Probabilistische Liniensuche für stochastische Optimierung

Denise Schmitz

Geboren am 17. März 1995 in Düsseldorf

1. Februar 2017

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jochen Garcke

Zweitgutachter: Dr. Markus Siebenmorgen

Institut für numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

In vielen Situationen haben wir eine Funktion, $f : \mathbb{R}^n \to \mathbb{R}$ mit Dimension n, gegeben, und möchten diese optimieren, indem wir das globale Minimum oder Maximum herausfinden. Diese Funktion könnte zum Beispiel eine Kostenfunktion sein, bei der das globale Minimum aussagt, wann die Kosten optimiert werden, oder eine Gewinnfunktion, bei der das globale Maximum darstellt, wie wir die Gewinne maximieren können.
Wenn wir uns daran erinnern, wie dieses Problem in der Schule oder im Studium gelöst wurde, dann würde die Funktion analytisch differenziert werden und die erhaltene Ableitung mit Null gleichgesetzt. Also müsste die Bedingung $\nabla f(x) = 0$ erfüllt sein. Daraus bekämen wir die Extremwerte und müssten dann herausfinden, ob ein Minimum oder ein Maximum vorliegt, indem wir die zweite Ableitung gebrauchen oder uns Funktionswerte in der Nähe anschauen.
Soll dieses Verfahren nicht per Hand, sondern durch einen Rechner durchgeführt werden, kann es dazu kommen, dass die Ableitung nicht analytisch berechnet werden kann oder dass die Funktionswerte gestört sind und deshalb falsche Ergebnisse geliefert werden. Dies kann zum Beispiel bei Rechenungenauigkeiten oder Rundungsfehlern auftreten. Um dieses Problem zu vermeiden, werden Verfahren zur Approximation an das Minimum bzw. Maximum benutzt. Das bedeutet, dass nicht der genaue, optimale Wert x berechnet wird, sondern es wird sich diesem Punkt so gut wie möglich angenähert.

Die meistgenutzten Verfahren, die für dieses Problem verwendet werden, sind die Abstiegsverfahren mit Schrittweitensuche. Sie sind das Standardwerkzeug in der deterministischen, also reproduzierbaren Optimierung, da sie Stabilität und Effizienz sicherstellen. Ein solches Abstiegsverfahren zeichnet sich dadurch aus, dass es sich iterativ dem Minimum bzw. Maximum annähert. Wenn dabei jedoch die Funktion oder ihr Gradient gestört ist, muss ein geeignetes Verfahren gefunden werden, das mit der Störung zurecht kommt, da dies nicht für alle Abstiegsverfahren gilt.
Ein weiteres Problem bei hohen Dimensionen ist, dass zwar ein Minimum bzw. ein Maximum gefunden wird, dieses jedoch nur ein lokales und kein globales Extremum ist. So würden zwar gute Werte, aber keine optimalen ausgegeben werden. Um sich diesem Problem anzunähern, müssen drei Hinweise befolgt werden, die das Problem unwahrscheinlicher machen, es jedoch nicht komplett ausschließen: Erstens sollte der Startwert so gut wie möglich gewählt werden, zweitens sollten weitere Informationen zur Hilfe gezogen werden, und drittens kann ein stochastisches Verfahren helfen. Ein stochastisches Verfahren kann versuchen, die gestörten Werte herauszufiltern und die Störungen somit nicht bzw. wenig in die Rechnung mit einzubeziehen.

Das Verfahren, das in dieser Arbeit betrachtet wird, ist das *Stochastische Gradientenverfahren*, welches ein Abstiegsverfahren mit Schrittweitensuche ist, das mit Störun-

gen umgehen und helfen kann, das Problem mit den lokalen Extrema zu verkleinern. Das stochastische Gradientenverfahren ist im Moment das Standardverfahren für die Optimierung von mehrdimensionalen Funktionen mit gestörten Gradienten.
Diese Bachelorarbeit stützt sich dabei auf die Arbeit *Probabilistic Line Searches for Stochastic Optimization* von Maren Mahsereci und Philipp Hennig, welche ein solches stochastisches Gradientenverfahren mithilfe von wahrscheinlichkeitstheoretischen Grundlagen gebaut, getestet und der Öffentlichkeit zugänglich gemacht haben [1].
Neben dem üblichen Fundament eines Abstiegsverfahrens benutzt die Methode außerdem einen Gauß-Prozess-Ersatz für eindimensionale Probleme und einen Bayes-Optimierer, der die erwartete Verbesserung der Entwicklungspunkte (auch Auswertungspunkte genannt) berechnet. Zusätzlich wird eine probabilistische, also eine Wahrscheinlichkeiten berücksichtigende Art der Wolfe-Terminierungsbedingungen verwendet, um den Abstieg zu kontrollieren und die Wahrscheinlichkeit zu berechnen, dass die Wolfe-Bedingungen erfüllt sind.
Das Ziel unseres stochastischen Gradientenverfahren ist es, eine Methode zu entwerfen, welche mit gestörten Werten und ohne die Eingabe von Parametern das Ergebnis eines gegebenen stochastischen Optimierungsproblems findet. Dabei soll es aber minimalen Rechenaufwand haben und die Notwendigkeit entfernen, eine Schrittweite für das stochastische Gradientenverfahren zu definieren.

In Kapitel 2 werden zuerst einmal die Grundlagen des Gradientenverfahrens dargelegt und daraufhin in Kapitel 3 die Grundlagen der Wahrscheinlichkeitstheorie erklärt, aus denen dann die einzelnen Elemente des Verfahrens zusammengebaut werden.
Wir werden außerdem in Kapitel 5 den stochastische Fall des Gradientenverfahrens für eine Datenmenge, im Sinne des maschinellen Lernens, betrachten. Dies kann benötigt werden, wenn für eine Menge von Auswertungspunkten eine Funktion gefunden werden soll, welche die Menge bestmöglich approximiert, also den Abstand von der Funktion zu der Menge minimiert. Damit können dann die einzelnen Punkte der Datenmenge besser ausgewertet werden. Dies geschieht, indem eine Verlustfunktion $\mathcal{L}(x)$ minimiert wird. Diese Verlustfunktion ist definiert als der Durchschnitt von einzelnen Verlustfunktionen $\ell(\hat{x}, x)$, die den Schaden angeben, wenn statt des exakten Werts x der gestörte Wert $\hat{x}$ angenommen wird [2]. In diesem Fall treten zum Beispiel dann gestörte Gradienten auf, wenn die Verlustfunktion $\mathcal{L}(x)$ mit dem Optimierungsparameter $x \in \mathbb{R}^n$ nur über eine Teilmenge $\{d_j\}_{j=1\dots m}$ der Datenpunkte statt über die Gesamtmenge $\{d_i\}_{i=1\dots M}$ entwickelt wird, wobei m viel kleiner als M sein soll. Dies wird gemacht, da eine Auswertung über die gesamte Datenmenge zu rechenaufwendig wäre und die Approximation der Teilmenge sehr nah an das exakte Ergebnis herankommt.
In Kapitel 6 werden die Parameter, die vom Benutzer im stochastischen Gradientenverfahren gewählt werden müssten, nacheinander eliminiert, damit der Benutzer das

bestmögliche Ergebnis geliefert bekommt, ohne es negativ beeinflussen zu können.
Am Ende werden einige numerische Experimente mit dem Matlab-Code, der zu [1] gehört, und dem selbst geschriebenen Python-Code ausgeführt. Dabei wird zum Beispiel geprüft, ob die Wahl der Parameter sinnvoll war oder wie die Funktionswerte sich während des Vorgangs verhalten.
Zum Schluss folgt eine Zusammenfassung der Ergebnisse und ein Rückblick auf das Gelernte. Außerdem wollen wir uns anschauen, wie das Thema fortgeführt werden kann.

Eigene Leistung

Während des Schreibens dieser Arbeit orientierte ich mich an den folgenden Zielen:

1. Wiederholung und Vertiefung der Themen *Abstiegsverfahren* und *Stochastik*.
2. Einarbeitung in die Themen *Gauß-Prozess* und *Maschinelles Lernen*.
3. Aufarbeitung der Vorgehensweise in [1] und die Umstellung der bereits bekannten Elemente in die probabilistische Art.
4. Implementierung des stochastischen Gradientenverfahrens in Python auf der Grundlage des vorgegebenen Matlab-Codes aus [1].
5. Analyse der Parameter und deren Einflüsse, ausgewertet durch numerische Experimente.

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich tatkräftig beim Schreiben dieser Bachelorarbeit unterstützt haben. Als Erstes möchte ich mich speziell bei Herrn Prof. Dr. Jochen Garcke für die Betreuung und spannende Themenauswahl bedanken. Sie hatten immer ein offenes Ohr für meine Fragen und haben mir einige wichtige Richtungshinweise gegeben. Außerdem geht mein Dank an Herrn Dr. Markus Siebenmorgen, der die Zweitkorrektur übernommen hat.
Im weiteren bedanke ich mich von Herzen bei meiner Familie und meinen Freunden für die Unterstützung in meinem gesamten Studium. Ihnen habe ich vielleicht manchmal ein bisschen viel über Mathe geredet und ständig Fragen gestellt, die ich mir danach selbst beantwortet habe. Hannes David Becker und meiner Familie danke ich insbesondere, da sie meine Launen ertragen mussten und mich, nicht nur während der Bachelorarbeit, sondern während des gesamten Studiums emotional gestützt haben. Ohne euch hätte mich vielleicht öfter mal der Mut verlassen.
Meinen Eltern, Irina Schmitz, Michael Stieleke und Julian Schröteler, der mir auch mathematisch unter die Arme gegriffen hat, danke ich obendrauf noch, da sie meine Bachelorarbeit Korrektur gelesen haben und mir so einige peinliche Fehler ersparten.

Notation

Wir werden als Erstes die Notation dieser Bachelorarbeit aufstellen. Die folgenden mathematischen Objekte werden als bekannt vorausgesetzt oder werden innerhalb der Bachelorarbeit definiert und befinden sich zum Nachschlagen in dieser Liste:

$\mathbb{R}^n$	n-dimensionaler Raum $\mathbb{R} \times \ldots \times \mathbb{R}$ mit $n \in \mathbb{N}$
$f : X \to Y$	Abbildung f aus dem Raum X in den Raum Y
$\nabla^i f$ oder kurz $f^{(i)}$	i-te Ableitung einer Funktion f
$\{x_k\}$	Folge, bestehend aus den Folgegliedern $x_k,\ k \in \{1, 2, \ldots\}$
$\{x_k\} \to x$	Folge x_k konvergiert gegen den Wert x
$x \ll y$	x viel kleiner als y
$\lvert x \rvert$	Betrag von x
$\lVert x \rVert$	Beliebige Norm von x
X^T	Transponierter Vektor bzw. transponierte Matrix von X
X^{-1}	Inverse Matrix von X
A^c	Komplement von A in der Übermenge Ω, definiert als $\Omega \setminus A$
$\mathcal{B}(X)$	Borelsche σ-Algebra, kleinste σ-Algebra, die alle offenen Mengen enthält
$\mathbb{E}(X)$	Erwartungswert von X
$Var(X)$	Varianz von X
$Cov(X, Y)$	Kovarianz von X und Y
$X \sim \mathcal{N}(\mu, \sigma^2)$	X ist normalverteilt mit Mittelwert μ und Varianz σ^2
$X \sim \mathcal{N}_n(\mu, \Sigma)$	X ist multivariat normalverteilt in Dimension n mit dem Mittelwertvektor μ und der Kovarianzmatrix Σ
$(X_t) \sim \mathcal{GP}(m_t, k_{tt'})$	X_t ist ein Gauß-Prozess mit der Mittelwertfunktion $m_t = m(t)$ und der Kovarianzfunktion $k_{tt'} = k(t, t')$
$X_n \xrightarrow{\mathrm{d}} X$	Die Folge X_n konvergiert in Verteilung gegen X
$\mathbb{1}_B$	Indikatorfunktion von x, $\mathbb{1}_B(x) = \begin{cases} 1, \text{ wenn } x \in B \\ 0, \text{ wenn } x \notin B \end{cases}$
$\mathbf{I}$	Einheitsmatrix
$\mathbf{0}$	Nullvektor

2 Liniensuche

Um das gewünschte Verfahren zusammenzubauen, benötigen wir mehrere Unterroutinen und Prozesse. Diese werden nun erläutert, in den folgenden Kapiteln in die probabilistische Art verwandelt und im Kapitel 5 daraufhin für das maschinelle Lernen umgebaut. Unser Optimierungsproblem besteht in diesem Kapitel immer aus einer Funktion $f : \mathbb{R}^n \to \mathbb{R}$, deren Minimum x_* mithilfe der Ableitung ∇f und einem initiierten Startwert x_0 gefunden werden soll.
Unser Ziel dabei ist es, das Gradientenverfahren zu verstehen, um daraus das stochastische Gradientenverfahren zu bauen. Beim Gradientenverfahren benutzen wir die Ableitung der Funktion, um eine Richtung zu finden, in die der momentane Entwicklungspunkt verändert werden soll. Das bedeutet, dass wir unsere Abstiegsrichtung auf die Ableitung setzen und somit eine gut gewählte Richtung haben, da die Funktion an dem aktuellen Punkt in genau diese Richtung am steilsten sinkt. Deswegen wird das Gradientenverfahren auch Verfahren des steilsten Abstiegs genannt.
Zuerst werden wir uns das allgemeine Konzept der Abstiegsverfahren anschauen, worauf das speziellere Gradientenverfahren folgt. Danach müssen wir klären, wie wir einen Entwicklungspunkt auswählen und was unsere Kriterien für einen zugelassenen Punkt sind. Diese Informationen sind Standardinhalte von Numerik-Vorlesungen.

2.1 Abstiegsverfahren

In den allgemeinen Abstiegsverfahren mit Schrittweitensuche wird bei einem Punkt x_0 gestartet und sich Schritt für Schritt einem Minimum x_* angenähert, indem eine Suchrichtung s_k und eine Schrittweite α_k gewählt werden und sich dann der Punkt um diese Werte verändert, indem $x_{k+1} = x_k + a_k s_k$ gesetzt wird. Der Punkt x_k wird solange um die Suchrichtungen und Schrittweiten verändert, bis ein gewisses Abbruchkriterium erfüllt ist. Meistens wird abgebrochen, falls der momentane Punkt nahe genug an unserem Minimum liegt, $\|\nabla f(x_k)\| < \varepsilon$, oder der neue Wert nur sehr schwach vom alten Wert abweicht, $\|x_{k+1} - x_k\| < \varepsilon$ oder $\|f(x_{k+1}) - f(x_k)\| < \varepsilon$, wobei bei allen drei Bedingungen ε klein gewählt wird. Eine andere Art der Abbruchkriterien ist es, nur eine bestimmte Anzahl von Iterationen durchzuführen.

Um zu definieren, was genau eine Abstiegsrichtung und eine Schrittweite ist und wann sie zulässig sind, benutzen wir die nachfolgenden Definitionen, Sätze, Lemmata und Beweise, welche in veränderter Schreibweise aus [3] stammen.

Definition 2.1: (Abstiegsrichtung)

Es seien $f : \mathbb{R}^n \to \mathbb{R}$ und $x \in \mathbb{R}^n$. Ein Vektor $s \in \mathbb{R}^n$ heißt Abstiegsrichtung von f in x, wenn ein $\alpha_0 > 0$ existiert, mit $f(x + \alpha s) < f(x)$ für $\alpha \in (0, \alpha_0]$.

Unsere Suchrichtung ist durch den Typ des Abstiegsverfahren, welches benutzt wird, bestimmt und die Schrittweite wird in den meisten Fällen berechnet, indem eine eindimensionale Hilfsfunktion $F(\alpha) = f(x_k + \alpha s_k)$ minimiert wird. Dies ist jedoch sehr kostenintensiv, weswegen es vermieden werden sollte. Wir sprechen hierbei immer von Kosten im Sinne von Rechenaufwand, der im Computer entstehen, wenn Berechnungen ausgeführt werden. Diese Rechenkosten sollen so klein wie möglich gehalten werden.
Bei beiden Werten, der Richtung und der Weite, muss sichergestellt werden, dass sich die Approximation tatsächlich zum vorherigen Wert verbessert, weswegen $\nabla f(x_k)^\mathsf{T} s_k < 0$ und $f(x_k + \alpha_k s_k) < f(x_k)$ gefordert werden. Dass dieses erste Kriterium auch wirklich eine Abstiegsrichtung fordert, besagt das folgende Lemma.

Lemma 2.2:

Es seien $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar und $s \in \mathbb{R}^n$. Gilt $\nabla f(x)^\mathsf{T} s < 0$, dann ist s eine Abstiegsrichtung von f in x.

Fassen wir nun alles bisherige zu einem Algorithmus für das allgemeine Abstiegsverfahren zusammen, so sieht dieser wie folgt aus:

Algorithmus 2.1 Allgemeines Abstiegsverfahren

Eingabe: Startwert $x_0 \in \mathbb{R}^n$, Funktion $f : \mathbb{R}^n \to \mathbb{R}$
Ausgabe: Approximation des Minimums x_*

1: $k = 0$
2: **while** Abbruchbedingung nicht erfüllt **do**
3: Bestimme Suchrichtung $s_k \in \mathbb{R}^n$, sodass $\nabla f(x_k)^\mathsf{T} s_k < 0$
4: Bestimme Schrittweite $\alpha_k \in \mathbb{R}$, sodass $f(x_k + \alpha_k s_k) < f(x_k)$
5: Setze $x_{k+1} = x_k + \alpha_k s_k$
6: $k = k + 1$

Schritt 4 wird dabei Liniensuche genannt. Der Schritt wird exakte Liniensuche genannt, wenn die Schrittweite exakt berechnet wird, dies ist jedoch sehr kostenintensiv. Praktikabler ist eine inexakte, kostengünstige Liniensuche, die eine passende Verkleinerung der Werte liefert.
Das Verfahren gibt einen stationären Punkt zurück, also ein Extremum. Dies wird bewiesen, nachdem definiert wurde, wann eine Suchrichtung, bzw. eine Schrittweite, zulässig ist:

Definition 2.3: (Zulässige Suchrichtung)

Die Teilfolge $\{s_l\}$ der durch den Algorithmus 2.1 erzeugten Suchrichtungen $\{s_k\}$ heißt zulässig, falls gilt:

(i) $\nabla f(x_k)^\mathsf{T} s_k < 0$ für alle $k \geq 0$
(ii) $\left\{ \frac{\nabla f(x_l)^\mathsf{T} s_l}{\|s_l\|} \right\} \to 0 \Rightarrow \{\nabla f(x_l)\} \to 0$

Definition 2.4: (Zulässige Schrittweite)

Die Teilfolge $\{\alpha_l\}$ der durch den Algorithmus 2.1 erzeugten Schrittweiten $\{\alpha_k\}$ heißt zulässig, falls gilt:

(i) $f(x_k + \alpha_k s_k) \leq f(x_k)$ für alle $k \geq 0$
(ii) $f(x_k + \alpha_k s_k) - f(x_k) \to 0 \Rightarrow \left\{ \frac{\nabla f(x_l)^\mathsf{T} s_l}{\|s_l\|} \right\} \to 0$

Satz 2.5: (Globaler Konvergenzbeweis)

Sei $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar. Algorithmus 2.1 terminiere nicht endlich und erzeuge Folgen $\{x_k\}$, $\{s_k\}$ und $\{\alpha_k\}$. Sei x_* ein Häufungspunkt von $\{x_k\}$ und $\{x_l\}$ eine gegen x_* konvergente Teilfolge, sodass die Suchrichtungsfolge $\{s_l\}$ und die Schrittweitenfolge $\{\alpha_l\}$ zulässig sind. Dann ist x_* ein stationärer Punkt von f.

Beweis:

Sei x_* ein Häufungspunkt von $\{x_k\}$ und $\{x_l\}$ eine Teilfolge, die gegen x_* konvergiert, sodass $\{s_l\}$ und $\{\alpha_l\}$ zulässig sind. Mit Definition 2.4 (i) gilt Monotonie der Folge $\{f(x_k)\}$, das heißt $f(x_{k+1}) = f(x_k + \alpha_k s_k) \leq f(x_k)$. Daher besitzt $\{f(x_k)\}$ einen Grenzwert $\varphi \in \mathbb{R} \cup \{-\infty\}$, insbesondere folgt auch $\{f(x_l)\} \to \varphi$. Da f stetig und $\{x_l\} \to x_*$ gilt aber auch $\{f(x_l)\} \to f(x_*)$. Es folgt somit $\varphi = f(x_*)$ und $f(x_k) \to f(x_*)$. Damit gilt

$$f(x_*) - f(x_0) = \lim_{k\to\infty} f(x_k) - f(x_0) = \sum_{k=0}^{\infty} \underbrace{(f(x_{k+1}) - f(x_k))}_{\to 0,\text{ sonst konv. die Reihe nicht}} ,$$

also $f(x_k + \alpha_k s_k) - f(x_k) \to 0$. Da $\{\alpha_l\}$ zulässig, folgt mit Definition 2.4 (ii) $\left\{ \frac{\nabla f(x_l)^\mathsf{T} s_l}{\|s_l\|} \right\} \to 0$. Da $\{s_l\}$ zulässig ist, folgt mit Definition 2.3 (ii) $\{\nabla f(x_l)\} \to 0$. Da ∇f stetig ist, ergibt sich $\nabla f(x_*) = \lim_{l\to\infty} \nabla f(x_l) = 0$. □

Wir haben somit bewiesen, dass unser Abstiegsverfahren einen stationären Punkt liefert, wenn es unendlich lange läuft. Das bedeutet, wir erhalten eine gute Approximation an das Minimum, wenn es durch ein geeignetes Abbruchkriterium beendet wird.

Da die Liniensuche nur die Länge des Schrittes und nicht die Richtung beschreibt, kann dieser Schritt mit anderen Algorithmen zur Anpassung der Suchrichtung kombiniert werden. Das hierauf folgende Gradientenverfahren ist eine solche Verschärfung der Liniensuche. Andere Beispiele sind das Konjugierte-Gradienten-Verfahren, das Newton-Verfahren oder das Quasi-Newton-Verfahren [4]. Das Gradientenverfahren bietet sich von diesen Verfahren jedoch am meisten an, da es die zweite Ableitung nicht benötigt und für die Optimierung einer Funktion gebaut wurde, im Gegensatz zum Konjugierten-Gradienten-Verfahren.

2.2 Gradientenverfahren

Das Gradientenverfahren, auch Verfahren des steilsten Abstiegs, ist ein Abstiegsverfahren mit der Eigenschaft, dass die Suchrichtung mithilfe des negativen Gradienten $-\nabla f$ berechnet wird. Dadurch haben wir eine gut gewählte Suchrichtung. Das Gradientenverfahren konvergiert im Allgemeinen nur sehr langsam, hat jedoch einen größeren Konvergenzbereich als zum Beispiel das Newton-Verfahren. Es ist die einfachste Methode s_k auszurechnen, da, im Gegenteil zu anderen Abstiegsverfahren, nicht die zweite Ableitung berechnet werden muss.
Wenn die Funktion nicht gestört ist, verändert sich Algorithmus 2.1 somit zu:

Algorithmus 2.2 Gradientenverfahren

Eingabe: Startwert $x_0 \in \mathbb{R}^n$, Funktion $f : \mathbb{R}^n \to \mathbb{R}$
Ausgabe: Approximation des Minimums x_*

1: $k = 0$
2: **while** Abbruchbedingung nicht erfüllt **do**
3: Berechne die Suchrichtung $s_k = -\nabla f(x_k)$
4: Bestimme Schrittweite $\alpha_k \in \mathbb{R}$, sodass $f(x_k + \alpha_k s_k) < f(x_k)$
5: Setze $x_{k+1} = x_k + \alpha_k s_k$
6: $k = k + 1$

Wir können leicht nachweisen, dass die Wahl der Schrittweite eine Abstiegsrichtung definiert, also dass die Anforderung $\nabla f(x_k)^\mathsf{T} s_k < 0$ erfüllt ist:

$$\nabla f(x_k)^\mathsf{T}(-\nabla f(x_k)) = -\|\nabla f(x_k)\|^2 < 0$$

Sie erfüllt sogar die Anforderungen einer zulässigen Suchrichtung, der Beweis dafür kann in [5] nachgeschlagen werden.

Wir sehen also, dass unser Gradientenverfahren eine zuverlässige und einfache Wahl für das Optimierungsproblem ist, um die Suchrichtung zu berechnen. Die gewählte Suchrichtung ist außerdem zulässig und wird deshalb eine gute Approximation des Minimums berechnen, wie in Satz 2.5 bewiesen. Als Verfahren für die Schrittweite könnte wieder das Minimieren einer eindimensionalen Hilfsfunktion benutzt werden.
Die Ableitung einer Funktion mit gestörten Werten zu berechnen, funktioniert allerdings schlechter, da die Ableitung wieder gestört sein wird. Deswegen wird für diesen Fall die Hilfe der Stochastik benötigt, um das Minimum zu approximieren. Dieser Fall wird in einem späteren Kapitel betrachtet.
Die zwei Hauptprobleme bei diesem Verfahren sind jedoch einerseits, dass der Gradient, auch wenn er nicht gestört ist, nicht die beste Suchrichtung ist, die gefunden werden kann, und andererseits, dass die Schrittweite einen drastischen Effekt auf die Effizienz des Verfahrens aufweist. Diese Schrittweite ist schwer zu wählen und außerdem virtuell

niemals perfekt. Wenn sie zum Beispiel zu groß initialisiert wurde, kann das Verfahren instabil werden. Die meisten Algorithmen, die sich diesen Problemen annähern, enthalten einen adaptiven Hilfseffekt, um die Lernrate zu bestimmen. Der Algorithmus aus [6] passt die Schrittweite sogar in jedem Schritt an, aber keiner der bekannten Algorithmen verändert die Größe des momentanen Abstiegsschritt. Dies werden wir in Kapitel 6 betrachten und in unserem Verfahren berücksichtigen.

Wir benötigen ein Verfahren mit geringem Aufwand, um die Schrittweite für das Gradientenverfahren auszuwählen, ohne Ableitungen einer weiteren Funktion berechnen zu müssen, da dies zu höheren Gesamtkosten führt. Dafür werden die oft gebrauchten Wolfe-Terminierungsbedingungen benutzt. Sie müssen von einer Schrittweite erfüllt werden, damit sie den Entwicklungspunkt verbessert und der neue Entwicklungspunkt x_{k+1} gesetzt wird.

2.3 Wolfe-Terminierungsbedingungen

Da die Liniensuche nur ein Hilfsschritt in den Abstiegsverfahren ist, muss sie nicht den genauen Wert der Ableitung kennen. Es genügt, einen Punkt zu finden, der nahe genug am Minimum liegt. Dadurch können viele Rechenkosten gespart werden.
Um einen solchen Punkt zu finden und zu prüfen, ob er zulässig ist, werden gewisse Bedingungen benötigt. Bei unserem Verfahren benutzen wir die Wolfe-Bedingungen oder auch Wolfe-Powell-Bedingungen, die aus einer Abstiegsbedingung (auch Armijo-Bedingung genannt) und einer Krümmungsbedingung bestehen. Sie sind die am weitesten verbreiteten Bedingungen für solche Optimierungsprobleme. Für die Funktion f sehen sie wie folgt aus:

$$\begin{aligned} &\text{(W-I)} \quad f(x_{k+1}) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^\mathsf{T} s_k \text{ und} \\ &\text{(W-II)} \quad \nabla f(x_{k+1})^\mathsf{T} s_k \geq c_2 \nabla f(x_k)^\mathsf{T} s_k \end{aligned}$$

mit den zwei Parametern $0 \leq c_1 < c_2 \leq 1$.
Die erste Bedingung vergleicht die tatsächlich erzielte Abnahme der Funktion mit der aus der linearen Approximation erwarteten Abnahme. Sie besagt also, dass akzeptable Funktionswerte unterhalb einer linearen Extrapolation der Steigung $c_1 \nabla f(x_k)$ liegen sollen, also, dass der Schritt klein genug gewählt wurde. Wenn dies nicht der Fall ist, muss die Schrittweite verkürzt werden. Die zweite Bedingung verlangt die Vergrößerung der Steigung um einen bestimmten Faktor c_2 von einem Schritt zum Nächsten, was heißt, dass der Schritt groß genug gewählt wird. Ansonsten muss der Schritt vergrößert werden.
Dass die Bedingungen zulässige Schrittweiten erzeugen, wird in den folgenden Sätzen erklärt und die Beweise sind in [3, 5] nachschlagbar.

Satz 2.6: (Wohldefiniertheit der Armijo-Schrittweitenregel)
Sei $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar, $x \in \mathbb{R}^n$, $s \in \mathbb{R}^n$ mit $\nabla f(x)^\mathsf{T} s < 0$, $\alpha \in (0,1)$, $c_1 \in (0,1)$. Dann existiert ein $\ell \in \mathbb{N}$ mit

$$f(x + \alpha^\ell s) \leq f(x) + c_1 \alpha^\ell \nabla f(x)^\mathsf{T} s.$$

Satz 2.7:
Sei $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar und die Teilfolge $\{x_l\}$ sei beschränkt. Weiter gebe es eine streng monoton wachsende Funktion $\varphi : [0,\infty) \to [0,\infty)$, sodass die durch den Algorithmus 2.1 erzeugten Suchrichtungen folgender Bedingung genügen

$$\|s_l\| \geq \varphi\left(\frac{-\nabla f(x_l)^\mathsf{T} s_l}{\|s_l\|}\right).$$

Dann ist die durch die Armijo-Regel erzeugte Schrittweitenfolge $\{\alpha_l\}$ zulässig.

Satz 2.8:
Sei f nach unten beschränkt und $\{s_k\}$ die zugehörigen Abstiegsrichtungen. Dann sind die durch die Wolfe-Bedingungen erzeugten Schrittweiten zulässig.

Durch die drei vorherigen Sätze wurde nun geklärt, dass immer eine zulässige Schrittweitenfolge mithilfe der Wolfe-Bedingungen gefunden wird, wenn die Funktion nach unten beschränkt ist. Der Fall, dass eine Funktion nicht nach unten beschränkt ist, interessiert uns dabei nicht allzu sehr, da eine solche Funktion auch kein globales Minimum besitzt. Ein lokales Minimum kann zwar mittels der Abstiegsverfahren gefunden werden, aber es könnte passieren, dass das Verfahren nicht terminiert oder sich die Entwicklungspunkte dem negativen Unendlichen annähern und nicht dem lokalen Minimum. Dies wird auch bei einer Funktion in den Experimenten in Kapitel 7 der Fall sein.
Für stetig differenzierbare, nach unten beschränkte Funktionen gibt es also immer ein Intervall, in dem die Wolfe-Bedingungen erfüllt sind. Dabei stellen wir uns noch die Frage, wie die in den Wolfe-Bedingungen vorkommenden Parameter c_1 und c_2 gewählt werden sollen. Die Wahl dieser Parameter hat einen großen Einfluss auf die Liniensuche, dabei gilt:

$c_1 = 0$	akzeptiert alle Werte unter $f(x_k)$
$c_1 = 1$	lehnt alle Punkte von konvexen Funktionen ab
$c_2 = 0$	nimmt nur Werte an, wo $\nabla f(x_{k+1}) \geq 0$ gilt
$c_2 = 1$	akzeptiert alle Punkte mit einer Steigung größer als $\nabla f(x_k)$

Wir sehen also, dass der Parameter c_1 eher in der Nähe von 0 gewählt werden sollte, damit auch konvexe Funktionen minimiert werden können und zusätzlich viele Schritt-

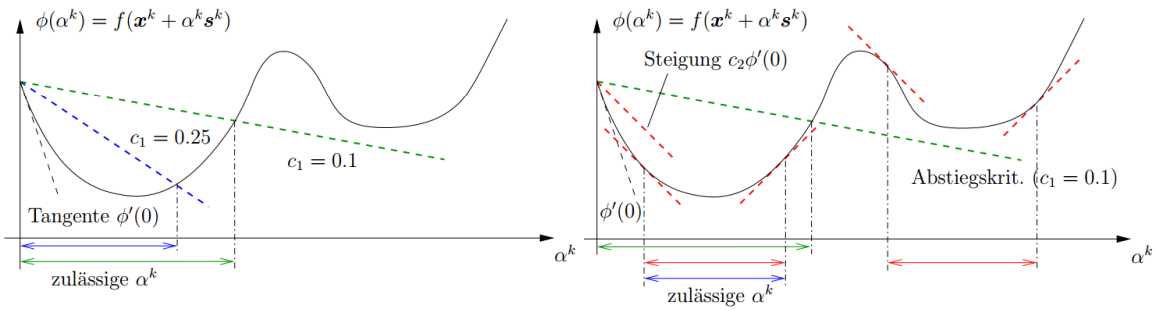

Abbildung 2.1: Veranschaulichung Abstiegs- und Krümmungsbedingung. Die linke Abbildung beschreibt dabei das Verhalten der Abstiegsbedingung und die rechte der Krümmungsbedingung mit der Annahme, dass $c_1 = 0.1$ gewählt wurde. Die Grafik stammt aus [4].

weiten angenommen werden. Dahingegen sollte c_2 eher in der Nähe von 0.5 bis 1 gewählt werden, damit genügend Punkte ausgewählt werden dürfen und die Steigung auch hinreichend wächst. Typischerweise wird $c_2 = 0.9$ gesetzt, wenn s_k durch ein Newton- oder Quasi-Newton-Verfahren bestimmt wird, bzw. $c_1 = 0.1$, wenn das Konjugierte-Gradienten-Verfahren benutzt wird [4].
In Kapitel 6 wird gezeigt, dass wir in unserem selbstgebauten stochastischen Verfahren $c_1 = 0.05$ und $c_2 = 0.8$ wählen sollten. Diese Werte wurden in [1] durch empirische Auswertungen festgesetzt. Ob diese Werte sinnvoll sind, wird dann in Kapitel 7 mithilfe numerischer Experimente gezeigt.

Die beiden Bedingungen (W-I) und (W-II) werden als schwache Wolfe-Terminierungsbedingungen bezeichnet. Diese reichen jedoch nicht immer aus, da sonst sehr viele Punkte erhalten werden können, an denen die Funktionswerte zwar klein sind, der Gradient allerdings sehr groß sein kann. Diese Punkte sollen lieber herausgefiltert werden, damit die Punkte gewählt werden können, die näher am Minimum liegen. Also benötigen wir die starke Form der Wolfe-Bedingungen, deren Auswirkungen in Abbildung 2.1 gezeigt werden. Bei ihnen verändert sich nur (W-II) zu

$$\text{(W-IIa)} \qquad |\nabla f(x_{k+1})^\mathsf{T} s_k| \geq |c_2 \nabla f(x_k)^\mathsf{T} s_k|.$$

Wir benötigen die Wolfe-Bedingungen jedoch in einer probabilistischen Weise, da wir davon ausgehen werden, dass unsere Funktion und deren Gradient gestört sind. Dafür benötigen wir einige stochastische Grundlagen, die im nächsten Kapitel erklärt werden. Wir werden durch die probabilistischen Wolfe-Bedingungen noch einen weiteren Parameter bekommen, der angibt, mit welcher Wahrscheinlichkeit die Bedingungen erfüllt werden müssen, damit eine Schrittweite gewählt wird. Dieser wird dann in Kapitel 6 auch auf einen festgelegten Wert gesetzt.

3 Stochastische Grundlagen

Bis jetzt haben wir gesehen, wie wir das Gradientenverfahren auf eine Funktion anwenden, die keine Störungen besitzt. Wenn jedoch eine Funktion mit Störung gegeben ist, werden einige wahrscheinlichkeitstheoretischen Grundlagen benötigt, um die Prozesse für den gestörten Fall umzubauen. Wir benutzen die Wahrscheinlichkeitstheorie, da das Gradientenverfahren unter Störungen nicht stabil ist. Der Grund dafür ist, dass der Schritt der Liniensuche bei den Wolfe-Bedingungen harte Entscheidungen treffen muss, um den Suchraum einzuschränken und die Effizienz, die wir bekommen, bei Störungen zusammenbricht. Es wird versucht, die gestörten Werte so gut wie möglich herauszufiltern, indem die Fehler abgeschätzt und nicht beachtet werden.
Um unser Verfahren aufzubauen, werden Elemente aus der mehrdimensionalen Wahrscheinlichkeitstheorie gebraucht. Um diese zu verstehen, muss jedoch erst einmal die Wahrscheinlichkeitstheorie für eindimensionale Probleme betrachtet werden, welche zu jeder Anfangsvorlesungen der Stochastik gehört. In den nächsten zwei Unterkapiteln wird deswegen die eindimensionale Wahrscheinlichkeitstheorie erläutert und daraufhin für das mehrdimensionale Problem verallgemeinert. Damit wird auf die multivariate Normalverteilung und den multivariaten Zentralen Grenzwertsatz gezielt. Diesen benötigen wir, um voraussetzen zu können, dass die Wahrscheinlichkeiten, die im Verfahren benutzt werden, normalverteilt sind. Dadurch können wir einige nützliche Eigenschaften der Normalverteilung anwenden.

3.1 Eindimensionale Wahrscheinlichkeitstheorie

Wir brauchen zunächst die elementare eindimensionale Wahrscheinlichkeitstheorie, um die multivariate Normalverteilung verstehen zu können, welche im Kapitel 4.1 benutzt werden soll, um den Gauß-Prozess aufzustellen. Die nachfolgenden wahrscheinlichkeitstheoretischen Formulierungen stammen zum größten Teil aus [7].

Definition 3.1: (σ-Algebra)

Sei Ω eine beliebige Menge und sei $\mathcal{F}$ eine Menge von Teilmengen von Ω mit der Eigenschaft, dass

(i) $\Omega \in \mathcal{F}$ und $\varnothing \in \mathcal{F}$,
(ii) falls $A \in \mathcal{F}$, dann ist auch $A^c \in \mathcal{F}$,
(iii) falls $A_n \in \mathcal{F}$, für alle $n \in \mathbb{N}$, dann ist auch $\bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}$.

Dann heißt $\mathcal{F}$ eine σ-Algebra und das Paar $(\Omega, \mathcal{F})$ heißt ein Messraum.

Definition 3.2: (Wahrscheinlichkeitsmaß)

Sei $(\Omega, \mathcal{F})$ ein Messraum und sei $\mathbb{P} : \mathcal{F} \to \mathbb{R}_+$ eine Abbildung mit folgenden Eigenschaften:

(i) $\mathbb{P}(\Omega) = 1$,

(ii) $\mathbb{P}(\varnothing) = 0$,

(iii) falls die Mengen $A_i \in \mathcal{F}$, $i \in \mathbb{N}$, paarweise disjunkt sind, dann gilt

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

Dann heißt $\mathbb{P}$ ein Wahrscheinlichkeitsmaß auf dem Messraum $(\Omega, \mathcal{F})$, und das Triple $(\Omega, \mathcal{F}, \mathbb{P})$ wird ein Wahrscheinlichkeitsraum genannt.

Definition 3.3: (Zufallsvariable)

Sei $(\Omega, \mathcal{F})$ ein Messraum und $f : \Omega \to \mathbb{R}$ eine reellwertige Funktion. Dann heißt f eine messbare Funktion von $(\Omega, \mathcal{F})$ nach $(\mathbb{R}, \mathcal{B})$, genau dann, wenn für alle $B \in \mathcal{B}$ gilt

$$f^{-1}(B) := \{\omega \in \Omega : f(\omega) \in B\} \in \mathcal{F}.$$

Wenn $(\Omega, \mathcal{F}, \mathbb{P})$ ein Wahrscheinlichkeitsraum und $X : \Omega \to \mathbb{R}$ eine messbare Funktion ist, so nennen wir X eine reellwertige Zufallsvariable.

Die Ausdrücke Wahrscheinlichkeitsmaß und Verteilung werden synonym verwendet. Also können wir nun die Verteilungsfunktion einer reellen Zufallsvariable definieren, indem wir sagen, der Wert der Verteilungsfunktion an der Stelle x entspricht der Wahrscheinlichkeit, dass die Zufallsvariable einen Wert kleiner oder gleich x annimmt. Formal ist sie also definiert als $F : \mathbb{R} \to [0, 1]$ mit

$$\begin{aligned} F(x) &:= \mathbb{P}(\{\omega \in \Omega : f(\omega) \leq x\}) = \mathbb{P}_f((-\infty, x]) \text{ oder auch abgekürzt} \\ &:= \mathbb{P}(X \leq x) \end{aligned}$$

in Abhängigkeit von der Zufallsvariable f bzw. X und dem Maß $\mathbb{P}$. Somit gelten die Eigenschaften $\lim_{x\to\infty} F(x) = 1$ und $\lim_{x\to-\infty} F(x) = 0$.

Definition 3.4: (Wahrscheinlichkeitsdichte)

Sei F Verteilungsfunktion eines Maßes auf $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$. Dann heißt F absolut stetig (bzgl. des Lebesgue-Maßes[1]), falls es eine positive, messbare Funktion $f : \mathbb{R} \to [0, \infty)$ gibt, sodass für alle $s < t \in \mathbb{R}$,

$$\mathbb{P}(s < X \leq t) = F(t) - F(s) = \int_s^t f(x) d\lambda(x)$$

gilt, wobei λ das Lebesgue-Maß ist. Wir nennen in diesem Fall die Funktion f die Wahrscheinlichkeitsdichte des Wahrscheinlichkeitsmaßes $\mathbb{P}$.

[1] Das Lebesgue-Maß ist das eindeutige Maß, welches jedem Intervall seine Länge zuordnet, das heißt, es gilt für alle $s, t \in \mathbb{R} : \mathbb{P}(s < X \leq t) = t - s$, siehe [7]. Wir schreiben auch dx statt $d\lambda(x)$ für das Integral des Lebesgue-Maßes.

Wenn eine Verteilungsfunktion nicht stetig ist, kann sie diskret sein. Dies ist der Fall, wenn sie nur auf einer endlichen oder abzählbar unendlichen Menge definiert ist oder sie auf einer beliebigen Menge definiert ist, aber nur endlich viele bzw. abzählbar viele Werte annehmen kann. Wir bezeichnen die Menge dieser Werte als I. Der diskrete Fall muss jedoch nicht allzu lange betrachtet werden, da die Normalverteilung benutzen werden soll, welche stetig ist. Trotzdem werden zum besseren Verständnis einige nachfolgende Definitionen auch im diskreten Fall betrachtet.
Wir haben bis jetzt gesehen, dass eine Verteilungsfunktion eine Zufallsvariable vollständig charakterisiert. Wir benötigen jedoch noch andere Kenngrößen, die eine Verteilung ausmachen.

Definition 3.5: (Erwartungswert)

Der Erwartungswert oder auch Mittelwert einer Zufallsvariable $X : \Omega \to \mathbb{R}$ ist definiert als

$$\mathbb{E}(X) := \sum_{i \in I} x_i \mathbb{P}(X = x_i) \text{ für diskrete Verteilungsfunktionen bzw.}$$
$$\mathbb{E}(X) := \int_\Omega x f(x) d\lambda(x) = \int_\Omega X d\mathbb{P} \text{ für stetige Verteilungsfunktionen.}$$

Er ist der Wert, welcher im Durchschnitt erwartet werden kann, wenn das Zufallsexperiment ausgeführt wird, zu dem die Zufallsvariable X gehört.

Definition 3.6: (Varianz)

Die Varianz gibt an, wie sehr sich die Verteilung um den Erwartungswert herum streut. Sie ist für eine Zufallsvariable $X : \Omega \to \mathbb{R}$ definiert als

$$Var(X) := \mathbb{E}\left((X - \mathbb{E}(X))^2\right) = \sum_{x \in I} (x - \mathbb{E}(X))^2 \mathbb{P}(X = x) \text{ im diskreten,}$$
$$Var(X) := \mathbb{E}\left(X - \mathbb{E}(X))^2\right) = \int_\Omega (X - \mathbb{E}(X))^2 d\mathbb{P} \text{ im stetigen Fall.}$$

Dabei muss $\mathbb{E}(|X|) < \infty$ gelten. Die Quadratwurzel der Varianz, $\sigma^2 = Var(X)$, wird im Allgemeinen als Standardabweichung bezeichnet.

Definition 3.7: (Kovarianz)

Die Kovarianz gibt den Zusammenhang zwischen zwei Zufallsvariablen mit gemeinsamer Wahrscheinlichkeitsverteilung an. Sie ist definiert durch

$$Cov(X, Y) = \mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))],$$

wobei $\mathbb{E}(X)$, $\mathbb{E}(Y)$ und $\mathbb{E}(XY)$ existieren müssen. Wenn $Cov(X, Y) = 0$ gilt, sind die Zufallsvariablen X und Y unkorreliert, was für zwei normalverteilten Variablen bedeutet, dass sie unabhängig sind.

Die Kovarianz wird auch benötigt, um den Korrelationskoeffizienten

$$\rho = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

zu berechnen. Dieser gibt den linearen Zusammenhang zwischen den Zufallsvariablen an und kann Werte zwischen -1 für vollständigen negativen und $+1$ für vollständigen positiven Zusammenhang annehmen. Dabei bedeutet ein negativer Zusammenhang, dass große Werte von X kleine Werte von Y implizieren und ein positiver Zusammenhang, dass große Werte von X große Werte von Y implizieren. Wenn der Koeffizient den Wert 0 annimmt, gibt es keinen Zusammenhang zwischen den beiden Zufallsvariablen [9].

Nun fehlt in den Grundlagen der Wahrscheinlichkeitstheorie nur noch die bedingte Wahrscheinlichkeit, die in Kapitel 4.1 benötigt wird.

Definition 3.8: (Bedingte Wahrscheinlichkeit)
Sei $(\Omega, \mathcal{F}, \mathbb{P})$ ein Wahrscheinlichkeitsraum und seien $A, B \in \mathcal{F}$. Sei $\mathbb{P}(B) > 0$. Dann heißt

$$\mathbb{P}(A|B) := \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

die bedingte Wahrscheinlichkeit von A gegeben B.

Dabei ist $\mathbb{P}(A \cap B) \leq \min(\mathbb{P}(A), \mathbb{P}(B))$ die Wahrscheinlichkeit des gleichzeitigen Eintretens von A und B. Nun stellt $\mathbb{P}(A|B)$ die Wahrscheinlichkeit dar, dass A eintritt, wenn angenommen wurde, dass B bereits eingetreten ist.

Nun haben wir alle elementaren eindimensionalen Grundbausteine der Stochastik beisammen und können uns darum kümmern, was es heißt, wenn eine Zufallsvariable normalverteilt ist, und wie wir dies für unser Verfahren benutzen können.

3.2 Eindimensionale Normalverteilung

Die Normalverteilung oder auch Gauß-Verteilung ist eine stetige Wahrscheinlichkeitsverteilung, deren besondere Bedeutung mit dem Zentralen Grenzwertsatz zusammenhängt. Sie hat die Wahrscheinlichkeitsdichte

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

mit dem Mittelwert $\mu \in \mathbb{R}$, der Standardabweichung $\sigma > 0$ und der Varianz σ^2. Die Verteilungsfunktion ist definiert als

$$F(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right).$$

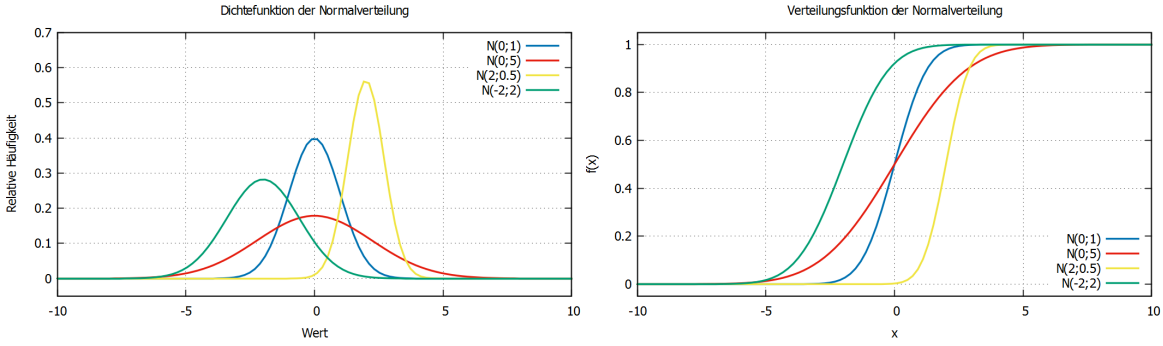

Abbildung 3.1: Veranschaulichung der Normalverteilung $\mathcal{N}(\mu, \sigma^2)$ für verschiedene Werte des Mittelwerts μ und der Varianz σ^2.

Dabei ist $\text{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x exp(-t^2)dt$ die gaußsche Fehlerfunktion.
Wenn eine Zufallsvariable X normalverteilt ist, wird es mit $X \sim \mathcal{N}(\mu, \sigma^2)$ abgekürzt. Eine Zufallsvariable für die $X \sim \mathcal{N}(0, 1)$ gilt, heißt standardnormalverteilt. Abbildung 3.1 zeigt die Dichte- und Verteilungsfunktionen von verschiedenen Normalverteilungen. Sie sind symmetrisch zum Mittelwert μ und besitzt dort auch ihr Maximum.

Die Normalverteilung ist meistens die erste Wahl, wenn es, wie bei uns, um die Verteilung von Abweichungen oder Fehlern geht. Der Grund dafür ist der Zentrale Grenzwertsatz, welcher besagt, dass Verteilungen in vielen Fällen als normalverteilt angenommen werden können. Sein Beweis kann in [7] nachgelesen werden.

Satz 3.9: (Zentraler Grenzwertsatz)

Seien $X_i,\ i \in \mathbb{N}$ unabhängige, identisch verteilte Zufallsvariablen mit $\mathbb{E}(X_i) = \mu$ und $Var(X_i) = \sigma^2 < \infty$. Dann konvergiert

$$Z_n := \frac{\sum_{i=1}^{n}(X_i - \mu)}{\sqrt{n\sigma^2}}$$

in Verteilung gegen eine Gaußsche Zufallsvariable mit dem Mittelwert 0 und der Varianz 1.

Eine Folge von reellen Zufallsvariablen $\{X_n\}_{n\in\mathbb{N}}$ konvergiert dabei in Verteilung gegen eine Zufallsvariable X, $X_n \xrightarrow{\text{d}} X$, genau dann, wenn die Verteilungsfunktionen $F_n(x) = \mathbb{P}(X_n \leq x)$ der Zufallsvariablen X_n schwach gegen die Verteilungsfunktion $F(x) = \mathbb{P}(X \leq x)$ der Zufallsvariable X konvergiert. Das bedeutet, dass für alle $c \in \mathbb{R}$, für die F stetig ist, $F_n(c) \to F(c)$ gilt.

Für uns bedeutet der Zentrale Grenzwertsatz nun, dass wir davon ausgehen können, dass das Maß, welches wir später im Verfahren benutzen wollen, als normalverteilt angenommen werden kann. Nun muss dies natürlich noch für eine mehrdimensionale Wahrscheinlichkeit gezeigt werden.

3.3 Mehrdimensionale Wahrscheinlichkeitstheorie

Wir benötigen in unserem Verfahren eine mehrdimensionale Normalverteilung, da wir die eindimensionale nicht auf multivariate Funktionen ausführen können. Dafür werden die Definitionen der eindimensionalen Prozesse minimal umgeschrieben. Bei uns im speziellen Beispiel des Codes und der Experimente wird es die bivariate Normalverteilung für zweidimensionale Funktionen sein.
Die Informationen in diesem und dem nächsten Kapitel stammen aus [8, 9, 10].

In der multivariaten Form der Wahrscheinlichkeitstheorie gibt es statt einer Zufallsvariable einen Zufallsvektor $X : (\Omega, \mathcal{F}, \mathbb{P}) \to (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$. Er wird geschrieben als $X = (X_1, \ldots, X_n)^\mathsf{T}$, wobei die einzelnen Komponenten Zufallsvariablen $X_1, \ldots, X_n$ sind. Der Zufallsvektor heißt stetig, wenn es eine Funktion $f(x) = f(x_1, \ldots, x_n)$ gibt, sodass für alle $s_i < t_i \in \mathbb{R},\ i \in \{1, \ldots, n\}$ gilt, dass

$$\mathbb{P}(s_1 < X_1 \leq t_1, \ldots, s_n < X_n \leq t_n) = \int_{s_1}^{t_1} \ldots \int_{s_n}^{t_n} f(u_1, \ldots, u_n) du_1 \ldots du_n.$$

Die Funktion f ist dann die multivariate Dichte des Zufallsvektors X. Für eine diskrete Zufallsvariable mit den möglichen Werten $\tilde{x}_1, \tilde{x}_2, \ldots \in \mathbb{R}^n$ ist die diskrete Dichte bestimmt durch

$$f(\tilde{x}_i) = \mathbb{P}(X = \tilde{x}_i).$$

Die zugehörigen n-dimensionalen Verteilungsfunktionen sind definiert als

$$\begin{aligned} F(x) = F(x_1, \ldots, x_n) &= \mathbb{P}(X_1 \leq x_1, \ldots, X_n \leq x_n) \\ &= \begin{cases} \int_{-\infty}^{x_1} \ldots \int_{-\infty}^{x_n} f(u_1, \ldots, u_n) du_1 \ldots du_n & X \text{ stetig} \\ \sum_{\tilde{x_i} \leq x} f(\tilde{x_i}) & X \text{ diskret,} \end{cases} \end{aligned}$$

wobei $\tilde{x_i} \leq x$ bedeutet, dass jede Komponente in $\tilde{x_i}$ kleiner oder gleich der entsprechenden Komponente in x ist. F besitzt, wie auch in der eindimensionalen Variante, die Eigenschaften:

$$\begin{aligned} &0 \leq F(x) \leq 1 \text{ für alle } x, \\ &\lim_{x \to \infty} F(x) = 1, \text{ wobei alle Komponenten } x_i \to \infty, \\ &\lim_{x_i \to -\infty} F(x) = 0 \text{ für alle Komponenten } x_i, \end{aligned}$$

und außerdem noch die erweiterte Eigenschaft:

$$f(x) = \frac{\partial^n F(x)}{\partial x_1 \ldots \partial x_n}, \text{ wenn } f(x) \text{ an der Stelle } x \text{ stetig ist.}$$

Es können wieder die drei stochastischen Kennwerte der Zufallsvariable, die nun ein Zufallsvektor ist, angegeben werden. Die einzelnen Komponenten des Erwartungswerts eines Zufallsvektors X sind definiert durch den Erwartungswert der einzelnen Komponenten $X_1, \ldots, X_n$. Er wird als Erwartungswertvektor bezeichnet und wird abgekürzt durch:

$$\mu := \mathbb{E}(X) = (\mathbb{E}(X_1), \ldots, \mathbb{E}(X_n))^\mathsf{T}$$

mit den einzelnen Komponenten $\mu_i := \mathbb{E}(X_i)$. Dabei muss für alle $i \in \{1, \ldots, n\}$ die Eigenschaft $\mu_i < \infty$ erfüllt sein.
Für die Varianzen der einzelnen Komponenten gilt

$$\sigma_i^2 := Var(X_i) = \mathbb{E}\left((X_i - \mathbb{E}(X_i))^2\right) =: \sigma_{ii}.$$

Sie sind die Diagonaleinträge für die Kovarianzmatrix mit den Einträgen

$$\sigma_{ij} := Cov(X_i, X_j) = \mathbb{E}[(X_i - \mathbb{E}(X_i))(X_j - \mathbb{E}(X_j))],$$

wobei $\mathbb{E}(X_i)^2 < \infty$ für alle $i \in \{1, \ldots, n\}$ eintreffen muss. Insgesamt sieht die Kovarianzmatrix, welche symmetrisch ist, folgendermaßen aus:

$$\Sigma := Cov(X, X) = \mathbb{E}[(X - \mathbb{E}(X))(X - \mathbb{E}(X))^\mathsf{T}] = \begin{pmatrix} \sigma_{11} & \cdots & \sigma_{1n} \\ \vdots & & \vdots \\ \sigma_{n1} & \cdots & \sigma_{nn} \end{pmatrix}.$$

Dies ist im Unterschied zur eindimensionalen Kovarianz nur für einen Zufallsvektor gegeben. Wenn hingegen die Kovarianz zweier Zufallsvektoren $X = (X_1, \ldots, X_p)^\mathsf{T}$ und $Y = (Y_1, \ldots, Y_q)^\mathsf{T}$ untersucht werden soll, betrachten wir

$$\begin{aligned} \Sigma_{XY} := Cov(X, Y) &= \mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))^\mathsf{T}] \\ &= \begin{pmatrix} Cov(X_1, Y_1) & \cdots & Cov(X_1, Y_q) \\ \vdots & & \vdots \\ Cov(X_p, Y_1) & \cdots & Cov(X_p, Y_q) \end{pmatrix}. \end{aligned}$$

Nun haben wir wieder alle Bausteine der diesmal mehrdimensionalen Stochastik beschrieben und müssen nun die mehrdimensionale Normalverteilung verstehen. Für unseren Korrelationskoeffizienten gelten in der zweidimensionalen Verteilung allerdings die selben Rechenregeln wie in der eindimensionalen Wahrscheinlichkeitstheorie, da die beiden Werte dort für zwei Zufallsvariablen berechnet wurden und dies nun die einzelnen Komponenten unseres Zufallsvektors darstellen.

3.4 Mehrdimensionale Normalverteilung

Bei der mehrdimensionalen Normalverteilung betrachten wir eine Verallgemeinerung der eindimensionalen Normalverteilung für einen n-dimensionalen Raum. Ein Zufallsvektor $X = (X_1, \ldots, X_n)^\mathsf{T} \in \mathbb{R}^n$ ist multivariat normalverteilt, wenn er die Dichte

$$f(x) = \frac{1}{\sqrt{(2\pi)^n det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^\mathsf{T}\Sigma^{-1}(x-\mu)\right)$$

besitzt, wobei $\mu = \mathbb{E}(X)$ dem Erwartungswertvektor entspricht und Σ der Kovarianzmatrix, welche positiv definit sein muss. Eine Matrix ist positiv definit genau dann, wenn alle Eigenwerte der Matrix positiv sind. Eine andere Definition der Verteilung ist, dass X n-dimensional normalverteilt ist, wenn $c^\mathsf{T}X$ eindimensional normalverteilt ist, für alle $c \in \mathbb{R}^n$.
Bei der mehrdimensionalen Normalverteilung wird es diesmal mit $X \sim \mathcal{N}_n(\mu, \Sigma)$ abgekürzt, im speziellen Fall für $n = 2$ wird es eine bivariate Normalverteilung genannt.
Für die Verteilungsfunktion gibt es keine Formel, da die Integrale numerisch berechnet werden müssen.

Auch der Zentrale Grenzwertsatz kann für einen Zufallsvektor formuliert werden, dabei wird er auf den Erwartungsvektor $\mathbf{0}$ und die Kovarianzmatrix Σ standardisiert.

Satz 3.11: (Zentraler Grenzwertsatz für Zufallsvektoren)
Seien $X_i,\ i \in \mathbb{N}$ unabhängige, identisch verteilte Zufallsvektoren in $\mathbb{R}^n$ mit Erwartungswertvektor μ und Kovarianzmatrix Σ. Dann gilt für $\bar{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$, dass

$$\sqrt{n}(\bar{X}_n - \mu) \xrightarrow{\mathrm{d}} Z, \text{ mit } Z \sim N_n(0, \Sigma).$$

Durch den Zentralen Grenzwertsatz für mehrdimensionale Zufallsvektoren haben wir nun herausgefunden, dass wir unsere mehrdimensionalen Wahrscheinlichkeiten als mehrdimensional normalverteilt mit Erwartungswert $\mathbf{0}$ und Kovarianz Σ annehmen können. Wie wir dies genau benutzen wollen, betrachten wir im folgenden Kapitel.

4 Probabilistische Liniensuche

Da die Grundlagen definiert wurden, kommen wir nun zu den zusätzlichen Elementen, wie der Idee eines Gauß-Prozesses, also einen robusten, aber kostengünstigen Ersatz, und der eines weit verbreiteten Werkzeugs der Bayes-Optimierung, dem *Expected Improvement*. Diese wurden vorher noch nicht benötigt, doch sie halten unser Verfahren zusammen. Dazu kommen noch die probabilistischen Formulierungen der Wolfe-Bedingungen, die im Kapitel 2 bereits angesprochen wurden. Dieses Kapitel stützt sich zu weiten Teilen auf die Veröffentlichung [1], die in der Einleitung schon vorgestellt wurde.

4.1 Gauß-Prozess

In einer weiteren Vorgehensweise zur globalen Optimierung geht es darum, ein Objekt f mit einem Wahrscheinlichkeitsmaß $p(f)$, zu modellieren. Meistens wird dies durch einen Gauß-Prozess erreicht, was bedeutet, dass die Wahrscheinlichkeit, einen bestimmten Wert anzunehmen, normalverteilt ist. Der Unterschied besteht darin, dass statt einem Erwartungswert und Kovarianzen jeweils eine Funktion gegeben ist. Der Gauß-Prozess besitzt eine besonders hohe Transparenz und Verständlichkeit, weil er hauptsächlich auf Linearer Algebra und der multivariaten Normalverteilung im Sinne der Fehlerrechnung basiert.
In einem Gauß-Prozess soll eine approximierende Funktion an diskrete Stützstellen gefunden werden, dabei kann es als überwachtes Maschinenlernverfahren zur Modellierung durch endlich viele diskrete Trainingsdaten verwendet werden. Was dies bedeutet, werden wir in Kapitel 5 erläutern.

Auf der Suche nach dem Minimum werden die Auswertungspunkte durch eine Nutzenfunktion $u[p(f)]$ ausgewählt. Es wird also eine Liste von möglichen Punkten aufgestellt, woraus danach in Kapitel 4.2 mithilfe eines Bayes-Optimierers der bestmögliche ausgewählt wird. An das Wahrscheinlichkeitsmaß $p(f)$ gibt es somit zwei Voraussetzungen. Erstens muss es robust zu Unregelmäßigkeiten des Objektes sein und zweitens muss es analytische Berechnungen von diskreten Punkten erlauben, damit in der Liniensuche kein anderer Optimierer aufgerufen werden muss. Beide Eigenschaften sind durch einen einmal integrierten Wiener-Prozess erfüllt. Ein Wiener Prozess ist dabei ein stochastisches Modell für die Brownsche Bewegung. Die Brownsche Bewegung ist die zeitliche Irrfahrt kleiner Teilchen in einer Flüssigkeit, siehe [11].
Wir benutzen den zentrierten Gauß-Prozess, da dieser sehr nah an einen Wiener-Prozess heran kommt, das heißt, dass der Erwartungswert, bzw. der Erwartungswertvektor konstant $\mathbf{0}$ ist. Es wird mit $p(f) \sim \mathcal{GP}(m(t), k(t, t'))$ bezeichnet, wenn das Wahrscheinlichkeitsmaß ein Gaußscher Prozess ist. Dabei wäre normalerweise $m(t) = \mathbb{E}(X_t)$ die Erwartungswertfunktion, welche bei uns jedoch entfällt. Die Kovarianzfunktion, die bei

uns durch

$$k(t,t') = Cov(X_t, X_{t'}) = \theta^2 \left[\frac{1}{3}\min^3(\tilde{t},\tilde{t'}) + \frac{1}{2}|t-t'|\min^2(\tilde{t},\tilde{t'})\right] \tag{4.1}$$

definiert ist, tritt an die Stelle von der Kovarianzmatrix Σ aus dem vorherigen Kapitel 3.3. Wir werden sie auch mit $k_{tt'} = k(t,t')$ abkürzen. Hierbei ist $\tilde{t} := t+\tau$ und $\tilde{t'} := t'+\tau$ eine Verschiebung um eine Konstante $\tau > 0$, die sicherstellt, dass wir in einem positiven Bereich liegen.

Bei unserem Gauß-Prozess-Ersatz nehmen wir an, dass die Regression auf f und ∇f aus N Beobachtungen besteht, die jeweils Paare (y_t, y'_t) sind. Dabei is y der gestörte Funktionswert und y' der gestörte Ableitungswert. Eine Liniensuche sammelt dabei typischerweise weniger als 10 Datenpunkte in einem Durchlauf, deswegen lässt sich der Prozess mit einer Gramschen Matrix und dem Lösen eines linearen Gleichungssystems praktisch mit den selben niedrigen Kosten realisieren. Dies wird auch im Programmiercode angewendet und in Kapitel 7 werden Experimente mit der Anzahl der maximalen Datenpunkte durchgeführt. Die Datenpunkte werden die Kandidaten für unseren Entwicklungspunkt sein. Eine Gram-Matrix ist dabei definiert als die Matrix G mit den Einträgen $G_{ij} := \delta(v_i, v_j)$ mit $i,\ j \in \{1, \cdots, n\}$ für einen Vektor $v = (v_1, \cdots, v_n)^\mathsf{T}$ und eine Bilinearform δ. Dabei interessiert uns die Bilinearform nicht allzu sehr, da wir stattdessen die untenstehende Funktion benutzen werden.
Das Gaußsche Maß, also das normalverteilte Maß, ist unter linearen Abbildungen abgeschlossen, weswegen wir den Wiener Prozess darstellen können als

$$p(f, f') \sim \mathcal{GP}\left(0, \begin{bmatrix} k & k^{\partial} \\ {}^{\partial}k & {}^{\partial}k^{\partial} \end{bmatrix}\right), \tag{4.2}$$

mit

$${}^{\partial^i}k^{\partial^i} = \frac{\partial^{i+j} k(t,t')}{\partial t^i \partial t'^j},$$

also gilt

$$\begin{aligned} k^{\partial}(t,t') &= \theta^2 \left[\mathbb{1}_{t<t'} t^2/2 + \mathbb{1}_{t\geq t'}(tt' - t'^2/2)\right], \\ {}^{\partial}k(t,t') &= \theta^2 \left[\mathbb{1}_{t'<t} t'^2/2 + \mathbb{1}_{t'\geq t}(tt' - t^2/2)\right], \text{ und} \\ {}^{\partial}k^{\partial}(t,t') &= \theta^2 \min(t,t'). \end{aligned}$$

Wenn wir nun eine Menge von Auswertungen (T, Y, Y') gegeben haben, welche jeweils Vektoren (t_i, y_i, y'_i) sind, wobei t_i den t-Wert des Kandidaten i in der Formel für den nächsten Entwicklungspunkt $x_{k+1} = x_k + \alpha_k t_i s_k$ darstellt, und diese unabhängige Wahr-

scheinlichkeiten besitzen, bekommen wir, dass $p(f|Y, Y')$ ein Gauß-Prozess ist mit der Mittelwertsfunktion

$$\mu(t) = \underbrace{\begin{bmatrix} k_{Tt} \\ k^{\partial}_{Tt} \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} k_{TT} + \sigma_f^2 \mathbf{I} & k^{\partial}_{TT} \\ {}^{\partial}k_{TT} & {}^{\partial}k^{\partial}_{TT} + \sigma_{f'}^2 \mathbf{I} \end{bmatrix} \right)^{-1}}_{=:g^{\mathsf{T}}(t)} \begin{bmatrix} Y \\ Y' \end{bmatrix}, \tag{4.3}$$

und der Kovarianzfunktion

$$\tilde{k}(t, t') = k_{tt'} - g^{\mathsf{T}}(t) \begin{bmatrix} k_{Tt'} \\ k^{\partial}_{Tt'} \end{bmatrix}. \tag{4.4}$$

Dabei sind σ_f und $\sigma_{f'}$ die Geräuschpegel von f und ∇f. Sie zeigen an, wie die Funktion und ihre Ableitung gestört sind. In Kapitel 6 wird dies genauer erläutert.
Die erhaltene marginale Varianz wird bezeichnet mit $\mathbb{V}(t) = \tilde{k}(t, t)$, sie wird nachher für den Bayes-Optimierer benötigt.
Eine nützliche Eigenschaft von μ ist, dass es stückweise kubisch ist. Um dies zu sehen bemerken wir, dass es mindestens drei nicht verschwindende Ableitungen gibt:

$${}^{\partial^2}k(t, t') = \theta^2 \mathbb{1}_{t \le t'}(t' - t),\ {}^{\partial^3}k = -\theta^2 \mathbb{1}_{t \le t'} \text{ und } {}^{\partial^2}k^{\partial}(t, t') = \theta^2 \mathbb{1}_{t \le t'}.$$

Die Form von μ ist entscheidend für unser Ziel: Wenn N Werte von f und ∇f gesammelt sind, können alle lokalen Minima von μ in linearer Laufzeit mit einem einzigen Durchlauf durch die Kandidaten $t_{i-1} < t < t_i,\ i = 1, \ldots, N$ gefunden werden, wobei $t_0 = 0$ der Startpunkt ist, an dem (y_0, y_0') von der vorherigen Liniensuche vererbt wurde. Für die meisten Liniensuchen gilt dabei, wie bereits angemerkt, $N < 10$. An jeder Stelle t ist $\mu(t)$ ein kubisches Polynom mit mindestens einem Minimum, welches von einer trivialen quadratischen Rechnung der drei Skalare $\mu'(t),\ \mu''(t)$ und $\mu'''(t)$ gefunden werden kann. Ein ergänzender Vorteil ist, dass die Existenz von weiteren höheren Ableitungen nicht benötigt wird. Im Algorithmus werden diese Eigenschaften nach jeder Auswertung (y_N, y_N') benutzt, um eine kurze Liste von Kandidaten für die nächste Entwicklung aufzustellen. Diese bestehen aus weniger als N lokalen Optimierern t_i von $\mu(t)$ und einem additiven Entwicklungsknoten $t_{max} + \alpha$, mit t_{max} dem momentan größten entwickelten t und α dem Entwicklungsschritt, der bei $\alpha = 1$ startet und sich dann in jedem Schritt verdoppelt.

Um uns deutlich zu machen, wie dies alles mit einer Gram-Matrix berechnet werden kann und wie diese Matrix aussieht, betrachten wir nun den Pseudocode. Wir bekommen wieder die Menge von Auswertungen (T, Y, Y'), welche die Kandidaten und zugehörigen Werte darstellt und erhalten daraus die Mittelwertsfunktion $\mu(t)$ und die Kovarianzfunktion $\tilde{k}(t, T)$. Auch die Ableitungen dieser Funktionen werden im Gauß-Prozess

berechnet. Da diese jedoch analog berechnet werden, lassen wir sie der Übersichtlichkeit halber weg.

Algorithmus 4.1 Gauß-Prozess

Eingabe: Kandidaten $T = (t_1, \cdots, t_N)$, Anzahl der Kandidaten N, Funktionswerte $Y = (f(x_1), \ldots, f(x_N))$, Ableitungswerte $Y' = (f'(x_1), \ldots, f'(x_N))$

Ausgabe: $\mu(t)$, $\tilde{k}(t, T)$

1: $G = \begin{bmatrix} k(T,T) & k^{\partial}(T,T) \\ {}^{\partial}k(T,T) & {}^{\partial}k^{\partial}(T,T) \end{bmatrix}$
2: $Sig = [\sigma_f, \ldots, \sigma_f, \sigma_{f'}, \ldots, \sigma_{f'}]$ mit N-mal σ_f, $2N$-mal $\sigma_{f'}$
3: $G = G + \text{diag}(Sig)$
4: $b = [Y, Y']^{\mathsf{T}}$
5: Löse Lineares Gleichungssystem $GA = b$ nach A
6: $\mu(t) = [k(t, T'), k^{\partial}(t, T')] \cdot A$
7: Löse Lineares Gleichungssystem $GB = [k(t, T'), k^{\partial}(t, T')]^{\mathsf{T}}$ nach B
8: $\tilde{k}(t, T) = k(t, t) - ([k(t, T'), k^{\partial}(t, T')] \cdot B)$

Der Ausdruck $k(T, T)$ definiert dabei in Schritt 1 die Blockmatrix mit den Einträgen $k_{ij} = k(T[i], T[j])$. Für die Bezeichnungen $k(t, T)$ gilt dies einseitig, das bedeutet, es wird ein Vektor mit den Einträgen $k_i = k(t, T[i])$ berechnet. Analoges gilt für die jeweiligen Ableitungen.

Somit haben wir diskrete Kandidatenpunkte für die nächste Entwicklungsstufe mithilfe der Funktionen $\mu(t)$ konstruiert. Für weitere Informationen zum Gauß-Prozess siehe [12]. Nun müssen wir unter ihnen nur noch den geeignetsten Punkt finden. Dafür definieren wir eine Nutzenfunktion, die maximiert werden soll.

4.2 Erwartete Verbesserung

Um zu entscheiden, für welchen Punkt f und ∇f angewendet werden sollen, also welcher Kandidat ausgewählt wird, benutzen wir ein Werkzeug der Bayes-Optimierung. *Expected Improvement* ist die erwartete Menge unter unserem Gauß-Prozess, an der $f(t)$ kleiner ist als ein momentan bester Wert $\eta = \min_{i=0,\ldots,N}\{\mu(t_i)\}$. Die erwartete Verbesserung ist definiert als

$$\begin{aligned} u_{EI}(t) &= \mathbb{E}_{p(f(t)|Y,Y')}[\min\{0, \eta - f(t)\}] \\ &= \frac{\eta - \mu(t)}{2}\left(1 + \text{erf}\left(\frac{\eta - \mu(t)}{\sqrt{2\mathbb{V}(t)}}\right)\right) + \sqrt{\frac{\mathbb{V}(t)}{2\pi}}\exp\left(-\frac{(\eta - \mu(t))^2}{2\mathbb{V}(t)}\right). \end{aligned}$$

Diese Funktion ist die Nutzenfunktion, das heißt, sie gibt uns an, welchen Nutzen die einzelnen Punkte auf unseren Algorithmus haben. Der nächste Entwicklungspunkt wird also aus den Kandidaten gewählt, indem geprüft wird, welcher Punkt diese Formel maximiert. Dieser Punkt wird dann mit der Wahrscheinlichkeit multipliziert, dass die

Wolfe-Bedingungen erfüllt sind. Unsere Wolfe-Bedingungen sind aber bis jetzt nur in der eindimensionalen, ungestörten Variante gegeben, deswegen schauen wir uns nun an, wie sie für unser Verfahren genutzt werden können.

4.3 Probabilistische Wolfe-Terminierungsbedingungen

Wir wollen jetzt die in Kapitel 2.3 vorgestellten Wolfe-Bedingungen in eine probabilistische Form bringen. Wir definieren dafür die zwei Variablen a_t und b_t, welche beide lineare Projektionen von f und ∇f sind, indem wir c_1 und c_2 aus den deterministischen Wolfe-Bedingungen benutzen. Dabei soll gelten:

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(x_k) \\ f'(x_k) \\ f(x_{k+1}) \\ f'(x_{k+1}) \end{bmatrix} \geq 0.$$

Wir bemerken als Erstes, dass es positive Einschränkungen an unsere Variablen gibt, und als Zweites, dass die Wolfe-Bedingungen erfüllt sind, wenn $a_t, b_t > 0$ gilt. Dies wird genutzt, um eine Kurzschreibweise für die Bedingungen aufzustellen. Der Gauß-Prozess (4.2) auf f impliziert eine bivariate Normalverteilung, bei jedem Wert von t, wie folgt:

$$\begin{aligned} p(a_t, b_t) &\sim \mathcal{N}_2\left(\begin{bmatrix} m_t^a \\ m_t^b \end{bmatrix}, \begin{bmatrix} C_t^{aa} & C_t^{ab} \\ C_t^{ba} & C_t^{bb} \end{bmatrix}\right), \\ \text{mit } m_t^a &= \mu(0) - \mu(t) + c_1 t\mu'(0) \text{ und } m_t^b = \mu'(t) - c_2\mu'(0), \\ C_t^{aa} &= \tilde{k}_{00} + (c_1 t)^2\ {}^{\partial}\tilde{k}_{00}^{\partial} + \tilde{k}_{tt} + 2[c_1 t(\tilde{k}_{00}^{\partial} - {}^{\partial}\tilde{k}_{0t}) - \tilde{k}_{0t}], \\ C_t^{bb} &= c_2^2\ {}^{\partial}\tilde{k}_{00}^{\partial} - 2c_2\ {}^{\partial}\tilde{k}_{0t}^{\partial} + {}^{\partial}\tilde{k}_{tt}^{\partial} \text{ und} \\ C_t^{ab} = C_t^{ba} &= -c_2(\tilde{k}_{00}^{\partial} + c_1 t\,{}^{\partial}\tilde{k}_{00}^{\partial}) + (1 + c_2)^{\partial}\tilde{k}_{0t} + c_1 t\,{}^{\partial}\tilde{k}_{0t}^{\partial} - \tilde{k}_{tt}^{\partial}. \end{aligned}$$

Die Wahrscheinlichkeit, dass die Wolfe-Bedingungen für den Kandidaten t erfüllt sind, $p_t^{Wolfe} = p(a_t > 0 \wedge b_t > 0)$, kann als Integral über eine bivariate Normalverteilung dargestellt werden:

$$p_t^{Wolfe} = \int_{-\frac{m_t^a}{\sqrt{C_t^{aa}}}}^{\infty} \int_{-\frac{m_t^b}{\sqrt{C_t^{bb}}}}^{\infty} \mathcal{N}_2\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho_t \\ \rho_t & 1 \end{bmatrix}\right) da\ db \tag{4.5}$$

Dabei wird der Korrelationskoeffizient $\rho_t = C_t^{ab}/\sqrt{C_t^{aa} C_t^{bb}}$ verwendet.
Die Liniensuche berechnet diese Wahrscheinlichkeit für alle Kandidatenpunkte nach jeder Auswertung. Wenn einer die Wolfe-Bedingungen mit $p_t^{Wolfe} > c_w$ erfüllt, wobei $0 \leq c_w \leq 1$, dann wird er akzeptiert und zurückgegeben. Wenn mehrere Punkte die Ungleichung erfüllen, wird derjenige von ihnen mit dem kleinsten $\mu(t)$ zurückgegeben.

Wenn kein Kandidat die Bedingung erfüllt, wird der beste von ihnen ausgewählt, der, bei dem μ minimal ist. In Kapitel 6 werden wir sehen, wie c_w gesetzt werden soll.
In der Programmierung kann die Formel für p_t^{Wolfe} effizient ausgerechnet werden, indem ein schon benutzbarer Code [13] für die bivariate Normalverteilung verwendet wird, welcher auf [14] beruht. Dieser musste nur noch in Python übersetzt werden. Dabei braucht ein Computer für jeden Aufruf etwa 100 Mikrosekunden und eine Liniensuche benötigt weniger als 50 Aufrufe, weswegen sich die Laufzeit nicht signifikant verschlechtert.

Betrachten wir nun die starken Wolfe-Bedingungen, da diese meist bei deterministischen Optimierern benutzt werden. Eine praktische probabilistische Auswertung dieser Bedingung ist numerisch sehr umständlich, da die Verteilung von $|\nabla f|$ eine nichtzentrale χ-Verteilung ist, welche individualisierte Berechnungen verlangt. Sie ist eine Verallgemeinerung der χ-Verteilung, wobei der Erwartungswert keine Rolle spielt. Dabei wird eine Verteilung so standardisiert wie im Zentralen Grenzwertsatz, damit eine nicht-zentrale χ-Verteilung herauskommt. Dies sieht wie folgt aus: Seien X_i unabhängig normalverteilte Zufallsvariablen mit Mittelwert μ_i und Varianz σ_i^2 für $i \in \{1, \cdots, k\}$. Dann ist

$$Z = \sqrt{\sum_{i=1}^{k} \left(\frac{X_i}{\sigma_i}\right)^2}$$

nicht-zentral χ-verteilt mit $k > 0$ dem Grad der Freiheit.
Gebräuchlicher ist jedoch die speziellere χ^2-Verteilung, die besagt, dass für unabhängige standardnormalverteilte Zufallsvariablen X_i mit $i \in \{1, \cdots, k\}$ die Summe $Z = \sum_{i=1}^{k} X_i^2$ eine χ^2-Verteilung besitzt [9].

Eine Art, die starken Wolfe-Bedingungen probabilistisch zu gestalten, ist, große positive Gradienten nicht zu akzeptieren: Angenommen es gilt $\nabla f(x_k) < 0$, wobei dies komponentenweise gemeint ist, dann kann die starke Krümmungsbedingung geschrieben werden als

$$0 \leq b_t := \nabla f(x_{k+1})^\mathsf{T} s_k - c_2 f(x_k) \leq -2c_2 \nabla f(x_k)^\mathsf{T} s_k.$$

Dabei ist der Wert $-2c_2 \nabla f(x_k)^\mathsf{T} s_k$ zu 95 Prozent beschränkt durch

$$-2c_2 \nabla f(x_k) \lesssim -2c_2 \left(|\mu'(0)| + 2\sqrt{\mathbb{V}'(0)} \right) =: \bar{b}.$$

Nun kann eine Approximation an die starken Wolfe-Bedingungen erreicht werden, indem die unendliche obere Integrationsschranke in (4.5) durch $(\bar{b} - m_t^b)/\sqrt{C_t^{bb}}$ ersetzt wird. Somit werden jetzt nur noch weniger und auch bessere Punkte die Wolfe-Bedingungen erfüllen. Diese Veränderung bringt uns außerdem keine neuen Kosten.

Um zu verdeutlichen, wie die Bedingungen als Algorithmus berechnet werden, betrachten wir den Pseudocode. In Schritt 1 werden die Komponenten, wie oben beschrieben, berechnet. Danach wird geprüft, ob sich unsere Werte berechnen lassen, und wenn nicht, wird die Wahrscheinlichkeit anders ermittelt. Wenn die Komponenten C_t^{aa} und C_t^{bb} groß genug sind, wird der Korrelationskoeffizient berechnet und die obere Integrationsschranke für die starke Krümmungsbedingung gesetzt. Mithilfe der Funktion, die in [14] beschrieben ist, berechnet Schritt 7 die bivariate normalverteilte Wahrscheinlichkeit.

Algorithmus 4.2 Probabilistische Wolfe-Bedingungen

Eingabe: Mittelwertsfunktion $\mu(t)$, Kovarianzfunktion $\tilde{k}(t,t')$ und deren Ableitungen, Wolfe-Parameter c_1, und c_2
Ausgabe: Wahrscheinlichkeit p_t^{Wolfe}

1: Berechne m_t^a, m_t^b, C_t^{aa}, C_t^{bb} und C_t^{ab}
2: **if** C_t^{aa} und C_t^{bb} sehr klein **then**
3: Gib $\mathbb{1}_{m_t^a \geq 0} \mathbb{1}_{m_t^b \geq 0}$ zurück
4: **else**
5: $\rho = C_t^{ab} / \sqrt{C_t^{aa} C_t^{bb}}$
6: $upper = \left(2c_2 \left(|\mu'(0)| + 2\sqrt{\mathbb{V}'(0)}\right) - m_t^b\right) / \sqrt{C_t^{bb}}$
7: Berechne die Wahrscheinlichkeit, dass für Werte x, y mit dem Korrelationskoeffizienten ρ gilt $-m_t^a / \sqrt{C_t^{aa}} < x$ und $-m_t^b / \sqrt{C_t^{bb}} < y < upper$ und gib diese zurück

Nun haben wir alle Bauteile für unseren Algorithmus zusammen und können das Verfahren als Algorithmus aufstellen. Hierbei benutzen wir die einzelnen Elemente aus diesem Kapitel ohne die Eingaben und Ausgaben genau zu definieren, sie können in den Algorithmen 4.1 und 4.2 nachgeschaut werden.

Algorithmus 4.3 Probabilistische Liniensuche

Eingabe: Entwicklungspunkt x_k, Funktionswert $f(x_k)$, Ableitungswert $\nabla f(x_k)$, Maximale Anzahl an Kandidatenpunkten N_{max}, Wolfe-Parameter c_1, c_2 und c_w
Ausgabe: Schrittweite t für den nächsten Schritt

1: $T = [0]$, $N = 1$, $Y = [f(x_0)]$, $Y' = [f'(x_0)]$ und $t = 1$
2: Gauß-Prozess für T ausführen
3: **while** $N < N_{max}$ **do**
4: Kandidaten t an T, Y und Y' anhängen, $N = N + 1$
5: Gauß-Prozess für T ausführen
6: **if** $p_t^{Wolfe} > c_w$ **then**
7: Gib den Kandidaten t zurück
8: **else**
9: Finde neuen minimalen Kandidaten t mithilfe von $\mu(t)$, $\tilde{k}(t,T)$, deren Ableitungen und der Nutzenfunktion $u_{EI}(t)$
10: $N = N + 1$
11: Gauß-Prozess für T ausführen
12: Besten Kandidaten $t_* \in T$ mithilfe der Nutzenfunktion finden und zurückgeben

5 Maschinelles Lernen

Die vorherigen Verfahrensgrundlagen sind für eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ definiert. Wir möchten gerne betrachten, was passiert, wenn stattdessen ein Neuronales Netzwerk, also eine Menge von Datenpunkten benutzt wird. Dies ist ein Anwendungsbeispiel für unser Verfahren, da in der Praxis meist keine Funktion gegeben sind, sondern nur zusammenhängende Daten. Gesucht wird dann eine Funktion, die unsere Menge approximieren kann und somit den Abstand der Menge zur Funktion minimiert. Dies wird zum Beispiel bei automatisierten Diagnosen oder bei der Texterkennung verwendet. Das Thema des maschinellen Lernens ist eng mit dem Bereich des *Data-Mining* verwandt, bei dem jedoch das Finden von Strukturen im Vordergrund steht. Oft geht es dabei um Kosten- und Belohnungsoptimierung, wie bei uns mit der Nutzenfunktion.

5.1 Allgemeine Definition des maschinellen Lernens

Beim maschinellen Lernen oder *Machine Learning* geht es darum, dass der Rechner durch einen Algorithmus selbstständig aus Trainingsdaten lernt, wie eine bestimmte Aufgabe gelöst wird, und sie daraufhin immer wieder lösen kann. Durch die Trainingsdaten erkennt der Algorithmus ein Muster und kann dieses Muster auf die folgenden Dateien anwenden. Maschinelles Lernen hängt stark mit Statistik und Wahrscheinlichkeitstheorie zusammen, da der Algorithmus aus den Statistiken der Trainingsdaten die Gesetzmäßigkeiten erkennt und dann berechnen kann, wie hoch die Wahrscheinlichkeit ist, dass die neuen Dateien den Gesetzmäßigkeiten entsprechen.

Es gibt drei verschiedene Typen des maschinellen Lernens [15]:

1. Überwachtes Lernen: Der Algorithmus bekommt Paare von Ein- und Ausgabewerten gegeben, und ihm werden korrekte Funktionswerte während des Lernens übermittelt. Das Ziel ist es, Assoziationen herzustellen. Als Anwendungsgebiete gelten zum Beispiel Handschrifterkennung oder Spam-Filter.

2. Unüberwachtes Lernen: Der Algorithmus erzeugt für eine Eingabe ein Muster, welches das Einfügen neuer Daten ermöglicht. Ein Anwendungsgebiet ist das Finden von Strukturen in einer Datenmenge.

3. Bestärkendes Lernen: Der Algorithmus arbeitet mit einer dynamischen Umgebung und einem Ziel, welches er erfüllen soll. Daraufhin gibt es Belohnungen, die auch negativ ausfallen können, anhand derer eine Nutzenfunktion aufgestellt wird. Der Algorithmus wird versuchen die Nutzenfunktion zu maximieren. Ein Anwendungsgebiet ist das Erlernen eines Spiels durch künstliche Intelligenz. Als Beispiel für diese Lernart gilt der Mensch.

Außerdem wird unterschieden zwischen Batch-Lernen, was bedeutet, dass der Algorithmus alle Daten gleichzeitig bekommt, und kontinuierlichem Lernen, bei dem die Daten sich nacheinander entwickeln [16].

Wir sehen also, dass unser Verfahren für Datenpunkte ein maschineller Lernalgorithmus mit bestärkendem, kontinuierlichem Lernen sein wird, da wir eine Nutzenfunktion $u_{EI}(t)$ haben und unsere Daten aufeinander folgend im Algorithmus berechnet werden. Die Kandidatenpunkte werden nacheinander erstellt und nach weniger als 10 Kandidatenpunkten wird derjenige mit dem höchsten Nutzen ausgewählt.
Der Gauß-Prozess ist jedoch an sich ein überwachtes maschinelles Lernverfahren, welches für Regressionen und Klassifikationen benutzt wird. Die Regressionen und die Klassifikation sind Verfahren, wobei die Regression die Ausgabedaten als reellwertige Zahlenwerte ausgibt und die Klassifikation als Klassencodes, welche die Ausgabewerte in definierte Klassen einteilt [17]. Durch die Gauß-Prozess-Regression kann aus den Trainingsdateien der Erwartungswert und die Varianzen berechnet werden und somit eine Schätzung der Werte an den Stellen und für den Fehler bestimmt werden.

5.2 Aufbau des Problems für eine Datenmenge

Gehen wir nun davon aus, dass statt einer Funktion eine Menge von Datenpunkten bzw. ein Neuronales Netzwerk gegeben ist. Es soll also eine Verlustfunktion $\mathcal{L}(x) : \mathbb{R}^n \to \mathbb{R}$ mithilfe seiner Ableitung $\nabla\mathcal{L}(x) : \mathbb{R}^n \to \mathbb{R}^n$ und den Prozessen aus dem vorherigen Kapitel optimiert werden. Wie schon in der Einleitung angesprochen, kann es dabei vorkommen, dass die Gradienten gestört werden, da nur über eine Teilmenge $\{d_j\}_{j=1\dots m}$ von Datenpunkten statt über die Gesamtmenge $\{d_i\}_{i=1\dots M}$ entwickelt wird. Dies spart Rechenkosten und der Algorithmus kommt trotzdem sehr nahe an das exakte Ergebnis. Die Verlustfunktion ist dabei definiert als:

$$\mathcal{L}(x) \coloneqq \frac{1}{M}\sum_{i=1}^{M} \ell(x, d_i) \approx \frac{1}{m}\sum_{j=1}^{m} \ell(x, d_j) =: \hat{\mathcal{L}}(x) \quad \text{mit } m \ll M. \tag{5.1}$$

Dabei ist $\ell(\hat{y}, y)$, wie in [2] definiert, die Verlustfunktion, welche die Kosten angibt, wenn statt dem wahren Wert y ein falscher Wert $\hat{y}$ vorausgesagt wird. Die Gesamt-Verlustfunktion $\mathcal{L}(x)$ gibt also an, wie groß der durchschnittliche Verlust ist, wenn wir an unserer Menge $\{d_i\}$ eine Funktion approximieren wollen.
Wenn die Indizes j dabei durch den Zentralen Grenzwertsatz unabhängig identisch in $[1, M]$ verteilt sind, ist der Fehler $\hat{\mathcal{L}}(x) - \mathcal{L}(x)$ unvoreingenommen und ungefähr normalverteilt. Die Indizes sind dabei unabhängig identisch verteilt, wenn alle die selbe Verteilung besitzen, das heißt, dass sie mit gleicher Wahrscheinlichkeit gleiche Werte annehmen, sich jedoch nicht beeinflussen.

In unserem Optimierungsproblem wollen wir also $y(t) = \hat{\mathcal{L}}(x(t))$ minimieren. Die Liniensuche arbeitet entlang einem mehrdimensionalen Gebiet $x(t) = x_i + ts_i,\ t \in \mathbb{R}$, dabei ist s_i die Richtung, entlang derer gearbeitet wird, und t der Skalar, der entscheidet, wie weit gegangen wird. Entlang dieser Richtung sammelt die Liniensuche Funktionswerte $f(t) = \mathcal{L}(x(t))$ und entwirft Gradienten $f'(t) = s_i^\mathsf{T} \nabla\mathcal{L}(x(t)) \in \mathbb{R}$. Da das Gradientenverfahren betrachtet wird, gilt wieder $s_i = -\nabla\mathcal{L}(x_i)$, dies muss jedoch nicht der Fall sein, da die probabilistische Liniensuche auch mit anderen Abstiegsverfahren kombiniert werden kann. Dabei akzeptiert unser Verfahren gestörte Funktionswerte $y_t = y(t)$ und gestörte Gradientenwerte $y'_t = y'(t)$ im Punkt t nur, wenn sie normalverteilt sind, also

$$p(y_t, y'_t|f) \sim \mathcal{N}_2\left(\begin{bmatrix} f(t) \\ f'(t) \end{bmatrix}, \begin{bmatrix} \sigma_f^2 & 0 \\ 0 & \sigma_{f'}^2 \end{bmatrix}\right). \tag{5.2}$$

Dabei meint $p(y_t, y'_t|f)$ das Wahrscheinlichkeitsmaß, welches von der bedingten Wahrscheinlichkeit ausgeht, dass y_t und y'_t bestimmte Werte annehmen, nachdem f angenommen wurde.
Wir können Formel (5.2) so interpretieren: Die Wahrscheinlichkeit, dass die gestörten Werte y_t und y'_t eintreffen, während die Funktion f betrachtet wird, ist bivariat normalverteilt mit dem Erwartungswertvektor $\mathbb{E}(X)$ und der Kovarianzmatrix Σ wie folgt:

$$\mathbb{E}(X) = \begin{bmatrix} f(t) \\ f'(t) \end{bmatrix} \text{ und } \Sigma = \begin{bmatrix} \sigma_f^2 & 0 \\ 0 & \sigma_{f'}^2 \end{bmatrix}.$$

Das bedeutet im Speziellen, dass der erwartete Funktionswert $f(t)$ und der erwartete Wert der Ableitung $f'(t)$ ist. Dies ist analytisch gesehen einleuchtend, da wir bei der Betrachtung einer Funktion an der Stelle t immer $f(t)$ erwarten würden, analog gilt dies auch für die Ableitung.
Weiterhin sind die Varianzen der einzelnen Komponenten σ_f^2, bzw. $\sigma_{f'}^2$, was bedeutet, dass diese beiden Werte die Geräusch- oder Störungspegel sind, da sie angeben, wie weit die Streuung der Werte um den Erwartungswert liegen. Die Kovarianzen zwischen den beiden Komponenten betragen 0, das heißt, dass es keinen Zusammenhang zwischen den Störungen von Funktion und Ableitung gibt. Dies ist analytisch gesehen nicht einleuchtend, weil eine Störung der Funktion eigentlich auch eine Störung der Ableitung verursachen sollte, dies wird aber in [1] nicht betrachtet. Wie die Schätzung der Varianzen aussehen sollte, wird in Kapitel 6 geschildert.

Betrachten wir nun unsere einzelnen Bauteile und passen das Verfahren an unsere neue Optimierungsaufgabe an. In Kapitel 4.1 wurde schon angesprochen, dass der Gauß-Prozess als überwachtes Maschinenlernverfahren zur Modellierung durch endlich viele diskrete Trainingsdaten verwendet werden kann. Deswegen bleibt der Gauß-Prozess wie

er war und es wird statt der alten Definition von f und ∇f einfach die neue Definition übernommen. Wir benutzen hierbei die Gauß-Prozess-Regression und die Regressionen auf f und ∇f beziehen sich an dieser Stelle auf die Trainingsdaten. Das Berechnen des Erwartungswerts und der Varianzen benötigen wir in dem Sinne, dass σ_f und $\sigma_{f'}$ mithilfe der Regression berechnet werde können und dann in die Formeln (4.3) und (4.4) eingesetzt werden. Wie diese Parameter berechnet werden, wird im nächsten Kapitel erläutert.
Auch das Verfahren der *Expected Improvement* bleibt in diesem Sinne gleich. Dieses dient uns nun allerdings als Nutzenfunktion für das bestärkende Lernen des Algorithmus.
Die Wolfe-Bedingungen verändern sich jedoch in

$$\begin{aligned} &\text{(W-I)} && f(t) \leq f(0) + c_1 t f'(0), \\ &\text{(W-II)} && f'(t) \geq c_2 f'(0), \text{ und in der starken Form in} \\ &\text{(W-IIa)} && |f'(t)| \geq |c_2 f'(0)|. \end{aligned}$$

Wenn die neuen Definitionen allerdings eingesetzt werden, zeigt sich, dass der einzige Unterschied darin besteht, dass α verschwunden ist und dafür das t auftaucht, weswegen sich in der probabilistischen Schreibweise der Wolfe-Bedingungen nichts verändert, da wir dort t bereits für die Kandidaten benutzen.
So ist es möglich, dass das Verfahren mit minimalen Änderungen und anderen Definitionen der Funktion und ihres Gradienten eine Menge von Datenpunkten approximiert.

6 Wahl der Parameter

Wir wollen nun alle Parameter des Verfahrens eliminieren, damit der Benutzer die Liniensuche benutzen kann, ohne Entscheidungen treffen zu müssen. Unser Algorithmus wird damit zu einem Black-Box-Algorithmus.
Im Moment gibt es noch die sechs freien Parameter c_1, c_2, c_w, θ, σ_f und $\sigma_{f'}$, welche gesetzt oder anderweitig ausgetauscht werden müssen. Die Parameter für die Wolfe-Bedingungen c_1, c_2 und c_w können durch die Entscheidung des Programmierers, nicht des Benutzers gesetzt werden. Dafür müssen sie jedoch so gut wie möglich gewählt werden, damit sie später nicht mehr verändert werden müssen und trotzdem den Algorithmus voranbringen. In Kapitel 2.3 wurde bereits betrachtet, was passiert, wenn die Parameter in die möglichen Extremwerte 0 und 1 verändert werden. Später werden wir uns in den numerischen Experimenten noch anschauen, was passiert, wenn die Wahl der Werte um kleinere Schritte abgewandelt werden.
Der Wert θ aus der Kovarianzfunktion (4.1) kann eliminiert werden, indem die Liniensuche standardisiert wird. Die Geräuschpegel σ_f und $\sigma_{f'}$ können mit einem minimalen Aufwand geschätzt und daraufhin auch standardisiert werden.
Das Ergebnis wird ein parameterfreier Algorithmus sein, der auch die problematische Schrittweite entfernt.

Die Wahl der Parameterwerte haben Maren Mahsereci und Philipp Hennig in [1] getroffen. Sie haben die Werte durch empirische Versuche ausgewählt. Wir werden uns in einigen Experimenten in Kapitel 7 anschauen, ob diese Entscheidungen auch gut getroffen sind.

6.1 Wolfe-Parameter

Unser probabilistischer Algorithmus erbt die Wolfe-Parameter c_1 und c_2 von seinem deterministischen Vorgänger des nicht gestörten Falls. Wir werden die Standardwerte für eine "gnädige" Liniensuche benutzen, welche $c_1 = 0.05$ und $c_2 = 0.8$ sind. Diese Werte akzeptieren die meisten Abstiegsrichtungen, halten jedoch auch unerwünschte Punkte fern.
Der Parameter c_w ist der Wert, über dem die Wahrscheinlichkeit, dass die Wolfe-Bedingungen erfüllt sind, liegen muss. Dieser kommt nur in der probabilistischen Liniensuche vor. Wenn keine gestörten Werte vorliegen, sind die Wolfe-Bedingungen entweder erfüllt oder nicht, also liegt die Wahrscheinlichkeit bei 0 oder 1. In diesem Fall könnten wir unseren Parameter c_w beliebig wählen. Wir betrachten deswegen nur den probabilistischen, gestörten Fall, um zu entscheiden, wie der Wert gewählt werden soll. Wir bemerken, dass p_t^{Wolfe} auch kleine Werte annimmt, wie zum Beispiel bei $a_{t_1} = b_{t_1} = 0$, wo $p_t^{Wolfe} \approx 0.2$ sein kann, oder $a_{t_1} = b_{t_1} = -\varepsilon$ für $\varepsilon \to 0$, wo $p_t^{Wolfe} \approx 0.4$ erreichen

könnte, wenn die Auswertungen schlecht gewählt werden. Also benötigen wir einen Wert, der die konkurrierenden Wünsche nach Präzision und wenigen Aufrufen ausgleicht. Dies passiert zum Beispiel bei der Wahl $c_w = 0.3$.
In den Experimenten, die in [1] ausgeführt wurden, hat sich herausgestellt, dass der Wert nur ungefähr in der Nähe von 0.3 liegen muss, da die meisten Wahrscheinlichkeiten von p_t^{Wolfe} entweder sehr weit über oder sehr weit unter diesem Wert lagen. Diese Experimente und auch andere, werden wir in Kapitel 7 betrachten.
Somit haben wir die Wolfe-Parameter "gnädig" gesetzt, was heißt, dass die meisten Schrittweiten α_k angenommen werden. Damit muss der Benutzer keine Zeit mehr damit aufwenden, Werte für sie zu setzen.

6.2 Maßstab θ

Der Maßstab θ kommt in der Kovarianzfunktion (4.1) vor und zieht sich dann durch alle Ableitungen durch, da er als Skalarmultiplikator in der Funktion steht. Er stellt die angenommene Varianz dar. Wir wollen ihn eliminieren, indem wir das Optimierungsobjekt skalieren, dafür wird als Erstes $\theta = 1$ gesetzt. Nun müssen zwei Fälle unterschieden werden. Der erste Fall ist, dass eine Funktion optimiert werden soll, der zweite Fall ist, dass eine Datenmenge benutzt wird.
Im ersten Fall berechnen wir einen Hilfsfaktor $\beta = |\nabla f(x_k)^\mathsf{T} s_k|$, mit diesem können wir nun die alten Werte der Funktion und seines Gradienten auf die neuen Werte $\tilde{f}$ und $\nabla\tilde{f}$ setzen:

$$\tilde{f}(x_{k+1}) = \frac{f(x_{k+1}) - f(x_k)}{\alpha_k \beta} \text{ und } \nabla\tilde{f}(x_{k+1}) = \frac{\nabla f(x_{k+1})^\mathsf{T} s_k}{\beta}.$$

Nun gilt $\tilde{f} = 0$ und $\nabla\tilde{f} = -1$ wenn sich der Entwicklungspunkt von einem Schritt zum Nächsten nicht verändert.
Im zweiten Fall werden y_i und y_i' so verändert, dass sie besondere Eigenschaften erfüllen. Die Veränderungen sind fast genauso wie im ersten Fall. Wir ersetzen also y_i durch $(y_i - y_0)/|y_0'|$ und y_i' durch $y_i'/|y_0'|$. Dann erfüllen sie die Eigenschaften $y_0 = 0$ und $y_0' = -1$.
Mit diesen Wahlen ist dafür gesorgt, dass die Objektreichweite in der Standardreichweite von Liniensuchen $0 < t < 10$ liegt. Dies hilft uns bei der Berechnung der Schrittweite α, siehe Kapitel 6.4.
Da durch β bzw. den Betrag von y_0' geteilt wird, kommt es allerdings zu einer Nicht-Normalverteilung. Wie bei c_w wurde in den Experimenten in [1] jedoch herausgefunden, dass dies keinen bemerkbaren empirischen Effekt hat.

6.3 Geräuschpegel

Wir benötigen in unseren Formeln (4.3), (4.4) und (5.2) die zwei Parameter σ_f und $\sigma_{f'}$, die eine Standardverteilung für die Störungen von den Funktions- und Gradientenwerten darstellen. Diese Parameter sollen nicht nur gesetzt, sondern auch an das Problem angepasst werden, indem während der Liniensuche versucht wird, die Werte so gut wie möglich zu verändern.
Für den Fall, das eine Funktion betrachtet wird, werden σ_f und $\sigma_{f'}$ folgendermaßen gesetzt:

$$\sigma_f = \frac{\sqrt{Var(f(x_k))}}{\alpha_k} \text{ und } \sigma_{f'} = \sqrt{(s_k.^2)^\mathsf{T} Var(f(x_k))},$$

wobei mit $.^2$ das elementweise Quadrieren eines Vektors oder einer Matrix gemeint ist. Danach werden sie noch durch den Faktor β aus Kapitel 6.2 geteilt, damit sie skaliert sind.
Im Modell (5.1) kann die Varianz des Verlustes und seines Gradienten direkt innerhalb des Stapels von Datenpunkten mit einem kleinen Rechenaufwand geschätzt werden. Also werden für den Fall einer Datenmenge im Sinne des maschinellen Lernens die Parameter durch die empirische Statistik

$$\hat{S}(x) := \frac{1}{m}\sum_{j=1}^{m} \ell^2(x, y_j) \text{ und } \nabla\hat{S}(x) := \frac{1}{m}\sum_{j=1}^{m} \nabla\ell^2(x, y_j)$$

berechnet. Mit ihnen schätzen wir am Anfang der Liniensuche

$$\begin{aligned}\sigma_f^2 =& \frac{1}{m-1}\left(\hat{S}(x_k) - \hat{\mathcal{L}}(x_k)^2\right) \text{ und}\\ \sigma_{f'}^2 =& \left(s_i.^2\right)^\mathsf{T}\left[\frac{1}{m-1}\left(\nabla\hat{S}(x_k) - \left(\nabla\hat{\mathcal{L}}(x_k)\right).^2\right)\right].\end{aligned}$$

Nun soll die empirische Schätzung noch, wie bei θ, skaliert werden. Dafür wird σ_f durch $\sigma_f/|y'(0)|$ und σ_f' durch $\sigma_{f'}/|y'(0)|$ ersetzt. Dabei sind die Kosten dieser Auswertung klein, auch wenn die Berechnung von $\ell(x, y_i)$ selbst teurer ist als die Summation über j. Für diesen Ansatz wird jedoch ein Stapel mit einer Größe, die größer ist als 1, benötigt. Falls nur eine Einzelprobe vorhanden ist, kann stattdessen ein Durchschnitt benutzt werden.
Bei den Formeln für beide Fälle nehmen wir bewusst an, dass die Störung des Gradienten unabhängig von der Störung der Funktion ist, dies geht jedoch in der Praxis nicht immer, da eine Funktion und ihre Gradienten zusammenhängen. Wir wollen die Parameter trotzdem so wählen, denn dadurch, dass wir die Störungen für jede Dimension einzeln abschätzen, erfassen wir oft die inhomogene Struktur unter den Elementen des Gradienten und den Effekt auf die Störung entlang der projizierten Richtung.

6.4 Schrittweite

Nun kümmern wir uns noch um den wohl entscheidendsten Parameter, nämlich den, der uns schon im nicht probabilistischen Gradientenverfahren gestört hat, da wir ihn nicht exakt berechnen können: die Schrittweite α.
Falls sie zu groß initialisiert wird oder zu schnell wächst, kann das stochastische Gradientenverfahren instabil werden, da die individuellen Schritte nicht auf Robustheit überprüft werden. Unser Ziel war es in Kapitel 4 eine geeignete Schrittgröße zu finden, indem wir uns eine Liste von möglichen Auswertungspunkten geschrieben und dann mit dem Verfahren *Expected Improvement* den geeignetsten ausgewählt haben. Dieser musste dann auch noch die Wolfe-Bedingungen erfüllen, um angenommen zu werden. Falls es keinen Punkt gab, der die Bedingungen erfüllt, wurde der beste Kandidatenpunkt ausgewählt.
Wir wollen aber auch eine gute Schrittgröße finden, falls die Länge der Richtung s_i falsch skaliert ist. Diese Länge ist im stochastischen Gradientenverfahren proportional zu unserer Schrittgröße α_k. Solche Skalierprobleme bestehen normalerweise über die Zeit an, es wäre aber verschwenderisch, einen Algorithmus zu bauen, der in jeder Liniensuche den Maßstab anpasst. Stattdessen wollen wir die Schrittgröße im Algorithmus von einer Iteration in die Nächste fortpflanzen. Wieder muss zwischen den beiden Fällen, die es auch bei den anderen Parametern gab, unterschieden werden. Im ersten Fall wird $s_k = -\nabla f(x_k)$ gesetzt und am Anfang wird $\alpha_0 = 0.01$ initiiert, danach wird x_k entwickelt zu $x_{k+1} = x_k + \alpha_k t_* s_k$ und der neue Wert der Schrittweite wird $1.3 t_* \alpha_k$, wobei t_* der t-Wert des angenommenen Kandidaten ist. Dort hilft uns jetzt, dass $0 < t < 10$ gilt, weil α so nicht zu groß werden kann. Anders wäre es, wenn t größer als 10 sein könnte. Falls die neue Schrittweite jedoch trotzdem zu weit von den bisherigen Schrittweiten abweicht wird α auf den ungefähren Durchschnitt zurückgesetzt.
Beim zweiten Fall ist es fast genauso. Die initiierte Schrittweite wird wieder am Anfang auf $\alpha_0 = 0.01$ gesetzt und die initiierte Suchrichtung auf $s_0 = -\alpha_0 \nabla \hat{\mathcal{L}}(x_0)$. Danach wird jede Liniensuche bei $x_i = x_{i-1} + t_* s_i$ enden und die nächste Suchrichtung wird auf $s_{i+1} = -1.3 t_* \alpha_0 \nabla \hat{\mathcal{L}}(x_i)$ festgelegt.
Somit startet die nächste Liniensuche die Auswertung beim 1.3-fachen der Schrittgröße des Vorgänger.

7 Numerische Experimente

Wir wollen einige numerische Experimente am stochastischen Gradientenverfahren ausführen. Dafür wird der selbstgeschriebene Python-Code benutzt, der allerdings auf die gleiche Art arbeitet wie der vorgegebene Matlab-Code. Der einzige Unterschied ist, dass die beiden Programme unterschiedliche Zufallszahlgeneratoren benutzen, und dass es zu unterschiedlichen Rundungsfehlern kommen kann. Das Problem der Zufallszahlen wurde jedoch umgangen, indem der Generator des Matlab-Codes auf denselben wie in Python gesetzt wurde.

Die Experimente werden an vier verschiedenen Funktionen ausgeführt, drei von ihnen waren im Original-Code vorgegeben, eine weitere Funktion haben wir zu Versuchszwecken aufgestellt. Die drei vorhandenen Funktionen sind weitverbreitete Funktionen zum Testen von Optimierern: die Booth-Funktion $f_1(x) = (x_1+2x_2-7)^2+(x_2+2x_1-5)^2$, die Mccormick-Funktion $f_2(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$ und die Branin-Funktion $f_3(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos(x_1) + s$. Die Mccormick-Funktion ist dabei eine Funktion, die kein globales Minimum besitzt. Wie in Kapitel 2 bereits angesprochen, wird das Verfahren trotzdem versuchen, ein lokales Minimum zu finden, wenn dies näher zum Startwert liegt als der Abfall ins negative Unendliche. Bei der Branin-Funktion werden die typischen Werte für die Parameter $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $r = 6$, $s = 10$ und $t = 1/(8\pi)$ gesetzt. Für weitere Informationen zu diesen Funktionen siehe [18]. Die selbst aufgestellte Funktion ist $f_4(x) = 10x_1^2 - 2x_1 - 4x_1x_2 + x_2^2 + 1$, welche zu Vergleichszwecken dient.

Wir wollen nicht nur prüfen, ob die Wahl der Parameter gelungen ist, sondern auch, was mit dem Verfahren passiert, wenn wir einige Werte des Algorithmus verändern, die wir bis jetzt nicht betrachtet haben. Dafür werden wir uns als Erstes anschauen, was passiert, wenn sich der Startwert verändert. Dies kann negative Auswirkungen haben, da der Startwert eine der Verbesserungsmöglichkeiten zur Problembekämpfung war, die in der Einleitung genannt wurde.
Im Weiteren wird betrachtet, was die Störung auf die Funktion bewirkt, wie dies das Konvergenzverhalten beeinträchtigt und was passiert, wenn wir das Abbruchkriterium oder die maximale Anzahl von Kandidatenpunkten verändern. Dann wird getestet, ob die Parameter c_1, c_2 und c_w für die Wolfe-Bedingungen gut gesetzt sind und wie sich die Veränderung des initiierten Wertes von α_0 auswirkt. Dabei ist aufgefallen, dass Maren Mahsereci und Philipp Hennig in [1] zwar $c_2 = 0.8$ wählen, es im Original-Code aber auf $c_2 = 0.5$ gesetzt haben. Darauf sollte geachtet werden, falls die Auswertungen nachempfunden oder verglichen werden und es zu Unterschieden kommt. Da wir planen, diesen Wert zu verändern, stört es uns allerdings nicht.

Für jeden Versuch und jeden Wert, der geprüft werden soll, werden wir zehn Durchläufe starten und den Durchschnitt dieser Durchläufe betrachten. Dabei beeinflussen sich die Schritte, im Gegensatz zum Verfahren des maschinellen Lernens, nicht untereinander. In jedem Schritt werden alle Daten neu berechnet und das Verfahren wird von neu gestartet. Durch die Definition von Funktionsauswertungen kann daraus folgen, dass wir für mehrere Ausführungen mit den gleichen Eingabewerten unterschiedlich viele Entwicklungspunkte haben. Dies kommt daher, dass in den verschiedenen Ausführungen unterschiedlich viele Kandidaten erstellt werden müssen. Deswegen müssen wir bei der Berechnung der durchschnittlichen Punkte auf die minimale Anzahl von Ergebnissen kürzen. Aufgrund dieser Kürzung kann es allerdings zur Beeinträchtigung der Lösung kommen, wir betrachten also den Worst-Case der Anzahl von Entwicklungspunkten. Ein guter Ausgabepunkt ist hierbei ein Punkt, der eine kleine Distanz zum Minimum aufweist. Dabei muss aber auf das Kosten-Leistungs-Verhältnis geachtet werden.
Es wird von jedem Versuch und jeder Funktion ein Bild des Weges der Auswertungspunkte und ein Graph der Entwicklung der Funktionswerte ausgegeben. An ihnen wird dann überprüft, wie die Werte ausgewertet werden können und es kann gesehen werden, wie das Verfahren konvergiert.
Wir gehen von den Standardwerten aus und verändern jeweils einen Wert. Dabei sind die Standardwerte für die Startwerte der Funktionen f_1 bis f_3 vorgegeben und für f_4 wurde im Programm ein beliebiger, aber fester Punkt gesetzt. Als Standardstörung wird für die Funktion und ihre Ableitungen nach beiden Variablen jeweils 1 benutzt. Was dies genau bedeutet, wird in Kapitel 7.2 erläutert. Unser Abbruchkriterium wird eine festgelegte Anzahl von Funktionsauswertungen sein. Bei den Standardeinstellungen wird es maximal 50 Funktionsauswertungen und 7 Kandidatenpunkte geben. Diese Anzahl war in der Voreinstellung des vorgegebenen Codes enthalten. Die Parameter c_1, c_2, c_w und α_0 werden auf die Werte, die bereits in Kapitel 6 beschrieben wurden, gesetzt.

7.1 Startwertveränderung

Der Startwert x_0, welcher dem stochastischen Gradientenverfahren übergeben wird, hat einen großen Einfluss auf den Verlauf der Entwicklungspunkte. Falls er zu weit entfernt und deswegen zu nahe an einem lokalen Minimum oder am Abfall ins negative Unendliche liegt, wird immer eine schlechte Suchrichtung ausgewählt, da die Funktion zum falschen Wert abfällt. Nur mit viel Glück wird die Funktion oder das Verfahren derart gestört, dass sie zum globalen Minimum hin konvergiert.
Wir betrachten für dieses Experiment den angegebenen Startwert und neun weitere, die gut verteilt im Plot liegen. Dabei wird nicht bei jeder Funktion und jedem Startwert das Minimum erreicht. Dies wird in Abbildung 7.1 gezeigt, wobei der voreingestellte Startpunkt schwarz gekennzeichnet ist.

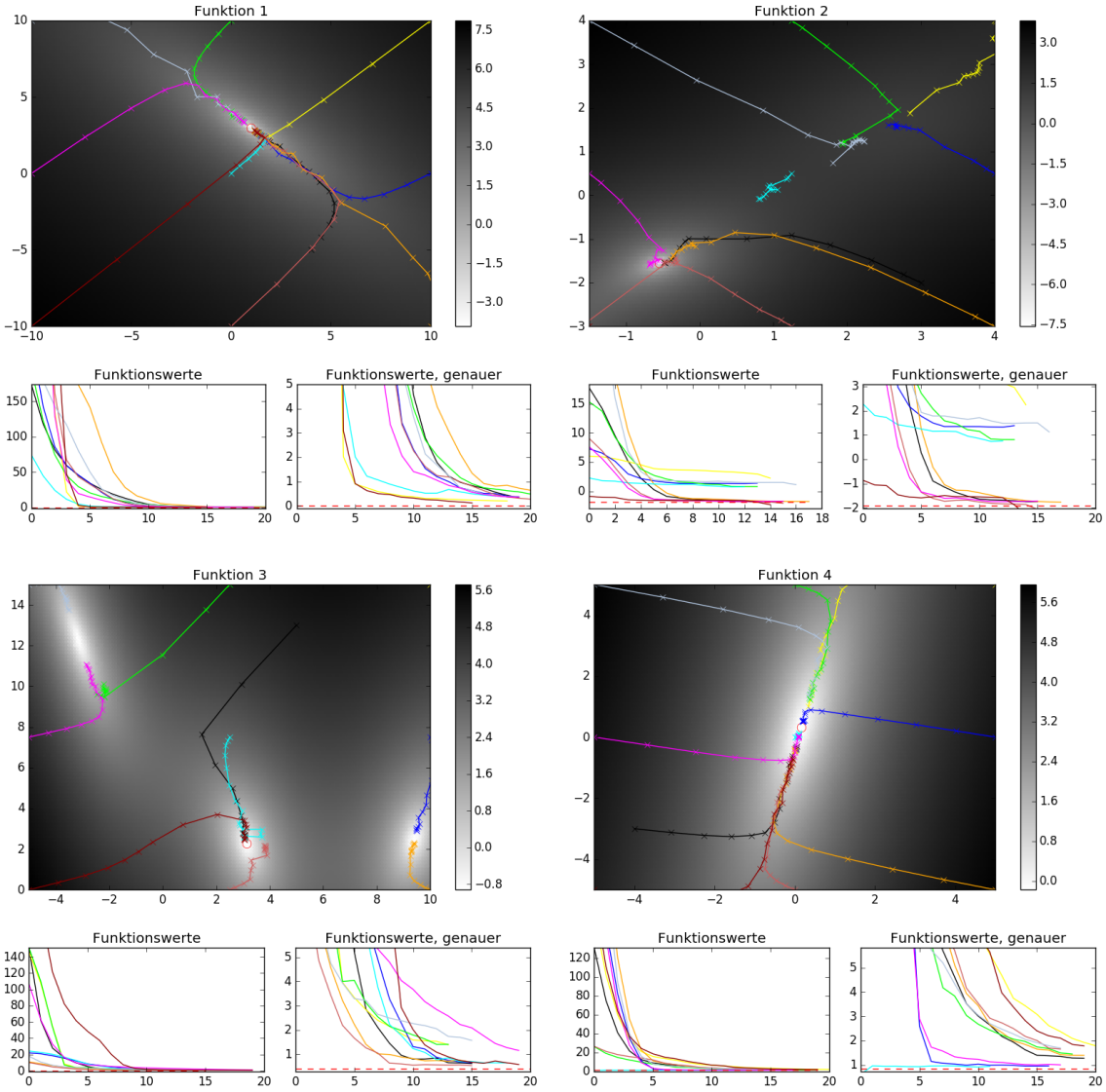

Abbildung 7.1: Auswirkung auf den Auswertungsweg und die Funktionswerte bei der Veränderung des Startwerts der Funktionen f_1 bis f_4 in neun weitere Punkte. Bei diesen sind $x_1 \in \{x_1^{min}, x_1^{max}, 0.5(x_1^{min} + x_1^{max})\}$ und $x_2 \in \{x_2^{min}, x_2^{max}, 0.5(x_2^{min} + x_2^{max})\}$, wobei x_1^{min}, x_1^{max}, x_2^{min} und x_2^{max} jeweils den Rand des Plots definieren. Die Standardeinstellung des Startwerts ist jeweils schwarz gekennzeichnet.

Wir sehen in dieser Abbildung, dass unser Verfahren immer eine Approximation des Minimums erreicht, wenn die Funktion nur ein Minimum in der Nähe besitzt, siehe Funktion f_1 und f_4. Bei Funktion f_2 stört ein Wert, der zwar größer ist als das Minimum, aber zu dem die Gradienten auch fallen. Außerdem gilt $\lim_{x_1,x_2 \to -\infty} f_2(x) = -\infty$, was die Wege der Entwicklungspunkte beeinträchtigt. Als Beispiel kann der Weg, welcher bei $(-3, -1.5)$ startet, genommen werden. Dieser unternimmt keinen Schritt in die Richtung des lokalen Minimums und ist deswegen im Bild nicht zu sehen. Bei f_3 ist das Problem,

dass es mehrere lokale Minima in der Nähe des globalen Minimums gibt. Startwerte, die zu nahe an diesen lokalen Minima liegen, finden nicht den Weg zum globalen Minimum. Betrachten wir nun die Funktionswerte, dann sehen wir, dass sie für die Funktionen f_1 und f_4 gegen das Minimum konvergieren. Dies haben wir bereits aus dem Plot des Weges geschlossen. Bei f_2 hingegen konvergieren die Funktionswerte für manche Startwerte gegen $-\infty$ und bei f_4 konvergieren sie gegen die drei verschiedenen Minima.

7.2 Störungsauswirkung

Dass eine Funktion ohne Störung auch mit dem normalen Gradientenverfahren gelöst werden kann, wurde schon angesprochen. Bei ihr kann der Parameter c_w beliebig gewählt werden. Nun betrachten wir, was passiert, wenn sich die Störung von 0 bis 5 verändert. Dabei bedeutet Störung um den Wert ε, dass die gestörten Werte definiert sind als

$$\hat{f}(x) = f(x) + \varepsilon \text{randn}() \text{ und } \nabla\hat{f}(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \end{pmatrix} + \varepsilon \begin{pmatrix} \text{randn}() \\ \text{randn}() \end{pmatrix},$$

mit randn() einer standardnormalverteilten Zufallszahl. Dies bedeutet, dass wir nicht die Störung selbst festlegen sondern ihre Größenordnung. Die Störung mit dem Wert 0 ist dabei unser Ausgangswert, der in Abbildung 7.2 in schwarz dargestellt ist.

Wenn wir kurz überlegen, was es bedeuten wird, wenn wir die Funktion stören, kommen wir zu einer relativ simplen Idee. Je mehr die Funktion gestört wird, desto schlechter werden die Ergebnisse. Dieser Gedankengang ist einleuchtend, aber die Frage bleibt offen, wie stark die Werte abweichen. Betrachten wir dafür die Abbildung 7.2.
Wir sehen, dass unser Standardwert von Störung 1 nahe an dem ungestörten Fall liegt und dass mit steigender Störung, wie schon angenommen, die Werte schlechter werden. Dabei gilt allerdings, dass für die Funktionen mit nur einem Minimum dieses fast immer erreicht wird. Hingegen wird für eine hohe Störung bei den Funktionen f_2 und f_3 nicht ansatzweise das Minimum approximiert.
Genau dieselben Aussagen liefern uns die Funktionswerte, obwohl wir bei ihnen sehen können, dass auch die Funktionen mit eindeutigem globalen Minimum schlecht approximiert werden. Wenn wir allerdings mehr Funktionsauswertungen zulassen, konvergieren die Funktionswerte, dies würden sie allerdings langsamer und mit größeren Abweichungen.

7.3 Abbruchkriterium

Betrachten wir nun unser Abbruchkriterium, also die Anzahl der Funktionsauswertungen. Der erste Gedanke, der uns kommt, ist, dass wenn wir weniger Funktionsauswertungen veranlassen, wir auch nicht so nahe an das Minimum herankommen. Wenn wir sehr

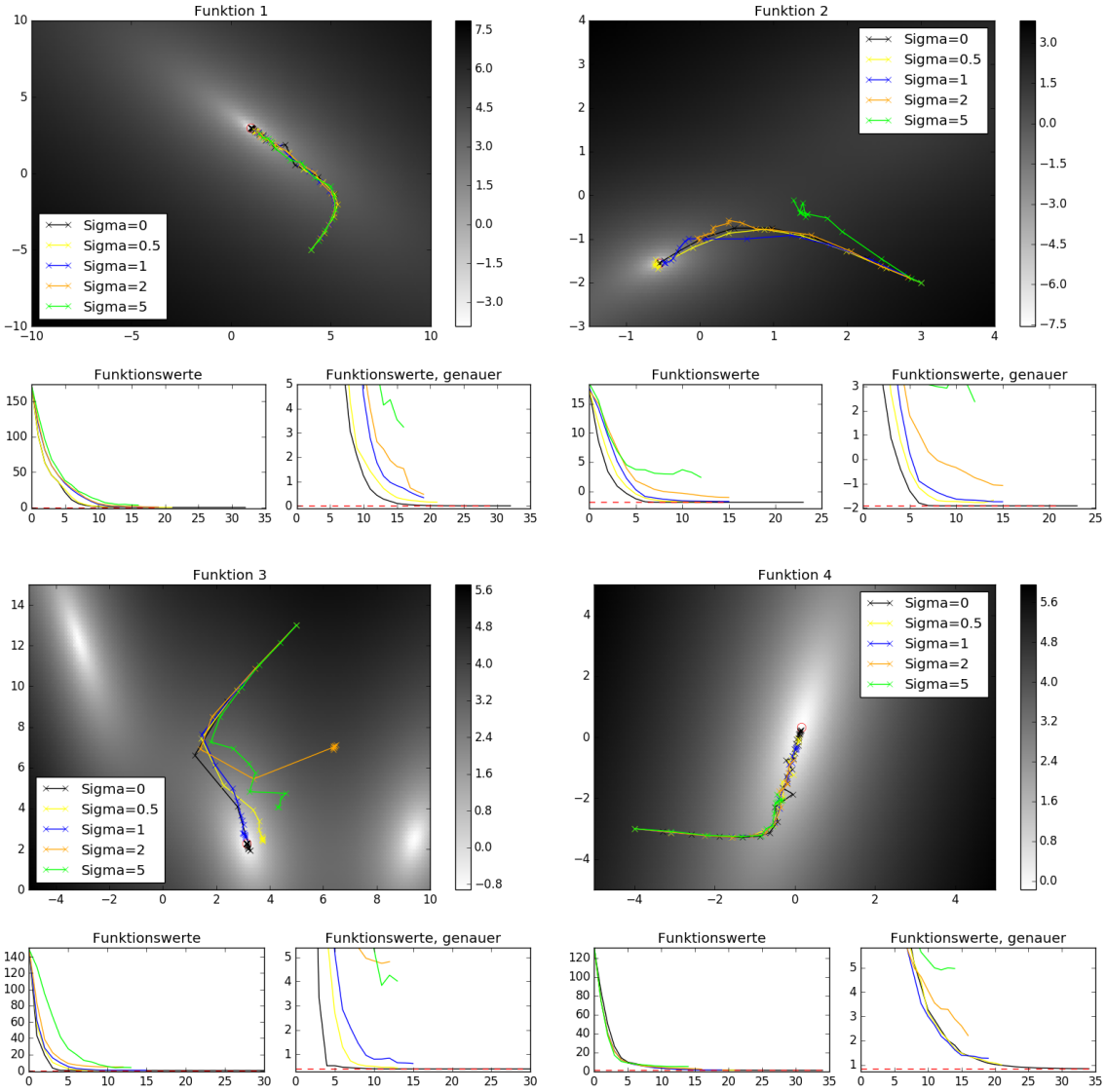

Abbildung 7.2: Auswirkung der Veränderung der Störungen bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Werte sind von 0 bis 5 gefächert. Der Fall ohne Störung ist dabei schwarz gekennzeichnet.

viele Auswertungen zulassen, müssten wir das Minimum finden und gut approximieren können, dabei muss aber darauf geachtet werden, ob es sich lohnt, mehr Auswertungen stattfinden zu lassen, da dies natürlich höhere Rechenkosten hervorbringt.
Eine Funktionsauswertung ist definiert als die Berechnung der einzelnen Prozesse für einen Punkt. Das bedeutet, dass für jeden Kandidatenpunkt eine Auswertung stattfindet. Deswegen kann es für einen Entwicklungspunkt zu maximal $N + 1$ Funktionsauswertungen kommen. Dabei ist N die maximale Anzahl von Kandidatenpunkten, welche im nächsten Unterkapitel verändert wird und nach der Schleife darf ein weiteres mal der Gauß-Prozess berechnet werden. Der Standardwert für N liegt bei 7, da er zwischen

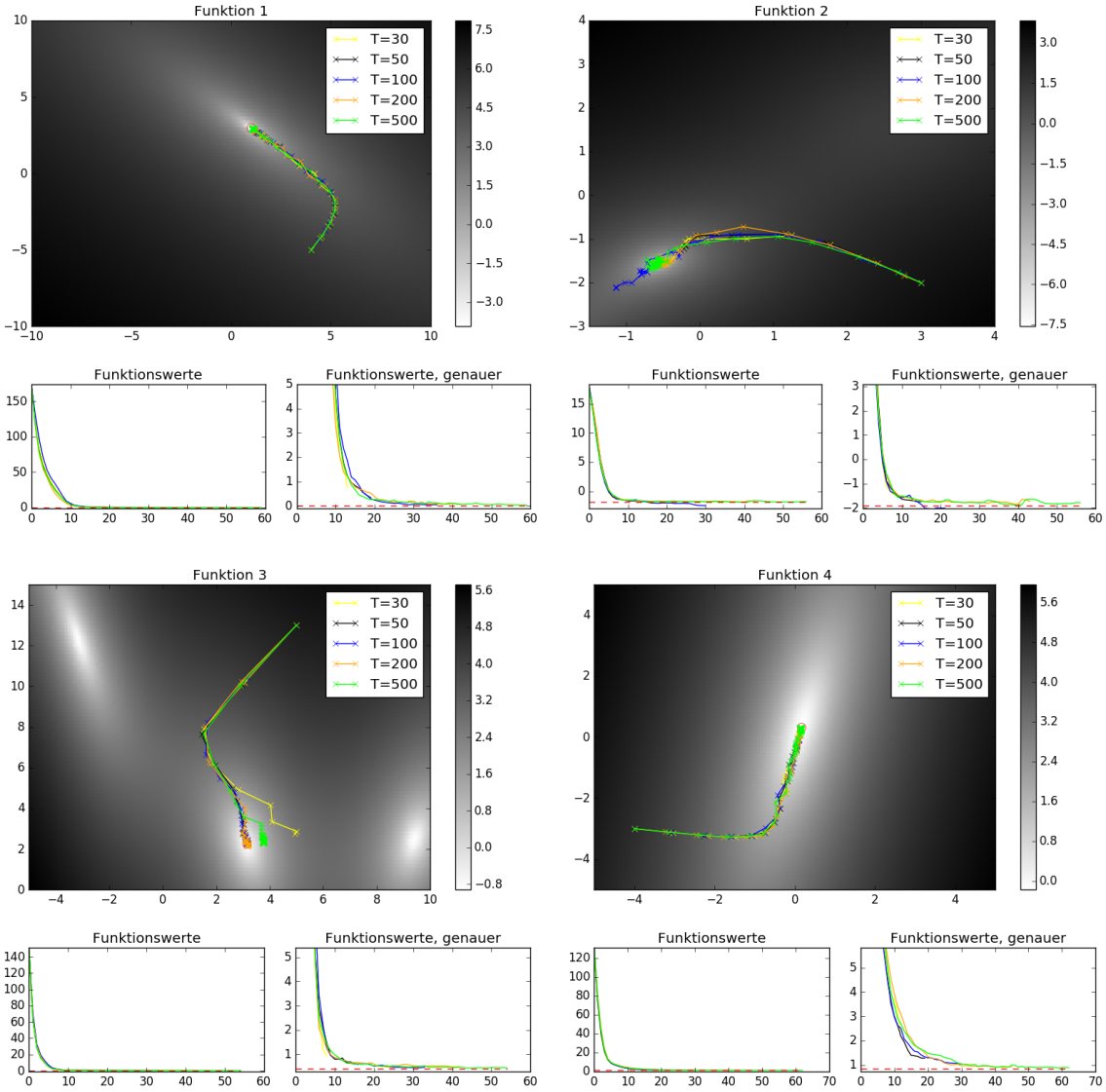

Abbildung 7.3: Auswirkung der Veränderung des Abbruchkriteriums bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Standardeinstellung $T = 50$ ist dabei schwarz gekennzeichnet.

0 und 10 liegen soll und es empirisch in [1] ausgewertet wurde, dass dieser Wert gut passt.

In Abbildung 7.3 sehen wir, dass für wenig Auswertungen nicht immer das Minimum erreicht wird. Allerdings verändert sich die Genauigkeit ab 100 Auswertungen nicht mehr allzu viel. Bei Funktion f_3 gibt es sogar eine Verschlechterung der Approximation, wenn zu viele Entwicklungspunkte berechnet werden. Außer bei Funktion f_2 ist $T = 100$ der beste Kompromiss von Genauigkeit und Kosten. Da die Werte für $T = 50$ jedoch bei jeder Funktion schnell konvergieren, ist dies eine gute Wahl für unsere Standardeinstellung.

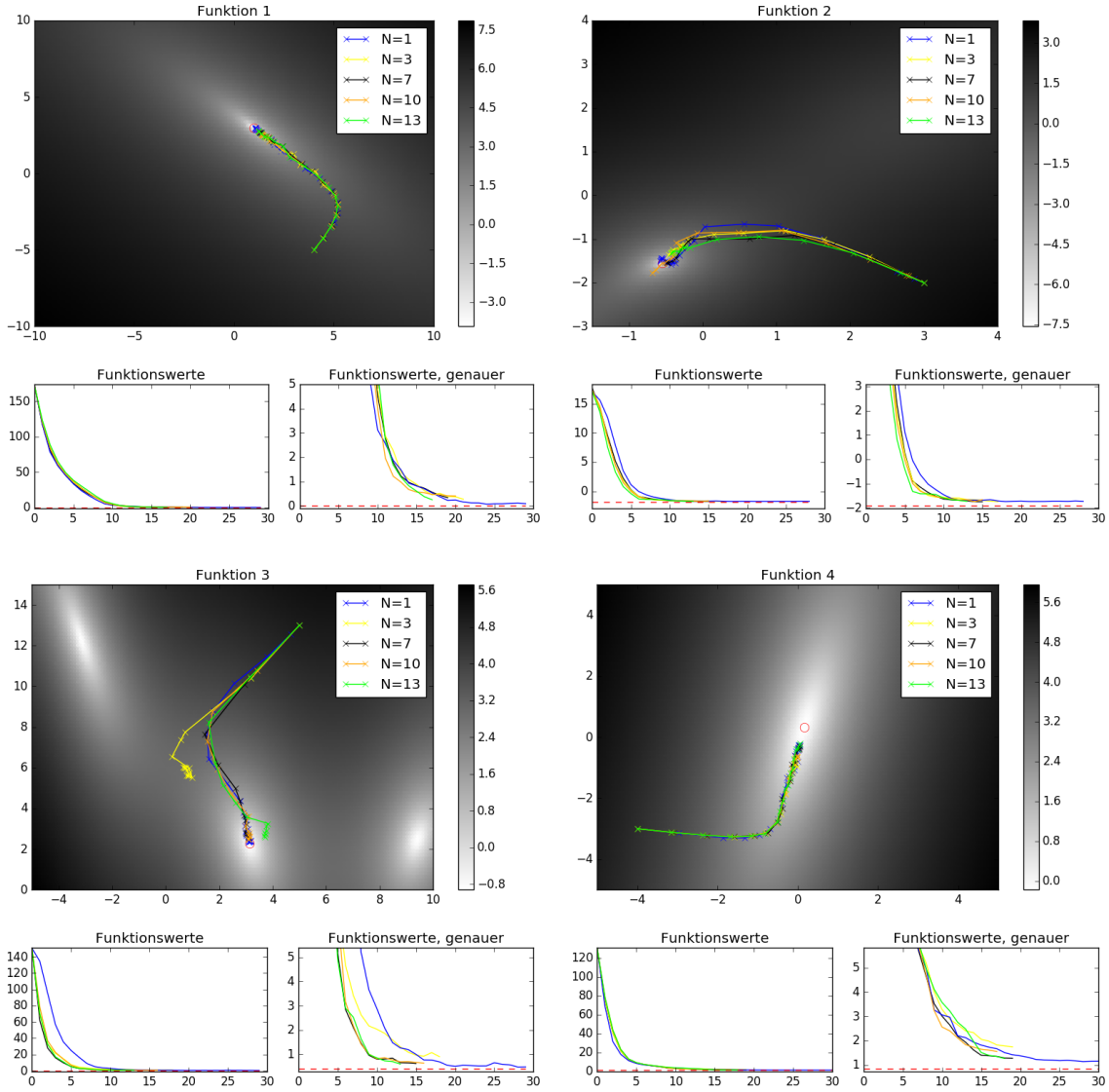

Abbildung 7.4: Auswirkung der Veränderung der Maximalen Anzahl von Kandidatenpunkten bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Standardeinstellung $N = 7$ ist dabei schwarz gekennzeichnet.

7.4 Anzahl der Kandidatenpunkte

Kommen wir nun zu der Anzahl der Kandidatenpunkte. Es ist einleuchtend, dass wir, wenn wir wenig Kandidaten erlauben, schlechte Entwicklungspunkte annehmen müssen und somit das Minimum nicht so schnell und gut approximieren können. Im Gegensatz dazu werden viele Entwicklungspunkte gute Punkte liefern, da aus einen größeren Menge ausgewählt werden kann. Hierbei muss natürlich wieder das Verhältnis von Genauigkeit und Rechenkosten betrachtet werden. Denn je mehr Kandidaten wir aufstellen, desto öfters müssen wir auch die Funktion auswerten. Da die Anzahl der Funktionsauswertun-

gen jedoch auf eine feste Zahl gesetzt wird, weil wir dies als Abbruchkriterium benutzen, kann es dazu führen, dass wir zwar gute, aber zu wenig Auswertungspunkte berechnen. Deswegen wird auch die Liste der Punkte kürzer, wenn N steigt. Dies sehen wir in Abbildung 7.4 an den Funktionswerten.

Betrachten wir nun die Ausgabeplots in der Abbildung 7.4. Wir sehen, dass es nicht so schlecht ist, wie wir gedacht haben, wenn wir nur einen Kandidatenpunkt benutzen. Dies kommt daher, dass wir mehr Punkte finden können, weil die Anzahl der verbleibenden Auswertungen pro Liniensuche nur um maximal zwei sinkt. Es sind zwei, da nach der Schleife für die Kandidaten noch ein weiteres mal der Gauß-Prozess aufgerufen werden kann, siehe Algorithmus 4.3. Das hat zur Folge, dass die Funktion langsamer zum Minimum hin konvergiert.
Die Fälle $N = 3$ und $N = 13$ sind eher schlecht gewählt, da sie für Funktion f_3 die Umgebung des Minimums nicht erreichen. Außerdem würde $N = 13$ nur wenige Auswertungspunkte zulassen. Im Endeffekt werden alle Verfahren, bis auf den einen Fall bei f_3, für die verschiedenen Einstellungen konvergieren. Dabei ist ein guter Kompromiss wieder die Standardeinstellung $N = 7$.

7.5 Veränderung der Wolfe-Parameter

Betrachten wir nun die Wolfe-Parameter. Sie wurden in [1] auf $c_1 = 0.05$, $c_2 = 0.8$ und $c_w = 0.3$ durch empirische Testauswertungen festgesetzt. Wir wollen versuchen, diese Experimente nachzustellen. Dabei werden wir jeweils die Werte 0, 0.5 und 1 ausprobieren und zusätzlich noch einen kleinen Schritt von 0.05 zu beiden Seiten des gesetzten Wert gehen, um zu schauen, in welche Richtung sich die Werte verbessern oder verschlechtern. In den Abbildung 7.5, 7.6 und 7.7 wird jeweils einer der Parameter verändert und der Standardwert ist schwarz dargestellt. Dies gibt uns nur eine grobe Ahnung davon, wie es sich auswirkt, da sich auch mehrere Parameter auf einmal verändert können. Für diesen Fall gibt es im Anhang eine Tabelle, die für f_1 alle möglichen Kombinationen betrachtet und den durchschnittlichen Abstand zum Minimum enthält. In ihr sind auch jeweils die besten, schlechtesten und gesetzten Werte für die Kombinationen gekennzeichnet.

Betrachten wir zuerst die Plots. Der erste Parameter c_1 stammte aus der Abstiegsbedingung (W-I), welche aussagt, dass ein angenommener Wert unter einer linearen Extrapolation der Steigung $c_1 \nabla f(x_k)$ liegen muss. Wir hatten festgehalten, dass die Wolfe-Bedingungen für $c_1 = 0$ alle Werte unter $f(x_k)$ annehmen und für $c_1 = 1$ alle Werte von konvexen Funktionen abgelehnt werden.
Wenn wir nun die Durchschnittspfade der Entwicklungspunkte für die Randwerte betrachten, sehen wir, dass $c_1 = 0$ relativ gute Auswertungspunkte annimmt, aber am Ende des Pfades werden Punkte ausgewählt, die unsere Entwicklung nicht verbessern.

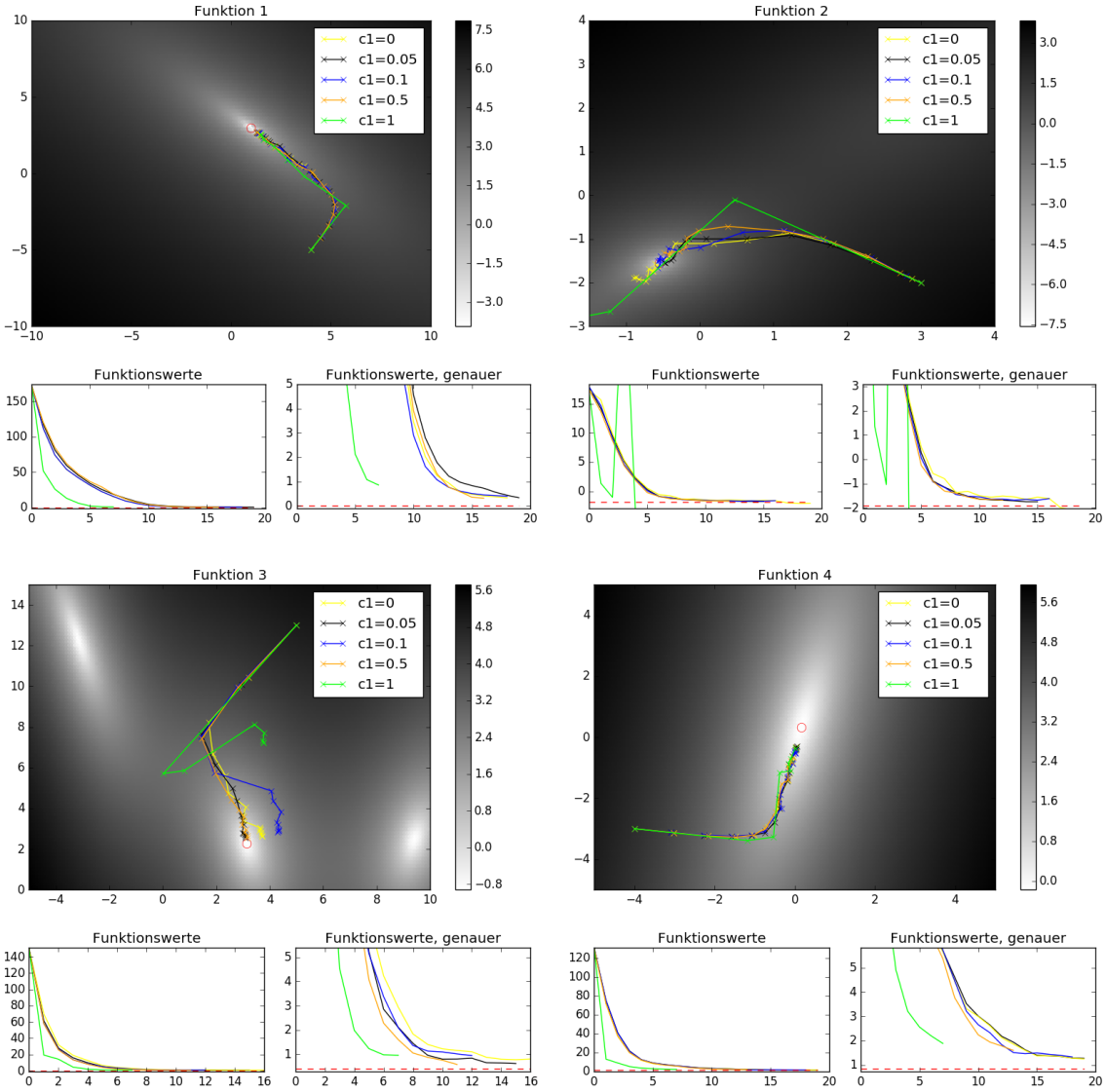

Abbildung 7.5: Auswirkung der Veränderung des Wolfe-Parameters c_1 bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Wahl $c_1 = 0.05$ ist dabei schwarz gekennzeichnet.

Deswegen konvergiert die Funktion zwar für diese Wahl des Parameters, aber nicht zum genauen Minimum hin.
Für den Fall $c_1 = 1$ wirken die Pfade sehr geradlinig und eckig. Deshalb wird nur eine schlechte Approximation an das Minimum gefunden. In der Liniensuche werden mehr Kandidaten benötigt, um einen Passenden zu finden, der die Wolfe-Bedingungen erfüllt und meistens wird keiner gefunden, weswegen der beste von ihnen zurückgegeben wird. Bei f_2 tritt wieder der Fall ein, dass die Entwicklungspunkte am Minimum vorbei gehen und ins negative Unendliche abfallen.

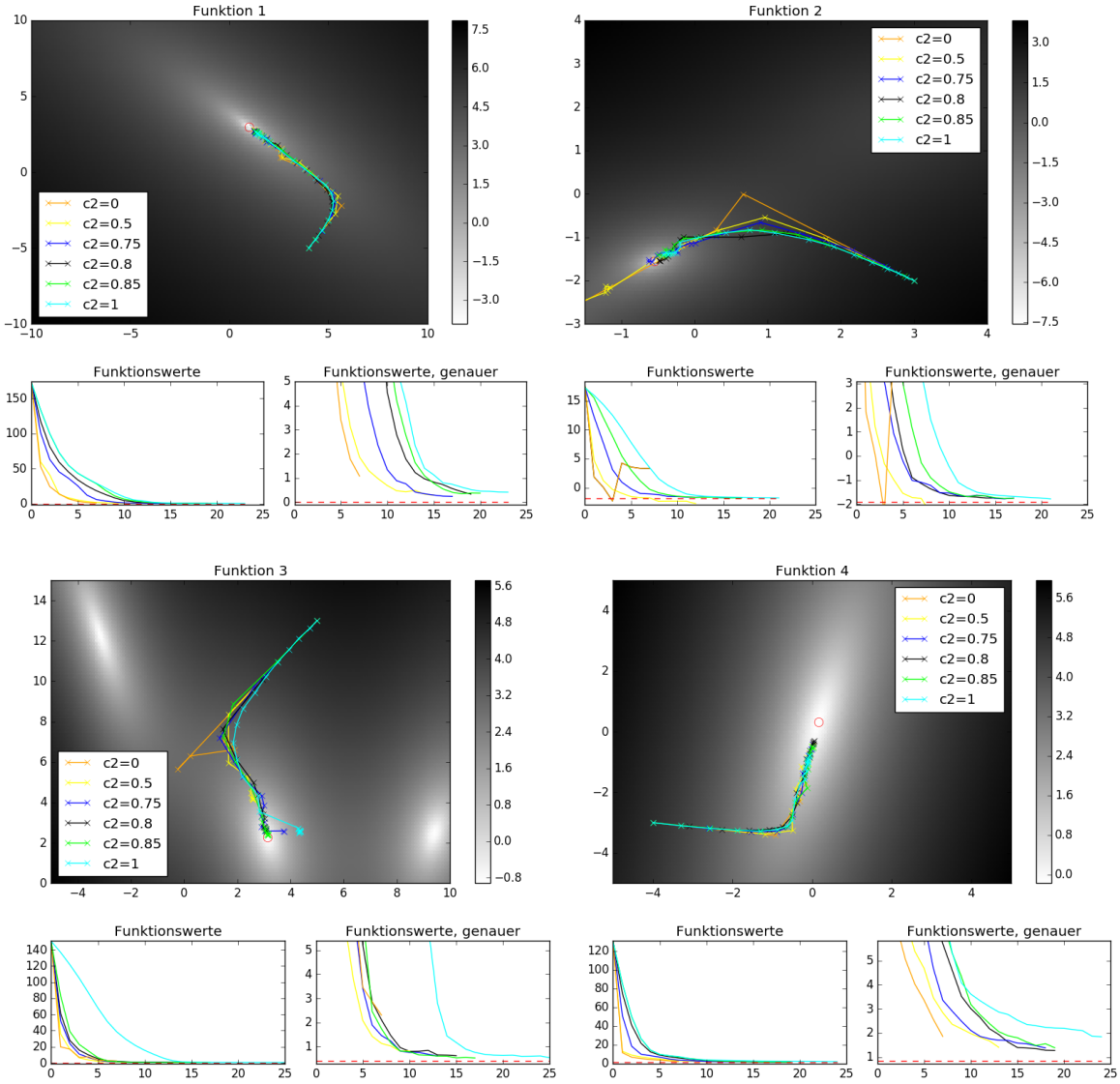

Abbildung 7.6: Auswirkung der Veränderung des Wolfe-Parameters c_2 bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Wahl $c_2 = 0.8$ ist dabei schwarz gekennzeichnet.

Die kleinen Veränderungen um einen Wert von 0.05 bewirken im Verhältnis nicht ganz so viel. Sogar die Veränderung auf 0.5 trägt keine Verschlechterungen mit sich. Wenn jedoch die Funktionsauswertung von f_3 verglichen werden, kommen wir zu dem Schluss, dass die Wahl von $c_1 = 0.05$ gut gewählt wurde.
Auch bei weiteren Experimenten, die im Rahmen dieser Bachelorarbeit betrieben wurden, hat sich herausgestellt, dass die Wahl gut gelungen ist. In diesen Experimenten wurde noch eine weitere Funktion benutzt und der Durchschnitt wurde über alle Funktionen gebildet. Die weitere Funktion wurde im Python-Code integriert und die Auswertung der Experimente wurde, wie auch der Programmiercode, dieser Arbeit beigefügt.

Die Tabelle aus dem Anhang unterstützt dies nicht ganz. Bei ihr liegen im Durchschnitt die Ausgaben von $c_1 = 0$ näher am Minimum. Dies ist aber nur die Ausgabe für Funktion f_1, für die anderen sehen wir am Plot, dass diese Wahl zu Problemen und Abweichungen führen kann.

Unser zweiter Wolfe-Parameter c_2 stammt aus der Krümmungsbedingung (W-II). In ihr wurde eine Vergrößerung der Steigung um einen bestimmten Faktor verlangt. Wie auch für c_1 hatten wir festgehalten, dass die Wolfe-Bedingungen für $c_2 = 0$ alle Werte annehmen, an denen $\nabla f(x_{k+1}) \geq 0$ gilt und für $c_2 = 1$ nur Werte mit einer Steigung größer als $\nabla f(x_k)$.
In Abbildung 7.6 sehen wir, wie sich die Wege und Werte entwickeln, wenn wir c_2 verändern. Wieder ist die Auswirkung auf Funktionen mit eindeutigem globalen Minimum gering. Jedoch werden für die Funktionen f_2 und f_3 direkt $c_2 = 0$ und $c_2 = 0.5$ ausgeschlossen, da nicht nur die Wege, sondern auch die Funktionswerte viel schlechter konvergieren. Auch die Werte 0.85 und 1 sehen im Vergleich der Funktionswerte so aus, als wären sie keine gute Wahl. Sie konvergieren langsamer, was höhere Rechenkosten nach sich ziehen kann.
Auch bei den weiteren Experimenten kam heraus, dass $c_2 = 0.8$ eine sehr gut Wahl ist. Dass im Matlab-Code $c_2 = 0.5$ gesetzt wurde könnte nur daher kommen, dass für die voreingestellten Werte und die ausgewählte Funktion dieser Wert am besten war. Dies ist jedoch nur eine Vermutung.
Betrachten wir nun auch wieder die Tabelle für Funktion f_1, dann sehen wir, dass auch hierbei fast immer $c_2 = 0.5$ die beste Wahl ist. Da jedoch für die Funktionen mit mehreren Minima dieser Wert zu weit abweicht, schließen wir ihn aus.

Der dritte Wolfe-Parameter c_w wird am meisten Auswirkung auf das Verfahren haben. Dies können wir sagen, da er die Schwelle ist, über der die Wahrscheinlichkeit liegen muss, dass ein Entwicklungspunkt die Wolfe-Bedingungen erfüllt, damit dieser angenommen wird.
Abbildung 7.7 zeigt uns, dass die Funktionswerte diesmal für alle Funktionen, nicht nur für die mit mehreren Minima, stark abweichen. Dies kann jedoch für die Funktionen f_1 und f_4 nur gesehen werden, wenn wir die genaueren Funktionswerte betrachten. Die Auswertungswege sehen relativ unauffällig aus, also weichen sie nicht stark ab. Unsere Standardwahl ist wieder mit schwarz gekennzeichnet und wir sehen, wenn wir uns alle Funktionen anschauen, dass er sehr gut gewählt ist.
Die Randwerte $c_w = 0$ und $c_w = 1$ werden direkt ausgeschlossen. Keiner der beiden Fälle hat ein gutes Konvergenzverhalten. Für den Wert 0 wird jeder Punkt angenommen, weswegen wir den Weg auch in eine vollkommen falsche Richtung entwickeln können. Dies sehen wir besonders für die dritte Funktion. Der Wert 1 wird fast nie wirklich benutzt,

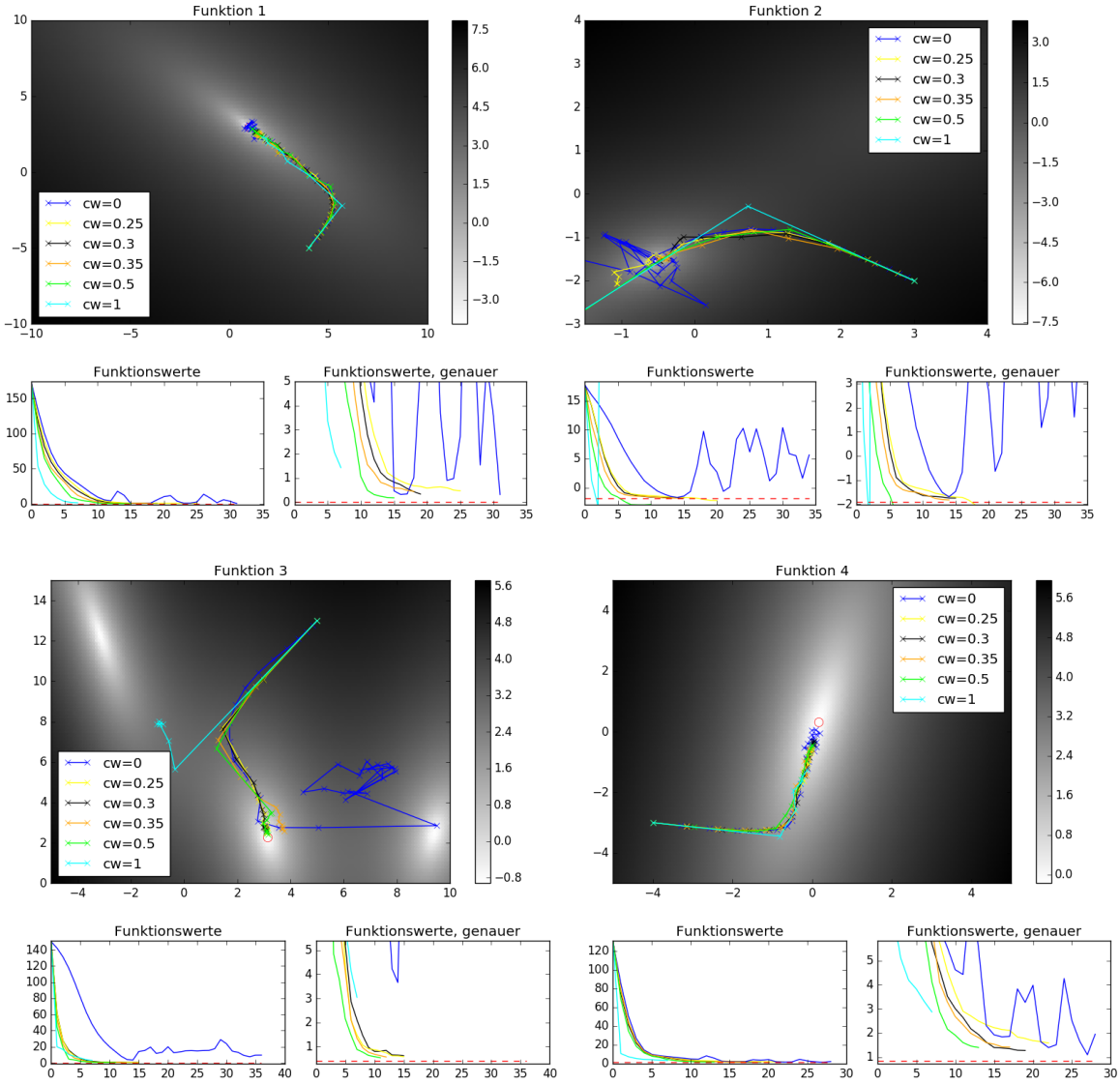

Abbildung 7.7: Auswirkung der Veränderung des Wolfe-Parameters c_w bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Die Wahl $c_w = 0.3$ ist dabei schwarz gekennzeichnet.

stattdessen wird immer der beste Wert aus den Kandidaten gewählt. Meistens werden die Wolfe-Bedingungen von keinem Kandidaten vollkommen, also mit $p_t^{Wolfe} = 1$ erfüllt. Deswegen kommt es zu Fehleinschätzungen und schlechten Entwicklungspunkten. Betrachten wir nun die Werte, welche näher am Standardwert liegen, dann sehen wir, dass es grundsätzlich keine starken Auswirkungen gibt, außer bei Funktion f_2 und den Werten $c_w = 0.5$ und $c_w = 0.25$, dort wird gegen das negative Unendliche konvergiert. Für die verbliebenen Werte sind die Entwicklungpfade und die Werte scheinbar gleich. Um eine Unterscheidung zu machen, schauen wir uns die Tabelle aus dem Anhang an. Wir betrachten jeweils den Durchschnitt der einzelnen Tabellen. Wenn wir alle mitein-

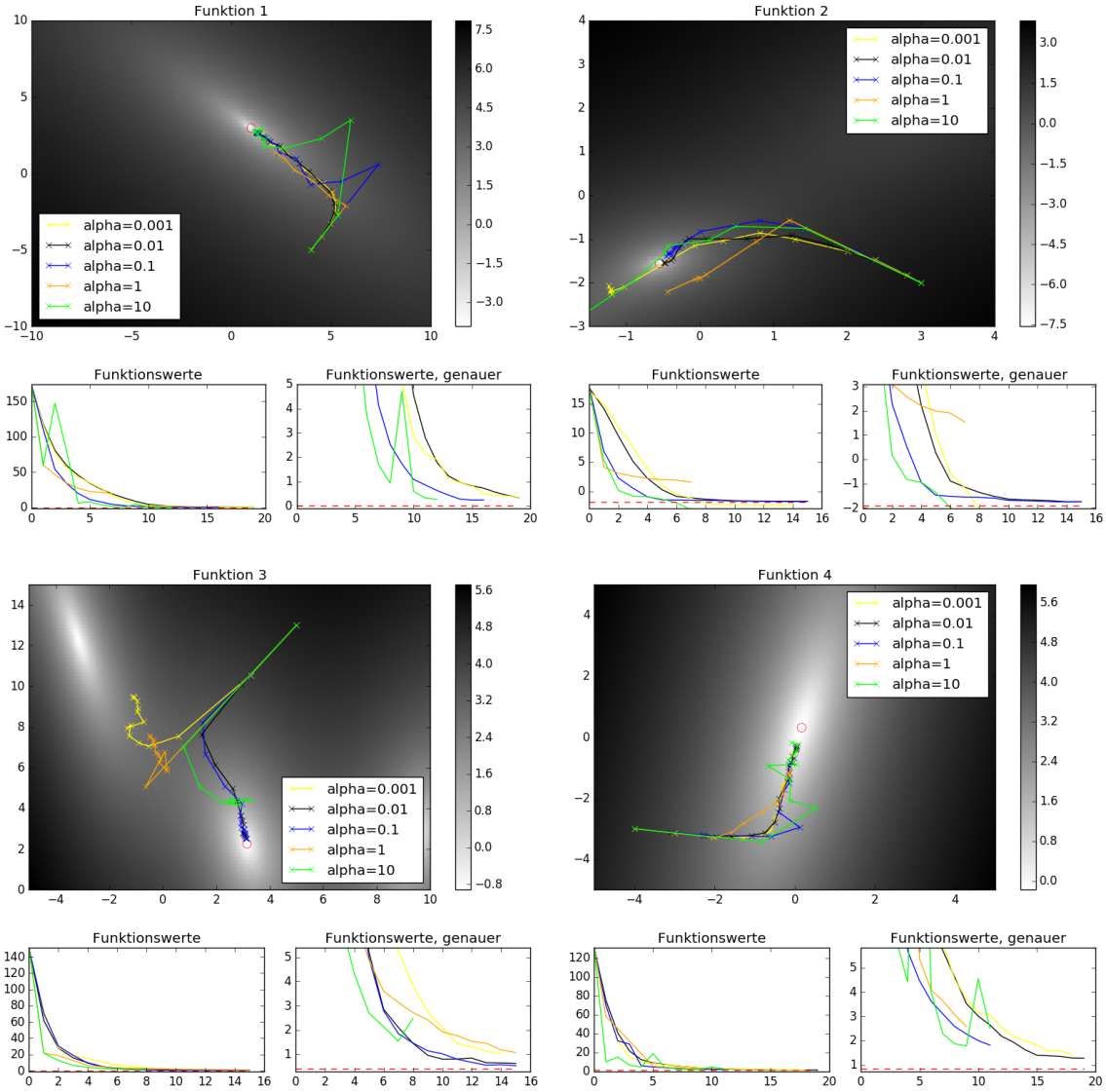

Abbildung 7.8: Auswirkung der Veränderung der initiierten Schrittweite α_0 bei den Funktionen f_1 bis f_4 auf den Auswertungsweg und die Funktionswerte. Dabei ist der Standardwert $\alpha_0 = 0.05$ schwarz gekennzeichnet.

ander vergleichen, ist der Durchschnitt bei $c_w = 0.5$ am kleinsten. Da wir diesen Wert jedoch schon ausgeschlossen haben, vergleichen wir nur noch die Tabellen der restlichen c_w-Werte, wobei der Durchschnitt bei $c_w = 0.3$ klein ist. Das minimale Ergebnis steht aber in der Tabelle von $c_w = 0.25$. Dies liefert uns aber auch kein klares Ergebnis.
In den anderen numerischen Experimenten, die den Durchschnitt aller Funktionen betrachten, lief jedoch $c_w = 0.3$ ein kleines bisschen besser als die anderen Werte.
Wir können uns hierbei also nicht genau festlegen. In Kapitel 6 wurde bereits angesprochen, dass der Wert nur ungefähr bei $c_w = 0.3$ liegen muss, da die meisten Wahrscheinlichkeiten viel größer oder viel kleiner sind. Dies bestätigt sich hiermit.

7.6 Initiierte Schrittweite

Nun kommen wir zu den letzten Experimenten. Wir haben bereits in Kapitel 6 beschrieben, wie sich die Schrittweite im Laufe des Verfahrens verändert. Dabei haben wir eine initiierte Schrittweite α_0 gesetzt. Nun wollen wir betrachten, wie sich die Entwicklungspunkte und die Funktionswerte verändern, wenn α_0 auf andere Werte gesetzt wird.

In Abbildung 7.8 wurden jeweils Werte mit dem Multiplikand 10 gewählt. Wir beginnen also bei $\alpha_0 = 0.001$ und gehen bis $\alpha_0 = 10$ hoch. Dabei können wir bereits erwarten, dass die Punkte für sehr kleine Werte nicht immer beim Minimum ankommen werden und für sehr Große ein schlechtes Konvergenzverhalten folgt. Dies spiegelt die Abbildung nicht ganz wider.
Wir sehen, dass unser Standardwert sehr gut gewählt wurde, da er schnell konvergiert und der einzige Wert $\alpha_0 = 0.1$, der besser gewählt wäre, zwischendurch schlecht gewählte Punkte hat, was für eine andere Anzahl von Funktionsauswertungen zu Fehlern führen kann. Sehr große Werte sollten nicht gesetzt werden, da es auch dort zu vielen Abweichungen kommt und teilweise nicht das Minimum erreicht wird, bzw. es wird daran vorbei entwickelt. Auch die Werte $\alpha_0 = 0.001$ und $\alpha_0 = 1$ konvergieren langsam und erreichen deswegen nicht immer das Optimum. Teilweise konvergieren sie auch zu anderen Werten, also zu anderen Minima.
Bei anderen Experimenten kam jedoch heraus, dass die Wahl $\alpha_0 = 0.1$ viel besser ist als unsere Standardeinstellung, da die Abweichungen für das Gesamtergebnis keine Beeinträchtigungen haben.
Wir sehen also, dass die Standardeinstellung eine gute, jedoch nicht die beste Wahl für unsere Funktion ist. In [1] wurden diese Experimente jedoch noch für Trainingsdaten im Sinne des maschinellen Lernens ausgeführt, weswegen die Autoren sehr wahrscheinlich auf diese Wahl kamen. Dies kann jedoch nicht nachgeprüft werden.

8 Zusammenfassung und Ausblick

In dieser Arbeit haben wir gesehen, wie das stochastische Gradientenverfahren für das Optimierungsproblem einer reellen und gestörte Funktion aufgebaut ist und aus welchen Grundlagen die einzelnen Elemente bestehen. Sie hat den wissenschaftlichen Artikel [1] als Fundament. Um das Verfahren aufbauen zu können, mussten wir zuerst einige Informationen, wie das Gradientenverfahren und die zugehörigen Wolfe-Bedingungen, aus Grundvorlesungen des Mathematik-Studiums zusammentragen und diese vertiefen. Danach konnten die Grundlagen für die probabilistische und gestörte Art umgebaut werden. Als Beispiel für eine Anwendung dieses Verfahrens haben wir das maschinelle Lernen und das Approximieren an eine Datenmenge betrachtet.
Wir haben versucht, aus unserem Verfahren einen Black-Box-Algorithmus zu bauen. Das heißt, wir haben keine zu setzenden Parameter, da wir alle Parameter eliminiert oder auf einen festen Wert gesetzt haben. Die Werte, die im Fundament vorgeschlagen wurden haben wir mithilfe von Experimenten auf ihr Zutun geprüft, indem wir im selbstgeschriebenen und kostengünstigen Python-Code verschiedene andere Werte eingesetzt und benutzt haben. Wir haben in den Experimenten auch geschaut, was passiert, wenn wir bestimmte Parameter des Verfahrens verändern, die nicht direkt aus der Idee stammen. Dies waren die Veränderung des Startwerts, die Skalierung der Störung oder das Abbruchkriterium, welche uns einen besseren Einblick in das Verhalten des Verfahrens gegeben haben. Das Ergebnis der Experimente zu den anderen Parametern war, dass die Werte gut gesetzt wurden. Es gab bei einigen zwar minimale Verbesserungen, doch keine so deutlichen, dass die Werte ausgetauscht werden sollten.
Wir haben somit herausbekommen, dass Optimierungen auch auf gestörte Einstellungen erweitert werden können. Dafür mussten schon bestehende Prinzipien mit den Ideen des Gauß-Prozesses und der Bayes-Optimierung verknüpft werden. Dabei bringt jede anständige Wahl der initiierten Werte eine gute Approximation an das Minimum des Optimierungsproblems.

Betrachten wir nun, wie mit diesen Ergebnissen weiter gearbeitet werden kann. Im *Maschinellen Lernen* und dem Gebiet des *Data-Mining* gibt es schon viele Anwendungen der Optimierung. Diese können nun auch für gestörte Werte einfach betrachtet werden, indem nur über eine kleinere Datenmenge approximiert wird. Dazu zählen zum Beispiel automatisierte Diagnoseverfahren und Spracherkennungen.
Auch könnte das Optimierungsverfahren für Funktionen mit einer weiteren Unterroutine verbunden werden, welche die Ableitungen selbst berechnen kann, da die Ableitungen bis jetzt noch in das Programm eingefügt werden. In der Praxis ist es jedoch häufig zu aufwendig, die Ableitungen analytisch zu finden, weswegen diese mit einem Computer berechnet werden sollen. Wenn die beiden Verfahren verknüpft werden, bekommen wir sehr mühelos, nur mit der Eingabe einer Funktion, die Approximation des Minimums.

Als weiteres Beispiel für ein Anwendungsgebiet betrachten wir die momentane Forschung. Die Autoren M. Mahsereci und P. Hennig arbeiten mit der Emmy Noether Group on Probabilistic Numerics weiter an der Umsetzung von schon bestehenden Algorithmen in die probabilistische Art. Sie arbeiten auch mit dem Deutschen Krebsforschungszentrum zusammen, um maschinelles Lernen und numerische Optimierungstechniken zur Quantifizierung und Minimierung von Unsicherheiten in der Strahlentherapie anzuwenden und vorausschauende Modelle für die Ergebnisse anzufertigen [19].
Die Forschung hat also noch viele Aufgaben, die durch unsere Problemstellung gelöst werden können.

A Tabelle zur Auswertung der Wolfe-Parameter

Dies sind Tabellen für die Auswertung der Approximationen von Funktion f_1 aus Kapitel 7. In den Plots konnte nur dargestellt werden, wie sich jeweils ein Wert der Wolfe-Bedingungen verändert, da es sonst zu unübersichtlich wäre. Deswegen zeigen diese Tabellen die Veränderung nach allen drei Parametern auf einmal. Jede einzelne Tabelle stellt die Auswertung eines anderen c_w-Wertes dar. Weiterhin ist jede Zeile ein c_1-Wert und jede Spalte ein c_2-Wert. Im Gesamten können wir es uns als eine 3D-Tabelle vorstellen. Die einzelnen Werte sind der Durchschnitt von zehn nacheinander ausgeführten Optimierungen. In grau ist der Standardwert gekennzeichnet, in grün die beste und in rot die schlechteste Wahl der Werte.

$c_w = 0$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	0.33862	0.52259	0.39189	0.50889	0.51611	0.51282
0.05	0.79881	0.57210	0.63842	0.51609	0.51609	0.51609	0.59293
0.1	0.79881	0.61316	0.51609	0.51609	0.51609	0.51609	0.57939
0.5	0.79881	0.71537	1.76346	1.76346	1.76346	1.76346	1.42800
1	0.79881	0.88953	1.60347	1.75535	1.90196	2.56011	1.55123
Durchschnitt	0.79881	0.62576	1.00881	0.98858	1.04250	1.17437	0.93960

$c_w = 0.25$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	0.28616	0.29075	0.29543	0.38119	0.25019	0.38376
0.05	0.79881	0.30220	0.27949	0.49197	0.36187	0.25734	0.41528
0.1	0.79881	0.29098	0.43630	0.36986	0.47078	0.24062	0.43456
0.5	0.79881	0.45811	0.32246	0.51046	0.38066	0.45705	0.48793
1	0.79881	0.69526	0.69526	0.96266	0.85770	0.85408	0.81063
Durchschnitt	0.79881	0.40654	0.40485	0.52608	0.41186	0.49044	0.50643

$c_w = 0.3$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	0.27446	0.37470	0.43730	0.34238	0.34661	0.42904
0.05	0.79881	0.34564	0.32834	0.47203	0.36906	0.33344	0.44122
0.1	0.79881	0.35209	0.31955	0.30450	0.30450	0.26506	0.39075
0.5	0.79881	0.28389	0.30875	0.35323	0.44103	0.48129	0.44450
1	0.79881	0.69526	0.69526	0.69526	0.69526	0.75790	0.72296
Durchschnitt	0.79881	0.39027	0.40532	0.45246	0.43044	0.43686	0.48569

$c_w = 0.35$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	0.28919	0.30314	0.28199	0.43997	0.34202	0.40919
0.05	0.79881	0.27958	0.33731	0.46979	0.38411	0.36940	0.43983
0.1	0.79881	0.21001	0.32488	0.25458	0.29966	0.40276	0.38178
0.5	0.79881	0.23879	0.18518	0.36316	0.31341	0.41787	0.38620
1	0.79881	0.69526	0.69526	0.69526	0.69526	0.69526	0.71252
Durchschnitt	0.79881	0.34257	0.36915	0.41296	0.42649	0.44546	0.46591

$c_w = 0.5$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	0.33899	0.22659	0.30178	0.23436	0.31690	0.36957
0.05	0.79881	0.36354	0.45200	0.23839	0.48391	0.15771	0.41573
0.1	0.79881	0.36795	0.25752	0.46775	0.35172	0.20373	0.40791
0.5	0.79881	0.53706	0.40751	0.27178	0.36218	0.30740	0.44746
1	0.79881	0.69526	0.69526	0.69526	0.69526	0.69526	0.71252
Durchschnitt	0.79881	0.46056	0.40778	0.39499	0.35472	0.33620	0.45882

$c_w = 1$	c_2						
c_1	0	0.5	0.75	0.8	0.85	1	Durchschnitt
0	0.79881	1.00510	1.00510	1.00510	1.00510	1.00043	0.97072
0.05	0.79881	1.00510	1.00510	1.00510	1.00510	1.00043	0.97072
0.1	0.79881	1.00510	1.00510	1.00510	1.00510	1.00043	0.97072
0.5	0.79881	0.82942	0.78613	0.79610	0.80077	0.77742	0.79811
1	0.79881	0.69526	0.69526	0.69526	0.69526	0.69526	0.71597
Durchschnitt	0.79881	0.90800	0.89934	0.90133	0.90227	0.89712	0.88448

Literatur

[1] M. Mahsereci und P. Hennig: *Probabilistic Line Searches for Stochastic Optimization*. In Advances in Neural Information Processing Systems 28, Seiten 181-189, Curran Associates, Inc., 2015.

[2] L. Bottou: *Large-Scale Machine Learning with Stochastic Gradient Descent*. In Proceedings of the 19th Int. Conf. on Computational Statistics, Seiten 177-186, Springer, 2010.

[3] J. Garcke: *Einführung in die Numerische Mathematik*. Mathematisch-Naturwissenschaftliche Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn, Vorlesungsmitschrift, 2015.

[4] K. Graichen: *Methoden der Optimierung und optimalen Steuerung*. Fakultät für Ingenieurwissenschaften und Informatik der Universität Ulm, Vorlesungsskript, 2014.

[5] B. von Harrach: *Einführung in die Optimierung*. Fachbereich Mathematik - IMNG der Universität Stuttgart, Vorlesungsskript, 2015.

[6] T. Schaul, S. Zhang und Y. LeCun: *No more pesky learning rates*. In 30th International Conference on Machine Learning, Seiten 343-351, 2013.

[7] A. Bovier: *Einführung in die Wahrscheinlichkeitstheorie*. Mathematisch-Naturwissenschaftliche Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn, Vorlesungsskript, 2013.

[8] G. Tutz und F. Leitenstorfer: *Mehrdimensionale Zufallsvariablen*. Ludwig-Maximilians-Universität München, Handout zur Vorlesung, `http://www.statistik.lmu.de/~leiten/Lehre/Material/Mulit_07/mehrdimZV.pdf` (2016).

[9] L. Fahrmeir, C. Heumann, R. Künstler, I. Pigeot und G. Tutz: *Statistik, der Weg zur Datenanalyse*. Seiten 302-362, 7.Auflage, Springer, 2016.

[10] N. Bäuerle: *Stochastik II*. Vorlesungsmitschrift, `http://mitschriebwiki.nomeata.de/Stochastik2.pdf`, 2007.

[11] Spektrum: *Brownsche Bewegung*. `http://www.spektrum.de/lexikon/physik/brownsche-bewegung/2040`, 1998.

[12] C. Rasmussen und C. Williams: *Gaussian Processes for Maschine Learning*. MIT, 2006

[13] A. Genz: http://www.math.wsu.edu/faculty/genz/software/matlab/bvn.m. Department of Mathematics and Statistics at Washington State University (2017).

[14] Z. Drezner und G. Wesolowsky: *On the computation of the bivariate normal integral*. Journal of Statistical Computation and Simulation. 35, Seiten 101-107, 1989.

[15] S. Russell und P.Norvig: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[16] Felix Burkhardt: *Maschinelles Lernen*. http://felix.syntheticspeech.de/ML.html (2017).

[17] E. Alpaydin: *Maschinelles Lernen*. Oldenbourg Wissenschaftsverlag, 2008.

[18] *Virtual Library of Simulation Experiments*: https://www.sfu.ca/~ssurjano/optimization.html. (2017).

[19] *Deutsches Krebsforschungszentrum in der Helmholtz-Gemeinschaft*: http://www.dkfz.de/en/medphys/optimization_algorithms/optimization_algorithms.html. (2017).