UNIVERSITÄT ZU LÜBECK
INSTITUTE OF MATHEMATICS AND
IMAGE COMPUTING

Anwendung des Nesterov-Algorithmus in der Bildregistrierung

Application of Nesterovs-algorithm in Image Registration

Bachelorarbeit

im Rahmen des Studiengangs
Mathematik in Medizin und Lebenswissenschaften
der Universität zu Lübeck

vorgelegt von
Sina Ueberberg

ausgegeben und betreut von
Prof. Dr. Jan Modersitzki
Institute of Mathematics and Image Computing

mit Unterstützung von
Thomas Polzin M. Sc.
Institute of Mathematics and Image Computing

Lübeck, den 05.November 2014

MIC

IM FOCUS DAS LEBEN

Eigenständigkeitserklärung

Erklärung
Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Lübeck, 05. November 2014

Sina Ueberberg

Zusammenfassung

In dieser Arbeit geht es um Nesterov-Algorithmen mit der Anwendung in der Bildregistrierung. Es wurden in dieser Arbeit vier Nesterov-Algorithmen implementiert und mit zwei akademischen Beispielen und fünf Beispieldatensätzen aus FAIR anhand verschiedener Merkmale hinsichtlich ihrer Performanz und Genauigkeit bezüglich eines Vergleichsminimierers bei den akademischen Beispielen oder eines Referenzbildes bei den Beispieldatensätzen verglichen. Um die Ergebnisse bezüglich der Laufzeiten vergleichbar zu machen, wurden außerdem die Ergebnisse der Standardverfahren, wie dem Newton- und dem Gradientenverfahren dargestellt und auch ausgewertet. Am Ende stellte sich heraus, dass bei den hier getesteten Nesterov-Algorithmen das Nesterov-Constant-Step-Scheme 1 am schnellsten und genauesten, das Nesterov-General-Scheme allerdings bei den eben genannten Kriterien am schlechtesten war.

Abstract

This thesis focus is on the application of Nesterov algorithms in image registration. Four Nesterov-algorithms were implemented and the algorithms were evaluated with two academical examples and five datasets from FAIR. The results were compared to the analytic minimizer and for the datasets with a reference picture. Furthermore the results were compared with standard methods like the Newton method and the Gradient method concerning its performance and its accuracy regarding. It could be observed, that the Nesterov-Constant-Step-Scheme 1 shows the most precise and fastest results but the Nesterov-General-Scheme shows the least precise results.

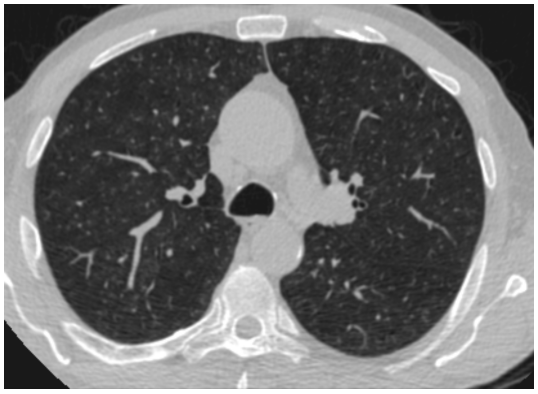
Inhaltsverzeichnis

1 Einleitung

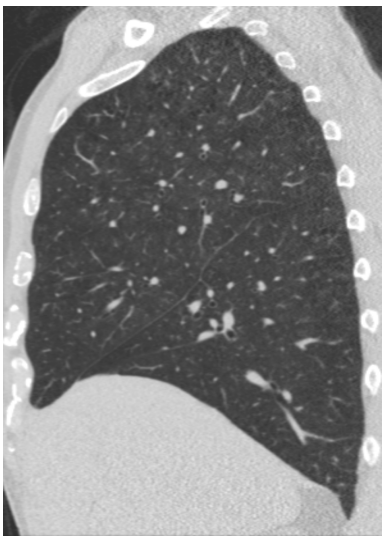
In dieser Arbeit werden verschiedene von Yuri Nesterov entwickelte Optimierungsalgorithmen und deren Anwendung in der Bildregistrierung untersucht. Diese werden mit den Standardverfahren der unregistrierten Optimierung [12], wie dem Newton-Verfahren oder dem Gradienten-Verfahren hinsichtlich ihrer Laufzeiten, ihrer Genauigkeiten und ihrer Performanz verglichen. Am Ende wird herausgestellt, dass das Nesterov-Constant-Step-Scheme 1 anhand der in dieser Arbeit verwendeten Datensätze eine Alternative zu den Standardverfahren ist. Das Nesterov-Constant-Step-Scheme 2 und das Nesterov-Verfahren nach [11] lieferten bei den hier getesteten Beispielen schwankende Ergebnisse und das Nesterov-General-Scheme war bei den Datensätzen aus FAIR die Methode, die am wenigsten Reduzierung erreichte. Die Vorteile und Probleme der Algorithmen werden in der Auswertung erläutert.

1.1 Motivation - Suche nach neuen Verfahren in der medizinischen Bildregistrierung

In der medizinischen Bildregistrierung werden zwei oder mehr Bilder des gleichen Objektes, die zu verschiedenen Zeitpunkten, an verschiedenen Orten, aus verschiedenen Blickwinkeln oder von verschiedenen Sensoren aufgenommen wurden, aufeinander ausgerichtet [3]. Die medizinische Bildgebung wurde durch die Einführung tomographischer Bildgebungstechniken, wie Computer-Tomographie (CT) und Magnetresonanz-Tomographie (MRT) in den 70er Jahren stark weiterentwickelt, hat ihre Ursprünge aber 1895 mit der Entdeckung der Röntgenstrahlen [6]. Mittlerweile können durch Techniken, wie CT und MRT Bilder im Submillimeterbereich aufgelöst werden [6]. Das bedeutet, dass zum Beispiel ein 3D-CT-Bild des Thorax aus ungefähr $400^3 = 64000000$ Voxeln besteht. Dies bedeutet einen hohen Rechen- und Speicheraufwand, da man ein Gleichungssystem mit 64000000 Variablen lösen muss. Die Registrierung solcher Datensätze ist ein Bestandteil des EMPIRE10-Wettbewerbs [9]. EMPIRE10 steht für Evaluation of Methods for Pulmonary Image REgistration 2010. Der EMPIRE10-Wettbewerb ist eine öffentliche Plattform zur Evaluierung von Registrierungs-Algorithmen für Lungen-CT-Bilder. Es wurden zahlreiche Algorithmen von verschiedenen Forschungsgruppen verglichen. Sie mussten 30 Datensätze registrieren, die jeweils von einem Patienten stammten. Die Ergebnisse sind auf der EMPIRE10-Website (http://empire10.isi.uu.nl)öffentlich einsehbar. Beispielhaft ist in Abbildung 1.1 der dritte Datensatz dieses Wettbewerbs zu sehen. Hierbei wird die sagittale (seitliche), die axiale (obere) und



(a) Axiale Ansicht



(b) Sagittale Ansicht

(c) Koronale Ansicht

Abbildung 1.1: CT-Bild eines Thorax der axialen, der sagittalen und der koronalen Ansicht [9]. Es handelt sich um den dritten Datensatz des EMPIRE10-Wettbewerbs.

die koronale (vordere) Ansicht dargestellt [9]. Dieser Datensatz hat eine Größe von $(412 \times 296 \times 458)$ Voxeln mit einer Auflösung von $0.748 \times 0.748 \times 0.7$ mm^3. Auf der vollen Auflösung müsste für eine deformierbare Registrierung ein Optimierungsproblem mit circa 56 Millionen Variablen gelöst werden, was viel Speicherplatz und einen hohen Rechenaufwand bedeutet.
Im Fachgebiet der Optimierung werden effiziente und schnelle Algorithmen entwickelt, um Optima von Zielfunktionen zu berechnen. Das Newton-Verfahren hat eine quadratische Konvergenzrate, benötigt aber zweite Ableitungen, was bei verrauschten und großen Datensätzen Probleme bereitet [8]. Weiterhin liefert es nur bei positiv definiten Hessematrizen sinnvolle Abstiegsrichtungen [12]. Approximationen, wie sie im Gauß-Newton-Verfahren verwendet werden, lindern diese Einschränkungen, verschlechtern aber unter Umständen die Konvergenzrate [8]. Auch das Gradienten-Verfahren, welches nur erste Ableitun-

gen benötigt, hat Vor- und Nachteile. Es benötigt viele Iterationen, wodurch eine lange Laufzeit entsteht. Dennoch benötigt es weniger Speicherplatz als das Newton-Verfahren, da die Hessematrix nicht benötigt wird, wodurch auch die Laufzeit je Iteration geringer ist [8]. Da dies in der medizinischen Bildregistrierung bei großen Bilddatensätzen wichtig ist, wird weiterhin nach weniger rechenintensiven Algorithmen gesucht. Eine Registrierungsmethode, die das Gradienten-Verfahren zur Optimierung verwendet, ist momentan führend im EMPIRE10-Wettbewerb. Dieses Verfahren heißt „gsyn“ und ist öffentlich über die ANTS-Software verfügbar [1, 13]. Eine Alternative zum Gradienten und Newton-Verfahren stellt das 1983 veröffentlichte Paper von Yuri Nesterov dar [11]. Es basiert ebenfalls auf Gradienten-Informationen, nutzt diese jedoch geschickter aus und erreicht somit eine (lokal) quadratische Konvergenzordnung [11]. Nesterov entwickelte seine Ideen weiter und schrieb sie in einem Buch [10] nieder. Diese Verfahren stellen in der Theorie eine Brücke zwischen Gradienten- und Newton-Verfahren dar, in der Praxis wurden in dieser Arbeit allerdings teilweise abweichende Ergebnisse erzielt.

1.2 Problemstellung und Ergebnisse

Die von Nesterov veröffentlichten Algorithmen wurden in dieser Arbeit zunächst implementiert und anschließend bezüglich der Anwendung in der Bildregistrierung evaluiert. Um einen möglichst repräsentativen Auswertungsprozess und eine Vielzahl von möglichen Szenarien abzudecken, wurden die Algorithmen mit dem Gradienten- und (Gauß-)Newton-Verfahren anhand von verschiedenen Datensätzen und Registrierungseinstellungen aus FAIR (Flexible Algorithm Image Registration) [8] und zwei akademischen Beispielen verglichen. Dabei wurden bei den akademischen Beispielen einmal die Voraussetzungen, die Nesterov trifft eingehalten und bei dem anderen Beispiel bewusst gebrochen. Dabei erkannte man Unterschiede zwischen den Verfahren. Am Ende zeigte sich, dass das Nesterov-Constant-Step-Scheme 1 die genauesten Ergebnisse lieferte und dabei meist eine mit dem Newton-Verfahren vergleichbare Laufzeit hatte. Bei dem 3D-Brain Datensatz wurde vom Nesterov-Constant-Step-Scheme 1 eine höhere Reduzierung erreicht, als beim Gauß-Newton-Verfahren, allerdings war die Laufzeit beim Constant-Step-Scheme 1 fast vier mal so lang, wie bei dem Gauß-Newton-Verfahren. Das Nesterov-General-Scheme zeigte einzig bei dem akademischen Beispiel eine gute Performanz. Das Nesterov-Verfahren nach [11] und das Nesterov-Constant-Step-Scheme 2 schwankten in ihren Ergebnissen.

1.3 Gliederung der Arbeit

Die Arbeit ist in folgende Bereiche gegliedert: In Kapitel 2 werden zunächst die analytischen Grundlagen, die Nesterov in seinem Paper [11] und seinem Buch [10] benötigt und darstellt, definiert. Danach werden die Optimierungsgrundlagen erläutert. Hierzu gehören die Standard-Optimierungsalgorithmen

wie das Newton-Verfahren oder das Gradienten-Verfahren. Im dritten Kapitel werden die von Nesterov beschriebenen Algorithmen, die in dieser Arbeit verwendet werden, vorgestellt. Dazu gehören der Algorithmus aus Nesterovs 1983 veröffentlichtem Paper [11], sowie das Nesterov-General-Scheme, das Nesterov-Constant-Step-Scheme 1 und das Nesterov-Constant-Step-Scheme 2 aus [10]. In den Experimenten, die in Kapitel 4 beschrieben sind, werden die verschiedenen Algorithmen anhand der Registrierung verschiedener Datensätze verglichen und die Ergebnisse dargestellt. Hierzu gehören zwei akademische Beispiele und fünf Bildregistrierungsbeispiele aus der Software FAIR [8]. In der Diskussion werden in Kapitel 5 die einzelnen Algorithmen miteinander verglichen und Schwächen sowie Stärken der einzelnen Algorithmen hervorgehoben. Im abschließenden Ausblick werden potenzielle Verbesserungsmöglichkeiten formuliert.

2 Grundlagen zur Erklärung der analytischen Hintergründe der Nesterov-Algorithmen und der Optimierungsgrundlagen dieser Arbeit

In diesem Kapitel werden zuerst die mathematischen Begriffe definiert, die Nesterov als Voraussetzungen trifft. Danach werden die Standardoptimierungsgrundlagen vorgestellt, die beim Vergleich mit den Nesterov-Algorithmen verwendet werden. In den analytischen Grundlagen (Abschnitt 2.1) wird unter anderem gezeigt, wie ein Konvexitätsparameter abgeschätzt werden kann und wie man die Lipschitzkonstante berechnet. In den Optimierungsgrundlagen (Abschnitt 2.2) wird erörtert, was ein Minimierer ist und wie die Standardverfahren, wie das Newton-Verfahren oder das Gradienten-Verfahren funktionieren.
Als Anfangsannahme gilt:
Wir haben das Problem (P):

Problem 2.1. *(P)*
Eine konvexe Funktion $f : \mathbb{R}^n \to \mathbb{R}$ ist zu minimieren. Das heißt:
Finde ein (x^*), so dass $f(x^*) \leq f(y)$ für alle $y \in \mathbb{R}^n$.

Im Folgenden wird genauer erläutert, was die Begriffe konvex und Minimierer bedeuten.

2.1 Analytische Grundlagen

Nesterov trifft im Paper [11] zunächst einige Voraussetzungen. Diese werden im Folgenden mathematisch definiert. Er beschreibt, dass die von ihm Entwickelten Algorithmen eine Lipschitzkonstante und einen Konvexitätsparameter μ benötigt. Für die Abschätzung dieser Parameter benötigt man die Definition eines Skalarproduktes und einer Norm. Dies wird im Folgenden definiert:

Definition 2.2. *(Skalarprodukt) [7, S. 45]*
Sei X ein $\mathbb{K}$-Vektorraum. Eine Abbildung $\langle \cdot, \cdot \rangle \colon X \times X \to \mathbb{K}$ heißt Skalarprodukt,

wenn

$$\left.\begin{aligned} &\forall x, y, z \in X, \quad \forall \lambda \in \mathbb{K}: \\ &\qquad \langle x, x\rangle \geq 0, \quad \langle x, x\rangle = 0 \Leftrightarrow x = 0 \\ &\qquad \langle \lambda x, y\rangle = \lambda \langle x, y\rangle \\ &\qquad \langle x, y\rangle = \overline{\langle y, x\rangle} \\ &\qquad \langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle. \end{aligned}\right\} \tag{2.1}$$

Definition 2.3. *(Norm) [7, S. 38]*
Sei X ein $\mathbb{K}$-Vektorraum. Eine Abbildung $\|\cdot\|\colon X \to \mathbb{R}$ heißt Norm, wenn

$$\left.\begin{aligned} &\forall x, y \in X, \quad \forall \lambda \in \mathbb{K}: \\ &\qquad \|x\| \geq 0, \quad \|x\| = 0 \Leftrightarrow x = 0 \\ &\qquad \|\lambda x\| = |\lambda| \|x\| \\ &\qquad \|x + y\| \leq \|x\| + \|y\|. \end{aligned}\right\} \tag{2.2}$$

Weiter heißt $(X, \|\cdot\|)$ normierter Raum.

Um die Optimalität in den akademischen Beispielen zu zeigen, kann man die Eigenwerte einer Matrix bestimmen und überprüfen, ob diese positiv oder negativ oder beides sind. Dafür wird zunächst definiert, was ein Eigenwert ist.

Definition 2.4. *(Eigenwerte) [4, S. 231]*
Sei $A \in \mathbb{R}^{n\times n}$ eine reelle quadratische Matrix. Ein Skalar $\lambda \in \mathbb{C}$, das

$$\det(A - I_n\lambda) \stackrel{!}{=} 0 \tag{2.3}$$

erfüllt, heißt Eigenwert von A. Dabei bezeichnet $I_n \in \mathbb{R}^{n\times n}$ die Einheitsmatrix mit $I_n = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$.

Satz 2.5. *(Eigenschaften von Eigenwerten)*

- *Es gibt maximal n verschiedene Eigenwerte.*
- *Falls die Matrix A symmetrisch ist, sind die Eigenwerte $\lambda_i \in \mathbb{R}, \quad i \in \{1, 2, \dots, n\}$.*

Über die Definitheit der Hessematrix kann gezeigt werden, dass ein Punkt x^* ein Minimierer ist. Dafür wird zunächst die Definitheit und danach die Hessematrix definiert.

Definition 2.6. *(Definitheit) [7, S. 207]*
Eine reelle, quadratische Matrix $A \in \mathbb{R}^{n\times n}$ heißt

1. *positiv definit, falls $\langle Ax, x\rangle > 0$ für alle $x \in \mathbb{R}^n \backslash \{0\}$ gilt.*

2. *positiv semidefinit, falls* $\langle Ax, x\rangle \geq 0$ *für alle* $x \in \mathbb{R}^n \backslash \{0\}$ *gilt.*
3. *negativ (semi-)definit, falls* $\langle Ax, x\rangle < 0$ *(bzw.* $\langle Ax, x\rangle \leq 0$*) für alle* $x \in \mathbb{R}^n \backslash \{0\}$ *gilt.*
4. *Eine Matrix heißt positiv (semi-)definit* $\Leftrightarrow \lambda_i > (\geq)\ 0$ *für alle* $i \in \{1, ..., n\}$ *sind. [4, S. 327].*

Definition 2.7. *(Raum der stetig-differenzierbaren Funktionen) [7, S. 205]*
$C^d(\mathbb{R}^n, \mathbb{R})$ *ist der Raum der d-mal stetig-differenzierbaren Funktionen mit Definitionsbereich* $\mathbb{R}^n$ *und Bildbereich* $\mathbb{R}$*. Für* $f \in C^d(\mathbb{R}^n, \mathbb{R})$ *gilt also* $f \colon \mathbb{R}^n \to \mathbb{R}$ *und ist d-mal stetig-differenzierbar.*

Definition 2.8. *(Hessematrix) [7, S. 207]*
Hat man eine Abbildung $f \colon U \to \mathbb{R}$*,* $f \in C^2(\mathbb{R}^n, \mathbb{R})$*, so kann man diese in Matrixform angeben. Die entsprechende Matrix* $H_f(x) \in \mathbb{R}^{n \times n}$ *mit*

$$H_f(x) = \begin{pmatrix} D_1 D_1 f(x) & \dots & D_1 D_n f(x) \\ \vdots & \ddots & \vdots \\ D_n D_1 f(x) & \dots & D_n D_n f(x) \end{pmatrix} \tag{2.4}$$

heißt Hessematrix von f *im Punkt* x*.* $H_f(x)$ *ist symmetrisch nach dem Satz von Schwarz [7, S. 204]. Hierbei bezeichnet* D_i *die Ableitung bezüglich der i-ten Variable,* $i \in \{1, 2, \dots, n\}$*.*

Definition 2.9. *(Kompakter Träger) [2, S. 81]*
Für $u \in C(\Omega) \to \mathbb{R}, \Omega \subseteq \mathbb{R}^n$ *heißt*

$$\operatorname{supp}(u) = \overline{\{x \in \Omega \colon u(x) \neq 0\}} \tag{2.5}$$

Träger (support) von u.

In der Optimierung werden üblicherweise Optimierer beschränkter Zielfunktionen gesucht. Es wird somit die Domain einer Funktion betrachtet:

Definition 2.10. *(Domain) [10, S. 104]*
Die Menge

$$\operatorname{dom} f = \{x \in \mathbb{R}^n \colon |f(x)| < \infty\} \tag{2.6}$$

heißt die Domain einer Funktion $f \colon \mathbb{R}^n \to \mathbb{R}$*.*

Da Nesterov seine Algorithmen für konvexe Funktionen definiert hat, wird im Folgenden die Konvexität von Mengen, dann die μ-Konvexität und als Spezialfall der μ-Konvexität die Konvexität definiert.

Definition 2.11. *(Konvexität von Mengen) [12, S. 8]*
Eine Menge $S \subset \mathbb{R}^n$ *ist eine konvexe Menge, falls eine Gerade, die zwei Punkte aus S verbindet, vollständig in S liegt. Das bedeutet, dass für zwei beliebige Punkte* $x \in S$ *und* $y \in S$ *gilt:* $\alpha x + (1 - \alpha) y \in S$ *für alle* $\alpha \in [0, 1]$*.*

Definition 2.12. *(μ-Konvexität) [12, S. 8]*
Eine Funktion $f\colon \mathbb{R}^n \to \mathbb{R}$ ist stark μ-konvex, wenn $\operatorname{dom} f$ konvex ist und $\forall x, y \in \operatorname{dom} f$, $\forall \alpha \in [0,1]$:

$$f(\alpha x + (1-\alpha)y) + \mu\frac{\alpha(1-\alpha)}{2} \leq \alpha f(x) + (1-\alpha)f(y) \tag{2.7}$$

Da Nesterov seine Algorithmen für μ-konvexe Funktionen definiert hat, ist es sinnvoll sich die Definition der μ-Konvexität anzusehen. Außerdem muss man sich überlegen, wie man in den Algorithmen den Konvexitätsparameter μ abschätzen kann. Dafür ist folgender Satz hilfreich:

Satz 2.13. *(Bedingungen für μ-konvexe Funktionen) [2, S. 459]*
Hinreichende Bedingungen für stark μ-konvexe Funktionen sind:

1. *Bedingung erster Ordnung: Die allgemeine Definition der μ-Konvexität wie in* (2.7).
2. *Bedingung zweiter Ordnung (für $f \in C^1(\mathbb{R}^n, \mathbb{R})$):*

$$\forall x, y \in \operatorname{dom} f\colon f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{\mu}{2}\|x-y\|^2 \tag{2.8}$$

3. *Bedingung dritter Ordnung (für $f \in C^2(\mathbb{R}^n, \mathbb{R})$):*

$$\langle H_f(x)y, y\rangle \geq \mu\|y\|^2 = \mu\langle y, y\rangle \tag{2.9}$$

Ein Spezialfall der μ-Konvexität ist die bekanntere Definition der Konvexität mit $\mu = 0$.

Definition 2.14. *(Konvexität von Funktionen) [12, S. 8]*

1. *Die Funktion $f\colon S \to \mathbb{R}$ ist eine konvexe Funktion, falls $S \subset \mathbb{R}^n$ eine konvexe Menge ist und für zwei beliebige Punkte x und y in S folgende Annahme erfüllt ist:*

$$\forall \alpha \in [0,1]\ , \forall x, y \in S\colon \quad f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y) \tag{2.10}$$

2. *Die Funktion f ist streng konvex, wenn in* (2.10) *echte Ungleichheit für $x \neq y$ und α aus dem offenen Intervall $(0,1)$ gilt. Es gilt also:*

$$\forall \alpha \in (0,1)\ , \forall x, y \in S\colon \quad f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y) \tag{2.11}$$

3. *Eine Funktion f ist konkav, wenn $-f$ konvex ist.*

Im Folgenden wird anhand zwei akademischer Beispiele Konvexität gezeigt.

Satz 2.15. *Die lineare Funktion $f\colon \mathbb{R}^n \to \mathbb{R}$ mit $f(x) = c^T x + \beta$ für einen beliebigen Vektor $c \in \mathbb{R}^n$ und $\beta \in \mathbb{R}$ ist konvex.*

Beweis.

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= c^T(\alpha x + (1-\alpha)y) + \beta \\ &= \alpha c^T x + (1-\alpha)c^T y + (\alpha + 1 - \alpha)\beta \\ &= \alpha f(x) + (1-\alpha) f(y) \end{aligned}$$

□

Satz 2.16. *Die quadratische Funktion $q\colon \mathbb{R}^n \to \mathbb{R}$ mit $q(x) = x^T H x$, wobei $H \in \mathbb{R}^{n\times n}$ eine symmetrische, positiv semidefinite Matrix ist, ist konvex.*

Beweis. Mit der Bedingung dritter Ordnung (2.9) gilt:

$$\begin{aligned} y^T H_q(x) y &\geq \mu y^T y \\ &= \mu \|y\|^2 \end{aligned}$$

□

Zur Veranschaulichung der Eigenschaften von μ betrachten wir das folgende Problem. Hierbei soll beispielhaft gezeigt werden, wie man μ bei einer konvexen Funktion abschätzen kann.

Beispiel 2.17. *Es sei $f\colon \mathbb{R}^2 \to \mathbb{R}$ eine quadratische Funktion mit*

$$f(x,y) = \frac{1}{2}x^2 + 2y^2. \tag{2.12}$$

Es gilt f zu minimieren. Für den Gradienten gilt : $\nabla f(x,y) = (x, 4y)^T$ und die Hessematrix $H_f(x,y) = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$ ist konstant und symmetrisch, positiv definit. Dann gilt mit der Bedingung dritter Ordnung aus Formel (2.9) $\mu = 1$.

Auch die Lipschitzkonstante ist in Nesterovs Algorithmen bedeutend. Diese ist folgendermaßen definiert:

Definition 2.18. *(Lipschitz-Stetigkeit) [14, S. 73]*
Eine Funktion $f \in C^1(\mathbb{R}^n, \mathbb{R})$ heißt Lipschitz-stetig mit Lipschitzkonstante $L \in \mathbb{R}^+$, falls gilt:

$$\forall x, y \in \mathbb{R}^n\colon \quad \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{2.13}$$

Eine weitere Bedingung für Lipschitz-Stetigkeit einer Funktion $f \in C^2(\mathbb{R}^n, \mathbb{R})$ lautet:

$$L I_n \geq H_f(x_k) \tag{2.14}$$

Dies kann man mit dem größten Eigenwert der Hessematrix abschätzen [15].

Beispiel 2.19. *Zur exemplarischen Berechnung von L betrachten wir wieder Beispiel 2.17. Mit (2.14) und der Berechnung des größten Eigenwertes gilt für das Problem $L = 4$.*

2.2 Grundlagen der Optimierung

Das Ziel, der in dieser Arbeit verwendeten Algorithmen, ist das Lösen des Problems P nach 2.1.

Definition 2.20. *(Minimierer) [12, S. 12f.]*

1. *Ein Punkt $x^* \in \mathbb{R}$ heißt globaler Minimierer, wenn $f(x^*) \leq f(x)$ für alle $x \in \mathbb{R}^n$ gilt.*
2. *x^* heißt lokaler Minimierer, wenn es für alle $x \in B(x^*, \delta)$ ein $\delta > 0$ gibt, mit $f(x^*) \leq f(x)$. $B(x^*, \delta)$ bezeichnet die Kugel mit Mittepunkt x^* und Radius δ.*
3. *x^* heißt strikter lokaler Minimierer, wenn es $\forall x \in B(x^*, \delta)$ ein $\delta > 0$ gibt, mit $f(x^*) < f(x)$.*

Satz 2.21. *(Bedingungen an Minimierer) [12, S. 15]*

1. *Notwendige Bedingung erster Ordnung:*
   Ist x^* ein lokaler Minimierer und $f \in C^1(\mathbb{R}^n, \mathbb{R})$, dann ist $\nabla f(x^*) = 0$.
2. *Notwendige Bedingung zweiter Ordnung:*
   Ist x^* ein lokaler Minimierer und $f \in C^2(\mathbb{R}^n, \mathbb{R})$, dann ist $\nabla f(x^*) = 0$ und $H_f(x^*)$ positiv semidefinit.
3. *Hinreichende Bedingung zweiter Ordnung:*
   Ist $f \in C^2(\mathbb{R}^n, \mathbb{R})$, $\nabla f(x^*) = 0$ und $H_f(x^*)$ ist positiv (semi-)definit, dann ist x^* ein (strikter) lokaler Minimierer.

Um die Unterschiede, beziehungsweise die Gemeinsamkeiten der Standard-Verfahren zu erkennen, wird zunächst eine allgemeine Minimierungsmethode vorgestellt und danach auf die einzelnen Algorithmen eingegangen.

Definition 2.22. *(Allgemeine iterative Methode der unrestringierten Optimierung) [12, S. 30]*
Eine allgemeine Methode der Optimierung sieht folgendermaßen aus:
1) Initialisierung:

Setze k=0 und wähle ein x_0.

2) Solange die Abbruchbedingungen nach 2.23 nicht erfüllt sind:

$$\left.\begin{array}{c} x_{k+1} = x_k + a_k p_k \\ k = k + 1 \end{array}\right\} \tag{2.15}$$

Im Folgenden bezeichnet x_k die k-te Iterierte des Algorithmus für $k \in \mathbb{N}_0$, p_k die Suchrichtung und a_k die Schrittweite.

Definition 2.23. *(Abbruchkriterien) [5]*
Die Abbruchbedingungen nach Gill, Murray und Wright sind folgendermaßen definiert:

1. $|f(x_k) - f(x_{k+1})| \leq \tau_1(1 + |f(x_{k+1})|)$
2. $\|x_k - x_{k+1}\| \leq \sqrt{\tau_2(1 + \|x_{k+1}\|)}$
3. $\|\nabla f(x_{k+1})\| \leq \sqrt[3]{\tau_3}(1 + |f(x_{k+1})|)$
4. $\|\nabla f(x_{k+1})\| \leq \varepsilon$
5. $k > k_{max}$

$\tau_1, \tau_2, \tau_3 \in \mathbb{R}^+$ sind die Toleranzwerte, ε die Maschinengenauigkeit und k_{max} die maximale Iterationszahl. Der Algorithmus wird abgebrochen, wenn $(1 \wedge 2 \wedge 3) \vee 4 \vee 5$.

Der Gradient beschreibt die Richtung des stärksten Anstiegs einer Funktion. Daher nutzt man zur Minimierung den negativen Gradienten (Die Richtung des steilsten Abstiegs):

Definition 2.24. *(Gradienten-Verfahren) [12, S. 20]*
Mit der Suchrichtung

$$p_k = -\nabla f(x_k) \tag{2.16}$$

ist mit Definition 2.22 das Gradienten-Verfahren definiert.

Zur Bestimmung einer Suchrichtung kann man auch Informationen über die lokale Krümmung der zu minimierenden Funktion nutzen. Dies macht sich das Newton-Verfahren zunutze:

Definition 2.25. *(Newton-Verfahren) [12, S. 22]*
In allen Punkten x_k mit $\nabla f(x_k) \neq 0$, in denen die Hesse-Matrix $H_f(x_k)$ positiv definit ist, ist mit der Newton-Richtung

$$p_k = -(H_f(x_k))^{-1} \nabla f(x_k) \tag{2.17}$$

und mit Definition 2.22 das Newton-Verfahren definiert.

Eine Abwandlung des Newton-Verfahren ist das Gauß-Newton-Verfahren. Für eine nähere Beschreibung des Gauß-Newton-Verfahren, das in FAIR genutzt wird siehe [12, S. 254].

Definition 2.26. *(Quasi-Newton-Verfahren) [12, S. 24]*
Für eine Folge von symmetrisch positiv definiten Matrizen $(B_k)_{k \in \mathbb{N}_0} \in \mathbb{R}^{n \times n}$ heißt ein Verfahren mit der Suchrichtung

$$p_k = -B_k^{-1} \nabla f(x_k) \tag{2.18}$$

und mit Definition 2.22 Quasi-Newton-Verfahren, wobei $\lim_{k\to\infty} B_k \to H_f(x_k)$.

Eine populäre Art zur Berechnung der symmetrisch positiv definiten Matrix B_k ist das in dieser Arbeit verwendete BFGS-Update. Für Details sei auf [12, S. 140] verwiesen.

Definition 2.27. *(Schrittweitenbestimmung) [10, S. 28]*
Zur Bestimmung der Schrittweiten gibt es verschiedene Methoden. Im Folgenden sind die in dieser Arbeit verwendeten aufgeführt.

1. *Die Folge* $\{a_k\}_{k=0}^{\infty}$ *wird vor Beginn der Iterationen fest gewählt. Zum Beispiel:*
   konstant: $a_k = h > 0$ *oder,*
   monoton fallend: $a_k = \frac{h}{\sqrt{k+1}}$

2. *Die Wolfe-Bedingung [12], die auch Armijo-Regel genannt wird:*

$$\phi(a_k) \leq \phi(0) + a_k\sigma\phi(0)' = \phi(0) + a_k\sigma\nabla f(x_k)^T p_k \tag{2.19}$$

   hierbei ist $\sigma \in (0,1)$ *und* $a_k \in [0,1]$*, sowie* $\phi(a_k) = f(x_k + a_k p_k)$ *und* $\phi\colon \mathbb{R} \to \mathbb{R}$ *die Schrittweitenfunktion.*

3 Methoden

In den Methoden werden die verschiedenen Algorithmen von Nesterov vorgestellt. Zunächst wird der Algorithmus aus dem Paper „A Method of solving a convex programming Problem with convergence rate $\mathcal{O}(1/k^2)$“ erläutert [11]. Danach wird auf die Algorithmen aus Nesterovs Buch „Introductory Lectures on Convex Programming Volume I: Basic course“ eingegangen [10]. Abschließend werden kurz wichtige Begriffe und Methoden der Bildregistrierung beschrieben.

3.1 Die Nesterov-Algorithmen nach [11] und [10]

Zunächst wird das Problem 2.1 einer Minimierung einer konvexen Funktion f betrachtet. Zum Lösen wird folgende Methode benutzt:

Algorithmus 1 Der Nesterov-Algorithmus nach [11]

---

1) Initialisierung:

Man wähle einen Punkt $y_0 \in \mathbb{R}^n$. Setze

$$k = 0, \quad a_0 = 1, \quad x_{-1} = y_0, \quad \beta_{-1} = \frac{\|y_0 - z\|}{\|\nabla f(y_0) - \nabla f(z)\|} \tag{3.1}$$

wobei $z \in \mathbb{R}^n$ ist, sodass gilt: $z \neq y_0$ und $\nabla f(z) \neq \nabla f(y_0)$.

2) Solange die Abbruchbedingungen aus Definition 2.23 nicht erfüllt sind:

a) Berechne den kleinsten Index $i \geq 0$, so dass

$$f(y_k) - f(y_k - 2^{-i}\beta_{k-1}\nabla f(y_k)) \geq 2^{-i-1}\beta_{k-1}\|\nabla f(y_k)\|^2. \tag{3.2}$$

b) Berechne damit

$$\left.\begin{aligned} &\beta_k = 2^{-i}\beta_{k-1}, \quad x_k = y_k - \beta_k \nabla f(y_k), \\ &a_{k+1} = \frac{1 + \sqrt{4a_k^2 + 1}}{2}, \\ &y_{k+1} = x_k + \frac{(a_k - 1)(x_k - x_{k-1})}{a_{k+1}}. \end{aligned}\right\} \tag{3.3}$$

c) $k = k + 1$

Der Algorithmus 1 verwendet 2 Schritte in einer Iteration. Dabei wird zunächst der Anfangspunkt $x_{-1} = y_0$ gesetzt und mit einer Schrittweitenbestimmung, die ähnlich der Armijo-Regel ist. Mit dieser wird x_k dann geupdated. Mit a_{k+1} wird die zweite Schrittweite berechnet. a_k ist eine vorher festgelegte Folge, mit immer kleiner werdenden Schrittweiten. Mit dieser Schrittweite a_{k+1} wird dann der eigentliche Startwert y_0 geupdatet, unter Zuhilfenahme der vorher berechneten Gradientenschritte und und des vorherigen Punktes. Dies wird so lange iteriert, bis das globale Minimum nach den Abbruchkriterien nach Definition 2.23 erreicht ist.

Algorithmus 2 Nesterov-General-Scheme [10]

1) Initialisierung:
Man wähle einen Punkt $x_0 \in \mathbb{R}^n$ und $\gamma_0 > 0$. Setze $k = 0$ und $\upsilon_0 = x_0$.
2) Solange die Abbruchbedingungen aus Definition 2.23 nicht erfüllt sind:

a) Berechne $\alpha_k \in (0, 1)$ durch Lösen der Gleichung

$$L\alpha_k^2 = (1 - \alpha_k)\gamma_k + \alpha_k\mu. \tag{3.4}$$

Setze $\gamma_{k+1} = (1 - \alpha_k)\gamma_k + \alpha_k\mu = L\alpha_k^2$.
b) Berechne

$$y_k = \frac{\alpha_k\gamma_k\upsilon_k + \gamma_{k+1}x_k}{\gamma_k + \alpha_k\mu}. \tag{3.5}$$

c) Schrittweitenbestimmung von a_k nach Armijo (2.19).
d) $x_{k+1} = x_k - a_k\nabla f(x_k)$
e) Danach:

$$\upsilon_{k+1} = \frac{1}{\gamma_{k+1}}[(1 - \alpha_k)\gamma_k\upsilon_k + \alpha_k\mu y_k - \alpha_k\nabla f(y_k)] \tag{3.6}$$

f) $k = k + 1$

Auch hier wird zunächst ein beliebiger Startpunkt x_0 gewählt und ein Vergleichswert υ_0 auf denselben Startwert gesetzt. Außerdem wird ein γ_k beliebig, aber fest gewählt. Mithilfe der Lipschitzkonstanten und des Konvexitätsparameters kann man die Schrittweite α_k aus einer quadratischen Gleichung berechnen, die so gewählt ist, dass immer eine Lösung der quadratischen Gleichung in $(0, 1)$ liegt. Danach wird y_k mit γ_k, α_k , υ_k, x_k, μ und γ_{k+1} so gewählt, dass υ_k und x_k gewichtet werden. Danach wird x_k mit der Armijo-Schrittweitenbestimmung aktualisiert. Also ein Schritt des Gradientenverfahrens auf x_{k+1} angewendet, damit dieses in den nächsten Schritt in der Berechnung von y_k einfließen kann. Danach wird υ_{k+1} aktualisiert, wobei auch hier auffällt, dass ein Schritt in die Richtung des negativen Gradienten gemacht wird, allerdings noch zwei Schritte in eine andere Richtung. Dies wird so lange iteriert, bis die Abbruchbedingungen aus Definition 2.23 erfüllt sind. Dies ist der Fall, wenn y_k nah genug am Minimum liegt.

Algorithmus 3 Nesterov-Constant-Step-Scheme 1 [10]

1) Initialisierung:
Man wähle einen Punkt $x_0 \in \mathbb{R}^n$ und $\gamma_0 > 0$. Setze $k = 0$ und $v_0 = x_0$.
2) Solange die Abbruchbedingungen aus Definition 2.23 nicht erfüllt sind:

a) Berechne $\alpha_k \in (0, 1)$ durch Lösen der Gleichung

$$L\alpha_k^2 = (1 - \alpha_k)\gamma_k + \alpha_k\mu. \tag{3.7}$$

Setze

$$\gamma_{k+1} = (1 - \alpha_k)\gamma_k + \alpha_k\mu = L\alpha_k^2. \tag{3.8}$$

b) Berechne

$$y_k = \frac{\alpha_k\gamma_k v_k + \gamma_{k+1}x_k}{\gamma_k + \alpha_k\mu}. \tag{3.9}$$

c) Setze

$$\left.\begin{aligned} x_{k+1} &= y_k - \frac{1}{L}\nabla f(y_k) \\ v_{k+1} &= \frac{1}{\gamma_{k+1}}[(1 - \alpha_k)\gamma_k v_k + \alpha_k\mu y_k - \alpha_k\nabla f(y_k)]. \end{aligned}\right\} \tag{3.10}$$

d) $k = k + 1$

Das Nesteroc-Constant-Step-Scheme 1 hat im Vergleich zum General-Scheme nur einen Unterschied. Die x_{k+1} werden nicht mit einer Armijo-Schrittweitenbestimmung berechnet, sondern mit einer festgelegten Schrittweite von $\frac{1}{L}$.

Algorithmus 4 Nesterov-Constant-Step-Scheme 2 [10]

1) Initialisierung:
Man wähle einen Punkt $x_0 \in \mathbb{R}^n$ und $\alpha_0 \in (0, 1)$. Setze $k = 0$ und $v_0 = x_0$ und $q = \frac{\mu}{L}$.
2) Solange die Abbruchbedingungen aus Definition 2.23 nicht erfüllt sind:

a) Setze $x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k)$.
b) Berechne $\alpha_{k+1} \in (0, 1)$ aus der Gleichung

$$\alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2 + q\alpha_k \tag{3.11}$$

und setze

$$\left.\begin{aligned} \beta_k &= \frac{\alpha_k(1 - \alpha_k)}{\alpha_k^2 + \alpha_{k+1}} \\ y_{k+1} &= x_{k+1} + \beta_k(x_{k+1} - x_k). \end{aligned}\right\} \tag{3.12}$$

c) $k = k + 1$

Beim Nesterov-Constant-Step-Scheme 2 werden nicht mehr die Konstanten μ

und L an sich betrachtet, sondern der Quotient aus beiden. $q = \frac{\mu}{L}$. Zunächst wird das x_{k+1} aktualisiert mit $\frac{1}{L}$ als Schrittweitenbestimmung. Danach wird die Schrittweite α_{k+1} mit einem im ersten Schritt beliebig, aber festem α aktualisiert. Mit β_k wird die neue und die alte Schrittweite α_k und α_{k+1} in Verhältnis gesetzt. Am Schluss jeder Iteration wird das y_{k+1} aktualisiert, indem der Abstand von x_{k+1} und x_k durch β gewichtet wird. Der Algortihmus wird iteriert, bis die Abbruchbedingung nach Definition 2.23 erfüllt sind und y_k das Minimum annähernd erreicht hat.
In den Algorithmen 1, 2, 3 und 4 wurde der Konvexitätsparameter $\mu = 0$ [10, S. 67] gewählt und die Lipschitzkonstante L in jedem Iterationsschritt durch den größten Eigenwert der Hessematrix abgeschätzt.

3.2 Grundlagen der Bildregistrierung

In der Bildregistrierung versucht man mehrere Bilder des gleichen Motivs zu überlagern [3, 6]. Im Folgenden wird von zwei verschiedenen Bildern ausgegangen. Gegeben sind ein Templatebild T und ein Referenzbild R. Die Aufgabe ist es, eine Transformation $y\colon \mathbb{R}^n \to \mathbb{R}^d$ zu bestimmen, so dass $T(y)$ ähnlich zu R ist, wobei $\Omega \subset \mathbb{R}^d$ ein kompakter Träger ist und $T : \Omega \to \mathbb{R}$ und $R : \Omega \to \mathbb{R}$ abbilden. Es sollen somit korrespondierende Strukturen passend ausgerichtet werden. In der Modellierung mit FAIR geschieht dies durch Optimierung einer Zielfunktion

$$J(y) = D(T(y), R) + \alpha S(y), \qquad \alpha \in \mathbb{R}^+ \tag{3.13}$$

[8]. Die Funktion D beschreibt hierbei das Distanzmaß, welches angibt, wie ähnlich sich $T(y)$ und R sind. S ist ein Regularisierer, der die Transformation y glättet und die Möglichkeit zur Integration von (zum Beispiel physikalischem) Vorwissen ermöglicht. Der Regularisierungsparameter α gibt die Möglichkeit den Einfluss der beiden Funktionen zu gewichten. In dieser Arbeit wurde als Distanzmaß die Sum of squared differences verwendet:

Definition 3.1. *(Sum of squared differences (SSD)) [8, S. 71]*
Die Summe der quadrierten Differenzen ist folgendermaßen definiert:

$$D^{SSD}[T(y), R] = \frac{1}{2}\int_\Omega (T(y(x)) - R(x))^2 dx \tag{3.14}$$

Die Summe der quadrierten Differenzen ist ein intuitives Distanzmaß, da es die Grauwertabstände zwischen Referenz- und Templatebild quadratisch bestraft. Ein ähnliches Distanzmaß ist das Integral über den Betrag der Grauwertdifferenzen. Dieser ist allerdings nicht überall stetig-differenzierbar, was in der ableitungsbasierten Optimierung für Probleme sorgt. SSD ist gut geeignet, wenn zwei korrespondierende Strukturen ähnliche Grauwerte haben, wie es bei der monomodalen Registrierung der Fall ist. Monomodal bedeutet, dass beide Bilder mit dem gleichen Gerät, zum Beispiel CT, aufgenommen wurden.

Bei multimodaler Registrierung gilt die Annahme gleicher Intensitäten nicht zwangsläufig und andere Distanzmaße können vorteilhaft sein.
In der Bildregistrierung werden verschiedene Transformationen vorgenommen. Im Bereich der parametrischen Transformation werden in dieser Arbeit zwei weit verbreitete Arten betrachtet, die affine-lineare Transformation und die rigide Transformation.
Mit der affinen Transformation können Vektoren skaliert, geschert, rotiert und transliert werden.

Definition 3.2. *(Affin-lineare Transformationen) [8, S. 47]*
Affin-lineare Transformationen haben die Form

$$y : \mathbb{R}^d \to \mathbb{R}^d \quad y(x) = Ax + b, \tag{3.15}$$

wobei in $A \in \mathbb{R}^{d\times d}$ die Skalierung (Vergrößerung, Verkleinerung), Scherung und Rotation (Drehung, Spiegelung) einfließt und in $b \in \mathbb{R}^d$ die Translation (Verschiebung). Dabei ist $d \in \mathbb{N}$ die Raumdimension.

Typischerweise gilt $d \in \{2, 3, 4\}$, da in der medizinischen Bildregistrierung noch keine Anwendung für $d \geq 5$ notwendig ist. Ein Spezialfall der affin-linearen Transformationen sind rigide Transformationen.

Definition 3.3. *(rigide Transformationen) [8, S. 47]*
Rigide Transformationen ermöglichen nur Rotationen und Translationen. Das bedeutet, dass A orthogonal ist, also $A^T = A^{-1}$. Dies gilt insbesondere für Drehmatrizen.

Beispiel 3.4. *Eine Drehung um den Winkel $\theta \in \mathbb{R}$ wäre im zweidimensionalen Fall durch die Matrix*

$$A = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{3.16}$$

beschrieben.

Definition 3.5. *(Nicht-parametrische Transformationen) [8, S. 47]*
Bei der nicht-parametrischen Transformation kann jeder Punkt des Bildgebiets individuell transformiert werden. Dies kann zum Beispiel durch Addition einer vom betrachteten Punkt x abhängigen Deformation $u : \mathbb{R}^d \to \mathbb{R}^d$ beschrieben werden.

$$y(x) = x + u(x) \tag{3.17}$$

Synonym werden für nicht-parametrisch auch oft die Begriffe elastisch oder deformierbar verwendet.

Definition 3.6. *(Regularisierer) [8, S. 117]*
Der elastische Regularisierer ist das elastische Potential, dass die Energie, die durch die Deformation eines elastischen Materials entsteht, misst.

$$S(y) = \frac{1}{2} \int_\Omega \mu^E \langle \nabla y, \nabla y \rangle + (\lambda^E + \mu^E)(\nabla \cdot y)^2 dx \tag{3.18}$$

FAIR

Die in dieser Arbeit verwendeten Datensätze sind Beispiele aus FAIR [8]. In FAIR werden auch Grundlagen und Methoden der Bildregistrierung vorgestellt. Die FAIR-Software benötigt MATLAB. Da MATLAB sich gut eignet, um numerische Berechnungen durchzuführen, wurden alle Implementierungen dieser Thesis in MATLAB durchgeführt. Hierzu wird die Minimierung von $J(y)$ (3.13) im „Discretize-Then-Optimize-Framework“ vorgenommen und die Methoden der Numerischen Optimierung finden Anwendung. Hierzu werden diskretisierte Formulierungen für Distanzmaße und Regularisierer benötigt. Da die Diskretisierung nicht im Fokus dieser Arbeit lag, sei für die Details auf [8] verwiesen.

4 Experimente

Zunächst werden zwei akademische Funktionen mit den verschiedenen Algorithmen optimiert. Dabei ist die eine Funktion eine strikt konvexe, bei der sowohl der Konvexitätsparameter μ als auch die Lipschitzkonstante L bekannt ist. Um die Nesterov-Algorithmen auch für eine nicht konvexe Funktion zu testen, wird die Rosenbrock-Funktion verwendet. Danach werden die Algorithmen 1-4, sowie das Gradienten-Verfahren und das Gauß-Newton-Verfahren anhand der verschiedenen Datensätze aus FAIR grafisch und tabellarisch verglichen. Die maximale Schrittzahl ist dabei pro Level auf 100 Iterationen begrenzt, genauso wie die maximale Anzahl von Schrittweitenbestimmungen (Linesearches) je Iteration 100 beträgt. Die Abbruchtoleranzen nach Definition 2.23 sind mit $\tau_1 = 10^{-3}$, $\tau_2 = 10^{-1}$ und $\tau_3 = 10^{-2}$ gewählt.

4.1 Akademische Beispiele

Für die akademischen Beispiele wurde $B_0 = I_n$ und BFGS-Updates für das Quasi-Newton-Verfahren gewählt. Als Schrittweitenbestimmung wurde für das Newton-Verfahren, das Quasi-Newton-Verfahren und das Gradienten-Verfahren die Armijo-Schrittweite festgelegt.

Die Rosenbrock Funktion

Definition 4.1. *Die Rosenbrock Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ ist mit*

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2 \tag{4.1}$$

mit dem Gradienten

$$\nabla f(x, y) = \begin{pmatrix} -400(y - x^2) + 2x - 2 \\ 200(y - x^2) \end{pmatrix} \tag{4.2}$$

und der Hessematrix

$$H_f(x, y) = \begin{pmatrix} 1200x^2 + 2 - 400y & -400x \\ -400x & 200 \end{pmatrix} \tag{4.3}$$

definiert.

Das Problem P nach 2.1 soll ausgehend von $(x_0, y_0)^T = (-1.2, 1)$ minimiert werden.

Satz 4.2. *Die Rosenbrock-Funktion ist nicht konvex:*

Beweis. Wähle zunächst zwei Punkte $g_1 = (0, 0)$ und $g_2 = (-1, 2)$ und $\alpha = 0.5$. Dann ist zu zeigen:

$$\begin{aligned} & f(\alpha g_1 + (1-\alpha)g_2) \leq \alpha f(g_1) + (1-\alpha) f(g_2) \\ \Leftrightarrow\ & f(0.5 \cdot (0,0) + 0.5 \cdot (-1,2)) \leq 0.5 \cdot 1 + 0.5 \cdot 104 \\ & \Leftrightarrow 58.5 \leq 52.5 \lightning \end{aligned}$$

Daraus folgt, die Rosenbrock-Funktion ist nicht konvex. □

Satz 4.3. *Der einzige stationäre Punkt der Rosenbrock-Funktion ist* $x^* = (1, 1)^T$.

Beweis. Berechne zuerst x^* mit $\nabla f(x, y) \stackrel{!}{=} 0$.

$$\begin{aligned} -400(y - x^2) + 2x - 2 &= 0 \\ 200(y - x^2) &= 0 \end{aligned}$$

Aus $200(y - x^2) = 0$ folgt $y = x^2$. Damit folgt:

$$\begin{aligned} -400(x^2 - x^2) + 2x - 2 &= 0 \\ \Leftrightarrow 2x - 2 &= 0 \\ \Rightarrow x &= 1 \\ \Rightarrow y &= 1 \end{aligned}$$

Daraus folgt $x^* = (1, 1)$ ist die eindeutige Lösung.

□

Satz 4.4. *Zeige dann, dass* $H_f(x^*)$ *symmetrisch, positiv definit ist.*

Beweis. $H_f(1,1) = \begin{pmatrix} 1200 + 2 - 400 & -400 \\ -400 & 200 \end{pmatrix} = \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix}$.
Die Matrix ist symmetrisch. Da die Eigenwerte $\lambda_1 = -(\sqrt{250601} - 501) = 0.399$ und $\lambda_2 = \sqrt{(}250601)501 = 1001.6$ größer null sind, ist die Matrix symmetrisch, positiv definit und x^* ein Minimierer nach Satz 2.21 (3). □

Da die verschiedenen Algorithmen den Minimierer bestimmen sollen, kann man anhand des Minimierers die verschiedenen Verfahren analysieren. In Tabelle 4.1 sind die Ergebnisse der Optimierung der Rosenbrock-Funktion durch die verschiedenen Algorithmen dargestellt. Dabei wurden die Anzahl der Iterationen, die Laufzeit, der berechnete Minimierer und das Minimum aufgezeigt. In Abbildung 4.1 werden die Höhenlinien mit dem Minimum der Rosenbrock-Funktion aufgezeigt. Das Minimum wird hierbei als x dargestellt.

Tabelle 4.1: Ergebnisse der Optimierung der Rosenbrock-Funktion

Verfahren	Iterationen (k)	Laufzeit (in s)	(x_k, y_k)	$f(x_k, y_k)$
Newton	13	0.04	(0.70, 0.49)	0.09
Quasi-Newton	25	0.02	(0.92, 0.85)	0.01
Gradienten	48	0.03	(0.93, 0.87)	0.00
Algorithmus 1	100	0.08	(-59.72, 785.52)	773351843.64
Algorithmus 2	100	0.13	(-1.06, 1.04)	5.05
Algorithmus 3	4	0.01	(-1.00, 1.01)	4.01
Algorithmus 4	6	0.01	(-1.03, 1.06)	4.11

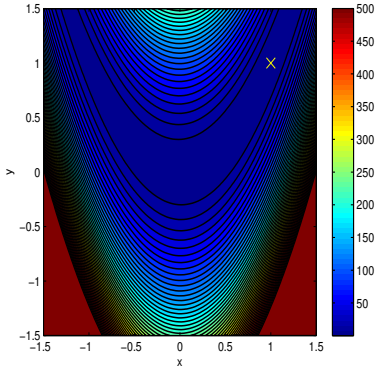


Abbildung 4.1: Höhenlinien der Rosenbrock-Funktion mit dem Minimum der Funktion als x dargestellt

Die quadratische Funktion

Nun wird die quadratische und somit nach Beispiel 2.17 konvexe Funktion

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x, y) = \frac{1}{2}x^2 + 2y^2 + 10x + 10y \tag{4.4}$$

mit dem Gradienten

$$\nabla f(x, y) = \begin{pmatrix} x + 10 \\ 4y + 10 \end{pmatrix} \tag{4.5}$$

und der Hessematrix

$$H_f(x, y) = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \tag{4.6}$$

ausgehend vom Startpunkt $(x_0, y_0)^T = (-1.2, 1)$ minimiert. Hierbei ist der Konvexitätsparameter $\mu = 1$ bekannt.

Satz 4.5. *Der einzige stationäre Punkt ist* $x^* = (-10, -2.5)^T$.

Tabelle 4.2: Ergebnisse der Optimierung von der Quadratischen Funktion (4.4)

Verfahren	Iterationen (k)	Laufzeit (in s)	(x_k, y_k)	$f(x_k, y_k)$
Newton	1	0.0005	(-10.00, -2.50)	-62.50
Quasi-Newton	3	0.001	(-10.00, -2.502)	-62.49
Gradienten	8	0.001	(-10.00, -2.50)	-62.50
Algorithmus 1	9	0.002	(-9.95, -2.50)	-62.49
Algorithmus 2	2	0.004	(-10.00, -2.50)	-62.50
Algorithmus 3	11	0.004	(-9.34, -2.50)	-62.28
Algorithmus 4	100	0.006	(-5.1, -2.50)	-50.48

Beweis. Berechne zunächst x^* mit $\nabla f(x, y) \stackrel{!}{=} 0$.

$$\begin{aligned} x + 10 &= 0 \\ 4y + 10 &= 0 \end{aligned}$$

Daraus folgt, dass $x^* = (-10, -2.5)$.

□

Satz 4.6. *Zeige $H_f(x^*)$ ist symmetrisch, positiv definit.*

Beweis. $H_f(-10, -2.5) = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$.
Die Matrix ist symmetrisch. Da die Eigenwerte $\lambda_1 = 1$ und $\lambda_2 = 4$ größer null sind, ist die Matrix symmetrisch, positiv definit und x^* ist Minimierer. □

In Tabelle 4.2 werden die Ergebnisse der Optimierung der quadratischen Funktion anhand von Iterationen, Laufzeit, Minimierer und Minimum gezeigt. In Abbilung 4.2 werden die Höhenlinien der quadratischen Funktion dargestellt.

4.2 Beispiele aus FAIR

Es wurden drei Datensätze betrachtet: zwei zweidimensionale und ein dreidimensionaler.

Hands

Der Datensatz „Hands“ umfasst zwei 2-D-Röntgenaufnahmen von rechten Händen verschiedener Menschen [8]. Das Templatebild T, das Referenzbild R und die initiale Betragsdifferenz $|T - R|$ sind in Abbildung 4.3 dargestellt.

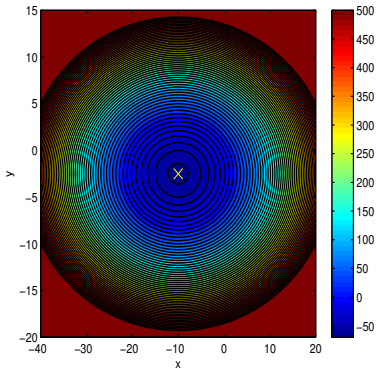
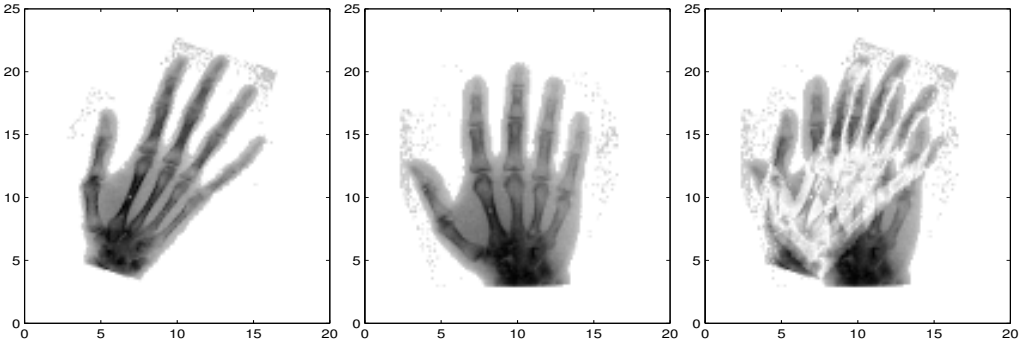


Abbildung 4.2: Höhenlinien der quadratischen Funktion mit dem Minimum als x dargestellt



(a) T (b) R (c) $|T - R|$

Abbildung 4.3: Templatebild T, Referenzbild R und initiale Betragsdifferenz $|T - R|$ des Hands Datensatzes

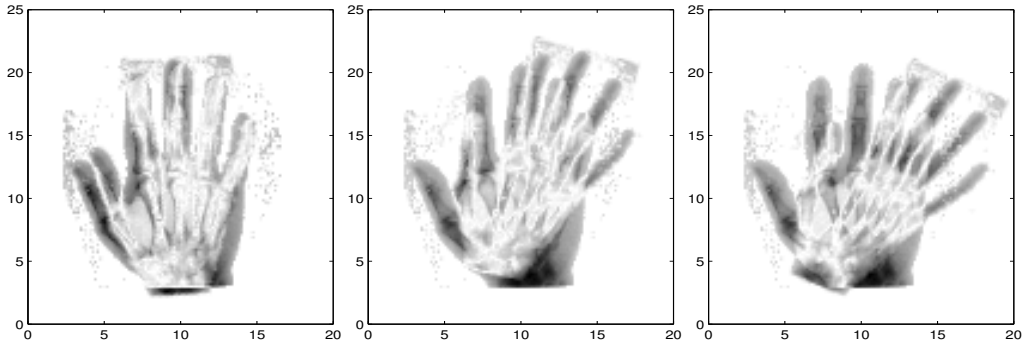
Die Größe des Datensatzes auf dem gröbsten Level ist $m = [8, 8]$, dabei ist m_i die Anzahl der Pixel in der i-ten Raumrichtung. Auf dem feinsten Gitter beträgt $m = [128, 128]$.

Hands MLPIR

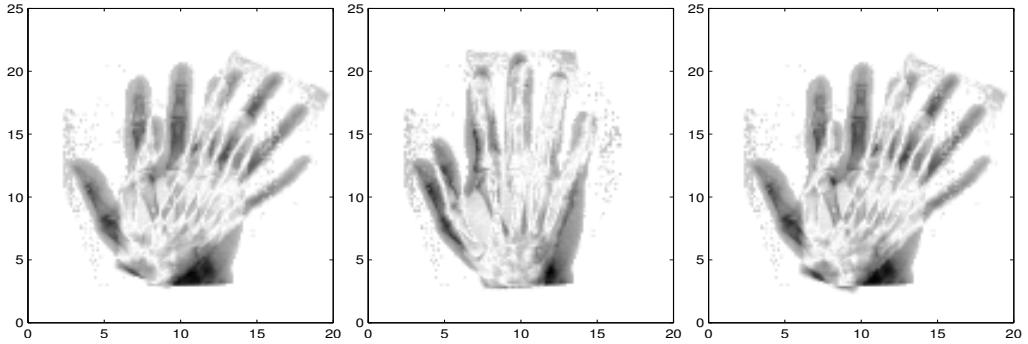
Der Hands-Bilddatensatz wird mit einer rigiden Bildregistrierung im Multilevelansatz registriert, wobei alle Level durchlaufen werden. Es wird die Spline-Interpolation verwendet, sowie das SSD-Distanzmaß und rigide Transformationen. Um einen quantitativen Vergleich aufzeigen zu können, werden die Iterationen pro Level, die Laufzeit der einzelnen Algorithmen und die Reduzierung der Zielfunktion angegeben. Die Laufzeit wird in Sekunden angegeben. Die Ergebnisse der Optimierung sind in Tabelle 4.3 aufgelistet. Dabei bezeichnet $\frac{J(y_k)}{J(y_0)}$ die Reduzierung. Die Betragsdifferenzbilder nach der Registrierung für den grafischen Vergleich sind in Abbildung 4.4 dargestellt.

Tabelle 4.3: Ergebnisse der Optimierung des Beispiels Hands MLPIR

Verfahren	Iterationen pro Level	Laufzeit (in s)	$\frac{J(y_k)}{J(y_0)}$	θ	b_1	b_2
Gauß-Newton	(7, 4, 8, 1, 1)	5.73	0.34	-0.43	-4.32	6.05
Gradienten	(100, 100, 26, 6, 100)	44.19	0.64	-0.07	-2.66	0.7
Algorithmus 1	(100, 66, 2, 4, 5)	22.12	0.70	0.11	-1.01	0.17
Algorithmus 2	(7, 2, 5, 2, 1,)	7.67	0.69	6.38	-1.35	-0.11
Algorithmus 3	(100, 1, 9, 20, 1)	15.83	0.35	-0.39	-4.13	5.08
Algorithmus 4	(6, 100, 12, 10, 1)	16.36	0.66	0.06	-1.78	0.48



(a) Gauß-Newton-Verfahren (b) Gradienten-Verfahren (c) Nesterov-Algorithmus aus Paper [11]



(d) Nesterov-General-Scheme (e) Nesterov-Constant-Step-Scheme 1 (f) Nesterov-Constant-Step-Scheme 2

Abbildung 4.4: Grafischer Vergleich der verschiedenen Optimierungsverfahren anhand des Beispiels Hands MLPIR

Hands MLIR SSD mfElas

Der hier verwendete Bilddatensatz ist „Hands“. Zur Registrierung wurde eine elastische Registrierung mit den Konstanten $\alpha = 10^3$, $\mu^E = 1$ und $\lambda^E = 0$ verwendet. Die Implementierung war hierbei matrixfrei. Es wurde ein Multilevel-

Tabelle 4.4: Ergebnisse der Optimierung des Beispiels Hands MLIR SSD mfElas

Verfahren	Anzahl der Iterationen pro Level	Laufzeit (in s)	Reduzierung $J(y_k)/J(y_0)$
Gauß-Newton	(3, 5, 5, 3, 2)	7.64	0.05
Gradienten	(8, 11, 12, 15, 6)	18.42	0.13
Algorithmus 1	(7, 4, 9, 4, 3)	17.01	0.13
Algorithmus 2	(2, 2, 2, 1, 1)	8.14	0.35
Algorithmus 3	(8, 10, 11, 5, 4)	13.07	0.10
Algorithmus 4	(12, 9, 16, 7, 2)	16.97	0.14

Ansatz mit fünf Leveln und einem maximalen m=[128, 128] verwendet. Das Distanzmaß war SSD und interpoliert wurde mit Splines. Als Vorregistrierung für die nicht parametrische Registrierung diente eine affine Registrierung auf dem gröbsten Level. In Tabelle 4.4 sind die Ergebnisse der Optimierung des Beispiels Hands MLIR SSD mfElas anhand der Iterationszahlen pro Level, der Laufzeit und der Reduzierung $\frac{J(y_k)}{J(y_0)}$ aufgelistet. Die Ergebnisse der verschiedenen Optimierungsverfahren sind in Abbildung 4.5 dargestellt. Die transformierten Bilder $T(y)$ sind in Abbildung 4.6 dargestellt, um die Plausibilität des Ergebnisses zu überprüfen.

HNSP

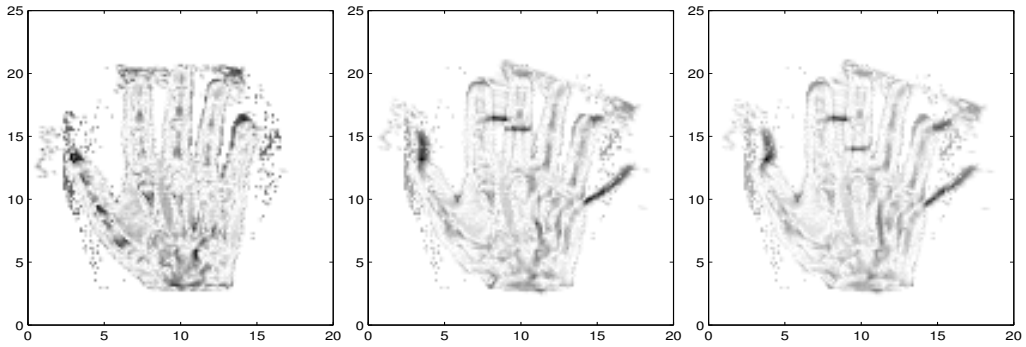
Der Datensatz „HNSP“ stammt aus dem Human Neuro Scanning Project und zeigt zwei Gehirnschnitte unter dem Lichtmikroskop [8]. Dieser wird auf sechs Leveln registriert. Das gröbste Level hat $m = [16, 8]$ Pixel und das feinste Level ist $m = [512, 256]$ Pixel groß. Das Templatebild (T), das Referenzbild (R) und die initiale Betragsdifferenz ($|T - R|$) sind in Abbildung 4.7 dargestellt. Für alle Registrierungen dieses Datensatzes wurde die Spline-Interpolation zur Auswertung der deformierten Templatebilder und der zugehörigen Ableitungen verwendet.

HNSP MLPIR SSD affine2D

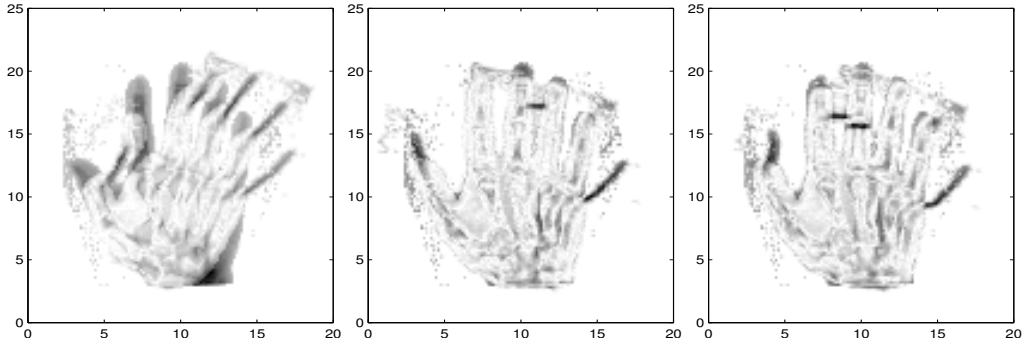
Bei diesem Beispiel wurde SSD als Distanzmaß verwendet und die Transformation war affin. Die quantitativen Ergebnisse sind in Tabelle 4.5 zu finden. Die Ergebnisse nach der Registrierung sind in Abbildung 4.8 dargestellt.

HNSP MLIR SSD mfElas

Es wurde der Datensatz „HNSP“ verwendet. Der Regularisierer war elastisch mit den Parametern $\alpha = 5 \cdot 10^2$, $\mu^E = 1$ und $\lambda^E = 0$, das Distanzmaß war SSD



(a) Gauß-Newton-Verfahren (b) Gradienten-Verfahren (c) Nesterov-Algorithmus aus Paper [11]



(d) Nesterov-General-Scheme (e) Nesterov-Constant-Step-Scheme 1 (f) Nesterov-Constant-Step-Scheme 2

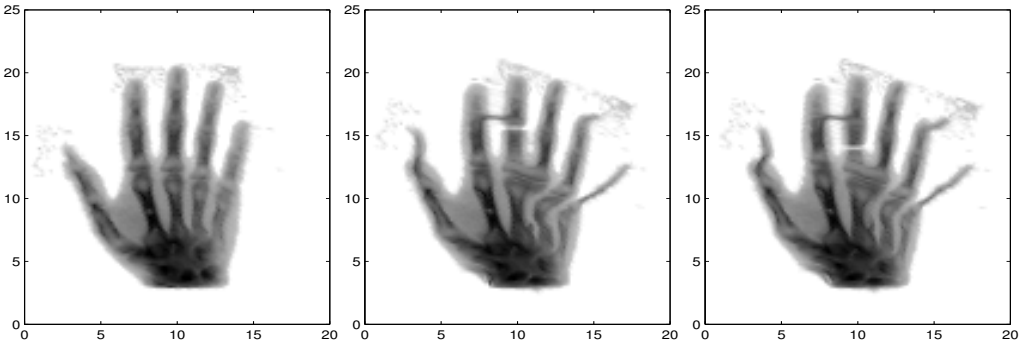
Abbildung 4.5: Grafischer Vergleich der verschiedenen Optimierungsverfahren anhand des Beispiels Hands MLIR SSD mfElas

und es wurde eine affine Transformation verwendet. Die qualitativen Ergebnisse sind in 4.6 aufgelistet. Interpoliert wurde mit Splines. In Abbildung 4.9 sind die grafischen Ergebnisse nach der Optimierung mit den verschiedenen Verfahren dargestellt.

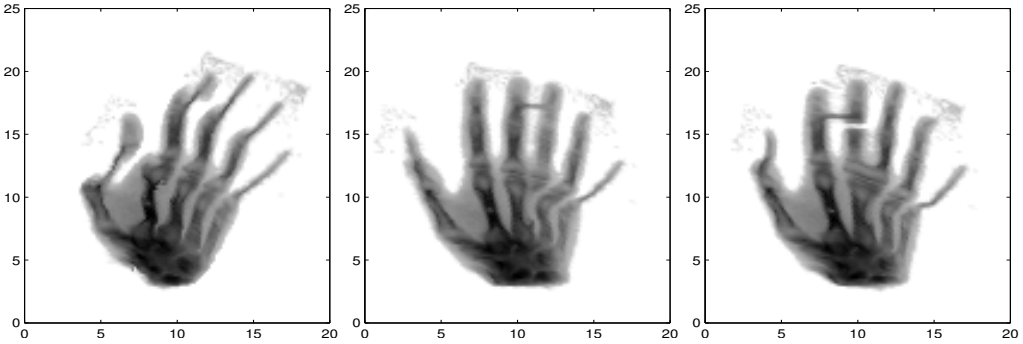
3D-Brain

Der Datensatz 3D-Brain stellt verschiedene Schnitte des 3D-MRT Bildes eines Gehirns dar [8]. Der Datensatz hat auf dem gröbsten Level eine Größe von $m = [16, 8, 16]$ Voxeln und auf dem feinsten Level eine Größe von $m = [128, 64, 128]$ Voxeln. Zur Visualisierung wird ein axialer Schnitt aus der Mitte des Datensatzes verwendet und registriert. Das Templatebild (T), das Referenzbild (R) und die initiale Betragsdifferenz ($|T-R|$) sind in Abbildung 4.10 dargestellt.

Während der Registrierung des 3D-Brain Datensatzes wurde eine lineare Interpolation verwendet und als Distanzmaß wurde SSD gewählt. Die Transformation war affin und der Regularisierer matrixfrei und elastisch. In 4.7 sind die qualitativen Ergebnisse der Optimierung von 3D-Brain dargestellt. Die regis-

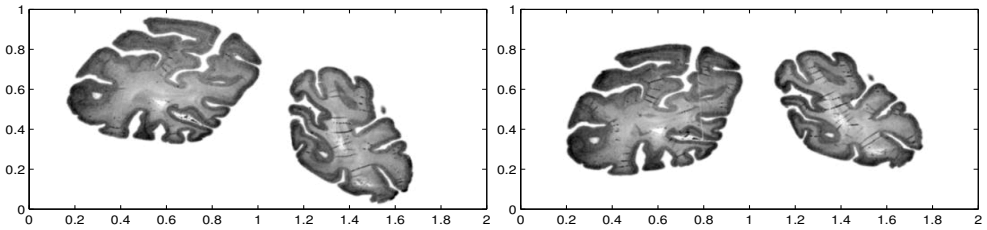


(a) Gauß-Newton-Verfahren (b) Gradienten-Verfahren (c) Nesterov-Algorithmus aus Paper [11]

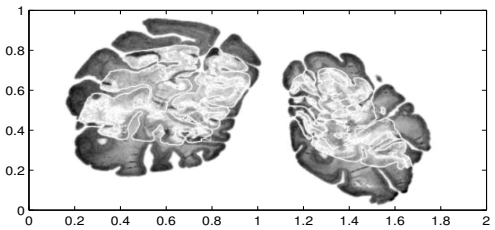


(d) Nesterov-General-Scheme (e) Nesterov-Constant-Step-Scheme 1 (f) Nesterov-Constant-Step-Scheme 2

Abbildung 4.6: Die transformierten Template-Bilder im Beispiel Hands MLIR SSD mfElas



(a) T (b) R



(c) $|T - R|$

Abbildung 4.7: Templatebild T, Referenzbild R und initiale Betragsdifferenz $|T - R|$ des HNSP-Datensatzes

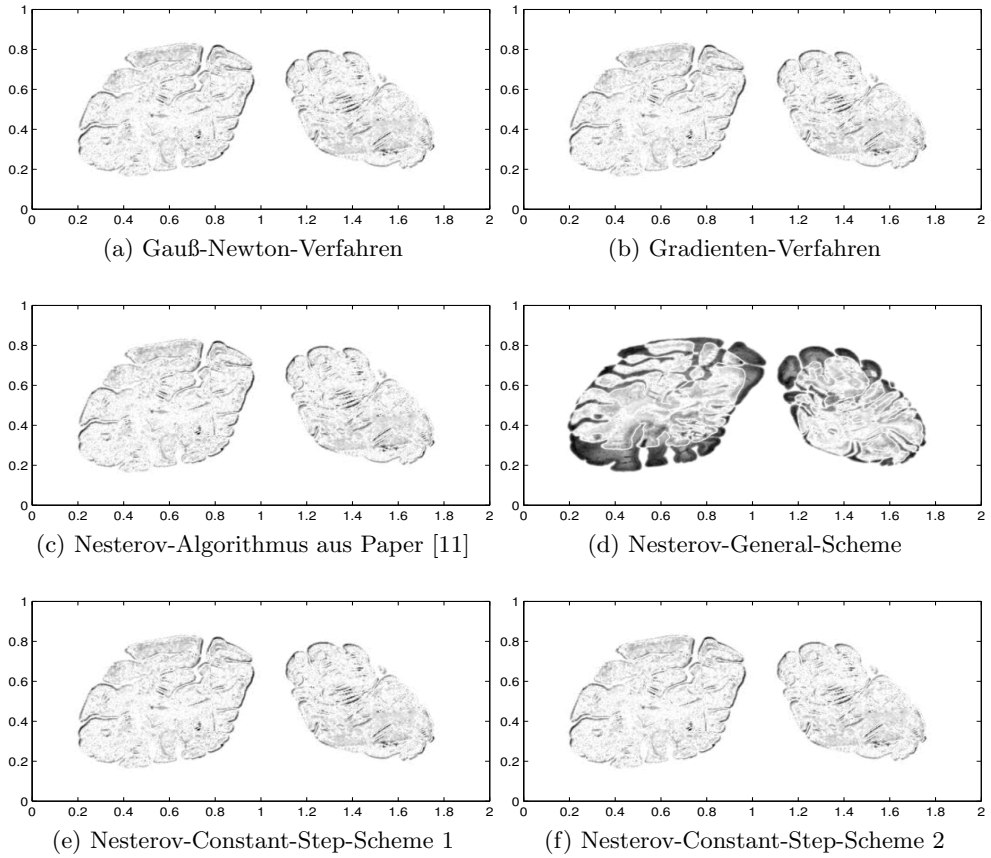
Tabelle 4.5: Ergebnisse der Optimierung des Beispiels HNSP MLPIR SSD affine2D

Verfahren	Iterationen pro Level	Laufzeit (in s)	Reduzierung $J(y_k)/J(y_0)$	$(A_{1,1}, A_{1,2}, b_1$ $(A_{2,1}, A_{2,2}, b_2)$
Gauß-Newton	(5, 2, 2, 1, 1, 1)	7.02	0.03	(0.96, 0.30, -0.12, -0.31, 0.97, 0.34)
Gradienten	(100, 100, 100, 100, 100, 66)	120.39	0.03	(0.96, 0.30, -0.12, -0.32, 0.97, 0.34)
Algorithmus 1	(100, 87, 54, 57, 18, 23)	70.59	0.03	(0.96, 0.30, -0.12, -0.31, 0.97, 0.34)
Algorithmus 2	(2, 2, 3, 4, 8, 14)	43.57	0.52	(0.98, 0.02, -0.03, -0.199, 1.16, 0.12)
Algorithmus 3	(100, 48, 5, 7, 17, 1)	25.82	0.03	(0.96, 0.30, -0.12, -0.31, 0.97, 0.34)
Algorithmus 4	(66, 24, 21, 15, 18, 8)	29.72	0.03	(0.96, 0.31, -0.12, -0.31, 0.97, 0.34)

Tabelle 4.6: Ergebnisse der Optimierung des Datensatzes HNSP MLIR SSD mfElas

Verfahren	Anzahl der Iterationen pro Level	Laufzeit (in s)	Reduzierung $J(y_k)/J(y_0)$
Gauß-Newton	(3, 3, 3, 3, 2, 2)	12.73	0.01
Gradienten	(30, 63, 46, 84, 88, 100)	163.32	0.01
Algorithmus 1	(4, 4, 5, 3, 4, 2)	24.98	0.02
Algorithmus 2	(2, 3, 3, 2, 2, 2)	47.61	0.29
Algorithmus 3	(7, 8, 5, 5, 4, 1)	22.34	0.02
Algorithmus 4	(6, 7, 7, 6, 2, 2)	16.61	0.02

trierten Bilder des 3D-Brain Datensatzes werden in Abbildung 4.11 gezeigt.



(a) Gauß-Newton-Verfahren (b) Gradienten-Verfahren

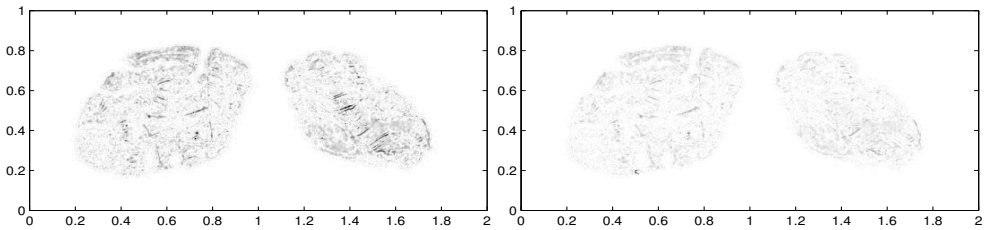
(c) Nesterov-Algorithmus aus Paper [11] (d) Nesterov-General-Scheme

(e) Nesterov-Constant-Step-Scheme 1 (f) Nesterov-Constant-Step-Scheme 2

Abbildung 4.8: Grafischer Vergleich der verschiedenen Optimierungsverfahren des Beispiels HNSP MLPIR SSD affine 2D

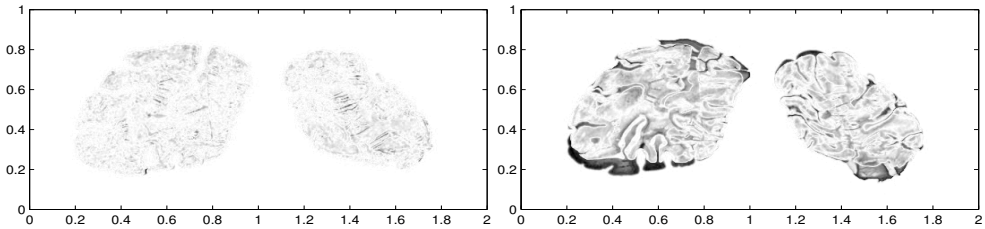
Tabelle 4.7: Ergebnisse der Optimierung des Datensatzes 3D-Brain

Verfahren	Anzahl der Iterationen pro Level	Laufzeit (in s)	Reduzierung $J(y_k)/J(y_0)$
Gauß-Newton	(6, 8, 10, 10)	230.96	0.35
Gradienten	(14, 15, 16, 14)	212.70	0.35
Algorithmus 1	(14, 11, 13, 11)	263.60	0.35
Algorithmus 2	(2, 1, 1, 1)	106.30	0.54
Algorithmus 3	(12, 21, 5, 22)	871.66	0.34
Algorithmus 4	(26, 19, 21, 19)	247.97	0.35



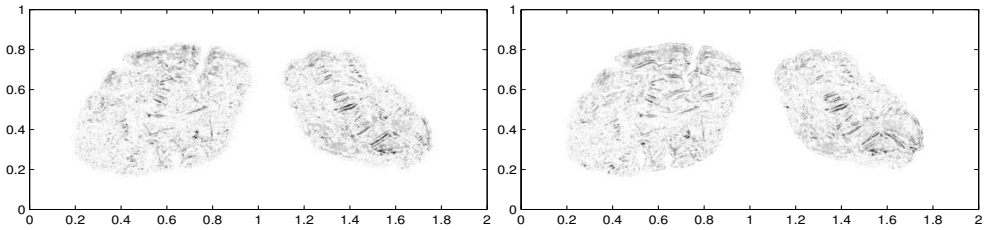
(a) Gauß-Newton-Verfahren

(b) Gradienten-Verfahren



(c) Nesterov-Algorithmus aus Paper [11]

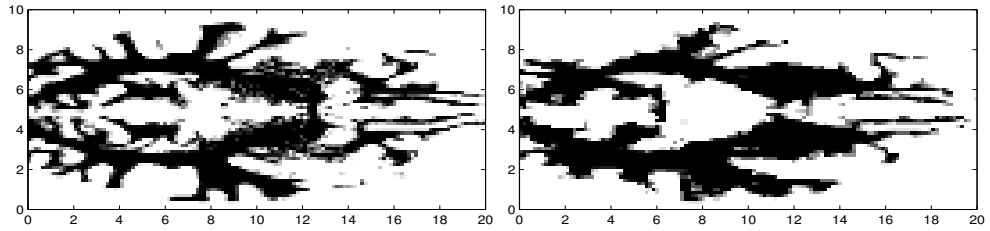
(d) Nesterov-General-Scheme



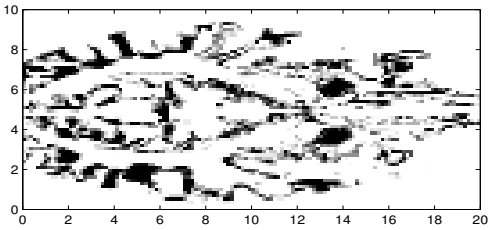
(e) Nesterov-Constant-Step-Scheme 1

(f) Nesterov-Constant-Step-Scheme 2

Abbildung 4.9: Grafischer Vergleich der verschiedenen Optimierungsverfahren anhand des Beispiels HNSP MLIR SSD mfElas

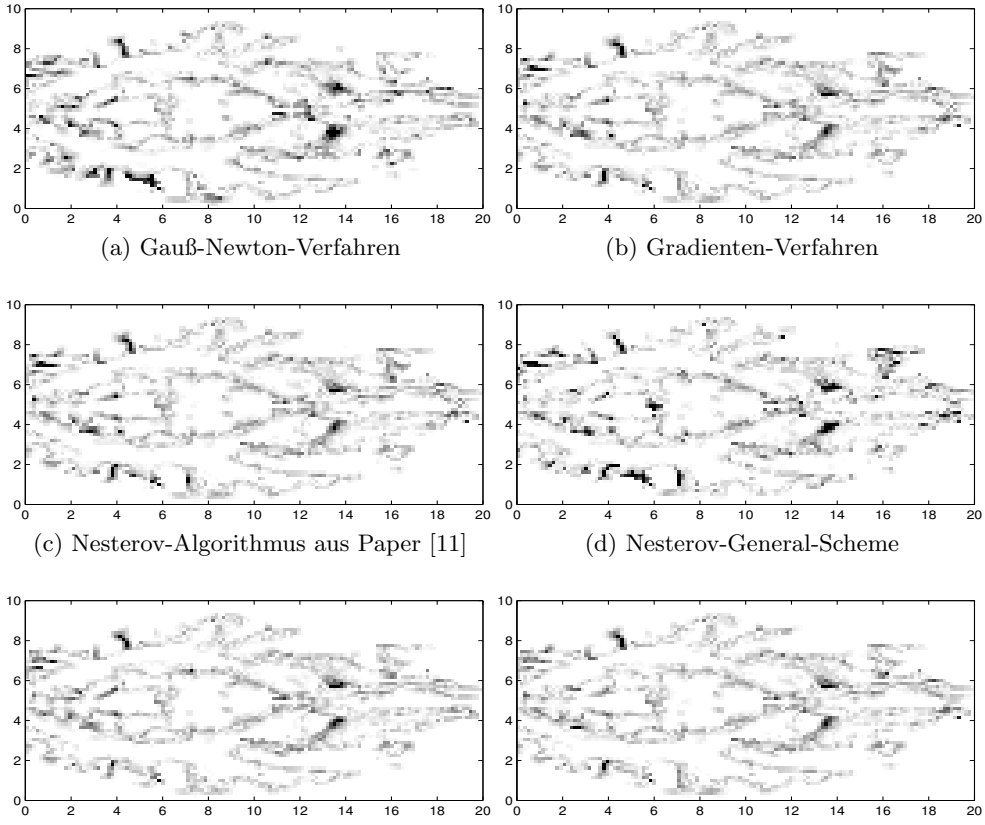


(a) T (b) R



(c) $|T - R|$

Abbildung 4.10: Templatebild T, Referenzbild R und initiale Betragsdifferenz $|T - R|$ des 3D-Brain Datensatzes



(a) Gauß-Newton-Verfahren

(b) Gradienten-Verfahren

(c) Nesterov-Algorithmus aus Paper [11]

(d) Nesterov-General-Scheme

(e) Nesterov-Constant-Step-Scheme 1

(f) Nesterov-Constant-Step-Scheme 2

Abbildung 4.11: Grafischer Vergleich der verschiedenen Optimierungsverfahren anhand des Beispiels 3D-Brain

5 Auswertung und Diskussion

In diesem Kapitel werden die erzielten Resultate erörtert und dargelegt, wo die Stärken und Schwächen der einzelnen Algorithmen liegen. Im anschließenden Ausblick werden mögliche Ansätze zur Verbesserung gegeben. Als schlecht wird im Folgenden eine lange Laufzeit, viele Iterationen und einen ungenauer Minimierer bezeichnet.

5.1 Auswertung

Die Rosenbrock-Funktion

Bei dem Beispiel der Rosenbrock-Funktion sollte die Stabilität der Nesterov-Algorithmen bezüglich der Konvexität getestet werden. Hierbei wird deutlich, dass vor allem das Nesterov-Verfahren [11], da es viele Iterationen benötigt und das errechnete Ergebnis sehr weit von dem berechneten Minimierer entfernt liegt, als schlecht beachtet wird. Aber auch die restlichen Nesterov-Algorithmen erreichen nicht das Minimum 0, sondern brechen beim errechneten Minimum von 5.05, 4.01 und 4.11 ab. Positiv hervorzuheben ist bei dem Beispiel der Rosenbrock-Funktion vor allem das Gradienten-Verfahren. Dieses löst das Minimierungsproblem in einer schnelleren Laufzeit als das Newton-Verfahren und ist näher an x^*. Dafür benötigt das Gradienten-Verfahren allerdings 48 Iterationen, wohingegen das Newton-Verfahren nur 13 Iterationen benötigt.

Die quadratische Funktion

Das Beispiel der quadratischen Funktion ist ein streng-konvexes Beispiel. Hierbei wird Nesterovs Annahme nicht verletzt. Das beste Ergebnis unter den Nesterov-Algorithmen erzielte das Nesterov-General-Scheme mit 2 Iterationen und einer Laufzeit von 0.004 Sekunden. Nur noch das Newton-Verfahren benötigt weniger Iterationen und hat eine geringere Laufzeit mit 0.0005 Sekunden. Am schlechtesten war bei dem Beispiel der quadratischen Funktion das Nesterov-Constant-Step-Scheme 2. Dieses benötigte hundert Iterationen, hatte dadurch die längste Laufzeit mit 0.006 Sekunden und erzielte das schlechteste Ergebnis mit einem Minimum von -50.48.

Hands MLPIR

Für das Beispiel Hands MLPIR ist das Nesterov-Constant-Step-Scheme 1 annähernd so gut wie das Gauß-Newton-Verfahren. Es benötigt zwar mehr Iterationen pro Level und hat dadurch eine schlechtere Laufzeit, dafür aber eine ähnliche Reduzierung. Dies wird auch grafisch in Abbildung 4.4 deutlich. Während alle anderen Hände hauptsächlich auf der unteren Handfläche übereinander liegen, stimmen beim Nesterov-Constant-Step-Scheme 1 sogar die Finger größtenteils überein. Die anderen Nesterov-Verfahren sind eher mit dem Gradienten-Verfahren zu vergleichen, benötigen allerdings weniger Iterationen um ein ähnliches Ergebnis wie das des Gradienten-Verfahrens zu erzielen. Dieser Vorteil wird auch an der Laufzeit deutlich. Während das Gradienten-Verfahren 44.19 Sekunden benötigt, ist die schlechteste Laufzeit unter den Nesterov-Algorithmen das Nesterov-Verfahren [11] mit 22.12 Sekunden.

Hands MLIR SSD mfElas

Bei dem Beispiel Hands MLIR SSD mfElas ist das Nesterov-Constant-Step-Scheme 1 von der Reduzierung zwischen dem Gauß-Newton-Verfahren und dem Gradienten-Verfahren. Auch die anderen Nesterov-Algorithmen erzielen ein mit dem Gradienten-Verfahren vergleichbares Ergebnis. Einzig das Nesterov-General-Scheme hat nur eine Reduzierung von 0.35. Dies wird auch grafisch deutlich. Während bei allen anderen Algorithmen kaum noch das Referenzbild in Abbildung 4.5 zu erkennen ist, ist beim Nesterov-General-Scheme ein Teil der unteren Handfläche noch zu erkennen. In Abbildung 4.6 ist zu erkennen, dass ein niedriger Zielfunktionswert nicht unbedingt eine gute Registrierung bedeutet. Die Hand hat für alle Verfahren außer dem Gauß-Newton-Verfahren 6 Finger. Der Beitrag des Regularisiers zur Zielfunktion J war zu gering und α hätte erhöht werden müssen. Die schlechte Registrierung ist nicht allein Schuld der Optimierungsverfahren.

HNSP MLPIR SSD affine2D

Das Beispiel HNSP MLPIR SSD affine2D zeigt, dass bis auf das Nesterov-General-Scheme alle anderen Verfahren annähernd die gleiche Reduzierung haben. Einzig in der Laufzeit sind Unterschiede erkennbar. So ist das Gauß-Newton-Verfahren mit 7.02 Sekunden das schnellste, das Gradienten-Verfahren mit 120.39 Sekunden das langsamste. Das Gauß-Newton-Verfahren benötigte allerdings auch die wenigsten Iterationen, wohingegen das Gradienten-Verfahren die meisten brauchte. Das Nesterov-Constant-Step-Scheme 1 hatte unter den Nesterov-Algorithmen die beste Laufzeit mit 35.82 Sekunden. Das Nesterov-Verfahren [11] die schlechteste mit 70.59 Sekunden. Das Nesterov-General-Scheme lieferte das schlechteste Ergebnis, allerdings brach das Verfahren auch bei der Schrittweitenbestimmung zwischenzeitlich ab, was zu der niedrigen Iterationenanzahl führte. Das schlechte Ergebnis des Nesterov-General Scheme

wird auch in Abbildung 4.8 d) erkennbar.

HNSP MLPIR SSD mfElas

Auch bei dem Beispiel HNSP SSD mfElas sind die Ergebnisse der Reduzierung bei den Nesterov-Algorithmen, mit Ausnahme des Nesterov-General-Scheme, nur minimal schlechter als das Gauß-Newton-Verfahren und das Gradienten-Verfahren. Das Nesterov-General-Scheme allerdings erreicht nur eine Reduzierung von 0.29 was wiederrum auf die Schrittweitenbestimmung zurückzuführen ist. Bei den Laufzeiten ist das Nesterov-Constant-Step-Scheme 1 mit 16.61 Sekunden dem Gauß-Newton-Verfahren mit 12.73 Sekunden sehr ähnlich.

3D-Brain

Bei dem Beispiel 3D-Brain sieht man in Tabelle 4.7, dass alle Verfahren, mit Ausnahme des Nesterov-General-Scheme annähernd gleich gut sind. Das Nesterov-Constant-Step-Scheme 1 allerdings benötigt für die Registrierung 803.05 Sekunden. Das Gradienten-Verfahren hingegen nur 211.97 Sekunden. Das Nesterov-General-Scheme brach vorzeitig wegen fehlender Schrittweitenbestimmung ab und lieferte somit auch bei diesem Beispiel kein gutes Ergebnis.

5.2 Diskussion

Der Nesterov Algorithmus aus Paper [11]

Ein Vorteil des Nesterov-Algorithmus [11] ist, dass er nicht von der Lipschitzkonstante L und dem Parameter für stark-konvexe Funktionen μ abhängt, da diese Paramameter zu grob abgeschätzt werden können. Allerdings muss bei dem Algorithmus ein Punkt $z \in \mathbb{R}^n$ gewählt werden, über den Nesterov neben der Aussage, dass $z \neq y_0$ ist, keine weiteren Annahmen hat, sodass nicht deutlich wird, wie z geschickt zu wählen ist. Im praktischen Vergleich fällt auf, dass der Algorithmus verhältnismäßig viele Iterationen benötigt und dadurch in der Praxis eine schlechte Konvergenz hat. Ein weiterer Nachteil, der vor allem bei dem akademischen Beispiel der Rosenbrock-Funktion auffiel, ist die Konzeption für konvexe Funktionen. Ist die Zielfunktion nicht konvex, können keine vernünftigen Ergebnisse erzielt werden. Dies kann auch in der medizinischen Bildregistrierung zum Problem werden, wenn keine genaue Aussage über die Konvexität der zu optimierenden Zielfunktion gemacht werden kann. Auch bei den Datensätzen aus FAIR fällt auf, dass der Nesterov-Algorithmus qualitativ dem Gradienten-Verfahren ähnelt, aber nur bedingt bessere Laufzeiten liefert.

Das Nesterov-General-Scheme

Das Nesterov-General-Scheme liefert bei konvexen Beispielen mit bekannter Lipschitzkonstante und bekanntem Konvexitätsparameter μ eine annähernd lokal quadratische Konvergenz. Dies sieht man an einem ähnlichen Iterationsverlauf wie beim Newton-Verfahren für das akademische Beispiel der Rosenbrock-Funktion. Darüber hinaus benötigt die Methode bei den Datensäzen aus FAIR auf jedem Level sehr wenige Iterationen. Bei nicht konvexen Beispielen oder bei unbekannter Konvexität liefert das Nesterov-General-Scheme keine genauen Ergebnisse.

Nesterov-Constant-Step-Scheme 1

Das Nesterov-Constant-Step-Scheme 1 benötigt keine Schrittweitenbestimmung nach Armijo. Das ermöglicht einen guten Fortschritt je Iteration auch an Stellen, wo durch „Zig-Zagging-Effekte“ oder ähnliches die Schrittweite sehr oft iteriert werden muss. Das Verfahren liefert eine gute Lösung auch bei der Rosenbrock-Funktion, obwohl diese nicht konvex ist. Allerdings wird bei diesem Verfahren das Wissen von L und μ vorausgesetzt. Bei den Datensätzen aus FAIR wird allerdings deutlich, dass das Constant-Step-Scheme 1, obwohl es annähernd gute Lösungen liefert, im Vergleich mit dem Gauß-Newton-Verfahren durch die hohe Anzahl von Iterationen und die Notwendigkeit L zu bestimmen, langsam ist.

Nesterov-Constant-Step-Scheme 2

Wie auch das Nesterov-Constant-Step-Scheme 1 benötigt das Nesterov-Constant-Step-Scheme 2 keine Schrittweitenbestimmung. Allerdings wird bei dem akademischen Beispiel der Rosenbrock-Funktion deutlich, dass das Nesterov-Constant-Step-Scheme 2 bei einer nicht streng-konvexen Funktion kein vernünftiges Ergebnis liefert. Des Weiteren ist das Nesterov-Constant-Step-Scheme 2 im Vergleich mit dem Gauß-Newton-Verfahren eine mittelmäßige Lösung.

Fazit

Alles in allem ist das Nesterov-Constant-Step-Scheme 1 eine gute Alternative zum Newton-Verfahren, da es in den meisten Fällen näher am Referenzbild bzw am Minimierer war und eine bessere Laufzeit hatte. Ein besseres Ergebnis, als des Gauß-Newton-Verfahren erreichte das Nesterov-Constant-Step-Scheme 1 allerdings nur beim Datensatz 3D-Brain und bei dem Beispiel hatte es eine im Vergleich zum Gauß-Newton-Verfahren langsame Laufzeit. Das Nesterov-Verfahren [11] und das Nesterov-General-Scheme hingegen erreichten meistens die Genauigkeit des Gradienten-Verfahren, waren allerdings nicht besser. Das Nesterov-Constant-Step-Scheme 2 war bei einigen Testfällen gut, bei den meisten allerdings weniger und im Vergleich erzielte es die ungenauesten Ergebnis-

se, da es selten bis zum Ende durchlief, sondern meistens wegen der fehlenden Schrittweitenberechnung abbrach. Dies ist als negativ zu bewerten.

5.3 Ausblick

Es gibt einige mögliche Verbesserungen der Umsetzung der Algorithmen, die nun aufgezählt werden. Ein Problem in der Implementierung war das Abschätzen des Konvexitätsparameters und der Lipschitzkonstanten. Da die Lipschitzkonstante in der aktuellen Implementierung in jeder Iteration neu berechnet wird, wird die Laufzeit verschlechtert. Ein weiterer Ansatzpunkt ist der Konvexitätsparameter μ, der mit 0 gewählt wurde. Auch hier könnte man die Implementierung verbessern, indem man versucht μ geschickt zu wählen.
Eine weitere Möglichkeit zur Optimierung könnte die Wahl von z im Nesterov-Algorithmus sein. Für ein geschickt gewähltes z kann man eventuell bessere Ergebnisse erzielen als für ungschickt gewählte z, da sich bei unseren Experimenten die Wahl von z als nicht unerheblich herausgestellt hat. Für das Nesterov-Verfahren [11] und das General-Scheme sind eventuell andere Schrittweitenbestimmungen wie zum Beispiel Goldstein oder Wolfe-Powell [12] sinnvoll. Der Versuch mit Nesterovs Verfahren eine Brücke zwischen dem Gradienten und dem (Gauß-)Newton-Verfahren zu schlagen, ist in der Theorie möglich, in der praktischen Umsetzung allerdings fehlen klarere Bestimmungsvorgaben für μ und L und eine Ausdehnung auf sehr große Optimierungsprobleme, wie sie zum Beispiel in dem EMPIRE10-Wettbewerb bestehen, sollte durchgeführt werden, um eventuell eine Beschleunigung der Optimierung erreichen zu können. Des Weiteren ist die Annahme einer strikt-konvexen Funktion nicht immer realisierbar.

Literaturverzeichnis

[1] Brian B Avants, Charles L Epstein, Murray Grossman und James C Gee, Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain, in *Medical Image Analysis*, 12(1):26–41, 2008.

[2] Stephen Boyd und Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2009.

[3] Bernd Fischer und Jan Modersitzki, Ill-posed medicine - an introduction to image registration, in *Inverse Problems*, 24(3):034008, 2008.

[4] Gerd Fischer, *Lineare algebra: Eine Einführung für Studienanfänger*, Springer-Verlag, 2008.

[5] Philip E Gill, Walter Murray und Margaret H Wright, Practical optimization, in , 1981.

[6] Heinz Handels, *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie*, Springer-Verlag, 2009.

[7] Rupert Lasser und Frank Hofmaier, *Analysis 1+2: Ein Wegweiser zum Studienbeginn*, Springer, 2012.

[8] Jan Modersitzki, *FAIR: flexible algorithms for image registration*, 6, SIAM, 2009.

[9] Keelin Murphy, Bram Van Ginneken, Joseph M Reinhardt, Sven Kabus, Kai Ding, Xiang Deng, Kunlin Cao, Kaifang Du, Gary E Christensen, Vincent Garcia et al., Evaluation of registration methods on thoracic CT: the EMPIRE10 challenge, in *Medical Imaging, IEEE Transactions on*, 30(11):1901–1920, 2011.

[10] Yu Nesterov, Introductory Lectures on Convex Programming Volume I: Basic course, in *Center for Operations Research and Econometrics, Universit e catholique de Louvain, Louvain-la-Neuve, Belgium*, 1996.

[11] Yurii Nesterov, A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$, in *Soviet Mathematics Doklady*, 27, 372–376, 1983.

[12] Jorge Nocedal und SJ Wright, Numerical optimization, in *Springer*, 2006.

[13] Gang Song, Nicholas Tustison, Brian Avants und James C Gee, Lung CT image registration using diffeomorphic transformation models, in *Medical*

Image Analysis for the Clinic: A Grand Challenge, 23–32, 2010.

[14] Christiane Tretter, *Analysis I*, Birkhäuser Springer, 2013.

[15] S V N Vishwanathan, Notes on Optimization, 2010, abgerufen am 04.11.2014.