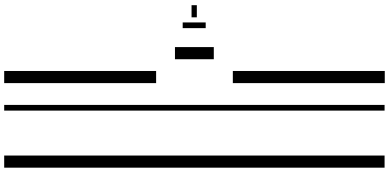

Optimierung von Formfunktionalen mit NCG und Quasi-Newton-Methoden

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Institut für Numerische und Angewandte Mathematik

Betreuung:
Prof. Dr. Benedikt Wirth

Eingereicht von:
Jonas Wanka

Münster, 19. November 2015

Zusammenfassung

Höhenlinien, oder auch Level-Set Funktionen genannt, bieten nicht nur Geographen wichtige Informationen. Level-Set Funktionen finden auch in der Bildverarbeitung Anwendung. Ziel dieser Arbeit ist es, Level-Set Funktionen mit der Formoptimierung zu kombinieren und diese mithilfe von nicht-linearen Verfahren zu diskretisieren.

Dazu schaffen wir im ersten Teil der Arbeit die notwendigen analytischen Grundlagen der Formoptimierung, der Shape Sensitivity Analysis, und stellen das Chan-Vese Energiefunktional sowie die genutzen Algorithmen vor, mit welchen wir Bilder segmentieren wollen. Im zweiten Teil implementieren wir das Gradientenabstiegsverfahren, das nichtlineare cg-Verfahren und das Quasi-Newton Verfahren für Formfunktionen mit Level-Set Funktionen und vergleichen diese mit der klassischen Heaviside-Implementierung.

Eidesstattliche Erklärung

Hiermit versichere ich, *Jonas Wanka*, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Gedanklich, inhaltlich oder wörtlich Übernommenes habe ich durch Angabe von Herkunft und Text oder Anmerkung belegt bzw. kenntlich gemacht. Dies gilt in gleicher Weise für Bilder, Tabellen, Zeichnungen und Skizzen, die nicht von mir selbst erstellt wurden. Alle auf der CD beigefügten Programme sind von mir selbst programmiert worden mit Ausnahme der im Quellecode markierten Methoden.

Münster, 19. November 2015

Jonas Wanka

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Münster, 19. November 2015

Jonas Wanka

Danksagung

Ich möchte mich an dieser Stelle bei all denjenigen bedanken, die mich während meines Studiums persönlich als auch fachlich unterstützt haben.

Mein besonderen Dank richtet sich an Prof. Dr. Benedikt Wirth, der auf jegliche Frage eine anschauliche Antwort wusste und mir die Möglichkeit gab, mich mit diesem Thema zu beschäftigen.

Ein herzliches Dankeschön geht an meine Familie, die mich jahrelang während meines Studiums nicht nur finanziell sondern auch durch bedingungslosen Rückhalt unterstützt hat. Danke, dass ihr immer für mich da seid.

Besonders bedanke ich mich bei meinen Freunden, die selbst die langweiligste Vorlesung zu einer lustigen Veranstaltung haben werden lassen, Stunden des Zettelrechnens im Fluge haben vergehen lassen und auch außerhalb der Universität genügend Ideen für Ablenkungen gegeben haben. Danke für diese wundervolle und unvergessliche Zeit.

Inhaltsverzeichnis

1. Einleitung

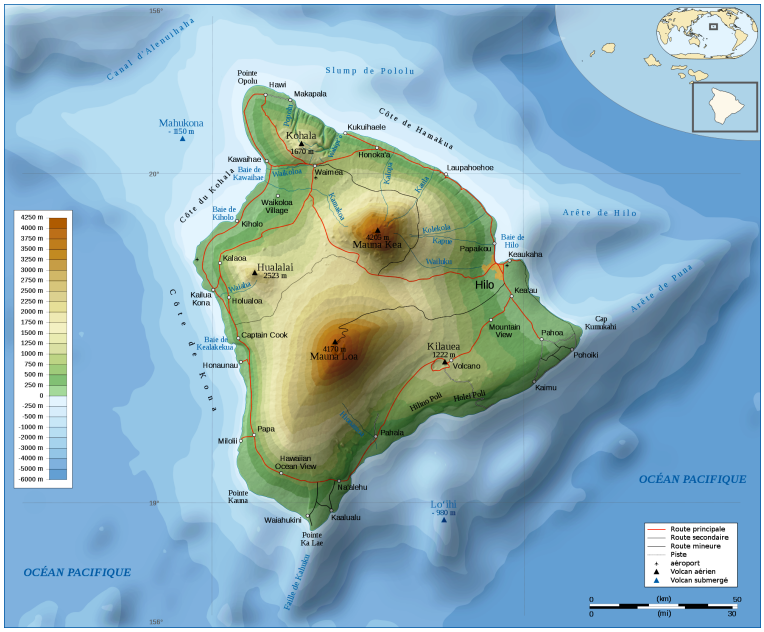

Abbildung 1.1.: Höhenkarte von Hawaii [Quelle [18]]

Nicht nur bei topographischen Karten sind Höhenlinien, oder auch Level-Set Funktionen genannt, wichtige Informationsträger. In den letzten Jahren haben Level-Set Funktionen deutlich an Nutzen gewonnen. Immer mehr mathematische Probleme vor allem in der Bildverarbeitung lassen sich mithilfe dieser Funktionen einfacher modellieren und diskretisieren. So wird nun auch in der Formoptimerung der Level-Set-Ansatz genutzt, um bekannte Probleme einfacher zu simulieren.

Der Ansatz besteht darin Strukturen durch eine implizite Funktion darzustellen, wobei der Rand durch die Null-Niveaumenge dargestellt wird. Im Inneren des Objekts oder der Form ist die Funktion positiv und außerhalb negativ. Mithilfe dieser Darstellungen lassen sich einfache Evolutionsgleichungen entwickeln, die eine initial gewählte Funktion so lange bewegen, bis sie die Konturen des Objektes darstellen. Vor allem die Null-Niveaumenge ist hierbei interessant, da sie die grundlegende Form des Objektes angibt.

Des Weiteren spielt die Shape Sensitivity Analysis eine große Rolle, da die Evoluti-

onsgleichungen der Level-Set Funktionen auf dem Gradienten am Rand des Objektes angewiesen sind. Jedoch ist für Formen als Untermannigfaltigkeit der Ableitungsbegriff nicht definiert und muss erarbeitet werden. Die Shape Sensitivity Anaylsis bietet uns die analytische Grundlage, um Grenzwerte mit Ableitungen zu identifizieren, die wir im Anschluss für numerische Simulationen nutzen können.

Ein Schwerpunkt dieser Arbeit soll es sein, die benötigten Grundlagen für Optimierungsverfahren mit Blick auf die Bildverarbeitung zu erarbeiten. Hierbei werden die Level-Set Funktionen zur Diskretisierung genutzt und mithilfe der Shape Sensitivity Analysis der Ableitungsbegriff definiert. Ein besonderes Augenmerk soll dann auf der Erweiterung der numerischen Verfahren liegen. Hierbei wollen wir die bekannten linearen Verfahren zu höherer Ordnung verallgemeinern und das Konvergenzverhalten untersuchen. Die Schwierigkeit bei der Nutzung von nicht-linearen Verfahren ist, dass die Ableitung der Shape-Analysis nur auf dem Rand des Objektes, also der Null-Niveaumenge der Level-Set Funktion existiert. Jedoch wird diese in jedem Schritt bewegt und somit muss auch die Ableitung "mitbewegt"werden, um bei der Berechnung des nächsten Schrittes wieder genutzt werden zu können.

Eine Einordnung dieser Arbeit in die Bildverarbeitung und eine kurze Einführung der benötigten Begriffe für Level-Set Funktionen und Gradientenabstiegsverfahren wird in **Kapitel 2** gegeben. Im Anschluss wollen wir uns mit den analytischen Grundlagen beschäftigen. In **Kapitel 3** werden dazu die nötigen Formableitungen angegeben und als wichtiges Hilfsmittel der Struktursatz hergeleitet. Im Anschluss erarbeiten wir die nötigen Grundlagen, um nicht-lineare Verfahren mit den Level-Set Funktionen zu nutzen. Dazu wollen wir in **Kapitel 4** den Transport des Gradienten verdeutlichen und zwei nicht-lineare Verfahren, das Quasi-Netwon und das nicht-lineare cg-Verfahren, angeben und vergleichen. In **Kapitel 5** geben wir als Anwendungsbeispiel die Bildsegmentierung mit dem Chan-Vese Energiefunktional an, welches die Verfahren in der Praxis vergleichen soll. Zum Abschluss wollen wir unsere Ergebnisse zusammenfassen und einen Ausblick auf weitere interessante Fragestellungen geben.

2. Grundlagen

2.1. Bildverarbeitung in der Mathematik

Die automatische Verarbeitung digitaler Bilder ist in vielen Gebieten von fundamentaler Bedeutung, etwa in der Medizin (Bilder aus der Tomographie), in den Naturwissenschaften (Bilder aus der Mikroskopie), in der Kunst (Gemälde) oder auch im alltäglichen Leben (Bilder aus Digitalkameras). Als Grundlage zur Verarbeitung dieser Bilder dienen häufig mathematische Verfahren und Modelle, die die gewünschten Eigenschaften des Bildes wiederherstellen oder andere Manipulationen und Rückschlüsse aus den Daten zulassen. Die Grundprobleme umfassen zum Beispiel Entrauschen, Scharfzeichnen, Kantenerkennung oder das Inpainting (Quelle [4]). All diese Verfahren nutzen die Optimierung eines Energiefunktionals, um den gewünschten Zustand zu erreichen. Zur Einführung wollen wir uns mit der Bildsegmentierung beschäftigen.

2.1.1. Bildsegmentierung

Zu Beginn stellt sich die Frage: Wie repräsentieren wir ein Bild?
Definition 2.1.1 (Bilder in der Mathematik)**.** *Ein Bild ist eine Funktion u, die jedem Punkt im Definitionsbereich, z.B. dem $\mathbb{R}^2$, einen Farbwert zuordnet. Sei also $D \subset \mathbb{R}^2$ ein Schwarzweißbild, dann gilt:*

$$u : D \to [0, 1] \tag{2.1}$$

Das Intervall [0,1] beschreibt hierbei die Grauwerte des Bildes.

Die Erzeugung von inhaltlich zusammenhängenden Regionen durch Zusammenfassung benachbarter Pixel entsprechend eines bestimmten Homogenitätskriteriums bezeichnet man als *Segmentierung*. Ziel ist es eine parametrisierte Kurve $C(s) : [0, 1] \to \mathbb{R}$ entlang der Kanten des Vorder-/Hintergrundes zu finden. Diese Kontur kann mehrere Objekte dem Vorder- bzw. Hintergrund zuordnen.

Im „klassischen“ Modell werden Kanten über den Gradienten des Bildes u_0 erkannt. Es wird jeweils das $inf_C J_1(C)$ für eine Kurve C gesucht für das gilt:

$$J_1(C) = \alpha \int_0^1 |C'(s)|^2 ds + \beta \int_0^1 |C''(s)| ds - \lambda \int_0^1 |\nabla u_0(C(s))|^2 ds \tag{2.2}$$

α, β, λ sind dabei positive Gewichte der Terme. Die ersten beiden Terme beschreiben die *innere Energie*, also die Glattheit der Kurve, und der dritte Term die *externe Energie*, also die Anziehung der Kurve an die Kante des Objektes im Bild.
Wir wollen $|\nabla u_0(C(s))|$ maximieren und dabei eine gewisse Glattheit der Kurve erhalten.

Ein vereinfachtes Modell, welches annimmt, dass u_0 aus zwei konstanten Regionen besteht, wurde 2001 von Chan und Vese vorgestellt [Quelle [5]]. Dadurch lässt sich das Energiefunktional (2.2) wie folgt vereinfachen. Nehmen wir an, dass das Bild nur aus zwei stückweise-konstanten Regionen mit Werten c_1 im Inneren des Objekts und c_2 außerhalb besteht und die zu erkennende Kante C_0 am Rand von c_1 liegt. Um die Energie in beiden Regionen widerzuspiegeln, schreiben wir jeweils für beide Regionen:

$$F_1(C) + F_2(C) = \int_{inside(C)} |u_0(x,y) - c_1|^2 dxdy + \int_{outside(C)} |u_0(x,y) - c_2|^2 dxdy \ , \tag{2.3}$$

wobei C eine beliebige Kurve ist. Somit ist klar, dass C_0 gerade der Minimierer des sogenannten Fitting Terms (2.3) ist:

$$inf_C(F_1(C) + F_2(C)) \approx 0 \approx F_1(C_0) + F_2(C_0) \tag{2.4}$$

Fügen wir diesem Fitting-Term (2.3) zwei weitere regularisierende Eigenschaften der Kurve C hinzu, erhalten wir für unser Energiefunktional:

$$\begin{aligned} J_{CV}(c_1, c_2, C) &= \mu \cdot Length(C) + \nu \cdot Area(inside(C)) \\ &+ \lambda_1 \int_{inside(C)} |u_0(x,y) - c_1|^2 dxdy \\ &+ \lambda_2 \int_{outside(C)} |u_0(x,y) - c_2|^2 dxdy, \end{aligned} \tag{2.5}$$

wobei $\mu \geq 0$, $\nu \geq 0$, λ_1,$\lambda_2 > 0$ feste Parameter sind. Im Normalfall wird $\lambda_1 = \lambda_2 = 1$ und $\nu = 0$ gesetzt. Wir betrachen also das Minimierungsproblem:

$$\inf_{c_1, c_2, C} J_{CV}(c_1, c_2, C) \tag{2.6}$$

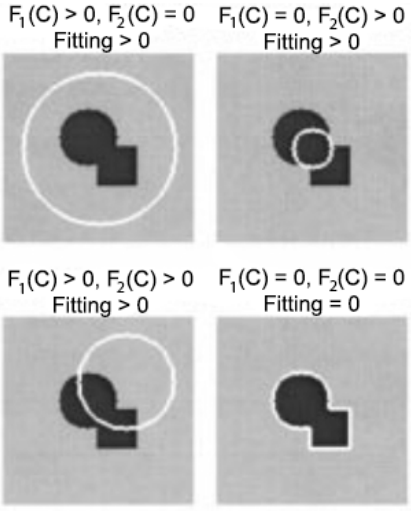

Abbildung 2.1.: Veranschaulichung des Fitting Terms für alle Fälle der beiden Regionen. [Quelle [5]]

Theorem 2.1.2 (Existenz eines Minimieres)**.** *Für unser Minimierungsproblem* (2.6) *existiert ein Minimierer.*

Beweis. Wir überführen unser Energiefunktional J_{CV} in ein vereinfachtes Mumford-Shah Funktional, für welches die Existenz für den zweidimensionalen Fall zum Beispiel in [7] gezeigt wurde. Das Mumford-Shah Funktional ist gegeben durch

$$\begin{aligned} J_{MS}(u, C) &= \mu \cdot Length(C) \\ &+ \lambda \int_{\Omega} |u_0(x,y) - u(x,y)|^2 dxdy \\ &+ \int_{\Omega \setminus C} |\nabla u(x,y)|^2 dxdy. \end{aligned} \tag{2.7}$$

Die Energie wird minimiert, wenn wir also möglichst glatte Regionen R_i des Bildes mit scharfen Kanten C haben. Reduzieren wir J_{MS} auf stückweise konstante Funktionen u_i, zum Beispiel $u_i = c_i$ auf jeder Region R_i und setzen $c_i = average(u_0)$ erhalten wir das **minimal partition problem**.
Wählen wir in unserem Modell $\nu = 0$ und $\lambda_1 = \lambda_2 = \lambda$ erhalten wir genau diese spezielle Form des Mumford-Shah-Modells und haben damit die Existenz eines Minimieres gezeigt. □

2.1.2. Formoptimierung

Im Vergleich zur Bildsegmentierung beschreibt die Formoptimierung jene Art von Optimierungsproblemen, bei welchen die zu optimierende Variable nicht etwa eine Menge oder Funktion ist, sondern die Form und Struktur eines geometrischen Objekts. Diese Optimierungsaufgabe spielt angefangen bei der Oberflächenstruktur von Flugzeugtragflächen über die Konstruktion eines Kranarmes bis hin zur optimalen Gestaltung von Muschelschalen oder Brückenbögen eine Rolle. Als mathematische Grundlage dient die *Shape Sensitivity Analysis* mit der wir uns in Kapitel 3 ausführlicher beschäftigen wollen.

2.2. Level-Set Funktionen

Die Level-Set Funktion ist ein numerisches Werkzeug, um geometrische Objekte oder deren Bewegung approximativ zu verfolgen. Dieses Verfahren eignet sich vor allem, wenn das Modell stark auf dem Rand der Struktur beruht und ist somit besonders für die Erkennung von Objekten oder für die Formoptimerung geeignet.

Zu Beginn nehmen wir als Beispiel ein Bild $D \subset \mathbb{R}^2$. In diesem Bild befindet sich ein Objekt Ω, welches durch eine Kurve C umrandet ist. Die Level-Set Funktion für D ist dann implizit gegeben durch eine Lipschitz-Funktion $\phi : D \to \mathbb{R}$:

$$\begin{cases} C & = \{(x,y) \in D : \phi(x,y) = 0\} \\ inside(C) & = \{(x,y) \in D : \phi(x,y) > 0\} \\ outside(C) & = \{(x,y) \in D : \phi(x,y) < 0\} \end{cases} \tag{2.8}$$

Die Kurve C wird also mit der Null-Niveaumenge von ϕ identifiziert. Für das Innere des Objekts ist $\phi > 0$ und für das Äußere gilt $\phi < 0$.

Ein Beispiel für die Level-Set Funktion ist die signierte Distanzfunktion.
Definition 2.2.1 (Signierte Distanzfunktion)**.** *Sei* (Ω, d) *ein metrischer Raum, dann ist die signierte Distanzfunktion definiert durch:*

$$f(x) = \begin{cases} d(x, \Omega^c) & wenn\ x \in \Omega\ , \\ -d(x, \Omega) & wenn\ x \in \Omega^c, \end{cases}$$

wobei $d(x, \Omega) = \inf_{y \in \Omega} d(x, y)$.

Bemerkung. Handelt es sich bei Ω um eine Teilmenge des euklidischen Raums $\mathbb{R}^n$, so ist f fast überall differenzierbar und erfüllt die Eigenschaft:

$$|\nabla f| = 1$$

Die signierte Distanzfunktion erfüllt also per Definition die Level-Set Eigenschaften aus (2.8).

Die Level-Set Funktionen werden nun mit einer Evolutionsgleichung des jeweiligen Energiefunktionals entwickelt. Das heißt, wir entwickeln unsere Level-Set Funktion $\phi(t, x, y)$ in Normalenrichtung mit Geschwindigkeit F in einer fiktiven Zeit t und erhalten:

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| F \ , \ \phi(0, x, y) = \phi_0(x, y) \ ,$$

wobei die Menge $\{(x, y) | \phi_0(x, y) = 0\}$ eine initiale Kontur beschreibt. Die zeitliche Veränderung von ϕ ist also gerade gegeben durch die Bewegung von ϕ um F.

Ein Spezialfall dieser Evolutionsgleichung ist die Bewegung mit mittlerer Krümmung. Hierbei ist $F = div(\frac{\nabla \phi(x,y)}{|\nabla \phi(x,y)|})$ die Krümmung der Niveaukurve von ϕ am Punkt (x, y). Damit erhalten wir:

$$\begin{cases} \frac{\partial \phi}{\partial t} = |\nabla \phi| div(\frac{\nabla \phi}{|\nabla \phi|}), t \in (0, \infty), (x, y) \in \mathbb{R}^2 \\ \phi(0, x, y) = \phi_0(x, y), (x, y) \in \mathbb{R}^2 \end{cases}$$

Wir wollen nun unser Beispiel mit dem Modell von Chan und Vese (2.5) in eine Level Set-Formulierung umschreiben. Wir nutzen als Indikatorfunktionen für ϕ die Heaviside Funktion $H : \mathbb{R} \to \{0, 1\}$ und das Diracmaß $\delta_0 : \mathbb{R} \to \{0, 1\}$, welche definiert sind durch:

$$H(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \quad \delta_0(x) = \frac{d}{dx} H(x) \tag{2.9}$$

Mithilfe dieser Funktionen können wir unsere Formulierung auf das gesamte Gebiet

erweitern und erhalten für die einzelnen Terme in (2.5):

$$\begin{aligned}
Length\{\phi = 0\} &= \int_\Omega |\nabla H(\phi(x,y))| dxdy = \int_\Omega \delta_0(\phi(x,y))|\nabla\phi(x,y)|dxdy, \\
Area\{\phi \geq 0\} &= \int_\Omega H(\phi(x,y))dxdy, \\
\int_{\phi>0} |u_0(x,y) - c_1|^2 dxdy &= \int_\Omega |u_0(x,y) - c_1|^2 H(\phi(x,y))dxdy, \\
\int_{\phi<0} |u_0(x,y) - c_2|^2 dxdy &= \int_\Omega |u_0(x,y) - c_2|^2 (1 - H(\phi(x,y)))dxdy
\end{aligned}$$

Somit erhalten wir für die Energie $F(c_1, c_2, \phi)$:

$$\begin{aligned}
F(c_1, c_2, \phi) = \quad & \mu \int_\Omega \delta_0(\phi(x,y)) \, |\nabla\phi(x,y)|dxdy \\
& + \nu \int_\Omega H(\phi(x,y))dxdy \\
& + \lambda_1 \int_\Omega |u_0(x,y) - c_1|^2 H(\phi(x,y))dxdy \\
& + \lambda_2 \int_\Omega |u_0(x,y) - c_2|^2 (1 - H(\phi(x,y)))dxdy
\end{aligned}$$

Halten wir ϕ fest und minimieren die Energie $F(c_1, c_2, \phi)$ im Bezug auf die Konstanten c_1 bzw. c_2, erhalten wir die Euler-Lagrange-Gleichung für c_1 und c_2 :

$$c_1(\phi) = \frac{\int_\Omega u_0(x,y) H(\phi(x,y))dxdy}{\int_\Omega H(\phi(x,y))dxdy} \tag{2.10}$$

$$c_2(\phi) = \frac{\int_\Omega u_0(x,y)(1 - H(\phi(x,y)))dxdy}{\int_\Omega (1 - H(\phi(x,y)))dxdy}\,, \tag{2.11}$$

für $\int_\Omega H(\phi(x,y))dxdy > 0$ bzw. $\int_\Omega (1 - H(\phi(x,y)))dxdy > 0$. In unserem Fall sind c_1 und c_2 nicht beschränkt und somit gilt tatsächlich:

$$\begin{cases} c_1(\phi) = average(u_0) \text{ für } \{\phi \geq 0\} \\ c_2(\phi) = average(u_0) \text{ für } \{\phi < 0\} \end{cases}$$

Der finale Schritt ist nun die Heaviside Funktion H und das Diracmaß δ_0 durch eine regularisierte Variante H_ϵ bzw. δ_ϵ zu ersetzen, sodass für $\epsilon \to 0$ gilt $H_\epsilon \to H$ und $\delta_\epsilon \to \delta_0$. Dieser Schritt dient zur Glättung des vorhandenen Sprungs an der Kante. Auf die Wahl der Regularisierung für die Heaviside Funktion und des Diracmaßes wird im

im Abschnitt 5.1.3 dieser Arbeit diskutiert.

Theorem 2.2.2 (Level-Set-Formulierung für Chan und Vese Funktional)**.** *Die Level-Set-Formulierung des Modells von Chan und Vese ist gegeben durch:*

$$\begin{aligned}
\frac{\partial \phi}{\partial t} &= \delta_\epsilon(\phi)[\mu div(\frac{\nabla \phi}{|\nabla \phi|}) - \nu - \lambda_1(u_0 - c_1)^2 \\
&+ \lambda_2(u_0 - c_2)^2] \ in\ (0,\infty) \times \Omega \\
\phi(0,x,y) &= \phi_0(x,y)\ in\ \Omega \\
\frac{\delta_\epsilon(\phi)}{|\nabla \phi|}\frac{\partial \phi}{\partial \overrightarrow{n}} &= 0\ auf\ \partial\Omega
\end{aligned}$$

Beweis. Ein ausführlicher Beweis wie aus dem Energiefunktional die Level-Set-Formulierung folgt, erfolgt mit der Gâteaux-Ableitung (siehe Definition 3.1.1) und der Euler-Lagrange-Gleichung. Der Beweis kann in [12] eingesehen werden. □

Die Idee der Level-Set-Funktion kann auch genutzt werden, um Formen zu bewegen und zu optimieren. Hierbei muss aber vor allem auf die Ableitung des Randes acht gegeben werden, da nicht sichergestellt werden kann, dass diese existieren. Diese Diskussion wird in Kapitel 3 geführt. Zuerst wollen wir uns einige grundlegende Minimierungsverfahren in Erinnerung rufen, die zur Lösung des Chan-Vese Energiefunktionals dienen können.

2.3. Gradientenabstiegsverfahren

Die Idee des Gradientenabstiegsverfahren besteht darin, eine Funktion dadurch zu minimieren, sie in Richtung des steilsten Abstiegs, also des negativen Gradienten, entlang zu laufen. Das allgemeine Gradientenabstiegsverfahren ist wie folgt definiert:

Algorithmus 2.3.1 (allgemeines Gradientenabstiegsverfahren)**.** *Sei $f \in C^1$ die zu minimierende Funktion. Sei x_0 der Anfangswert. Dann wiederhole folgende Schritte bis ein Abbruchkriterium erfüllt ist:*

- *Wähle als Abstiegsrichtung $d = -\nabla f(x_n)$.*
- *Führe Schritt mit Schrittweite τ durch: $x_{n+1} = x_n + \tau d$*

Als eines der einfachsten weiterführenden Gradientenabstiegsverfahren gilt das *Conjugate Gradient Verfahren*, kurz cg-Verfahren.

Die Idee des cg-Verfahrens besteht darin, dass für eine symmetrische und positiv defi-

nite Matrix A das Minimieren der quadratischen Form

$$f(x) := \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

äquivalent zum Lösen der Gleichung $Ax = b$ ist. Hierbei bezeichnet $\langle\cdot,\cdot\rangle$ das Standardskalarprodukt. Der Gradient von f an der Stelle x_k ist gerade $\nabla f\big|_{x_k} = Ax_k - b := -r_k$. Die Änderung zum normalen Gradientenverfahren besteht darin, die Funktion f anstelle in Richtung des Residuums r_k in eine andere Richtung d_k über einen Unterraum zu minimieren. Die Richtungen d_k sind dabei alle A-konjugiert, das heißt, es gilt

$$\langle Ad_i, d_j\rangle = 0 \ \forall i \neq j.$$

Algorithmus 2.3.2 (cg-Verfahren)**.** *Das cg-Verfahren zum Lösen von $Ax = b$ für eine symmetrische, positiv definite Matrix $A \in \mathbb{R}^{n\times n}$ lautet: Wähle $x_0 \in \mathbb{R}^n$ beliebig und setze*

$$\begin{aligned} r_0 &= b - Ax_0 \\ d_0 &= r_0. \end{aligned}$$

Führe folgende Schritte solange durch bis ein Abbruchkriterium erfüllt ist, z.B. $\|r_{k+1}\| < \epsilon$.

- *Speichere Produkt zur effizienten Berechnung $z = Ad_k$.*
- *Aktualisiere x_k mit Richtung d_k und Gradient bzw. Residuum*

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{d_k^T z}, \\ x_{k+1} &= x_k + \alpha_k d_k, \\ r_{k+1} &= r_k - \alpha_k z. \end{aligned}$$

- *Korrigiere Suchrichtung d_{k+1}*

$$\begin{aligned} \beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \\ d_{k+1} &= r_{k+1} + \beta_k d_k. \end{aligned}$$

Bemerkung. In Algorithmus 2.3.2 gibt es verschiedene Möglichkeiten für die Wahl von β. Die hier angegeben ist die von Fletcher-Reeves. Eine andere oft genutzte Möglichkeit

ist die von Polak-Ribière. Diese lautet $\beta_k = \frac{(r_{k+1}^T)(r_{k+1}-r_k)}{r_k^T r_k}$.

Bevor wir uns von dem linearen Ansatz lösen, benötigen wir ein Verfahren zur Bestimmung der idealen Schrittweite α. Hier wollen wir das Backtracking-Linesearch mit Armijo-Goldstein Bedingung nutzen. Diese Bedingung ist definiert durch

$$f(x_0 + \alpha r_0) \leq f(x_0) + \alpha c d_0^T \nabla f(x) \tag{2.12}$$

für ein $c \in (0, 1)$. Damit wird sichergestellt, dass der Schritt in Abstiegsrichtung nicht zu weit gemacht wird und so zum Beispiel das Minimum übergangen wird. Eine solche Schrittweite lässt sich durch das Backtracking-Linesearch finden, welches wie folgt definiert ist.

Algorithmus 2.3.3 (Backtracking-Linesearch)**.** *Sei d die Abstiegsrichtung und setze initiale Schrittweite $\alpha_0 > 0$ und Parameter $\tau \in (0, 1)$ und $c \in (0, 1)$. Setze Iterationszähler $n = 0$. Wiederhole folgendene Schritte solang bis* (2.12) *erfüllt ist.*

- $n = n + 1$
- $\alpha_n = \tau \alpha_{n-1}$

Das heißt, die Schrittweite wird solange um den Faktor τ verringert, zum Beispiel halbiert wenn $\tau = \frac{1}{2}$, bis die Armijo-Goldstein Bedingung erfüllt ist.

Theorem 2.3.4 (Terminiertheit des Backtracking-Linesearch Verfahrens)**.** *Sei $f : \mathbb{R}^n \to \mathbb{R}$ und ∇f stetig. Sei d echte Abstiegsrichtungen, d.h. es gilt $\nabla f(x) d^T < 0$. Zudem existiert $C_1 > 0$ und $C_2 > 0$, sodass*

$$C_1 \geq ||d|| \geq C_2 ||\nabla f(x)||^q$$

für ein $q > 0$ gilt. Dann terminiert das Backtracking-Linesearch Verfahren in endlichen Schritten und α_n erfüllt die Armijo-Goldstein Bedingung (2.12)*.*

Beweis. Dieser Beweis ist Quelle [3] entnommen. Nehmen wir an der Algorithmus terminiert nicht in endlichen Schritten, dann wir die Armijo-Goldstein Bedingung nie erfüllt und es gilt

$$\frac{f(x + \alpha_0 \tau^n d^T) - f(x)}{\alpha_0 \tau^n} > c \nabla f(x) d^T.$$

Da $\tau^n \to 0$ für $n \to \infty$, da $\tau < 1$, gilt für die Ungleichung

$$\nabla f(x) d^T > c \nabla f(x) d^T,$$

was ein Widerspruch zu $c \in (0,1)$ und $\nabla f(x)d^T \neq 0$ ist. □

Bemerkung. Die Konstanten c und τ werden im Allgemeinen als $\frac{1}{2}$ gewählt, so wie von Armijo im Paper 1966 veröffentlicht [2].

Wir wollen uns nun von dem linearen Ansatz lösen und nicht-lineare Verfahren betrachten. Eine natürliche Fortsetzung ist die Einführung des *nicht-linearen cg-Verfahrens.*

Sei f dazu eine nicht-lineare Funktion. Im Vergleich zum linearen cg-Verfahren, welches die Lösung der linearen Gleichung $Ax = b$ bzw. $A^TAx = A^Tb$ sucht, findet das nicht-lineare cg-Verfahren ein lokales Minimum basierend auf dem Gradienten ∇f. Wir ersetzen daher, dass im cg-Verfahren auftretende Residuum r_k durch den Gradienten $\nabla f(x_k)$. Da wir die Schrittweite α_k im nichtlinearen Fall nicht mehr exakt bestimmen können, verwenden wir das Backtracking-Linesearch Verfahren in Richtung d_k zur Bestimmung von α_k.

Algorithmus 2.3.5 (nicht-lineares cg-Verfahren)**.** *Sei x_0 ein beliebiger Startwert. Gegeben eine zu minimierende Funktion $f(x)$ gibt der Gradient $\nabla_x f$ die Richtung des steilsten Anstiegs an, sodass man zum Minimieren mit dem negativen Gradient als Startrichtung beginnen kann*

$$r_0 = -\nabla f(x_0).$$

Nun sucht man die ideale Schrittweite mithilfe von Backtracking-Linesearch in Richtung r_0, sodass die Armijo-Goldstein Bedingung (2.12) *erfüllt wird und macht anschließend den ersten initialen Schritt:*

$$x_1 := x_0 + \alpha_0 r_0$$

Nach dieser ersten Iteration wiederholt man folgende Schritte, wobei d die konjugierte Richtung ist und $d_0 = r_0$ gilt.

- *Berechne den steilsten Abstieg*

$$r_n = -\nabla_x f(x_n).$$

- *Berechne β_n wie in Algorhitmus 2.3.2 mit Fletcher-Reeves oder Polak-Ribière.*
- *Aktualisiere konjugierte Suchrichtung d_n*

$$d_n = r_n + \beta_n d_{n-1}.$$

- *Führe Backtracking-Linesearch in Richtung d_n durch, sodass α_n die Armijo-Goldstein Bedingung erfüllt.*
- *Führe Schritt durch*

$$x_{n+1} = x_n + \alpha_n d_n.$$

Diese Schritte werden wiederholt bis ein gegebenes Konvergenzkriterium erfüllt ist.

Eine andere Möglichkeit nicht lineare Funktionen zu optimieren ist das Newton-Verfahren. Jedoch benötigt dieses Verfahren die Jacobi- oder Hessematrix des Problems, die oft schwer zu berechnen oder nicht verfügbar ist. Um dieses Problem zu umgehen, wollen wir als zweites nicht-lineares Optimierungsverfahren das *Quasi-Newton Verfahren* nutzen. Das Quasi-Newton Verfahren approximiert dabei die Jacobi- bzw. Hessematrix in jedem Schritt aus den vorhandenen Parametern und erspart so die aufwendige Berechnung. Hierbei wird das Minimierungsproblem

$$B_{n+1} = \arg\min_B ||B - B_k||$$

gelöst.

Algorithmus 2.3.6 (Quasi-Newton Verfahren)**.** *Wähle Startvektor x_0 und initialisiere die Approximation der Hessematrix B_n durch $B_0 = Id$. Wiederhole nun folgende Schritte bis ein gegebene Konvergenzkriterum erfüllt ist.*

- *Erhalte Suchrichtung d_n durch Lösen von*

$$B_n d_n = -\nabla f(x_n).$$

- *Führe Backtracking Linesearch in Richtung d_n durch, sodass α_n die Armijo-Goldstein Bedingung* (2.12) *erfüllt.*
- *Führe Schritt durch*

$$x_{n+1} = x_n + \alpha_n d_n.$$

- *Berechne neue Approximation der Hessematrix*

$$\begin{aligned} s_k &= \alpha_k d_k, \\ y_n &= \nabla f(x_{n+1}) - \nabla f(x_n), \\ B_{n+1} &= B_n + \frac{y_n y_n^T}{y_n^T s_n} - \frac{B_n s_n s_n^T B_n}{s_n^T B_n s_n}. \end{aligned}$$

Sollte es während der Iteration dazu kommen, dass d_k keine „echte“ Abstiegsrichtung ist, also das $-\nabla f(x_n) \cdot d_k > 0$ gilt, initialisieren wir den Algorithmus neu. Das heißt, wir setzen wieder $B = Id$ und iterieren weiter.

Bemerkung. Es handelt sich hierbei um den *Broyden-Fletcher-Goldfarb-Shanno Algorithmus*, kurz BFGS Algorithmus. Anstatt der Berechnung von B und des Lösens des Gleichungsystems $B_n r_n = -\nabla f(x_n)$, kann auch direkt B_{n+1}^{-1} bestimmt werden. Es ist gegeben durch:

$$B_{n+1}^{-1} = B_n^{-1} + \frac{(s_n^T y_n + y_n^T B_n^{-1} y_n)(s_n s_n^T)}{(s_n^T y_n)^2} - \frac{B_n^{-1} y_n s_n^T + s_n y_n^T B_n^{-1}}{s_n^T y_n}$$

Um einen kurzen Einblick über die Konvergenzordnung der Verfahren zu erhalten, wollen wir diese mit der Rosenbrock-Funktion testen. Die Rosenbrock-Funktion ist gegeben durch:

$$f(x, y) = 100 * (y - x^2)^2 + (1 - x)^2$$

Es handelt sich hierbei um eine nicht-konvexe, offensichtlich nicht-lineare Gleichung, die ihr Minimum bei $(1, 1)$ annimmt.

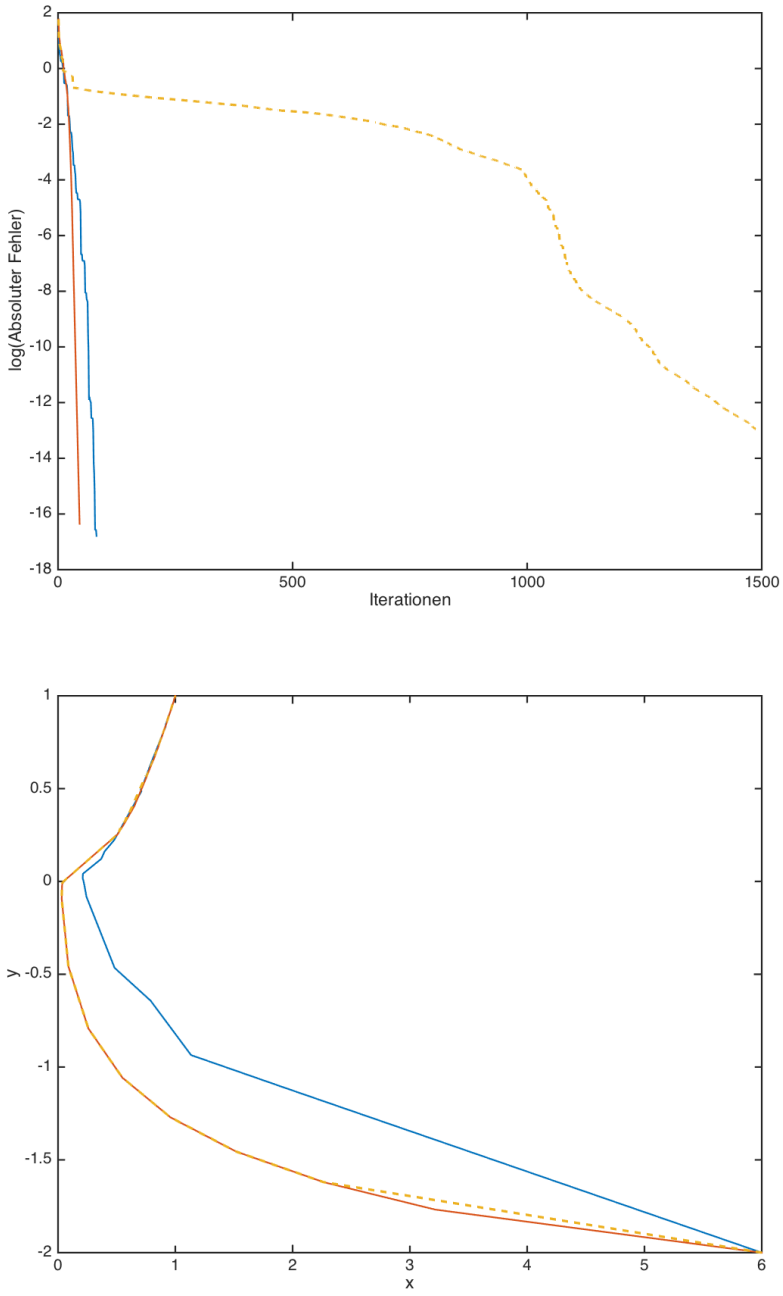

Abbildung 2.2.: Oben: Logarithmus des absolute Fehler im Vergleich zur Iteration. Unten: Pfad der Iterationen. Startwert unten rechts bei $(6, 2)$. In beiden Fällen ist blau das nicht-lineare cg-Verfahren und orange das Quasi-Netwon Verfahren und zum Vergleich gestrichelt der normale Gradientenabstieg.

Hierbei ist zu beobachten, dass der erste Schritt des Gradientenabstiegs- und Quasi-Newton-Verfahren gleich ist, da die approximierte Hessematrix im Quasi-Newton Verfahren noch die Identität ist und somit ein normaler Abstiegsschritt gemacht wird. In Abbildung 2.3 ist zudem zu sehen, dass zu Beginn das nicht-lineare cg-Verfahren schneller konvergiert. Es wird allerdings nach wenigen Schritten vom Quasi-Newton Verfahren überholt. Dies kann jedoch an der genutzten Funktion oder dem Startwert liegen.

Beide Verfahren konvergieren superlinear, wie in Abbildung 2.3 gut zu sehen ist. Superlineare Konvergenz heißt, dass für die Verfahren gilt $\lim_{n\to\infty} \frac{\|x_{n+1}-x^*\|}{\|x_n-x^*\|} \to 0$, wobei x^* das gesuchte Minimum ist. Für die Beweise zur superlinearen Konvergenz und weitere Diskussionen sei auf [13] verwiesen. In Abbildung 2.3 ist gut zu sehen, wie die Konvergenzen der beiden Verfahren zwischen der linearen und quadratischen Konvergenz (gestrichelt) liegen.

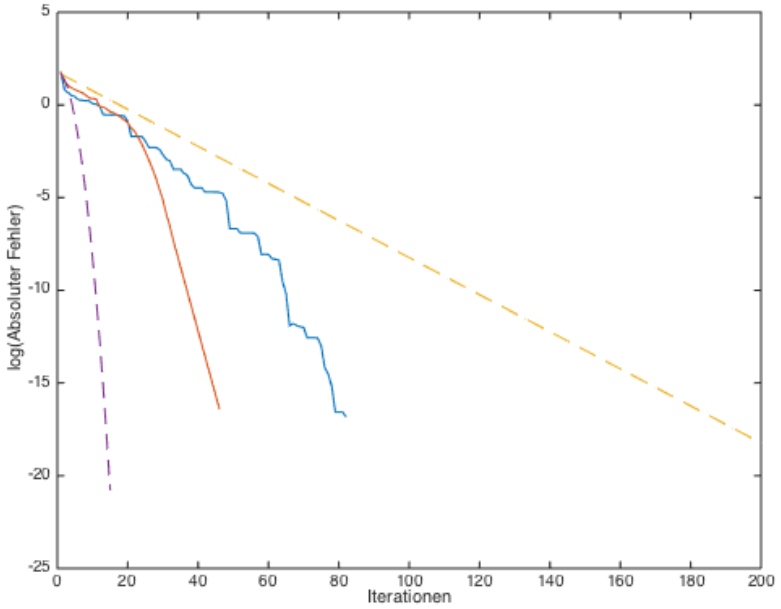

Abbildung 2.3.: Die Konvergenzen der nicht-linearen Verfahren im Detail. Auch hier ist blau das nicht-lineare cg-Verfahren und orange das Quasi-Netwon Verfahren. Gestrichelt ist hier beispielhaft lineare bzw. quadratische Konvergenz.

3. Shape Sensitivity Analysis

Die theoretische Grundlage zur Anwendung der hier vorgestellten Optimierungsverfahren auf Formfunktionen ist die *Shape Sensitivity Analysis*, die im folgenden Kapitel kurz vorgestellt wird. Hierzu werden grundlegende Resultate und Aussagen vorgestellt, die für das weitere Vorgehen in dieser Arbeit von Nöten sind. Für eine ausführlichere Darstellung des Themas sei auf die Literatur von Sokolowski und Zolesio [17] oder von Delfour und Zolesio [6] verwiesen. Zudem basiert dieses Kapitel auf der Diplomarbeit von Martin Pach [14].

In der Shape Sensitivity Analysis werden auf Basis von Transformationen des Gebietes Ω mithilfe der „Velocity-Methode“ sogennante Formableitungen eingeführt. Diese erlauben uns die Transformation als Abstiegsrichtung für ein Gradientenabstiegsverfahren, siehe 2.3, zu identifizieren. Das heißt, wir können eine Transformation $T(\Omega) = \tilde{\Omega} \subset D$ finden, sodass für ein Energiefunktional J gilt:

$$J(\tilde{\Omega}) < J(\Omega)$$

Somit bietet die Shape Sensitivity Analysis ein einfaches Werkzeug, um Gradientenabstiegsverfahren auch in der Formoptimierung zu nutzen.

3.1. Formfunktionen und Formableitungen

Im folgenden Abschnitt wollen wir uns einige grundlegende Definitionen und Folgerungen zur Semi-Differenzierbarkeit basierend auf den Gâteaux- und Hadamard-Semiableitungen in topologischen Vektorräumen in Erinnerung rufen. Für eine weitere Diskussion sei auf [16] verwiesen.

Definition 3.1.1 (Semiableitungen)**.** *Sei E ein topologischer Vektorraum, $x_0 \in E$*

beliebig, U Umgebung um x_0 und $f : U \to E$ eine reellwertige Funktion.

- *Die Gâteaux-Semiableitung im Punkt $x \in U$ in Richtung $v \in E$ existiert, falls folgender Limes existiert:*

$$\lim_{\epsilon \to 0} \frac{f(x + \epsilon v) - f(x)}{\epsilon} \tag{3.1}$$

  Falls dieser Grenzwert existiert, bezeichnen wir ihn mit $df(x; v)$.

- *Die Hadamard-Semiableitung im Punkt $x \in U$ in Richtung $v \in E$ existiert, falls folgender Limes existiert:*

$$\lim_{\epsilon \to 0, w \to v} \frac{f(x + \epsilon w) - f(x)}{\epsilon} \tag{3.2}$$

  Falls dieser Grenzwert existiert, bezeichnen wir ihn mit $d_H f(x; v)$.

Falls $df(x; v)$ und $d_H(x; v)$ existieren, gilt offensichtlich $df(x; v) = d_H(x; v)$.

Für die Formfunktionen wollen wir nun ähnlich der Semiableitungen Formableitungen definieren. Zunächst sei jedoch die Definition der Formfunktionen vorangestellt.

Definition 3.1.2 (Formfunktionen)**.** *Sei $\emptyset \neq D \subset \mathbb{R}^n$ Teilmenge und bezeichne $\mathcal{K}(\Omega) = \{\Omega : \Omega \subset D\}$ die Potenzmenge von D. Eine Formfunktion ist eine Abbildung*

$$J : \mathcal{A} \to E$$

von einer Familie $\mathcal{A}$ zulässiger Gebiete aus $\mathcal{K}(\Omega)$ in einen topologischen Raum E, sodass für jeden Homöomorphismus T von $\overline{\Omega}$ mit der Eigenschaft $T(\Omega) = \Omega$ für alle $\Omega \in \mathcal{A}$ gilt

$$J(\Omega) = J(T(\Omega)).$$

Bemerkung. Der Definitionsbereich D kann eine physikalische oder mechanische Einschränkung an die Form beschreiben oder sogar eine Untermannigfaltigkeit des $\mathbb{R}^n$ sein. Im Allgemeinen kann D so groß und der Rand δD so glatt wie nötig gewählt werden. Im unbeschränkten Fall gilt $D = \mathbb{R}^n$.

Obwohl die Menge $\mathcal{A}$ im Allgemeinen keine Vektorraumstruktur besitzt, ist es möglich die Grenzwerte von Differenzenquotienten in eine unendlich-dimensionale Richtung um ein Element $\Omega \in \mathcal{A}$ zu betrachten. Sei dazu $V \in C^1(\mathbb{R}^d, \mathbb{R}^d)$ ein Vektorfeld. Dann definieren wir folgende Abbildungen:

$$T(s, x) := x + sV(x) \ \forall x \in \mathbb{R}^d \ , s \geq 0$$

Für s hinreichend klein ist die Abbildung

$$T(s,\cdot) : \mathbb{R} \to \mathbb{R}^d \, , x \mapsto x + sV(x) \; \forall x \in \mathbb{R}^d \, , s \geq 0 \tag{3.3}$$

ein lokaler C^1-Diffeomorphismus, denn es gilt:

$$(\mathbb{I} + sV) \in C^1(\mathbb{R}^d, \mathbb{R}^d) \text{ und } \lim_{s \to 0} det(\mathbb{I} + sV) = 1$$

$det(A)$ steht hierbei für die Determinante von A. Das heißt, für jedes beschränkte Gebiet Ω existiert ein $s_0 > 0$, sodass $T_{s_0}(\cdot)\ \Omega$ diffeomorph auf $\Omega_{s_0} = T_{s_0}(\Omega)$ abbildet. $T_s(\Omega)$ ist dabei folgendermaßen definiert

$$T_s(\Omega) := \{T_s(x) : \forall x \in \Omega\},$$

wobei $T_s(x) := T(s,x) \forall x \in \Omega$ gilt.
Diese Methode wird als „Störung der Identität“ bezeichnet. Mit dieser Definition ist es nun möglich die Semiableitung $dJ(\Omega; V)$ als Grenzwert, falls dieser existiert, eines Differenzenquotientens zu identifizieren. Dazu konstruieren wir uns eine Abbildung $T_s : \Omega \to \Omega_s$ wie in (3.3). Wir definieren nun die Semiableitung $dJ(\Omega; V)$ analog zu (3.1) und erhalten:

$$\lim_{s \to 0} \frac{J(T_s(\Omega)) - J(\Omega)}{s} =: dJ(\Omega; V) \tag{3.4}$$

Betrachtet man das Vektorfeld V aus einem topologischen Vektorraum, so lässt sich die Stetigkeit und Linearität der Abbildung

$$V \mapsto dJ(\Omega; V)$$

untersuchen und die Resultate sind analog zu denen in Banachräumen. Dieser Ansatz hat jedoch Einschränkungen, da Variationen von Ω nur entlang der Linie von $\mathbb{I} + V$ nach $\mathbb{I}$ auftreten. Sie ist daher nicht anwendbar, wenn das Definitionsgebiet D eine Untermannigfaltigkeit des $\mathbb{R}^d$ mit nichtverschwindender Krümmung ist.

So wie die Hadamard-Semiableitung eine Verallgemeinerung der Gâteaux-Semiableitung darstellt, wollen wir nun eine allgemeinere Methode zur Variation von Gebieten einführen.

Sei dazu $V \in C^0(\mathbb{R}^d, \mathbb{R}^d)$ ein Vektorfeld. Wir betrachten den von V erzeugten Fluss,

also die Abbildung $x : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d$ mit folgenden Eigenschaften:

$$\begin{aligned} \dot{x}(s, \bar{x}) = &\quad V(x(s, \bar{x})) \quad \forall s \geq 0 \\ x(0, \bar{x}) = &\quad \bar{x} \qquad\quad \forall \bar{x} \in \mathbb{R}^d \end{aligned}$$

Damit können wir nun wieder eine Abbildung $T_s : \mathbb{R}^d \to \mathbb{R}^d$ definieren, mit der wir das Gebiet Ω variieren:

$$T_s(\bar{x}) := x(s, \bar{x}) \ \forall s \geq 0 \ \forall \bar{x} \in \mathbb{R}^d$$

Die Geschwindigkeit des Punktes $X(s, \bar{x})$ zum Zeitpunkt s hängt dabei nur von $V(x(s, \bar{x}))$ und nicht von $V(\bar{x})$ ab, welches die Idee hinter der Velocity-Methode ist.

Erweitern wir diese Konstruktion auf zeitabhängige Vektorfelder, erhalten wir folgendes. Sei dazu $V \in C^0(\mathbb{R}^+ \times \mathbb{R}^d, \mathbb{R}^d)$. Dann betrachten wir wieder den von V erzeugten Fluss:

$$\begin{aligned} \dot{x}(s, \bar{x}) &= V(s, x(s, \bar{x})) \quad \forall s \geq 0 \\ x(0, \bar{x}) &= \bar{x} \qquad\qquad\quad \forall \bar{x} \in \mathbb{R}^d \end{aligned} \tag{3.5}$$

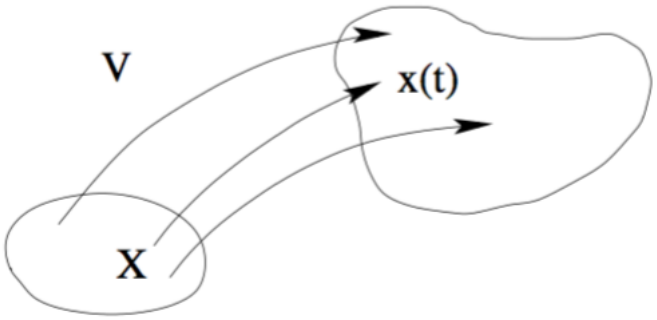

Abbildung 3.1.: Transport von Ω durch das Geschwindigkeitsfeld V [Quelle [14]]

Dann ist die Abbildung $T : \mathbb{R}_0^+ \times \mathbb{R}^d \to \mathbb{R}^d$ definiert durch:

$$T(s, \bar{x}) := x(s, \bar{x}) \ \forall s \geq 0 \ \forall \bar{x} \in \mathbb{R}^d \tag{3.6}$$

Die Funktion x aus (3.5) ist offensichtlich die Lösung einer gewöhnlichen Differentialgleichung. Es ist leicht zu sehen, dass die Lösung stetig von den Anfangsdaten abhängt und somit ist auch T stetig. Wie schon in Gleichung (3.2) wollen wir $d_H J(\Omega; V)$ einer

Shapefunktion J als Grenzwert eines Differenzenquotienten defineren:

$$\lim_{s\to 0} \frac{J(T(s,\Omega)) - J(\Omega)}{s} =: d_H J(\Omega; V) \ ,$$

wobei $T(s,\Omega) = \{T(s,\bar{x}) : \bar{x} \in \Omega\}$ ist.

Nun wollen wir zeigen, dass die *Velocity-Methode* eine Verallgemeinerung der „Störung der Identität“ ist, so wie die Hadamard-Semiableitung eine Verallgemeinerung der Gâteaux-Semiableitung ist.

Wir wollen dazu ein zeitabhängiges Vektorfeld $V : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d$ einem konstanten Vektorfeld $\tilde{V} : \mathbb{R}^d \to \mathbb{R}^d$ zuordnen, sodass die Transformation des zeitabhängigen Vektorfeldes mit der *Velocity-Methode* übereinstimmt mit der Transformation des konstanten Vektorfeldes mithilfe „Störung der Identität“. Betrachten wir dazu die Transformation

$$T(s,\bar{x}) = x(s) = \bar{x} + s\tilde{V}(\bar{x}) \quad \forall \bar{x} \in \mathbb{R}^d \ , \ s \geq 0$$

Das Vektorfeld V muss jetzt so gewählt werden, dass die Funktion x folgende Differentialgleichung löst:

$$\begin{aligned} \dot{x} &= V(s, x(s)) \quad \forall s \geq 0 \\ x(0) &= \bar{x} \end{aligned} \tag{3.7}$$

Da es sich um eine einfache Transportgleichung handelt, sieht die Lösung von (3.7) wie folgt aus :

$$V(s,\bar{x}) := \tilde{V}(T^{-1}(s,\bar{x})) \quad \forall \bar{x} \in \mathbb{R}^d \ , \ s \geq 0$$

Ist $\tilde{V}$ genügend glatt und s genügend klein, erhalten wir mit dieser Lösung aus (3.7):

$$\begin{aligned} \frac{\partial V}{\partial s}(s,x)\big|_{s=0} &= -[D\tilde{V}(x)]\tilde{V}(x) \quad \forall x \in \mathbb{R}^d \\ V(0,x) &= \tilde{V}(x) \qquad\qquad\quad \forall x \in \mathbb{R}^d \end{aligned} \tag{3.8}$$

Dabei ist $D\tilde{V}(x)$ die Jacobimatrix von $\tilde{V}$ im Punkt x. Aus Gleichung (3.8) erkennt man, dass die Punkte aus Ω von dem Start-Vektorfeld $V(0,x) = \tilde{V}(x)$ und dem Beschleunigungsfeld $\dot{V}(0,x) = -[D\tilde{V}(x)]\tilde{V}(x)$ beeinflusst werden. Die beiden Methoden werden, wie auch die Hadamard- und Gâteaux-Semiableitung, die gleichen Ableitungen produzieren, wenn es sich bei dem Konstruktionsgebiet zum Beispiel um den $\mathbb{R}^d$ handelt.

Im folgenden Theorem wollen wir die Verbindungen zwischen Hadamard-Semiableitung

und der *Velocity-Methode* festhalten, indem wir die Hadamard-Semiableitung einer reellwertige Funktion $f : \mathbb{R}^d \to \mathbb{R}$ mit der Semiableitung der *Velocity-Methode* in Verbindungen bringen.

Theorem 3.1.3 (Hadamard-Semiableitung und *Velocity-Methode*)**.** *Sei $U_{\bar{x}}$ eine Umgebung um einen Punkt $\bar{x} \in \mathbb{R}^d$ und $f : U_{\bar{x}} \to \mathbb{R}$. Dann ist f Hadamard-semidifferenzierbar im Punkt $\bar{x}$ in Richtung $v \in \mathbb{R}^d$ genau dann, wenn es ein $\tau > 0$ gibt, sodass für alle zeitabhängigen Vektorfelder $V : [0, \tau] \times \mathbb{R}^d \to \mathbb{R}^d$, die den folgenden Bedingungen genügen*

i. $V(\cdot, \bar{x}) \in C([0, \tau]; \mathbb{R}^d) \quad \forall \bar{x} \in \mathbb{R}^d$,

ii. $\exists c > 0$, *sodass* $\forall \bar{x}, y \in \mathbb{R}^d$ *gilt* $\|V(\cdot, y) - V(\cdot, \bar{x})\|_{C([0,\tau];\mathbb{R}^d)} \leq c\|y - \bar{x}\|$,

iii. der Grenzwert

$$d_H f(x, V) := \lim_{s \to 0} \frac{f(T(V)(s, \bar{x})) - f(\bar{x})}{s}$$

existiert. Hierbei ist $T(V)(s, \bar{x}) := x(s)$ die Lösung der Differentialgleichung

$$\begin{aligned} \dot{x} &= V(s, x(s)) \quad \forall s \geq 0 \\ x(0) &= \bar{x} \ , \end{aligned}$$

falls $V(0, \bar{x}) = v$ ist, dann folgt direkt hieraus, dass

$$d_H f(\bar{x}, V) := d_H f(\bar{x}, \tilde{V}) \quad \text{wobei} \quad \tilde{V}(s, \bar{x}) = v \quad \forall s \geq 0$$

gilt.

Beweis. Für einen Beweis wird auf [6] verwiesen. □

Theorem 3.1.3 motiviert uns die Hadamard-Semiableitung auch für Formfunktion zu definieren. Im folgenden Abschnitt wollen wir daher unsere Ergebnisse zusammentragen.

3.2. Erste Ordnung Semiableitungen

Zu den Bedingungen aus Theorem 3.1.3 müssen wir im beschränkten Fall $D \subset \mathbb{R}^d$ gewährleisten, dass das transformierte Gebiet in D enthalten ist. Dies können wir

erreichen, indem wir folgende Forderung an V stellen:

$$\forall x \in \partial D \ , \ \forall s \in [0,\tau] \text{ gilt } V(s,x) \in TR_{\partial D}(x)$$

$TR_{\partial D}(x)$ bezeichnet den Tangentialraum von ∂D im Punkt $x \in \partial D$. Der Rand ∂D des Konstruktionsgebiets D muss dafür genügend glatt sein, welches aber in den meisten Optimerungsproblemen gewährleistet ist. Im beschränkten Fall müssen folgende drei Bedinungen erfüllt werden:

$$V(\cdot,x) \in C([0,\tau];\mathbb{R}^d) \quad \forall x \in \mathbb{R}^d \tag{3.9}$$

$$\exists c > 0, \text{ sodass } \forall x,y \in \mathbb{R}^d \text{ gilt } \|V(\cdot,y) - V(\cdot,x)\|_{C([0,\tau];\mathbb{R}^d)} \leq c\|y-x\| \tag{3.10}$$

$$\forall x \in \partial D \ , \ \forall s \in [0,\tau] \text{ gilt } V(s,x) \in TR_{\partial D}(x) \tag{3.11}$$

Zur Notation: Eine Transformation

$$T_s(V)(\Omega) := T(V)(s,\Omega),$$

die von einem Vektorfeld, welches die Bedingungen (3.9) bis (3.11) erfüllt, erzeugt wird, bildet Gebiete $\Omega \in D$ auf Gebiete $\Omega_s \in D$ ab. Zudem bezeichnen wir mit $Lip(D,\mathbb{R}^n)$ den Raum aller Lipschitz-stetigen Funktionen von D nach $\mathbb{R}^n$.

Definition 3.2.1 (Semiableitung und Ableitung der Formfunktion)**.** *Sei $\mathcal{V}$ ein topologischer Untervektorraum von $Lip(D,\mathbb{R}^d)$ und sei J eine reellwertige Formfunktion.*

i. Wenn V ein zeitabhängiges Vektorfeld ist, welches den Bedingungen (3.9) *und* (3.10) *genügt, dann sagen wir J besitzt eine Euler-Semiableitung bei Ω in Richtung V, falls folgender Grenzwert existiert*

$$dJ(\Omega;V) := \lim_{s\to 0} \frac{J(\Omega_s(V)) - J(\Omega)}{s}$$

ii. Für ein $\tilde{V} \in Lip(D,\mathbb{R}^d)$ und zeitabhängiges Vektorfeld V

$$V(s,x) = \tilde{V}(x) \qquad \forall s \in [0,\tau], \quad \forall x \in D$$

benutzen wir die Notation $dJ(\Omega;V)$ oder einfach $dJ(\Omega;\tilde{V})$.

iii. Sei $\tilde{V} \in \mathcal{V}$. Wir sagen, J hat eine Hadamard-Semiableitung bei Ω in Richtung $\tilde{V}$

bezüglich $\mathcal{V}$, falls für alle zeitabhängigen Vektorfelder V gilt:

- *Bedingungen* (3.9) *und* (3.10) *sind erfüllt,*
- $V(s,.) \in \mathcal{V}$,
- $V(0,x) = \tilde{V}(x) =: V(0)$,
- *der Grenzwert $dJ(\Omega;V)$ existiert.*

In diesem Fall wird die Semiableitung mit $d_H J(\Omega;\tilde{V})$ bezeichnet und natürlich gilt

$$d_H J(\Omega;\tilde{V}) = dJ(\Omega;V(0)).$$

iv. J heißt differenzierbar bei Ω in $\mathcal{V}$, falls J in jede Richtung $\tilde{V} \in \mathcal{V}$ eine Euler-Semiableitung besitzt und die Abbildung

$$dJ(\Omega;.) : \mathcal{V} \to \mathbb{R} \tag{3.12}$$

linear und stetig, d.h. ein Element aus dem Dualraum $\mathcal{V}'$ von $\mathcal{V}$ ist.

Diese Definition der Euler-Semiableitung ist sehr allgemein gehalten und kann zum Beispiel sehr leicht auf Formfunktionen, die auf Untermannigfaltigkeiten des $\mathbb{R}^d$ definiert sind, erweitert werden. Sie schließt zudem Fälle ein, in denen $dJ(\Omega;V)$ nicht nur von $V(0)$ abhängen kann, sondern auch von $V(s)$ für eine Umgebung um $s = 0$. Wenn $dJ(\Omega;V)$ nur von $V(0)$ abhängt, dann kann die Analyse auf stationäre Vektorfelder beschränkt werden, da die Hadamard-Semiableitungen existieren.

Zur besseren Veranschaulichung wollen wir ein einfaches Beispiel betrachten.

Beispiel 3.2.2. *Seien $V \in C_0^\infty(\mathbb{R}^d,\mathbb{R}^d)$ und $\partial\Omega \in C^1$. Die Formfunktion J sei wie folgt definiert:*

$$J(\Omega) = \int_\Omega 1 \; dx$$

Es ist $\Omega_s = T_s(V)(\Omega)$, wobei $T_s(V)(.)$ Ω diffeomorph auf Ω_s abbildet. Dann gilt mithilfe des Transformationssatzes:

$$J(\Omega_s) = \int_{\Omega_s} 1 \; dy = \int_\Omega |detDT_s(V)(x)|dx$$

$DT_s(V)(x)$ ist wieder die Jacobi-Matrix von $T_s(V)(.)$ bezüglich x. Leiten wir J ab,

erhalten wir weiter:

$$\begin{aligned} dJ(\Omega;V) = \frac{\partial J(\Omega_s)}{\partial s}\Big|_{s=0} &= \frac{\partial}{\partial s}\int_\Omega |detDT_s(V)(x)|dx\Big|_{s=0} \\ &= \int_\Omega \frac{\partial |detDT_s(V)(x)|}{\partial s}\Big|_{s=0} dx \end{aligned}$$

Es gilt weiter für die Taylorentwicklung bezüglich s der Transformation $T_s(V)(.)$*:*

$$\begin{aligned} T_s(V)(x) &= T_0(V)(x) + \frac{\partial}{\partial s}T_s(V)(x)\Big|_{s=0} s + o(s^2) \\ &= \mathbb{I} + V(x)s + o(s^2) \end{aligned}$$

Damit folgt für die Jacobi-Matrix von $T_s(V)(x)$*:*

$$DT_s(V)(x) = \mathbb{I} + DV(x)s + o(s^2)$$

Setzen wir dies in die Formableitung ein, erhalten wir:

$$\begin{aligned} dJ(\Omega_s)\big|_{s=0} &= \int_\Omega \frac{\partial}{\partial s} detDT_s(V)(x)\big|_{s=0} dx \\ &= \int_\Omega \frac{\partial}{\partial s} det(\mathbb{I} + DV(x)s + o(s^2))\big|_{s=0} dx \\ &= \int_\Omega tr(DV(x))dx \\ &= \int_{\partial\Omega} V(x)\cdot\vec{n}\; d\nu \end{aligned}$$

$\vec{n}$ beschreibt hierbei die äußere Normale an $\partial\Omega$. Man erkennt, dass nur der Normalenanteil von V auf dem Rand von Ω in die Formableitung eingeht. Diese Beobachtung wollen wir im nächsten Kapitel genauer untersuchen.

3.3. Struktursatz

Wir beschäftigen uns wieder mit stationären Vektorfeldern $V \in \mathcal{V}$, wobei $\mathcal{V}$ ein topologischer Untervektorrau von $Lip(\mathbb{R}^d, \mathbb{R}^d)$ ist. Zur Vereinfachung schauen wir uns an dieser Stelle nur den unbeschränkten Fall $D = \mathbb{R}^d$ an. Der beschränkte Fall liefert ähnliche Resultate, ist jedoch technisch aufwendiger. Für eine genauere Diskussion sei hier auf [6] verwiesen.

Die Formableitung hängt im Wesentlichen von der Wahl des topologischen Untervektorraums von $Lip(\mathbb{R}^d, \mathbb{R}^d)$ ab. Wir wollen uns hier nur auf glatte Vektorfelder beschränken. Sei also $\mathcal{V} = C_0^\infty(\mathbb{R}^d, \mathbb{R}^d)$ der Raum der unendlich oft differenzierbaren Funktionen mit kompaktem Träger. Damit sind Bedingungen (3.9) und (3.10) für alle $V \in \mathcal{V}$ erfüllt.

Definition 3.3.1. *Sei J eine reellwertige Formfunktion und sei $\Omega \subset \mathbb{R}^d$ eine Teilmenge des $\mathbb{R}^d$.*

i. Wir sagen, die Formfunktion J ist formableitbar bei Ω, falls sie bei Ω in alle Richtungen $V \in \mathcal{V}$ differenzierbar ist.

ii. Die Abbildung in (3.12) *definiert eine Vektordistribution $\frac{\partial J}{\partial \Omega}$ in $\mathcal{V}'$, für welche wir folgende Notation einführen:*

$$\langle \frac{\partial J}{\partial \Omega}, V \rangle := dJ(\Omega; V)$$

$\langle \frac{\partial J}{\partial \Omega}, \cdot \rangle$ bedeutet hierbei die Anwendung von $\frac{\partial J}{\partial \Omega}$ auf ein Element $V \in \mathcal{V}$.

Im Folgenden wollen wir die Vektordistribution $\langle \frac{\partial J}{\partial \Omega}, \cdot \rangle$ untersuchen.

Lemma 3.3.2. *Die in* (3.6) *definierte Abbildung $T(s, x)$ ist für ein $V \in \mathcal{V} = C_0^\infty(\mathbb{R}^d, \mathbb{R}^d)$ ein C^1-Diffeomorphismus.*

Beweis. Das dynamische System F zu einem $V \in \mathcal{V}$ ist unabhängig von der Zeit, sprich autonom, denn es gilt $F(s, x) = F(x) := V(x)$. Des Weiteren ist F linear beschränkt, denn es gilt

$$\|F(s, x)\| \leq L\|x\| + \|F(0)\|,$$

wobei L die Lipschitz-Konstante von V ist. Somit ist für jedes Intervall $[0, a], a > 0$ die Abbildung $T(s, x)$ mit $s \in [0, a]$ definiert, da die einzelnen Integralkurven auf ganz $[0, a]$ definiert sind. Nach dem Satz von der Diffeomorphie der Zustandsabbildung ist $T_s(V)(x) = T(s, x), s \in [0, a], T_s(V)(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$ ein Diffeomorphismus.
Für Details sei auf [8] verwiesen. □

Theorem 3.3.3 (Struktursatz)**.** *Sei J eine reellwertige Formfunktion. J sei formableitbar für eine Teilmenge Ω des $\mathbb{R}^d$.*
Dann ist der Träger von $\langle \frac{\partial J}{\partial \Omega}, \cdot \rangle$ in $\partial\Omega$ enthalten.

Beweis. Sei $V \in C_0^\infty(\mathbb{R}^d, \mathbb{R}^d)$ mit $V = 0$ auf $\partial\Omega$. Zu zeigen ist, dass $T_s(V)(\Omega) = \Omega$ ist.

Sei dazu $x_0 \in \partial\Omega$. Für die Trajektorie $\bar{x}(s)$ von x_0 gilt:

$$\begin{aligned} \dot{\bar{x}}(s) &= V(\bar{x}(s)) \quad \forall s \geq 0 \\ \bar{x}(0) &= x_0 \\ \dot{\bar{x}}(0) &= V(\bar{x}(0)) = 0 \end{aligned}$$

Die eindeutige Lösung ist hier gegeben durch $\bar{x}(s, x_0) = x_0$. Das heißt also, dass der Rand $\partial\Omega$ von Ω auf sich selbst abgebildet wird:

$$T_s(V)(\partial\Omega) = \partial\Omega$$

Der Rand von Ω teilt den $\mathbb{R}^d$ in zwei disjunkte Zusammenhangskomponenten, zum einen in $\Omega \backslash \partial\Omega = int(\Omega)$ und zum anderen in $\mathbb{R}^d \backslash \bar{\Omega}$. Da $T_s(V)(\cdot)$ nach Lemma 3.3.2 als Diffeomorphismus stetig ist, gilt dies auch für die Bilder der beiden Komponenten. Weil stetige Abbildungen beschränkte Mengen auf beschränkte Mengen abbilden, muss $T_s(V)(int(\Omega)) = int(\Omega)$ gelten. Somit folgt

$$\begin{aligned} & T_s(V)(\Omega) && = \Omega && \forall s \geq 0 \\ \Rightarrow\ & J(T_s(V)(\Omega)) && = J(\Omega) && \forall s \geq 0 \\ \Rightarrow\ & dJ(\Omega; V) && = 0 \end{aligned}$$

Es gilt also $\langle \frac{\partial J}{\partial \Omega}, V \rangle = 0$ für alle $V \in \mathcal{V}$ mit $V\big|_{\partial\Omega} = 0$. □

Bemerkung. Die Aussage von Theorem 3.3.3 lässt sich auf Vektorfelder $V \in C_0^\infty(\mathbb{R}^d, \mathbb{R}^d)$ ohne Normalenanteil übertragen.

$$V \cdot \vec{n} = 0 \text{ auf } \partial\Omega \Rightarrow dJ(\Omega; V) = 0$$

Auch hier kann man zeigen, dass der Rand von Ω auf sich selbst abgebildet wird und dann wie in Theorem 3.3.3 schlussfolgern.

Zuletzt wollen wir uns noch weitere Beispiele für Formableitungen anschauen. Diese Beispiele dienen als Grundlage für die spätere Formoptimierung und unseres Anwendungsbeispiels des Chan-Vese Energiefunktionals.

Beispiel 3.3.4. *Sei $J(\Omega) = \int_\Omega f(x)dx$ mit $f : \mathbb{R}^n \to \mathbb{R}$ glatt. Dann folgt mit dem Transformationssatz für ein Pfad $\gamma(t)$ der $\partial\Omega$ entlang läuft:*

$$J(\Omega_s) = \int_{\Omega_s} f dx = \int_\Omega f \circ T_s(V)\gamma(t) dx$$

Für die Formableitung erhalten wir damit:

$$\begin{aligned} dJ(\Omega_s; V) &= \lim_{s\to 0} \frac{1}{s} \int_\Omega (f \circ T_s(V))\gamma(s) - (f \circ T_s(V))\gamma(0) dx \\ \underset{f \text{ glatt genug}}{\Rightarrow} dJ(\Omega_s; V) &= \int_\Omega \lim_{s\to 0} \frac{1}{s}((f \circ T_s(V))\gamma(s) - (f \circ T_s(V))\gamma(0)) dx \\ &= \int_\Omega (\nabla f V(0) + f \operatorname{div} V(0)) dx \\ &= \int_\Omega \operatorname{div}(FV(0)) dx \underset{Gauß}{=} \int_{\partial\Omega} FV(0) dS \ , \end{aligned}$$

wobei F die Stammfunktion von f ist.

Beispiel 3.3.5. *Sei $J(\Omega) = \int_{\partial\Omega} f(x) dS(x)$ mit $f : \mathbb{R}^n \to \mathbb{R}$ glatt. Dann folgt durch Transformation:*

$$J(\Omega_s) = \int_{\partial\Omega_s} f(x) dS(x) = \int_{\partial\Omega} (f(x) \circ T_s(V))\omega(s) dS(x)$$

mit

$$\omega(s) = \gamma(s) || DT_s^{-T} \vec{n} ||$$

für einen Pfad γ der $\partial\Omega$ entlang läuft. Dann können wir schlussfolgern:

$$\begin{aligned} dJ(\Omega; V) &= \int_{\partial\Omega} \lim_{s\to 0} \frac{1}{s}((f \circ T_s(V))\omega(s) - (f \circ T_0(V))\omega(0)) dS(x) \\ &= \int_{\partial\Omega} \nabla f \cdot V(0) + f\omega'(0)) dS(x). \end{aligned}$$

Es kann gezeigt werden (siehe [17] Seite 80), dass

$$w'(0) = div V(0) - DV(0)\vec{n} \cdot \vec{n}$$

gilt.

Definition 3.3.6 (Tangentiale Divergenz)**.** *Sei $\Omega \subset \mathbb{R}^n$, $\partial\Omega$ der Rand von Ω und V ein Vektorfeld. Dann ist die tangentiale Divergenz definiert durch:*

$$div_{\partial\Omega} V = div V - DV\vec{n} \cdot \vec{n}$$

Damit folgt

$$dJ(\Omega; V) = \int_{\partial\Omega} (\nabla f \cdot V(0) + f div_{\partial\Omega} V(0)) dS(x)$$

4. Formoptimierung mit Level-Set Funktionen

Im nun folgenden Kapitel wollen wir die Ergebnisse der Kapitel 2 und 3 zusammenfügen und die Shape Sensitivity Analysis mit der Level-Set-Methode verbinden. Diese diskretisieren wir dann mithilfe des nicht linearen cg-Verfahrens und des quasi-Newton Verfahrens numerisch.

Zu Beginn müssen wir jedoch allgemeine Vorbereitungen treffen und uns überlegen, wie wir die beiden Verfahren verbinden und nutzen können.

4.1. Vorbereitungen

4.1.1. Formableitung als Abstiegsrichtung

Die erste Vorüberlegung ist, dass wir in den Algorithmen 2.3.1, 2.3.5 und 2.3.6 jeweils den Gradienten als Abstiegsrichtung benötigen bzw. daraus eine konjugierte Abstiegsrichtung herleiten. Jedoch haben wir gesehen, dass für Formfunktionen der Ableitungsbegriff nicht definiert ist. Hier nutzen wir nun die Shape Sensitivity Analysis, die uns genau diesen Begriff bereitstellt und identifizieren daher ∇f mit $\langle \frac{\partial J}{\partial \Omega}, \cdot \rangle$. Die Formableitung ist, wie wir mit dem Struktursatz 3.3.3 gesehen haben, nur auf dem Rand des Objektes definiert, so wie wir es erwarten würden. Wir wollen also das Objekt mittels einer initialen Level-Set-Funktion, die wir mithilfe der Formableitung bewegen, erkennen.

Mit dieser Vorüberlegung können wir nun das einfache Gradientenabstiegsverfahren für Formfunktionen aufstellen. Dieses lautet wie folgt:
Algorithmus 4.1.1 (Gradientenabstiegsverfahren für Formfunktionen)**.** *Sei J eine differenzierbare Formfunktion, die zu minimieren sei. Dann wiederhole folgende Schrit-*

te bis ein Abbruchkriterium erfüllt ist:

- *Bestimme Normalenanteil der Abstiegsrichtung $V = \tilde{V} \cdot \vec{n}$ zu Ω mit der Formableitung $V = -dJ(\Omega_n)$.*
- *Löse anschließend die Differentialgleichung $\dot{x}(t) = V(x(0))$ mit $x(0) = x_0$ für alle Trajektorien $x \in \partial\Omega_n$, um Ω_{n+1} zu erhalten.*

Anschaulich bedeutet das, dass wir die Abstiegsrichtung mithilfe der Formableitung bestimmen und anschließend den Rand unserer Form in diese Richtung bewegen.

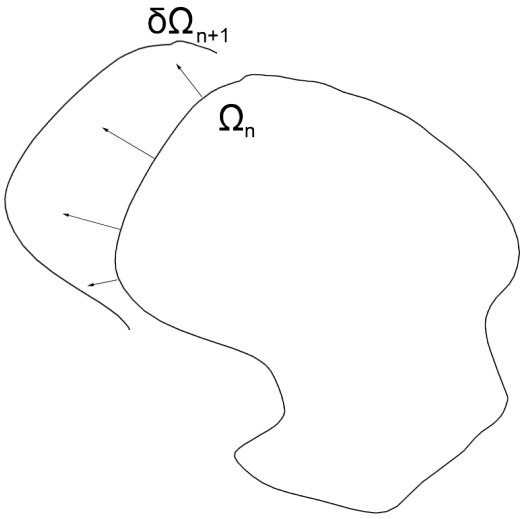

Abbildung 4.1.: Teilstück der Bewegung von $\partial\Omega_n$ in Normalenrichtung.

Betrachten wir diesen Algorithmus im Zusammenhang mit Level-Set Funktionen, stellen wir fest, dass die Bewegung des Objekts nur für $x \in \partial\Omega$ definiert ist. Level-Set Funktionen beruhen aber gerade darauf, das gesamte Gebiet implizit darzustellen. Daher erweitern wir in der nächsten Vorüberlegung die Formableitung auf das gesamte Gebiet, um den Gradientenabstieg für Level-Set Funktionen mit Formableitung definieren zu können.

4.1.2. Erweiterung der Formableitung auf D

Wie bereits erwähnt, ist die Formableitung nur auf dem Rand des Objektes definiert, was aber bei der Diskretisierung mit Level-Set Funktionen zu Problemen führt. In diesem Fall müsste der Rand explizit dargestellt werden. Wir wollen jedoch den Rand

durch die signierte Distanzfunktion, siehe 2.2.1, als implizite Level-Set Funktion darstellen. Daher kann die Kante zwischen Pixeln liegen und wir müssten zwischen Pixeln, in denen der Vorzeichenwechsel stattfindet, die Distanzfunktion interpolieren, sodass wir die explizite Null-Niveaumenge erhalten, um dort die Formableitung zu bestimmen und die Level-Set Funktion zu bewegen. Um dieses Schwierigkeit zu umgehen, wollen wir die Formableitung auf das gesamte Gebiet $D \subset \mathbb{R}^2$ erweitern.

Hierzu wählen wir eine Projektion P, die von $\partial\Omega$ auf ganz D surjektiv abbildet. Wie diese Projektion P explizit aussieht, wollen wir im nächsten Abschnitt erörtern, da es verschiedene Ansätze für die Wahl von P gibt. Nutzen wir diese Projektion P, können wir nun das Gradientenabstiegsverfahren mit Level-Set Funktionen für Formfunktionen wie folgt definieren:

Algorithmus 4.1.2 (Gradientenabstiegsverfahren mit Level-Set Funktionen für Formfunktionen)**.** *Sei ϕ die implizite Darstellung von D. Dann wiederhole folgende Schritte bis ein Abbruchkriterium erfüllt ist:*

- *Bestimme Normalenanteil der Abstiegsrichtungen V zu $\{\phi_n = 0\}$ mit $V = -dJ(\{\phi_n = 0\})$.*
- *Projiziere die Formableitung auf das gesamte Gebiet, das heißt, $\forall y \in \mathbb{R}^n$ existiert ein $x \in \{\phi_n = 0\}$, sodass gilt V(y) = V(P(x)).*
- *Führe fiktive Zeit t ein, um die Evolutionsgleichung $\dot{\phi}(x,t) = V(x)|\nabla\phi(x,t)|$ mit $\phi(x,0) = \phi_n(x)$ zu erhalten. Setze die Lösung der Differentialgleichung gleich ϕ_{n+1}.*

Die Definition einer solchen Projektion P führt bei der Implementierung zu einer Vereinfachung. Im letzten Schritt wollen wir die explizite Implementierung für den Algorithmus 4.1.2 angeben.

4.1.3. Gradientabstieg für Formfunktionen mit Level-Set Funktionen

Zuerst wollen wir auf die explizite Wahl von P eingehen. Eine intuitive Wahl von P ist die Erweiterung des Randes in Normalenrichtung. Wie in Abbildung 4.1 sehen wir, dass für die Berechnung von $\partial\Omega_{n+1}$ die Informationen von $\partial\Omega_n$ zur Verfügung stehen,

da der Rand nur in Normalenrichtung bewegt wird. Das heißt explizit gilt $\forall y \in D$

$$P(y) = \arg\min_{x \in \Omega}(x, y) \ .$$

Die Existenz dieses Minimierers kann in [10] Kapitel 3.3 nachgelesen werden. Zudem sieht man, dass der Minimierer orthogonal auf dem Rand von Ω steht. Dieser Minimierer existiert, solang Ω ein geschlossenerer Unterraum von $\mathbb{R}^n$ ist. Gehen wir nun weiter davon aus, dass unser Gebiet mit der signierten Distanzfunktion belegt ist, so ergibt sich für alle $y \in D$

$$P(y) = y - \frac{\nabla\phi(y)}{|\nabla\phi(y)|} \cdot |\phi(y)|.$$

Die Formel ergibt sich, indem wir von jedem Punkt y aus jeweils in Normalen-Richtung $-\frac{\nabla\phi(y)}{|\nabla\phi(y)|}$ um die Entfernung zum Rand $|\phi(y)|$ laufen. Diese Werte werden uns direkt durch die signierte Distanzfunktion vorgegeben. Da $|\nabla\phi(y)| = 1$ gilt, können wir weiter vereinfachen und erhalten:

$$P(y) = y - \nabla\phi(y) \cdot |\phi(y)| \ . \tag{4.1}$$

Mithilfe der Projektion P können wir nun den einfachen Gradientenabstieg für Formfunktion mit Level-Set Funktionen angeben. In Algorithmus 4.1.2 ersetzen wir die ersten beiden Schritte durch:

- Für jeden Punkt $y \in D$ bestimme projizierten Randpunkt $x \in \partial\Omega$ mit $x = y - \nabla\phi(y) \cdot |\phi(y)|$
- Bestimme anschließend für jeden Punkt $y \in D$ die Abstiegsrichtung $V = -dJ(y)$

Im Anschluss muss nur noch auf dem gesamten Gebiet die Differentialgleichung gelöst werden und wir erhalten die nächste Iteration von ϕ.
Bemerkung. Da wir die Eigenschaft $|\nabla\phi(y)| = 1$ nutzen, müssen wir nach jedem Schritt unser Gebiet neu mit der signierten Distanzfunktion belegen. Für eine genauere Diskussion sei auf Abschnitt 5.1.2 verwiesen.

4.1.4. Erweiterung von NCG- und Quasi-Newton-Methoden

Nachdem wir nun die Grundlagen gelegt haben, fehlt uns nur noch der finale Schritt. Sowohl im nicht-linearen cg-Verfahren als auch im Quasi-Newton-Verfahren gibt es Elemente, die wie der Gradient beim Gradientenabstiegsverfahren nur auf dem Rand $\partial\Omega$

existieren. Auch diese Elemente wollen wir mithilfe unser Projektion P auf das gesamte Gebiet erweitern. Ein weiterer Punkt ist, dass bei den beiden Methoden Elemente aus der vorherigen Iteration genutzt werden müssen. Wir müssen also sicherstellen, dass die Elemente des Randes mitbewegt werden, sodass im nächsten Schritt die Elemente zur Verfügung stehen. Dazu wollen wir uns beispielhaft den Transport des konjugierten Gradienten d im nicht-linearen cg-Verfahren anschauen.

In jedem Iterationsschritt wird in der Berechnung von d_n, die alte konjugierte Suchrichtung d_{n-1} benötigt, da

$$d_n = r_n + \beta_n d_{n-1}$$

gilt. Dies ist jedoch kein Problem, da wir d_n mithilfe von der Projektion P auf das gesamte Gebiet in Normalenrichtung erweitert haben. Des Weiteren treiben wir unsere Evolution von ϕ nur in Normalenrichtung voran und die Projektion P erweitert d auch in Normalenrichtung, sodass sichergestellt ist, dass für jedes $x \in \partial\Omega$ das richtige Element genutzt wird. Bei genauerer Betrachtung stellen wir fest, dass für alle Elemente, dank der Konstruktion von P, diese Eigenschaft gilt.

Wir haben damit alle Vorüberlegungen abgeschlossen und können den Algorithmus für das nicht-lineare cg-Verfahren zur Formoptimierung mit Level-Set Funktionen angeben:
Algorithmus 4.1.3 (NCG-Methode zur Formoptimierung mit Level-Set Funktionen)**.** *Sei ϕ_0 ein beliebiger Startwert für die Repräsentation von Ω. Gegeben eine zu minimierende Formfunktion J. Bestimme für jeden Punkt $x \in D$ projizierten Randpunkt $y \in \{\phi = 0\}$ mit der Projektion die in* (4.1) *definiert wurde.*

$$r_0(x_0) = -dJ(\phi_0(y_0)) \ \forall x_0 \in D$$

Löse die Differentialgleichung $\dot{\phi}(x,t) = r_0(x)|\nabla\phi(x,t)|$ mit $\phi(x,0) = \phi_0(x)$ für alle $x \in D$ um ϕ_1 zu erhalten. Nach dieser ersten Iteration wiederholt man folgende Schritte bis ein gegebenes Konvergenzkriterium erfüllt ist. Hierbei ist d die konjugierte Richtung und es gilt $d_0 = r_0$.

- *Projiziere Punkte auf den Rand mit Projektion P wie in* (4.1)*.*
- *Berechne den steilsten Abstieg für alle $x \in D$*

$$r_n(x) = -dJ(\phi_n(x)).$$

- *Berechne β_n wie in Algorhitmus 2.3.2 mit Fletcher-Reeves oder Polak-Ribière.*

- *Update konjugierte Suchrichtung* d_n

$$d_n(x) = r_n(x) + \beta_n d_{n-1}(x).$$

- *Löse Differentialgleichung* $\dot{\phi}(x,t) = d_n(x)|\nabla\phi(x,t)|$ *mit* $\phi(x,0) = \phi_n(x)$ *für alle* $x \in D$ *um* ϕ_{n+1} *zu erhalten.*

- *Reinitialisiere* ϕ_{n+1} *mit der signierten Distanzfunktion.*

Bemerkung. Zu beachten ist, dass d_n durch die Erweiterung von r_n auf dem gesamten Rechengebiet $\mathbb{R}^d$ vorhanden ist und nicht nur auf $\{\phi_n = 0\}$ existiert.

Für das Quasi-Newton-Verfahren oder auch BFGS-Verfahren nutzen wir den limited-Memory BFGS-Algorithmus. Dieser Algorithmus unterscheidet sich vom normalen BFGS-Verfahren nur in dem Punkt, dass die Hessenmatrix B nie explizit gespeichert wird, sondern in jeder Iteration aus den letzten $m \in \mathbb{N}$ Iterationen rekonstruiert wird. So kann mit einfacher Vektor- und Skalarmultiplikation gerechnet werden, anstatt die ganze Matrix in jedem Schritt zu nutzen und zu speichern. Ein weiterer Vorteil ist, dass wir im normalen BFGS-Verfahren die Hessenmatrix bewegen bzw. projizieren müssten. Beim limited-Memory BFGS-Verfahren können wir jeweils die m gespeicherten Iterationen bewegen und damit unsere Berechnungen vereinfachen. Eine genauere Diskussion zum limited-Memory BFGS-Algorithmus kann in [15] und [9] eingesehen werden.

Algorithmus 4.1.4 (Quasi-Newton-Methode zur Formoptimierung mit Level-Set Funktionen)**.** *Wähle Startfunktion* ϕ_0*. Sei* $g_k = dJ$ *für eine zu minimierende Formfunktion* J*. Seien für die letzten* m *Iterationen die Werte* $s_k = x_{k+1} - x_k$ *und* $y_k = g_{k+1} - g_k$ *gespeichert. Dazu sei* $\rho_k = \frac{1}{y_k^T s_k}$ *in jedem Schritt. Wiederhole nun folgende Schritte bis ein gegebene Konvergenzkriterum erfüllt ist.*

- *Setze* $q = -g_k$*.*

- *Projiziere alle* m *gespeicherten Einträge von* s_k *und* y_k *auf den aktuellen Rand mit Projektion* P *wie in* (4.1)*.*

- *Für* $i = k-1$ *bis* $k-m$ *bestimme*

$$\alpha_i = \rho_i s_i^T q,$$
$$q = q - \alpha_i y_i.$$

- *Bestimme approximierte Hessenmatrix* $H_k = \frac{y_{k-1}^T s_{k-1}}{y_{k-1}^T y_{k-1}}$.

- *Bestimme Abstiegsrichtung* $z = H_k q$.
- *Löse Gleichungssystem für* $i = k - m$ *bis* $k - 1$

$$\beta_i = \rho_i y_i^T z,$$
$$z = z + s_i(\alpha_i - \beta_i).$$

- *Löse Differentialgleichung* $\dot{\phi}(x,t) = z(x)|\nabla\phi(x,t)|$ *mit* $\phi(x,0) = \phi_n(x)$ *für alle* $x \in D$ *um* ϕ_{n+1} *zu erhalten.*
- *Reinitialisiere* ϕ_{n+1} *mit der signierten Distanzfunktion.*

Sollte es während der Iteration dazu kommen, dass d_k *keine „echte“ Abstiegsrichtung ist, also das* $z \cdot q > 0$ *gilt, initialisieren wir den Algorithmus neu. Das heißt, wir setzen die Anzahl der gespeicherten Iterationen auf null und iterieren weiter.*
Bemerkung. Im BFGS-Algorithmus müssen wir im Vergleich zum nicht-linearen cg-Verfahren mehr Daten bewegen, da nicht nur Informationen aus dem letzten Schritt sondern aus den m letzten Schritten benötigt werden und sich der Rand und somit die Normale verändert haben kann.

4.2. Zusammenfassung

Wie wir gesehen haben, ist die Verbindung von Shape Sensitivity Analysis und Level-Set Funktionen mit einigen zusätzlichen Überlegungen möglich. Vor allem kommt es hier auf die Wahl der Projektion P an, die allen Elemente im Rechengebiet D ein Element auf dem Rand des Obejekts $\partial\Omega$ zuordnet und so die Erweiterung, zum Beispiel der Abstiegsrichtung, auf das gesamte Gebiet zulässt. Diese Projektion vereinfacht zudem das Nutzen von Informationen aus den vorherigen Iterationsschritten, die bei weiterführenden Verfahren benötigt werden. Diese Informationen sind dank der Projektion P auf dem gesamten Rechengebiet vorhanden.

5. Bildsegmentierung als Anwendungsbeispiel

Im folgenden Kapitel wollen wir unsere Erkenntnisse und Algorithmen für die Bildsegmentierung mit dem Chan-Vese Energiefunktional (2.5) nutzen. Dazu vergleichen wir das allgemeine Gradientenabstiegsverfahren mit dem nicht-linearen cg- und dem Quasi-Newton-Verfahren. Dazu erläutern wir zu Beginn die Implementierung der Verfahren und vergleichen im Anschluss das numerische Verhalten der Verfahren sowohl untereinander als auch im Vergleich zu ihrer „klassischen" Segmentierungs-Implementierung mithilfe der Heaviside-Funktion.

5.1. Diskretisierung

5.1.1. Level-Set-Gleichung und Reinitalisierung

Dieser Abschnitt orientiert sich in Teilen an der Diplomarbeit von Martin Pach [14]. Sei $\Omega \subset \mathbb{R}^2$ ein Gebiet mit glattem Rand, welches durch die Level-Set-Funktion ϕ beschrieben wird.

$$\begin{aligned} \partial\Omega &= \{x \in \mathbb{R}^2 | \phi(x) = 0\} \\ \Omega &= \{x \in \mathbb{R}^2 | \phi > 0\} \end{aligned}$$

So ist die äußere Normale und die mittlere Krümmung an $\{\phi = 0\}$ gegeben durch:

$$\vec{n} = -\frac{\nabla\phi}{|\nabla\phi|} \quad \& \quad \kappa = div(\vec{n}) \tag{5.1}$$

Unter Reintialisierung von Level-Set Funktion versteht man das Ersetzen der Level-Set-Funktion ϕ durch eine neue Level-Set-Funktion ϕ_{neu}, die die gleiche Form beschreibt

wie ϕ, jedoch bessere Eigenschaften besitzt. Dafür belegen wir ϕ mit der signierten Distanzfunktion 2.2.1, sodass gilt:

$$\phi_{neu} = \phi \text{ für } \{\phi = 0\} \quad \& \quad |\nabla \phi_{neu}| = 1$$

Wie schon vorher beschrieben, führen wir diese Reinitalisierung nach jeder Iteration durch, da die signierte Distanzfunktion bessere Konditionen zum Berechnen liefert. Zum einen ist die Berechnung der mittleren Krümmung an $\{\phi = 0\}$ gegeben durch

$$\kappa = \operatorname{div}(\frac{\nabla \phi}{|\nabla \phi|}) \ .$$

Hierbei sieht man, dass es für $|\nabla \phi|$ nahe Null zu erheblichen Rundungsfehlern kommen kann. Zudem nutzen wir die Eigenschaft $|\nabla \phi| = 1$ bei der Berechnung der Abstiegsrichtung, sodass es einem unmittelbaren Einfluss auf die Minimierung gibt.

Die Evolution der Gleichung, also das Lösen der Differentialgleichung $\dot{\phi}(x,t) = v(x)|\nabla \phi(x,t)|$ mit $\phi(x,0) = \phi_0(x)$, vollziehen wir mit dem Upwind-Verfahren, da dieses deutlich stabiler ist als zum Beispiel das explizite Euler-Verfahren.

5.1.2. Formableitung für Chan-Vese Funktional

Bevor wir genauer auf die Implementierung eingehen, bestimmen wir zuerst die Formableitung unseres Energiefunktionals (2.5). Wie wir in Beispiel 3.3.4 gesehen haben, können wir die Formableitung für ein Funktional J einfach bestimmen. Wir erhalten für $(x,y) \in \{\phi = 0\}$:

$$dJ_{CV}(\phi(x,y)) = -|u_0(x,y) - c_1|^2 + |u_0(x,y) - c_2|^2 + \kappa(x,y) \ , \tag{5.2}$$

wobei κ die Krümmung von ϕ in (x,y) ist. Das Minus vor dem ersten Term kommt von der Normalen, da wir jeweils die nach außengerichtete Normale betrachten, wie in Gleichung (5.1) zu sehen ist.

5.1.3. Heaviside-Regularisierung als Vergleichskriterium

Wir wollen kurz auf die genutzte Heaviside-Regularisierung eingehen, da diese als Vergleichskriterium eine Rolle spielt. Wir benötigen eine möglichst glatte Funktion, die

dennoch bei $x = 0$ schnellstmöglich von 0 auf 1 springt. Zwei mögliche Regularisierungen sind:

$$H_{1,\epsilon}(z) = \begin{cases} 1 \text{ wenn } z > \epsilon, \\ 0 \text{ wenn } z < -\epsilon, \\ \frac{1}{2}[1 + \frac{z}{\epsilon} + \frac{1}{\pi} sin(\frac{\pi z}{\epsilon})] \text{ wenn } |z| \leq \epsilon \end{cases} \tag{5.3}$$

$$H_{2,\epsilon}(z) = \frac{1}{2}(1 + \frac{2}{\pi} arctan(\frac{z}{\epsilon})) \tag{5.4}$$

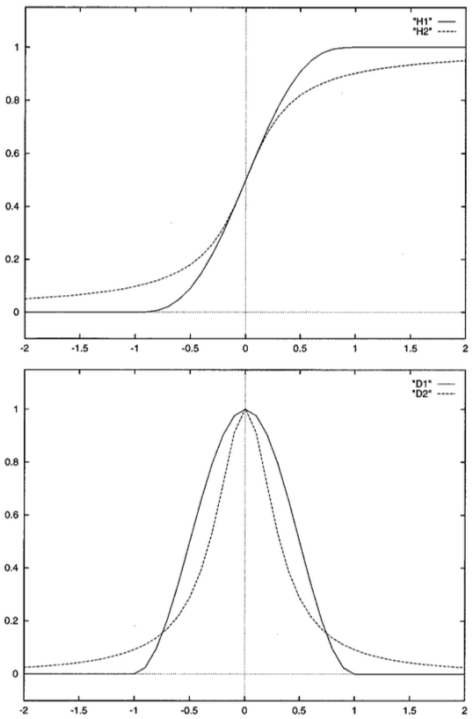

Abbildung 5.1.: Graphen der Heaviside-Funktionen (oben) und des Dirac-Maßes (unten). Durchgezogenen Linie $H_{1,\epsilon}$, gestrichelt Linie $H_{2,\epsilon}$. [Quelle [5]]

Wir nutzen in unserer Implementierung $H_{2,\epsilon}(z)$ da diese auf dem gesamten Gebiet nicht null ist, sodass auch weiter entfernte Kanten erkannt werden. Für eine genauere Diskussion über die Wahl der Heaviside-Regularisierung sei auf [5] verwiesen.

5.2. Numerischer Vergleich

Für den numerischen Vergleich wollen wir das nicht-lineare cg-Verfahren und Quasi-Newton-Verfahren mit Formableitungen mit dem allgemeinen Gradientenabstieg für

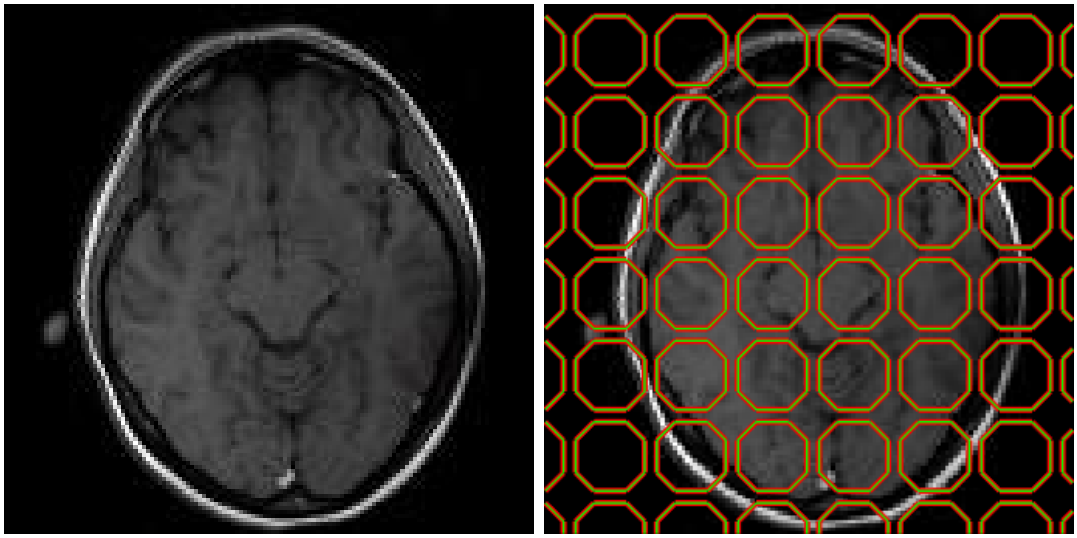

Abbildung 5.2.: Links ein Gehirn im Querschnitt und rechts mit initialer Level-Set Funktion.

Formableitungen vergleichen. Zudem vergleichen wir die beiden Verfahren mit ihrer Implementation mithilfe der Heaviside-Funktion.

Im weiteren Verlauf dieses Abschnittes ist $\mu = 0.02$ und als Abbruchkriterium wählen wir $|\nabla J| < \epsilon$ für $\epsilon = 10^{-1}$. Zudem wurde bei allen Algorithmen zum Lösen der Differentialgleichung das Upwind-Verfahren genutzt. Die Zeitschrittweite wird so gewählt, dass die CFL-Bedingung erfüllt ist, jedoch der Rand nicht weiter als 5 Pixel bewegt wird.

Die Laufzeitanalysen wurden alle auf einem Mac Mini mit 2,6Ghz Intel Core i7 und 16GB Ram ausgeführt.

In Abbildung 5.2 sehen wir unser erstes Beispiel zur Bildsegmentierung. Wir haben die Startwerte als kleine Kreise gewählt, damit man sieht, dass topologische Änderungen keine Rolle für die Algorithmen spielen. Die Abbildungen 5.3 zeigen die Ergebnisse der Optimierung mit Formableitungen. Gut zu erkennen ist, dass das Gradientenabstiegsverfahren vor allem im oberen Bereich das Bild nicht so schnell wie die anderen Algorithmen segmentieren kann. Dies spiegelt sich auch im Fehler, der in Abbildungen 5.4 gezeigt wird, wieder. Jedoch ist zu erkennen, dass sich der Fehler der Verfahren im Laufe der Iterationen annähert. Dies liegt vor allem an Diskretisierungsfehlern in den nicht-linearen Verfahren. Eigentlich würden wir ein ähnliches Verhalten wie in Abbildung 2.2 erwarten, jedoch kommt es vor allem durch die Reinitalisierung und durch die Projektion P schnell zu einem Gleichgewichtszustand. Obwohl wir bei der Reintialisierung darauf achten, auch Kanten zwischen Objekten zuzulassen, verhindert die

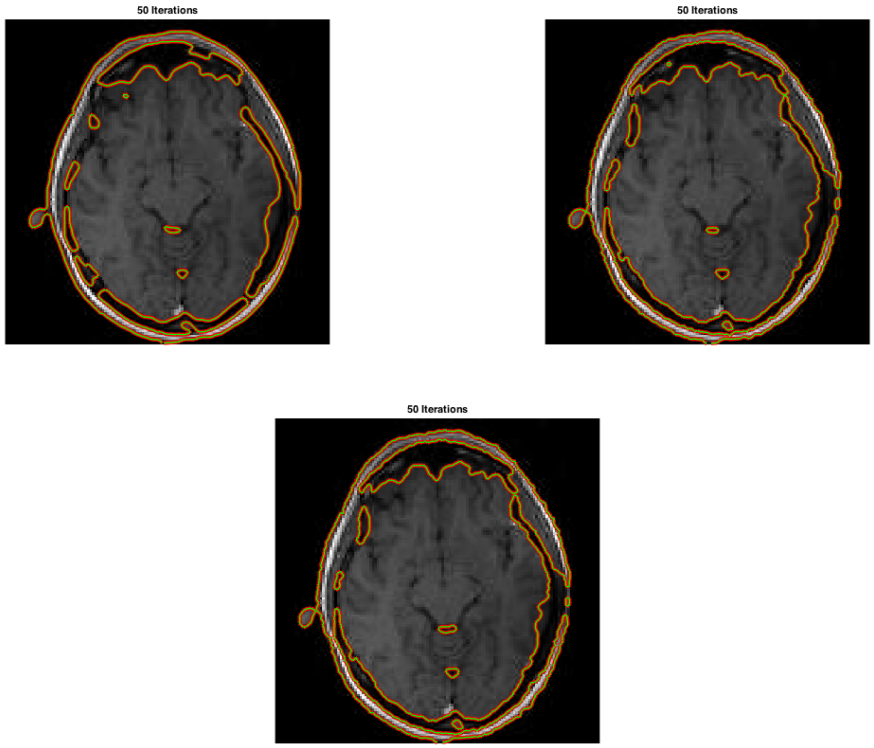

Abbildung 5.3.: Ergebnis nach 50 Iterationen. Oben links Gradientenabstieg, oben rechts nicht-lineares cg-Verfahren und unten L-BFGS-Verfahren jeweils mit Formfunktionen.

Erweiterung der Formableitung, also die Projektion P, die Verbesserung in der Konvergenz, da wir jeweils nur auf den nächsten Pixel runden können. So wird entweder der gleiche Pixel bei wenig bewegtem Rand gewählt oder der Nachbarpixel, welches einen Sprung im Fehler ergibt. Eine Lösung wäre hier zum Beispiel, wenn man zwischen den Pixeln interpolieren würde oder die Projektion anders kontinuierlich fortsetzt, sodass auch Formableitungen und Rechenwerte zwischen Pixeln gewählt werden können. Diesem Fakt ist auch zu schulden, dass sich die Energie in allen Verfahren ähnlich verhält. Zudem wird zur Bestimmung des Chan-Vese Funktionals, sowohl bei der Energie als auch bei der Formableitung, nur pixelweise gerechnet und nicht die explizite signierte Distanzfunktion genutzt. Dies lässt nur ungenaue Approximationen zu und ließe sich auch nur verhindern, wenn man die Funktionswerte zwischen Pixeln interpolieren würde oder andere kontinuierliche Ansätze zum Fortsetzen nutzen würde.

Wir wollen nun einen kurzen Vergleich zur Implementierung mit der Heaviside-Funktion geben. Wie in Tabelle 5.1 zu sehen, sind die Implementierungen mit der Heaviside-Funktion schneller als die mit den Formfunktionen. Dies liegt vor allem an der Projek-

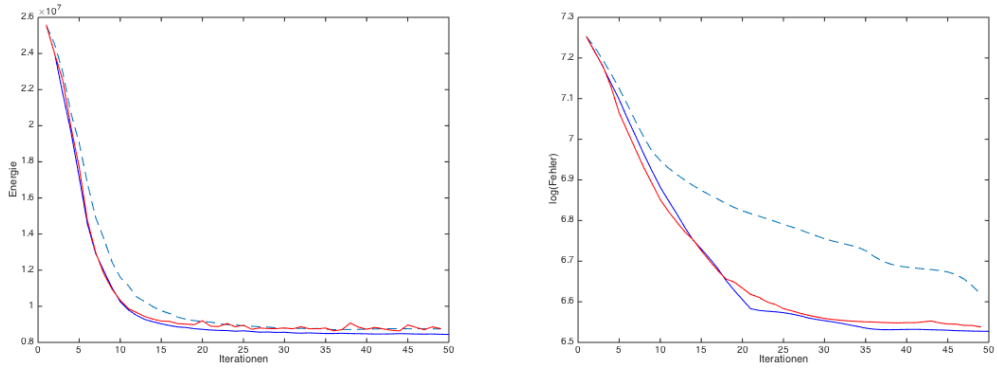

Abbildung 5.4.: Ergebnis nach 50 Iterationen. Links die Energie und rechts der logarithmische Fehler. Gestrichelt der Gradientenabstieg, blau das nicht-lineare cg-Verfahren und in rot das L-BFGS-Verfahren.

Laufzeit	Gradientenabstieg	ncg-Verfahren	L-BFGS
Formfunktion	42.145 s	69.454 s	94.282 s
Heaviside	16.890 s	27.934 s	16.180 s

Tabelle 5.1.: Mittelung von drei Zeitmessungen für Gradientenabstieg, ncg-Verfahren und L-BFGS für Formfunktionen.

tion, da in jedem Schritt für jeden Pixel der projizierte Pixel bestimmt werden muss. Vor allem kommt dies im L-BFGS Verfahren zu Geltung, da dort für alle gespeicherten Schritte die Projektion durchgeführt werden muss, was den großen Anstieg in der Laufzeit erklärt.

Schaut man sich die Lösung mithilfe der Heaviside-Funktion in Abbildung 5.5 an, erkennt man gut, dass das Gradientenabstiegsverfahren nach 50 Iterationen deutlich schlechtere Ergebnisse liefert als die Implementierung mit der Formfunktion. Hingegen ist das nicht-lineare cg-Verfahren nach 50 Schritt schon fast im optimal Zustand. Es werden nur noch kleinere Artefakte im Inneren des Gehirns nicht exakt approximiert. Gleiches kann auch festgestellt werden, wenn die Energie in Abbildung 5.6 verglichen wird. Das nicht-lineare cg-Verfahren erreicht nach wenigen Schritten einen besseren Energiezustand als die Verfahren mit Formfunktionen, die auf einem höheren Energielevel in ein Gleichgewichtszustand übergehen. Jedoch konvergiert das Gradientenabstiegsverfahren mit Formfunktionen deutlich schneller als die Implementierung mit der Heaviside-Funktion. Dies kann unter anderem daran liegen, dass die Glättung der Heaviside-Funktion den Gradienten an den Kanten verfälscht.

Als zweites kurzes Beispiel schauen wir uns den Kameramann in Abbildung 5.7 an. Dieses Bild ist deutlich größer als unser erstes Beispiel, sodass wir vor allem die Laufzeit

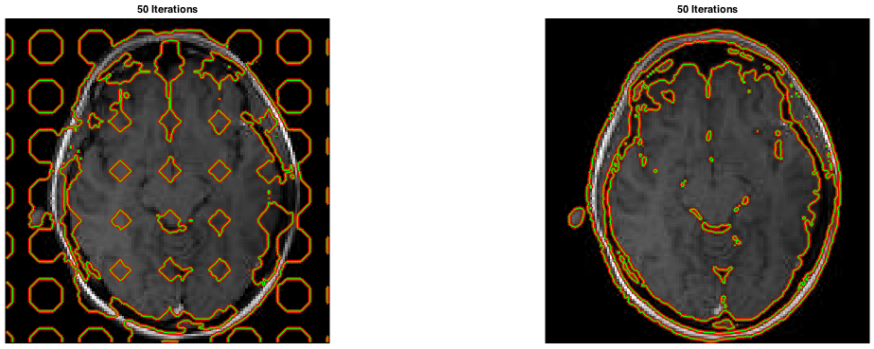

Abbildung 5.5.: Ergebnis nach 50 Iterationen. Links Gradientenabstieg und rechts nicht-lineares cg-Verfahren mit Heaviside Funktion.

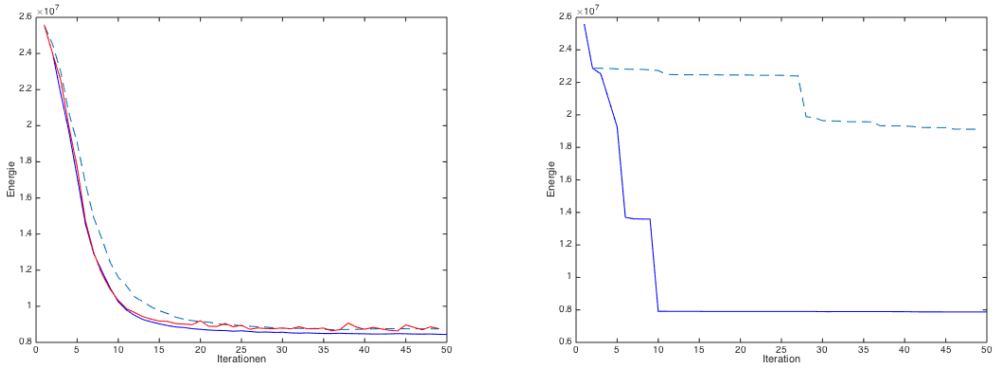

Abbildung 5.6.: Links die Energie für Formfunktionen und rechts für die Heaviside-Funktionen. Gestrichelt das Gradientenabstiegsverfahren, blau das nicht-lineare cg-Verfahren und rot das L-BFGS-Verfahren.

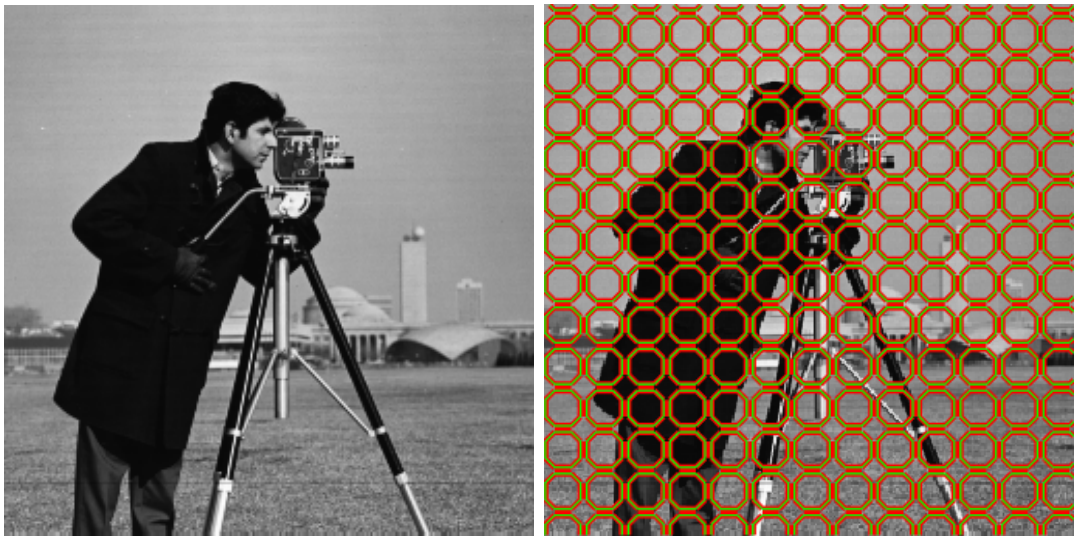

Abbildung 5.7.: Links ein Kameramann und rechts mit initialer Level-Set Funktion.

Laufzeit	Gradientenabstieg	ncg-Verfahren	L-BFGS
Formfunktion	17.918 s	30.909 s	16.574 s
Heaviside	36.773 s	64.399 s	34.948 s

Tabelle 5.2.: Mittelung aus drei Messungen für Gradietenabstieg, ncg-Verfahren und L-BFGS für die Heaviside-Funktion.

besser vergleichen können.

Was vor allem bei großen Bildern mit homogenen Gebieten auffällt ist, dass die Verfahren mit Formfunktionen eine anfänglich bessere Konvergenz liefern. In Abbildung 5.8 ist dies gut zu erkennen. Während das L-BFGS Verfahren mit der Heaviside Funktion nach 25 Iterationen noch keine Segmentierung geliefert hat, hat das Verfahren mit Formfunktionen schon eine gute Segmentierung berechnet.

In Abbildung 5.8 ist die Energie in Abhängigkeit von der Anzahl der Iterationen für die Implementierung mit den Formfunktionen zu sehen. Zu Beginn sind die nicht-linearen Verfahren schneller als der Gradientenabstieg, jedoch verhindert im weiteren Verlauf der Diskretisierungsfehler die bessere Konvergenz.

In Abbildung 5.9 sieht man die Konvergenz. Vor allem in homogenen Gebieten liefert das Verfahren schnell gute Ergebnisse. Lediglich in inhomogenen Gebieten sind einige Iterationen nötig, um genau zu segmentieren, wie gut am Gras zu erkennen ist.

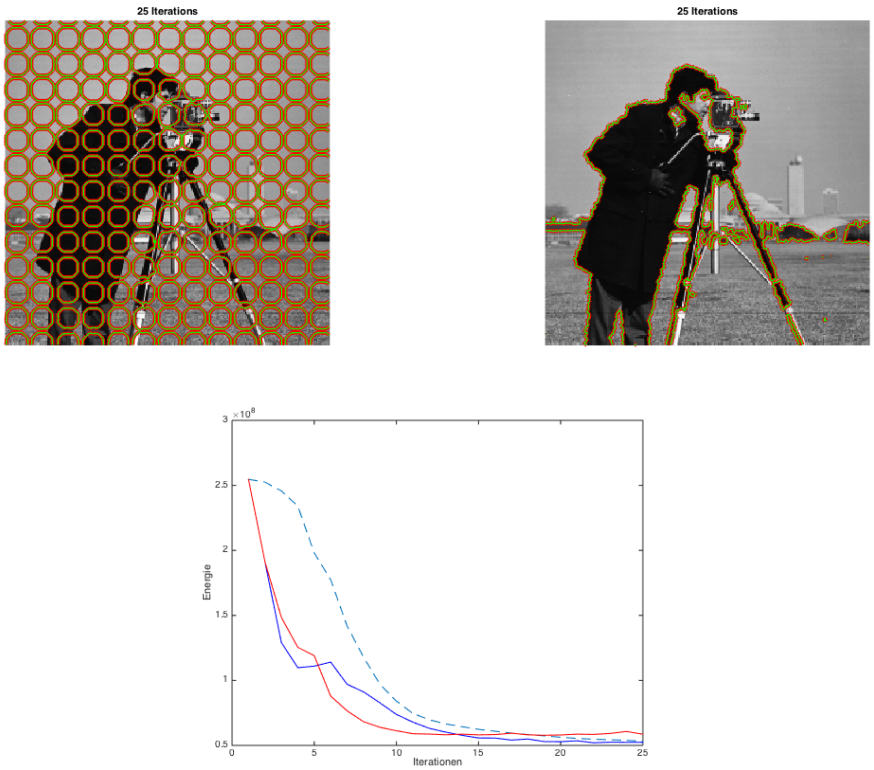

Abbildung 5.8.: Nach 25 Iterationen: oben links für die Heaviside-Funktion, gestrichelt Gradietenabstieg, blau das ncg-Verfahren und in rot L-BFGS. Rechts die Lösung des L-BFGS Algorithmus. Unten die Energie der beiden Verfahren im Vergleich zum gestrichelten Gradientenabstieg.

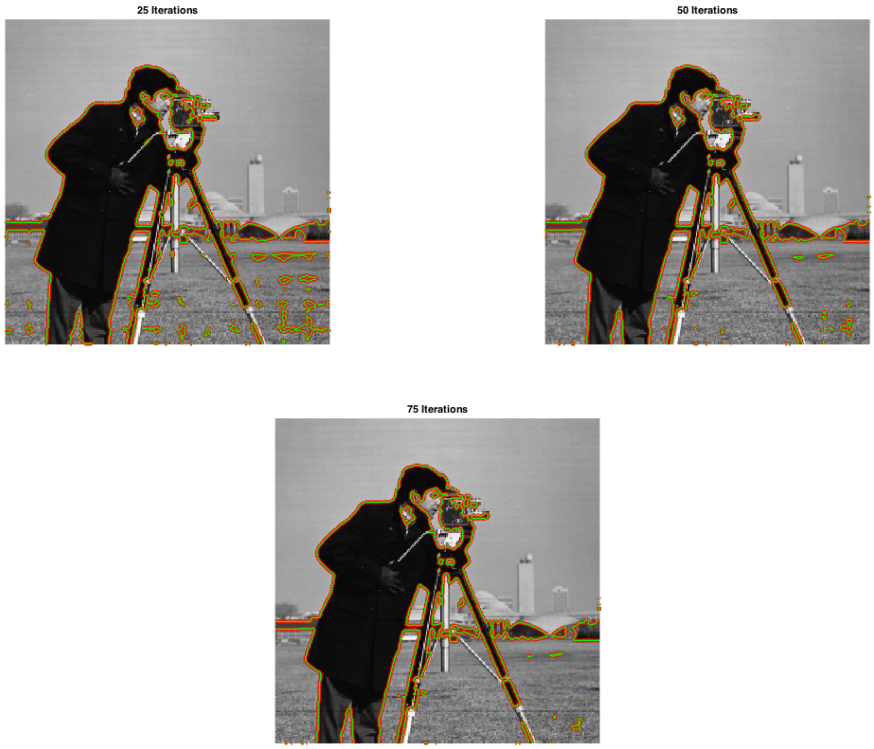

Abbildung 5.9.: Verlauf der Iterationen für den Gradientenabstieg mit Formfunktionen.

5.3. Zusammenfassung

Wie wir in der numerischen Analyse der Algorithmen gesehen haben, gibt es Vorteile und Nachteile für beiden Implementierungen.

Die Implementierung mithilfe von Formfunktionen ist einfacher als die mit der Heaviside-Funktion, da keine Indikatorfunktion genutzt werden muss und die Verfahren direkt auf das Funktional mit Formableitung angewendet werden können. Lediglich die Projektion P muss implementiert werden, welche aber durch das belegen des Rechengebietes mit der signierten Distanzfunktion einfach zu bewerkstelligen ist. Vor allem für das Gradientenabstiegsverfahren lohnt sich diese Erweiterung. Würden die im Abschnitt 5.2 angesprochenen Probleme, vor allem die kontinuierliche Fortsetzung für die Projektion zwischen Zellen, beseitigt, sollten vor allem die Verfahren höherer Ordnung eine bessere Konvergenz liefern. Jedoch müssen im Vergleich zur Heaviside-Funktion Abstriche bezüglich der Laufzeit gemacht werden, da die Bestimmung der projizierten Pixel Zeit beansprucht. Dennoch ist jetzt schon die anfängliche Konvergenz bei Formfunktionen besser als bei den Verfahren mit der Heaviside-Funktion.

6. Fazit und Ausblick

In dieser Arbeit haben wir Level-Set Funktionen mit Formfunktionen für nicht-lineare Verfahren kombiniert. Dazu haben wir im ersten Teil die nötigen Begriffe aus der Bildverarbeitung erläutert und die Level-Set Funktionen in Zusammenhang mit der Bildverarbeitung im speziellen der Bildsegmentierung gebracht. Des Weiteren haben wir die grundlegenden Eigenschaften der Level-Set Funktion erläutert und die drei Verfahren *Gradientenabstieg*, *nicht-lineares cg-Verfahren* und *Quasi-Newton-Verfahren* vorgestellt. Im Anschluss haben wir die nötigen Sätze aus der Shape Sensitivity Analysis hergeleitet, die die analytischen Grundlagen für die Formoptimierung und in unserem Fall vor allem eine Definition für die Formableitung liefern. Abschließend haben wir den Struktursatz bewiesen, der zeigt, dass die Formableitung auf dem Rand der Form existiert, so wie wir es erwarten würden.

Im zweiten Teil der Arbeit haben wir die nötigen Verbindungen zwischen Level-Set Funktionen, Formoptimierung und Algorithmen geschaffen. Dazu haben wir schrittweise die Algorithmen angepasst und uns vor allem dem Problem der Erweiterung auf das gesamte Rechengebiet gestellt. Dazu haben wir eine Projektion P eingeführt, die die Formableitung auf das gesamte Gebiet erweitert, indem ein Referenzpixel auf dem Rand in Normalenrichtung zu jedem Pixel bestimmt wird. Die Projektion P nutzen wir auch, um Informationen aus alten Iterationsschritten, die im nicht-linearen cg-Verfahren und im Quasi-Newton-Verfahren nötig sind, zu transportieren. Diese würden wie die Formableitung nur auf dem Rand existieren und im speziellen nur auf dem Rand der vorherigen Iteration.

Im letzen Teil haben wir die Verfahren sowohl untereinander als auch mit ihrer klassischen Implementierung mit der Heaviside-Funktion verglichen. Bezüglich der Laufzeit konnten die Verfahren die mit der Formableitung implementiert werden keine Verbesserung erzielen, da vor allem die Projektion der Pixel Rechenzeit beansprucht. Zudem kommt es zu Diskretisierungsfehlern, sodass die Verfahren nach kurzer Zeit in einen stationären Zustand übergehen. Dies liegt vor allem daran, dass beispielsweise zur Bestimmung der Normalen oder Projektion jeweils nur das nächstliegende

Pixel gewählt wird, sodass gleiche Veränderungen in der zugrundeliegenden Level-Set-Funktion Sprünge auslösen. Jedoch ist die Implementierung einfacher zu handhaben, da direkte Funktionale mit ihrer Ableitung genutzt werden können. Es muss lediglich die Projektion implementiert werden, welches durch die signierte Distanzfunktion begünstigt wird. Zudem ist die Konvergenz zu Beginn deutlich besser, sodass schon in wenigen Schritten eine gute Segmentierung berechnet werden kann.

Vor allem die Beseitigung von Diskretisierungsfehlern würde die Konvergenz der Verfahren mit Formfunktionen verbessern. Zudem könnte die Wahl der Projektion P verändert werden und andere Ansätze genutzt werden. Zum Beispiel könnte jeweils rückwirkend der alte referenzierte Punkt auf dem Rand ermittelt werden. Zudem wäre es möglich, die Implementierung für andere Funktionale testen. Hierbei sind natürlich auch Anwendungen außerhalb der Bildsegmentierung möglich. Dabei sollte wieder ein Augenmerk auf die Erweiterung der Formableitung auf das gesamte Rechengebiet liegen.

A. Matlab-Code

Im folgenden werden kurz die genutzten Programme vorgestellt. Die gesamte Implementierung mit Testfunktionen kann der beigefügten CD entnommen werden.

A.1. Implementierung mit Formfunktionen

backtrack.m

```
% Backtracking Line-Search mit Armijo-Goldstein Bedingung
% Input:
% x : als Matrix
% d : Abstiegsrichtung
% f : Funktion die zu minimieren ist
% gradf : Gradient dieser Funktion in jedem Punkt als Matrix
% mark : 0,1; Mark gibt an, ob überprüft werden soll, ob die Abstiegsrichtung
% "echt" ist.
%
% Output:
% Alpha : Schrittweite; -1 falls d keine Abstiegsrichtung

function alpha = backtrack(x,d,f,gradf,mark)
```

bfgs.m

```
% BFGS Algorithmus zur Minimierung
% Input:
% f : Funktion die minimiert werden soll
% gradf : Gradient von f
% x0 : Startwert für die Iteration
% eps : Epsilion eps für das Abbruchkriterium norm(grad) < eps
% P : Das Bild was Segmentiert wird; nur zur Darstellung nach jeder Iteration
% invMethode: Marker; 0 falls Hessematrix direkt bestimmt werden soll, 1
% falls die inverse Hessematrix bestimmt werden soll
%
```

```
11 % Output:
12 % x : approximiertes Minimum

14 function x = bfgs(f,gradf,x0,eps,P,invMethode)
```

buildVelocity.m

```
1 % Berechnet das Vektorfeld V für eine gegebene signierte Distanzfunktion
2 % Input:
3 % phi : Level-Set Funktion
4 % shapeDev : Formableitung in jedem Punkt
5 %
6 % Output :
7 % Vx, Vy : Vektorfeld für x und y Komponente

9 function [Vx, Vy] = buildVelocity( phi, shapeDev )
```

energy.m

```
1 % Chan-Vese Energiefunktional
2 % Input:
3 % phi0 : Level-Set Funktion
4 % mu : Gewichtung für Regularisierung
5 % P : zu segmentierendes Bild
6 %
7 % Output:
8 % x : Energie für phi0 in jedem Pixel

10 function x = energy(phi0,mu,P)
```

gradDesc.m

```
1 % Einfacher Gradientenabstieg für Formfunktionen
2 % Input:
3 % f : Funktion die zu minimieren ist
4 % gradf : Gradient von f
5 % x0 : Startwert
6 % eps : Abbruchkriterium norm(grad) < eps
7 % P : Bild; nur zur Darstellung in jeder Iteration
8 % sol : Lösung des Minimierungsproblems
9 %
10 % Output:
11 % x : approximiertes Minimum
12 % error : Fehler in jedem Schritt
13 % energy : Energie in jedem Schritt
```

```
function [phi, error, energy] = gradDesc( phi0,f,gradf,eps,P, sol )
phi = phi0;
shape = feval(gradf,phi);
iter = 1;

while (norm(shape) > eps && iter < 100)
    error(iter) = norm(phi-sol);
    energy(iter) = feval(f,phi);
    shape = feval(gradf,phi);

    [Vx, Vy] = buildVelocity(phi,shape);

    solve = solvePDE(phi,Vx, Vy, P);

    phi = reinit_SD(solve,0.5,10);

    showphi(P,phi,iter);
    iter = iter + 1;
end
end
```

lmbfgs.m

```
% Limited-Memory BFGS zur Minimierung
% Input:
% f : Funktion die zu minimieren ist
% gradf : Gradient von f in jedem Punkt
% x0 : Startwert
% eps : Parameter für das Abbruchkriterium norm(grad) < eps
% P : Bild; nur zur Datstellung in jeder Iteration
% restartIter : nach wievielen Iterationen die Hessenmatrix neu
% angesetzt werden soll
% sol : Lösung des Minimierungsproblems
%
% Output :
% x : approximiertes Minimum
% error : Fehler in jedem Schritt
% energy : Energie in jedem Schritt
% Orientiert an https://en.wikipedia.org/wiki/Limited-memory_BFGS

function [x, error, energy] = lmbfgs(f,gradf,x0,eps,P,restartIter,sol)

k = 1;      % Iterationszähler
x = double(x0(:));
```

```
22 g = feval(gradf,reshape(x,size(P)));
23 g = -g(:);
24 error(1) = norm(reshape(x,size(P))-sol);
25 energy(1) = feval(f,reshape(x,size(P)));

27 m = 0;        % Anzahl der zwischengespeicherten Informationen
28 %%% erste Iteration
29 x_old = x;
30 solve = solvePDEnormal(reshape(x,size(P)),reshape(g,size(P)),P);
31 x = reinit_SD(solve,0.5,10);
32 x = x(:);

34 k = k + 1;
35 %%%

37 %%Hauptschleife
38 while (norm(g) > eps && k < 26)
39     error(k) = norm(reshape(x,size(P))-sol);
40     energy(k) = feval(f,reshape(x,size(P)));
41     g_old = g;

43     g = feval(gradf,reshape(x,size(P)));
44     g = g(:);
45     % benötigte Zwischenergebnisse speichern
46     movedS = movedNormal(reshape(x - x_old,size(P)),reshape(x,size(P)));
47     movedY = movedNormal(reshape(g - g_old,size(P)),reshape(x,size(P)));
48     s(k-1,:) = movedS(:);
49     y(k-1,:) = movedY(:);
50     for ii = k-m:k-2
51         movedS = movedNormal(reshape(s(ii,:),size(P)),reshape(x,size(P)));
52         movedY = movedNormal(reshape(y(ii,:),size(P)),reshape(x,size(P)));
53         s(ii,:) = movedS(:);
54         y(ii,:) = movedY(:);
55     end
56     m = m + 1;
57     if m > restartIter
58         m = 0;
59     end

61     % H^-1 rekonstruieren
62     q = -g;
63     alpha = zeros(m-1);
64     for i=k-1:-1:k-m
65         p = 1 / (y(i,:) * s(i,:)');
```

```
66          alpha(i) = p * (s(i,:) * q);
67          q = q - alpha(i) * y(i,:)';
68      end
69      % H^-1 aufbauen
70      if m > 0
71          H = (y(k-1,:) * s(k-1,:)') / (y(k-1,:) * y(k-1,:)');
72      else
73          H  = 1;
74      end
75      z = H * q;

77      % Gleichungssystem lösen
78      beta = zeros(m-1);
79      for i = k-m:k-1
80          p = 1 / (y(i,:) * s(i,:)');
81          beta(i) = p * (y(i,:) * z);
82          z = z + (s(i,:)'*(alpha(i)-beta(i)));
83      end
84      % Schritt machen mit Abstiegsrichtung z
85      x_old = x;
86      if (z' * g > 0)
87          m = -1;
88          disp(['Keine Abstiegsrichtung in Schritt ' num2str(k)])
89          solve = solvePDEnormal(reshape(x,size(P)),-reshape(g,size(P)),P);
90      else
91          solve = solvePDEnormal(reshape(x,size(P)),reshape(z,size(P)),P);
92      end

94      x = reinit_SD(solve,0.5,10);
95      x = x(:);
96      %x = x + a*z;

98      showphi(P,reshape(x,size(P)),k);
99      k = k + 1;
100     %energy1(reshape(x,size(P)),0.02,P)
101 end
102 end
```

movedNormal.m

```
1  % Projektion P
2  % Input:
3  % toMove : was bewegt werden soll
4  % phi : zugrundeliegende Level-Set Funktion
5  %
```

```
% Output:
% moved : projezierte Werte

function moved = movedNormal( toMove, phi )
% Initalisierung
dx = 1;
[k,l] = size(toMove);
moved = zeros(k,l);

for i = 1:k-1
    for j = 1:l-1
        % Bestimme Referenzpixel
        normale(1) = (phi(i+1,j) - phi(i,j))/dx;
        normale(2) = (phi(i,j+1) - phi(i,j))/dx;
        moved_i = i - round(phi(i,j) * normale(1));
        moved_j = j - round(phi(i,j) * normale(2));
        if (moved_i < 1)
            moved_i = 1;
        end
        if (moved_i > k-1)
            moved_i = k-1;
        end
        if (moved_j < 1)
            moved_j = 1;
        end
        if (moved_j > l-1)
            moved_j = l-1;
        end

        % Wende Projektion an
        moved(i,j) = toMove(moved_i,moved_j);
    end
end
end
```

nlcg.m

```
% Nicht-lineares cg-Verfahren zur Minimierung
% Input:
% f : Funktion die zu minimieren ist
% gradf : Gradient von f
% x0 : Startwert
% eps : Abbruchkriterium norm(grad) < eps
% P : Bild; nur zur Darstellung in jeder Iteration
% sol : Lösung des Minimierungsproblems
```

```
9  %
10 % Output:
11 % x : approximiertes Minimum
12 % error : Fehler in jedem Schritt
13 % energy : Energie in jedem Schritt
14 % Orientiert an http://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method

16 function [x, error, energy] = nlcg(f,gradf,x0,eps,P,sol)
17 % Initalisierung
18 x = double(x0);
19 itere = 1;

21 % Erster Schritt
22 grad = feval(gradf,x);
23 d=-grad(:);
24 s = d;
25 error(1) = norm(x-sol);
26 energy(1) = feval(f,x);
27 x = solvePDEnormal(x,reshape(s,size(P)),P);
28 x = reinit_SD(x,0.5,10);
29 showphi(P,x,itere);

31 % Hauptschleife
32 while (norm(grad) > eps && itere < 25)
33     itere = itere +1;
34     error(itere) = norm(x-sol);
35     energy(itere) = feval(f,x);



38     d_old=d;
39     s_old=s;

41     grad = feval(gradf,x);
42     d=-grad(:);

44     beta=(d'*(d-d_old))/(d_old'*d_old); % beta_n Polak Ribiere
45     %beta=(d'*d)/(d_old'*d_old); %beta_n Fletcher Reeves
46     s=d+beta*s_old;
47     x = solvePDEnormal(x,reshape(s,size(P)),P);
48     x = reinit_SD(x,0.5,10);
49     showphi(P,x,itere);
50 end
51 end
```

shapeDerMoved.m

```
% Erweiterte Formableitung auf das gesamte Rechengebiet
% Input:
% phi0 : Level-Set Funktion
% P : zu segmentierendes Bild
%
% Output :
% x : ausgewertete und projezierte Formableitung auf dem gesamten Gebiet

function x = shapeDerMoved(phi0,P)
    % Bestimme Formableitung
    eps = 0;
    dx = 1;
    inidx = find(phi0>=0); % Vordergrund index
    outidx = find(phi0<0); % Hintergrund index

    c1 = sum(sum(P.*Heaviside(phi0)))/(length(inidx)+eps); % Durchschnitt in Phi0
    c2 = sum(sum(P.*(1-Heaviside(phi0))))/(length(outidx)+eps); % außerhalb

    int1 = (P - c1).^2;
    int2 = (P - c2).^2;

    kru = kappa(phi0);

    x = int2 - int1 + kru;
   [k,l] = size(x);

for i = 1:k
    for j = 1:l
        % Referenzpixel bestimmen
        if (i == k)
            normale(1) = (-phi0(i,j))/dx;
        else
            normale(1) = (phi0(i+1,j) - phi0(i,j))/dx;
        end
        if (j == l)
            normale(2) = (-phi0(i,j))/dx;
        else
            normale(2) = (phi0(i,j+1) - phi0(i,j))/dx;
        end
        moved_i = i - round(phi0(i,j) * normale(1));
        moved_j = j - round(phi0(i,j) * normale(2));

        if (moved_i < 1)
```

```
44                 moved_i = 1;
45             end
46             if (moved_i > k)
47                 moved_i = k;
48             end
49             if (moved_j < 1)
50                 moved_j = 1;
51             end
52             if (moved_j > l)
53                 moved_j = l;
54             end

56             % Projektion anwenden
57             x(i,j) = x(moved_i,moved_j);
58         end
59     end

61 end
```

solvePDEnormal.m

```
1  % Upwind-Verfahren für Normalenvektorfeld V
2  % Input :
3  % phi : Level-Set Funktion
4  % Vnormale : Vektorfeld in Normalenrichtung
5  % P : Bild
6  %
7  % Output :
8  % solve : Lösung der PDGL
9  % Orientiert an https://en.wikipedia.org/wiki/Upwind_scheme

11 function solve = solvePDEnormal( phi, Vnormale , P)
```

A.2. Implementierung mit der Heaviside-Funktion

backtracking.m

```
1  % Backtracking Line-Search mit Armijo-Goldstein Bedingung
2  % Input:
3  % x : als Matrix
4  % d : Abstiegsrichtung
5  % f : Funktion die zu minimieren ist
```

```
% gradf : Gradient dieser Funktion in jedem Punkt als Matrix
% mark : 0,1; Mark gibt an, ob überprüft werden soll, ob die Abstiegsrichtung
% "echt" ist.
%
% Output:
% Alpha : Schrittweite; -1 falls d keine Abstiegsrichtung

function alpha = backtrack(x,d,f,gradf,mark)
```

bfgs.m

```
% BFGS Algorithmus zur Minimierung
% Input:
% f : Funktion die minimiert werden soll
% gradf : Gradient von f
% x0 : Startwert für die Iteration
% eps : Epsilion eps für das Abbruchkriterium norm(grad) < eps
% P : Das Bild was Segmentiert wird; nur zur Darstellung nach jeder Iteration
% invMethode: Marker; 0 falls Hessematrix direkt bestimmt werden soll, 1
% falls die inverse Hessematrix bestimmt werden soll
%
% Output:
% x : approximiertes Minimum

function x = bfgs(f,gradf,x0,eps,P,invMethode)
```

energy.m

```
% Chan-Vese Energiefunktional
% Input:
% phi0 : Level-Set Funktion
% mu : Gewichtung für Regularisierung
% P : zu segmentierendes Bild
%
% Output:
% x : Energie für phi0 in jedem Pixel

function x = energy(phi0,mu,P)
```

graddesc.m

```
% Einfacher Gradientenabstieg
% Input:
% f : Funktion die zu minimieren ist
% gradf : Gradient von f
```

```
% x0 : Startwert
% eps : Abbruchkriterium norm(grad) < eps
% P : Bild; nur zur Darstellung in jeder Iteration
% sol : Lösung des Minimierungsproblems
%
% Output:
% x : approximiertes Minimum
% error : Fehler in jedem Schritt
% energy : Energie in jedem Schritt

function [x, error, energy] = graddesc( f,gradf,x0,eps,P, sol )
```

gradientImageSeg.m

```
% Gradient des Chan-Vese Energiefunktionals
% Input:
% phi0 : Level-Set Funktion
% mu : Gewichtung für Regularisierung
% P : zu segmentierendes Bild
%
% Output:
% x : Gradient für phi0 in jedem Pixel

function j = gradientImageSeg(phi0, mu, P)
```

lmbfgs.m

```
% Limited-Memory BFGS zur Minimierung
% Input:
% f : Funktion die zu minimieren ist
% gradf : Gradient von f in jedem Punkt
% x0 : Startwert
% eps : Parameter für das Abbruchkriterium norm(grad) < eps
% P : Bild; nur zur Datstellung in jeder Iteration
% restartIter : nach wievielen Iterationen die Hessenmatrix neu
% angesetzt werden soll
% sol : Lösung des Minimierungsproblems
%
% Output :
% x : approximiertes Minimum
% error : Fehler in jedem Schritt
% energy : Energie in jedem Schritt
% Orientiert an https://en.wikipedia.org/wiki/Limited-memory_BFGS

function [x, error,energy] = lmbfgs(f,gradf,x0,eps,P,restartIter, sol)
```

maskcircle2.m

```
% Startwert-Erstellung von Level-Set Funktionen
% Input:
% I : Bild
% Type : Typ der Maske die erstellt werden soll (small, medium, large, whole)
%
% Output:
% m : Level-Set Funktion des angegebenen Typen
% Quelle : http://www.mathworks.com/matlabcentral/fileexchange/
% 23445-chan-vese-active-contours-without-edges/content/maskcircle2.m

function m = maskcircle2(I,type)
```

nlcg.m

```
% Nicht-lineares cg-Verfahren zur Minimierung
% Input:
% f : Funktion die zu minimieren ist
% gradf : Gradient von f
% x0 : Startwert
% eps : Abbruchkriterium norm(grad) < eps
% P : Bild; nur zur Darstellung in jeder Iteration
% sol : Lösung des Minimierungsproblems
%
% Output:
% x : approximiertes Minimum
% error : Fehler in jedem Schritt
% energy : Energie in jedem Schritt
% Orientiert an http://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method

function [x, error, energy] = nlcg(f,gradf,x0,eps,P, sol)
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Allaire, G. ; Dapogny, C. ; Frey, P. : Shape optimization with a level set based mesh evolution method. In: *Computer Methods in Applied Mechanics and Engineering* (2014)

[2] Armijo, L. : Minimization of Functions having Lipschitz Continuous First Partial Derivatives. In: *Pacific Jorunal of Mathematics* 16 (1966), Nr. 1 12

[3] Astolfi, A. : *Optimization An introduction* 11

[4] Bredies, K. ; Lorenz, D. : *Mathematische Bilderverarbeitung*. Springer Vieweg, 2011 3

[5] Chan, T. F. ; Vese, A. : Active Contours Without Edges. (2001) 4, 5, 38, 60

[6] Delfour, M. ; Zolesio, J.-P. : *Shapes and Geometries*. SIAM, 2011 17, 22, 25

[7] Fronasier, M. ; March, R. : Existence of Minimizers of the Mumford-Shah Functional With Singular Operators in Two Space Dimensions. 5

[8] Königsberger, K. : *Analysis 2*. Springer, 2002 26

[9] Liu, D. C. ; Nocedal, J. : On The Limited Memory BFGS Method For Large Scale Optimization. In: *Mathematical Programming 45* (1989) 34

[10] Luenberger, D. G.: *Optimization by Vector Space Methods*. Wiley Interscience, 1969 32

[11] Mumford, D. ; Shah, J. : Optimal approximation by piecewise smooth functions and associated variational problems. In: *Communications on Pure and Applied Mathematics* (1989)

[12] Neslihan, Ö. : *Image Segmentation And Smoothing Via Partial Differential Equations*, Diplomarbeit, 2009 9

[13] Nocedal, J. ; Wright, S. J.: *Numerical Optimization*. Springer, 1999 16

[14] Pach, M. : *Levelsetverfahren in der Shapeoptimierung*, Universität Duisburg-Essen, Diplomarbeit, 2005 17, 20, 36, 60

[15] Schraudolph, N. N. ; Yu, J. ; Günter, S. : A Stochastic Quasi-Newton Method for Online Convex Optimization. 34

[16] Shapiro, A. : On Concepts of Directional Differentiability. In: *Journal of Optimization Theory and Applications* 66 (1990), Nr. 3 17

[17] Sokolowski, J. ; Zolesio, J.-P. : *Introduction to Shape Optimization*. Springer, 1998 17, 28

[18] Wissenschaft Verlagsgesellschaft mbH, S. der: *http://www.wissenschaft-schulen.de/alias/material/hoehenkarte-der-hauptinsel-von-hawai/1063530* 1, 60