Ein Filter-Trust-Region-Verfahren für nichtlineare Gleichungssysteme

Diplomarbeit

von

Markus Kaiser

Betreuerin

Prof. Dr. Kathrin Klamroth

5. August 2008

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Angewandte Mathematik II

Inhaltsverzeichnis

Kapitel 1

Einführung

Nichtlineare Gleichungssysteme (NLG) treten bei den verschiedensten mathematischen, physikalischen, technischen oder wirtschaftswissenschaftlichen Problemstellungen auf. Im Gegensatz zu linearen Gleichungssystemen kann a priori keine Aussage über die Lösbarkeit eines nichtlinearen Gleichungssystems getroffen werden. Auch ist es im Allgemeinen nicht möglich, ein NLG durch algebraische Umformungen zu lösen. Aus diesem Grund bedient man sich in der Regel eines iterativen Verfahrens, um die Lösung eines NLGs auf numerischem Wege zu ermitteln. Ähnlich vielfältig wie die Situationen in denen ein nichtlineares Gleichungssystem auftreten kann, sind die Methoden es zu lösen. Zu den bekanntesten gehören das Newton-Verfahren mit seinen vielfältigen Erweiterungen und Varianten, das Gauss-Newton-Verfahren, das Gradientenverfahren, auch Verfahren des steilsten Abstiegs genannt, und das Halley-Verfahren.
In dieser Diplomarbeit soll ein Verfahren vorgestellt werden, das die Vorzüge der Trust-Region-Verfahren und der relativ jungen Filter-Methoden miteinander verbindet. Das so entstehende Filter-Trust-Region-Verfahren (FTR) konvergiert aufgrund der verwendeten Trust-Region-Techniken global gegen eine lokale Minimalstelle der Zielfunktion. Die Verwendung des mehrdimensionalen Filters dagegen gewährleistet die Zulässigkeit der ermittelten Lösung und kann die benötigte Rechenzeit reduzieren, da überflüssige Iterationen vermieden werden.
In Kapitel 2 werden die notwendigen theoretischen Vorbereitungen zur Entwicklung des eigentlichen Algorithmus dargestellt. Dazu wird zunächst ein grundlegendes Trust-Region-Verfahren erläutert, dessen Konvergenzbeweis die Basis für die Konvergenzanalyse des FTR-Algorithmus bildet. Anschließend wird das Prinzip des Filters anhand eines Filter-SQP-Verfahrens eingeführt. Ein Abschnitt über die Grundlagen der multikriteriellen Optimierung, durch den der theoretische Hintergrund für den mehrdimensionalen Filter vorbereitet wird, bildet den Abschluss dieses Kapitels. Kapitel 3 ist dem FTR-Algorithmus mitsamt der Einführung des mehrdimensionalen Filters, einer die Konvergenz sichernden Akzeptanzbedingung, ersten Überlegungen zur Testschrittbestimmung und der Konvergenzanalyse vorbehalten.
Kapitel 4 stellt eine Implementierung des FTR-Algorithmus in Matlab vor, in der

verschiedene Modellfunktionen zur Bestimmung eines geeigneten Testpunktes, unterschiedliche Aktualisierungsschritte für den Trust-Region-Radius und weitere variable Parameter zur Beeinflussung der Trust-Region-Updates und des mehrdimensionalen Filters, für eine hohe Anpassungsfähigkeit sorgen. In Kapitel 5 werden diese unterschiedlichen Varianten des Algorithmus mit Hilfe einer Menge von Testbeispielen miteinander verglichen.

Mein besonderer Dank gilt vor allem Herrn Dipl.-Technomath. Alexander Thekale sowie Frau Prof. Dr. Kathrin Klamroth für eine optimale (!) Betreuung und Unterstützung während dieser Diplomarbeit. Desweiteren danke ich Herrn Tobias Rasp und Herrn Dipl.-Ing. Sleman Saliba für die Möglichkeiten meinen Algorithmus an praktischen Beispielen zu erproben. Außerdem möchte ich meiner Freundin Simona Peruzzi für das Finden meiner zahlreichen "nichtmathematischen" Fehler und für ihre Unterstützung in jeder möglichen Hinsicht danken.

Kapitel 2

Literaturüberblick und Algorithmen

In diesem Kapitel werden die beiden wesentlichen Ideen des Filter-Trust-Region-Algorithmus erläutert. Zum einen wird ein grundlegender Trust-Region-Algorithmus vorgestellt und dessen Konvergenzeigenschaften untersucht, da dies die Basis für die Konvergenzanalyse des Filter-Trust-Region-Algorithmus liefert. Außerdem wird das Prinzip des Filters so erläutert, wie es von Roger Fletcher und Sven Leyffer eingeführt wurde. Um den theoretischen Hintergrund für den mehrdimensionalen Filter bereit zu stellen, bildet ein kurzer Abschnitt, der die grundlegenden Begriffe der multikriteriellen Optimierung erläutert, den Abschluss dieses Kapitels.

2.1 Trust-Region-Verfahren

Ein Trust-Region-Verfahren ist eine iterative Methode das Minimum eines nichtlinearen Optimierungsproblems ohne Nebenbedingungen zu bestimmen. Die grundlegende Idee dabei ist, die Zielfunktion iterativ innerhalb eines bestimmten Gebietes um den aktuellen Iterationspunkt durch eine einfacher zu minimierende Modellfunktion zu approximieren. Dieses Gebiet, in dem man der Modellfunktion "vertraut" (engl.: Trust-Region), ist namensgebend für des Verfahren. Die Trust-Region wird folgendermaßen definiert:

Definition 2.1 Trust-Region *wird die Menge $\mathcal{B}_k$ aller Punkte, die innerhalb einer Kugel mit Radius Δ_k um den aktuellen Iterationswert x_k liegen, genannt:*

$$\mathcal{B}_k = \{x \in \mathbb{R}^n | \, \|x - x_k\|_k \leq \Delta_k\}$$

Dabei ist Δ_k der Trust-Region-Radius *und $\|\cdot\|_k$ eine vom Iterationsschritt k abhängige Norm.*

Man hofft, dass sich für einen Punkt, für den die Modellfunktion einen besseren Wert ergibt, als für den aktuellen Iterationspunkt, auch der Wert der Zielfunktion in ähn-

licher Weise verbessert hat. In Abhängigkeit davon, wie genau diese Vorhersage der Verbesserung des Wertes der Zielfunktion durch die Modellfunktion war, kann das Gebiet, in dem man dem Modell vertraut, vergrößert oder verkleinert werden. Damit wird erreicht, dass mit jedem akzeptierten Iterationsschritt der Wert der Zielfunktion kleiner wird, was letztendlich zur Konvergenz des Verfahrens führt.
Das folgende Kapitel basiert überwiegend auf dem sechsten Kapitel des Buches "Trust-Region-Methods" von Andrew Conn, Nicholas Gould und Philippe Toint [4].

2.1.1 Ein grundlegender Trust-Region-Algorithmus

Es sei eine reellwertige zweimal stetig differenzierbare nach unten beschränkte Funktion $f(x)$ gegeben. Ziel des in diesem Kapitel betrachteten Algorithmus ist es, für das unrestringierte Problem

$$\min_{x \in \mathbb{R}} f(x) \tag{2.1}$$

eine Lösung, d.h. ein lokales Minimum zu finden. Dazu wird für den aktuellen Iterationswert x_k des Algorithmus eine Modellfunktion $m_k(x)$ erzeugt, welche die Zielfunktion $f(x)$ innerhalb der Trust-Region approximieren soll.
Nachdem man eine Modellfunktion $m_k(x)$ bestimmt hat, wird mit deren Hilfe ein Testpunkt $x_k + s_k$ berechnet, der sich aus dem letzten Iterationspunkt durch Addition des Testschritts s_k ergibt. Da der Wert der Zielfunktion $f(x_k + s_k)$ im Testpunkt möglichst klein sein soll, wird man den Testschritt so wählen, dass der Wert der Modellfunktion $m_k(x_k + s_k)$ möglichst klein wird. Dabei ist allerdings zu beachten, dass der Testpunkt innerhalb der Trust-Region um x_k liegen muss. Man erhält somit das Ersatzproblem

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & m_k(x_k + s_k) \\ \text{s.t.} & \|s_k\|_k \leq \Delta_k. \end{array}$$

Nun wird die Zielfunktion an der Stelle $x_k + s_k$ ausgewertet und die in der Modellfunktion erreichte Verbesserung mit der Verbesserung der Zielfunktion verglichen. Ist die Vorhersage durch die Modellfunktion ausreichend genau, akzeptiert man den neuen Punkt als nächsten Iterationspunkt und behält den Trust-Region-Radius bei. Im Falle einer sehr genauen Vorhersage kann der Trust-Region-Radius auch vergrößert werden. Stellt man dagegen fest, dass die Modellfunktion das Verhalten der Zielfunktion schlecht modelliert, wird der Testpunkt abgelehnt und der Trust-Region-Radius verkleinert, in der Hoffnung, dass die Modellfunktion in einem kleineren Gebiet eine bessere Vorhersage liefert.
Ausgehend von diesen Überlegungen lässt sich ein grundlegender Trust-Region Algorithmus formulieren, der gegen eine lokale Lösung des Problems (2.1) konvergiert:

Algorithmus 2.1 *Basic Trust-Region-Algorithmus (BTR)*

Schritt 1: Initialisierung
Ein Startwert x_0 und ein anfänglicher Trust-Region-Radius Δ_0 sind gegeben; außerdem die Konstanten η_1, η_2, γ_1 und γ_2, für die gilt:
$0 < \eta_1 \leq \eta_2 < 1$ *und* $0 < \gamma_1 \leq \gamma_2 < 1$.
Berechne $f(x_0)$ und initialisiere den Iterationszähler mit $k = 0$.
Schritt 2: Wahl der Modellfunktion
Wähle eine Norm $\|\cdot\|_k$ und eine Modellfunktion m_k innerhalb der Trust-Region $\mathcal{B}_k$.
Schritt 3: Bestimmen des Testschritts
Berechne einen Testschritt s_k, der den Wert der Modellfunktion m_k ausreichend verringert und der die Bedingung $x_k + s_k \in \mathcal{B}_k$ erfüllt.
Schritt 4: Akzeptanztest
Berechne $f(x_k + s_k)$ und definiere $\rho_k := \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}$.
Ist $\rho_k \geq \eta_1$, dann ist $x_{k+1} = x_k + s_k$, andernfalls ist $x_{k+1} = x_k$.
Schritt 5: Aktualisierung des Trust-Region-Radius
Setze

$$\Delta_{k+1} \in \begin{cases} [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & falls \quad \rho_k < \eta_1 \\ [\gamma_2 \Delta_k, \Delta_k] & falls \quad \rho_k \in [\eta_1, \eta_2) \\ [\Delta_k, \infty) & falls \quad \rho_k \geq \eta_2. \end{cases}$$

Setze $k = k + 1$ und gehe zu Schritt 2.

Es bleibt zunächst offen, welche Werte für die Konstanten η_1, η_2, γ_1 und γ_2 verwendet werden. In [4] werden beispielsweise die Werte $\eta_1 = 0.01$, $\eta_2 = 0.9$ und $\gamma_1 = \gamma_2 = 0.5$ vorgeschlagen.
Für die Konvergenzanalyse dieses Algorithmus ist es nötig, verschiedene Arten von Iterationsschritten zu unterscheiden:

Definition 2.2 *Man nennt alle Iterationsschritte, für die $\rho_k \geq \eta_1$ ist,* erfolgreich. *Gilt sogar $\rho_k \geq \eta_2$ wird der zugehörige Iterationsschritt* sehr erfolgreich *genannt. Alle anderen Iterationsschritte werden als* nicht erfolgreich *bezeichnet. Dementsprechend definiert man die* Menge der erfolgreichen Iterationsschritte $\mathcal{S} = \{k \geq 0 | \rho_k \geq \eta_1\}$ *und die* Menge der sehr erfolgreichen Iterationsschritte $\mathcal{V} = \{k \geq 0 | \rho_k \geq \eta_2\}$.

Auch die Wahl der Modellfunktion innerhalb jeder Iteration des BTR-Algorithmus ist noch nicht spezifiziert. In der Praxis verwendet man beispielsweise eine quadratische Funktion der Form

$$m_k(x_k + s) = m_k(x_k) + \langle g_k, s \rangle + \frac{1}{2} \langle s, H_k s \rangle, \tag{2.2}$$

dabei sei $m_k(x_k) = f(x_k)$, $g_k = \nabla_x f(x_k)$ und H_k eine symmetrische Approximation an die Hessematrix $\nabla_{xx} f(x_k)$. Ist $H_k \neq 0$ nennt man m_k eine Modellfunktion zweiter

Ordnung. Im Allgemeinen sagt man, eine Modellfunktion ist von l-ter Ordnung, wenn die höchste darin auftretende Ableitung die l-te ist.
Im BTR-Algorithmus fehlt des weiteren eine Abbruchbedingung. Eine solche ist in der Praxis natürlich unerlässlich, aber für die Konvergenzanalyse nicht erforderlich, da man dann annehmen kann, dass der Algorithmus eine unendliche Folge $\{x_k\}$ von Iterationswerten erzeugt.
Ein weiterer wichtiger Gesichtspunkt jedes Trust-Region-Algorithmus ist die Form der Trust-Region. Idealerweise sollte diese das Gebiet wiederspiegeln, in dem man annimmt, dass die Modellfunktion die Zielfunktion ausreichend gut approximiert. Da die Trust-Region von der verwendeten Norm erzeugt wird, kann es sinnvoll sein diese problem- und iterationsabhängig zu wählen, was durch die Schreibweise $\|\cdot\|_k$ im Algorithmus angedeutet wird. Die am häufigsten verwendeten Normen sind die Manhattannorm (ℓ_1-Norm), die euklidische Norm (ℓ_2-Norm) und die Tchebychefnorm (ℓ_∞-Norm). Im Folgenden wird die euklidische Norm vereinfacht mit $\|\cdot\|$ bezeichnet.

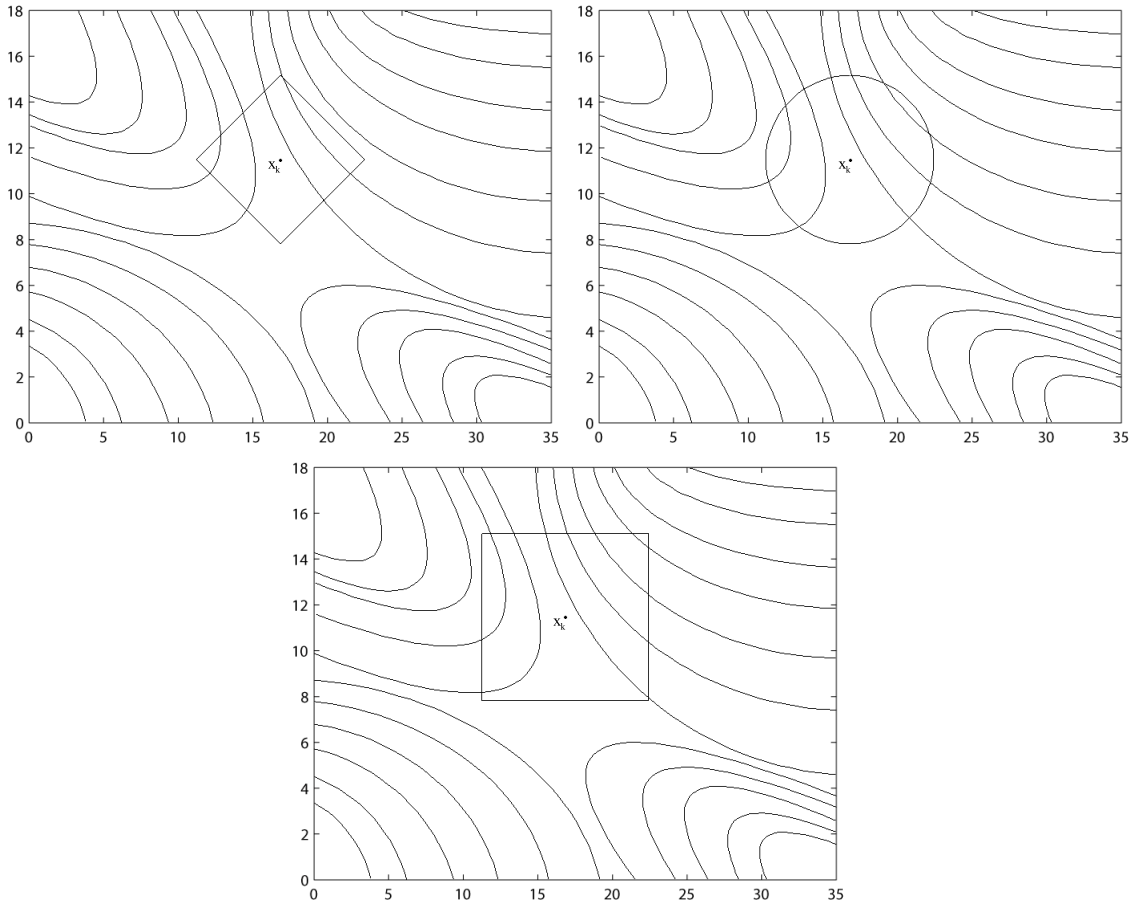


Abbildung 2.1: Unterschiedliche Gestalt der Trust-Region bei Verwendung der ℓ_1 (Darstellung links oben), ℓ_2 (Darstellung rechts oben) bzw. der ℓ_∞ (untere Darstellung) Norm und gleichem Trust-Region-Radius.

2.1.2 Bedingungen für die Konvergenzanalyse

Im Laufe der Konvergenzanalyse werden Annahmen bezüglich der Problemstellung, der Modellfunktion und der verwendeten Norm getroffen. Diese werden hier einmalig aufgelistet, in den folgenden Sätzen und Beweisen wird nur noch auf sie verwiesen.

Die ersten Annahmen betreffen die Problemstellung an sich:
AF 1: Das Minimierungsproblem hat die Form $\min_{x\in\mathbb{R}} f(x)$, wobei die Zielfunktion $f(x)$ zweimal stetig differenzierbar ist.
AF 2: $f(x)$ ist nach unten beschränkt; es existiert demnach eine Konstante $\kappa_{lbf} \in \mathbb{R}$ so, dass für alle $x \in \mathbb{R}$ gilt: $f(x) \geq \kappa_{lbf}$.
AF 3: Die Hessematrix der Zielfunktion ist beschränkt; es existiert also eine Konstante $\kappa_{ufh} > 0$ so, dass für alle $x \in \mathbb{R}$ gilt: $\|\nabla_{xx} f(x)\| \leq \kappa_{ufh}$.

Die folgenden Annahmen betreffen die Modellfunktion:
AM 1: In jedem Iterationsschritt k ist die Modellfunktion m_k innerhalb der $\mathcal{B}_k$ Trust-Region zweimal differenzierbar.
AM 2: Im aktuellen Iterationspunkt x_k stimmen die Werte der Ziel- und der Modellfunktion überein; das heißt, für alle k gilt: $m_k(x_k) = f(x_k)$.
AM 3: Die Gradienten der Ziel- und der Modellfunktion stimmen für alle k in x_k überein: $g_k := \nabla_x m_k(x_k) = \nabla_x f(x_k)$.
AM 4: Die Hessematrix der Modellfunktion ist beschränkt innerhalb der Trust-Region, das heißt: $\|\nabla_{xx} m_k(x)\| \leq \kappa_{umh} - 1$ für alle $x \in B_k$ und für alle k, wobei $\kappa_{umh} \geq 1$ eine Konstante und unabhängig von k ist.

Eine weitere Annahme wird für die unterschiedlichen Normen $\|\cdot\|_k$ der einzelnen Iterationsschritte getroffen:
AN 1: Es gibt eine Konstante $\kappa_{une} \geq 1$ so, dass für alle k und für alle $x \in \mathbb{R}^n$ gilt:

$$\tfrac{1}{\kappa_{une}}\|x\|_k \leq \|x\| \leq \kappa_{une}\|x\|_k.$$

Dabei ist $\|\cdot\| := \|\cdot\|_2$ die euklidische Norm und die Normen $\|\cdot\|_k$ nennt man gleichmäßig äquivalent (engl.: uniformly equivalent) zu $\|\cdot\|$.

2.1.3 Der Cauchy-Punkt und die Verringerung des Wertes der Modellfunktion

Ein entscheidender Punkt im BTR-Algorithmus ist die Bestimmung der Schrittweite s_k, die so gewählt werden sollte, dass der entstehende Testpunkt $x_k + s_k$ den Wert der Modellfunktion innerhalb der Trust-Region ausreichend reduziert.
Eine naheliegende Strategie hierfür ist die Untersuchung des Verhaltens der Modellfunktion entlang des steilsten Abstiegs $-g_k$ innerhalb der Trust-Region $\mathcal{B}_k$. Dabei ist g_k der Gradient der Zielfunktion im aktuellen Iterationspunkt. Die Richtung des steilsten Abstiegs entspricht demnach der Richtung des negativen Gradienten. Wie der

Name vermuten lässt, erwartet man, dass sich der Funktionswert der Modellfunktion entlang dieser Halbgeraden zumindest lokal am schnellsten verringert.

Definition 2.3 *Man nennt die Strecke, die in x_k beginnt, in Richtung des negativen Gradienten verläuft und am Rand der Trust-Region endet,* Cauchy-Bogen *(engl.: Cauchy-arc) $c_k^C(t)$ mit*

$$c_k^C(t) := \{x | x = x_k - t g_k, \quad t \geq 0, \quad x \in \mathcal{B}_k\}.$$

Unter der Annahme, dass eine quadratische Modellfunktion, also ein Funktion der Form (2.2), verwendet wird, ist es möglich, das Minimum von $m_k(x)$ auf dem Cauchy-Bogen eindeutig zu bestimmen.

Definition 2.4 *Das Minimum der Modellfunktion auf dem Cauchy-Bogen*

$$x_k^C = x_k - t_k^C g_k = \arg \min_{\substack{t \geq 0 \\ x_k - t g_k \in \mathcal{B}_k}} m_k(x_k - t g_k) \tag{2.3}$$

wird Cauchy-Punkt *genannt.*

Da in der folgenden Betrachtung die Krümmung der Modellfunktion eine entscheidende Rolle spielt, definiert man eine weitere Konstante

$$\beta_k := 1 + \max_{x \in \mathcal{B}_k} \|\nabla_{xx} m_k(x)\|. \tag{2.4}$$

β_k ist eine obere Schranke für die Krümmung der Modellfunktion innerhalb der Trust-Region. Betrachtet man Bedingung AM 4, sieht man, dass $\beta_k \leq \kappa_{umh}$ gilt.
Die Lage des Cauchy-Punktes ist abhängig von der Krümmung der Modellfunktion. Für eine quadratische Modellfunktion $m_k(x)$ lässt sich dieser Zusammenhang folgendermaßen darstellen: Ist $m_k(x)$ konvex, so liegt ihr Minimum entweder innerhalb der Trust-Region, dann ist es identisch mit dem Cauchy-Punkt, oder außerhalb der Trust-Region, dann liegt der Cauchy-Punkt auf dem Rand von $\mathcal{B}_k$. Ist die Modellfunktion nicht konvex, so liegt der Cauchy-Punkt ebenfalls auf dem Rand der Trust-Region. Der folgende Satz zeigt, dass es eine untere Schranke gibt, um die der Cauchy-Punkt den Wert der Modellfunktion, verglichen mit dem Wert im letzten Iterationspunkt, mindestens verbessert.

Satz 2.1 *Es sei eine quadratische Modellfunktion wie in (2.2) gegeben. Dann gilt:*

$$m_k(x_k) - m_k\left(x_k^C\right) \geq \frac{1}{2} \|g_k\| \min\left(\frac{\|g_k\|}{\beta_k}, \nu_k^C \Delta_k\right)$$

wobei β_k in (2.4) definiert wurde und $\nu_k^C := \frac{\|g_k\|}{\|g_k\|_k}$.

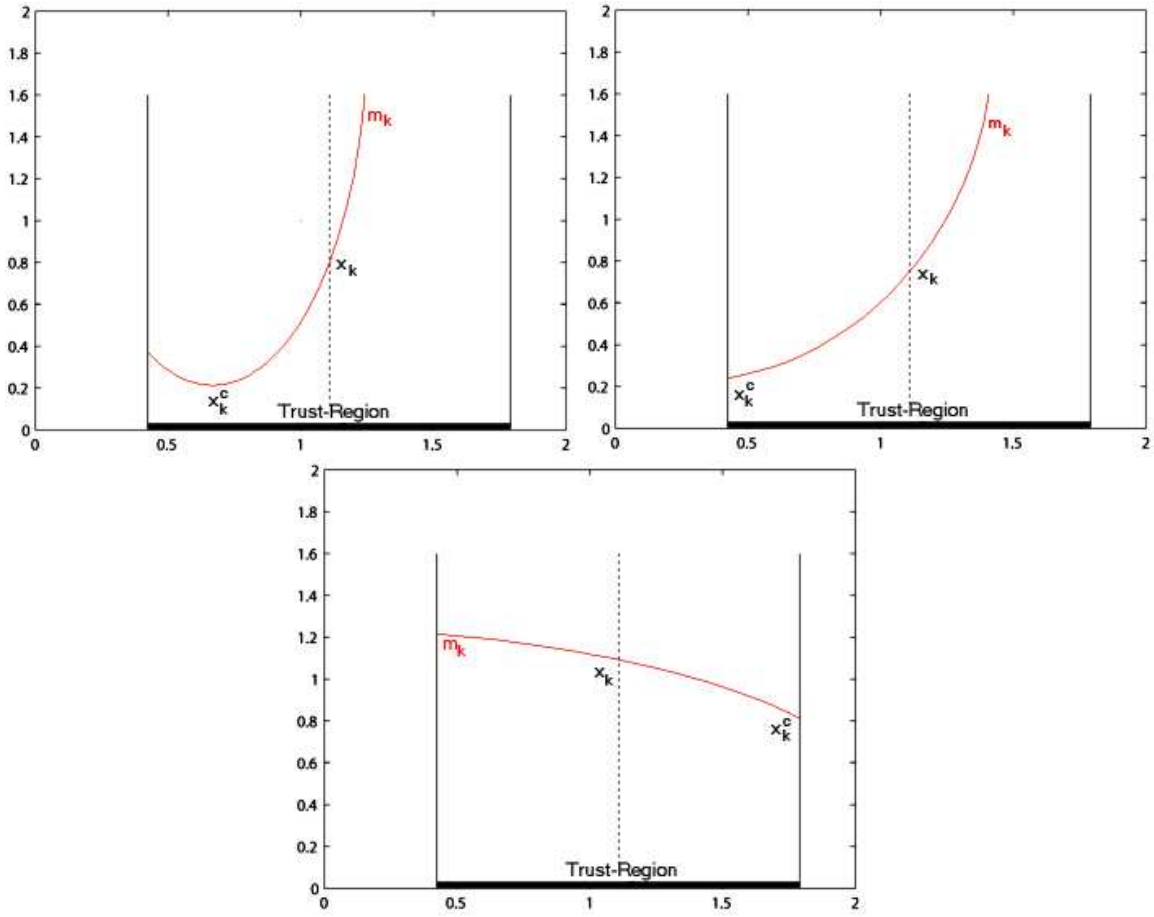


Abbildung 2.2: Darstellung der drei möglichen Fälle für die Lage des Cauchy-Punktes: in den beiden Abbildungen in der oberen Reihe sind die Modellfunktionen konvex, wobei der Cauchy-Punkt im linken Bild innerhalb der Trust-Region und im rechten Bild auf deren Rand liegt; darunter ist eine nicht konvexe Modellfunktion abgebildet.

Beweis 1 Da der Cauchy-Punkt laut Definition auf dem Cauchy-Bogen liegt, kann man die Variable s in der Definition der quadratischen Modellfunktion (2.2) durch $-tg_k$ ersetzen und erhält damit für alle $t \geq 0$:

$$\begin{aligned} m_k\left(x_k - tg_k\right) &= m_k\left(x_k\right) + \langle g_k, -tg_k\rangle + \frac{1}{2}\left\langle -tg_k, H_k\left(-tg_k\right)\right\rangle \\ &= m_k\left(x_k\right) - t\left\|g_k\right\|^2 + \frac{1}{2}t^2\left\langle g_k, H_k g_k\right\rangle \end{aligned} \tag{2.5}$$

Da, wie in Abbildung 2.2 dargestellt, die Lage des Cauchy-Punktes nicht einheitlich ist, muss eine Fallunterscheidung bezüglich der Krümmung von $m_k(x)$ und der Lage von x_k^C vorgenommen werden.
Zunächst betrachtet man den Fall, dass die Krümmung der Modellfunktion in Richtung des steilsten Abstiegs, d.h. in Richtung des negativen Gradienten $-g_k$, positiv ist, also $\langle g_k, H_k g_k\rangle > 0$. In diesem Fall ist m_k über dem Cauchy-Bogen konvex. Daher

gibt es ein eindeutiges Minimum der Modellfunktion und man kann den Wert des Parameters t bestimmen, für den es angenommen wird. Sei m_k also für t_k^* minimal. Für den Wert der ersten Ableitung von (2.5) nach t im Minimum gilt $0 = \|g_k\|^2 - t_k^* \langle g_k, H_k g_k \rangle$. Daraus folgt für t_k^*:

$$t_k^* = \frac{\|g_k\|^2}{\langle g_k, H_k g_k \rangle}. \tag{2.6}$$

Nun können zwei Unterfälle auftreten. Liegt das Minimum innerhalb der Trust-Region, gilt also $t_k^* \|g_k\|_k \leq \Delta_k$, so kann man aus (2.5) folgern:

$$\begin{aligned}
& & m_k(x_k - t g_k) &= m_k(x_k) - t\|g_k\|^2 + \tfrac{1}{2}t^2 \langle g_k, H_k g_k \rangle \\
& \overset{t=t_k^*=t_k^C}{\Rightarrow} & m_k(x_k^C) &= m_k(m_k) - t_k^*\|g_k\|^2 + \tfrac{1}{2}(t_k^*)^2 \langle g_k, H_k g_k \rangle \\
& \overset{2.6}{\Rightarrow} & m_k(x_k) - m_k(x_k^C) &= \tfrac{\|g_k\|^4}{\langle g_k, H_k g_k \rangle} - \tfrac{1}{2}\tfrac{\|g_k\|^4}{\langle g_k, H_k g_k \rangle} \\
& & &= \tfrac{1}{2}\tfrac{\|g_k\|^4}{\langle g_k, H_k g_k \rangle} \geq \tfrac{\|g_k\|^2}{2\beta_k}.
\end{aligned}$$

Die zweite Zeile folgt aus der ersten, da im hier betrachtetem Fall der Punkt, in dem die Modellfunktion minimal ist, dem Cauchy-Punkt entspricht; es gilt also $t_k^C = t_k^*$. t_k^C ist dabei der Wert für t, der die Gleichung $x_k^C = x_k - t g_k$ erfüllt. Die dritte Zeile folgt durch Einsetzen des oben bestimmten Terms für t_k^*. Die letzte Ungleichheit folgt aus

$$\langle g_k, H_k g_k \rangle \overset{C.S.U.}{\leq} \|g_k\| \cdot \|H_k g_k\| \leq \|g_k\|^2 \cdot \|\nabla_{xx} m_k(x_k)\| \overset{2.4}{\leq} \|g_k\|^2 \beta_k.$$

Im zweiten Unterfall liegt das Minimum der Modellfunktion nicht innerhalb der Trust-Region und somit ist $t_k^* \|g_k\|_k > \Delta_k$. Deshalb gilt für den Cauchy-Punkt: $t_k^C \|g_k\|_k = \Delta_k$. Für den optimalen Parameter t_k^* gilt weiterhin (2.6), weswegen man folgern kann:

$$\begin{aligned}
& & t_k^* &= \tfrac{\|g_k\|^2}{\langle g_k, H_k g_k \rangle} \\
& \overset{t_k^*\|g_k\|_k > \Delta_k}{\Rightarrow} & \tfrac{\Delta_k}{\|g_k\|_k} &< \tfrac{\|g_k\|^2}{\langle g_k, H_k g_k \rangle} \\
& \overset{t_k^C\|g_k\|_k = \Delta_k}{\Rightarrow} & t_k^C &< \tfrac{\|g_k\|^2}{\langle g_k, H_k g_k \rangle} \\
& \Rightarrow & \langle g_k, H_k g_k \rangle &< \tfrac{\|g_k\|^2}{t_k^C}
\end{aligned}$$

Mit Hilfe dieser Ungleichung kann man, ähnlich wie im vorherigen Fall, die Verbesserung, die durch den Cauchy-Punkt erreicht wird, abschätzen:

$$\begin{aligned}
m_k(x_k) - m_k(x_k^C) &= t_k^C \|g_k\|^2 - \frac{1}{2}\left(t_k^C\right)^2 \langle g_k, H_k g_k \rangle \\
&> \nu_k^C \|g_k\| \Delta_k - \frac{1}{2}\nu_k^C \|g_k\| \Delta_k \\
&= \frac{1}{2}\nu_k^C \|g_k\| \Delta_k,
\end{aligned}$$

wobei die Ungleichung aus

$$t_k^C \|g_k\|^2 = \frac{\|g_k\|}{\|g_k\|_k} \|g_k\| \, t_k^C \|g_k\|_k = \nu_k^C \|g_k\| \, \Delta_k$$

und

$$\frac{1}{2} \langle g_k, H_k g_k \rangle \left(t_k^C\right)^2 < \frac{1}{2} \frac{\|g_k\|^2}{t_k^C \|g_k\|_k} \left(t_k^C\right)^2 \|g_k\|_k = \frac{1}{2} \nu_k^C \|g_k\| \, \Delta_k$$

folgt.
Bisher wurde die Werteverbesserung bezüglich einer konvexen Modellfunktion untersucht, nun bleibt noch der zweite Fall zu betrachten, bei dem die Krümmung der Modellfunktion entlang des steilsten Abstiegs negativ oder null ist und demzufolge $\langle g_k, H_k g_k \rangle \leq 0$ gilt. Dann erhält man aus (2.5):

$$\begin{aligned} m_k \left(x_k - t g_k\right) &= m_k \left(x_k\right) - t \|g_k\|^2 + \frac{1}{2} t^2 \langle g_k, H_k g_k \rangle \\ &\leq m_k \left(x_k\right) - t \|g_k\|^2 \end{aligned} \tag{2.7}$$

für alle $t \geq 0$. Da im nicht konvexen Fall der Cauchy-Punkt auf dem Rand der Trust-Region liegt, gilt $t_k^C \|g_k\|_k = \Delta_k$. Das ergibt zusammen mit (2.7) folgende Abschätzung:

$$\begin{aligned} m_k \left(x_k\right) - m_k \left(x_k^C\right) &\geq t_k^C \|g_k\|^2 = \|g_k\|^2 \frac{t_k^C \|g_k\|_k}{\|g_k\|_k} = \|g_k\|^2 \frac{\Delta_k}{\|g_k\|_k} \\ &= \nu_k^C \|g_k\| \, \Delta_k \geq \frac{1}{2} \nu_k^C \|g_k\| \, \Delta_k \end{aligned}$$

Man hat nun für jeden möglichen Fall gezeigt, dass die durch den Cauchy-Punkt erreicht Verbesserung größer oder gleich einer unteren Schranke ist. Da im Satz behauptet wird, dass die Verbesserung in jedem Fall größer oder gleich dem Minimum der drei Schranken ist, ist damit auch die Aussage des Satzes gezeigt. □

Bei der bisherigen Betrachtung wurde davon ausgegangen, dass die Modellfunktion quadratisch ist. Setzt man dies nicht voraus, kann eine exakte Minimalstelle der Modellfunktion möglicherweise nicht mehr so einfach ermittelt werden. Deshalb begnügt man sich mit einer Approximation. Hier bietet sich das Schrittweitenverfahren von Armijo [5] an. Bei diesem Verfahren wird ausgehend von einem Startpunkt entlang einer Suchrichtung ein Punkt ermittelt, der den Wert der gegebenen Funktion ausreichend reduziert. Dabei variiert man die Schrittweite, um die die Testpunkte vom Startpunkt entfernt sind, so lange bis man einen geeigneten Punkt gefunden hat.
Angewandt auf die Suche nach dem Minimum der Modellfunktion entlang des Cauchy-Bogens bedeutet das: Man bestimmt die kleinste nichtnegative ganze Zahl $j = j_c$, für die der Punkt

$$x_k(j) := x_k - \kappa_{bck}^j \frac{\Delta_k}{\|g_k\|_k} g_k \tag{2.8}$$

die Bedingung

$$m_k\left(x_k(j)\right) \leq m_k\left(x_k\right) + \kappa_{ubs}\left\langle g_k, x_k(j) - x_k\right\rangle \tag{2.9}$$

erfüllt. Dabei sind $\kappa_{bck} \in (0,1)$ und $\kappa_{ubs} \in (0, \frac{1}{2})$ Konstanten. Dieses Verfahren wird oft auch als backtracking-Verfahren bezeichnen, da man vom Punkt $x_k(0)$, der auf dem Rand der Trust-Region liegt, rückwärts in Richtung des letzten Iterationspunktes x_k sucht.

Definition 2.5 *Der Punkt $x_k^{AC} := x_k\left(j_c\right)$ wird als* approximierter Cauchy-Punkt *bezeichnet.*

Im nächsten Satz wird, wie bereits für den exakten Cauchy-Punkt geschehen, eine Abschätzung für die Verbesserung des Modellfunktionswertes in einem Iterationsschritt vom aktuellen Wert x_k zum approximierten Cauchy-Punkt hergeleitet.

Satz 2.2 *Unter der Annahme AM 1 ist der approximierte Cauchy-Punkt x_k^{AC} wohldefiniert, das bedeutet, dass es ein endliches j_c gibt, für das der Punkt $x_k^{AC} = x_k\left(j_c\right)$ die Bedingung (2.9) erfüllt. Darüber hinaus gilt folgende Ungleichung:*

$$m_k\left(x_k\right) - m_k\left(x_k^{AC}\right) \geq \kappa_{dcp}\left\|g_k\right\| \min\left(\frac{\left\|g_k\right\|}{\beta_k}, \nu_k^C \Delta_k\right)$$

wobei $\kappa_{dcp} \in (0,1)$ eine vom Iterationsschritt k unabhängige Konstante ist und β_k sowie ν_k^C wie in Satz 2.1 definiert sind.

Beweis 2 Für die nichtnegative ganze Zahl j sei die Bedingung (2.9) nicht erfüllt, somit gilt:

$$m_k\left(x_k(j)\right) > m_k\left(x_k\right) + \kappa_{ubs}\left\langle g_k, x_k(j) - x_k\right\rangle.$$

Setzt man für $x_k(j)$ die Definition (2.8) ein und führt die Variable t_j mit $t_j = \frac{\kappa_{bck}^j \Delta_k}{\left\|g_k\right\|_k}$ ein, so erhält man die Ungleichung:

$$m_k\left(x_k - t_j g_k\right) > m_k\left(x_k\right) - \kappa_{ubs} t_j \left\|g_k\right\|^2, \tag{2.10}$$

da

$$-\kappa_{ubs} t_j \left\|g_k\right\|^2 = \kappa_{ubs}\left\langle g_k, -g_k t_j\right\rangle \overset{(2.8)}{=} \kappa_{ubs}\left\langle g_k, x_k(j) - x_k\right\rangle.$$

Wendet man nun den Satz von Taylor auf die linke Seite der Ungleichung (2.10) an, so gilt für einen Punkt $\xi_j \in [x_k, x_k - t_j g_k]$:

$$m_k\left(x_k\right) - t_j \left\|g_k\right\|^2 + \frac{1}{2} t_j^2 \left\langle g_k, \nabla_{xx} m_k\left(\xi_j\right) g_k\right\rangle > m_k\left(x_k\right) - \kappa_{ubs} t_j \left\|g_k\right\|^2. \tag{2.11}$$

Andererseits gilt wegen $\xi_j \in \mathcal{B}_k$ und (2.4) die Ungleichung $\left\|\nabla_{xx} m_k\left(\xi_j\right)\right| < \beta_k$. Damit kann man die linke Seite von (2.11) abschätzen durch:

$$m_k\left(x_k\right) - t_j \left\|g_k\right\|^2 + \frac{1}{2} t_j^2 \left\langle g_k, \nabla_{xx} m_k\left(\xi_j\right) g_k\right\rangle < m_k\left(x_k\right) - t_j \left\|g_k\right\|^2 + \frac{1}{2} t_j^2 \beta_k \left\|g_k\right\|^2, \tag{2.12}$$

da

$$\langle g_k, \nabla_{xx} m_k(\xi_j) g_k \rangle \overset{C.S.U.}{\leq} \nabla_{xx} m_k(\xi_j) \|g_k\|^2 \overset{(2.4)}{<} \beta_k \|g_k\|^2 .$$

Vergleicht man nun die beiden Ungleichungen (2.11) und (2.12) miteinander, so erhält man:

$$m_k(x_k) - t_j \|g_k\|^2 + \frac{1}{2} t_j^2 \beta_k \|g_k\|^2 > m_k(x_k) - \kappa_{ubs} t_j \|g_k\|^2$$

und da $t_j \neq 0$ damit auch $t_j > \frac{2(1-\kappa_{ubs})}{\beta_k}$. Da $\kappa_{bck} < 1$, muss ein endliches $j_c > j$ existieren, für das

$$t_{j_c} = \frac{\kappa_{bck}^{j_c} \Delta_k}{\|g_k\|_k} < \frac{2(1-\kappa_{ubs})}{\beta_k} \tag{2.13}$$

gilt und somit die Bedingung (2.9) erfüllt ist. Der approximierte Cauchy-Punkt ist damit wohldefiniert und es gilt:

$$\begin{aligned} m_k(x_k) - m_k\left(x_k^{AC}\right) &\overset{(2.9)}{\geq} -\kappa_{ubs} \langle g_k, x_k(j_c) - x_k \rangle \\ &\overset{(2.8)}{=} -\kappa_{ubs} \left\langle g_k, -\kappa_{bck}^{j_c} \frac{\Delta_k}{\|g_k\|_k} g_k \right\rangle \\ &= \kappa_{ubs} \kappa_{bck}^{j_c} \nu_k^C \Delta_k \|g_k\| . \end{aligned} \tag{2.14}$$

Ist $j_c \geq 1$, was nach (2.8) der Fall ist, wenn x_k^{AC} innerhalb der Trust-Region liegt, so gilt:

$$\kappa_{bck}^{j_c} = \kappa_{bck} \kappa_{bck}^{j_c - 1} \overset{(2.13)}{\geq} 2\kappa_{bck}(1-\kappa_{ubs}) \frac{\|g_k\|_k}{\beta_k \Delta_k}.$$

Diese Ungleichung folgt, da j_c der kleinste Wert ist, für den (2.13) erfüllt ist und für $j_c - 1$ demzufolge die Umkehrung von (2.13) gilt.
Setzt man dieses Abschätzung für $\kappa_{bck}^{j_c}$ nun in (2.14) ein, so erhält man die Ungleichung

$$m_k(x_k) - m_k\left(x_k^{AC}\right) \geq 2\kappa_{bck}\kappa_{ubs}(1-\kappa_{ubs}) \frac{\|g_k\|^2}{\beta_k}.$$

Ist dagegen $j_c = 0$, so liegt x_k^{AC} auf dem Rand der Trust-Region und es folgt aus (2.14) sofort folgende Abschätzung für die Verbesserung des Wertes der Modellfunktion:

$$m_k(x_k) - m_k\left(x_k^{AC}\right) \geq \kappa_{ubs} \nu_k^C \Delta_k \|g_k\| .$$

Zusammen ergibt sich aus den Abschätzungen der beiden Fälle die Behauptung des Satzes mit $\kappa_{dcp} = \min(\kappa_{ubs}, 2\kappa_{bck}(1-\kappa_{ubs}))$. □

Der approximierte Cauchy-Punkt muss selbst bei quadratischen Modellfunktionen nicht mit dem exakten übereinstimmen. Dennoch kann man, wie oben gesehen, für die durch diese beiden Punkte erreichte Verbesserung des Wertes der Modellfunktion ähnliche Abschätzungen herleiten. Da beide Punkte außerdem die selbe Rolle bei der Betrachtung der Konvergenz des Verfahrens spielen, kann man die Unterscheidung zwischen x_k^C und x_k^{AC} zugunsten einer einfacheren Beweisführung, aufgeben. Im

Folgenden wird also nur noch die Bezeichnungen für den exakten Cauchy-Punkt verwendet, obwohl die Aussagen ebenso für den approximierten Cauchy-Punkt gelten. Aus diesem Grund werden nun die beiden Abschätzungen in einer weiteren Annahme zusammengefasst:

AA 1: Für alle Iterationsschritte k gilt:

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{mdc} \|g_k\| \min\left(\frac{\|g_k\|}{\beta_k}, \Delta_k\right)$$

mit einer Konstanten $\kappa_{mdc} \in (0, 1)$.

Die Konstante κ_{mdc} lässt sich abhängig davon, ob $x_k + s_k$ ein exakter oder ein approximierter Cauchy-Punkt ist, genauer bestimmen. Im Fall eines exakten Cauchy-Punktes gilt $\kappa_{mdc} = \frac{1}{2\kappa_{une}}$, andernfalls ist $\kappa_{mdc} = \kappa_{dcp}\frac{1}{\kappa_{une}}$.
Dies folgt aus den beiden oben bewiesenen Abschätzungen und der Tatsache, dass sich $\nu_k^C = \frac{\|g_k\|}{\|g_k\|_k}$ aufgrund der Annahme AN 1 für alle k durch $\nu_k^C \geq \frac{1}{\kappa_{une}} > 0$ abschätzen lässt.
Damit lässt sich der folgende Satz formulieren:

Satz 2.3 *Seien die Annahmen AM 3 und AA 1 erfüllt und sei* $\nabla_x f(x_k) \neq 0$. *Dann gilt* $m_k(x_k + s_k) < m_k(x_k)$ *und* $s_k \neq 0$.

Beweis 3 Aus AM 3 und $\nabla_x f(x_k) \neq 0$ folgt $g_k \neq 0$. Damit ergibt sich:

$$\kappa_{mdc} \|g_k\| \min\left(\frac{\|g_k\|}{\beta_k}, \Delta_k\right) > 0$$

und somit aus AA 1 die Ungleichung $m_k(x_k + s_k) < m_k(x_k)$, also die Behauptung.□

Als Konsequenz aus Satz 2.3 lässt sich festhalten, dass

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)},$$

mit dem im BTR-Algorithmus die Genauigkeit der Modellfunktion gemessen wird, immer wohldefiniert ist, solange die Ableitung der Zielfunktion in x_k nicht 0 ist. In einigen Quellen (beispielsweise in [1]) findet man auch die äquivalente Bezeichnung $\frac{\text{ared}_k}{\text{pred}_k}$ für den Quotienten aus tatsächlicher (engl.: actual reduction) und vorhergesagter (engl.: predicted reduction) Verringerung der Zielfunktion.
Natürlich wird man den Wert der Modellfunktion weiter, als um den in Annahme AA 1 angeführten Wert, reduzieren, wenn dies ohne unakzeptabel großen zusätzlichen Rechenaufwand möglich ist. Aus diesem Grund kann es sinnvoll sein, das Problem

$$\min_{x \in \mathcal{B}_k} m_k(x) \tag{2.15}$$

exakt zu lösen.

Definition 2.6 *Man bezeichnet eine Lösung des Problems (2.15) als eine* globale Minimalstelle der Modellfunktion *(engl.: model minimizer) innerhalb der Trust-Region und schreibt dafür* x_k^M.

Für den Punkt x_k^M kann man folgende Aussage zeigen:

Satz 2.4 *Die Schrittweiten* s_k *seien so gewählt, dass für alle Iterationsschritte* k *die Abschätzung*

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{amm}\left(m_k(x_k) - m_k\left(x_k^M\right)\right)$$

für eine Konstante $\kappa_{amm} \in (0, 1]$ *gilt. Dann folgt daraus:* κ_{mdc} *kann so gewählt werden, dass die Annahme AA 1 zutrifft.*

Beweis 4 Da x_k^M eine globale Minimalstelle der Modellfunktion innerhalb der Trust-Region ist, gilt $m_k\left(x_k^M\right) \leq m_k\left(x_k^C\right)$. Wählt man demnach als Schrittweite $s_k = x_k^M - x_k$, so ist AA 1 mit

$$\kappa_{mdc} = \frac{1}{\kappa_{une}} \min\left(\frac{1}{2}, \kappa_{dcp}\right)$$

erfüllt. Daraus folgt AA 1 für ein beliebiges s_k, das die Voraussetzung des Satzes erfüllt, mit

$$\kappa_{mdc} = \kappa_{amm}\frac{1}{\kappa_{une}} \min\left(\frac{1}{2}, \kappa_{dcp}\right)$$

und damit die Behauptung des Satzes. □

Satz 2.4 ist deshalb wichtig, weil er zeigt, dass die hier entwickelte Konvergenztheorie auf Implementationen des BTR-Algorithmus, die auf der Berechnung der Minimalstelle der Modellfunktion x_k^M beruhen, angewandt werden kann.

2.1.4 Konvergenz erster Ordnung

In diesem Abschnitt wird gezeigt, dass der BTR-Algorithmus global gegen eine Nullstelle der ersten Ableitung der Zielfunktion konvergiert. Das heißt, dass unter den gemachten Voraussetzungen alle Häufungspunkte x_* der Folge x_k der vom Algorithmus erzeugten Iterationswerte unabhängig von der Wahl des Startpunkts x_0 und des anfänglichen Trust-Region-Radius Δ_0, die Bedingung $\nabla_x f(x_*) = 0$ für das Ausgangsproblem (2.1) erfüllen.
Zunächst einmal wird im nächsten Satz eine Abschätzung für die Abweichung des Wertes der Modellfunktion von dem der Zielfunktion im neuen Iterationspunkt $x_k + s_k \in \mathcal{B}_k$ hergeleitet:

Satz 2.5 *Die Annahmen AF 1, AF 3 und AM 1 - AM 4 seien erfüllt. Dann gilt für alle Iterationsschritte k :*

$$|f(x_k+s_k)-m_k(x_k+s_k)| \leq \max(\kappa_{ufh},\kappa_{umh})(\nu_k^s)^2\Delta_k^2 \tag{2.16}$$

mit $x_k+s_k \in \mathcal{B}_k$ und

$$\nu_k^s = \frac{\|s_k\|}{\|s_k\|_k}.$$

Die beiden Konstanten κ_{ufh} und κ_{umh} bezeichnen dabei die in den Annahmen AF 3 und AM 4 definierten oberen Schanken der Normen der Hessematrizen der Zielfunktion bzw. der Modellfunktion.
Ist darüber hinaus auch die Annahme AN 1 erfüllt, gilt sogar

$$|f(x_k+s_k)-m_k(x_k+s_k)| \leq \kappa_{ubh}\Delta_k^2 \tag{2.17}$$

mit $\kappa_{ubh} := \kappa_{une}^2 \max(\kappa_{ufh},\kappa_{umh})$.

Beweis 5 Aufgrund der Annahmen AF 1 und AM 1, in denen die zweimalige Differenzierbarkeit der Zielfunktion bzw. der Modellfunktion gefordert wird, kann man die Werte der beiden Funktionen im neuen Iterationspunkt x_k+s_k mit Hilfe einer Taylorentwicklung bestimmen. So ist

$$f(x_k+s_k) = f(x_k)+\langle s_k, \nabla_x f(x_k)\rangle + \frac{1}{2}\langle s_k, \nabla_{xx} f(\xi_k)\, s_k\rangle$$

für ein $\xi_k \in [x_k, x_k+s_k]$ und

$$m_k(x_k+s_k) = m_k(x_k)+\langle s_k, g_k\rangle + \frac{1}{2}\langle s_k, \nabla_{xx} m_k(\zeta_k)\, s_k\rangle$$

für ein $\zeta_k \in [x_k, x_k+s_k]$. Subtrahiert man die Entwicklung der Modellfunktion von der Entwicklung der Zielfunktion und bildet davon den Absolutbetrag, so ergibt sich:

$$\begin{aligned}
|f(x_k+s_k)-m_k(x_k+s_k)| &\overset{\text{AM2, AM3}}{=} \frac{1}{2}|\langle s_k, \nabla_{xx} f(\xi_k)\, s_k\rangle - \langle s_k, \nabla_{xx} m_k(\zeta_k)\, s_k\rangle| \\
&\overset{\triangle-\text{Ungl.}}{\leq} \frac{1}{2}|\langle s_k, \nabla_{xx} f(\xi_k)\, s_k\rangle| + \frac{1}{2}|\langle s_k, \nabla_{xx} m_k(\zeta_k)\, s_k\rangle| \\
&\overset{\text{C.S.U., AF3, AM4}}{\leq} \frac{1}{2}\|s_k\|^2\kappa_{ufh} + \frac{1}{2}\|s_k\|^2(\kappa_{umh}-1) \\
&= \frac{1}{2}(\kappa_{ufh}+\kappa_{umh}-1)\|s_k\|^2 \\
&= \frac{1}{2}(\kappa_{ufh}+\kappa_{umh}-1)(\nu_k^s)^2\|s_k\|_k^2 \\
&\overset{\|s_k\|_k\leq\Delta_k}{\leq} \frac{1}{2}(\kappa_{ufh}+\kappa_{umh}-1)(\nu_k^s)^2\Delta_k^2 \\
&\leq \max(\kappa_{ufh},\kappa_{umh})(\nu_k^s)^2\Delta_k^2.
\end{aligned}$$

Damit ist (2.16) und somit der erste Teil des Satzes gezeigt. Um daraus (2.17) zu folgern, wendet man AN 1 auf $\nu_k^s = \frac{\|s_k\|}{\|s_k\|_k}$ an, wodurch sich sofort der zweite Teil des Satzes ergibt. □

An der Abschätzung kann man erkennen, dass der Fehler sich proportional zum Quadrat des Trust-Region-Radius verhält. Das bedeutet, je kleiner der Trust-Region Radius, desto geringer ist die Abweichung zwischen Ziel- und Modellfunktion. Mit dem nächsten Satz wird diese Anschauung untermauert. Es wird gezeigt, dass ein Iterationsschritt erfolgreich sein muss, wenn der aktuelle Iterationswert keine Nullstelle der ersten Ableitung ist und der Trust-Region-Radius klein genug ist.

Satz 2.6 *Die Annahmen AF 1, AF 3, AN 1, AM 1 - AM 4 und AA 1 seien erfüllt. Darüberhinaus sei $g_k \neq 0$ und*

$$\Delta_k \leq \frac{\kappa_{mdc} \|g_k\| (1-\eta_2)}{\kappa_{ubh}}.$$

Dann ist der Iterationsschritt k sehr erfolgreich und es gilt: $\Delta_{k+1} \geq \Delta_k$.

Beweis 6 Für die verwendeten Konstanten gilt nach den jeweiligen Definitionen in der Beschreibung des BTR-Algorithmus bzw. in AA 1: $\eta_2 \in (0,1)$ und $\kappa_{mdc} \in (0,1)$. Daraus folgt, dass $\kappa_{mdc}(1-\eta_2) < 1$ ist und man kann den Trust-Region-Radius wie folgt abschätzen:

$$\begin{aligned}
\Delta_k &\leq \frac{\kappa_{mdc} \|g_k\| (1-\eta_2)}{\kappa_{ubh}} \overset{\text{Def. } \kappa_{ubh}}{=} \frac{\kappa_{mdc} \|g_k\| (1-\eta_2)}{\kappa_{une}^2 \max(\kappa_{ufh}, \kappa_{umh})} \\
&< \frac{\|g_k\|}{\kappa_{une}^2 \max(\kappa_{ufh}, \kappa_{umh})} \overset{\kappa_{une} \geq 1}{\leq} \frac{\|g_k\|}{\max(\kappa_{ufh}, \kappa_{umh})} \overset{\beta_k < \kappa_{umh}}{<} \frac{\|g_k\|}{\beta_k}.
\end{aligned}$$

Dabei ergibt sich $\kappa_{une} \geq 1$ aus AN 1 und $\beta_k < \kappa_{umh}$ aus (2.4) und AM 4. Eingesetzt in AA 1 ergibt dies:

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{mdc} \|g_k\| \min\left(\frac{\|g_k\|}{\beta_k}, \Delta_k\right) = \kappa_{mdc} \|g_k\| \Delta_k. \tag{2.18}$$

Mit dieser Ungleichung und der Voraussetzung an Δ_k aus dem Satz kann man zeigen:

$$\begin{aligned}
|\rho_k - 1| &= \left| \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} - \frac{m_k(x_k) - m_k(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \right| \\
&\overset{\text{AM } 2}{=} \left| \frac{f(x_k + s_k) - m_k(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \right| \\
&\overset{(2.17),\,(2.18)}{\leq} \frac{\kappa_{ubh}}{\kappa_{mdc} \|g_k\|} \cdot \Delta_k \leq \frac{\kappa_{ubh}}{\kappa_{mdc} \|g_k\|} \cdot \frac{\kappa_{mdc} \|g_k\| (1-\eta_2)}{\kappa_{ubh}} \\
&= 1 - \eta_2.
\end{aligned}$$

In der vorletzten Zeile ist es möglich, auf den Betrag zu verzichten, da alle Faktoren des Produkts größer oder gleich null sind. Nun betrachtet man die beiden Fälle $\rho_k \geq 1$

und $\rho_k < 1$. Im ersten Fall ist offensichtlich $\rho_k \geq \eta_2$, da $\eta_2 \in (0,1)$ gewählt war. Im zweiten Fall kann man die Ungleichung folgendermaßen umformen:

$$\begin{aligned} & |\rho_k - 1| \leq 1 - \eta_2 \\ \Rightarrow \quad & 1 - \rho_k \leq 1 - \eta_2 \\ \Rightarrow \quad & \rho_k \geq \eta_2. \end{aligned}$$

Damit hat man für alle möglichen Fälle gezeigt: $\rho_k \geq \eta_2$; das bedeutet nach Definition 2.2, dass der Iterationsschritt k sehr erfolgreich ist. Außerdem folgt die zweite Behauptung des Satzes $\Delta_{k+1} \geq \Delta_k$, da, wie im Algorithmus 2.1 beschrieben, bei sehr erfolgreichen Iterationsschritten der Trust-Region-Radius folgendermaßen aktualisiert wird: $\Delta_{k+1} \in [\Delta_k, \infty)$. □

Als nächstes wird gezeigt, dass der Trust-Region-Radius nicht zu klein werden kann, solange man nicht ausreichend nahe bei einer Nullstelle der ersten Ableitung der Zielfunktion angelangt ist. Diese Eigenschaft ist sehr wichtig, da durch sie gewährleistet wird, dass der Algorithmus nicht gegen einen Punkt konvergiert, der keine Minimalstelle ist.

Satz 2.7 *Die Annahmen AF 1 - AF 3, AN 1, AM 1 - AM 4 und AA 1 seien erfüllt. Des weiteren existiere eine Konstante $\kappa_{lbg} > 0$ so, dass $\|g_k\| \geq \kappa_{lbg}$ für alle Iterationsschritte k gilt. Dann existiert eine Konstante $\kappa_{lbd} > 0$ mit $\Delta_k \geq \kappa_{lbd}$ für alle k.*

Beweis 7 Angenommen der $k+1$-te Iterationsschritt sei der erste für den

$$\Delta_{k+1} \leq \frac{\gamma_1 \kappa_{mdc} \kappa_{lbg} (1 - \eta_2)}{\kappa_{ubh}} \tag{2.19}$$

gilt. Außerdem weiß man aufgrund des fünften Schrittes im BTR-Algorithmus, in dem der Trust-Region-Radius aktualisiert wird, dass $\gamma_1 \Delta_k \leq \Delta_{k+1}$ und somit

$$\Delta_k \leq \frac{\kappa_{mdc} \kappa_{lbg} (1 - \eta_2)}{\kappa_{ubh}}$$

gilt.
Da $\|g_k\| \geq \kappa_{lbg}$ vorausgesetzt wurde, folgt:

$$\Delta_k \leq \frac{\kappa_{mdc} \|g_k\| (1 - \eta_2)}{\kappa_{ubh}}.$$

Damit sind die Voraussetzungen für Satz (2.6) erfüllt und es folgt $\Delta_k \leq \Delta_{k+1}$.
Dies widerspricht aber der Annahme, dass der $k+1$-te Iterationsschritt der erste ist, für den (2.19) gilt. Sie muss deswegen verworfen werden und folglich wurde die Behauptung des Satzes mit

$$\kappa_{lbd} = \frac{\gamma_1 \kappa_{mdc} \kappa_{lbg} (1 - \eta_2)}{\kappa_{ubh}}$$

gezeigt. □

Mit Hilfe der beiden vorangegangenen Ergebnisse kann man nun zeigen, dass der eindeutige Häufungspunkt der vom Algorithmus erzeugten Folge von Iterationswerten eine Nullstelle der ersten Ableitung der Zielfunktion ist, falls nur endlich viele Iterationsschritte erfolgreich waren.

Satz 2.8 *Die Annahmen AF 1, AF 3, AN 1, AM 1 - AM 4 und AA 1 seien erfüllt. Außerdem sei die Anzahl der erfolgreichen Iterationsschritte endlich. Dann gilt für alle ausreichend großen k:*

$$x_k = x_*,$$

wobei x_* eine Nullstelle der ersten Ableitung der Zielfunktion ist.

Beweis 8 Sei k_0 der Index des letzten erfolgreichen Iterationsschrittes. Dann gilt für alle $j > 0 : x_* = x_{k_0+1} = x_{k_0+j}$, da, wie in Schritt 4 des BTR-Algorithmus beschrieben, nach jedem nicht erfolgreichen Iterationsschritt der alte Iterationswert aus dem letzten erfolgreichen Iterationsschritt beibehalten wird.
Da somit alle Iterationsschritte ab einem hinreichend großen k nicht erfolgreich sind, folgt aus dem Aktualisierungsschritt des BTR-Algorithmus für den Trust-Region-Radius, dass die Folge $\{\Delta_k\}$ gegen null konvergiert.
Wäre $\|g_{k_0+1}\| > 0$, so würde aus Satz 2.6 folgen, dass es einen erfolgreichen Iterationsschritt mit Index größer als k_0 gibt. Da dies aber den Vorraussetzungen des Satzes widerspricht, muss $\|g_{k_0+1}\| = 0$ sein und damit ist x_* eine Nullstelle der ersten Ableitung der Zielfunktion. □

Im letzten Satz wurde die Konvergenz des BTR-Algorithmus für den Fall gezeigt, in dem $\mathcal{S}$, also die Menge aller erfolgreichen Iterationsschritten, endlich ist.
Es bleibt der Fall zu betrachten, in dem es unendlich viele erfolgreiche Iterationsschritte gibt. Zunächst soll zu diesem Zweck gezeigt werden, dass mindestens einer der Häufungspunkte der Folge von Iterationswerten eine Nullstelle der ersten Ableitung der Zielfunktion ist.

Satz 2.9 *Die Annahmen AF 1 - AF 3, AN 1, AM 1 - AM 4 und AA 1 seien erfüllt. Dann gilt:*

$$\liminf_{k\to\infty} \|\nabla_x f(x_k)\| = 0$$

Beweis 9 Annahme: Für alle Iterationsschritte k gilt $\|\nabla_x f(x_k)\| \geq \epsilon$ mit $\epsilon > 0$. Da wegen AM 3 die Gleichung $\|\nabla_x f(x_k)\| = \|g_k\|$ erfüllt ist, folgt auch für den Gradienten der Modellfunktion $\|g_k\| \geq \epsilon$.
Betrachtet man nun einen erfolgreichen Iterationsschritt mit Index k, ergibt sich dafür aus AA 1 die folgende Ungleichung:

$$f(x_k) - f(x_{k+1}) \overset{k\in\mathcal{S}}{\geq} \eta_1 (m_k(x_k) - m_k(x_k + s_k)) \geq \kappa_{mdc}\epsilon\eta_1 \min\left(\frac{\epsilon}{\kappa_{umh}}, \kappa_{lbd}\right),$$

wobei nach (2.4) und AM 4 $\kappa_{umh} \geq \beta_k$ ist und nach Satz 2.7 $\kappa_{lbd} \leq \Delta_k$ gilt. Summiert man dies über alle erfolgreichen Iterationsschritte mit Indizes von 0 bis k, so erhält man

$$f(x_0) - f(x_{k+1}) = \sum_{\substack{j=0 \\ j \in \mathcal{S}}}^{k} (f(x_j) - f(x_{j-1})) \geq \sigma_k \kappa_{mdc} \epsilon \eta_1 \min\left(\frac{\epsilon}{\kappa_{umh}}, \kappa_{lbd}\right).$$

Dabei ist σ_k die Anzahl der erfolgreichen Iterationsschritte, die bis zur k-ten Iteration aufgetreten sind. Da es insgesamt aber unendlich viele solcher Iterationsschritte gibt, folgt

$$\lim_{k \to \infty} \sigma_k = \infty$$

und somit, dass die Differenz $f(x_0) - f(x_{k+1})$ unbeschränkt ist. Das ist jedoch ein Widerspruch zu der Tatsache, dass die Zielfunktion, wie in Bedingung AF 2 gefordert, nach unten beschränkt ist. Deshalb ist die zu Beginn des Beweises gemachte Annahme falsch und es folgt die Behauptung des Satzes. □

Der letzte Satz ist vergleichbar mit einer Existenzaussage, da er besagt, dass, falls die Folge von Iterationswerten Häufungspunkte hat, mindestens einer davon eine Nullstelle der ersten Ableitung der Zielfunkion ist. Nun soll noch, als endgültiges Konvergenzresultat, gezeigt werden, dass dies für alle Häufungspunkte dieser Folge gilt.

Satz 2.10 *Die Annahmen AF 1 - AF 3, AN 1, AM 1 - AM 4 und AA 1 seien erfüllt. Dann gilt:*

$$\lim_{k \to \infty} \|\nabla_x f(x_k)\| = 0.$$

Beweis 10 Annahme: Es existiert eine Teilfolge erfolgreicher Iterationsschritte mit Indizes $\{t_i\} \subseteq \mathcal{S}$ so, dass

$$\|\nabla_x f(x_{t_i})\| \overset{\text{AM 3}}{=} \|g_{t_i}\| \geq 2\epsilon$$

mit $\epsilon > 0$ und für alle i.
Durch Satz 2.9 wird gewährleistet, dass es zu jedem Index t_i einen Iterationsschritt $l(t_i) > t_i$ gibt, der erfolgreich ist und für den gilt: $\left\|g_{l(t_i)}\right\| < \epsilon$. Der Iterationsschritt $l(t_i)$ sei der erste auf t_i folgende, für den diese Bedingung erfüllt ist.
Mit der Abkürzung $l_i := l(t_i)$ erhält man eine weitere Teilfolge in $\mathcal{S}$ mit den Indizes $\{l_i\}$ so, dass $\|g_k\| \geq \epsilon$ für $t_i \leq k < l_i$ und $\|g_{l_i}\| < \epsilon$ gilt.
Nun soll die Teilfolge erfolgreicher Iterationsschritte, deren Indizes in der Menge

$$\mathcal{K} := \{k \in \mathcal{S} | t_i \leq k < l_i\}$$

liegen, wobei t_i und l_i zu den oben definierten Teilfolgen gehören, betrachtet werden. Aufgrund von AA 1, der Tatsache, dass $\mathcal{K} \subseteq \mathcal{S}$, und der obigen Annahme, kann man für alle $k \in \mathcal{K}$ folgern:

$$f(x_k) - f(x_{k+1}) \geq \eta_1 (m_k(x_k) - m_k(x_k + s_k)) \geq \kappa_{mdc} \epsilon \eta_1 \min\left(\frac{\epsilon}{\kappa_{umh}}, \Delta_k\right). \quad (2.20)$$

Jedoch ist die Folge der Funktionswerte $\{f(x_k)\}$ monoton fallend und wegen AF 2 nach unten beschränkt. Demnach muss sie konvergent sein und die linke Seite von (2.20) muss gegen null konvergieren, wenn k gegen unendlich geht.
Damit erhält man

$$\lim_{\substack{k\to\infty \\ k\in\mathcal{K}}} \Delta_k = 0,$$

da $\frac{\epsilon}{\kappa_{umh}}$ konstant ist. Das bedeutet für ein ausreichend großes $k \in \mathcal{K}$, dass der zweite Term im Minimum auf der rechten Seite von (2.20) der kleinere ist, und die Ungleichung lässt sich so umformen:

$$\Delta_k \leq \frac{1}{\kappa_{mdc}\epsilon\eta_1}\left(f(x_k) - f(x_{k+1})\right).$$

Aufgrund dieser Abschätzung kann man für ein ausreichend großes i folgern:

$$\|x_{t_i} - x_{l_i}\| \leq \sum_{\substack{j=t_i \\ j\in\mathcal{K}}}^{l_i-1} \|x_j - x_{j+1}\| \overset{(1)}{\leq} \sum_{\substack{j=t_i \\ j\in\mathcal{K}}}^{l_i-1} \nu_j^s \Delta_j \overset{(2)}{\leq} \frac{\kappa_{une}}{\kappa_{mdc}\epsilon\eta_1}\left(f(x_{t_i}) - f(x_{l_i})\right).$$

Ungleichung (1) ergibt sich, da

$$\|x_j - x_{j+1}\| = \|s_k\| = \frac{\|s_k\|}{\|s_k\|_k} \cdot \|s_k\|_k \leq \frac{\|s_k\|}{\|s_k\|_k} \cdot \Delta_k$$

gilt. Um Ungleichung (2) zu erhalten, wendet man AN 1 auf ν_j^s an, benutzt die oben stehende Abschätzung für Δ_k und vereinfacht den Ausdruck mit Hilfe der Teleskopsumme.
Wegen AF 2, wonach die Zielfunktion nach unten beschränkt ist, und der Monotonie der Folge $\{f(x_k)\}$ muss die rechte Seite der Ungleichung gegen null konvergieren und man erhält somit, dass $\|x_{t_i} - x_{l_i}\|$ für $i \to \infty$ gegen null geht.
Da der Gradient der Zielfunktion aufgrund von AF 1 stetig ist, kann man weiter folgern, dass $\|\nabla_x f(x_{t_i}) - \nabla_x f(x_{l_i})\|$ und, wegen AM 3, dass $\|g_{t_i} - g_{l_i}\|$ ebenfalls gegen null konvergieren. Dies widerspricht aber den Definitionen der Folgen $\{t_i\}$ und $\{l_i\}$, anhand derer man sieht, dass $\|g_{t_i} - g_{l_i}\| \geq \epsilon$ gilt.
Insgesamt hat man damit gezeigt, dass keine Teilfolge $\{t_i\}$ existieren kann, welche die zu Beginn des Beweises gemachte Annahme erfüllt. Diese muss demnach falsch sein, womit die Behauptung des Satzes gezeigt ist. □

Unter einigen weiteren Voraussetzungen lässt sich für den BTR-Algorithmus auch die Konvergenz zweiter Ordnung zeigen. Dies wird in [4] ausführlich behandelt.
Insgesamt lässt sich das Ergebnis der vorangegangenen Konvergenzuntersuchung folgendermaßen zusammenfassen:
Der BTR-Algorithmus konvergiert unter erfüllbaren Voraussetzungen gegen eine Nullstelle der ersten Ableitung der Zielfunktion. Dieses Resultat wird in Kapitel 3 verwendet, um herzuleiten, dass der Filter-Trust-Region-Algorithmus das selbe Konvergenzverhalten zeigt.

2.2 Ein Filter-Verfahren für nichtlineare Optimierungsaufgaben

In diesem Abschnitt soll das Prinzip eines zweidimensionalen Filters beschrieben werden. Dabei dient der Artikel "Nonlinear Programming without a penalty function" [9], in dem Sven Leyffer und Roger Fletcher 1997 dieses neue Konzept vorgestellt haben, als Grundlage. Die Motivation hinter der Einführung des Filters war der Wunsch, ein nichtlineares Minimierungsproblem mit Nebenbedingungen ohne Verwendung einer Straffunktion zu lösen. In einem Filter-Verfahren benötigt man eine zugrundeliegende iterative Lösungsmethode für das Minimierungsproblem. Dabei können verschiedene Methoden zum Einsatz kommen: In diesem Kapitel werden die Iterationspunkte von einem SQP-Verfahren erzeugt, später im Filter-Trust-Region-Algorithmus übernimmt diese Rolle das Trust-Region-Verfahren. Unabhängig davon, wie die Iterationspunkte erzeugt werden, ist es die Aufgabe des Filters, den Fortschritt der Iterationswerte zu überwachen und zu steuern.

2.2.1 Problemformulierung

Der in diesem Kapitel vorgestellte Algorithmus dient dazu, eine lokale Lösung eines restringierten nichtlinearen Minimierungsproblems zu finden. Dazu betrachtet man folgende vereinfachte Problemstellung ohne Gleichheitsnebenbedingungen:

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } c(x) \leq 0. \end{aligned} \tag{2.21}$$

Dabei seien $f : \mathbb{R}^n \to \mathbb{R}$ und $c : \mathbb{R}^n \to \mathbb{R}^m$ zweimal stetig differenzierbar. Es soll nun für Probleme dieser Form ein Algorithmus entwickelt werden, der nicht auf die Verwendung einer Straffunktion angewiesen ist und auch bei inkonsistenten Problemstellungen, die in der Praxis beispielsweise aufgrund von Modellierungsungenauigkeiten auftreten können, gegen eine nach gewissen Gesichtspunkten optimale Lösung konvergiert.
Die Konvergenzgeschwindigkeit des Algorithmus hängt stark von dem zugrundeliegenden Iterationsverfahren ab. Für die Problemstellung (2.21) bietet sich beispielsweise die Methode der sequentiellen quadratischen Optimierung (SQP) an [2]. Dabei wird (2.21) durch Erzeugen und Lösen einer Folge von quadratischen Optimierungsproblemen gelöst. Diese entstehen, indem man die nichtlinearen Nebenbedingungen durch lineare und die nichtlineare Zielfunktion durch quadratische Approximationen der Lagrange-Funktion des Problems ersetzt. Die einzelnen quadratischen Teilprobleme werden mit Hilfe eines Trust-Region-Verfahrens gelöst. Die Lösung eines solchen Teilproblems wird anschließend dem Filter übergeben, um zu entscheiden, ob der Iterationsschritt erfolgreich war.
Hier soll es genügen, folgende Aussagen über das allgemeine Konvergenzverhalten von SQP-Verfahren festzuhalten: Unter bestimmten Voraussetzungen kann gezeigt

werden, dass sie in der Nähe einer Lösung quadratisch konvergieren. Allerdings werden Erweiterungen benötigt, um globale Konvergenz von einem beliebigen Startpunkt zu einer Lösung zu gewährleisten. Beispielsweise wird dies durch Verwendung einer Straffunktion ermöglicht. Diese besteht aus einer Linearkombination der Zielfunktion mit einem Maß für die Größe der Verletzung der Nebenbedingungen.
Dadurch erhält man ein Entscheidungskriterium für die Frage, ob ein Iterationsschritt des SQP-Verfahrens akzeptiert werden sollte. Dies geschieht nämlich genau dann, wenn dadurch eine ausreichende Verringerung des Wertes der Straffunktion erreicht wird [3]. Ein Beispiel einer Straffunktion aus [3] ist

$$\psi(x) = f(x) + \rho \left\| c(x) \right\|_1 ,$$

wobei $f(x)$ die Zielfunktion, $c(x)$ der Funktionsvektor der Nebenbedingungen des Problems und $\|\cdot\|_1$ die ℓ_1-Norm ist, außerdem ist $0 < \rho \in \mathbb{R}$ der Strafparameter.
Es gibt einige Schwierigkeiten, die bezüglich der Straffunktion und insbesondere bei der Wahl des Parameters der Straffunktion auftreten. Wählt man den Strafparameter kleiner als einen nicht a-priori bekannten Schwellwert, hat die Straffunktion kein lokales Minimum in der Lösung von (2.21). Deshalb kann eine zu kleine Wahl des Strafparameters zu einer unzulässigen Lösung oder einem unbeschränkten Anstieg des Werts der Straffunktion führen. Wird der Parameter dagegen zu groß gewählt, führt dies zu langsamer Konvergenz.
Die angeführten Nachteile, die mit der Verwendung einer Straffunktion einhergehen, werden bei einem Verfahren, das auf das Konzept des Filters zurückgreift, vermieden. Im hier vorgestellten Verfahren wird versucht mit möglichst wenig Eingriffen in die Folge der SQP-Iterationsschritte durch die Verwendung des Filteransatzes globale Konvergenz zu erreichen.
Zur Erleichterung der Schreibweise werden folgende Notationen eingeführt: Der Gradient von f wird definiert als $g = g(x) = \nabla f(x)$, die Jacobimatrix der Nebenbedingungen als $A = A(x) = \nabla c^T(x)$ und die Lagrange-Funktion als $\mathcal{L}(x, \lambda) = f(x) + \lambda^T c(x)$. Die Hessematrix der Lagrange-Funktion sei $W = \nabla^2 \mathcal{L}(x, \lambda)$. Geklammerte hochgestellte Indizes $^{(k)}$ bezeichnen den k-ten Iterationsschritt und $f^{(k)} = f\left(x^{(k)}\right)$. Größen, die sich auf eine lokale Lösung von (2.21) beziehen, sind mit einem * gekennzeichnet.

2.2.2 Der Filteransatz

Es gibt bei einem nichtlinearen Optimierungsproblem zwei üblicherweise konkurrierende Ziele, nämlich die Minimierung der Zielfunktion f und das Bestreben, die Nebenbedingungen zu erfüllen. Formal lassen sich diese beiden Ziele folgendermaßen schreiben:

$$\min f(x) \tag{2.22}$$

und

$$\min h\left(c(x)\right), \tag{2.23}$$

wobei

$$h\left(c(x)\right) := \left\|c^+(x)\right\|_1 = \sum_{j=1}^{m} c_j^+(x)$$

und $c_j^+ := max\left(0, c_j\right)$ ist.
Man sieht, auch das Ziel eine zulässige Lösung, d.h. eine Lösung, welche die Nebenbedingungen erfüllt, zu erhalten, kann als Minimierungsproblem (2.23) angesehen werden. Die ℓ_1-Norm wurde als Maß für die Abweichung in den Nebenbedingungen gewählt, da dies in der später zu erläuternden Restaurationsphase (engl.: restoration phase) den Vorteil einer vergleichsweise einfach zu minimierenden Zielfunktion mit sich bringt. Diese Wahl ist allerdings nicht die einzig Mögliche, sondern es kann auch eine beliebige andere Norm verwendet werden.
Durch eine Straffunktion würden (2.22) und (2.23) zu einem einzigen Minimierungsproblem kombiniert werden. Es erscheint jedoch intuitiv sinnvoller, sie getrennt zu betrachten, da die beiden Zielsetzungen durchaus gegensätzlich sein können. Außerdem hat die Minimierung von $h\left(c(x)\right)$ insofern eine höhere Priorität, als dass $h\left(c(x)\right) = 0$ für eine zulässige Lösung des gesamten Problems eine notwendige Bedingung ist. Um den Filter definieren zu können, benötigt man zuerst eine abgewandelte Definition aus der multikriteriellen Optimierung:

Definition 2.7 *Seien* $\left(f^{(k)}, h^{(k)}\right)$ *bzw.* $\left(f^{(l)}, h^{(l)}\right)$ *die in* $x^{(k)}$ *bzw.* $x^{(l)}$ *berechneten Werte von* $f(x)$ *und* $h\left(c(x)\right)$. *Ein Paar* $\left(f^{(k)}, h^{(k)}\right)$ dominiert *ein anderes Paar* $\left(f^{(l)}, h^{(l)}\right)$ *genau dann, wenn sowohl* $f^{(k)} \leq f^{(l)}$ *als auch* $h^{(k)} \leq h^{(l)}$ *gilt.*

Der hier definierte Begriff der Dominanz unterscheidet sich von dem in der multikriteriellen Optimierung gebräuchlichen. Dieser wird in Kapitel 2.3 dargestellt.
Damit ist es möglich, den Filter so zu definieren, wie er als Akzeptanzkriterium z.B. in SQP- oder Trust-Region-Algorithmen verwendet werden kann.

Definition 2.8 *Ein* Filter *ist eine Liste bestehend aus Paaren* $\left(f^{(l)}, h^{(l)}\right)$ *derart, dass kein Paar ein anderes dominiert. Ein Paar* $\left(f^{(k)}, h^{(k)}\right)$ *ist* akzeptabel *für den Filter, wenn es von keinem anderen Paar, das sich bereits im Filter befindet, dominiert wird.*

Man sieht, dass im Filter lediglich zwei Skalare pro Eintrag gespeichert werden müssen und nicht etwa der ganze Vektor $x^{(l)}$. Daraus folgt zunächst, dass vergleichsweise wenig Speicherplatz benötigt wird. Allerdings kann es durchaus sinnvoll sein, zu jedem Filtereintrag ebenfalls den dazugehörenden Iterationspunkt zu speichern, beispielsweise um backtracking-Strategien anwenden zu können. Mit backtracking bezeichnet man ein Verfahren, das zu einem früheren Iterationspunkt zurückkehren kann, um von dort aus einen alternativen Iterationsschritt durchzuführen, wenn dies für das Erreichen einer Lösung des Problems sinnvoll erscheint.

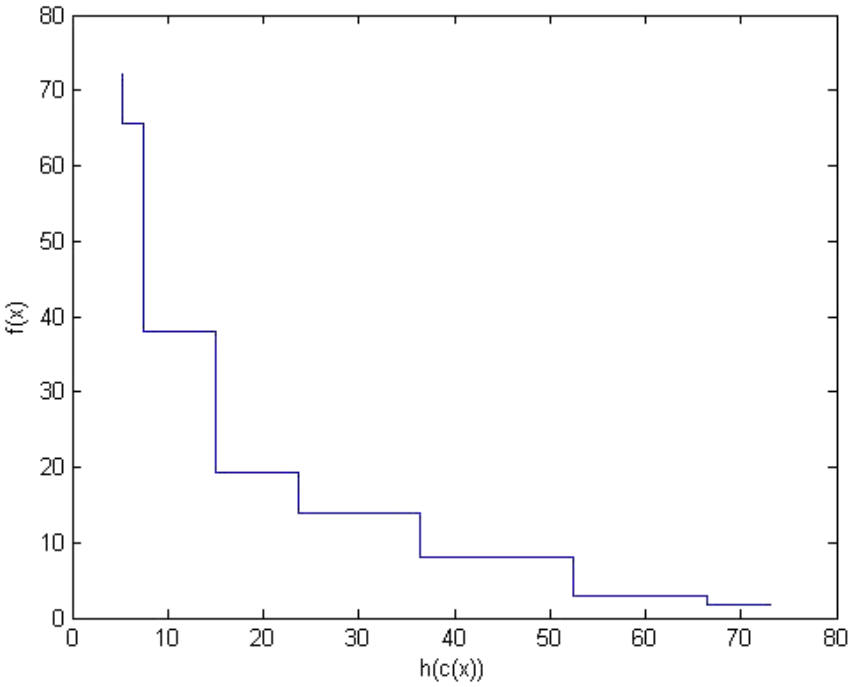


Abbildung 2.3: Beispiel für einen Filter mit sieben Einträgen; die von den Filtereinträgen dominierten Punkte liegen rechts bzw. oberhalb des Graphen, akzeptable Punkte links bzw. unterhalb davon.

Ein solcher Filter soll nun auf das SQP-Trust-Region-Verfahren angewandt werden. Dazu ist es allerdings zuvor erforderlich, sequentiell die quadratischen Teilprobleme zu lösen. Diese lassen sich in der Form

$$(QP^{(k)}) \begin{cases} \min_d & \frac{1}{2} d^T W^{(k)} d + d^T g^{(k)} \\ s.t. & {A^{(k)}}^T d + c^{(k)} \leq 0 \\ & \|d\|_\infty \leq \Delta^{(k)} \end{cases}$$

darstellen, wobei die Approximationen der Zielfunktion und der Nebenbedingungen im letzten Iterationswert $x^{(k)}$ eines erfolgreichen Iterationsschrittes gebildet wurden. Dabei ist $W^{(k)}$ die Hessematrix der Lagrange-Funktion, $g^{(k)}$ der Gradient der Zielfunktion, $A^{(k)}$ die Jacobi-Matrix des Vektors der Nebenbedingungen, $c^{(k)}$ der Funktionswert der Nebenbedingungen in $x^{(k)}$ und $\Delta^{(k)}$ der verwendete Trust-Region-Radius. Durch eine Lösung $d^{(k)}$ dieses Teilproblems erhält man den nächsten Testschritt für das eigentliche Problem (2.21) und somit den Testpunkt $x^{(k+1)} = x^{(k)} + d^{(k)}$. Dieser wird vom Filter akzeptiert, wenn das zugehörige Paar $\left(f^{(k+1)}, h^{(k+1)}\right)$ von keinem anderen der bereits im Filter vorhandenen Einträge dominiert wird. Andernfalls wird der Schritt abgelehnt und der Trust-Region-Radius Δ verkleinert, um so einen geeigneteren Testpunkt zu erhalten. Das üblicher Weise verwendete Akzeptanzkriterium

für einen Iterationsschritt, das eine Verbesserung im Wert der Straffunktion durch den neuen Iterationspunkt fordert, wird somit ersetzt durch die Bedingung, dass ein neuer Punkt für den Filter akzeptabel ist.
In einem konventionellen Trust-Region-Verfahren folgt die Konvergenz, da sich mit Verkleinerung des Trust-Region-Radius der Testschritt immer weiter der Richtung des negativen Gradienten, also der Richtung des steilsten Abstiegs annähert. Dadurch wird eine Verbesserung, die genügt um den Iterationsschritt zu akzeptieren, erreicht. In einem SQP-Trust-Region-Verfahren dagegen kann die Verkleinerung des Trust-Region-Radius zu einem unzulässigen Teilproblem $(QP^{(k)})$ führen, falls der aktuelle Wert $x^{(k)}$ unzulässig für (2.21) ist. Diese Schwierigkeit kann auftreten, wenn mehrere aufeinander folgende Iterationsschritte zurückgewiesen werden.
Ein weiterer Fall, in dem ein unzulässiges $(QP^{(k)})$ entstehen kann, ist, wenn die linearisierten Nebenbedingungen selbst inkonsistent sind, das Teilproblem also keinen zulässigen Bereich besitzt. In diesen beiden Fällen gibt es die Möglichkeit mit Hilfe der Restaurationsphase näher an den zulässigen Bereich zu gelangen. Dies wird im Abschnitt über die Erweiterungen des Algorithmus genauer erläutert. Ist das Problem $(QP^{(k)})$ dagegen zulässig, so kann man annehmen, dass der Algorithmus einen Punkt findet, der als Kandidat für den nächsten Iterationsschritt verwendet werden kann.
Unter Berücksichtigung der bisherigen Erkenntnisse kann man einen grundlegenden Filter-SQP-Algorithmus formulieren:

Algorithmus 2.2 *Basic Filter SQP*

Startwert $x^{(0)}$ und anfänglicher Trust-Region-Radius $\Delta^{(0)}$ sind gegeben, setze $k = 0$.
Repeat
  If *Teilproblem QP ist unzulässig,* **then**
    Bestimme einen Punkt $x^{(k+1)}$ und einen Trust-Region-Radius $\Delta^{(k+1)}$ in der Restaurationsphase.
  else
    - *Bestimme die Lösung $d^{(k)}$ von $(QP^{(k)})$.*
    - *Berechne den Testpunkt $x^{(k+1)} = x^{(k)} + d^{(k)}$.*
    If $\left(f^{(k+1)}, h^{(k+1)}\right)$ *ist für den Filter akzeptabel,* **then**
      - *Akzeptiere $x^{(k+1)}$ und füge $\left(f^{(k+1)}, h^{(k+1)}\right)$ zum Filter hinzu.*
      - *Entferne alle Punkte, die von $\left(f^{(k+1)}, h^{(k+1)}\right)$ dominiert werden aus dem Filter.*
      - *Vergrößere eventuell den Trust-Region-Radius Δ.*
    else
      - *Lehne den Iterationspunkt ab.*
      - *Setze $x^{(k+1)} = x^{(k)}$.*
      - *Verkleinere den Trust-Region-Radius Δ.*
    Endif
  Endif
  Setze $k = k + 1$.
until *Konvergenz*

Auf den in Algorithmus 2.2 nur angedeuteten Aktualisierungvorgang des Trust-Region-Radius wird im Zusammenhang mit Algorithmus 2.4 genauer eingegangen. Das Vorgehen in den beiden Algorithmen unterscheidet sich nur darin, dass die im folgenden Abschnitt eingeführten Erweiterungen bei der Bestimmung des Trust-Region-Radius für die nächste Iteration in Algorithmus 2.4 berücksichtigt werden müssen.

2.2.3 Erweiterungen des Basic-Filter-SQP-Algorithmus

Nachdem das Grundkonzept eines Filter-Verfahrens für nichtlineare Optimierungsprobleme beschrieben wurde, das implementierbar ist und für viele Beispielprobleme konvergiert, wird der Algorithmus nun um einige Erweiterungen ergänzt. Dabei wird darauf geachtet, dass die Erweiterungen möglichst wenig Rechenaufwand benötigen und den Ablauf des Algorithmus in Situationen, in denen er bereits zufriedenstellend funktioniert, nicht unterbrechen. Das Ziel dieses Vorgehens ist es, einen Algorithmus zu entwickeln, für den ein Konvergenzbeweis geführt werden kann, da dies für Algorithmus 2.2 nicht möglich ist. Der in diesem Kapitel entstehende Filter-SQP-Algorithmus bildet somit die Grundlage für den Nachweis der globalen Konvergenz von Filter-SQP-Verfahren, der in einer anderen Veröffentlichung [11], nach ein paar weiteren Modifikationen am Algorithmus, erbracht wurde, hier aber nicht thematisiert werden soll. Stattdessen wird der Basic-Filter-SQP-Algorithmus schrittweise um sechs Erweiterungen ergänzt, wodurch der vollständige Filter-SQP-Algorithmus entsteht.

Korrekturschritt zweiter Ordnung
Eine wichtige Eigenschaft eines jeden SQP-Algorithmus ist die gute Konvergenzgeschwindigkeit in der Nähe eines Minimums, da sich ein SQP-Verfahren nahe der Lösung einem Newton-Verfahren annähert. Dies hat zur Folge, dass das SQP-Verfahren wie das Newton-Verfahren in diesem Bereich quadratisch konvergiert. Allerdings kann die Verwendung einer Straffunktion dieses Konvergenzverhalten verhindern, da durch sie nicht immer der ganze Newton-Schritt $x_k + s_k$, wobei x_k der letzte Iterationspunkt und s_k der berechnete Iterationsschritt ist, akzeptiert wird, sondern nur ein kleinerer Schritt $x_k + \lambda s_k$ mit $0 \leq \lambda \leq 1$ in die selbe Richtung. Diese Situation kann durch eine nicht differenzierbare Straffunktion verursacht beliebig nahe der eigentlichen Lösung des Problems auftreten und wird als Maratos-Effekt bezeichnet [16].
Der Maratos-Effekt kann vermieden werden, indem man einen Korrekturterm berechnet, der die Beiträge der zweiten Ableitung in den Nebenbedingungen der quadratischen Teilprobleme berücksichtigt. Man bezeichnet dieses Vorgehen als Korrekturschritt zweiter Ordnung (SOC). Ein weiterer Vorteil bei der Verwendung von SOC-Schritten ist, dass damit oft die Verletzung der Nebenbedingung im Testpunkt reduziert wird. Dadurch wird die Wahrscheinlichkeit, dass er für den aktuellen Trust-Region-Radius Δ akzeptiert wird, erhöht und damit auch die Wahrscheinlichkeit, Δ nicht verkleinern zu müssen. Dies ist gewöhnlich wünschenswert, da bei einem kleinen Trust-Region-Radius auch nur kleine Iterationsschritte gemacht werden können, was die Konvergenzgeschwindigkeit verringert und eventuell zu einem unzulässigen

Teilproblem führen kann. Dies würde wiederum eine vermeidbare Restaurationsphase nach sich ziehen.
Der SOC-Schritt wird berechnet, wenn $x^{(k+1)} =: x^{(k+1,0)}$ vom Filter des Basic-filter-SQP-Algorithmus nicht akzeptiert wird, dabei wird ein zweiter Index eingeführt, um neben dem Iterationszähler der SQP-Schritte auch den Zähler der SOC-Schritte festzuhalten. Dazu wird ein neues quadratisches Teilproblem der Form

$$(QP_2)\begin{cases} \min_d & \frac{1}{2}d^T W^{(k)} d + d^T g^{(k)} \\ s.t. & {A^{(k)}}^T d + c^{(k+1,l)} - {A^{(k)}}^T d^{(k,l)} \leq 0 \\ & \|d\|_\infty \leq \Delta \end{cases}$$

aufgestellt. Dabei ist $l \in \mathbb{N}_0$ mit $c^{(k+1,0)} := c^{(k+1)}$ und $d^{(k,0)} := d^{(k)}$, der nicht akzeptierte Testschritt, also die Lösung von $(QP^{(k)})$. Im Unterschied zu $(QP^{(k)})$ werden in (QP_2) die Nebenbedingungen auch quadratisch approximiert. Eine ausführliche Herleitung von (QP_2) ist in [8] zu finden. Sei $\hat{d}^{(k)} := d^{(k,l+1)}$ die Bezeichnung der Lösung von (QP_2).
Der Punkt $x^{(k+1,l)} = x^{(k)} + d^{(k,l+1)}$, der durch Verwendung des SOC-Testschritts erzeugt wurde, wird dann vom Filter getestet. Ist $\left(f^{(k+1,l)}, h^{(k+1,l)}\right)$ akzeptabel, dann wird der Schritt akzeptiert und der Trust-Region-Radius kann vergrößert werden. Andernfalls wird eine Folge von SOC-Schritten berechnet, die eine Folge von Testpunkten $x^{(k+1,l)}$, $l \in \mathbb{N}_0$ erzeugt, bis eine der folgende Eigenschaften erfüllt ist:

1. Ein Punkt $x^{(k+1,L)}$ wird gefunden, der von Filter akzeptiert wird.
2. Ein unzulässiges quadratisches Teilproblem wird entdeckt.
3. Die Konvergenzrate $r = \frac{h^{(k+1,l)}}{h^{(k+1,l-1)}}$ der SOC-Schritte ist zu gering.
4. Ein beinahe zulässiger Punkt mit $h^{(k+1,l)} < \epsilon$ wird erzeugt, wobei ϵ die Toleranzgrenze des zur Lösung von (QP_2) verwendeten Verfahrens ist.

Im ersten Fall ist der nächste Iterationspunkt $x^{(k+1)} = x^{(k+1,L)}$. Der Iterationsschritt war also erfolgreich und es kann mit dem nächsten Teilproblem $(QP^{(k+1)})$ fortgefahren werden. In den anderen drei Fällen wird der Iterationsschritt abgelehnt und der Trust-Region-Radius Δ wird trotz SOC-Schritten verkleinert.

Obere Schranke für die Verletzung der Nebenbedingungen
Es kann vorkommen, dass eine Folgen von Punkten mit $f^{(k+1)} < f^{(k)}$, $h^{(k+1)} > h^{(k)}$ und $h^{(k)} \to \infty$ vom Filter akzeptiert wird. Um dies zu vermeiden, führt man eine zusätzliche Bedingung für die Akzeptanz eines Punktes ein. Man legt eine obere Schranke u für die Verletzung der Nebenbedingungen fest und fordert $h\left(c(x)\right) \leq u$. Dazu wird der Filter mit dem Eintrag $(-\infty, u)$ initialisiert. Dieser Eintrag steht somit als Vergleichskriterium für alle Iterationspunkte des SQP-Algorithmus zur Verfügung.

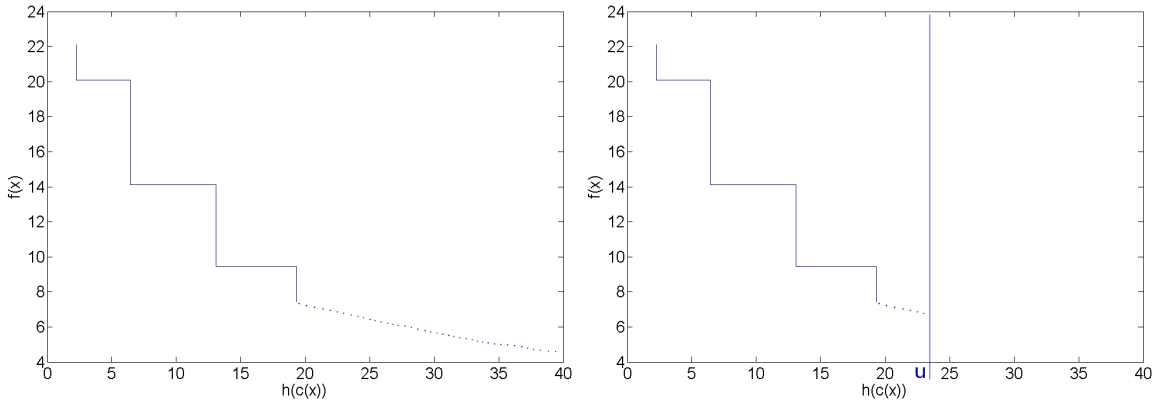


Abbildung 2.4: Das linke Bild zeigt die oben beschriebene Situation, in der sehr viele unnötige Iterationspunkte (durch die gepunktete Linie dargestellt) vom Filter akzeptiert werden. Im rechten Bild sieht man, wie dies durch die Schranke u verhindert wird.

Eine zusätzlich Funktion übernimmt die Schranke u in Verbindung mit der Restaurationsphase, die im nächsten Abschnitt erläutert wird. Wird nämlich ein Punkt, der als Ergebnis einer Restaurationsphase vom Filter auf Akzeptanz getestet wird, abgelehnt, so werden die Filtereinträge, welche die Akzeptanz des neuen Punktes verhindern, gelöscht. Um zu vermeiden, dass dieses Vorgehen zu einem Kreisen des Algorithmus führt, wird gleichzeitig der Wert u verringert.
Es ist im Sinne der Idee des Filters, dass die Schranke möglichst selten verwendet werden muss. Deshalb wird in der Regel ein großer Wert für u gewählt. Ein Vorschlag dafür wäre beispielsweise:

$$u := \max\left\{ubd, 1.25 \cdot h\left(c(x_H)\right)\right\},$$

wobei ubd eine problemabhängige Konstante und x_H der Filtereintrag mit der größten Nebenbedingungsverletzung ist. In [9] wird beispielsweise der Wert $ubd = 100$ verwendet. Graphisch gesehen bedeutet diese zusätzliche Bedingung, dass alle unzulässigen Iterationswerte in einem Intervall um die Grenze des zulässigen Gebiets liegen müssen.

Restaurationsphase
Die schon mehrfach erwähnte Restaurationsphase ist eine Möglichkeit, dem unerwünschten Effekt eines unzulässigen quadratischen Teilproblems zu begegnen.
Die Unzulässigkeit eines $(QP^{(k)})$ kann, bei Verwendung eines SQP-Trust-Region-Ansatzes, aus der wiederholten Verkleinerung des Trust-Region-Radius resultieren. Eine weitere Ursache dafür kann die lineare Approximation der nichtlinearen Nebenbedingungen sein, da diese möglicherweise das zulässige Gebiet des Ausgangsproblems nicht erhalten.

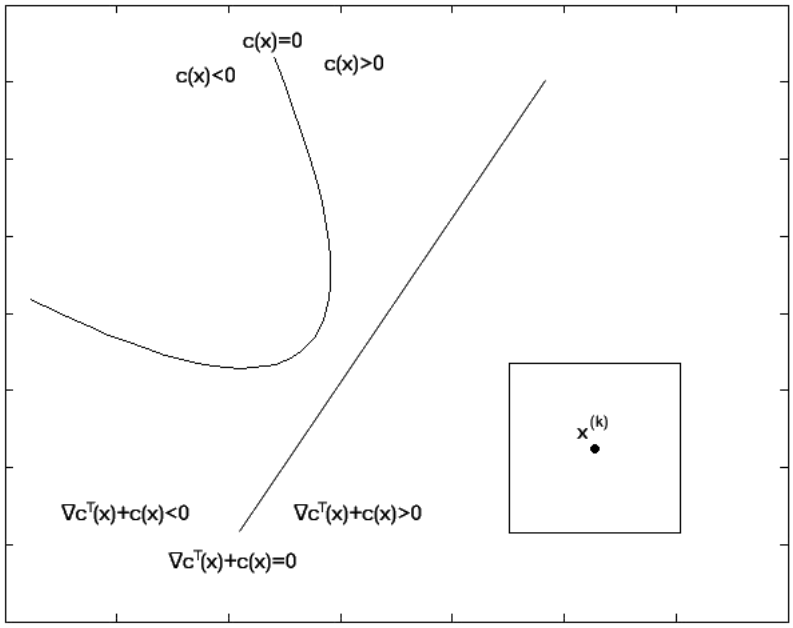


Abbildung 2.5: Beispiel für ein unzulässiges $(QP^{(k)})$, das durch einen zu kleinen Trust-Region-Radius verursacht wird. Die rechteckige Form der Trust-Region resultiert aus der Verwendung der ℓ_∞-Norm.

In solchen Fällen ist die Verwendung einer Restaurationsphase sinnvoll. Dafür unterteilt man zunächst die Nebenbedingungen eines Teilproblems $(QP^{(k)})$ in die beiden Indexmengen

$$J \subset \{1, 2, ..., m\} \text{ und } J^{\perp} = \{1, 2, ..., m\} \setminus J.$$

Dabei enthält die Menge J die Indizes der durch $x^{(k)}$ verletzten Nebenbedingungen und $J^{\perp}$ die der erfüllten Nebenbedingungen. Um sich dem Bereich der zulässigen Lösungen von $(QP^{(k)})$ zu nähern, wird das Problem

$$(LP_1) \begin{cases} \min_d & \sum_{j \in J} {a_j^{(k)}}^T d \\ s.t. & {a_j^{(k)}}^T d + c_j^{(k)} \leq 0, \quad j \in J^{\perp} \\ & \|d\|_\infty \leq \Delta \end{cases}$$

gelöst. Dabei ist $a_j^{(k)} = \nabla c_j \left(x^{(k)}\right)$, c der Vektor der Nebenbedingungen aus $(QP^{(k)})$ und Δ der Trust-Region-Radius. Durch die Minimierung dieser Zielfunktion wird versucht, die Verletzung der Nebenbedingungen zu reduzieren, da ein negativer Wert von ${a_j^{(k)}}^T d$ gleichbedeutend ist mit einer Verringerung des Wertes der Nebenbedingung j durch den Schritt d. Die verwendeten Nebenbedingungen ${a_j^{(k)}}^T d + c_j^{(k)} \leq 0, \quad j \in J^{\perp}$ sorgen dafür, dass alle bereits erfüllten Nebenbedingungen von $(QP^{(k)})$ auch weiterhin

erfüllt bleiben.
Das Ziel der Restaurationsphase ist es, das nichtlineare Optimierungsproblem

$$(F)\begin{cases} \min_x & \sum_{j\in J} c_j^+(x) \\ \\ s.t. & c_j(x) \leq 0, \quad j \in J^\perp \end{cases}$$

zu lösen, das durch die Mengen J und $J^\perp$ definiert wird. Das Vorgehen dabei ist folgendes: Es werden in einer Folge von SQP-ähnlichen Iterationsschritten Probleme der Art (LP_1) gelöst, um eine Schrittweite d für eine Lösung des Problems (F) zu bestimmen. Der dadurch erzeugte Punkt ist, nach Konstruktion von (F), ein zulässiger Punkt des Problems $(QP^{(k)})$ und das eigentliche SQP-Verfahren kann mit diesem Startpunkt den nächsten Iterationsschritt durchführen.
Nach jedem Iterationsschritt in der Restaurationsphase werden gegebenenfalls die Mengen J und $J^\perp$ aktualisiert. Danach wird die Zulässigkeit des Systems

$$(LP)\begin{cases} {a_j^{(k)}}^T dc_j^{(k)} \leq 0, \qquad j = 1, 2, ..., m \\ \\ \quad \|d\|_\infty \leq \Delta \end{cases}$$

überprüft. Dieses Gleichungssystem wurde aus den Nebenbedingungen von $(QP^{(k)})$ abgeleitet. Falls ein Punkt, der für (LP) zulässig ist, gefunden wird, endet die Restaurationsphase.
Andernfalls wird ein neues quadratisches Teilproblem bestimmt, indem man den Term $\frac{1}{2}d^T W^{(k)} d$ in die Zielfunktion von (LP_1) einfügt und so die zweiten Ableitungen der Nebenbedingungen von $(QP^{(k)})$ berücksichtigt. Man erhält:

$$(QP_{R-Ph})\begin{cases} \min_d & \sum_{j\in J} {a_j^{(k)}}^T d + \frac{1}{2}d^T W^{(k)} d \\ \\ s.t. & {a_j^{(k)}}^T d + c_j^{(k)} \leq 0, \quad j \in J^\perp \\ \\ & \|d\|_\infty \leq \Delta. \end{cases}$$

Dabei ist

$$W(x, \lambda) = \sum_{j\in J} \nabla^2 c_j(x) + \sum_{j\in J^\perp} \lambda_j \nabla^2 c_j(x).$$

Die Lagrange-Multiplikatoren λ_j werden mit Hilfe der Lösung von (LP_1) bestimmt. Diese wird demnach zur Initialisierung des neuen quadratischen Problems (QP_{R-Ph}) verwendet.
Die bisherigen Betrachtungen für den Basic-filter-SQP-Algorithmus können auch hier angewandt werden. Demnach kann auch in der Restaurationsphase eine Folge von SOC-Schritten verwendet werden, falls die Lösung von (QP_{R-Ph}) nicht als nächster Iterationsschritt akzeptiert wird.
Es gibt für die Lösung des Problems (F) zwei konkurrierende Ziele, analog zur Darstellung des Problems (P): die Aufrechterhaltung der Zulässigkeit der Bedingungen

aus $J^\perp$ und die Minimierung der Verletzung der Nebenbedingungen aus J. Zwar gibt es für (F) eine geeignete Straffunktion, nämlich die Summe über die gesamte Verletzung der Nebenbedingungen. Trotzdem erhält man durch den Gebrauch eines Filters eine höhere Flexibilität in der Akzeptanz von Iterationsschritten, weswegen die Verwendung eines Filters auch in der Restaurationsphase sinnvoll ist.

Definition 2.9 *Ein* Restaurationsfilter *ist eine Liste bestehend aus Paaren* $(h_J, h_{J^\perp})$*, in der kein Paar ein anderes dominiert. Dabei sind*

$$h_J := h\left(c_J(x)\right) := \sum_{j \in J} c_j^+(x)$$

und

$$h_{J^\perp} := h\left(c_{J^\perp}(x)\right) := \sum_{j \in J^\perp} c_j^+(x).$$

Da sich die Mengen J und $J^\perp$ in jedem Iterationsschritt ändern, wäre es ideal, auch die Filtereinträge entsprechend zu aktualisieren. Dies würde jedoch bedeuten, dass alle Werte $c_j\left(x^l\right)$ für alle Einträge l im Filter gespeichert werden müssten. Um Speicherplatz einzusparen verzichtet man auf das Anpassen der Filtereinträge. Stattdessen werden Einträge, die vor einer Änderung der Menge J in den Filter aufgenommen wurden, gelöscht, wenn sie Einträge blockieren, die nach dieser Änderung von J im Filter getestet werden. Die Gefahr beim Entfernen von Filtereinträgen ist immer, dass der Algorithmus zu kreisen beginnt. Dem beugt man vor, indem die Schranke u für die Verletzung der Nebenbedingungen auf den Wert

$$u = \max\left(\hat{h}^{(k,L)}, \frac{u}{10}\right)$$

verkleinert wird. Dabei ist $\hat{h}^{(k,L)}$, wie zu Beginn dieses Kapitels definiert, die ℓ_1-Norm der Nebenbedingungsverletzung im Punkt $\hat{x}^{(k,L)}$, dabei sei $\hat{x}^{(k,L)}$ der Punkt, der im Iterationsschritt k der Restaurationsphase als Kandidat für den Testpunkt ermittelt wurde.
Als Resultat der Restaurationsphase erhält man entweder einen für $(QP^{(k)})$ zulässigen Iterationspunkt und kann im eigentlichen SQP-Verfahren fortfahren, oder die Lösung von (F) bleibt unzulässig für $(QP^{(k)})$. Im zweiten Fall muss der Algorithmus abgebrochen werden, da das Ausgangsproblem lokal unzulässig ist.
Mit Hilfe dieser Überlegungen lässt sich nun eine Beschreibung der Restaurationsphase des Filter-SQP-Algorithmus angeben:

Algorithmus 2.3 *Restaurationsphase*

Ein $x^{(k)}$ *(unzulässig für* $(QP^{(k)})$*), der Trust-Region-Radius* Δ *und die Schranke* u *aus dem SQP-Filter sind gegeben.*
Repeat

- *Löse* (LP_1).

- *Bestimme die Indexmengen J und $J^{\perp}$.*

If (LP_1) *ist zulässig* **then**
- *Kehre zum SQP-Algorithmus zurück.*
- *Lösche den Restaurationsfilter.*

else
- *Löse* (QP_{R-Ph}).
- $d^{(k)}$ *sei die Lösung von* (QP_{R-Ph}).
- *Setze* $x^{(k+1)} = x^{(k)} + d^{(k)}$.

If $(h_J^{(k+1)}, h_{J^{\perp}}^{(k+1)})$ *ist akzeptabel für Restaurationsfilter* **then**
- *Akzeptiere* $x^{(k+1)}$ *und füge* $(h_J^{(k+1)}, h_{J^{\perp}}^{(k+1)})$ *zum Restaurationsfilter hinzu.*
- *Entferne alle Punkte, die von* $(h_J^{(k+1)}, h_{J^{\perp}}^{(k+1)})$ *dominiert werden, aus dem Filter.*
- *Vergrößere eventuell den Trust-Region-Radius* Δ.

else
- *Löse eine Folge von SOC-Schritten.*
- $\hat{d}^{(k)}$ *sei Ergebnis der SOC-Schritte.*
- *Setze* $x^{(k+1)} = x^{(k)} + \hat{d}^{(k)}$.

If $(\hat{h}_J^{(k+1)}, \hat{h}_{J^{\perp}}^{(k+1)})$ *ist für den Restaurationsfilter akzeptabel* **then**
- *Akzeptiere* $\hat{x}^{(k+1)}$ *und füge* $(\hat{h}_J^{(k+1)}, \hat{h}_{J^{\perp}}^{(k+1)})$ *zum Restaurationsfilter hinzu.*
- *Entferne alle Punkte, die von* $(\hat{h}_J^{(k+1)}, \hat{h}_{J^{\perp}}^{(k+1)})$ *dominiert werden, aus dem Filter.*
- *Vergrößere eventuell den Trust-Region-Radius* Δ.

else

If J *wurde seit dem Iterationsschritt* $k-1$ *geändert und* $\hat{h}^{(k,L)} < u$ **then**
- *Akzeptiere den besten Wert aus den SOC-Schritten (setze* $x^{(k+1)} = \hat{x}^{(k+1)}$*).*
- *Entferne alle diesen Eintrag blockierenden Werte aus dem Filter.*
- *Reduziere die Schranke der Nebenbedingungverletzung auf* $u = \max(\hat{h}^{(k,L)}, \frac{u}{10})$.

else
- *Lehne den Iterationsschritt ab (setze* $x^{(k+1)} = x^{(k)}$*).*
- *Verkleinere den Trust-Region-Radius* Δ.

endif

endif

endif

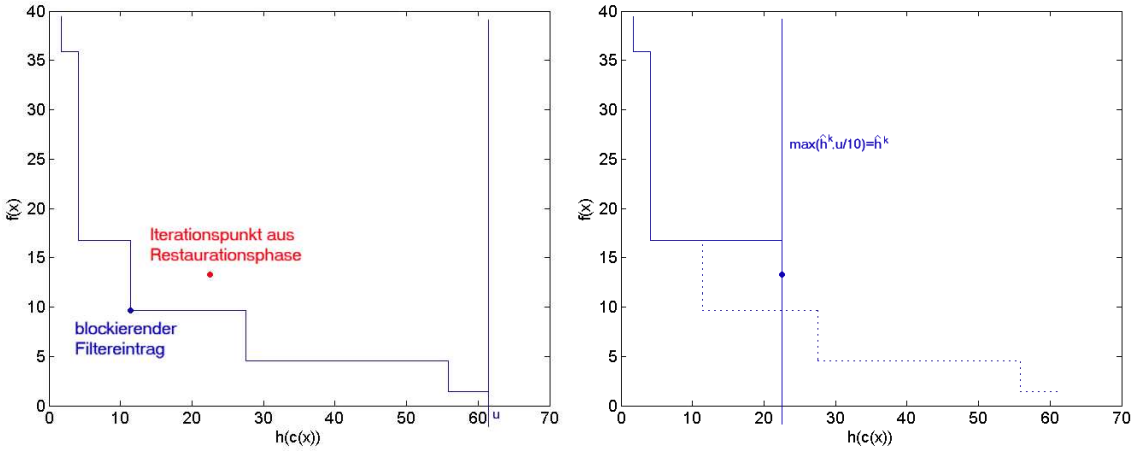
Setze $k = k + 1$.

until *Konvergenz oder Rückkehr zum SQP-Algorithmus*

Nach Abschluss der Restaurationsphase wird der Wert der oberen Schranke für die Verletzung der Nebenbedingungen u auf den Wert, den er vor der Restaurationsphase hatte zurückgesetzt. Der Trust-Region-Radius Δ dagegen wird dem Filter-SQP-Algorithmus übergeben, da er die Genauigkeit der linearen Approximation an die Zielfunktion $f(x)$ im Punkt $x^{(k)}$ wiederspiegelt.

Einträge aus dem Filter entfernen

Man betrachtet nun den Fall, dass (2.21) eine globale Lösung x^{**} und eine schlechtere lokale Lösung x^* besitzt. Außerdem sei $x^{(0)}$ ein zulässiger Punkt, der nahe bei x^{**} liegt. Es gibt jedoch eine Teilfolge von Iterationsschritten $x^{(k)}$, die gegen x^* konvergiert. Gilt dann $f^{(0)} \leq f^*$, so verhindert der Filtereintrag $\left(f^{(0)}, h^{(0)}\right)$, dass das lokale Minimum angenommen werden kann. Dies wäre jedoch wünschenswert, falls die Folge der Iterationswerte nicht gegen das globale Minimum konvergieren kann.
In diesem Fall wäre es von Vorteil, eine backtracking-Strategie zurück zu $x^{(0)}$ verwenden zu können. Da man aber nicht die vorherigen Iterationspunkte $x^{(j)}$ sondern nur die Filtereinträge $\left(f^{(j)}, h^{(j)}\right)$ gespeichert hat, ist dies nicht möglich. Man nennt $\left(f^{(0)}, h^{(0)}\right)$ einen blockierenden Eintrag. Um die Konvergenz gegen x^* zu ermöglichen, werden Iterationspunkte, die aus einer Restaurationsphase stammen, stets in den Filter aufgenommen, auch wenn sie eigentlich nicht akzeptiert werden. Folglich muss jeder Filtereintrag, der einen solchen Punkt blockiert, gelöscht werden.
Das Löschen von Einträgen kann zu einem Kreisen führen, zum Beispiel wenn der Algorithmus in einem späteren Iterationsschritt zu $x^{(0)}$ zurückkehrt. Um das zu verhindern, verringert man, wie bereits erwähnt, die obere Schranke für die Verletzung der Nebenbedingungen auf den Wert $u = \max\left(\hat{h}^{(k)}, \frac{u}{10}\right)$, wobei $\hat{h}^{(k)}$ wieder die ℓ_1-Norm der Verletzung der Nebenbedingungen im neuen Punkt, der zum Filter hinzu gefügt wird, ist. Da sich diese Schranke wie ein Filtereintrag mit $f = -\infty$ verhält, wird dadurch in Verbindung mit der im nächsten Abschnitt erläuterten Mindestverbesserung ein Kreisen verhindert. Wird der Wert der Schranke immer wieder reduziert, so nähern sich die Iterationswerte $x^{(k)}$ beliebig weit der Grenze des zulässigen Gebietes an.



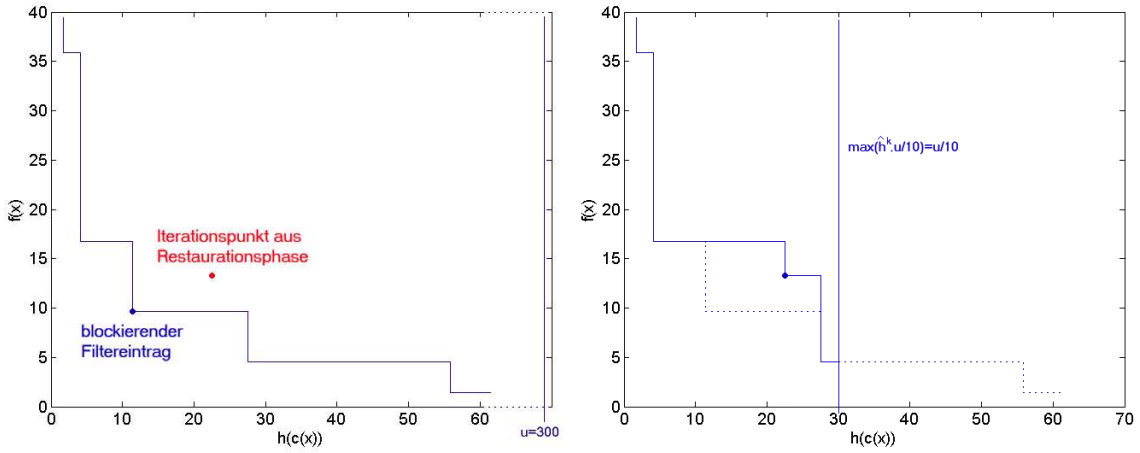


Abbildung 2.6: Die Abbildungen zeigen das Vorgehen, falls ein blockierender Filtereintrag gelöscht werden muss, insbesondere die Anpassung der Schranke u für die Verletzung der Nebenbedingungen. In der oberen Reihe ist $\max(\hat{h}^{(k)}, \frac{u}{10}) = \hat{h}^{(k)}$, in der unteren wird der Fall $\max(\hat{h}^{(k)}, \frac{u}{10}) = \frac{u}{10}$ dargestellt. Die gestrichelte Linie in den beiden rechten Abbildungen deutet den früheren Verlauf des Filters an.

Ausreichende Verbesserung

Die Akzeptanzbedingung des Filter besagt, dass ein neuer Iterationspunkt nicht von einem Eintrag aus dem Filter dominiert werden darf. Sie ermöglicht Häufungspunkte der Folge der Iterationswerte im (f, h)-Raum, die keine KKT-Punkte sind. Um diese Situation zu vermeiden, wird in Algorithmen, die mit einer Straffunktion arbeiten, üblicherweise gefordert, dass in jedem Iterationsschritt eine ausreichende Verbesserung des Wertes der Straffunktion erreicht wird. Dieses Prinzip wird für den Filter-SQP-Algorithmus angepasst. Es wird ein Korridor um den Filter erzeugt, der verhindert, dass neue Punkte, die sehr nahe an den bisherigen Einträgen liegen, akzeptiert werden.

Das bedeutet, ein neuer Iterationswert $x^{(k+1)}$ ist akzeptabel für den Filter, wenn

$$h^{(k+1)} \leq \beta h^{(l)} \tag{2.24}$$

oder

$$f^{(k+1)} \leq f^{(l)} - \max\left(\alpha_1 \Delta q^{(l)}, \alpha_2 h^{(l)} \mu^{(l)}\right) \tag{2.25}$$

für jeden Filtereintrag l gilt. Dabei sind α_1, α_2 und β kleine positive Konstanten (beispielsweise: $\alpha_1 = 0.99$, $\alpha_2 = 0.25$ und $\beta = 0.0001$). Die Ungleichung (2.24) erzeugt den Korridor parallel zu den senkrechten Abschnitten des Filters. Parallel zu den waagerechten Teilen entsteht er durch die zweite Ungleichung, wobei $\Delta q^{(l)} := f^{(l)} - f^{(l-1)}$ die Verbesserung der Zielfunktion f aus $(QP^{(k)})$ durch $x^{(l)}$ ist.

Dabei ist darauf zu achten, dass $\Delta q^{(l)} > 0$ gilt, wenn eine Verringerung des Wertes von f erreicht wird. Da es durchaus auch möglich ist, dass ein Anstieg des Wertes von f durch $x^{(l)}$ verursacht wird, ist es nötig, den Korridor in diesem Fall anders zu

definieren. Das erreicht man, indem man einen Schätzwert $\mu^{(l)}$ bestimmt, der gleich der kleinsten Zehnerpotenz ist, die größer als $\left\|\lambda^{(l)}\right\|_\infty$ ist, wobei $\lambda^{(l)}$ der Vektor der zugehörigen Lagrange-Multiplikatoren ist. Um keine zu großen oder zu kleinen Werte zu erhalten, beschränkt man $\mu^{(l)}$ aber auf das Intervall $[10^{-6}, 10^6]$. Der Wert $h^{(l)}\mu^{(l)}$ kann dann als Alternative zur von $x^{(l)}$ erzeugten Reduzierung des Wertes von f verwendet werden. Beide Werte $\Delta q^{(l)}$ und $\mu^{(l)}$ werden zusammen mit den Koordinaten $f^{(l)}$ und $h^{(l)}$ der Filtereinträge gespeichert, um für spätere Akzeptanztests anderer Punkte verfügbar zu sein.

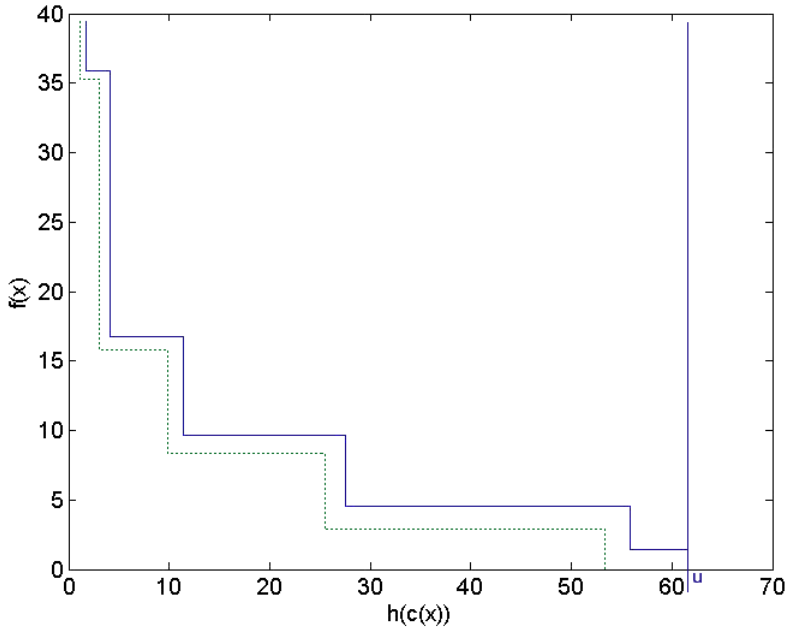


Abbildung 2.7: Darstellung eines Filters (durchgezogene Linie) mit dazugehöriger Mindestverbesserung (gestrichelte Linie).

Die North-West- und die South-East-Regel

Der Filter ist ein gutes Kriterium um zu entscheiden, ob ein Iterationsschritt akzeptiert werden sollte, solange der Wert der Verletzung der Nebenbedingungen des neuen Punktes, also $h^{(k+1)}$, nicht kleiner bzw. größer als der kleinste bzw. größte Wert der bereits vorhandenen Filtereinträge ist.

Es besteht beispielsweise die Möglichkeit, dass eine Folge von Filtereinträgen erzeugt wird, deren Werte $\{f^{(k)}\}$ monoton steigen, aber die dazugehörenden Werte $\{h^{(k)}\}$ monoton fallen. In diesem Fall konvergiert diese Teilfolge nicht gegen einen KKT-Punkt. Jedoch werden die Iterationspunkte aus dieser Folge in den Filter aufgenommen, falls sie eine hinreichende Verbesserung bezüglich der Verletzung der Nebenbedingungen bewirken.

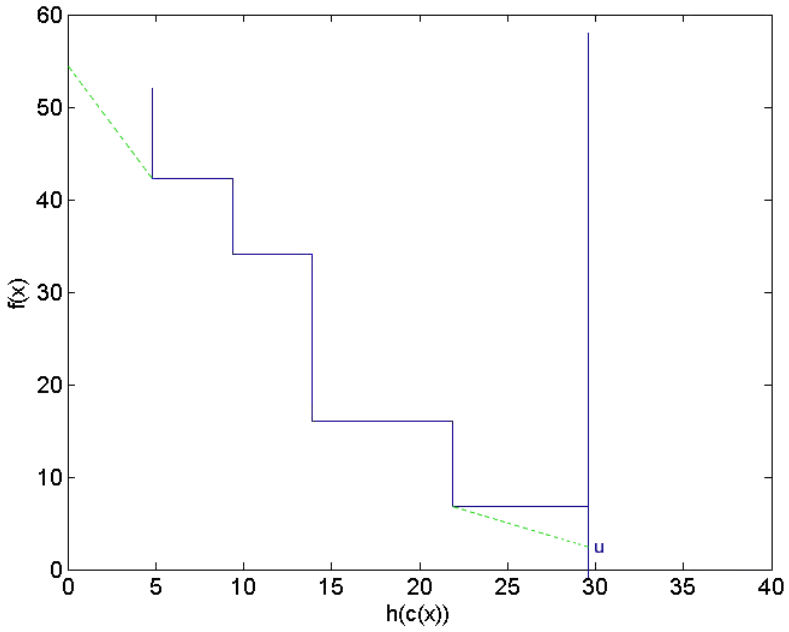


Abbildung 2.8: Darstellung der North-East- und der South-West-Regel als gestrichelte Linie.

Um zu vermeiden, dass solche, für die Lösung des Problems wenig hilfreichen Punkte in den Filter aufgenommen werden, wird eine zusätzliche Akzeptanzbedingung in Form einer Straffunktion eingeführt, die vom Filtereintrag $\left(f^{(1)}, h^{(1)}\right)$ mit der kleinsten Nebenbedingungsverletzung $h^{(1)}$ abhängt. Betrachtet man die graphische Darstellung eines Filters, entspricht dies dem Filtereintrag, der am weitesten links steht. Der neue Iterationspunkt $x^{(k+1)}$ muss dann zusätzlich zur ausreichenden Reduktion, die im vorangegangenem Abschnitt erläutert wurde, eine Verringerung des Wertes der Straffunktion, die unter Verwendung des Parameters $\mu = 1000\mu^{(1)}$ entsteht, erreichen. Der Wert dieser Straffunktion für den Referenzpunkt $\left(f^{(1)}, h^{(1)}\right)$ ist $f^{(1)} + \mu h^{(1)}$. Dies bedeutet, dass ein neuer Punkt, für den $h^{(k+1)} \leq h^{(1)}$ gilt, nur dann akzeptiert wird, wenn er die zusätzliche Bedingung

$$f^{(k+1)} + \mu h^{(k+1)} \leq f^{(1)} + \mu h^{(1)}$$

erfüllt. Dies wird auch North-West-Regel genannt.
Äquivalent soll durch die South-East-Regel verhindert werden, dass eine Folge von Punkten in den Filter aufgenommen wird, für die gilt: $h \to \infty$ und $f \to 0$. Zwar ist hier schon die obere Schranke u für die Nebenbedingungsverletzung eingeführt worden, aber deren Anfangswert wird in der Regel so groß gewählt, dass sich der Einsatz der zusätzlichen South-East-Regel rechtfertigt. Es wird die gleiche Akzeptanzbedingung wie bei der North-West-Regel verwendet, allerdings wählt man $\mu = \frac{\mu^{(L)}}{1000}$, wobei $\mu^{(L)}$ sich auf den Filtereintrag mit der größten Nebenbedingungsverletzung bezieht.

Graphisch gesprochen ist Eintrag $\left(f^{(L)}, h^{(L)}\right)$ der am weitesten rechts liegende.

Der Filter-SQP-Algorithmus
Die in diesem Kapitel vorgestellten Methoden dienen dazu, den Basic-Filter-SQP-Algorithmus zu erweitern, um Stabilität zu gewährleisten und die Konvergenzgeschwindigkeit zu erhöhen. Das Resultat der Erweiterung wird im folgenden Algorithmus dargestellt:

Algorithmus 2.4 *Filter-SQP*

Ein Startwert $x^{(0)}$ und ein Trust-Region-Radius $\Delta^{(0)}$ sind gegeben. Setze $k = 0$.
Repeat
Löse $(QP^{(k)})$, um eine Schrittweite $d^{(k)}$ für den Testschritt zu erhalten.
If *(QP) ist unzulässig* **then**
- *Führe die Restaurationsphase durch (siehe Algorithmus 2.3).*
- *Kehre zum SQP-Algorithmus zurück, wenn ein $x^{(k+1)}$ gefunden wurde, dessen zugehöriges QP zulässig ist, andernfalls: Abbruch des Algorithmus, da das Problem lokal unzulässig ist.*

else
Setze $x^{(k+1)} = x^{(k)} + d^{(k)}$.
If *$(f^{(k+1)}, h^{(k+1)})$ ist akzeptabel für den Filter* **then**
- *Akzeptiere $x^{(k+1)}$ und füge $(f^{(k+1)}, h^{(k+1)})$ zum Filter hinzu.*
- *Entferne alle Punkte, die von $(f^{(k+1)}, h^{(k+1)})$ dominiert werden aus dem Filter.*
- *Vergrößere eventuell den Trust-Region-Radius Δ.*

else
If *$(h^{(k+1)} > 0)$* **then**
- *Löse eine Folge von SOC-Schritten.*
- *$\hat{d}^{(k)}$ sei Lösung der SOC-Schritte.*
- *Setzte $\hat{x}^{(k+1)} = x^{(k)} + \hat{d}^{(k)}$.*

If *$(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ ist akzeptabel für den Filter* **then**
- *Akzeptiere $\hat{x}^{(k+1)}$ und füge $(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ zum Filter hinzu.*
- *Entferne alle Punkte, die von $(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ dominiert werden aus dem Filter.*
- *Vergrößere eventuell den Trust-Region-Radius Δ.*

endif
endif
If *kein zulässiger Punkt wurde gefunden* **then**
If *erster Iterationsschritt nach einer Restaurationsphase* **then**
- *Akzeptiere den besten Wert aus einem SOC-Schritt.*
- *Setze $x^{(k+1)} = \hat{x}^{(k+1)}$.*
- *Entferne alle diesen Eintrag blockierenden Werte aus dem Filter.*
- *Verkleinere die Schranke u auf den Wert $u = \max(\hat{h}^{(k,L)}, \frac{u}{10})$.*

- **else**

- *Lehne den Iterationsschritt ab.*
- *Setze* $x^{(k+1)} = x^{(k)}$.
- *Verkleinere den Trust-Region-Radius* Δ.

endif
endif
endif
endif
Setze $k = k + 1$.
until *Konvergenz*

In Algorithmus 2.4 ist das Kriterium, nach dem entschieden wird, ob der Trust-Region-Radius erhöht wird, nicht spezifiziert. In [9] wird Δ genau dann vergrößert, wenn $\Delta = \left\|d^{(k)}\right\|$ gilt und zusätzlich entweder der SQP-Schritt oder ein SOC-Schritt mit ausreichend guter Konvergenzrate (beispielsweise $r \leq 0.1$) akzeptiert wird. Ist diese Bedingung erfüllt, wird der Trust-Region-Radius verdoppelt: $\Delta = 2\Delta$. Soll Δ dagegen verkleinert werden, so wird der neue Wert durch $\Delta = 0.5 \cdot \min\left(\Delta, d^{(k)}\right)$ bestimmt.
In diesem Kapitel wurde anhand eines exemplarischen Algorithmus gezeigt, welche Motivation der Einführung des Filteransatzes zu Grunde liegt. Ein wichtiger Grund ist die Vermeidung der Schwierigkeiten, die bei Verwendung einer Straffunktion auftreten. Außerdem wurden einige Techniken vorgestellt, mit denen der erste Ansatz des Filter-SQP-Algorithmus verbessert wurde. Diese werden zwar im Filter-Trust-Region-Algorithmus nicht mehr verwendet, zeigen aber, welche Weiterentwicklung die Akzeptanzbedingung des dort verwendeten Filters darstellt, da durch sie alle hier vorgestellten Erweiterungen überflüssig werden [10].

2.3 Grundlegende Begriffe der multikriteriellen Optimierung

Der mehrdimensionale Filter, der im Filter-Trust-Region-Algorithmus eine entscheidende Rolle spielt, kann als Menge von Punkten angesehen werden, die alle Repräsentanten der effizienten Menge eines multikriteriellen Optimierungsproblems sind. Aus diesem Grund ist es nützlich, einen kurzen Exkurs in die multikriterielle Optimierung zu unternehmen, um einige grundlegende Definitionen zu erläutern. Es soll hier weder auf Methoden noch auf Lösungsverfahren für multikriterielle Optimierungsaufgaben eingegangen werden. Das Ziel dieses Abschnitts ist es lediglich, einige für das Verständnis des mehrdimensionalen Filters hilfreiche Begriffe einzuführen. Dieser Abschnitt stützt sich überwiegend auf das Buch "Multicriteria Optimization" von Matthias Ehrgott [7].
Die Multikriterielle Optimierung befasst sich mit Problemstellungen, bei denen mehrere Ziele verfolgt werden. Ein typisches Einführungsbeispiel ist der Kauf eines Produkts, wobei unter verschiedenen Alternativen eine Entscheidung getroffen werden soll. Dabei gibt es für gewöhnlich mehrere Kriterien, z.B. Funktionalität, Preis, Aussehen, usw., die bei der Kaufentscheidung eine Rolle spielen, sich vielleicht widersprechen und eventuell auch unterschiedliche Wichtigkeit für den Käufer aufweisen. Mathematisch formuliert, entspricht dies der gleichzeitigen Minimierung mehrerer Zielfunktion oder der Minimierung einer vektorwertigen Zielfunktion, wobei jede Funktion bzw. Komponente des Funktionsvektors einem Entscheidungskriterium entspricht. Die Aufgabe der multikriteriellen Optimierung ist es, bezogen auf obiges Beispiel, eine oder mehrere der zur Auswahl stehenden Alternativen als, in gewisser Hinsicht, optimale Produkte auszuzeichnen und so eine Entscheidungshilfe für den Käufer zu geben.

2.3.1 Ordnungsrelationen und Ordnungskegel

Die erste Schwierigkeit bei der Bewertung der verschiedenen Werte der vektorwertigen Zielfunktion besteht darin, eine geeignete Relation zu finden, um die Werte vergleichen zu können. Da es sich bei den Funktionswerten um Vektoren aus dem $\mathbb{R}^n$ und nicht um reelle Zahlen handelt, kann man normalerweise nicht ohne weiteres entscheiden, ob ein Punkt kleiner als ein anderer ist oder nicht. Der Grund dafür ist, dass es keine kanonische Anordnung für die Vektoren des $\mathbb{R}^n$ gibt, sondern unterschiedliche Möglichkeiten existieren, um eine Ordnungsrelation auf dem $\mathbb{R}^n$ zu definieren.
Dazu seien zunächst folgende Eigenschaften einer Relation auf einer nichtleeren Teilmenge A des $\mathbb{R}^n$ definiert:

Definition 2.10 *Eine binäre Relation $R \subset A \times A$ auf der Menge A heißt*

- reflexiv*, wenn $(a, a) \in R$ für alle $a \in A$;*
- irreflexiv*, wenn $(a, a) \notin R$ für alle $a \in A$;*

- symmetrisch, *wenn für* $a, b \in A$ *gilt:* $(a, b) \in R \Rightarrow (b, a) \in R$;
- asymmetrisch, *wenn für* $a, b \in A$ *gilt:* $(a, b) \in R \Rightarrow (b, a) \notin R$;
- antisymmetrisch, *wenn für* $a, b \in A$ *gilt:* $(a, b) \in R$ *und* $(b, a) \in R \Rightarrow b = a$;
- transitiv, *wenn für* $a, b, c \in A$ *gilt:* $(a, b) \in R$ *und* $(b, c) \in R \Rightarrow (a, c) \in R$;
- zusammenhängend, *wenn für* $a, b \in A$ *gilt:* $a \neq b \Rightarrow (a, b) \in R$ *oder* $(b, a) \in R$.

Anhand dieser Eigenschaften lassen sich verschiedene Arten von Ordnungen unterscheiden:

Definition 2.11 *Eine binäre Relation* R *auf der Menge* A *heißt*

- Quasiordnung, *wenn sie reflexiv und transitiv ist;*
- Halbordnung, *wenn sie reflexiv, transitiv und antisymmetrisch ist;*
- strenge Halbordnung, *wenn sie transitiv und asymmetrisch ist;*
- totale Ordnung, *wenn sie eine zusammenhängende Halbordnung ist.*

Vergleicht man die Definitionen der Halbordnung und der totalen Ordnung, erkennt man leicht den grundlegenden Unterschied: Eine Halbordnung ist im Allgemeinen nicht zusammenhängend, d.h. es können Punkte $x, y \in A$ existieren, für die weder $(x, y) \in R$ noch $(y, x) \in R$ gilt. Das bedeutet für das Ziel beliebige Werte einer vektorwertigen Zielfunktion miteinander zu vergleichen, dass, falls dies durch eine totale Ordnung geschieht, ein eindeutiger Wert als optimal bezeichnet werden kann. Bei Verwendung einer Halbordnung dagegen ist es lediglich möglich, eine Menge von Punkten zu bezeichnen, die untereinander nicht mehr verglichen werden können, jeder für sich aber besser ist, als alle anderen Punkte, die mit ihm vergleichbar sind.
Einige häufig verwendete Relationen auf dem $\mathbb{R}^n$ sind folgendermaßen definiert:

Definition 2.12 *Seien* $x = (x_1, ..., x_n), y = (y_1, ..., y_n) \in \mathbb{R}^n$. *Dann gilt:*

- $x \leq y$, *d.h.* $x_i \leq y_i$ *für* $i = 1, ..., n$, *heißt* schwach komponentenweise Ordnung;
- $x < y$, *d.h.* $x_i \leq y_i$ *für* $i = 1, ..., n$ *und* $x \neq y$, *heißt* komponentenweise Ordnung;
- $x \ll y$, *d.h.* $x_i < y_i$ *für* $i = 1, ..., n$ *und* $x \neq y$, *heißt* streng komponentenweise Ordnung,
- $x \leq_{lex} y$, *d.h.* $x_k \leq y_k$ *mit* $k := \min \{i : x_i \neq y_i\}$ *oder* $x = y$, *heißt* lexikographische Ordnung.

Es ist leicht zu sehen, dass alle in der letzten Definition vorgestellten Ordnungen totale oder Halbordnungen auf $\mathbb{R}^n$ sind. Im einzelnen heißt das: "$\leq$" definiert eine Halbordnung auf $\mathbb{R}^n$, "$<$" und "$\ll$" beschreiben jeweils strenge Halbordnungen auf $\mathbb{R}^n$ und durch "$\leq_{lex}$" wird eine totale Ordnung auf $\mathbb{R}^n$ definiert.
Ordnungen, beispielsweise die in der letzten Definition vorgestellten, können durch geometrische Objekte im $\mathbb{R}^n$ beschrieben werden. Genauer gesagt werden Kegel verwendet, um die nichtnegativen Elemente des $\mathbb{R}^n$ bezüglich einer Ordnungsrelation darzustellen:

Definition 2.13 *Eine Teilmenge $K \subseteq \mathbb{R}$ heißt* Kegel*, wenn $\lambda x \in K$ für alle $x \in K$ und für alle $\lambda \in \mathbb{R}$ mit $\lambda > 0$.*

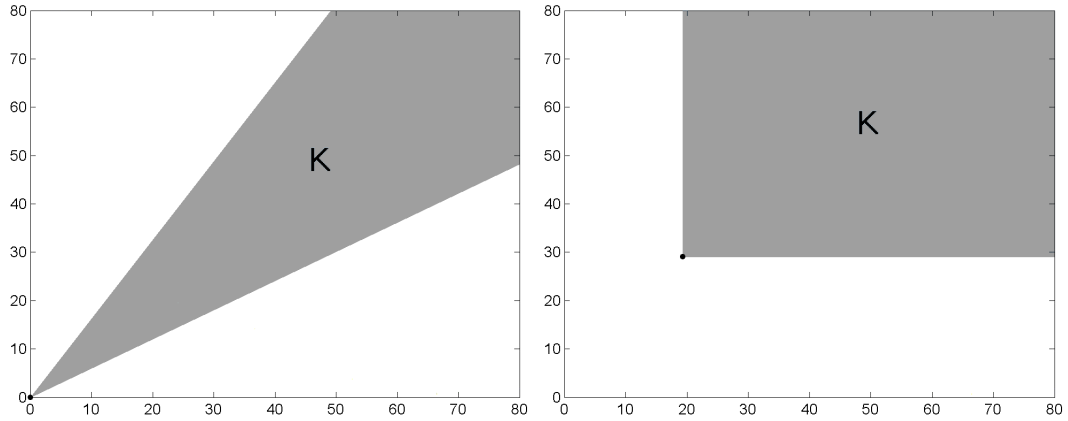


Abbildung 2.9: Im linken Bild ist ein Kegel $K \subset \mathbb{R}^2$ abgebildet. Das rechte Bild zeigt die Menge $K = \{x \in \mathbb{R}^2 : 30 \leq x_1 \wedge 20 \leq x_2\}$, die durch das Verschieben eines Kegels an den Punkt $(20, 30)$ entsteht.

Um die Beziehung zwischen Kegeln und Ordnungsrelationen genauer untersuchen zu können, benötigt man folgende Definition:

Definition 2.14 *Eine Ordnungsrelation $\preceq$ auf dem $\mathbb{R}^n$ ist* kompatibel bezüglich der Skalarmultiplikation*, wenn*

$$\forall x, y \in \mathbb{R}^n, \forall \lambda \in \mathbb{R} \text{ und } \lambda > 0 : \; x \preceq y \Rightarrow \lambda x \preceq \lambda y$$

gilt. Die Ordnungsrelation $\preceq$ auf dem $\mathbb{R}^n$ ist kompatibel bezüglich der Addition*, wenn*

$$\forall x, y, z \in \mathbb{R}^n : \; x \preceq y \Rightarrow x + z \preceq y + z.$$

Es ist leicht ersichtlich, dass alle in Definition 2.12 beschriebenen Relationen kompatibel bezüglich Addition und Skalarmultiplikation sind.
Durch folgenden Satz wird deutlich, wie der zu einer vorgegeben Ordnungsrelation gehörende Kegel definiert ist:

Satz 2.11 *$\preceq$ sei eine Ordnungsrelation auf dem $\mathbb{R}^n$, die kompatibel zur Skalarmultiplikation ist. Dann ist die Menge*

$$K_{\preceq} := \{y - x : x \preceq y\}$$

ein Kegel. Dieser wird Ordnungskegel bezüglich $\preceq$ *genannt.*

Beweis 11 Seien $x, y \in \mathbb{R}$ und $u = y - x \in K_{\preceq}$. Dann gilt $x \preceq y$ und damit auch $\lambda x \preceq \lambda y$ für alle $\lambda > 0$. Daraus folgt die Gleichung

$$\lambda u = \lambda(y - x) = \lambda y - \lambda x \in K_{\preceq}$$

und damit die Behauptung des Satzes. □

Andererseits lässt sich aus einem gegebenen Kegel K eine Ordnungsrelation $\preceq_K$ ableiten:

Satz 2.12 *Sei K ein Kegel und $\preceq_K$ eine Relation auf $\mathbb{R}^n$, definiert durch*

$$x \preceq_K y \Leftrightarrow y - x \in K.$$

Dann ist $\preceq_K$ kompatibel zur Addition und zur Skalarmultiplikation. Außerdem gilt:

1. *$\preceq_K$ ist reflexiv, falls $0 \in K$;*
2. *$\preceq_K$ ist transitiv, falls K konvex ist, d.h. falls $\alpha x_1 + (1 - \alpha)x_2 \in K$ für alle $x_1, x_2 \in K$ und alle $0 < \alpha < 1$ gilt;*
3. *$\preceq_K$ ist antisymmetrisch, falls $K \cap (-K) \subset \{0\}$.*

Das bedeutet, dass $\preceq_K$ eine Halbordnung ist, wenn K alle drei Bedingungen erfüllt.

Beweis 12 Seien $x, y, z \in \mathbb{R}^n$ und $\lambda > 0 \in \mathbb{R}$. Aus $x \preceq_K y$ folgt $y - x \in K$ und, da K ein Kegel ist, auch $\lambda(y - x) \in K$. Damit folgt aus der Definition von $\preceq_K$: $\lambda x \preceq_K \lambda y$ also die Kompatibilität bezüglich der Skalarmultiplikation. Die Kompatibilität bezüglich der Addition folgt, da sich aus $y - x = (y + z) - (x + z) \in K$ die Beziehung $(x + z) \preceq_K (y + z)$ ergibt.
Zu 1.: Mit $x \in \mathbb{R}$ ist $x - x = 0 \in K$ und daher gilt $x \preceq_K x$.
Zu 2.: Sei $x \preceq_K y$ und $y \preceq_K z$. Dann gilt: $y - x \in K$ und $z - y \in K$. Da K konvex ist, folgt $\frac{1}{2}(y - x + z - y) \in K$ und damit auch $(y - x + z - y) = z - x \in K$. Dies ist aber per Definition nichts anderes als $x \preceq_K z$.
Zu 3.: Mit $x, y \in \mathbb{R}^n$ folgt aus $x \preceq_K y$ und $y \preceq_K x$ nach der Definition der Relation: $y - x \in K$ und $x - y \in K$. Das bedeutet aber $y - x \in K \cap (-K) = \{0\}$ und somit $y = x$. □

Bei einem mehrdimensionalen Filter wird durch jeden Filtereintrag x_k ein verschobener Kegel $K_k = \{x \in \mathbb{R}^n : x_k < x\}$ erzeugt, in dem alle Punkte liegen, die aufgrund des Eintrags x_k nicht in den Filter aufgenommen werden können, dabei bildet jedes x_k die Spitze des von ihm erzeugten Kegels. Für den gesamten Filter bedeutet dies, dass die Menge der Punkte, die nicht akzeptiert werden, eine Vereinigung von n Kegeln ist, wobei n die Anzahl der Einträge im Filter bezeichnet.

2.3.2 Pareto-Optimalität und effiziente Menge

Bisher wurde der Begriff des multikriteriellen Optimierungsproblems nur anhand eines einleitenden Beispiels erläutert. Nun soll er allgemeiner beschrieben werden:

Definition 2.15 *Seien $n, p \in \mathbb{N}$, wobei $p > 1$, $X \subseteq \mathbb{R}^n$, $X \neq \{\}$ und $f_1, ..., f_p$ Funktionen mit $f_i : X \to \mathbb{R}$ für $i = 1, ..., p$. Außerdem sei eine Ordnungsrelation $\preceq$ auf $\mathbb{R}^p$ definiert. Dann ist*

$$\min_{x \in X} (f_1(x), ..., f_p(x))$$

ein multikriterielles Optimierungsproblem *(MCOP). Dabei bezeichnet man*

- *die Menge X als* zulässige Menge*,*
- *$(\mathbb{R}^p, \preceq)$ als* Zielfunktionsraum*,*
- *$f = (f_1(x), ..., f_p(x))$ als* Zielfunktionsvektor *bzw. f_i als* i-te Zielfunktion *und*
- *$Y := f(X) \subseteq \mathbb{R}^n$ als* Bildmenge der zulässigen Menge*.*

Betrachtet man ein solches multikriterielles Optimierungsproblem, kann man folgende Definition festhalten:

Definition 2.16 *Eine Lösung $x^* \in X$ eines (MCOP) heißt* Pareto-optimal*, wenn es kein $x \in X$ gibt, so dass $f(x) < f(x^*)$ ist. Ist x^* eine Pareto-optimale Lösung, so heißt $f(x^*)$* effizient*. Seien $x_1, x_2 \in X$, so sagt man, x_1* dominiert *x_2 und $f(x_1)$* dominiert *$f(x_2)$, wenn $f(x_1) < f(x_2)$ gilt. Die Menge aller Pareto-optimalen Lösungen $x^* \in X$ heißt* Pareto-Menge *X_{Par}. Die Menge aller effizienten Punkte $y = f(x^*) \in Y$ mit $x^* \in X_{Par}$ wird* effiziente Menge *Y_{eff} genannt.*

Wie man anhand der letzten Definition sieht, wird zur Festlegung der Pareto-optimalen Punkte die komponentenweise Ordnung "$<$" verwendet. In ähnlicher Weise werden zwei weitere Mengen "optimaler" Punkte unter Verwendung schwach bzw. stark komponentenweisen Ordnung definiert:

Definition 2.17 *Eine Lösung $x^* \in X$ eines (MCOP) heißt* schwach Pareto-optimal*, falls es kein $x \in X$ gibt, so dass $f(x) \ll f(x^*)$. Der Punkt $y^* = f(x^*)$ wird dann* schwach effizient *genannt. Eine Lösung $x^* \in X$ eines (MCOP) heißt* streng Pareto-optimal*, falls es kein $x \in X$, $x \neq x^*$ gibt, so dass $f(x) \leq f(x^*)$.*

In diesem Kapitel wurde die zu Beginn erwähnte effiziente Menge erläutert, aus deren Repräsentanten die Filtereinträge des multikriteriellen Filters bestehen. Abschließen soll dieses Kapitel ein Hinweis auf die unterschiedlich Verwendung des Begriffs der Dominanz. In der multikriteriellen Optimierung wird gesagt: Der Wert $f(x_1)$ dominiert den Wert $f(x_2)$, falls $f(x_1) < f(x_2)$ gilt, was nach Definition der komponentenweisen

Ordnung erstens $f(x_{1,i}) \leq f(x_{2,i})$ für alle $i = 1, ..., p$ und zweitens $f(x_1) \neq f(x_2)$ bedeutet. Diese zweite Bedingung wird, wenn der Begriff Dominanz im Kontext des Filters verwendet wird, aufgegeben und nur die erste zur Definition verwendet. Das bedeutet, dass jeder Filtereintrag $f(x)$ von sich selbst dominiert wird.

Kapitel 3

Der Filter-Trust-Region-Algorithmus

Im folgenden Kapitel soll der Filter-Trust-Region-Algorithmus anhand des 2004 erschienenen Artikels "A multidimensional filter algorithm for nonlinear equations and nonlinear least-squares" [12] erläutert werden. Die Autoren Nicholas I. M. Gould, Sven Leyffer und Philippe L. Toint entwerfen einen Algorithmus, um nichtlineare Gleichungssysteme und nichtlineare Ausgleichsprobleme (engl.: nonlinear least-squares problems) zu lösen, der die Effizienz der Filtermethoden und die Stabilität der Trust-Region-Verfahren vereinen soll. Für diesen Algorithmus wird mit Hilfe der Erkenntnisse über das Konvergenzverhalten von Trust-Region-Verfahren eine globale Konvergenzaussage bewiesen.

3.1 Problemstellung

Der Filter-Trust-Region Algorithmus (FTR-Algorithmus) wird verwendet, um ein nichtlineares Gleichungssystem

$$c(x) = 0 \tag{3.1}$$

zu lösen. Die linke Seite von (3.1) wird hier als mehrdimensionaler Funktionsvektor dargestellt. Dabei ist $c : \mathbb{R}^n \to \mathbb{R}^m$ eine zweimal stetig differenzierbare Funktion.
Da es sich bei dem FTR-Algorithmus um ein Optimierungsverfahren, genauer ein Minimierungsverfahren handelt, muss (3.1) in eine andere, jedoch äquivalente Problemformulierung transformiert werden, damit man den Algorithmus darauf anwenden kann. Dabei geht man folgendermaßen vor:
Zunächst unterteilt man die m Zeilen des Funktionsvektors c mit Hilfe von p Indexmengen $\mathcal{I}_j$ mit $j = 1, ..., p$. Eine Zeile darf dabei mehrmals, auch in unterschiedlichen Mengen $\mathcal{I}_j$, auftreten. Jede Zeile muss aber mindestens in einer Indexmenge stehen, da sie sonst nicht für die Berechnung der Lösung des nichtlinearen Gleichungssystems berücksichtigt wird. Formal bedeutet das: Die Zeilen von c werden in p nicht

notwendigerweise disjunkte Teilmengen $\{c_i(x)\}_{i\in\mathcal{I}_j}$ mit $j = 1, ..., p$ unterteilt, für die gilt: $\{1, ..., m\} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup ... \cup \mathcal{I}_p$. Obwohl die Indexmengen nur natürliche Zahlen beinhalten, bietet sich der Ausdruck "die Funktion j liegt in der Menge $\mathcal{I}_i$" für eine Komponente von c an, wenn $j \in \mathcal{I}_i$ gilt. Damit kann die folgende Argumentation etwas einfacher formuliert werden.
Für jede Gruppe von Funktionen wird ein Fehler

$$\theta_j(x) := \left\| c_{\mathcal{I}_j}(x) \right\| \text{ für alle } j = 1, ..., p$$

definiert, wobei $\|\cdot\|$ die euklidische Norm und $c_{\mathcal{I}_j}$ ein Funktionsvektor ist, der genau die Zeilen von c enthält, die in der Indexmenge $\mathcal{I}_j$ enthalten sind. $\theta_j(x)$ kann man demnach als Maß für die Abweichung vom optimalen Wert 0 interpretieren. Diese Maß gibt an, wie groß der Gesamtfehler aller Funktionen aus der Indexmenge $\mathcal{I}_j$ ist. Ein Punkt x^* ist somit genau dann eine Lösung von (3.1), wenn

$$\forall j \in \{1, ..., p\} : \quad \theta_j(x^*) = 0$$

gilt. Damit auch der Fehler für das gesamte Gleichungssystem angegeben werden kann, werden die Fehler der einzelnen Funktionsgruppen zu einem Vektor zusammengefasst: $\theta(x) = (\theta_1(x), \theta_2(x), ..., \theta_p(x))$. Außerdem werden die abkürzenden Schreibweisen $\theta_k := \theta(x_k)$ und $\theta_{j,k} := \theta_j(x_k)$ eingeführt.
Die Idee ist nun, eine Funktion zu modellieren, die von $\theta(x)$ abhängt und die ihr Minimum an der selben Stelle annimmt, an der $\theta(x) = 0$ ist. Verwendet man diese Funktion als Zielfunktion in einem Minimierungsalgorithmus, so erhält man mit der Lösung des Minimierungsproblems auch die Lösung des Gleichungssystems (3.1). Eine Funktion, die diese Anforderungen erfüllt und die im FTR-Algorithmus verwendet wird ist $f(x) = \frac{1}{2}\|\theta(x)\|^2$. Das Problem

$$\min_{x\in\mathbb{R}} f(x) = \frac{1}{2}\|\theta(x)\|^2 \tag{3.2}$$

ist das zu (3.1) äquivalente nichtlineare Ausgleichsproblem. Falls die Indexmengen $\mathcal{I}_j$ nicht disjunkt sind, die einzelnen Gleichungen des $c(x) = 0$ aber gleichwertig behandelt werden sollen, kann man die Modellierung dahingehend erweitern, dass man an die ensprechenden Komponenten des Vektors $\theta(x)$ ganzzahlige Gewichte multipliziert, um so das mehrfache Auftreten einer oder mehrerer Funktionen in den Indexmengen zu kompensieren.
Zusammenfassend bedeutet dies: Das Ziel des weiteren Vorgehens wird es sein, ein lokales Minimum x_* von $f(x)$ zu bestimmen. Denn ist zusätzlich $f(x_*) = 0$, so hat man damit auch eine Lösung des nichtlinearen Gleichungssystems (3.1) gefunden.

3.2 Theoretische Grundlagen

Effiziente Möglichkeiten (3.1) zu lösen erhält man beispielsweise durch die Verwendung des Newton-Verfahrens [19] oder des Gauß-Newton-Verfahrens [1]. Beide berechnen

in Iteration k ausgehend von einem Iterationspunkt x_k eine Schrittweite s_k und mit deren Hilfe einen Testpunkt $x_k^+ = x_k + s_k$. Von diesem erwartet man, dass er einen besseren, das heißt kleineren Funktionswert als der vorherige Iterationspunkt x_k erzielt.
Der Nachteil dieser beiden Verfahren ist, dass sie nicht global konvergent sind, also nicht von einem beliebigen Startpunkt konvergieren. Im Gegensatz dazu kann für den hier vorgestellten Filter-Trust-Region-Algorithmus die globale Konvergenz gegen ein lokales Minimum gezeigt werden.
Die Hauptidee der Filter-Algorithmen ist es, dass neue Iterationspunkte des zugrunde liegenden Iterationsalgorithmus vom Filter akzeptiert werden können, wenn sie nicht in allen Kriterien schlechter sind als ein bereits im Filter vorhandener Punkt. Beim gewöhnlichen zweidimensionalen Filter gibt es die normalerweise widersprüchlichen Ziele einen möglichst kleinen Wert in der Zielfunktion und eine möglichst geringe Verletzung der Nebenbedingungen zu erreichen; sie bilden die Kriterien für die Akzeptanz eines Testpunktes.
Dieses Vorgehen lässt sich auch auf den hier vorliegenden Kontext der nichtlinearen Gleichungssysteme übertragen, da man das Problem (3.1) als Kombination einzelner Teilprobleme betrachten kann. Dazu minimiert man jede Komponenten von c, also $\{c_i(x)\}_{i=1}^m$ beziehungsweise jeden Fehler $\{\theta_i(x)\}_{i=1}^p$ einer Gruppe von Funktionen separat. Dies sind im Allgemeinen m beziehungsweise p verschiedene, normalerweise widersprüchliche Ziele. Dies führt zur Einführung eines mehrdimensionalen Filters, da für eine beliebige Anzahl von Gleichungen zwei Kriterien nicht ausreichen.

3.2.1 Der mehrdimensionale Filter

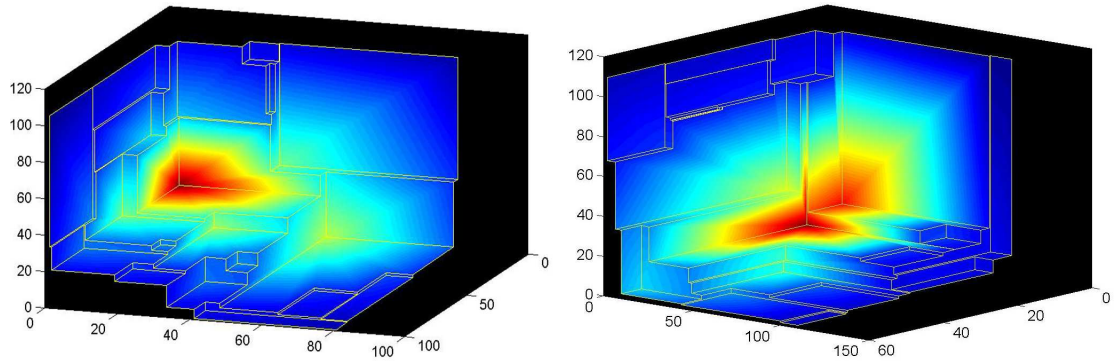


Abbildung 3.1: Zwei Beispiele für mehrdimensionale Filter bestehend aus dreidimensionalen Punkten. Die Bereiche der Filter, die nahe am Nullpunkt liegen, sind rot gefärbt. Je weiter vom Nullpunkt entfernt sie liegen, desto blauer wird die Färbung. Zum Bestimmen der Entfernung wurde die euklidische Norm verwendet.

Um den mehrdimensionalen Filter zu beschreiben, benötigt man zunächst, ähnlich wie beim zweidimensionalen Filter, den Begriff der Dominanz. Dieser wird für den

mehrdimensionalen Filter folgendermaßen definiert:

Definition 3.1 *Man sagt, ein Punkt* x_1 dominiert *einen Punkt* x_2 *genau dann, wenn*

$$\theta_j(x_1) \leq \theta_j(x_2) \quad \forall j = 1, ..., p.$$

Da unter Verwendung der Bezeichnungen und Notationen aus Definition 3.1 der Punkt x_2 für keine Gruppe von Funktionen eine geringere Abweichung vom gewünschten Ergebnis $\|\theta(x_k)\| = 0$ als x_1 erzielt, ist er für die Ermittlung einer Minimalstelle der Funktion $f(x)$ nicht von Interesse. Alle relevanten Punkte bzw. deren Fehlervektor θ werden dagegen im Filter gespeichert. Dabei ist der mehrdimensionale Filter ähnlich wie im zweidimensionalen Fall definiert:

Definition 3.2 *Ein* Filter $\mathcal{F}$ *ist eine Liste von p-Tupeln der Form* $(\theta_{1,k}, ..., \theta_{p,k})$ *so, dass es für alle* $k \neq l$ *mindestens ein* $j \in \{1, ..., p\}$ *gibt, für das gilt:* $\theta_{j,k} < \theta_{j,l}$.

Es gibt unterschiedliche Möglichkeiten, die Akzeptanzbedingung für einen Filter zu formulieren. Die naheliegendste ist folgende: Ein neuer Iterationswert x_k^+ wird vom Filter akzeptiert, wenn er nach Definition 3.1 von keinem Punkt, der bereits im Filter enthalten ist, dominiert wird. Im Sinne der multikriteriellen Optimierung bedeutet das: der Filter enthält Repräsentanten der effizienten Menge des $p-$kriterielle Problems durch welches die Unzulässigkeit in allen p Gruppen von Funktionen minimiert werden soll.
Es wird sich aber herausstellen, dass es für den Konvergenzbeweis des Algorithmus nötig ist, eine strengere Akzeptanzbedingung aufzustellen. Ein Testpunkt x_k^+ wird nur dann akzeptiert, wenn der zugehörige Fehlervektor $\theta_k^+ := \theta\left(x_k^+\right)$ deutlich von keinem anderen Filtereintrag dominiert wird. Dies führt zu folgender Definition:

Definition 3.3 *Ein Punkt* x_k^+ *ist* akzeptabel *für einen Filter* $\mathcal{F}$ *genau dann, wenn*

$$\forall \theta_l \in \mathcal{F} \quad \exists j \in \{1, ..., p\} \quad \theta_j\left(x_k^+\right) < \theta_{j,l} - \gamma_\theta \delta\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right), \tag{3.3}$$

wobei $\gamma_\theta \in \left(0, \frac{1}{\sqrt{p}}\right)$ *eine kleine positive Konstante ist und* $\delta(\cdot, \cdot)$ *einer der folgenden Ausdrücke:*

$$\delta\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right) = \|\theta_l\|, \ \textit{oder} \tag{3.4}$$
$$\delta\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right) = \left\|\theta_k^+\right\|, \ \textit{oder} \tag{3.5}$$
$$\delta\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right) = \min\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right). \tag{3.6}$$

Die drei alternativen Ausdrücke (3.4), (3.5) und (3.6) beinhalten jeweils ein Maß für den Gesamtfehler θ. Dabei ist bei Verwendung von (3.4) die Mindestverbesserung, die ein neuer Punkt erreichen muss, um in den Filter aufgenommen zu werden, abhängig vom Gesamtfehler der im Filter enthaltenen Punkte. Wird die zweite Alternative (3.5) verwendet, geht der Gesamtfehler des Testpunktes in die Berechnung mit ein. (3.6)

ist die schwächste der drei Formulierungen, da durch sie immer der kleinere Wert der beiden Normen verwendet wird. Ein sinnvoller Aspekt dieser Modellierung ist es, dass ein Testpunkt eine immer geringere Verbesserung verursachen muss, um vom Filter akzeptiert zu werden, je näher man sich bereits am Nullpunkt befindet.

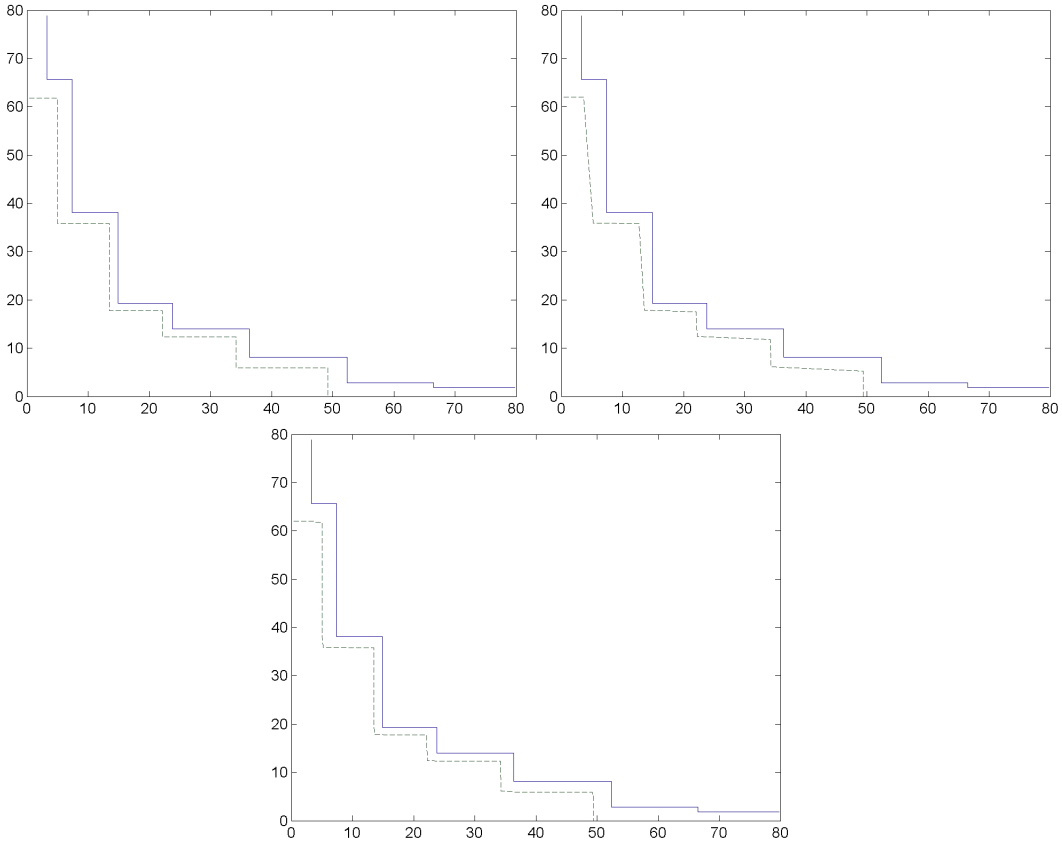


Abbildung 3.2: Die Abbildungen zeigen, wie sich der Korridor der Mindestverbesserung um den Filter verändert, je nachden welche der Normen (3.4), (3.5) oder (3.6) in der Akzeptanzbedingung verwendet wird. Im Bild links oben wurde die Norm der Punkte im Filter verwendet, im Bild rechts daneben die Norm des Testpunktes und im Bild darunter das Minimum der beiden Normen.

Die Akzeptanzbedingung (3.3) ist so formuliert, dass immer Punkte existieren, die sie erfüllen, vorausgesetzt, es wurde noch kein x_* mit $f(x_*) = 0$, das heißt noch keine Lösung des Problems (3.2) gefunden. Das folgt bei Verwendung von (3.4) oder (3.6) aus der Wahl von γ_θ. Da der Definitionsbereich dieser Konstante so festgelegt wurde, dass die rechte Seite von (3.3) für jeden Punkt im Filter und für mindestens ein j

positiv ist. Dies sieht man, da mit $\theta_{j,l} \geq \theta_{i,l} \quad \forall i \in \{1, \ldots, p\}$ gilt:

$$\begin{aligned} \theta_{j,l} - \underbrace{\gamma_\theta}_{<\frac{1}{\sqrt{p}}} \underbrace{\delta\left(\|\theta_l\|, \left\|\theta_k^+\right\|\right)}_{\leq\|\theta_l\|} &> \theta_{j,l} - \frac{\|\theta_l\|}{\sqrt{p}} = \theta_{j,l} - \sqrt{\frac{\theta_{1,l}^2 + \ldots + \theta_{p,l}^2}{p}} \\ &\geq \theta_{j,l} - \sqrt{\frac{\theta_{j,l}^2 + \ldots + \theta_{j,l}^2}{p}} = \theta_{j,l} - \sqrt{\frac{p \cdot \theta_{j,l}^2}{p}} = 0. \end{aligned}$$

Verwendet man (3.5), so ist ebenfalls die Existenz von akzeptablen Punkten gesichert. Dies folgt, da vorausgesetzt wurde, dass für alle Filtereinträge $\|\theta_l\| > 0$ ist. Denn daher muss es ein $j \in 1, ..., p$ geben, für das $\theta_{j,l}$ echt positiv ist. Das bedeutet, jeder Punkt, der die Ungleichung

$$\gamma_\theta \left\|\theta_k^+\right\| + \theta_j\left(x_k^+\right) < \theta_{j,l} \tag{3.7}$$

erfüllt, wird vom Filter akzeptiert. Um zu zeigen, dass es tatsächlich Punkte gibt, die diese Ungleichung erfüllen, genügt es ein Beispiel anzugeben: Sei $\theta_i\left(x_k^+\right) \ll \theta_j\left(x_k^+\right)$ für alle $i \in 1, ..., p$, $i \neq j$. Dann folgt aus (3.7):

$$\begin{aligned} & \gamma_\theta \theta_j\left(x_k^+\right) + \theta_j\left(x_k^+\right) < \theta_{j,l} \\ \Rightarrow \quad & \left(\gamma_\theta + 1\right)\theta_j\left(x_k^+\right) < \theta_{j,l}. \end{aligned}$$

Also erfüllt ein Punkt, der in der j-ten Gruppe von Funktionen einen Fehler verursacht, der kleiner als $\frac{\theta_{j,l}}{\gamma_\theta+1}$ ist und der gleichzeitig in allen anderen Gruppen von Funktionen einen noch sehr viel geringeren Fehler verursacht, die Ungleichung (3.7) und wird daher vom Filter akzeptiert.
Unabhängig davon, welche Norm in der Akzeptanzbedingung verwendet wird, ist auch die Wahl von γ_θ von großer Bedeutung. Die Wahrscheinlichkeit, dass ein Punkt vom Filter akzeptiert wird steigt, je kleiner der Parameter γ_θ gewählt wurde. In [12] wird der Wert $\gamma_\theta = \min(0.001, \frac{1}{2\sqrt{p}})$ verwendet.

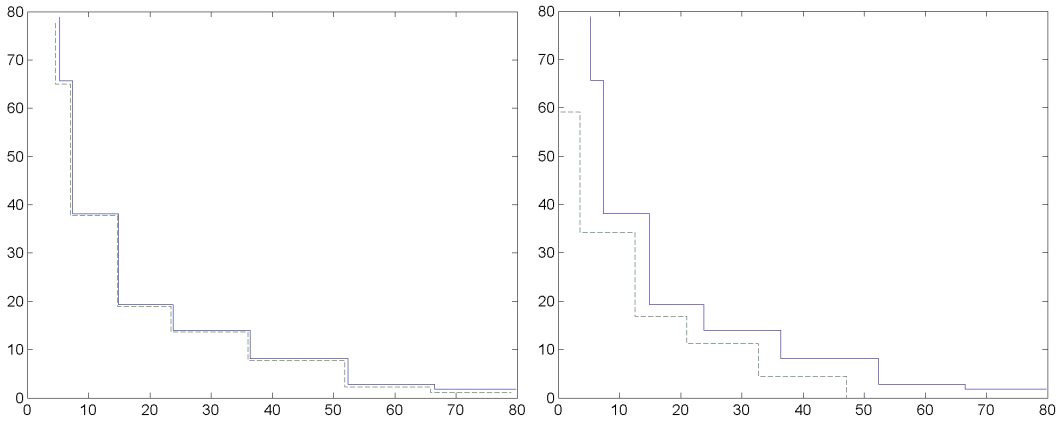


Abbildung 3.3: Beispiel für den Einfluss des Parameters γ_θ auf die benötigte Mindestverbesserung. Im linken Bild wurde $\gamma_\theta = 0.01$ gewählt und im rechten $\gamma_\theta = 0.1$.

3.2.2 Veränderung der Filtereinträge

Unter der Voraussetzung, dass ein Testpunkt x_k^+ im Sinne von (3.3) akzeptiert werden kann, ergänzt man den Filter um den neuen Punkt. Das ist notwendig, um Iterationsschritte zu vermeiden, die von x_k^+ dominierte Punkte liefern würden. Diese tragen, wie oben gesehen, nichts zur Lösung des Problems bei. Ein solches Vorgehen kann allerdings dazu führen, dass ein bereits im Filter gespeicherter Wert θ_l von dem neuen Eintrag θ_k^+ dominiert wird. Ein solches θ_l muss aus dem Filter entfernt werden. In Abhängigkeit von der in der Akzeptanzbedingung (3.3) verwendeten Norm wird der Eintrag θ_l aus dem Filter entfernt, wenn die jeweils zutreffende der folgenden drei Bedingungen erfüllt ist.
Bei Verwendung der Norm (3.4) ist

$$\exists \theta_q \in \mathcal{F} \quad \forall j \in 1, ..., p: \quad \theta_{j,l} - \gamma_\theta \|\theta_l\| \geq \theta_{j,q} - \gamma_\theta \|\theta_q\|$$

die Bedingung für die Entfernung des Eintrags θ_l.

$$\exists \theta_q \in \mathcal{F} \quad \forall j \in 1, ..., p: \quad \theta_{j,l} \geq \theta_{j,q}$$

gilt als Bedingung, falls (3.5) verwendet wird. In diesen beiden Fällen folgen die jeweiligen Ungleichungen sofort aus (3.3), wenn man die entsprechende Norm einsetzt. Im dritten Fall, also bei Verwendung des Minimums der beiden Normen (3.6), ergibt sich die Bedingung

$$\exists \theta_q \in \mathcal{F} \quad \forall j \in 1, ..., p: \quad \theta_{j,l} - \gamma_\theta \|\theta_l\| \geq \theta_{j,q}.$$

Wenn in diesem Fall ein Eintrag θ_l aus dem Filter entfernt werden muss, ist diese Ungleichung notwendiger Weise erfüllt. Das folgt, da damit auch die Ungleichheitskette

$$\theta_{j,k}^+ \leq \theta_{j,q} - \gamma_\theta \min\left(\left\|\theta_k^+\right\|, \|\theta_q\|\right) \leq \theta_{j,q} \leq \theta_{j,l} - \gamma_\theta \|\theta_l\| \leq \theta_{j,l} - \min\left(\left\|\theta_k^+\right\|, \|\theta_l\|\right)$$

gültig ist und diese genau die Situation beschreibt, in der θ_l entfernt werden muss.

3.2.3 Berechnung eines Testpunktes

Unabhängig vom Filter muss im zugrundeliegenden iterativen Trust-Region-Verfahren in jeder Iteration ein Testschritt s_k und ein sich daraus ergebender Testpunkt $x_k^+ = x_k + s_k$ erzeugt werden. Dazu sei $m_k(x)$ eine Modellfunktion von $f(x)$ und $\mathcal{B}_k = \{x_k + s : \|s\|_k \leq \Delta_k\}$ die Trust-Region, innerhalb der man annimmt, dass das Modell hinreichend genau ist. Die verwendeten Normen $\|\cdot\|_k$ müssen nicht in jedem Iterationsschritt identisch sein, es ist lediglich nötig, dass sie gleichmäßig äquivalent zur euklidischen Norm sind.
Für den Konvergenzbeweis ist es nötig, das in jeder Iteration k der Testschritt s_k eine ausreichende Verbesserung des Wertes der Modellfunktion bewirkt. Formal bedeutet dies:

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{mdc} \|g_k\| \, min\left(\frac{\|g_k\|}{\beta_k}, \Delta_k\right), \tag{3.8}$$

wobei $g_k = \nabla m_k(x_k)$, β_k eine positive obere Schranke der Norm der Hessematrix von m_k und κ_{mdc} eine Konstante aus dem Intervall $(0,1)$ ist. Diese Ungleichung wurde bereits für den Basic-Trust-Region-Algorithmus für beliebige Modellfunktionen hergeleitet und entsprach in Kapitel 2 der Voraussetzung AA 1.
Im Filter-Trust-Region-Algorithmus kann im Gegensatz zu klassischen Trust-Region-Verfahren, die Bedingung

$$\|s_k\|_k \leq \Delta_k \tag{3.9}$$

aufgegeben werden, wenn man annimmt, dass dadurch ein besserer Testpunkt erzeugt werden kann. Gilt für den Testschritt $\|s_k\|_k > \Delta_k$, so bleibt die Ungleichung (3.8) natürlich erfüllt, da die Verbesserung des Wertes der Zielfunktion höchstens größer werden kann, als dies der Fall ist, wenn (3.9) berücksichtigt werden muss. Denn die Suchrichtung wird nicht davon beeinflusst, wie groß die Schrittweite sein darf. Man verändert demnach nicht das prinzipielle Verfahren einen Testpunkt zu finden, sondern erhöht nur die Anzahl der in Frage kommenden Punkte.
Ein zweiter Aspekt, der sich aus dem Verzicht auf die Bedingung (3.9) ergibt, ist folgender: Da man die Trust-Region nicht beachten muss, kann man in einem Schritt zur Nullstelle der Modellfunktion gelangen. Ist die Modellfunktion beispielsweise eine lineare Approximation des Funktionsvektors $c(x)$, die mit Hilfe der Taylorentwicklung im Punkt x_k erzeugt wurde, so gelangt man mit Hilfe der Schrittweite $s_k = -J_k^{-1} c(x_k)$ direkt zu ihrer Nullstelle. Dabei ist J_k die Jacobimatrix des Funktionsvektors $c(x)$ in x_k.
Eine solche Modellfunktion, die durch lineare Approximation des Funktionsvektors $c(x)$ entsteht nennt man Gauss-Newton-Modell

$$m_k^{GN}(x_k + s) = \frac{1}{2} \sum_{i=1}^{p} \|c_{\mathcal{I}_i}(x_k) + J_{\mathcal{I}_i}(x_k)\, s\|^2,$$

der Zielfunktion $f(x)$. Dabei ist $f(x)$ wie in (3.2) definiert und es gilt:

$$f(x) = \frac{1}{2}\|\theta(x)\|^2 = \frac{1}{2}\sum_{i=1}^{p} (\theta_i(x))^2 = \frac{1}{2}\sum_{i=1}^{p} \|c_{\mathcal{I}_i}(x)\|^2.$$

Bestimmt man die Taylorentwicklung erster Ordnung des Funktionsvektors c in der umgeformten Zielfunktion, erhält man genau das Gauss-Newton-Modell m_k^{GN}.
Ein weiteres Beispiel einer Modellfunktion ist das Full-Newton-Modell

$$m_k^{N}(x_k + s) = m_k^{GN}(x_k + s) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i} c_j(x_k) \left\langle s, \nabla^2 c_j(x_k)\, s \right\rangle.$$

Dieses Modell entsteht auf die selbe Art wie das Gauss-Newton-Modell, nur wird bei

der Taylorentwicklung zusätzlich der Term der zweiten Ableitung berücksichtigt. Mit den Bezeichnungen $g_t(x)$ für den Gradienten und $H_t(x)$ für die Hessematrix einer Funktion t an der Stelle x und einer weiteren Umformulierung der Zielfunktion

$$f(x) = \frac{1}{2}\sum_{i=1}^{p} \|c_{\mathcal{I}_i}(x)\|^2 = \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i} (c_j(x))^2 \tag{3.10}$$

ergibt sich das Full-Newton-Modell aus folgender Umformung:

$$\begin{aligned}
m_k^N(x_k+s) &= \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(c_j(x_k) + g_{c_j}(x_k)\, s_k^\top + \frac{1}{2}s_k H_{c_j}(x_k)\, s_k^\top\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\Bigg((c_j(x_k))^2 + \left(g_{c_j}(x_k)\, s_k^\top\right)^2 + 2c_j(x_k)\, g_{c_j}(x_k)\, s_k^\top \\
&\quad + \left(\frac{1}{2}s_k H_{c_j}(x_k)\, s_k^\top\right)^2 + c_j(x_k)\, s_k H_{c_j}(x_k)\, s_k^\top \\
&\quad + g_{c_j}(x_k)\, s_k^\top s_k H_{c_j}(x_k)\, s_k^\top\Bigg) \\
&= m_k^{GN}(x_k+s) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\Bigg(\left(\frac{1}{2}s_k H_{c_j}(x_k)\, s_k^\top\right)^2 \\
&\quad + c_j(x_k)\, s_k H_{c_j}(x_k)\, s_k^\top + g_{c_j}(x_k)\, s_k^\top s_k H_{c_j}(x_k)\, s_k^\top\Bigg) \\
&= m_k^{GN}(x_k+s) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\Bigg(\underbrace{\frac{1}{4}s_k H_{c_j}(x_k)\, s_k^\top + g_{c_j}(x_k)\, s_k^\top}_{\longrightarrow 0 \text{ für } s_k\to 0} + c_j(x_k)\Bigg) \\
&\quad \cdot s_k H_{c_j}(x_k)\, s_k^\top \\
&= m_k^{GN}(x_k+s) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i} c_j(x_k)\left\langle s, \nabla^2 c_j(x_k)\, s\right\rangle
\end{aligned}$$

wobei alle Vektoren als Zeilenvektoren aufzufassen sind.
Diese beiden Modellfunktionen basieren auf Approximationen des Funktionsvektors $c(x)$. Durch die lineare bzw. quadratische Approximation der Zielfunktion $f(x)$ kann man weitere Modellfunktionen erhalten. Allerdings haben die numerischen Tests ergeben, dass für die Mehrzahl der Beispielprobleme das Gauss-Newton- und das Full-Newton-Modell die deutlich besseren Resultate liefern.

3.3 Der Algorithmus

Der folgende Algorithmus basiert auf den bisher in diesem Kapitel angestellten Überlegungen:

Algorithmus 3.1 *Filter-Trust-Region-Algorithmus*

Schritt 1: Initialisierung
Ein Startpunkt x_0 und ein anfänglicher Trust-Region-Radius $\Delta_0 > 0$ sind gegeben, ebenso die Konstanten $0 < \gamma_0 \leq \gamma_1 < 1 \leq \gamma_2$, $\gamma_\theta \in \left(0, \frac{1}{\sqrt{p}}\right)$ und $0 < \eta_1 < \eta_2 < 1$ sowie die Indexmengen $\{\mathcal{I}_j\}_{j=1}^{p}$.
Berechne $c_0 = c(x_0)$ und θ_0.
Setze $k = 0$, initialisiere einen boolschen Wert RESTRICT mit "false" und den Filter als leere Menge.
Schritt 2: Test auf Optimalität
Falls $\theta_k = 0$ oder $\|\nabla f(x_k)\| = 0$, STOP
Schritt 3: Berechnung des Testschritts
Bestimme einen Schritt s_k der (3.8) erfüllt.
Ist RESTRICT="true" muss s_k ebenfalls (3.9) erfüllen.
Berechne den Testpunkt $x_k^+ = x_k + s_k$.
Schritt 4: Berechnung der Abweichung im Testschritt
Berechne $c\left(x_k^+\right)$ und $\theta_k^+ = \theta\left(x_k^+\right)$. Bestimme $\rho_k = \frac{f(x_k) - f\left(x_k^+\right)}{m_k(x_k) - m_k\left(x_k^+\right)}$.
Schritt 5: Akzeptanztest
Falls x_k^+ vom Filter akzeptiert wird:
Setze $x_{k+1} = x_k^+$, setze RESTRICT="false" und füge θ_k^+ zum Filter hinzu, wenn entweder $\rho_k < \eta_1$ gilt oder (3.9) nicht erfüllt ist.
Falls x_k^+ nicht vom Filter akzeptiert wird:
Ist (3.9) erfüllt und gilt $\rho_k \geq \eta_1$, setze $x_{k+1} = x_k^+$ und RESTRICT="false", anderenfalls setze $x_{k+1} = x_k$ und RESTRICT="true".
Schritt 6: Aktualisierung des Trust-Region-Radius
Ist $\|s_k\|_k \leq \Delta_k$, wird der Trust-Region-Radius folgendermaßen aktualisiert:

$$\Delta_{k+1} \in \begin{cases} [\gamma_0 \Delta_k, \gamma_1 \Delta_k] & falls \quad \rho_k < \eta_1 \\ [\gamma_1 \Delta_k, \Delta_k] & falls \quad \rho_k \in [\eta_1, \eta_2) \\ [\Delta_k, \gamma_2 \Delta_k] & falls \quad \rho_k \geq \eta_2. \end{cases}$$

Anderenfalls setze $\Delta_{k+1} = \Delta_k$, $k = k + 1$ und gehe zu Schritt 2.

Anhand der Variable *RESTRICT* wird bestimmt, ob die Bedingung (3.9) für den Testschritt erfüllt sein muss. Man verzichtet auf diese Einschränkung, wenn man aufgrund eines guten Iterationsschrittes, d.h. $\rho \geq \eta_1$ oder x_k^* wird vom Filter akzeptiert, davon ausgehen kann, dass der Algorithmus schneller gegen ein lokales Minimum der Zielfunktion konvergiert, wenn die Umgebung, in der der nächste Iterationspunkt gesucht wird, vergrößert wird. Dies entspricht den Überlegungen, die bei der Aktualisierung des Trust-Region-Radius in gewöhnlichen Trust-Region-Verfahren und in Schritt 6 von Algorithmus 3.1 zur Vergrößerung des Trust-Region-Radius führen. Im Unterschied dazu wird durch die Entscheidung (3.9) nicht zu beachten, Δ_{k+1} nicht auf einen reellen Wert gesetzt, sondern das Vorgehen entspricht der Zuordnung $\Delta_{k+1} = \infty$. Der Vorteil dabei ist, dass der Algorithmus eventuell weniger Iterationsschritte benötigt

um gegen ein lokales Minimum zu konvergieren, falls die Modellfunktion, welche die Zielfunktion lokal gut approximiert, auch global eine gute Näherung darstellt. Ist die Modellfunktion aber lediglich in einer kleinen Umgebung um den aktuellen Iterationspunkt eine gute Approximation der Zielfunktion, so kann es geschehen, dass alle Iterationen in denen die Beschränkung (3.9) aufgehoben wurde, nicht erfolgreich sind, was zu einer großen Anzahl an unnötigen Iterationen führen kann. Dieser negative Effekt trat bei den numerischen Test zu diesem Algorithmus allerdings nur sehr vereinzelt auf.
Es ist, vor allem im Hinblick auf die folgende Konvergenzanalyse, zu beachten, dass die Anzahl der Punkte, die für den Filter akzeptabel sind, nicht mit der Anzahl der Punkte, die tatsächlich in den Filter aufgenommen werden, übereinstimmen muss.

3.4 Konvergenzanalyse

Um die Konvergenzeigenschaften des Filter-Trust-Region-Algorithmus nachzuweisen, werden zunächst, ähnlich wie bei der Konvergenzanalyse des Basic-Trust-Region-Algorithmus, folgende Annahmen aufgestellt:

A1: $c(x)$ ist zweimal stetig differenzierbar auf $\mathbb{R}^n$.
A2: Die Iterationswerte x_k liegen alle in einem offenem beschränktem Gebiet innerhalb des $\mathbb{R}^n$.
A3: $m_k(x)$ ist für alle k zweimal stetig differenzierbar auf $\mathbb{R}^n$.
A4: Für alle k gilt: $m_k(x_k) = f(x_k)$ und $g_k = \nabla m_k(x_k) = \nabla f(x_k)$.

Zum Zweck der Konvergenzanalyse gelte weiterhin, dass der Algorithmus nicht terminiert und demnach keines der beiden Kriterien des Optimalitätstests in Schritt 2 des FTR-Algorithmus für ein x_k aus der Folge der Iterationspunkte erfüllt ist.
Aus A1, A2 und A3 folgt, dass es eine Konstante $\kappa_u > 0$ gibt, so dass die Ungleichungen

$$\|c(x)\| \leq \kappa_u, \qquad \left\|\nabla^2 c_i(x)\right\| \leq \kappa_u \qquad \text{und} \qquad \left\|\nabla^2 m_k(x)\right\| \leq \kappa_u \tag{3.11}$$

für alle k und für alle x aus der konvexen Hülle der Menge der Iterationswerte x_k erfüllt sind.
Genauer gesagt folgt die Existenz einer oberen Schranke κ_1 für die Norm der Funktionswerte von $c(x)$, da c eine stetige Funktion mit einem beschränkten Gebiet als Definitionsmenge ist. Die Existenz der oberen Schranken κ_2 für die Norm der zweiten Ableitungen der Funktionen $c_i(x)$ und κ_3 für die zweite Ableitung der Modellfunktion $m_k(x)$ lässt sich mit Hilfe der selben Argumente folgern: Es handelt sich bei den Funktionen $\nabla^2 c_i(x)$ und $\nabla^2 m_k(x)$ um stetige Funktionen, die auf einem beschränkten Gebiet definiert sind. Wählt man $\kappa_u = \max(\kappa_1, \kappa_2, \kappa_3)$ so ergibt sich die Aussage (3.11).
Aus den selben Gründen sind auch die Normen der Funktionen $c_i(x)$ und $\nabla c_i(x)$ nach

oben beschränkt. Damit kann man zusammen mit der zweiten Ungleichung aus (3.11) und der Umformung der Zielfunktion (3.10) folgern:

$$\begin{aligned}
\left\|\nabla^2 f(x)\right\| &= \left\|\nabla^2\left(\frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(c_j(x)\right)^2\right)\right\| = \left\|\frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\nabla^2\left(\left(c_j(x)\right)^2\right)\right\| \\
&= \left\|\frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\nabla\left(2c_j(x)\cdot\nabla c_j(x)\right)\right\| \\
&= \left\|\frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(2\nabla c_j(x)\cdot\nabla c_j(x)+2c_j(x)\cdot\nabla^2 c_j(x)\right)\right\| \\
&\leq \sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left\|\left(\nabla\left(c_j(x)\right)\right)\right\| + \sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(\left\|c_j(x)\right\|\cdot\left\|\nabla^2\left(c_j(x)\right)\right\|\right) \leq \kappa_4
\end{aligned}$$

Die letzte Ungleichung folgt für eine Konstante κ_4, da $\|\nabla^2 f(x)\|$ so umgeformt wurde, dass es nur noch aus beschränkten Termen besteht. Definiert man nun $\kappa_u = \max(\kappa_1, \kappa_2, \kappa_3, \kappa_4)$, so gilt zusätzlich zu den Abschätzungen aus (3.11) auch $\|\nabla^2 f(x)\| \leq \kappa_u$.
Mit Hilfe der Annahmen A1 - A4 und den daraus gefolgerten Abschätzungen können Aussagen über das Konvergenzverhalten des FTR-Algorithmus hergeleitet werden. Diese sind in den folgenden Sätzen zusammengestellt:
Dazu muss man eine Fallunterscheidung bezüglich der Anzahl der Iterationspunkte die in den Filter aufgenommen werden, erstellen. Zunächst betrachtet man den Fall, dass unendlich viele Werte in den Filter des FTR-Algorithmus hinzugefügt werden:

Satz 3.1 *Es gelten die Annahmen A1 und A2. Werden dann unendlich viele Werte θ_k zum Filter des FTR-Algorithmus hinzugefügt, so gilt:*

$$\lim_{k\to\infty}\|c_k\| = \lim_{k\to\infty}\|\nabla f(x_k)\| = 0 \tag{3.12}$$

Beweis 13 Seien $\{k_i\}$ die Indizes der Teilfolge der Iterationsschritte, bei denen der Fehlervektor $\theta_{k_i} = \theta^+_{k_i-1}$ des Testpunktes in den Filter aufgenommen wird.
Angenommen, es würde eine Teilfolge $\{k_j\} \subseteq \{k_i\}$ derart existieren, dass $\left\|\theta_{k_j}\right\| \geq \epsilon_1$ für ein $\epsilon_1 > 0$.
Dann wäre $\left\{\left\|\theta_{k_j}\right\|\right\}$ nach oben und unten beschränkt und es müsste eine weitere Teilfolge $\{k_l\} \subseteq \{k_j\}$ geben, so dass

$$\lim_{l\to\infty}\theta_{k_l} = \theta_\infty \qquad \text{mit} \qquad \|\theta_\infty\| \geq \epsilon_1 \tag{3.13}$$

Darüber hinaus folgt aus der Definition der Folgen, insbesondere der von $\{k_l\}$, dass θ_{k_l} für jeden Wert der Variable l vom Filter akzeptiert wird. Das bedeutet, dass wegen (3.3) für jedes l ein $j \in \{1, ..., p\}$ existiert mit

$$\theta_{j,k_l} - \theta_{j,k_{l-1}} < -\gamma_\theta\delta\left(\left\|\theta_{k_{l-1}}\right\|, \|\theta_{k_l}\|\right). \tag{3.14}$$

Diese Folgerung hängt nicht davon ab, ob der Eintrag $\theta_{k_{l-1}}$ sich noch im Filter befindet oder aufgrund des Eintrags θ_{k_l} daraus entfernt wurde. Jedoch mit (3.13) und den drei alternativen Definitionen für $\delta = \left(\left\|\theta_{k_{l-1}}\right\|, \|\theta_{k_l}\|\right)$ (vgl. (3.4), (3.5) und (3.6)) muss es ein $\epsilon_2 > 0$ geben mit

$$\delta\left(\left\|\theta_{k_{l-1}}\right\|, \|\theta_{k_l}\|\right) \geq \epsilon_2$$

für alle hinreichend großen l. Deshalb kann mit (3.14) gefolgert werden, dass

$$\theta_{j,k_l} - \theta_{j,k_{l-1}} < -\gamma_\theta \epsilon_2$$

für hinreichend große l gilt. Aus (3.13) folgt jedoch, dass die linke Seite dieser Ungleichung gegen null konvergiert, wenn l gegen unendlich geht. Damit ergibt sich ein Widerspruch, da auf beiden Seiten der Ungleichung negative Zahlen stehen und die Konstante auf der rechten Seite immer größer sein soll als der Wert auf der linken Seite. Die zu Beginn des Beweises gemachte Annahme muss daher falsch sein und somit gilt:

$$\lim_{i \to \infty} \|\theta_{k_i}\| = 0. \tag{3.15}$$

Nun wird ein beliebiges $l \notin \{k_i\}$ betrachtet. Dafür sei $k_{i(l)}$ der letzte Iterationsschritt vor dem l-ten für den $\theta_{k_{l(i)}}$ in den Filter aufgenommen wird. Jeder erfolgreiche Iterationsschritt, für den der Fehlervektor θ des zugehörigen Testpunktes nicht in den Filter aufgenommen wird, führt zu einer Reduzierung des Zielfunktionswertes, da $\rho_k \geq \eta_1$ bei einem solchen Iterationsschritt gelten muss. Dies folgt aus Schritt 5 des FTR-Algorithmus. Daher kann man für alle $l \notin \{k_i\}$ schließen, dass

$$f(x_l) \leq f\left(x_{k_{i(l)}}\right)$$

gilt. Dies ist gleichbedeutend mit

$$\|\theta(x_l)\| \leq \left\|\theta\left(x_{k_{i(l)}}\right)\right\|.$$

Zusammen mit (3.15) und der Tatsache, dass $k_{i(l)} \in \{k_i\}$, erhält man die Aussage

$$\lim_{k \to \infty} \|\theta_k\| = 0,$$

woraus die Behauptung des Satzes (3.12) folgt. □

Nun wird der Fall betrachtet, in dem nur endlich viele Einträge in den Filter aufgenommen werden und die Anzahl der Iterationspunkte, die eigentlich für den Filter akzeptabel sind, ebenfalls endlich ist. Bei nur endlich vielen akzeptablen Punkten muss es einen Iterationsschritt k_0 geben, für den gilt, dass keiner der Fehlervektoren von späteren Iterationspunkten x_k mit $k \geq k_0$ vom Filter akzeptiert wird. Das heißt, dass für keinen dieser Punkte x_k die Akzeptanzbedingung des Filters aus Schritt 5 des FTR-Algorithmus erfüllt ist.
Das bedeutet, dass ab diesem Iterationsschritt k_0 der FTR-Algorithmus identisch mit dem normalen Trust-Region-Verfahren ist. In diesem Fall werden, ebenfalls nach

Schritt 5 des Algorithmus, nur noch Testpunkte, für die die Bedingung (3.9) erfüllt ist, als neue Iterationspunkte akzeptiert. Testpunkte, die (3.9) dagegen nicht erfüllen, werden verworfen und haben somit keine Auswirkung auf den aktuellen Iterationswerte oder die Anpassung des Trust-Region-Radius.
Daher und aus der Forderung der Annahmen A1 - A4, können die Resultate der Konvergenzanalyse des Basic-Trust-Region-Algorithmus für diesen Fall übernommen werden (vgl. Satz 2.8 und Satz 2.9):

Korollar 3.1 *Es gelten die Annahmen A1 - A4 und es sind nur endlich viele Testpunkte x_k^+ für den Filter akzeptabel. Weiterhin sei die Anzahl der erfolgreichen Iterationsschritte endlich. Dann terminiert der Algorithmus in Schritt 1 mit $x_k = x_*$ und $\nabla f(x_*) = 0$.*

Korollar 3.2 *Es gelten die Annahmen A1 - A4 und es sind nur endlich viele Testpunkte x_k^+ für den Filter akzeptabel. Weiterhin sei die Anzahl der erfolgreichen Iterationsschritte unendlich. Dann gilt:*

$$\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

Damit wurde bereits für zwei Fälle die Konvergenz des FTR-Algorithmus gegen Nullstellen der ersten Ableitung der zu minimierenden Zielfunktion gezeigt. Der einzige Fall, den es noch zu betrachten gilt, ist folgender: Es werden nur endlich viele Einträge in den Filter aufgenommen, aber es gibt unendlich viele, die akzeptabel sind. In dieser Situation ergeben sich für ein hinreichend großes k drei Arten von Iterationsschritten:

- Iterationsschritte, bei denen x_k^+ für den Filter nicht akzeptabel ist und (3.9) nicht erfüllt ist. Diese Iterationsschritte brauchen für die Konvergenzanalyse nicht berücksichtigt zu werden. Die Gründe dafür wurden bereits in der Argumentation vor Korollar 3.1 dargelegt.

- Iterationsschritte, bei denen x_k^+ für den Filter nicht akzeptabel ist, aber (3.9) erfüllt ist, sind reine Trust-Region-Iterationsschritte.

- Bei der dritten Art von Iterationsschritten ist x_k^+ für den Filter akzeptabel, aber θ_k^+ wird nicht in den Filter aufgenommen, da $\rho_k \geq \eta_1$ und (3.9) erfüllt ist. Dies sind wiederum normale Trust-Region-Iterationsschritte.

Eine Folge der betrachteten Iterationsschritten kann demnach so angesehen werden, als stamme sie von einem Basic-Trust-Region-Algorithmus. Damit sind die dafür gezeigten Konvergenzresultate übertragbar. Als Konsequenz ist auch eine Folgerung aus Korollar 3.1 und Korollar 3.2 für den FTR-Algorithmus gültig:

Satz 3.2 *Es gelten die Annahmen A1 - A4. Dann gibt es eine Teilfolge $\{k_i\}$, so dass $\liminf_{i\to\infty} \|\nabla f(x_{k_i})\| = 0$. Darüber hinaus ist $\liminf_{k\to\infty} \|c(x_k)\| = 0$, wenn unendlich viele Werte in den Filter aufgenommen werden.*

Dieser Satz ist keine Garantie dafür, dass eine Teilfolge $\{\|c_k\|\}$ gegen Null konvergiert, sondern nur dafür, dass eine Nullstelle der ersten Ableitung der Zielfunktion $f(x)$ erreicht wird. Dieses Resultat entspricht durchaus den Erwartungen, da es passieren kann, dass (3.1) keine Lösung hat oder dass die Folge der Iterationspunkte gegen ein lokales Minimum von $f(x)$ konvergiert. Die einzige Möglichkeit, solche Situationen zu vermeiden, ist die Verwendung von globalen Optimierungsverfahren, die jedoch hier nicht thematisiert werden sollen. Die zweite Aussage des Satzes ($\liminf_{k\to\infty} \|c(x_k)\| = 0$ gilt, falls unendlich viele Punkte in den Filter aufgenommen werden) ist wahr, da eine Folge von unendlich vielen Punkten, aufgrund der verwendeten Akzeptanzbedingung, nur dann in den Filter aufgenommen werden kann, wenn sie gegen den Nullpunkt konvergiert.
Aufgrund der bisher erhaltenen Ergebnisse ist es zulässig, der Argumentation der Konvergenzanalyse des Basic-Trust-Region-Algorithmus weiter zu folgen. Es kann demnach gezeigt werden, dass der Limes Inferior in Korollar 3.2 durch den Limes ersetzt werden kann, wenn zusätzlich für ein $k_0 > 0$ und eine Konstante $\kappa_\Delta \geq 1$ gilt:

$$\|s_k\|_k \leq \kappa_\Delta \Delta_k \qquad \forall k \geq k_0. \tag{3.16}$$

Im gewöhnlichen Trust-Region-Kontext ist diese Bedingung automatisch erfüllt, da die Schrittweite in solchen Verfahren auf jeden Fall kleiner als der Trust-Region-Radius sein muss. Für den hier vorgestellten FTR-Algorithmus ist dies allerdings nicht sofort ersichtlich, da theoretisch auch unbeschränkte Schrittweiten möglich sind.
Dennoch kann man ein stärkeres Konvergenzresultat erhalten, wenn angenommen wird, dass der Testschritt s_k immer kleiner als eine implementationsabhängige obere Schranke, die mitunter auch sehr groß sein kann, ist. Die obige Argumentation bleibt dadurch unverändert und man kann das abschließende Konvergenzresultat für den Filter-Trust-Region-Algorithmus formulieren (vgl. Satz 2.10):

Satz 3.3 *Es gelten die Annahmen A1 - A4 sowie (3.16). Dann gilt entweder* $\|\nabla f(x_k)\| = 0$ *für einen Iterationsschritt* k *oder* $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$. *Werden unendlich viele Werte in den Filter aufgenommen, dann gilt sogar:* $\lim_{k\to\infty} \|c(x_k)\| = 0$.

Die zweite Behauptung folgt analog zu Argumentation nach Satz 3.2.
In diesem Kapitel wurde der FTR-Algorithmus vorgestellt und sein Konvergenzverhalten untersucht. In den folgenden Kapiteln wird eine auf dieser Grundlage basierende Implementierung dargestellt und damit erhaltene numerische Ergebnisse präsentiert.

Kapitel 4

Numerische Umsetzung des Algorithmus

In diesem Kapitel soll eine Implementierung des FTR-Algorithmus vorgestellt werden, wobei das Hauptaugenmerk auf den in Algorithmus 3.1 nicht explizit ausgeführten Schritten liegt. Insbesondere die Bestimmung des Testschrittes s_k ist von Interesse, da diese von entscheidender Bedeutung für die Effizienz des Algorithmus ist. Zur Ermittlung eines geeigneten Testpunktes $x_k^+ = x_k + s_k$ werden vier unterschiedliche Methoden vorgestellt. Außerdem wird die Aktualisierung des Trust-Region-Radius anhand zweier unterschiedlicher Vorgehensweisen sowie einige Details der Ein- und Ausgabe der vorliegenden Implementierung betrachtet.
Zunächst soll allerdings noch auf Funktionen eingegangen werden, die für die Handhabung des mehrdimensionalen Filter benötigt werden. Die hier vorgestellten Funktionen dienen dazu, einen Filter zu initialisieren, ihn zu aktualisieren, eine graphische Darstellung für zwei- oder dreidimensionale Filter zu erzeugen und die Verbesserung, die erreicht wird, wenn ein Punkt in den Filter aufgenommen wird, zu berechnen.

4.1 Implementierung des mehrdimensionalen Filters

Im vorherigen Kapitel wurde der mehrdimensionale Filter als Liste bestehend aus n-dimensionalen Einträgen definiert. Da die hier vorgestellte Implementierung in Matlab realisiert wird, ist es naheliegend, den Filter als Matrix zu speichern, deren Zeilenvektoren die einzelnen Filtereinträge enthalten. Dabei wird darauf geachtet, dass die Filtereinträge nach der Initialisierung oder einer Veränderung des Filters lexikographisch sortiert werden. In einer zusätzlichen Spalte wird der Iterationszähler des zugrunde liegenden iterativen Verfahrens, beispielsweise des hier verwendeten Trust-Region-Verfahrens, festgehalten. Dadurch ist sofort ersichtlich, aus welchen Iterationsschritten die Einträge des Filters hervorgegangen sind. Insbesondere lässt sich dadurch sehr

einfach feststellen, ob sich der Filter während eines Iterationsschrittes geändert hat. Diese Abfrage kann in Schritt 5 von Algorithmus 3.1 zum Erzeugen der Fallunterscheidung, ob der Testpunkt x_k^+ vom Filter akzeptiert wird oder nicht, verwendet werden.

Initialisierung des Filters

Es gibt verschiedene Ausgangssituationen, die der Erzeugung eine Filters F zugrunde liegen können. Die einfachste Möglichkeit ist die Initialisierung eines Filters, der noch keine Punkte enthält. Dies geschieht, indem man F eine leere Matrix zuweist. Soll der Filter aus einer gegebenen Menge von Punkten erstellt werden, so kann man unterscheiden, ob die Menge einen oder mehrere Punkte enthält. Im ersten Fall kann der einzige Punkt ungeprüft in den Filter aufgenommen werden. Die Matrix F besteht dann aus genau einem Zeilenvektor. Andernfalls muss anhand einer Akzeptanzbedingung überprüft werden, welche Punkte der Ausgangsmenge in den Filter aufgenommen werden. Die Matrix F enthält dann nur die Punkte, die diese Bedingung erfüllen.
In den Kapiteln 2 und 3 wurden bereits zwei mögliche Akzeptanzbedingungen eingeführt:

AB 1: Ein Punkt ist akzeptabel für den Filter, wenn er von keinem anderen Punkt dominiert wird (vgl. Definition 2.8)
AB 2: Ein Punkt ist akzeptabel für den Filter, wenn er die in Definition 3.3 beschriebene Mindestverbesserung erreicht.

Für den FTR-Algorithmus ist es unerheblich, welche Akzeptanzbedingung man im Initialisierungsschritt verwendet, da der Filter nach Schritt 1 von Algorithmus 3.1 als leere Menge erzeugt wird.
Um jedoch dem allgemeinen Fall einer Initialisierung, ausgehend von einer Menge von Punkten, gerecht zu werden, verwenden die beiden im Folgenden vorgestellten Funktionen als Akzeptanzbedingung AB 1. Der Unterschied zur Erzeugung eines Filters mit Hilfe von AB 2 liegt darin, dass bei Verwendung der Dominanzbedingung (AB 1) genau die Punkte in den Filter aufgenommen werden, zu denen es keinen anderen Punkt, der in allen Komponenten einen kleineren Wert aufweist, gibt.
Folgendes Beispiel zeigt, warum dies für den Initialisierungsschritt eines Filters ein Vorteil gegenüber der Verwendung von AB 2 ist: Seien x_1 und x_2 zwei Punkte der Punktmenge M aus deren Elementen der Filter gebildet werden soll. Dabei gilt $x_{1,k} = x_{2,k} + \epsilon_k$ für alle Koordinaten k und Konstanten $\epsilon_k > 0$, wobei jedes ϵ_k kleiner als die geforderte Mindestverbesserung in der entsprechenden Koordinate ist. x_1 und x_2 erfüllen für alle anderen Punkte der Menge M die Akzeptanzbedingung. Es gibt nun zwei Möglichkeiten, wie der Filter initialisiert wird. Entweder er wird sukzessive durch Hinzufügen einzelner Punkte erzeugt. In diesem Fall werden die Testpunkte nur anhand der bereits im Filter vorhandenen Punkte überprüft. Oder jeder Punkt aus der zugrundeliegenden Menge M wird mit allen anderen Punkten aus M verglichen. In den Filter werden in beiden Fällen nur diejenigen Punkte aufgenommen, die die Akzeptanzbedingung für jeden der jeweiligen Referenzpunkte erfüllen.
Betrachtet man den ersten Fall, so sieht man, dass der Punkt aus der Menge $\{x_1, x_2\}$

im entstehenden Filter enthalten sein wird, der zuerst getestet wurde, da x_1 die Akzeptanzbedingung des Filters für x_2 nicht erfüllt und x_2 wegen x_1 nicht aufgenommen werden kann. Das kann dazu führen, dass der eigentlich bessere Punkt x_2 zugunsten des Punktes x_1 abgelehnt wird. Wird der Filter wie im zweiten Fall beschrieben initialisiert, so wird weder x_1 noch x_2 in den Filter aufgenommen, da keiner der beiden Punkte die Akzeptanzbedingung für den jeweils anderen erfüllt. In beiden beschriebenen Fällen enthält der Filter nach dem Initialisierungsschritt evtl. nicht alle geeigneten Punkte der Ausgangsmenge M.
Da in der Akzeptanzbedingung AB 1 auf die Forderung nach einer Mindestverbesserung verzichtet wird, wird unabhängig von der Reihenfolge, in der die Punkte getestet werden, jeder Punkt in den Filter aufgenommen, der von keinem anderen dominiert wird. Es entsteht demnach ein eindeutig bestimmter Filter und der oben beschriebene negative Effekt, der bei Verwendung von AB 2 entstehen kann, tritt nicht auf.
Aufgrund dieser Überlegungen kann man nun einen Algorithmus zu Initialisierung eines Filters formulieren (Die Bezeichnungen aller Algorithmen in diesem Kapitel ergeben sich aus dem Namen der Matlab-Funktion, in der der jeweilige Algorithmus realisiert wurde):

Algorithmus 4.1 init_filter.m

Eine Menge M von Punkten x_k sei gegeben mit $k \in \{1, ..., n\}$; n sei die Anzahl der Punkte.
If $n = 0$ **then**
Initialisiere den Filter als leere Matrix.
elseif $n = 1$ **then**
Initialisiere den Filter als Zeilenvektor $(x_1, 0)$.
else
- *Bestimme alle Punkte x_i, die von keinem anderen Punkt aus M dominiert werden.*
- *Erstelle eine Matrix mit den Zeilenvektoren $(x_i, 0)$, wobei x_i die nicht dominierten Punkte sind. Diese Matrix entspricht dem Filter.*

end

In Algorithmus 4.1 werden die Filtereinträge ausgewählt, indem jeder Punkt mit allen anderen Punkten der Ausgangsmenge M verglichen wird.
Der folgenden Algorithmus basiert auf dem Graef-Younes-Verfahren [20], erzeugt aber im Gegensatz dazu eine Matrix, die nur nicht dominierte Punkte enthält und stellt eine Möglichkeit dar, die Anzahl der benötigten Vergleiche zu reduzieren und somit den Filter in kürzerer Rechenzeit aufstellen zu können:

Algorithmus 4.2 init_filter_graef.m

Eine Menge M von Punkten x_k sei gegeben mit $k \in \{1, ..., n\}$; n sei die Anzahl der Punkte.

If $n = 0$ **then**
Initialisiere den Filter als leere Matrix.
elseif $n = 1$ **then**
Initialisiere den Filter als Zeilenvektor $(x_1, 0)$.
else
Initialisiere eine Matrix T *mit dem Zeilenvektor des ersten Punktes aus* M.
For $i = 2$ **to** n
- *Überprüfe, ob der* i*-te Punkt aus* M *von einem Punkt aus* T *dominiert wird.*
- *Ist dies nicht der Fall, so füge den Punkt* x_i *zu* T *hinzu.*

end
- *Sei* m *die Anzahl der Punkte in* T.
- *Kehre die Reihenfolge der Punkte in* T *um.*
- *Initialisiere eine Matrix* S *mit dem Zeilenvektor des ersten Punktes aus der umgekehrten Matrix* T.

For $i = 2$ **to** m
- *Überprüfe, ob der* i*-te Punkt aus der umgekehrten Matrix* T *von einem Punkt aus* S *dominiert wird.*
- *Ist dies nicht der Fall, so füge den Punkt* x_i *zu* S *hinzu.*

end
Ergänze jeden Zeilenvektor aus S *um den Iterationszähler* 0 *in der letzten Spalte. Die entstehende Matrix entspricht dem Filter.*
end

Dieses Verfahren ist in [15] angegeben. Um zu überprüfen, ob durch Verwendung der Funktion init_filter_graef.m tatsächlich eine Verbesserung der Rechenzeit erreicht wird, wurde ein numerischer Test durchgeführt. Wie bei allen weiteren Tests wurde dafür ein PC mit 1 GB Arbeitsspeicher und einem 1.66 GHz Dual-Core-Prozessor sowie Matlab 7 (R14) verwendet. Die Ergebnisse dieses Test sind in den Tabellen 4.1 und 4.2 dargestellt. Es wurden n Punkte der Dimension m mit zufälligen Koordinateneinträgen erzeugt und die benötigte Rechenzeit der beiden Funktionen init_filter.m und init_filter_graef.m gemessen. Für jedes Paar (n, m) wurde der Test zehn Mal mit unterschiedlichen Zufallswerten durchgeführt und die durchschnittliche Rechenzeit bestimmt, um ein möglichst representatives Ergebnis zu erhalten. Diese Werte sind in den beiden folgenden Tabellen aufgeführt.
Man sieht anhand der ermittelten Werte, dass die Funktion init_filter_graef.m vor allem für die Initialisierung eines Filters mit sehr vielen Punkten deutlich weniger Rechenzeit benötigt. Dies bestätigt die Gründe, die zur Entwicklung dieses Verfahrens geführt haben. Im FTR-Algorithmus ist es unerheblich, welche der beiden Funktionen verwendet wird, da der Filter als leere Menge initialisiert wird. Um auch in einem allgemeineren Kontext möglichst kurze Laufzeiten zu erreichen, wird der Filter jedoch immer mit init_filter_graef.m initialisiert.

	20	50	75	100	200	500	750	1000
20	0,0012	0,0031	0,0032	0,0045	0,0094	0,0152	0,0296	0,0312
50	0,0109	0,0157	0,0187	0,0219	0,0438	0,1032	0,1484	0,1938
75	0,0249	0,0343	0,0436	0,0547	0,0969	0,2266	0,3438	0,4595
100	0,0391	0,0547	0,0780	0,0938	0,1719	0,4078	0,6187	0,8358
200	0,1609	0,2391	0,3125	0,3750	0,6844	1,7577	2,6500	3,5875
500	1,5719	2,1171	2,6095	3,1298	5,6594	13,155	19,233	25,258
750	4,7843	6,0282	7,2328	8,5406	14,336	31,636	45,534	59,481
1000	10,609	12,823	15,145	17,572	28,131	59,155	87,855	114,16

Tabelle 4.1: Rechenzeiten in Sekunden von init_filter.m, dabei ist in den Spalten die Dimension und in den Zeilen die Anzahl der zu testenden Punkte angegeben.

	20	50	75	100	200	500	750	1000
20	0,0004	0,0015	0,0016	0,0031	0,0069	0,0112	0,0158	0,0201
50	0,0047	0,0077	0,0110	0,0125	0,0263	0,0655	0,1048	0,1343
75	0,0063	0,0188	0,0250	0,0312	0,0577	0,1484	0,2281	0,3172
100	0,0141	0,0282	0,0423	0,0547	0,0998	0,2717	0,4218	0,5704
200	0,0470	0,1110	0,1562	0,2031	0,4032	1,1437	1,7812	2,4641
500	0,3236	0,6703	0,9751	1,3234	2,8219	8,1247	12,611	17,137
750	0,7189	1,5094	2,2610	3,0891	6,6516	19,114	29,374	39,619
1000	1,2687	2,7032	4,1656	5,6281	12,430	34,839	53,127	71,358

Tabelle 4.2: Rechenzeiten in Sekunden von init_filter_graef.m, dabei ist in den Spalten die Dimension und in den Zeilen die Anzahl der zu testenden Punkte angegeben.

Aktualisierung des Filters

Immer wenn ein Punkt in den Filter aufgenommen wird, muss die Matrix des Filters verändert werden. Im FTR-Algorithmus besteht der Filter aus den Fehlervektoren θ_k der Iterationswerte x_k. Ein neuer Fehlervektor wird zum Filter hinzugefügt, wenn er für den Filter akzeptabel ist, d.h. wenn er von keinem Filtereintrag dominiert wird, außerhalb des Korridors um den Filter, der alle Punkte mit ungenügender Verbesserung enthält, liegt und entweder die Modellfunktion eine schlechte Vorhersage der Verbesserung des Zielfunktionswertes liefert oder die Schrittweite zum Testpunkt größer als der aktuelle Trust-Region-Radius ist.

Wird ein neuer Punkt in den Filter aufgenommen, so müssen die bereits im Filter vorhandenen Einträge überprüft und diejenigen, die durch den neu hinzu gekommenen Eintrag überflüssig geworden sind, entfernt werden. Um den aktualisierten Filter zu erstellen, ist es möglich, die Funktionen init_filter.m bzw. init_filter_graef.m zu verwenden, wenn darin die Akzeptanzbedingung AB 1 durch AB 2 ersetzt wird, um die erforderliche Mindestverbesserung zu berücksichtigen. Dazu wählt man als Ausgangsmenge zum Initialisieren des Filters die Menge aller potentiellen Filtereinträge des

neuen Filters, d.h. alle Einträge des alten Filters und den Punkt, der auf seine Akzeptanz getestet werden soll.
Da die Einträge des Filters vor der Aktualisierung im bisherigen Verlauf des Algorithmus schon mindestens einmal miteinander verglichen worden sein müssen - entweder bei der Initialisierung oder bei einer früheren Aktualisierung des Filters - genügt es, die alten Filtereinträge bezüglich des neuen Punktes zu überprüfen. Dadurch werden einige Vergleiche von Punkten und somit Rechenzeit eingespart.
Zusammenfassend lässt sich mit Hilfe dieser Überlegungen der folgende Algorithmus formulieren, wobei im Unterschied zur Initialisierung die Akzeptanzbedingung AB 2 verwendet wird, damit die Konvergenz des FTR-Algorithmus gesichert ist:

Algorithmus 4.3 add_to_filter_env.m

Ein Filter F, ein Testpunkt x_k und der Iterationszähler k sind gegeben, außerdem die Parameter γ_θ und δ.
If *x_k erfüllt die von γ_θ abhängige Akzeptanzbedingung für alle Filtereinträge in der durch δ ausgewählten Abschätzung* **then**
- *Führe für jeden Filtereintrag einen Akzeptanztest bzgl. x_k durch.*
- *Entferne alle Einträge, für die der Test fehlschlägt aus dem Filter.*
- *Füge den Zeilenvektor (x_k, k) an der passenden Stelle zum Filter hinzu.*

else
Der aktualisierte Filter entspricht dem ursprünglichen.
Endif

Der Parameter γ_θ in Algorithmus 4.3 bestimmt, wie in Definition 3.3, die Größe des Korridors der Mindestverbesserung um den Filter. Mit Hilfe des Parameters δ wird die Norm festgelegt, die zur Bestimmung der Mindestverbesserung verwendet wird. Ebenfalls nach Definition 3.3 gibt es dafür die Möglichkeiten, die ℓ_2-Norm des neuen Punktes, die ℓ_2-Normen der Filtereinträge oder das Minimum dieser beiden Ausdrücke zu wählen.
Die Funktion add_to_filter.m unterscheidet sich nur dadurch von add_to_filter_env.m, dass bei ersterer die Akzeptanzbedingung AB 1 verwendet wird. Ein Testpunkt wird demnach immer dann akzeptiert, wenn er von keinem Filtereintrag dominiert wird. Der Filter, der aus allen nicht dominierten Punkten besteht, der also durch Verwendung von add_to_filter.m entsteht, wird in den, im Anschluss erläuterten, Funktionen zum Messen der Größe der Filterverbesserung durch einen Testpunkt und in derjenigen, die einen dreidimensionalen Filter graphisch darstellt, verwendet.

Bestimmen der Filterverbesserung

Es kann nützlich sein, die Verbesserung eines Filters, verursacht durch einen neu aufgenommenen Punkt, messen zu können. Dies ermöglicht den Vergleich und die Bewertung verschiedener Testpunkte oder die Dokumentation der Entwicklung des Filters im Laufe eines iterativen Verfahrens. In diesem Abschnitt sollen zwei Funktionen vorgestellt werden, durch die ein Wert für die Verbesserung, die ein Testpunkt für einen Filter erreicht, bestimmt werden kann.

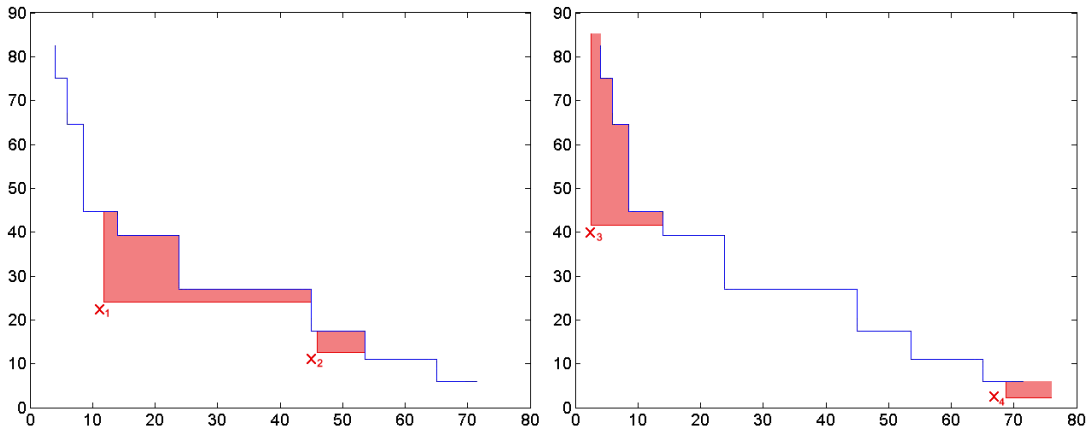


Abbildung 4.1: Darstellung von verschiedenen Filterverbesserungen (rote Bereiche).

In Abbildung 4.1 werden die Filterverbesserungen für verschiedene Punkte dargestellt. Für die Punkte x_1 und x_2 kann die erreichte Verbesserung sofort anhand des Flächeninhalts des rot gefärbten Gebietes abgelesen werden. Die im rechten Bild dargestellten Verbesserungen sind nicht so einfach zu messen, da die Punkte x_3 und x_4 in jeweils einer Koordinate einen kleineren Wert haben als alle bereits im Filter enthaltenen Punkte und somit der Zuwachs des vom Filter dominierten Gebietes unendlich groß ist.
Anschaulich betrachtet ist es das Ziel eines Filter-Verfahrens, eine Folge von Iterationswerten zu erzeugen, die gegen den Ursprung des Koordinatensystems konvergiert. Deshalb sollten in einem Vergleich von verschiedenen Testpunkten diejenigen, die zwar in einer Koordinate einen sehr guten Wert, in anderen Koordinaten aber sehr schlecht sind (dies ist beispielsweise bei Punkt x_4 in Abbildung 4.1 der Fall), nicht am besten bewertet werden. Dies wäre jedoch der Fall, wenn man ihnen eine unendlich große Verbesserung zuordnen würde. Eine Möglichkeit die Verbesserung dieser Punkte realistischer zu betrachten, ist die Einführung einer oberen Schranke s_i für jede Koordinate i, die verhindert, das ein Punkt eine unendlich große Verbesserung erreicht.
Die genaue Definition dieser Schranke ist eine Frage der Modellierung. In den beiden im Folgenden vorgestellten Funktionen wird sie für die i-te Koordinate mit Hilfe von

$$m_i = \max\left(\{x_i \in \mathbb{R} : \text{x ist Filtereintrag}\} \cup \{y_i \in \mathbb{R} : \text{y ist der Testpunkt}\}\right)$$

als

$$s_i = m_i + \frac{1}{m_i}$$

definiert, wobei x_i bzw. y_i die i-te Koordinate des Punktes x bzw. y bezeichnen.
Diese Definition beruht auf den beiden folgenden Überlegungen: Zum einen muss die Schranke größer als der größte Wert der Filtereinträge in dieser Koordinate sein, um zu gewährleisten, dass die gesamte relevante Verbesserung berücksichtigt wird. Außerdem muss sie auf jeden Fall größer als der Wert des Testpunktes in dieser Koordinate

sein, da nur dann in jedem Fall ein Gebiet mit positivem Flächeninhalt entsteht. Daraus ergibt sich die Forderung $s_i > m_i$. Zum anderen wird durch die Addition des Quotienten $\frac{1}{m_i}$ erreicht, dass die Schranke umso näher am Testpunkt liegt, je größer dessen i-te Koordinate ist. Dies führt in Situationen, wie beispielsweise der in Abbildung 4.2 dargestellten dazu, dass auch Testpunkte, die eigentlich eine unendlich große Filterverbesserung erreichen, sinnvoll miteinander verglichen werden können. So wird im Beispiel von Abbildung 4.2 dem Punkt mit kleinerem Wert in der ersten Koordinate, also x_5, eine größere Verbesserung zugeschrieben als dem Punkt x_6.

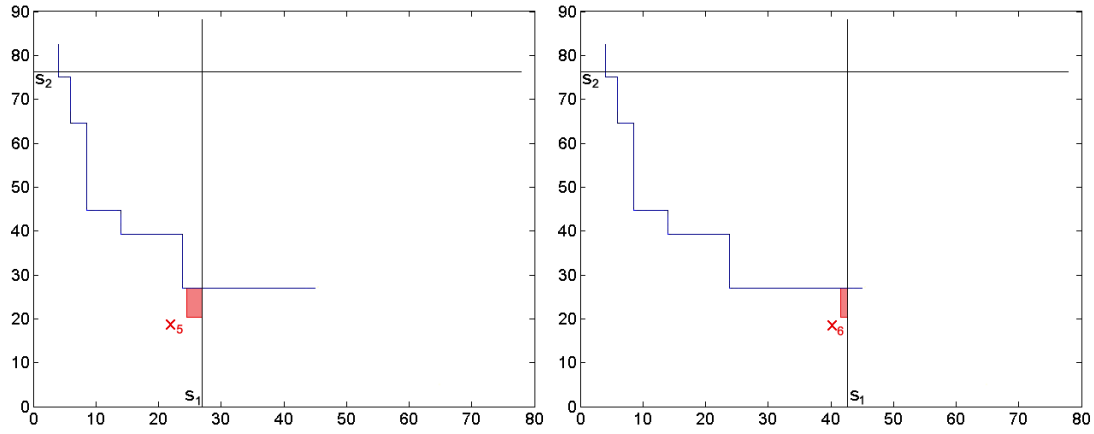


Abbildung 4.2: Darstellung der Beschränkungen des Verbesserungsbereichs für unterschiedliche Punkte. Der Abstand der Schranke s_1 zu den Punkten x_5 bzw. x_6 ist nicht maßstabsgetreu, um den Unterschied deutlicher zu machen.

In der folgenden Argumentation, die sich aus geometrischen Überlegungen ergibt, wird der Begriff Filter auch für die graphische Darstellung, d.h. die mehrdimensionale Treppenfunktion, die durch Verbinden der Punkte eines Filters entsteht, verwendet. Um die durch einen Testpunkt x_k erreichte Verbesserung zu messen, muss der Filter vor einem Aktualisierungsschritt mit dem durch die Aktualisierung entstehenden neuen Filter verglichen werden. Sind die Matrizen der beiden Filter identisch, so wurde x_k nicht akzeptiert und es wurde somit auch keine Verbesserung erzielt.
Unterscheiden sich die beiden Matrizen, so lässt sich die Größe der Verbesserung geometrisch als Volumen, das vom alten und von dem verbesserten Filter eingeschlossen wird, beschreiben. In Abbildung 4.1 beispielsweise ist der neue Filterverlauf durch rote Linien gekennzeichnet. Eine Verschlechterung des Filters ist durch dessen Akzeptanzbedingung ausgeschlossen. Das bedeutet, dass die Veränderung durch einen Aktualisierungsschritt, falls der Testpunkt akzeptiert wurde, auf jeden Fall echt positiv sein muss.
Die ersten Schritte der beiden Funktionen, die der Berechnung der Filterverbesserung dienen, sind identisch. Zunächst wird die Matrix F des ursprünglichen Filters um Einträge der Form $x_{n+i} = (0, ..., 0, s_i, 0, ..., 0)$ für jede Koordinate i des n-dimensionalen

Filters ergänzt, um die oberen Schranken, wie in Abbildung 4.2 dargestellt, zu realisieren.
Danach wird der Ursprung des Koordinatensystems in den Testpunkt verschoben. Alle Punkte, die nach dieser Koordinatentransformation in allen Koordinaten positive Werte haben, werden unverändert übernommen. Bei den übrigen Punkten werden die negativen Werte in einer oder mehreren Koordinaten durch den Wert 0 ersetzt. Dies entspricht einer Projektion dieser Punkte auf die positiven Abschnitte der Koordinatenebenen. Mit der somit entstandenen Punktmenge wird ein neuer Filter initialisiert, um die überflüssigen aus Projektionen entstandenen Punkte zu entfernen. Der Graph dieses Filter bildet zusammen mit den Koordinatenebenen (nach der Koordinatentransformation) den Rand des Gebiets G, dessen Volumen der Filterverbesserung entspricht und somit bestimmt werden muss, um deren Größe zu ermitteln.
Um dieses Volumen zu berechnen, sollen zwei Ansätze vorgestellt werden: Die erste Variante basiert auf der Überlegung, dass die mit dem Gebiet G assoziierte n-dimensionale geometrische Figur aus n-dimensionalen Quadern zusammengesetzt werden kann. Ist es also möglich, das Volumen der einzelnen Quader bestimmen, so ergibt sich aus der Summe dieser Werte die gesuchte Verbesserung. Die Quader werden anhand eines Gitters, auf dessen Gitterpunkten alle Filtereinträge liegen, festgelegt. Jeder dieser Quader entspricht dann einer Gitterzelle. In Abbildung 4.3 ist dies für ein zweidimensionales Beispiel dargestellt.

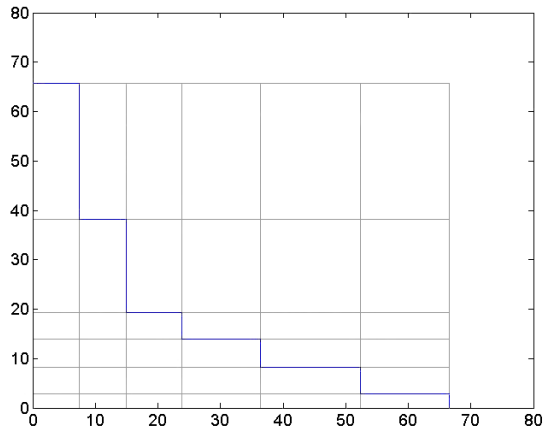


Abbildung 4.3: Gitter zur Bestimmung des Flächeninhalts unter einem zweidimensionalen Filter.

Für alle Quader, die innerhalb des Gebiets G, das vom Filter und den Koordinatenebenen eingeschlossen wird, liegen, bestimmt man das Volumen. Die gesuchte Verbesserung erhält man danach durch Aufsummieren aller so erhaltenen Werte.
Aus diesen Überlegungen ergibt sich der erste Algorithmus zur Berechnung der Verbesserung, die ein Testpunkt für einen Filter erreicht:

Algorithmus 4.4 verb_mess_raster.m

Schritt 1: Initialisierung
Ein Filter F mit n Einträgen x_k der Dimension m und ein Testpunkt x_0 sind gegeben. Initialisiere eine Variable $V = 0$, in der die Verbesserung gespeichert wird und deren Wert die Ausgabe dieser Funktion ist.
Initialisiere einen m-dimensionalen Vektor $a = (1, ..., 1)$.

Schritt 2: Beschränkungen einfügen
Falls x_0 von F nicht akzeptiert wird: STOP.
Andernfalls füge die Einträge $x_{n+i} = (0, ..., 0, s_i, 0, ..., 0)$ für $i = 1, ..., m$ zu F hinzu.

Schritt 3: Koordinatentransformation
- *Ersetze die j-te Koordinate des Eintrags x_i aus F durch $x_{i,j} = \max(0, x_{i,j} - x_{0,j})$ für $i = 1, ..., n + m$ und $j = 1, ..., m$.*
- *Entferne alle Einträge die von einem anderen Eintrag dominiert werden und bezeichne den so entstandenen Filter, der durch die $\nu \times m$-Matrix M repräsentiert wird, mit F_M.*

Schritt 4: Koordinaten der Gitterpunkte
Sortiere die Einträge der Spalten von M aufsteigend, so dass $m_{1,i} \leq ... \leq m_{n_1,i}$ für die i-te Spalte gilt, $i = 1, ..., m$.

Schritt 5: Testpunkt der aktuellen Gitterzelle
Bestimme einen Testpunkt t mit den Koordinaten $t_i = \frac{m_{a_i,i} + m_{a_i+1,i}}{2}$ für $i = 1, ..., m$.

Schritt 6: Volumen der aktuellen Gitterzelle
Falls t von F_M akzeptiert wird:
Berechne die Seitenlänge $l_i = m_{a_i+1,i} - m_{a_i,i}$ der aktuellen Gitterzelle für $i = 1, ..., m$ sowie den Inhalt $v_Z = \prod_{i=1}^{m} l_i$ und aktualisiere $V = V + v_Z$.

Schritt 7: Aktualisierung des Zählers
- *Erhöhe a_1 um 1, d.h. $a_1 = a_1 + 1$.*
- *Überprüfe für alle a_i, $i = 1, ..., m - 1$ mit a_1 beginnend aufsteigend nacheinander ob $a_i = n_1$ gilt. Ist dies der Fall, so setze $a_{i+1} = a_{i+1} + 1$ und $a_i = 1$.*
- *Ist $a_m = n_1$: STOP.*
- *Gehe zu Schritt 5.*

Der Nachteil von Algorithmus 4.4 ist der vergleichsweise hohe Rechenaufwand und die damit verbundene lange Rechenzeit, da bei m Punkten der Dimension n die Anzahl der entstehenden Quader $(m-1)^n$ ist. Für jeden dieser Quader muss die Akzeptanz des Punktes t überprüft und gegebenenfalls das Volumen berechnet werden. Ein wenig reduzieren ließe sich dieser Aufwand, wenn im Aktualisierungsschritt des Vektors a folgende zusätzliche Regelung eingeführt wird: Ist ein Punkt t nicht akzeptabel für den Filter, so setze $a(1) = n_1$. Dies ist möglich, da die Testpunkte der übersprungenen Quader von dem bereits nicht akzeptierten t dominiert werden und somit ebenfalls nicht akzeptiert werden können. Damit kann die Anzahl der zu überprüfenden Test-

punkte t deutlich reduziert werden, aber der Aufwand beim Berechnen der Volumen der einzelnen Quader ändert sich nicht.
Eine größere Reduzierung der Rechenzeit erreicht man durch die Verwendung eines anderen Ansatzes zur Berechnung der durch einen Testpunkt erzielten Verbesserung. Wie in Algorithmus 4.4 wird das Gebiet, dessen Volumen bestimmt werden soll, durch eine Koordinatentransformation so verschoben, dass es von den positiven Koordinatenebenen und dem Graph des Filters begrenzt wird. Es wird nun allerdings keine Unterteilung in Quader vorgenommen, sondern man benutzt die Überlegung, dass der Graph eines m-dimensionalen Filters aus mehreren $(m-1)$-dimensionalen Filtern zusammengesetzt ist. In Abbildung 4.4 wird dies anhand eines dreidimensionalen Beispiels graphisch demonstriert, wobei vor allem in der Mitte des rechten Bildes die Filterform der einzelnen Abschnitte des Graphen deutlich zu erkennen sind.

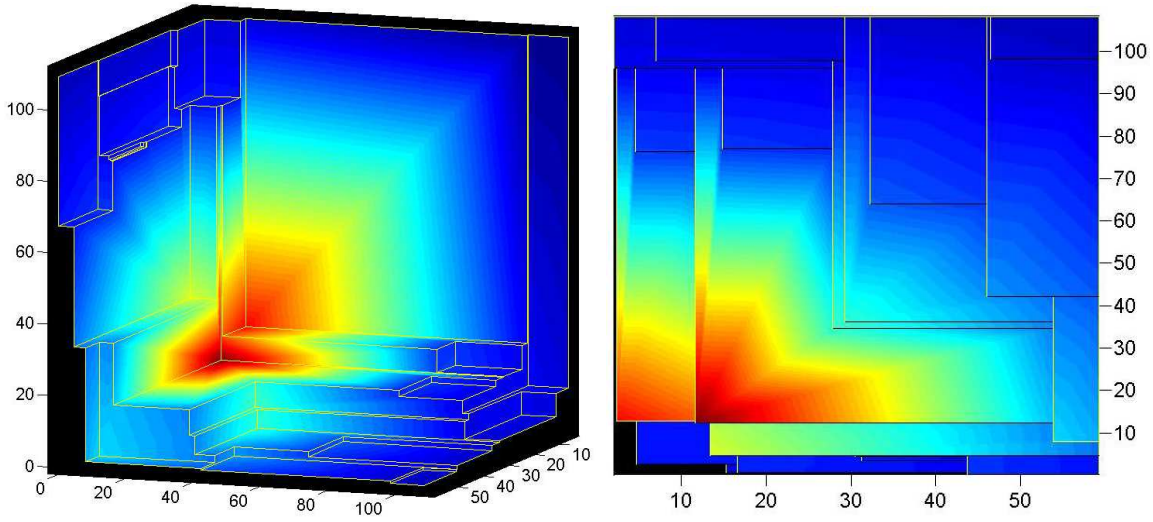


Abbildung 4.4: Im linken Bild ist ein dreidimensionaler Filter dargestellt. Im rechten Bild wurde die Ansicht so verschoben, dass man parallel zur x_1-Achse senkrecht auf den unteren Teil des Filter sieht.

Da jeder solche Abschnitt des Filters in der Ebene eines Filtereintrages liegt, die parallel zu einer Koordinatenebene ist, lässt sich die zu bestimmende Verbesserung, also das Volumen des Gebiet unter dem Filter als $V = \sum_{i=1}^{n} v_i \cdot h_i$ berechnen. Dabei ist n die Anzahl der Einträge im Filter, v_i das Volumen des in der Ebene des i-ten Filtereintrages liegenden Abschnitts des Filters und h_i der Abstand dieses Abschnittes zur parallelen Koordinatenebene.
Das bedeutet, das Volumen unter einem m-dimensionalen Filter wird durch Berechnung der Volumen unter n $(m-1)$-dimensionalen Filtern ermittelt. Man fährt mit dieser rekursiven Vereinfachung des Problems solange fort, bis die Filter, unter welchen man das Volumen bestimmen möchte, zweidimensional sind. Der Flächeninhalt unter einem zweidimensionalen Filter lässt sich sehr einfach bestimmen, da die $(m-1)$-dimensionalen Abschnitte des Filters in diesem Fall Strecken sind und sich

der Flächeninhalt somit als Summe der Flächeninhalte von Rechtecken ergibt.
Diese Überlegungen zur rekursiven Berechnung der durch einen Testpunkt erreichten Verbesserung führt zu folgendem Algorithmus:

Algorithmus 4.5 verb_mess_fil.m

Schritt 1: Initialisierung
Ein Filter F mit n Einträgen x_k der Dimension m und ein Testpunkt x_0 sind gegeben.
Initialisiere eine Variable $V = 0$, in der die Verbesserung gespeichert wird und deren Wert die Ausgabe dieser Funktion ist.

Schritt 2: Beschränkungen einfügen
Falls x_0 von F nicht akzeptiert wird: STOP.
Andernfalls füge die Einträge $x_{n+i} = (0, ..., 0, s_i, 0, ..., 0)$ für $i = 1, ..., m$ zu F hinzu.

Schritt 3: Koordinatentransformation
- *Ersetze die j-te Koordinate des Eintrags x_i aus F durch $x_{i,j} = \max(0, x_{i,j} - x_{0,j})$ für $i = 1, ..., n + m$ und $j = 1, ..., m$.*
- *Entferne alle Einträge, die jetzt in jeder Koordinate den Wert 0 haben.*
- *Entferne alle Einträge, die von einem anderen Eintrag dominiert werden und bezeichne den so entstandenen Filter, der durch die $n_1 \times m$-Matrix M repräsentiert wird, mit F_M.*

Schritt 4: Rekursionsende
Falls $m = 2$, berechne $V = V + \sum_{i=1}^{n_1} (m_{i,1} \cdot (m_{i+1,2} - m_{i,2}))$, STOP

Schritt 5: Rekursionsschritt
Führe für jeden Filtereintrag x_i von F_M folgende Schritte durch:
- *Definiere $hoehe_i := m_{i,1}$.*
- *Gilt $m_{j,1} = hoehe_i$ für einen Filtereintrag x_j mit $j \neq i$, so füge den Zeilenvektor z mit den Koordinaten $z_k = \max(x_{i,k}, x_{j,k})$ zu F_M hinzu.*
- *Falls $hoehe_i > 0$:*
  - *Sei M^1 die Matrix M ohne den ersten Spaltenvektor.*
  - *Initialisiere eine Matrix M^2 und speichere alle Zeilenvektoren m_j^1 darin, für die $m_{j,1} \leq hoehe_i$ gilt.*
  - *Entferne alle dominierten Punkte aus M^2.*
  - *Rufe die Funktion verb_mess_fil.m mit der Matrix M^2 und dem Punkt x_i auf; sei v_1 das Ergebnis dieses Funktionsaufrufs.*
    Berechne $V = V + v_1 \cdot hoehe_i$.

Im Rekursionsschritt von Algorithmus 4.5 wird zunächst eine Koordinate i der m-dimensionalen Filtereinträge als Höhe ausgewählt. Anschließend wird das Volumen der Abschnitte des Filters bestimmt, die senkrecht zur i-ten Koordinatenachse liegen. Für jeden Filtereintrag gibt es einen solchen Abschnitt G_k. Anschaulich dargestellt kann dieser für einen Eintrag x_k folgendermaßen beschrieben werden:
Grundsätzlich ist das gesuchte Gebiet G_k innerhalb der betreffenden Ebene $E_k :=$

$\{P \in \mathbb{R}^m : P_i = x_{k,i}\}$ durch x_k und die Schranken s_j beschränkt. Daraus ergibt sich zunächst eine rechteckige bzw. quaderförmige Gestalt von G_k. Daraus müssen nun alle Teilgebiete entfernt werden, die von einem anderen Filtereintrag x_l dominiert werden. x_l dominiert genau dann einen Teil der Ebene E_k, falls

$$x_{l,i} \leq x_{k,i} \tag{4.1}$$

gilt. Für jeden Eintrag x_l mit dieser Eigenschaft wird ein quaderförmiges Teilgebiet aus G_k entfernt. Dadurch erhält G_k die Form eines $(m-1)$-dimensionalen Filters.
Da im nächsten Rekursionsschritt nur noch die $(m-1)$-dimensionale Ebene E_k betrachtet wird, wird in Schritt 5 von Algorithmus 4.5 die Matrix M^2 erzeugt, welche die Projektionen aller Punkte, die zur Begrenzung von G_k beitragen, d.h. die x_{n+i}, die in Schritt 2 des Algorithmus erzeugt wurden, sowie alle x_l für die $x_{l,i} \leq x_{k,i}$ gilt, auf die Ebene E_k enthält.
Bei dieser Vorgehensweise muss ein Spezialfall beachtet werden. Liegen in einer Ebene E_k mehrere Filtereinträge, so würde ein Teil von E_k zugleich von allen diesen Einträgen dominiert werden. Dies würde dazu führen, dass das betreffende Gebiet G^* nicht in die Berechnung des Volumens mit eingeht, da es aus den Gebieten G_k jedes Eintrags aufgrund der Dominanz durch einen weiteren Eintrag entfernt werden würde. Dieses Problem kann nicht dadurch behoben werden, dass man in (4.1) die Relation $\leq$ durch $<$ ersetzt, denn dann würde G^* mehrfach in die Volumenberechnung eingehen. Eine Möglichkeit G^* korrekt in die Berechnung einzubinden, ist das Hinzufügen eines zusätzlichen Eintrags mit den Koordinaten $z_{(i,j),k} = \max(x_{i,k}, x_{j,k})$ in den Filter F für jedes Paar x_i, x_j, das in der selben Ebene E_k liegt. Dadurch erfüllt F zwar nicht länger die Eigenschaften eines Filters, aber das ist im weiteren Verlauf des Algorithmus auch nicht notwendig. Liegen nur zwei Filtereinträge in einer Ebene, wird G^* durch das zu $z_{(i,j)}$ gehörende Gebiet vollständig beschrieben. Liegen mehr als zwei Einträge in einer Ebene, werden auch mehrere Einträge $z_{(i,j)}$ zum Filter hinzugefügt und die Vereinigung der dazugehörenden Gebiete entspricht G^*.
Noch eine weitere Ergänzung ist notwendig, damit das zu $z_{(i,j)}$ gehörende Gebiet $G_{(i,j)}$ ein positives Volumen besitzt. Da jeder der beiden Punkt x_i und x_j $G_{(i,j)}$ vollständig dominieren, muss verhindert werden, dass diese Punkte in der nächsten Rekursion für das zu $z_{(i,j)}$ gehörende Gebiet berücksichtigt werden. Dies geschieht in Schritt 3 des Algorithmus 4.5. Dort werden nach der Koordinatentransformation alle Filtereinträge entfernt, die in jeder Koordinate den Wert 0 aufweisen. Dies sind genau die Projektionen der Punkte aus der vorangegangenen Rekursion, die $G_{(i,j)}$ vollständig dominiert haben, also die Filtereinträge x_i und x_j.
Die Motivation für Algorithmus 4.5 entstand aufgrund der langen Laufzeit von Algorithmus 4.4. Anhand der folgenden Tabelle 4.3 werden die Rechenzeiten der beiden Algorithmen gegenübergestellt. Bei dem zugrundeliegende numerischen Test wurde für jede Problemgröße die benötigte Zeit für 10 zufällig erzeugte Beispiele gemessen. Die sich daraus ergebenden durchschnittlichen Werte sind in Tabelle 4.3 aufgeführt.

Dimension	3	4	5	6	7
verb_mess_fil.m	0.0016	0.0093	0.0124	0.0485	0.1678
verb_mess_raster.m	0.0140	0.4235	5.9501	604.89	9682.5

Tabelle 4.3: Rechenzeiten in Sekunden der Funktionen verb_mess_fil.m und verb_mess_raster.m bei unterschiedlichen Dimension der Filtereinträge.

Man sieht, dass die Volumenberechnung mit Hilfe der Funktion verb_mess_fil.m wesentlich schneller erfolgt. In der folgenden Tabelle 4.4 sind die Rechenzeiten der Funktion verb_mess_fil.m für Probleme höherer Dimension angeführt. Für diese war die benötigte Laufzeit von verb_mess_raster.m zu lang, als dass ein Vergleich der beiden Funktionen durchführbar gewesen wäre.

Dimension	8	9	10	15	20	25
verb_mess_fil.m	0.2218	0.4500	1.8062	38.912	474.74	5310.6

Tabelle 4.4: Rechenzeiten in Sekunden der Funktion verb_mess_fil.m für Probleme höherer Dimension.

Eine zusätzlich Anwendung der Funktionen zur Bestimmung der erreichten Filterverbesserung ist die Möglichkeit, eine alternative Akzeptanzbedingung für den Filter zu entwickeln. Beispielsweise wäre es denkbar, dass ein Punkt x_t in den Filter aufgenommen wird, wenn er die Bedingung $V \geq f(\|x_t\|)$ erfüllt, wobei V die durch x_t erreichte Verbesserung und $f : \mathbb{R} \to \mathbb{R}$ eine reellwertige Funktion ist, die von der euklidische Norm $\|x_t\|$ des Testpunktes abhängt. Dies ist lediglich ein möglicher Modellierungsvorschlag für eine solche Akzeptanzbedingung. Es erscheint aber sinnvoll, dass diese von der Norm des Testpunktes abhängig ist, da ein Punkt, der beispielsweise ein minimale Verbesserung erreicht, nur dann akzeptiert werden sollte, wenn er sehr nahe am Ursprung liegt.
Da die benötigten Rechenzeiten bei Problemen größerer Dimension auch bei Verwendung von verb_mess_fil.m zum Teil sehr hoch sind, sind die Einsatzmöglichkeiten einer Akzeptanzbedingung, die auf dieser Funktion beruht, jedoch begrenzt.

Graphische Darstellung des Filters
Bei zwei- bzw. dreidimensionalen Filtern ist eine graphische Darstellung möglich. Implementiert wurde dies in den Funktionen fplot2d.m und fplot3d.m. Beide Funktionen beruhen darauf, dass der Filter, ähnlich wie bei der Funktion verb_mess_fil.m, in einzelne Abschnitte, die parallel zu einer der Koordinatenachsen - bzw. Koordinatenebenen im dreidimensionalen Fall - liegen, unterteilt wird. Im Unterschied zur Volumenberechnung ist es allerdings erforderlich, nicht nur die Abschnitte, die parallel zu einer bestimmten Achse - bzw. Ebene - des Koordinatensystems liegen zu bestimmen, son-

dern auch diejenigen, die parallel zur zweiten Achse - bzw. zu den beiden anderen Ebenen - liegen. Der Filter kann nun dargestellt werden, indem jeder dieser Abschnitte einzeln abgebildet wird. Beispiele für die Resultate dieser beiden Funktionen sind die Abbildungen 2.3 auf Seite 25 und 3.1 auf Seite 49.
Für die Darstellung eines zweidimensionalen Filters mit Korridor der Mindestverbesserung steht die Funktion plot2d_env_var.m zur Verfügung. Dieser Funktion müssen außer dem Filter auch zwei Parameter übergeben werden. Einerseits γ_θ, durch das die Größe des Korridors verändert werden kann. Diese Parameter wurde in Definition 3.3 eingeführt. Andererseits kann mit Hilfe eines Parameters δ ausgewählt werden, welcher der Ausdrücke (3.4), (3.5) oder (3.6) zur Erzeugung des Korridors verwendet werden soll. Abbildung 3.2 zeigt das Resultat dieser Funktion für die drei unterschiedlichen Varianten zur Bestimmung der Mindestverbesserung. Die bisher vorgestellten Algorithmen dienen hauptsächlich der Handhabung des mehrdimensionalen Filters, im folgenden Kapitel werden nun Einzelheiten der eigentlichen Implementation des FTR-Algorithmus vorgestellt.

4.2 Implementierung des FTR-Algorithmus

Die numerische Umsetzung des Filter-Trust-Region-Algorithmus orientiert sich im Wesentlichen an Algorithmus 3.1. Da dieser allerdings nicht in allen Einzelheiten spezifiziert ist, sollen die nötigen Ergänzungen hier aufgeführt werden. Eine Implementierung des vollständigen Algorithmus stellt die Matlab-Funktion ftr.m (bzw. ftr2.m) dar.

Berechnen des Testpunktes
Eine effiziente Bestimmung eines geeigneten Testpunktes ist entscheidend für die Laufzeit des FTR-Algorithmus. Der Testpunkt wird anhand einer Modellfunktion bestimmt, die die Zielfunktion approximiert. Als Testpunkt kann dann beispielsweise die Minimalstelle der Modellfunktion gewählt werden. Es genügt jedoch schon, einen Punkt zu ermitteln, in dem das Modell einen niedrigeren Wert annimmt als im aktuellen Iterationswert.
In Kapitel 3 wurden bereits das Gauss-Newton-Modell und das Full-Newton-Modell vorgestellt. Zusätzlich zu diesen beiden Modellfunktionen, die durch lineare bzw. quadratische Approximation der Komponenten $c_i(x)$ des Funktionsvektors $c(x)$ des nichtlinearen Gleichungssystems $c(x) = 0$ entstehen, sind in ftr.m zwei weitere Modellfunktionen implementiert. Diese entstehen aus einer linearen bzw. quadratischen Approximation der eigentliche Zielfunktion $f(x) = \frac{1}{2} \|\theta(x)\|^2$, die mit Hilfe der Abweichungen, die ein Iterationswert x in den einzelnen Funktionsgruppen verursacht, definiert wird (vgl. Kapitel 3). Im folgenden Abschnitt werden die Methoden vorgestellt, mit denen bei Verwendung der jeweiligen Modellfunktion ein Testschritt bestimmt werden kann.
Bei Verwendung einer **linearen Approximation der Funktion** $f(x)$ ist der Wert eines Punktes $x_k + s_k$ ausgehend vom aktuellen Iterationswert x_k in der Modellfunktion

gegeben durch

$$m_k^{lin}(x_k + s) = f(x_k) + \nabla f(x_k)\, s^\top.$$

Dabei ist x_k der Punkt in dem die Approximation und damit das Modell gebildet wurde und s_k der Testschritt in der k-ten Iteration. Der Gradient der Funktion f, s_k sowie alle weiteren in diesem Abschnitt auftretenden Vektoren werden als Zeilenvektoren behandelt. Die Approximation des Gradienten sowie die der Hessematrix, die für die quadratischen Modellfunktionen benötigt werden, erfolgt im Algorithmus ftr.m mit Hilfe von finiten Differenzen, d.h. die erforderlichen partiellen Ableitungen werden durch den entsprechenden Differenzenquotienten angenähert.
Zur Ermittlung des Testpunktes im linearen Modell wird zunächst die globale Minimalstelle x_{tr} von $m_k^{lin}(x)$ innerhalb der Trust-Region $\mathcal{B}_k$ bestimmt. Dies geschieht durch Addition des Produkts aus dem normierten negativen Gradienten der Funktion $f(x)$ im Punkt x_k mit dem Trust-Region-Radius Δ_k zu den Koordinaten des aktuellen Iterationspunktes:

$$x_{tr} = x_k - \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}\Delta_k. \tag{4.2}$$

x_{tr} bezeichnet tatsächlich die globale Minimalstelle von $m_k^{lin}(x)$ auf der Trust-Region $\mathcal{B}_k$, da es sich bei der Modellfunktion um eine lineare Funktion mit der konstanten Ableitung

$$\left(m_k^{lin}(x)\right)' = \nabla f(x_k)$$

für alle $x \in \mathcal{B}_k$ handelt und die Trust-Region mit Hilfe der euklidischen Norm gebildet wurde. Deshalb ist der euklidische Abstand aller Randpunkte der Trust-Region zu x_k gleich und das globale Minimum wird in dem Randpunkt angenommen, der auf der Halbgerade ausgehend von x_k entlang des steilsten Abstiegs $-\nabla f(x_k)$ liegt. Die so ermittelte Minimalstelle der Modellfunktion entspricht nach Definition 2.4 dem Cauchy-Punkt.
Ist die Länge des Testschrittes nicht auf die Trust-Region beschränkt, oder ist der Wert der Modellfunktion in x_{tr} kleiner null, so ist es möglich mit sehr geringem zusätzlichen Rechenaufwand einen geeigneteren Kandidaten für den Testpunkt zu ermitteln. Dazu wird ausgehend von x_{tr} mit Hilfe zweier Fallunterscheidungen der tatsächliche Testpunkt x_{lin} bestimmt. Man betrachtet zunächst den Wert der Modellfunktion in x_{tr}. Ist dieser kleiner Null, so wird die Linearität der Modellfunktion ausgenutzt, um den Punkt auf der Strecke zwischen x_k und x_{tr} ausfindig zu machen, in dem der Wert der Modellfunktion gleich 0 ist. Damit ergibt sich

$$x_{lin} = x_{tr} - (x_k - x_{tr})\,\frac{m_k^{lin}(x_{tr})}{m_k^{lin}(x_k) - m_k^{lin}(x_{tr})} \tag{4.3}$$

als nächster Testpunkt. Von diesem kann angenommen werden, dass er näher an der Lösung x^* des nichtlinearen Gleichungssystems liegt als x_{tr}, da für den Wert der Zielfunktion in dieser Lösung $f(x^*) = 0$ gilt. Ist $m_k^{lin}(x_{tr}) < 0$, so ist es unerheblich, ob die Bedingung (3.9) (d.h. $\|s_k\| \leq \Delta_k$) erfüllt sein muss oder nicht, da der Testpunkt von sich aus innerhalb der Trust-Region liegt.

Ist dagegen $m_k^{lin}(x_{tr}) > 0$, so ist $x_{lin} = x_{tr}$, falls (3.9) erfüllt sein muss. Ist die Schrittweite dagegen nicht durch den Trust-Region-Radius beschränkt, so kann der Testpunkt erneut anhand von (4.3) bestimmt werden. Dadurch erhält man wiederum einen Punkt auf der Geraden, welche durch die Punkte x_k und x_{tr} verläuft, in dem der Wert der Modellfunktion gleich 0 ist. In diesem Fall liegt dieser Punkt jedoch außerhalb der Trust-Region.
Gilt $m_k^{lin}(x_{tr}) = 0$, so entspricht der Testpunkt dem ermittelten Cauchy-Punkt: $x_{lin} = x_{tr}$ unabhängig davon, ob Bedingung (3.9) erfüllt sein muss oder nicht.
Im folgenden Algorithmus sind die vorangegangenen Überlegungen zusammengefasst:

Algorithmus 4.6
Die Zielfunktion $f(x)$, der Iterationswert x_k und der Trust-Region-Radius Δ_k sind gegeben.
Berechne mit Hilfe von (4.2) den Wert x_{tr}.
If $m_k^{lin}(x_{tr}) \leq 0$ **then**
    Berechne den Testpunkt x_{lin} anhand von (4.3).
else
    If *RESTRICT="true"* **then**
        Für den Testpunkt gilt $x_{lin} = x_{tr}$.
    else
        Berechne den Testpunkt x_{lin} anhand von (4.3).
    endif
endif

Dabei bezieht sich "RESTRICT" auf die in Algorithmus 3.1 eingeführte boolsche Variable, durch die angezeigt wird, ob ein Testschritt durch den Trust-Region-Radius beschränkt sein muss (RESTRICT="true") oder nicht (RESTRICT="false"). Da man mit Hilfe von (4.3) für $m_k^{lin}(x_{tr}) = 0$ das gewünschte Ergebnis $x_{lin} = x_{tr}$ erhält, muss dieser Fall nicht gesondert betrachtet werden.
Die **quadratische Approximation der Funktion** $f(x)$ entsteht unter Berücksichtigung eines zusätzlichen quadratischen Terms der zweiten Ableitung von $f(x)$, um den das lineare Modell $m_k^{lin}(x)$ erweitert wird. Damit hat die quadratische Modellfunktion die Form:

$$m_k^q(x_k + s) = f(x_k) + \nabla f(x_k)\, s^\top + \frac{1}{2} s H_{f,k} s^\top,$$

wobei mit $H_{f,k}$ die Approximation der Hessematrix der Funktion $f(x)$ im Iterationswert x_k bezeichnet wird.
Zur Bestimmung des Testpunktes werden zunächst zwei verschiedene dafür in Frage kommende Punkte berechnet. Einerseits ermittelt man analog zum linearen Fall den Punkt x_{tr}, der entlang des negativen Gradienten von $f(x)$ ausgehend vom Iterationswert x_k auf dem Rand der Trust-Region liegt, dabei handelt es sich auch bei der quadratischen Modellfunktion um den Cauchy-Punkt. Aus x_{tr} wird analog zum Vorgehen beim linearen Modell $m_k^{lin}(x)$ der Punkt x_{lin} bestimmt, der dort als Testpunkt verwendet wird. Da der Gradient der quadratischen Modellfunktion jedoch nicht konstant ist und nur im Entwicklungspunkt x_k mit $\nabla f(x_k)$ übereinstimmt, ist x_{lin} keine

Null- oder Minimalstelle von $m_k^q(x)$. Allerdings kann man annehmen, dass der Wert von $m_k^q(x)$, falls x_{lin} in einer ausreichend kleinen Umgebung um x_k liegt, in x_{lin} kleiner ist als der in x_k.
Als Alternative zu x_{lin} wird, falls die Schrittweite auf die Trust-Region beschränkt ist, mit Hilfe des Truncated Conjugate Gradient Verfahrens zusätzlich der so genannte Steihaug-Toint-Punkt bestimmt [4]. Hierbei werden solange CG-Schritte ausgeführt, bis das CG-Verfahren konvergiert, der CG-Schritt die Trust-Region verlässt oder man einen Punkt mit negativer Krümmung erreicht hat. In den letzten beiden Fällen wird eine Richtung mit negativer Krümmung bestimmt und der Punkt zurückgegeben, an dem man in dieser Richtung den Rand der Trust-Region schneidet. Der Steihaug-Toint-Punkt stellt eine sehr guten Approximation der Lösung des Minimierungsproblems

$$\begin{array}{c} \min_x q(s) = \nabla f(x_k)\, s^\top + \frac{1}{2} s H_{f,k} s^\top \\ \text{s.t. } \|s\| \leq \Delta_k \end{array}, \tag{4.4}$$

und somit des Punktes an dem die quadratische Modellfunktion innerhalb der Trust-Region den kleinsten Wert annimmt, dar. Ist die Bedingung (3.9) im aktuellen Iterationsschritt nicht gefordert, so ist der Punkt, der (4.4) ohne Berücksichtigung der Nebenbedingung minimiert, eine gute Wahl für den Testpunkt. Es ist demnach nötig, eine Nullstelle der ersten Ableitung von $q(s)$ zu bestimmen. Diese entspricht einer Lösung des Gleichungssystems $sH_{f,k} = -\nabla f(x_k)$, falls $H_{f,k}$ positiv definit ist. In Matlab kann die Lösung s^* eines solchen Gleichungssystems mit Hilfe der Funktion "mldivide" bestimmt werden, die in diesem Fall durch die Eingabe "$s^* = -(H_{f,k} \setminus \nabla f(x_k))$" aufgerufen wird. Die zweite Alternative x_{alt} für den Testpunkt ist somit entweder der Steihaug-Toint-Punkt oder ist gegeben durch $x_k + s^*$.
Als eigentlicher Testpunkt x_q wird derjenige ausgewählt, in dem die Modellfunktion $m_k^q(x)$ den kleineren Wert annimmt. Dies sollte bis auf wenige Ausnahmen in x_{alt} geschehen. Nur falls (3.9) nicht erfüllt sein muss und demnach der "\"-Operator verwendet wird, besteht die Möglichkeit, dass $m_k^{qu}(x_{lin}) < m_k^{qu}(x_q)$ ist, falls die Hessematrix nicht positiv definit bzw. singulär ist, da in diesen Situationen s^* nicht zu einer Minimalstelle der Modellfunktion führt bzw. nicht durch die Matlab-Funktion "mldivide" bestimmt werden kann.
Der Testpunkt im quadratischen Modell ergibt sich also nach den vorangegangenen Überlegungen aus folgendem Algorithmus:

Algorithmus 4.7
Die Zielfunktion $f(x)$, der Iterationswert x_k und der Trust-Region-Radius Δ_k sind gegeben.
Berechne anhand von Algorithmus (4.6) den Punkt x_{lin}.
if *RESTRICT="true"* **then**
  Definiere $p_k := -\nabla f(x_k)$, $s_k := x_k$, $d_k := 0$, $g_k := \nabla f(x_k)$ und $g_{tmp} := \nabla f(x_k)$.
  while $\|g_k\| > \epsilon$ **do**
    Bestimme den Wert $\kappa = p_k H_{f,k} p_k^T$
    if $\kappa \leq 0$ **then**

Berechne $\sigma = \frac{-d_k p_k^T + \sqrt{(d_k p_k^T)^2 + \|p_k\|^2(\Delta_k^2 - \|d_k\|^2)}}{\|p_k\|^2}$.
Setze $s_k = s_k + \sigma p_k$.
STOP (Verlasse die while-Schleife)
endif
Bestimme den Wert $\alpha = \frac{\|g_k\|^2}{\kappa}$.
if $\sqrt{\|d_k - \alpha p_k\|} \geq \Delta_k$ **then**
Berechne $\sigma = \frac{-d_k p_k^T + \sqrt{(d_k p_k^T)^2 + \|p_k\|^2(\Delta_k^2 - \|d_k\|^2)}}{\|p_k\|^2}$.
Setze $s_k = s_k + \sigma p_k$.
STOP (Verlasse die while-Schleife)
endif
Setze $s_k = s_k + \alpha p_k$, $g_k = g_k + \alpha H_{f,k} p_k$, $\beta := \frac{\|g_k\|^2}{\|g_{tmp}\|^2}$, $g_{tmp} = g_k$,
$p_k = -g_k + \beta p_k$ *und* $d_k := s_k - x_k$.
end
Definiere $x_{alt} := s_k$.
else
Bestimme die Lösung s^* *des Gleichungssystems* $sH_{f,k} = -\nabla f(x_k)$ *und definiere* $x_{alt} := x_k + s^*$.
endif
Bestimme denjenigen der Punkte x_{lin} *und* x_{alt}, *in dem die Modellfunktion* $m_k^q(x)$ *den kleineren Wert annimmt und verwende diesen als Testpunkt* x_q.

Das in Algorithmus 4.7 verwendete CG-Verfahren basiert auf der in [4] verwendeten Methode zur Bestimmung des Steihaug-Toint-Punktes, wurde jedoch so verändert, dass die Verwendung eines beliebigen von 0 verschiedenen Startpunktes x_k möglich ist. Das CG-Verfahrens terminiert, sobald ein Punkt ermittelt wurde, in dem die Norm $\|g_k\|$ des Gradienten kleiner als eine Konstante $\epsilon > 0$ ist.
Die für den Konvergenzbeweis des FTR-Verfahrens nötige Mindestverbesserung des Wertes der Modellfunktion ist beim quadratischen Modell durch die Verwendung des Steihaug-Toint-Punktes als Kandidat für den Testpunkt gewährleistet [4].
Die dritte im Algorithmus ftr.m implementierte Modellfunktion ist das **Gauss-Newton-Modell**

$$m_k^{GN}(x_k + s_k) = \frac{1}{2} \sum_{i=1}^{p} \left\| c_{\mathcal{I}_i}(x_k) + J_{\mathcal{I}_i}(x_k)\, s_k^{\top} \right\|^2,$$

das bereits in Kapitel 2 vorgestellt wurde. Um die Koordinaten des Testpunktes zu diesem Modell zu ermitteln, bestimmt man zunächst eine Nullstelle von $m_k^{GN}(x)$. Diese ist, wie bereits in Kapitel 3.2.3 erläutert, durch $x_0 = x_k - J_k^+ c_k$ gegeben. Dabei sei J_k^+ die Pseudoinverse der Jacobimatrix von $c(x)$ im Iterationswert x_k und c_k der Funktionswert von $c(x)$ in x_k. Liegt der Punkt x_0 innerhalb der Trust-Region oder muss die Bedingung (3.9) im aktuellen Iterationsschritt nicht erfüllt sein, so kann x_0 als Testpunkt verwendet werden.
Ist jedoch $\Delta_k < \|x_k - x_0\|$ und muss (3.9) berücksichtigt werden, so erfüllt x_0 die Bedingungen, die an den Testpunkt gestellt werden, nicht. In diesem Fall werden zwei

alternative Punkte bestimmt. Einerseits der Punkt x_1, in dem die Strecke zwischen x_k und x_0 den Rand der Trust-Region um x_k schneidet. Liegt der Punkt x_0 in einer ausreichend kleinen Umgebung um den Rand der Trust-Region, so ist zu erwarten, dass der Wert der Modellfunktion in x_1 kleiner ist als der in x_k und x_1 kann als Testpunkt verwendet werden.
Als zweiten Punkt betrachtet man wieder den Punkt, der sich als Schnittpunkt der Halbgeraden ausgehen von x_k in Richtung des negativen Gradienten mit dem Rand der Trust-Region ergibt. Nun wird das Minimum der Funktionswerte von $m_k^{GN}(x)$ in diesen beiden Punkten ermittelt und der zugehörige Wert der Variable x als Testpunkt verwendet.
In algorithmischer Darstellung lässt sich das Vorgehen wie folgt beschreiben:

Algorithmus 4.8
Das nichtlineare Gleichungssystem $c(x) = 0$, der Iterationswert x_k und der Trust-Region-Radius Δ_k sind gegeben.
Bestimme die Jacobimatrix J_k der Funktion $c(x)$ in x_k und die Nullstelle $x_0 = x_k - J_k^+ c_k$ der Modellfunktion $m_k^{GN}(x)$.
if *RESTRICT="false"* **then**
  Für den Testpunkt x_{GN} gilt: $x_{GN} := x_0$.
else
  if $\Delta_k \geq \|x_k - x_0\|$ **then**
    Für den Testpunkt x_{GN} gilt: $x_{GN} := x_0$.
  else
    Bestimme den Schnittpunkt $x_1 := x_k + \frac{(x_0 - x_k)\Delta_k}{\|x_0 - x_k\|}$ des Randes der Trust-Region mit der Strecke $\overline{x_k x_0}$.
    Bestimme den Punkt $x_2 := x_k - \Delta_k \frac{\nabla m_k^{GN}(x)}{\nabla \|m_k^{GN}(x)\|}$.
    Verwende denjenigen der Punkte x_1 und x_2 als Testpunkt, in dem die Modellfunktion $m_k^{GN}(x)$ den kleineren Wert annimmt.
  endif
endif

Die vierte im Algorithmus ftr.m enthaltene Modellfunktion ist das ebenfalls bereits in Kapitel 2 vorgestellte **Full-Newton-Modell**

$$m_k^N(x_k + s) = m_k^{GN}(x_k + s) + \frac{1}{2} \sum_{i=1}^{p} \sum_{j \in \mathcal{I}_i} c_j(x_k) \left\langle s, \nabla^2 c_j s^\top \right\rangle .$$

Die Berechnung des Testpunktes in dieser Modellfunktion geschieht analog zur Vorgehensweise beim quadratischen Modell der Zielfunktion. Auch dabei werden zwei Kandidaten für den Testpunkt bestimmt. Zunächst berechnet man den Steihaug-Toint-Punkt mit Hilfe des in Algorithmus (4.7) beschriebenen CG-Verfahrens, falls die Bedingung (3.9) erfüllt sein muss. Dazu ist es notwendig, die Modellfunktion $m_k^N(x)$ umzuformulieren, da sie im Gegensatz zu dem quadratischen Modell der Zielfunktion

nicht in der Form einer quadratischen Gleichung $q(x) = xAx^\top + bx^\top + c$ mit einer Matrix A, einem Vektor b und einer Konstanten c vorliegt. Da die Kenntnis von A und b für die Anwendung des CG-Verfahrens unerlässlich ist, formt man die Modellfunktion $m_k^N(x)$ folgendermaßen um:

$$\begin{aligned}
m_k^N(x_k+s) &= m_k^{GN}(x_k+s) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i} c_j(x_k)\left\langle s, H_{c_j}(x_k)\,s^\top\right\rangle \\
&= \frac{1}{2}\sum_{i=1}^{p}\left\|c_{\mathcal{I}_i}(x_k) + J_{\mathcal{I}_i}(x_k)\,s^\top\right\|^2 + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i} c_j(x_k)\left\langle s, H_{c_j}(x_k)\,s^\top\right\rangle \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(c_j(x_k) + g_{c_j}(x_k)\,s^\top\right)^2 + \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(sc_j(x_k)\,H_{c_j}(x_k)\,s^\top\right) \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\Big(c_j^2(x_k) + 2c_j(x_k)\,g_{c_j}(x_k)\,s^\top + sg_{c_j}^\top(x_k)\,g_{c_j}(x_k)\,s^\top \\
&\quad + sc_j(x_k)\,H_{c_j}(x_k)\,s^\top\Big) \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\Big(s\left(c_j(x_k)\,H_{c_j}(x_k) + g_{c_j}^\top(x_k)\,g_{c_j}(x_k)\right)s^\top \\
&\quad + 2c_j(x_k)\,g_{c_j}(x_k)\,s^\top + c_j^2(x_k)\Big).
\end{aligned}$$

Dabei bezeichnet $H_{c_j}(x_k)$ bzw. $g_{c_j}(x_k)$ die Hessematrix bzw. den Gradienten der j-ten Komponente des Funktionsvektors $c(x)$ in x_k. Mit

$$A := \sum_{i=1}^{p}\sum_{j\in\mathcal{I}_i}\left(c_j(x_k)\,H_{c_j}(x_k) + g_{c_j}^\top(x_k)\,g_{c_j}(x_k)\right) \tag{4.5}$$

und

$$b := 2c_j(x_k)\,g_{c_j}(x_k) \tag{4.6}$$

lässt sich das CG-Verfahren zur Bestimmung des Steihaug-Toint-Punktes anwenden, falls die Schrittweite durch den Trust-Region-Radius beschränkt ist, also Bedingung (3.9) in Iterationsschritt k erfüllt sein muss. Andernfalls wird wiederum analog zum Vorgehen beim quadratischen Modell mit Hilfe der Matlab-Funktion "mldivide" ein stationärer Punkt der Modellfunktion, d.h. eine Lösung s^* des Gleichungssystems $sA = -b$, bestimmt.
Um für die Situationen, in denen mit Hilfe des Matlab-Operators "mldivide" kein geeigneter Testpunkt gefunden werden kann, eine Alternative zur Verfügung zu haben, berechnet man als zweiten Kandidaten für den Testpunkt den Punkt x_{GN} analog zu Algorithmus 4.8. Als Testpunkt wird schließlich derjenige verwendet, in dem die Funktion $m_k^N(x)$ den niedrigeren Wert annimmt.
Insgesamt erhält man zur Berechnung des Testpunktes im Full-Newton-Modell folgenden Algorithmus:

Algorithmus 4.9
Das nichtlineare Gleichungssystem $c(x) = 0$, der Iterationswert x_k und der Trust-Region-Radius Δ_k sind gegeben.
Berechne x_{GN} anhand von Algorithmus 4.8.
Bestimme mit Hilfe von (4.5) und (4.6) die Matrix A und den Vektor b.
if *RESTRICT="true"* **then**
    Ermittle analog zu Algorithmus 4.7 den Steihaug-Toint-Punkt s_k.
    Definiere $x_{alt} := s_k$.
else
    Bestimme die Lösung s^* des Gleichungssystems $sA = -b$ und definiere $x_{alt} := x_k + s^*$.
endif
Bestimme denjenigen der Punkte x_{GN} und x_{alt}, in dem die Modellfunktion $m_k^N(x)$ den kleineren Wert annimmt und verwende diesen als Testpunkt x_N.

Die Unterschiede der vier Modellfunktionen lassen sich im Wesentlichen in zwei Punkten zusammenfassen: Betrachtet man die Modellierung der Modelle, so sieht man, dass das Gauss-Newton- und das Full-Newton-Modell aus Approximationen der Funktionen $c_i(x)$ des nichtlinearen Gleichungssystems gebildet wurden. Das lineare und das quadratische Modell stellen dagegen Approximationen der eigentlichen Zielfunktion $f(x)$ dar. Das bewirkt einerseits einen Unterschied bezüglich der benötigten Rechenzeit, so muss z.B. der Gradient bzw. die Hessematrix für die Modellfunktionen $m_k^{lin}(x)$ und $m_k^q(x)$ lediglich für eine reellwertige Funktion $f(x)$ bestimmt werden. Bei $m_k^{GN}(x)$ und $m_k^N(x)$ ist dies für alle Funktionen $c_i(x)$ erforderlich. Andererseits sollte die Funktion $f(x)$, die eigentlich aus einer Verknüpfung von Funktionen $f(c(x))$ besteht, im Allgemeinen genauer durch die Approximationen von $c(x)$ beschrieben werden, als durch die $f(x)$ approximierenden Modelle.
Der zweite Unterschied bezieht sich zunächst ebenfalls auf die Genauigkeit der Modellfunktionen. Die quadratischen Approximationen sollten den Verlauf der Zielfunktion im Allgemeinen besser beschreiben als die entsprechenden linearen Alternativen. Allerdings ist der Aufwand zur Bestimmung eines Testpunktes dort auch von einer höheren Ordnung. Aus diesem Grund ist es möglich, in den Algorithmen 4.7 und 4.9 die zusätzliche Berechnung der Testpunkte der jeweiligen linearen Modellfunktion durchzuführen, ohne dass sich dadurch die Größenordnung des Rechenaufwands ändert. Andererseits erreicht man durch die Berechnung mehrerer Alternativen für einen Testpunkt eventuell eine schnellere Konvergenz des Verfahrens, da die Modellfunktion in den zusätzlichen Kandidaten für den Testpunkt durchaus bessere Werte annehmen kann.
Die im anschließenden Kapitel vorgestellten numerischen Tests zeigen wie sich die Unterschiede der vier Modelle auf verschiedene Beispielprobleme auswirken.

Aktualisierung des Trust-Region-Radius
In Schritt 6 von Algorithmus 3.1 werden in Abhängigkeit von der Genauigkeit ρ_k der Modellfunktion lediglich Wertebereiche angegeben, in denen der Trust-Region-Radius

Δ_{k+1} für den nächsten Iterationsschritt liegen soll:

$$\Delta_{k+1} \in \begin{cases} [\gamma_0\Delta_k, \gamma_1\Delta_k] & falls \quad \rho_k < \eta_1 \\ [\gamma_1\Delta_k, \Delta_k] & falls \quad \rho_k \in [\eta_1, \eta_2) \\ [\Delta_k, \gamma_2\Delta_k] & falls \quad \rho_k \geq \eta_2. \end{cases}$$

Dabei gilt für die Parameter: $0 < \gamma_0 \leq \gamma_1 < 1 \leq \gamma_2$ sowie $0 < \eta_1 < \eta_2 < 1$. Für eine explizite Implementierung des Filter-Trust-Region-Algorithmus muss allerdings aus jedem dieser Intervalle ein Wert ausgewählt werden, den der Trust-Region-Radius für den jeweiligen Fall in der nächsten Iteration annehmen soll. Dazu stehen im Algorithmus ftr.m zwei verschiedene Möglichkeiten zur Verfügung.
Ein naheliegender Ansatz Δ_{k+1} zu bestimmen, entsteht durch Multiplikation des alten Radius Δ_k mit einer Konstanten, die im Fall einer sehr guten Vorhersage durch die Modellfunktion größer als 1 ist, im Fall einer ausreichend genauen Vorhersage gleich 1 und sonst kleiner als 1 ist. Hierfür wählt man diese Konstanten als γ_1 und γ_2 und erhält folgende Aktualisierungsvorschrift:

$$\Delta^1_{k+1} = \begin{cases} \gamma_1 \cdot \Delta_k & falls \quad \rho_k < \eta_1 \\ \Delta_k & falls \quad \rho_k \in [\eta_1, \eta_2) \\ \gamma_2 \cdot \Delta_k & falls \quad \rho_k \geq \eta_2 \end{cases}$$

Dabei gilt für die Parameter analog zu der in Algorithmus 3.1 angegebenen Aktualisierungsvorschrift: $0 < \gamma_1 < 1 < \gamma_2$. In Anlehnung an das Vorgehen in [4] könnte man beispielsweise die Werte $\gamma_1 = 0.25$ und $\gamma_2 = 2.5$ verwenden.
Eine weitere Möglichkeit, den Trust-Region-Radius zu aktualisieren, berücksichtigt zusätzlich die Länge des Testschrittes s_k. Dies erscheint sinnvoll, da die Genauigkeit der Modellfunktion weniger von Δ_k, das nur die obere Schranke der Länge des Testschritts darstellt, als vielmehr von der eigentlichen Länge des Testschrittes $\|s_k\|$ abhängt. Numerische Versuche in Kapitel 5 werden diese anschauliche Überlegung überprüfen und zeigen, ob der folgende Aktualisierungsschritt tatsächlich schneller zum Ziel führt als der obige Ansatz:

$$\Delta^2_{k+1} = \begin{cases} \max\left(\gamma_1 \|s_k\|, \Delta_k\right) & falls \quad \rho_k < \eta_1 \\ \Delta_k & falls \quad \rho_k \in [\eta_1, \eta_2) \\ \gamma_2 \|s_k\| & falls \quad \rho_k \geq \eta_2 \end{cases}$$

Für γ_1 und γ_2 gelten auch hier die Einschränkungen: $0 < \gamma_1 < 1 < \gamma_2$. Vergleicht man die Werte von Δ^1_{k+1} und von Δ^2_{k+1} für gleiche Werte der Parameter γ_1, γ_2, η_1 und η_2 sowie bei gleichen Werten für Δ_k und ρ_k in der vorhergehenden Iteration, so sieht man, dass für einen beliebigen Testschritt s_k die Ungleichung $\Delta^1_{k+1} \geq \Delta^2_{k+1}$ gilt. Im folgenden Kapitel werden anhand einiger Parameterkombinationen diese beiden Aktualisierungsvorschriften in numerischen Tests miteinander verglichen.

Initialisierung und Ausgabe des FTR-Algorithmus
Alle für den Algorithmus nötigen Parameter sind im ersten Schritt von Algorithmus

3.1 zusammengefasst. Da die Art und Weise der Eingabe jedoch nicht bei allen offensichtlich ist, sollen die folgenden Anmerkungen am Beispiel des Algorithmus ftr.m zeigen, wie diese realisiert werden kann. Ein Aufruf von ftr.m aus dem Matlab Command Window heraus hat die Form

$$\begin{aligned}[\text{q,r,a}] \;=\; &\text{ftr(Funktionsvektor,Gruppeneinteilung,Startpunkt,}\Delta_0\text{,update,}\gamma_\theta\text{,}\\ &\eta_1,\eta_2,\gamma_1,\gamma_2\text{,Modellwahl).}\end{aligned}$$

Aufgrund der Vorgehensweise in den beiden implementierten Möglichkeiten, den Trust-Region-Radius zu aktualisieren kann auf den in Algorithmus 3.1 angegebenen Parameter γ_0 verzichtet werden. Für die verwendeten Parameter γ_θ, η_1, η_2, γ_1, γ_2 sollte die Beschränkungen, die im ersten Schritt von Algorithmus 3.1 angegeben sind, beachtet werden. Das bedeutet für die Parameter, welche die Größe des Trust-Region-Radius in der nächsten Iteration festlegen: $0 < \gamma_1 < 1 \leq \gamma_2$, wobei in den meisten Fällen die Wahl $\gamma_2 > 1$ sinnvoll ist.
Durch die beiden Parameter η_1 und η_2 können die Grenzen der drei Fälle des Aktualisierungsschrittes des Trust-Region-Radius festgelegt werden. Sie werden aus dem Intervall $[0, 1]$ gewählt, wobei $\eta_1 < \eta_2$ gilt.
Der Wert γ_θ beeinflusst die Größe des Korridors um den Filter, in dem die Punkte liegen, die zwar von keinem Filtereintrag dominiert werden, aber auch keine ausreichende Verbesserung erzielen und somit nicht vom Filter akzeptiert werden. γ_θ sollte im Intervall $(0, \frac{1}{\sqrt{p}})$ liegen, um die Konvergenz der Verfahrens zu gewährleisten, wobei $p \in \mathbb{N}$ die Anzahl der Gruppen bezeichnet, in welche die Funktionen des Gleichungssystems eingeteilt werden.
Diese fünf Parameter müssen nicht beim Funktionsaufruf spezifiziert werden; es können auch Standardwerte, die im Algorithmus festgelegt sind, verwendet werden. Dazu übergibt man beim Aufrufen der Funktion ftr.m an der Stelle des betreffenden Parameters den Wert 0. Die Werte, auf die der Algorithmus in diesem Fall zurückgreift, sind: $\gamma_\theta = 10^{-5}$, $\eta_1 = 0.1$, $\eta_2 = 0.9$, $\gamma_1 = 0.25$ und $\gamma_2 = 2.5$.
Natürlich ist es von Vorteil, einen Startpunkt angeben zu können, der in der Nähe der (vermuteten) Lösung liegt, da die Zeit und die Anzahl der Iterationsschritte, die das Verfahren benötigt, um eine Lösung des Gleichungssystems zu ermitteln, sehr von der Lage des Startpunktes abhängt. Sollte allerdings kein geeigneter Startpunkt bekannt sein, so ist es möglich, stattdessen lediglich dessen Dimension, d.h. die Dimension der vektorwertigen Variable bzw. die Anzahl der reellen Variablen, anzugeben. In diesem Fall verwendet der Algorithmus den Nullvektor als Startpunkt.
Ähnlich wird auch mit dem anfänglichen Trust-Region-Radius verfahren. Er kann entweder spezifiziert oder mit dem Wert 0 übergeben werden. Im zweiten Fall verwendet der Algorithmus $\Delta_0 = 1$.
Mit Hilfe der Parameter "update" bzw. "Modellwahl" kann man auswählen, in welcher Weise der Trust-Region-Radius aktualisiert bzw. welche Modellfunktion zur Bestimmung des Testschrittes verwendet werden soll. Zulässige Werte für "update" sind 1 und 2. Für "update"=1 wird der im vorherigen Abschnitt beschriebene Wert Δ^2_{k+1} verwendet. Ist "update"=2, wird damit die Alternative Δ^1_{k+1} gewählt. Der Wert des

Parameters "Modellwahl" kann aus der Menge $\{1, 2, 3, 4\}$ gewählt werden. Dabei entspricht der Wert 1 dem linearen Modell der Zielfunktion, 2 dem Gauss-Newton-Modell, 3 dem Full-Newton-Modell und 4 dem quadratische Modell der Zielfunktion.
Das nichtlineare Gleichungssystem wird dem Algorithmus als n-dimensionaler Funktionsvektor übergeben. Dabei können die Funktionen auf zwei unterschiedliche Arten vorliegen. Zum einen wird der in Matlab vorhandene Datentyp "function-handle" unterstützt. Soll dieser verwendetet werden, so wird der Parameter "Funktionsvektor" im Funktionsaufruf als Array, in dessen Zellen die einzelnen Funktionen als "function-handle" stehen, übergeben. Alternativ können die Funktionen auch als Matlab-Funktionsdateien vorliegen. In diesem Fall muss jedoch die Funktion ftr1.m verwendet werden, die bis auf diesen Unterschied bei der Initialisierung völlig mit ftr.m identisch ist. Ein Beispiel für die Eingabe von Funktionen als Dateien ist im einleitenden Kommentar der Datei ftr1.m beschrieben.
Der letzte Parameter des Funktionsaufrufs ist die "Gruppeneinteilung". Dieser gibt an, wie die in [12] erläuterte Einteilung der Zeilen des nichtlinearen Gleichungssystems bzw. der Komponenten des n-dimensionalen Funktionsvektors vorgenommen werden soll. "Gruppeneinteilung" wird in Form einer Matrix übergeben, deren Einträge natürliche Zahlen $a_{i,j}$ mit $a_{i,j} \in \{1, ..., n\}$ sind. Ein Eintrag mit dem Wert k verweist auf die k-te Komponente des Funktionsvektors. Jeder Zeilenvektor der Matrix "Gruppeneinteilung" erzeugt dann eine Gruppe von Funktionen. Somit kann anhand der Anzahl der Zeilen dieser Matrix die Anzahl der Gruppen abgelesen werden und anhand der Anzahl der Spalten ist ersichtlich, aus wie vielen Funktionen eine Gruppe besteht. Übergibt man im Funktionsaufruf als Wert des Parameters "Gruppeneinteilung" statt einer Matrix die Zahl 1, so erzeugt der Algorithmus für jede Funktion eine eigene Gruppe. Dies ist nach [12] in vielen Fällen eine gute Wahl. Außerdem kann der Parameter "Gruppeneinteilung" mit dem Wert 2 übergeben werden. In diesem Fall werden alle Funktionen zu einer einzigen Gruppe zusammengefasst.
Die dreigeteilte Ausgabe der Funktion ftr.m besteht aus dem Punkt r in dem der Algorithmus terminiert ist, einem Vektor q, in dem die Funktionswerte $c(r)$ gespeichert sind, die der Punkt r in den einzelnen Komponenten des Funktionsvektors verursacht und einer Matrix a, in deren Zeilen für jeden Iterationsschritt relevante Größen gespeichert sind, die den Ablauf des Algorithmus beschreiben. Diese Größen sind für den k-ten Iterationsschritt der Wert der Variable "RESTRICT", die Koordinaten des Iterationspunktes, eine boolsche Variable, die genau dann wahr ist, wenn der Testpunkt akzeptiert wurde, der Fehlervektor $\theta(x_k)$, eine boolsche Variable die genau dann wahr ist, wenn $\theta(x_k)$ in den Filter aufgenommen wurde, die Funktionswerte $c_i(x_k)$, der Wert der Zielfunktion, der Trust-Region-Radius Δ_k und der Wert der Variable ρ_k.
Bei der in diesem Kapitel vorgestellten Implementierung des FTR-Algorithmus sind an entscheidenden Stellen mehrere Alternativen auswählbar, so gibt es beispielsweise mehrere Möglichkeiten den Testschritt bzw. die Modellfunktion zu bestimmen. Außerdem wurden für verschiedene Parameter der Funktion ftr.m bisher nur Definitionsbereiche angegeben. Im anschließenden Kapitel werden die unterschiedlichen Varianten des Algorithmus anhand von numerischen Tests gegenübergestellt sowie verschiedene Parametereinstellungen miteinander verglichen.

Kapitel 5

Numerische Ergebnisse

In diesem Kapitel werden die Ergebnisse einer Reihe von numerischen Tests vorgestellt, mit deren Hilfe verschiedene Parametereinstellungen bzw. Varianten der in Kapitel 4 dargestellten Implementation des FTR-Algorithmus miteinander verglichen werden. Dazu wird eine Menge bestehend aus 54 Testbeispielen, die in Anhang B genauer beschrieben sind, betrachtet.

5.1 Modellfunktionen

Für den Vergleich der Modellfunktionen werden für die übrigen Parameter die bereits erwähnten Standardwerte $\gamma_\theta = 10^{-5}$, $\eta_1 = 0.1$, $\eta_2 = 0.9$, $\gamma_1 = 0.25$ und $\gamma_2 = 2.5$ verwendet. Der Trust-Region-Radius wird nach der Aktualisierungsvorschrift für Δ_{k+1}^2 erneuert und jede Zeile des zu lösenden Gleichungssystems bildet eine eigene Gruppe. Die folgende Tabelle stellt eine zusammenfassende Übersicht dar, in der einige relevante Werte aufgeführt sind:

Modell-funktion	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen			durchschnittlich benötigte	
			Zeit	Iterationen	Fehler	Zeit	Iterationen
Gauss-Newton	38	33	3	4	9	256.07	263.39
linear	15	2	2	27	10	313.31	2434.5
Full-Newton	32	5	0	0	22	18.612	25.094
quadratisch	37	7	3	2	12	318.93	59.797

Tabelle 5.1: Vergleich der Modellfunktionen

Die ausführlichen Ergebnisse dieses Tests befinden sich in Anhang C. Ein Testbeispiel gilt in dieser wie auch in allen weiteren Untersuchungen dieses Kapitels als gelöst, falls der Algorithmus terminiert, da ein Iterationswert x^* gefunden wurde, für den

entweder

$$\|\nabla f(x^*)\| < \epsilon_1$$

oder

$$\max_{j\in\{1,\ldots,p\}} (\theta_j(x^*)) < \epsilon_2$$

gilt. Dabei besagt die erste dieser Bedingungen, dass die Norm des Gradienten der Zielfunktion $\|\nabla f(x)\|$ in der vermeintlichen Lösung x^* kleiner als eine Konstante $\epsilon_1 > 0$ ist. Im Algorithmus ftr.m wird für diese Schranke der Wert $\epsilon_1 = 10^{-8}$ verwendet. In der zweiten Abbruchbedingung wird das Maximum der Fehler $\theta_j(x^*)$, den x^* in der j-ten Gruppe von Funktionen verursacht, ermittelt. Ist dieses, und somit auch alle übrigen $\theta_i(x^*)$ mit $j \neq i \in \{1, ..., n\}$ kleiner als eine Konstante $\epsilon_2 > 0$, so kann das Testbeispiel ebenfalls als gelöst angesehen werden. In ftr.m gilt auch für diese Schranke der Wert $\epsilon_2 = 10^{-8}$.
Darüber hinaus sind zwei weitere Abbruchbedingungen implementiert, die es ermöglichen, auch bei Problemen, die der Algorithmus innerhalb einer festgesetzten Zeit oder nach einer gewissen Anzahl von Iterationen nicht lösen kann, den besten bisher ermittelten Wert der Variable x auszugeben. Zu diesem Zweck wird die Berechnung abgebrochen, falls nach n Iterationsschritten bzw. m verstrichenen Sekunden noch keine geeignete Lösung gefunden wurde. Beim Vergleich der Modellfunktionen werden die Werte $n = 10000$ und $m = 21600$, dies entspricht einer Rechenzeit von 6 Stunden, verwendet. In Tabelle 5.1, sowie in allen weiteren Tabellen in diesem Kapitel wird die Anzahl der auf diese Weise beendeten Berechnungen in der Spalte "Abbruch wegen Iterationen" bzw. "Abbruch wegen Zeit" für die jeweilige Parametereinstellung festgehalten.
Im FTR-Algorithmus wird der Wert

$$\rho_k = \frac{f(x_k) - f\left(x_k^+\right)}{m_k(x_k) - m_k\left(x_k^+\right)}$$

ermittelt, um die Genauigkeit der Modellfunktion zu beschreiben. Dabei ist x_k der Wert der Variable x in der k-ten Iteration, x_k^+ der Testpunkt, $f(x)$ die Zielfunktion und $m_k(x)$ die Modellfunktion. Ändert sich der Funktionswert der Modellfunktion nicht, so kann ρ_k nicht berechnet werden, da sich dann für den Nenner des obigen Quotienten der Wert 0 ergibt. Aus diesem Grund wurde eine fünfte Abbruchbedingung in ftr.m implementiert, welche die Berechnung in einer solchen Situation abbricht, bevor dieser Fehler auftritt. Die Anzahl der durch diese Abbruchbedingung beendeten Berechnungen wird in den tabellarischen Übersichten in diesem Kapitel in der Spalte "Abbruch wegen Fehler" festgehalten.
Die Werte in der dritten Spalte Tabelle 5.1 beziffern für die Modellfunktion m_k die Anzahl der Testbeispiele, die mit keiner Modellfunktion schneller als mit m_k gelöst werden können. In den nächsten drei Spalten wird die Anzahl der Testbeispiele festgehalten, für die der Algorithmus durch eine der drei Abbruchbedingungen, die implizieren, dass keine geeignete Lösung für das Problem gefunden wurde, beendet wurde. In den letzten beiden Spalten kann die durchschnittlich benötigte Rechenzeit in Sekunden sowie die durchschnittliche Anzahl der benötigten Iterationen abgelesen werden.

Bei der Berechnung dieser letzten beiden Werte wurden nur die mit der jeweiligen Modellfunktion gelösten Testbeispiele betrachtet. Das bedeutet, dass für die Modellfunktionen der Durchschnitt über unterschiedliche Teilmengen der Testbeispiele gebildet wurde.
Aus Tabelle 5.1 scheint eindeutig hervorzugehen, dass das Gauss-Newton-Modell für die meisten verwendeten Testbeispiele die beste Wahl ist. Allerdings bleiben für jede Modellfunktion eine Reihe von Probleme ungelöst. Aus Anhang C, in dem die Testergebnisse ausführlich dargestellt sind, ergibt sich andererseits, dass nur eines der Probleme überhaupt nicht gelöst werden kann. Daher liegt der Schluss nahe, dass die Wahl der Modellfunktion problemabhängig erfolgen sollte, da zu beinahe jedem Problem eine Lösung gefunden werden kann, falls die richtige Modellfunktion verwendet wird. Es gibt jedoch keine Modellfunktion, mit der alle Testbeispiele gelöst werden können.
Betrachtet man die letzte Spalte der Tabelle, so bestätigt sich die Vermutung, dass die Zielfunktion in der Regel besser durch die Modellfunktionen, die aus quadratischen Approximationen hervorgehen, angenähert wird. Denn die Anzahl der durchschnittlich benötigten Iterationsschritte, die das Full-Newton-Modell bzw. das quadratischen Modell benötigen, um eine Lösung zu ermitteln, ist deutlich geringer als bei den linearen Varianten. Auffällig ist außerdem, dass das Full-Newton-Modell, falls es ein Testbeispiel löst, dafür durchschnittlich sehr wenig Rechenzeit benötigt. Der Nachteil dieser Modellfunktion hingegen ist die sehr hohe Anzahl der Probleme, bei denen bei der Berechnung von ρ_k der Nenner den Wert 0 annimmt.
Eine Möglichkeit verschiedene Algorithmen bzw. Varianten eines Algorithmus anschaulich miteinander zu vergleichen, sind Leistungsprofile (engl.: performance profiles) [6]. Um ein solches zu erzeugen, wird für jedes berücksichtigte Testbeispiel p die Zeit $t_{p,s}$ ermittelt, welche die Algorithmusvariante s benötigt, um es zu lösen. Weiterhin wird für jedes Paar (p, s) die relative Leistung (engl.: performance ratio)

$$r(p,s) = \frac{t_{p,s}}{\min_{1 \leq s \leq n_s} \{t_{p,s}\}}$$

ermittelt, wobei n_s die Anzahl der zu vergleichenden Algorithmusvarianten ist. Um eine Aussage über die Leistung von s auf der Menge P der Testbeispiele zu erhalten, betrachtet man die Funktion

$$\phi_s(\tau) := \frac{1}{n_p} \text{size}\left(\{p \in P : r(p,s) \leq \tau\}\right),$$

wobei n_p die Anzahl der Testbeispiele bezeichnet und die Funktion $\text{size}\,(M)$ die Anzahl der Elemente einer Menge M bestimmt. In einem Leistungsprofil werden die Graphen $\phi_s(\tau)$ für verschieden Werte von s abgebildet und können auf diese Weise verglichen werden.
In Abbildung 5.1 ist ein Leistungsprofil für die vier implementierten Modellfunktionen dargestellt. Dabei werden die bereits anhand von Tabelle 5.1 angestellten Betrachtungen bestätigt. Man sieht, dass mit Hilfe des Gauss-Newton-Modells deutlich mehr Testbeispiele am schnellsten gelöst werden als unter Verwendung der anderen

Modellfunktionen. Außerdem ist ersichtlich, dass durch das quadratische Modell oder durch das Full-Newton-Modell beinahe so viele Problem wie durch das Gauss-Newton-Modells gelöst werden können, wenn auch nach durchschnittlich längerer Rechenzeit. In [12] wird ein Vergleich zweier Varianten des FTR-Algorithmus durchgeführt, wobei die erste das Gauss-Newton-Modell und die zweite das Full-Newton-Modell zur Testschrittberechnung verwendet. Die dort dargestellten Ergebnisse decken sich mit den hier festgestellten Vorzügen des Gauss-Newton-Modells. Die geringe Anzahl der Probleme, die mit Hilfe des linearen Modells gelöst werden können, ist nicht darauf zurückzuführen, dass dies mit diesem Modell prinzipiell nicht möglich wäre, sondern darauf, dass die Anzahl der benötigten Iterationsschritte, wie aus Tabelle 5.1 zu ersehen, oft größer ist als die maximale Anzahl von Iterationen, nach denen die Berechnung abgebrochen wird.

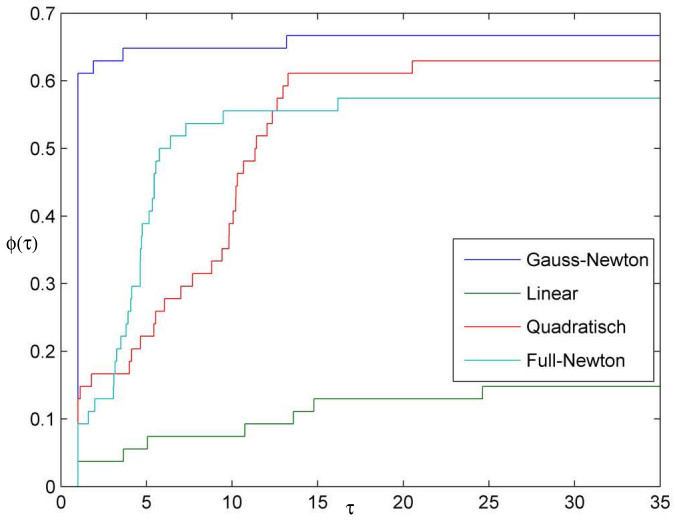


Abbildung 5.1: Leistungsprofil der Modellfunktionen

5.2 Aktualisierung des Trust-Region-Radius

Für alle weiteren Parametertests in diesem Kapitel wird die Menge der Testbeispiele verändert, um die, verglichen mit den vier Modellfunktionen, größere Anzahl der Parametereinstellungen in möglichst kurzer Rechenzeit an ausreichend vielen Testbeispielen erproben zu können. Dazu wird zunächst jede Kombination aus den ursprünglichen 54 Testbeispielen und den vier Modellfunktionen betrachtet. Aus den sich daraus ergebenden Testbeispielberechnungen werden 117 ausgewählt, da aufgrund von Rechenzeiteinsparungen nicht alle Kombinationen von s und p berücksichtigt werden können. Die folgenden Test zu den unterschiedlichen Parametereinstellungen werden anhand dieser neuen Testmenge, deren Elemente in der Tabelle in Anhang B gekennzeichnet sind, durchgeführt.

Die Werte aller Parameter, die bei einem Test nicht variiert werden, sind wie zu Beginn diese Kapitels beschrieben, gewählt (d.h. $\gamma_\theta = 10^{-5}$, $\eta_1 = 0.1$, $\eta_2 = 0.9$, $\gamma_1 = 0.25$, $\gamma_2 = 2.5$, $\Delta_{k+1} = \Delta_{k+1}^2$ und Unterteilen der n Zeilen des jeweiligen Gleichungssystems in n Gruppen). Wie schon zuvor beim Vergleich der Modellfunktionen wird das Ergebnis jedes Parametertests zunächst tabellarisch dargestellt. Anschließend wird für die dem Anschein nach besten Parametereinstellungen ein Leistungsprofil erstellt, da es zu unübersichtlich wäre alle getesteten Varianten zu berücksichtigen.
Zunächst werden die Parameter η_1 und η_2 untersucht. Durch sie wird festgelegt, welcher der drei Fälle, in Abhängigkeit von der Genauigkeit ρ_k der Modellfunktion, für die Aktualisierung des Trust-Region-Radius angewandt wird. In Algorithmus 3.1 wird die Bedingung $0 < \eta_1 < \eta_2 < 1$ für diese Parameter gefordert. In einem diesbezüglichen Parametertest wurden 25 Kombinationen von Werten der beiden Parameter gegenübergestellt. Dieser ergab zusammengefasst folgende Ergebnisse, wobei die mit $\star$ versehenen Parametereinstellungen zusätzlich in einem Leistungsprofil verglichen werden:

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen Zeit	Abbruch wegen Iterationen	Abbruch wegen Fehler	durchschnittlich benötigte Zeit	durchschnittlich benötigte Iterationen
$\eta_1 = 0.01$, $\eta_2 = 0.1$ $\star$	109	32	0	6	2	5.1277	237.46
$\eta_1 = 0.01$, $\eta_2 = 0.2$	108	22	0	7	2	5.9738	275.74
$\eta_1 = 0.01$, $\eta_2 = 0.5$	107	20	0	8	2	5.4235	248.47
$\eta_1 = 0.01$, $\eta_2 = 0.75$	108	23	0	7	2	6.2091	295.19
$\eta_1 = 0.01$, $\eta_2 = 0.9$	109	18	0	6	2	6.6639	311.98
$\eta_1 = 0.01$, $\eta_2 = 0.95$	108	22	0	7	2	5.1615	220.05
$\eta_1 = 0.1$, $\eta_2 = 0.2$	108	13	0	7	2	5.0615	224.86
$\eta_1 = 0.1$, $\eta_2 = 0.5$	108	39	0	7	2	6.3888	281.97
$\eta_1 = 0.1$, $\eta_2 = 0.75$	108	35	0	7	2	5.5088	245.75
$\eta_1 = 0.1$, $\eta_2 = 0.9$	108	29	0	7	2	5.2393	206.68
$\eta_1 = 0.1$, $\eta_2 = 0.95$	108	35	0	7	2	4.5204	176.24
$\eta_1 = 0.2$, $\eta_2 = 0.5$	108	25	0	7	2	6.0127	272.27
$\eta_1 = 0.2$, $\eta_2 = 0.75$	108	27	0	7	2	5.4466	243.51
$\eta_1 = 0.2$, $\eta_2 = 0.9$ $\star$	110	38	0	5	2	5.8891	275.35
$\eta_1 = 0.2$, $\eta_2 = 0.95$	108	36	0	7	2	4.2610	155.56
$\eta_1 = 0.4$, $\eta_2 = 0.5$	109	38	0	6	2	5.5355	272.43
$\eta_1 = 0.4$, $\eta_2 = 0.75$	108	28	0	7	2	5.2759	257.34
$\eta_1 = 0.4$, $\eta_2 = 0.9$	111	30	0	4	2	6.7747	339.18
$\eta_1 = 0.4$, $\eta_2 = 0.95$	110	27	0	5	2	6.0359	294.14
$\eta_1 = 0.6$, $\eta_2 = 0.75$	111	34	0	4	2	5.3745	268.16

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen Zeit	Abbruch wegen Iterationen	Abbruch wegen Fehler	durchschnittlich benötigte Zeit	durchschnittlich benötigte Iterationen
$\eta_1 = 0.6$, $\eta_2 = 0.9$	113	29	0	2	2	6.3384	323.08
$\eta_1 = 0.6$, $\eta_2 = 0.95$	109	34	0	6	2	3.5942	133.98
$\eta_1 = 0.8$, $\eta_2 = 0.9$ $\star$	114	42	0	1	2	5.2002	250.96
$\eta_1 = 0.8$, $\eta_2 = 0.95$	115	30	0	0	2	6.7081	344.14
$\eta_1 = 0.9$, $\eta_2 = 0.95$ $\star$	115	37	0	0	2	6.9990	368.29

Tabelle 5.2: Übersicht des Parametertests für η_1 und η_2

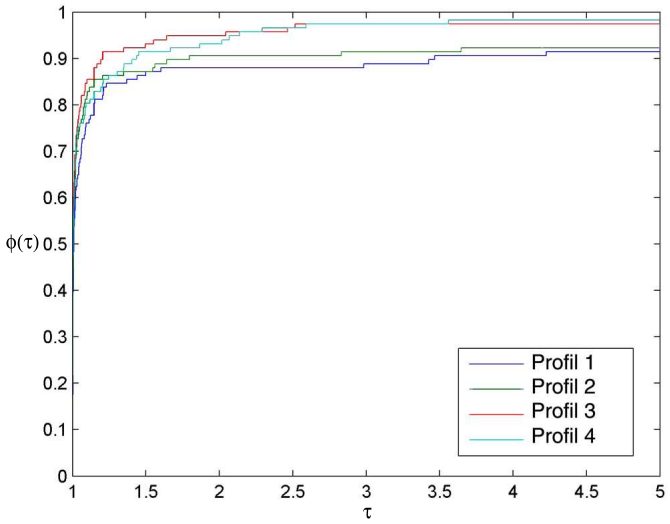


Abbildung 5.2: Leistungsprofil für geeignete Werte von η_1 und η_2. Dabei bezieht sich Profil 1 auf die Kombination $\eta_1 = 0.01$, $\eta_2 = 0.1$, Profil 2 auf $\eta_1 = 0.2$, $\eta_2 = 0.9$, Profil 3 auf $\eta_1 = 0.8$, $\eta_2 = 0.9$ und Profil 4 auf $\eta_1 = 0.9$, $\eta_2 = 0.95$.

Viele der Parametereinstellung, die zum Teil sehr unterschiedliche Werte für η_1 und η_2 beinhalten (z.B. $\eta_1 = 0.01$, $\eta_2 = 0.1$ bzw. $\eta_1 = 0.6$, $\eta_2 = 0.95$), erzielen sehr ähnliche Ergebnisse. Dies entspricht nicht dem Verhalten von gewöhnlichen Trust-Region-Verfahren. Jedoch ist dies dadurch erklärbar, dass nach Algorithmus 3.1 bei der Berechnung die Beschränkung des Testschrittes s_k auf die Trust-Region nur dann notwendig ist, falls im vorherige Iterationsschritt $k-1$ der Fehlervektor des Testpunktes x_{k-1}^+ nicht vom Filter akzeptabel war und entweder die Vorhersage der Verbesserung des Funktionswertes durch die Modellfunktion sehr schlecht, d.h. $\rho_{k-1} < \eta_1$, oder die Norm des Testschrittes größer als der Trust-Region-Radius war. Da die Trust-Region aus diesem Grund nicht in jedem Iterationsschritt berücksichtigt werden muss, ist der Einfluss der Parameter, durch die sie verändert wird, auch nicht so stark, als dies bei herkömmlichen Trust-Region-Verfahren der Fall ist.

Allerdings weisen sowohl Tabelle 5.2 als auch das zugehörige Leistungsprofil darauf hin, dass durch Verwendung von sehr großen Werten für η_1 und η_2 (z.B. $\eta_1 = 0.9$ und $\eta_2 = 0.95$) mehr Testbeispiele gelöst werden konnten. Dies ist eine direkte Folge davon, dass für diese Parametereinstellungen deutlich weniger bzw. keine Berechnung aufgrund zu vieler Iterationen abgebrochen werden musste. Sind die Werte sowohl für η_1 als auch für η_2 sehr groß, so wird der Trust-Region-Radius in der Regel häufiger verringert als bei kleineren Werten dieser Parameter. Für den FTR-Algorithmus scheint dies eine vorteilhafte Vorgehensweise zu sein, vermutlich aufgrund der Iterationsschritte, die ohne Beschränkung auf die Trust-Region durchgeführt werden und somit einen Ausgleich für eine rasche Verkleinerung des Trust-Region-Radius darstellen.
Die folgenden beiden Untersuchungen betreffen das Verhalten des Algorithmus bei Veränderung der Parameter γ_1 und γ_2. Dazu wird zunächst der Aktualisierungsschritt des Trust-Region-Radius zu

$$\Delta_{k+1}^2 = \begin{cases} \max\left(\gamma_1 \left\|s_k\right\|, \Delta_k\right) & falls \quad \rho_k < \eta_1 \\ \Delta_k & falls \quad \rho_k \in [\eta_1, \eta_2) \\ \gamma_2 \left\|s_k\right\| & falls \quad \rho_k \geq \eta_2 \end{cases}$$

betrachtet. Die folgende Tabelle beinhaltet die Zusammenfassung des Vergleichs von 30 Kombinationen verschiedener Werten der beiden Parameter γ_1 und γ_2. Für die mit $\star$ gekennzeichneten Parameterbelegungen wird im Anschluss an die tabellarische Aufstellung ein Leistungsprofil erstellt:

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen			durchschnittlich benötigte	
			Zeit	Iterationen	Fehler	Zeit	Iterationen
$\gamma_1 = 0.1$, $\gamma_2 = 1.5$	108	26	0	6	3	4.5350	191.57
$\gamma_1 = 0.1$, $\gamma_2 = 2$	107	32	0	7	3	4.7414	193.15
$\gamma_1 = 0.1$, $\gamma_2 = 2.5$	106	26	0	7	4	4.7462	195.32
$\gamma_1 = 0.1$, $\gamma_2 = 5$	108	22	0	7	2	4.6270	204.56
$\gamma_1 = 0.1$, $\gamma_2 = 7.5$	107	28	0	7	3	5.9298	260.18
$\gamma_1 = 0.1$, $\gamma_2 = 10$	108	28	0	6	3	6.6232	328.54
$\gamma_1 = 0.2$, $\gamma_2 = 1.5$	107	17	0	6	4	6.9144	337.46
$\gamma_1 = 0.2$, $\gamma_2 = 2$	106	35	0	7	4	4.7149	200.45
$\gamma_1 = 0.2$, $\gamma_2 = 2.5 \star$	106	41	0	6	5	6.3856	292.97
$\gamma_1 = 0.2$, $\gamma_2 = 5$	108	33	0	7	2	5.2066	212.81
$\gamma_1 = 0.2$, $\gamma_2 = 7.5$	109	31	0	6	2	5.4501	262.96
$\gamma_1 = 0.2$, $\gamma_2 = 10$	109	32	0	5	3	6.8795	360.00
$\gamma_1 = 0.25$, $\gamma_2 = 1.5$	108	25	0	6	3	5.6179	278.53

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen Zeit	Abbruch wegen Iterationen	Abbruch wegen Fehler	durchschnittlich benötigte Zeit	durchschnittlich benötigte Iterationen
$\gamma_1 = 0.25$, $\gamma_2 = 2$	107	21	0	7	3	4.6201	181.55
$\gamma_1 = 0.25$, $\gamma_2 = 2.5$	108	28	0	7	2	5.2626	206.68
$\gamma_1 = 0.25$, $\gamma_2 = 5$	108	28	0	6	3	6.3512	310.96
$\gamma_1 = 0.25$, $\gamma_2 = 7.5$	110	24	0	5	2	6.5473	347.67
$\gamma_1 = 0.25$, $\gamma_2 = 10$ $\star$	110	29	0	5	2	8.6138	398.32
$\gamma_1 = 0.5$, $\gamma_2 = 1.5$	107	28	0	7	3	5.4553	236.55
$\gamma_1 = 0.5$, $\gamma_2 = 2$ $\star$	108	33	0	6	3	6.0776	287.19
$\gamma_1 = 0.5$, $\gamma_2 = 2.5$	107	28	0	7	3	5.6254	250.79
$\gamma_1 = 0.5$, $\gamma_2 = 5$	109	27	0	5	3	6.7377	346.27
$\gamma_1 = 0.5$, $\gamma_2 = 7.5$	109	24	0	5	3	7.0736	355.65
$\gamma_1 = 0.5$, $\gamma_2 = 10$	109	30	0	5	3	7.1962	355.53
$\gamma_1 = 0.75$, $\gamma_2 = 1.5$	106	30	0	8	3	5.2559	189.08
$\gamma_1 = 0.75$, $\gamma_2 = 2$	106	28	0	8	3	5.5302	215.86
$\gamma_1 = 0.75$, $\gamma_2 = 2.5$	106	32	0	8	3	5.6293	223.17
$\gamma_1 = 0.75$, $\gamma_2 = 5$	106	35	0	8	3	5.7947	233.85
$\gamma_1 = 0.75$, $\gamma_2 = 7.5$	106	32	0	8	3	5.5452	226.74
$\gamma_1 = 0.75$, $\gamma_2 = 10$ $\star$	108	33	0	6	3	6.4244	292.56

Tabelle 5.3: Übersicht des Parametertests für γ_1 und γ_2 bei Trust-Region-Radius-Update zu Δ^2_{k+1}

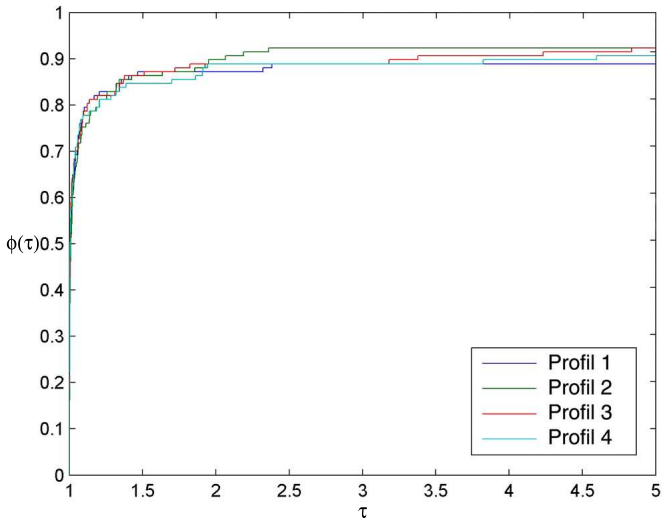


Abbildung 5.3: Leistungsprofil für geeignete Werte von γ_1 und γ_2 bei Trust-Region-Radius-Update zu Δ^2_{k+1}. Dabei bezieht sich Profil 1 auf die Kombination $\gamma_1 = 0.2$, $\gamma_2 = 2.5$, Profil 2 auf $\gamma_1 = 0.25$, $\gamma_2 = 10$, Profil 3 auf $\gamma_1 = 0.5$, $\gamma_2 = 2$ und Profil 4 auf $\gamma_1 = 0.75$, $\gamma_2 = 10$.

Sowohl aus Tabelle 5.3 als auch aus dem Leistungsprofil ist ersichtlich, dass es nur geringe Unterschiede bei der Anzahl der gelösten Testbeispiele gibt; sie liegt bei jeder Parametereinstellung zwischen 106 und 110. Jedoch erzielen die Werte $\gamma_2 = 7.5$ und $\gamma_2 = 10$ geringfügig bessere Ergebnisse, als dies durch die Wahl eines kleineren Wertes für diesen Parameter auf der verwendeten Testmenge möglich ist. Für γ_1 empfiehlt sich aufgrund von Tabelle 5.3 ein Wert von 2 oder 2.5.
Auch für den alternativen Aktualisierungsschritt des Trust-Region-Radius

$$\Delta_{k+1}^1 = \begin{cases} \gamma_1 \cdot \Delta_k & falls \quad \rho_k < \eta_1 \\ \Delta_k & falls \quad \rho_k \in [\eta_1, \eta_2) \\ \gamma_2 \cdot \Delta_k & falls \quad \rho_k \geq \eta_2 \end{cases}$$

wurde ein Vergleich der selben 30 Parametereinstellungen für γ_1 und γ_2 durchgeführt und die zusammengefassten Ergebnisse in der nachfolgenden Tabelle, sowie im dazugehörenden Leistungsprofil dargestellt:

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen			durchschnittlich benötigte	
			Zeit	Iterationen	Fehler	Zeit	Iterationen
$\gamma_1 = 0.1$, $\gamma_2 = 1.5$ $\star$	109	37	0	6	2	6.1168	191.57
$\gamma_1 = 0.1$, $\gamma_2 = 2$	108	30	0	6	3	5.5455	193.15
$\gamma_1 = 0.1$, $\gamma_2 = 2.5$	107	25	0	6	4	5.7262	195.32
$\gamma_1 = 0.1$, $\gamma_2 = 5$	107	30	0	6	4	5.2473	204.56
$\gamma_1 = 0.1$, $\gamma_2 = 7.5$	105	22	0	7	5	5.5149	260.18
$\gamma_1 = 0.1$, $\gamma_2 = 10$	108	25	0	6	3	7.0914	328.54
$\gamma_1 = 0.2$, $\gamma_2 = 1.5$	106	22	0	7	4	4.6411	337.46
$\gamma_1 = 0.2$, $\gamma_2 = 2$	106	28	0	7	4	4.8057	200.45
$\gamma_1 = 0.2$, $\gamma_2 = 2.5$	107	24	0	6	4	5.2492	292.97
$\gamma_1 = 0.2$, $\gamma_2 = 5$ $\star$	108	33	0	5	4	7.3842	212.81
$\gamma_1 = 0.2$, $\gamma_2 = 7.5$ $\star$	110	27	0	5	2	7.2186	262.96
$\gamma_1 = 0.2$, $\gamma_2 = 10$	106	32	0	7	4	3.8861	360.00
$\gamma_1 = 0.25$, $\gamma_2 = 1.5$	108	33	0	6	3	5.8581	278.53
$\gamma_1 = 0.25$, $\gamma_2 = 2$	108	28	0	6	3	6.5531	181.55
$\gamma_1 = 0.25$, $\gamma_2 = 2.5$	108	22	0	6	3	5.4324	206.68
$\gamma_1 = 0.25$, $\gamma_2 = 5$	107	25	0	7	3	5.0646	310.96
$\gamma_1 = 0.25$, $\gamma_2 = 7.5$	107	26	0	6	4	6.3353	347.67
$\gamma_1 = 0.25$, $\gamma_2 = 10$ $\star$	109	27	0	5	3	12.379	398.32
$\gamma_1 = 0.5$, $\gamma_2 = 1.5$	108	27	0	6	3	6.5179	236.55
$\gamma_1 = 0.5$, $\gamma_2 = 2$	107	28	0	7	3	5.2261	287.19
$\gamma_1 = 0.5$, $\gamma_2 = 2.5$	107	23	0	7	3	5.6871	250.79

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen Zeit	Iterationen	Fehler	durchschnittlich benötigte Zeit	Iterationen
$\gamma_1 = 0.5$, $\gamma_2 = 5$	108	30	0	6	3	6.8855	346.27
$\gamma_1 = 0.5$, $\gamma_2 = 7.5$	107	22	0	6	4	6.3413	355.65
$\gamma_1 = 0.5$, $\gamma_2 = 10$	108	26	0	6	3	6.4773	355.53
$\gamma_1 = 0.75$, $\gamma_2 = 1.5$	108	26	0	6	3	7.2495	189.08
$\gamma_1 = 0.75$, $\gamma_2 = 2$	107	22	0	7	3	6.8861	215.86
$\gamma_1 = 0.75$, $\gamma_2 = 2.5$	107	18	0	7	3	6.8881	223.17
$\gamma_1 = 0.75$, $\gamma_2 = 5$	105	17	0	7	5	6.3912	233.85
$\gamma_1 = 0.75$, $\gamma_2 = 7.5$	106	14	0	7	4	7.1084	226.74
$\gamma_1 = 0.75$, $\gamma_2 = 10$	107	27	0	6	4	7.1777	292.56

Tabelle 5.4: Übersicht des Parametertests für γ_1 und γ_2 bei Trust-Region-Radius-Update zu Δ_{k+1}^1

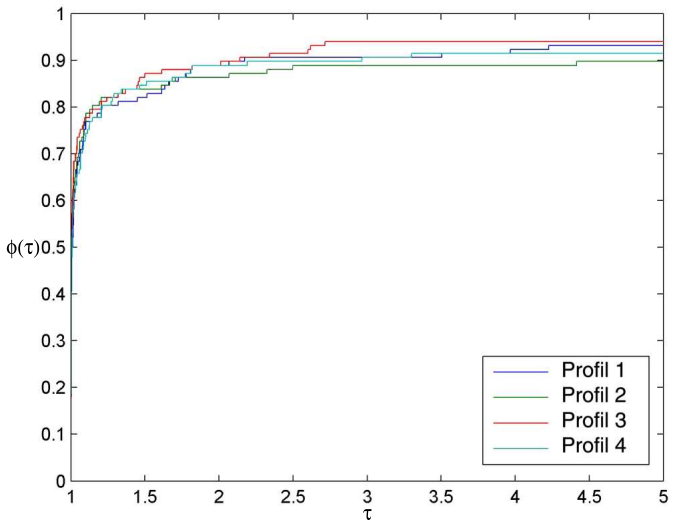


Abbildung 5.4: Leistungsprofil für geeignete Werte von γ_1 und γ_2 bei Trust-Region-Radius-Update zu Δ_{k+1}^1. Dabei bezieht sich Profi 1 auf die Kombination $\gamma_1 = 0.1$, $\gamma_2 = 1.5$, Profil 2 auf $\gamma_1 = 0.2$, $\gamma_2 = 5$, Profil 3 auf $\gamma_1 = 0.2$, $\gamma_2 = 7.5$ und Profil 4 auf $\gamma_1 = 0.25$, $\gamma_2 = 10$.

Die Unterschiede zwischen den einzelnen Parametereinstellungen sind ähnlich geringfügig wie bei der vorherigen Untersuchung unter Verwendung des Aktualisierungsschrittes $\Delta_{k+1} = \Delta_{k+1}^2$. Im Unterschied dazu sind hier jedoch keine Tendenzen zu erkennen, aufgrund derer eine Empfehlung eines bestimmten Parameterwertes möglich wäre. Die beiden Parameterkombinationen, mit denen auf der verwendeten Menge von Testbeispielen die meisten Probleme gelöst werden können bzw. die meisten Probleme am schnellsten gelöst werden können, sind $\gamma_1 = 0.2$, $\gamma_2 = 7.5$ bzw. $\gamma_1 = 0.1$, $\gamma_2 = 1.5$.

Um einen Vergleich der beiden alternativen Aktualisierungsschritte des Trust-Region-Radius durchzuführen, werden nun jeweils die beiden am geeignetsten erscheinenden Parametereinstellung der beiden vorangegangenen Untersuchungen betrachtet. Dies sind für Δ_{k+1}^2 die Kombinationen $\gamma_1 = 0.2$, $\gamma_2 = 2.5$ und $\gamma_1 = 0.25$, $\gamma_2 = 10$ und für Δ_{k+1}^1 die bereits erwähnten Wertepaare $\gamma_1 = 0.2$, $\gamma_2 = 7.5$ und $\gamma_1 = 0.1$, $\gamma_2 = 1.5$. Damit ergibt sich folgende tabellarische Zusammenstellung der Ergebnisse:

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen Zeit	Abbruch wegen Iterationen	Abbruch wegen Fehler
$\Delta_{k+1} = \Delta_{k+1}^2$, $\gamma_1 = 0.2$, $\gamma_2 = 2.5$	106	51	0	6	5
$\Delta_{k+1} = \Delta_{k+1}^2$, $\gamma_1 = 0.25$, $\gamma_2 = 10$	110	37	0	5	2
$\Delta_{k+1} = \Delta_{k+1}^1$, $\gamma_1 = 0.1$, $\gamma_2 = 1.5$	109	58	0	6	2
$\Delta_{k+1} = \Delta_{k+1}^1$, $\gamma_1 = 0.2$, $\gamma_2 = 7.5$	110	45	0	5	2

Tabelle 5.5: Übersicht des Parametertests für γ_1, γ_2 und Δ_{k+1}

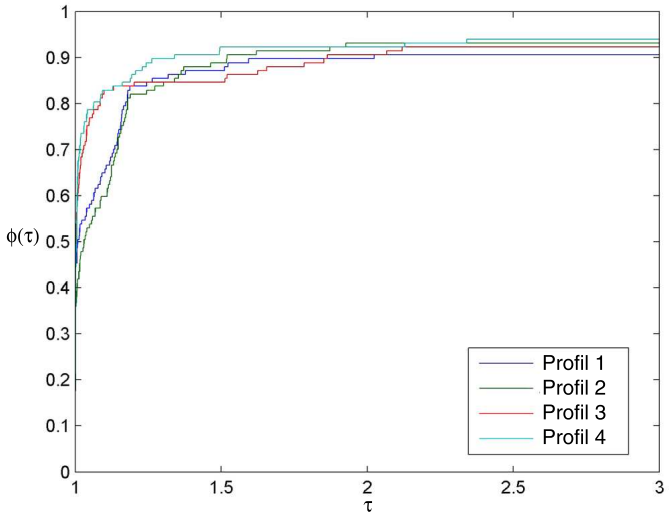


Abbildung 5.5: Leistungsprofil zum Vergleich der Trust-Region-Radius Aktualisierungsschritte. Dabei bezieht sich Profil 1 auf Δ_{k+1}^2, $\gamma_1 = 0.2$, $\gamma_2 = 2.5$, Profil 2 auf Δ_{k+1}^2, $\gamma_1 = 0.25$, $\gamma_2 = 10$, Profil 3 auf Δ_{k+1}^1, $\gamma_1 = 0.1$, $\gamma_2 = 1.5$ und Profil 4 auf Δ_{k+1}^1, $\gamma_1 = 0.2$, $\gamma_2 = 7.5$.

Weder das Leistungsprofil noch die tabellarische Zusammenfassung dieser Tests zeigen deutliche Unterschiede der beiden betrachteten Aktualisierungsschritte auf. Allerdings können mit dem Trust-Region-Radius-Update $\Delta_{k+1} = \Delta_{k+1}^1$ durchschnittlich mehr Testbeispiele am schnellsten gelöst werden. Dies wird im Leistungsprofil dadurch deutlich, dass die zugehörigen Graphen für $\tau < 1.2$ eindeutig größerer Werte als diejenigen, die zum Aktualisierungsschritt $\Delta_{k+1} = \Delta_{k+1}^2$ gehören, annehmen. In

[4] wird dagegen der Aktualisierungsschritt $\Delta_{k+1} = \Delta_{k+1}^2$ mit den Parameterwerten $\gamma_1 = 0.25$ und $\gamma_2 = 2.5$ als gute Wahl empfohlen. Allerdings werden dort nur reine Trust-Region-Verfahren betrachtet. Dies legt erneut den Schluss nahe, dass Parametereinstellungen, die sich für allgemeine Trust-Region-Verfahren als geeignet erwiesen haben, nicht ohne Weiteres auf das FTR-Verfahren übertragen lassen.

5.3 Mehrdimensionaler Filter

Zunächst soll der Parameter γ_θ, der die Größe des Korridors der Mindestverbesserung um den mehrdimensionalen Filter beeinflusst, betrachtet werden. Es gibt für dessen Wahl grundsätzlich die beiden Möglichkeiten, einen konstanten Wert zu verwenden oder γ_θ abhängig von der Anzahl p der Gruppen von Funktionen zu berechnen. In Tabelle 5.6 sind die Ergebnisse für unterschiedliche Belegungen von γ_θ zusammengefasst, wobei zehn feste sowie zehn variable Werte verwendet werden:

Parameter-einstellung	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen			durchschnittlich benötigte	
			Zeit	Iterationen	Fehler	Zeit	Iterationen
$\gamma_\theta = 10^{-1}$	95	30	0	12	10	2.6854	79.768
$\gamma_\theta = 10^{-2}$	102	32	0	11	4	6.7747	127.84
$\gamma_\theta = 10^{-3}$	106	25	0	9	2	4.7108	166.82
$\gamma_\theta = 10^{-4}$ $\star$	110	34	0	5	2	6.6100	303.74
$\gamma_\theta = 10^{-5}$	108	27	0	7	2	5.2587	206.68
$\gamma_\theta = 10^{-6}$ $\star$	108	35	0	7	2	5.9201	239.16
$\gamma_\theta = 10^{-7}$	108	27	0	6	3	6.9228	321.98
$\gamma_\theta = 10^{-8}$	108	24	0	6	3	6.8436	314.85
$\gamma_\theta = 10^{-9}$	108	28	0	6	3	6.8457	314.85
$\gamma_\theta = 10^{-10}$	108	19	0	6	3	6.8446	314.80
$\gamma_\theta = \sqrt{p}^{-1}$	93	18	0	14	10	3.3255	96.656
$\gamma_\theta = \sqrt{p}^{-2}$	93	19	0	13	11	3.6619	138.10
$\gamma_\theta = \sqrt{p}^{-3}$	94	16	0	12	11	2.7211	81.500
$\gamma_\theta = \sqrt{p}^{-4}$	94	20	0	13	10	2.6868	70.287
$\gamma_\theta = \sqrt{p}^{-5}$	100	27	0	11	6	3.2904	129.77
$\gamma_\theta = \sqrt{p}^{-6}$	102	24	0	10	5	3.4170	85.422
$\gamma_\theta = \sqrt{p}^{-7}$ $\star$	102	27	0	11	4	5.7548	111.63
$\gamma_\theta = \sqrt{p}^{-8}$	104	25	0	10	3	6.4738	163.18
$\gamma_\theta = \sqrt{p}^{-9}$	105	19	0	9	3	6.8278	163.29
$\gamma_\theta = \sqrt{p}^{-10}$ $\star$	106	20	0	9	2	4.0909	111.02

Tabelle 5.6: Übersicht des Parametertests für γ_θ

Das Leistungsprofil setzt sich hier aus zwei konstanten sowie aus zwei von p abhängigen

Werten für γ_θ zusammen:

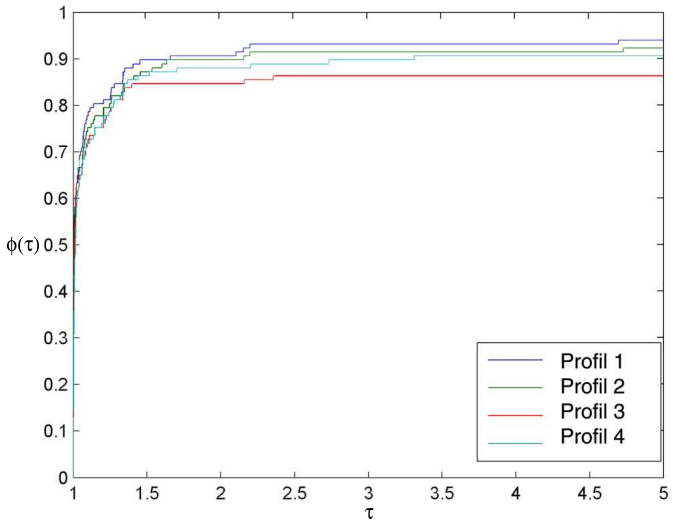


Abbildung 5.6: Leistungsprofil für Werte von γ_θ. Dabei bezieht sich Profil 1 auf $\gamma_\theta = 10^{-4}$, Profil 2 auf $\gamma_\theta = 10^{-6}$, Profil 3 auf $\gamma_\theta = \sqrt{p}^{-7}$ und Profil 4 auf $\gamma_\theta = \sqrt{p}^{-10}$.

Auffällig ist, dass die Wahl von γ_θ offenbar die Anzahl der Probleme, für die der Algorithmus aufgrund eines Fehlers bei der Berechnung von ρ_k abgebrochen werden muss, vergleichsweise stark beeinflusst. Die Anzahl der Testbeispiele, die deswegen abgebrochen werden mussten, schwankt zwischen 2 und 11.
Der Algorithmus löste im Allgemeinen mit einem kleineren Wert für γ_θ mehr Testbeispiele. Dies deutet darauf hin, dass der Korridor der Mindestverbesserung um den Filter nicht zu groß sein sollte, da sonst zu viele Iterationswerte abgelehnt werden. Allerdings liegt der für die betrachtete Menge von Testbeispielen am besten geeignete Wert bei $\gamma_\theta = 10^{-4}$ und somit im mittleren Wertebereich von γ_θ. Außerdem scheint es, dass die Wahl eines konstanten Wertes für γ_θ einer variablen Bestimmung vorzuziehen ist. Ein Grund dafür ist vermutlich, dass die Menge der Testbeispiele überwiegend aus Gleichungssystemen mit vergleichsweise wenig Zeilen besteht und deshalb der Wert $\sqrt{p}^{-i}$ deutlich kleiner als beispielsweise 10^{-i} ist.
Die Einteilung der Funktionen in Gruppen ist eine weitere Möglichkeit den FTR-Algorithmus zu variieren. Die folgende Tabelle bzw. das folgende Leistungsprofil zeigt die Ergebnisse eines Vergleichs zwischen den beiden kanonischen Varianten die Unterteilung vorzunehmen: beim ersten Testlauf wird jeder Funktion eine eigene Gruppe zugewiesen, beim zweiten bilden alle Funktionen eine einzige Gruppe. Für diese Untersuchung wird die Testmenge um sechs Testbeispiele, die aus den Problemen ATB9 und ATB10 abgeleitet wurden, reduziert, da andernfalls die Berechnungen mit Hilfe der zweiten getesteten Variante zu viel Zeit in Anspruch nehmen würde. Für die verbleibenden 111 Testbeispiele ergeben sich folgende Ergebnisse:

Einteilung der Funktionen	gelöste Beispiele	am schnellsten gelöst	Abbruch wegen			durchschnittlich benötigte	
			Zeit	Iterationen	Fehler	Zeit	Iterationen
in p Gruppen	102	101	0	7	2	5.3466	218.80
in eine Gruppe	58	1	0	15	38	6.2904	256.67

Tabelle 5.7: Vergleich zweier Gruppeneinteilungen

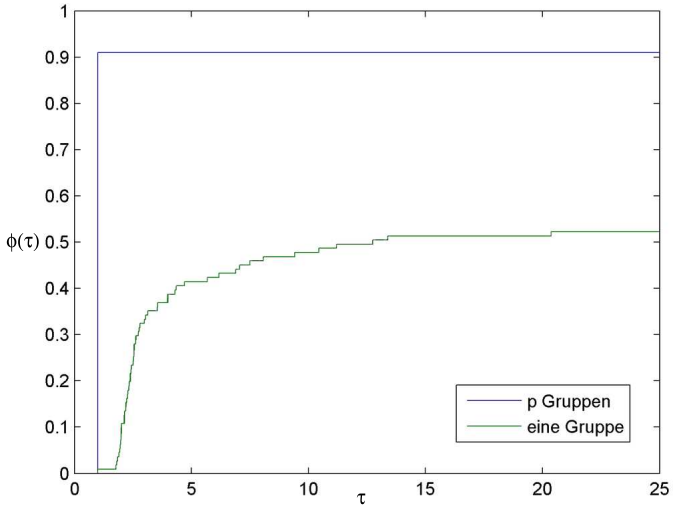


Abbildung 5.7: Leistungsprofil der beiden Gruppeneinteilungen

Indem man alle Funktionen zu einer einzigen Gruppe zusammenfasst, wird die Wirkungsweise des Filters stark beeinträchtigt, da als Folge davon die Fehlervektoren und somit auch die Einträge des Filters nur aus einer Koordinate bestehen. Im Leistungsprofil ist der dadurch entstehende Nachteil deutlich zu erkennen. Die Einteilung der m Funktionen in m Gruppen erweist sich dagegen, wie auch in [12] dargestellt, meist als geeignete Einstellung. Dies liegt nach [12] vor allem darin begründet, dass Filter höherer Dimension durchschnittlich mehr Punkte und demnach auch mehr Iterationswerte akzeptieren, als ein Filter, dessen Einträge weniger Koordinaten aufweisen.

5.4 Zusammenfassung der Ergebnisse

Die vorangegangenen numerischen Tests lassen eine Reihe von Aussagen über die Wahl geeigneter Parameterwerte und Variationsmöglichkeiten des FTR-Algorithmus zu.
Für den Parameter γ_θ zur Beeinflussung der Größe des Korridors der Mindestverbesserung um den mehrdimensionalen Filter kann man aus dem betreffenden numerischen

Test recht deutlich ersehen, dass der konstante Wert $\gamma_\theta = 10^{-4}$ die besten Resultate erzielt. Für die Parameter γ_1, γ_2, η_1 und η_2 sowie für die beiden Alternativen zur Aktualisierung des Trust-Region-Radius haben sich wenig signifikante Ergebnisse ergeben. Die deutlichste Tendenz kann für η_1 und η_2 gezeigt werden. Diese beiden Parameter sollten demnach am oberen Rand ihres Wertebereichs liegen, da unter Verwendung von $\eta_1 = 0.9$ und $\eta_2 = 0.95$ die besten Resultate erzielt werden können. Variationen der Einstellungen der anderen Faktoren zur Steuerung der Trust-Region ergeben nur geringfügige Unterschiede auf der betrachteten Menge von Testbeispielen. Als empfehlenswerte Kombination bieten sich jedoch die Werte $\Delta_{k+1} = \Delta^1_{k+1}$, $\gamma_1 = 0.2$ und $\gamma_2 = 7.5$ an. Mit dieser Wahl der Parameter können die meisten Probleme gelöst werden und es kann ebenfalls für vergleichsweise viele Probleme in kürzester Laufzeit eine Lösung ermittelt werden.
Der Vergleich der beiden verschiedenen Gruppeneinteilungen der Zeilen des zugrundeliegenden nichtlinearen Gleichungssystems zeigt dagegen deutlich, dass es vorteilhaft ist, möglichst viele Funktionsgruppen zu erzeugen. Es sollte demnach im Allgemeinen für ein NLG bestehend aus m Zeilen ein m-dimensionaler Filter verwendet werden.
Die Wahl der Modellfunktion ist für die Berechnung der Lösung eines NLG von entscheidender Bedeutung. Im allgemeinen kann aufgrund der Testergebnisse das Gauss-Newton-Modell für die meisten Probleme als geeignet angesehen werden. Jedoch ist es aufgrund der relativ großen Anzahl der Testbeispiele, die damit nicht gelöst werden konnten, nötig darauf hinzuweisen, dass die Resultate, die mit einer bestimmten Modellfunktion erzielt werden, sehr problemspezifisch sind. So ist beispielsweise das lineare Modell, dass im Allgemeinen die schlechtesten Ergebnisse erzielt, die einzige Modellfunktion, mit der die Testbeispiele A282, A297 und A298 gelöst werden können.

Kapitel 6

Ausblick

Die während dieser Diplomarbeit entwickelte Implementierung des Filter-Trust-Region-Algorithmus konnte bereits auf zwei explizite Problemstellungen angewandt werden. Zum einen konnten Lösungen für nichtlineare Gleichungssysteme für mechanische Kontaktprobleme, die während der Diplomarbeit von Herrn Tobias Rasp auftraten, bestimmt werden.
Die zweite Möglichkeit zur Erprobung der praktischen Tauglichkeit der Implementierung ergab sich im Zusammenhang mit dem Online Target Date Assignment Problem (OTDAP) [17]. Für das von Herrn Dipl.-Ing. Sleman Saliba dafür aufgestellte nichtlineare Gleichungssystem konnte für Beispiele mit bis zu 10 Funktionen und 10 Variablen unter Verwendung des Standardwertes für den Startpunkt $x_0 = 0$ eine Lösung bestimmt werden (Testbeispiele ATB3-ATB8). Für Beispiele mit bis zu 100 Funktionen und 100 Variablen (Testbeispiele ATB9 und ATB10) konnte die von Herrn Saliba vermutet Lösung $x^* = (x^*(1), ..., x^*(n))$ bestätigt werden. Dabei ergibt sich x^* aus der Rekursionsformel

$$x^*(0) = 1$$

und

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

wobei $f(i)$ die i-te Fibonacci-Zahl bezeichnet. Daraus ergeben sich zum Teil sehr große Koordinatenwerte für $x^*(i)$ und folglich auch der Grund, warum diese Testbeispiele nicht vom Startwert $x_0 = 0$ ausgehend lösbar sind. Der Standardwert des Startpunktes liegt in diesem Fall zu weit von der Lösung entfernt, d.h. $\|x_0 - x^*\|$ ist zu groß, als dass diese in annehmbarer Rechenzeit erreicht werden könnte.
Um die Zuverlässigkeit des FTR-Algorithmus, insbesondere im Hinblick auf den Fehler der im Zusammenhang mit der Berechnung von ρ_k auftritt, zu erhöhen, sind weitere ausführliche numerische Tests oder entsprechend angepasste Implementierungsvarianten notwendig.
Das hier vorgestellte FTR-Verfahren zur Berechnung einer Lösung von nichtlinearen Gleichungssystemen wurde mit nur geringen Ergänzungen in [13] unter dem Namen Filtrane veröffentlicht und erreicht den dort durchgeführten numerischen Tests

zufolge ein deutlich höheres Maß an Zuverlässigkeit und Effizienz als vergleichbare Trust-Region-Verfahren. Die wesentliche Weiterentwicklung von Filtrane verglichen mit dem FTR-Algorithmus, der hier vorgestellt wurde, ist die Möglichkeit nichtlineare Ungleichungen zu berücksichtigen. So wird ein Vektor $x^* \in \mathbb{R}$ bestimmt, der die Bedingungen

$$c_E(x) = 0$$

und

$$c_I(x) \geq 0$$

mit $c_E(x) : \mathbb{R}^n \to \mathbb{R}^m$ und $c_I(x) : \mathbb{R}^n \to \mathbb{R}^q$ erfüllt. Dies wird durch eine abgeänderte Definition des Fehlervektors $\theta(x) := (c_E(x), [c_I(x)]_-)^\top$ mit $[c_I(x)]_- = \min\{0, c_I(x)\}$ ermöglicht.
Eine weitere Änderung ist der Verzicht auf die in Algorithmus 3.1 verwendete Variable RESTRICT. Stattdessen wird ein Parameter $\tau_k \geq 1$ eingeführt und die Anforderung an die Läge der Schrittweite als $\|s_k\|_k \leq \tau_k \Delta_k$ formuliert. Dies entspricht für $\tau = 1$ der im FTR-Algorithmus verwendeten Bedingung (3.9). τ_k wird in den gleichen Situation größer als 1 gewählt, in denen in Algorithmus 3.1 RESTRICT="false" gesetzt, also die Beschränkung der Länge des Testschrittes auf die Trust-Region aufgehoben wird. Die Verwendung von τ_k führt also dazu, das die unbeschränkten Schritte im FTR-Algorithmus durch Schritte ersetzt werden, deren Länge zwar wesentlich größer als der Trust-Region-Radius sein darf - in [13] wird als maximaler Wert $\tau = 1000$ empfohlen - aber dennoch beschränkt bleiben. Die prinzipielle Vorgehensweise des übrigen Algorithmus stimmt mit der des FTR-Algorithmus überein.

Anhang A

Liste der implementierten Funktionen

In der folgenden Liste sind alle auf der beiliegenden CD enthaltenen Funktionen aufgeführt. Dabei handelt es sich um die in Kapitel 4 vorgestellten Algorithmen zur Handhabung des mehrdimensionalen Filters und um alle für die Ausführung des FTR-Algorithmus notwendigen Funktionen. Außerdem enthält die Zusammenstellung die Matlab-Dateien zur Erzeugung der verwendeten Testbeispiele.

- **add_to_filter.m** - Fügt einen Testpunkt zu einem Filter hinzu, falls er von keinem Filtereintrag dominiert wird und entfernt alle Einträge, die von einem akzeptierten Testpunkt dominiert werden, aus dem Filter.
- **add_to_filter_env.m** - Fügt einen Testpunkt zu einem Filter hinzu, falls dieser die nötige Akzeptanzbedingung erfüllt und entfernt alle Filtereinträge, die aufgrund des Testpunktes nicht länger akzeptiert werden.
- **eindeutigkeit_vektoren.m** - Entfernt aus einer Matrix jeden Zeilenvektor z_i, der identisch mit einem Zeilenvektor z_j, $j < i$ ist. Dadurch entsteht eine Matrix, in der kein Zeilenvektor mehrfach auftritt.
- **fplot2d.m** - Stellt einen zweidimensionalen Filter graphisch dar.
- **fplot3d.m** - Stellt einen dreidimensionalen Filter graphisch dar.
- **fplot3d_2dfilter.m** - Ergänzt die Matrix eines zweidimensionalen Filters um die für die graphische Darstellung nötigen Zwischenpunkte.
- **ftr.m** - Bestimmt die Lösung eines Nichtlinearen Gleichungssystems durch Anwendung des FTR-Algorithmus.
- **ftr2.m** - Analog zu ftr.m, unterstützt jedoch die Eingabe der Zeilen des Gleichungssystems als Matlab-Dateien.

- **full_newton_modell.m** - Approximiert den Wert der Zielfunktion in einem Punkt x_1 mit Hilfe des Full-Newton-Modells entwickelt im Iterationswert x_k.
- **full_newton_modell2.m** - Analog zu full_newton_modell.m, unterstützt jedoch die Eingabe der Zeilen des Gleichungssystems als Matlab-Dateien.
- **gauss_newton_modell.m** - Approximiert den Wert der Zielfunktion in einem Punkt x_1 mit Hilfe des Gauss-Newton-Modells entwickelt im Iterationswert x_k.
- **gauss_newton_modell2.m** - Analog zu gauss_newton_modell.m, unterstützt jedoch die Eingabe der Zeilen des Gleichungssystems als Matlab-Dateien.
- **init_filter.m** - Initialisiert einen Filter bestehend aus nicht dominierten Punkten, ausgehend von einer beliebigen Menge von Punkten.
- **init_filter_graef.m** - Initialisiert einen Filter bestehend aus nicht dominierten Punkten, ausgehend von einer beliebigen Menge von Punkten. Basiert auf dem Graef-Younes-Verfahren.
- **init_schwach.m** - Initialisiert einen Filter, der aus schwach Pareto-optimalen Punkten besteht, ausgehend von einer beliebigen Menge von Punkten.
- **lineares_modell.m** - Approximiert den Wert der Zielfunktion in einem Punkt x_1 mit Hilfe eines linearen Modells der Zielfunktion entwickelt im Iterationswert x_k.
- **lineares_modell2.m** - Analog zu lineares_modell.m, unterstützt jedoch die Eingabe der Zeilen des Gleichungssystems als Matlab-Dateien.
- **plot2d_env_var.m** - Stellt einen zweidimensionalen Filter und den Korridor der Mindestverbesserung graphisch dar.
- **quadratisches_modell.m** - Approximiert den Wert der Zielfunktion in einem Punkt x_1 mit Hilfe eines quadratischen Modells der Zielfunktion entwickelt im Iterationswert x_k.
- **quadratisches_modell2.m** - Analog zu quadratisches_modell.m, unterstützt jedoch die Eingabe der Zeilen des Gleichungssystems als Matlab-Dateien.
- **test_hock_schittkowski.m** - Erzeugt eine Auswahl von Testbeispielen, die aus [14] und [18] übernommen wurden.
- **testbeispiele.m** - Erzeugt eine Auswahl von Testbeispielen, die in Zusammenarbeit mit Herrn Dipl.-Technomath. Alexander Thekale entstanden bzw. von Herrn Dipl.-Ing. Sleman Saliba für das OTDAP erstellt wurden.
- **verb_mess_fil.m** - Ermittelt die Verbesserung, die ein Testpunkt in einem Filter erzeugt.

- **verb_mess_raster.m** - Ermittelt die Verbesserung, die ein Testpunkt in einem Filter erzeugt.
- **vol_verb_fil.m** - Wird von verb_mess_fil.m zur rekursiven Berechnung des Volumens unter einem Filter benötigt.

Anhang B

Testbeispiele

Die folgende Liste enthält alle in Kapitel 5 verwendeten Testbeispiele. Diese wurden in den Matlab-Dateien test_hock_schittkowski.m bzw. testbeispiele.m gespeichert und stehen nach deren Ausführen in der von Algorithmus ftr.m benötigten Form zur Verfügung.

Bezeichnung	Anzahl		Quelle	Änderungen	Verwendung für Parametertest
	Funktionen	Variablen			
A001	3	3	[14]	1)	Q, FN
A006	2	2	[14]		GN, L, Q, FN
A007	2	2	[14]	2)	GN, L, Q, FN
A009	2	2	[14]	2)	GN, L, Q, FN
A026	2	3	[14]		GN, L, Q, FN
A027	2	3	[14]	2)	GN, L, Q, FN
A028	2	3	[14]		GN, L, Q, FN
A039	3	3	[14]	2)	GN, Q, FN
A040	4	4	[14]	2)	GN, Q, FN
A042	3	4	[14]	2)	GN, Q, FN
A046	3	5	[14]		Q, FN
A047	4	5	[14]		GN, Q, FN
A048	3	5	[14]		GN, L, Q, FN
A049	3	5	[14]		
A050	4	5	[14]		GN, Q, FN
A051	4	5	[14]		GN, Q, FN
A052	4	5	[14]	2)	GN, Q, FN
A061	3	3	[14]	2)	
A077	3	5	[14]	2)	GN, Q, FN
A205	3	2	[18]	3)	GN, L, Q, FN
A219	3	4	[18]	2)	GN, L, Q

Bezeichnung	Anzahl		Quelle	Änderungen	Verwendung für
	Funktionen	Variablen			Parametertest
A240	3	3	[18]	3)	GN, L, Q, FN
A241	5	3	[18]	3)	GN, L, Q, FN
A242	10	3	[18]	3)	
A255	7	4	[18]	3)	GN, Q, FN
A256	4	4	[18]	3)	GN, L, Q, FN
A261	5	4	[18]	3)	GN, L, Q, FN
A267	11	5	[18]	3)	
A271	6	6	[18]	3)	GN, L, Q, FN
A272	13	6	[18]	3)	
A282	10	10	[18]	3)	
A286	10	20	[18]	3)	
A287	5	20	[18]	3)	
A288	5	20	[18]	3)	
A294	5	6	[18]	3)	GN, L, Q, FN
A295	9	10	[18]	3)	
A296	15	16	[18]	3)	
A297	29	30	[18]	3)	
A298	49	50	[18]	3)	
A299	99	100	[18]	3)	
A352	20	4	[18]	3)	
A373	7	9	[18]	2), 3)	
A394	21	20	[18]	3)	
A395	51	50	[18]	3)	
ATB1	3	3			GN, L, Q, FN
ATB2	3	3			GN, L, Q, FN
ATB3	3	3	[17]		GN, L, Q, FN
ATB4	4	4	[17]		GN, Q, FN
ATB5	5	5	[17]		GN, Q, FN
ATB6	6	6	[17]		GN
ATB7	7	7	[17]		GN
ATB8	10	10	[17]		GN
ATB9	50	50	[17]		GN, L, Q, FN
ATB10	100	100	[17]		GN, L, Q, FN

Folgende Änderungen wurden bei den Testbeispielen durchgeführt:

1) Eine Ungleichung der Form $x_j \leq a$, $a \in \mathbb{R}$ wurde durch Einführen einer weiteren Variable x_i und einer weiteren Gleichung $x_i - |x_i| = 0$ durch $x_j + x_i = a$ ersetzt.

2) Um aus dem zugrundeliegenden nichtlinearen Optimierungsproblem ein homogenes nichtlineares Gleichungssystem zu entwickeln, wurde eine Konstante $c \in \mathbb{R}$ zur Zielfunktion addiert, so dass diese in der Lösung des Gleichungssystems den Wert 0 annimmt.
3) Die aus einer Summe bestehende Zielfunktion des zugrundeliegenden Optimierungsproblems wurde so unterteilt, dass jeder Summand eine Zeile des entstehenden Gleichungssystems bildet.

In der letzten Spalte der vorangegangenen Tabelle sind für das Testbeispiel p die Modellfunktionen m angegeben, für die das Paar (p, m) eines der 117 Elemente der reduzierten Testmenge für die Parametertests darstellt. Dabei bezeichnet GN das Gauss-Newton-Modell, L das lineare Modell, Q das quadratische Modell und FN das Full-Newton-Modell.

Anhang C

Ausführliche Testergebnisse

In der folgenden Tabelle sind die numerischen Ergebnisse des Vergleichs der vier Modellfunktionen dargestellt. Für die Parameter wurden die Werte $\gamma_\theta = 10^{-5}$, $\eta_1 = 0.1$, $\eta_2 = 0.9$, $\gamma_1 = 0.25$ und $\gamma_2 = 2.5$ verwendet. Der Trust-Region-Radius wurde nach der Aktualisierungsvorschrift für Δ_{k+1}^2 berechnet und jede Zeile eines Gleichungssystems bildete eine eigene Gruppe.
$f(x^*)$ bezeichnet dabei den Wert der Zielfunktion in der ermittelten Lösung x^*. Für die Kennzeichnung der Modellfunktionen - Gauss-Newton-Modell (GN), lineares Modell (L), quadratisches Modell (Q) und Full-Newton-Modell (FN) - wurden die angegebenen Abkürzungen verwendet.
In der letzten Spalte wurde die Ursache für das Terminieren des Algorithmus festgehalten. Dabei bedeutet die Abkürzung

- A01: Die Norm des Gradienten im letzten Iterationswert war kleiner als $\epsilon = 10^{-8}$.
- A02: Der Wert der Zielfunktion im letzten Iterationswert war kleiner als $\epsilon = 10^{-8}$.
- B01: Der Wert der Modellfunktion im Testpunkt war gleich dem im aktuellen Iterationspunkt, d.h. die Berechnung von ρ_k war nicht möglich.
- B02: Der Algorithmus konnte nach 10000 Iterationsschritten keine Lösung ermitteln.
- B03: Der Algorithmus konnte nach 6 Stunden Laufzeit keine Lösung ermitteln.

In den Fällen A01 und A02 kann das Testbeispiel als gelöst betrachtet werden, musste hingegen eine der Abbruchbedingungen B01, B02 oder B03 angewandt werden, so wurde keine geeignete Lösung des nichtlinearen Gleichungssystems gefunden.

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
A001	GN	3	3	10000	311.33	1.178e-009	B02
	L	3	3	10000	246.03	2.3194e-007	B02
	Q	3	3	44	3.11	1.0342e-013	A01
	FN	3	3	35	2.75	2.9178e-014	A01
A006	GN	2	2	17	0.141	4.0092e-018	A02
	L	2	2	10000	151.44	9.4035e-010	B02
	Q	2	2	48	1.328	7.7274e-016	A01
	FN	2	2	17	0.438	3.1029e-017	A02
A007	GN	2	2	16	0.125	1.2547e-017	A02
	L	2	2	10000	140.59	1.0846e-008	B02
	Q	2	2	31	0.875	5.5355e-017	A02
	FN	2	2	17	0.438	9.8573e-018	A02
A009	GN	2	2	12	0.094	1.6258e-016	A01
	L	2	2	10000	149.69	6.5067e-006	B02
	Q	2	2	16	0.437	2.0152e-013	A01
	FN	2	2	12	0.297	1.6258e-016	A01
A026	GN	2	3	160	2.953	1.6564e-013	A01
	L	2	3	10000	186.33	6.2706e-010	B02
	Q	2	3	31	1.453	7.1044e-013	A01
	FN	2	3	21	0.813	4.8269e-018	A02
A027	GN	2	3	19	0.203	4.6893e-015	A01
	L	2	3	10000	182.05	9.461e-012	B02
	Q	2	3	24	1.125	1.4869e-013	A01
	FN	2	3	16	0.625	2.2017e-014	A01
A028	GN	2	3	23	0.234	5.9012e-013	A01
	L	2	3	4088	69.469	3.5132e-012	A01
	Q	2	3	20	0.937	1.7749e-012	A01
	FN	2	3	10	0.375	2.512e-013	A01
A039	GN	3	4	16	0.281	1.7711e-017	A02
	L	3	4	10000	299.41	1.5553e-007	B02
	Q	3	4	28	2.875	3.6047e-017	A02
	FN	3	4	17	1.531	3.7787e-017	A02
A040	GN	4	4	11	0.25	3.5955e-018	A02
	L	4	4	10000	394.58	7.4843e-009	B02
	Q	4	4	19	2.515	1.3994e-016	A01
	FN	4	4	11	1.188	3.5955e-018	A02
A042	GN	3	4	17	0.297	1.9837e-017	A02
	L	3	4	10000	320.14	1.6479e-006	B02
	Q	3	4	35	3.578	2.8715e-017	A02
	FN	3	4	17	1.375	1.9837e-017	A02

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
A046	GN	3	5	10000	565.09	1.0368e-015	B02
	L	3	5	7937	301.09	3.1831e-011	A01
	Q	3	5	31	4.484	4.8967e-013	B01
	FN	3	5	34	4.641	5.2096e-016	A01
A047	GN	4	5	234	14.656	2.106e-017	A02
	L	4	5	10000	494.27	7.786e-010	B02
	Q	4	5	41	7.687	3.6296e-013	A01
	FN	4	5	2	0.282	1.7821	B01
A048	GN	3	5	13	0.25	2.04e-012	A01
	L	3	5	2087	70.172	2.2925e-012	A01
	Q	3	5	22	3.156	3.1541e-012	A01
	FN	3	5	13	1.438	2.04e-012	A01
A049	GN	3	5	1024	53.75	6.2559e-011	A01
	L	3	5	10000	376.14	3.2773e-010	B02
	Q	3	5	28	4.078	4.9012e-011	A01
	FN	3	5	6	0.656	0.069943	B01
A050	GN	4	5	19	0.516	2.3819e-012	A01
	L	4	5	10000	478	8.8538e-012	B02
	Q	4	5	34	6.375	1.6907e-012	A01
	FN	4	5	19	2.813	7.634e-013	A01
A051	GN	4	5	11	0.313	2.697e-012	A01
	L	4	5	10000	493.27	1.3508e-012	B02
	Q	4	5	19	3.547	1.4656e-012	A01
	FN	4	5	11	1.61	2.697e-012	A01
A052	GN	4	5	33	0.875	3.4408e-017	A02
	L	4	5	10000	510.34	3.0288e-006	B02
	Q	4	5	48	8.938	8.173e-016	A01
	FN	4	5	44	6.39	3.9513e-017	A02
A061	GN	3	3	195	3.859	6.6928e-014	A01
	L	3	3	10000	155.72	6.8676e-006	B02
	Q	3	3	40	1.875	8.2468e-014	B01
	FN	3	3	1	0.047	87.512	B01
A077	GN	3	5	29	0.453	4.1324e-018	A02
	L	3	5	10000	269.8	0.00026555	B02
	Q	3	5	40	4.031	5.9543e-017	A01
	FN	3	5	17	1.328	3.7439e-017	A02
A205	GN	3	2	17	0.187	3.0523e-017	A02
	L	3	2	6767	170.48	1.7211e-011	A01
	Q	3	2	36	1.438	4.1298e-017	A02
	FN	3	2	17	0.609	3.0523e-017	A02

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
A219	GN	3	4	21	0.359	2.7696e-017	A02
	L	3	4	327	8.843	1.8836e-016	A01
	Q	3	4	35	3.531	5.1177e-017	A02
	FN	3	4	1	0.078	35466	B01
A240	GN	3	3	17	0.25	5.0077e-013	A01
	L	3	3	59	1.265	3.8382e-012	A01
	Q	3	3	37	2.453	4.9774e-013	A01
	FN	3	3	17	0.953	1.0597e-012	A01
A241	GN	5	3	11	0.25	8.7873e-026	A02
	L	5	3	10000	423.55	5.3414e-010	B02
	Q	5	3	19	2.062	56.137	B01
	FN	5	3	19	2.078	56.232	B01
A242	GN	10	3	13	0.593	7.2405e-013	A01
	L	10	3	88	6.375	8.0768e-011	A01
	Q	10	3	15	3.219	6.1334e-011	A01
	FN	10	3	13	2.422	7.2405e-013	A01
A255	GN	7	4	18	0.703	4.6864e-017	A02
	L	7	4	10000	709.74	1.0768e-008	B02
	Q	7	4	36	8.032	2.3718e-014	A01
	FN	7	4	18	3.312	4.6864e-017	A02
A256	GN	4	4	15	0.344	1.302e-011	A01
	L	4	4	348	12.719	4.0013e-011	A01
	Q	4	4	28	3.671	1.292e-011	A01
	FN	4	4	15	1.594	1.302e-011	A01
A261	GN	5	4	11	0.313	7.5003e-012	A01
	L	5	4	99	4.625	1.0868e-010	A01
	Q	5	4	25	4.063	1.3867e-011	A01
	FN	5	4	11	1.453	7.5003e-012	A01
A267	GN	11	5	1	0.079	1.8516	A01
	L	11	5	10000	1426.7	2.3409e-008	B02
	Q	11	5	100	53.703	5.6423e-007	A01
	FN	11	5	1	0.422	1.8516	A01
A271	GN	6	6	13	0.578	1.0603e-013	A01
	L	6	6	30	2.11	5.1053e-013	A01
	Q	6	6	21	7.672	7.8509e-013	A01
	FN	6	6	13	3.703	1.0603e-013	A01
A272	GN	13	6	34	9.735	804.5	A01
	L	13	6	10000	2605	5.4355e-008	B02
	Q	13	6	48	40.187	2.6822e-011	A01
	FN	13	6	3	3.922	23.424	B01

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
A282	GN	10	10	2405	1173.4	0.27723	B01
	L	10	10	4464	1089.5	2.3664e-012	A01
	Q	10	10	1157	1995.1	0.27723	B01
	FN	10	10	10	22.25	6.4717e+024	B01
A286	GN	10	20	4859	7284.9	1.0789e-012	A01
	L	10	20	7475	2695.1	1.123e-011	A01
	Q	10	20	37	198.47	2.2586e-013	A01
	FN	10	20	31	393.73	1.8829e-014	A01
A287	GN	5	20	100	33.688	1.1344e-012	A01
	L	5	20	1068	139.48	3.1388e-012	A01
	Q	5	20	81	167.78	2.404e-013	A01
	FN	5	20	20	74.985	2.6918e-014	A01
A288	GN	5	20	2990	2348.3	7.6691e-012	A01
	L	5	20	10000	1918.8	1.1093e-010	B02
	Q	5	20	56	158.52	2.2367e-011	A01
	FN	5	20	13	26.218	4.5176e-011	A01
A294	GN	5	6	361	15.219	9.297e-014	A01
	L	5	6	2499	105.48	7.1559e-012	A01
	Q	5	6	36	7.906	5.7751e-014	A01
	FN	5	6	17	2.859	3.4053e-015	A01
A295	GN	9	10	225	17.187	1.3158e-012	A01
	L	9	10	1259	157.09	4.7679e-012	A01
	Q	9	10	101	96.141	1.7645e-013	A01
	FN	9	10	7	4.844	4.8506	B01
A296	GN	15	16	403	118.88	1.453e-013	A01
	L	15	16	1234	440.52	8.1383e-012	A01
	Q	15	16	159	590.39	5.2095e-014	A01
	FN	15	16	9	24.188	11.461	B01
A297	GN	29	30	3262	21606	1.3911e+008	B03
	L	29	30	1514	2009.8	3.3733e-012	A01
	Q	29	30	911	21609	3.8718e+009	B03
	FN	29	30	5	84.391	19.691	B01
A298	GN	49	50	801	21616	2.2115e+017	B03
	L	49	50	2412	11153	4.7248e-012	A01
	Q	49	50	202	21608	3.2566e+011	B03
	FN	49	50	6	461.05	32.532	B01
A299	GN	99	100	70	21933	2.8989e+024	B03
	L	99	100	699	21612	2611.3	B03
	Q	99	100	18	21991	2.5652e+018	B03
	FN	99	100	5	4564.3	64.659	B01

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
A352	GN	20	4	61	14.172	1.0554e-017	A02
	L	20	4	5600	1733	2.2176e-016	A02
	Q	20	4	29	20.172	6.8099e-026	A02
	FN	20	4	13	7.547	2.5624e-025	A02
A373	GN	7	9	10000	3171.3	2.265e-005	B02
	L	7	9	10000	9061.7	9341.9	B02
	Q	7	9	10000	13465	2471.7	B02
	FN	7	9	2	1.312	1.6822e+010	B01
A394	GN	21	20	8	2.438	6.2781e-005	A01
	L	21	20	419	230.3	6.2781e-005	A01
	Q	21	20	58	450.17	7.7228e+010	A01
	FN	21	20	8	44.812	6.2781e-005	A01
A395	GN	51	50	6	15.672	4.0031e-006	A01
	L	51	50	414	2027.1	4.0031e-006	A01
	Q	51	50	44	7054.8	1.8982e+005	A01
	FN	51	50	6	707.08	4.0031e-006	A01
ATB1	GN	3	3	2	0.031	1.8874e-028	A02
	L	3	3	10000	250.61	0.13073	B02
	Q	3	3	40	2.672	8.4359e-019	A02
	FN	3	3	3	0.172	2.2187e-031	A02
ATB2	GN	3	3	318	4.672	0.5	A01
	L	3	3	10000	257.44	1.6664e-011	B02
	Q	3	3	708	48.125	7.5309e-017	A01
	FN	3	3	320	18.359	0.5	A01
ATB3	GN	3	3	6	0.094	7.5501e-019	A02
	L	3	3	2693	62.532	3.822e-014	A01
	Q	3	3	12	0.828	6.4757e-023	A02
	FN	3	3	7	0.39	2.1127e-029	A02
ATB4	GN	4	4	6	0.14	4.2447e-017	A02
	L	4	4	10000	398.39	8.3444e-010	B02
	Q	4	4	21	2.875	1.4873e-017	A02
	FN	4	4	10	1.328	4.4789e-021	A02
ATB5	GN	5	5	7	0.25	2.2351e-019	A02
	L	5	5	10000	595.5	0.0013941	B02
	Q	5	5	245	59.141	1.1908e-015	A01
	FN	5	5	19	4.047	6.2748e-017	A01
ATB6	GN	6	6	7	0.313	3.6323e-016	A01
	L	6	6	10000	850.03	0.0027973	B02
	Q	6	6	1344	543.05	9.8112e-008	B01
	FN	6	6	12	4.438	0.15842	B01

Test-beispiel	Modell-funktion	Funktionen	Variablen	benötige Iterationen	benötige Zeit	$f(x^*)$	Abbruch wegen
ATB7	GN	7	7	8	0.515	2.5318e-020	A02
	L	7	7	10000	1113	0.01644	B02
	Q	7	7	10000	7946.9	0.41021	B02
	FN	7	7	13	13.672	0.33497	B01
ATB8	GN	10	10	9	1.047	3.5315e-017	A02
	L	10	10	10000	2364.4	0.012822	B02
	Q	10	10	10000	20157	0.7341	B02
	FN	10	10	7	8.203	21.511	B01
ATB9	GN	50	50	1	2.219	2.5561e-012	A01
	L	50	50	55	205.88	1.504e-012	A01
	Q	50	50	4	475.72	2.1815e-012	B01
	FN	50	50	1	110.42	2.5561e-012	A01
ATB10	GN	100	100	0	0	1.5971e-021	A02
	L	100	100	0	0	1.5971e-021	A02
	Q	100	100	0	0	1.5971e-021	A02
	FN	100	100	0	0	1.5971e-021	A02

Alle weiteren Ergebnisse der in Kapitel 5 zusammengefassten numerischen Test befinden sich auf der beiliegenden CD im Ordner "numerische Ergebnisse". Dort bezeichnen die Unterordner den jeweiligen Test. In diesen Ordnern befinden sich, als Matlab-Variablen gespeichert, die tabellarisch zusammengefassten Testergebnisse für die einzelnen Parametereinstellungen in den Dateien "Z_zusammenfassung.mat", wobei durch $Z \in \{1, ..., 30\}$ auf die Zeile der betreffenden Tabelle in Kapitel 5 und damit auf die verwendete Parametereinstellung verwiesen wird.
Außerdem ist für jede einzelne Berechnung eine Matrix gespeichert, in welcher der Verlauf der wichtigsten Größen in jeder Iteration festgehalten wurde. Diese Matrizen befinden sich in den Dateien "Z_B_M.mat", wobei B auf das betrachtete Testbeispiel und M auf die verwendete Modellfunktion verweist.

Literaturverzeichnis

[1] W. Alt. *Nichtlineare Optimierung*. Friedr. Vieweg und Sohn Verlagsgesellschaft, 2002.

[2] P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta Numerica 1995*, 4. pp. 1-51, 1995.

[3] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. Methods for nonlinear constraints in optimization calculations. In: Duff, I.S., Watson, G.A., eds., The State of the Art in Numerical Analysis, pp. 363–390, Clarendon Press, Oxford, 1997.

[4] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *Trust-region methods*. SIAM series on optimization. Society for Industrial and Applied Mathematics, 2000.

[5] J.E. Dennis and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, Philadelphia, 1996.

[6] E. Dolan and J. More. Benchmarking optimization software with performance profiles. Mathematical Programming, 91(2), pp. 201–213, 2002.

[7] M. Ehrgott. *Multicriteria Optimization*. Springer-Verlag, 2000.

[8] R. Fletcher. *Practical methods of optimization*. Wiley, Chichester [u.a.], 2. ed., repr. edition, 1996.

[9] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. Numerical Analysis Report NA/171, University of Dundee, UK, 1997. http://www.maths.dundee.ac.uk/ fletcher/.

[10] R. Fletcher, S. Leyffer, and Ph.L. Toint. A brief history of filter methods. Preprint ANL/MCS-P1372-0906, Argonne National Laboratory, Mathematics and Computer Science Division, 2006.

[11] R. Fletcher, S. Leyffer, and Ph.L. Toint. On the global convergence of a filter-SQP algorithm. *SIAM J. Optim.*, 13(1):44–59, 2002.

[12] N.I.M. Gould, S. Leyffer, and Ph.L. Toint. A multidimensional filter algorithm for nonlinear equations and nonlinear least-squares. *SIAM J. Optim.*, 15(1):pp. 17–38, 2004.

[13] N.I.M. Gould and Ph.L. Toint. Filtrane, a fortran 95 filter-trust-region package for solving nonlinear feasibility problems. Transactions of the ACM on Mathematical Software, vol. 33(1), pp. 3-25, 2007.

[14] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Springer Verlag, 1981.

[15] J. Jahn and U. Rathje. Graef-Younes method with bachward iteration. In: K.-H. Küfer, H. Rommelfanger, C. Tanner und K. Winkler (Hrsg.), Multicriteria Decision Making and Fuzzy Systems - Theory, Methods and Application, pp. 75–81, Shaker Verlag, Aachen, 2006.

[16] F. Jarre and J. Stoer. *Optimierung*. Springer Verlag, 2004.

[17] S. Saliba. Online-optimization of large-scale vehicle dispatching problems. Dissertation, Universität Kaiserslautern, 2008.

[18] K. Schittkowski. *More Test Examples for Nonlinear Programming Codes*. Springer Verlag, 1987.

[19] J. Werner. *Numerische Mathematik 1*. Friedr. Vieweg und Sohn Verlagsgesellschaft, 1992.

[20] Y.M. Younes. Studies on discrete vector optimization. Dissertation, University of Demiatta, Egypt, 1993.

Hiermit erkläre ich, dass ich diese Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Erlangen, den 5. August 2008 Markus Kaiser