Das Trust-Region-Steihaug Verfahren zur numerischen Optimierung

Bachelorarbeit

vorgelegt von

Rau Mika

an der

Universität Konstanz

Studiengang Finanzmathematik

Fachbereich Mathematik und Statistik

Betreut durch Prof. Dr. Stefan Volkwein

Konstanz, 2020

Konstanzer Online-Publikations-System (KOPS)
URL: http://nbn-resolving.de/urn:nbn:de:bsz:352-2-1hvelaz07gc0z3

Inhaltsverzeichnis

1 Einleitung **2**

1.1 Grundlagen der Optimierung 2

1.2 Grundlagen der Numerik 3

2 Das Trust-Region Verfahren und der Steihaug-Ansatz **4**

2.1 Das Trust-Region Verfahren 4

2.2 Der Steihaug-Algorithmus 8

2.3 Konvergenz 12

3 Andere Verfahren **15**

3.1 Das Gradienten-Verfahren 16

3.2 Das globalisierte Newton-Verfahren 16

4 Anwendung **18**

5 Fazit **25**

6 Anhang **26**

1 Einleitung

Die Menschheit hat sich stets mithilfe von Optimierungen weiterentwickelt. Prozesse müssen schneller werden, Werkstoffe leichter und Aktiendepots profitabler. Darum soll es in dieser Arbeit gehen, aber nicht die Anwendungen in der realen Welt, sondern die Basis dazu- die Algorithmen. Spezieller wird das Trust-Region Verfahren in Verbindung mit dem Steihaug-Ansatz behandelt. Diese Kombination wird zur Minimierung von mehrdimensionalen Funktionen genutzt.
Während einfache Liniensuchverfahren einen geringen Rechenaufwand haben, stoßen diese bei komplizierten Funktionen schnell an ihre Grenzen. Newton-Verfahren liefern bei diesen Problemen zwar bessere Ergebnisse, doch vor allem bei hochdimensionalen Funktionen benötigen diese Verfahren Unterstützung bei der Lösung von Gleichungssystemen. Das Trust-Region Verfahren, welches zuerst unter dem Namen "Maximization by Qudadratic Hill-Climbing"[2] veröffentlicht wurde, bildet die Grundlage und ermöglicht die Bewertung einer Approximation an die zu minimierende Funktion. Der Steihaug-Ansatz nutzt dazu eine Taylor-Approximation zweiten Grades, da diese einfacher zu optimieren ist. Die Idee geht zurück auf den norwegischen Mathematiker Trond Steihaug [1].

1.1 Grundlagen der Optimierung

Es sei die zu minimierende Funktion geben durch

$$f : \mathbb{R}^n \longrightarrow \mathbb{R} \qquad \text{mit } n \in \mathbb{N}$$

und es gelte zusätzlich, dass f zweimal stetig differenzierbar ist. Dadurch wird die Stetigkeit und eine gewisse Glätte der Funktion sichergestellt. Es wird das Ziel sein, für f eine lokale Minimalstelle zu finden, also ein $x \in \mathbb{R}^n$ für das folgende Behauptung gilt

$$\exists \mathrm{U} \subset \mathbb{R}^n : \forall \tilde{x} \in \mathrm{U} : f(x) \leq f(\tilde{x})$$

wobei U eine Umgebung von x ist. Falls ein lokales Maximum gesucht werden sollte, ist diese Fragestellung identisch zur Suche einer lokalen Maximalstelle von $-f$.
Sei weiter ein $x_0 \in \mathbb{R}^n$ gegeben. Dies ist der Startpunkt für die iterative Suche des lokalen Minimums.
Das hinreichende Optimalitätskriterium zweiter Ordnung besagt, dass aus

$$\nabla f(x^*) = 0 \quad \text{und} \quad \nabla^2 f(x^*) \succ 0$$

bereits folgt, dass es sich bei x^* um ein lokales Minimum handelt. Hierbei beschreibt $\nabla f(x^*)$ den Gradienten der Funktion und $\nabla^2 f(x^*) \succ 0$ die positive Definitheit der Hesse-Matrix, ausgewertet an der Stelle x^*.

1.2 Grundlagen der Numerik

Da das Thema dieser Arbeit nicht nur ausschließlich theoretisch angegangen wird, sondern zusätzlich auch die praktische Anwendbarkeit betrachtet wird, werden nun einige Grundlagen dazu erläutert. Die Numerik ist ein Teilgebiet der Mathematik und beschäftigt sich mit dem näherungsweise Lösen von mathematischen Problemen mit Hilfe von Algorithmen.
Der Algorithmus, welcher später behandelt wird, wurde in Python verfasst. Eine Implementierung in andere Programme, wie MATLAB, ist ebenfalls denkbar. Zudem wird auch ein Pseudocode zur Verfügung gestellt.
Während in der Analysis meist Probleme entweder exakt gelöst werden oder nur die Existenz beziehungsweise nur Abschätzungen von Lösungen gegeben werden kann, liegt der Fokus der Numerik auf der Anwendbarkeit. Es sind somit exakte Ergebnisse erwünscht. Gleichzeitig muss das Problem aber von einem Algorithmus mit möglichst geringem Aufwand lösbar sein. Die Numerik, im Speziellen die Algorithmen, lebt von der Abwägung zwischen der Genauigkeit des Ergebnisses und dem steigenden Aufwand.
Dabei ist der Rechenaufwand für jeweils einen Schritt und die gesamt Anzahl der Iterationen wichtig. Während einfache Verfahren oft einen geringen Aufwand pro Schritt verursachen, sind diese meist nicht sehr genau, was dazu führt, dass mehr Schritte berechnet werden müssen. Gleichzeitig kann es aber auch passieren, dass zwar sehr genaue Schritte berechnet werden können, diese aber aufgrund von Modellannahmen und dem enormen Rechenaufwand nicht mehr praktikabel sind. Dieses Thema wird am Ende der Arbeit nochmal aufgegriffen.
Ein weiterer wichtiger Punkt ist die Rechenweise von Programmen. Zahlen werden in Bits gespeichert, dies kann zu Rundungsfehlern führen. Diese fallen zunächst nicht direkt auf und sind nicht relevant. Aber bei Brüchen, in denen der Nenner extrem klein ist, oder auch bei der Multiplikation von sehr kleinen Zahlen, kann dies zu instabilen Berechnungen führen. Zusätzlich arbeitet Python mit einer begrenzten Anzahl an Nachkommastellen, was dazu führen kann, dass nur ein gewisser Grad an Genauigkeit erreicht werden kann.
Ähnliches gilt für Abschnitt **1.1** , dort wird gefordert, dass der Gradient exakt den Wert 0 annimmt. Dies ist für die meisten Probleme und Algorithmen unrealistisch. Daher wird als Bedingung für einen solchen Punkt nur gefordert, dass die euklidische Norm des Gradienten einen Wert annimmt, welcher kleiner als einen gegebene Abbruchbedingung ist. Dieser wird für gewöhnlich als ε beschrieben und ermöglicht die Genauigkeit der Lösung zu steuern. Dadurch wird auch der benötigte Aufwand beeinflusst.

2 Das Trust-Region Verfahren und der Steihaug-Ansatz

Das Trust-Region Verfahren ist der Kern dieser Arbeit und geht zurück auf ein Paper von Goldfeld, Quandt und Trotter. In "Maximization by Quadratic Hill-Climbing"[2] wurde die Grundidee dazu gelegt. 1982 fasste Sorensen [4] verschiedene Modifikationen zusammen, das Verfahren wurde unter dem Namen Trust-Region bekannt.
Aufgabe des Verfahrens ist es, eine mehrdimensionale Funktion, wie in Abschnitt **1.1** erläutert, zu minimieren. Da diese in der Regel komplex sind, wird die Funktion approximiert. Im Folgenden wird dazu die Taylor-Reihe verwendet. Diese Approximation zweiten Grades ist leichter zu optimieren. Dies ist Aufgabe des Steihaug-Algorithmus und wird in Abschnitt **2.2** erläutert. Die Lösung wird gefunden, indem auf eine Mischung aus Liniensuch- und Newton-Conjugate-Gradient-Verfahren zurückgegriffen wird. Da Approximation und Funktion aber nicht identisch sind, sondern nur eine gewisse Ähnlichkeit besitzen, muss die gefundene Iteration in einem Bereich um den Ausgangspunkt der Approximation liegen. Dieser wird als Trust-Region bezeichnet. Anschließend wird überprüft, ob die gefundene Iteration auch als Lösung für f sinnvoll ist.

2.1 Das Trust-Region Verfahren

Seien eine Funktion f und ein Startpunkt x_0, wie in Abschnitt **1.1** beschrieben, gegeben. Mithilfe der Taylor-Reihe von f an x_s kann die Funktion, für alle $s \in \mathbb{N}_0$, durch eine Funktion zweiten Grades approximiert werden. Diese wird als t_s definiert, und es gilt

$$t_s(p) = f(x_s) + g_s^T p + \frac{1}{2} p^T H_s p \tag{1}$$

mit $p \in \mathbb{R}^n$. Zusätzlich ist gefordert, dass $g_s = \nabla f(x_s)$ und $H_s = \nabla^2 f(x_s)$. Theoretisch ist es auch möglich, dass es sich bei H_s nur um eine Approximation der Hesse-Matrix handelt. Dieser Fall wird im Folgenden aber nicht behandelt. Somit ist t_s die Taylor-Approximation zweiten Grades von f an der Stelle x_s.
Da es sich hierbei um eine quadratische Funktion handelt, ist diese leichter zu minimieren, als f. Dies wird in Abschnitt **2.2** behandelt.
Es gilt unabhängig vom gewählten f und x_s immer

$$t_s(0) = f(x_s),\ \nabla t_s(0) = \nabla f(x_s) \text{ und } \nabla^2 t_s(0) = \nabla^2 f(x_s)$$

Die Funktionswerte und ersten beiden Ableitungen stimmen am Stützpunkt überein. Dennoch lässt sich keine allgemeingültige Aussage über den weiteren Verlauf von f treffen. Da aber t_s ebenfalls stetig differenzierbar ist, sind t_s und f in einem Bereich um x_s ähnlich. Dies beruht auf der Taylor-Entwicklung und Konvergenz des Restterms, also der Differenz

zwischen f und t_s, gegen 0 für $p \to 0$.
Sei ein $\Delta_s > 0$ gegeben, dann beschreibt Γ_s, mit

$$\Gamma_s = \{\delta \in \mathbb{R}^n : \|\delta\|_2 \leq \Delta_s\}$$

eine n-dimensionale Kugel. Diese besitzt als Mittelpunkt den Ursprung und Radius Δ_s. Für alle $p \in \Gamma_s$ wird angenommen, dass

$$f(x_s + p) \approx t_s(p)$$

gilt. Der Algorithmus erhält somit seinen Namen, da darauf vertraut wird, dass diese Abschätzung für den Bereich stimmt. Da aber diese Eigenschaft von vielen Faktoren abhängt, wird bei jedem Schritt des Algorithmus abgeschätzt, ob diese Annahme noch realistisch ist oder ob der Radius verändert werden muss.
Zuerst werden einige Begriffe erörtert und definiert.

- $x_0 \in \mathbb{R}^n$ ist der Startpunkt der Suche.
- p_s ist der s-te Optimierungsschritt.
- x_s beschreibt die s-te Iterierte, und es gilt $x_s = x_{s-1} + p_{s-1}$ für $s \geq 1$.
- $\Delta_{\text{Limit}} > 0$ dient als Beschränkung und soll Radien von Γ_s über diesem Wert verhindern.
- $\varepsilon > 0$ ist die Abbruchbedingung wie beschreiben in Abschnitt **1.2**.
- $\Delta_0 \in (0, \Delta_{\text{Limit}}]$ ist der Radius der ersten Trust-Region.
- Tmax $\in \mathbb{N}$ ist die maximale Schrittanzahl.
- $\eta \in (0, \frac{1}{4})$ bestimmt, wie sensitiv auf Unterschiede von f und t_s reagiert wird.

Sei

$$\tilde{p}_s = \arg\min_{p \in \Gamma_s} f(x_s) + g_s^T p + \frac{1}{2} p^T H_s p$$

die Lösung zu (1), welche in Γ_s liegt. Daraus folgt

$$\begin{aligned} & t_s(\tilde{p}_s) \leq t_s(0) \\ \Leftrightarrow \quad & 0 \leq t_s(0) - t_s(\tilde{p}_s) \end{aligned} \tag{2}$$

$t_s(\tilde{p}_s)$ ist durch diesen Schritt kleiner als $t_s(0)$, doch da es sich bei t_s nur um eine Approximation für f handelt, ist dies nur die zu erwartende Minimierung durch den s-ten

Schritt. Diese wird definiert durch

$$\mathrm{PredR}_s = t_s(0) - t_s(\tilde{p}_s)$$

während die tatsächliche Minimierung durch

$$\mathrm{ActR}_s = f(x_s) - f(x_s + \tilde{p}_s)$$

bestimmt wird.
Da aber beide Ergebnisse keinen Aufschluss auf die Ähnlichkeit der Funktionen geben oder ob die gefundene Iteration für f eine Minimierung bringt, wird das relative Verhältnis betrachtet und definiert durch

$$\rho_s = \frac{\mathrm{ActR}_s}{\mathrm{PredR}_s} = \frac{f(x_s) - f(x_s + \tilde{p}_s)}{t_s(0) - t_s(\tilde{p}_s)} \tag{3}$$

Aus (2) folgt, dass $\mathrm{PredR}_s > 0$. Sollte ρ_s daher negativ oder sehr klein sein, bedeutet dies, dass die Übertragung des Schrittes von t_s zu f nicht effizient ist oder es möglicherweise sogar zu einer Vergrößerung geführt hat. Der folgende Algorithmus beschreibt, wie die Trust-Region in Abhängigkeit von ρ_s verändert wird. Dadurch ergibt sich folgender Algorithmus [6].

Algorithm 1 Trust-Region Algorithmus

```
Sei x_0 ∈ ℝ^n, Δ_Limit > 0, Δ_0 ∈ (0, Δ_Limit), η ∈ (0, 1/4), ε > 0, Tmax ∈ ℕ
for s ∈ ℕ_{0∪≤Tmax} do
    if ‖∇f(x_s)‖_2 < ε then
        return x_s
    end if
    Berechne p_s als Minimum von (1) mit p_s ∈ Γ_s
    und ρ_s aus (3)
    if ρ_s < 1/4 then
        Δ_{s+1} = 1/4 Δ_s
    else if ρ_s > 3/4 und ‖p_s‖_2 = Δ_s then
        Δ_{s+1} = min(2Δ_s, Δ_Limit)
    else
        Δ_{s+1} = Δ_s
    end if
    if ρ_s > η then
        x_{s+1} = x_s + p_s
    else
        x_{s+1} = x_s
    end if
end for
return x_s
```

Sei

$$\tilde{p}_s = \arg\min_{p \in \Gamma_s} f(x_s) + g_s^T p + \frac{1}{2} p^T H_s p$$

die Lösung zu (1), welche in Γ_s liegt. Dann gilt für die nächste Iteration $x_{s+1} = x_s + \tilde{p}_s$. Zuerst wird überprüft, ob

$$\|\nabla f(x_{s+1})\|_2 < \varepsilon$$

gilt. Ist dies der Fall, handelt es sich um einen stationären Punkt. Dieser wird als Lösung ausgegeben und der Algorithmus wird beendet. Falls nicht, muss überprüft werden, ob beide Funktionen in diesem Bereich ähnlich genug sind. Hierfür wird abgeschätzt, ob der Schritt $\tilde{p}_s$ auch für die Minimierung von f sinnvoll ist und sich übertragen lässt.
Hierfür wird PredR_s und ActR_s genutzt. Da beide Ergebnisse alleine keine große Aussagekraft besitzen, beschreibt ρ_s die relative Effektivität der Übertragung des s-ten Schrittes. Wenn $\rho_s < \frac{1}{4}$, ist die tatsächliche Minimierung der Funktion f kleiner als die Erwartete oder möglicherweise erhöht sich f sogar durch die Iteration. Denn in (2) und (3) wurde bereits gezeigt, dass $\mathrm{Pred}_s > 0$ gilt. Somit folgt aus $\rho \leq 0$ folgendes:

$$\begin{aligned}
\rho_s &\leq 0 \\
\frac{\mathrm{ActR}_s}{\mathrm{PredR}_s} &\leq 0 \\
\mathrm{ActR}_s &\leq 0 \\
f(x_s) - f(x_s + \tilde{p}_s) &\leq 0 \\
f(x_s) &\leq f(x_s + \tilde{p}_s)
\end{aligned}$$

Es konnte somit keine Verringerung von f erreicht werden. Sollte dies der Fall sein oder ρ_s klein sein, kann der Trust-Region nicht vertraut werden, die Funktionen sind an der Stelle x_{s+1} zu unterschiedlich. Folglich ist Γ_s zu groß und in diesem Fall gilt

$$\Delta_{s+1} = \frac{1}{4}\Delta_s$$

Falls gilt $\rho_s > \frac{3}{4}$ ist die tatsächliche Minimierung ähnlich groß oder sogar größer als die Erwartete. Daraus lässt sich aber nicht direkt auf eine Ähnlichkeit schließen. Falls zusätzlich aber $\|\tilde{p}_s\|_2 = \Delta_s$ gilt, ist nicht nur die Effizienz des Schrittes gut, sondern der Radius der Trust-Region wurde vollkommen ausgeschöpft. Daher ist es sinnvoll Γ_s zu vergrößern, um die mögliche Schrittweite zu vergrößern und somit die Zahl der benötigten Iterationen zu minimieren. Dadurch wird versucht, den Gesamtaufwand gering zu halten. Es gilt dann

$$\Delta_{s+1} = \min(2\Delta_s, \Delta_{\mathrm{Limit}})$$

wobei Δ_{Limit} als Beschränkung dient, um zu extreme Werte für Δ_s ausschließen.

Sollte keiner der beiden Fälle eintreten oder $\tilde{p}_s$ nicht die gesamte Trust-Region ausnutzen, wird diese nicht verändert, und es gilt

$$\Delta_{s+1} = \Delta_s$$

Nachdem der Radius von Γ_{s+1} aktualisiert wurde, wird nun überprüft, ob die gefundene Iteration

$$x_{s+1} = x_s + \tilde{p}_s$$

akzeptiert wird. Dazu ist $\eta \in (0, \frac{1}{4})$ gegeben. Wenn $\rho_s > \frac{1}{4}$, folgt daraus, dass $\rho_s > \eta$. Die Iteration wird angenommen und der Prozess beginnt erneut mit neuem Radius Δ_{s+1} und t_{s+1} als neue Approximation an der Stelle x_{s+1}.
Sollte $\rho_s \in (\eta, \frac{1}{4}]$ gelten, hat sich f zwar durch den gefundenen Schritt verkleinert und wird übernommen, dadurch wird der Gesamtaufwand gering gehalten. Dennoch wird die Trust-Region wie oben gezeigt verkleinert, um im nächsten Schritt ein konservativeres Ergebnis zu erhalten. Die Funktionen sind schlichtweg nicht ähnlich genug. Der nächste Schritt wird mit dem aktualisierten Radius Δ_{s+1} berechnet.
Sollte $\rho_s \leq \eta$ gelten, ist das gefundene Ergebnis nicht akzeptabel, daher wird die Trust-Region verkleinert und es wird eine neuen Lösung zu (1) gesucht, welche in Γ_{s+1} liegt.
Dieser Vorgang wird fortgeführt, bis ein Schritt mit $\rho > \eta$ gefunden wird. Doch da es sich um einen numerischen Algorithmus handelt, welcher seine Anwendung auf Computern finden soll, ist es auch möglich, dass der Algorithmus extrem langsam konvergiert. Die Variable Tmax stellt sicher, dass der Algorithmus abgebrochen wird, und x_{Tmax} als beste gefundene Lösung ausgegeben wird.

2.2 Der Steihaug-Algorithmus

In vorherigen Abschnitt wurde $\tilde{p}_s$, eine Lösung zu (1), als gegeben angenommen. Im Folgenden wird beschrieben, wie mit Hilfe des Steihaug-Ansatz eine solche Lösung gefunden werden kann. Dieser geht zurück auf den norwegischen Professor Trond Steihaug und greift auf verschiedene Verfahren zurück. Er nutzt Liniensuch- und das Conjugate-Gradient-Newton-Verfahren, um das vorhandene Cauchy-Verfahren zu verbessern und erweitern. Dies liefert Vorteile bei der Berechnung, dem Aufwand und der Genauigkeit des Ergebnisses.
Für eine bessere Notation wird folgendes definiert:

$$\mathbb{N}_{0\cup<n} = \mathbb{N}_0 \cup \mathbb{N}_{<n}$$

Die zu minimierende Funktion wird an der Stelle x_s durch die Taylor-Reihe approximiert. Wie schon in Abschnitt **2.1** ist diese zweiten Grades und wird durch folgenden Ausdruck beschrieben

$$t_s(p) = f(x_s) + \nabla f(x_s)^T p + \frac{1}{2} p^T \nabla^2 f(x_s) p$$

Daraus ergeben sich folgende Eigenschaften

$$\begin{aligned} \nabla t_s(p) &= \nabla f(x_s) + \nabla^2 f(x_s) p \\ \nabla^2 t_s(p) &= \nabla^2 f(x_s) \end{aligned}$$

sowie

$$\begin{aligned} \nabla t_s(0) &= \nabla f(x_s) =: \nabla f_s \\ \nabla^2 t_s(0) &= \nabla^2 f(x_s) =: \nabla^2 f_s \end{aligned}$$

Sei nun ein $\delta_s > 0$ als Abbruchbedingung gegeben. Denkbar ist $\delta_s = 10^-9$, wie in (1.2) eingeführt. Eine andere Möglichkeit wird von [6, S. 169 Algorithmus 7.1] vorgeschlagen.

$$\delta_s = \min(0.5, \sqrt{\|\nabla f_s\|_2}) \|\nabla f_s\|_2$$

Dabei orientiert sich δ_s für jeden Schritt individuell an der aktuellen Norm des Gradienten. Dadurch wird eine schneller Konvergenz erzielt [6, S. 171]. Dies wird in dieser Arbeit verwendet.
Für eine bessere Übersicht werden diverse Variablen definiert.

$$z_0 = 0 \qquad r_0 = \nabla f_s \qquad d_0 = -\nabla f_s$$

Zuerst wird überprüft, ob t_s konvex in Richtung d_0 ist. Gilt $\nabla f_s^T \nabla^2 f_s \nabla f_s \leq 0$, ist dies nicht der Fall, die Funktion fällt in Richtung d_0 dauerhaft ab. Es existiert somit kein stationärer Punkt in diese Richtung. Folglich wird eine Schrittweite gewählt, welche die Trust-Region voll ausnutzt. Dazu wird der Gradient auf die Länge eins normiert, um diesen dann mit dem Radius Δ_s zu multiplizieren.

$$\tilde{p}_s = -\nabla f_s \frac{\Delta_s}{\|\nabla f_s\|_2}$$

Es wurde der bestmögliche Schritt gefunden und der Algorithmus wird beendet.
Ist t_s in Richtung d_0 konvex, wird die Suche nach der Minimalstelle gestartet. Diese ist iterativ und wird maximal n-mal wiederholt. Dabei ist n die Dimension des Definitionsbereiches. Sei $k \in \mathbb{N}_{0 \cup < n-1}$, zu Beginn wird die Konvexität der Approximation in

Algorithm 2 CG-Steihaug Algorithmus

Sei der Startpunkt x_s gegeben
Setze $z_0 = 0$, $r_0 = \nabla f_s$, $d_0 = -\nabla f_s$, $n =$ Dimension des Definitionsbereichs
Berechne $\delta_s = \min(0.5, \sqrt{\|\nabla f_s\|_2}) \cdot \|\nabla f_s\|_2$
if $\nabla f_s^T \nabla^2 f_s \nabla f_s \leq 0$ **then**
  $p_s = -\nabla f_s \frac{\Delta_s}{\|\nabla f_s\|_2}$
  return p_s
end if
for $k \in \mathbb{N}_{0 \cup < n}$ **do**
  if $d_k^T \nabla^2 f_s d_k \leq 0$ **then**
    Finde $\tau \in \mathbb{R}_{>0}$, so dass $t_s(p_s)$ durch $p_s = z_k + \tau d_k$ minimiert und $\|p_s\|_2 = \Delta_s$ gilt
    return p_s
  end if
  $\alpha_k = \frac{r_k^T r_k}{d_k^T \nabla^2 f_s d_k}$
  $z_{k+1} = z_k + \alpha_k d_k$
  if $\|z_{k+1}\|_2 \geq \Delta_s$ **then**
    Finde $\tau \in \mathbb{R}_{>0}$, so dass $p_s = z_k + \tau d_k$ und $\|p_s\|_2 = \Delta_s$ gilt
    return p_s
  end if
  $r_{k+1} = r_k + \alpha_k \nabla^2 f_s d_k$
  if $\|r_{k+1}\|_2 < \delta_s$ **then**
    $p_s = z_{k+1}$
    return p_s
  end if
  $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$
end for
$p_s = z_n$
return p_s

Richtung d_k untersucht. Sollte diese nicht gegeben sein, wird dies behandelt wie die erste **if**-Bedingung. Der Unterschied ist, dass bereits durch z_k eine Iteration der Minimalstelle gefunden wurde. Dennoch ist es das Ziel, einen möglichst großen Schritt zu gehen. Es wird nun ein $\tau \in \mathbb{R}$ gesucht, dass folgende Eigenschaften erfüllt.

$$\min_{\tau>0} t_s(z_k + \tau d_k) = \Psi(\tau) \qquad \text{mit } \|z_k + \tau d_k\|_2 = \Delta_s$$

Die Approximation soll abhängig von τ minimiert werden, wobei die Lösung noch in der Trust-Region liegen muss. Da Ψ in τ fällt, muss die Lösung auf dem positiven Rand liegen. Umformungen ergeben

$$\begin{aligned}
\Delta_s &= \|p_k\|_2 \\
\Delta_s^2 &= \|z_k + \tau d_k\|_2^2 \\
\Delta_s^2 &= (z_k + \tau d_k)^T (z_k + \tau d_k) \\
\Delta_s^2 &= z_k^T z_k + \tau d_k^T z_k + \tau z_k^T d_k + \tau^2 d_k^T d_k \\
0 &= \tau^2 \cdot \|d_k\|_2^2 + 2\tau z_k^T d_k + \|z_k\|_2^2 - \Delta_s^2
\end{aligned}$$

Diese quadratische Gleichung kann gelöst werden und es gilt

$$\tau_{1,2} = \frac{-2 \cdot z_k^T d_k + \sqrt{(-2 \cdot z_k^T d_k)^2 - 4 \cdot \|d_k\|_2^2 \cdot (\|z_k\|_2^2 - \Delta_s^2)}}{2 \cdot \|d_k\|_2^2}$$

In [1, Theorem 2.1] wird gezeigt, dass $z_k^T d_k > 0$ und $\tau > 0$ gelten, somit kommt für τ nur ein Lösung in Frage, obwohl theoretisch 2 möglich sind.
Ist die Approximation für die Suchrichtung konvex, wird die Schrittweite α_k und die neue Iterierte z_{k+1} berechnet. Diese orientieren sich am Conjugate-Gradient-Verfahren (CG-Verfahren), welches in [7, Abschnitt 5] näher erläuter wird. Dabei sind die verschiedenen Suchrichtungen d_k H_s-konjugiert. Das bedeutet

$$d_i^T H_s d_j = 0 : \forall i \neq j$$

Dabei hat [8, S. 22] eine gute Erklärung verfasst. Bei einer quadratischen Funktion, wie t_s, ergeben die Konturlinien eine Ellipse. Zwei Vektoren, die H_s-konjugiert sind, sind in dieser Darstellung nicht rechtwinklig aufeinander. Verzerrt man den Graphen, sodass die Konturlinien Kreise bilden, dann stehen diese Vektoren rechtwinklig aufeinander. Eine weitere Eigenschaft ist die lineare Unabhängigkeit [6, Theorem 5.3]. Dadurch spannen $d_0, \ldots, d_{n-1}$ den gesamten Definitionsbereich $\mathbb{R}^n$ auf, und es werden für den Prozess maximal n verschiedene Richtungen und somit Wiederholungen benötigt.
Sollte z_k bereits außerhalb der Trust-Region liegen, wird der Algorithmus gestoppt und äquivalent zum Abbruch bei fehlender Konvexität, τ berechnet. Dabei liegt die Lösung

wieder auf dem Rand, und $\tau > 0$ muss gelten.
Liegt z_k noch in der Trust-Region, wird das Residual r_k berechnet. Dieses gibt ungefähr an, wie weit die Lösung z_k von der Minimalstelle t_s entfernt ist. Ist die Norm kleiner als die Abbruchbedingung δ_s, wurde eine Lösung gefunden und der Algorithmus wird beendet.
Ist das Residual noch zu groß wird eine neue Richtung berechnet und der Prozess beginnt erneut. Dabei sollte nach n Durchläufen spätestens eine Lösung gefunden sein. Da es sich aber um einen numerischen Prozess handelt, bei dem Rundungs- und andere Fehler auftreten können, wird die letzte Lösung ausgegeben, falls nach n Berechnungen die Abbruchbedingung nicht eintritt.

2.3 Konvergenz

Im folgenden Abschnitt wird die Konvergenz des Trust-Region-Verfahrens und Steihaug-Algorithmus gezeigt. Der Beweis orientiert sich an [6, Theorem 7.3] und [7, Abschnitt 3 und 5].
Zuerst werden wichtige Eigenschaften des Steihaug-Verfahrens und der generierten Richtungen gezeigt. Diese orientieren sich am CG-Verfahren wie in [7, Abschnitt 5]. Induktiv wird dort gezeigt, dass folgende Eigenschaften gelten.
Seien die Voraussetzungen, wie in (2) beschreiben, gegeben. Zum besseren Verständnis werden die genutzten Begriffe aufgelistet.

$$\alpha_k = \frac{r_k^T r_k}{d_k^T H d_k} \tag{4}$$

$$z_{k+1} = z_k + \alpha_k d_k \tag{5}$$

$$r_{k+1} = r_k + \alpha_k H d_k \tag{6}$$

$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{7}$$

$$d_{k+1} = -r_{k+1} + \beta_{k+1} d_k \tag{8}$$

mit

$$g := \nabla f_s,\ H := \nabla^2 f_s,\ r_0 = g = -d_0,\ z_0 = 0$$

Dann gelten

$$
\begin{aligned}
r_i^T r_j &= 0 && \forall i,j \in \mathbb{N}_{0\cup\leq n},\ i \neq j && (9)\\
d_i^T H d_j &= 0 && \forall i,j \in \mathbb{N}_{0\cup\leq n},\ i \neq j && (10)\\
d_i^T r_j &= 0 && \forall i,j \in \mathbb{N}_{0\cup\leq n},\ i < j && (11)\\
d_i^T r_j &= -\|r_i\|_2^2 && \forall i,j \in \mathbb{N}_{0\cup\leq n},\ i \geq j && (12)\\
r_i^T H p_i &= -d_i^T H d_i && \text{und} && (13)\\
r_i^T H p_j &= 0 && \forall i,j \in \mathbb{N}_{0\cup\leq n},\ i \neq j,\ i \neq j+1 && (14)
\end{aligned}
$$

In dieser Arbeit werden nur Aussage (10) und (11) bewiesen. Die anderen Beweise sind ähnlich zu diesen, wurde aber schon ausführlich von [7] und [6] bewiesen.
Es wird folgende Hilfsaussage aus [7, Gleichung 5:2] zitiert.

$$
d_k = -\|r_k\|_2^2 \sum_{j=0}^{k} \frac{r_j}{\|r_j\|_2^2} \tag{15}
$$

Die Aussagen (10) und (11) werden nun induktiv bewiesen.
Zum Induktionsanfang sei $i = 0$ und $j = 1$. Dann lässt sich durch Anwendung der Definitionen (4)-(8) folgern, dass

$$
\begin{aligned}
\beta_1 g^T H g &= \frac{r_1^T r_1}{r_0^T r_0} g^T H g && \text{folgt mit (6)}\\
&= \frac{g^T g - 2\alpha_0 g^T H g + \alpha_0^2 g^T H H g}{g^T g} g^T H g && \text{folgt mit (4)}\\
&= \frac{g^T g - 2\frac{g^T g}{g^T H g} g^T H g + \alpha_0^2 g^T H H g}{g^T g} g^T H g\\
&= \frac{-g^T g + alpha_0^2 g^T H H g}{g^T g} g^T H g && \text{folgt mit (4)}\\
&= -g^T H g + \frac{g^T g}{g^T H g}^2 g^T H H g \frac{g^T H g}{g^T g}\\
&= -g^T H g + \alpha_0 g^T H H g
\end{aligned}
$$

Mit diesem Zwischenergebnis folgt

$$
\begin{aligned}
d_0^T H d_1 &= -g^T H(-r_1 - \beta_1 g)\\
&= -g^T H(-(r_0 + \alpha_0 H d_0) - \beta_1 g)\\
&= g^T H g - \alpha_0 g^T H H g + \beta_1 g^T H g\\
&= g^T H g - \alpha_0 g^T H H g - g^T H g + \alpha_0 g^T H H g\\
&= 0
\end{aligned}
$$

und

$$\begin{aligned} d_0^T r_1 &= -g^T(r_0 - \alpha_0 H g) \\ &= -g^T g + \frac{g^T g}{g^T H g} g^T H g \\ &= 0 \end{aligned}$$

Somit ist der Induktionsanfang für (10) und (11) bewiesen.
Gelten die Aussagen für alle $r_0, \ldots, r_{k-1}, d_0, \ldots, d_{k-1}$. Sei $i < k$ dann gilt mit (15)

$$\begin{aligned} r_i^T d_k &= -\|r_k\|_2^2 r_i^T \sum_{j=0}^{k} \frac{r_j}{\|r_j\|_2^2} && \text{mit (9) folgt} \\ &= -\|r_k\|_2^2 \end{aligned}$$

Mit diesem Zwischenergebnis und (6) folgt

$$\begin{aligned} d_k^T r_{i+1} &= d_k^T (r_i + \alpha_i H d_i) \\ \|r_k\|_2^2 &= \|r_k\|_2^2 + \alpha_i d_k^T H d_i \\ 0 &= d_k^T H d_i \end{aligned}$$

Somit ist (10) bewiesen. Sei für (11) ein $i < k$ gegeben, mit der Induktionsvoraussetzung folgt

$$\begin{aligned} d_i^T r_k &= d_i^T (r_{k-1} + \alpha_{k-1} H d_{k-1}) \\ &= d_i^T r_{k-1} + \alpha_{k-1} d_i^T H d_{k-1} && \text{gilt } i \neq k-1 \text{ folgt mit (10) und (11) direkt} \\ &= 0 \end{aligned}$$

gilt $i = k - 1$ dann folgt

$$\begin{aligned} &= d_i^T r_i + \alpha_i d_i^T h d_i && \text{mit (4) und (12)} \\ &= -r_i^T r_i + r_i^T r_i \\ &= 0 \end{aligned}$$

Mit den Aussagen (9)-(14) wird nun folgender Satz bewiesen [6, vgl. Theorem 7.3]. Dieser besagt, dass die berechneten Schritte z_i in ihrer Länge streng monoton wachsen, aber der finale Schritt Γ_s nicht verlässt.

$$0 = \|z_0\|_2 < \ldots < \|z_i\|_2 < \|z_{i+1}\|_2 < \ldots < \|p_s\|_2 \leq \Delta_s$$

Sei $j \in \mathbb{N}_{0\cup\leq n}$. Terminiert Algorithmus **2**, da $d_j^T \nabla^2 f_s d_j \leq 0$, dann gilt $\|p_s\|_2 = \|z_j + \tau d_j\|_2 = \Delta_s$. Gleiches gilt für den Fall, dass $\|z_j\|_2 \geq \Delta_s$, da die letzte Schrittweite so gekürzt wird, dass der Algorithmus auf dem Rand von Γ_s terminiert.
Folgende Aussage wurde bereits von [6, S. 173] induktiv bewiesen:

$$z_j^T d_j > 0 \qquad \forall j \in \mathbb{N}_{0\cup\leq n} \tag{16}$$

Damit kann gefolgert werden, dass

$$\begin{aligned} \|z_{j+1}\|_2^2 &= \|z_j + \alpha_j d_j\|_2^2 && \text{mit (5)} \\ &= \|z_j\|_2^2 + 2\alpha_j z_j^T d_j + \alpha_j^2 \|d_j\|_2^2 \end{aligned}$$

Da $\alpha_j > 0$, (16) und $\|d_j\|_2^2 > 0$, gilt die folgende Aussage:

$$\|z_{j+1}\|_2 > \|z_j\|_2$$

Terminiert der Algorithmus **2** bei der ersten Iteration, handelt es sich um den gleichen Schritt wie beim Cauchy-Verfahren [6, S. 71]. Für dieses kann abgeschätzt werden [6, Abschnitt 4.2], dass für ein $\lambda \in (0, 1]$ gilt

$$t_s(0) - t_s(\tilde{p}_s) \geq \lambda \|g_s\|_2 \min\left(\Delta_s, \frac{\|g_s\|_2}{\|H_s\|_2}\right)$$

Diese Abschätzung gilt für das Cauchy-Verfahren. Mit Abschnitt **2.3** wurde gezeigt, dass das Trust-Region-Steihaug-Verfahren (TRS-Verfahren) Iterationen liefert, welche mindestens so gut sind. Daher konvergiert die durch Algorithmus **1** generirte Folge x_i für $i \longrightarrow \infty$ gegen eine Lösung x^*, welche ein lokales Minimum beschreibt. Sollte f nicht nach unten beschränkt sein, existiert dieses nicht zwingend. Daher kann eine unendliche Folge generiert werden mit $f(x_i) \longrightarrow -\infty$ für $i \longrightarrow \infty$.

3 Andere Verfahren

Um das Trust-Region-Steihaug Verfahren zu bewerten, wird im Folgenden das Gradienten- und globalisierte Newton-Verfahren mit der Armijo-Schrittweitenregelung erläutert und an drei Funktionen verglichen. Konkret wird dabei die Anzahl der benötigten Iterationen, Genauigkeit der Lösung und die benötigte Zeit beachtet. Dazu wird der Gradient an der errechneten Lösung ausgewertet und in der euklidischen Norm betrachtet. Um auch kleine Unterscheide bei Werten nahe 0 zu erkennen, wird zusätzlich der natürliche Logarithmus des Ergebnisses, der Norm angewandt.
Weiter wird der Einfluss unterschiedlicher Funktionen und Startpunkte x_0 betrachtet und welche Probleme in anderen Fällen auftreten könnte.

3.1 Das Gradienten-Verfahren

Das Gradienten-Verfahren ist ein Liniensuchverfahren. Dabei wird erst eine Abstiegsrichtung bestimmt und davon abhängig eine Schrittweite. Sei dazu f und x_s wie in Abschnitt **1.1** gegeben. Falls d_s eine Abstiegsrichtung ist, verringert sich die Funktion in diese Richtung, und es gilt

$$\exists t \in \mathbb{R}_{>0} : \forall \hat{t} \in (0, t] : f(x_s + \hat{t}d) < f(x_s)$$

Dies ist erfüllt durch

$$d_s^T \nabla f(x_s) < 0$$

Die Eigenschaft gilt insbesondere für den negativen Gradienten– die Richtung des steilsten Abstiegs. Dies und die folgenden Eigenschaften wurden bereits von Ulbrich und Ulbrich [3, S. 18] gezeigt. Somit gilt

$$d_s = -\nabla f(x_s)$$

Um eine sinnvolle Schrittweite σ_i zu wählen, nutzt der Algorithmus die Armijo-Bedingung:

$$f(x_s + \sigma_i d_s) \leq f(x_s) + \sigma_i v \nabla f(x_s)^T d_s \qquad i \in \mathbb{N}_0$$

Dabei gilt $v \in (0, 1)$ und $\sigma_0 = 1$. Ist diese Bedingung für σ_0 nicht erfüllt, wird eine neue Schrittweite iterativ errechnet. Es gilt

$$\sigma_{n+1} = \sigma_n \cdot \beta$$

Wobei $\beta \in (0, 1)$. Das Verfahren wird fortgeführt, bis eine Schrittweite ermittelt würde, welche die Armijo-Bedingung erfüllt. Durch [3, Lemma 7.5] ist sichergestellt, dass eine solche existiert. Damit hat das Verfahren eine neue Iterierte gefunden, und es gilt

$$x_{s+1} = x_s + \sigma_i d_s$$

Dabei beschreibt σ_i die Schrittweite, welche die Armijo-Bedingung erfüllt.
Für diese wird überprüft, ob die Abbruchbedingung erfüllt ist und es sich somit um einen stationären Punkt handelt. Dieses Verfahren wird fortgeführt, bis ein solcher Punkt gefunden wird. In [3, Satz 7.7] ist die Konvergenz des gesamten Gradienten-Verfahrens erläutert und sichergestellt.

3.2 Das globalisierte Newton-Verfahren

Ähnlich wie der Steihaug-Ansatz, nimmt auch das Newton-Verfahren die Taylor-Approximation zweiten Grades zur Hilfe. Sei eine Funktion f und ein Punkt x_s wie zuvor beschreiben

geben. Für die Approximation aus (1) gilt dann

$$t_s(p) = f(x_s) + g_s^T p + \frac{1}{2} p^T H_s p$$

mit $g_s = \nabla f(x_s)$ und $H_s = \nabla^2 f(x_s)$. Da es sich bei einem Minimum um einen stationären Punkt handeln muss, gilt für das Minimum p^* von t_s

$$0 = \nabla t_s(p^*) = g_s + H_s p^*$$

Durch Umformung ergibt sich

$$p^* = -H_s^{-1} g_s \tag{17}$$

Falls H_s vollen Rang besitzt, existiert die Inverse und t_s besitzt an der Stelle p^* einen stationären Punkt. Da es sich aber auch um ein Maximum oder einen Sattelpunkt handeln kann, muss überprüft werden, ob es sich um eine Abstiegsrichtung handelt. Ulbrich und Ulbrich [3, S. 49] schlagen dazu folgende Bedingung vor: Wähle $\lambda \in (0, 1)$, $\mu > 0$ und $c > 0$, dann wird die Richtung aus (17) akzeptiert, falls

$$-\nabla f(x_s)^T p^* \geq \min\{\lambda, \mu \|\nabla f(x_s)\|_2^c\} \|p^*\|_2 \|\nabla f(x_s)\|_2$$

Sollte diese Bedingung nicht erfüllt sein oder hat H_s nicht vollen Rang, konnte keine passende Suchrichtung gefunden werden. Daher wird als Richtung der negative Gradient gewählt. Es wird also das Gradienten-Verfahren aus Abschnitt **3.1** in diesem Schritt angewandt. Probleme können auftreten, wenn das Verfahren nicht gegen ein lokales Minimum, sondern einen Sattelpunkt konvergiert. Dies ist möglich, da an diesen Stellen der Gradient ebenfalls den Wert Null annimmt. Die Abstiegsbedingung kann n diesem Fall nicht ausreichend sein, da sich dem Sattelpunkt auch mit einer absteigenden Folge genährt werden kann.
Nachdem die Richtung bestimmt wurde, wird mit der Armijo-Regel wie in Abschnitt **3.1** die Schrittweite bestimmt und folglich die neue Iterierte.
Der Konvergenzbeweis ist ähnlich zu Abschnitt **3.1** und wird in [3][Satz 10.10] näher erläutert. Wichtig ist, dass lokal quadratische Konvergenz möglich ist.
Problematisch ist bei diesem Verfahren, dass wie beim Trust-Region-Verfahren auch, nicht nur der Gradient sondern auch die Hesse-Matrix vorhanden sein muss. Zusätzlich wird einen Lösung für das Gleichungssystem

$$H_s p^* = g_s$$

wie in Abschnitt **2.2**, benötigt.

4 Anwendung

Da nicht nur die theoretische Auseinandersetzung das Ziel dieser Arbeit ist, wird in diesem Abschnitt die praktische Anwendung betrachtet. Dazu werden drei verschiedene Funktionen behandelt.

1. Rosenbrock Funktion

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

1960 fand diese ihre erste Anwendung durch den englischen Mathematiker Howard Rosenbrock [5]. Er nutzte diese, um verschiedene Optimierungsalgorithmen zu untersuchen und zu vergleichen. Seitdem bildet die Funktion ein Benchmark für den Test von neuen Algorithmen.

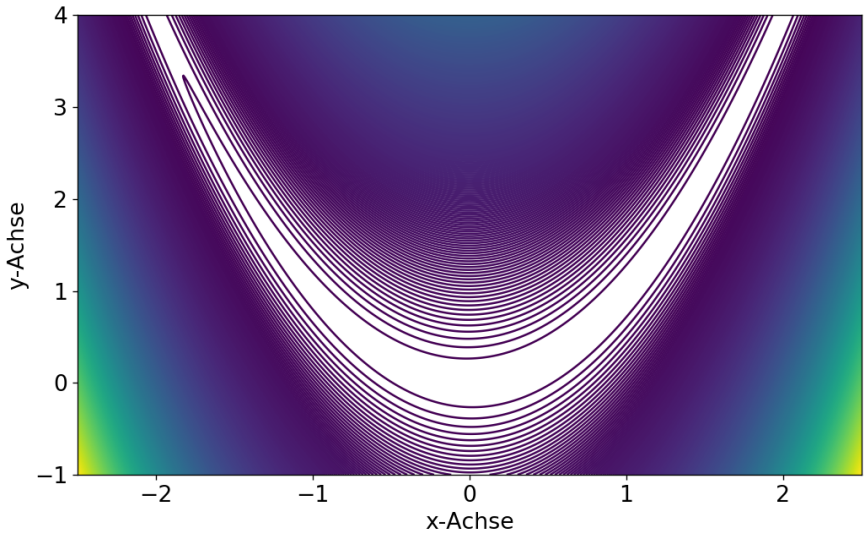

Abbildung 1: Rosenbrock Funktion

Während das parabelförmige Tal sehr flach ist, sind die Wände extrem steil. Dies und die kurvige Form machen es für viele Algorithmen schwer, zum Minimum bei $(1, 1)$ zu gelangen. Denn der Algorithmus muss es schaffen, mit der Hilfe von Geraden durch das kurvige Tal zu gelangen.

Als Startpunkte werden im folgenden $(-2, 2)$, $(0, 3)$ und $(-1, 0)$ gewählt.

Für den TRS-Algorithmus wurden folgende Parameter gewählt $\Delta_0 = 1$, $\Delta_{\text{Limit}} = 2$, $\varepsilon = 10^{-9}$ und $\eta = \frac{1}{5}$. Sowie für das Gradienten-Verfahren zusätzlich Gmax $= 5000$, $\alpha = 0.05$ und $\beta = 0.5$. Diese werden auch für die Armijo-Schrittweitenregelung des Newton-Verfahrens genutzt.

Um die Verfahren zu vergleichen, werden 4 Parameter genutzt. Erstens die Anzahl der Iterationen, zweitens die Norm des Gradienten an der jeweils errechneten Lösung, drittens der Abstand zum tatsächlichen Minimum und als Letztes die benötigte Zeit in 10^{-2} Sekunden. Ebenfalls wird die Kontur der Funktion, sowie der Verlauf der gefundenen Iteration gezeichnet. Für ein besseres Verständnis der Konvergenz, wird die Norm des

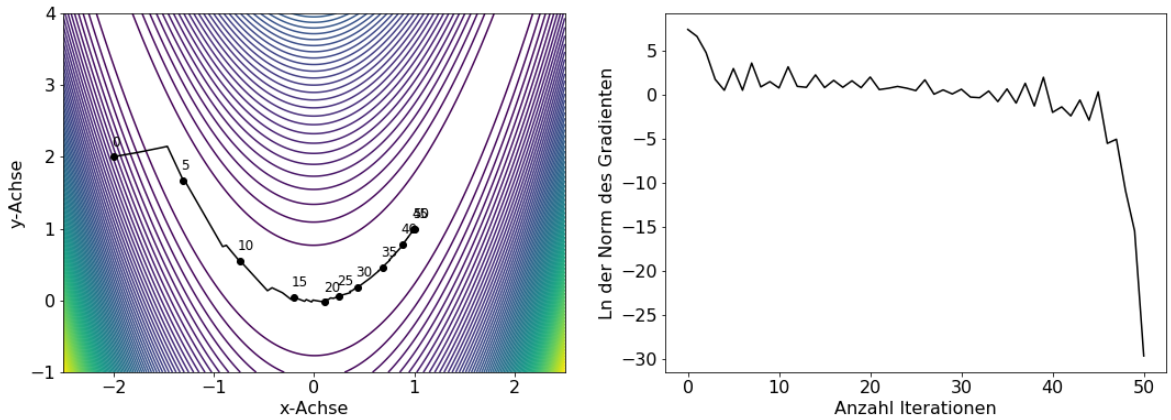

Abbildung 2: Trust-Region Verfahren mit Startpunkt $x_0 = (-2, 2)$

Gradienten, ausgewertet an jeder Iterierten, auch gezeichnet. Da diese Werte schnell gegen 0 konvergieren, wird der natürliche Logarithmus dieser Werte betrachtet.
Die Abbildung (2) zeigt den Verlauf der Iterationen, die durch den TRS-Algorithmus generiert wurden. Auffällig ist, dass bereits nach wenigen Schritte das Tal erreicht wurde. Dennoch hat der Algorithmus Schwierigkeiten, dort voranzukommen. Dies ist besonders gut am Verlauf des rechten Graphen zu erkennen. Dieser zeigt den Ln der Norm des Gradienten an den gefundenen Iterationen an. Dort ist der Abstieg zu Beginn auffällig, wie zuvor schon beschreiben. Auch die Probleme im Tal sind hier durch einen flachen Verlauf zu erkennen. An diesen Stellen, hat der Algorithmus das Problem, eine sinnvolle Richtung zu finden. Es wird versucht mit möglichst langen Geraden die Form des Tales anzunähern, ohne dabei den Funktionswert zu erhöhen. Dennoch kann der Algorithmus, auch wegen der Hesse-Matrix, die Form relativ gut erkennen, um mit guten Schrittweiten und Richtungen zum Minimum zu gelangen.
In den letzten Iterationen zeigt sich die Stärke des Verfahrens, da die Approximation aus (1) und die Funktion dort sehr ähnlich sind. Die gefundenen Iterationen stimmen stark mit den Minimum der Funktion überein. Dies führt zu einer schnellen Konvergenz. Dadurch schafft es der Algorithmus bereits nach 39 Schritten zu terminieren. An der gefundenen Lösung nimmt die Norm des Gradienten den Wert $1.3 \cdot 10^{-13}$ und der ln des Gradienten, den Wert -29.67. Der Abstand zum tatsächlichen Minimum beträgt nur noch $2.2 \cdot 10^{-16}$ Diese Werte sind für die meisten Anwendungen völlig ausreichend und können, falls benötigt, auch durch ein kleineres ε exakter werden. Wobei durch Rundungsfehler von Python oder MATLAB Grenzen gesetzt sind. Durch sinnvolle Anpassungen und Skalierung kann dies aber umgangen werden.

Das Newton-Verfahren, welches in Abbildung 3 rot dargestellt ist, findet das Minimum bereits nach 25 Schritten. Das rechte Schaubild zeigt ein ähnliches Verhalten des Newton- und Trust-Region-Verfahrens, dennoch schafft es das Newton-Verfahren früher eine schnelle Konvergenz zu erreichen, da Richtungen und Schrittweiten sich nicht

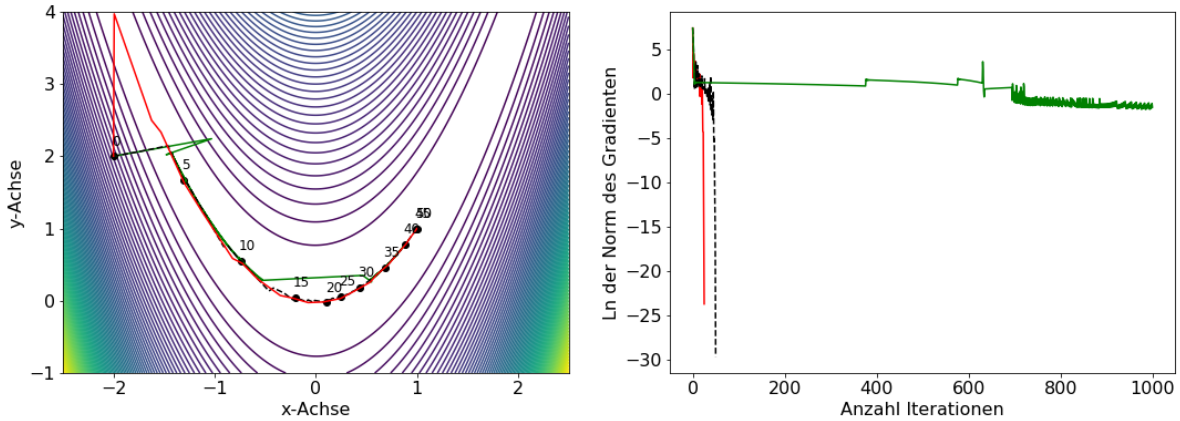

Abbildung 3: TRS-Verfahren in schwarz, Newton-Verfahren in rot und Gradienten-Verfahren in grün mit Startpunkt $x_0 = (-2, 2)$

so extrem von der schwierigen Form beeinflussen lassen. Die Abbildung zeigt zudem das Problem, welches bei der Gradienten-Methode, die in grün dargestellt ist, auftreten kann. Während die generierten Iterationen des Algorithmus zwar schnell ins Tal kommen, entstehen große Probleme bei der Konvergenz. Die Iterationen springen von einer Seite des Tales zur Anderen. Dabei nähren sich die Iterationen kaum dem Minimum an. Dies ist der Einfachheit des Algorithmus geschuldet. Zwar benötigt dieser nur die erste Ableitung der Funktion, durch die fehlende Hesse-Matrix, kann dieser aber keine Krümmung der Funktion erkennen.

		Iterationen	$\|\nabla f(\cdot)\|_2$	Abstand zu Minimum	10^{-2} Sekunden
	TR	50	$1.3 \cdot 10^{-13}$	$2.2 \cdot 10^{-16}$	0.1052
$x_0 = (-2, 2)$	NT	25	$4.9 \cdot 10^{-11}$	$4.5 \cdot 10^{-13}$	0.0543
	GR	5000	0.00426	0.00729	6.925
	TR	31	$6.4 \cdot 10^{-14}$	$1.1 \cdot 10^{-16}$	0.062
$x_0 = (0, 3)$	NT	15	$1.4 \cdot 10^{-13}$	$1.4 \cdot 10^{-14}$	0.030
	GR	5000	0.00356	0.00448	7.054
	TR	39	$8.0 \cdot 10^{-14}$	$6.4 \cdot 10^{-14}$	0.071
$x_0 = (-1, 0)$	NT	21	$1.8 \cdot 10^{-15}$	$6.2 \cdot 10^{-14}$	0.038
	GR	5000	0.00321	0.00454	7.124

Selbst nach 5000 Schritten schafft es der Algorithmus nur zum Punkt $(0.9980, 0.9959)$ und hat somit einen Abstand von 0.00729 zum tatsächlichen Minimum. Die Norm des Gradienten beträgt 0.0032, während der Newton Algorithmus einen Wert von 4.910^{-11} erreicht. Zudem ist die benötigte Zeit auffällig. Diese wurde als Durchschnittswert von mehreren Läufen errechnet und in 10^{-2} Sekunden angegeben. Dabei benötigt das Newton-Verfahren die kürzeste Zeit, während das TRS ungefähr doppelt so lange benötigt. Dieses Ergebnis ist auch für die anderen Startpunkte ähnlich. Da die Dimension des Definiti-

onsbereichs nur zwei beträgt, ist die Berechnung der Inversen der Hesse-Matrix, für das Newton-Verfahren, einfach. Bei der Betrachtung der benötigten Zeit pro Iteration fällt auf, dass sowohl Newton als auch TRS durchschnittlich 0.00210^{-2} Sekunden dazu benötigen. Somit gleicht sich das Bestimmen der Inversen des Newton-Verfahrens mit der erhöhten Anzahl von einfachen Berechnungen beim TRS aus. Da aber die Schritte des Newton-Verfahrens signifikant besser sind, ist der Gesamtaufwand des TRS höher. Diese Eigenschaft wird sich bei höheren Dimensionen ändern. Das Gradienten-Verfahren benötigt zwar nur 0.001410^{-2} Sekunden pro Schritt und ist damit fast doppelt so schnell wie die anderen Verfahren. Da aber die gefundenen Richtung nicht effizient sind, ist das Verfahren für diese Funktion nicht passend. Insgesamt sind die gefundenen Ergebnisse auch für die andern Startpunkte übertragbar.

2. $f(x, y) = -e^{-((x-\pi)^2+(y-\pi)^2)} - \sin(x)\sin(y)$

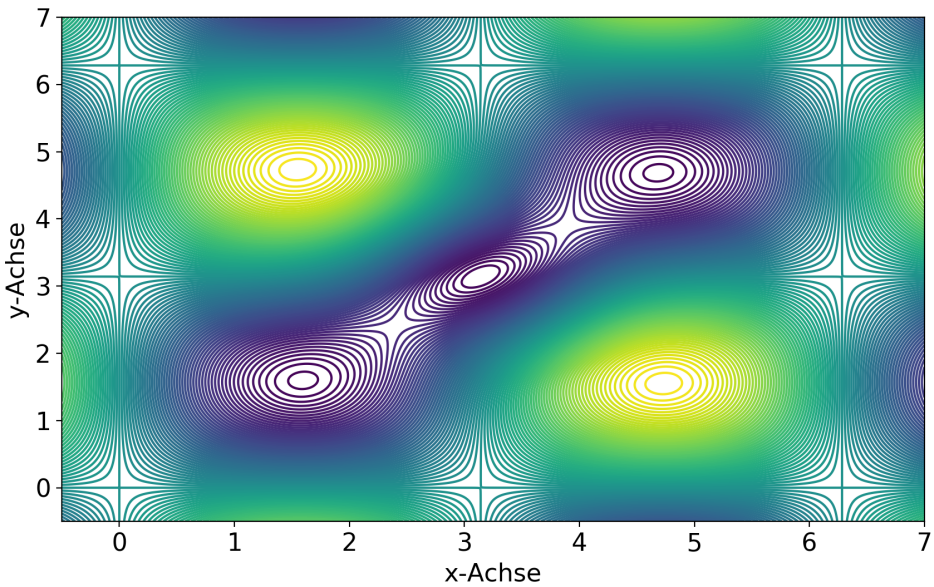

Abbildung 4: zweite Testfunktion

Die zweite Testfunktion besteht aus zwei Teilen. Der erste Teil ist eine Exponentialfunktion, wobei der Exponent keine positiven Werte annimmt. Dadurch entsteht ein Minimum bei (π, π), da für diese Werte der Exponent den Wert 0 annimmt. Somit ergibt sich dort ein Funktionswert von -1. Für Werte, welche einen größeren Abstand zum Minimum haben, flacht der Graph extrem ab und konvergiert gegen 0.
Der zweite Teil besteht aus zwei Sinusfunktionen. Diese bilden ein Schachbrettmuster mit wechselnden Hoch- und Tiefpunkten.
Dies ist in Abbildung 4 zu erkennen. Dabei sind die gelben Stellen Maxima, während die Violetten Minima anzeigen. Interessant ist der Bereich um (π, π), da die e-Funktion nur in diesem einen signifikanten Einfluss besitzt. Dadurch existieren lokale Minima bei

$(\frac{\pi}{2}, \frac{\pi}{2}), (\pi, \pi)$ und $(\frac{3\pi}{2}, \frac{3\pi}{2})$. Dabei handelt es sich bei den beiden äußeren sogar um globale Minima. Zwischen den Tiefpunkten liegen Sattelpunkt, genauso wie bei den restlichen Hoch- und Tiefpunkten.
Die folgende Tabelle zeigt die verschiedenen Bewertungsparameter für 3 verschiedene Startpunkte.

		Iterationen	$\|\nabla f(\cdot)\|_2$	Abstand zu Minimum	10^{-2} Sekunden
	TR	6	$1.0 \cdot 10^{-13}$	$1.2 \cdot 10^{-13}$	0.0174
$x_0 = (2.5, 4.75)$	NT	7	$8.2 \cdot 10^{-11}$	$9.6 \cdot 10^{-11}$	0.0167
	GR	15	$1.6 \cdot 10^{-10}$	$1.9 \cdot 10^{-10}$	0.0138
	TR	7	$3.7 \cdot 10^{-13}$	$3.1 \cdot 10^{-13}$	0.0160
$x_0 = (2.25, 5.25)$	NT	5	$4.0 \cdot 10^{-11}$	2.22198	0.0121
	GR	15	$3.7 \cdot 10^{-10}$	$4.3 \cdot 10^{-10}$	0.0135
	TR	7	$9.4 \cdot 10^{-10}$	$1.1 \cdot 10^{-9}$	0.0467
$x_0 = (3.5, 1.75)$	NT	30	$3.0 \cdot 10^{-12}$	$3.5 \cdot 10^{-12}$	0.0641
	GR	16	$8.2 \cdot 10^{-10}$	$9.6 \cdot 10^{-10}$	0.0152

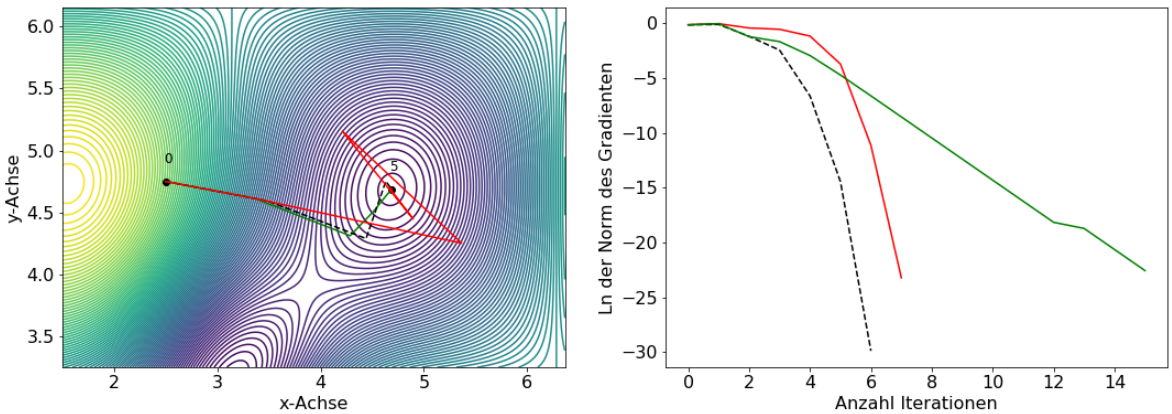

Abbildung 5: TRS-Verfahren in schwarz, Newton-Verfahren in rot und Gradienten-Verfahren in grün mit Startpunkt $x_0 = (2.5, 4.75)$

Abbildung 5 zeigt den Verlauf für den ersten Startpunkt. Dabei laufen alle Verfahren in einer ähnlichen Richtung zum Tiefpunkt. Auffällig ist nur das Verhalten des Newton-Verfahrens, da dieser mehrfach über das eigentliche Minimum springt. Dennoch endet der Algorithmus nach sieben Iterationen, das TRS-Verfahren benötigt nur sechs, während das Gradienten-Verfahren am langsamsten konvergiert. Das TRS benötigt zwar wenige Schritte, die hohe Anzahl an Rechnungen für einen Schritt machen den Algorithmus aber zeitaufwändig. Das Newton-Verfahren kann ebenfalls die Inverse einfach berechnen und benötigt, trotz einer höheren Zahl an Iterationen, weniger Zeit. Am schnellsten ist das Gradienten-Verfahren, obwohl es die doppelte Anzahl an Iterationen benötigt. Die Berechnung sind simpel genug, um die erhöhte Schrittzahl auszugleichen.

Wie in Abschnitt **3.2** angesprochen, kann das Verfahren gegen Sattelpunkte konvergieren,

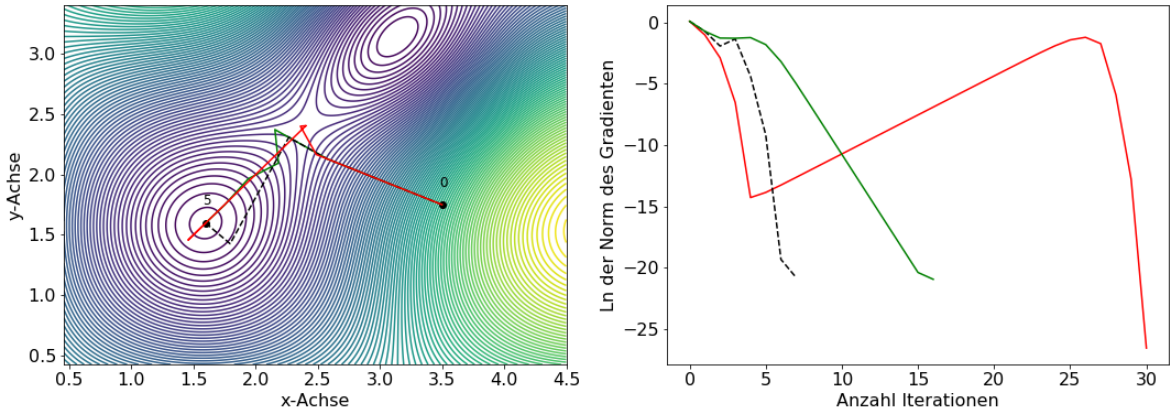

Abbildung 6: TRS-Verfahren in schwarz, Newton-Verfahren in rot und Gradienten-Verfahren in grün mit Startpunkt $x_0 = (3.5, 1.75)$

da diese ebenfalls stationäre Punkte sind und sich diesen mit der Hilfe von Abstiegsrichtungen genährt werden kann. Da das Verfahren die Abstiegsrichtung d_k genau so wählt, dass diese Punkte getroffen werden. Das ist auch der Fall für den zweiten Startpunkt. Der Newton-Algorithmus findet zwar bereits nach fünf Iterationen eine Lösung, an der der Gradient klein genug ist. Der Abstand zum nächsten Tiefpunkt beträgt aber 2.22198, daher handelt es sich um einen Sattelpunkt.
In Abbildung 6 ist ein ähnliches Verhalten des Newton-Algorithmus zu erkennen. Nach 4 Iterationen findet dieser einen Punkt mit einem kleinen Gradienten, welcher aber noch zu groß für die Abbruchbedingung ist. Trotzdem schafft es das Verfahren nochmal eine Abstiegsrichtung zu finden. Zwar erhöht sich dafür der Wert des Gradienten, aber nach 30 Schritten erreicht das Verfahren das tatsächliche Minimum. Wieder ist die extrem schnelle Konvergenz vom Newton- und TRS-Verfahren, durch den steilen Abfall der rechten Graphen, zu erkennen.
Das Gradienten-Verfahren erreicht wieder nur einen geringe Konvergenzgeschwindigkeit, benötigt dennoch weniger Zeit als die anderen Verfahren.
Besonders bei dieser Funktion zeigt sich die Stärke des TRS-Verfahrens. Es konvergiert verhältnismäßig schnell zum Minimum, ohne dabei von Sattelpunkt extrem angezogen zu werden. Und auch bei höheren Dimensionen wird das Verfahren effizient bleiben.

3. $f(x, y) = x^4 - 2x^3 + (y-1)^2 + (y+1)^2(x-2)^2$

Die letzte Testfunktion orientiert sich im weitesten Sinne an der Rosenbrock Funktion, aber ohne dabei zu extreme Formen anzunehmen. Wieder wurden 3 Startpunkte gewählt, und es ergab sich folgendes Ergebnis.

		Anzahl Iterationen	$\|\nabla f(\cdot)\|_2$	Abstand zu Minimum	10^{-2} Sekunden
	TR	8	$7.3 \cdot 10^{-8}$	$1.2 \cdot 10^{-8}$	0.1593
$x_0 = (3, 5)$	NT	8	$1.7 \cdot 10^{-16}$	$< 10^{-18}$	0.0177
	GR	100	$1.1 \cdot 10^{-7}$	$1.5 \cdot 10^{-8}$	0.1702
	TR	12	$8.1 \cdot 10^{-12}$	$2.9 \cdot 10^{-12}$	0.0284
$x_0 = (-2, 1)$	NT	11	$3.5 \cdot 10^{-15}$	$6.0 \cdot 10^{-16}$	0.0225
	GR	100	$1.2 \cdot 10^{-8}$	$9.2 \cdot 10^{-10}$	0.1836
	TR	7	$9.0 \cdot 10^{-14}$	$4.7 \cdot 10^{-14}$	0.0143
$x_0 = (0, 3.75)$	NT	8	$2.2 \cdot 10^{-12}$	$2.7 \cdot 10^{-13}$	0.0163
	GR	100	$5.9 \cdot 10^{-8}$	$2.1 \cdot 10^{-8}$	0.2369

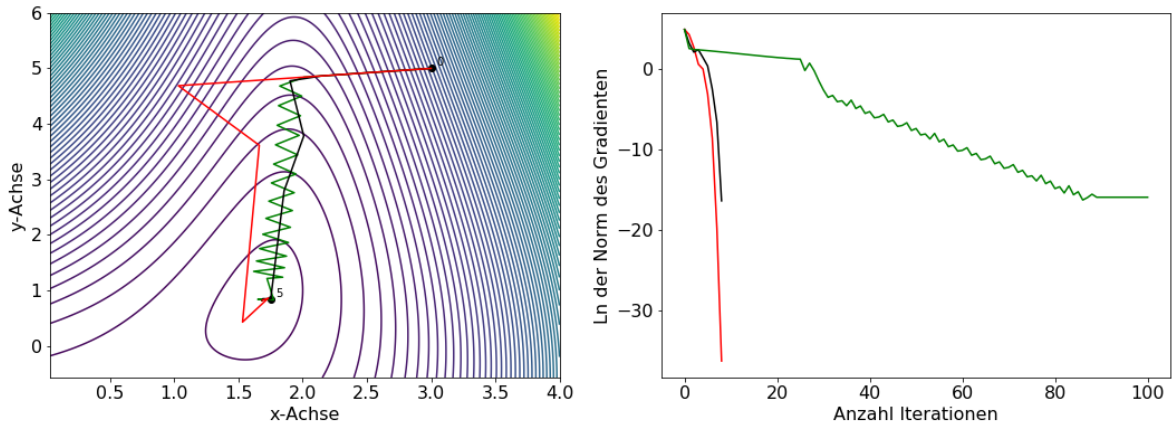

Abbildung 7: dritte Testfunktion mit Startpunkt $x_0 = (3, 5)$

Abbildung 7 ist vergleichbar zum Verlauf bei der Rosenbrock Funktion. Hier ist extrem gut der Unterschied zwischen Gradienten- und TRS-Verfahren zu erkennen. Beide finden schnell den weg in ein Tal, ab dort Unterscheiden sich aber die Richtungen enorm. Während der grüne Graph von einer Seite des Tales zur Anderen springt, schafft es das TRS-Verfahren, durch die Wahl von zwei konjugierten Richtungen, die Form und Krümmung des Tales wahrzunehmen. Auch ist die Vergrößerung der Trust-Region nach wenigen Schritten wesentlich. Dadurch steigt die Effizienz des Verfahrens enorm.
Als letzter Punkt ist relevant, dass das Gradienten-Verfahren es nie schafft die Abbruchbedingung zu erreichen, sondern wird durch die maximale Schrittzahl gestoppt. Das liegt nicht an der zu geringen Einschränkung, sondern an der Funktion und Python selber. Denn die Abstände zum tatsächlichen Minimum sind vergleichbar zu den anderen Funktion, da aber trotzdem noch der Gradient verhältnismäßig groß ist, kommt Python bei der Berechnung der Schrittweite an seine Grenzen. um dieses Problem, müsst das Verfahren angepasst werden. Dies überschreitet aber das Ausmaß dieser Arbeit.
Das gleiche geschieht beim ersten Startpunkt für das TRS-Verfahren. Die Funktionswerte

und Abstände werden so klein, dass Rechnungen instabil werden und durch Rundungsfehler keine neuen sinnvollen Iterationen mehr ausgerechnet werden können.

5 Fazit

Bei der numerischen Minimierung von Funktionen stehen Aufwand und Genauigkeit im Vordergrund. Dabei muss für jedes Problem abgewägt werden, welch Ressourcen und Voraussetzungen gegeben sind und welche Ergebnisse benötigt werden. Dabei hat jedes in dieser Arbeit vorgestellt Verfahren eine Existenzberechtigung. Das Gradienten-Verfahren benötigt nur die erste Ableitung und kommt mit wenig Aufwand zur nächsten Iteration. Dabei können aber Suchrichtung und Schrittweite durch Rundungsfehler und komplexe Funktionen schnell an ihre Grenze kommen.
Das globalisierte Newton-Verfahren besitzt lokal eine sehr schnelle Konvergenz und ausgewählte Suchrichtungen führen schnell zum gewünschten Ziel. Doch es wird immer auch die zweite Ableitung benötigt, sowie die Berechnung einer Lösung für das Gleichungssystem an jeder Iterierten. Dafür werden spezielle Verfahren benötigt, welche in der Numerik behandelt werden.
Das Trust-Region-Steihaug-Verfahren schließt die Lücke zwischen diesen Verfahren. Es wird zwar auch die Hesse-Matrix benötigt, dabei werden aber nur Matrixprodukte berechnet. Für die steigt zwar ebenfalls der Aufwand an, sowie die Anzahl der Schritte des Steihaug-Algorithmus. Dennoch ist dieser signifikant geringer als beim Newton-Verfahren. Gleichzeitig ist die Genauigkeit beider Verfahren trotzdem noch vergleichbar.
Zudem basiert die Idee auf einem Ansatz aus den 50er Jahren, wurde später in 1982/83 erstmals populär und genau dokumentiert. Seitdem wird das Verfahren für verschiedene Ansprüche und Voraussetzungen angepasst und weiterentwickelt. Der Ansatz und die Ideen werden auch noch in Zukunft ihre Relevanz besitzen und in neuen Algorithmen ihren Einsatz finden.

6 Anhang

Im Anhang befindet sich der Code zum Trust-Region- und Steihaug-Algorithmus. Dieser wurde in Python verfasst. Zudem wurden Kommentare am Ende der Zeilen eingefügt. Diese beziehen sich auch darauf, wie ein effizienter Algorithmus entsteht.

```
import numpy as np
from math import sqrt
from numpy.linalg import norm

def CG_Steihaug(f, df, ddf, x, delta):
    '''
    Steihaug zur Berechnung von Schritten fuer den
    Trust-Region-Algorithmus Input: f, df, ddf sind die Funktion
    und ersten beiden Ableitungenx ist der Startpunkt der Suche
    '''
    n = np.size(x)          #Dimension des Def. Ber.
    z = np.zeros(n)
    r = df(x)
    d = -r
    f = f(x)
    df1 = r
    ddf1 = ddf(x)
    A = norm(df1)
    B = min(0.5, sqrt(A))
    eps = A*B        #Berchnung der Abbruchbedingung
    ddf_df = np.dot(ddf1, d)
    C = np.dot(d, ddf_df) #Produkt Hesse-Matrix und Suchrichtung
    if C <= 0:            #Konvexitaet in Richtung des Gradienten
        p = d*delta/norm(d)
        return p                          #Ausgabe der Loesung
    for j in np.arange(0, n, 1):     #CG-Verfahren max. n Wiederh.
        C = np.dot(d,ddf_df)
        if C <= 0:                #Konvexitaet in Suchrichtung
            tau = (-2*np.dot(z,d) + sqrt((2*np.dot(z,d))**2 -
                4*np.dot(d,d)*(np.dot(z,z) - delta**2)))
                /(2*np.dot(d,d)) #Berechnung maximalen Schrittweite
            p = z + tau * d
            return p                      #Ausgabe der Loesung
        rr =  np.dot(r, r)       #Norm des Residual ins Quadrat
        alpha = rr/C
```

```
        z1 = z + alpha*d            #Berechnug neue Iterierte
        if norm(z1) >= delta:    #ist gefundener Schritt in TR
            tau = (-2*np.dot(z,d) + sqrt((2*np.dot(z,d))**2 -
                  4*np.dot(d,d)*(np.dot(z,z) - delta**2)))
                  /(2*np.dot(d,d))  #Berechnung maximale Schrittweite
            p = z+tau*d
            return p                        #Ausgabe der Loesung
        r_1 = r + alpha*ddf_df       #Berechnung neues Residual
        if norm(r_1) < eps:    #Abbruch, falls Residual klein
            p = z1
            return p                        #Ausgabe der Loesung
        beta = np.dot(r_1, r_1)/r #Aktualisierung der Variabeln
        d = -r_1 + beta*d
        ddf_df = np.dot(ddf1, d)
        r = r_1
        z = z1
    return z   #Ausgabe Loesung, falls maximale Schrittzahl


def Trust_Region(x, deltaL = '2', delta = '1', minimum_fit = '1/5',
                 f, df, ddf, tmax = '50', epsilon = '10**-8'):
    '''
    Trust-Region-Algorithmus zur Berechnung von Tiefpunkten
    Input: f, df, ddf sind die Funktion und ersten beiden
    Ableitungen, x = der Startpunkt der Suche,
    deltaL = maximaler Radius der TR, delta = Anfangsradius,
    minimum_fit = Mindestwert zur Uebernahme eines Schrittes,
    tmax = maximale Anzahl vonbSchritten,
    epsilon = Abbruchbedingung
    '''
    for k in np.arange(0, tmax, 1):
        df_1 = df(x)
        if norm(df_1) < epsilon: #Ueberpruefung stationaerer Punkt
            return x                        #Ausgabe der Loesung
        p = CG_Steihaug(f,df,ddf,x,delta) #neue Iteration
        ddf_1 = ddf(x)
        q = (f(x)-f(x+p))/(-np.dot(df_1, p) - 1/2*
             np.dot(np.dot(p, ddf_1), p)) #Berechung der Effizienz
        if q < 1/4:                 #Aktualisierung Radius der TR
            delta = 1/4*delta
        else:
```

```
        if q > 3/4 and norm(p) == delta:
            delta = min(2*delta, deltaL)
    if q > minimum_fit: #Ueberpruefung ob Iterierte akzeptiert
        x = x + p
return x   #Ausgabe der Loesung, falls maximale Schrittzahl
```

Literatur

[1] **Steihaug, T**: The Conjugate Gradient Method and Trust Regions in Large Scale Optimization, SIAM Journal on Numerical Analysis Vol. 20 No. 3 1983

[2] **Goldfield, S; Quandt, R; Trotter, H**: Maximization by Quadratic Hill-Climbing, Econometrica 1966

[3] **Ulbrich, M; Ulbrich, S**: Nichtlineare Optimierung, Birkhäuser 2011

[4] **Sorensen, D**: Newton's Method with a Model Trust Region Modification, SIAM Journal on Numerical Analysis Vol. 19 No. 2 1982

[5] **Rosenbrock, H**: An Automatic Method for finding the Greatest or Least Value of a Function , The Computer Journal Vol. 3 No. 3 1960

[6] **Nocedal, J; Wright, S**: Numerical Optimization, Springer Second Edition 2006.

[7] **Hestens, M; Stiefel, E** Methods of Conjugate Gradients for Solving Linear Systems, Journal of Research of the National Bureau of Standards Vol. 49 No. 6 1952

[8] **Shewchuk, J**: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994