TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Bachelorarbeit

Stabile Implementierung der Randelementmethode auf stark adaptierten Netzen

Ausgeführt am Institut für
Analysis und Scientific Computing
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.Math. Dr.techn. Dirk Praetorius

durch
Markus Mayr
Täglicher Markt 7/10
3500 Krems an der Donau

Inhaltsverzeichnis

1 Einleitung **4**

2 Affine Randelemente: Distanzbestimmung **5**
2.1 Abstand zwischen Punkt und affinem Randstück 5
2.2 Abstand zwischen zwei affinen Randstücken 9

3 Semi-analytische Berechnung bestimmter Doppelintegrale **13**
3.1 Interpolation 13
3.2 Gauß-Quadratur 14
3.3 Semi-analytische Berechnung des Doppelintegrals 15
3.4 Approximierende Matrix 19

4 Lösungsverfahren und numerische Beispiele **22**
4.1 Problemstellung und Lösungsansatz 22
4.1.1 Indirekte Randintegralmethode 22
4.1.2 Galerkin-Verfahren 22
4.1.3 Fehlerschätzer 24
4.1.4 Netzverfeinerung 24
4.1.5 Vorkonditionierung der Steifigkeitsmatrix $\boldsymbol{V}$ 25
4.1.6 Lösungsalgorithmus 26
4.1.7 Adaptiver Algorithmus 27
4.2 Beispiel Schlitz-Geometrie 28
4.3 Beispiel Z-Shape 33

A Implementierung des V-Operators **40**
A.1 Übersicht 40
A.2 Globale Variablen und Präprozessorkonstanten 40
A.3 `mexFunction` 41
A.4 `computeVij` 43
A.5 `computeVij_analytic` 43
A.6 `computeVij_semianalytic` 44
A.7 `slpWrapper` 45
A.8 `slp` 46
A.9 `dist2` 47
A.10 `ptoseg` 47

B Implementierung der Vorkonditionierung des V-Operators **49**

Abbildungsverzeichnis **51**

Literatur **51**

1 Einleitung

In dieser Arbeit beschäftigen wir uns mit der Randelementmethode, insbesondere zur Lösung der homogenen Laplace-Gleichung

$$\begin{aligned} \Delta u &= 0 \qquad && \text{in } \Omega \subset \mathbb{R}^2, \\ u &= g \qquad && \text{auf } \Gamma := \partial\Omega, \end{aligned}$$

wobei $\Delta u := \partial_x^2 u + \partial_y^2 u$ den Laplace-Operator bezeichnet und $\Omega \subset \mathbb{R}^2$ eine beschränkte Teilmenge von $\mathbb{R}^2$ mit Rand $\Gamma := \partial\Omega$ ist.
Im Zuge der Berechnung zerlegen wir den Rand in affine Randstücke $T_j = [\boldsymbol{a}, \boldsymbol{b}]$, wobei $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2$ Punkte in der Ebene sind und $[\boldsymbol{a}, \boldsymbol{b}]$ die konvexe Hülle der Punkte $\boldsymbol{a}, \boldsymbol{b}$ bezeichnet, also all jene Punkte einer Strecke von $\boldsymbol{a}$ nach $\boldsymbol{b}$. In Abschnitt 2 stellen wir einige Eigenschaften für affine Randstücke zusammen. Insbesondere interessiert uns hierbei die Abstandsbestimmung zwischen zwei Randstücken $T_j = [\boldsymbol{a}_i, \boldsymbol{b}_i]$ und $T_k = [\boldsymbol{a}_j, \boldsymbol{b}_j]$ und dessen effiziente Berechnung. Als Vorbereitung untersuchen wir die Abstandsbestimmung zwischen einem Punkt und einem affinen Randstück um abschließend zu zeigen, dass der Abstand zwischen T_j und T_k unter der Voraussetzung $T_j \cap T_k = \emptyset$ in zwei der Endpunkte angenommen wird. Daher genügt es, den Abstand zwischen je zwei Endpunkten zu berechnen und anschließend das Minimum über alle berechneten Werte zu ermitteln um den Abstand zwischen T_j und T_k zu berechnen.
In Abschnitt 3 beschäftigt uns die approximative bzw. semi-analytische Berechnung des Doppelintegrals

$$\int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}}, \tag{1}$$

wobei κ eine *asymptotisch glatte* Kernfunktion bezeichnet. Eine formale Definition dieses Begriffs folgt später, doch insbesondere interessieren wir uns für die Funktion $\log|\boldsymbol{x} - \boldsymbol{y}|$, die asymptotisch glatt ist. Unsere Strategie ist es, dass äußere Integral durch Gauss-Quadratur zu ersetzen. Unter einer bestimmten *Zulässigkeitsbedingung* können wir zeigen, dass der approximative Wert für wachsenden Quadraturgrad exponentiell schnell gegen den exakten Wert konvergiert. Abschließend betrachten wir eine Matrix $A \in \mathbb{R}^{N \times N}$, deren Einträge $A_{j,k}$ durch (1) gegeben sind. Wir bilden dann eine Matrix A_p so, dass die Einträge durch den approximativen Wert gegeben sind, wo immer das durch die Zulässigkeitsbedingung möglich ist und durch den exakten Wert sonst überall. Am Ende zeigen wir dann, dass die so definierte Matrix A_p in der Frobeniusnorm für wachsenden Quadraturgrad exponentiell schnell gegen A geht. Die in Abschnitt 2 bewiesenen Sätze werden hierbei für die Realisierung der Zulässigkeitsbedingung benötigt, in die der Abstand zwischen Strecken eingeht.
Im letzten Abschnitt 4 wird zunächst die Randelementmethode für die homogene Laplace-Gleichung mit Dirichlet-Randbedingung vorgestellt. Wir verwenden den sogenannten *indirekten Ansatz* um anschließend mithilfe eines Galerkin-Verfahrens zu lösen. Im Zuge des Galerkin-Verfahrens wird eine Matrix verwendet, für deren Berechnung sich die Erkenntnisse aus Abschnitt 3 als nützlich erweisen. Schließlich werden einige zuvor vorgestellte Techniken anhand numerischer Beispiele verglichen. So werden adaptive Netzverfeinerung und uniforme Netzverfeinerung gegenübergestellt und es wird untersucht, wie sich die Vorkonditionierung auf die auftretende Galerkin-Matrix auswirkt.

2 Affine Randelemente: Distanzbestimmung

Ziel dieses Abschnitts ist das Finden eines effizienten Algorithmus zur Bestimmung des minimalen Abstandes zwischen zwei Strecken in der euklidischen Ebene. Zunächst aber beschäftigen wir uns aber mit der Bestimmung des minimalen Abstandes zwischen einem Punkt und einer Strecke.

2.1 Abstand zwischen Punkt und affinem Randstück

Definition 2.1. *Seien $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2$ mit $\boldsymbol{a} \neq \boldsymbol{b}$. Dann heißt*

$$T := [\boldsymbol{a}, \boldsymbol{b}] := \left\{ \frac{1}{2} (\boldsymbol{a} + \boldsymbol{b} + \lambda(\boldsymbol{b} - \boldsymbol{a})) \quad \middle| \quad \lambda \in [-1, 1] \right\}$$

affines Randstück *von $\boldsymbol{a}$ und $\boldsymbol{b}$.*

Bemerkung 2.2. *Affine Randstücke sind in $\mathbb{R}^2$ bezüglich der euklidischen Topologie klarerweise abgeschlossen. Sie sind außerdem beschränkt und daher nach dem Satz von Heine-Borel kompakt.*

Wir betrachten im Folgenden den Abstand zwischen einem Punkt und einem affinen Randstück. Der Abstand ist bezüglich der euklidischen Norm $\| \cdot \|_2$ definiert, für die wir in diesem und im folgenden Abschnitt nur $| \cdot |$ schreiben.

Definition 2.3. *Sei $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Dann heißt*

$$\gamma : [-1, 1] \to T : \lambda \mapsto \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b} + \lambda(\boldsymbol{b} - \boldsymbol{a}))$$

die zu T zugehörige Parametrisierung.

Wir wollen einige leicht einzusehende Eigenschaften dieser Parametrisierung festhalten.

Bemerkung 2.4.

- *Die Abbildung ist bijektiv, denn mit*

  $$\boldsymbol{x} \mapsto \begin{cases} \frac{2x_1 - a_1 - b_1}{b_1 - a_1} & \text{falls } a_1 \neq b_1 \\ \frac{2x_2 - a_2 - b_2}{b_2 - a_2} & \text{falls } a_2 \neq b_2 \end{cases}$$

  ist eine Umkehrabbildung gegeben.

- *Die Parametrisierung ist stetig. Sie ist außerdem stetig differenzierbar, da die Ableitung*

  $$\gamma' : \lambda \mapsto \frac{1}{2}(\boldsymbol{b} - \boldsymbol{a})$$

  offensichtlich stetig ist.

- *Auch die Umkehrabbildung ist stetig.*

- *Eine bijektive, stetig differenzierbare Abbildung, deren Umkehrabbildung stetig ist, bezeichnet man als* stetig differenzierbaren Homöomorphismus. *γ ist ein stetig differenzierbarer Homöomorphismus.*

Wir stellen nun fest, dass die Menge aller affinen Randstücke unter punktweise angewendeten Translationsabbildungen und linearen Abbildungen, abgeschlossen ist.

Definition 2.5. *Sei f eine Abbildung und $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Dann ist*

$$f(T) := \{f(\boldsymbol{x}) \;\mid\; \boldsymbol{x} \in T\}$$

das Bild von T unter der Abbildung f. Für den Fall, dass f eine lineare Abbildung ist, die durch eine Matrix A beschrieben wird, schreiben wir für $f(T)$ auch AT.

Lemma 2.6. *Seien $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück, $\tau_{\boldsymbol{t}} : \boldsymbol{x} \mapsto \boldsymbol{x} - \boldsymbol{t}$ mit $\boldsymbol{t} \in \mathbb{R}^2$ eine Translationsabbildung und f eine lineare Abbildung. Dann sind $\tau_{\boldsymbol{t}}(T)$ und $f(T)$ ebenfalls affine Randstücke und es gilt*

$$\tau_{\boldsymbol{t}}(T) = [\tau_{\boldsymbol{t}}(\boldsymbol{a}), \tau_{\boldsymbol{t}}(\boldsymbol{b})] \quad bzw. \quad f(T) = [f(\boldsymbol{a}), f(\boldsymbol{b})]\,. \tag{2}$$

Beweis. Sei $\boldsymbol{x} \in T$. Dann existiert ein $\lambda \in [-1, 1]$ mit $\boldsymbol{x} = \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b} + \lambda(\boldsymbol{b} - \boldsymbol{a})$. Es folgt

$$\begin{aligned}
\tau_{\boldsymbol{t}}(\boldsymbol{x}) &= \boldsymbol{x} - \boldsymbol{t} = \frac{1}{2}\left(\boldsymbol{a} + \boldsymbol{b} + \lambda(\boldsymbol{b} - \boldsymbol{a})\right) - \boldsymbol{t} \\
&= \frac{1}{2}\left((\boldsymbol{a} - \boldsymbol{t} + \boldsymbol{b} - \boldsymbol{t} + \lambda\left((\boldsymbol{b} - \boldsymbol{t}) - (\boldsymbol{a} - \boldsymbol{t})\right)\right) \\
&= \frac{1}{2}\left(\tau_{\boldsymbol{t}}(\boldsymbol{a}) + \tau_{\boldsymbol{t}}(\boldsymbol{b}) + \lambda\left(\tau_{\boldsymbol{t}}(b) - \tau_{\boldsymbol{t}}(a)\right)\right) \in [\tau_{\boldsymbol{t}}(a), \tau_{\boldsymbol{t}}(b)]\,.
\end{aligned}$$

Dies zeigt die Inklusion $\tau_{\boldsymbol{t}}([\boldsymbol{a}, \boldsymbol{b}]) \subseteq [\tau_{\boldsymbol{t}}(\boldsymbol{a}), \tau_{\boldsymbol{t}}(\boldsymbol{b})]$. Die andere Inklusion folgt analog.
Zu zeigen bleibt noch, dass dies auch für lineare Abbildungen f gilt. Wir wählen also wieder einen Punkt $\boldsymbol{x} = \frac{1}{2}\left(\boldsymbol{a} + \boldsymbol{b} + \lambda(\boldsymbol{b} - \boldsymbol{a})\right) \in T$ auf dem affinen Randstück. Dann gilt

$$f(\boldsymbol{x}) = \frac{1}{2}\left(f(\boldsymbol{a}) + f(\boldsymbol{b}) + \lambda\left(f(\boldsymbol{a}) - f(\boldsymbol{b})\right)\right) \in [f(\boldsymbol{a}), f(\boldsymbol{b})]\,.$$

Dies zeigt die Inklusion $f([\boldsymbol{a}, \boldsymbol{b}]) \subseteq [f(\boldsymbol{a}), f(\boldsymbol{b})]$. Die andere Inklusion erfolgt wiederum analog. □

Definition 2.7. *Sei $\boldsymbol{x} \in \mathbb{R}^2$ und $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Dann ist*

$$\operatorname{dist}(\boldsymbol{x}, T) := \inf\{|\boldsymbol{x} - \boldsymbol{y}| \;\mid\; \boldsymbol{y} \in T\}$$

der Abstand *zwischen $\boldsymbol{x}$ und T.*

Wir stellen fest, dass der Abstand unter Translation und Multiplikation mit einer orthogonalen Matrix invariant ist.

Lemma 2.8. *Seien $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^2$ Punkte und $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Seien weiters $\tau_{\boldsymbol{t}}$ eine Translation und A eine orthogonale Abbildung. Dann gilt:*

$$\operatorname{dist}(\tau_{\boldsymbol{t}}(\boldsymbol{x}), \tau_{\boldsymbol{t}}(T)) = \operatorname{dist}(\boldsymbol{x}, T)\,, \tag{3}$$

$$\operatorname{dist}(A\boldsymbol{x}, AT) = \operatorname{dist}(\boldsymbol{x}, T)\,. \tag{4}$$

Beweis. Für eine beliebige Translation $\tau_{\boldsymbol{t}}$ und $\boldsymbol{y} \in T$ beliebig gilt

$$|\tau_{\boldsymbol{t}}(\boldsymbol{x}) - \tau_{\boldsymbol{t}}(\boldsymbol{y})| = |\boldsymbol{x} - \boldsymbol{t} - \boldsymbol{y} + \boldsymbol{t}| = |\boldsymbol{x} - \boldsymbol{y}|.$$

Der Abstand stimmt für alle $\boldsymbol{y} \in T$ überein. Daher stimmen auch die Infima überein. Für eine beliebige orthogonale Abbildung A und $\boldsymbol{y} \in T$ beliebig gilt wegen der Linearität der Abbildung

$$|A\boldsymbol{x} - A\boldsymbol{y}| = |A(\boldsymbol{x} - \boldsymbol{y})|$$

und wegen der Orthogonalität gilt weiter

$$|A(\boldsymbol{x} - \boldsymbol{y})| = |\boldsymbol{x} - \boldsymbol{y}|.$$

Der Abstand stimmt also für alle $\boldsymbol{y} \in T$ überein, weshalb wiederum auch die Infima übereinstimmen. □

Im folgenden wollen wir auch den Durchmesser bzw. die Länge eines Randstücks verwenden, die wir einfach als Abstand der beiden Endpunkte definieren.

Definition 2.9. *Seien $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}$ und $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Dann heißt*

$$\operatorname{diam}(T) = |\boldsymbol{b} - \boldsymbol{a}|$$

Durchmesser *bzw.* Länge *von T.*

Mit Hilfe von Lemma 2.8 können wir für weitere Betrachtungen annehmen, dass $T = [\boldsymbol{a}, \boldsymbol{b}]$ gegeben ist durch $\boldsymbol{a} = \mathbf{0}$ und $\boldsymbol{b} = (\operatorname{diam}(T), 0)^T$, wie folgendes Lemma zeigt.

Lemma 2.10. *Sei $T = [\boldsymbol{a}, \boldsymbol{b}]$ ein affines Randstück. Seien weiters $\tau_{\boldsymbol{a}} : \boldsymbol{x} \mapsto \boldsymbol{x} - \boldsymbol{a}$ und*

$$A = \frac{1}{\operatorname{diam}(T)} \begin{pmatrix} b_1 - a_1 & b_2 - a_2 \\ -(b_2 - a_2) & b_1 - a_1 \end{pmatrix}$$

Abbildungen. Dann gilt

$$A\tau_{\boldsymbol{a}}(\boldsymbol{a}) = \mathbf{0}, \qquad A\tau_{\boldsymbol{a}}(\boldsymbol{b}) = (\operatorname{diam}(T), 0)^T$$

sowie

$$\operatorname{dist}(\boldsymbol{x}, T) = \operatorname{dist}(A\tau_{\boldsymbol{a}}(\boldsymbol{x}), A\tau_{\boldsymbol{a}}(T)) \quad \textit{für alle } \boldsymbol{x} \in \mathbb{R}^2.$$

Beweis. Klarerweise gilt $A\tau_{\boldsymbol{a}}(\boldsymbol{a}) = A\mathbf{0} = \mathbf{0}$. Mit $\tau_{\boldsymbol{a}}\boldsymbol{b} = (b_1 - a_1, b_2 - a_2)^T$ gilt für $A\tau_{\boldsymbol{a}}(\boldsymbol{b})$

$$\begin{aligned} &\frac{1}{\operatorname{diam}(T)} \begin{pmatrix} b_1 - a_1 & b_2 - a_2 \\ -(b_2 - a_2) & b_1 - a_1 \end{pmatrix} \begin{pmatrix} b_1 - a_1 \\ b_2 - a_2 \end{pmatrix} \\ &\quad = \frac{1}{\operatorname{diam}(T)} \begin{pmatrix} (b_1 - a_1)^2 + (b_2 - a_2)^2 \\ -(b_1 - a_1)(b_2 - a_2) + (b_1 - a_1)(b_2 - a_2) \end{pmatrix} \\ &\quad = \frac{1}{\operatorname{diam}(T)} \begin{pmatrix} \operatorname{diam}(T)^2 \\ 0 \end{pmatrix} = \begin{pmatrix} \operatorname{diam}(T) \\ 0 \end{pmatrix}. \end{aligned}$$

Zu zeigen bleibt noch, dass $\operatorname{dist}(\boldsymbol{x}, T) = \operatorname{dist}(A\tau_{\boldsymbol{a}}(\boldsymbol{x}), A\tau_{\boldsymbol{a}}(T))$ für alle $\boldsymbol{x} \in \mathbb{R}^2$. Dies folgt mit Lemma 2.8, denn $\tau_{\boldsymbol{a}}$ ist eine Translationsabbildung und A ist eine orthogonale Matrix, wie folgende Rechnung zeigt:

$$\begin{aligned} AA^T &= \frac{1}{\operatorname{diam}(T)^2} \begin{pmatrix} b_1 - a_1 & b_2 - a_2 \\ -(b_2 - a_2) & b_1 - a_1 \end{pmatrix} \begin{pmatrix} b_1 - a_1 & -(b_2 - a_2) \\ b_2 - a_2 & b_1 - a_1 \end{pmatrix} \\ &= \frac{1}{\operatorname{diam}(T)^2} \begin{pmatrix} (b_1 - a_1)^2 + (b_2 - a_2)^2 & 0 \\ 0 & (b_1 - a_1)^2 + (b_2 - a_2)^2 \end{pmatrix} \\ &= \frac{1}{\operatorname{diam}(T)^2} \begin{pmatrix} \operatorname{diam}(T)^2 & 0 \\ 0 & \operatorname{diam}(T)^2 \end{pmatrix} = I_2 \end{aligned}$$

Somit folgt die Behauptung. □

Satz 2.11. *Seien $\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2$ und $T = [\boldsymbol{a}, \boldsymbol{b}]$. Dann existiert ein eindeutiges $\boldsymbol{y} \in T$, sodass $|\boldsymbol{x} - \boldsymbol{y}| = \operatorname{dist}(\boldsymbol{x}, T)$.*

Beweis. Wegen Lemma 2.10 kann man ohne Beschränkung der Allgemeinheit annehmen, dass $\boldsymbol{a} = \boldsymbol{0}$ und $\boldsymbol{b} = (\operatorname{diam}(T), 0)^T$.
Um die Behauptung zu zeigen, geben wir in Abhängigkeit von $\boldsymbol{x}$ ein $\boldsymbol{y} \in T$ an und zeigen, dass $|\boldsymbol{x} - \boldsymbol{y}| < |\boldsymbol{x} - \boldsymbol{z}|$ für alle $\boldsymbol{z} \in T \backslash \{\boldsymbol{y}\}$ ist. Dies impliziert sowohl, dass das Infimum in einem Punkt tatsächlich angenommen wird, wie auch, dass dieser Punkt eindeutig ist. Hierzu unterscheidet man in Abhängigkeit von $\boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2$ drei Fälle:

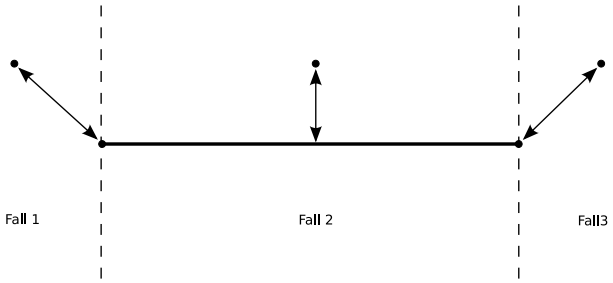


Abbildung 1: Die drei zu unterscheidenden Fälle

Fall 1: $x_1 < 0$. In diesem Fall ist der eindeutige Punkt gegeben als $\boldsymbol{y} = (0, 0)^T$, wie wir zeigen werden. Sei $\boldsymbol{z} = (z_1, 0)^T \in T$ ein weiterer Punkt des affinen Randstücks, dann ist klarerweise $z_1 > 0$.
Nun gilt

$$|\boldsymbol{y} - \boldsymbol{x}| = \sqrt{(-x_1)^2 + (-x_2)^2} < \sqrt{(z_1 - x_1)^2 + (-x_2)^2} = |\boldsymbol{z} - \boldsymbol{x}|,$$

womit die Behauptung gezeigt ist.
Fall 2: $x_1 \in [0, \operatorname{diam}(T)]$. In diesem Fall wählt man $\boldsymbol{y} = (x_1, 0)^T$. Sei $\boldsymbol{z} \in T$ wiederum ein anderer Punkt aus T, dann gilt

$$|\boldsymbol{y} - \boldsymbol{x}| = x_2 < \sqrt{(z_1 - x_1)^2 + x_2^2} = |\boldsymbol{z} - \boldsymbol{x}|$$

für $z_1 \neq x_1$.

Fall 3: $x_1 > \operatorname{diam}(T)$. Der eindeutige Punkt ist gegeben durch $\boldsymbol{y} = (\operatorname{diam}(T), 0)^T$. Sei $\boldsymbol{z} \in T \backslash \{\boldsymbol{y}\}$ ein weiterer Punkt, dann ist klarerweise $z_1 < y_1$. Es gilt

$$|\boldsymbol{y} - \boldsymbol{x}| = \sqrt{(y_1 - x_1)^2 + x_2^2} < \sqrt{(z_1 - x_1)^2 + x_2^2} = |\boldsymbol{z} - \boldsymbol{x}|.$$

Insgesamt folgt die Existenz jenes eindeutigen Punktes $\boldsymbol{y} \in T$, in dem der Abstand zu einem weiteren Punkt $\boldsymbol{x}$ angenommen wird. □

Wir bemerken, dass die Distanzabbildung für festes T in $\boldsymbol{x}$ stetig bezüglich der euklidischen Topologie ist.

Lemma 2.12. *Sei T ein beliebiges, aber festes, affines Randstück. Dann ist die Abbildung* $\operatorname{dist}(\cdot, T) : \mathbb{R}^2 \to \mathbb{R}$ *Lipschitz-stetig mit Lipschitz-Konstante* 1.

Beweis. Seien $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^2$ fest aber beliebig. Nach Satz 2.11 existieren eindeutige Punkte $\boldsymbol{z}, \boldsymbol{z}' \in T$, sodass $\operatorname{dist}(\boldsymbol{x}, T) = \operatorname{dist}(\boldsymbol{x}, \boldsymbol{z}) = |\boldsymbol{x} - \boldsymbol{z}|$ und $\operatorname{dist}(\boldsymbol{y}, T) = \operatorname{dist}(\boldsymbol{y}, \boldsymbol{z}') = |\boldsymbol{x} - \boldsymbol{z}'|$. Wegen der Minimalitätseigenschaft des Abstandes und wegen der Dreiecksungleichung gilt einerseits

$$|\boldsymbol{x} - \boldsymbol{z}| - |\boldsymbol{y} - \boldsymbol{z}'| \leq |\boldsymbol{x} - \boldsymbol{z}'| - |\boldsymbol{y} - \boldsymbol{z}'| = |\boldsymbol{x} - \boldsymbol{y} + \boldsymbol{y} - \boldsymbol{z}'| - |\boldsymbol{y} - \boldsymbol{z}'| \leq |\boldsymbol{x} - \boldsymbol{y}|$$

und andererseits

$$|\boldsymbol{y} - \boldsymbol{z}'| - |\boldsymbol{x} - \boldsymbol{z}| \leq |\boldsymbol{y} - \boldsymbol{z}| - |\boldsymbol{x} - \boldsymbol{z}| \leq |\boldsymbol{x} - \boldsymbol{y}|.$$

Insgesamt erhält man so

$$|\operatorname{dist}(\boldsymbol{x}, T) - \operatorname{dist}(\boldsymbol{y}, T)| = ||\boldsymbol{x} - \boldsymbol{z}| - |\boldsymbol{y} - \boldsymbol{z}'|| \leq |\boldsymbol{x} - \boldsymbol{y}|,$$

womit die Behauptung folgt. □

Abschließen möchte ich diesen Teil mit der Angabe eines Algorithmus zur Berechnung des Abstandes zwischen einem Punkt und einer Strecke. Algorithmus 2.1 ist dem Beweis von Satz 2.11 nachempfunden.
Zunächst wird dieselbe Transformation durchgeführt, wie in Lemma 2.10 Die anfängliche Berechnung von $\boldsymbol{y}$ und $\boldsymbol{z}$ entspricht hierbei der Translation $\tau_{\boldsymbol{a}}$, die darauf folgende Berechnung von y_1 und y_2 der Drehung.
Dann werden dieselben drei Fälle unterschieden, wie im Beweis zu Satz 2.11.

2.2 Abstand zwischen zwei affinen Randstücken

Ziel dieses Abschnittes ist das Finden eines Algorithmus zur Bestimmung des Abstandes zwischen zwei Strecken.

Definition 2.13. *Seien $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{a}_k, \boldsymbol{b}_k \in \mathbb{R}^2$ mit $\boldsymbol{a}_j \neq \boldsymbol{b}_j$ und $\boldsymbol{a}_k \neq \boldsymbol{b}_k$. Seien weiters $T_j = [\boldsymbol{a}_j, \boldsymbol{b}_j], T_k = [\boldsymbol{a}_k, \boldsymbol{b}_k]$ zwei affine Randstücke. Dann ist der Abstand zwischen T_j und T_k definiert durch*

$$\operatorname{dist}(T_j, T_k) := \inf \{|\boldsymbol{x} - \boldsymbol{y}| \;\; | \;\; \boldsymbol{x} \in T_j, \boldsymbol{y} \in T_k\}$$

Algorithm 1 Abstandsbestimmung zwischen Punkt und affinem Randstück

function GETDISTANCE($\boldsymbol{x}$, $\boldsymbol{a}$, $\boldsymbol{b}$)
  $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{a}$;
  $\boldsymbol{z} = \boldsymbol{b} - \boldsymbol{a}$;
  $n = |\boldsymbol{z}|$;
  $y_1 = \frac{z_1}{n} y_1 - \frac{z_2}{n} y_2$;
  $y_2 = \frac{z_1}{n} y_1 + \frac{z_2}{n} y_2$;
  if $y_1 < 0$ **then**
    return $\sqrt{y_1^2 + y_2^2}$;
  else if $y_1 > n$ **then**
    return $\sqrt{(y_1 - n)^2 + y_2^2}$;
  else
    return y_2;
  end if
end function

Bemerkung 2.14. *Klarerweise gilt*

$$\inf \{ |\boldsymbol{x} - \boldsymbol{y}| \quad | \quad \boldsymbol{x} \in T_j, \boldsymbol{y} \in T_k \} = \inf \{ \operatorname{dist}(\boldsymbol{x}, T_k) \quad | \quad \boldsymbol{x} \in T_j \}. \tag{5}$$

Die Infima in (5) sind sogar Minima, denn laut Bemerkung 2.2 ist T_j kompakt, und laut Bemerkung 2.12 ist $\operatorname{dist}(\cdot, T_k)$ *stetig. Daher ist* $\{ \operatorname{dist}(\boldsymbol{x}, T_k) \mid \boldsymbol{x} \in T_j \}$ *kompakt.*

In weiterer Folge ist zu zeigen, dass es genügt, die Abstände der vier Eckpunkte zum jeweils anderen affinen Randstück zu betrachten, falls die Randstücke nicht schneiden. Zunächst aber wenden wir uns dem Fall zu, dass sie einander schneiden, also dass der Abstand zwischen ihnen gleich 0 ist.

Lemma 2.15. *Seien $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{a}_k, \boldsymbol{b}_k \in \mathbb{R}^2$ und $T_j = [\boldsymbol{a}_j, \boldsymbol{b}_j]$, $T_k = [\boldsymbol{a}_k, \boldsymbol{b}_k]$. Dann schneiden sich T_j und T_k genau dann, wenn das Gleichungssystem*

$$\left(\boldsymbol{b}_j - \boldsymbol{a}_j, \boldsymbol{a}_k - \boldsymbol{b}_k\right) \boldsymbol{x} = \boldsymbol{a}_k + \boldsymbol{b}_k - \boldsymbol{a}_j - \boldsymbol{b}_j$$

eine Lösung $\boldsymbol{x}$ hat mit $\|\boldsymbol{x}\|_\infty \leq 1$.

Beweis. Dass die affinen Randstücke

$$T_j = \left\{ \frac{1}{2} \left(\boldsymbol{a}_j + \boldsymbol{b}_j + \lambda_j (\boldsymbol{a}_j + \boldsymbol{b}_j) \right) \quad \middle| \quad \lambda_j \in [-1, 1] \right\},$$
$$T_k = \left\{ \frac{1}{2} \left(\boldsymbol{a}_k + \boldsymbol{b}_k + \lambda_k (\boldsymbol{a}_k + \boldsymbol{b}_k) \right) \quad \middle| \quad \lambda_k \in [-1, 1] \right\}$$

einander in einem Punkt schneiden, ist äquivalent zu der Tatsache, dass $\lambda_j, \lambda_k \in [-1, 1]$ existieren mit

$$\frac{1}{2} \left(\boldsymbol{a}_j + \boldsymbol{b}_j + \lambda_j (\boldsymbol{a}_j + \boldsymbol{b}_j) \right) = \frac{1}{2} \left(\boldsymbol{a}_k + \boldsymbol{b}_k + \lambda_k (\boldsymbol{a}_k + \boldsymbol{b}_k) \right).$$

Durch elementare Umformungen erhält man

$$\lambda_j (\boldsymbol{b}_j - \boldsymbol{a}_j) - \lambda_k (\boldsymbol{b}_k - \boldsymbol{a}_k) = \boldsymbol{a}_k + \boldsymbol{b}_k - \boldsymbol{a}_j - \boldsymbol{b}_j.$$

Schreibt man dies mit $\boldsymbol{x} = (\lambda_j, \lambda_k)^T$ als Gleichungssystem, so erhält man

$$\left(\boldsymbol{b}_j - \boldsymbol{a}_j, \boldsymbol{a}_k - \boldsymbol{b}_k\right) \boldsymbol{x} = \boldsymbol{a}_k + \boldsymbol{b}_k - \boldsymbol{a}_j - \boldsymbol{b}_j.$$

$\lambda_j, \lambda_k \in [-1, 1]$ ist klarerweise äquivalent zu $\|\boldsymbol{x}\|_\infty \leq 1$. □

Der folgende Satz liefert nun die Grundlage für den gesuchten Algorithmus.

Satz 2.16. *Seien $T_j = [\boldsymbol{a}_j, \boldsymbol{b}_j], T_k = [\boldsymbol{a}_k, \boldsymbol{b}_k]$ zwei affine Randstücke mit leerem Schnitt. Dann gilt*

$$\operatorname{dist}(T_j, T_k) = \min\left\{\operatorname{dist}(\boldsymbol{a}_j, T_k), \operatorname{dist}(\boldsymbol{b}_j, T_k), \operatorname{dist}(\boldsymbol{a}_k, T_j), \operatorname{dist}(\boldsymbol{b}_k, T_j)\right\}$$

Beweis. Ohne Beschränkung der Allgemeinheit können wir mit Lemma 2.10 annehmen, dass $\boldsymbol{a}_j = \boldsymbol{0}$ und $b_{j,2} = 0$, d.h. $T_j = [0, b_{j,1}] \times \{0\}$.
Wegen Bemerkung 2.14 wird der Abstand zwischen den affinen Randstücken tatsächlich angenommen. Sei nun $\boldsymbol{x} \in T_k$ ein Punkt für den $\operatorname{dist}(\boldsymbol{x}, T_j) = \operatorname{dist}(T_k, T_j)$ gilt.
Fall 1: Ist $x_1 \leq 0$, so wird der Abstand zu T_j laut dem Beweis von Satz 2.11 eindeutig im Punkt $\boldsymbol{a}_j = \boldsymbol{0}$ angenommen.

$$\operatorname{dist}(T_k, T_j) = \operatorname{dist}(\boldsymbol{x}, T_j) = \operatorname{dist}(\boldsymbol{x}, \boldsymbol{a}_j) = \operatorname{dist}(\boldsymbol{x}, \boldsymbol{0}) = |\boldsymbol{x}|.$$

Fall 2: Ist $x_1 \geq b_{j,1}$, so gilt wiederum laut dem Beweis von Satz 2.11

$$\operatorname{dist}(T_k, T_j) = \operatorname{dist}(\boldsymbol{x}, T_j) = \operatorname{dist}(\boldsymbol{x}, \boldsymbol{b}_j).$$

Fall 3: Sei nun $x_1 \in (0, b_{j,1})$. Der Abstand ist dann klarerweise gegeben durch $\operatorname{dist}(\boldsymbol{x}, T_j) = |x_2|$. Ist $\boldsymbol{x} = \boldsymbol{a}_k$ oder $\boldsymbol{x} = \boldsymbol{b}_k$, so ist nichts mehr zu zeigen.
Sei $\gamma_k(\lambda) = \frac{1}{2}(\boldsymbol{a}_k + \boldsymbol{b}_k + \lambda(\boldsymbol{b}_k - \boldsymbol{a}_k))$ die zu T_k zugehörige Parametrisierung. Sei weiter $\boldsymbol{x} \in T_k \backslash \{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ und $\lambda_x \in (-1, 1)$ so gegeben, dass $\gamma_k(\lambda_x) = \boldsymbol{x}$ ist. Da γ_k gemäß Bemerkung 2.4 stetig ist, wählen wir nun ein $\delta > 0$, sodass

$$\gamma_k((\lambda_x - \delta, \lambda_x + \delta) \cap (-1, 1)) \subseteq (0, b_{j,1}) \times \mathbb{R}.$$

Es existieren $\lambda_y, \lambda_z \in (\lambda_x - \delta, \lambda_x + \delta) \cap (-1, 1)$, sodass $|\lambda_y| < |\lambda_x| < |\lambda_z|$. Mit $\boldsymbol{y} = \gamma_k(\lambda_y), \boldsymbol{z} = \gamma_k(\lambda_z)$ ist nun wegen Satz 2.11 und wegen $y_1, z_1 \in (0, 1)$ der Abstand zu T_j gegeben durch $|y_2|$ bzw. $|z_2|$. Wegen der Minimalität von $\operatorname{dist}(\boldsymbol{x}, T_j)$ erhält man

$$|x_2| \leq |y_2| = \left|\frac{1}{2}\left(a_{k,2} + b_{k,2} + (b_{k,2} - a_{k,2})\lambda_y\right)\right| \leq |b_{k,2} - a_{k,2}||\lambda_y|,$$
$$|x_2| \leq |z_2| = \left|\frac{1}{2}\left(a_{k,2} + b_{k,2} + (b_{k,2} - a_{k,2})\lambda_z\right)\right| \leq |b_{k,2} - a_{k,2}||\lambda_z|.$$

Durch Subtraktion der beiden Ungleichungen und Herausheben folgt

$$0 \leq |b_{k,2} - a_{k,2}|\left(|\lambda_y| - |\lambda_z|\right).$$

Voraussetzungsgemäß ist $|\lambda_y| - |\lambda_z| < 0$. Somit folgt $|b_{k,2} - a_{k,2}| \leq 0$ bzw. $b_{k,2} - a_{k,2} = 0$. Insbesondere sind die Strecken also parallel. Ist $a_{k,1} \in (0, b_{j,1})$ oder $b_{k,1} \in (0, b_{j,1})$, so gilt wegen Satz 2.11

$$\operatorname{dist}(\boldsymbol{a}_k, T_j) = a_{k,2} = x_2 = \operatorname{dist}(\boldsymbol{x}, T_j) \quad \text{oder} \quad \operatorname{dist}(\boldsymbol{b}_k, T_j) = b_{k,2} = x_2 = \operatorname{dist}(\boldsymbol{x}, T_j).$$

Sei $a_{k,1} \leq b_{k,1}$. Dann ist $a_{k,1} \leq 0$ und $b_{k,1} \geq 0$. Daher gilt

$$-\frac{a_{k,1} + b_{k,1}}{b_{k,1} - a_{k,1}} = -\frac{|b_{k,1}| - |a_{k,1}|}{|b_{k,1}| + |a_{k,1}|} =: \lambda_y \in [-1, 1].$$

Durch elementare Rechnung erhält man

$$\boldsymbol{y} := \gamma_k(\lambda_y) = \begin{pmatrix} 0 \\ a_{k,2} + b_{k,2} \end{pmatrix} = \begin{pmatrix} 0 \\ x_2 \end{pmatrix}.$$

Wegen Satz 2.11 gilt

$$\operatorname{dist}(\boldsymbol{y}, \boldsymbol{a}_j) = \operatorname{dist}(\boldsymbol{y}, T_j) = \operatorname{dist}(\boldsymbol{x}, T_j).$$

Der Fall $b_{k,2} < a_{k,2}$ folgt aus Symmetriegründen analog, womit der Beweis abgeschlossen ist. □

Bemerkung 2.17. *Im Allgemeinen existieren keine eindeutigen Punkte* $\boldsymbol{x} \in T_j, \boldsymbol{y} \in T_k$ *sodass* $\operatorname{dist}(T_j, T_k) = \operatorname{dist}(\boldsymbol{x}, \boldsymbol{y})$. *Man betrachte hierfür beispielsweise zwei parallele Randstücke* $T_j = \left[(0,0)^T, (1,0)^T\right]$, $T_k = \left[(1,0)^T, (1,1)^T\right]$, *wo sogar für alle* $\boldsymbol{x} \in T_j$ *gilt, dass* $\operatorname{dist}(\boldsymbol{x}, T_k) = \operatorname{dist}(T_j, T_k) = 1$.

Algorithm 2 Abstandsbestimmung zwischen zwei affinen Randstücken

```
function getDistance(T_j = [a_j, b_j], T_k = [a_k, b_k])
    if T_j und T_k schneiden einander then
        return 0
    else
        return min { getDistance(a_j, T_k), getDistance(b_j, T_k),
                     getDistance(a_k, T_j), getDistance(b_k, T_j ) }
    end if
end function
```

3 Semi-analytische Berechnung bestimmter Doppelintegrale

Ziel dieses Abschnitts ist die approximative Berechnung des Integrals

$$A_{jk} = \int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}} \tag{6}$$

beziehungsweise als Spezialfall davon die Berechnung des Integrals

$$A_{jk} = \int_{T_j} \int_{T_k} \log|\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}} ds_{\boldsymbol{x}} \tag{7}$$

unter bestimmten Voraussetzungen an die affinen Randstücke T_j, T_k und den Integranden $\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. Dabei wendet man beispielsweise auf das äußere Integral eine interpolatorische Quadraturformel an, was in diesem Fall die *Gauß-Quadratur* sein wird. Die angesprochenen Voraussetzungen garantieren schließlich exponentielle Konvergenz des Approximationsfehlers, was abschließend gezeigt wird.

3.1 Interpolation

In diesem Abschnitt wird die Chebyshev-Interpolation auf einem Intervall $[-1, 1]$ definiert. Im folgenden bezeichnet $\mathcal{P}_p$ die Menge aller Polynome vom Grad p.

Definition 3.1. *Seien $x_0, ..., x_p \in [-1, 1]$ paarweise verschieden* Knoten *und $y_0, ..., y_p$ die zugehörigen* Funktionswerte*. Die* Lagrange'sche Interpolationsaufgabe *lautet dann folgendermaßen: Finde ein Polynom q vom Grad $p - 1$, sodass*

$$q(x_i) = y_i \qquad \text{für alle } i = 0, ..., p.$$

Ferner definieren wir die zu den Knoten $x_0, ..., x_p$ zugehörigen Lagrange-Polynome durch

$$L_j(x) = \prod_{\substack{i=1 \\ i \neq j}}^{p} \frac{x - x_i}{x_j - x_i}.$$

[8, Theorem 1.6] zeigt, dass die Lagrange'sche Interpolationsaufgabe eine eindeutige Lösung besitzt, die gegeben ist durch

$$q = \sum_{j=0}^{p} y_j L_j.$$

Wir definieren den Interpolationsoperator $\mathcal{I}_p u$ durch

$$\mathcal{I}_p u := \sum_{j=0}^{p} u(x_j) L_j.$$

Außerdem definieren wir die Lebesgue-Konstante durch

$$\Lambda_p := \max_{x \in [-1,1]} \sum_{j=0}^{p} |L_j(x)|.$$

[2, Satz 4.16] stellt einen Bezug zwischen der Bestapproximation und der Polynominterpolation her und ist später von Bedeutung.

Satz 3.2. *Sei Λ_p die Lebesgue-Konstante zu p paarweise verschiedenen Knoten $x_1, ..., x_p$, die alle im Intervall $[-1, 1]$ liegen. Dann gilt für jedes $u \in \mathcal{C}[-1, 1]$*

$$\|\mathcal{I}_p u - u\|_\infty \leq (1 + \Lambda_p) \min_{v \in \mathcal{P}_p} \|u - v\|_\infty, \tag{8}$$

wobei $\mathcal{P}_p$ die Menge aller Polynome vom Grad p bezeichnet. □

Eine einfache Fehlerabschätzung für den Interpolationsfehler liefert [8, Theorem 1.17]. Unter der Voraussetzung $u \in \mathcal{C}^{p+1}([-1, 1])$ gilt:

$$\|u - \mathcal{I}_p u\|_{\infty,[-1,1]} \leq \max_{x \in [-1,1]} \left| \prod_{j=0}^{n} (x - x_j) \right| \frac{1}{(p+1)!} \|u^{(p+1)}\|_{\infty,[-1,1]} \tag{9}$$

Bei der Chebyshev-Interpolation wählt man die Knoten in Bezug auf diese Abschätzung optimal, indem man den ausschließlich durch die Knotenwahl bestimmten Term

$$\max_{x \in [-1,1]} \left| \prod_{j=0}^{n} (x - x_j) \right|$$

minimiert. Die Lösung dieses Minimierungsproblems sind die Nullstellen der Chebyshev-Polynome. Diese sind gegeben durch

$$x_j = \cos\left(\frac{2j - 1}{p} \frac{\pi}{2} \right) \qquad \text{für } j = 1, ..., p.$$

Siehe dazu auch [8, Definition 1.22 und Theorem 1.24].
Im folgenden bezeichnet $\mathcal{I}_p$ immer den Chebyshev-Interpolationsoperator

3.2 Gauß-Quadratur

Der Einfachheit halber verwenden wir im Folgenden einen einschränkenden Begriff der Quadratur, indem wir bei der Definition auf die Gewichtsfunktion verzichten bzw. annehmen, dass diese die konstante 1-Funktion ist. Man spricht dann auch von der *klassischen Gauß-Quadratur*.
Unter *Quadratur* versteht man die approximative Berechnung eines Integrals der Form

$$\int_{-1}^{1} f(x) dx,$$

indem man es durch eine Summe

$$Q_n(f) := \sum_{k=0}^{n} \omega_k f(x_k)$$

ersetzt und diese anschließend auswertet. Die x_k heißen hierbei *Knoten*, die ω_k *Gewichte*. Eine Quadratur heißt *exakt* für eine Funktion f, falls $Q_n(f) = \int_{-1}^{1} f(x) dx$ ist.

Eine Quadratur hat *Exaktheitsgrad* m, wenn sie für alle Polynome bis zum Grad m exakt ist.
Eine Quadratur heißt *interpolatorisch* (vom Grad n), falls für jede integrierbare, auf $(-1,1)$ stetige Funktion f

$$Q_n(f) = \int_{-1}^{1} p(x)dx$$

gilt, wobei p das Interpolationspolynom von f vom Grad n zu den Knoten $x_0, ..., x_n$ ist.

3.3 Semi-analytische Berechnung des Doppelintegrals

Ehe wir uns der (exponentiell schnellen) Konvergenz der semi-analytischen Berechnung zuwenden, müssen einige der Voraussetzungen definiert werden, die hierfür benötigt werden.

Definition 3.3. *Seien T_j, T_k affine Randstücke und sei $\eta > 0$ fest. Dann heißt das Paar (T_j, T_k)* η-zulässig*, genau dann wenn*

$$\eta \operatorname{dist}(T_j, T_k) \geq \min\{\operatorname{diam}(T_j), \operatorname{diam}(T_k)\}. \tag{10}$$

Andernfalls heißt das Paar (T_j, T_k) unzulässig.

Bei den betrachteten Doppel-Integralen sollen insbesondere Integranden mit einer Singularität auf der Diagonalen zugelassen werden, die aber ansonsten glatt sind. Die folgende Abschätzung ermöglicht dann die hinreichende Kontrolle der partiellen Ableitungen trotz dieser Singularität.

Definition 3.4. *Die Kern-Funktion $\kappa(\boldsymbol{x}, \boldsymbol{y})$ heißt* asymptotisch glatt*, falls sie glatt ist für $\boldsymbol{x} \neq \boldsymbol{y}$ und Konstanten $c_1, c_2 > 0$ und eine* Ordnung der Singularität *$s \in \mathbb{R}$ existieren, sodass*

$$|\partial_x^\alpha \partial_y^\beta \kappa(\boldsymbol{x}, \boldsymbol{y})| \leq c_1 (c_2 |\boldsymbol{x} - \boldsymbol{y}|)^{-(|\alpha|+|\beta|+s)} (\alpha + \beta)! \tag{11}$$

für alle Multiindizes $\alpha, \beta \in \mathbb{N}_0^d$ mit $|\alpha| + |\beta| \geq 1$ gilt.

Es zeigt sich, dass asymptotisch glatte Kernfunktionen besonders gut durch Polynome interpoliert werden können. Das wesentliche Argument hierfür wird [3, Lemma 3.13] sein:

Lemma 3.5. *Sei $J \subseteq \mathbb{R}$ ein abgeschlossenes und beschränktes Intervall und angenommen die Funktion u erfüllt für Konstanten $C_u, \gamma_u \geq 0$*

$$\|u^{(n)}\|_{\infty,J} \leq C_u \gamma_u^n n! \qquad \text{für alle } n \in \mathbb{N}_0. \tag{12}$$

Dann gilt für alle $k \in \mathbb{N}_0$

$$\min_{v \in \mathcal{P}_k} \|u - v\|_{\infty,J} \leq C_u 4e(1 + \gamma_u |J|)(k+1) \left(1 + \frac{2}{\gamma_u |J|}\right)^{-(k+1)}. \tag{13}$$

□

Lemma 3.6. *Seien T_j, T_k zwei η-zulässige, affine Randstücke mit* $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$. *Sei κ eine asymptotisch glatte Kernfunktion mit Konstanten $c_1, c_2 > 0$ und Singularitätsordnung $s \geq 0$. Dann gilt mit*

$$C_{\eta,j,k} := 4e\frac{c_1}{(c_2\operatorname{dist}(T_j,T_k))^s}\left(1+\frac{\eta}{c_2}\right) \tag{14}$$

die Abschätzung

$$\sup_{\boldsymbol{y}\in T_k}\|\kappa(\cdot,\boldsymbol{y})-\mathcal{I}_p\kappa(\cdot,\boldsymbol{y})\|_{\infty,T_j} \leq C_{\eta,j,k}(1+\Lambda_p)(p+1)\left(1+\frac{2c_2}{\eta}\right)^{-(p+1)} \tag{15}$$

Beweis. Wir definieren die Konstanten

$$C_\kappa := \frac{c_1}{(c_2\operatorname{dist}(T_j,T_k))^s} \geq 0 \qquad \text{und} \qquad \gamma_\kappa := \frac{\eta}{2c_2} \geq 0$$

und stellen fest, dass wir für die Konstante $C_{\eta,j,k}$ dann kurz

$$C_{\eta,j,k} = 4eC_\kappa\,(1+2\gamma_\kappa)$$

schreiben können.
Sei $\boldsymbol{y} \in T_k$ fest gewählt. Bezeichnet γ_j die Parametrisierung von T_j, so gilt wegen $|\gamma_j'| = \left|\frac{\boldsymbol{b}_j-\boldsymbol{a}_j}{2}\right| = \frac{\operatorname{diam}(T_j)}{2}$ und der Kettenregel

$$|\partial_{\boldsymbol{x}}^n\kappa| = |\partial_x^n(\kappa(\gamma_j(x),\boldsymbol{y}))| = \left(\frac{\operatorname{diam}(T_j)}{2}\right)^n |(\partial_x^n\kappa)(\gamma_j(x),\boldsymbol{y})|\,.$$

Mithilfe von (11) und den Konstanten C_κ, γ_κ gilt die Abschätzung

$$\begin{aligned}
&\left|\left(\frac{\operatorname{diam}(T_j)}{2}\right)^n(\partial_x^n\kappa)(\gamma_j(x),\boldsymbol{y})\right| \\
&\qquad\leq \left(\frac{\operatorname{diam}(T_j)}{2}\right)^n c_1(c_2|\gamma_j(x)-\boldsymbol{y}|)^{-n-s}n! \\
&\qquad= \left(\frac{\operatorname{diam}(T_j)}{2}\right)^n c_1(c_2|\boldsymbol{x}-\boldsymbol{y}|)^{-n-s}n! \\
&\qquad= \frac{c_1}{(c_2|\boldsymbol{x}-\boldsymbol{y}|)^s}\left(\frac{\operatorname{diam}(T_j)}{2c_2|\boldsymbol{x}-\boldsymbol{y}|}\right)^n n! \\
&\qquad\leq C_\kappa\gamma_\kappa^n n!.
\end{aligned}$$

T_j und T_k sind zulässig, weshalb κ auf $T_j \times T_k$ definitionsgemäß glatt ist. Daher sind die Voraussetzungen für Lemma 3.5 erfüllt, und es gilt

$$\min_{v\in\mathcal{P}^p}\|\kappa(\gamma_j(\cdot),\boldsymbol{y})-v\|_{\infty,[-1,1]} \leq C_\kappa 4e(1+2\gamma_\kappa)(p+1)\left(1+\frac{1}{\gamma_\kappa}\right)^{-(p+1)} \tag{16}$$

$$= C_{\eta,j,k}(p+1)\left(1+\frac{1}{\gamma_\kappa}\right)^{-(p+1)} \tag{17}$$

für alle $p \in \mathbb{N}_0$, wobei $\mathcal{P}^p$ die Menge aller Polynome vom Grad p ist. Aus (17) folgt weiter

$$\min_{v \in \mathcal{P}^p} \|\kappa(\gamma_j(\cdot), \boldsymbol{y}) - v\|_{\infty,[-1,1]} \leq C_{\eta,j,k}(p+1) \left(1 + \frac{2c_2}{\eta}\right)^{-(p+1)}.$$

Mithilfe von Satz 3.2 folgt

$$\|\kappa(\cdot, \boldsymbol{y}) - \mathcal{I}_p \kappa(\cdot, \boldsymbol{y})\|_{\infty,T_j} \leq C_{\eta,j,k}(1+\Lambda_p)(p+1) \left(1 + \frac{2c_2}{\eta}\right)^{-(p+1)}.$$

Da $\boldsymbol{y}$ beliebig in T_k war und die rechte Seite von $\boldsymbol{y}$ unabhängig ist, folgt die Behauptung für das Supremum, womit der Beweis von (15) abgeschlossen ist. □

Mit diesen Erkenntnissen können wir nun den folgenden Satz zeigen.

Satz 3.7. *Seien T_j, T_k zwei η-zulässige affine Randstücke mit zugehörigen Parametrisierungen γ_j, γ_k und $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$. Sei weiters $\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} : (\boldsymbol{x}, \boldsymbol{y}) \mapsto \kappa(\boldsymbol{x}, \boldsymbol{y})$ eine asymptotisch glatte Kernfunktion mit Konstanten c_1, c_2, s. Sei*

$$\widetilde{C}_{\eta,j,k} := 8e \frac{c_1 \operatorname{diam}(T_j) \operatorname{diam}(T_k)}{(c_2 \operatorname{dist}(T_j, T_k))^s} \left(1 + \frac{\eta}{c_2}\right). \tag{18}$$

Dann gilt für das Integral

$$\begin{aligned} A_{jk} &= \int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}} && (19) \\ &= \frac{\operatorname{diam}(T_j)}{2} \int_{-1}^{1} \int_{T_k} \kappa(\gamma_j(x), \boldsymbol{y}) ds_{\boldsymbol{y}} dx && (20) \end{aligned}$$

und für den durch Gauß-Quadratur von A_{jk} zum Grad p entstehenden Term

$$(A_p)_{jk} = \frac{\operatorname{diam}(T_j)}{2} \sum_{\ell=0}^{p} \omega_\ell \int_{T_k} \kappa(\gamma_j(x_\ell), \boldsymbol{y}) ds_{\boldsymbol{y}} \tag{21}$$

die Abschätzung

$$|A_{jk} - (A_p)_{jk}| \leq \widetilde{C}_{\eta,j,k}(p+1)(1+\Lambda_p) \left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}. \tag{22}$$

Beweis. Die Gauß-Quadratur ist interpolatorisch mit Exaktheitsgrad $2p+1$. Siehe dazu [8, Korollar 6.38], wobei jedoch beachtet werden muss, dass dort in der Quadraturformel die Summation bei 1 beginnt anstatt bei 0. Da die Gauß-Quadratur interpolatorisch vom Grad $2p+1$ ist, gilt

$$\begin{aligned} (A_p)_{jk} &= \frac{\operatorname{diam}(T_j)}{2} \sum_{\ell=0}^{p} \omega_\ell \int_{T_k} \kappa(x_\ell, \boldsymbol{y}) ds_{\boldsymbol{y}} \\ &= \frac{\operatorname{diam}(T_j)}{2} \int_{-1}^{1} \int_{T_k} \mathcal{I}_{2p+1} \kappa(\gamma_j(x), \boldsymbol{y}) ds_{\boldsymbol{y}} dx, \end{aligned}$$

wobei $\mathcal{I}_{2p+1}$ den Chebyshev-Interpolationsoperator vom Grad $2p+1$ in x bezeichnet. Wegen der Additivität des Integrals und durch Hineinziehen des Betrages erhält man

$$\begin{aligned}
&|A_{jk} - (A_p)_{jk}| \\
&\quad = \frac{\operatorname{diam}(T_j)}{2} \left| \int_{-1}^{1} \int_{T_k} \kappa(\gamma_j(x), \boldsymbol{y}) ds_{\boldsymbol{y}} dx - \int_{-1}^{1} \int_{T_k} \mathcal{I}_{2p+1} \kappa(\gamma_j(x), \boldsymbol{y}) ds_{\boldsymbol{y}} dx \right| \\
&\quad = \frac{\operatorname{diam}(T_j)}{2} \left| \int_{-1}^{1} \int_{T_k} \kappa(\gamma_j(x), \boldsymbol{y}) - \mathcal{I}_{2p+1} \kappa(\gamma_j(x), \boldsymbol{y}) ds_{\boldsymbol{y}} dx \right| \\
&\quad \leq \frac{\operatorname{diam}(T_j)}{2} \int_{-1}^{1} \int_{T_k} \left| \kappa(\gamma_j(x), \boldsymbol{y}) - \mathcal{I}_{2p+1} \kappa(\gamma_j(x), \boldsymbol{y}) \right| ds_{\boldsymbol{y}} dx \\
&\quad = \operatorname{diam}(T_j) \operatorname{diam}(T_k) \sup_{\boldsymbol{y} \in T_k} \left\| \kappa(\gamma_j(\cdot), \boldsymbol{y}) - \mathcal{I}_{2p+1} \kappa(\gamma_j(\cdot), \boldsymbol{y}) \right\|_{\infty, [-1,1]}.
\end{aligned}$$

Wir wenden nun das Ergebnis aus Lemma 3.6 an und erhalten

$$\begin{aligned}
|A_{jk} - (A_p)_{jk}| &\leq C_{\eta,j,k} \operatorname{diam}(T_j) \operatorname{diam}(T_k) \ (1+\Lambda_p) 2(p+1) \left(1 + \frac{2c_2 \operatorname{dist}(T_j, T_k)}{\operatorname{diam}(T_j)} \right)^{-2(p+1)} \\
&= \widetilde{C}_{\eta,j,k} (1+\Lambda_p)(p+1) \left(1 + \frac{2c_2 \operatorname{dist}(T_j, T_k)}{\operatorname{diam}(T_j)} \right)^{-2(p+1)},
\end{aligned}$$

womit der Beweis abgeschlossen ist. □

Das folgende Lemma wird es uns ermöglichen, eine Abschätzung zu finden, wenn nicht $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$ sondern $\operatorname{diam}(T_k) < \operatorname{diam}(T_j)$ gilt und sonst die Voraussetzungen aus Satz 3.7 erfüllt sind.

Lemma 3.8. *Seien T_j, T_k η-zulässige Randstücke, κ eine asymptotisch glatte Kernfunktion und*

$$A_{jk} := \int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}}.$$

Dann gilt

$$A_{jk} = A_{kj}.$$

Beweis. Da T_j und T_k zulässig sind, folgt aus der Zulässigkeitsbedingung (10)

$$\operatorname{dist}(T_j, T_k) \geq \frac{\min\{\operatorname{diam}(T_j), \operatorname{diam}(T_k)\}}{\eta} > 0.$$

Aus Definition 2.13, der Definition des Abstandes zwischen affinen Randstücken, folgt $T_j \cap T_k = \emptyset$. Da $\kappa(\boldsymbol{x}, \boldsymbol{y})$ asymptotisch glatt ist, ist sie insbesondere glatt auf $T_j \times T_k$. Die Menge $T_j \times T_k$ ist beschränkt und abgeschlossen und daher nach dem Satz von Heine-Borel kompakt. Nach [6, Seite 249, Satz 6] ist κ als stetige Abbildung auf einem kompakten Intervall auf $T_j \times T_k$ integrierbar. Nach dem Satz von Fubini (s. [6, Seite 289]) gilt daher

$$\int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}} = \int_{T_k} \int_{T_j} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{x}} ds_{\boldsymbol{y}},$$

weshalb die Behauptung folgt. □

Bemerkung 3.9. *Lemma 3.8 gilt unter bestimmten Voraussetzungen an κ auch falls die Randstücke T_j und T_k unzulässig sind. Beispielsweise ist $\kappa(\boldsymbol{x},\boldsymbol{y}) := \log|\boldsymbol{x}-\boldsymbol{y}|$ lokal, d.h. über jedes Kompaktum, integrierbar auf ganz $\mathbb{R}^2\times\mathbb{R}^2$. Klarerweise ist diese Funktion dann insbesondere über $T_j\times T_k$ integrierbar für alle T_j, T_k, da wir im Beweis zu Lemma 3.8 bereits angemerkt haben, dass diese Menge kompakt ist.*

Bemerkung 3.10. *Seien T_j, T_k zwei η-zulässige Randstücke. Ist* $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$ *kann Satz 3.7 angewandt werden, um A_{jk} zu approximieren. Ist* $\operatorname{diam}(T_k) < \operatorname{diam}(T_j)$, *so kann A_{jk} approximiert werden, indem wir $A_{jk} = A_{kj}$ benutzen und anschließend A_{kj} mit Hilfe von Satz 3.7 approximieren.*

Bemerkung 3.11. *Der Ausdruck $1+\Lambda_p$ wächst für die Chebyshev-Interpolation im wesentlichen logarithmisch in p. Daher konvergiert $(A_p)_{jk}$ für wachsenden Quadraturgrad p exponentiell schnell gegen A_{jk}.*

3.4 Approximierende Matrix

Abschließend betrachten wir die Situation, dass $T_1, T_2, ..., T_n$ affine Randstücke sind. Wir betrachten die Matrizen A, A_p, wobei die Einträge von A genau durch (19) gegeben sind und A_p eine A approximierende Matrix ist.
Bei der Definition der approximierenden Matrix unterscheiden wir im Sinne von Bemerkung 3.10 drei Fälle.

Definition 3.12. *Seien $T_1, T_2, ..., T_n$ affine Randstücke und $A \in \mathbb{R}^{n\times n}$ gegeben durch* (19). *Dann ist die A* approximierende Matrix *A_p folgendermaßen definiert.*

- *Sind T_j* und *T_k* unzulässig, *so ist*

$$(A_p)_{jk} := A_{jk}. \tag{23}$$

- *Sind T_j* und *T_k* zulässig und $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$, *so ist*

$$(A_p)_{jk} := \frac{\operatorname{diam}(T_j)}{2} \sum_{\ell=0}^{p} \omega_\ell \int_{T_k} \kappa(\gamma_j(x_\ell), \boldsymbol{y}) ds_{\boldsymbol{y}}. \tag{24}$$

- *Sind hingegen T_j* und *T_k* zulässig und $\operatorname{diam}(T_j) > \operatorname{diam}(T_k)$, *so ist*

$$(A_p)_{jk} := \frac{\operatorname{diam}(T_k)}{2} \sum_{\ell=0}^{p} \omega_\ell \int_{T_j} \kappa(\boldsymbol{x}, \gamma_k(y_\ell)) ds_{\boldsymbol{x}}. \tag{25}$$

Wir wollen nun zeigen, dass die approximierende Matrix bezüglich der Frobenius-Norm für wachsenden Quadraturgrad exponentiell schnell gegen die gegebene Matrix konvergiert. Die Frobenius-Norm ist hierbei gegeben durch

$$\|A\|_{\mathcal{F}} := \left(\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}^2\right)^{1/2} \qquad \text{für} \qquad A \in \mathbb{R}^{n\times n}.$$

Satz 3.13. *Seien $T_1, T_2, ..., T_n$ affine Randstücke. Sei $\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ eine asymptotisch glatte Kernfunktion mit Konstanten c_1, c_2 und Singularitätsordnung $s \geq 0$. Sei $\widetilde{C}_{\eta,j,k}$ wie in (18). Seien $A \in \mathbb{R}^{n\times n}$ eine Matrix, deren Einträge gegeben sind durch*

$$A_{jk} := \int_{T_j} \int_{T_k} \kappa(\boldsymbol{x}, \boldsymbol{y}) ds_{\boldsymbol{y}} ds_{\boldsymbol{x}},$$

und sei A_p die A approximierende Matrix gemäß Definition 3.12. Dann gilt

$$\|A - A_p\|_{\mathcal{F}} \leq n \max_{j=1,...,n} \max_{k=1,...,n} \widetilde{C}_{\eta,j,k}(p+1)(1+\Lambda_p) \left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}.$$

Beweis. Wir betrachten die Differenz zwischen A und A_p in einem festen Eintrag $(A - A_p)_{jk}$. Sind T_j und T_k unzulässig, so ist die Differenz definitionsgemäß 0.
Sind T_j und T_k hingegen zulässig, so unterscheiden wir zwei Fälle. Ist $\operatorname{diam}(T_j) \leq \operatorname{diam}(T_k)$, so ist $(A_p)_{jk}$ gegeben durch (24). Nach Satz 3.7 gilt

$$|A_{jk} - (A_p)_{jk}| \leq \widetilde{C}_{\eta,j,k}(p+1)(1+\Lambda_p) \left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}. \tag{26}$$

Ist hingegen $\operatorname{diam}(T_j) > \operatorname{diam}(T_k)$, so ist wegen Lemma 3.8 $A_{jk} = A_{kj}$. $(A_p)_{jk}$ ist gegeben durch (25). Darauf können wir nun wieder Satz 3.7 anwenden und erhalten dieselbe Abschätzung wie in (26). Insgesamt gilt also für beliebiges j, k die Abschätzung (26). Somit erhält man

$$\begin{aligned}
\|A - A_p\|_{\mathcal{F}}^2 &= \sum_{j,k=1}^{n} (A_{jk} - (A_p)_{jk})^2 \\
&\leq \sum_{j,k=1}^{n} \widetilde{C}_{\eta,j,k}^2 \left((p+1)(1+\Lambda_p)\left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}\right)^2 \\
&\leq \sum_{j,k=1}^{n} \max_{j=1,...,n} \max_{k=1,...,n} \widetilde{C}_{\eta,j,k}^2 \left((p+1)(1+\Lambda_p)\left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}\right)^2 \\
&= n^2 \max_{j=1,...,n} \max_{k=1,...,n} \widetilde{C}_{\eta,j,k}^2 \left((p+1)(1+\Lambda_p)\left(1 + \frac{2c_2}{\eta}\right)^{-2(p+1)}\right)^2.
\end{aligned}$$

Zieht man nun auf beiden Seiten die Wurzel, so folgt die Behauptung. □

Bemerkung 3.14. *Im Zuge der Randelementmethode mit Galerkin-Verfahren tritt die sogenannte Steifigkeitsmatrix auf. Deren Einträge A_{jk} sind gegeben durch*

$$\int_{T_j} \int_{T_k} \log|\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}} ds_{\boldsymbol{x}}. \tag{27}$$

Hierbei berechnen wir an Stelle der exakten Matrix die approximierende Matrix A_p. In der Praxis können so Auslöschungseffekte vermieden werden, die ansonsten insbesondere

für kurze Randelemente auftreten. Um diese Auslöschungseffekte zu verstehen, betrachte man beispielsweise die Stammfunktion des inneren Integrals von (27)

$$F_{\boldsymbol{x}}(\boldsymbol{y}) = \int_{T_k} \log|\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}}$$

für festes $\boldsymbol{x} \in \mathbb{R}^2$*. Die Stammfunktion ist klarerweise stetig. Daher gilt für* $T_k = [\boldsymbol{a}_k, \boldsymbol{b}_k]$ *mit geringen Durchmesser* $\operatorname{diam}(T_k)$ *aber* $F_{\boldsymbol{x}}(\boldsymbol{a}_k) \approx F_{\boldsymbol{x}}(\boldsymbol{b}_k)$*, weshalb für den Ausdruck*

$$F_{\boldsymbol{x}}(\boldsymbol{b}_k) - F_{\boldsymbol{x}}(\boldsymbol{a}_k) = \int_{T_k} \log|\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}},$$

starke Auslöschungseffekte auftreten.
Für zulässige Randstücke wird das äußere Integral durch Gauß-Quadratur ersetzt. Dabei wird $F_{\boldsymbol{x}}$ *mehrmals ausgewertet und mit stets positiven Quadraturgewichten multipliziert. Die* $F_{\boldsymbol{x}}$ *haben in der Praxis alle dasselbe Vorzeichen (was insbesondere durch Skalierung des Integrationsbereichs erreicht werden kann). Daher werden bei der Gauß-Quadratur lediglich Werte gleichen Vorzeichens addiert, weshalb keine Auslöschung auftritt. In der Praxis beobachtet man daher, dass die Auswertung mittels Gauß-Quadratur für zulässige Randstücke genauer ist als die Berechnung mithilfe der Stammfunktion.*
Die obige Argumentation trifft auch auf das Doppelintegral (27) *zu. Insbesondere weil die Auswertung des Doppelintegrals auf mehrere Auswertungen von Integralen über das innere Randstück zurückgeführt wird, erscheint es aus numerischer Sicht günstig, zuerst über das längere der beiden Randelemente zu integrieren. Dadurch wird jene Operation, bei der stärkere Auslöschungen auftreten, ans Ende des Algorithmus gestellt. Die Vertauschbarkeit der Integrationsreihenfolge für unzulässige Randstücke gilt ebenfalls, wie wir bereits in Bemerkung 3.9 festgestellt haben. Die C-Funktion, die für die Auswertung der Doppelintegrale verantwortlich ist, führt daher die Vertauschungen durch, wie man in Abschnitt A.5 nachlesen kann.*

4 Lösungsverfahren und numerische Beispiele

4.1 Problemstellung und Lösungsansatz

In diesem Abschnitt betrachten wir die homogene Laplace-Gleichung in der Ebene mit gegebenen Dirichlet-Daten. Das Problem ist also gegeben durch

$$-\Delta u = 0 \qquad \text{auf } \Omega \subset \mathbb{R}^2, \tag{28}$$

$$u|_\Gamma = g \qquad \text{auf } \Gamma := \partial\Omega, \tag{29}$$

wobei $\Delta u := \partial_x^2 u + \partial_y^2 u$ den Laplace-Operator bezeichnet, $\Omega \subset \mathbb{R}^2$ eine beschränkte Teilmenge von $\mathbb{R}^2$ mit glattem Rand $\Gamma := \partial\Omega$ ist. Durch Skalierung können wir ferner annehmen, dass $\operatorname{diam}(\Omega) < 1$ gilt.

4.1.1 Indirekte Randintegralmethode

Sei $\widetilde{V}$ gegeben durch

$$\widetilde{V}\phi := -\frac{1}{2\pi}\int_{T_i}\int_{T_j} \log|\boldsymbol{x}-\boldsymbol{y}|\phi(\boldsymbol{y})ds_{\boldsymbol{y}}ds_{\boldsymbol{x}} \tag{30}$$

und bezeichne $\gamma_0 \in L(H^1(\Omega), H^{1/2}(\Gamma))$ den Spuroperator, der einer $C^\infty(\Omega)$-Funktion eine Funktion zuordnet, sodass

$$\gamma_0 u = u|_\Gamma.$$

Gemäß [9, Theorem 12] gilt mit diesen Bezeichnungen

$$-\Delta\widetilde{V}\phi = 0 \qquad \text{für alle } \phi \in H^{-1/2}(\Gamma). \tag{31}$$

Wir machen nun den sogenannten *indirekten Ansatz* $u = \widetilde{V}\phi$. Wegen (31) ist die Laplace-Gleichung erfüllt, und es gilt

$$V\phi = g, \tag{32}$$

wobei $V := \gamma_0\widetilde{V} \in L(H^{-1/2}(\Gamma), H^{1/2}(\Gamma))$. Ziel ist es nun, aus (32) eine Funktion ϕ zu bestimmen, die die obige Gleichung erfüllt, denn dann ist $\widetilde{V}\phi$ die Lösung des Problems.

4.1.2 Galerkin-Verfahren

Da das Problem (32) im Allgemeinen nicht exakt lösbar ist, lösen wir es lediglich näherungsweise mithilfe des *Galerkin-Verfahrens*, bei dem wir $H^{-1/2}(\Gamma)$ durch einen endlichdimensionalen Unterraum ersetzen.
Da V wegen $\operatorname{diam}(\Omega) < 1$ ein symmetrischer und elliptischer Isomorphismus auf Γ ist, existiert ein äquivalentes Skalarprodukt $\langle\!\langle\cdot,\cdot\rangle\!\rangle$ auf $\widetilde{H}^{-1/2}$ mit $\langle\!\langle\phi,\psi\rangle\!\rangle := \langle V\phi,\psi\rangle$, wobei $\langle\cdot,\cdot\rangle$ das erweiterte L^2-Skalarprodukt bezeichnet. Weiters bezeichne $|\!|\!|\cdot|\!|\!|$ die induzierte Norm.
Sei nun ϕ die eindeutige Lösung und bezeichne g die Dirichlet-Daten am Rand. Dann gilt

$$\langle\!\langle\phi,\psi\rangle\!\rangle = \langle g,\psi\rangle \qquad \text{für alle } \psi \in \widetilde{H}^{-1/2}. \tag{33}$$

Wir betrachten nun eine Partition $\mathcal{T}_h = \{ T_1, T_2, ..., T_N \}$ des Randes Γ in N affine Randstücke $T_1, T_2, ..., T_N$ und definieren die zugehörigen charakteristischen Funktionen durch

$$\chi_i(\boldsymbol{x}) = \begin{cases} 1 & \text{für } \boldsymbol{x} \in T_i \\ 0 & \text{sonst} \end{cases}$$

für alle $i = 1, ..., N$. Mit X_h bezeichnen wir den durch $\chi_1, \chi_2, ..., \chi_N$ aufgespannten Raum. Man sucht nun eine Lösung $\phi_h \in X_h$ mit

$$\langle\!\langle \phi_h, \psi_h \rangle\!\rangle = \langle g, \psi_h \rangle \qquad \text{für alle } \psi_h \in X_h,$$

was gleichbedeutend ist mit

$$\langle\!\langle \phi_h, \chi_i \rangle\!\rangle = \langle g, \chi_i \rangle \qquad \text{für alle } i = 1, ..., N, \tag{34}$$

da $\{\chi_1, ..., \chi_N\}$ eine Basis von X_h ist. Sei im folgenden $\phi_h \in X_h$ eine Funktion, die (34) erfüllt. Der Fehler $\phi - \phi_h$ steht bezüglich des Energieskalarprodukts $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ orthogonal auf X_h, das heißt, es gilt

$$\langle\!\langle \phi - \phi_h, \psi_h \rangle\!\rangle = 0 \qquad \text{für alle } \psi_h \in X_h. \tag{35}$$

Diese Eigenschaft bestimmt die diskrete Lösung ϕ_h eindeutig. Die Galerkin-Lösung ist die Best-Approximation bezüglich des Unterraums X_h. Das heißt es gilt

$$|\!|\!| \phi - \phi_h |\!|\!| \leq |\!|\!| \phi - \psi_h |\!|\!| \qquad \text{für alle } \psi_h \in X_h. \tag{36}$$

Schreiben wir nun (34) um, so erhalten wir

$$\sum_{j=1}^{N} x_{h;j} \langle V\chi_j, \chi_i \rangle = \langle g, \chi_i \rangle, \tag{37}$$

wobei $\boldsymbol{x}_h := (x_{h;1}, ..., x_{h;N})$ der Koordinatenvektor bezüglich der Basis $\chi_1, ..., \chi_N$ ist. Bezeichnen wir nun mit $\boldsymbol{V}_{ij} := \langle V\chi_j, \chi_i \rangle$, und sei $\boldsymbol{g}_h \in \mathbb{R}^N$ mit den Koeffizienten $g_{h;i} := \langle g, \chi_i \rangle$, so können wir (37) umschreiben als

$$\boldsymbol{V}\boldsymbol{x}_h = \boldsymbol{g}_h.$$

Die Matrix $\boldsymbol{V}$ wird als Steifigkeitsmatrix bezeichnet. Schreibt man $\boldsymbol{V}_{ij}$ in Integralschreibweise an, so ist

$$\boldsymbol{V}_{ij} = -\frac{1}{2\pi} \int_{T_i} \int_{T_j} \log |\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}} ds_{\boldsymbol{x}}.$$

Wie bereits in Bemerkung 3.14 erwähnt, betrachten wir genau genommen nicht die Matrix $\boldsymbol{V}$, sondern eine approximierende Matrix $\boldsymbol{V}_p$ die entsteht, indem man — im Wesentlichen, für Details sei auf Abschnitt 3 verwiesen — bei bestimmten Einträgen das äußere Integral durch Gauß-Quadratur vom Grad p ersetzt. Vom analytischen Standpunkt konvergiert $\boldsymbol{V}_p$ in der Frobeniusnorm für wachsendes p exponentiell schnell gegen $\boldsymbol{V}$. Jedoch treten bei der analytischen Berechnung von $\boldsymbol{V}$ Auslöschungseffekte auf, sodass in der Praxis die approximierende Matrix besser geeignet ist. Dies betrifft aber lediglich die stabile Berechnung von $\boldsymbol{V}$. Im Folgenden unterscheiden wir jedoch nicht, auf welche Weise $\boldsymbol{V}$ berechnet wird und schreiben auch für die approximierende Matrix einfach $\boldsymbol{V}$.

4.1.3 Fehlerschätzer

Durch das Galerkin-Verfahren wird lediglich eine approximative Lösung ermittelt. Um eine Aussage über die Genauigkeit der Lösung treffen zu können, interessiert man sich für $|||\phi - \phi_h|||$, wobei ϕ die exakte Lösung von (32) und ϕ_h die Galerkin-Lösung bezeichnet. Da die exakte Lösung und deren Energienorm nicht bekannt sind, betrachten wir stattdessen die folgenden Fehlerschätzer.
Hierzu betrachten wir die Gitternetze $\mathcal{T}_h$ und $\mathcal{T}_{h/2}$, wobei $\mathcal{T}_{h/2}$ aus $\mathcal{T}_h$ entsteht, indem jede Kante in zwei gleichgroße Kanten geteilt wird. Es Bezeichnen nun X_h und $X_{h/2}$ die zugehörigen Unterräume von $\widetilde{H}^{-1/2}(\Gamma)$ und weiters $\phi_h \in X_h$ und $\phi_{h/2} \in X_{h/2}$ die jeweiligen Galerkin-Lösungen. Der Fehlerschätzer η_h ist dann definiert durch

$$\eta_h := |||\phi_{h/2} - \phi_h|||. \tag{38}$$

Wegen der Galerkin-Orthogonalität (35) gilt

$$|||\phi - \phi_h|||^2 = |||\phi - \phi_{h/2}|||^2 + |||\phi_{h/2} - \phi_h|||^2 = |||\phi - \phi_{h/2}|||^2 + \eta_h^2 \tag{39}$$

woraus $\eta_h \leq |||\phi - \phi_h|||$ folgt. Man sagt daher auch, dass der η_h-Schätzer *effizient* mit Konstante $C_{\text{eff}} = 1$. Außerdem kann man unter der Saturationsannahme

$$|||\phi - \phi_{h/2}||| \leq C_{\text{sat}} |||\phi - \phi||| \qquad \text{für ein } C_{\text{sat}} \in (0,1)$$

zeigen, dass der Schätzer η_h *zuverlässig* ist, das heißt, es gilt

$$\exists C_{\text{rel}} : \quad |||\phi - \phi_h||| \leq C_{\text{rel}} \eta_h.$$

Siehe dazu auch [7, Glg (31)]. Obwohl die Saturationsannahme in der Praxis beobachtet werden kann, ist sie im Kontext der Randelementmethode unbewiesen.
Ein Nachteil des η_h-Fehlerschätzers ist die Tatsache, dass er keine Aussage über den Fehler an einer bestimmten Stelle macht. Insbesondere bei der Netzverfeinerung interessieren wir uns aber auch für den Fehler auf bestimmten Randelementen, weshalb wir den auf der L^2-Norm basierenden lokalen h-$h/2$-Schätzer μ_h betrachten, der definiert ist durch

$$\mu_h^2 := \sum_{i=1}^{N} \mu_h(T_i)^2, \quad \text{wobei} \quad \mu_h(T_i)^2 = \operatorname{diam}(T_i) \, \|\phi_{h/2} - \phi_h\|^2_{L^2(T_i)}. \tag{40}$$

Man kann zeigen, dass der μ_h-Schätzer und der η_h-Schätzer unter gewissen Regularitätseigenschaften des Randnetzes äquivalent sind, das heißt es existieren Konstanten C_1, C_2, sodass $C_1 \mu_h \leq \eta_h \leq C_2 \mu_h$ gilt; siehe [5, Theorem 3.2]. Der μ_h-Schätzer ist insbesondere ebenfalls *zuverlässig* und *effizient*.

4.1.4 Netzverfeinerung

Wählt man ein Gitternetz $\mathcal{T}_h'$, sodass $X_h \subseteq X_{h'}$, so wird der Fehler wegen (36) bezüglich der Energienorm im Allgemeinen kleiner. Bei der Wahl von $\mathcal{T}_h'$, also der Verfeinerung von $\mathcal{T}_h$, kann man verschiedene Strategien verfolgen.
Einerseits kann man das Netz *uniform verfeinern*. Das bedeutet, dass man jede Kante in jedem Verfeinerungsschritt in eine feste Anzahl von Kanten unterteilt. Beispielsweise

kann man die Kantenlänge in jedem Verfeinerungsschritt halbieren, was in den folgenden Beispielen stets gemeint ist, wenn wir von uniformer Netzverfeinerung sprechen. Verfeinern wir ein Gitternetz $\mathcal{T}_h$ einmal uniform, so wird das uniform verfeinerte Netz kurz mit $\mathcal{T}_{h/2}$ bezeichnet.
Andererseits können nur bestimmte Randstücke zur Verfeinerung ausgewählt werden. Man spricht dann von *adaptiver Netzverfeinerung*. Günstig erscheint es, das Netz in Abhängigkeit des Fehlers zu verfeinern. Genauer verwenden wir *Dörflers Markierungsstrategie* [4]. Hierbei bestimmen wir für eine feste Konstante $\theta \in (0,1)$ eine minmale Teilmenge $\mathcal{M}_h \subseteq \mathcal{T}_h$, sodass

$$\theta \sum_{T \in \mathcal{T}_h} \mu_h(T)^2 \leq \sum_{T \in \mathcal{M}_h} \mu_h(T)^2 \tag{41}$$

wobei $\mu_h(T)$ den zuvor in (40) definierten μ_h-Schätzer auf einem Randstück T bezeichnet. Es stellt sich heraus, dass der Fehler in der Energienorm $|||\phi - \phi_h|||$ bei adaptiver Netzverfeinerung schneller gegen 0 geht, als bei uniformer. Konkret ist der Fehler bei uniformer Netzverfeinerung in Abhängigkeit des Randes ein $\mathcal{O}(N^{-r})$ mit $r < 3/2$, während bei adaptiver Netzverfeinerung die optimale Konvergenzrate von $\mathcal{O}(N^{-3/2})$ auftritt.

4.1.5 Vorkonditionierung der Steifigkeitsmatrix $\boldsymbol{V}$

Die adaptive Netzverfeinerung wirkt sich jedoch störend auf die Konditionszahl der Steifigkeitsmatrix aus. Wir betrachten im folgenden stets die Konditionszahl bezüglich der ℓ_2-Norm. Gemäß [1, Theorem 2.1] gilt, wenn man es auf die betrachteten Beispiele spezialisiert,

$$\operatorname{cond}(\boldsymbol{V}) \preceq N \qquad \text{bei uniformer Netzverfeinerung,} \tag{42}$$

$$\operatorname{cond}(\boldsymbol{V}) \preceq N \log N \left(\frac{h_{\max}}{h_{\min}}\right)^2 \qquad \text{bei adaptiver Netzverfeinerung,} \tag{43}$$

wobei N die Anzahl der Randelemente bezeichnet und $h_{\max} = \max\{\ \operatorname{diam}(T)\ \mid T \in h\ \}$ und $h_{\min} = \min\{\ \operatorname{diam}(T)\ \mid T \in h\ \}$. Im adaptiven Fall kann die Konditionszahl der Steifigkeitsmatrix also wesentlich schneller wachsen, was sich potenziell störend auf das Lösen des Gleichungssystems während des Galerkin-Verfahrens auswirkt.
Die Situation kann man jedoch durch Vorkonditionierung deutlich verbessern, was ebenfalls in [1, Theorem 2.1] gezeigt wird. Hierzu sei

$$\boldsymbol{D}_{ij} := \begin{cases} \boldsymbol{V}_{ij} & \text{für } i = j \\ 0 & \text{sonst.} \end{cases}$$

Anstelle von $\boldsymbol{V}\boldsymbol{x} = \boldsymbol{g}$ löst man nun mit $\widehat{\boldsymbol{V}} := \boldsymbol{D}^{-1/2}\boldsymbol{V}\boldsymbol{D}^{-1/2}$ und $\widehat{\boldsymbol{g}} := \boldsymbol{D}^{-1/2}\boldsymbol{g}$ das Gleichungssystem

$$\widehat{\boldsymbol{V}}\boldsymbol{y} = \widehat{\boldsymbol{b}}.$$

Klarerweise gilt dann $\boldsymbol{x} = \boldsymbol{D}^{-1/2}\boldsymbol{y}$.
Betrachtet man nun die Konditionszahl von $\widehat{\boldsymbol{V}}$ in Bezug auf die Anzahl der Gitternetzkanten N des Netzes $\mathcal{T}_h$, so gilt gemäß [1, Theorem 2.1] für den schwach singulären Integraloperatore V, den wir betrachten, und einen Rand der Dimension 1

$$\operatorname{cond}(\widehat{\boldsymbol{V}}) =\preceq (N(1 + \log(Nh_{\min})\frac{1 + |\log h_{\min}|}{1 + |\log h_{\max}|}. \tag{44}$$

Wir können nun $h_{\max}$ mit 1 abschätzen, da das die Länge des größten Randelements unseres Gitternetzes ist. Außerdem kann man $h_{\min}$ nach unten hin durch 2^{-v} abschätzen, wobei v die Anzahl der Verfeinerungsschritte ist. In jedem Schritt markiert der verwendete Markierungsalgorithmus zumindest einen festen, prozentualen Anteil p der Randelemente. Daher hängt die Zahl der Verfeinerungsschritte im wesentlichen logarithmisch von der Anzahl der Randelemente N ab. Die folgende Rechnung zeigt dies, wobei n die Anzahl der Randelemente des Startnetzes ist:

$$n(1+p)^v \leq N \tag{45}$$

$$v \leq \frac{\log N - \log n}{\log(1+p)} \tag{46}$$

Daher gilt

$$|\log h_{\min}| \geq |-v \log 2| \succeq \log N. \tag{47}$$

Nach oben hin kann man $h_{\min}$ in unseren Beispielen ebenfalls mit 1 abschätzen. Insgesamt erhält man

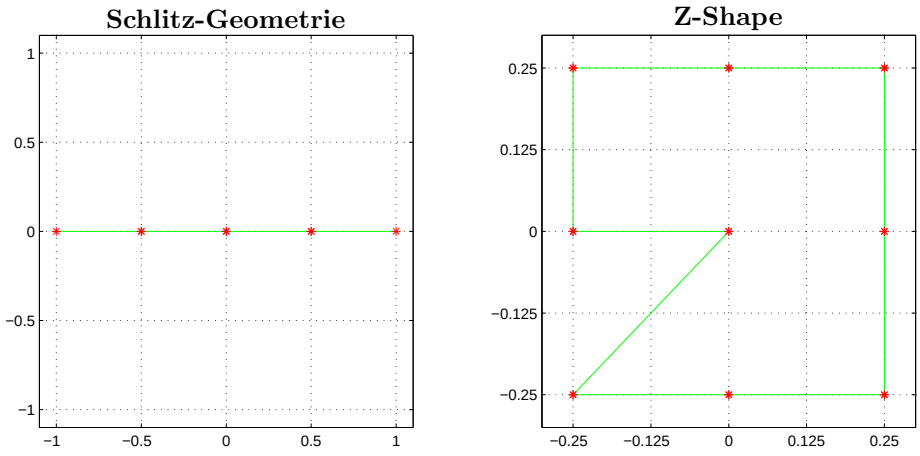
$$\begin{aligned} \operatorname{cond}(\widehat{\boldsymbol{V}}) &\preceq N(1+\log(N h_{\min}))\frac{1+|\log h_{\min}|}{1+|\log h_{\max}|} \\ &\preceq N(1+\log(N))\log(N) \preceq N\log(N) + N\log(N)^2 \preceq N\log(N)^2. \end{aligned} \tag{48}$$

Man erwartet also, dass die Konditionszahl keinenfalls schneller wächst als $cN\log(N)^2$ für eine Konstante $c \in \mathbb{R}$.

4.1.6 Lösungsalgorithmus

Wir geben nun in Listenform einen Algorithmus an, der ϕ_h in Bezug auf ein Netz $\mathcal{T}_h$ berechnet. Als Eingabeparameter erhält der Algorithmus einerseits die Triangulierung $\mathcal{T}_h$ und andererseits eine von g abhängige Funktion `buildRightHandSide`, die zu gegebenen g einen Vektor $\boldsymbol{g}_h$ bezüglich einem Netz $\mathcal{T}_h$ erzeugt, dessen Einträge durch $\boldsymbol{g}_{h,i} = \langle g, \chi_i \rangle$ gegeben sind. Zurückgegeben wird die Galerkin-Lösung ϕ_h, beziehungsweise deren Koeffizientenvektor $\boldsymbol{x}_h$ bezüglich der kanonischen Basis von X_h.

1. Erzeuge eine Matrix $\boldsymbol{V}$ in Bezug auf die Triangulierung $\mathcal{T}_h$.
2. Berechne die rechte Seite $\boldsymbol{g}_h$ in Bezug auf die Triangulierung $\mathcal{T}_h$.
3. Löse das (optional vorkonditionierte) lineare Gleichungssystem
$$\boldsymbol{V}\boldsymbol{x}_h = \boldsymbol{g}_h$$
4. Gib $\boldsymbol{x}_h$ zurück.



(a) Schlitz-Geometrie

(b) Z-Shape

Abbildung 2: Die unterschiedlichen Beispiel-Geometrien

4.1.7 Adaptiver Algorithmus

Schließlich formulieren wir den adaptiven Algorithmus, der $\phi_{h'}$ bezüglich einem Netz $\mathcal{T}_{h'}$ berechnet, sodass der Fehlerschätzer $\eta_{h'}$ kleiner einer festen Konstante ist. Wie auch der vorhergehende Algorithmus, erhält der adaptive Algorithmus als Eingabeparameter einerseits die Triangulierung $\mathcal{T}_h$, also das Startnetz, und andererseits eine von g abhängige Funktion `buildRightHandSide`, die zu gegebenen g einen Vektor $\boldsymbol{g}_h$ bezüglich einem Netz $\mathcal{T}_h$ erzeugt, dessen Einträge durch $\boldsymbol{g}_{h,i} = \langle g, \chi_i \rangle$ gegeben sind. Zurückgegeben wird die Galerkin-Lösung $\phi_{h'}$, beziehungsweise genauer deren Koeffizientenvektor $\boldsymbol{x}_{h'}$ bezüglich der kanonischen Basis von $X_{h'}$.

1. Ermittle $\boldsymbol{x}_h$ mit dem Lösungsalgorithmus aus Unterabschnitt 4.1.6.
2. Verfeinere $\mathcal{T}_h$ uniform zu einem Netz $\mathcal{T}_{h/2}$.
3. Ermittle $\boldsymbol{x}_{h/2}$ mit dem Lösungsalgorithmus aus Unterabschnitt 4.1.6.
4. Berechne $\eta_h = |||\phi_h - \phi_{h/2}|||$
   (a) Setze $\boldsymbol{x}_h$ auf das verfeinerte $h/2$-Netz fort.
   (b) Berechne η_h als $(\boldsymbol{x}_{h/2} - \boldsymbol{x}_h)\boldsymbol{V}_{h/2}(\boldsymbol{x}_{h/2} - \boldsymbol{x}_h)$, wobei $\boldsymbol{V}_{h/2}$ die Steifigkeitsmatrix in Bezug auf das $h/2$-Netz ist.
5. Falls η_h hinreichend klein ist, so gib $\boldsymbol{x}_{h/2}$ zurück.
6. Berechne mithilfe von $\phi_{h/2}$ und ϕ_h die lokalen Anteile des μ_h-Schätzers $\mu_h(T_i)$ für alle $T_i \in \mathcal{T}_h$.
7. Verfeinere das Netz $\mathcal{T}_h$ mit der Dörfler-Markierungsstrategie zu einem Netz $\mathcal{T}_{h'}$.
8. Setze $\mathcal{T}_h := \mathcal{T}_{h'}$ und gehe zum ersten Schritt.

4.2 Beispiel Schlitz-Geometrie

Wir betrachten die Laplace-Gleichung

$$-\Delta u = 0 \qquad \text{auf } \Omega \subset \mathbb{R}^2, \tag{49}$$

$$u|_\Gamma = 1 \qquad \text{auf } \Gamma := \partial\Omega, \tag{50}$$

mit dem Startnetz aus Abbildung 2(a), dessen Gitterpunkte gegeben sind durch

$$\{(-1,0),(-0.5,0),(0,0),(0.5,0),(1,0)\}$$

Die Lösung ϕ, die wir durch den direkten Ansatz erhalten, weist an den Enden $(-1,0)$ bzw. $(1,0)$ jeweils Singularitäten auf. Außerdem kann man berechnen, dass $|||\phi|||^2 = \pi$.
In den Abbildungen 3 und 4 betrachten wir den Fehler der Lösung gegenüber der analytisch exakt bekannten Lösung in der Energienorm, sowie diverse Fehlerschätzer. Entlang der x-Achse ist die Anzahl der Gitternetzkanten aufgetragen. Beide Achsen sind logarithmisch skaliert.
In Blau ist der Fehler der auf dem h-Netz ermittelten Lösung gegenüber der bekannten Lösung in der Energienorm dargestellt. In Lila ist der h-$h/2$-Schätzer geplottet und in Türkis der μ_h-Schätzer. Die rote Linie zeigt den Fehler auf dem $h/2$-Netz.
In Abbildung 3 wurde uniforme Netzverfeinerung verwendet. Man sieht, dass der Fehler in etwa mit $\mathcal{O}(N^{-1/2})$ gegen 0 strebt. Außerdem erkennt man im Plot die Effektivität und Zuverlässigkeit des h-$h/2$-Schätzer anhand der Parallelität des tatsächlichen Fehlers und des Fehlerschätzers. Der Fehlerschätzer beschreibt den tatsächlichen Fehler auch in der Größenordnung sehr gut.
Man sieht außerdem ebenfalls wegen der Parallelität der entsprechenden Geraden die Äquivalenz des h-$h/2$-Schätzers und des lokalen μ_h-Schätzers.
Gerechnet wurde bis ungefähr 4000 Gitternetzkanten. Der Fehler in der Energienorm ist dann im Bereich von 0.1.
In Abbildung 4 wurde hingegen adaptive Netzverfeinerung verwendet. Man sieht, dass der Fehler hierbei wesentlich schneller gegen 0 strebt, nämlich wie $\mathcal{O}(N^{-3/2})$. Die Fehlerschätzer verhalten sich ähnlich gut wie bei uniformer Netzverfeinerung.
Bei rund 7000 Elementen sieht man, dass der Fehler auf dem $h/2$-Netz im Plot verschwindet. Dies begründet sich dadurch, dass $|||\phi - \phi_{h/2}|||^2 < 0$, weshalb die Wurzel daraus komplex ist und nicht geplottet wird. Dies steht klarerweise nicht im Einklang mit der Theorie.
Genau genommen wird bei der Berechnung von $|||\phi - \phi_{h/2}|||$ die Galerkin-Orthogonalität verwendet, wegen der

$$|||\phi - \phi_{h/2}|||^2 = |||\phi|||^2 - |||\phi_{h/2}|||^2$$

gilt.
Bei rund 9000 Elementen verschwindet der Fehler auf dem h-Netz im Plot aus den selben Gründen ebenfalls. Die Implementierung läuft bis rund 7000 Elemente stabil und erreicht so in diesem Beispiel eine Genauigkeit von 10^{-5} in der Energienorm.
In Abbildung 5 betrachten wir nun die Konditionszahl der beim Galerkin-Verfahren auftretenden Steifigkeitsmatrix bei uniformer und adaptiver Netzverfeinerung mit und ohne Vorkonditionierung. Hierbei ist auf der x-Achse die Zahl der Gitternetzkanten und auf

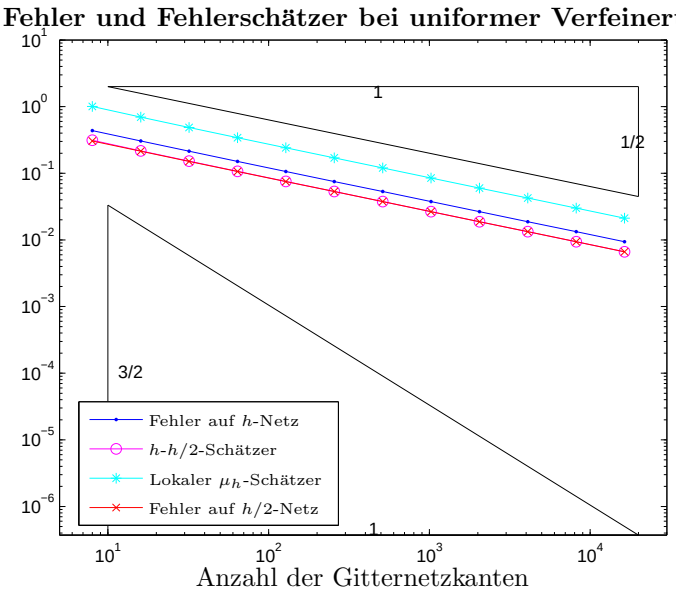


Abbildung 3: Fehler und Fehlerschätzer bei uniformer Netzverfeinerung (Schlitz-Geometrie)

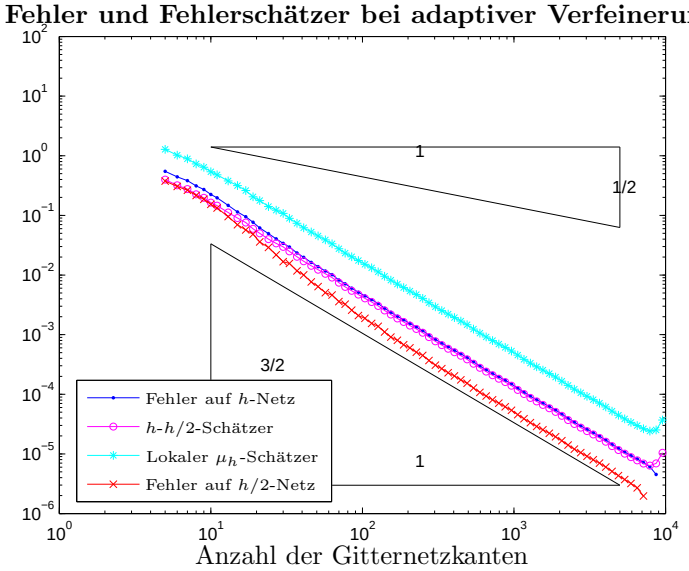


Abbildung 4: Fehler und Fehlerschätzer bei adaptiver Netzverfeinerung (Schlitz-Geometrie)

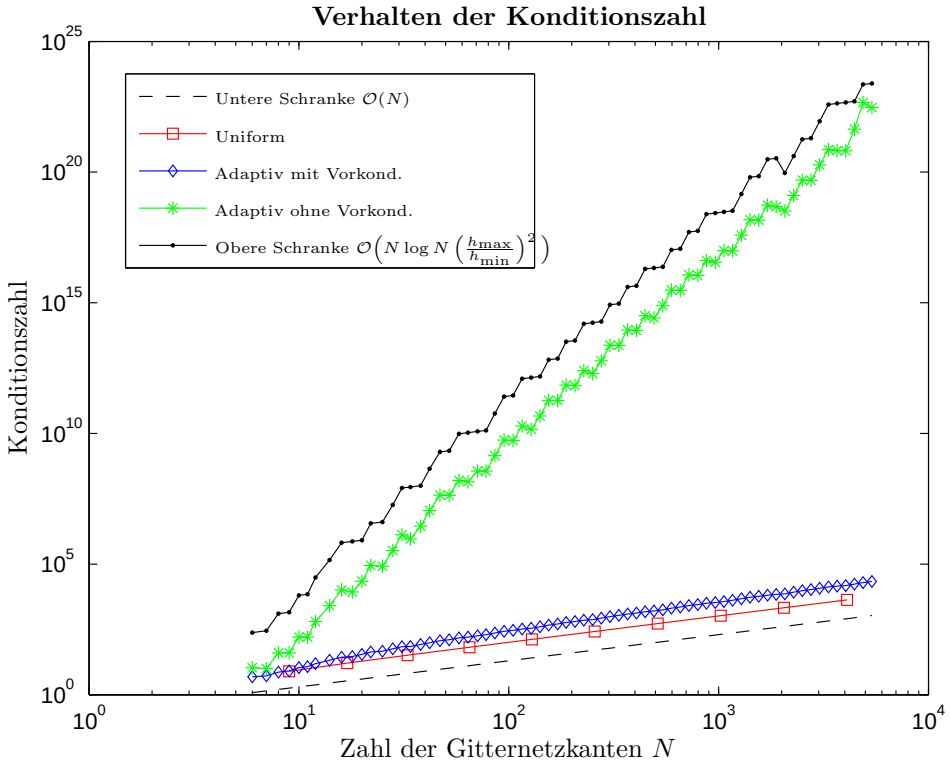


Abbildung 5: Konditionszahl von V (Schlitz-Geometrie)

der y-Achse die jeweilige Konditionszahl aufgetragen. Beide Achsen sind logarithmisch skaliert.
Im uniformen Fall, der in Rot dargestellt ist, wächst die Konditionszahl lediglich linear in der Zahl der Gitternetzkanten. Verfeinert man das Netz hingegen adaptiv, so wächst die Konditionszahl wie $N \log N \left(\frac{h_{\max}}{h_{\min}}\right)^2$, wobei $h_{\max}$ den Durchmesser der längsten Kannte und $h_{\min}$ jenen der kürzesten Kante bezeichnet.
Durch Skalierung mit einer Diagonalmatrix lässt sich das Wachstum der Konditionszahl auf $\mathcal{O}(N \log(N)^2)$ reduzieren, was in blau dargestellt ist. Bei uniformer Netzverfeinerung hat die Vorkonditionierung keinen Effekt, weshalb dieser Fall nicht abgebildet ist.
In der Praxis wirkt sich die schlechte Kondition der Matrix interessanterweise nicht auf die Stabilität des Verfahrens aus. Betrachtet man wiederum dieselbe Situation, die zu Abbildung 4 geführt hat, so ergeben sich gegenüber diesem Plot keine sichtbaren Unterschiede.
Abschließend untersuchen wir die Stabilität und Genauigkeit des Algorithmus in Abhängigkeit der Vorkonditionierung, des gewählten Gauß-Quadraturgrades bei der Berechnung von $\boldsymbol{V}$, sowie der Maschinengenauigkeitskonstante EPS bei der Berechnung von V. Den Gauß-Quadraturgrad wählen wir dabei entweder als 4, 8, 16 oder 32 und die Maschinengenauigkeitskonstante als $10^{-12}, 10^{-14}$ oder 10^{-15}. Es stellt sich heraus, dass die Wahl dieser Parameter für die Stabilität des Algorithmus unkritisch ist. In allen Fällen treten erste Instabilitäten bei 7000 Gitternetzkanten auf.

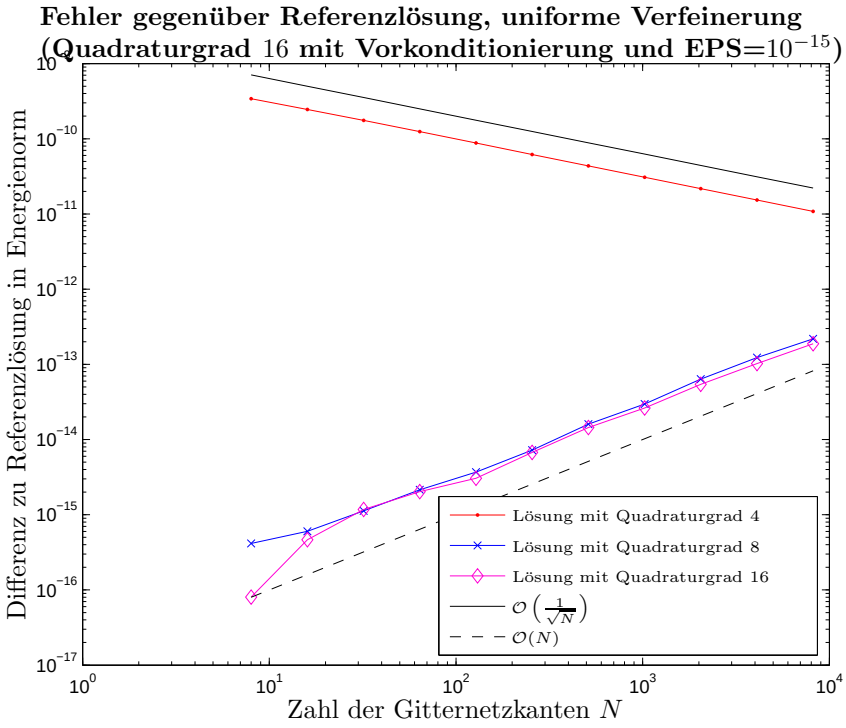


Abbildung 6: Genauigkeit gegenüber Referenzlösung bei uniformer Verfeinerung (Schlitz-Geometrie)

Um die Genauigkeit zu untersuchen, wählen wir jene Lösung mit Vorkonditionierung, Quadraturgrad 32 und Maschinengenauigkeitskonstante 10^{-15} als Referenzlösung aus. Dabei können folgende Beobachtungen unabhängig von der Verfeinerungsstrategie gemacht werden:

- Die Vorkonditionierung verbessert die Genauigkeit des Verfahrens bei diesem Beispiel nicht.
- Die Wahl der Maschinengenauigkeitskonstante EPS ist in Bezug auf die Genauigkeit insofern unkritisch, dass Unterschiede erst kurz, bevor der Algorithmus instabil wird, auftreten.

Aus diesen Gründen lassen wir die Vorkonditionierung und EPS= 10^{-15} fest und variieren lediglich den Quadraturgrad.
Abbildung 6 zeigt den Fehler gegenüber der Referenzlösung bei uniformer Netzverfeinerung. Zu bemerken ist, dass der Fehler für Quadraturgrad 4 anfangs mit ca. 10^{-9} deutlich höher ist als für Quadraturgrad 8 oder 16. Mit wachsender Gitternetzkantenanzahl und kleiner werdender Kantenlänge nähern sich diese Fehler aber einander an.
Bei adaptiver Netzverfeinerung muss man zusätzlich darauf achten, dass das verfeinerte Netz von der zuvor ermittelten Lösung abhängt. Das bedeutet, dass sich bei unterschiedlicher Parameterwahl auch unterschiedliche Netze ergeben können. Das wollen wir nach Möglichkeit vermeiden, weshalb wir in jedem Schritt nur eine Lösung auswählen anhand derer das Netz weiter verfeinert wird. Diese Lösung ist stets die Referenzlösung. Diese Strategie macht es möglich, die Energienorm der Differenz zweier Lösungen sehr einfach

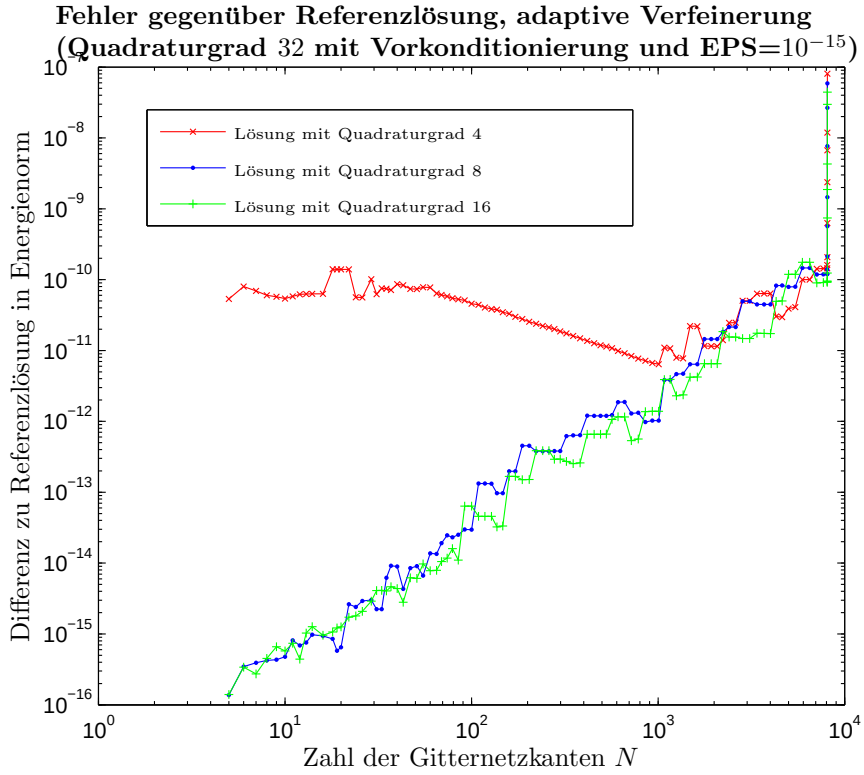
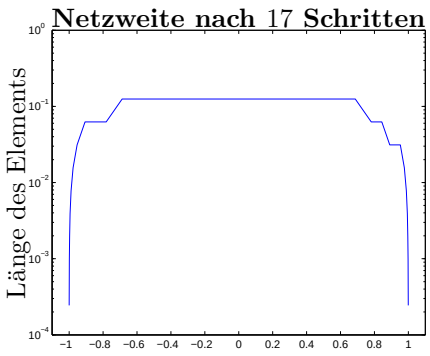
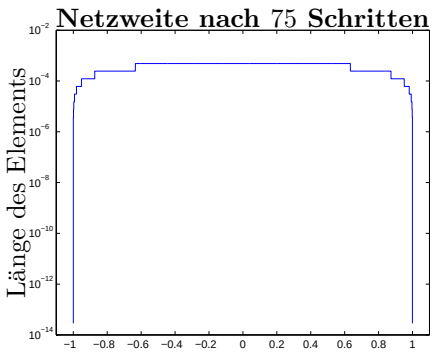


Abbildung 7: Genauigkeit gegenüber Referenzlösung bei adaptiver Verfeinerung (Schlitz-Geometrie)

zu berechnen, hat jedoch den Nachteil, dass der bestimmte Wert nur den in *einem* Schritt gemachten Fehler beschreibt. Das Wechselspiel zwischen Fehler und Netzverfeinerung betrachten wir nicht.
Abbildung 7 stellt nun die Ergebnisse dar. Es ergibt sich ein ähnliches Bild, wie bei uniformer Netzverfeinerung. So ist der Fehler bei Quadraturgrad 4 gegenüber der Referenzlösung anfangs verhältnismäßig groß, wird jedoch mit kleiner werdender Gitternetzkantenlänge etwas kleiner, bis er sich schließlich dem Fehler für Quadraturgrad 8 bzw. 16 annähert. Anfangs liegt der Fehler für Quadraturgrad 8 bzw. 16 größenordnungsmäßig bei 10^{-16} und steigt dann mit zunehmender Zahl der Randelemente an. Die Fehlergröße verhält sich dabei annähernd so wie $\mathcal{O}(N^4)$.
Abschließend betrachten wir die Auswirkungen der adaptiven Netzverfeinerung auf das Netz. Hierzu plotten wir jeweils in der Mitte jedes Elements des Netzes die Länge des jeweiligen Elements. Die y-Achse, entlang derer die Längen der Elemente aufgetragen sind ist logarithmisch skaliert. Abbildung 8(a) zeigt den so entstanden Plot nach 17 Verfeinerungsschritte, während Abbildung 8(b) der entsprechende Plot nach 75 Verfeinerungsschritten ist, bevor der Algorithmus abbricht. Als übrige Parameter für die Plots wurden Quadraturgrad 16, Maschinengenauigkeitskonstante 10^{-12} und aktivierte Vorkonditionierung gewählt.
Die Plots zeigen die effektive Netzverfeinerung nahe der Singularität. Das Netz in Abbildung 8(b) verfügt beispielsweise über rund 10000 Elemente. Während man bei uniformer Netzverfeinerung nur Netzweite $1/5000 = 2 \cdot 10^{-4}$ hätte, geht diese im adaptiven Fall



(a) Netzweite nach 17 Schritten



(b) Netzweite nach 75 Schritten

Abbildung 8: Netzweite bei adaptiver Netzverfeinerung bei Gauß-Quadraturgrad 16, EPS 10^{-12} und mit Vorkonditionierung. (Schlitz-Geometrie)

nahe der Singularität auf unter 10^{-13} während sie fern der Singularität deutlich gröber ist. Oder anders gesagt, erreichen wir in Abbildung 8(a) nach 17 Verfeinerungsschritten mit nur 37 Gitternetzelementen bereits eine ähnliche Netzweite nahe der Singularität wie bei uniformer Netzverfeinerung mit 10000 Elementen.
Betrachten wir schließlich noch die minimale und maximale Gitterweite im Verlaufe des Verfeinerungsprozesses, so entsteht der Plot in Abbildung 9. Als Parameter wurden wiederum Quadraturgrad 16 und EPS 10^{-12}, sowie aktivierte Vorkonditionierung gewählt. Entlang der x-Achse ist die Zahl der Gitternetzkanten aufgetragen, während entlang der y-Achse die jeweiligen Größen bei logarithmischer Skalierung aufgetragen sind. Wir können beobachten, dass die maximale Gitternetzweite, also die Länge des längsten Elements im gesamten Netz im wesentlichen wie $\mathcal{O}(1/N)$ kleiner wird, während die minimale Gitternetzweite sich wie $\mathcal{O}(1/N^4)$ verhält.

4.3 Beispiel Z-Shape

Wiederum betrachten wir das Problem

$$-\Delta u = 0 \qquad \text{auf } \Omega \subset \mathbb{R}^2, \tag{51}$$

$$u|_\Gamma = 1 \qquad \text{auf } \Gamma := \partial\Omega, \tag{52}$$

bezüglich des Startnetzes aus Abbildung 2(b), dessen Gitterpunkte gegeben sind durch

$$\left\{(0,0)\,,\left(-\frac{1}{4},-\frac{1}{4}\right),\left(0,-\frac{1}{4}\right),\left(\frac{1}{4},-\frac{1}{4}\right),\left(\frac{1}{4},0\right),\left(\frac{1}{4},\frac{1}{4}\right),\left(0,\frac{1}{4}\right),\left(-\frac{1}{4},\frac{1}{4}\right),\left(-\frac{1}{4},0\right)\right\}$$

Die Lösung ϕ weist an den Ecken des Netzes Singularitäten auf. Hierbei geht ϕ in allen außenliegenden Ecken gegen $+\infty$ und in der einspringenden Ecke gegen $-\infty$. Ein Plot einer approximativen Zwischenlösung findet sich in Abbildung 14(b) am Ende dieses Abschnitts.
Die folgenden Abbildungen 10 und 11 zeigen, wie jene im vorhergehenden Beispiel, zwei Fehlerschätzer, sowie den Fehler gegenüber der tatsächlichen Lösung. In diesem Fall ist die tatsächliche Lösung jedoch nicht bekannt. Deren Energienorm kann aber mithilfe

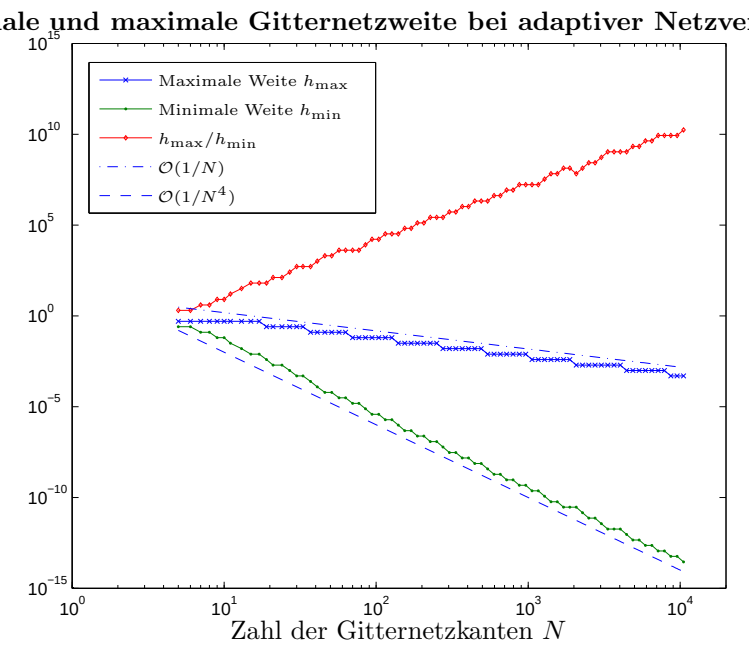


Abbildung 9: Minimale und maximale Gitternetzweite bei adaptiver Netzverfeinerung (Schlitz-Geometrie)

des *Aitken'schen* Δ^2*-Verfahrens* approximiert werden. Beim Aitken'schen Δ^2-Verfahren handelt es sich um ein Verfahren zur Beschleunigung der Konvergenz von Folgen. Wir wenden dieses auf die Folge der $|||\phi_h|||^2$ an, die gegen die Energie der exakten Lösung ϕ konvergiert und erhalten so

$$|||\phi|||^2 \approx 5.101700132914669.$$

Um die Energienorm der Differenz zwischen einer Galerkin-Lösung und der exakten Lösung zu bestimmen, verwendet man wiederum die Galerkin-Orthogonalität, wegen der

$$|||\phi - \phi_h|||^2 = |||\phi|||^2 - |||\phi_h|||^2$$

gilt, wobei ϕ die exakte Lösung von (33) und $\phi_h \in X_h$ die Galerkin-Lösung in Bezug auf h bezeichnet.
Entlang der x-Achse der Abbildungen 10 und 11 ist die Zahl der Gitternetzkanten aufgetragen. Beide Achsen sind logarithmisch skaliert. In Blau ist der Fehler der auf dem h-Netz ermittelten Lösung gegenüber der bekannten Lösung in der Energienorm dargestellt. In Lila ist der h-$h/2$-Schätzer geplottet und in türkis der μ_h-Schätzer. Die rote Linie zeigt den Fehler auf dem $h/2$-Netz.
In Abbildung 10 wurde das Netz uniform verfeinert. Man erkennt, wie auch schon im Fall der Schlitz-Geometrie, dass der Fehler ein $\mathcal{O}(N^{-1/2})$ ist. Außerdem erkennt man auch in diesem Fall die Korrektheit und Verlässlichkeit des h-$h/2$-Schätzers anhand der Parallelität des Fehlerschätzers und des tatsächlichen Fehlers im Plot.
Abbildung 11 zeigt den Fehler und die Fehlerschätzer bei adaptiver Netzverfeinerung.
Im Unterschied zur Schlitz-Geometrie ist in Abbildung 11 keine Instabilität zu erkennen. Die Implementierung bleibt bis zur Speichergrenze der Maschine (32 GB, ca. 22000

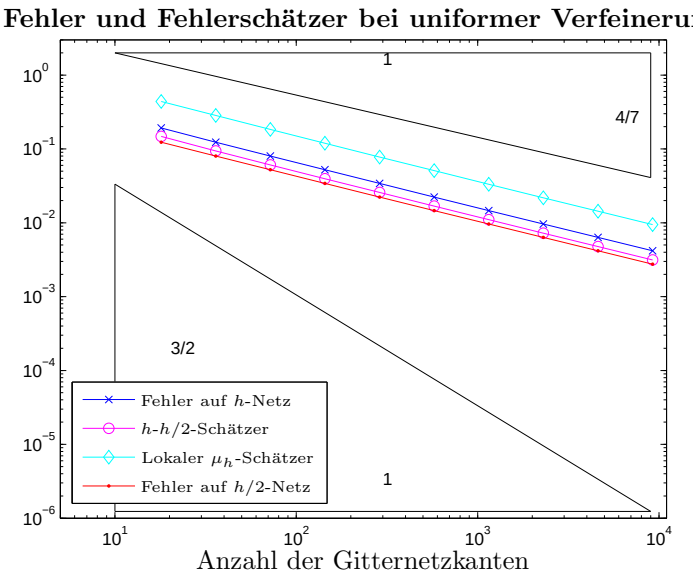


Abbildung 10: Fehler und Fehlerschätzer bei uniformer Netzverfeinerung (Z-Shape)

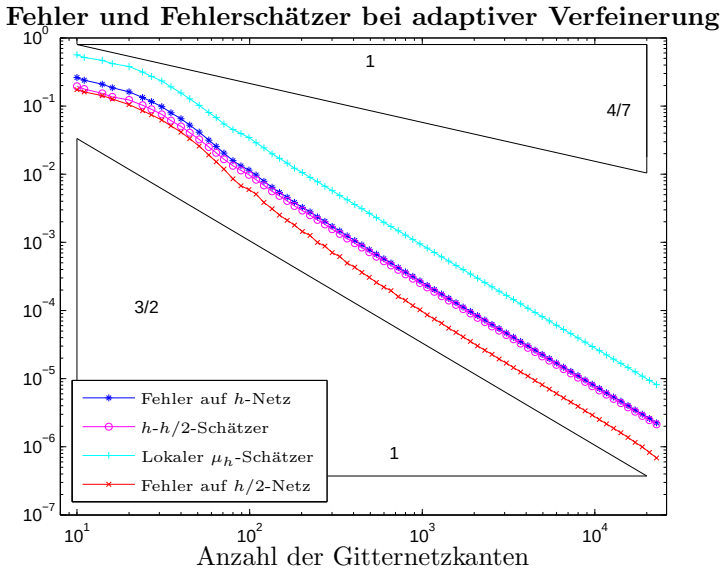


Abbildung 11: Fehler und Fehlerschätzer bei adaptiver Netzverfeinerung (Z-Shape)

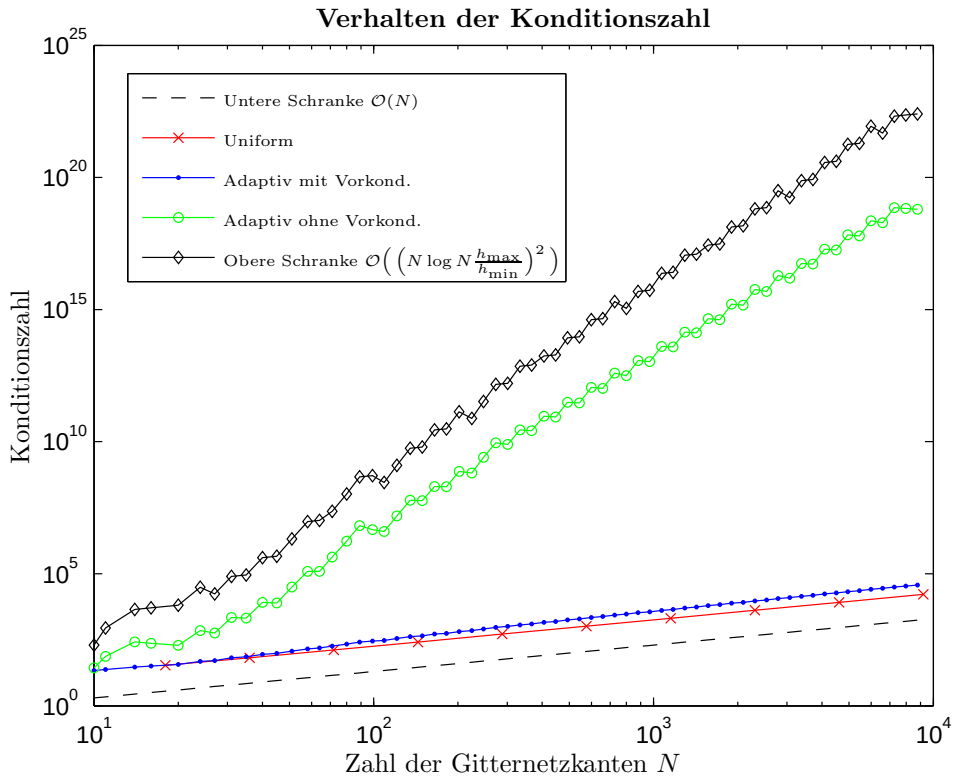


Abbildung 12: Konditionszahl von V (Z-Shape)

Gitternetzkanten) stabil. Hierbei ist zu bemerken, dass eine Matrix mit 22000×22000 Einträgen lediglich ungefähr 3.5 GB benötigt. Jedoch benötigt die uniforme Verfeinerung dann bereits rund 14 GB und für die Lösung des Gleichungssystems während des Galerkin-Verfahrens wird nochmals so viel Zusatzspeicher benötigt. In anderen Worten benötigt das Programm rund $\mathcal{O}(8N^2)$ Zusatzspeicher

In Abbildung 12 betrachten wir die Konditionszahl der Steifigkeitsmatrix bei adaptiver und uniformer Netzverfeinerung. Auf der x-Achse ist die Zahl der Gitternetzkanten aufgetragen, auf der y-Achse die jeweilige Konditionszahl. Beide Achsen sind logarithmisch skaliert.

In Rot ist die Konditionszahl bei uniformer Netzverfeinerung dargestellt, die mit wachsender Gitternetzweite linear wächst. In Grün ist die Konditionszahl bei adaptiver Netzverfeinerung abgebildet. Diese ist ein $\mathcal{O}\left(N \log N \left(\frac{h_{\max}}{h_{\min}}\right)^2\right)$, wobei $h_{\max}$ der Durchmesser der längsten Gitternetzkante und $h_{\min}$ der Durchmesser der kürzesten Gitternetzkante ist. Verglichen mit der Schlitz-Geometrie wächst die Konditionszahl deutlich moderater. Wendet man die bereits beschriebene Vorkonditionierungsstrategie an, so wächst die Konditionszahl wie $\mathcal{O}(N \log(N)^2$.

Im uniformen Fall hat die Vorkonditionerung keinerlei Auswirkungen, weshalb dieser Fall nicht abgebildet ist. Auf die Stabilität des Algorithmus wirkt sich die Vorkonditionierung nicht aus.

Abschließend betrachten wir eine unterschiedliche Wahl einiger frei wählbarer Parameter

in der Implementierung des Einfachschichtpotentialoperators und untersuchen die Auswirkungen auf Stabilität und Genauigkeit des Algorithmus.
Konkret wählen wir, wie auch im Fall der Schlitz-Geometrie, die Maschinengenauigkeitskonstante EPS entweder 10^{-12}, 10^{-14} oder 10^{-15}. Den Quadraturgrad wählen wir mit 4, 8, 16 oder 32. Außerdem werden alle Rechnungen mit und ohne Vorkonditionierung durchgeführt.
Interessanterweise zeigen sich auch hier keine Auswirkungen auf die Stabilität des Algorithmus. In allen Fällen kann bis 22000 Randelemente gerechnet werden, ohne dass Instabilitäten auftreten. Alle Plots sehen im wesentlichen gleich aus wie Abbildung 11.
Bei der Genauigkeit ergeben sich gewisse Unterschiede, allerdings höchstens in der Größenordnung von 10^{-8}, was deutlich unter der tatsächlichen Genauigkeit von $2 \cdot 10^{-6}$ liegt.
Abbildung 13 zeigt nun die Differenz einiger Lösungen gegenüber einer Referenzlösung bei adaptiver Netzverfeinerung. Das Netz wurde hierbei anhand der Referenzlösung verfeinert. Diese wurde mit den Parametern Quadraturgrad 32, EPS 10^{-15} sowie mit aktivierter Vorkonditionierung der $\boldsymbol{V}$-Matrix gewählt. Entlang der x-Achse ist die Anzahl der Gitternetzkanten aufgetragen. Entlang der y-Achse ist die Energienorm der Differenz der jeweiligen Lösung und der Referenzlösung aufgetragen. Beide Achsen sind logarithmisch skaliert. Im Plot wurden EPS und die Vorkonditionierung für Quadraturgrad 4, 8 und 16 festgehalten, da die Unterschiede bei Änderung dieser Parameter lediglich in einer Größenordnung von 10^{-12} oder darunter liegen und somit vergleichsweise unbedeutend sind.
Um dies zu illustrieren, wurde zusätzlich jene Lösung mit Quadraturgrad 32, EPS 10^{-15} und ohne Vorkonditionierung geplottet. Diese stimmt mit der Refernezlösung also bis auf die Vorkonditionierung überein. Die Differenz liegt lediglich im Bereich von 10^{-14}.
Abschließend betrachten wir die Auswirkungen der adaptiven Netzverfeinerung auf das Netz. Hierzu plotten wir jeweils in der Mitte jedes Elements des Netzes dessen Länge. Die z-Achse, entlang derer die Längen der Elemente aufgetragen sind, ist logarithmisch skaliert. Abbildung 14(a) zeigt den so entstandenen Plot mit rund 22000 Elementen bzw. nach 74 Verfeinerungsschritten. Wiederum fällt auf, wie stark das Netz nahe den Singularitäten verfeinert wird.
Abbildung 14(b) zeigt den Plot einer Treppenfunktion, die man als Zwischenergebnis für ϕ_h erhält. Die Geometrie ist hierbei in Pünktchen eingezeichnet. Es wurde lediglich ein Netz mit rund 700 Randelementen verwendet um diesen Plot zu erzeugen.
Betrachtet man abschließend wiederum die maximale und minimale Gitternetzweite über den Netzverfeinerungsprozess hinweg, so entsteht der Plot in Abbildung 15. Hierbei ist entlang der x-Achse die Zahl der Kanten des jeweiligen Gitternetzes aufgetragen. Entlang der y-Achse sind die maximale Gitternetzweite $h_{\max}$, also die Länge des größten Elements des Netzes, die minimale Gitternetzweite $h_{\min}$, sowie der Quotient $h_{\max}/h_{\min}$ aufgetragen. Beide Achsen sind logarithmisch skaliert.

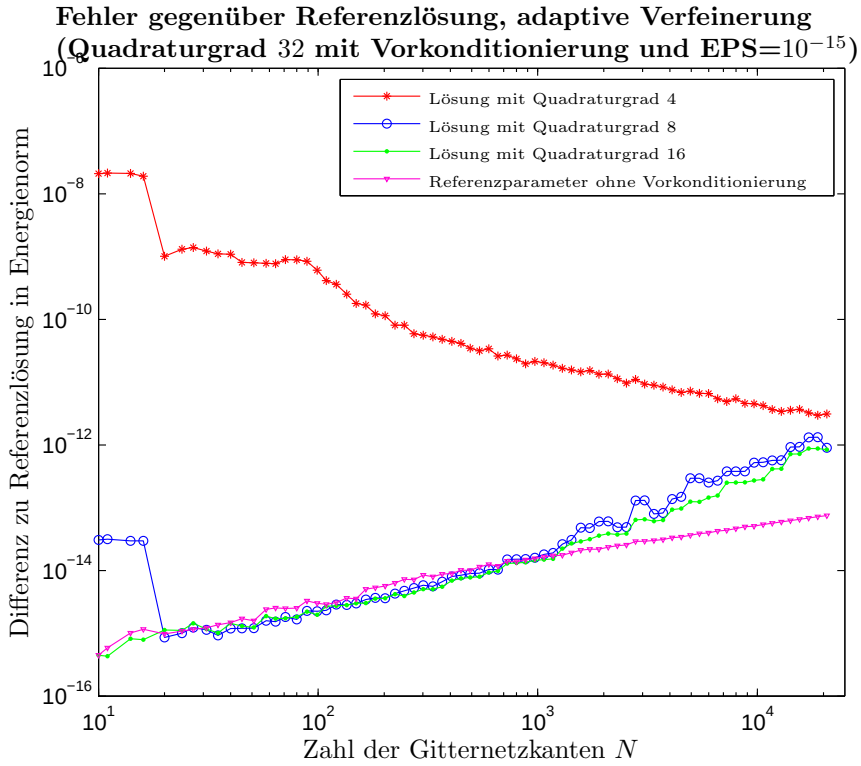
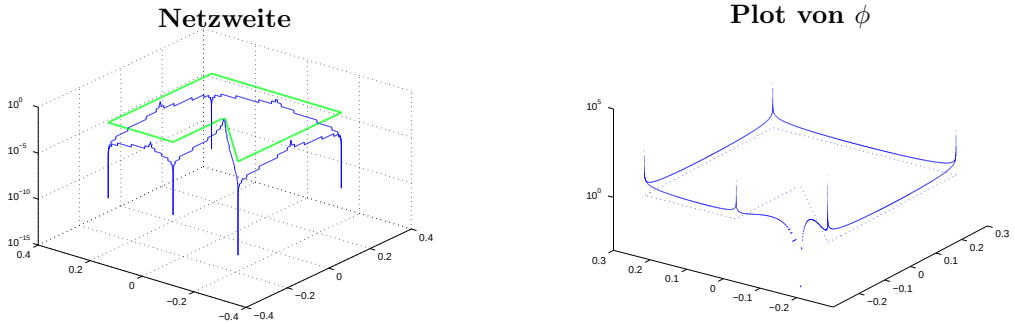


Abbildung 13: Genauigkeit gegenüber Referenzlösung bei adaptiver Verfeinerung (Z-Shape)



(a) Netzweite bei adaptiver Verfeinerung

(b) Zwischenergebnis für ϕ_h mit ca. 700 Randelementen

Abbildung 14: Endnetz und Zwischenergebnis mit ca. 700 Randelementen (Z-Shape).

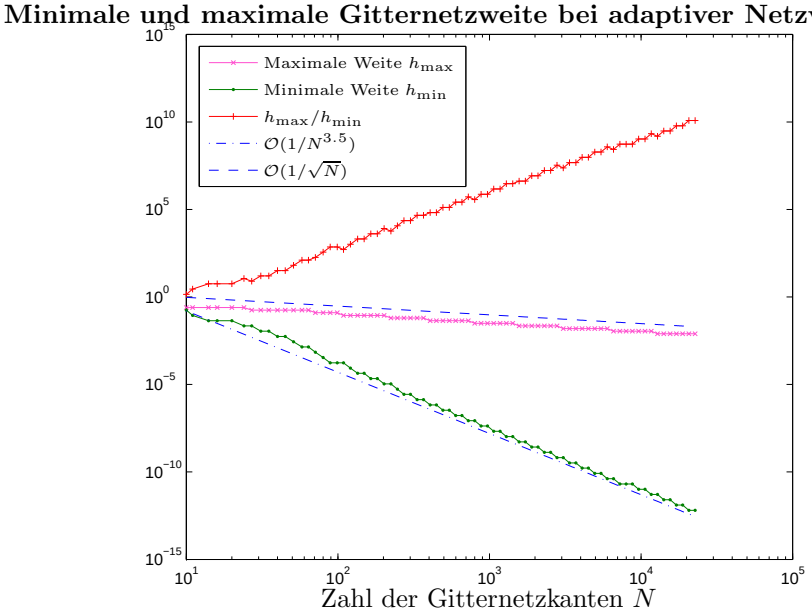


Abbildung 15: Netzweite bei adaptiver Netzverfeinerung (Z-Shape).

A Implementierung des V-Operators

Dieser Abschnitt enthält den Programmcode zur stabilen Berechnung des $\boldsymbol{V}$-Operators, der im Laufe dieser Bachelorarbeit weiterentwickelt wurde. Der Code wird im Zuge des FWF-Projektes P21732, während dem unter anderem die Programm-Bibliothek HILBERT[1] für adaptive Randelementmethode entsteht, gewartet und weiterentwickelt.
HILBERT enthält unter anderem stabile Implementierungen des Einfach- und Doppelschichtpotential-Operators und des hypersingulären Integral-Operators für die Laplace-Gleichung in 2D und Randelemente niedrigster Ordnung. Weiters sind diverse Fehlerschätzer, Markierungsstrategien, Netzverwaltungsroutinen und einige Visualisierungswerkzeuge enthalten. Darüber hinaus werden einige Beispiele zur Verfügung gestellt, in denen adaptive Algorithmen für die Integralformulierung des Dirichlet- und des Neumann-Problems sowie des gemischten Problems (mit Dirichlet- und Neumann-Bedingungen) implementiert sind. HILBERT ist für akademische Zwecke frei verfügbar.[2]

A.1 Übersicht

Die Funktion `mexFunction` bildet die Schnittstelle zu Matlab. Sie lest Eingabeparameter ein und allokiert Speicher für die Matrix $\boldsymbol{V}$. Anschließend ruft sie für jeden Matrix-Eintrag die Funktion `computeVij` auf, die dann für die Berechnung der Doppelintegrale der Gestalt

$$\int_{T_j}\int_{T_k} \log|\boldsymbol{x}-\boldsymbol{y}|ds_{\boldsymbol{y}}ds_{\boldsymbol{x}}$$

verantwortlich ist. Die Funktion stellt sicher, dass $\operatorname{diam}(T_j) \geq \operatorname{diam}(T_k)$, und unterscheidet anschließend, ob T_i und T_j zulässig oder unzulässig sind. Sind die Randelemente zulässig, so wird die Funktion `computeVij_semianalytic` zur Berechnung des Doppelintegrals aufgerufen. Sind die Randelemente hingegen unzulässig, so wird die Funktion `computeVij_analytic` aufgerufen.
Die Funktionen `slpWrapper` und `slp` dienen der Berechnung der einfachen Integrale

$$\int_{T_k} \log|\boldsymbol{x}-\boldsymbol{y}|ds_{\boldsymbol{y}}$$

für festes $\boldsymbol{x} \in \mathbb{R}^2$.
Die Funktion `ptoseg` bestimmt die Distanz zwischen einem Punkt und einem Randstück. Sie wird insbesondere von der Funktion `dist2` verwendet, die die Distanz zwischen zwei affinen Randstücken bestimmt.

A.2 Globale Variablen und Präprozessorkonstanten

Die Präprozessorkonstanten `GAUSS4_POINTS`, `GAUSS8_POINTS`, `GAUSS16_POINTS` wie auch `GAUSS32_POINTS` enthalten die Quadraturpunkte für 4, 8, 16 bzw. 32-Punkt Gauß-Quadratur. Die Konstanten `GAUSS4_WEIGHTS`, `GAUSS8_WEIGHTS`, `GAUSS16_WEIGHTS` beziehungsweise `GAUSS32_WEIGHTS` sind ebenfalls definiert. Sie enthalten die Quadraturgewichte für

[1] **H**ILBERT **I**s a **L**ovely **B**oundary **E**lement **R**esearch **T**ool
[2] siehe `http://www.asc.tuwien.ac.at/abem`.

4, 8, 16 bzw. 32-Punkt Gauß-Quadratur enthalten. All diese Konstanten initialisieren Arrays, d.h. sie sind von der Form {1, 2, 3, 4}.
Außerdem sind die Präprozessorkonstanten DEFAULT_EPS, DEFAULT_GAUSS_ORDER sowie DEFAULT_ETA definiert.
Zudem sind folgende globale Variablen definiert und initialisiert:

Listing 1: Globale Variablen

```
24 double eps = DEFAULT_EPS;
25 int gauss_order = DEFAULT_GAUSS_ORDER;
26 double* gauss_point;
27 double* gauss_wht;
```

A.3 mexFunction

Diese Funktion erhält als Eingabe die Arrays coordinates und elements, die die Triangulierung beschreiben. Hierbei ist coordinates eine $nC \times 2$-Matrix, wobei in jeder Zeile die Koordinaten eines Punktes aus $\mathbb{R}^2$ stehen und nC die Zahl der Randknoten des Randnetzes bezeichnet. elements ist eine $nE \times 2$-Matrix, wobei nE die Zahl der Randelemente des Randnetzes bezeichnet. Hierbei beschreibt jede Zeile ein Randelement, wobei an Stelle der Koordinaten lediglich die Nummer der entsprechenden Zeile in coordinates gespeichert wird. Alternativ kann ein weiterer Parameter η übergeben werden. Dabei handelt es sich um die Zulässigkeitskonstante.
Zurückgegeben wird die zur Triangulierung passende Steifigkeitsmatrix $\boldsymbol{V}$.

Listing 2: mexFunction

```
50 void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[]) {
51   const char* functionName = mexFunctionName();
52   char errorMessage[255];
53   int nC, nE;
54   int i, j;
55   double* V;
56   double* elements;
57   double* coordinates;
58   double* ptr;
59   double eta;
60   double a0,a1, b0,b1, c0,c1, d0,d1;
61   int aidx, bidx, cidx, didx;
62   double gauss4_point[]   = GAUSS4_POINTS;
63   double gauss4_weight[]  = GAUSS4_WEIGHTS;
64   double gauss8_point[]   = GAUSS8_POINTS;
65   double gauss8_weight[]  = GAUSS8_WEIGHTS;
66   double gauss16_point[]  = GAUSS16_POINTS;
67   double gauss16_weight[] = GAUSS16_WEIGHTS;
68   double gauss32_point[]  = GAUSS32_POINTS;
69   double gauss32_weight[] = GAUSS32_WEIGHTS;
70
71   if (nlhs != 1) {
72     sprintf(errorMessage,"Use␣either␣V␣=␣%s(coordinates,elements)\n"
73             "␣␣␣␣␣␣␣␣or␣V␣=␣%s(coordinates,elements,eta).",
74             functionName,functionName);
75     mexErrMsgTxt(errorMessage);
76   }
77
78   if ((nrhs != 2) && (nrhs != 3) && (nrhs != 5)) {
79     sprintf(errorMessage,"Use␣either␣V␣=␣%s(coordinates,elements)\n"
80             "␣␣␣␣␣␣␣␣or␣V␣=␣%s(coordinates,elements,eta)\n"
81             "␣␣␣␣␣␣␣␣or␣V␣=␣%s(coordinates,elements,eta,eps,gauss_order).\n",
82             functionName,functionName,functionName);
83     mexErrMsgTxt(errorMessage);
84   }
```

```
85
86    /* Read input data */
87    coordinates = mxGetPr(prhs[0]);
88    nC = mxGetM(prhs[0]);          /* number of nodes */
89
90    elements = mxGetPr(prhs[1]);
91    nE = mxGetM(prhs[1]);          /* number of elements */
92
93    if (nrhs == 3 || nrhs == 5) {
94      ptr = mxGetPr(prhs[2]);
95      eta = *ptr;
96      if (nrhs == 5) {
97          eps = mxGetScalar(prhs[3]);
98          gauss_order = (int) mxGetScalar(prhs[4]);
99      }
100     else {
101         eps = DEFAULT_EPS;
102         gauss_order = DEFAULT_GAUSS_ORDER;
103     }
104   }
105   else {
106     eta = 0;
107   }
108
109   switch (gauss_order) {
110     case 4:
111       gauss_point = gauss4_point;
112       gauss_wht   = gauss4_weight;
113       break;
114     case 8:
115       gauss_point = gauss8_point;
116       gauss_wht   = gauss8_weight;
117       break;
118     case 16:
119       gauss_point = gauss16_point;
120       gauss_wht   = gauss16_weight;
121       break;
122     case 32:
123       gauss_point = gauss32_point;
124       gauss_wht   = gauss32_weight;
125       break;
126     default:
127       sprintf(errorMessage, "Gauss order must either be 4, 8, 16 or 32.\n");
128       mexErrMsgTxt(errorMessage);
129   }
130
131   /* Allocate output data */
132   plhs[0] = mxCreateDoubleMatrix(nE,nE,mxREAL);
133   V = mxGetPr(plhs[0]);
134
135   /* Fill matrix V and use symmetry to reduce building time */
136   for (i=0; i<nE; ++i) {
137
138     aidx = (int) elements[i]-1;        /* 1st node of element Ti = [a,b] */
139     a0 = coordinates[aidx];
140     a1 = coordinates[aidx+nC];
141
142     bidx = (int) elements[i+nE]-1;    /* 2nd node of element Ti = [a,b] */
143     b0 = coordinates[bidx];
144     b1 = coordinates[bidx+nC];
145
146     for (j=i; j<nE; ++j) {
147       cidx = (int) elements[j]-1;      /* 1st node of element Tj = [c,d] */
148       c0 = coordinates[cidx];
149       c1 = coordinates[cidx+nC];
150
151       didx = (int) elements[j+nE]-1;  /* 2nd node of element Tj = [c,d] */
152       d0 = coordinates[didx];
153       d1 = coordinates[didx+nC];
154
```

```
155         V[i + j*nE] = computeVij(a0,a1,b0,b1, c0,c1,d0,d1,eta);
156         V[j + i*nE] = V[i + j*nE];
157       }
158     }
159   }
```

A.4 computeVij

Diese Funktion erhält als Eingabeparameter die Endpunkte $\boldsymbol{a} = [\texttt{a0}, \texttt{a1}], \boldsymbol{b} = [\texttt{b0}, \texttt{b1}]$ eines Randstückes T_i sowie die Endpunkte $\boldsymbol{c} = [\texttt{c0}, \texttt{c1}], \boldsymbol{d} = [\texttt{d0}, \texttt{d1}]$ des Randstückes T_j. Weiters erhält sie `eta`, die Zulässigkeitskonstante. Die Funktion überprüft zunächst die Zulässigkeitsbedingung für die Randstücke T_i, T_j und ruft, je nachdem ob die Zulässigkeitsbedingung erfüllt ist oder nicht, `computeVij_semianalytic` oder `computeVij_analytic` auf um

$$\frac{-1}{2\pi} \int_{T_i} \int_{T_j} \log |\boldsymbol{x} - \boldsymbol{y}| ds_{\boldsymbol{y}} ds_{\boldsymbol{x}} \tag{53}$$

zu berechnen. Anschließend wird der berechnete Wert zurückgegeben.

Listing 3: computeVij

```
165 double computeVij(double a0, double a1, double b0, double b1,
166                   double c0, double c1, double d0, double d1, double eta) {
167   /*
168    * INPUT:  elements Ti = [a,b], Tj = [c,d] with a,b,c,d \in \R^2
169    * OUTPUT: Galerkin integral -1/(2pi) \int_{Tj} \int_{Ti} log|x-y| ds_y ds_x
170    */
171
172   double hi = (b0-a0)*(b0-a0) + (b1-a1)*(b1-a1); /* hi = norm(b-a)^2 */
173   double hj = (d0-c0)*(d0-c0) + (d1-c1)*(d1-c1); /* hj = norm(d-c)^2 */
174   double tmp;
175
176   /* For stability reasons, we guarantee   hj <= hi   to ensure that *
177    * outer integration is over smaller domain. This is done by       *
178    * swapping Tj and Ti if necessary.                                */
179
180   if (hj > hi) {
181     tmp = a0; a0 = c0; c0 = tmp;   /* swap a and c */
182     tmp = a1; a1 = c1; c1 = tmp;
183     tmp = b0; b0 = d0; d0 = tmp;   /* swap b and d */
184     tmp = b1; b1 = d1; d1 = tmp;
185     tmp = hi; hi = hj; hj = tmp;   /* ensure that hj <= hi */
186   }
187
188   if ( eta == 0) { /* compute all matrix entries analytically */
189
190     return computeVij_analytic(a0,a1, b0,b1, c0,c1, d0,d1);
191
192   }
193   else { /* compute admissible matrix entries semi-analytically */
194
195     if ( dist2(a0,a1,b0,b1,c0,c1,d0,d1) > eta*sqrt(hj) ) {
196       return computeVij_semianalytic(a0,a1, b0,b1, c0,c1, d0,d1);
197     }
198     else {
199       return computeVij_analytic(a0,a1, b0,b1, c0,c1, d0,d1);
200     }
201   }
202 }
```

A.5 computeVij_analytic

Diese Funktion berechnet (53) analytisch.

Listing 4: computeVij_analytic

```
208 double computeVij_analytic(double a0, double a1, double b0, double b1,
209                            double c0, double c1, double d0, double d1) {
210
211   /*
212    * INPUT:  elements Ti = [a,b], Tj = [c,d] with a,b,c,d \in \R^2
213    * OUTPUT: Galerkin integral -1/(2pi) \int_{Tj} \int_{Ti} log|x-y| ds_y ds_x
214    */
215
216   double hi = (b0-a0)*(b0-a0) + (b1-a1)*(b1-a1); /* hi = norm(b-a)^2 */
217   double hj = (d0-c0)*(d0-c0) + (d1-c1)*(d1-c1); /* hj = norm(d-c)^2 */
218   double tmp, val, det;
219   double x[2], y[2], z[2];
220   double zxp[2], zxm[2], zyp[2], zym[2];
221   double lambda, mu;
222
223   x[0] = 0.5*(b0 - a0);             /* x = (b-a)/2 */
224   x[1] = 0.5*(b1 - a1);
225   y[0] = 0.5*(c0 - d0);             /* y = (c-d)/2 */
226   y[1] = 0.5*(c1 - d1);
227   z[0] = 0.5*(a0 + b0 - c0 - d0);   /* z = (a+b-c-d)/2 */
228   z[1] = 0.5*(a1 + b1 - c1 - d1);
229
230   zxp[0] = z[0] + x[0];             /* zxp = z+x = (2b-c-d)/2 */
231   zxp[1] = z[1] + x[1];
232   zxm[0] = z[0] - x[0];             /* zxm = z-x = (2a-c-d)/2 */
233   zxm[1] = z[1] - x[1];
234   zyp[0] = z[0] + y[0];             /* zyp = z+y = (a+b-2d)/2 */
235   zyp[1] = z[1] + y[1];
236   zym[0] = z[0] - y[0];             /* zym = z-y = (a+b-2c)/2 */
237   zym[1] = z[1] - y[1];
238
239   /* There hold different recursion formulae if Ti and Tj *
240    * are parallel (det = 0) or not                        */
241
242   det = x[0]*y[1] - x[1]*y[0];
243
244   if ( fabs(det) <= eps*sqrt(hi*hj) ) { /* case that x and y are linearly dependent, i.e., Ti and Tj ar
245
246     if ( fabs(x[0]) < fabs(x[1]) )
247       lambda = y[1] / x[1];
248     else
249       lambda = y[0] / x[0];
250
251     val = 0.5*( lambda * ( slpWrapper(1, y, zxm) - slpWrapper(1, y, zxp) )
252                  + slpWrapper(0, x, zyp) + slpWrapper(0, x, zym) );
253   }
254
255   else { /* case that x and y are linearly independent */
256
257     lambda = (z[0]*y[1] - z[1]*y[0]) /det;
258     mu = (x[0]*z[1] - x[1]*z[0]) /det;
259
260     val = 0.25 * ( -8 + (lambda+1)*slpWrapper(0, y, zxp) - (lambda-1)*slpWrapper(0, y, zxm)
261                    + (mu+1)*slpWrapper(0, x, zyp) - (mu-1)*slpWrapper(0, x, zym) );
262   }
263
264   return -0.125*sqrt(hi*hj)*val /PI; /* = - 1/(2*PI) * |Ti|/2 * |Tj|/2 * val */
265 }
```

A.6 computeVij_semianalytic

Diese Funktion berechnet (53) semi-analytisch mittels Gauß-Quadratur.

Listing 5: computeVij_semianalytic

```
271 double computeVij_semianalytic(double a0, double a1, double b0, double b1,
```

```
272                                 double c0, double c1, double d0, double d1) {
273
274   /*
275    * INPUT:  elements Ti = [a,b], Tj = [c,d] with a,b,c,d \in \R^2
276    * OUTPUT: approximate Galerkin integral -1/(2pi) \int_{Tj} \int_{Ti} log|x-y| ds_y ds_x,
277    *         where outer integration is performed by Gaussian quadrature
278    */
279
280   int k;
281   double hi = (b0-a0)*(b0-a0) + (b1-a1)*(b1-a1); /* hi = norm(b-a)^2 */
282   double hj = (d0-c0)*(d0-c0) + (d1-c1)*(d1-c1); /* hj = norm(d-c)^2 */
283   double u[2], v[2];
284   double val = 0;
285   double sx0 = 0;
286   double sx1 = 0;
287
288
289   u[0] = 0.5*(a0-b0);
290   u[1] = 0.5*(a1-b1);
291
292   for (k=0; k<gauss_order; ++k){
293       /* transformation of quadrature nodes from [-1,1] to [a,b] */
294       sx0 = ((1-gauss_point[k])*c0+(1+gauss_point[k])*d0)*0.5;
295       sx1 = ((1-gauss_point[k])*c1+(1+gauss_point[k])*d1)*0.5;
296
297       v[0] = sx0 - 0.5*(a0+b0);
298       v[1] = sx1 - 0.5*(a1+b1);
299
300       /* inner product wht*func(sx) */
301       val += gauss_wht[k]*slpWrapper(0, u, v);
302   }
303
304   return -0.0625*sqrt(hi*hj)*val /PI; /* = - 1/(2*PI) * |Ti|/2 * |Tj|/2 * 0.5 * int(log |.|^2) */
305 }
```

A.7 `slpWrapper`

Diese Funktion erhält als Eingabeparameter einen Integer `k`, sowie zwei Vektoren $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2$, die mittels der Arrays `u`, `v` an die Funktion übergeben werden. Sie berechnet das Integral

$$\int_{-1}^{+1} s^k \log|s\boldsymbol{u} + \boldsymbol{v}| ds$$

und gibt den berechneten Wert zurück.

Listing 6: slpWrapper

```
311 double slpWrapper(int k, double u[2],double v[2]) {
312   /*
313    * INPUT: Vectors u,v \in \R^2, integer k
314    * OUTPUT: value of SLP-type integral
315    *         \int_{-1}^{+1} s^k \log |s*u+v|^2 ds
316    */
317
318   double a = u[0]*u[0] + u[1]*u[1];          /* a = <u,u> */
319   double b = 2 * ( u[0]*v[0] + u[1]*v[1] ); /* b = 2 <u,v> */
320   double c = v[0]*v[0] + v[1]*v[1];          /* c = <v,v> */
321   return slp(k,a,b,c);
322 }
```

A.8 `slp`

Diese Funktion erhält als Eingabe einen Integer `k` sowie drei Gleitkommazahlen `a`, `b` sowie `c` und berechnet das Integral

$$\int_{-1}^{+1} s^k \log |as^2 + bs + c| ds.$$

Der berechnete Wert wird anschließend zurückgegeben.

Listing 7: slp

```
325 double slp(int k, double a, double b, double c) {
326   /*
327    * INPUT: scalars a,c>0 and b \in \R, integer k
328    * OUPUT: value of SLP-type integral
329    *        \int_{-1}^{+1} s^k \log |a*s^2+b*s+c| ds
330    */
331
332   double val;
333   double tmp;
334   double D;
335
336   /* Ensure that discriminant is either positive or zero */
337   tmp = 4*a*c - b*b; /* Note that by theory tmp >= 0 */
338   if (tmp > eps*4*a*c)
339     D = sqrt(tmp);
340   else
341     D = 0;
342
343   /* The case k=0 */
344   if (D == 0) {
345     tmp = b + 2*a;
346     if (fabs(tmp) > eps*a)
347       val = tmp * log( 0.25*tmp*tmp /a );
348     else
349       val = 0;
350     tmp = b - 2*a;
351     if (fabs(tmp) > eps*a)
352       val -= tmp * log( 0.25*tmp*tmp /a );
353     val = 0.5*val /a - 4;
354   }
355   else { /* case D > 0 */
356     tmp = c - a;
357     if (fabs(tmp) < eps*c)
358       val = 0.5*PI;
359     else if (a < c)
360       val = atan( D /tmp );
361     else
362       val = atan( D /tmp ) + PI;
363
364     val = ( 0.5*( (b+2*a) * log(a+b+c) - (b-2*a) * log(a-b+c) ) + D*val )/a - 4;
365   }
366
367   /* The case k=1 */
368   if (k==1) {
369     val = -b*(2+val);
370
371
372     tmp = a+b+c;
373     if (fabs(tmp) > eps*a)
374       val += tmp * log(tmp);
375
376     tmp = a-b+c;
377     if (fabs(tmp) > eps*a)
378       val -= tmp * log(tmp);
379
380     val /= (2*a);
```

```
381     }
382
383     return val;
384 }
```

A.9 dist2

Diese Funktion erhält als Eingabeparameter die Endpunkte $\boldsymbol{a} = [\texttt{a0}, \texttt{a1}], \boldsymbol{b} = [\texttt{b0}, \texttt{b1}]$ eines Randstückes T_i sowie die Endpunkte $\boldsymbol{c} = [\texttt{c0}, \texttt{c1}], \boldsymbol{d} = [\texttt{d0}, \texttt{d1}]$ des Randstückes T_j. Sie berechnet daraufhin den Abstand zwischen den Randstücken $\text{dist}(T_i, T_j)$, indem sie den Abstand zwischen jedem Endpunkt und der jeweils anderen Strecke ermittelt und über diese Werte das Minimum bildet. Siehe dazu auch Abschnitt 2.

Listing 8: dist2

```
396 double dist2(double a0, double a1, double b0, double b1,
397                double c0, double c1, double d0, double d1) {
398
399 double test;
400 double best = sqrt((a0-c0)*(a0-c0)+(a1-c1)*(a1-c1));
401 double val;
402
403 /* The only possibility that these two segments intersect: */
404 if ((fabs(a0-d0)<=eps && fabs(a1-d1)<=eps) || (fabs(b0-c0)<=eps && fabs(b1-c1)<=eps))
405     best=0;
406 /* If segements do not intersect: */
407 else{
408     /* Calculate the distance from A to segment [C,D]: */
409     test = ptoseg(a0, a1, c0, c1, d0, d1);
410     if ((test-best)<=eps){
411         best = test;
412     }
413     /* Calculate the distance from B to segment [C,D]: */
414     test = ptoseg(b0, b1, c0, c1, d0, d1);
415     if ((test-best)<=eps){
416         best = test;
417     }
418     /* Calculate the distance from C to segment [A,B]: */
419     test = ptoseg(c0, c1, a0, a1, b0, b1);
420     if ((test-best)<=eps){
421         best = test;
422     }
423     /* Calculate the distance from D to segment [A,B]: */
424     test = ptoseg(d0, d1, a0, a1, b0, b1);
425     if ((test-best)<=eps){
426         best = test;
427     }
428 }
429
430 return best;
431 }
```

A.10 ptoseg

Diese Funktion erhält als Eingabeparameter einen Punkt $\boldsymbol{p} = [\texttt{p0}, \texttt{p1}]$ sowie die Endpunkte $\boldsymbol{a} = [\texttt{a0}, \texttt{a1}]$ und $\boldsymbol{b} = [\texttt{b0}, \texttt{b1}]$ eines Randstückes $T_i = [\boldsymbol{a}, \boldsymbol{b}]$. Sie berechnet anschließend die Distanz $\text{dist}(\boldsymbol{p}, T_i)$ zwischen dem Punkt $\boldsymbol{p}$ und des Randstückes T_i und gibt den berechneten Wert zurück.

Listing 9: ptoseg

```
445 double ptoseg(double p0, double p1, double a0, double a1,double b0, double b1){
```

```
446
447 double t, tmp1, tmp2;
448 double val;
449
450 tmp1 = b0-a0;
451 tmp2 = b1-a1;
452
453 if (tmp1<=eps && tmp2<=eps){
454     tmp1 = p0-a0;
455     tmp2 = p1-a1;
456 }
457 else{
458     t = ((p0-a0)*(b0-a0)+(p1-a1)*(b1-a1))/((b0-a0)*(b0-a0)+(b1-a1)*(b1-a1));
459
460     if (t<=eps){
461         tmp1 = p0-a0;
462         tmp2 = p1-a1;
463     }
464     else if ((1-t)<=eps){
465         tmp1 = p0-b0;
466         tmp2 = p1-b1;
467     }
468     else{
469         tmp1 = p0 - a0 - t*tmp1;
470         tmp2 = p1 - a1 - t*tmp2;
471     }
472 }
473 val  = sqrt(tmp1*tmp1 + tmp2*tmp2);
474 return val;
475 }
```

B Implementierung der Vorkonditionierung des V-Operators

Die Funktion `precondV` ist für die Vorkonditionierung der Steifigkeitsmatrix und des zugehörigen Vektors verantwortlich. Zum Einsatz kommt eine einfache Diagonalskalierung. Sei also

$$\boldsymbol{V}\boldsymbol{x} = \boldsymbol{g},$$

ein lineares Gleichungssystem, wobei $\boldsymbol{V}$ die Steifigkeitsmatrix bezeichnet und $\boldsymbol{g}$ die zugehörigen Randdaten enthält. Betrachtet man die Diagonalmatrix $\boldsymbol{D}$, die gegeben ist durch

$$\boldsymbol{D}_{ij} := \delta_{ij}\boldsymbol{V}_{ij}$$

und setzt

$$\widehat{\boldsymbol{V}} := \boldsymbol{D}^{-1/2}\boldsymbol{V}\boldsymbol{D}^{-1/2}$$
$$\widehat{\boldsymbol{g}} := \boldsymbol{D}^{-1/2}\boldsymbol{g},$$

wobei zu bemerken ist, dass alle $\boldsymbol{V}_{ii} \neq 0$ sind für $i = 1, ..., N$, weshalb $\boldsymbol{D}$ vollen Rang hat und $\boldsymbol{D}^{-1/2}$ überhaupt erst definiert ist. Man erhält so ein weiteres Gleichungssystem

$$\widehat{\boldsymbol{V}}\boldsymbol{y} = \widehat{\boldsymbol{g}}.$$

Man löst nun das vorkonditionierte System und stellt fest, dass $\boldsymbol{y}$ genau dann das vorkonditionierte System löst, wenn $\boldsymbol{x} := \boldsymbol{D}^{-1/2}\boldsymbol{y}$ das ursprüngliche System löst.
Die Funktion `precondV` erhält nun als Eingabe die Matrix $\boldsymbol{V}$ und den Vektor $\boldsymbol{g}$. Sie gibt den Vektor $\boldsymbol{d} = (\boldsymbol{D}_{11}^{-1/2}, ..., \boldsymbol{D}_{NN}^{-1/2})$ zurück. Weiters modifiziert sie die Eingabeparameter, sodass nach dem Aufruf an Stelle von $\boldsymbol{V}$ die Matrix $\widehat{\boldsymbol{V}}$ steht und an Stelle von $\boldsymbol{g}$ der Vektor $\widehat{\boldsymbol{g}}$.
Diese Funktion ist in C implementiert, da Matlab für die Multiplikation einer Diagonalmatrix mit einer vollbesetzten Matrix $\mathcal{O}(N^3)$ Operationen benötigt, aber lediglich $\mathcal{O}(N^2)$ Operationen notwendig sind. Diese Implementierung ist daher bedeutend schneller für große Matrizen $\boldsymbol{V}$.

Listing 10: precondV

```
4  void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[])
5  {
6      const char* function_name = mexFunctionName();
7      char errorMsg[255];
8      double *A = NULL, *b = NULL, *d = NULL;
9      size_t mSize = 0;
10     int i = 0, j = 0;
11
12     if (nlhs != 1 || nrhs != 2) {
13         sprintf(errorMsg, "The␣function␣signature␣is\n"
14             "\td␣=␣%s(A,␣b)\n"
15             "when␣A␣is␣a␣nxn-Matrix␣and␣b␣is␣a␣n-dimensional␣vector.\n",
16             function_name);
17         mexErrMsgTxt(errorMsg);
18     }
19
20     A = mxGetPr(prhs[0]);
21     b = mxGetPr(prhs[1]);
22     mSize = mxGetM(prhs[0]);
23
24     if (mSize != mxGetM(prhs[1])) {
25         sprintf(errorMsg, "Matrix␣and␣vector␣dimensions␣must␣match.");
26         mexErrMsgTxt(errorMsg);
27     }
28
29     /* Initialize the d-vector. */
30     plhs[0] = mxCreateDoubleMatrix(mSize, 1, mxREAL);
31     d = mxGetPr(plhs[0]);
32
33     for (i = 0; i < mSize; ++i)
34             d[i] = pow(A[i*mSize + i], -0.5);
35
36     /* Modify A */
37     for (i = 0; i < mSize; ++i) {
38         for (j = 0; j < mSize; ++j) {
39             A[i*mSize+j] = d[i] * A[i*mSize+j] * d[j];
40         }
41     }
42
43     /* Modify b */
44     for (i = 0; i < mSize; ++i) {
45         b[i] *= d[i];
46     }
47 }
```

Abbildungsverzeichnis

1 Die drei zu unterscheidenden Fälle 8
2 Die unterschiedlichen Beispiel-Geometrien 27
3 Fehler und Fehlerschätzer bei uniformer Netzverfeinerung (Schlitz-Geometrie) 29
4 Fehler und Fehlerschätzer bei adaptiver Netzverfeinerung (Schlitz-Geometrie) 29
5 Konditionszahl von V (Schlitz-Geometrie) 30
6 Genauigkeit gegenüber Referenzlösung bei uniformer Verfeinerung (Schlitz-Geometrie) 31
7 Genauigkeit gegenüber Referenzlösung bei adaptiver Verfeinerung (Schlitz-Geometrie) 32
8 Netzweite bei adaptiver Netzverfeinerung bei Gauß-Quadraturgrad 16, EPS 10^{-12} und mit Vorkonditionierung. (Schlitz-Geometrie) 33
9 Minimale und maximale Gitternetzweite bei adaptiver Netzverfeinerung (Schlitz-Geometrie) 34
10 Fehler und Fehlerschätzer bei uniformer Netzverfeinerung (Z-Shape) . . . 35
11 Fehler und Fehlerschätzer bei adaptiver Netzverfeinerung (Z-Shape) . . . 35
12 Konditionszahl von V (Z-Shape) 36
13 Genauigkeit gegenüber Referenzlösung bei adaptiver Verfeinerung (Z-Shape) 38
14 Endnetz und Zwischenergebnis mit ca. 700 Randelementen (Z-Shape). . . 38
15 Netzweite bei adaptiver Netzverfeinerung (Z-Shape). 39

Literatur

[1] Ainsworth, M., W. McLean und T. Tran: *The conditioning of boundary element equations on locally refined meshes and preconditioning by diagonal scaling*. SIAM J. Numer. Anal., 36(6):1901–1932 (electronic), 1999.

[2] Auzinger, W. und D. Praetorius: *Numerische Mathematik*, 2007.

[3] Börm, S., M. Löhndorf und J. M. Melenk: *Approximation of integral operators by variable-order interpolation*. Numer. Math., 99(4):605–643, 2005.

[4] Dörfler, W.: *A Convergent Adaptive Algorithm for Poisson's Equation*. SIAM Journal on Numerical Analysis, 33(3):1106–1124, 1996.

[5] Ferraz-Leite, S. und D. Praetorius: *Simple a posteriori error estimators for the h-version of the boundary element method*. Computing, 83(4):135–162, 2008.

[6] Königsberger, K.: *Analysis 2*. Springer-Verlag, 2004.

[7] Mund, P., E. P. Stephan und J. Weisse: *Two-level methods for the single layer potential in* $\mathbf{R}^3$. Computing, 60(3):243–266, 1998.

[8] Plato, R.: *Numerische Mathematik kompakt*. Vieweg Verlag, 2006.

[9] Praetorius, D.: *Introduction to Boundary Element Method*. 2007.