Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

B-Spline-Kollokation mit der Kombinationstechnik

Sebastian Hasler

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dirk Pflüger
Betreuer/in:	M.Sc. Michael Rehme
Beginn am:	26. Oktober 2017
Beendet am:	26. April 2018

Kurzfassung

Wir implementieren eine Methode zur Lösung partieller Differentialgleichungen mithilfe einer Tensorproduktbasis aus B-Splines. Im Gegensatz zur Finite-Elemente-Methode, müssen dabei keine Integrale berechnet werden, was eine Beschleunigung mit sich bringt. Um in mehreren Dimensionen den Fluch der Dimensionalität zu umgehen, verwenden wir die Kombinationstechnik, mit welcher Probleme auf dünnen Gittern gelöst werden können. Die so erhaltene Lösung ist im Allgemeinen keine exakte Dünngitter-Lösung, stellt aber für bestimmte Probleme – wie zum Beispiel für die Poisson-Gleichung – eine gute Approximation dar. In der Hoffnung, diese Approximation zu verbessern, implementieren wir ein iteratives Verfahren, welches das Residuum an den Gitterpunkten weiter verringern soll. Wie sich später herausstellt, geschieht zwar Letzteres, doch wird die Lösung dabei in ihrer Gesamtheit nicht besser. Des Weiteren ist die Methode ungenau bei der Lösung von Problemen mit unstetigen Randbedingungen.

Inhaltsverzeichnis

1 Einleitung

Die *B-Spline-Kollokation mit der Kombinationstechnik* (BSKK) ist eine numerische Methode, um partielle Differentialgleichungen auf dünnen Gittern zu lösen. Bei der Verwendung von klassischen, gleichmäßigen Diskretisierungsschemas in mehreren Dimensionen spürt man schnell den sogenannten Fluch der Dimensionalität: Die Rechenzeit und der Speicheraufwand steigen exponentiell mit der Anzahl an Dimensionen. Diskretisierungen auf den in [Zen91] eingeführten dünnen Gittern haben $O\left(N \cdot \log(N)^{d-1}\right)$ Freiheitsgrade, wobei N die Anzahl an Randpunkten in einer Koordinatenrichtung und d die Dimensionalität ist [BG04; Gar12; GG08]. Das sind wesentlich weniger als die N^d Punkte eines regulären Gitters. Durch die Verwendung von dünnen Gittern kann somit die exponentielle Abhängigkeit von der Dimensionalität zu einem gewissen Grad umgangen werden. Eine Herangehensweise, um Probleme auf dünnen Gittern zu lösen, ist die Kombinationstechnik [GSZ92], bei der Lösungen auf unterschiedlichen anisotropen vollen Gittern zu einer Lösung auf einem dünnen Gitter kombiniert werden.

Bei der B-Spline-Kollokation wird die Lösung einer partiellen Differentialgleichung als Linearkombination von B-Splines [Boo78; Höl03] dargestellt und deren Koeffizienten durch das Lösen eines Gleichungssystems bestimmt. Diese Methode wurde auf vollen Gittern in [GMI12] untersucht. Sie erfordert im Gegensatz zur wohlbekannten Finite-Elemente-Methode [Höl03; KW17] keine Integration und ermöglicht daher eine beschleunigte Berechnung. Durch die Verallgemeinerung der Kombinationstechnik auf die B-Spline-Kollokation erwarten wir eine noch bessere Performanz und die Möglichkeit in höherdimensionalen Räumen rechnen zu können als bei der Verwendung von vollen Gittern.

Bei partiellen Differentialgleichungen führt die Kombinationstechnik allerdings nur unter bestimmten Bedingungen zu einer exakten Dünngitter-Lösung [Gar12; HGC06]. Beispielsweise erfüllt die Poisson-Gleichung $-\Delta u = f$ mit dem Laplace-Operator Δ diese Bedingungen zwar nicht [HGC06], die Lösung dieser Gleichung mit der Kombinationstechnik liefert aber dennoch eine gute Approximation [BGRZ94]. Wir beschreiben im Rahmen dieser Arbeit ein iteratives Verfahren, mit dem wir versuchen werden, die Genauigkeit dieser Approximation zu verbessern. Teil der Arbeit ist die Implementierung der B-Spline-Kollokation mit der Kombinationstechnik und dieses Verfahrens im Software-Paket SG++ [SGpp] – einem in C++ [Cpp] geschriebenen Werkzeugkasten für dünne Gitter.

Zuerst ordnen wir in Kapitel 2 diese Arbeit in den Kontext der Simulationspipeline [BZBP13] ein. Anschließend gibt Kapitel 3 einen Einblick in die notwendigen grundlegenden Kenntnisse über B-Splines, die Kollokationsmethode und die Kombinationstechnik. Kapitel 4 ist eine Aufführung der dieser Arbeit zugrundeliegenden Problemstellung einschließlich einer Beschreibung des oben genannten iterativen Verfahrens. Danach beschreiben wir in Kapitel 5 unsere Implementierung der notwendigen Funktionalitäten in SG++. In Kapitel 6 visualisieren wir einige Ergebnisse, um die Funktionsweise der Software zu validieren und Aussagen über die Effizienz des Algorithmus

sowie die Qualität der Ergebnisse machen zu können. Schließlich fassen wir in Kapitel 7 die Arbeit zusammen und geben in Kapitel 8 einen Ausblick auf mögliche zukünftige Arbeiten an diesem Thema.

2 Einordnung in die Simulationspipeline

Ein prädestiniertes Anwendungsgebiet für die B-Spline-Kollokation ist die Simulation. Dabei soll ein Phänomen, das in der Realität stattfinden könnte, simuliert werden. Auf diese Weise können auch dort Erkenntnisse gewonnen werden, wo die Theorie keine direkte Antwort liefert und auch kein Experiment durchgeführt werden kann. Die Theorie auf der einen Seite lässt oft keine analytischen Lösungen zu und ein Experiment auf der anderen Seite ist manchmal zu langsam durchzuführen, zu teuer, unerwünscht oder gar unmöglich [BZBP13; Meh18]. In diesen Fällen braucht es dann eine Simulation, um interpretierbare und zuverlässige Ergebnisse zu gewinnen. Dazu sind mehrere aufeinander aufbauende Schritte notwendig. Sie bilden die sogenannte Simulationspipeline [BZBP13], welche in Abbildung 2.1 wie in [Meh18] dargestellt ist.

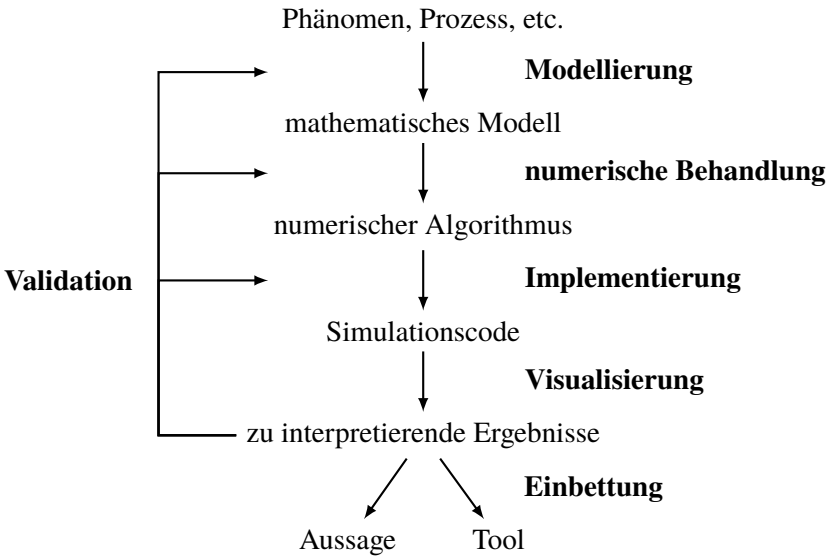


Abbildung 2.1: Die Simulationspipeline [Meh18]

2.1 Modellierung

Die Modellierung ist der erste Schritt der Simulationspipeline: Das Phänomen wird in ein – meist mathematisches – Modell überführt, das heißt in ein vereinfachendes Abbild der Realität [BZBP13]. Dabei gibt es oft mehr als ein geeignetes Modell. Zum Beispiel betrachten molekulardynamische Modelle einzelne Moleküle oder Atome mit ihren Wechselwirkungen, wie sie durch die Newtonschen Gesetze beschrieben werden. Kontinuumsmechanische Modelle dringen dagegen nicht so weit in die

Materie ein und fassen die Messgrößen anstatt dessen als stetige Funktionen auf, die im Rechengebiet durch partielle Differentialgleichungen beschrieben werden, welche sich typischerweise aus den Erhaltungsgesetzen ergeben [Meh18].

Bei der Wahl des Modells entscheidet sich folglich, welche Effekte berücksichtigt werden und welche nicht. Da kein Modell die Realität genau abbildet, gibt es zwangsläufig einen Modellierungsfehler, der das Ergebnis der Simulation verfälscht. Dies bedeutet jedoch nicht, dass immer ein molekulardynamisches Modell oder gar ein noch genaueres quantenmechanisches Modell verwendet werden sollte, denn das Modell bestimmt, welche numerischen Algorithmen zur Anwendung kommen können und in welchem Rahmen sich der Rechenaufwand bewegen wird. Demnach muss das Modell der Anwendung entsprechend gewählt werden. Bei der Wahl zwischen einem molekulardynamischen und einem kontinuumsmechanischen Modell ist es entscheidend, für welche räumliche Skala man sich interessiert. Erscheint die Messgröße auf dieser Skala als homogen, genügt ein kontinuumsmechanisches Modell. Will man hingegen molekulardynamische Effekte auf kleinster Skala betrachten, so benötigt man ein molekulardynamisches Modell [Meh18].

Da wir bei der B-Spline-Kollokation Lösungen als stückweise Polynome darstellen, also als stetige Funktionen [Boo78], wird ein kontinuumsmechanisches Modell vorausgesetzt. Dementsprechend liefert die B-Spline-Kollokation auch nur für diejenigen Phänomene zuverlässige Ergebnisse, für die eine kontinuumsmechanische Modellierung sinnvoll ist. In dieser Arbeit beschäftigen wir uns nicht weiter mit der Modellierung, sondern setzen voraus, dass ein Modell in Form von partiellen Differentialgleichungen bereits vorhanden ist. Als dauerhaftes Beispiel verwenden wir die Wärmeleitungsgleichung [Meh18], die in Abschnitt 3.5 genauer beschrieben wird. Da die in dieser Arbeit vorgestellte Methode jedoch – bis auf die eben genannten Anforderungen – das Modell nicht weiter vorgibt, lassen sich damit die unterschiedlichsten Problemstellungen lösen.

2.2 Numerische Behandlung

Da eine analytische Lösung oft nicht möglich ist, müssen die aus einem Modell hervorgehenden Gleichungen diskretisiert werden. Dabei erhält man typischerweise ein Gleichungssystem mit endlich vielen Unbekannten, welches auf Rechnern mit endlicher Rechenleistung gelöst werden kann [Meh18]. Dies geschieht bei der B-Spline-Kollokation, indem die Lösung – wie in Abschnitt 3.1 beschrieben – als Linearkombination endlich vieler B-Splines dargestellt wird. Die Unbekannten sind dabei die Koeffizienten der einzelnen B-Splines und bilden einen Lösungsraum endlicher Dimension.

Man unterscheidet zwischen der räumlichen und der zeitlichen Diskretisierung. Während ein molekulardynamisches Modell bereits räumlich diskret ist, da endlich viele Partikel im Raum Platz finden, müssen die hier relevanten kontinuumsmechanischen Modelle sowohl im Raum, als auch gegebenenfalls in der Zeit diskretisiert werden. Die zeitliche Diskretisierung ist notwendig, falls man sich nicht nur für einen spezifischen Zeitpunkt interessiert, sondern das Rechengebiet auch die Zeitdimension umfasst [Meh18]. Bei der zeitlichen Diskretisierung werden wir die Wahl haben, ob wir die Zeitdimension einfach analog zu den Raumdimensionen betrachten, oder ob wir in der Zeit separat diskretisieren und Zeitschrittverfahren [Han02] verwenden.

Die numerische Behandlung umfasst neben der Diskretisierung auch das Finden eines Lösungsalgorithmus, mit dem die Werte der Unbekannten, die sich aus der Diskretisierung ergeben, berechnet werden können [Meh18]. Die BSKK gibt neben der Diskretisierung auch den Lösungsalgorithmus vor; nämlich die Kombinationstechnik, welche in Abschnitt 3.8 vorgestellt wird.

3 Grundlagen

Dieses Kapitel gibt einen Überblick über B-Splines, die Kollokationsmethode und die Kombinationstechnik. Die B-Splines bilden eine Basis der stückweisen Polynome und deshalb beginnen wir mit Letzteren.

3.1 Stückweise Polynominterpolation

Definition 3.1.1
Wir notieren den Raum aller Polynome $\mathbb{R} \to \mathbb{R}$ vom Grad höchstens n als

$$\Pi_n := \left\{ p : \mathbb{R} \to \mathbb{R} \mid \exists a_0, a_1, \ldots, a_n \in \mathbb{R} : p(x) = \sum_{i=0}^{n} a_i x^i \right\}. \tag{3.1}$$

Da wir stückweise Polynomen verwenden wollen, müssen wir den Definitionsbereich einschränken. Deshalb definieren wir wie in [PZ18]:

Definition 3.1.2
Sei $f : X \to Y$ eine Funktion und $T \subseteq X$. Dann definieren wir

$$\begin{aligned} f|_T : T &\to \mathbb{R} \\ x &\mapsto f(x) \end{aligned} \tag{3.2}$$

als die Funktion, die man erhält, wenn der Definitionsbereich von f auf T beschränkt wird.

Definition 3.1.3
Sei R ein Raum von Funktionen $X \to Y$ und $T \subseteq X$. Dann definieren wir

$$R|_T := \{ f|_T \mid f \in R \} \tag{3.3}$$

als den Raum, in dem der Definitionsbereich aller Funktionen aus R auf T beschränkt wurde.

Nun können wir den Definitionsbereich von Polynomen einschränken: Wir betrachten $\Pi_n|_X$ für eine unendliche Teilmenge $X \subset \mathbb{R}$ und interessieren uns für dessen Dimension.

Lemma 3.1.1
Sei $X \subseteq \mathbb{R}$ unendlich. Dann ist

$$\dim \Pi_n|_X = n + 1. \tag{3.4}$$

Beweis. Es genügt eine $n+1$-elementige Basis von $\Pi_n|_X$ anzugeben. Aus Definition 3.1.1 folgt direkt, dass

$$B_n := \{\phi_i \mid 0 \leq i \leq n\} \tag{3.5}$$

mit

$$\begin{aligned} \phi_i : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto x^i \end{aligned} \tag{3.6}$$

ein Erzeugendensystem von Π_n bildet. Somit ist $B_n|_X$ ein Erzeugendensystem von $\Pi_n|_X$. Wenn eine Linearkombination p der Elemente von $B_n|_X$

$$p(x) = \sum_{i=0}^{n} a_i x^i \tag{3.7}$$

gleich der Nullfunktion auf X ist, hat p unendlich viele Nullstellen, weil X unendlich ist. Infolgedessen müssen alle Koeffizienten a_i gleich 0 sein. Somit ist $B_n|_X$ eine linear unabhängige Menge und demnach eine Basis von $\Pi_n|_X$. Da die $n+1$ Basisfunktionen ϕ_i linear unabhängig sind, sind sie insbesondere paarweise ungleich und folglich ist $B_n|_X$ eine $n+1$-elementige Menge. □

Als Korollar bemerken wir, dass ebenfalls $\dim \Pi_n = \dim \Pi_n|_{\mathbb{R}} = n+1$ ist [PZ18]. Die Einschränkung des Definitionsbereich auf unendliche Teilmengen $X \subset \mathbb{R}$ ändert also nichts an der Dimension von Π_n.

Wir haben jetzt alles, was wir für die stückweisen Polynome brauchen, welche wir ebenfalls angelehnt an [PZ18] definieren.

Definition 3.1.4
Sei $\tau = (\tau_0, \tau_1, \ldots, \tau_N)$ eine streng monoton wachsende Folge von Knoten. Dann schreiben wir für $0 \leq i < n$

$$\tau[i] := \begin{cases} [\tau_i, \tau_{i+1}) & \text{falls } i < n-1 \\ [\tau_i, \tau_{i+1}] & \text{falls } i = n-1 \end{cases} \tag{3.8}$$

und bezeichnen

$$\Pi_{n,\tau} := \left\{ f : [\tau_0, \tau_N] \to \mathbb{R} \mid \forall\, 0 \leq i < N : f|_{\tau[i]} \in \Pi_n|_{\tau[i]} \right\} \tag{3.9}$$

als den Raum der stückweisen Polynome vom Grad höchstens n zu den Knoten τ.

Ein stückweises Polynom ist zusammengesetzt aus N Polynomen vom Grad höchstens n mit jeweils $n+1$ Freiheitsgraden. Demnach erhalten wir folgendes Lemma [PZ18], welches ein Spezialfall des später vorkommenden Lemmas 3.1.3 ist.

Lemma 3.1.2
Für $\Pi_{n,\tau}$ aus Definition 3.1.4 gilt

$$\dim \Pi_{n,\tau} = N \cdot (n+1). \tag{3.10}$$

Die Dimension von $\Pi_{n,\tau}$ steigt also nicht nur mit dem Polynomgrad, sondern auch mit der Anzahl an Teilintervallen. Dies ermöglicht es, Funktionen zu interpolieren, ohne dabei – wie bei der Polynominterpolation – den Polynomgrad mit der Stützstellenanzahl erhöhen zu müssen. Bei der stückweisen Polynominterpolation soll eine auf einem Intervall definierte Funktion $f : [a, b] \to \mathbb{R}$ an den Stützstellen $\tau = (\tau_0, \tau_1, \ldots, \tau_N)$ mit

$$a = \tau_0 < \tau_1 < \cdots < \tau_N = b \tag{3.11}$$

durch ein stückweises Polynom $\hat{f} \in \Pi_{n,\tau}$ interpoliert werden [PZ18]. Es sollen also die Bedingungen

$$\hat{f}(\tau_i) = f(\tau_i) \quad \forall\, 0 \le i \le N \tag{3.12}$$

erfüllt sein. Häufig sind zusätzlich Glattheitseigenschaften erwünscht; mindestens aber, dass der Interpolant $\hat{f}$ stetig ist. Die Glattheitseigenschaften können wir mit der nächsten Definition aus [PZ18] formalisieren.

Definition 3.1.5
Für $X \subseteq \mathbb{R}$ bezeichnen wir mit

- $C^{-1}(X)$ *die Menge aller,*
- $C^0(X)$ *die Menge der stetigen und*
- $C^k(X)$ *die Menge der k mal stetig differenzierbaren*

Funktionen $X \to \mathbb{R}$.

Wir suchen nun einen Interpolanten aus $\Pi_{n,\tau} \cap C^k([a, b])$, wobei n der Polynomgrad ist und k die Glattheitsbedingung beschreibt. Abbildung 3.1 zeigt eine solche Interpolation mit $n = 1$ und $k = 0$, das heißt der Interpolant ist stückweise linear und stetig.

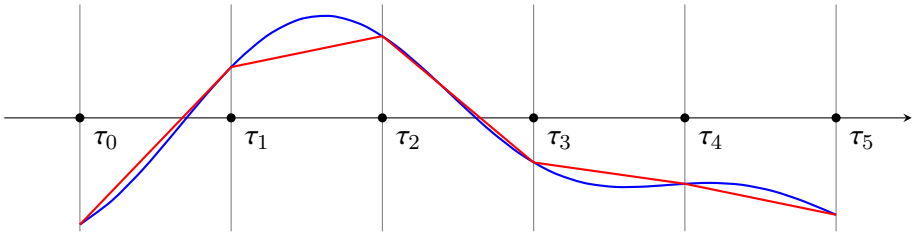


Abbildung 3.1: Interpolation in $\Pi_{1,\tau} \cap C^0([\tau_0, \tau_N])$ mit $N = 5$

Bei der Frage, wie k gewählt werden darf, um eine Lösung zu erhalten, ist die Dimension von $\Pi_{n,\tau} \cap C^k([a, b])$ entscheidend.

Lemma 3.1.3
Für $\Pi_{n,\tau}$ aus Definition 3.1.4 und $-1 \le k \le n$ gilt

$$\dim\left(\Pi_{n,\tau} \cap C^k([\tau_0, \tau_N])\right) = n + 1 + (N - 1) \cdot (n - k). \tag{3.13}$$

Ein Beweis für den Spezialfall $k = n - 1$ findet sich beispielsweise in [Dah08].

3.2 Spline-Raum

Wählen wir für $\Pi_{n,\tau} \cap C^k([\tau_0, \tau_N])$ zum Beispiel $n = k$, erhalten wir nach Lemma 3.1.3

$$\dim\left(\Pi_{n,\tau} \cap C^n([\tau_0, \tau_N])\right) = n + 1. \tag{3.14}$$

Diese Dimension genügt uns also nicht, um bei festem Grad n in beliebig vielen Stützstellen zu interpolieren. Das ist keine Überraschung, wenn man sich überlegt, dass

$$\Pi_{n,\tau} \cap C^n([\tau_0, \tau_N]) = \Pi_n|_{[\tau_0,\tau_N]} \tag{3.15}$$

ein einfacher Polynomraum ist. Eine Interpolation in diesem Raum entspricht also der normalen Polynominterpolation. Da bei der stückweisen Polynominterpolation die $N + 1$ Bedingungen aus Gleichung (3.12) erfüllt sein sollen, benötigen wir um die Existenz einer Lösung zu garantieren mindestens $N + 1$ Freiheitsgrade, also

$$\dim\left(\Pi_{n,\tau} \cap C^k([\tau_0, \tau_N])\right) \overset{!}{\geq} N + 1. \tag{3.16}$$

Damit dies bei festem Grad n für eine beliebig große Anzahl an Teilintervallen N erfüllt sein kann, muss wegen Lemma 3.1.3 $n > k$ gelten. Mit anderen Worten: Bei der Interpolation durch stückweise Polynome vom Grad n dürfen wir $n-1$-fache stetige Differenzierbarkeit des Interpolanten verlangen. Dies motiviert zur folgenden Definition:

Definition 3.2.1
Für eine streng monoton wachsende Folge von Knoten $\tau = (\tau_1, \tau_2, \ldots, \tau_N)$ bezeichnen wir

$$S_{n,\tau} := \Pi_{n,\tau} \cap C^{n-1}([\tau_0, \tau_N]) \tag{3.17}$$

als den Spline-Raum *vom Grad n.*

Mit Lemma 3.1.3 erhalten wir

$$\dim S_{n,\tau} = N + n. \tag{3.18}$$

Wird eine Funktion im Spline-Raum interpoliert, können also neben den $N + 1$ Bedingungen aus Gleichung (3.12) $n - 1$ weitere Bedingungen formuliert werden. Die bereits zuvor erwähnte Abbildung 3.1 zeigt tatsächlich eine Spline-Interpolation durch einen linearen Spline.

Um Interpolationsprobleme im Spline-Raum zu lösen, wollen wir zuerst eine Basis bestimmen. Dann könnten wir den Interpolanten als Linearkombination der Basisfunktionen darstellen, diese in die $N + n$ Interpolationsbedingungen einsetzen und erhielten so ein lineares Gleichungssystem (LGS), wobei die Koeffizienten der Linearkombination die Unbekannten sind. Wie man LGS effizient löst ist immer noch aktueller Forschungsgegenstand [Han02], aber nicht im Rahmen dieser Arbeit.

Eine Basis von $S_{n,\tau}$ erhalten wir folgendermaßen: Wir benutzen die Schreibweise

$$x_+^k := \begin{cases} x^k & \text{falls } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{3.19}$$

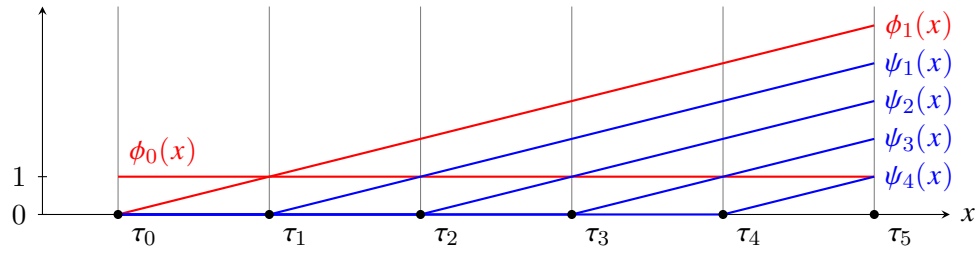
und nehmen die Basis

$$\{\phi_k \mid 0 \leq k \leq n\} \cup \{\psi_i \mid 1 \leq i < N\} \tag{3.20}$$

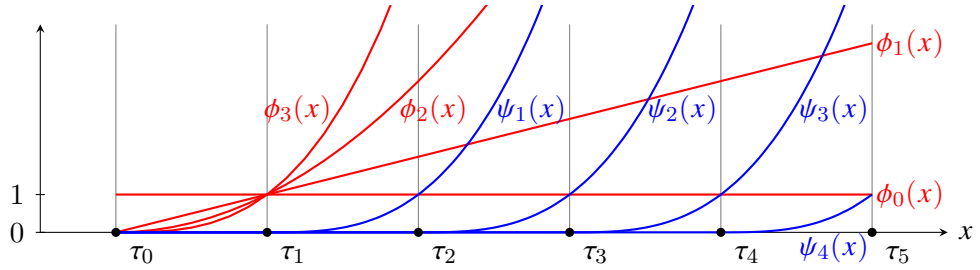
definiert durch

$$\begin{aligned} \phi_k(x) &= (x - \tau_0)^k \\ \psi_i(x) &= (x - \tau_i)_+^n \end{aligned} \tag{3.21}$$

[Lub]. Die lineare Unabhängigkeit dieser Funktionen lässt sich ähnlich wie bei der Basis des Polynomraums aus Gleichung (3.5) leicht zeigen und die Anzahl stimmt mit dim $S_{n,\tau}$ aus Gleichung (3.18) überein, also handelt es sich in der Tat um eine Basis des $S_{n,\tau}$. Abbildung 3.2 zeigt diese Basis für den linearen und kubischen Spline-Raum mit $N = 5$ Teilintervallen. Sie besteht lediglich aus einer Basis des Polynomraums (die Funktionen ϕ_k), die pro weiteres Teilintervall um eine $n-1$ mal stetig differenzierbare Funktion ψ_i ergänzt wurde. Erst die n-te Ableitung $\psi_i^{(n)}$ ist an der Stelle τ_i undefiniert und nicht stetig.



(a) $n = 1$



(b) $n = 3$

Abbildung 3.2: Basen zweier Spline-Räume $S_{n,\tau}$ mit $N = 5$

Ein Element des Spline-Raums hat – in dieser Basis dargestellt – die Form

$$\hat{f}(x) = \sum_{k=0}^{n} \alpha_k \phi_k(x) + \sum_{i=1}^{N-1} \beta_i \psi_i(x). \tag{3.22}$$

Zur Lösung eines Interpolationsproblems müssen wir diese Darstellung in die Interpolationsbedingungen einsetzen. Machen wir das zum Beispiel mit Gleichung (3.12), erhalten wir

$$\sum_{k=0}^{n} \alpha_k \phi_k(\tau_i) + \sum_{i=1}^{N-1} \beta_i \psi_i(\tau_i) = f(\tau_i) \quad \forall\, 0 \le i \le N. \tag{3.23}$$

Von der Tatsache, dass dies nur $N+1$ Bedingungen sind und somit für eine quadratische Matrix noch $n-1$ weitere Zeilen fehlen, sehen wir zunächst einmal ab. Die Einträge der Matrix sind die in Gleichung (3.23) vorkommenden Werte $\phi_k(\tau_i)$ und $\psi_i(\tau_i)$; die rechte Seite ist dabei der Spaltenvektor $(f(\tau_i))_{0 \le i \le N}^T$. Nun haben die Basisfunktionen ϕ_k und ψ_i große Träger: Diese erstrecken sich

durchschnittlich über mehr als die Hälfte des Rechengebietes $[\tau_0, \tau_N]$. Folglich wird die resultierende Matrix auch entsprechend zu über der Hälfte mit nicht verschwindenden Einträgen besetzt sein. Da dünn besetzte Matrizen – wie zum Beispiel Bandmatrizen – wesentlich effizienter zu lösen sind [Han02], wäre es wünschenswert eine Basis zu finden, deren Basisfunktionen möglichst kleine Träger haben.

3.3 B-Splines

Wie wir in Abschnitt 3.2 gesehen haben, ist eine einfach zu findende Basis nicht zwingend auch für den Anwendungsfall geeignet. Auf der Suche nach Basisfunktionen mit möglichst kleinem Träger betrachten wir zuerst einmal nur den Fall konstanter Splines, also aus $S_{0,\tau}$. Nach Gleichung (3.18) ist $\dim S_{0,\tau} = N$, also ist

$$\{B_{i,0} : [\tau_0, \tau_N] \to \mathbb{R} \mid 0 \leq i < N\} \tag{3.24}$$

mit

$$B_{i,0}(x) := \begin{cases} 1 & \text{falls } x \in \tau[i] \\ 0 & \text{sonst} \end{cases} \tag{3.25}$$

offenkundig eine Basis von $S_{0,\tau}$. Wir nennen sie die konstanten B-Splines. Zur Erinnerung an Definition 3.1.4: $\tau[i]$ ist das Teilintervall, welches bei τ_i anfängt und bei τ_{i+1} aufhört.

Konstante Splines sind nicht sonderlich nützlich. Eine Basis mit ähnlich lokalen Basisfunktionen gibt es auch für die linearen Splines $S_{1,\tau}$, nämlich die sogenannten Hutfunktionen [Han02]

$$\{B_{i,1} : [\tau_0, \tau_N] \to \mathbb{R} \mid -1 \leq i < N\} \tag{3.26}$$

mit

$$B_{i,1}(x) := \begin{cases} \frac{x-\tau_i}{\tau_{i+1}-\tau_i} & \text{falls } x \in \tau[i] \\ \frac{\tau_{i+2}-x}{\tau_{i+2}-\tau_{i+1}} & \text{falls } x \in \tau[i+1] \\ 0 & \text{sonst.} \end{cases} \tag{3.27}$$

Bei den Hutfunktionen auf dem Rand stehen hier die Intervalle $\tau[-1]$ und $\tau[N]$, die es eigentlich gar nicht gibt. Um dies zu beheben, setzen wir einfach die Knoten ins negative und positive Unendliche fort. Dann können wir sogar alle $i \in \mathbb{Z}$ in Gleichung (3.27) einsetzen. Wir nennen diese Hutfunktion die linearen B-Splines. Sie beinhalten eine Basisfunktion mehr als bei den konstanten B-Splines, denn mit jeder Inkrementierung des Polynomgrades muss wegen $\dim S_{0,\tau} = N + n$ eine Basisfunktion dazukommen.

Die Bedingung $x \in \tau[i]$ lässt sich auch anders ausdrücken:

$$x \in \tau[i] \Leftrightarrow B_{i,0}(x) = 1 \quad \forall i \in \mathbb{Z}. \tag{3.28}$$

Mit dieser Schreibweise lässt sich Gleichung (3.27) kompakter schreiben:

$$B_{i,1}(x) = \frac{x - \tau_i}{\tau_{i+1} - \tau_i} B_{i,0}(x) + \frac{\tau_{i+2} - x}{\tau_{i+2} - \tau_{i+1}} B_{i+1,0}(x). \tag{3.29}$$

Wir starten also mit zwei nebeneinanderliegenden konstanten B-Splines, multiplizieren den Linken mit einer über seinem Träger von 0 auf 1 steigenden linearen Funktion und behandeln den rechten symmetrisch. Deren Summe ist dann ein linearer B-Spline. Dieses Prinzip können wir für beliebige Grade zu einer Rekursionsgleichung verallgemeinern [BSBF]:

Definition 3.3.1
Sei $\tau = (\tau_0, \tau_1, \ldots, \tau_N)$ eine streng monoton wachsende Folge von Knoten. Für $n \in \mathbb{N}_0$ und $i \in \mathbb{Z}$ heißen die $B_{i,n}$, definiert durch

$$\begin{aligned} B_{i,0}(x) &:= \begin{cases} 1 & \text{falls } x \in \tau[i] \\ 0 & \text{sonst,} \end{cases} \\ B_{i,n}(x) &:= \frac{x - \tau_i}{\tau_{i+n} - \tau_i} B_{i,n-1}(x) + \frac{\tau_{i+n+1} - x}{\tau_{i+n+1} - \tau_{i+1}} B_{i+1,n-1}(x) \quad \text{für } n \geq 1, \end{aligned} \tag{3.30}$$

B-Splines vom Grad n.

Das ist genau das, was wir gebraucht haben, denn es gilt:

Lemma 3.3.1
Für alle $n \in N_0$ ist

$$\{B_{i,n} \mid -n \leq i < N\} \tag{3.31}$$

eine Basis des $S_{n,\tau}$.

Ein Beweis findet sich zum Beispiel in [Lub].

Die B-Splines vom Grad n haben einen Träger, der $n+1$ Teilintervalle umfasst. Wenn man also n relativ klein wählt – üblich sind etwa die in Abbildung 3.3 gezeigten kubischen Splines mit $n = 3$ – erhält man bei der B-Spline-Interpolation eine Bandmatrix, welche effizient gelöst werden kann [Han02]. Selbstverständlich muss man n mindestens groß genug wählen, um die geforderten Glattheitsbedingungen zu erfüllen.

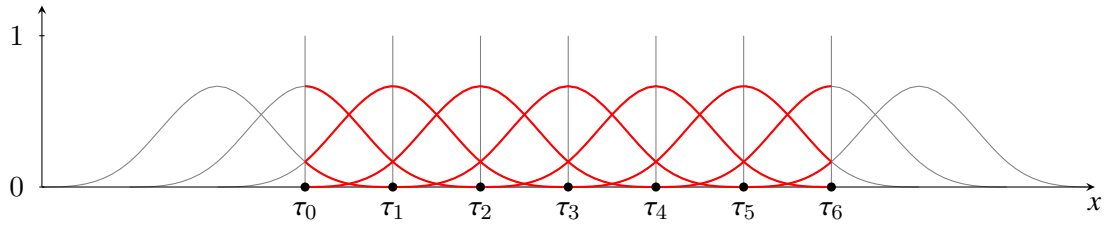


Abbildung 3.3: Kubische B-Splines bezüglich $(\tau_0, \tau_1, \ldots, \tau_4)$

Beim Lösen eines Interpolationsproblems würden wir die Knoten der B-Splines gerne auf den Stützstellen platzieren. Dabei gibt es jedoch ein Problem: Mit unserer bisherigen Notation war $N+1$ die Anzahl der Knoten und nach Gleichung (3.18) $N+n$ die zugehörige Anzahl an Freiheitsgraden. Würden wir pro Knoten eine Stützstelle verwenden, wären das zu viele Freiheitsgrade, denn wir brauchen genauso viele Freiheitsgrade wie Stützstellen. Sei nun $M+1$ die Stützstellenanzahl, dann soll konkret $N+n = M+1$ gelten und folglich brauchen wir nur $N+1 = M-n+2 = M+1-(n-1)$ Knoten. Das erreichen wir, indem alle außer $n-1$ Stützstellen als Knoten verwendet werden.

Definition 3.3.2
Sei $s = (x_0, x_1, \ldots, x_M)$ eine streng monoton wachsende Folge von Stützstellen und $n \geq 1$ ungerade, wobei $M \geq n$ gilt. Dann bezeichnet

$$s^{NAK} := (x_0, x_r, x_{r+1}, \ldots, x_{M-r-1}, x_{M-r}, x_M) \tag{3.32}$$

mit

$$r := \frac{n+1}{2} \tag{3.33}$$

die Not-a-Knot *Knoten zu s.*

Bei den *Not-a-Knot* Knoten werden also rechts vom ersten und links vom letzten jeweils $r-1$ Knoten herausgenommen. Da dabei $M-n+2$ Knoten übrig bleiben und wir mindestens zwei Knoten für einen Spline-Raum brauchen, soll $M \geq n$ gelten; dann können wir den zugehörigen Spline-Raum $S_{n,s^{\text{NAK}}}$ betrachten. s^{NAK} lässt sich als Folge von Knoten $(\tau_0, \tau_1, \ldots, \tau_{M-(n-1)})$ auffassen, also gilt nach Gleichung (3.18)

$$\dim S_{n,s^{\text{NAK}}} = M - (n-1) + n = M + 1. \tag{3.34}$$

Dies ist genau die Dimension, die wir brauchen, um in $M+1$ Stützstellen zu interpolieren. Die B-Splines zu den Knoten s^{NAK} nennen wir *Not-a-Knot* B-Splines. Sie sind für den kubischen Fall in Abbildung 3.4 dargestellt.

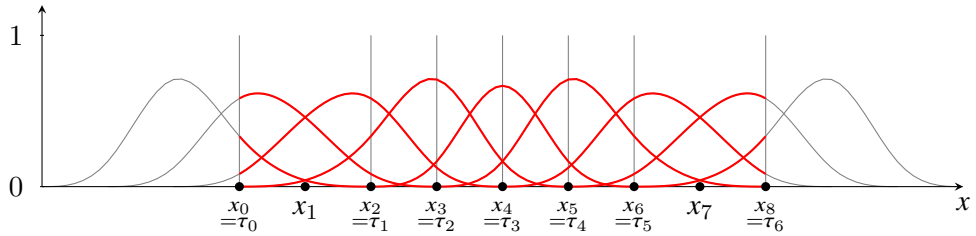


Abbildung 3.4: Kubische *Not-a-Knot* B-Splines bezüglich $(x_0, x_1, \ldots, x_8)^{\text{NAK}}$

3.4 Tensorprodukt

Definition 3.4.1
Seien $f : X_f \to \mathbb{R}$ und $g : X_g \to \mathbb{R}$ Funktionen. Dann definieren wir das Tensorprodukt von f und g durch

$$\begin{aligned} f \otimes g : X_f \times X_g &\to \mathbb{R} \\ (x, y) &\mapsto f(x) \cdot g(y). \end{aligned} \tag{3.35}$$

Definition 3.4.2
Seien V, W zwei Funktionsräume, F eine Basis von V und G eine Basis von W. Dann definieren wir den Tensorproduktraum von V und W durch

$$V \otimes W := \operatorname{span}\{f \otimes g \mid f \in F, g \in G\}. \tag{3.36}$$

Beweis der Wohldefiniertheit. Für zwei Basen F, Σ von V und zwei Basen G, Γ von W müssen wir

$$\operatorname{span}\{f \otimes g \mid f \in F, g \in G\} = \operatorname{span}\{\sigma \otimes \gamma \mid \sigma \in \Sigma, \gamma \in \Gamma\} \tag{3.37}$$

zeigen. Wir nummerieren die Elemente von $\Sigma = \{\sigma_i \mid i \in I\}$ und $\Gamma = \{\gamma_j \mid j \in J\}$. Seien $f \in F$ und $g \in G$ beliebig. Da Σ eine Basis von V ist, lässt sich $f \in V$ als Linearkombination

$$f = \sum_{i \in I} \alpha_i \sigma_i \tag{3.38}$$

darstellen. Analog ist

$$g = \sum_{j \in J} \beta_j \gamma_j. \tag{3.39}$$

Nehmen wir das Tensorprodukt, erhalten wir

$$(f \otimes g)(x, y) = f(x) \cdot f(y) = \left(\sum_{i \in I} \alpha_i \sigma_i(x) \right) \left(\sum_{j \in J} \beta_j \gamma_j(y) \right) = \sum_{i \in I} \sum_{j \in J} \alpha_i \beta_j \left(\sigma_i \otimes \gamma_j \right)(x, y) \tag{3.40}$$

und demnach ist $f \otimes g \in \operatorname{span}\{\sigma \otimes \gamma \mid \sigma \in \Sigma, \gamma \in \Gamma\}$, und zwar für alle $f \in F$ und $g \in G$. Es folgt

$$\operatorname{span}\{f \otimes g \mid f \in F, g \in G\} = \operatorname{span}\{\sigma \otimes \gamma \mid \sigma \in \Sigma, \gamma \in \Gamma\} \tag{3.41}$$

und analog folgt die umgekehrte Teilmengenbeziehung. □

Lemma 3.4.1
Seien wieder V, W zwei Funktionsräume, F eine Basis von V und G eine Basis von W. Dann ist

$$\{f \otimes g \mid f \in F, g \in G\} \tag{3.42}$$

eine Basis von $V \otimes W$.

Beweis. In Definition 3.4.2 lesen wir, dass $\{f \otimes g \mid f \in F, g \in G\}$ ein Erzeugendensystem von $V \otimes W$ ist. Es bleibt die lineare Unabhängigkeit zu zeigen. Wir nummerieren wieder die Elemente von $F = \{f_i \mid i \in I\}$ und $G = \{g_j \mid j \in J\}$. Angenommen

$$\sum_{i \in I} \sum_{j \in J} \alpha_{i,j} \left(f_i \otimes g_j \right) \tag{3.43}$$

ist die Nullfunktion. Dann gilt für alle x, y

$$\sum_{i \in I} \sum_{j \in J} \alpha_{i,j} \left(f_i \otimes g_j \right)(x, y) = \sum_{i \in I} \sum_{j \in J} \alpha_{i,j} f_i(x) g_j(y) = \sum_{i \in I} f_i(x) \sum_{j \in J} \alpha_{i,j} g_j(y) = 0. \tag{3.44}$$

Da F eine Basis von V ist, sind die f_i linear unabhängig und folglich gilt für alle $i \in I$ und für alle y

$$\sum_{j \in J} \alpha_{i,j} g_j(y) = 0. \tag{3.45}$$

Analog sind auch die g_j linear unabhängig, also sind alle Koeffizienten $\alpha_{i,j}$ gleich 0. Demnach ist $\{f \otimes g \mid f \in F, g \in G\}$ linear unabhängig. □

Eine direkte Folgerung von Lemma 3.4.1 ist

$$\dim(V \otimes W) = \dim V \cdot \dim W. \tag{3.46}$$

Zur Lösung von Problemen in mehreren Dimensionen werden wir in jeder Dimension einen Spline-Raum nehmen und das Problem in deren Tensorproduktraum lösen. Als Basisfunktionen können wir nach Lemma 3.4.1 die Tensorprodukte der zugehörigen B-Splines nehmen.

3.5 Partielle Differentialgleichungen

Wir wollen nicht interpolieren, sondern partielle Differentialgleichungen numerisch lösen. Das sind Differentialgleichungen, die partielle Ableitungen beinhalten [Han02]. Sie haben die Form

$$\mathcal{L}u = f. \tag{3.47}$$

Dabei ist $\mathcal{L}$ ein Differentialoperator, der partielle Ableitungen enthält, $f : X^d \to \mathbb{R}$ gegeben und $u : X^d \to \mathbb{R}$ die gesuchte Lösung. Mit $\mathcal{L} = -\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2}$ und $d = 2$ Dimensionen hätten wir die Gleichung

$$-\frac{\partial^2}{\partial x^2}u(x,y) - \frac{\partial^2}{\partial y^2}u(x,y) = f(x,y). \tag{3.48}$$

Dies ist der zweidimensionale Fall der sogenannten Poisson-Gleichung

$$-\Delta u = f \tag{3.49}$$

mit

$$\Delta = \sum_{k=1}^{d} \frac{\partial^2}{\partial x_k^2} \tag{3.50}$$

[Meh18], welche wir als dauerhaftes Beispiel für eine Gleichung, die es zu lösen gilt, verwenden. Δ heißt Laplace-Operator und ordnet einer Funktion die Summe ihrer zweiten Ableitungen in allen Dimensionen zu. Die Poisson-Gleichung ist der stationäre Fall der Wärmeleitungsgleichung

$$\frac{\partial}{\partial t}u(\vec{x},t) - c\Delta u(\vec{x},t) = f(\vec{x},t), \tag{3.51}$$

(bei der sich der Laplace-Operator nur auf die Raumdimensionen bezieht). Sie beschreibt die Änderung der Temperatur im Berechnungsgebiet. Dabei beschreibt $f(\vec{x},t)$ den Einfluss zusätzlicher Wärmequellen. Ohne diese hat man die homogene Wärmeleitungsgleichung

$$\frac{\partial}{\partial t}u(\vec{x},t) - c\Delta u(\vec{x},t) = 0. \tag{3.52}$$

Die physikalische Herleitung findet sich zum Beispiel in [Meh18].

Viele Differentialgleichungen sind – wie auch die Wärmeleitungsgleichung – nicht eindeutig lösbar, sondern haben viele Lösungen. Um die Lösung eindeutig zu machen, müssen zusätzliche Bedingungen an die Lösung geknüpft werden; in der Regel in Form von Randbedingungen. Suchen wir zum Beispiel eine Funktion $u : [0,1]^2 \to \mathbb{R}$, welche die folgende Form der Poisson-Gleichung erfüllt:

$$\Delta u = 0. \tag{3.53}$$

Für jede konstante und jede lineare Funktion u in kartesischen Koordinaten (x, y) gilt

$$\Delta u = \frac{\partial^2}{\partial x^2}u + \frac{\partial^2}{\partial y^2}u = 0 + 0 = 0, \tag{3.54}$$

also sind das alles Lösungen. Damit nicht genug: Es gibt noch viele weitere Arten von Lösungen für Gleichung (3.53). Fügen wir jedoch auf dem Rand $\partial[0,1]^2$ die Randbedingungen

$$\begin{aligned} u(0,y) &= 0 \quad \forall y \in [0,1] \\ u(1,y) &= 1 \quad \forall y \in [0,1] \\ u(x,0) &= x \quad \forall x \in [0,1] \\ u(x,1) &= x \quad \forall x \in [0,1] \end{aligned} \tag{3.55}$$

hinzu, erhalten wir die eindeutige Lösung [Han02]

$$u(x, y) = x. \tag{3.56}$$

3.6 Kollokation

Aber wie wollen wir nun partielle Differentialgleichungen lösen und was hat das mit Interpolation zu tun? Bei der Interpolation hat man Stützstellen – im Folgenden x_i mit $i \in I$ genannt – und will, dass der Interpolant an den Stützstellen bestimmte Werte annimmt. Stellt man den Interpolanten in einer Basis

$$B = \{b_m \mid m \in M\} \tag{3.57}$$

dar, erhält man also die Bedingungen

$$\sum_{m \in M} \alpha_m b_m(x_i) = f(x_i) \quad \forall i \in I, \tag{3.58}$$

wobei f die zu interpolierende Funktion ist. Auch zur numerischen Lösung einer partiellen Differentialgleichung $\mathcal{L}u = f$ wollen wir die Lösung u in einer gewählten Basis darstellen:

$$u = \sum_{m \in M} \alpha_m b_m. \tag{3.59}$$

Um die numerische Lösung zu berechnen werden wir ebenfalls Stützstellen verwenden, das heißt die Lösung soll an den Stützstellen die Differentialgleichung erfüllen. Die Bedingungen sind dann

$$\mathcal{L}\left(\sum_{m \in M} \alpha_m b_m\right)(x_i) = f(x_i) \quad \forall i \in I \tag{3.60}$$

und falls der Differentialoperator $\mathcal{L}$ linear ist, wird daraus ein LGS

$$\sum_{m \in M} \alpha_m \mathcal{L} b_m(x_i) = f(x_i) \quad \forall i \in I, \tag{3.61}$$

mit Einträgen $\mathcal{L}b_m(x_i)$ und den Unbekannten α_m. Diese Methode nennen wir Kollokation und wenn die B-Splines respektive deren Tensorprodukte als Basis genutzt werden, B-Spline-Kollokation. Der Laplace-Operator Δ ist linear, denn für Skalare $\alpha, \beta \in \mathbb{R}$ und Funktionen u, v in kartesischen Koordinaten $(x_1, x_2, \ldots, x_d)$ gilt

$$\Delta(\alpha u + \beta v) = \sum_{k=1}^{d} \frac{\partial^2}{\partial x_k^2}(\alpha u + \beta v) = \sum_{k=1}^{d} \left(\alpha \frac{\partial^2}{\partial x_k^2} u + \beta \frac{\partial^2}{\partial x_k^2} v\right) = \alpha \Delta u + \beta \Delta v. \tag{3.62}$$

Wir können also zum Beispiel die Poisson-Gleichung $-\Delta u = f$ mit der Kollokationsmethode lösen. Im Folgenden suchen wir eine Funktion $u : [0, 1]^2 \to \mathbb{R}$, die

$$\Delta u = 0 \tag{3.63}$$

erfüllt. Wie in Abschnitt 3.5 beschrieben, müssen wir Randbedingungen spezifizieren, damit die Lösung eindeutig ist. Wir können zum Beispiel Dirichlet-Randbedingungen [Meh18]

$$u(x, y) = u_D(x, y) \quad \forall (x, y) \in \partial[0, 1]^2 \tag{3.64}$$

verwenden, wobei $\partial[0,1]^2$ den Rand des Rechengebietes $[0,1]^2$ bezeichnet. Das heißt Dirichlet-Randbedingungen geben am Rand die Werte der Lösung vor. Eine weitere häufig verwendete Art von Randbedingungen sind Neumann-Randbedingungen [Meh18], welche die Ableitung in Richtung der Normale des Randes vorgeben. Um die Randbedingungen bei der Kollokation überhaupt miteinbeziehen zu können, müssen auf dem Rand Stützstellen liegen. Die einfachste Möglichkeit ist es, die Stützstellen äquidistant auf einem regulären Gitter zu verteilen, wie es in Abbildung 3.5 dargestellt ist. Die rot eingefärbten Stützstellen werden dann für die Randbedingungen verwendet und die restlichen für die eigentliche Differentialgleichung $\Delta u = 0$. Setzen wir hier die Darstellung von u in einer Basis $\{b_m \mid m \in M\}$ ein und nutzen die Linearität des Laplace-Operators Δ aus, erhalten wir für $0 \leq i, j \leq 4$ analog zu Gleichung (3.61)

$$\sum_{m \in M} \alpha_m \Delta b_m(x_i, y_j) = 0 \qquad \text{falls } 1 \leq i, j \leq 3, \tag{3.65}$$

$$\sum_{m \in M} \alpha_m b_m(x_i, y_j) = u_D(x_i, y_j) \qquad \text{sonst.} \tag{3.66}$$

Da wir in diesem Fall mit $5 \cdot 5 = 25$ Stützstellen auch 25 Bedingungen haben, sollte auch der Funktionsraum, in dem wir die Lösung suchen eine Dimension von 25 haben, damit sich ein quadratisches LGS der Seitenlänge 25 ergibt. Nach Gleichung (3.46) bietet es sich also an, in beiden Dimensionen jeweils einen fünfdimensionalen Raum R_1 respektive R_2 zu nehmen und die Lösung in deren Tensorproduktraum $R_1 \otimes R_2$ zu suchen. Da wir *Not-a-Knot* B-Splines verwenden wollen, wählen wir

$$R_1 = R_2 = S_{n,s^{\text{NAK}}}. \tag{3.67}$$

Dies ist nach Gleichung (3.34) ein fünfdimensionaler Raum. Sei nun $B_1 = B_2$ die zugehörige B-Spline-Basis. Als Basis B aus Gleichung (3.57), die wir für die Kollokation verwenden, können wir dann nach Lemma 3.4.1

$$B = \{b_1 \otimes b_2 \mid b_1 \in B_1, b_2 \in B_2\} \tag{3.68}$$

verwenden.

R_1 und R_2 sowie B_1 und B_2 waren in diesem Beispiel lediglich deshalb gleich, weil wir ein quadratisches reguläres Gitter verwendet haben. Hat das Gitter nicht in jeder Dimension gleich viele Gitterpunkte, werden in den unterschiedlichen Dimensionen unterschiedliche B-Spline-Basen verwendet.

3.7 Volle und dünne Gitter

Möchte man hochdimensionale Probleme auf regulären Gittern wie in Abbildung 3.5 lösen, spürt man schnell den sogenannten Fluch der Dimensionalität. Die Anzahl der Gitterpunkte auf regulären Gittern steigt exponentiell mit der Dimensionalität des Raumes, denn bei n Gitterpunkten pro Dimension und d Dimensionen ist die Gesamtanzahl der Gitterpunkte n^d. Die Wahl der Stützstellenanzahl, ist ein Kompromiss zwischen weniger Rechenzeit durch weniger Stützstellen und mehr Genauigkeit durch mehr Stützstellen. Aber ist ein reguläres Gitter diesbezüglich überhaupt die optimale Möglichkeit, die Stützstellen im Raum zu verteilen? Nein, denn bei der Interpolation

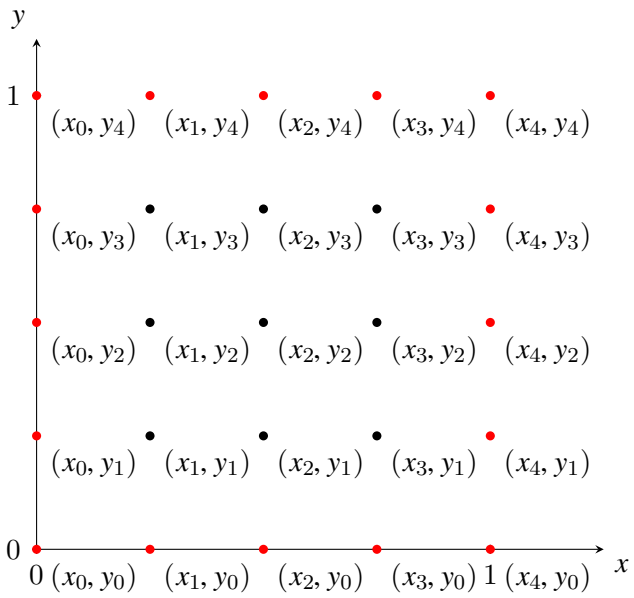


Abbildung 3.5: Gleichmäßiges Gitter mit 5 Gitterpunkten pro Dimension

ausreichend Glatter Funktionen hat sich herausgestellt, dass man unter Verwendung von sogenannten dünnen Gittern mit wesentlich weniger Stützstellen ein annähernd gleichwertiges Ergebnis erhält [BG04; Bun11].

Wir gehen im Folgenden auf die Konstruktion eines dünnen Gitters ein. Eine genauere Einführung findet sich in [Gar12]. Da dünne Gitter – wie in Abbildung 3.7 dargestellt – eine Kombination aus mehreren vollen Gittern sind, formalisieren wir zuerst Letztere.

Definition 3.7.1
Sei $l \in \mathbb{N}_0$, dann bezeichnet

$$V_l := \left\{ \frac{i}{2^l} \mid 0 \leq i \leq 2^l \right\} \tag{3.69}$$

das eindimensionale reguläre Gitter mit $2^l + 1$ Gitterpunkten.

Wir beschäftigen uns in dieser Arbeit nur mit dem Rechengebiet $[0, 1]^d$, wobei d die Dimensionalität ist. Deshalb liegen die Gitterpunkte in diesem Bereich.

Definition 3.7.2
Wir schreiben

$$\begin{aligned} W_0 &:= V_0, \\ W_l &:= V_l \setminus V_{l-1} \quad \forall l \geq 1. \end{aligned} \tag{3.70}$$

Mehrdimensionale Gitter sind dann das kartesische Produkt mehrerer eindimensionaler regulärer Gitter.

Definition 3.7.3
Sei $\bar{l} = (l_1, \ldots, l_d) \in \mathbb{N}_0^d$ ein d-stelliger Multiindex. Dann bezeichnet

$$V_{\bar{l}} := \bigtimes_{k=1}^{d} V_{l_k}, \tag{3.71}$$

$$W_{\bar{l}} := \bigtimes_{k=1}^{d} W_{l_k} \tag{3.72}$$

das Tensorgitter von $V_{l_1}, \ldots, V_{l_d}$ respektive $W_{l_1}, \ldots, W_{l_d}$. Wir nennen diese volle Gitter, und falls $l_1 = l_2 = \cdots = l_d$ gilt, reguläre Gitter.

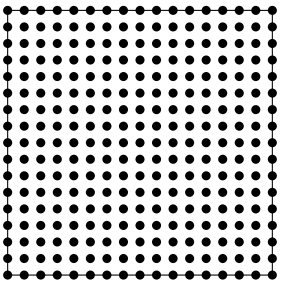


Abbildung 3.6: Gleichmäßiges Gitter $V_{(4,4)}$

Definition 3.7.4
Sei $d \in \mathbb{N}_1$ und $q \in \mathbb{N}_0$, dann bezeichnet

$$SG_q^d := \bigcup_{\substack{\bar{l} \in \mathbb{N}_0^d \\ \|\bar{l}\|_{\ell^1} \leq q}} W_{\bar{l}} \tag{3.73}$$

das d-dimensionale dünne Gitter vom Level q, wobei $\|\bar{l}\|_{\ell^1} = \sum_{k=1}^{d} l_k$ die Summennorm bezeichnet.

In Abbildung 3.7 ist zum Beispiel die Zusammensetzung des SG_4^2 dargestellt und man sieht, dass gegenüber dem regulären Gitter $V_{(4,4)}$ aus Abbildung 3.6 einige Gitterpunkte eingespart wurden. Nach [Gar12] enthält ein dünnes Gitter SG_q^d asymptotisch

$$|\mathrm{SG}_q^d| \in O\left(2^q \cdot q^{d-1}\right) = O\left(h^{-1} \cdot \log(h^{-1})^{d-1}\right) \tag{3.74}$$

Gitterpunkte, wobei $h = 2^{-q}$ die Gitterweite an der höchstaufgelösten Stelle ist. Dies sind wesentlich weniger als die $\left(h^{-1}+1\right)^d$ Gitterpunkte eines regulären Gitters. Um Probleme auf dünnen Gittern zu lösen, kann man nun eine Basis konstruieren, die je Gitterpunkt eine Basisfunktion enthält. Die hierarchische Basis aus [Gar12] ist eine solche.

3.8 Kombinationstechnik

Eine weitere Möglichkeit zur Lösung von Dünngitterproblemen ist die Kombinationstechnik [Gar12; GSZ92]. Wir werden sie hier erst für den Fall der Interpolation beschreiben und in Kapitel 4 auf die Kollokation verallgemeinern. Zunächst betrachten wir folgende Vorgehensweise aus [Gar12] zur

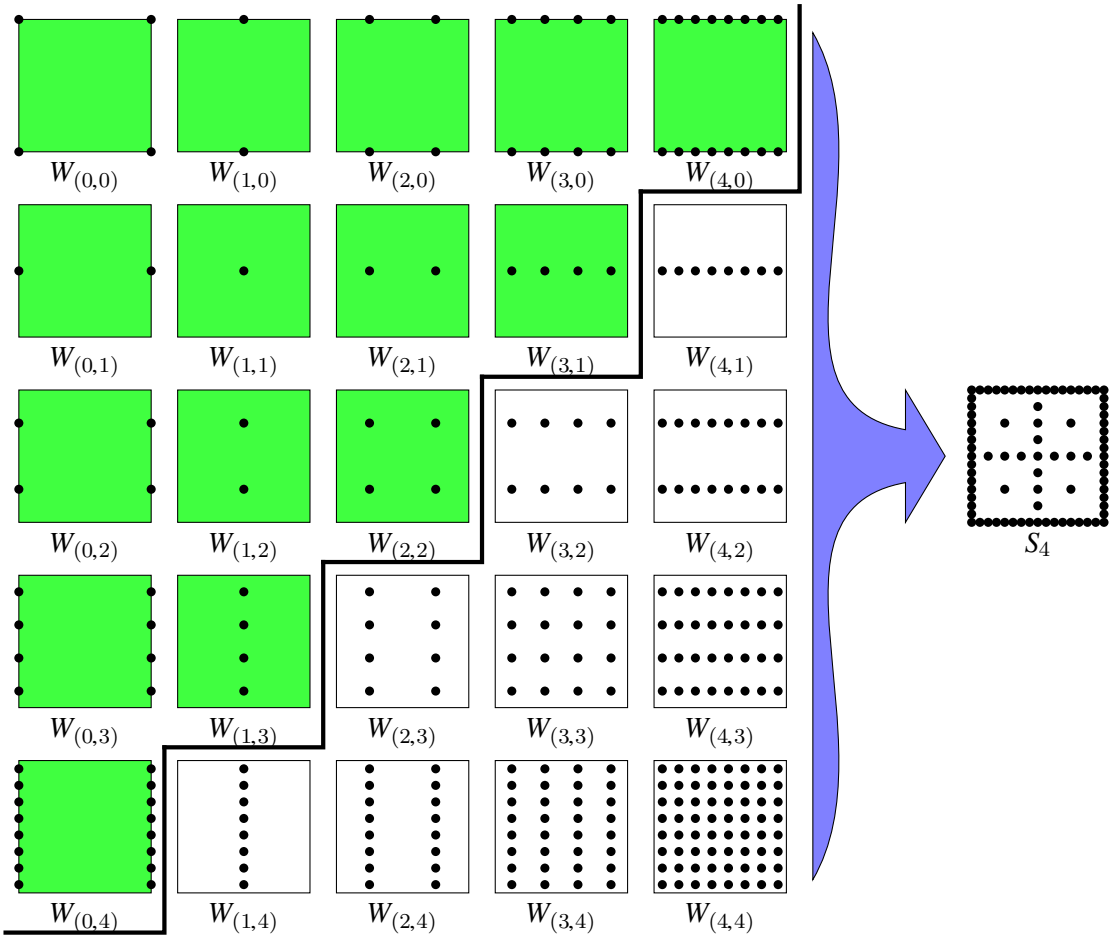


Abbildung 3.7: Dünnes Gitter SG_4^2

Lösung eines Interpolationsproblems auf S_q^d: Wir nehmen alle Gitter, aus denen S_q^d zusammengesetzt ist; das sind nach Definition 3.7.4 alle $W_{\bar{l}}$ mit $\bar{l} \in \mathbb{N}_0^d$ und $\|\bar{l}\|_{\ell^1} \leq q$. Bezüglich jedem dieser Gitter $W_{\bar{l}}$ berechnen wir eine Teillösung $w_{\bar{l}}$, die auf allen $x \in W_{\bar{l}}$ das Residuum der Summe der vorherigen Teillösungen interpoliert und außerdem

$$w_{\bar{l}}(x) = 0 \quad \forall x \in V_{\bar{l}} \setminus W_{\bar{l}} \tag{3.75}$$

erfüllt. Wenn man diese Teillösung auf die Summe der vorherigen Teillösung addiert, ändert sich deren Residuum an letzteren Stellen folglich nicht. Die Gesamtlösung ist dann die Summe der Teillösungen

$$s_q^d := \sum_{\substack{\bar{l} \in \mathbb{N}_0^d \\ \|\bar{l}\|_{\ell^1} \leq q}} w_{\bar{l}}. \tag{3.76}$$

Damit die Lösung s_q^d an den Stützstellen beweisbar exakt interpoliert, wurde in [Gar12] eine hierarchische Hutbasis verwendet, mit der $w_{\bar{l}}(x)$ für $x \in V_{\bar{l}} \setminus W_{\bar{l}}$ automatisch null ist. Die Kombinationstechnik ist nun eine Technik, mit der s_q^d berechnet werden kann, ohne die $w_{\bar{l}}$ zu berechnen:

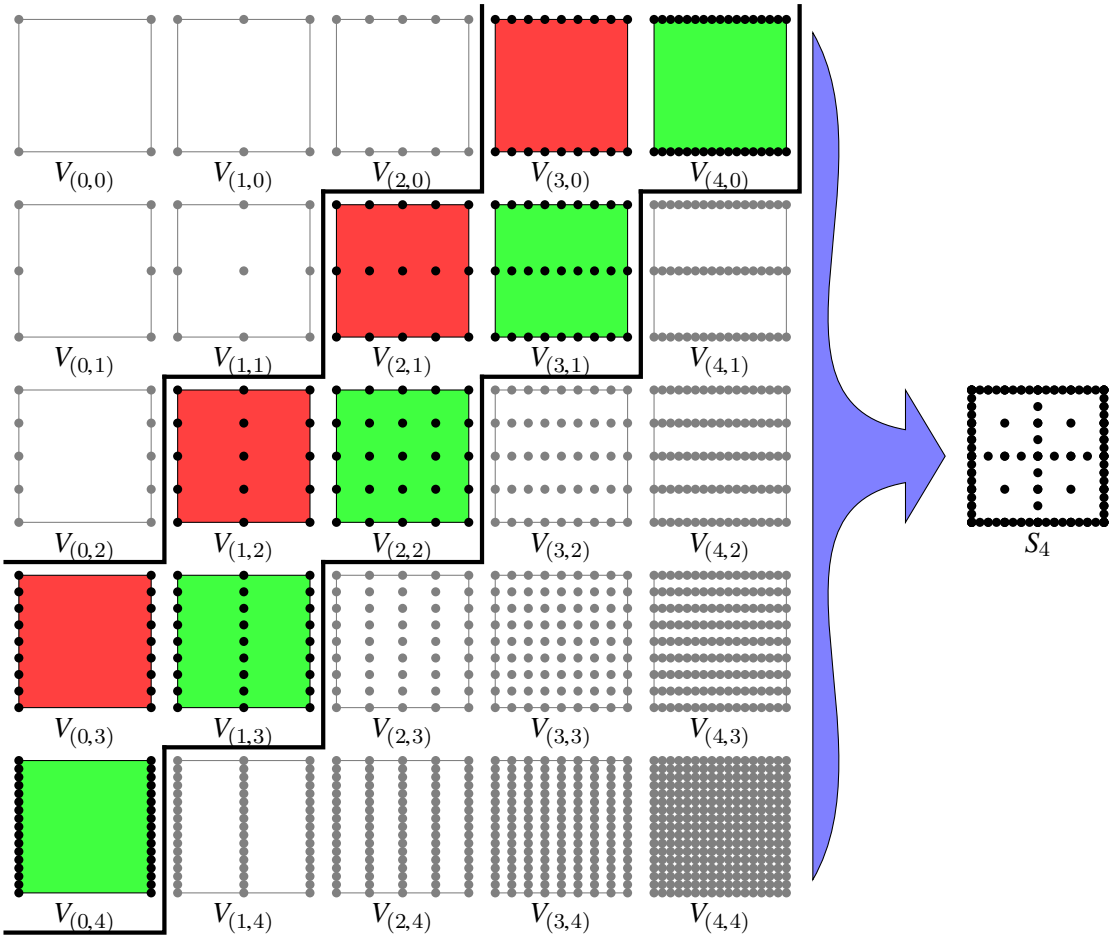


Abbildung 3.8: Kombinationstechnik für SG_4^2

Lemma 3.8.1
Sei $\bar{l} \in \mathbb{N}_0^d$ und $v_{\bar{l}}$ eine Lösung auf $V_{\bar{l}}$. Mit s_q^d aus Gleichung (3.76) *gilt dann*

$$s_q^d = \sum_{k=0}^{d-1} (-1)^k \binom{d-1}{k} \sum_{\substack{\bar{l} \in \mathbb{N}_0^d \\ \|\bar{l}\|_{\ell^1} \leq q-k}} v_{\bar{l}}. \tag{3.77}$$

Ein Beweis findet sich ebenfalls in [Gar12]. Nach diesem Schema berechnet man einfach Lösungen für die vollen Gitter $V_{\bar{l}}$ mit $q - d < \|\bar{l}\|_{\ell^1} \leq q$ und kombiniert diese wie in Lemma 3.8.1 angegeben. Das Schema ist darüber hinaus prädestiniert für eine parallele Berechnung. Abbildung 3.8 stellt sie für $d = 2$ Dimensionen und Level $q = 4$ dar. Mit ihr müssen augenscheinlich weniger Teillösungen berechnet werden als mit Gleichung (3.76).

4 Problemstellung

Im Rahmen dieser Arbeit implementieren wir die B-Spline-Kollokation mit der Kombinationstechnik (BSKK) in das Software-Paket SG++ [SGpp]. Dazu müssen wir zuerst die Kombinationstechnik auf die Kollokation verallgemeinern.

4.1 Kollokation mit der Kombinationstechnik

Das Entscheidende ist dabei, dass bei der Kombinationstechnik lediglich Lösungen auf vollen Gittern kombiniert werden. Wie wir die Lösung einer partiellen Differentialgleichung $\mathcal{L}u = f$ auf einem vollen Gitter berechnen können, wissen wir bereits seit Abschnitt 3.6. Diese Lösungen können wir problemlos nach dem Schema der Kombinationstechnik kombinieren. Die Vorgehensweise ist dabei wie folgt: Auf Eingabe einer partiellen Differentialgleichung und von Randbedingungen, wird mittels der B-Spline-Kollokation auf mehreren vollen Gittern jeweils eine Lösung berechnet. Diese Lösungen werden anschließend mit der Kombinationstechnik kombiniert, um eine approximative Lösung auf einem dünnen Gitter zu erhalten. Dementsprechend wird durch den Aufbau des dünnen Gitters bestimmt, auf welchen vollen Gittern zuvor eine Lösung berechnet wird.

Nach [Gar12] ist die Kombinationstechnik genau dann eine exakte Projektion in den Dünngitter-Raum, wenn die partiellen Projektionen auf die Funktionsräume der beteiligten vollen Gitter paarweise kommutieren. Das heißt es muss für alle beteiligten Gitter V_1, V_2

$$P_{V_1} P_{V_2} = P_{V_2} P_{V_1} \tag{4.1}$$

gelten, wobei jeweils P_V die Projektion auf den zum Gitter V gehörigen Funktionsraum bezeichnet [HGC06]. Mit anderen Worten: Die beweisbare Exaktheit der Lösung an den Stützstellen geht im Allgemeinen verloren [Gar12]. Zum Beispiel erfüllt die Poisson-Gleichung $-\Delta u = f$ diese Bedingung nicht [HGC06]. Für sie erhalten wir aber dennoch eine gute Approximation [BGRZ94]. In Abschnitt 4.2 stellen wir einen Ansatz vor, diese Approximation zu verbessern.

4.2 Iterative Verbesserung

Da die bei der BSKK erhaltene Dünngitterlösung im Allgemeinen nicht exakt ist, sondern nur eine Approximation liefert, untersuchen wir folgenden Algorithmus zur iterativen Verbesserung dieser Approximation: Als erstes wird mittels der BSKK eine erste Approximation u_0 für $\mathcal{L}u = f$ berechnet. Dann berechnen wir ebenfalls der mittels BSKK, anfangend bei $i = 1$, eine Approximation v_i für

$$\mathcal{L}v = f - \mathcal{L}u_{i-1} \tag{4.2}$$

und setzen

$$u_i := u_{i-1} + v_i = u_0 + \sum_{k=1}^{i} v_k \quad \forall i \geq 1. \tag{4.3}$$

Sei $r_i := \mathcal{L}_i - f$ das zugehörige Residuum. Wir erwarten, dass r_i mit steigender Anzahl an Iterationen i betragsmäßig signifikant abnimmt. Sollte diese Erwartung erfüllt werden, erlaubte das uns, das Residuum an den Punkten des dünnen Gitters beliebig zu drücken. Der Algorithmus kann dann abgebrochen werden, sobald $|r_i|$ an den Gitterpunkten eine vordefinierte Schwelle unterschreitet. Es stellt sich außerdem die Folgefrage, ob dabei zusammen mit dem Residuum an den Gitterpunkten auch das Residuum zwischen den Gitterpunkten abnimmt, sprich, ob die Lösung sich allumfassend verbessert.

5 Implementierung

In diesem Kapitel beschreiben wir die Implementierung der BSKK im Software-Paket SG++ [SGpp] – einem Werkzeugkasten für dünne Gitter, geschrieben in C++. Das Fundament bilden dabei die Datenstrukturen, durch welche die Zutaten, mithilfe derer sich eine Kollokation auf einem vollen Gitter beschreiben lässt, in der Software repräsentiert werden. Dies umfasst die B-Splines (Abschnitt 5.1), partielle Differentialgleichungen (Abschnitt 5.2) und Randbedingungen (Abschnitt 5.3). Wie in Abschnitt 5.4 erläutert, wird dazu anschließend ein LGS für die Koeffizienten der Basisfunktionen aufgestellt und gelöst. Hierdurch erhalten wir eine Lösung der Kollokation auf dem verwendeten vollen Gitter. Mit den Lösungen auf unterschiedlichen anisotropen vollen Gittern lässt sich dann – wie in Abschnitt 4.1 bereits beschrieben – die Kombinationstechnik durchführen. Da sie in SG++ schon vorhanden war, gehen wir auf ihre Implementierung nicht weiter ein. Die mit der Kombinationstechnik erhaltene Lösung soll schließlich durch den in Abschnitt 4.2 vorgestellten iterativen Algorithmus verbessert werden, dessen Implementierung wir in Abschnitt 5.5 behandeln.

5.1 B-Splines

Ein Spline-Raum und die zugehörige B-Spline-Basis sind durch den Grad n und die Knoten τ eindeutig bestimmt. Lässt man dabei offen, welche Knoten verwendet werden, reden wir im Folgenden von einer Basisfamilie. Die B-Splines vom Grad 3 sind also ein Beispiel für eine Basisfamilie. Während es plausibel ist, bei der BSKK auf allen beteiligten vollen Gittern den selben Grad zu verwenden, sind die Knoten bei *Not-a-Knot* B-Splines abhängig von den Stützstellen, insbesondere von deren Anzahl. Deshalb haben wir uns entschieden, in der Software die Information über die Basisfamilie von der Information über die Stützstellen zu trennen.

Alle hier beschriebenen Klassen befinden sich dort im Combigrid-Modul mit dem Namensraum `sgpp::combigrid`. Zur Repräsentation einer Basisfamilie gibt es die polymorphische Klasse `UnaryBasis`, deren Name deutlich macht, dass es sich bei den zugehörigen Basisfunktionen noch nicht um Tensorprodukte handelt. `UnaryBasis` besitzt momentan eine einzige Implementierung namens `BSplines`. Hierfür verwendeten wir eine C++-Programmiertechnik namens *konzeptbasierter Polymorphie* [Par12; Par17], welche es ermöglicht, Objekte polymorphischer Klassen *by-value* zu übergeben, anstatt Zeiger zu verwenden. In Listing 5.1 wird solch ein `UnaryBasis`-Objekt konstruiert. Wie man dort auch sieht, benötigt man um eine Basis auszuwerten zusätzlich die Stützstellen, eine Auswertestelle und die Ableitungen, an denen man interessiert ist. In der `BSpline`-Implementierung werden aus den Stützstellen *Not-a-Knot* Knoten generiert und anschließend die zugehörigen B-Splines berechnet. Auf diese Weise ist die Anzahl der B-Splines gleich der Stützstellenanzahl. Da wir bei der Kollokation typischerweise die Werte der Ableitungen der B-Splines brauchen, gibt

es den Parameter `derivatives`, in dem man angibt, welche Ableitungen berechnet werden sollen. Für diejenigen B-Splines deren Träger die Auswertestelle enthält, sind nach der Ausführung die berechneten Werte im zurückgegebenen `BasisValueStorage`-Objekt gespeichert.

Listing 5.1 Auswertung von kubischen B-Splines

```
1 UnaryBasis basis = BSplines{3}; // Degree 3
2 std::vector<int32_t> derivatives{0}; // 0th derivative
3 std::vector<double> gridPoints{0, 0.25, 0.5, 0.75, 1};
4 double evalPoint = 0.35;
5 BasisValueStorage storage = basis.valuesAt(derivatives, gridPoints, evalPoint);
```

Aus der Berechnungsvorschrift für die B-Splines (Definition 3.3.1) rekapitulieren wir

$$\begin{aligned} B_{i,0}(x) &:= \begin{cases} 1 & \text{falls } x \in \tau[i] \\ 0 & \text{sonst,} \end{cases} \\ B_{i,n}(x) &:= \frac{x-\tau_i}{\tau_{i+n}-\tau_i} B_{i,n-1}(x) + \frac{\tau_{i+n+1}-x}{\tau_{i+n+1}-\tau_{i+1}} B_{i+1,n-1}(x) \quad \text{für } n \geq 1. \end{aligned} \tag{5.1}$$

Dementsprechend können wir stets aus zwei konsekutiven B-Splines vom Grad $n-1$ einen B-Spline vom Grad n berechnen, wie es in Abbildung 5.1 skizziert ist. Sind wir an Werten an einer bestimmten Stelle $\tilde{x}$ interessiert, müssen nach dieser Vorschrift die dafür benötigten B-Splines niedrigerer Grade nur an $\tilde{x}$ ausgewertet werden. Dies gibt uns Spielraum zur Optimierung. Wegen $B_{i,0}(\tilde{x}) = 0 \Leftrightarrow \tilde{x} \notin \tau[i]$ gibt es für jede Stelle $\tilde{x}$ nur einen konstanten B-Spline, der dort ungleich null ist. Für $n \geq 1$ gilt: Falls $B_{i,n-1}(\tilde{x}) = B_{i+1,n-1}(\tilde{x}) = 0$ ist, dann ist auch $B_{i,n}(\tilde{x}) = 0$. Infolgedessen müssen die in Abbildung 5.1 ausgegrauten B-Splines nicht ausgewertet werden, um die Werte der kubischen B-Splines an der eingezeichneten Stelle $\tilde{x}$ zu berechnen. Bei der Kollokation brauchen wir auch die Werte der Ableitungen der B-Splines. Für diese gibt es die Rekursionsgleichung [DBSC]

$$\frac{d}{dx} B_{i,n}(x) = \frac{n}{\tau_{i+n}-\tau_i} B_{i,n-1}(x) - \frac{n}{\tau_{i+n+1}-\tau_{i+1}} B_{i+1,n-1}(x) \tag{5.2}$$

welche eine Berechnung nach ähnlichem Schema zulässt. Die Berechnungen wurden nicht rekursiv implementiert, sondern wir bedienten uns der Technik des dynamischen Programmierens [Bel57], um auch mehrfach benötigte Werte nur einmal zu berechnen.

Da *Not-a-Knot* B-Splines verwendet werden, muss nach Definition 3.3.2 der Grad ungerade sein. Momentan unterstützt die Software B-Splines der Grade 1, 3 und 5. Höhere Grade wurden bisher nicht gebraucht, da – wie wir in Abschnitt 3.3 gesehen haben – mit dem Grad der B-Splines auch deren Träger wächst. Bei Bedarf muss zur Unterstützung höherer Grade lediglich die `BSplines`-Implementierung angepasst werden. Alle anderen Komponenten sind unabhängig von der verwendeten Basis, weshalb prinzipiell beliebige Basen verwendet werden könnten.

Gehen wir nun in mehrere Dimensionen, müssen wir eine Tensorproduktbasis

$$\left\{ \bigotimes_{k=1}^{d} b_k \mid \forall\, 1 \leq k \leq d : b_k \in B_k \right\} \tag{5.3}$$

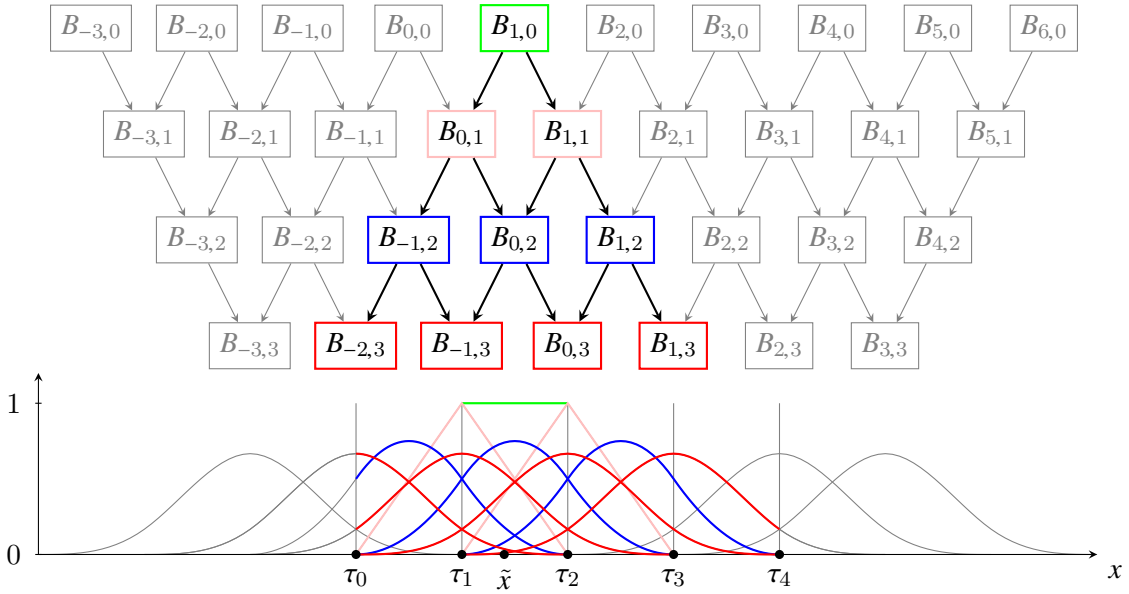


Abbildung 5.1: Berechnung der Werte kubischer B-Splines an der Stelle $\tilde{x} \in \tau[1]$

auswerten, wofür wir über alle Basisfunktionen iterieren könnten. Falls sie jedoch einen kleinen Träger haben, verschwinden die meisten Basisfunktionen an der Auswertestelle. Bei den B-Splines ist das der Fall und diese Vorgehensweise dementsprechend ineffizient. Auch hier können wir wieder optimieren: Wegen

$$\left(\bigotimes_{k=1}^{d} b_k\right)(x_1, x_2, \ldots, x_d) = \prod_{k=1}^{d} b_k(x_k) \tag{5.4}$$

und dem Satz vom Nullprodukt dürfen wir bereits unter den Basisfunktionen auf den eindimensionalen Gittern diejenigen aussortieren, die an der Auswertestelle verschwinden. Dies wird noch genauer in Abschnitt 5.4 erklärt.

5.2 Partielle Differentialgleichungen

Partielle Differentialgleichungen haben die Form

$$\mathcal{L}u = f, \tag{5.5}$$

mit einer gegebenen Funktion f, einer gesuchten Funktion u und einem Differentialoperator $\mathcal{L}$, der partielle Ableitungen enthält. Im zweidimensionalen Fall heißt das es gibt eine Funktion g, sodass

$$\mathcal{L}u(x,y) = g\left(x, y, u(x,y), u_x(x,y), u_y(x,y), u_{xx}(x,y), u_{xy}(x,y), u_{yx}(x,y), u_{yy}(x,y), \ldots\right) \tag{5.6}$$

gilt, wobei wir hier die Notation

$$u_x := \frac{\partial}{\partial x} u \tag{5.7}$$

verwenden. Die ersten beiden Operanden x, y erlauben es dem Differentialoperator, ortsabhängig zu sein. Dadurch wird es möglich, kompliziertere partielle Differentialgleichungen – wie zum Beispiel die Konvektions-Diffusions-Gleichung [Sch06] – zu formulieren und zu lösen.

In der Software wollen wir bei der Auswertung von $\mathcal{L}u$ nicht alle möglichen Ableitungen von u berechnen, sondern nur diejenigen, von denen das Ergebnis tatsächlich abhängt. Zum Beispiel werden (in d Dimensionen) zur Auswertung von

$$\Delta u = \sum_{k=1}^{d} \frac{\partial^2}{\partial x_k^2} u \tag{5.8}$$

lediglich die zweiten partiellen Ableitungen von u benötigt. Aus diesem Grund gibt es die `DifferentialOperands`-Klasse, die beschreibt, welche partiellen Ableitungen ein Differentialoperator als Operanden hat. In Listing 5.2 konstruieren wir ein `DifferentialOperands`-Objekt, das die Operanden $\frac{\partial^2}{\partial x^2} u(x, y)$ (Zeile 3) und $\frac{\partial^2}{\partial y^2} u(x, y)$ (Zeile 4) beschreibt, also die Operanden des zweidimensionalen Laplace-Operators.

Listing 5.2 Konstruktion der partiellen Diffferentialgleichung $\Delta u(x, y) = -1$

```
1  int32_t dimensionality = 2;
2  DifferentialOperands operands{dimensionality};
3  operands.addOperand({2, 0});
4  operands.addOperand({0, 2});
5
6  auto g = [](VectorView<const double> point, VectorView<const double> operandValues) {
7    return operandValues[0] + operandValues[1]; // Add the values of the operands
8  };
9  DifferentialOperator laplaceOperator{operands, g};
10
11 auto f = [](VectorView<const double> point) { return -1.0; };
12 DifferentialEquation pde{laplaceOperator, f};
```

Zur Repräsentation von Differentialoperatoren gibt es die `DifferentialOperator`-Klasse, deren Objekte aus einem `DifferentialOperands`-Objekt sowie einem Funktor, der die Berechnungsvorschrift vorgibt, zusammengesetzt sind. Zur Auswertung eines Differentialoperators werden eine Auswertestelle sowie die Werte der Operanden an diesen Funktor übergeben, der daraufhin das Ergebnis zurückgibt. Der Funktor stellt also die Funktion g aus Gleichung (5.6) dar, mit dem Unterschied, dass er nur die Werte der benötigten Operanden erhält. In Listing 5.2 erhält er zwei Werte, deren Semantik wie gesagt durch das `DifferentialOperands`-Objekt bestimmt ist, und gibt deren Summe zurück.

Eine partielle Differentialgleichung $\mathcal{L}u = f$ wird schließlich durch ein `DifferentialEquation`-Objekt dargestellt, welches aus einem `DifferentialOperator`-Objekt und einem weiteren Funktor für die rechte Seite f besteht.

5.3 Randbedingungen

Wie in Abschnitt 3.5 erläutert, sind Differentialgleichungen oft nicht eindeutig lösbar. Um eine eindeutige Lösung zu erhalten, müssen wir typischerweise Randbedingungen spezifizieren und diese infolgedessen auch in der Software repräsentieren. Häufig verwendete Arten von Randbedingungen sind Dirichlet-Randbedingungen der Form

$$u(\vec{x}) = u_D(\vec{x}) \quad \text{für } \vec{x} \in \partial\Omega \tag{5.9}$$

und Neumann-Randbedingungen der Form

$$\frac{\partial}{\partial \vec{n}} u(\vec{x}) = u_N(\vec{x}) \quad \text{für } \vec{x} \in \partial\Omega, \tag{5.10}$$

wobei $\partial\Omega$ den Rand des Rechengebiets Ω und $\vec{n}$ dessen Normalenvektor an der Stelle $\vec{x}$ bezeichnet [Han02]. Diese Art von Randbedingungen sind also partielle Differentialgleichungen, welche durch die Lösung auf $\partial\Omega$ erfüllt sein sollen. Folglich können wir die in Abschnitt 5.2 beschriebene Repräsentation partieller Differentialgleichung nicht nur im Inneren des Rechengebietes verwenden, sondern auch für die Randbedingungen. Wir rechnen auf dem Gebiet $[0,1]^d$ und darauf wird ein Kollokationsproblem demnach durch eine partielle Differentialgleichung

$$\mathcal{L}_{\text{interior}} u = f_{\text{interior}} \tag{5.11}$$

im Inneren und in jeder Dimension $k \in \{1, \ldots, d\}$ jeweils durch zwei weitere partielle Differentialgleichungen

$$\begin{aligned} \mathcal{L}_{\text{lower},k} u &= f_{\text{lower},k} \quad \text{auf } \{\vec{x} \mid x_k = 0\}, \\ \mathcal{L}_{\text{upper},k} u &= f_{\text{upper},k} \quad \text{auf } \{\vec{x} \mid x_k = 1\} \end{aligned} \tag{5.12}$$

– das heißt auf dem unteren und oberen Rand in dieser Dimension – beschrieben. Solche Beschreibung werden in der Software durch die `BoundaryCollocationTask`-Klasse repräsentiert, die beispielhaft in Listing 5.3 verwendet wird. Dabei ist `task.interior()` die partielle Differentialgleichung $\mathcal{L}_{\text{interior}} u = f_{\text{interior}}$ im Inneren, `task.lowerBoundary(k)` die auf dem unteren Rand ($\mathcal{L}_{\text{lower},k} u = f_{\text{lower},k}$) und `task.upperBoundary(k)` die auf dem oberen Rand ($\mathcal{L}_{\text{upper},k} u = f_{\text{upper},k}$) in Dimension k. In diesem Beispiel ist

$$\mathcal{L}_{\text{interior}} u = \Delta u = -1 = f_{\text{interior}} \tag{5.13}$$

und es wurden homogene Dirichlet-Randbedingungen [Han02]

$$\begin{aligned} \mathcal{L}_{\text{lower},k} u = u = 0 = f_{\text{lower},k} \\ \mathcal{L}_{\text{upper},k} u = u = 0 = f_{\text{upper},k} \end{aligned} \tag{5.14}$$

für alle Dimensionen $k \in \{1, \ldots, d\}$ verwendet.

Listing 5.3 `BoundaryCollocationTask` für $\Delta u = -1$ mit homogenen Dirichlet-Randbedingungen

```
1 int32_t dimensionality = 2;
2 BoundaryCollocationTask task{dimensionality};
3 task.interior() = {laplaceOperator(dimensionality), fConstant(-1)};
4 task.lowerBoundary(0) = {identityOperator(dimensionality), fConstant(0)};
5 task.lowerBoundary(1) = {identityOperator(dimensionality), fConstant(0)};
6 task.upperBoundary(0) = {identityOperator(dimensionality), fConstant(0)};
7 task.upperBoundary(1) = {identityOperator(dimensionality), fConstant(0)};
```

Für unterschiedliche Dimensionen k überschneiden sich die Geltungsbereiche der Randbedingungen aus Gleichung (5.12) an allen Stellen, die in mehr als einer Dimension am Rand ist. Im Zweidimensionalen passiert das also an den Ecken und im Dreidimensionalen an den Ecken und Kanten. Zum Beispiel müsste eine Lösung an der Stelle $\vec{x} = \vec{0}$ alle Randbedingungen $\mathcal{L}_{\text{lower},k} u(\vec{x}) = f_{\text{lower},k}(\vec{x})$ erfüllen. Weil wir genau eine lineare Gleichung pro Stützstelle formulieren wollen, stellt das ein

Problem dar. Dieses lösen wir, indem wir die Gleichungen, die an einer solchen Stelle gelten sollen zu einer einzigen partiellen Differentialgleichung

$$\left(\sum_{\substack{k \in \{1,\ldots,d\} \\ x_k \in \{0,1\}}} \mathcal{L}_{\vec{x},k} \right) u = \sum_{k \in \{1,\ldots,d\}} f_{\vec{x},k} \tag{5.15}$$

mit

$$\mathcal{L}_{\vec{x},k} := \begin{cases} \mathcal{L}_{\text{lower},k} & \text{falls } x_k = 0, \\ \mathcal{L}_{\text{upper},k} & \text{falls } x_k = 1, \\ 0 & \text{sonst,} \end{cases} \qquad f_{\vec{x},k} := \begin{cases} f_{\text{lower},k} & \text{falls } x_k = 0, \\ f_{\text{upper},k} & \text{falls } x_k = 1, \\ 0 & \text{sonst} \end{cases} \tag{5.16}$$

summieren.

Möchte man aus einem vorhandenden `BoundaryCollocationTask`-Objekt die Differentialgleichung auslesen, welche an einem Gitterpunkt erfüllt sein soll, so hängt das Resultat davon ab, ob sich dieser Punkt im Inneren oder auf dem Rand befindet und, in letzterem Fall, auf was für einer Stelle des Randes. Deshalb gibt es die Methode `BoundaryCollocationTask::equation`, zu der sich ein Beispielcode in Listing 5.4 findet. Sie bekommt die Ausmaße eines Tensorgitters sowie den Multiindex eines Punktes darin übergeben und gibt die richtige Differentialgleichung für diesen Punkt zurück. Liegt der Punkt im Inneren oder auf einer Stelle des Randes, an der nur eine Randbedingung gültig ist, so ist der Rückgabewert die partielle Differentialgleichung, die für diese Stelle spezifiziert wurde. Anderenfalls wird die in Gleichung (5.15) aufgeschriebene Summe der entsprechenden Differentialgleichungen zurückgegeben; dargestellt durch ein neues `DifferentialEquation`-Objekt.

Listing 5.4 Auslesen der partiellen Differentialgleichung in einer Ecke; Fortsetzung von Listing 5.3

```
8  std::vector<int64_t> bounds{5, 5};
9  std::vector<int64_t> multiindex{0, 5};
10 DifferentialEquation pde = task.equation(bounds, multiindex);
```

5.4 Lineares Gleichungssystem

Eine vollständige Beschreibung einer Kollokation auf einem vollen Gitter besteht nun aus dem vollen Gitter selbst, einem `UnaryBasis`-Objekt pro Dimension um auf dem Gitter die Basisfunktionen vorzugeben sowie einem `BoundaryCollocationTask`-Objekt, welches das Problem beschreibt. Analog zu Definition 3.7.3 haben wir dabei pro Dimension $k \in \{1, \ldots, d\}$ ein eindimensionales reguläres Gitter V_{l_k}, deren kartesisches Produkt

$$V_{\vec{l}} := \bigtimes_{k=1}^{d} V_{l_k} \tag{5.17}$$

das volle Gitter bildet. Tatsächlich unterstützen wir für die hier verwendeten eindimensionalen Gitter nicht nur reguläre, sondern beliebige. Auf Eingabe dieser Daten können wir das zugehörige LGS aufstellen und lösen.

Dafür ist es notwendig, an jedem Gitterpunkt einen Differentialoperator auszuwerten und deshalb überlegen wir uns zuerst, wie das am effizientesten geschieht. Die gegebenen `UnaryBasis`-Objekte definieren auf dem vollen Gitter für jede Dimension $k \in \{1, \ldots, d\}$ eine Basis

$$B_k =: \{b_{k,i} \mid i \in I_k\}. \tag{5.18}$$

Da wir die Lösung u als Linearkombination

$$u = \sum_{i_1 \in I_1} \sum_{i_2 \in I_2} \cdots \sum_{i_d \in I_d} \alpha_{\vec{i}} \bigotimes_{k=1}^{d} b_{k,i_k} \tag{5.19}$$

darstellen, lässt sich die partielle Differentialgleichung $\mathcal{L}u = f$ mit linearem Differentialoperator $\mathcal{L}$ umformen zu

$$\mathcal{L}u = \sum_{i_1 \in I_1} \sum_{i_2 \in I_2} \cdots \sum_{i_d \in I_d} \alpha_{\vec{i}} \mathcal{L} \bigotimes_{k=1}^{d} b_{k,i_k} = f. \tag{5.20}$$

Die zur Auswertung des Differentialoperators benötigten Operanden haben dann die Form

$$\begin{aligned} \frac{\partial^{a_1}}{\partial x_1^{a_1}} \frac{\partial^{a_2}}{\partial x_2^{a_2}} \cdots \frac{\partial^{a_d}}{\partial x_d^{a_d}} \left(\bigotimes_{k=1}^{d} b_{k,i_k} \right) (\vec{x}) &= \frac{\partial^{a_1}}{\partial x_1^{a_1}} \frac{\partial^{a_2}}{\partial x_2^{a_2}} \cdots \frac{\partial^{a_d}}{\partial x_d^{a_d}} \prod_{k=1}^{d} b_{k,i_k}(x_k) \\ &= \prod_{k=1}^{d} \frac{\partial^{a_k}}{\partial x_k^{a_k}} b_{k,i_k}(x_k), \end{aligned} \tag{5.21}$$

wobei in jeder Dimension $k \in \{1, \ldots, d\}$ die Koordinate x_k nur die Werte des für diese Dimension gegebenen eindimensionalen Gitters annehmen kann. Demnach können wir zuerst die Werte aller benötigten Ableitungen berechnen und sie später zusammen multiplizieren. Das gegebene `BoundaryCollocationTask`-Objekt gibt uns mit den enthaltenen Gleichungen Auskunft darüber, in welcher Dimension wir welche Ableitungen der eindimensionalen Basisfunktionen benötigen. Daraufhin lassen wir an allen Gitterpunkten der eindimensionalen Gitter, mithilfe des für diese Dimension gegebenen `UnaryBasis`-Objekts, diese Ableitungen berechnen.

Um mithilfe dieser Werte das LGS aufzustellen, iterieren wir anschließend über jeden Punkt des vollen Gitters und fügen jeweils eine Zeile zum – anfangs leeren – LGS hinzu, welche die zu diesem Gitterpunkt gehörige Bedingung beschreibt. Mit der Methode `BoundaryCollocationTask::equation` lesen wir jeweils die am Gitterpunkt – im Folgenden als $\vec{x}$ bezeichnet – spezifizierte Differentialgleichung $\mathcal{L}u = f$ aus. Dann ist unter Verwendung von Gleichung (5.20)

$$\sum_{i_1 \in I_1} \sum_{i_2 \in I_2} \cdots \sum_{i_d \in I_d} \alpha_{\vec{i}} \left(\mathcal{L} \bigotimes_{k=1}^{d} b_{k,i_k} \right) (\vec{x}) = f(\vec{x}) \tag{5.22}$$

die zugehörige Bedingung, die wir zum LGS hinzufügen müssen. Die Einträge haben also die Form

$$\left(\mathcal{L} \bigotimes_{k=1}^{d} b_{k,i_k} \right) (\vec{x}). \tag{5.23}$$

Hier reicht es also, diejenigen eindimensionalen Basisfunktionen b_{k,i_k} zu betrachten, deren Träger die Auswertestelle enthält, denn mit Gleichung (5.21) und dem Satz vom Nullprodukt ist ersichtlich, dass der Eintrag sonst verschwindet. Um die Werte der nicht verschwindenden Einträge zu erhalten,

nutzen wir die zuvor berechneten Ableitungen und berechnen nach dem Schema von Gleichung (5.21) die Werte der für $\mathcal{L}$ benötigten Operanden. Damit werten wir daraufhin den Differentialoperator aus.

Da die Stützstellenanzahl gleich der Anzahl an Einträgen in einer Zeile des LGS ist, erhält man mit dieser Vorgehensweise ein quadratisches LGS. Dies ist notwendig dafür, dass das LGS weder über-, noch unterbestimmt ist [Han02]. Die eindeutige Lösbarkeit ist jedoch nicht garantiert, sondern hängt von der Problemstellung ab. Unlösbare respektive nicht eindeutig lösbare Kollokationsprobleme führen natürlicherweise zu einem unlösbaren respektive nicht eindeutig lösbaren LGS. Der Nutzer der Software steht in der Verantwortung, das Problem so zu formulieren, dass es eindeutig lösbar ist. Tut er das nicht, ist es vom verwendeten LGS-Löser abhängig, ob dieser einen Fehler, eine Warnung oder nichts dergleichen generiert. Die Lösung des LGS kann nämlich mit unterschiedlichen Lösern geschehen. Für die in Kapitel 6 vorgestellten Ergebnisse haben wir die C++-Bibliothek Eigen [Eigen] verwendet.

5.5 Iterative Verbesserung

In Abschnitt 4.2 hatten wir einen iterativen Algorithmus zur Verbesserung des Residuums folgendermaßen beschrieben: Als erstes soll mittels BSKK eine Approximation u_0 für $\mathcal{L}u = f$ berechnet werden. Anfangend bei $i = 1$ soll anschließend eine Approximation v_i für

$$\mathcal{L}v = f - \mathcal{L}u_{i-1} \tag{5.24}$$

berechnet werden, wobei

$$u_i := u_0 + \sum_{k=1}^{i} v_k \quad \forall i \geq 0 \tag{5.25}$$

gesetzt ist. Die Differentialgleichungen auf den Rändern werden dabei analog zu Gleichung (5.24) behandelt.

Wir können das aber noch vereinheitlichen: Sei $u_{-1} = 0$ die konstante Nullfunktion auf dem Rechengebiet. Dann können wir eine Iteration durchführen, das heißt wir berechnen mittels der BSKK eine Approximation v_0 für

$$\mathcal{L}v = f - \mathcal{L}u_{-1}. \tag{5.26}$$

Wegen $u_{-1} = 0$ ist das Ergebnis genau $v_0 = u_0$ und folglich gilt

$$u_i = \sum_{k=0}^{i} v_k \quad \forall i \geq 0. \tag{5.27}$$

Der Algorithmus ist also äquivalent, wenn man mit der konstanten Nullfunktion als erste Approximation anfängt und dafür eine Iteration mehr durchführt. Dies ist nicht mehr Rechenaufwand, da lediglich die Berechnung der ersten Approximation durch eine Iteration ersetzt wurde. Die Implementierung fiel uns aber in dieser Variante leichter, da wir nur die Iteration implementieren mussten. Tatsächlich gibt es in der Software nur den iterativen Algorithmus. Falls man die iterative Verbesserung nicht verwenden möchte, führt man eben nur eine Iteration durch.

Es stellt sich noch die Frage, wie wir die Summe mehrerer Lösungen $u_i = \sum_{k=0}^{i} v_k$ effizient in der Software darstellen können. Die v_k sind jeweils eine Kombination von mehreren Lösungen auf vollen Gittern. Eine Summe der Gesamtlösung erhalten wir, indem wir die Lösungen auf den vollen Gittern summieren. Dies funktioniert rundheraus, denn dafür müssen wir lediglich die Koeffizienten, aus denen eine Lösung auf einem vollen Gitter besteht, summieren. In jeder Iteration kommt ein neues v_k dazu, dessen Koeffizienten zu den Summen beitragen. Nach dem Summieren kann die Lösung v_k wieder gelöscht werden, sodass nie mehr Speicherplatz als für zwei Lösungen benötigt wird. Außerdem geht das Auswerten der Summe genauso schnell wie bei einer einzelnen Lösung.

6 Visualisierung und Auswertung

6.1 Interpolation

Zur Validation starten wir im Eindimensionalen mit dem einfachsten Spezialfall der Kollokation, der Interpolation. Das eindimensionale dünne Gitter S_l^1 entspricht nach Definition 3.7.4 dem regulären Gitter V_l aus Definition 3.7.1. Mit dünnen Gittern oder der Kombinationstechnik hat das also noch nichts zu tun. Bei der Interpolation soll an den Stützstellen

$$\mathrm{id}u = f \tag{6.1}$$

gelten, wobei u der Interpolant ist. Das ist eine partielle Differentialgleichung mit der Identität als Differentialoperator. Da wir in einem Spline-Raum $S_{n,\tau}$ interpolieren, und dieser den Polynomraum $\Pi_n \subset S_{n,\tau}$ enthält, muss die Interpolation von Polynomen vom Grad höchstens n exakt sein.

Zunächst benötigen wir eine Metrik, mit der wir die Qualität des Interpolanten messen. Dazu verteilen wir äquidistante Stellen $x_1, \ldots, x_m$ auf dem Rechengebiet – wir verwendeten $m = 150$ – und berechnen jeweils das Residuum

$$r_i := r(x_i) = \mathrm{id}u(x_i) - f(x_i) = u(x_i) - f(x_i). \tag{6.2}$$

Die zwei im Folgenden verwendeten Metriken sind die Maximumsnorm

$$\|r\|_{\max} := \max\{|r_i| \mid 1 \le i \le m\} \tag{6.3}$$

und die ℓ^2-Norm

$$\|r\|_{\ell^2} := \sqrt{\sum_{i=1}^{m} r_i^2} \tag{6.4}$$

des Vektors $r := (r_i)_{i=1,\ldots,m}$.

Abbildung 6.1 zeigt die Interpolation des kubischen Polynoms

$$f(x) = x(x - 0.5)(x - 1) \tag{6.5}$$

mittels kubischer B-Splines auf unterschiedlichen Leveln. Das reguläre Gitter V_l vom Level l aus Definition 3.7.1 besitzt $2^l + 1$ Gitterpunkte. Nach Lemma 3.1.1 ist $\dim \Pi_3 = 4$ und deshalb benötigen wir mindestens 4 Stützstellen, um kubische Polynome exakt zu interpolieren. Da bei Level 1 nur $2^1 + 1 = 3$ Stützstellen verwendet werden, ist es nicht überraschend, dass f erst ab Level 2 exakt interpoliert wird. Analog zeigt Abbildung 6.2 die Interpolation des Polynoms

$$g(x) = x(x - 0.25)(x - 0.5)(x - 0.75)(x - 1) \tag{6.6}$$

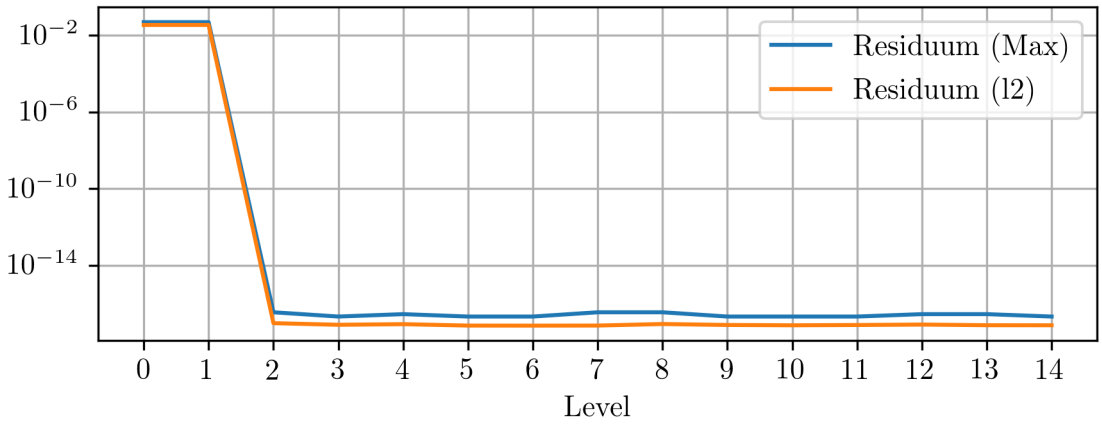


Abbildung 6.1: Residuen bei Interpolation von $f(x) = x(x - 0.5)(x - 1)$ mittels kubischer B-Splines

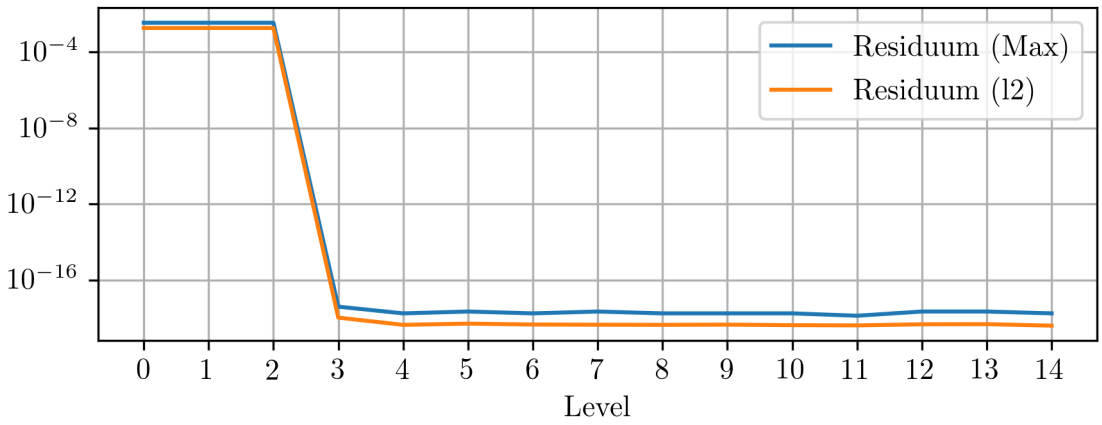


Abbildung 6.2: Residuen bei Interpolation von $g(x) = x(x - 0.25)(x - 0.5)(x - 0.75)(x - 1)$ mittels B-Splines vom Grad 5

mittels B-Splines desselben Grades; das heißt vom Grad 5. Da hier mindestens 6 Stützstellen benötigt werden, entspricht es der Erwartung, dass die Interpolation ab Level 3 mit $2^3 + 1 = 9$ Stützstellen exakt ist.

Eine exakte Interpolation ist nicht möglich, wenn die zu interpolierende Funktion nicht im Spline-Raum liegt, wie zum Beispiel

$$f(x) = \sin(10\pi x). \tag{6.7}$$

Die Auswertung der Interpolation dieser Funktion mittels B-Splines vom Grad 5 findet sich in Abbildung 6.3. Da sich f in der verwendeten Basis nicht exakt darstellen lässt, steigt die Qualität der Approximation erst mit steigendem Level, das heißt mit steigender Stützstellenanzahl. Interessanterweise tritt dieser Effekt erst ab Level 4 ein. Dies können wir mit dem Nyquist-Shannon-Abtasttheorem [Sha49] erklären: Zur exakten Rekonstruktion eines Signals, in dem Frequenzen bis

zu einer Frequenz $f_{\max}$ vorkommen, ist eine Abtastrate von mehr als $2f_{\max}$ notwendig und hinreichend. Wegen $f(x) = \sin(5 \cdot 2\pi x)$ ist die einzige enthaltene Frequenz $f_{\max} = 5$. Daher brauchen wir eine Abtastrate größer als 10. Da wir von den beiden Stützstellen, die auf dem Rand des Intervalls $[0, 1]$ liegen, nur eine zur Abtastrate zählen dürfen, brauchen wir also mindestens 12 Stützstellen, was wir erstmals bei Level 4 mit $2^4 + 1 = 17$ Stützstellen erreichen. Bei niedrigeren Leveln tritt *Unterabtastung* auf. Zwar geht es hier nicht um eine exakte Rekonstruktion, aber dennoch ist es nicht verwunderlich, dass bei einer Unterabtastung keine qualitativ hochwertige Lösung zustande kommt. Des Weiteren ist zu beobachten, dass die ℓ^2-Norm des Residuums in beiden Fällen nicht unter 10^{-16} sinkt. Dies liegt in der Größenordnung der technisch bedingten numerischen Ungenauigkeit. Wir verwendeten Fließkommazahlen mit doppelter Genauigkeit nach Norm IEEE 754 [I75408]. Diese haben eine Mantisse mit 53 Bit, was knapp 16 Dezimalstellen entspricht.

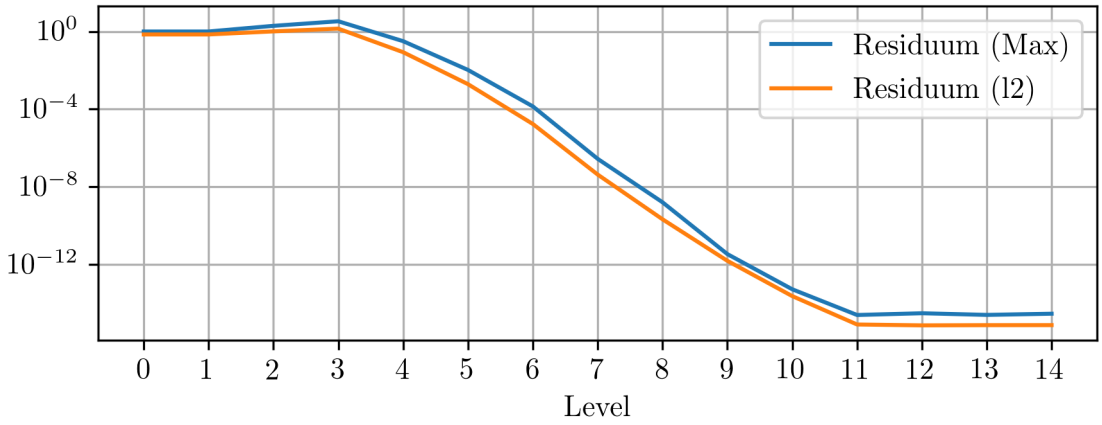


Abbildung 6.3: Residuen bei Interpolation von $f(x) = \sin(10\pi x)$ mittels B-Splines vom Grad 5

Dasselbe können wir in mehreren Dimensionen probieren. Um eine Unterabtastung zu verhindern, wählen wir diesmal eine Funktion mit einer niedrigeren Frequenz, und zwar

$$f(x, y, z) = \sin(\pi x) \cdot \sin(\pi y) \cdot \sin(\pi z). \tag{6.8}$$

Das ist also der Sinus von π mal dem Abstand zum Ursprung. Die Metriken der Interpolation dieser Funktion finden sich in Abbildung 6.4. Die Genauigkeit verbessert sich ab Level 3, welches im Dreidimensionalen das erste Level mit einem Gitterpunkt im Inneren ist. Wir beobachten, dass die ℓ^2-Norm sowie das Maximum der Residuen im Dreidimensionalen mit steigendem Level etwas langsamer fallen als im Eindimensionalen.

6.2 Eindimensionale Kollokation

Kommen wir nun zur Kollokation. Auch hier starten wir wieder im Eindimensionalen mit der Differentialgleichung

$$-\frac{\partial^2}{\partial x^2} u(x) = 1. \tag{6.9}$$

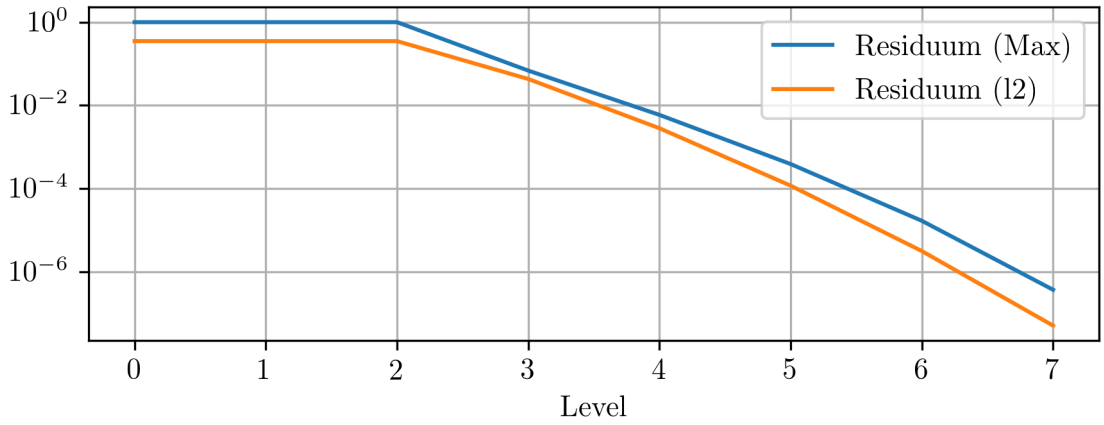


Abbildung 6.4: Residuen bei Interpolation von $f(x, y, z) = \sin(\pi x) \cdot \sin(\pi y) \cdot \sin(\pi z)$ auf dünnen Gittern unterschiedlicher Level (mittels B-Splines vom Grad 5)

Dies ist der eindimensionale Fall der Poisson-Gleichung aus Gleichung (3.49) mit rechter Seite $f(x) = 1$. Am Rand verwenden wir homogene Dirichlet-Randbedingungen

$$u(0) = u(1) = 0. \tag{6.10}$$

Die analytische Lösung dieses Problems ist eine nach unten geöffnete Parabel mit den Nullstellen 0 und 1, wie sie in Abbildung 6.5 zu sehen ist. Tatsächlich handelt es sich dabei um eine Abbildung der Lösung, die wir bei der Kollokation mit B-Splines vom Grad 5 auf dem regulären Gitter des Levels 1 erhielten, also mit $2^1 + 1 = 3$ Stützstellen. Da die Lösung in diesem Fall exakt darstellbar ist, reicht diese Stützstellenanzahl.

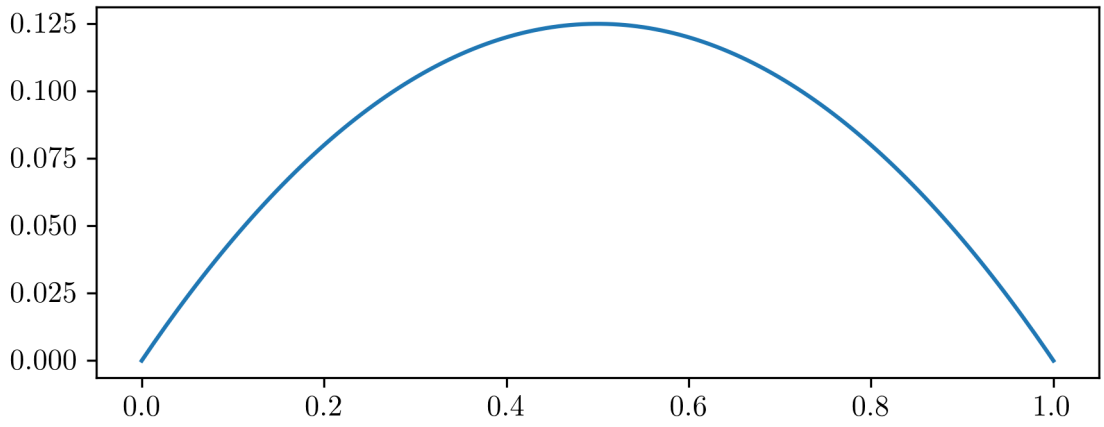


Abbildung 6.5: Lösung der Gleichungen (6.9) und (6.10) auf regulärem Gitter des Levels 1 (mittels B-Splines vom Grad 5)

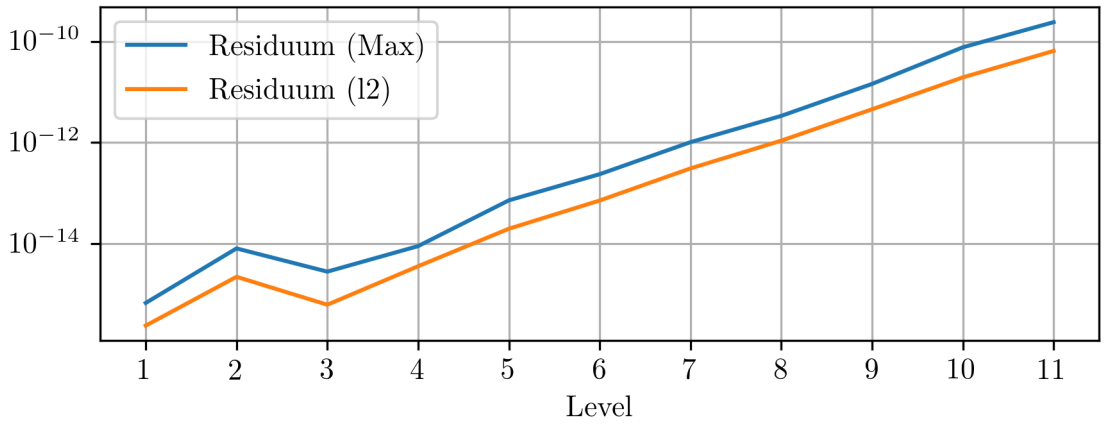


Abbildung 6.6: Residuen beim Lösen der Gleichungen (6.9) und (6.10) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

Abbildung 6.6 zeigt die Metriken dieser Kollokation bei unterschiedlichen Leveln. Aber wieso sinkt mit steigendem Level die Genauigkeit? Hierfür bieten wir folgende Erklärung an: Die inneren Stützstellen tragen mit Zeilen der Form

$$\sum_{i \in I} \alpha_i \frac{\partial^2}{\partial x^2} b_i(x) = f(x) \tag{6.11}$$

zum LGS bei. Die Koeffizienten sind also zweite Ableitungen. Mit jeder Inkrementierung des Levels wird die Stützstellenanzahl verdoppelt. Da wir die Stützstellen als Knoten für die B-Splines verwenden, werden diese dabei entlang der x-Achse um den Faktor 2 gestaucht. Wegen

$$\frac{d^2}{dx^2} b(2x) = \frac{d}{dx} 2b'(2x) = 2\frac{d}{dx} b'(2x) = 4b''(2x) \tag{6.12}$$

sind folglich die Einträge nach einer Inkrementierung des Levels viermal so groß und die Lösung um diesen Faktor ungenauer. Auf eine Leveldifferenz von 10 müsste dieser Effekt eine Verschlechterung der Genauigkeit um den Faktor $4^{10} \approx 10^6$ ausmachen. Genau das beobachten wir in Abbildung 6.6.

6.3 Schwache Formulierung

Bevor wir zur Kollokation kommen, machen wir einen kurzen Einschub über die Metriken, mit denen wir unsere Lösungen beurteilen. Sind die Metriken aus Gleichungen (6.2) und (6.3) immer sinnvoll? Man stelle sich vor, man habe eine Lösung, deren Differenz zur exakten Lösung überall klein ist, aber hochfrequent schwankt. Da solch eine Schwankung stark zur zweiten Ableitung beitrüge, führte sie zum Beispiel mit dem Laplace-Operator zu einem hohen Residuum, obwohl der tatsächliche Fehler klein wäre.

Um dem entgegenzuwirken, schauen wir zur Finite-Elemente-Methode, einer anderen Methode zur Lösung partieller Differentialgleichungen [Meh18]. Dabei wird die Gleichung $\mathcal{L}u = f$ über dem Rechengebiet Ω in ihre schwache Form [Höl03; KW17]

$$\int_{\Omega} \phi(\vec{x}) \mathcal{L}u(\vec{x}) d\vec{x} = \int_{\Omega} \phi(\vec{x}) f(\vec{x}) d\vec{x} \tag{6.13}$$

gebracht und für einen Menge von Testfunktionen $\Phi \ni \phi$ gelöst. Ebenso wird die Lösung daran gemessen, wie gut sie die schwache Formulierung erfüllt [KW17].

Diese Metrik können wir aber auch nutzen um unsere Lösungen bei der BSKK zu beurteilen. Wir wählen in d Dimensionen die Ansatzfunktionen

$$\Phi = \{\phi_{\bar{i}} \mid \bar{i} \in \mathbb{N}_0^d, \|\bar{i}\|_{\max} \leq 2^l\} \tag{6.14}$$

mit dem Level l des bei der Kollokation verwendeten dünnen Gitters und

$$\phi_{\bar{i}}(\vec{x}) := \begin{cases} 1 & \text{falls } \forall\ 1 \leq k \leq d : x_k \leq \frac{i_k + 0.5}{2^l} \\ 0 & \text{sonst.} \end{cases} \tag{6.15}$$

Haben wir eine numerische Lösung u einer partiellen Differentialgleichung $\mathcal{L}u = f$, sind wir für alle $\phi_{\bar{i}} \in \Phi$ am Residuum der schwachen Form

$$\begin{aligned} r_{\bar{i}} &= \int_{[0,1]^d} \phi_{\bar{i}}(\vec{x}) f(\vec{x}) d\vec{x} - \int_{[0,1]^d} \phi_{\bar{i}}(\vec{x}) \mathcal{L}u(\vec{x}) d\vec{x} \\ &= \int_{[0,1]^d} \phi_{\bar{i}}(\vec{x}) \left(f(\vec{x}) - \mathcal{L}u(\vec{x})\right) d\vec{x} \end{aligned} \tag{6.16}$$

interessiert. Um es nicht analytisch lösen zu müssen, berechnen wir diese Residuen mittels Quadratur. Als Metrik nehmen wir dann deren ℓ^2-Norm. Was ist nun der Vorteil zur bisherigen Metrik? Durch das Integral erhoffen wir uns, hochfrequente Schwankungen aus dem Residuum zu entfernen. In den folgenden Abbildungen ist diese Metrik als *Residuum der schwachen Form* gekennzeichnet.

6.4 Zweidimensionale Kollokation

Im Zweidimensionalen lautet die Poisson-Gleichung, wenn wir wieder als rechte Seite $f(x, y) = 1$ verwenden,

$$-\Delta u(x, y) = 1. \tag{6.17}$$

Abbildung 6.7 zeigt die Residuen bei der Lösung dieser Gleichung mit homogenen Dirichlet-Randbedingungen

$$u(x) = 0 \quad \forall x \in \partial[0, 1]^2 \tag{6.18}$$

mittels B-Splines vom Grad 5. Für die Residuen wurden im Gegensatz zu den 150 im Eindimensionalen, jetzt 150^2 Stichproben genommen.

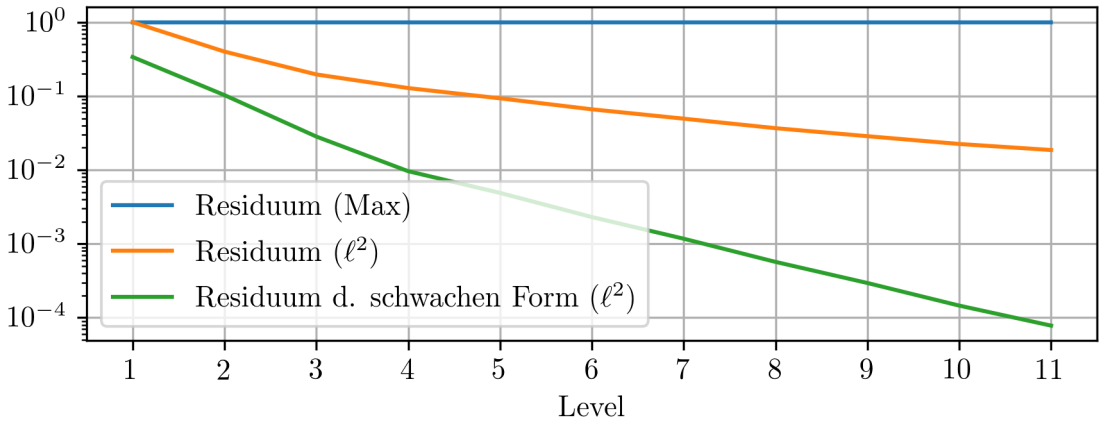


Abbildung 6.7: Residuen beim Lösen der Gleichungen (6.17) und (6.18) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

Hier stellt sich die Frage, wieso das betragsmäßig größte Residuum stets $10^0 = 1$ ist. Abbildung 6.8 zeigt exemplarisch die Lösung auf Level 11, gibt jedoch auch keine Antwort auf diese Frage. In Abbildung 6.9 sind die Residuen visualisiert und man erkennt, dass das Residuum in den Ecken stets -1 ist. Dies widerspricht nicht den Bedingungen, die wir an die Lösung geknüpft haben, da wir an den Stützstellen auf dem Rand lediglich die Randbedingungen einfordern. Mit steigendem Level wird das Gitter feiner und infolgedessen der Eckbereich kleiner, in dem das Residuum betragsmäßig groß ist.

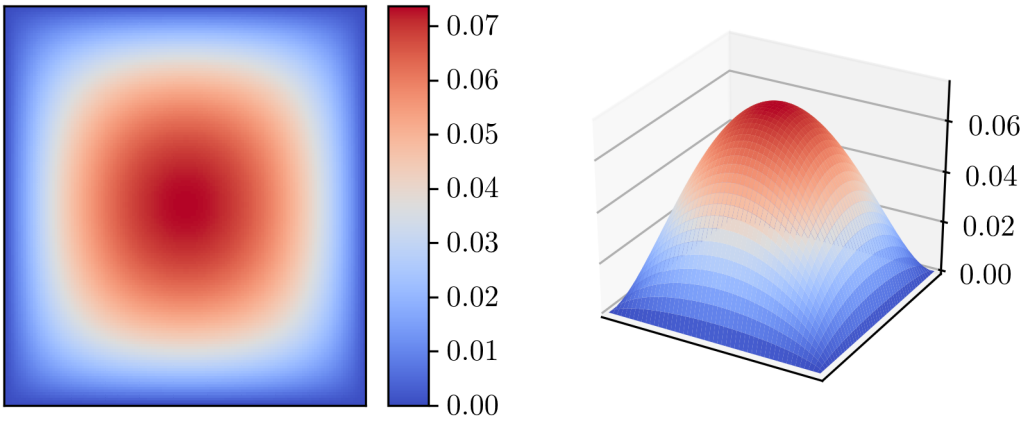


Abbildung 6.8: Lösung der Gleichungen (6.17) und (6.18) auf Level 11 (mittels B-Splines vom Grad 5)

Abgesehen von diesen Ausreißern ist aber auch die ℓ^2-Norm der Residuen in Abbildung 6.7 viel höher als bei unseren bisherigen Ergebnissen. Dies ist offenbar die in Abschnitt 4.1 erwähnte Ungenauigkeit bei der Kollokation mit der Kombinationstechnik, wegen der wir die iterative Verbesserung implementiert haben.

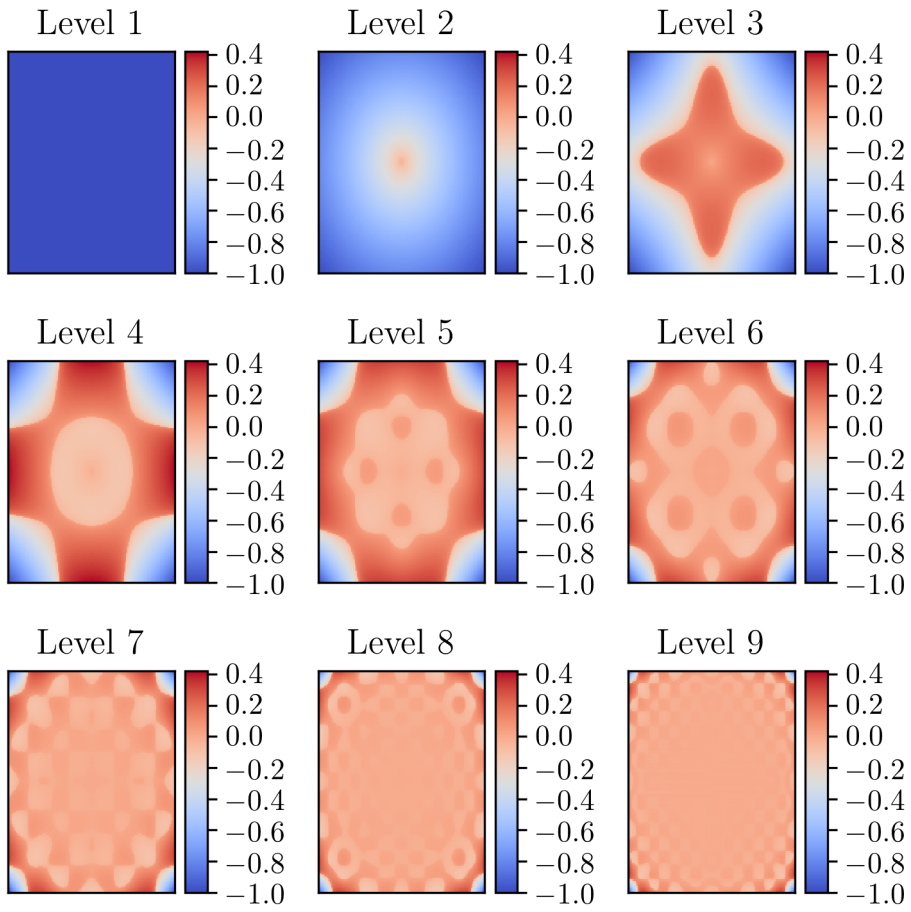


Abbildung 6.9: Visualisierung der Residuen beim Lösen der Gleichungen (6.17) und (6.18) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

6.5 Iterative Verbesserung

Exemplarisch wenden wir die iterative Verbesserung auf die Level-6-Lösung aus Abbildung 6.7 an. Die resultierenden Residuen nach den Iterationen sind in Abbildung 6.10 aufgezeichnet. Die iterative Verbesserung ist – wie in Abschnitt 4.2 erklärt – darauf ausgelegt, die Residuen an den Gitterpunkten zu verbessern. Um dies zu messen haben wir in der Abbildung zusätzlich zu den bisherigen Metriken die Maximumsnorm

$$\|r\|_{\max,\mathrm{SG}_l^d} := \max\left\{r(x) \mid x \in \mathrm{SG}_l^d\right\} \tag{6.19}$$

sowie die ℓ^2-Norm

$$\|r\|_{\ell^2,\mathrm{SG}_l^d} := \sqrt{\sum_{x\in\mathrm{SG}_l^d} r(x)^2} \tag{6.20}$$

dieser Residuen aufgezeichnet. In der Tat steigt die Genauigkeit an den Stützstellen um knapp eine Größenordnung pro Dimension. Dieses Ziel hat die iterative Verbesserung demnach erreicht. Jedoch geht dies offensichtlich mit einer Genauigkeitsminderung zwischen den Stützstellen einher,

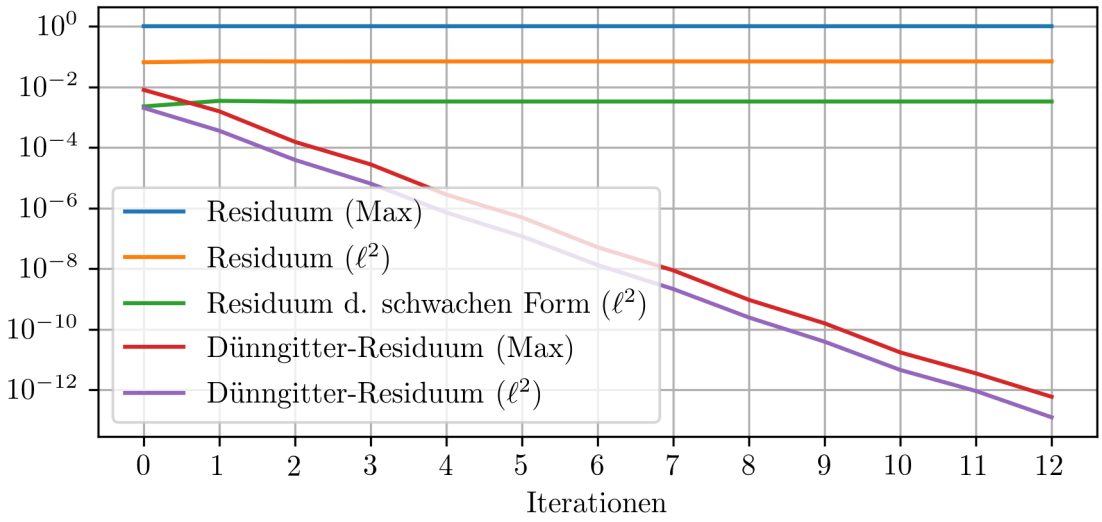


Abbildung 6.10: Iterative Verbesserung der Lösung der Gleichungen (6.17) und (6.18) auf Level 6 (mittels B-Splines vom Grad 5)

denn insgesamt bleibt die Genauigkeit gleich; sowohl in der Metrik des Residuums, als auch in der Metrik des Residuums der schwachen Formulierung. Dieselbe Beobachtung konnten wir auch bei der Lösung anderer Kollokationsprobleme feststellen. Wir kommen also zu dem Schluss, dass die iterative Verbesserung zwar das Ziel erreicht, das Residuum an den Gitterpunkten zu drücken, die Lösung in ihrer Gesamtheit aber nicht verbessert.

6.6 Vergleich zur Kollokation auf vollen Gittern

Wir haben dünne Gitter verwendet, weil wir wir hofften, auf einem dünnen Gitter des Levels l ähnliche Ergebnisse, wie auf einem vollen Gitter des Levels l zu erhalten; und das bei wesentlich weniger Gitterpunkten, siehe Gleichung (3.74). Das soeben auf dünnen Gittern gelöste Problem haben wir auch auf vollen gelöst und Abbildung 6.11 zeigt die zugehörigen Residuen. Die Genauigkeit steigt (in der logarithmischen Skala) doppelt so schnell wie in Abbildung 6.7, in der es um die Dünngitter-Lösungen ging. Beispielsweise ist die Vollgitter-Lösung auf Level 5 von der gleichen Genauigkeit, wie die Dünngitter-Lösung auf Level 10. Da wir uns hierbei im Zweidimensionalen befinden, enthält das dünne Gitter SG^2_{10} das volle Gitter $V_{(5,5)}$. Dieses Problem auf dünnen Gittern zu lösen hat also keinen Vorteil gebracht, sondern nur mehr Rechenleistung benötigt (als auf dem vollen Gitter des halben Levels).

Wir haben bereits in Abbildung 6.9 gesehen, dass die Genauigkeit der Dünngitter-Lösungen vor allem in den Eckbereichen schlecht ist. Man könnte vermuten, das komme daher, dass es dort kaum Gitterpunkte gibt, außer denen, die als Randbedingungen zum LGS beitragen. In Abbildung 6.12 sind auch die Residuen der Vollgitter-Lösungen visualisiert und dort machen wir selbige Beobachtung. Demnach ist dieses Problem in den Eckbereichen inhärent schwieriger zu lösen und dann ist es nicht verwunderlich, dass die Verwendung von dünnen Gittern keinen Vorteil bringt.

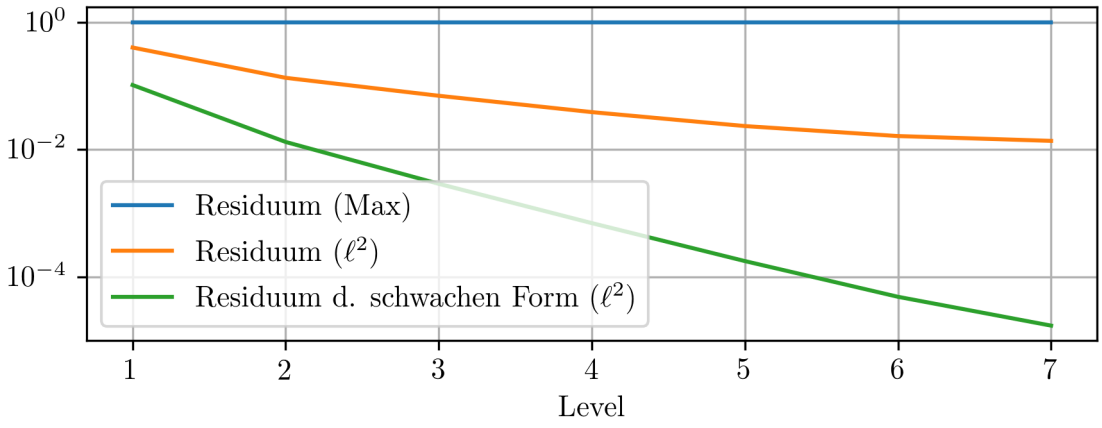


Abbildung 6.11: Residuen beim Lösen der Gleichungen (6.17) und (6.18) auf *vollen Gittern* unterschiedlicher Level (mittels B-Splines vom Grad 5)

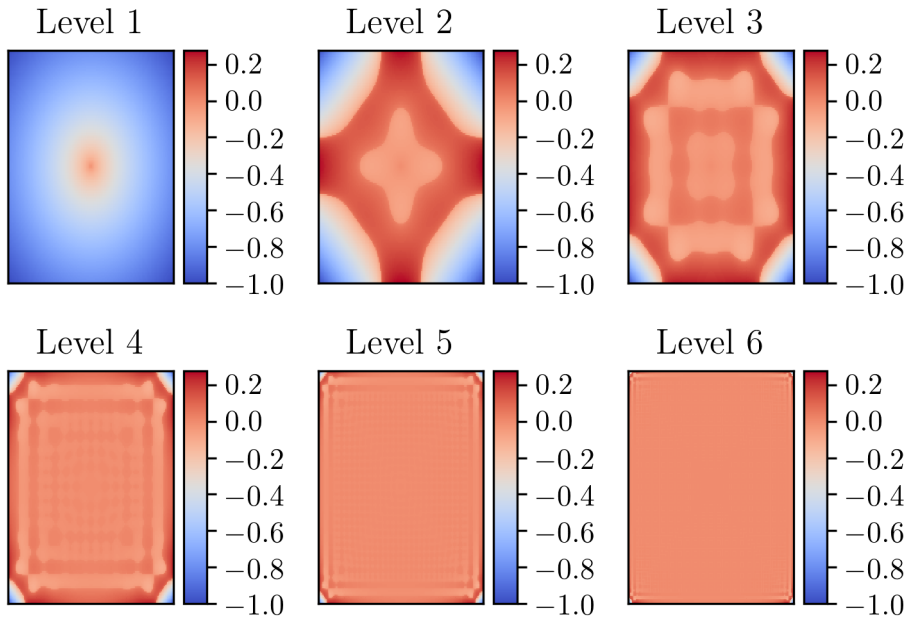


Abbildung 6.12: Visualisierung der Residuen beim Lösen der Gleichungen (6.17) und (6.18) auf *vollen Gittern* unterschiedlicher Level (mittels B-Splines vom Grad 5)

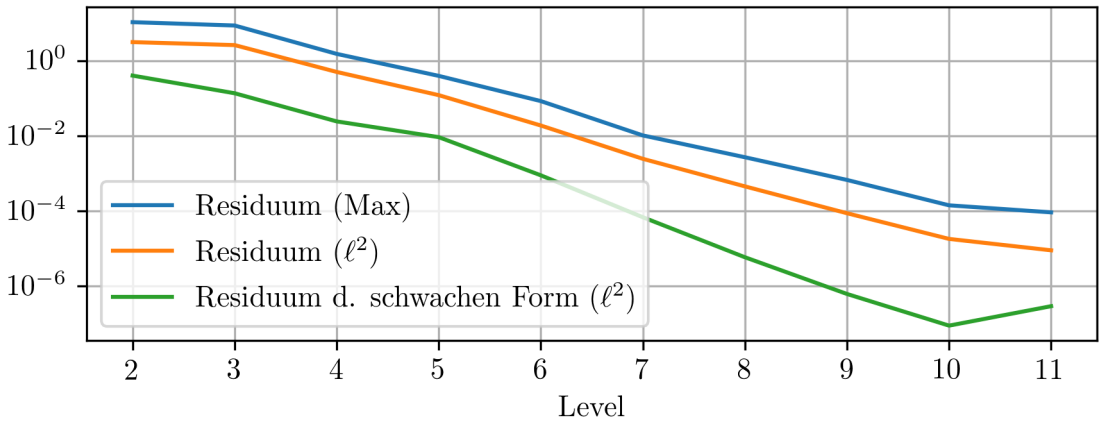


Abbildung 6.13: Residuen beim Lösen der Gleichungen (6.21) und (6.22) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

Wir nutzen im Folgenden ein glattes Problem ohne Singularitäten für den Vergleich zwischen dünnen und vollen Gittern. Diesmal lösen wir die zweidimensionale Laplace-Gleichung [Han02]

$$\Delta u(x, y) = 0 \tag{6.21}$$

und verwenden die Dirichlet-Randbedingungen

$$u(x, y) = r^7 \cdot \sin(7\phi), \tag{6.22}$$

wobei hier (r, ϕ) die Polarkoordinaten des Punktes $\left(x - \frac{1}{2}, y - \frac{1}{2}\right)$ sind.

Bei diesem Problem erhalten wir auf dünnen Gittern die in Abbildung 6.13 und auf vollen Gittern die in Abbildung 6.14 aufgezeichneten Residuen. Diesmal sind die Genauigkeiten von dünnen und vollen Gittern mit vergleichbarer Anzahl an Gitterpunkten ähnlich. Beispielsweise hat das dünne Gitter SG_9^2 vom Level 9 etwas weniger Gitterpunkte als das volle Gitter $V_{(6,6)}$ und dennoch ist die Lösung auf dem dünnen Gitter SG_9^2 knapp genauer. Davon, dass die Lösung auf einem dünnen Gitter ähnlich gut ist, wie die auf dem vollen Gitter desselben Levels, sind wir allerdings auch bei diesem Beispiel weit entfernt.

6.7 Hohe Dimensionalität

Da wir überhaupt dünne Gitter verwenden, um dem Fluch der Dimensionalität entgegenzuwirken, müssen wir natürlich auch verifizieren, dass die Software mit Problemen in vielen Dimensionen zurechtkommt. Wieder lösen wir die Gleichung

$$-\Delta u(\vec{x}) = 1 \tag{6.23}$$

mit homogenen Dirichlet-Randbedingungen

$$u(\vec{x}) = 0 \quad \text{für } x \in \partial[0, 1]^5 \tag{6.24}$$

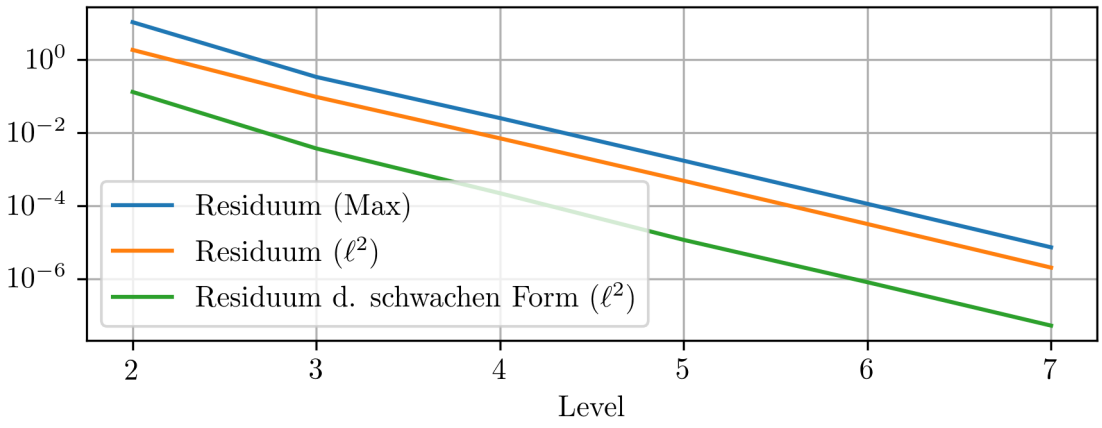


Abbildung 6.14: Residuen beim Lösen der Gleichungen (6.21) und (6.22) auf *vollen Gittern* unterschiedlicher Level (mittels B-Splines vom Grad 5)

aber diesmal in fünf Dimensionen. Abbildung 6.15 zeigt die zugehörigen Residuen. Das dünne Gitter vom Level 5 ist im Fünfdimensionalen das Erste mit einer Stützstelle im Inneren, weshalb wir bei den vorherigen Leveln als Lösung die konstante Nullfunktion mit dem konstanten Residuum 1 erhalten. Ab Level 5 steigt die Genauigkeit, jedoch nur sehr langsam.

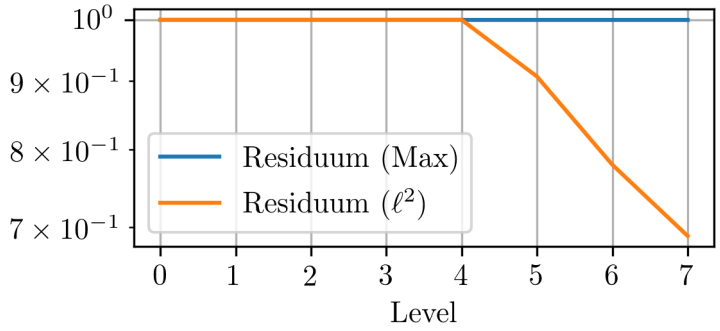


Abbildung 6.15: Residuen beim Lösen der Gleichungen (6.23) und (6.24) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

6.8 Problematische Randbedingungen

Bisher hatten wir nur Beispiele mit unproblematischen Randbedingungen. Wieder lösen wir die Laplace-Gleichung und zwar diesmal mit den Dirichlet-Randbedingungen

$$u(x,y) = \begin{cases} 1 & \text{falls } x = 0, \\ 0 & \text{sonst} \end{cases} \quad \text{für } (x,y) \in \partial[0,1]^2. \tag{6.25}$$

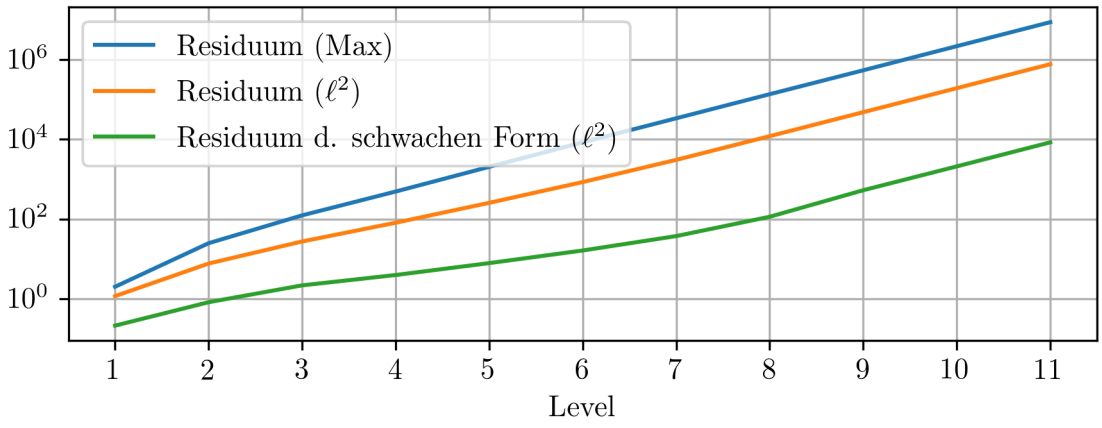


Abbildung 6.16: Residuen beim Lösen der Gleichungen (6.21) und (6.25) auf unterschiedlichen Leveln (mittels B-Splines vom Grad 5)

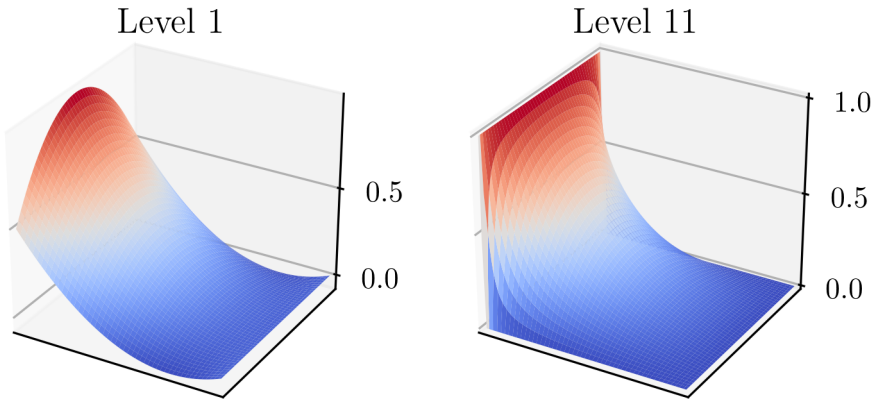


Abbildung 6.17: Zwei Lösungen der Gleichungen (6.21) und (6.25) (mittels B-Splines vom Grad 5)

Die Residuen bei der Lösung dieses Problems sind in Abbildung 6.16 dargestellt. Anstatt zu steigen, sinkt die Genauigkeit mit steigendem Level. Wie in Abbildung 6.17 exemplarisch gezeigt, sind jedoch bei niedrigen Leveln die Randbedingungen sehr schlecht erfüllt. Mit steigendem Level steigt auch die Qualität der Randwerte; auf Kosten des Residuums bezüglich der partiellen Differentialgleichung für das Innere.

Hier haben wir wieder das in Abschnitt 6.6 erläuterte Problem: Gerade in den Ecken, in denen das dünne Gitter wenige Stützstellen besitzt, hat das Problem Singularitäten. Die bei dünnen Gittern verwendete Anordnung der Gitterpunkte ist dann optimal, wenn das Problem bestimmte Glattheitsbedingungen erfüllt, als da sind, dass die exakte Lösung beschränkte zweite gemischte Ableitungen hat [BG04; Gar12]. Da dies in den Singularitäten nicht der Fall ist, wäre ein Gitter geeignet, das in den betroffenen Ecken besonders fein aufgelöst ist [Bun92; GG08].

7 Zusammenfassung

In dieser Arbeit haben wir unsere Implementierung der B-Spline-Kollokation mit der Kombinationstechnik im Software-Paket SG++ vorgestellt und einige Ergebnisse ausgewertet. Die B-Spline-Kollokation ist eine numerische Methode, bei der die Lösung einer partiellen Differentialgleichung mithilfe von B-Splines diskretisiert wird. Die Koeffizienten der B-Splines ergeben sich durch das Lösen eines LGS. Im Gegensatz zur Finite-Elemente-Methode, bei der die schwache Formulierung einer partiellen Differentialgleichung gelöst wird, sind bei der B-Spline-Kollokation keine Integrale zu berechnen, was eine schnellere Berechnung ermöglicht.

Auf die B-Spline-Kollokation wendeten wir die Kombinationstechnik an, mit der Probleme auf dünnen Gittern gelöst werden können. Ein dünnes Gitter der Dimensionalität $d \geq 2$, auf dessen Rand in jeder Koordinatenrichtung N Punkte liegen, besitzt mit seinen $O(N \cdot \log(N)^d)$ Punkten wesentlich weniger als ein reguläres Gitter mit N^d Punkten. Davon erhofften wir uns, den Fluch der Dimensionalität zu umgehen und die Berechnung in mehreren Dimensionen noch weiter beschleunigen zu können. Bei unserem Vergleich zwischen Lösungen auf vollen und dünnen Gittern mit derselben Anzahl an Stützstellen in Abschnitt 6.6 stellten wir jedoch keine signifikant unterschiedlichen Genauigkeiten der Lösungen fest.

Da die durch die Kombinationstechnik erhaltene Lösung partieller Differentialgleichungen im allgemeinen keine exakte Dünngitter-Lösung ist, implementierten wir ein iteratives Verfahren zur Verbesserung des Residuums an den Gitterpunkten. Dieses Ziel wurde – unseren Auswertungen zufolge – erreicht. Jedoch trat unsere Hoffnung, dass sich im Zuge dessen auch das Residuum zwischen den Gitterpunkten verbessert, nicht ein. Die untersuchten Lösungen wurden mit der iterativen Verbesserung in ihrer Gesamtheit nicht besser.

Zur Lösung eines Problems sind dünne Gitter optimal, wenn das Problem bestimmte Glattheitseigenschaften erfüllt. Probleme mit unstetigen Randbedingungen stellen dagegen eine Limitierung der Methode dar. Zur qualitativ hochwertigen Lösung solcher Probleme ist nahe der Unstetigkeiten eine hohe Auflösung des Gitters notwendig, die wir mit der Kombinationstechnik nicht haben.

8 Ausblick

Unsere Motivation für die BSKK, war die postulierte bessere Performanz im Vergleich zur Finite-Elemente-Methode, bei welcher Integrale berechnet werden müssen. Es wäre interessant, diese Behauptung in der Praxis zu überprüfen. Am repräsentativsten wäre es dabei, zwei *optimierte* Löser gegeneinander antreten zu lassen; einen für die Finite-Elemente-Methode und einen für die BSKK, wobei für Letztere noch keine optimierte Implementierung existiert.

Ein weiteres erwähnenswertes Thema ist die Lösung von nichtlinearen Problemen. Zur Lösung einer partiellen Differentialgleichung $\mathcal{L}u = f$ mit der Kollokationsmethode hatten wir in Abschnitt 3.6 die Linearität von $\mathcal{L}$ vorausgesetzt. Es bleibt zu untersuchen, ob und wie sich die BSKK auf nichtlineare partielle Differentialgleichungen verallgemeinern lässt.

8.1 Zeitliche Diskretisierung

Beim Lösen von zeitabhängigen partiellen Differentialgleichungen mit der B-Spline-Kollokation kann man die Zeitdimension analog zu den Raumdimensionen betrachten. Wenn das Problem in der Zeitdimension allerdings nicht glatt genug ist, würde eine Dünngitter-Lösung darunter leiden [BG04; Gar12]. In diesem Fall würde es sich möglicherweise anbieten, in der Zeit separat zu diskretisieren und ein Zeitschrittverfahren anzuwenden. Dabei wird ausgehend von einer Lösung zum Startzeitpunkt t_0 eine Lösung für den darauf folgenden Zeitpunkt t_1 berechnet und so weiter.

Im Folgenden schlagen wir einige Zeitschrittverfahren vor, mit denen eine zeitabhängige partielle Differentialgleichung

$$\frac{\partial}{\partial t}u(\vec{x},t) = \mathcal{L}u(\vec{x},t) + f(t,u(\vec{x},t)) \quad \text{auf } \Omega \times [t_0, t_{\text{end}}] \tag{8.1}$$

gelöst werden kann. Wir verwenden die Schreibweisen $t_i := t_0 + i \cdot dt$ und $u^{(i)} := u(\vec{x},t_i)$, wobei t_0 der Startzeitpunkt und dt die Länge der Zeitschritte ist. Für jeden Zeitschritt soll eine mithilfe von B-Splines diskretisierte Lösung berechnet werden. Dabei gehen wir davon aus, dass wir durch die Randbedingungen des Problems eine Lösung für den Startzeitpunkt t_0 erhalten können; sei es durch Interpolation von Dirichlet-Randbedingungen oder durch Kollokation im Fall von gemischten Randbedingungen. Schließlich sollte die Lösung zum Zeitpunkt t_0 nicht von der Zukunft abhängen.

Ein wohlbekanntes Zeitschrittverfahren ist das explizite Euler-Verfahren [Han02], das hier die Form

$$u^{(i+1)} = u^{(i)} + h\frac{\partial}{\partial t}u^{(i)} = u^{(i)} + h\left(\mathcal{L}u^{(i)} + f\left(t_i, u^{(i)}\right)\right) \tag{8.2}$$

hat. Wir können an allen Gitterpunkten $\vec{x}$ den Wert $u^{(i+1)} = u(\vec{x}, t_{i+1})$ berechnen und diese Werte über dem Rechengebiet interpolieren. Dieses Verfahren erfordert also eine Interpolation pro Zeitschritt, die wir zum Beispiel mit der Kombinationstechnik durchführen können.

Beim impliziten Euler-Verfahren [Han02]

$$u^{(i+1)} = u^{(i)} + h\frac{\partial}{\partial t}u^{(i+1)} = u^{(i)} + h\left(\mathcal{L}u^{(i+1)} + f(t_{i+1}, u^{(i+1)})\right) \tag{8.3}$$

kommt dagegen die Lösung zum nächsten Zeitpunkt $u^{(i+1)}$ auch auf der rechten Seite vor. Falls jedoch $f(t, u(\vec{x}, t)) =: f(t)$ unabhängig von $u(\vec{x}, t)$ ist, lässt sich Gleichung (8.3) umformen zu

$$(id - h\mathcal{L})u^{(i+1)} = u^{(i)} + hf(t_{i+1}). \tag{8.4}$$

Das ist also eine partielle Differentialgleichung mit dem Differentialoperator $(id - h\mathcal{L})$. Dieser ist linear, also können wir $u^{(i+1)}$ mit der Kollokationsmethode approximieren.

Unter der gleichen Voraussetzung lässt sich auch das Crank-Nicolson-Verfahren [Han02]

$$\begin{aligned} u^{(i+1)} &= u^{(i)} + \frac{h}{2}\left(\frac{\partial}{\partial t}u^{(i)} + \frac{\partial}{\partial t}u^{(i+1)}\right) \\ &= u^{(i)} + \frac{h}{2}\left(\mathcal{L}u^{(i)} + f\left(t_i, u^{(i)}\right) + \mathcal{L}u^{(i+1)} + f(t_{i+1})\right) \end{aligned} \tag{8.5}$$

zu einer Berechnungsvorschrift

$$(id - \frac{h}{2}\mathcal{L})u^{(i+1)} = u^{(i)} + \frac{h}{2}\left(\mathcal{L}u^{(i)} + f(t_i) + f(t_{i+1})\right) \tag{8.6}$$

umformen. Auch in diesem Fall ist der Differentialoperator linear. Sowohl beim impliziten Euler-Verfahren, als auch beim Crank-Nicolson-Verfahren muss also ein Kollokationsproblem pro Zeitschritt gelöst werden.

Eine Abwandlung des Crank-Nicolson-Verfahrens ist die Methode von Heun [Han02]. Bei dieser Methode wird zuerst eine Näherung $\tilde{u}^{(i+1)}$ durch einen Schritt des expliziten Euler-Verfahrens berechnet. Diese Näherung wird dann in Gleichung (8.5) als Approximation für $u^{(i+1)}$ eingesetzt, wodurch die resultierende Berechnungsvorschrift

$$\begin{aligned} u^{(i+1)} &= u^{(i)} + \frac{h}{2}\left(\frac{\partial}{\partial t}u^{(i)} + \frac{\partial}{\partial t}\tilde{u}^{(i+1)}\right) \\ &= u^{(i)} + \frac{h}{2}\left(\mathcal{L}u^{(i)} + f\left(t_i, u^{(i)}\right) + \mathcal{L}\tilde{u}^{(i+1)} + f(t_{i+1}, \tilde{u}^{(i+1)})\right) \end{aligned} \tag{8.7}$$

explizit ist. Durch Einsetzen der Formel des expliziten Euler-Verfahrens für $\tilde{u}^{(i+1)}$ in Gleichung (8.7) lässt sich $u^{(i+1)}$ in einem Schritt berechnen. Es ist also wie beim expliziten Euler-Verfahren eine Interpolation pro Zeitschritt durchzuführen. Das Gleiche funktioniert analog mit dem Verfahren von Runge-Kutte [Han02]

Bei den hier aufgelisteten Zeitschrittverfahren können die Lösungen der einzelnen Zeitschritte zum Beispiel mit der BSKK berechnet werden. Eine Implementierung und Untersuchung dieser Verfahren mithilfe der BSKK steht aus.

Literaturverzeichnis

[Bel57] R. Bellman. *Dynamic Programming*. P (Rand Corporation). Princeton University Press, 1957. ISBN: 978-0486428093 (zitiert auf S. 32).

[BG04] H.-J. Bungartz, M. Griebel. „Sparse grids“. In: *Acta Numerica*. Bd. 13. University of Cambridge, 2004. ISBN: 9780511569975. DOI: 10.1017/S0962492904000182 (zitiert auf S. 7, 25, 53, 57).

[BGRZ94] H.-J. Bungartz, M. Griebel, D. Röschke, C. Zenger. „Pointwise Convergence Of The Combination Technique For Laplace's Equation“. English. In: *East-West J. Numer. Math.* 2.1 (1994), S. 21–45. ISSN: 0928-0200 (zitiert auf S. 7, 29).

[Boo78] C. D. Boor. *A Practical Guide to Splines*. Springer, 1978. ISBN: 3-540-90356-9 (zitiert auf S. 7, 10).

[BSBF] *B-spline Basis Functions: Definition*. URL: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-basis.html (zitiert auf S. 18).

[Bun11] H.-J. Bungartz. *Concepts for Higher Order Finite Elements on Sparse Grid*. 2011 (zitiert auf S. 25).

[Bun92] H.-J. Bungartz. *An Adaptive Poisson Solver Using Hierarchical Bases And Sparse Grids*. 1992 (zitiert auf S. 53).

[BZBP13] H.-J. Bungartz, S. Zimmer, M. Buchholz, D. Pflüger. *Modellbildung und Simulation*. Springer, 2013. ISBN: 978-3-642-37656-6. DOI: 10.1007/978-3-642-37656-6 (zitiert auf S. 7, 9).

[Cpp] *Standard C++*. URL: https://isocpp.org (zitiert auf S. 7).

[Dah08] W. Dahmen. *Numerik für Ingenieure und Naturwissenschaftler*. Deutsch. Hrsg. von A. Reusken. Zweite, korrigierte Auflage. Springer-Lehrbuch. Springer, 2008. ISBN: 978-3-540-76493-9. DOI: 10.1007/978-3-540-76493-9 (zitiert auf S. 15).

[DBSC] *Derivatives of a B-spline Curve*. URL: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-derv.html (zitiert auf S. 32).

[Eigen] *Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.* URL: http://eigen.tuxfamily.org (zitiert auf S. 38).

[Gar12] J. Garcke. „Sparse Grids in a Nutshell“. In: *Lecture Notes in Computational Science and Engineering*. Bd. 88. Springer, 2012. ISBN: 978-3-642-31702-6. DOI: 10.1007/978-3-642-31703-3_3 (zitiert auf S. 7, 25–29, 53, 57).

[GG08] T. Gerstner, M. Griebel. „Sparse Grids“. In: *Encyclopedia of Quantitative Finance* (2008) (zitiert auf S. 7, 53).

[GMI12] J. Goh, A. A. Majid, A. I. M. Ismail. „Cubic B-Spline Collocation Method for One-Dimensional Heat and Advection-Diffusion Equations“. In: *Journal of Applied Mathematics* 2012 (2012). Article ID 458701, 8 pages. DOI: 10.1155/2012/458701 (zitiert auf S. 7).

[GSZ92] M. Griebel, M. Schneider, C. Zenger. „A combination technique for the solution of sparse grid problems“. In: *Iterative methods in linear algebra (Brussels, 1991)* (1992), S. 263–281 (zitiert auf S. 7, 26).

[Han02] M. Hanke-Bourgeois. *Grundlagen der numerischen Mathematik und des wissenschaftlichen Rechnens*. Bd. 178. 3. Springer, 2002. ISBN: 978-3-8348-0708-3. DOI: 10.1007/978-3-8348-9309-3 (zitiert auf S. 10, 16, 18, 19, 22, 23, 35, 38, 51, 57, 58).

[HGC06] M. Hegland, J. Garcke, V. Challis. *The combination technique and some generalisations*. Aug. 2006. DOI: 10.1016/j.laa.2006.07.014 (zitiert auf S. 7, 29).

[Höl03] K. Höllig. *Finite Element Methods with B-Splines*. Frontiers in Applied Mathematics. Society for Industrial und Applied Mathematics, 2003. ISBN: 9780898715330. DOI: 10.1137/1.9780898717532 (zitiert auf S. 7, 46).

[I75408] „IEEE Standard for Floating-Point Arithmetic“. In: *IEEE Std 754-2008* (Aug. 2008), S. 1–70. DOI: 10.1109/IEEESTD.2008.4610935 (zitiert auf S. 43).

[KW17] K. Knothe, H. Wessels. *Finite Elemente – Eine Einführung für Ingenieure*. 5. Aufl. Springer, 2017. ISBN: 978-3-662-49351-9. DOI: 10.1007/978-3-662-49352-6 (zitiert auf S. 7, 46).

[Lub] G. Lube. *Numerische Mathematik I*. URL: https://lp.uni-goettingen.de/get/text/1012 (zitiert auf S. 17, 19).

[Meh18] M. Mehl. *Grundlagen des Wissenschaftlichen Rechnens*. Version: WS17/18. 2018. URL: https://www.ipvs.uni-stuttgart.de/abteilungen/sgs/lehre/lehrveranstaltungen/vorlesungen/WS1718/NSG (zitiert auf S. 9–11, 22–24, 46).

[Par12] S. Parent. *Value Semantics and Concepts-based Polymorphism*. 2012. URL: https://www.youtube.com/watch?v=_BpMYeUFXv8 (zitiert auf S. 31).

[Par17] S. Parent. *Better Code: Runtime Polymorphism*. 2017. URL: https://www.youtube.com/watch?v=QGcVXgEVMJg (zitiert auf S. 31).

[PZ18] D. Pflüger, S. Zimmer. *Numerische und Stochastische Grundlagen*. Version: WS17/18. 2018. URL: https://www.ipvs.uni-stuttgart.de/abteilungen/sgs/lehre/lehrveranstaltungen/vorlesungen/WS1718/NSG (zitiert auf S. 13–15).

[Sch06] F. Schwabl. *Statistische Mechanik*. Springer-Lehrbuch. Springer, 2006. ISBN: 9783540310952. DOI: 10.1007/3-540-31097-5 (zitiert auf S. 33).

[SGpp] *SG++: General Sparse Grid Toolbox*. URL: http://sgpp.sparsegrids.org (zitiert auf S. 7, 29, 31).

[Sha49] C. E. Shannon. „Communication in the Presence of Noise“. In: *Proceedings of the IRE*. Bd. 86. Feb. 1949, S. 10–21 (zitiert auf S. 42).

[Zen91] C. Zenger. „Sparse grids“. In: *Parallel Algorithms for Partial Differential Equations*. Bd. 31. Wolfgang Hackbusch, 1991. ISBN: 3-528-07631-3 (zitiert auf S. 7).

Alle URLs wurden zuletzt am 25. 04. 2018 geprüft.

Erklärung

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

Ort, Datum, Unterschrift