoliert werden, damit der Interpolationsfehler möglichst klein wird.
Die Abschätzung des Interpolationsfehlers zum Interpolanten $\hat{F}_3$ ist in unseren Beispiel immer

eine obere Schranke. Die Genauigkeit der Fehlerabschätzung hängt aber leider von der Genauigkeit der Interpolanten der inneren und der äußeren Funktion ab. Aus diesem Grund wird die Abschätzung des Interpolationsfehlers zum Interpolanten $\hat{F}_3$ umso genauer, desto kleiner die Interpolationsfehler der inneren und der äußeren Funktion sind.

7.4 Funktionsinterpolation durch Hintereinanderschaltung von Funktionen

In Kapitel 5 ist das Verfahren zur Funktionsinterpolation durch Hintereinanderschaltung von Funktionen (FHF-Verfahren) vorgestellt worden. In diesem Verfahren wird die zu interpolierende Funktion nicht direkt interpoliert sondern mittels zwei ineinandergeschachtelter Funktionen. Dabei berechnet sich die innnere Funktion durch den Betrag der Ableitung von der Ausgangsfunktion und die äußere Funktion durch die innere Funktion und die zu interpolierende Funktion (siehe Abschnitt 5.6). In diesem Kapitel werden wir nun das FHF-Verfahren mit dem einfachen adaptiven Dünngitter-Interpolationsverfahren (ADG-Verfahren) vergleichen und die Ergebnisse diskutieren. Wir werden dabei drei verschiedene Typen von Funktionen besprechen:

1. glatte Funktionen
2. Funktionen mit Polstelle
3. Funktionen mit Sprung

Betrachten wir nun folgende Beispielfunktionen:

- **zu 1.**
  - **GLATT1D**
$$G1D(x) = \sin(\pi \cdot x)$$
  - **GLATT2D**
$$G2D(\mathbf{x}) = \begin{pmatrix} \cos(2\pi x_1 - \pi) \cdot \sin(2\pi x_2 - \pi) \\ \sin(2\pi x_1 - \pi) \cdot \sin(2\pi x_2 - \pi) \\ \cos(2\pi x_2 - \pi) \end{pmatrix}$$
- **zu 2.**
  - **POL1D**
$$P1D(x) = \left| x - \frac{1}{3} \right|^{-\frac{1}{3}}$$
  - **POL2D**
$$P2D(\mathbf{x}) = \left| x_1 - \frac{1}{3} \right|^{-\frac{1}{3}} + \left| x_2 - \frac{1}{3} \right|^{-\frac{1}{3}}$$

- **zu 3.**
  - **SPRUNG1D**

$$S1D(x) = \begin{cases} x & \text{, falls } x < \frac{1}{\pi} \\ x+1 & \text{, sonst} \end{cases}$$

  - **SPRUNG2D**

$$S2D(\mathbf{x}) = \begin{cases} x_1 + x_2 & \text{, falls } x_1 < \frac{1}{\pi} \ \wedge \ x_2 < \frac{1}{\pi} \\ x_1 + 1 & \text{, falls } x_1 < \frac{1}{\pi} \ \wedge \ x_2 \geq \frac{1}{\pi} \\ x_2 & \text{, falls } x_1 \geq \frac{1}{\pi} \ \wedge \ x_2 < \frac{1}{\pi} \\ x_1 + x_2 + 1 & \text{, sonst} \end{cases}$$

Die Sprünge und Polstellen befinden sich bewusst auf Parallelen zu den Achsen, da dünne Gitter mit Singularitäten, die nicht auf den Parallelen zu den Achsen liegen, große Schwierigkeiten haben, wie wir in Abschnitt 7.1 gesehen haben. Dieses Problem gilt insbesondere für das FHF-Verfahren, weil die hintereinandergeschalteten Funktionen im FHF-Verfahren auf adaptiven dünnen Gittern interpoliert werden. Somit kommt dieses Problem zweimal zum Tragen, einmal bei der inneren Funktion unbd einmal bei der äußeren Funktion. Es macht deswegen keinen Sinn Funktionen mit Singularitäten, die nicht auf den Parallelen zu den Achsen liegen, zu betrachten. Die Beispielfunktionen sind in den Abbildungen 7.24 bis 7.26 zu sehen.

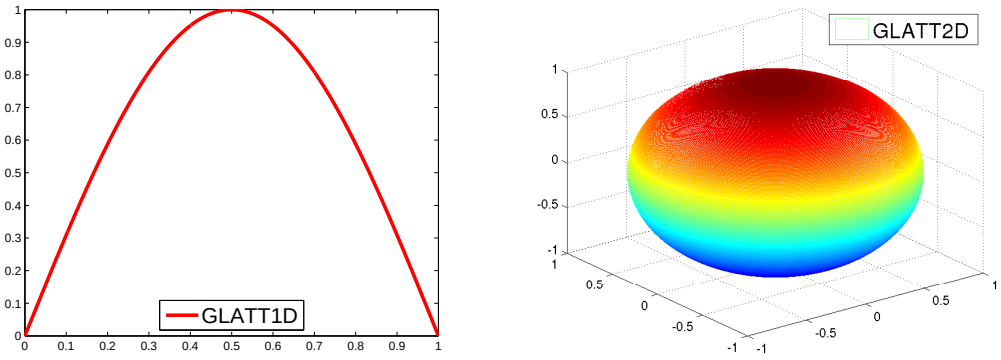


Abb. 7.24: Funktionsplots zu GLATT1D und GLATT2D

Mit beiden Verfahren haben wir bis zum maximalen Level 15 bei eindimensionalen Funktionen und bis zum maximalen Level 10 bei zweidimensionalen Funktionen interpoliert. Die Berechnung des Interpolationsfehlers (siehe dazu Abschnitt 3) haben wir mit der Fehlerberechnungsmethode mit Produktgitter-Punkten (FP) durchgeführt. Wir haben den Interpolationsfehler der eindimensionalen Funktionen mit der Schrittweite $h = 10^{-5}$ und den Interpolationsfehler der zweidimensionalen Funktionen mit der Schrittweite $h = \frac{1}{250}$ berechnet.
An Hand der zweidimensionalen Funktionen soll gezeigt werden, dass unser Algorithmus auch in höheren Dimensionen anwendbar ist. Leider kann man zur Zeit mit dem FHF-Verfahren noch nicht in sehr hohe Dimensionen vorstoßen, da für die Berechnung der inneren Funktion

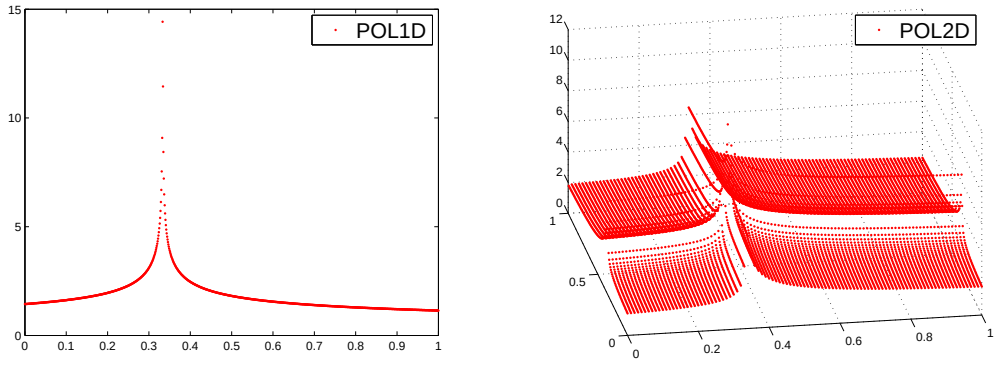


Abb. 7.25: Funktionsplots zu POL1D und POL2D

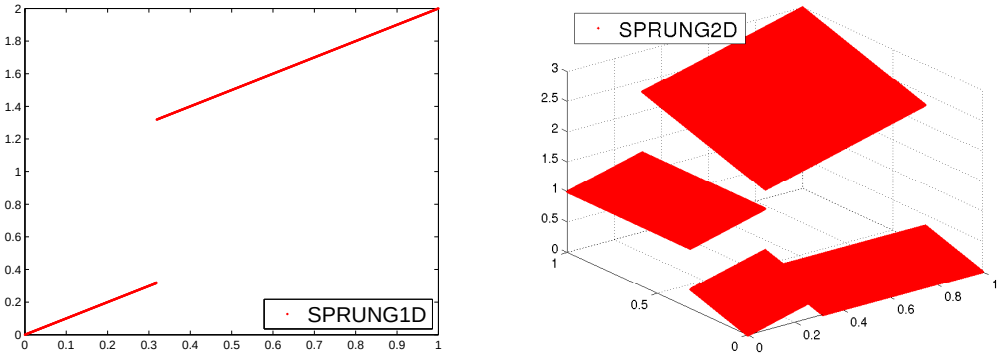


Abb. 7.26: Funktionsplots zu SPRUNG1D und SPRUNG2D

g einmal ein Produktgitter des maximalen Levels gespeichert werden muss und so der Fluch der Dimension hier sehr schnell zuschlägt. Als erstes befassen wir uns mit den glatten Funktionen GLATT1D und GLATT2D. In Abbildung 7.27 sind die interpolierten Funktionen mit dem ADG-Verfahren und in Abbildung 7.28 die interpolierten Funktionen mit FHF-Verfahren zu sehen. Vergleicht man die Graphen der Interpolanten, gewonnen mit dem ADG-Verfahren, mit denen, die mit dem FHF-Verfahren gewonnen wurden, so sind keine optischen Unterschiede zu erkennen. Beide Verfahren interpolieren so gut, dass selbst zur Originalfunktion keine Unterschiede zu erkennen sind.
Betrachten wir deswegen die Ergebnisse, die wir mit der FP berechnet haben und schauen uns die Konvergenzgraphen in den Abbildungen 7.29 und 7.30 an. Beide Verfahren zeigen für die glatten Funktionen dasselbe Konvergenzverhalten an, wobei die Konvergenzrate bei zwei liegt. Der einzige Unterschied ist, dass das FHF-Verfahren mehr Punkte als das ADG-Verfahren benötigt, um dieselbe Genauigkeit bei der Interpolation der glatten Funktionen GLATT1D und GLATT2D zu erhalten. Besonders fällt auf, wenn wir die Unterschiede zwischen den Dimensionen betrachten, dass der Abstand zwischen der Fehlerkurve vom ADG-Verfahren und der Fehlerkurve vom FHF-Verfahren mit der Dimension größer wird. Das heißt, dass das Ergebnis der FHF-Methode im Vergleich zur ADG-Methode schlechter wird und wir wesentlich mehr

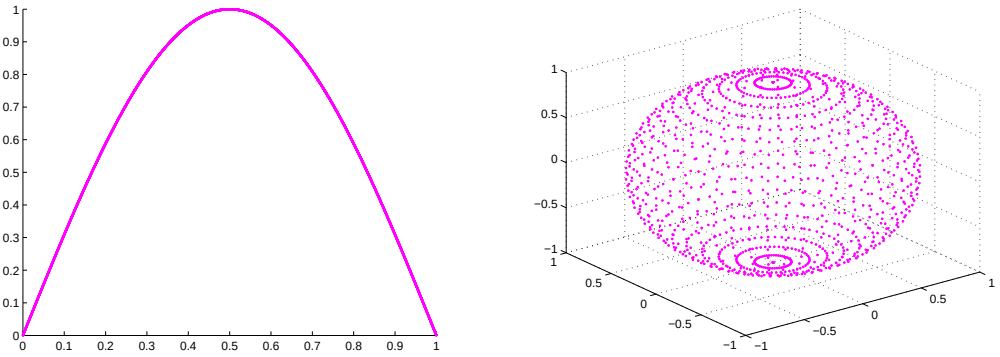


Abb. 7.27: F zu GLATT1D und GLATT2D mit einem ADG bis Level 15, bzw. Level 10

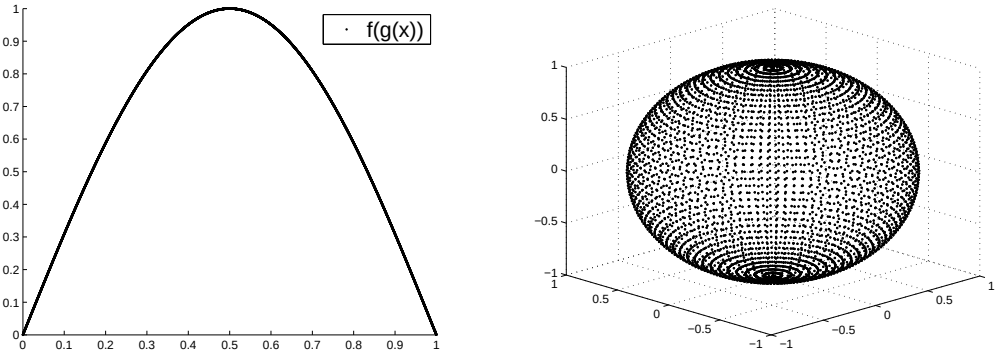


Abb. 7.28: F zu GLATT1D und GLATT2D mit MON bis Level 15, bzw. Level 10

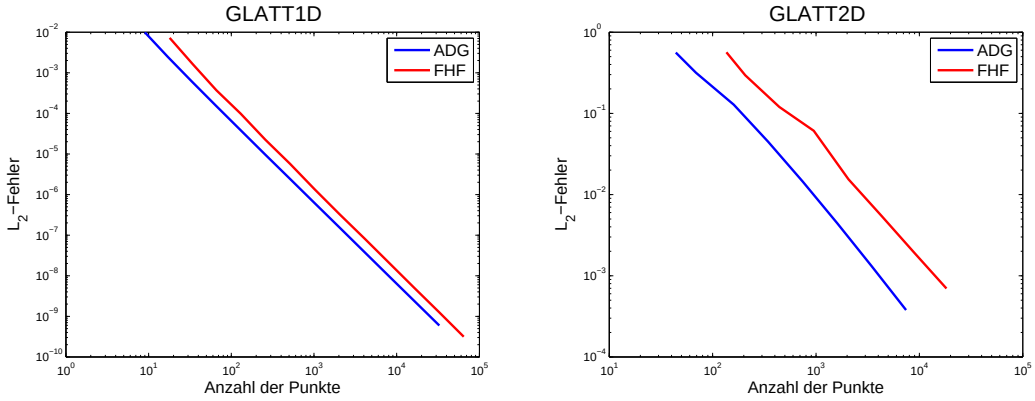


Abb. 7.29: Konvergenzplots zu GLATT1D und GLATT2D mit L_2-Fehler

Gitterpunkte zur Interpolation mit der FHF-Methode benötigen, um die Genauigkeit der Interpolation mit der ADG-Methode zu erreichen, wenn die Dimension größer wird.

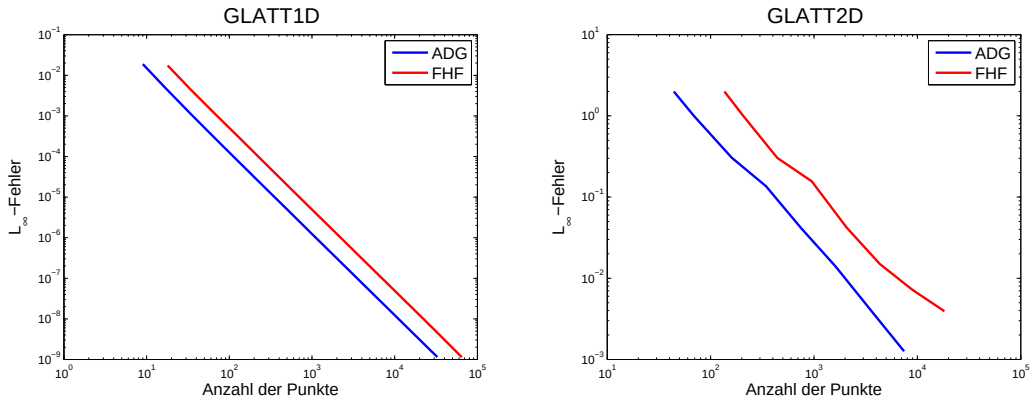


Abb. 7.30: Konvergenzplots zu GLATT1D und GLATT2D mit L_∞-Fehler

Funktion	ADG	FHF
GLATT1D	2.005017e+00	2.037593e+00
GLATT2D	1.422221e+00	1.356441e+00
POL1D	2.366482e-01	1.869139e+00
POL2D	1.488030e+00	2.491130e+00
SPRUNG1D	3.847888e+00	9.141188e+00
SPRUNG2D	1.410381e+00	1.061308e-01

Tabelle 7.27: Konvergenzrate errechnet aus dem L_2-Fehler mit der Konvergenzgraphenmethode (KGM)

Gehen wir nun über zu den Beispielfunktionen POL1D und POL2D. In den Abbildungen 7.31 und 7.32 sind schon optisch Unterschiede zwischen dem Interpolanten zur Funktion POL2D mit dem FHF-Verfahren und dem mit dem ADG-Verfahren zu beobachten. Das FHF-Verfahren interpoliert genauer. Die Interpolanten zur Funktion POL1D, die mit dem FHF-Verfahren und dem ADG-Verfahren ermittelt wurden, sehen identisch aus.
Zu klären bleibt, ob der Aufwand der FHF-Methode geringer ist als der Aufwand der ADG-Methode um eine ähnliche Genauigkeit zu erreichen. Sehen wir uns die Konvergenzgraphen in den Abbildungen 7.33 und 7.34 an, so sehen wir, dass die Genauigkeit des Interpolanten gegenüber der Originalfunktion, gewonnen mit dem FHF-Verfahren, in beiden Fällen höher ist als die Genauigkeit des Interpolanten gegenüber der Originalfunktion, gewonnen mit dem ADG-Verfahren. Hinzu kommt außerdem, dass die Konvergenzrate des FHF-Verfahrens größer ist als die des ADG-Verfahrens. Auf Grund dieser Festellung schauen wir uns die Konvergenzrate und den Interpolationsfehler zu den Funktionen POL1D und POL2D noch genauer an.
Beginnen wir mit der Funktion POL1D und der Betrachtung des Interpolationsfehlers. Man sieht in Tabelle 7.28, dass bei gleichem Level das FHF-Verfahren zwar mehr Gitterpunkte als das ADG-Verfahren benötigt, aber auch eine höhere Genauigkeit aufweist. Diese Aussage gilt für jedes Level und jede der betrachteten Fehlernormen (L_2 und L_∞). Ab dem Level 10 ist ein sehr großer Abfall des Interpolationsfehlers des FHF-Verfahrens zu bemerken. Dieser Abfall des Interpolationsfehlers begründet sich aber nicht auf der guten Interpolation, sondern auf

der Fehlerberechnung, da wir mit der gewählten Anzahl an Punkten den Interpolationsfehler ab dem zehnten Level nicht mehr genau erfassen können. Betrachte zum Abfall des Interpolationsfehlers die Abbildungen 7.33 und 7.34. Abhilfe würde eine kleinere Schrittweite des benutzten Produktgitters bringen, aber auch mehr Zeit bei der Berechnung mit sich bringen. Deswegen haben wir uns gegen eine kleinere Schrittweite entschieden. Zudem würde eine Veränderung der Schrittweite nichts an der Tatsache ändern, dass das FHF-Verfahren die Funktion POL1D besser interpoliert als das ADG-Verfahren.

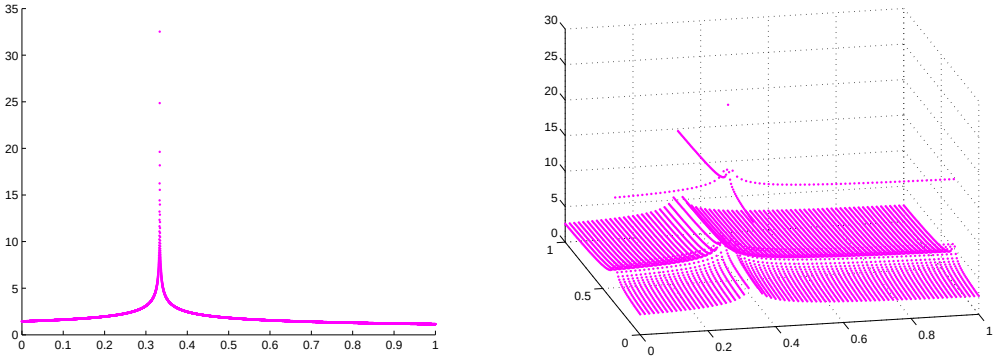


Abb. 7.31: F zu POL1D und POL2D mit einem ADG bis Level 15, bzw. Level 10

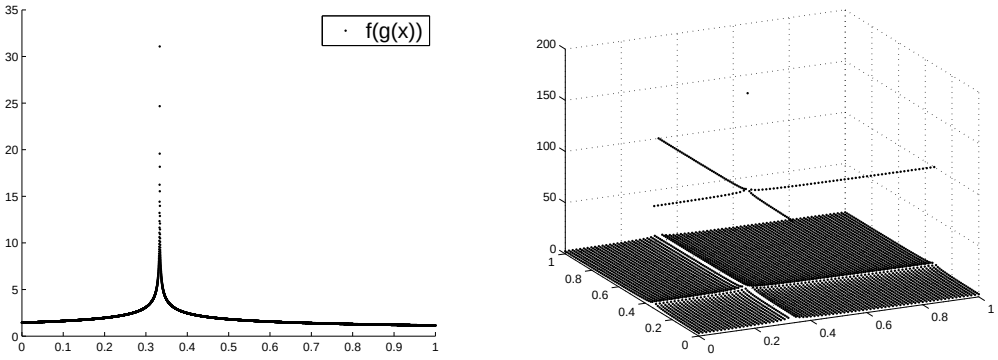


Abb. 7.32: F zu POL1D und POL2D mit MON bis Level 15, bzw. Level 10

Dieser extreme Abfall des Interpolationsfehlers des FHF-Verfahrens ist auch in der Konvergenzrate, wobei die Konvergenzrate mit der Steigungsmethode berechnet wurde (Tabelle 7.29), zu erkennen. Ab dem zehnten Level schnellt die Konvergenzrate des FHF-Verfahrens von knapp eins auf vier hoch. Betrachtet man aber nur die Konvergenzraten vor dem zehnten Level, ist trotzdem eindeutig zu sehen, dass die Konvergenzrate des FHF-Verfahrens (hier bei ca. 0.5) größer ist als die Konvergenzrate des ADG-Verfahrens (hier bei ca. 0.2) zur Funktion POL1D.

Diskutieren wir nun die Ergebnisse zur Interpolation der Funktion POL2D. Betrachten wir

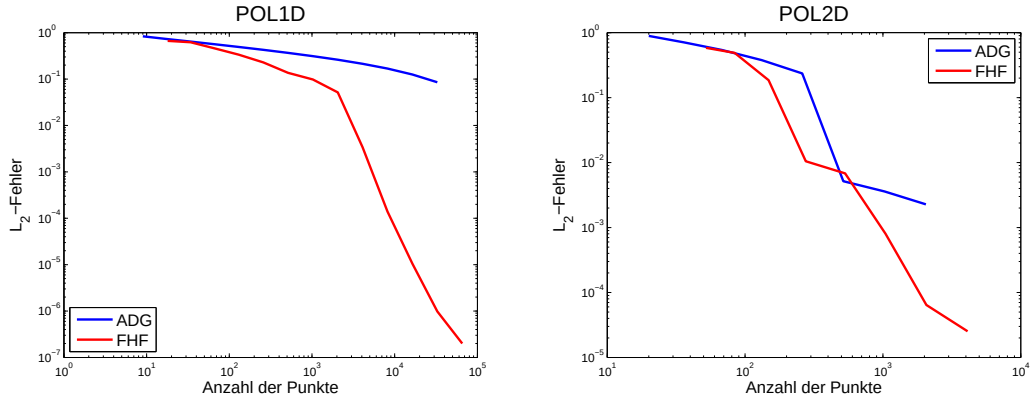


Abb. 7.33: Konvergenzplots zu POL1D und POL2D mit L_2-Fehler

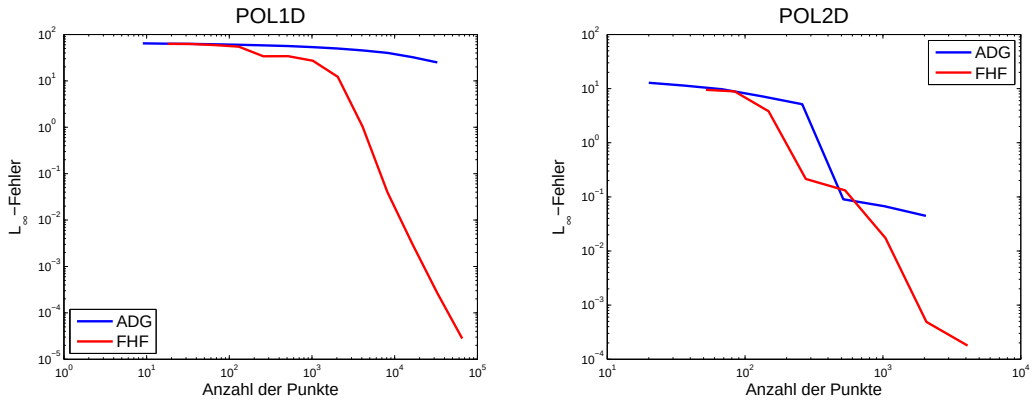


Abb. 7.34: Konvergenzplots zu POL1D und POL2D mit L_∞-Fehler

die berechneten Interpolationsfehler (Tabelle 7.30), so sehen wir, dass die FHF-Methode zum gleichen Level wie die ADG-Methode zwar doppelt soviele Gitterpunkte benötigt, aber auch doppelt so genau ist wie die ADG-Methode. Vergleichen wir aber den Fehler, welcher zum Level L bei der Interpolation mit dem ADG-Verfahren gemacht wird, mit dem Fehler, der zum Level $L-1$ bei der Interpolation mit dem FHF-Verfahren gemacht wird, so zeigt sich, dass beide Verfahren ungefähr die gleiche Anzahl an Gitterpunkten brauchen, aber das FHF-Verfahren die Funktion POL2D genauer interpoliert als das ADG-Verfahren.
Der Verlauf der Konvergenzrate ist bei beiden Verfahren schwer zu interpretieren (Tabelle 7.31), da die Konvergenzraten zwischen großen und kleinen Werten hin und her springen. In den Konvergenzgraphen (Abbildungen 7.33 und 7.34) ist dies durch die treppenförmige Gestalt der Fehlerkurve zu beobachten. Betrachten wir aber den gemittelten Wert in Tabelle 7.27, so fällt auf, dass das FHF-Verfahren auch hier bei der zweidimensionalen Funktion POL2D durch seine bessere Konvergenzrate und seine höhere Genauigkeit als das ADG-Verfahren überzeugt.

max.	ADG			FHF		
Level	Punkte	L_2-Fehler	L_∞-Fehler	Punkte	L_2-Fehler	L_∞-Fehler
3	9	8.339456e-01	6.425717e+01	18	6.657734e-01	6.264796e+01
4	17	7.353547e-01	6.355893e+01	34	6.202929e-01	6.235541e+01
5	33	6.465849e-01	6.267941e+01	66	4.596508e-01	5.928504e+01
6	65	5.664529e-01	6.157076e+01	130	3.359946e-01	5.458674e+01
7	129	4.938879e-01	6.017528e+01	258	2.282476e-01	3.398929e+01
8	257	4.279191e-01	5.841371e+01	514	1.360720e-01	3.403221e+01
9	513	3.676660e-01	5.620279e+01	1026	9.792859e-02	2.718616e+01
10	1025	3.123160e-01	5.339575e+01	2050	5.161632e-02	1.220919e+01
11	2049	2.611439e-01	4.991315e+01	4098	3.382261e-03	1.032093e+00
12	4097	2.134075e-01	4.538916e+01	8194	1.394021e-04	3.947898e-02
13	8193	1.686202e-01	4.003246e+01	16386	1.050111e-05	3.044188e-03
14	16385	1.256434e-01	3.241871e+01	32770	9.809212e-07	2.661103e-04
15	32769	8.546571e-02	2.500498e+01	65538	2.002658e-07	2.793870e-05

Tabelle 7.28: Interpolationsfehler und benötigte Gitterpunkte zu POL1D

max. Level	ADG	FHF
3		
4	1.978261e-01	1.112560e-01
5	1.939538e-01	4.518730e-01
6	1.951830e-01	4.622826e-01
7	2.000005e-01	5.641230e-01
8	2.080110e-01	7.504339e-01
9	2.195592e-01	4.759053e-01
10	2.357209e-01	9.252044e-01
11	2.583433e-01	3.934537e+00
12	2.913367e-01	4.602284e+00
13	3.398934e-01	3.731296e+00
14	4.244799e-01	3.420562e+00
15	5.559418e-01	2.292322e+00

Tabelle 7.29: Konvergenzrate zu POL1D errechnet aus dem L_2-Fehler mit der Steigungsmethode (SM)

max. Level	ADG	FHF
3		
4	3.884034e-01	3.724871e-01
5	4.249249e-01	1.704913e+00
6	5.398284e-01	4.613604e+00
7	6.981011e-01	6.579397e-01
8	5.571152e+00	3.169332e+00
9	5.271297e-01	3.697724e+00
10	6.650275e-01	1.355133e+00

Tabelle 7.31: Konvergenzrate zu POL2D errechnet aus dem L_2-Fehler mit der Steigungsmethode (SM)

max.	ADG			FHF		
Level	Punkte	L_2-Fehler	L_∞-Fehler	Punkte	L_2-Fehler	L_∞-Fehler
3	20	8.907668e-01	1.281162e+01	52	5.840593e-01	9.517498e+00
4	36	7.089495e-01	1.137056e+01	84	4.885138e-01	8.892775e+00
5	68	5.410636e-01	9.723843e+00	148	1.859937e-01	3.818927e+00
6	132	3.782184e-01	7.223519e+00	276	1.049152e-02	2.140842e-01
7	260	2.356254e-01	5.145721e+00	532	6.812773e-03	1.320229e-01
8	516	5.173947e-03	9.000250e-02	1044	8.042270e-04	1.732683e-02
9	1028	3.597735e-03	6.709387e-02	2068	6.422575e-05	4.892742e-04
10	2052	2.271948e-03	4.448436e-02	4116	2.527122e-05	1.785892e-04

Tabelle 7.30: Interpolationsfehler und benötigte Gitterpunkte zu POL2D

Nun werden wir noch die letzten beiden Beispielfunktionen SPRUNG1D und SPRUNG2D besprechen. Bis jetzt haben wir gesehen, dass das FHF-Verfahren zwar das ADG-Verfahren bei den glatten Funktionen nicht übertreffen kann, aber Funktionen mit Singularitäten besser interpoliert. Wir haben als erste Singularität Polstellen betrachtet. Sind unsere dort gewonnenen Ergebnisse auch auf andere Singularitäten zu übertragen? Wie wir gleich sehen werden, müssen wir das leider mit nein beantworten, denn bei den Sprungfunktionen verschlechtert sich nur die Interpolation durch die Hintereinanderschaltung. Vielleicht kann man aber durch eine bessere Wahl der inneren Funktion und der äußeren Funktion die Interpolation mit der FHF-Methode verbessern, aber wir haben noch keine geeigneten Funktionen gefunden.

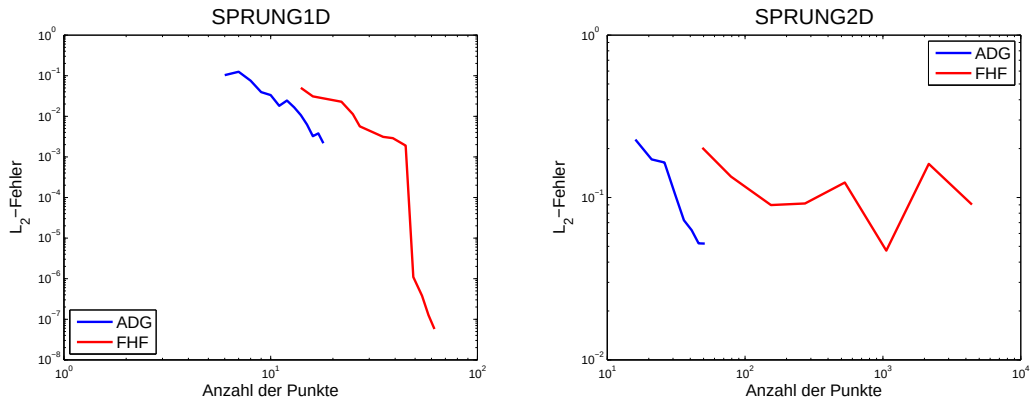


Abb. 7.35: Konvergenzplots zu SPRUNG1D und SPRUNG2D mit L_2-Fehler

In den Abbildungen 7.37 und 7.38 sind die interpolierten Funktionen SPRUNG1D und SPRUNG2D zu einem gegebenen Maximallevel zu sehen. Der Interpolant der Funktion SPRUNG1D, der mit dem FHF-Verfahren konstruiert wurde, ist optisch etwas genauer als der Interpolant, der mit dem ADG-verfahren berechnet wurde. Bei der Funktion SPRUNG2D ist es genau anders herum, da ist die Genauigkeit des Interpolanten berechnet mit dem ADG-Verfahren eindeutig höher als die Genauigkeit des Interpolanten berechnet mit dem FHF-Verfahren. Betrachten wir nun die Konvergenzgraphen in den Abbildungen 7.35 und 7.36. Was man in allen vier Graphen sehen kann, ist, dass das FHF-Verfahren sehr viel mehr Gitterpunkte

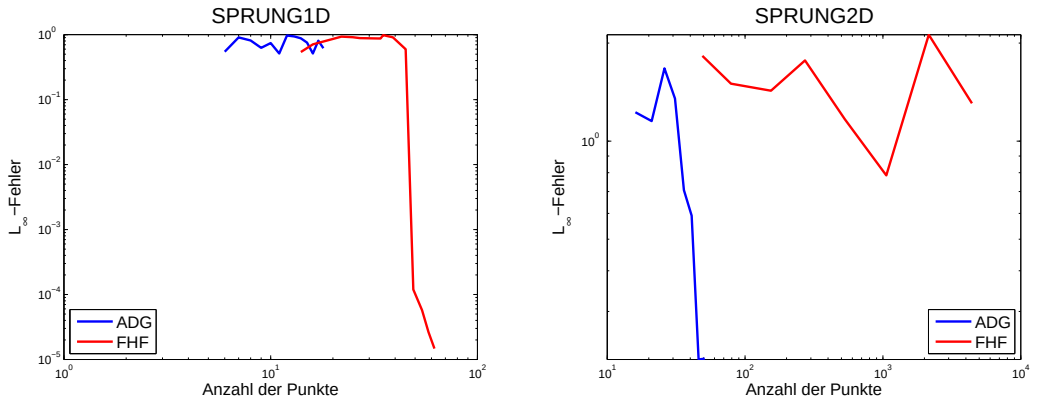


Abb. 7.36: Konvergenzplots zu SPRUNG1D und SPRUNG2D mit L_∞-Fehler

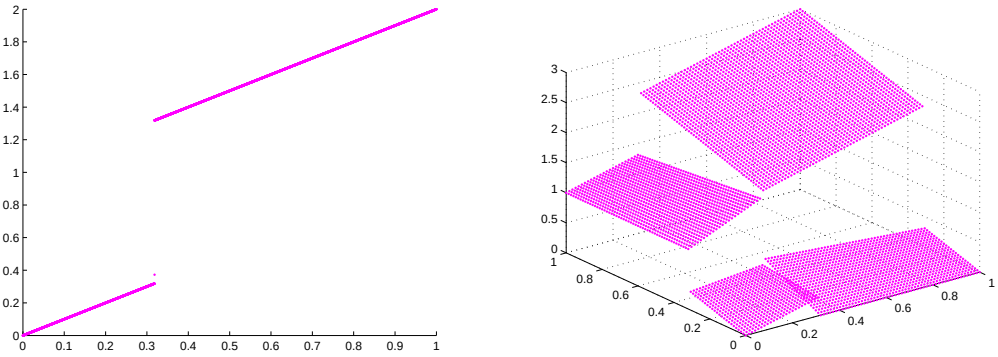


Abb. 7.37: F zu SPRUNG1D und SPRUNG2D mit einem ADG bis Level 15, bzw. Level 10

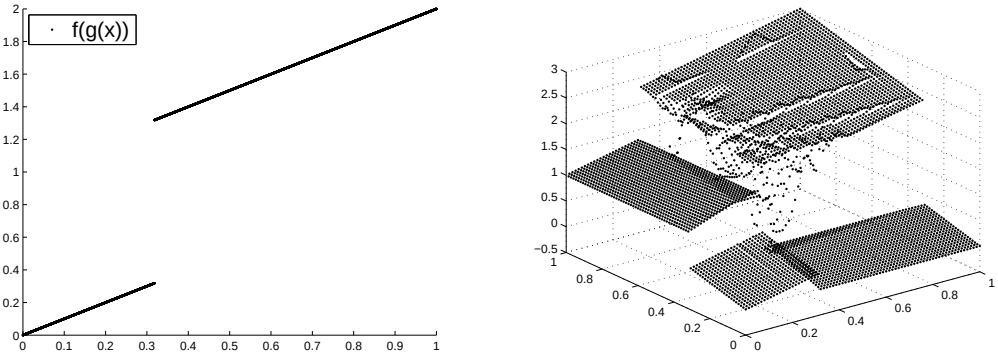


Abb. 7.38: F zu SPRUNG1D und SPRUNG2D mit MON bis Level 15, bzw. Level 10

benötigt als das ADG-Verfahren, um dieselbe Genauigkeit des Interpolanten zu erhalten. Außerdem oszilliert die L_2-Fehlerkurve des Interpolanten zur Funktion SPRUNG2D, berechnet mit der FHF-Methode, zum Ende hin sehr stark. Das FHF-Verfahren ist also speziell in zwei

Dimension für Sprungfunktionen nicht stabil genug.

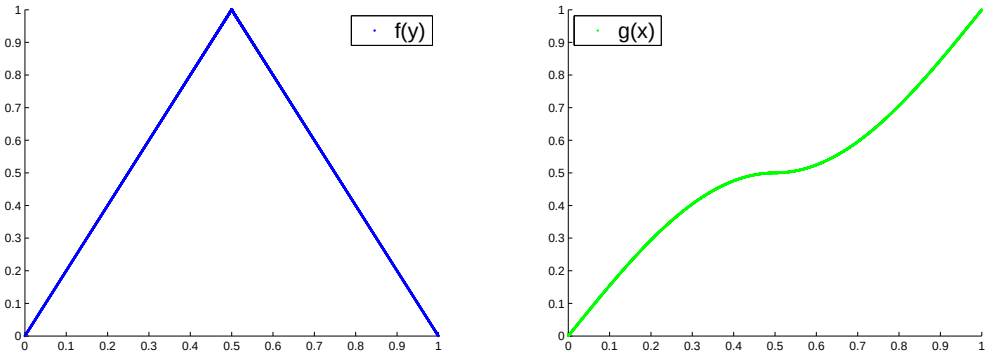


Abb. 7.39: f und g zu GLATT1D mit FHF mit Level 15

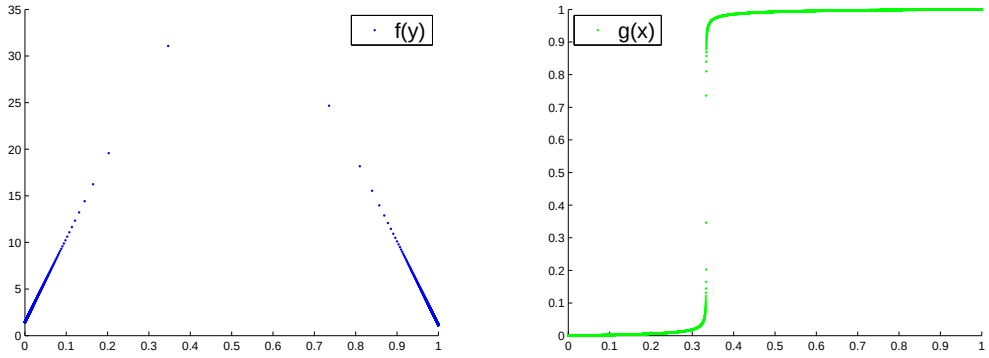


Abb. 7.40: f und g zu POL1D mit FHF mit Level 15

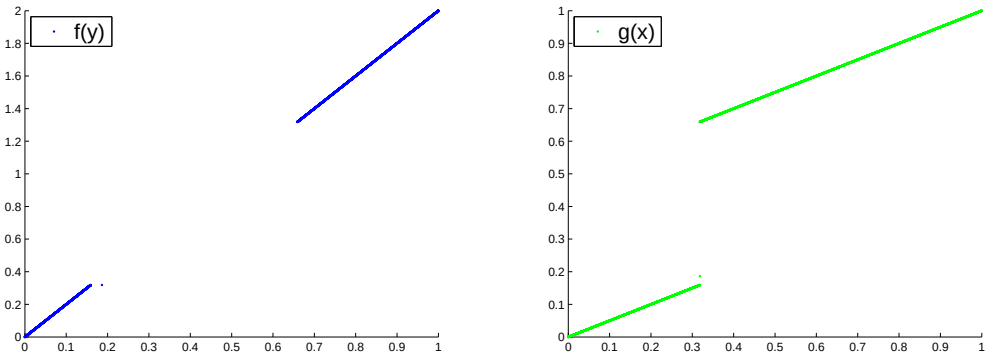


Abb. 7.41: f und g zu SPRUNG1D mit FHF mit Level 15

Zum Schluss werden wir uns noch die innere Funktion g und die äußere Funktion f, die wir während des FHF-Verfahrens aus der Originalfunktion F berechnen für die eindimensionalen Funktion GLATT1D, POL1D und SPRUNG1D anschauen (siehe Abbildungen 7.39 bis 7.41) und an ihnen erläutern warum nur die Funktion mit Polstelle besser interpoliert werden kann. Wie erwartet ist die äußere Funktion in jedem Fall eine lineare Funktion. Bei den Funktionen GLATT1D und POL1D entspricht sie einer Hutfunktion und bei der Funktion SPRUNG1D der Geraden $f(x) = 2 \cdot x$. Die innere Funktion steht für die Verteilung der Gitterpunkte auf dem Intervall [0,1] durch die Originalfunktion.
Betrachten wir nun die Funktionen nacheinander und beginnen mit der funktion POL1D. Durch die innere Funktion werden die Punkte von der Polstelle nach links und rechts weg verschoben und durch die äußere Funktion nach oben verschoben. In Abbildung 7.40 ist das gut zu erkennen. Die Polstelle wird also verbreitert, so dass sie einfacher auszuwerten ist. Die Folge ist, dass der Interpolationsfehler, den wir zur Originalfunktion machen, geringer ausfällt, als wenn wir die Funktion POL1D direkt interpoliert hätten.
Bei glatten Funktionen hilft uns diese Technik aber nicht weiter, da keine Singularitäten, also Problemstellen, vorhanden sind. Wir erhalten zu einer glatten Funktion mit dem FHF-Verfahren eine glatte steigende innere Funktion und eine lineare äußere Funktion. Siehe dazu Abbildung 7.39. Wir brauchen dabei für die innere Funktion zumeist genauso viele Punkte wie das ADG-Verfahren für die komplette Originalfunktion, aber zusätzliche benötigen wir noch mindestens einen Punkt für die äußere Funktion. Das heißt aber, dass wir insgesamt immer mehr Punkte brauchen mit dem FHF-Verfahren, um die Originalfunktion genauso gut zu interpolieren wie das ADG-Verfahren.
Die Sprungfunktion ist unser Problemkind. Der Interpolationsfehler, der an der Sprungstelle von der inneren Funktion gemacht wird, wird durch die äußere Funktion verstärkt. Die Hoffnung, dass wie bei den Polfunktionen so der Maximalfehler minimiert wird, ist damit erloschen.
In zwei Dimensionen sind die Argumente für alle Verfahren die gleichen, weswegen wir darauf nicht genauer eingehen werden.

Fazit

Wir haben gesehen, dass mit der FHF-Methode, d.h mit der Variante für die Ausgangsfunktion eine hintereinandergeschaltete Funktion zu wählen, eine Funktion mit Polstellen besser als mit dem ADG-Verfahren interpoliert werden kann. Nicht nur die Genauigkeit ist höher, sondern auch die Gitterpunktanzahl geringer und die Konvergenzrate größer.
Bei glatten Funktionen hat das ADG-Verfahren den Vorteil nur eine Funktion auf einem adaptiven dünnen Gitter interpolieren zu müssen. In diesem Fall kann unser FHF-Verfahren in der Genauigkeit zwar mithalten, aber die dafür benötigte Gitterpunktanzahl ist immer höher als die benötigte Gitterpunktanzahl für die Interpolation mit dem ADG-Verfahren.
Bei allen übrigen Funktionen mit Singularitäten sollte das FHF-Verfahren nicht angewendet werden, da durch die bestehende Auswahl der inneren Funktion die Interpolanten, berechnet mit dem FHF-Verfahren, gegenüber den Interpolanten, berechnet mit dem ADG-Verfahren, sogar an Genauigkeit verlieren.
Die Hoffnung besteht aber auch solche singulären Funktionen mit diesem FHF-Verfahren besser interpolieren zu können, wenn man eine andere Wahl der inneren und äußeren Funktion trifft. Bei den glatten Funktionen braucht man sich aber keine großen Hoffnungen zu machen, da es

hier keine Problemstellen gibt, die man verbessern kann.

8 Fazit und Ausblick

Ein Ziel dieser Arbeit war, das Verhältnis von Aufwand zu Genauigkeit bei der Interpolation von Funktionen zu verbessern, so dass hochdimensionale und vektorwertige Funktionen besser interpoliert werden können. Mit regulären Dünngitter-Interpolationsverfahren kann der Aufwand gegenüber Produktgitter-Interpolationsverfahren schon gesenkt werden bei annähernd gleichem Interpolationsfehler. Wir haben gezeigt, dass man mit dem adaptiven Dünngitter-Interpolationsverfahren den Aufwand bei gleichbleibender Genauigkeit noch weiter reduzieren kann. Aus diesem Grund können hochdimensionale Funktionen mit geringem Speicheraufwand interpoliert werden. Dies gilt insbesondere für singuläre Funktionen.
Durch die Aufteilung einer vektorwertigen Funktion $F : \bar{\Omega} \to \mathbb{R}^m$ in m einfache Teilfunktionen $F_j : \bar{\Omega} \to \mathbb{R}$ mit $j = 1, \ldots, m$ ist es uns gelungen, die Interpolation einer solchen Funktion effizienter durchzuführen. Da zu jeder Teilfunktion ein eigener Interpolant konstruiert wird, der von den Interpolanten der anderen Teilfunktionen nicht abhängt, kann man jede Teilfunktion auf adaptiven dünnen Gittern optimal interpolieren. Desweiteren könnte eine Parallelisierung der Interpolation der Teilfunktionen helfen, die Interpolation vektorwertiger Funktionen schneller zu gestalten. Dies bleibt aber anderen vorbehalten.
Auf Grund der Feststellung, die wir im Kapitel 4 gemacht haben, dass der Interpolationsfehler einer Hintereinanderschaltung von zwei Dünngitter-Interpolanten durch die einzelnen Interplationsfehler nach oben abgeschätzt werden kann, haben wir ein Verfahren mit Hilfe der adaptiven dünnen Gitter konstruiert, das uns erlaubt Funktionen mit Polstellen mit einem besseren Verhältnis von Aufwand zu Genauigkeit als mit einem adaptiven Dünngitter-Interpolationsverfahren zu interpolieren. Dabei lassen sich die benutzten Funktionen zur Hintereinanderschaltung aus der zu interpolierenden Funktion berechnen. Wir haben dazu die innere Funktion mit dem Integral des Betrags der Ableitung der zu interpolierenden Funktion gleichgesetzt. Eine Frage die sich an dieser Stelle ergibt ist die Folgende. Könnten wir eventuell durch eine andere Wahl der inneren Funktion auch andere Funktionstypen mit diesem Verfahren besser als mit einem adaptiven Dünngitter-Verfahhren interpolieren? Die Antwort dieser interessanten Frage bleibt offen für weitere Untersuchungen. Dabei kann die vorliegende Arbeit als Grundlage für weitere Forschungen genutzt werden. Weiter bliebe in Zukunft zu klären, ob man durch eine geschickte Wahl der äußeren Funktion eine Verbesserung des Verfahrens erreichen könnte. Ein ernsthaftes Problem bei der Funktionsinterpolation durch Hintereinanderschaltung ist noch, dass wir zur Zeit innerhalb des Verfahrens ein komplettes Produktgitter zur Bestimmung der inneren Funktion aufstellen müssen und so nur Funktionen bis zur Dimension zwei interpolieren können. Die Idee die Hintereinanderschaltung zur Verbesserung der Interpolation zu benutzen, konnte leider im Rahmen dieser Arbeit nicht weiter untersucht werden und sollte, dem vielversprechenden ersten Ergebnis nach zu urteilen, in anderen Arbeit eingehender betrachtet werden.
Es ist festzuhalten, dass in dieser Arbeit eine Vielzahl von Algorithmen zur Interpolation auf dünnen Gittern erläutert wurden und die Dünngitter-Algebra aus [Sch98] durch die Hintereinanderschaltung erweitert wurde. Desweiteren wurde die Interpolationsqualität auf adaptiven

dünnen und regulären dünnen Gittern an Hand einiger unterschiedlicher Funktionstypen genau untersucht und so ein Beitrag zur besseren Interpolation von hochdimensionalen Funktionen geleistet.

9 Anhang

9.1 Wichtige Räume und Normen

Definition 9.1 *Für $1 \leq p < \infty$ ist die p-Norm auf $\mathbb{R}^n$ gegeben durch*

$$|\mathbf{x}| := \begin{cases} (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}} & \text{für } 1 \leq p < \infty, \\ \max_{i=1,\ldots,n} |x_i| & \text{für } p = \infty, \end{cases} \tag{9.1}$$

wenn $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$.

Definition 9.2 *Sei $\Omega \subseteq \mathbb{R}^n$ offen, $1 \leq k < \infty$ und $m \in \mathbb{N}_0$.*

$C^m(\Omega) = \{u : \ u : \Omega \to \mathbb{R}$ *und u ist m-mal stetig differenzierbar in $\Omega\}$*
ist Ω zusätzlich noch beschränkt:
$C^m(\bar{\Omega}) = \{u : \ u \in C^m(\Omega)$ *und $D^{\boldsymbol{\alpha}} u$ kann stetig auf $\bar{\Omega}$ fortgesetzt werden $\forall\, |\boldsymbol{\alpha}|_k \leq m\}$*

Definition 9.3 *Sei $\Omega \in \mathbb{R}^n$ offen, $m \in \mathbb{N}_0$, $k \in \{1, \infty\}$ und $1 \leq p \leq \infty$..*

$$W^{m,p}(\Omega) = \{u \in L^p(\Omega) : \ D^{\boldsymbol{\alpha}} u \in L^p(\Omega) \text{ existiert } \forall\, |\boldsymbol{\alpha}|_k \leq m\} \tag{9.2}$$

definiert dann den Banach-Raum der Funktionen mit schwachen partiellen Ableitungen der Ordnung $\leq m$ in $L^p(\Omega)$, auch Sobolev-Raum genannt, und

$$\|u\|_{m,p} = \left(\sum_{|\boldsymbol{\alpha}|_k \leq m} \int_\Omega |D^{\boldsymbol{\alpha}} u|_k^p \, dx \right)^{\frac{1}{p}}$$
$$bzw. \ \|u\|_{m,\infty} = \max_{|\boldsymbol{\alpha}|_k \leq m} \|D^{\boldsymbol{\alpha}} u\|_\infty$$

die zugehörige Norm.
Ist Ω zusätzlich noch beschränkt mit stückweise glattem Rand, dann erhalten wir den Sobolev-Raum

$$W^{m,p}(\bar{\Omega}) = \{u : \ u \in W^{m.p}(\Omega) \text{ und } \partial\Omega \in C^1\}. \tag{9.3}$$

Der Raum $W_0^{m,p}(\Omega)$ ist der Abschluss von $C_0^\infty(\Omega)$ in $W^{m,p}(\Omega)$

$$W_0^{m,p}(\Omega) = \{u : \ u \in W^{m,p} \text{ mit } u = 0 \text{ auf } \partial\Omega\} \tag{9.4}$$

Außerdem betrachtet man noch folgenden Spezialfall:
für $p = 2$ wird er mit $H^m(\Omega) = W^{m,2}(\Omega)$ definiert, da er dann ein Hilbertraum ist.

9.2 Binomialkoeffizienten

Definition 9.4 *Der verallgemeinerte Binnomialkoeffizient ist definiert durch*

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}$$

mit $\alpha \in \mathbb{R}$, $k \in \mathbb{N}$ *und*

$$\binom{\alpha}{0} = 1$$

.

Bemerkung *Ist* $\alpha \in \mathbb{N}$ *und* $\alpha > k$ *so gilt:*

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-\alpha)\cdots(\alpha-k+1)}{k!} = 0.$$

Satz 9.1 *Additionstheoreme für verallgemeinerte Binomialkoeffizienten.*
Seien $\alpha, \beta \in \mathbb{R}$ *und* $k \in \mathbb{N}$. *Dann gelten die folgenden Gleichungen:*

1. $\binom{\alpha}{k} + \binom{\alpha}{k+1} = \binom{\alpha+1}{k+1}$
2. $\sum_{i=0}^{k} \binom{\alpha+i}{i} = \binom{\alpha+k+1}{k}$
3. $\sum_{i=0}^{k} \binom{\alpha}{i}\binom{\beta}{k-i} = \binom{\alpha+\beta}{k}$.

Beweis:

1.

$$\begin{aligned}\binom{\alpha}{k} + \binom{\alpha}{k+1} &= \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!} + \frac{\alpha(\alpha-1)\cdots(\alpha-k)}{(k+1)!} \\ &= \frac{(k+1)\alpha(\alpha-1)\cdots(\alpha-k+1) + \alpha(\alpha-1)\cdots(\alpha-k)}{(k+1)!} \\ &= \frac{(\alpha-k+k+1)\alpha(\alpha-1)\cdots(\alpha-k+1)}{(k+1)!} \\ &= \frac{(\alpha+1)\alpha\cdots(\alpha+1-k)}{(k+1)!} = \binom{\alpha+1}{k+1}\end{aligned}$$

2. Beweis per Induktion
   IA:

$$\begin{aligned} n=0: &\quad \binom{\alpha}{0} = 1 = \binom{\alpha+1}{0} \\ n=1: &\quad \binom{\alpha}{0} + \binom{\alpha+1}{1} = 1+\alpha+1 = \alpha+2 = \binom{\alpha+2}{1}\end{aligned}$$

IS:

$$
\begin{aligned}
n \to n+1: \quad \sum_{k=0}^{n+1} &\overset{IV}{=} \binom{\alpha+n+1}{n} + \binom{\alpha+n+1}{n+1} \\
&= \frac{(\alpha+n+1)\cdots(\alpha+2)}{n!} + \frac{(\alpha+n+1)\cdots(\alpha+1)}{(n+1)!} \\
&= \frac{(\alpha+n+1)\cdots(\alpha+2)\cdot(\alpha+1+n+1)}{(n+1)!} \\
&= \frac{(\alpha+n+2)\cdots(\alpha+2)}{(n+1)!} = \binom{\alpha+n+2}{n+1}
\end{aligned}
$$

3. Beweis per Induktion
IA:

$$
\begin{aligned}
n=0 \quad & \binom{\alpha}{0}\cdot\binom{\beta}{0} = 1\cdot 1 = \binom{\alpha+\beta}{0} \\
n=1 \quad & \binom{\alpha}{0}\cdot\binom{\beta}{1} + \binom{\alpha}{1}\cdot\binom{\beta}{0} = \binom{\alpha}{1} + \binom{\beta}{1} = \alpha+\beta = \binom{\alpha+\beta}{1}
\end{aligned}
$$

IS:

$$
\begin{aligned}
& n \to n+1: \\
\sum_{k=0}^{n+1} \binom{\alpha}{k}\binom{\beta}{n+1-k} &= \frac{1}{n+1}\cdot\left[\sum_{k=0}^{n+1}(n+1-k+k)\binom{\alpha}{k}\binom{\beta}{n+1-k}\right] \\
&= \frac{1}{n+1}\cdot\left[\sum_{k=0}^{n+1}(n+1-k)\binom{\alpha}{k}\binom{\beta}{n+1-k} + \sum_{k=0}^{n+1}k\binom{\alpha}{k}\binom{\beta}{n+1-k}\right] \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n}\beta\binom{\alpha}{k}\binom{\beta-1}{n-k} + \sum_{k=1}^{n+1}\alpha\binom{\alpha-1}{k-1}\binom{\beta}{n+1-k}\right] \\
&\overset{IV}{=} \frac{1}{n+1}\left[\beta\cdot\binom{\alpha+\beta-1}{n} + \alpha\cdot\binom{\alpha+\beta-1}{n}\right] \\
&= \frac{\alpha+\beta}{n+1}\cdot\binom{\alpha+\beta-1}{n} = \binom{\alpha+\beta+1}{n+1}
\end{aligned}
$$

q.e.d.

10 Literaturverzeichnis

[Alt] Alt H. W.: *Lineare Funktionalanalysis*, Springer, Berlin, 2002

[Ba94] Balder R.: *Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern*, Dissertation, Institut für Informatik, Technische Universität München, Okt. 1994

[Bun92] Bungartz H.-J.: *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*, Dissertation, Institut für Informatik, Technische Universität München, 1992

[BuDo97] Bungartz H.-J., Dornseifer T.: *Sparse Grids: Recent Developments For Elliptic Partial Differential Equations*, TUM-I9702, SFB-Bericht Nr. 342/02/97 A, Institut für Informatik, Technische Universität München, Februar 1997

[BuGri97] Bungartz H.-J., Griebel M.: *A note on the complexity of solving Poisson's equation for spaces of bounded mixed derivatives*, J. Complexity, 15:167-199, 1999 (auch als SFB-Report 256, Nummer 524, Universität Bonn, 1997)

[BuGri04] Bungartz H.-J., Griebel M.: *Sparse grids*, Acta Numerica, 13:1-123, 2004

[Feuer05] Feuersänger C.: *Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen*, Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, Mai 2005

[Ga04] Garcke J.: *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern*, Dissertation, Institut für Numerische Simulation, Universität Bonn, 2004

[Ga06] Garcke J.: *Sparse Grid Tutorial*, August 2006

[GaGriTh00] Garcke J., Griebel M., Thess M.: *Data Mining with Sparse Grids*, SFB-Report 256, Nummer 675, Universität Bonn, Juli 2000

[GeGri03] Gerstner T., Griebel M.: *Dimension-Adaptive Tensor-Product Quadrature*, Computing, 71(1):65-87, 2003

[Gra02] Grădinaru V. C.: *Whitney Elements om Sparse Grids*, Dissertation, Mathematische Fakultät, Eberhard-Karls-Universität Tübingen, April 2002

[Gri97] Griebel M.: *Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences*,Computing, 61(2):151-179, 1998. also as Proceedings Large-Scale Scientific Computations of Engineering and Environmental Problems, 7. June - 11. June, 1997, Varna, Bulgaria, Notes on Numerical Fluid Mechanics 62, Vieweg-Verlag, Braunschweig, M. Griebel, O. Iliev, S. Margenov and P. Vassilevski (editors)

[GriOVas05] Griebel M., Oeltz D., Vassilevski P.: *Space-Time Approximation with Sparse Grids*, SIAM Journal on Scientific Computing, 28(2):701-727, 2005

[Ha06] Hahnen P.: *Nichtlineare numerische Verfahren zur multivariaten Dichteschätzung*, Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, November 2006

[Hau03] Hauser H.: *The Hironaka Theorem on Resolution of Singularities*, Bulletin (New series) of the American Mathematical Society, Volume 40, Number3, Pages 323-403, Mai 2003

[Hau04] Hauser H.: *Berkeley Lectures on Blowups and Resolution of Singularities*, Mai 2004

[Hau05] Hauser H.: *Singular Mobiles*, 16. März 2005

[HeSp99] Hemker P.W., Sprengel F.: *On the Representation of Functions and Finite Difference Operators on Adaptive Sparse Grids*, CWI Report MAS-R9933, ISSN 1386-3703, November 1999

[Kegl99] Kegl B.: *Principal Curves: Learning, Design and Applications*, Dissertation, The Department of Computer Science, Concordia University, Montreal, Quebec, Canada, Dez. 1999

[KeKrLiZe98] Kegl B., Krzyzak A., Linder T., Zeger K.: *A Polygonal Line Algorithm for Constructing Principal Curves*, Neural Information Processing Systems, 1998

[KeKrLiZe00] Kegl B., Krzyzak A., Linder T., Zeger K.: *Learning and Design of Principal Curves*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 3, pp. 281-297, März 2000

[Ko01] Koster F.: *Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern*, Dissertation, Institut für Numerische Simulation, Universität Bonn, Nov. 2001

[Kranz02] Kranz C. J.: *Untersuchungen zur Kombinationstechnik bei der numerischen Strömungssimulation auf versetzten Gittern*, Dissertation, Institut für Informatik, Technische Universität München, 2002

[Sch98] Schiekofer T.: *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen*, Dissertation, Institut für Numerische Simulation, Universität Bonn, Dez. 1998

[Thu94] Thurner V.: *Eine ganzzahlige Arithmetik zur adaptiven Lösung der Poisson-Gleichung mit hierarchischen Basen*, Diplomarbeit, Institut für Informatik, Technische Universität München, März 1994

[We] Werner D.: *Funktionalanalysis*, Springer, Berlin, 2002