der Bundeswehr
Universität München

Fakultät für Mathematik und Plasmatechnik
Institut für Mathematik und Operations Research

Stochastische Lösungsansätze zu mehrdimensionalen nichtlinearen Optimierungsproblemen

Sabine Schmied

Vorsitzender des Promotionsausschusses:	Univ.-Prof. Dr.-Ing. Stefan Lindenmeier
1. Berichterstatter:	Univ.-Prof. Dr. rer. nat. Dr.-Ing. Stefan Schäffler
2. Berichterstatter:	Priv.-Doz. Dr.-Ing. Gerhard Staude

Mit der Promotion erlangter akademischer Grad:
Doktor-Ingenieur
(Dr.-Ing.)

Tag der Prüfung: 20.03.2009

für

Robert und Tobias Schmied

und

Maria und Gerhard Welzel

Inhaltsverzeichnis

Einleitung

Viele Probleme aus dem Bereich technisch-wissenschaftlicher Anwendungen können als globale nichtlineare Optimierungsprobleme formuliert werden. Beispielsweise möchte man bei der Produktion möglichst wenig Material verwenden und dennoch eine bestmögliche Stabilität erreichen. Ein anderes Beispiel findet sich im Bauwesen. Hierbei sollen Funktionen an umfangreiche Mengen von Messdaten angepasst werden. Das abschließende Beispiel (vgl. Abschnitt 6.2.2) in dieser Arbeit untersucht genau die Fragestellung, wie die Infiltrationsrate von Wasser in den Boden in Abhängigkeit der kumulativen Niederschlagsenergie auf Basis verschiedenster Messdaten beschrieben werden kann. Dabei sind die „Messfehler“ also die Abweichungen von den Messdaten von der (idealisierenden) Funktion zu minimieren. Es liegt also ein klassisches nichtlineares Optimierungsproblem vor, die Suche nach der bestmöglichen Parameterkombination für die Funktion erfordert eine Suche über den gesamten Parameterraum (etwa den $\mathbb{R}^n$). Ziel bei der globalen Optimierung ist es damit auch, unter einer im allgemeinen großen Zahl gefundener lokaler Optima die globalen Optima zu lokalisieren. Im Gegensatz zur lokalen Optimierung, bei der eine optimale Lösung in der Nähe eines vorgegebenen Punktes gesucht wird, verlangt die globale Optimierung das Auffinden des besten lokalen Optimums.

Das Ziel dieser Arbeit ist es, ein ableitungsfreies Verfahren zur unrestringierten, globalen Optimierung, das an die zu optimierende Funktion nichts voraussetzt, mit Hilfe eines stochastischen Lösungsansatzes zu verbessern. Hierbei soll sich die Chance für das Auffinden eines Bereiches erhöhen, in dem sich ein globales Optimum der gegebenen Funktion befinden könnte. Außerdem sollten durch das Verfahren idealerweise alle globalen Optima und nicht nur ein globales Optimum gefunden werden, was natürlich nicht immer gewährleistet werden kann. Dennoch zeigen die Simulationsergebnisse die sich eröffnenden Möglichkeiten in diesem Punkt. Deshalb ist es auch wichtig, dass das Verfahren nicht deterministisch ist, sondern dass das zu bestimmende Verfahren bei wiederholter Ausführung mit gleichen Startbedingungen nicht immer identische Ergebnisse zu liefern imstande ist.

Die mathematischen Definitionen zu globalen und lokalen unrestringierten Optimierungsproblemen werden im ersten Kapitel aufgezeigt. Außerdem wird der Unterschied zwischen linearer und nichtlinearer Optimierung geklärt.

Im zweiten Kapitel werden verschiedene numerische Verfahren untersucht, ob sie die Bedingungen für das gesuchte Verfahren erfüllen.

Der Algorithmus des ableitungsfreien Optimierungsverfahrens von Nelder und Mead, der in den gebräuchlichsten Mathematikprogrammen implementiert ist, wird im dritten Kapitel behandelt. Er liefert Ansatzpunkte für eine Verfahrensverbesserung im Sinne der obigen Ausführungen. Insbesondere die Tatsache, dass er ein Algorithmus zur globalen Optimierung, ableitungsfrei und in den gängigsten Ma-

thematikpakten implementiert ist, schafft einen besonderen Anreiz zur weitergehenden Untersuchung. Auch dessen innere Konstruktion spielt wie sich zeigen wird eine wesentliche Rolle bei der Entscheidung, diesen Algorithmus zu erweitern.

Die Algorithmuserweiterung mit stochastischen Mitteln soll mit Hilfe eines aus dem Bereich der Zufallszahlengeneratroren stammenden Verfahrens durchgeführt werden. Zufallszahlen sind für nicht-deterministische Verfahren relevant. Deshalb wird im vierten Kapitel darauf eingegangen. Vor allem Zufallszahlen mit vorgegebener Wahrscheinlichkeitsdichte sind hier besonders interessant.

Die letztgenannten Ansätze werden dann im fünften Kapitel modifiziert und mit dem Verfahren von Nelder und Mead kombiniert. Die Beschreibung dort stellt die Kernidee der Arbeit dar. Die Realisierung mit Beispielrechnung erfolgt im Computerprogramm Mathematica.

Im Kapitel sechs wird dann dieser neue Algorithmus an bekannten Funktionen und an dem konkreten Praxisbeispiel aus dem Wasserwesen getestet und ausgewertet.

Das Kapitel sieben gibt zu den erzielten Ergebnissen einen Ausblick.

1 Optimierungsprobleme

1.1 Globale und lokale unrestringierte Optimierung

Gegenstand dieser Arbeit ist folgendes Optimierungsproblem. Gegeben ist eine Funktion

$$f : D \subseteq \mathbb{R}^n \to \mathbb{R}.$$

Gesucht ist ein Punkt $m = (m_1, \ldots, m_n) \in D$ mit $f(m) \leq f(x)$ für alle $x \in D$. Betrachtet wird ein so genanntes globales Minimierungsproblem, das durch folgende Schreibweise abgekürzt wird:

$$\min f(x) \text{ mit } x \in D.$$

Daneben gibt es auch globale Maximierungsprobleme. Beide Problemstellungen werden unter dem Begriff globale Optimierungsprobleme zusammengefasst. Im Folgenden werden ausschließlich Minimierungsprobleme betrachtet, da beide Problemstellungen äquivalent sind. Denn es gilt gemäß [BOR01]

$$\min f(x) \text{ mit } x \in D \equiv \max\ -f(x) \text{ mit } x \in D.$$

D bezeichnet dabei die Definitionsmenge der Funktion f. Sie soll hier ohne Einschränkungen als $D = \mathbb{R}^n$ festgelegt sein. Das Optimierungsproblem heißt in diesem Fall auch **unrestringiert**, ansonsten restringiert. Weiter sei an die Funktion f, die auch Zielfunktion genannt wird, keinerlei Voraussetzung geknüpft. Das bedeutet z.B., dass die Funktion weder stetig noch differenzierbar sein muss. Die zu optimierende Funktion ist eine reellwertige Funktion. Die Lösung des Optimierungsproblems erfolgt durch die Suche nach m, mit dem Ziel, die Funktion f zu minimieren oder maximieren. Der Vorgang werde mit **Optimierung** bezeichnet.

Gesucht ist das globale Optimum, d.h. das globale Minimum m bzw. entsprechend der folgenden Festlegung das globale Maximum M. Unter obigen Annahmen ist nicht klar, ob ein globales Optimum existiert. Es sei ein $a \in \mathbb{R}$ gegeben. Wenn vorausgesetzt wird, dass die Niveaumengen

$$\{x \in \mathbb{R}^n | f(x) \leq a\}$$

kompakt sind, ist die Existenz eines globalen Minimums gesichert. Dies wird entsprechend für globale Optima formuliert. Der Punkt m heißt dann auch **globales Minimum** und es ist

$$f(m) = \min \big\{ f(x) |\ x \in D \big\}.$$

Entsprechend heiße $M \in \mathbb{R}^n$ **globales Maximum**, falls

$$f(M) = \max\{f(x) | \, x \in D\}.$$

Die Optimierung wird in praktischen Anwendungen meist durch numerische Verfahren ermittelt. Dabei geschieht es häufig, dass nicht das globale Minimum gefunden wird, sondern ein Punkt $x_0 \in D$, der lediglich für Punkte in der Nähe von x_0 den kleinsten Funktionswert liefert. Das Verfahren liefert eine lokal gültige und optimale Lösung. Man spricht dann auch von einem lokalen Minimum. Entsprechend gibt es ein lokales Maximum, zusammen spricht man von lokalen Extrema. Lokale Extrema beziehen sich im Gegensatz zum globalen Extremum nur auf eine Umgebung eines Punktes der Definitionsmenge. Als Umgebung $U(x_0)$ eines Punktes $x_0 \in \mathbb{R}^n$ bezeichnet man eine offene Menge in $\mathbb{R}^n$, die den Punkt x_0 enthält. Speziell nennt man einen offenen Quader der Form

$$U_\delta(x_0) := \{x \in \mathbb{R}^n | x_0 - \delta < x < x_0 + \delta\} \quad \text{(komponentenweise Betrachtung)}$$

δ-Umgebung des Punktes x_0, wobei $\delta \in \mathbb{R}^n$ ist.

Eine Funktion f: $D = \mathbb{R}^n \rightarrow \mathbb{R}$ besitzt in $x_0 \in D$ ein lokales Extremum, wenn eine δ-Umgebung $U_\delta(x_0)$ existiert mit der Eigenschaft

$f(x) \geq f(x_0)$ für alle x $\in U_\delta(x_0)$ $\quad m_l := x_0$ heißt **lokales Minimum**
$f(x) \leq f(x_0)$ für alle x $\in U_\delta(x_0)$ $\quad M_l := x_0$ heißt **lokales Maximum**.

Es gibt Optimierungsverfahren, die einerseits wie beispielsweise das Newton-Verfahren darauf ausgerichtet sind, in einem lokalen Bereich entsprechend ein lokales Minimum zu finden, andererseits wie beispielsweise das Nelder/Mead-Verfahren auf der Suche nach globalen Minima manchmal lokale Minima ermitteln. In beiden Fällen ist keine Lösung des globalen Optimierungsproblems gefunden. Insbesondere im zweiten Fall, bei dem die Intention einer globalen Suche gegeben ist, ist das Verhalten des Verfahrens auch deshalb nicht günstig, weil die Suche deterministisch verläuft. Jedes kleinste lokale Minimum $min\{m_l | m_l \; lokales \; Minimum\}$ ist das globale.

1.2 Lineare und nichtlineare Optimierung

Im Weiteren werden grundlegende Optimierungsprobleme vorgestellt. Dabei werden zunächst entgegen obiger Annahme bestimmte Voraussetzungen an die Zielfunktion gestellt.

1.2.1 Lineare Optimierungsprobleme

Unter einem linearen Optimierungsproblem versteht man die Aufgabe, eine lineare Zielfunktion unter Beachtung von linearen Nebenbedingungen zu maximieren bzw. zu minimieren [DOM95]. Der Vorgang

heißt lineare Optimierung. Gegeben sind eine Matrix $A \in \mathbb{R}^{p,n}$ und zwei Vektoren $b = (b_1, \ldots, b_p)^T \in \mathbb{R}^p$ und $c = (c_1, \ldots, c_n)^T \in \mathbb{R}^n$. Eine zulässige Lösung ist ein Vektor $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$, der die Nebenbedingungen erfüllt. Insgesamt erhalten wir folgende Problemstellung:

$$\begin{aligned} &\min f(x_1, ..., x_n) = c_1x_1 + ... + c_nx_n \text{ mit} \\ &a_i^T x \leq b_i \qquad \text{für } i = 1, ..., p_1 \\ &a_i^T x \geq b_i \qquad \text{für } i = p_1 + 1, ..., p_2 \\ &a_i^T x = b_i \qquad \text{für } i = p_2 + 1, ..., p. \end{aligned}$$

Soll x komponentenweise nichtnegativ sein, sind entsprechende Nebenbedingungen $x_j \geq 0$ für $j = 1, ..., n$ zu stellen. Die lineare Optimierung hat das Ziel, unter allen zulässigen Vektoren x einen zu finden, der das Skalarprodukt minimiert [@WIS].

Lineare Optimierungsprobleme sind durch das Simplexverfahren lösbar [BOR01] und werden somit nicht weiter betrachtet.

1.2.2 Nichtlineare Optimierungsprobleme

Nichtlineare Optimierungsprobleme sind für die vorliegende Arbeit von Bedeutung. Im Gegensatz zu linearen Problemen besitzen nichtlineare Optimierungsprobleme eine nichtlineare Zielfunktion und/oder mindestens eine nichtlineare Nebenbedingung [DOM95]. Nichtlineare Optimierunsgsprobleme lassen sich mit $p, l \in \mathbb{N}$ folgendermaßen formulieren:

$$\begin{aligned} &\min f(x_1, ..., x_n) \text{ mit} \\ &g_i(x_1, ..., x_n) \leq 0 \text{ für } i = 1, ..., p \\ &h_j(x_1, ..., x_n) = 0 \text{ für } j = 1, ..., l \\ &\text{wobei } g_i : D \to \mathbb{R} \text{ und } h_j : D \to \mathbb{R}, \quad D \in \mathbb{R}^n. \end{aligned}$$

Es ist $x \in \mathbb{R}^n$ und die Funktionen f, g_i und h_j sind reellwertig. Der Zulässigkeitsbereich des nichtlinearen Optimierungsproblems ist damit gemäß [BOR01] so beschreibbar :

$$X = \{x \in D \subseteq \mathbb{R}^n \mid g_i(x) \leq 0,\ h_j(x)=0 \text{ für } i = 1, ..., p \text{ und } j = 1, ..., l\}$$

f ist die Zielfunktion, g_i sind die Ungleichungsrestriktionen und h_j die Gleichungsrestriktionen.

Im Folgenden werden einige Spezialfälle von nichtlinearen Optimierungsproblemen angegeben. Hierzu zählen die **quadratischen Optimierungsprobleme**. Sie behandeln Probleme, die sich von linearen Optimierungsproblemen nur durch die Zielfunktion unterscheiden. Diese enthält neben linearen Termen

auch quadratische Ausdrücke. Die Zielfunktion eines quadratischen Maximierungsproblems wird so formuliert:

$$\min f(x) = c^T x + \tfrac{1}{2} x^T C x$$

Dabei ist $x \in \mathbb{R}^n$ ein Spaltenvektor und $C \in \mathbb{R}^{n,n}$ ist eine symmetrische Matrix mit reellwertigen Koeffizienten.

Konvexe Optimierungsprobleme sind ein weiterer Spezialfall der nichtlinearen Optimierung. Vorausgesetzt wird hierbei, dass die Zielfunktion f lineare Nebenbedingungen und keine gemischten Terme enthält. Sie ist somit als Summe von Funktionen f_j, $j = 1, \ldots, n$ mit je einer Variablen $x_j \in \mathbb{R}$ darstellbar. Für die Funktion f_j wird vorausgesetzt, dass sie bei Minimierungsproblemen konvex und bei Maximierungsproblemen konkav ist.

Konvexe Mengen und Funktionen sind allgemein für die Theorie der Optimierungsprobleme von Bedeutung. Eine Funktion $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ heißt konvex bzw. konkav in einem Bereich $I \subseteq D$, wenn für je zwei beliebige Werte x_1, $x_2 \in I$ und jede reelle Zahl $\lambda \in [0, 1]$ gilt:

$$\begin{aligned} f(\lambda x_2 + (1-\lambda)x_1) &\leq \lambda f(x_2) + (1-\lambda) f(x_1), \qquad f \text{ heißt } \textbf{konvex}, \\ f(\lambda x_2 + (1-\lambda)x_1) &\geq \lambda f(x_2) + (1-\lambda) f(x_1), \qquad f \text{ heißt } \textbf{konkav}. \end{aligned}$$

Die Funktion f heißt strikt konvex, wenn die obige Ungleichung für $x_1 \neq x_2$ im strengen Sinne gilt. Anschaulich bedeutet dies für konvexe Funktionen, dass für beliebiges x und y die Sekante vom Punkt $(x_1,\ f(x_1))$ nach $(x_2,\ f(x_2))$ nicht unterhalb des Graphen der Funktion liegt. Auch die leere Menge ist konvex. Für die späteren Ausführungen (vgl. Kapitel 5) ist auch der Begriff der Konvexkombination wichtig. Für ein $\lambda \in [0, 1]$ und zwei Punkte $x_1, x_2 \in \mathbb{R}^n$ heißt

$$\lambda x_1 + (1-\lambda) x_2 = x_2 + \lambda(x_1 - x_2)$$

die Konvexkombination der beiden Punkte x_1 und x_2.

Ein konvexes Optimierungsproblem mit konvexen Funktionen f_j, $j = 1, \ldots, n$ wird folgendermaßen formuliert:

$$\min f(x) = \sum_{j=1}^{n} f_j(x_j) \text{ mit } Ax \leq b \text{ und } x \geq 0$$

Bei **kombinatorischen Optimierungsproblemen** ist der Definitionsbereich diskret, z.B. $D \subseteq \mathbb{Z}^n \subset \mathbb{R}^n$. Ein kombinatorisches Optimierungsproblem lässt sich somit folgendermaßen formulieren:

$$\min f(x) \text{ mit } x \in \mathbb{Z}^n.$$

1.3 Zusammenfassung

Es gibt eine Vielzahl verschiedener Typen von Optimierungsverfahren, die sich nach Art der Zielfunktion (linear/nichtlinear), anhand der Definitionsmenge der Zielfunktion (restringiert/unrestringiert) und aufgrund des Suchtyps (global/lokal) unterscheiden. Die grundlegende Formulierung solcher Probleme wurde in diesem Kapitel aufgeführt. Das Studium der Literatur zeigt, dass lineare Optimierungsprobleme effizient behandelbar sind [BOR01] und somit sollen sie kein Gegenstand der weiteren Betrachtung sein. Der Fokus richtet sich nun auf globale und lokale unrestringierte nichtlineare Optimierungsprobleme. Der folgende Abschnitt zeigt gänginge Optimierungsverfahren für diese Klasse von Problemstellungen. Die in Kapitel 6 untersuchten Optimierungsprobleme sind allesamt nichtlinear, unrestringiert und global formuliert.

2 Numerische Verfahren zur globalen und lokalen nichtlinearen Optimierung

Es gibt eine Vielzahl an (numerischen) Verfahren zur Lösung von Optimierungsproblemen (hier: Minimierungsproblemen). Deshalb wurden lediglich die für diese Arbeit als zentral erachteten Verfahren ausgewählt und sollen im folgenden kurz beschrieben werden. Für eine tiefergehende Beschreibung dieser Verfahren sei auf die angegebene Literatur verwiesen. Ziel der Arbeit ist es, unrestringierte globale Optimierungsprobleme mit Hilfe stochastischer Konzepte zu lösen.

Als Basisverfahren hierfür wurde das Nelder/Mead-Verfahren ausgewählt. Deswegen soll dieses Verfahren später im selbständigen Kapitel 3 ausführlich beschrieben werden. Das Nelder/Mead-Verfahren ist ein heuristisches Verfahren. Darum sollen hier weitere Vertreter der Klasse heuristischer Optimierungsverfahren beschrieben werden. Im Vordergrund steht dabei die Herangehensweise an Optimierungsprobleme. Weiter ist das Nelder/Mead-Verfahren ein ableitungsfreies Verfahren. Aber es gibt auch andere ableitungsfreie Verfahren; hier soll ein Mutation-Selektions-Verfahren gezeigt werden. Dies geschieht vor dem Hintergrund, dass das genannte Verfahren stochastische Komponenten enthält, indem neue Punkte zufällig generiert werden. Die Idee der Generierung zufälliger Punkte wird bei der Erweiterung des Nelder/Mead-Verfahrens im Kapitel 5 benötigt. Das Nelder/Mead-Verfahren nutzt zur Optimierung die Techniken der Liniensuche und Clusterbildung. Diese beiden Techniken werden hier ebenfalls beschrieben. Zuletzt weist das Nelder/Mead-Verfahren lokale Eigenschaften auf, die dazu führen, dass das Verfahren gegen lokale Optimalpunkte konvergiert. Exemplarisch soll durch die Beschreibung des Gradientenverfahrens und des Newton-Verfahrens anhand numerischer Ergebnisse die Wirkungsweise lokaler Optimierungsverfahren, die meist nicht ableitungsfrei sind, dem Nelder/Mead-Verfahren gegenüber gestellt werden.

2.1 Heuristiken in der Optimierung

2.1.1 Bergsteigeralgorithmus

Der Bergsteigeralgorithmus ist ein Suchalorithmus mit einer intuitiven, allgemeinen und einfachen Idee. Er besteht aus einer Schleife in der der Punkt, der den besten Wert der bisher ermittelten Werte hat, einen potentiellen Nachfolgepunkt generiert. Wenn dieser neue Punkt besser ist als sein Vorgänger, wird dieser mit dem neuen Punkt ersetzt. Dann beginnt die Schleife von neuem. Das Problem vom Bergsteigeralgorithmus ist das Verfangen in lokalen Optima und die Technik des Generierens eines neuen

Punktes ist aufgrund seiner Allgemeinheit nicht vorgegeben. Zur Vermeidung der genannten Probleme gibt es verschiedene Lösungsansätze, wie z.B. per Zufallsgenerator nach einer bestimmten Anzahl von Durchläufen komplett von vorne zu beginnen [WEI08].

1. Man wähle ein $x \in \mathbb{R}^n$.
2. Man bestimme ein $y \in \mathbb{R}^n$.
3. Ist $f(y) \leq f(x)$, setze $x := y$.
4. Falls ein vorgegebenes Abbruchkriterium erfüllt ist, wird das Verfahren beendet, ansonsten mit Schritt 2 fortgesetzt.

2.1.2 Sintflutalgorithmus

Ein weiteres heuristisches Optimierungsverfahren ist der Sintflutalgorithmus [@CBO]. Die Idee ist, eine zufällige Suche im Suchraum durch einen sich ändernden Akzeptanzwert mit der Zeit einzuschränken. Dies ist vergleichbar mit einem Gefäß, in das in diskreten Abständen eine bestimmte Menge Wasser eingefüllt wird und sich so der Wasserspiegel erhöht. Dazu werden ein erster Schwellenwert (Wasserstand) und eine Konstante (Wassermenge, die eingefüllt wird) festgelegt. Von einem zufälligen Startwert ausgehend wird nun iterativ ein neuer Punkt im Suchraum erzeugt und genau dann akzeptiert, wenn der Funktionswert an diesem Punkt oberhalb des Schwellenwertes liegt. Der Schwellenwert wird dabei regelmäßig um die Konstante erhöht. Bildlich verkleinern sich dadurch akzeptierbare Bereiche des Suchraums, so dass der Algorithmus anfänglich lokale Optima überwinden kann. Das Problem liegt auch hier in der allgemeinen Definition. Denn es ist nicht festzulegen, wie neue je nach Schwellenwert zulässige Punkte im damit eingeschränkten Definitionsbereich der Funktion gefunden werden können. Es sei denn, es wird die Umgebung des aktuellen Punktes untersucht. Mit der Zeit geht der Algorithmus dann jedoch in einen Bergsteigeralgorithmus über, der von zufälligen Startpunkten ausgeht und einfach nur den besten Punkt in dessen Umgebung sucht. Das hier beschriebene Vorgehen sucht nach Maxima einer Funktion. Wie dies für die Minimussuche äquivalent umformuliert werden kann, wurde bereits im ersten Kapitel erläutert und soll in folgender Darstellung der Einzelschritte verwendet werden.

1. Man wähle ein $x \in \mathbb{R}^n$ und eine Konstante $c \in \mathbb{R}$. Setze $D = \mathbb{R}^n$ und $d := f(x)$.
2. Setze $D := \{x \in \mathbb{R}^n | \ f(x) \leq d\}$. Man bestimme ein $y \in \mathbb{R}^n$, idealerweise aus D.
3. Ist $f(y) \leq d$, setze $x := y$ und $d := f(x)$, sonst $d := f(x) - c$.
4. Falls ein vorgegebenes Abbruchkriterium erfüllt ist, wird das Verfahren beendet, ansonsten mit Schritt 2 fortgesetzt.

2.1.3 Simulierte Abkühlung

Ein weiteres heuristisches Optimierungsverfahren ist die Simulierte Abkühlung. Der Algorithmus bildet einen Abkühlungsprozess nach, d.h. nach Erhitzen eines Metalls sorgt die langsame Abkühlung dafür, dass die Molekühle ausreichend Zeit haben, sich zu ordnen und stabile Kristalle zu bilden. Dadurch wird ein energiearmer Zustand nahe am Optimum erreicht. Die Temperatur ist hierbei eine Akzeptanzschwelle unter der sich ein Zwischenergebnis auch verschlechtern darf. Der Vorteil dieses Verfahrens liegt darin, dass ein lokales Optimum wieder verlassen werden kann [@CBO]. Ein mathematisches Modell im Sinne von einem Algorithmus zur Lösung eines Optimierungsproblems sieht in etwa folgendermaßen aus:

1. Man wähle ein $x \in \mathbb{R}^n$ und eine monoton fallende Folge $(T_t)_{t\in\mathbb{N}}$ mit $T_t \overset{n\to\infty}{\longrightarrow} 0$, sowie eine Folge $(N_t)_{t\in\mathbb{N}}$. Setze $t := 0,\ i := 1$.

2. Ist $i \leq N_t$, so bestimme man $y \in U_\delta(x)$, sonst setze $t = t + 1,\ i := 1$.

3. Ist $f(y) \leq f(x)$, setze $x := y$, sonst $f(y) > f(x)$ setze $x := y$ mit einer bestimmten Wahrscheinlichkeit, wobei dafür eine Wahrscheinlichkeitsdichte auf Basis einer Exponentialfunktion mit einem sich über die Zeit ändernden, verschärfenden Parameter gegeben ist.

4. Falls ein vorgegebenes Abbruchkriterium erfüllt ist, wird das Verfahren beendet, ansonsten mit Schritt 2 fortgesetzt.

2.1.4 Schwellenakzeptanz

Auch die Schwellenakzeptanz ist ein heuristischer Optimierungsalgorithmus. Der Unterschied zur Simulierten Abkühlung liegt in der Behandlung von Verschlechterungen der gefundenen Konfiguration [@CBO]. Während die Simulierte Abkühlung Verschlechterungen in der Bewertung eines Zwischenergebnisses nur mit einer bestimmten im Verlauf der Optimierung kleiner werdenden Wahrscheinlichkeit akzeptiert, akzeptiert die Schwellenakzeptanz dies stets, sofern die Verschlechterung nicht größer als ein vorgegebener Schwellenwert $c \in \mathbb{R}$ ist. Dieser Schwellenwert wird im Verlauf der Optimierung ebenfalls gesenkt.

Vorteile der Schwellenakzeptanz gegenüber der Simulierten Abkühlung ergeben sich deshalb vor allem im Hinblick auf die Laufzeit. Die Schwellenakzeptanz verzichtet auf die Ermittlung einer Zufallszahl und die aufwändige Berechnung der Exponentialfunktion und kann daher unterschiedliche Konfigurationen schneller durchprobieren. Der Unterschied zur Simulierten Abkühlung besteht im zweiten Teil von Schritt 3:

3. Schwellenakzeptanz im Falle $f(y) > f(x)$: Der Punkt y wird stets angenommen, falls $|f(y) - f(x)| < c$ gilt.

2.2 Ableitungsfreie Optimierungsverfahren

2.2.1 Idee ableitungsfreier (globaler) Verfahren

Oftmals ist es gar nicht oder nur schwer möglich, Ableitungen einer Funktion zu bestimmen. Das liegt beispielsweise daran, dass die Funktion nicht überall differenzierbar oder nicht stetig ist.

2.2.2 Mutations-Selektions-Verfahren

Das Mutations-Selektions-Verfahren ist ein ableitungsfreies Verfahren. Hierbei führt man zufällige Mutationen des aktuellen Iterationspunktes durch und selektiert die brauchbaren Mutationen. Das Grundprinzip ist wie folgt [ALT02]:

1. Startpunkt $x_0 \in \mathbb{R}^n$ wählen und $k = 0$ setzen.
2. Mutation: neuen Punkt v_k durch zufällige Änderung von x_k berechnen.
3. Selektion: wenn $f(v_k) < f(x_k)$, dann $x_{k+1} = v_k$ setzen, andernfalls $x_{k+1} = x_k$ setzen.
4. k inkrementieren und zu 2. gehen.

Es gibt verschiedene Möglichkeiten, eine Mutation von x_k zu definieren. Eine häufig benutzte Mutation ist

$$v_{ki} = x_{ki} + \sigma_k(r_{ki} - 0.5), \quad i = 1, ..., n,$$

wobei $r_k \in \mathbb{R}^n$ ein Zufallsvektor mit Komponenten aus dem Intervall [0,1] stammt. Die Suchrichtung ist der Vektor d_k mit den Komponenten

$$d_{ki} = r_{ki} - 0,5 \in [-0.5, 0.5], \quad i = 1, ..., n,$$

Die Schrittweite σ_k sollte man umso kleiner wählen, je näher man an einem Minimum ist [ALT02]. Das Verfahren ist in der Grundversion einfach zu implementieren. Es fehlt allerdings eine systematische Vorgehensweise zur Anpassung der Schrittweite. Ebenso fehlt ein Abbruchkriterium.

2.2.3 Stochastische Ansätze in der Optimierung

2.2.3.1 Metropolisalgorithmus

Der Metropolis-Algorithmus kann Zustände für eine beliebige Wahrscheinlichkeitsverteilung erzeugen [@WIM]. Voraussetzung ist lediglich, dass die Dichte an jedem Ort berechnet werden kann. Der Algorithmus benutzt eine vorgeschlagene Dichte, die vom derzeitigen Ort und dem möglichem nächsten Ort abhängt.

1. Wähle $x \in \mathbb{R}^n$.
2. Ändere x zu y nach Vorgabe.
3. Selektiere x, falls $f(x) \in f(y)$ und y, falls $f(y) \in f(x)$ auf Basis einer bestimmten Wahrscheinlichkeit.

2.2.3.2 Stochastisches Tunneln

Beim Stochastischen Tunneln wird die untersuchte Funktion abgetastet, indem zufällig von der aktuellen Lösung zu einer anderen Lösung gesprungen wird. Für die Berechnung der Wahrscheinlichkeiten wird meist das Metropolis-Kriterium gewählt, wobei der Parameter passend gewählt werden muss. Das Selektionskriterium im Schritt 3 ist der grundsätzliche Unterschied zum Metropolis-Algorithmus.

2.3 Techniken in der Optimierung

2.3.1 Liniensuche

Die Suche nach einem neuen Punkt $y \in \mathbb{R}^n$ bei den ableitungsfreien Verfahren erscheint oft willkürlich zu sein. Es wird eine optimale Schrittweite λ bestimmt [BOR01]. Ausgehend von einem Ausgangspunkt x_k und der Bewegungsrichtung d_k entsteht dann bei der Liniensuche ein neuer Iterationspunkt $x_{k+1} = x_k + \lambda \cdot d_k$. Für die Optimierung der Funktion gilt dann:

$$\text{min } f(x_k + \lambda \cdot d_k) \quad \text{ mit } \lambda \in [a, b], a, b \in \mathbb{R}$$

Hier kann die Technik der Liniensuche eine strukturierte Vorgehensweise ermöglichen. Ist nämlich geklärt, in welcher Richtung d_k von x_k aus nach einer neuen Funktion gesucht werden soll, kann ein Punkt $y \in \mathbb{R}^n$, der auf dieser Richtung entsprechend liegt, vorgegeben und die „Linie“ $x + \lambda(y - x)$, $\lambda \in [a, b]$ nach einem neuen Punkt abgesucht werden. Auf Basis der Punkte x, y entsteht eine eindimensionale Funktion $\hat{f}(\lambda) = f(x + \lambda(y - x))$. Ist $\lambda \in [0, 1]$ vorgegeben, wird auf der Strecke zwischen x und y gesucht, was im Kapitel 5 benutzt wird.

2.3.2 Direkte Suchmethoden für den Minimalpunkt

Zur Minimierung einer eindimensionalen Funktion $f : [a, b] \to \mathbb{R}$ gibt es auch direkte Suchmethoden, wie beispielsweise das Bisektionsverfahren, das Newton-Verfahren, die Methode des Goldenen Schnitts und das Gradientenverfahren zur Liniensuche.

Bisektionsverfahren

Wird das Bisektionsverfahren zur Suche eines Minimums eingesetzt, kann dies als Suche nach der Nullstelle der ersten Ableitung der Funktion f interpretiert werden. Gegeben ist das Intervall $[a, b]$ wobei

die Ableitung $f'(a) < 0$ und die Ableitung $f'(b) > 0$ ist. Dies bedeutet aufgrund der Stetigkeit der Funktion, dass die erste Ableitung im angegebenen Intervall mindestens einen Nulldurchgang hat.

Das Vorgehen kann wie folgt beschrieben werden [RIA05]: Initialisiere $a_1 = a$ und $b_1 = b$ und berechne $y_1 = f'(\frac{a_1+b_1}{2})$. Dann gilt:

$$\begin{array}{llll} \text{wenn} & y_1{>}0 & \text{setzte } a_2 = a_1 & \text{und} \quad b_2 = \frac{a_1+b_1}{2} \\ \text{wenn} & y_1{<}0 & \text{setzte } a_2 = \frac{a_1+b_1}{2} & \text{und} \quad b_2 = b_1 \end{array}$$

Nun ist $f'(a_2) < 0$ und $f'(b_2) > 0$ und die Intervalllänge halbiert. Danach wird der Funktionswert $y_2 = f'(\frac{a_2+b_2}{2})$ berechnet und das Verfahren wiederholt.

Newton-Verfahren

Das Newton-Verfahren ist ein schnell konvergierendes Verfahren zur Suche einer Nullstelle der ersten Ableitung [RIA05]. Die Funktion f wird im aktuellen Näherungspunkt x^k durch eine quadratische Funktion approximiert. Es gilt

$$q(x) = f(x_k) + (x - x_k) \cdot f'(x_k) + \tfrac{1}{2}(x - x_k)^2 \cdot f''(x_k)$$

mit den ersten und zweiten Ableitungen der Funktion f im Punkt x_k. Bildet man die Iterationsvorschrift

$$x_{k+1} = x_k - [f''(x_k)]^{-1} \cdot f'(x_k) \qquad \text{für } k = 1, 2, \cdots,$$

so konvergiert dann gegen ein $\bar{x} \in \mathbb{R}$, wenn $f(\bar{x}) = 0$ und $f''(\bar{x}) > 0$ und $x_k \in U(\bar{x})$.

Goldener Schnitt

Bei der Methode des Goldenen Schnitts erhält man für $\lambda < \mu \in [a, b]$ durch Vergleich von $f(\mu)$ und $f(\lambda)$ die Information darüber, ob die Lösung links von λ oder rechts von μ liegt (siehe Abbildung 2.1).

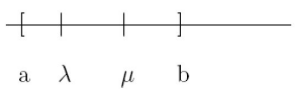

Abbildung 2.1: Goldener Schnitt, [JAN03]

Dann wird auf dem Intervall $[a, \mu]$ bzw. $[\lambda, b]$ weitergesucht [JAN03]. Der goldene Schnitt für das Suchintervall $[a, b]$ ist für $[0, 1]$ wie folgt definiert:

$$\alpha = \frac{-1+\sqrt{5}}{2} \approx 0,618$$

wobei $\lambda = \alpha^2$ und $\mu = \alpha$ ist (siehe Abbildung 2.2). Das Intervall verkleinert sich in beiden Fällen um den Faktor α, weil $\alpha = 1 - \alpha^2$.

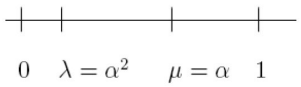

Abbildung 2.2: Goldener Schnitt, [JAN03]

Wenn $f(\mu) > f(\lambda)$ gilt, dann ist die Stelle λ identisch mit der neuen Stelle μ'. Bei $f(\mu) < f(\lambda)$ ist die Stelle μ identisch mit der neuen Stelle λ', d.h., dass eine der alten Funktionsauswertungen $f(\lambda)$ bzw. $f(\mu)$ in der nächsten Iteration mitverwendet werden kann.

Gradientenverfahren zur Liniensuche

Für die Liniensuche wird oft auch das Gradientenverfahren verwendet:

$$x^{k+1} = x^k + \alpha_k \nabla f(x_k)$$

Die nächste Iteration wird um α_k berechnet, um den kleinsten Wert der Linie zu finden. Der Algorithmus konvergiert zwar sehr langsam, ist aber robust. Abbildung 2.3 zeigt ein Beispiel des Gradientenverfahrens, bei dem das Ergebnis nach vier Schritten ermittelt wird.

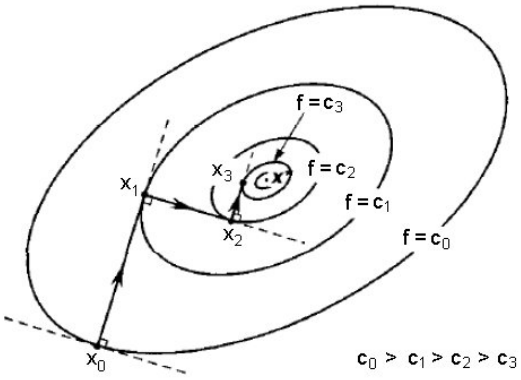

Abbildung 2.3: Gradientenverfahren zur Liniensuche, [GER07]

2.3.3 Liniensuche durch Kurvenanpassung

Das Rechnen mit der Liniensuche durch Kurvenanpassung nutzt möglichst viele Informationen aus den bisher bereits vorliegenden Iterationspunkten aus, um den Optimierungsprozess zu verbessern [BOR01]. Dies soll dazu dienen eine lokale Approximation mit Hilfe einer einfach handhabbaren Funktion zu gewinnen. Das ergibt dann eine Minimalstelle, die als vorläufiger Optimalpunkt verwendet wird.

Newton-Verfahren zur Kurvenanpassung

Die zu minimierende Funktion f muss zweimal differenzierbar sein. Ausgegangen wird von einem Iterationspunkt x_k mit den Funktionen $f(x_k)$, $f'(x_k)$ und $f''(x_k)$. Hieraus läßt sich eine quadratische Funktion $q(x)$ gewinnen, die bis zur zweiten Ableitung mit f übereinstimmt:

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \tfrac{1}{2} \cdot f''(x_k)(x - x_k)^2$$

Zur Ermittlung des Optimums wird die Funktion $q(x)$ abgeleitet [BOR01]. Dies führt zum bekannten Nullstellenverfahren mit der Iterationsvorschrift:

$$x_{k+1} = x_k - \tfrac{g(x_k)}{g'(x_k)}$$

In der Abbildung 2.4 erkennt man das Prinzip des Verfahrens. Der Punkt x_k ist die Schnittstelle der Funktion $f(x)$ mit der quadratischen Funktion $q(x)$. Der Iterationspunkt x_{k+1} ist die Minimalstelle von $q(x)$ und das Optimum von $f(x)$ ist durch den Punkt $\hat{x}$ gekennzeichnet.

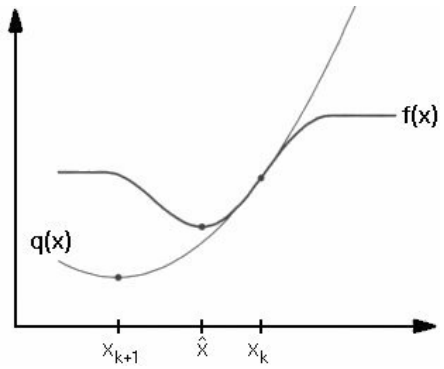

Abbildung 2.4: Newton-Verfahren zur Kurvenanpassung, [BOR01]

Approximation mit Regula Falsi

Im Gegensatz zum Newton-Verfahren werden für den Regula Falsi keine zwei Ableitungen benötigt, sondern erste Ableitungen an zwei verschiedenen Stellen [BOR01]. Es wird die folgende Funktion $q(x)$ gebildet:

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} \cdot \frac{f'(x_{k-1}) - f'(x_k)}{x_{k-1} - x_k}$$

Daraus ergibt sich folgende Iterationsvorschrift:

$$x_{k+1} = x_k - f'(x_k) \left[\frac{x_{k-1} - x_k}{f'(x_{k-1}) - f'(x_k)} \right]$$

Abbildung 2.5 zeigt die beiden Iterationspunkte x_{k+1} und x_{k-1} und das ermittelte Optimum $\hat{x}$.

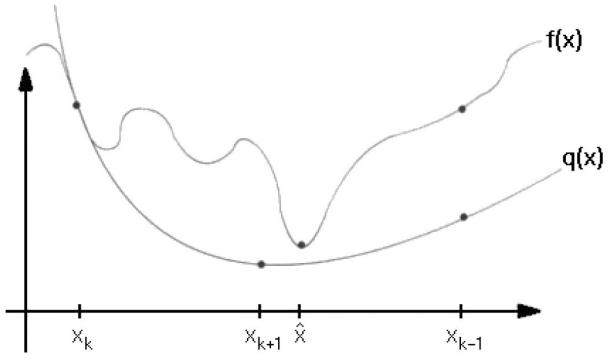

Abbildung 2.5: Approximation mit Regula Falsi, [BOR01]

Allgemeine quadratische Approximation

Beim Rechnen mit Kurvenapproximationen gibt es auch Methoden die Optima zu ermitteln ohne dass Ableitungen der Funktion notwendig sind. Hierzu seien drei Punkte mit ihren Funktionsauswertungen gegeben: $x_1 < x_2 < x_3$ mit $f_1{=}f(x_1)$, $f_2{=}f(x_2)$ und $f_3{=}f(x_3)$. Je nach Lage der Funktionswerte sind folgende Fälle zu unterscheiden:

- Fall (1): $f_1 > f_2 > f_3$
- Fall (2): $f_1 < f_2 < f_3$
- Fall (3): $f_1 > f_2 < f_3$
- Fall (4): $f_1 < f_2 > f_3$

Die Fälle werden in der Abbildung 2.6 nebeneinander aufgezeigt.

Wie zu sehen ist, gibt es im Fall (3) ein Minimum in $[x_1,x_3]$, falls f stetig bzw. quasikonvex ist. Entsprechend liegt das Maximum im Fall (4) im Intervall $[x_1,x_3]$. In den Fällen (1) und (2) sind die Funktionen möglicherweise monoton fallend, so dass in diesem Fall das Minimum in der Umgebung um x_3 liegt, bzw. wachsend mit Minimum in der Umgebung von x_1. Durch Verfeinerung kann die Annahme überprüft und gegebenenfalls korrigiert werden.

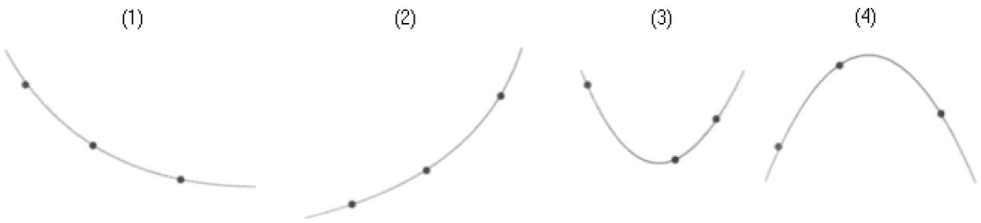

Abbildung 2.6: Vier Fälle bei Regula Falsi, [BOR01]

Zur Annäherung an die Optima durch einen Verfeinerungsalgorithmus gibt es das **Verfahren von Swan** [BOR01]. Ausgegangen wird hierbei vom Fall (3). Zuerst werden a, $a + \lambda$, $a - \lambda$ und $f(a)$, $f(a + \lambda)$, $f(a - \lambda)$ berechnet. Ein Tripel ist gefunden, wenn gilt, dass $f(a - \lambda) > f(a) < f(a + \lambda)$ ist. Daraufhin kann $\min\{f(a - \lambda), f(a + \lambda)\}$ bestimmt werden.

2.3.4 Clusterbildung

Es gibt Optimierungsverfahren, die clustern, wie z.B. durch Liniensuche gewonnene lokale Optimalpunkte, um daraus das globale Optimum zu bestimmen. Die Punkte werden dann zu **Punktewolken** zusammengefasst. Auch das Nelder/Mead-Verfahren (vgl. Kapitel 3) hat ein Clustering implementiert, indem es Punkte mit niedrigem Funktionswert zusammenfasst.

Bevor geclustert wird, sind Entscheidungen zu treffen. Diese betreffen die Wahl der Anzahl der Cluster, die Wahl des Ähnlichkeits- bzw. Distanzmaßes und die Wahl des Clusterverfahrens [GOE06].

2.3.4.1 Anzahl von Clustern

Für die Wahl der Anzahl der Cluster kann die Kennzahl **STRESS** (STandardized REsidual Sum of Squares) verwendet werden. Damit wird die Diskrepanz zwischen Disparitäten und Distanzen minimiert [ZIM03]. So steht d_{ij} für die Distanz zwischen den Objekten i und j und $\tilde{d}_{ij}$ für die Disparität zwischen den Objekten i und j.

$$STRESS(q,p) = \sqrt{\frac{\sum_i \sum_{j>i} (d(q,p)_{ij} - \tilde{d}(q,p)_{ij})^2}{\sum_i \sum_{j>i} d(q,p)_{ij}^2}}$$

Wenn die Kennzahl STRESS ermittelt ist, kann bei einer größeren Dimensionszahl ein **Ellenbogentest** (auch Scree-Test genannt) zur Bestimmung der Dimensionszahl durchgeführt werden [BAC94]. Dazu werden die STRESS-Koeffizienten in Abhängigkeit von der Dimensionszahl graphisch dargestellt. Die Verlaufskurve zeigt einen ellenbogenförmigen Knick, der um so stärker ausgeprägt ist, je geringer das Fehlerausmaß ist. In dem fiktiven Beispiel der Abbildung 2.7 ist die Dimensionszahl, die zur Berechnung am geeignetsten ist, die 4, weil hier ein ellenbogenförmiger Knick auftritt.

Bei Optimierungsproblemen wird häufig mit zwei Clustern gearbeitet.

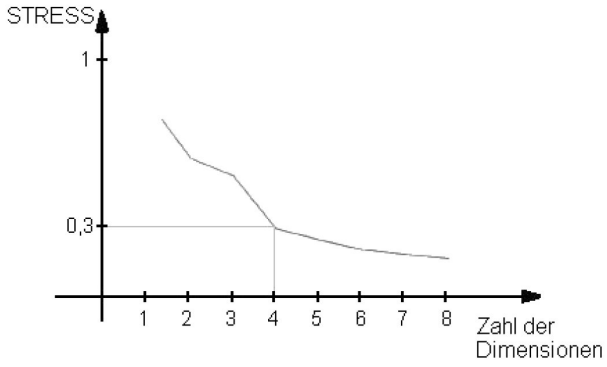

Abbildung 2.7: Ellenbogenkriterium

2.3.4.2 Auswahl eines Ähnlichkeits- bzw. Distanzmaßes

Wenn zwei Objekte (i und j) unterteilt vorliegen, muss zunächst festgelegt werden, wann sich diese Objekte ähnlich oder unähnlich sind [VOL00]. Zu diesem Zweck verwendet man Proximitätsmaße, die die Ähnlichkeit oder die Verschiedenheit von zwei Objekten zueinander angeben. Grundsätzlich lassen sich zwei Arten von Proximitätsmaßen unterscheiden:

1. **Ähnlichkeitsmaße** geben die Ähnlichkeit zwischen zwei Objekten an. Je größer der Wert eines Ähnlichkeitsmaßes, desto ähnlicher sind sich die beiden Objekte.

2. **Distanzmaße** messen die Unähnlichkeit zwischen zwei Objekten. Je größer der Wert des Distanzmaßes, desto unähnlicher sind sich die beiden Objekte

In der Regel wird in einer objektorientierten Clusteranalyse wegen der nachteiligen Effekte von Korrelationsmaßen mit einem Distanzmaß und nicht mit einem Ähnlichkeitsmaß gearbeitet [BAC94]. Deshalb werden im Weiteren nur Distanzmaße betrachtet. Die Kenngröße zur Ermittlung des Distanzmaßes ist die **Minkowski-Metrik**:

$$d(q,r)_{ij} = [\sum_{k=1}^{q} |x_{ik} - x_{jk}|^r]^{1/r}$$

Die Variablen q und r sind Metrikparameter, wobei q für die Anzahl der Dimensionen steht und r für die gewichteten Unterschiede der einzelnen Variablen [BAC94]. Wenn r einen hohen Wert hat, dann werden größere Unterschiede stärker gewichtet als kleinere Unterschiede, auch wenn die größeren Unterschiede nur in weniger Variablen sind. Im Folgenden werden die gebräuchlichsten Distanzmaße betrachtet, wobei d für die Distanz der Punkte i und j steht, und die beiden Objekte x_{ik} und x_{jk} die Koordinaten der Punkte i und j sind.

City-Block-Metrik

Die City-Block-Metrik ergibt sich aus der verallgemeinerten Minkowski-Metrik, wenn r=q=1 gesetzt wird.

$$CITY_{i,j} = d(q = 1, r = 1) = d_{ij} = \sum_{k=1}^{1} |x_{ik} - x_{jk}|$$

Es wird also die Distanz zwischen zwei Punkten als Summe der Abstände zwischen diesen Punkten gemessen.

Euklidische Distanz

Bei der Euklidischen Distanz ist die Quadratwurzel der Summe der quadrierten Differenzen der beiden Objekte x_{ik} und x_{jk} zu bestimmen [BAC94].

$$EUKLID_{i,j} = d(q = 2, r = 2) = d_{ij} = \sqrt{\sum_{k=1}^{2} (x_{ik} - x_{jk})^2}$$

Wenn die Euklidische Distanz für $q = 2$ berechnet wird, dann gleicht sie dem Satz von Pythagoras.

Quadrierte Euklidische Distanz

Die Quadrierte Euklidische Distanz ist die Summe der quadrierten Differenzen der beiden Objekte.

$$QEUKLID_{i,j} = d(q = 1, r = 2) = d_{ij} = \sum_{k=1}^{1} (x_{ik} - x_{jk})^2$$

Chebyshev

Das Distanzmaß Chebyshev wird auch Maximumsnorm genannt und ist die größte absolute Differenz der beiden Objekte.

$$CHEBYSHEV_{i,j} = d(q = \infty, r = \infty) = d_{ij} = \max_k |x_{ik} - x_{jk}|$$

Der Unterschied der Distanzmaße wird schematisch in Abbildung 2.8 mit der Distanz zwischen den Punkten x und y verdeutlicht:

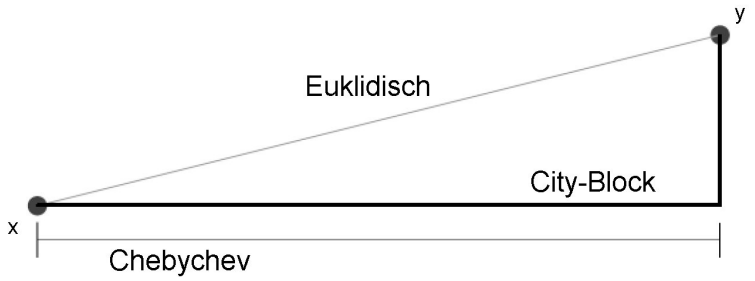

Abbildung 2.8: Distanzmaße im Vergleich

2.3.4.3 Clusterverfahren

Um aufzuzeigen, welches Clusterverfahren gewählt werden kann, werden im folgenden die gebräuchlichsten Clusterverfahren angesprochen. Für eine tiefergehende Beschäftigung sei auf die entsprechend angegebene Literatur verwiesen.

Modellbasierte Methoden
Unter Annahme einer bestimmten Verteilung und Anzahl von Clustern wird die Zugehörigkeit zu einem Cluster mittels einer Wahrscheinlichkeit bestimmt [THE04].

Hierarchische Clusterverfahren
Hierarchische Cluster-Verfahren operieren entweder top-down (divisiv) oder bottom-up (agglomerativ). Beim top-down Vorgehen wird mit einem großen Cluster gestartet und teilt diesen immer weiter auf, bis jedes Element einem Cluster zugeordnet ist. Beim bottom-up Vorgehen startet man hingegen mit einem Cluster pro Element und fügt so lange Cluster zusammen bis wieder ein allgemeines Cluster entsteht [HOT04].

Fuzzy-Clustering
Objektmengen können Elemente enthalten, die aufgrund mehrdeutiger Datenbefunde für keines der ermittelten Cluster prototypisch sind und mehreren Clustern zugeordnet werden könnten. Beim Fuzzy-Clustering werden Objekte unscharf auf Cluster verteilt, das vom bestimmten Zugehörigkeitsgrad abhängt. Im Spezialfall der Zugehörigkeit $= 1$ bzw. der Zugehörigkeit $= 0$ ist das Element einem Cluster vollständig bzw. überhaupt nicht zugehörig. Das Fuzzy Clustering fordert also keine hundertprozentige Zugehörigkeit der Objekte zu einem bestimmten Cluster, sondern erlaubt die Zugehörigkeit von α_i Prozent [THE04].

Expectation Maximization
Das Expectation Maximization ist ein probabilistisches Clusterverfahren [HOT04]. Im Gegensatz zu den meisten vorhergehenden Verfahren wird ein Element nicht deterministisch einem Cluster zugewiesen, sondern zu jedem Cluster die Wahrscheinlichkeit angegeben, dass ein Objekt dazugehört. Dabei soll eine gegebene Wahrscheinlichkeitsverteilung approximiert werden. Im so genannten Expectation-Schritt werden die Wahrscheinlichkeiten berechnet, im Maximization-Schritt wird dann die Wahrscheinlichkeitsverteilung selbst neu berechnet.

Partitionierungsverfahren
Beim Partitionierungsverfahren wird eine bestimmte Menge der Cluster vorgegeben und das Verfahren verteilt dann die Fälle solange auf die verschiedenen Cluster, bis ein Optimierungskriterium erfüllt ist.

2.4 Unrestringierte lokale Optimierungsverfahren

In diesem Abschnitt werden Verfahren vorgestellt, die eine Funktion ohne Nebenbedingung lokal minimieren.

2.4.1 Gradientenverfahren

Das Gradientenverfahren wird auch als **Methode des steilsten Abstiegs** bezeichnet. Ausgegangen wird von der Funktion $f : \mathbb{R}^n \to \mathbb{R}$, die stetige partielle Ableitungen besitzt. Diese Ableitungen werden zu einem Zeilenvektor $\nabla f(x)$ zusammengefasst, dem so genannten Gradienten. Der Spaltenvektor sei wie folgt definiert:

$$g(x) := \nabla f(x)^T$$

Das Verfahren ist durch die Iterationen

$$x_{k+1} = x_k - \alpha_k g(x_k)$$

definiert, wobei α_k die nichtnegative reele Zahl ist, welche die Funktion

$$f(x_k - \alpha g(x_k))$$

minimiert [RUD08]. Für gegebenes x_k und den Gradienten $g(x_k)$ an der Stelle x_k wird somit eine geeignete Abstiegsrichtung gesucht:

$$\min_{0 \leq \alpha < \infty} f(x_k - \alpha g(x_k))$$

2.4.2 Newton-Verfahren

Gegeben ist eine Abbildung $f : \mathbb{R}^n \to \mathbb{R}$, die zweimal stetig differenzierbar ist. Bei $\bar{x} \in \mathbb{R}^n$ existiere die Inverse $H_f^{-1}(\bar{x})$ der Hesse-Matrix und es sei $\nabla f(\bar{x}) = 0$. Liegt dann x_0 nahe genug bei $\bar{x}$, so konvergiert das iterative Verfahren $x_{k+1} := x_k - H_f^{-1}(x_k)\nabla f(x_k)$ nach [BOR01] gegen $\bar{x}$. Das Verfahren ist Grundlage für viele Abstiegsverfahren zur Lösung eines unrestringierten Optimierungsproblems

$$\min f(x) \text{ mit } x \in \mathbb{R}^n.$$

Unter geeigneten Voraussetzungen sind die vom Newton-Verfahren berechneten Richtungen Abstiegsrichtungen [ALT02]. Im Folgenden werden die wichtigsten Resultate zu diesem Verfahren aufgezeigt. Beim Newton-Verfahren wählt man zunächst einen Startpunkt x_0. Die Richtung

$$d_k = -H_f^{-1}(x_k)\nabla f(x_k)$$

heißt Newton-Richtung [ALT02]. Ist $H_f(x_k)$ positiv definit, dann ist diese Richtung ein Spezialfall der Abstiegsrichtungen. Da beim Newton-Verfahren als Schrittweite immer 1 gewählt wird, erhält man in der Regel kein Abstiegsverfahren, es sei denn, der Startpunkt wird hinreichend nahe an der Lösung gewählt. Man kann aber durch Verwendung einer geeigneten Schrittweite σ_k das Newton-Verfahren durch die Iterationsvorschrift

$$x_{k+1} = x_k - \sigma_k H_f^{-1}(x_k)\nabla f(x_k)$$

so modifizieren, dass es ein Abstiegsverfahren bleibt, aber der Konvergenzradius größer wird.

Gemäß [BOR01] ist die Konvergenzordnung ≥ 2, weshalb das Verfahren gerne zur lokalen Optimierung eingesetzt wird.

2.5 Zusammenfassung

Unrestringierte nichtlineare Optimierungsprobleme **ohne Nebenbedingungen** treten oft zur Lösung komplexerer Probleme als Teilprobleme auf. Berechnet werden Näherungslösungen mittels numerischer Iterationsverfahren [DOM95]. Mit solchen Verfahren werden sukzessive Punkte $\hat{x}$ berechnet, die unter gewissen Voraussetzungen gegen eine Maximal- oder Minimalstelle der Zielfunktion f konvergieren. Unterschieden werden hierbei Problemstellungen mit nur einer oder mehreren Variablen. Bei nur **einer Variablen** wird das Problem entsprechend durch

$$\min f(x) \text{ mit } x \in \mathbb{R}$$

formuliert. Bei Problemstellungen mit **mehreren Variablen** ist die Formulierung am Anfang des Kapitels 1.1 angebracht.

Zur Ermittlung von Extremalstellen werden zunächst zwei verschiedene Verfahrensklassen betrachtet [WEB07]. Zum einen sind dies Verfahren, die keine Ableitungen von f benutzen, wie die Methode des goldenen Schnitts und das Fibonacci-Verfahren. Verfahren der zweiten Klasse verwenden (partielle) Ableitungen, die aber die Differenzierbarkeit der Zielfunktion voraussetzen, z.B. das Bisektionsverfahren, das Newton-Verfahren und das Sekanten-Verfahren.

Ziel der vorliegenden Arbeit ist es, ein nicht deterministisch ableitungsfreies Verfahren zu finden bzw. ein bestehendes Verfahren entsprechend zu erweitern. Das bedeutet, dass ein Algorithmus gefunden werden muss, der nicht bei jeder Ausführung mit gleichen Startbedingungen und Eingaben gleiche Ergebnisse liefert. Von den in den Kapiteln 2.1 bis 2.4 beschriebenen Verfahren der nichtlinearen Optimierung sind daher nur die stochastischen Methoden relevant.

Die Verwendung von Clustern kollidiert nicht mit dem Ziel dieser Arbeit, weil der Funktion keine Einschränkungen unterliegen, und kann somit verwendet werden.

Das folgende Kapitel 3 befasst sich ausführlich mit dem globalen Optimierungsverfahren nach Nelder/Mead.

3 Optimierungsverfahren von Nelder und Mead

3.1 Definition

Das Verfahren von Nelder und Mead ist ein iteratives Verfahren zur globalen Optimierung, das heutzutage oft Verwendung findet. Beispielsweise wird es als Standardsuchverfahren im Programm Mathematica mit der Funktion *NMinimize*, in Matlab mit *fminsearch* und im Statistik-Programm R mit *optim* verwendet. Vorgestellt wurde dieses Verfahren im Jahre 1962 von Nelder und Mead im Paper „A simplex method for function minimization“, [NEL65].

Das Verfahren benutzt einen Simplex als Gundstruktur, weshalb es oft als „Simplexverfahren von Nelder/Mead“ bezeichnet wird. Das Simplexverfahren der linearen Optimierung ist aber unabhängig davon [ALT02]. Der Nelder/Mead-Algorithmus zählt zu den wenigen Algorithmen, die ohne Ableitungen der Funktion nach den Parametern auskommen, die bei manchen Problemstellungen einfach nicht mit vertretbarem Aufwand berechenbar sind. Durch sinnvolle Vergleiche der Funktionswerte mehrerer Punkte im Parameterraum wird ähnlich wie bei einer Regula falsi mit Schrittweitensteuerung die Tendenz der Werte und der Gradient Richtung Optimum angenähert. Das Verfahren konvergiert langsam, ist aber dafür einfach und relativ robust [@WID]. Als weitere Vorteile sind die gute Konvergenz und Stabilität für die meisten Problemstellungen zu nennen. Außerdem benötigt der Algorithmus zur Errechnung des Optimums im Vergleich zu anderen Verfahren relativ wenig Funktionsauswertungen.

Im Grundmodell wird mit einem Startpunkt $x_(0,0) \in \mathbb{R}$ begonnen, von dem aus die Eckpunkte eines n-Simplesex mittels Einheitsvektor ermittelt werden. Von dem so entstandenen Simplex werden dann die Funktionswerte an den Eckpunkten verglichen. In den weiteren Schritten wird bei einer Minimumsuche der Eckpunkt mit dem höchsten Funktionswert in Richtung des Optimums verbessert. Nach dieser grundsätzlichen Idee wird die Suche so lange weitergeführt, bis ein vorzugebendes Konvergenzkriterium erfüllt ist. Der Simplex wird sich dabei in Richtung des Optimums verlagern und sich schließlich um dieses herum zusammenziehen.

3.2 Algorithmus

Das Nelder/Mead-Verfahren verwendet drei Konstruktionsprinzipien zur Bestimmung neuer Punkte: Reflektion, Expansion und Kontraktion. Die Parameter müssen für diese Prinzipien zu Beginn fest-

gelegt werden: $0 < \alpha < 1$ (Kontraktionskonstante), $\beta > 1$ (Expansionskonstante) und $0 < \gamma \leq 1$ (Reflektionskonstante).

Im nächsten Schritt wird ein Startpunkt $x_{(k,0)} \in \mathbb{R}^n$ festgelegt. Der für das Verfahren notwendige Simplex wird aus den $n+1$ unabhängigen Vektoren $x_{(k,0)}, ..., x_{(k,n)} \in \mathbb{R}^n$ generiert:

$$S_k = \left\{ \sum_{i=0}^{n} \lambda_i x_{(k,i)} \mid \lambda_i \geq 0, i = 0, ..., n, \sum_{i=0}^{n} \lambda_i = 1 \right\}$$

Die Ecken des Startsimplexes S_0 werden beispielsweise mit dem Einheitsvektor $e_j \in \mathbb{R}^n$ gebildet [ALT02]:

$$x_{(k,j)} = x_{(k,0)} + e_j \qquad j = 1, ..., n$$

In jedem Iterationsschritt, d.h. ab hier beginnt die Schleife, wird dann der Eckpunkt des Simplexes S_k ermittelt, an dem der Funktionswert am größten ist. Dazu wird der Punkt $x_{(k,m)}$ bestimmt, wobei beim ersten Durchlauf $k = 0$ gesetzt wird.

$$f(x_{(k,m)}) = \mathsf{max}\{f(k,0), ..., f(k,n)\}$$

Es wird somit ein partitionierendes Clustering der Ecken des Simplexes, wie in Kapitel 2.3 angesprochen, verwendet. Dabei werden zwei Cluster $C_1 = \{x_{(k,m)}\}$ und $C_2 = \{x_{(k,i)} | i = 0, ..., n,\ i \neq m\}$ gebildet.

Außerdem wird der Eckpunkt $x_{(k,l)}$ des Simplexes mit dem kleinsten Funktionswert berechnet:

$$f(x_{(k,l)}) = \mathsf{min}\{f(k,0), ..., f(k,n)\}$$

Für weitere Berechnungen im Nelder/Mead-Verfahren ist dann noch die Ermittlung des Schwerpunktes der Ecken bzgl. $x_{(k,m)}$ notwendig. Es wird der Clustermittelpunkt des Clusters C_2 gebildet.

$$s_{(k,m)} = \frac{1}{n} \sum_{i=0,\ i \neq m}^{n} x_{(k,i)}.$$

3.2.1 Reflektion

Die Reflektion wird von $x_{(k,m)}$ am Schwerpunkt $s_{(k,m)}$ durchgeführt, d.h. der Punkt

$$x_r = s_{(k,m)} + \gamma(s_{(k,m)} - x_{(k,m)})$$

mit der Reflektionskonstante $0 < \gamma < 1$ wird berechnet. Wenn für den sich daraus ergebenden Funktionswert von x_r gilt, dass

$$f(x_{(k,l)}) \leq f(x_r) \leq f(x_{(k,m)})$$

dann ersetzt man $x_{(k,m)}$ mit x_r, weil davon ausgegangen wird, dass diese Suchrichtung erfolgsversprechend ist. Die restlichen Ecken von S_k bleiben unverändert. Der Index k wird inkrementiert und der Algorithmus beginnt von vorne.

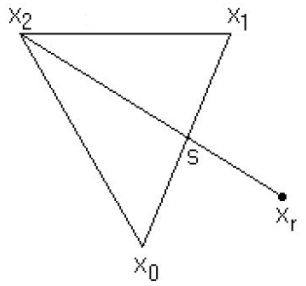

Abbildung 3.1: Reflektion x_r mit $\gamma{=}\frac{1}{2}$ und $j = 1$

3.2.2 Expansion

Wenn der Funktionswert von x_r einen noch besseren Wert erbracht hat, als der kleinste Funktionswert des Simplexes, wenn also gilt, dass

$$f(x_r) < f(x_{(k,l)})$$

dann wird eine Expansion durchgeführt.

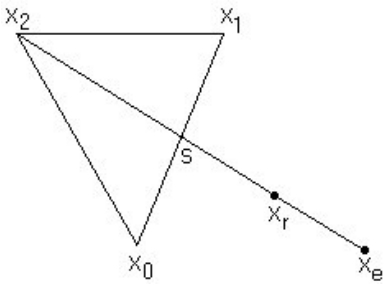

Abbildung 3.2: Expansion x_e mit $\beta{=}2$

Ausgehend von x_r in Richtung $x_r - s_{(k,m)}$ wird dann ein noch besserer Punkt gesucht, weil in dieser Richtung mehr Punkte mit einem noch niedrigeren Funktionswert vermutet werden. Für die Expansion berechnet man

$$x_e = s_{(k,m)} + \beta(x_r - s_{(k,m)})$$

mit der Expansionskonstanten $\beta > 1$. Dann wird $x_{(k,m)}$ durch den besseren der beiden Punkte x_r und x_e ersetzt, d.h.

$$x_{(k+1,m)} := \begin{cases} x_e, & \text{falls } f(x_e) < f(x_r) \\ x_r, & \text{falls } f(x_r) \leq f(x_e) \end{cases}$$

Die restlichen Ecken von S_k bleiben unverändert. Der nächste Iterationsschritt folgt mit einem neuen Simplex S_{k+1}.

3.2.3 Kontraktion innen

Ist der Funktionswert von x_r der größte aller bisher ermittelten Eckpunkte, also $f(x_r) \geq f(x_{(k,m)})$, dann geht man davon aus, dass man schon nahe am Optimum war und versucht deshalb zum Simplex mittels partieller innerer Kontraktion zurückzugehen [ALT02]. Dies wird von $x_{(k,m)}$ in Richtung $s_{(k,m)} - x_{(k,m)}$ durchgeführt, d.h. der neue Punkt ist

$$x_{ci} = s_{(k,m)} + \alpha(x_{(k,m)} - s_{(k,m)})$$

mit der Kontraktionskonstante $0 < \alpha < 1$. Man geht also vom schlechtesten Eckpunkt in Richtung des Eckenschwerpunktes. Wenn dieser neue Eckpunkt nicht schlechter als alle anderen verbleibenden Ecken ist, wird er beibehalten, d.h. der Simplex wurde kontrahiert. Die Schleife geht wieder zum Anfang (siehe Kapitel 3.2.1).

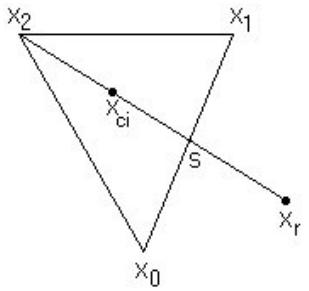

Abbildung 3.3: Kontraktion innen x_{ci} mit $\alpha{=}\frac{1}{2}$ und $j = 1$

3.2.4 Kontraktion außen

Gilt aber, dass $f(x_r) < f(x_{(k,m)})$, dann war es möglicherweise richtig, von $s_{(k,m)}$ in Richtung x_r zu gehen. Weil aber alle Ecken außer $x_{(k,m)}$ besser als x_r sind, sollte man wieder etwas näher an den Simplex gehen. Man führt eine partielle äußere Kontraktion von x_r in Richtung $s_{(k,m)} - x_r$ aus, d.h.

man berechnet

$$x_{ca} = s_{(k,m)} + \alpha(x_r - s_{(k,m)})$$

Man geht also vom Reflektionspunkt in Richtung des Eckenschwerpunktes. Wenn dieses neue Eck nicht schlechter als alle anderen verbleibenden Ecken ist, behält man es bei, d.h. der Simplex wurde kontrahiert und es geht wieder zum Anfang der Schleife.

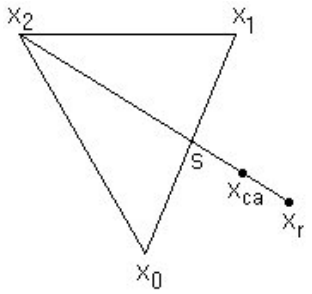

Abbildung 3.4: Kontraktion außen x_{ca} mit $\alpha{=}\frac{1}{2}$ und $j = 1$

3.2.5 Kontraktion total

Ist $f(x_c) < f(x_{(k,m)})$, dann setzt man $x_{(k+1,m)}{=}x_c$, und die restlichen Ecken von S_k bleiben unverändert. Andernfalls haben alle Versuche nichts gebracht. Dann führt man eine totale Kontraktion bzgl. $x_{(k,l)}$ durch, d.h. für $i \neq l$ setzt man

$$x_{(k+1,i)} = \frac{1}{2}(x_{(k,i)} + x_{(k,l)})$$

Dabei bleibt die Ecke mit minimalem Funktionswert gleich (hier: x_1). Alle anderen Ecken ersetzt man durch die Mittelpunkte zwischen ihnen und der besten Ecke. Damit werden alle Seitenlängen des Simplexes halbiert, so dass er schrumpft. Die Schleife geht wieder auf den Anfang zurück.

3.3 Abbruchkriterium

Für das Nelder/Mead-Verfahren gibt es verschiedene Abbruchkriterien. Zum Beispiel wird in der Matlab-Funktion *fminsearch* die Iteration abgebrochen, wenn der Simplex S_k hinreichend klein ist.

Nelder/Mead haben als Abbruchkriterium

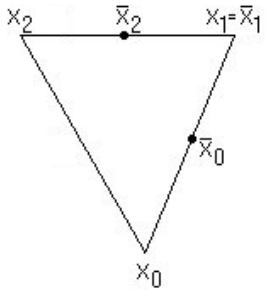

Abbildung 3.5: Kontraktion total mit $j = 1$

$$\left(\frac{1}{n+1}\sum_{i=0}^{n}(f(x_{(k,i)}) - \bar{f}_k)^2\right)^{\frac{1}{2}} < \varepsilon \text{ mit } \bar{f}_k = \frac{1}{n+1}\sum_{j=0}^{n} f(x_{(k,j)})$$

vorgeschlagen, wobei $\varepsilon > 0$ eine vorgegebene Genauigkeitsschranke ist. Dies bedeutet, dass die Standardabweichung der Funktionswerte an den Simplexecken hinreichend klein sein muss [ALT02].

3.4 Pseudocode von Nelder/Mead

Zur besseren Veranschaulichung des Verfahrens von Nelder/Mead wird der Pseudocode aufgezeigt. Die generierten Punkte können anhand der Abbildung 3.6 verfolgt werden. Hierbei steht x_0 für den Startpunkt, x_{min} für den Punkt im Simplex, der den kleinsten Funktionswert hat, sowie x_{max} mit dem größten Funktionswert. Außerdem ist der Eckenschwerpunkt s und die Reflektion x_r, Expansion x_e, innere Kontraktion x_{ci} und die äußere Kontraktion x_{ca} aufgezeigt. Die für die totale Kontratktion generierten Punkte können der Übersicht halber aus der Abbildung 3.6 entnommen werden.

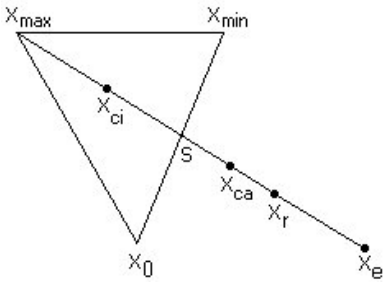

Abbildung 3.6: Skizze für Pseudocode

```
Modul: Nelder/Mead

Eingabe: gamma, beta, alpha, Anzahl der Dimensionen, zu minimierende Funktion, n-dimensionaler
Startpunkt, Anzahl der Nelder/Mead-Iterationen
Ausgabe: Punkte des zuletzt berechneten Simplexes mit dessen Funktionswerten
```

```
Prüfe Parameterbereiche für die eingegebenen Parameter
Initialisiere ersten n-dim. Eckpunkt des Simplexes mit {0,0,0,...,0}
Berechne restliche Eckpunkte des Startsimplexes mit Addition des Einheitsvektors
Berechne Funktionswerte der Eckpunkte

for 1 < i < Iterationen
  Berechne f_min und f_max
  Berechne x_min und x_max
  Berechne s = 1/n * ((SUMME (für i=1 bis n+1) über alle x_i)-x_max)
  Reflektion: x_r = s + gamma (s - x_max)

  if f(x_r) < f(x_min)
    Expansion: x_e = s + beta (x_r - s)

    if f(x_e) < f(x_r)
      x_max = x_e
      f(x_max) = f(x_e)

    else
      x_max = x_r
      f(x_max) = f(x_r)

   if f(x_r) >= f(x_max)
      Partielle innere Kontraktion: x_ci = s + alpha (x_max - s)
      Berechne f(x_ci)

   else
      Partielle äußere Kontraktion: x_ca = s + alpha (x_r - s)
      Berechne f(x_ca)

   if f(x_ci) < f(x_max)
      x_max = x_ci
      f(x_max) = f(x_ci)

   else
     for 1 < i < (Anzahl der Dimensionen + 1)
       Totale Kontraktion: x_i = 1/2 (x_i + x_min)
       Berechne f(x_i)
```

3.5 Erweiterungen vom Nelder/Mead-Algorithmus

Wie bei anderen Verfahren besteht immer die Gefahr, dass nur ein lokales Optimum erreicht wird. Diese Gefahr kann man durch mehrere Maßnahmen mindern [@WID]:

- Nach einer bestimmten Anzahl von Schritten wird neu angesetzt. Dazu wird der bisher beste Punkt beibehalten und um ihn herum ein neuer Start-Simplex aufgebaut, indem bei jedem weiteren Punkt die Koordinate nur eines Parameters z.B. um 5 % variiert wird.

- Additiv zu der eben genannten Erweiterung werden die zusätzlichen Simplex-Punkte durch Zufallswerte noch mehr flexibilisiert.

- Alternativ kann man sogar den Zeitpunkt der Neuansetzung per Zufallszahl variabel gestalten.

- Außerdem erzielen Variationen der Parameter α, γ und insbesondere die Variation von β Verbesserungen bzgl. des Erreichens vom globalen Optimum.

Alle diese Varianten erfordern eine intensive Beschäftigung mit dem jeweiligen konkreten Problem.

Die Reflektion (vgl. Kapitel 3.2.1) und Expansion (vgl. Kapitel 3.2.2) können als Liniensuche interpretiert (vgl. Kapitel 2.3.1) werden, indem die Konstanten γ und β variabel sind. Eine interessante Möglichkeit besteht dann darin, auf den Strecken $\overline{s, x_r} = s + \lambda(x_r - s)$ bzw. $\overline{x_r, x_e} = x_r + \lambda(x_e - x_r)$ mit $\lambda \in [0, 1]$ eine Liniensuche durchzuführen. Da dazu in dieser Arbeit ein ableitungsfreies Verfahren gesucht ist, können die klassischen Verfahren aus Kapitel 2 nicht verwendet werden. Ein dafür geeignetes Verfahren, das auf der Erzeugung von Zufallszahlen basiert, soll nun im nächsten Kapitel vorgestellt werden.

4 Erzeugung von Zufallszahlen

4.1 Definition von Zufallszahlen, Zufallsvariablen und Verteilungen

Als **Zufallszahl** wird das Ergebnis von speziellen Zufallsexperimenten bezeichnet [@WIZ]. Zur Erzeugung von Zufallszahlen gibt es verschiedene Verfahren. Echte Zufallszahlen werden mit Hilfe physikalischer Phänomene erzeugt: Münzwurf, Würfel. Diese Verfahren heißen physikalische Zufallszahlengeneratoren, sind aber zeitlich oder technisch sehr aufwändig. In der realen Anwendung genügt häufig eine Folge von Pseudozufallszahlen, das sind scheinbar zufällige Zahlen, die nach einem festen, reproduzierbaren Verfahren erzeugt werden. Sie sind also nicht zufällig, da sie sich vorhersagen lassen, haben aber ähnliche statistische Eigenschaften (gleichmäßige Häufigkeitsverteilung, geringe Korrelation) wie echte Zufallszahlenfolgen. Solche Verfahren heißen Pseudozufallszahlengeneratoren.

Eine **Zufallsvariable** bezeichnet eine Funktion, die den Ergebnissen eines Zufallsexperiments Werte zuordnet. Diese Werte werden als Realisationen der Zufallsvariablen bezeichnet. Die Zufallsvariable selbst wird üblicherweise mit einem Großbuchstaben bezeichnet (hier $X : \Omega \rightarrow \mathbb{R}$), während man für die Realisationen die entsprechenden Kleinbuchstaben verwendet (hier x). Die Funktion $F : \mathbb{R} \rightarrow$ [0,1] mit

$$F_X(x) = P(X \leq x),\ x \in \mathbb{R}$$

ist die Verteilungsfunktion der Zufallsvariablen X und gibt die Wahrscheinlichkeit P dafür an, dass X höchstens einen Wert von x annimmt [DOM01].

Verteilungsfunktionen sind wie Zufallsvariablen ebenfalls diskret oder stetig. Bei **diskreten Verteilungen**, die aus einer Folge von Tupeln aus Zufallswert und zugehöriger Wahrscheinlichkeit bestehen, lässt sich die Wahrscheinlichkeitsverteilung durch Summation aus der Wahrscheinlichkeitsfunktion erzeugen:

$$F_X(x) = \sum_{x_i \leq x} P(X = x_i)$$

Bei **stetigen Verteilungen** ist eine solche Summation nicht möglich, da der Wertebereich einer stetigen Variable eine überabzählbare Menge ist. Stattdessen wird die Verteilungsfunktion über die Dichtefunktion $f(x)$ berechnet:

$$F_X(x) = \int_{-\infty}^{x} f(x)dx$$

Die Erzeugung von Zufallsvariablen ist die Basis von vielen Algorithmen, die Zufallszahlen erzeugen. Somit können Zufallsvariablen für Zufallszahlen, die gleichmäßig im Intervall zwischen 0 und 1 verteilt sind, alle anderen Verteilungen von diesen Zufallszahlen generieren [DOM01].

4.2 Diskret verteilte Zufallszahlen

Eine Zufallsvariable X heißt diskret, wenn sie nur endlich viele oder höchstens abzählbar unendlich viele Werte annehmen kann [BOE04]. Unter Verwendung von im Intervall [0,1] gleichverteilten Zufallszahlen z_i lassen sich diskret verteilte Zufallszahlen x_i, die relativen Häufigkeiten entsprechen, ermitteln. Hierzu unterteilt man das Intervall [0,1] entsprechend den relativen Häufigkeiten in $n \in \mathbb{N}$ disjunkte Abschnitte z.B. [0,0.1], [0.1,0.3], ..., [0.8,1.0]. Fällt eine Zufallszahl z_i in das k-te Intervall (mit k=1,2,...,n), so erhält man die diskret verteilte Zufallszahl $x_i = k$. Im Folgenden wird ein Überblick der drei bekanntesten Verteilungen von diskreten Zufallszahlen gegeben.

Bernoulli-verteilte Zufallszahlen

Bei der Bernoulli-Verteilung werden wiederholt gleichverteilte Zufallszahlen z erzeugt, für die gilt: wenn $z \leq p$ dann $x = 1$ bzw. wenn $z > p$ dann $x = 0$ [KOE07]. Die Wahrscheinlichkeitsfunktion der Bernoulli-Verteilung ist gegeben durch

$$P_X(x) = \begin{cases} 1-p & \text{für } x\text{=0} \\ p & \text{für } x\text{=1} \end{cases}$$

Die Bernoulli-Verteilung hat einen Parameter p, für den gelten muss, dass $0 < p < 1$. Eine Bernoulli-verteilte Zufallsvariable X nimmt nur die zwei Werte 0 und 1 an [BOE04].

Binomialverteilte Zufallszahlen

Die Binomialverteilten Zufallszahlen haben die Wahrscheinlichkeit, dass bei n Versuchen mit Erfolgswahrscheinlichkeit p die Zahl der Erfolge x ist [KUE08]. Dies ergibt folgende Wahrscheinlichkeitsfunktion:

$$P_X(x) = \binom{n}{x} p^x (1-p)^{n-x} \qquad \text{für } x = 0, 1, 2, \ldots, n$$

Für die zwei Parameter n und p muss gelten, dass $n \in \mathbb{N}$ ist und $0 < p < 1$.

Poissonverteilte Zufallszahlen

Die Poissonverteilung hat die Wahrscheinlichkeit für das Auftreten von x Ereignissen im Einheitsintervall bei einem Poissonprozess mit der Rate λ [KUE08]. Die Wahrscheinlichkeitsfunktion der Poissonvertei-

lung ist definiert durch

$$P_X(x) = \frac{\lambda^x e^{-\lambda}}{x!} \qquad \text{für } x = 0, 1, 2, \ldots, n$$

Die Poissonverteilung hat einen Parameter λ, für den gelten muss λ>0 sein [BOE04].

4.3 Kontinuierlich verteilte Zufallszahlen

Kontinuierlich verteilte Zufallszahlen werden meist wie folgt erzeugt:

Gleichverteilte Zufallszahlen
Für die Erzeugung einer gleichverteilten Zufallszahl gilt: Zuerst wird eine Standardzufallszahl z ([0,1]-gleichverteilt) erzeugt. Gemäß $x{:=}a{+}z{\cdot}(b\text{-}a)$ wird eine gleichverteilte Zufallszahl x auf dem Intervall [a,b] erzeugt [KOE07].

Zufallszahlen einer beliebigen Verteilungsfunktion
Z sei eine im Intervall [0,1] gleichverteilte Zufallsvariable. Ist F eine Verteilungsfunktion, für die die Umkehrfunktion F^{-1} existiert, so besitzt die Zufallsvariable $X = F^{-1}(Z)$ die Verteilungsfunktion F.

Exponentialverteilte Zufallszahlen
Abbildung 4.1 zeigt die Erzeugung von exponentialverteilten Zufallszahlen:

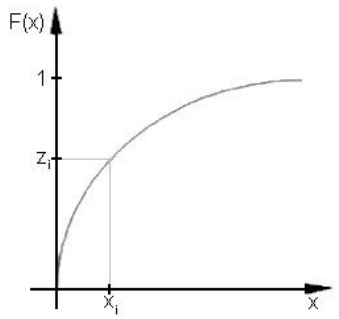

Abbildung 4.1: Exponentialverteilte Zufallszahlen, [DOM95]

Für die Verteilungsfunktion F(x) der Exponentialverteilung gilt:

$$F(x) = 1 - e^{-\delta x}$$

Gleichsetzen der Standardzufallszahl z mit $F(x)$ führt zu [DOM95]:

$$z = F(x) = -e^{-\delta x} + 1 \quad \Leftrightarrow \quad e^{-\delta x} = 1 - z$$

Durch Logarithmieren erhält man

$$-\delta \; x \; \ln \; e = \ln(1 - z) \quad \Leftrightarrow \quad x = -\tfrac{1}{\delta} \; \ln(1 - z) \;\; \text{oder} \;\; x := -\tfrac{1}{\delta} \; \ln(z)$$

und damit eine einfache Möglichkeit zur Erzeugung einer exponentialverteilten Zufallszahl x aus einer Standardzufallszahl z, wobei zum Einsparen von Rechenaufwand in der letzten Gleichung $1 - z$ durch z ersetzt werden kann.

4.4 Anwendungen von Zufallszahlen

4.4.1 Pseudozufallszahlen

Zufallszahlen werden meistens mit Algorithmen erzeugt [KUE08]. Die meisten Compiler stellen Subroutinen dafür zur Verfügung, wie z.B. der GNU-C Compiler mit dem Befehl $srand(iseed)$. Das Integer Argument $iseed$ spezifiziert, welche Folge von Zufallszahlen durch wiederholte Aufrufe der zwei Funktionen $rand()$ erzeugt wird. Der Aufruf $rand()$ erzeugt Integer-Zufallszahlen i mit $0 \leq i \leq 2147483647 = 2^{31} - 1 = RAND\_MAX$. Mit $x = rand()/(RAND\_MAX + 1)$ erzeugt man reelle Zufallszahlen x mit $0 \leq x < 1$.

Nahezu alle Programmiersprachen verfügen über solche Pseudozufallszahlengeneratoren, wie $rand()$. Sie erzeugen eine Zahlenfolge, die zwar zufällig aussieht, es aber nicht wirklich ist, da sie durch einen deterministischen Algorithmus berechnet wird.

Pseudozufallszahlengeneratoren gehen von einer ersten Zahl aus, dem so genannten Keim (seed) und berechnen anschließend eine deterministische Folge von Zahlen, die als Realsierung einer Gleichverteilung interpretiert werden kann [RIE05]. Da die Folge der Zahlen nur vom Wert des Keims abhängt, erhält man bei derselben Wahl des Keims dieselbe Folge von Zahlen, wodurch die Wiederholung einer Simulation z.B. unter abgeänderten Bedingungen mit denselben Zahlen möglich ist. Wenn kein Keim explizit angegeben ist, wird eine Zahl zufällig ausgewählt, z.B. der Wert der Nanosekunde, in dem das Programm gestartet wurde. Die Anzahl der Zahlen, die durch solch eine Konstruktion für alle erlaubten Keime erhalten werden können, ist endlich.

4.4.2 Lineare Kongruenz Generatoren

Linearen-Kongruenz-Generatoren haben auf einem Einheitsintervall von [0,1] folgende Form:

$$X_{k+1} := a \cdot X_k + c \;\text{mod}\; m \;\text{mit}\; k \in \mathbb{N}$$

Ausgegangen wird vom Keim X_0. Die Konstanten a, c, m können einmal gewählt werden und sind dann

festgelegt. Die optimalen Startparameter, um möglichst gutverteilte Pseudozufallszahlen zu bekommen, sind wie folgt definiert [@FHE]:

- c sollte kleiner sein als m
- c sollte auf ...z21 enden, wobei z eine gerade Zahl sein sollte
- m sollte eine Zweier- oder Zehnerpotenz sein
- m sollte groß gewählt werden, weil die Periode maximal $m-1$ lang ist, d.h. maximal $m-1$ verschiedene Zufallszahlen erzeugt werden können

4.4.3 Portable Generatoren

Portable Generatoren sind in einer höheren Programmiersprache geschriebene Generatoren, meist auch ähnlich der linearen Kongruenzgeneratoren. Am bekanntesten ist der **Park-Miller Generator**. Er erzeugt Integer-Zufallszahlen nach einem vereinfachten Prinzip der linearen Kongruenzgeneratoren [KUE08]:

$$X_{k+1} = a \cdot X_k \text{ mod } m \quad \text{mit} \quad a = 7^5,\ m = 2^{31} - 1 = 2147483647$$

Die Schwierigkeit der Implementierung liegt hier darin, bei gewöhnlicher Multiplikation Overflow oder andere Ausnahmen zu vermeiden.

4.5 Erzeugung von Zufallszahlen mit vorgegebener Wahrscheinlichkeitsdichte

4.5.1 Inversionsmethode

Bei der Inversionsmethode wird zuerst die Zufallsvariable X mit den Werten x_1, x_2, x_3,... realisiert, wobei $x_1<x_2<x_3<...$ gilt [LAW00]. Die ansteigende Verteilungsfunktion $F(x)$ ist im Intervall [0,1]. Der Algorithmus der Inversionsmethode zur Erzeugung von X ist wie folgt definiert:

1. Generiere $U \sim U$[0,1]
2. Return $X = F^{-1}(U)$

Die Zufallsvariable X besitzt damit die Verteilung, die der Verteilungsfunktion $F(x)$ zugrunde liegt [DOM01]. Folgendes Beispiel soll die Inversionsmethode verdeutlichen. Gegeben sei die Dichtefunktion zweier Zufallszahlen X_1 und X_2, die gemäß der angegebenen Dichte- bzw. Verteilungsfunktion verteilt sind:

$$f(x) = \frac{1}{\pi(x^2+1)} \qquad \text{mit der Verteilungsfunktion } F(x) = \frac{arctan(x)}{\pi} + 0,5$$

Diese Verteilungsfunktion besitzt eine Umkehrfunktion $F^{-1}(x)$ in geschlossener Form und ist stetig und monoton steigend auf dem Definitionsbereich [0,1]. Gemäß dem Algorithmus werden als erstes zwei Zufallsvariablen U_1 und $U_2 \sim U$[0,1] gebildet und wie in der Abbildung 4.2 gezeigt in die Umkehrfunktion eingesetzt.

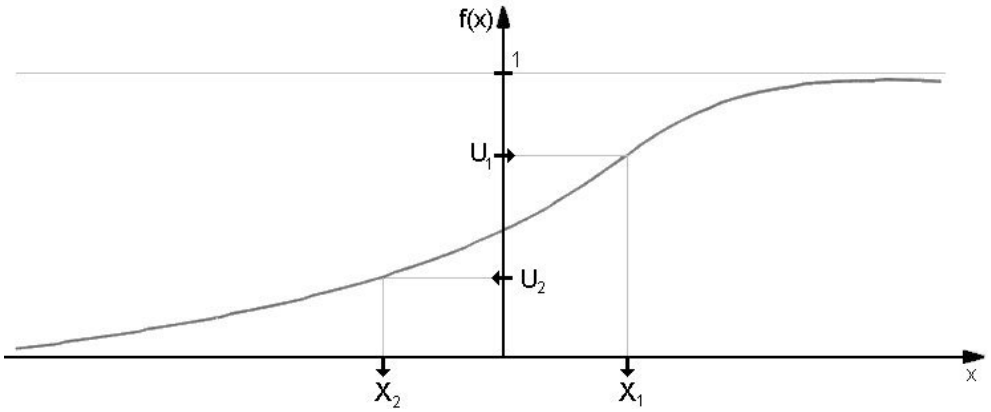

Abbildung 4.2: Verwendung der Inversionsmethode

Die Berechnung der Umkehrfunktion ist abhängig von der gegebenen Funktion sehr aufwändig und deshalb machmal nicht zu berechnen. Es gibt auch einige Vorteile dieser Methode. Falls die Umkehrfunktion berechenbar ist, so benötigt diese Methode nur eine einzige U[0,1]-verteilte Zufallszahl, um eine neue Zufallszahl zu generieren. Außerdem können mit dieser Methode leicht korrelierende Zufallszahlen erzeugt werden.

4.5.2 Kompositionsmethode

Verwendung findet die Kompositionsmethode, wenn sich die Verteilungsfunktion F aus mehreren Funktionen F_1, F_2,... abschnittsweise zusammensetzen kann [LAW00]:

$$F(x) = \sum_{j=1}^{\infty} p_j F_j(x) \qquad \text{mit } p_j \geq 0 \text{ und } \sum_{j=1}^{\infty} p_j = 1 \text{ und } F_j \text{ ist eine Verteilungsfunktion}$$

Der Grund für diese Stückelung liegt darin, dass die Zufallszahlen damit meist leichter erzeugt werden können als die Zufallszahlen zur Ausgangsfunktion F. Die Kompositionsmethode hat folgenden Algorithmus:

1. Realisiere eine positive Integer-Zufallsvariable J mit $P(J = j)$=p_j mit j=1,2,...

2. Return X mit der Verteilungsfunktion F_j

Im Schritt 1 wird eine Auswahl von Verteilungsfunktionen F_j mit der Wahrscheinlichkeit p_j generiert. Im zweiten Schritt wird X zurückgegeben, wobei X unabhängig zu J ist. Folgendes Beispiel zeigt anhand der Dichtefunktion $f(x) = 0,5e^{|x|}$ die Verwendung der Kompositionsmethode mit zugehöriger Verteilungsfunktion:

$$F(x) = \begin{cases} 0,5e^{x} & \text{falls -}\infty \leq \text{x} < 0 \\ 1 - 0,5e^{-x} & \text{sonst} \end{cases}$$

Beide Teile der Verteilungsfunktion sind gleichwahrscheinlich, somit gilt $p_1 = p_2 =$ 0,5. Zuerst generiert man nun U_1, $U_2 \sim U$(0, 1). Gilt $U_1 <$ 0,5 dann liefert die Methode $F_1^{-1}(U_2) =$ ln U_2 andernfalls $F_2^{-1}(U_2) =$ -ln U_2.

Die Kompositionsmethode kann auch **geometrisch interpretiert** werden, z.B. durch Teilung der Fläche unterhalb der Dichte mittels der y-Achse, um f zu ermitteln (siehe Abbildung 2.3). Für eine kontinuierliche Zufallsvariable X mit Dichte f könnte man die Fläche unter f aufteilen in Teilflächen p_1, p_2, ... bezogen auf den Ausbau von f in konvex-Kombinationen. Im ersten Schritt wird eine der Flächen gewählt und im zweiten Schritt die Verteilung bezogen auf die gewählte Region gewählt.

Die in Abbildung 4.3 gezeigte Doppelexponentialverteilung hat eine Dichte von $f(x) = 0,5^{|x|}$ für alle realen x. Aus der Grafik erkennt man, bezogen auf den normalisierten Faktor 0,5, dass $f(x)$ zwei exponentielle und aufeinanderfolgende Dichten hat.

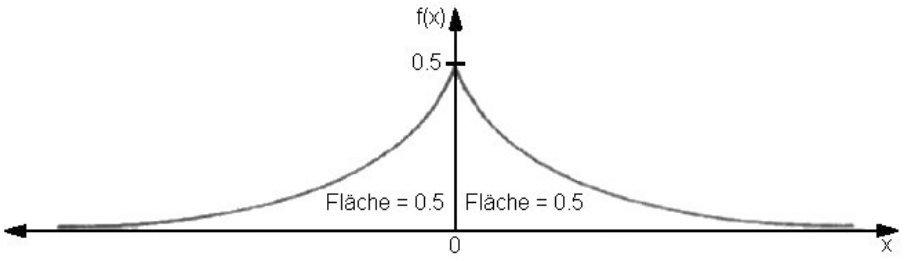

Abbildung 4.3: Erstes Beispiel zur Kompositionsmethode

Die Fläche unterhalb der Dichte kann auch mittels der x-Achse geteilt werden. Die Trapezverteilung hat die Dichte:

$$f(x) = \begin{cases} a + 2(1-a)x & \text{wenn } 0 \leq x \leq 1 \\ 0 & \text{sonst} \end{cases}$$

Wie in Abbildung 4.4 zu sehen ist, wird die Fläche f aufgeteilt in ein Rechteck mit der Fläche a und ein Dreick mit der Fläche $1 - a$.

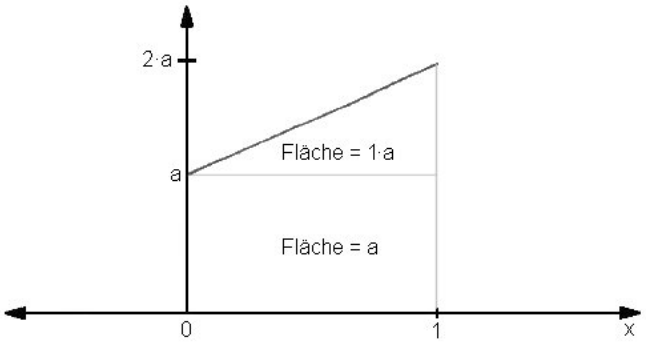

Abbildung 4.4: Zweites Beispiel zur Kompositionsmethode

4.5.3 Faltungsmethode

Der Faltungsmethode liegt die Beobachtung zugrunde, dass manche Verteilungen als Summe von unabhängigen und identisch verteilten Zufallsvariablen Y_j dargestellt werden können. Falls die Y_j leichter zu ermitteln sind als die Verteilung von X selbst, so kann X ermittelt werden als

$$X = Y_1 + Y_2 + ... + Y_m$$

Der Algorithmus zur Generierung der gewünschten Zufallszahlen [LAW00]:

1. Generiere unabhängige und gleichverteilte Zufallszahlen Y_1,Y_2,...,Y_m mit der Verteilungsfunktion G

2. Return $X = Y_1$+Y_2+...+Y_m

Die Faltungsmethode ist dann sinnvoll zu verwenden, wenn die Funktion nicht zu komplex ist, also nicht zu viele Y_j erzeugt werden müssen bzw. diese leicht zu erzeugen sind. Deutlich wird dies im folgenden Beispiel:

$$F(x) = \begin{cases} 0 & \text{wenn x} < X_1 \\ \frac{i-1}{n-1} + \frac{x-X_i}{(n-1)(X_{i+1}-X_i)} & \text{wenn } X_i \leq \text{x} < X_{i+1}, \text{ i=1,2,...,n} \\ 1 & \text{wenn x} \geq X_n \end{cases}$$

Die Abbildung 4.5 zeigt eine Beispielrealisierung mit U.

Dieses Beispiel zeigt, dass die Faltungsmethode auf der Inversionsmethode basiert, denn die Invertierung von $(X_I,\ X_{I+1})$ ergibt X.

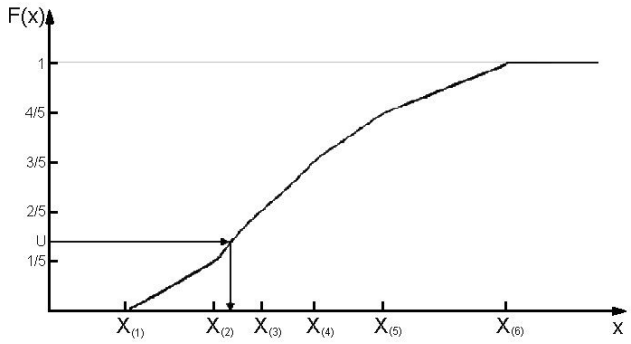

Abbildung 4.5: Anwendung der Faltungsmethode, [MAD07]

4.5.4 Annahme- und Ablehnungsmethode / Neumann-Algorithmus

Die Annahme-/Ablehnungsmethode von John v. Neumann geht von einer Dichtefunktion $f(x)$ aus, zu der eine weitere Funktion $t(x)$ ermittelt wird. Es gelte, dass $f(x) \leq t(x)$ ist. Die Funktion $t(x)$ kann auch eine Dichtefunktion sein. Es wird folgende Annahme für t getroffen:

$$c = \int_{-\infty}^{\infty} t(x)\ dx \geq \int_{-\infty}^{\infty} f(x)\ dx = 1$$

Die Funktion $t(x)$ kann zu einer Dichtefunktion werden, indem sie durch c geteilt wird. Die resultierende Funktion lautet dann $r(x) = t(x)/c$ (siehe Abbildung 4.6), mit der Zufallszahlen dann leichter generierbar sind [DOM01]. Im Folgenden wird von dieser Dichtefunktion ausgegangen.

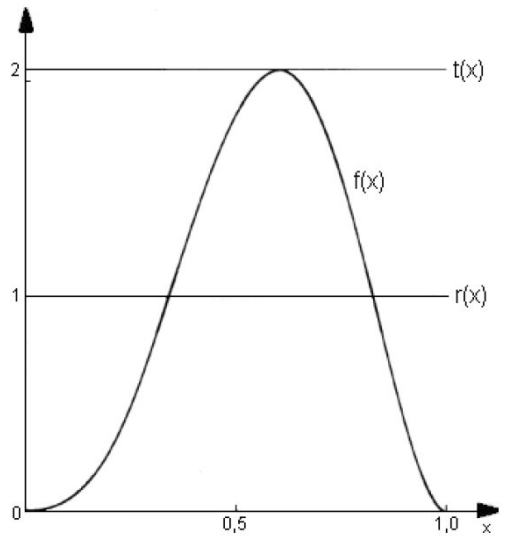

Abbildung 4.6: Annahme- und Ablehnungsmethode

Die Zufallsvariablen X (Realisationen) werden im Intervall [0,1] generiert und auf der x-Achse ange-

tragen und die Y-Zufallsvariablen werden mit der Dichte r erzeugt und auf $t(x)$ angetragen. Welche Paare (Y, U) als X angenommen werden, wird mit folgendem Algorithmus entschieden:

1. Erzeuge Y mit der Dichtefunktion $r(x)$
2. Erzeuge $U \sim U[0, 1]$
3. Wenn $U \leq f(Y)/t(Y)$ dann return $X = Y$ ansonsten werden Y und U verworfen und bei Schritt 1 wird erneut angefangen

Analog wird bei der Suche nach dem Minimum für die Akzeptanz von Y abgefragt, ob $U \geq f(Y)/t(Y)$ ist.

Ein Beispiel soll den Neumann-Algorithmus verdeutlichen. Gegeben sei folgende Funktion:

$$f(x) = 60x^3(1-x)^2$$

Abbildung 4.7 zeigt im ersten Bild die Dichtefunktionen f und r, wobei r meist gleichverteilt ist. Im mittleren Bild ist die erste Ableitung der Funktion f(x) dargestellt. Wie auf der x-Achse erkennbar ist, liegt das Maximum bei $x_{max} = 0,6$. Der Wert eingesetzt in die Funktionsgleichung ergibt $f(0,6) = 2,0736$, so dass gilt, dass $t = 2,0736$, das in der rechten Abbildung eingezeichnet ist.

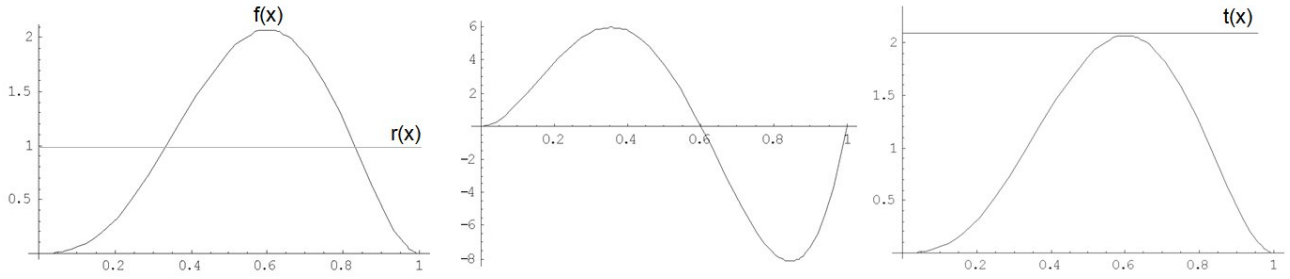

Abbildung 4.7: Beispiel für Annahme- und Ablehnungsmethode

Im folgenden Schritt beginnt nun der Neumann-Algorithmus, der im Weiteren n-Mal durchlaufen wird. Wie theoretisch eben beschrieben werden nun Zufallswerte auf der x- und y-Achse erzeugt. Weil die Y mit der Dichte r erzeugt werden, sind sie gleichförmig auf $t(X)$ verteilt, wie Abbildung 4.8 im linken Bild zeigt.

Im nächsten Schritt werden die Y als X angenommen, wenn erfüllt ist, dass $U \cdot t(Y) \leq f(Y)$ ist. Weil in diesem Beispiel das Maximum ermittelt wird, steht hier das $\leq$-Zeichen und nicht das $\geq$-Zeichen für die Minimumsuche. Wenn die Y als X angenommen werden, bedeutet das, dass die Punkte $P(Y, U \cdot t(Y))$ unter der Kurve der Dichte f liegen.

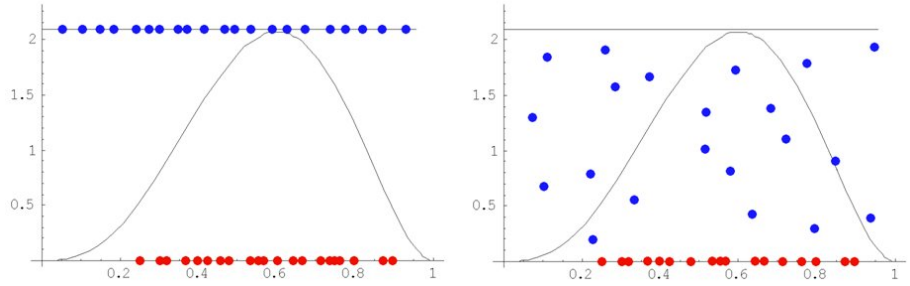

Abbildung 4.8: Beispiel für Annahme- und Ablehnungsmethode mit Punkteverteilung

In der Abbildung 4.8 ist auf dem rechten Bild zu sehen, dass die Konzentration von X und Y beim Maximum von $f(x)$ am höchsten ist. In den Regionen, in denen die Funktionswerte $f(x)$ niedrig also nahe 0 sind, ist $f(Y)/t(Y)$ klein. Dies hat zur Folge, dass die meisten Y zurückgewiesen werden und nur selten die Y als X akzeptiert werden. Analog werden in den Regionen, in denen die Y nahe dem Wert 0,6 sind, die meisten Y angenommen. Hier gilt, dass $f(Y)/t(Y)$ nahe 1 ist.

Somit dünnt der Algorithmus die Y von der $r(x)$ Dichte aus, wo $t(x)$ viel größer ist als $f(x)$, behält aber die meisten der Y bei denen $t(x)$ nur etwas höher ist als $f(x)$.

4.6 Optimierung mit Zufallszahlen

Für die vorliegende Arbeit sind nur kontinuierlich verteilte Zufallszahlen relevant, weil nichtdiskrete Funktionen optimiert werden sollen. Somit kommen im weiteren Pseudozufallszahlengeneratoren zum Einsatz, um kontinuierlich verteilte Zufallszahlen zu erzeugen.

Um das globale Optimum von nichtlinearen Funktionen zu ermitteln, können Zufallszahlen mit vorgegebener Wahrscheinlichkeitsdichte generiert werden. Hierzu zählt beispielsweise die Inversionsmethode. Sie hat aber den Nachteil, dass zur Ermittlung der Zufallszahlen eine Umkehrfunktion notwendig ist und in der vorliegenden Arbeit keine Restriktionen an die Funktion gemacht werden sollen. Außerdem kann es vorkommen, dass die Umkehrfunktion $F^{-1}(z)$ u.U. keine geschlossene Form aufweist, so dass keine Zufallszahlen berechnet werden können.

Eine weitere untersuchte Methode ist die Kompositionsmethode. Die Herausforderung hierbei ist es, eine geeignete Zerlegung der Funktion zu finden, was bei mehrdimensionalen Funktionen schwierig ist. Die Kompositionsmethode ist außerdem aufwändig, weil mitunter mehrere Zufallszahlengeneratoren für die Teilfunktionen benötigt werden.

Die Faltungsmethode, die ebenso Zufallszahlen mit vorgegebener Wahrscheinlichkeitsdichte erzeugt, ist für eine Verwendung in der vorliegenden Arbeit zu komplex zu berechnen, weil beim Rechnen mit mehrdimensionalen Funktionen bei der Faltungsmethode zu viele Zufallszahlen generiert werden müssen, um Ergebnisse zu erhalten.

Die Annahme- und Ablehnungsmethode ist die zuletzt betrachtete Methode und hat im Gegensatz zu den anderen Methoden viele Vorteile beim Berechnen der Zufallszahlen. Zum einen ist keine genaue

Analyse der Dichte- oder Verteilungsfunktion erforderlich. Zum anderen muss die Dichtefunktion nicht integrierbar sein. Als einzigen Nachteil ist der erhöhte Rechenaufwand zu nennen (Pidd 1992: „Throwing darts at a dart board“). Die Rechenzeit ist aber durch geeignete Modifikationen am Algorithmus zu verbessern.

Am Ende von Kapitel 3 wurde die Idee besprochen, die Reflektion bzw. Expansion im Nelder/Mead-Verfahren auf eine Liniensuche auf einer vorgegebenen Strecke durchzuführen. Dies kann durch geeignete Anpassung des Neumann-Algorithums erfolgen und wird im nächsten Kapitel beschrieben.

5 Kombination vom Nelder/Mead-Algorithmus mit Neumann-Algorithmus

5.1 Erweiterter Algorithmus

Das Problem vom Nelder/Mead-Algorithmus ist seine Determiniertheit. Deshalb wird bei Funktionen, die mehr als ein globales Optimum haben, mit diesem Algorithmus nur ein Optimum gefunden oder das Verfahren verfängt sich stets an einem möglicherweise nur lokalen Optimum. Um dies zu verbessern, wurde in dieser Arbeit der Nelder/Mead-Algorithmus erweitert. Die Grundlage vom erweiterten Algorithmus ist der Neumann-Algorithmus (Annahme-/Ablehnungsmethode), der in Kapitel 4.5.4 ausführlich beschrieben wurde. Mit der Einbindung dieses stochastischen Algorithmus ist der erweiterte Nelder/Mead-Algorithmus nicht mehr deterministisch, d.h. alle globalen Optima können nun möglicherweise berechnet werden.

Der Neumann-Algorithmus wird normalerweise für Berechnungen mit eindimensionalen Funktionen verwendet. Nachdem mit dem erweiterten Nelder/Mead-Algorithmus aber auch mehrdimensionale Funktionen untersucht werden sollen, wird die zu optimierende Funktion auf eine eindimensionale Funktion heruntergebrochen. Zur Realisierung dessen wird eine Konvexkombination zwischen zwei Punkten der Funktion durchgeführt (vgl. Kapitel 3.5). Als Dichtefunktion wird zur Vereinfachung $r(x) = 1$ gemäß Kapitel 4.5.4 gewählt.

Eine Konvexkombination ist eine Linearkombination von endlich vielen Punkten im Definitionsbereich der Funktion $f : \mathbb{R}^n \to \mathbb{R}$, bei der die Koeffizienten nicht negativ sind und ihre Summe 1 ergibt [@SCH]. Der Ausdruck

$$u + \lambda(v - u)$$

stellt für zwei Punkte $u, v \in \mathbb{R}^n$ eine Konvexkombination dar. Die Punkte u und v stehen für zwei Punkte des Definitionsbereichs der betrachtenden Funktion. Eine Beispielrechnung an der Rosenbrock-Funktion von Kapitel 6.1.3 soll dies verdeutlichen.

Funktion: $f : \mathbb{R}^2 \to \mathbb{R}$

$f(x_1, x_2) = 100(x_2 + x_1^2)^2 + (1 - x_1)^2$

Eckpunkte:	$u = (2,3)^T,\ v = (4,2)^T$
Konvexkombination:	$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} + \lambda \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 2+2\lambda \\ 3-\lambda \end{pmatrix}, \quad \lambda \in [0,1]$
somit gilt:	$x_1 = 2+2\lambda,\ x_2 = 3-\lambda$
eingesetzt:	$\hat{f} : \mathbb{R} \to \mathbb{R}$
	$\hat{f}(\lambda) := f(2+2\lambda, 3-\lambda)$
	$\hat{f}(\lambda) = 100(3-\lambda+(2+2\lambda)^2)^2 + (1-(2+2\lambda))^2$
	$\hat{f}(\lambda) = 1600\lambda^4 - 5600\lambda^3 + 10504\lambda^2 + 9804\lambda + 4901, \quad 0 \leq \lambda \leq 1$

In den folgenden zwei Abbildungen werden von der 50dimensionalen Funktion aus Kapitel 7.1.8 zwei Neumann-Durchläufe gezeigt. Es wurden in der Abbildung 5.1 sieben und in der Abbildung 5.2 drei der von Neumann generierten Punkte zugelassen, die somit zeigen, wo das lokale Optimum ist.

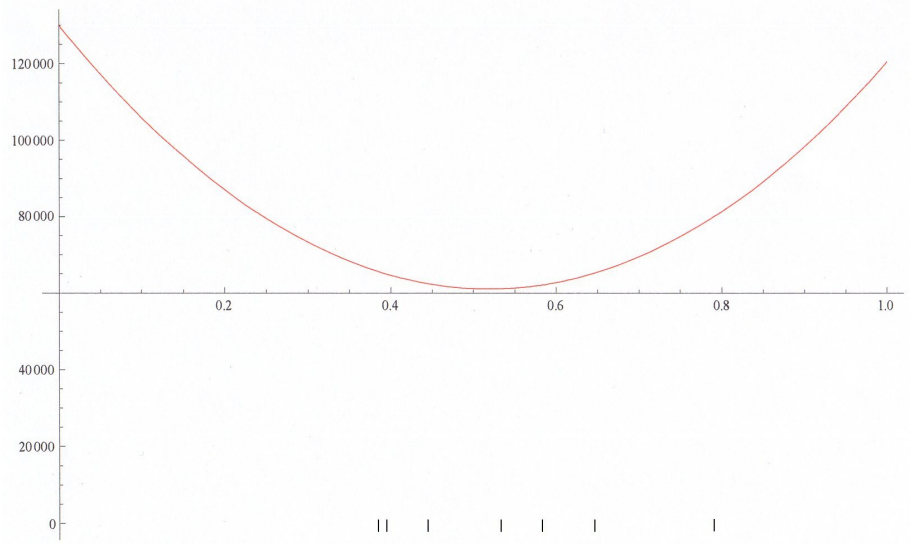

Abbildung 5.1: Erstes Beispiel von Neumann

Der Neumann-Algorithmus kommt im erweiterten Nelder/Mead-Algorithmus an folgenden Stecken zum tragen: Zuerst wird er zwischen dem Reflektionspunkt und dem Expansionspunkt (x_r und x_e) eingefügt. Eine weitere Strecke ist die vom schlechtesten Punkt hin zum inneren Kontraktionspunkt (x_m und x_{ci}). Außerdem wird auf der Strecke vom äußeren Kontraktionspunkt und dem Reflektionspunkt (x_{ca} und x_r) mit dem Neumann-Algorithmus ein Punkt mit möglichst optimalem (bei Minimumsuche niedirigem) Funktionswert ermittelt.

Voraussetzung an die zu optimierende Funktion bei Verwenden vom Nelder/Mead-Neumann-Algorithmus ist, dass sie positiv sein muss, weil mit dem Neumann-Algorithmus nur mit positiven Zahlen gerechnet werden kann.

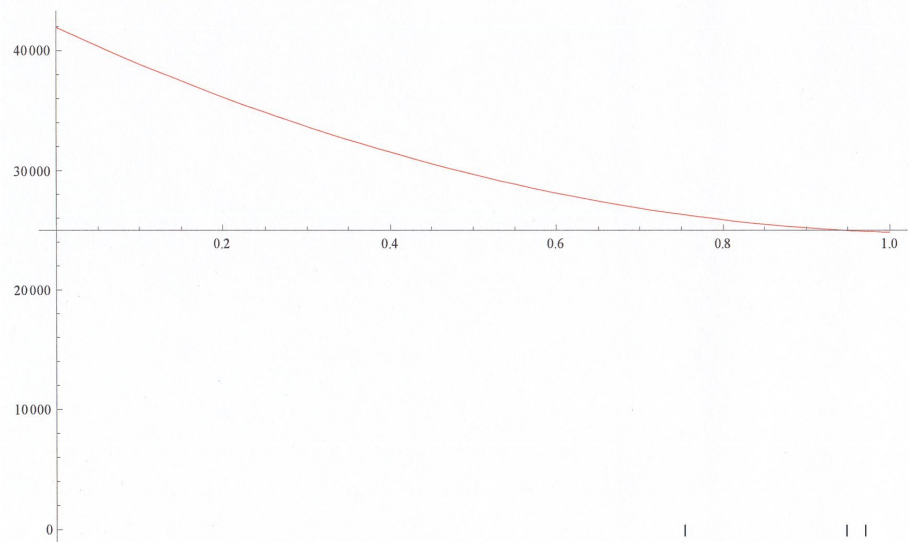

Abbildung 5.2: Zweites Beispiel von Neumann

5.2 Realisierung im Programm Mathematica

Realisiert wurde der mit Neumann erweiterte Nelder/Mead-Algorithmus in dieser Arbeit mit dem Programm **Mathematica** von der Firma Wolfram Research. Alternative Programme, wie Maple oder Matlab wurden wegen ihrer bekannten Trägheit bei Berechnungen von komplexeren Programmen und Ausführung von Schleifen nicht gewählt.

Der folgende Pseudocode zeigt das Modul **Neumann**. Hierzu müssen zuerst zwei Zufallszahlen im Intervall von [0,1] generiert werden. Zum einen ist es der λ-Wert, der einen zufälligen Abstand von a nach b generiert, und eine weitere Zufallszahl, die entscheidet, ob ermittelte Punkte angenommen oder abgelehnt werden (hier im Pseudocode: u). Dann kann eine Konvexkombination durchgeführt werden, bei der ein Punkt berechnet wird, der einen möglichst kleinen Funktionswert hat.

Wenn dann der Funktionswert des neu ermittelten Punktes kleiner als der bisherige maximale Funktionswert und größer als 0 ist, wird der Funktionswert vermerkt, wenn die Zufallszahl u größer als die Differenz vom neuen Funktionswert durch den maximalen Funktionswert ist. Hierbei wird der λ-Wert in die Hitliste und der Funktionswert des neuen Punktes in die Werteliste aufgenommen.

Ist der neue Funktionswert aber kleiner als 0, muss die Abfrage mit der Zufallsvariablen u nicht durchgeführt werden. In diesem Fall wird der λ-Wert gleich in die Hitliste und der Funktionswert in die Werteliste aufgenommen.

Es genügt den Algorithmus 20-mal zu durchlaufen, um ausreichend gute neue Punkte zu erhalten. Am Ende der Funktion werden beide Listen zur Weiterbearbeitung mit dem Nelder/Mead-Algorithmus zurückgegeben.

Im Folgenden wird der Pseudocode für das Modul Neumann aufgezeigt.

```
Modul: Neumann
-----------------------------------------------------------------------------
Eingabe: Anzahl der Durchläufe vom Neumann-Algorithmus, Punkt a, Punkt b, Funktionsgleichung,
Maximaler Funktionswert von Punkt a und b, aktueller maximaler Funktionswert f_max, Anzahl nach
wieviel gefundenen akzeptierten Punkten beim Neumann-Algorithmus abbrechen
Ausgabe: Hitliste und Werteliste
-----------------------------------------------------------------------------
Initialisiere Hitliste und Werteliste

for (i <= Anzahl Neumann-Iterationen) und (Länge der Hitliste <= bisherige Hits)
  Ermittle Zufallszahl lambda
  Ermittle Zufallszahl u
  Berechne Konvexkombination c=a+lambda(b-a)
  Berechne f(c)

  if (f(c) <= f_max) und (f(c) >= 0)
    if u > (f(c) / f_max)
      Schreibe lambda nach Hitliste
      Schreibe f(c) nach Werteliste

  if f(c) < 0
    Schreibe lambda nach Hitliste
    Schreibe f(c) nach Werteliste

  return Hitliste und Werteliste
```

Der Algorithmus des erweiterten Nelder/Mead ist im Pseudocode wie folgt:

```
Modul: Erweiterter Nelder/Mead
-----------------------------------------------------------------------------
Eingabe: gamma, beta, alpha, Anzahl der Dimensionen, zu minimierende Funktion, n-dimensionaler
Startpunkt, Anzahl der Nelder/Mead-Iterationen, Anzahl der Neumann-Iterationen
Ausgabe: Punkte des zuletzt berechneten Simplexes mit dessen Funktionswerten
-----------------------------------------------------------------------------
Prüfe Parameterbereiche für die eingegebenen Parameter
Initialisiere ersten n-dimensionalen Eckpunkt des Simplexes mit {0,0,0,...,0}
Berechne restliche Eckpunkte des Startsimplexes mit Addition des Einheitsvektors
Berechne Funktionswerte der Eckpunkte

for 1 < i < Iterationen
  Berechne f_min, f_max, x_min und x_max
  Berechne s = 1/n * (SUMME (für i=1 bis n+1) über alle x_i)-x_max)
  Reflektion: x_r = s + gamma (s - x_max)
  Berechne f(x_r)

  if f(x_r) < f(x_min)
    Expansion: x_e = s + beta (x_r - s)
    Berechne f(x_e)
    Neumann-Aufruf[Neumannanzahl, x_r, x_e, f(x), f(x_r), f(x_r), Abbruchkriterium]
    Neumann-Rückgabe[Hitliste, Werteliste]
    Hitliste = 0, 1, Hitliste
    Werteliste = f(x_r), f(x_e), Werteliste
    x_max = x_r + x_min * (x_e - x_r)
    f(x_max) = Hitliste(f(x_max))

    if f(x_r) <= f(x_max)
      Expansion: x_e = s + beta (x_r - s)
      Neumann-Aufruf[Neumannanzahl, x_r, x_e, f(x), f(x_r), f(x_r), Abbruchkriterium]
      Neumann-Rückgabe[Hitliste, Werteliste]
      Hitliste = 0, Hitliste
```

```
Werteliste = f(x_r), f(x_e), Werteliste
x_max = x_r + x_min*(x_e - x_r)
f(x_max) = Hitliste(f(x_max))

if x_r >= f(x_max)
  Partielle innere Kontraktion: x_c = s + alpha (x_max - s)
  Neumann-Aufruf[Neumannanzahl, x_max, x_c, f(x), f(x_max), f(x_max), Abbruchkriterium]
  Neumann-Rückgabe[Hitliste, Werteliste]
  Hitliste = 0, 1, Hitliste
  Werteliste = f(x_max), f(x_c), Werteliste
  x_c = x_max + x_min * (x_c - x_max)
  f(x_c) = Hitliste (f(x_c))

else
  Partielle äußere Kontraktion: x_c = s + alpha (x_r - s)
  Neumann-Aufruf[Neumannanzahl, x_c, x_r, f(x), f(x_r), f(x_r), Abbruchkriterium]
  Neumann-Rückgabe[Hitliste, Werteliste]
  Hitliste = 0, 1, Hitliste
  Werteliste = f(x_c), f(x_r), Werteliste
  x_c = x_c + x_min * (x_r - x_c)
  f(x_c) = Hitliste(f(x_c))

Berechne f(x_c)

if f(x_c) < f(x_max)
  x_max = x_c
  f(x_max) = f(x_c)

else
  for 1 < i < (Anzahl der Dimensionen + 1)
    Totale Kontraktion: x_i = 1/2 (x_i + x_min))
    Berechne f(x_i)
```

5.3 Beispielrechnung vom Nelder/Mead-Neumann-Algorithmus

Am Beispiel der Himmelblau-Funktion von Kapitel 6.1.1 wird nun der Nelder/Mead-Neumann-Algorithmus angewendet. Abbildung 5.3 zeigt den Startsimplex mit den Koordinaten x_0, x_1 und x_2. Im nächsten Schritt wird eine Expansion durchgeführt, so dass sich für den zweiten Simplex der Eckpunkt x_e ergibt.

Wie in Abbildung 5.4 zu sehen ist, wird im darauffolgenden Schritt der nächste Simplex durch Reflektion mit x_r generiert. Der vierte Simplex wird dann durch innere Kontraktion mit dem Eckpunkt x_{ci} dem Optimum angenähert.

In fast allen Berechnungen der Funktionen im Kapitel 6 werden der Nelder/Mead-Algorithmus und der Nelder/Mead-Neumann-Algorithmus in 60 Iterationen berechnet. Diese Anzahl hat sich aus den Erfah-

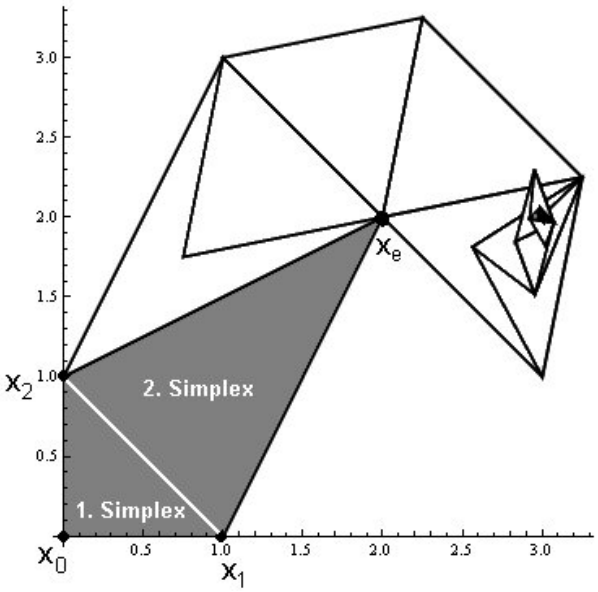

Abbildung 5.3: 1. und 2. Simplex

rungen von Testläufen als sinnvoll herausgestellt, weil somit die Ergebnisse mit 5-stelliger Genauigkeit berechnet werden konnten. Der Neumann-Algorithmus kommt im Schnitt bei 60 Iterationen ca. 45 mal zum Tragen. Um das Verhalten aufzuzeigen, werden im Weiteren von den ersten 30 Iterationen die vom Neumann-Algorithmus erzielten Hitlisten der obigen Testfunktion aufgezeigt. Bei den Iterationen 4, 6, 8, 13 und 28 wurde der vom Nelder/Mead-Algorithmus errechnete Wert übernommen. In den anderen 25 Fällen wurden die durch den Neumann-Algorithmus verbesserten Werte genommen.

1. {0.798251,0.552361,0.60237,0.79541,0.992697,0.715365,0.278564,0.660069,0.591862,0.952007}
2. {0.368251}
3. {0.574596,0.560688,0.180602,0.163264,0.653038,0.307746,0.658255,0.122241}
4. {}
5. {0.893846,0.172197,0.93982,0.928209,0.777467,0.821481,0.649284,0.709464}
6. {}
7. {0.610922,0.746411,0.919565,0.811016,0.656504,0.723067,0.644765,0.560777,0.74976,0.462481, 0.739414}
8. {}
9. {0.512684,0.469538,0.362249,0.560185,0.954798,0.779575}
10. {0.504416,0.781874,0.251738,0.540402,0.989928,0.904024,0.999052,0.916917,0.621298}
11. {0.36956,0.897397,0.739659,0.907468,0.703673,0.704621,0.881295,0.787704,0.259997,0.90264,0.99304}
12. {0.88292,0.474625,0.200832,0.686921,0.971787,0.62491,0.93799}
13. {}
14. {0.589376,0.641505,0.480602,0.736432,0.476459,0.544585,0.966429,0.377053,0.116905,0.838467, 0.991926}

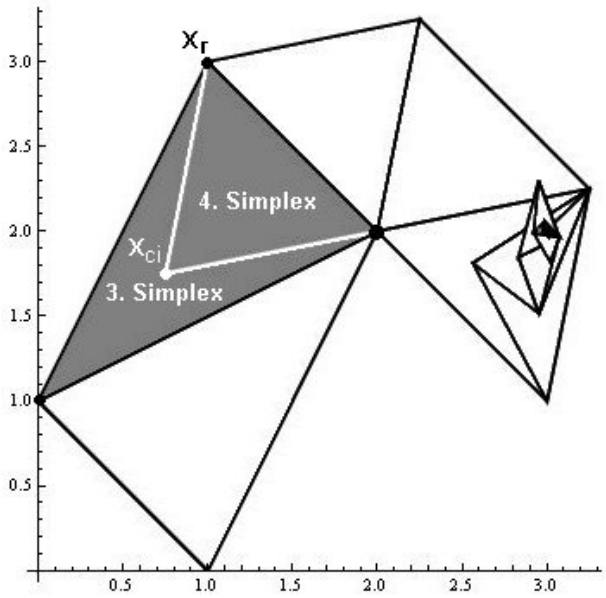

Abbildung 5.4: 3. und 4. Simplex

15. {0.196963,0.511324,0.915946,0.894103,0.949517,0.733997,0.537035,0.690095,0.655229,0.160558, 0.761126}
16. {0.638576,0.257531,0.29104,0.949662,0.919441,0.498786,0.280866,0.241255,0.376287,0.819126}
17. {0.108337,0.667781,0.386915,0.23349,0.414365,0.28418,0.306027,0.696959,0.029178,0.642263}
18. {0.631635,0.835339,0.629751,0.648612,0.967596,0.619434,0.480132,0.411303,0.575963,0.715785, 0.154988}
19. {0.337572,0.869692,0.745259,0.56425}
20. {0.965233,0.959203,0.901557,0.761555,0.366542,0.621847,0.452046,0.656614,0.755057,0.983699, 0.884571}
21. {0.310729,0.944187,0.106719,0.32234,0.910669,0.29356}
22. {0.84635,0.775632,0.605087,0.68458,0.898129,0.777403,0.884787,0.567347,0.815595,0.720997, 0.927626}
23. {0.434877,0.893615,0.761506,0.567514,0.279169,0.586282,0.108498,0.763582}
24. {0.99332,0.916732,0.972161,0.565188,0.385878,0.45669,0.693211,0.76205,0.301138,0.714287, 0.742126}
25. {0.819218,0.356248,0.900988,0.871241,0.479094,0.667617,0.717683,0.63567,0.816695}
26. {0.592049,0.407252,0.437597,0.132508,0.308849,0.496838,0.492154,0.86884,0.72462,0.336284, 0.317368}
27. {0.898686,0.766179,0.981986,0.489832,0.235794,0.620992,0.794296,0.896372,0.458012,0.579004, 0.793147}
28. {}
29. {0.865614,0.537501,0.773851,0.299372,0.383863,0.209358}
30. {0.361895,0.612387,0.692634,0.17555,0.04777,0.813655,0.0441227,0.656184,0.0400059}

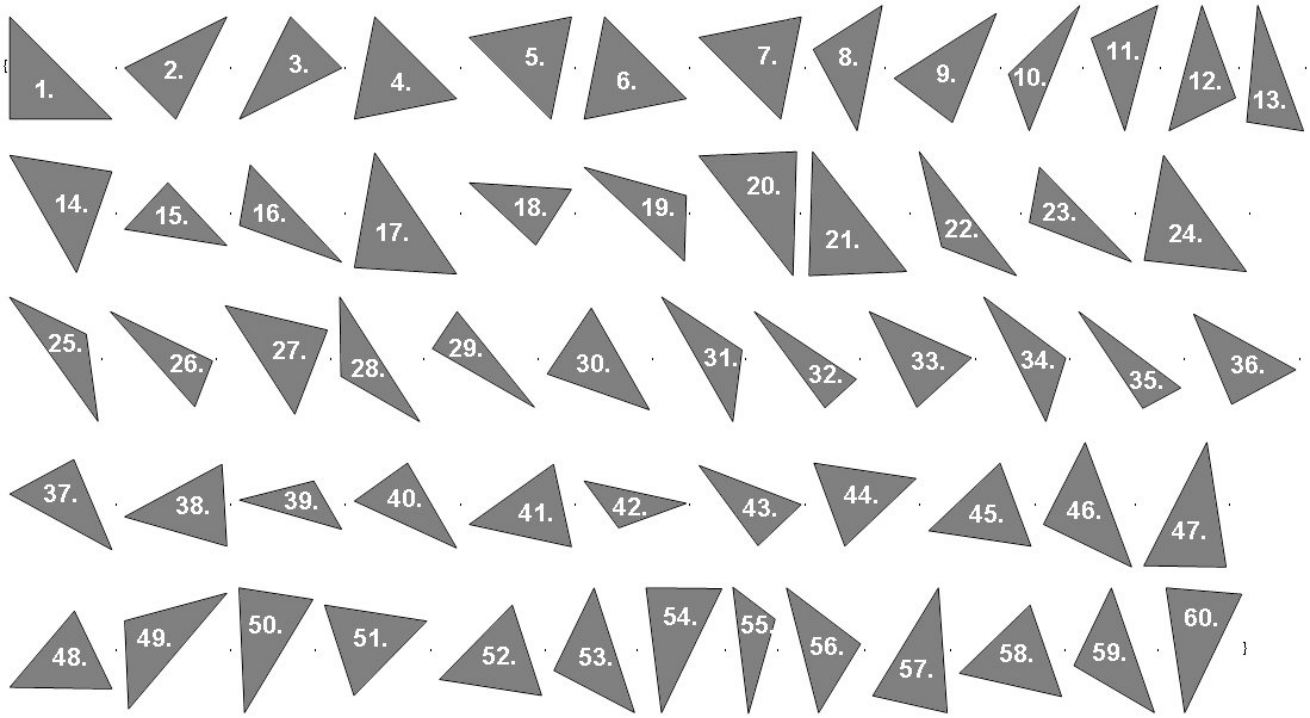

Abbildung 5.5: Alle Simplexe

In der Abbildung 5.5 werden die Dreiecke der 60 Iterationen von der Testfunktion aufgezeigt. Da die Dreiecke hier alle gleich groß gezeichnet sind, werden die Größenverhältnisse nicht real dargestellt.

6 Anwendungen vom Nelder/Mead-Neumann-Algorithmus

Im folgenden werden numerische Ergebnisse des Nelder/Mead-Neumann-Verfahrens mit Ergebnissen des klassischen Nelder/Mead-Verfahrens verglichen. Da das Nelder/Mead-Verfahren in der Software Mathematica als Verfahren der Funktion „NMinimize“ zur globalen Optimierung implementiert ist, werden auch numerische Ergebnisse des Aufrufs dieser Funktion aufgeführt. Da jedoch auch auf Nachfrage keine Details zur Implementierung des Verfahrens zu erfahren waren, ob es sich um den Standardalgorithmus wie im Kapitel 3 beschrieben handelt oder um ein erweitertes Nelder/Mead-Verfahren entsprechend der im Kapitel 3.5 aufgeführten Möglichkeiten, wurde der Standardalgorithmus zum Nelder/Mead-Verfahren eigens in Mathematica implementiert, um bessere Vergleichsmöglichkeiten zu bekommen. Insgesamt werden in den nachfolgenden Kapiteln mit den gewählten Testfunktionen das Nelder/Mead-Verfahren von Mathematica, das Nelder/Mead-Verfahren gemäß Kapitel 3 ohne Erweiterungen und das neue Nelder/Mead-Neumann-Verfahren von Kapitel 5 getestet und verglichen.

6.1 Berechnungen verschiedener Funktionen

6.1.1 Festlegungen

Zum besseren Vergleich werden für die meisten der folgenden Funktionen, für die das Optimum errechnet wird, gleiche Parameterwerte verwendet. Es gelte in den drei getesteten Varianten des Nelder/Mead-Verfahrens für die Reflektionskonstante γ=1 , für die Expansionskonstante β=6 und für die Kontraktionskonstante α=0,5.

Die Verfahren werden je 50 mal aufgerufen, um den Vorteil der nicht-deterministischen Arbeitsweise des Nelder/Mead-Neumann-Verfahrens aufzuzeigen. In diesen 50 Läufen werden grundsätzlich zunächst bis zu 60 Iterationen des Nelder/Mead-Algorithmus durchlaufen und innerhalb der Iterationsschritte der Neumann-Algorithmus, der entsprechend Kapitel 5 dann im Nelder/Mead-Algorithmus eingebunden ist, höchstens $4 \cdot 20$ mal aufgerufen. Das bedeutet, dass im Nelder/Mead-Neumann-Verfahren zusammen höchstens 4800 Iterationsschritte durchlaufen werden. Falls die Iterationszahl nicht ausreicht, um ein Abbruchkriterium zu erfüllen, wird dies an der entsprechenden Stelle gekennzeichnet und die Iterationszahl erhöht.

Als Startpunkte für die zweidimensionalen Testfunktionen seien alle Punkte $P_S \in \{(0|0), (3|3), (5, 3), (-3|-3), (20|20)\}$ gewählt, für die 50-dimensionale Funktion der Startpunkt $P_S = (-8, \ldots, -8)$.

6.1.2 Himmelblau-Funktion

Die zweidimensionale Funktion von Himmelblau [HIM72] hat mehrere lokale Maximalpunkte und vier globale Minimalpunkte. Der dreidimensionale Graph und der zweidimensionale Konturplot in Abbildung 6.1 zeigen einen Ausschnitt der Funktion, an dem die vier Koordinaten der globalen Optima eingezeichnet sind.

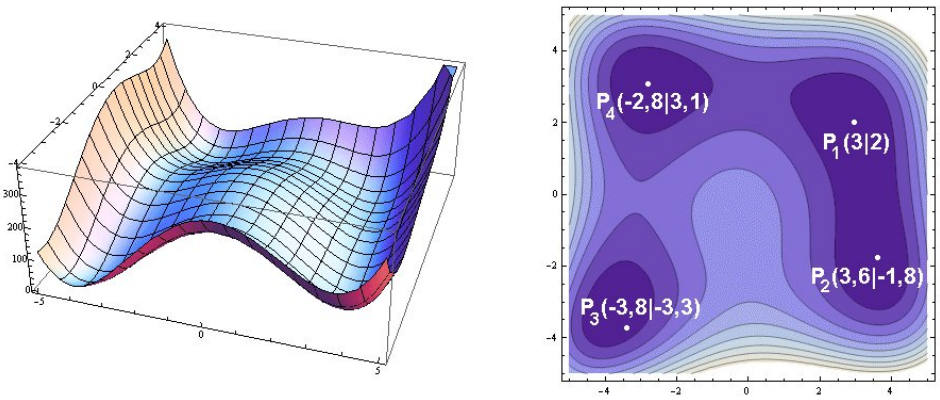

Abbildung 6.1: Himmelblau-Funktion

Die zugehörige Funktionsgleichung lautet:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Diese Funktion hat, wie analytisch nachweisbar ist, vier globale Minimalpunkte:

$$P_1(3|2),\ P_2(3,58443|-1,84813),\ P_3(-3.77931|-3.28319) \text{ und } P_4(-2,80512|3,13131)$$

jeweils mit dem Funktionswert $f(x_1, x_2) = 0$. Das Programm Mathematica bietet die Funktion *NMinimize* an, die den Nelder/Mead-Algorithmus implementiert hat, und man bekommt folgendes Ergebnis:

Eingabe:	$NMinimize[(x^2 + y - 11)^2 + (x + y^2 - 7)^2, \{x, y\}]$
Ausgabe:	$\{0., \{x- > 3., y- > 2.\}\}$

Der von Mathematica errechnete Optimalpunkt ist $P(3|2)$ mit dem Funktionswert $f(x_1, x_2) = 0$. Der Startpunkt liegt bei $P_S(0.870404|0.952376)$. Die von Mathematica berechneten Startpunkte sind immer im Bereich 0 bis 1. Wie oben bereits erwähnt ist es leider nicht möglich gewesen, die genaue Implementierung des Nelder/Mead-Verfahrens herauszubekommen. So ist bei diesem Startpunkt lediglich aus der Hilfe ersichtlich, dass ein Zufallszahlengenerator einen Startpunkt erzeugt. Dies ist jedoch

nicht befriedigend, da die Vergleichbarkeit der drei zu testenden Verfahren gewährleistet sein soll. Daher besteht auch die Möglichkeit, den Startpunkt frei zu wählen.

Bei Wahl des Startpunktes $P_S = (0|0)$ ergibt sich die in folgender Abbildung 6.2 gezeigte Punkteabfolge in den Iterationsschritten. Der Startpunkt ist blau, der gefundene Lösungspunkt grün eingezeichnet. Das Verfahren terminiert im Lösungspunkt $P(3|2)$.

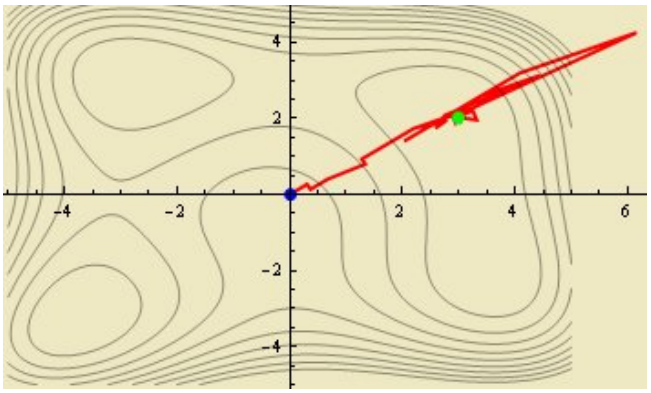

Abbildung 6.2: Suchverlauf der Himmelblau-Funktion

Entsprechend ergeben sich weitere Ergebnisse, die in folgender Tabelle zusammengefasst sind. Wie zu sehen ist, ermittelt der Algorithmus zwei der vier globalen Minima.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*
Nelder/Mead-Mathematica	$P_S(0\|0)$	$P(3\|2)$	0
Nelder/Mead-Mathematica	$P_S(3\|3)$	$P(3\|2)$	0
Nelder/Mead-Mathematica	$P_S(-3\|-3)$	$P(-3.77931\|-3.28319)$	$\sim 10^{-31}$
Nelder/Mead-Mathematica	$P_S(20\|20)$	$P(-3.77931\|-3.28319)$	$\sim 10^{-31}$

Als nächstes wird der Nelder/Mead-Algorithmus wie in Kapitel 3 beschrieben angewendet. Durchlaufen werden wie auch in den weiteren Berechnungen 60 Iterationen. Abbildung 6.3 zeigt den Startsimplex, der in ein Expansionsdreieck übergeht, usw. Auch hier wird nur einer der vier Optimalpunkte errechnet und zwar der Punkt $P(3|2)$. Bei Wahl der vier verschiedenen Startpunkte landet man ebenso wie beim Test mit der Mathematica Funktion *NMinimize* letztendlich bei zwei der vier globalen Optima.

Zum Vergleich der Startpunkte zeigt die Tabelle die Auswertungen. Die Funktionswerte liegen hier fast bei $f(x_1, x_2) = 0$. In der letzten Spalte ist die absolute Häufigkeit angegeben, mit der die Punkte bei 50 Läufen ermittelt wurden.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(0\|0)$	$P(3\|2)$	$4.57324 * 10^{-13}$	50
Nelder/Mead	$P_S(3\|3)$	$P(3\|2)$	$4.46079 * 10^{-16}$	50
Nelder/Mead	$P_S(-3\|-3)$	$P(-3.77931\|-3.28319)$	$7.80074 * 10^{-15}$	50
Nelder/Mead	$P_S(20\|20)$	$P(3\|2)$	$2.7519 * 10^{-13}$	50

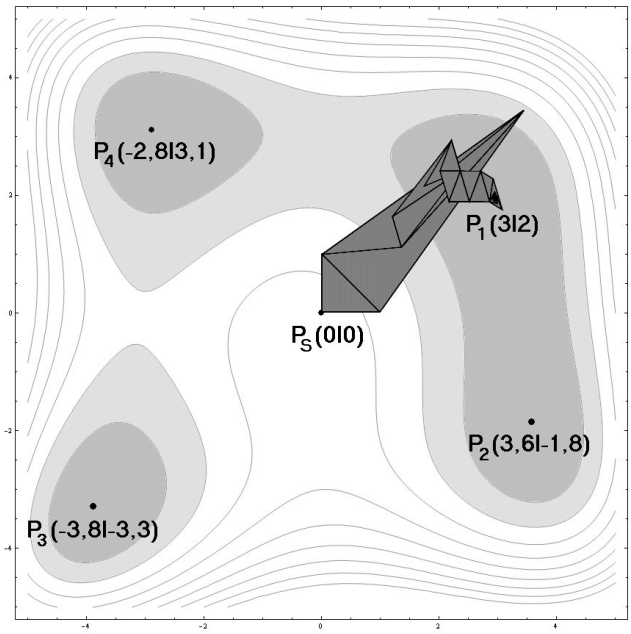

Abbildung 6.3: Himmelblau mit Nelder/Mead

Mit dem Nelder/Mead-Neumann-Algorithmus können alle vier globalen Optima (P_1, P_2, P_3, P_4) gefunden werden (siehe Abbildung 6.4), wenn ein Startpunkt gewählt wird, der vom Nullpunkt weiter weg liegt, wie z.B. hier $P_S(20|20)$. Eine Stärke des Verfahrens liegt also darin, vom gleichen Startpunkt aus und ohne Änderung von Verfahrensparametern mehrere Lösungen finden zu können. Handelt es sich dabei sogar um verschiedene globale Optimalpunkte, ist das eine wichtige Erweiterung zum bisherigen Nelder/Mead-Verfahren.

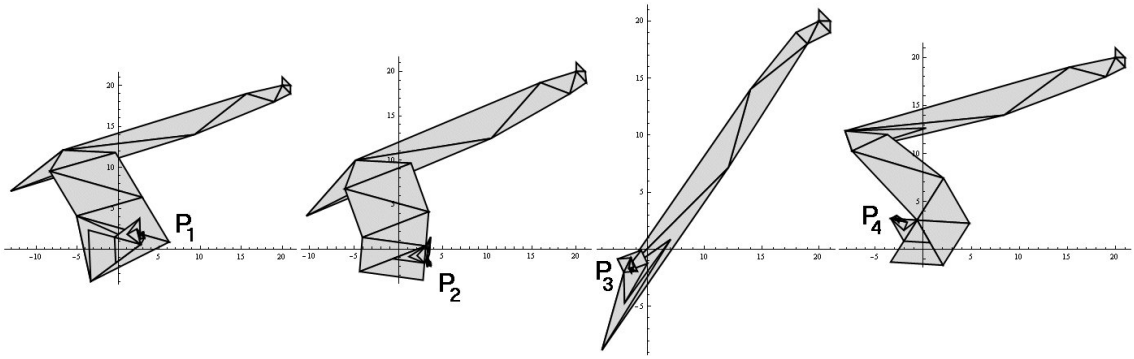

Abbildung 6.4: Himmelblau mit Nelder/Mead-Neumann

Es zeigt sich, dass je weiter man vom zentralen Geschehen entfernt beginnt das adaptierte Nelder/Mead-Verfahren umso besser funktioniert. Dass bei einer Wahl des Startpunktes im Bereich $[-3, 3] \times [-3, 3]$ das Verfahren stets genau einen Optimalpunkt liefert, liegt an den Eigenschaften des Nelder/Mead-Verfahrens, die auch trotz der gewählten Art der stochastischen Erweiterung überwiegen.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead-Neumann	$P_S(0\|0)$	$P(3\|2)$	$3.76722 * 10^{-13}$	50
Nelder/Mead-Neumann	$P_S(3\|3)$	$P(3\|2)$	$4.46079 * 10^{-16}$	50
Nelder/Mead-Neumann	$P_S(-3\|-3)$	$P(-3.77931\|-3.28319)$	$7.80074 * 10^{-15}$	50
Nelder/Mead-Neumann	$P_S(20\|20)$	$P_1(3,00001\|2,00002)$	$1.86916 * 10^{-8}$	23
		$P_2(3,58443\|-1,84813)$	$6.18634 * 10^{-11}$	7
		$P_3(-3,77931\|-3,28319)$	$1.56996 * 10^{-10}$	8
		$P_4(-2,80512\|3,13131)$	$6.27283 * 10^{-11}$	12

Durch Variation und Einstellen der Parameter kann das Verhalten eines Verfahrens begünstigt werden. Beim Nelder/Mead-Verfahren und seinen Erweiterungen, also auch beim Nelder/Mead-Neumann-Verfahren können beispielsweise die Konstanten α, β und γ angepasst werden. Bei der Himmelblau-Funktion hat sich gezeigt, dass die Variation der Konstanten das Verhalten bei der Suche nach den Optima von den gleichen Startpunkten aus nicht verändert.

6.1.3 Rosenbrock-Funktion

Die Rosenbrock-Funktion ist eine nichtlineare Funktion und die Suche nach Optimalpunkten konvergiert extrem langsam bei bestimmten Verfahren wie dem Gradientenverfahren (vgl. Kapitel 2.4.1). Es gibt ein globales Minimum, das in einem bananenartig geformten Tal liegt, das sich längs der Parabel $x_2 = x_1^2$ erstreckt. Wie in Abbildung 6.5 eingezeichnet ist, liegt das Minimum im Punkt $P(1|-1)$ und besitzt dabei einen Funktionswert von $f(x_1, x_2) = 0$.

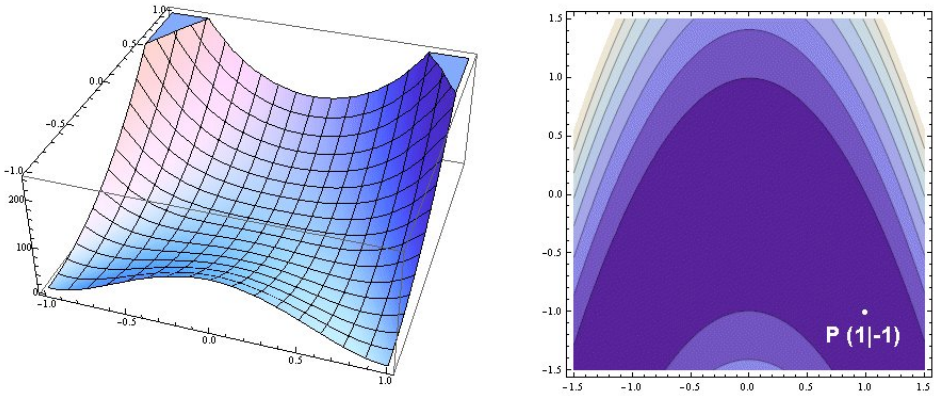

Abbildung 6.5: Rosenbrock-Funktion

Die Rosenbrock-Funktion ist wie folgt definiert [ROS60]:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = 100(x_2 + x_1^2)^2 + (1 - x_1)^2$$

Auch die Berechnung mit Mathematica ergibt beim Startpunkt $P_S(0.365626|0.132704)$ das gleiche Optimum:

Eingabe: $NMinimize[100(y+x^2)^2+(1-x)^2,\{x,y\}]$
Ausgabe: $\{0.,\{x->1.,y->-1.\}\}$

Bei den Berechnungen mit dem Nelder/Neumann-Algorithmus wurde unabhängig vom Startpunkt immer das gleiche und richtige Ergebnis erzielt. Beim Nelder/Mead-Algorithmus gibt es Abweichungen vom Optimum, wenn Startpunkte gewählt werden, die zu weit vom Optimum wegliegen. Abbildung 6.6 zeigt die Berechnung mit Mathematica mit dem Startpunkt $P_S(3|3)$ und der Lösung $P(1|-1)$.

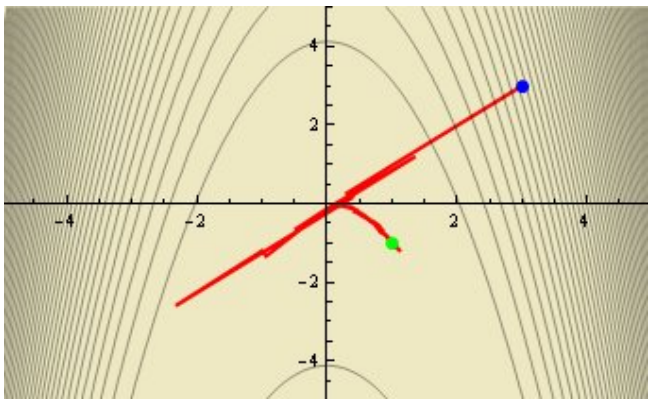

Abbildung 6.6: Suchverlauf der Rosenbrock-Funktion

Erhöht man bei der Rosenbrock-Funktion die Anzahl der Nelder/Mead-Iterationen von 60 auf 200 erhält man genauere Ergebnisse, die in der Tabelle aufgezeigt werden. Dies liegt daran, dass für die Erfüllung des Konvergenzkriteriums die 60 Iterationen nicht ausreichen.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(0\|0)$	$P(1\|-1)$	0	50
Nelder/Mead	$P_S(3\|3)$	$P(1\|-1)$	0	50
Nelder/Mead	$P_S(-3\|-3)$	$P(1\|-1)$	$5.68226*10^{-30}$	50
Nelder/Mead	$P_S(20\|20)$	$P(1\|-1)$	$2.88078*10^{-16}$	50
Nelder/Mead-Neumann	$P_S(0\|0)$	$P(1\|-1)$	0	50
Nelder/Mead-Neumann	$P_S(3\|3)$	$P(1\|-1)$	0	50
Nelder/Mead-Neumann	$P_S(-3\|-3)$	$P(1\|-1)$	0	50
Nelder/Mead-Neumann	$P_S(20\|20)$	$P(1\|-1)$	0	50

Der Wert der Expansionskonstanten β hat bei der Rosenbrock-Funktion keinen Einfluss auf die Ergebnisse.

6.1.4 Six-Hump-Camel-Back-Funktion

Die Six-Hump-Camel-Back-Funktion hat sechs lokale und zwei globale Minima [WIE07], die sich symmetrisch gegenüberliegen. Die Funktionsgleichung der Six-Hump-Camel-Back-Funktion ist gegeben durch:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = x_1^2 * (4 - 2,1 * x_1^2 + x^{\frac{4}{3}}) + x_1 * x_2 + x_2^2 * (-4 + 4 * x_2^2))$$

Mathematica errechnet mit dem Startpunkt $P_S(0.304936|0.266141)$ als Ergebnis nur ein Optimum und zwar bei $P(0.089842| - 0.712656)$ mit dem Funktionswert $f(x_1, x_2) = -1.03163$.

Eingabe: $NMinimize[x^2 * (4 - 2.1 * x^2 + x^{4/3}) + x * y + y^2 * (-4 + 4 * y^2), \{x, y\}]$
Ausgabe: $\{-1.03163, \{x- > 0.089842, y- > -0.712656\}\}$

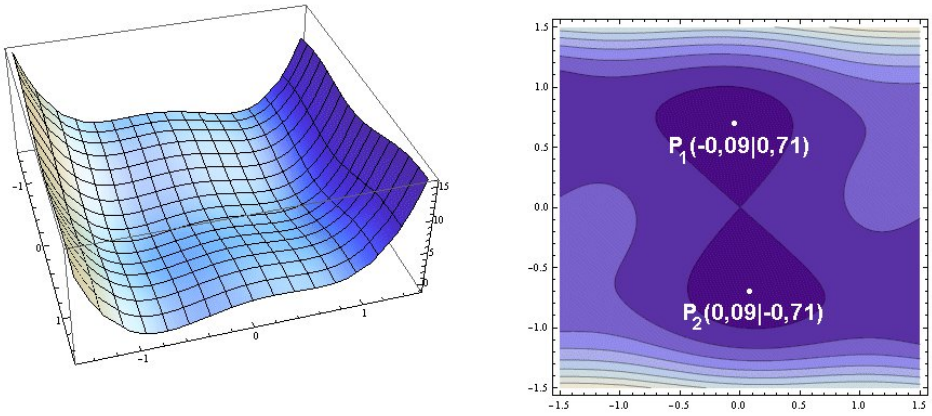

Abbildung 6.7: Six-Hump-Camel-Back-Funktion

Der Nelder/Mead-Algorithmus (gemäß Kapitel 3.4) findet auch nur ein Optimum.

In der Abbildung 6.8 ist der Verlauf mit Startpunkt $P_S(-3| - 3)$ aufgezeigt.

Wie schon bei der Rosenbrock-Funktion weichen die Ergebnisse aber ab, wenn Startpunkte gewählt werden, die zu weit vom Optimum entfernt sind. Der Nelder/Mead-Neumann-Algorithmus findet hingegen bei einem weiter entfernten Startpunkt, wie hier $P_S(20|20)$, beide Optima. Der zusätzlich gefundene Punkt $P_3(1.70361| - 0.796081)$ hat einen schlechteren, also höheren, Funktionswert $f(x_1, x_2) = -0.215464$. Er ist also nur ein lokales Mimimum und ist zu vernachlässigen, weil der Funktionswert schlechter ist, als die der beiden anderen Punkte mit $f(x_1, x_2) = -1.03163$. Somit können lokale Optima anhand des Funktionswertes bestimmt werden. Ein weiteres Beispiel hierzu ist der Startpunkt $P(3|3)$ vom Nelder/Mead-Algorithmus. Er verfängt sich lokal im Punkt $P(0.5| - 0.5)$. Da er den Funktionswert $f(x_1, x_2) = -0.126042$ hat ist er eindeutig ein lokales Minimum und kein globales.

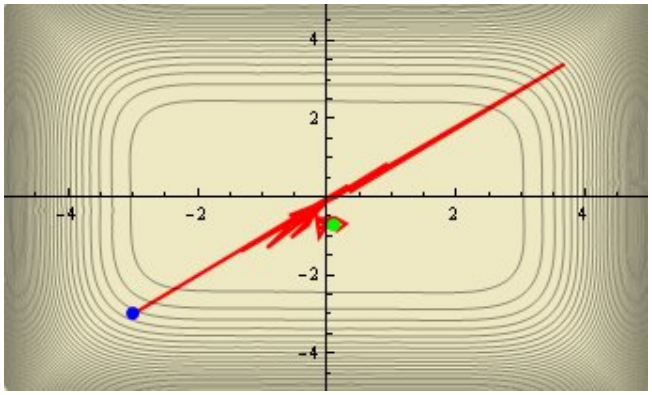

Abbildung 6.8: Suchverlauf der Six-Hump-Camel-Back-Funktion

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(0\|0)$	$P(-0.089842\|0.712656)$	-1.03163	50
Nelder/Mead	$P_S(3\|3)$	$P(0.5\|-0.5)$	-0.126042	50
Nelder/Mead	$P_S(-3\|-3)$	$P(-0.089842\|0.712656)$	-1.03163	50
Nelder/Mead	$P_S(20\|20)$	$P(1.70361\|-0.796084)$	-0.215464	50

Variationen der Expansionskonstanten ergeben wie auch schon bei der Himmelblau-Funktion bei Startpunkten nahe des Optimums keinen Unterschied. Deshalb wird in folgender Tabelle der Startwert $P_S(20|20)$ mit verschiedenen Werten für β betrachtet. Für mehr Genauigkeit wird die Anzahl der Nelder/Mead-Iterationen von 60 auf 100 erhöht.

β	*Ergebnis*	*Funktionswert*
2	$P(0.089842\|-0.712656)$	-1.03163
3	$P(-0.089842\|0.712656)$	-1.03163
4	$P(0.089842\|-0.712656)$	-1.03163
5	$P(0.089842\|-0.712656)$	-1.03163
6	$P(1.70361\|-0.796084)$	-0.215464
7	$P(0.089842\|-0.712656)$	-1.03163
8	$P(-0.089842\|0.712656)$	-1.03163
9	$P(-0.089842\|0.712656)$	-1.03163
10	$P(0.089842\|-0.712656)$	-1.03163
30	$P(-0.089842\|0.712656)$	-1.03163
50	$P(-0.089842\|0.712656)$	-1.03163
100	$P(-0.0892437\|0.711825)$	-1.03162

Wie aus der Tabelle zu sehen ist, werden bei verschiedenen β abwechselnd die beiden Optima gefunden. Ein Ausreißer ist nur β=6 mit dem lokalen Optimum bei $P(1.70361|-0.796084)$ und dem

Funktionswert $f(x_1, x_2) = -0.215464$.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead-Neumann	$P_S(0\|0)$	$P(-0.089842\|0.712656)$	-1.03163	50
Nelder/Mead-Neumann	$P_S(3\|3)$	$P_1(0.0898421\| - 0.712656)$	-1.03163	15
		$P_2(1.70361\| - 0.796084)$	-0.215464	35
Nelder/Mead-Neumann	$P_S(-3\| - 3)$	$P(-0.089842\|0.712656)$	-1.03163	50
Nelder/Mead-Neumann	$P_S(20\|20)$	$P_1(-0.0897982\|0.712642)$	-1.03163	24
		$P_2(0.0898375\| - 0.712655)$	-1.03163	25
		$P_3(1.70361\| - 0.796081)$	-0.215464	1

Der Nelder/Mead-Neumann errechnet bei verschiedenen β die gleichen Optima.

6.1.5 Shekel-Funktion

Die Shekel-Funktion ist eine gute Testfunktion, weil sie neben dem einen globalen Optimum mehrere lokale Optima hat, die nahe am globalen Optimum liegen. Es gibt daher viele Suchalgorithmen, die sich in den lokalen Optima verfangen. Abbildung 6.9 zeigt die Berge und Täler auf. Die Funktionsgleichung ist wie folgt:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = -\sum_{i=1}^{10} \left(\frac{1}{(x_1 - a_i)^2 + (x_2 - b_i)^2 + c_i - 0{,}5} \right)$$

Für die in der Funktion enthaltenen Parameter wurden die Werte $a = \{4, 2.5, 7.5, 8, 2, 2, 4.5, 8, 9.5, 5\}$, $b = \{4, 3.8, 5.6, 8, 1, 8.5, 9.5, 1, 3.7, 0.3\}$ und $c = \{0.7, 0.73, 0.76, 0.79, 0.82, 0.85, 0.88, 0.91, 0.94, 0.97\}$ gewählt [HOR95].

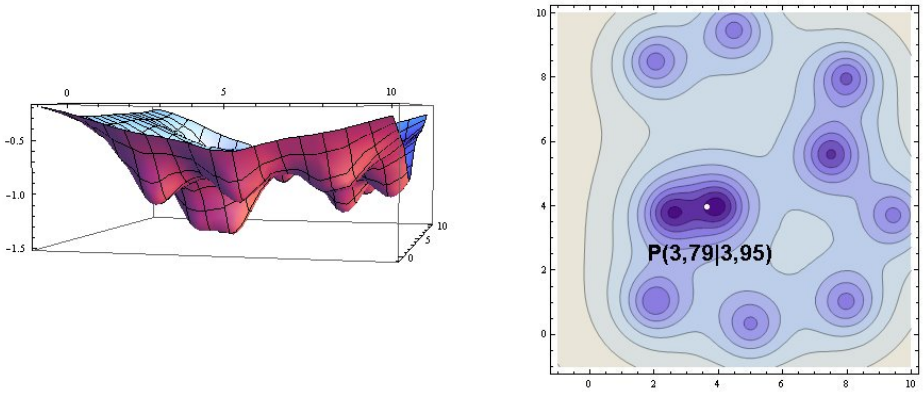

Abbildung 6.9: Shekel-Funktion

Mathematica verfängt sich lokal im Punkt $P(2.61683|3.79966)$ mit dem Funktionswert $f(x_1, x_2) = -2.05933$ bei Verwendung des Startpunktes $P_S(0.870404|0.952376)$.

Eingabe: $NMinimize[f[-Sum[1/((x-a[[i]])^2+(y-b[[i]])^2+c[[i]]),i,1,10]],\{x,y\}]$
Ausgabe: $\{-2.05933,\{x->2.61683,y->3.79966\}\}$

Der Nelder/Mead-Algorithmus findet außer beim negativen Startpunkt $P_S(-3|-3)$ das globale Optimum $P(3.79312|3.95113)$ mit dem Funktionswert $f(x_1,x_2)=-1.50219$. Der Suchverlauf der Shekel-Funktion mit dem Startpunkt $P_S(-3|-3)$ und dem richtigen Optimum ist anhand der Abbildung 6.10 ersichtlich.

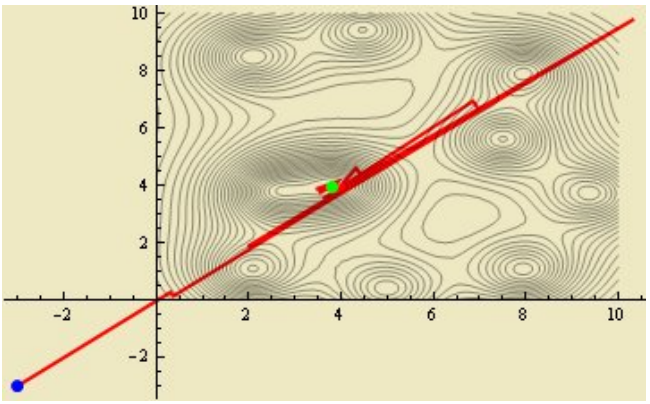

Abbildung 6.10: Suchverlauf der Shekel-Funktion

In den folgenden Tabellen wird die Shekel-Funktion mit 200 Iterationen gerechnet, wobei die β-Werte variiert werden.

Startpunkt	β	*Ergebnis*	*Funktionswert*
$P_S(0\|0)$	3	$P(3.79312\|3.95113)$	-1.50219
	6	$P(3.79312\|3.95113)$	-1.50219
	10	$P(3.79312\|3.95113)$	-1.50219
	30	$P(3.79312\|3.95113)$	-1.50219
	50	$P(2.09283\|1.0929)$	-1.13808
	100	$P(2.09283\|1.0929)$	-1.13808

Von niedrigeren β-Werten wird beim Startpunkt $P_S(0|0)$ immer das globale Optimum gefunden. Liegen die β-Werte höher, verfängt sich der Algorithmus lokal.

Startpunkt	β	*Ergebnis*	*Funktionswert*
$P_S(3\|3)$	3	$P(3.79312\|3.95113)$	-1.50219
	6	$P(3.79312\|3.95113)$	-1.50219
	10	$P(3.79312\|3.95113)$	-1.50219
	30	$P(3.79312\|3.95113)$	-1.50219
	50	$P(3.79312\|3.95113)$	-1.50219
	100	$P(3.79312\|3.95113)$	-1.50219

Beim Startpunkt $P_S(3|3)$ wird unabhängig von der Wahl des Expansionskonstantens immer das globale Optimum gefunden, weil es dem sehr nahe liegt.

Startpunkt	β	*Ergebnis*	*Funktionswert*
$P_S(-3\|-3)$	3	$P(4.98723\|0.403857)$	-1.08006
	6	$P(7.50537\|5.61157)$	-1.29241
	10	$P(3.79312\|3.95113)$	-1.50219
	30	$P(3.79312\|3.95113)$	-1.50219
	50	$P(2.09283\|1.0929)$	-1.13808
	100	$P(2.09283\|1.0929)$	-1.13808

Der Nelder/Mead-Algorithmus findet mit dem Startpunkt $P_S(-3|-3)$ bei β=10 und 30 das globale Optimum. Um dies aus den verschiedenen Ergebnissen ausfindig zu machen, müssen die Funktionswerte verglichen werden. Der Wert -1.50219 ist hier der niedrigste. Bei den Ergebnissen des Startpunktes $P_S(20|20)$ muss ebenso verfahren werden.

Startpunkt	β	*Ergebnis*	*Funktionswert*
$P_S(20\|20)$	3	$P(3.79312\|3.95113)$	-1.50219
	6	$P(3.79312\|3.95113)$	-1.50219
	10	$P(2.84129\|3.80849)$	-1.46424
	30	$P(3.79312\|3.95113)$	-1.50219
	50	$P(3.79312\|3.95113)$	-1.50219
	100	$P(7.94497\|7.89035)$	-1.15169

Im Weiteren wird nun die Shekel-Funktion mit dem Nelder/Mead-Neumann-Algorithmus berechnet. Wie in folgender Tabelle zu sehen ist, wird bis auf β=3 und 6 jedesmal das globale Optimum gefunden.

Startpunkt	β	*Ergebnis*	*Funktionswert*	*Anzahl*
$P_S(0\|0)$	3	$P(2.09283\|1.0929)$	-1.13808	50
	6	$P(2.84129\|3.80849)$	-1.46424	50
	10	$P_1(3.79312\|3.95113)$	-1.50219	48
		$P_2(2.84129, 3.80849)$	-1.46424	2
	30	$P_1(3.79312\|3.95113)$	-1.50219	46
		$P_2(2.84129\|3.80849)$	-1.46424	3
		$P_3(7.50537\|5.61157)$	-1.29241	1
	50	$P_1(3.79312\|3.95113)$	-1.50219	41
		$P_2(7.50537\|5.61157)$	-1.29241	3
		$P_3(2.09283\|1.0929)$	-1.13808	2
		$P_4(2.84129\|3.80849)$	-1.46424	4
	100	$P_1(3.79312\|3.95113)$	-1.50219	54
		$P_2(7.50537\|5.61157)$	-1.29241	3
		$P_3(2.84129\|3.80849)$	-1.46424	3

Wie schon beim Nelder/Mead-Algorithmus wird auch beim Nelder/Mead-Neumann beim Startpunkt $P_S(3|3)$ bei jedem β-Wert das globale Optimum und überhaupt kein lokales Optimum gefunden. Auf eine Aufstellung in einer Tabelle wird deshalb darauf verzichtet.

Bei den folgenden zwei Tabellen mit den Startpunkten $P_S(-3|-3)$ und $P_S(20|20)$ wird unabhängig von der Expansionskonstanten immer das globale aber auch lokale Optima gefunden.

Startpunkt	β	*Ergebnis*	*Funktionswert*	*Anzahl*
$P_S(-3\|-3)$	3	$P_1(2.09283\|1.0929)$	-1.13808	7
		$P_2(2.84129\|3.80849)$	-1.46424	6
		$P_3(3.79312\|3.95113)$	-1.50219	30
		$P_4(4.98723\|0.403857)$	-1.08006	4
		$P_5(7.94262\|1.07376)$	-1.07255	3
	6	$P_1(2.09283\|1.0929)$	-1.13808	5
		$P_2(2.84129\|3.80849)$	-1.46424	5
		$P_3(3.79312\|3.95113)$	-1.50219	35
		$P_4(9.37329\|3.7249)$	-1.0775	5
	10	$P_1(2.09283\|1.0929)$	-1.13808	11
		$P_2(3.79312\|3.95113)$	-1.50219	39
	30	$P_1(3.79312\|3.95113)$	-1.50219	45
		$P_2(7.50537\|5.61157)$	-1.29241	5

Startpunkt	β	*Ergebnis*	*Funktionswert*	*Anzahl*
	50	$P_1(2.09283\|1.0929)$	-1.13808	2
		$P_2(3.79312\|3.95113)$	-1.50219	45
		$P_3(7.50537\|5.61157)$	-1.29241	2
		$P_4(9.37329\|3.7249)$	-1.0775	1
	100	$P_1(2.09283\|1.0929)$	-1.13808	2
		$P_2(3.79312\|3.95113)$	-1.50219	36
		$P_3(7.50537\|5.61157)$	-1.29241	12

Startpunkt	β	*Ergebnis*	*Funktionswert*	*Anzahl*
$P_S(20\|20)$	3	$P_1(2.09283\|1.0929)$	-1.13808	2
		$P_2(3.79312\|3.95113)$	-1.50219	30
		$P_3(4.98723\|0.403857)$	-1.08006	2
		$P_4(7.50537\|5.61157)$	-1.29241	9
		$P_5(7.94262\|1.07376)$	-1.07255	7
	6	$P_1(2.84129\|3.80849)$	-1.46424	1
		$P_2(3.79312\|3.95113)$	-1.50219	35
		$P_3(7.50537\|5.61157)$	-1.29241	11
		$P_4(7.94262\|1.07376)$	-1.07255	3
	10	$P_1(2.09283\|1.0929)$	-1.13808	1
		$P_2(2.84129\|3.80849)$	-1.46424	4
		$P_3(3.79312\|3.95113)$	-1.50219	29
		$P_4(7.50537\|5.61157)$	-1.29241	10
		$P_5(7.94262\|1.07376)$	-1.07255	4
		$P_6(9.37329\|3.7249)$	-1.0775	2
	30	$P_1(2.09283\|1.0929)$	-1.13808	2
		$P_2(2.84129\|3.80849)$	-1.46424	2
		$P_3(3.79312\|3.95113)$	-1.50219	23
		$P_4(4.98723\|0.403857)$	-1.08006	3
		$P_5(7.50537\|5.61157)$	-1.29241	12
		$P_6(7.94262\|1.07376)$	-1.07255	7
		$P_7(9.37329\|3.7249)$	-1.0775	1
	50	$P_1(2.84129\|3.80849)$	-1.46424	2
		$P_2(3.79312\|3.95113)$	-1.50219	30
		$P_3(7.50537\|5.61157)$	-1.29241	14
		$P_4(7.94262\|1.07376)$	-1.07255	1
		$P_5(9.37329\|3.7249)$	-1.0775	3

Startpunkt	β	*Ergebnis*	*Funktionswert*	*Anzahl*
	100	$P_1(2.84129\|3.80849)$	-1.46424	1
		$P_2(3.79312\|3.95113)$	-1.50219	23
		$P_3(4.46667\|9.42203)$	-1.0576	1
		$P_4(4.98723\|0.403857)$	-1.08006	4
		$P_5(7.50537\|5.61157)$	-1.29241	17
		$P_6(7.94262\|1.07376)$	-1.07255	2
		$P_7(9.37329\|3.7249)$	-1.0775	1

6.1.6 Wellenfunktion

Dass sich auch der Nelder/Mead-Neumann-Algorithmus lokal verfangen kann wird am Beispiel dieser Funktion deutlich. Diese Funktion ist eine Beispielfunktion vom Programm Mathematica.

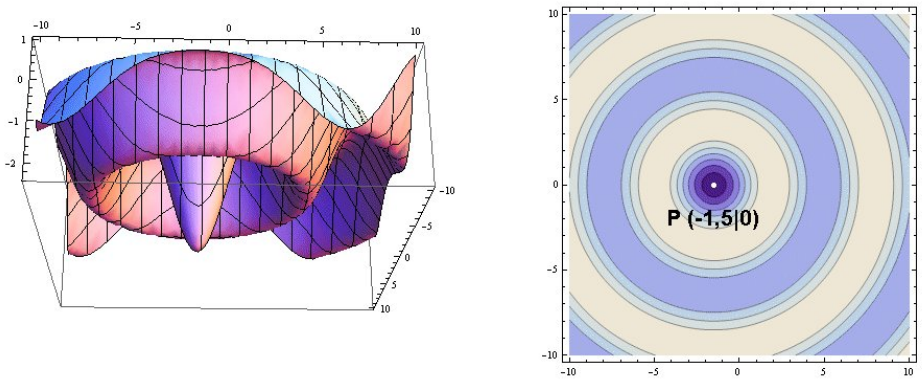

Abbildung 6.11: Weitere Funktion

Die Funktionsgleichung für die in Abbildung 6.11 gezeigte Funktion ist wie folgt definiert:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = Cos[Sqrt[x_1^2 + x_2^2 + 3x_1]]$$

Der in Mathematica implementierte Nelder/Mead-Algorithmus errechnet das richtige Optimum, das bei $P(-1.5|0)$ liegt und den Funktionswert $f(x_1, x_2) = -2.352409615243247$ hat. Gestartet wird bei dem Punkt $P_S(0.304936|0.266141)$.

Eingabe:	$NMinimize[-(Cos[Sqrt[x^2 + y^2 + 3x]]), \{x, y\}]$
Ausgabe:	$\{-2.352409615243247, \{x- > -1.5, y- > 0\}\}$

Der Nelder/Mead-Algorithmus erzielt beim Startpunkt $P_S(-3|-3)$ das richtige Optimum. Bei der Wahl der anderen drei Test-Startpunkte ist zu beobachten, dass sie sich lokal verfangen.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(0\|0)$	$P(-0.658884\|0.309372)$	-1.81494	50
Nelder/Mead	$P_S(3\|3)$	$P(3.79909\|3.69434)$	-1	50
Nelder/Mead	$P_S(-3\|-3)$	$P(-1.5\|1.14533 * 10^-10)$	-2.35241	50
Nelder/Mead	$P_S(20\|20)$	$P(20.8678\|22.1109)$	-1	50

Die Startpunkte $P_S(0|0)$ und $P_S(-3|-3)$ finden beim Nelder/Mead-Neumann-Algorithmus das richtige Optimum, wie in der Abbildung 6.12 mit dem Startpunkt $P_S(0|0)$ zu sehen ist.

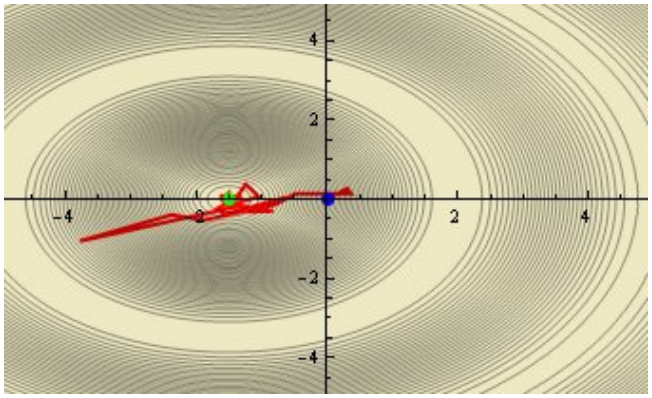

Abbildung 6.12: Suchverlauf der weiteren Funktion

Die Läufe mit den Startpunkten $P_S(3|3)$ und $P_S(20|20)$ verfangen sich lokal. Erhöht man aber bei diesen beiden Startpunkten die Expansionskonstante β auf 100, wird bei 50 Durchläufen zwei Mal das globale Optimum gefunden.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead-Neumann	$P_S(0\|0)$	$P(-1.5\|-1.2041*10^{-}7)$	-2.35241	50
Nelder/Mead-Neumann	$P_S(3\|3)$	$P_1(3.58482\|3.98409)$	-1	1
		$P_2(3.64127\|3.91098)$	-1	1
		$P_3(3.65714\|3.89003)$	-1	1
		$P_4(3.66511\|3.87944)$	-1	1
		$P_5(3.6719\|3.87039)$	-1	1
		$P_6(3.67521\|3.86595)$	-1	1
		$P_7(3.69321\|3.84175)$	-1	1
		$P_8(3.84638\|3.62556)$	-1	1
		$P_9(3.85232\|3.61679)$	-1	1
		$P_10(3.86289\|3.60108)$	-1	1
Nelder/Mead-Neumann	$P_S(-3\|-3)$	$P(-1.5\|5.43365*10^{-}8)$	-2.35241	50
Nelder/Mead-Neumann	$P_S(20\|20)$	$P_1(21.0473\|21.9278)$	-1	1
		$P_2(21.1306\|21.8419)$	-1	1
		$P_3(21.3269\|21.6366)$	-1	1
		$P_4(21.348\|21.6143)$	-1	1
		$P_5(21.3574\|21.6044)$	-1	2
		$P_6(21.3582\|21.6036)$	-1	1
		$P_7(21.6442\|21.2968)$	-1	1
		$P_8(21.7692\|21.1603)$	-1	1
		$P_9(21.9798\|20.9263)$	-1	1

6.1.7 Branin-Funktion

Die Branin-Funktion wird üblicherweise im Intervall von $-5 \leq x_1 \leq 10$ und $0 \leq x_2 \leq 15$ betrachtet und hat dort drei globale Optima [BRA72]. Die Optima liegen bei $P_1(-\pi|12.275)$, $P_2(\pi|2.275)$ und $P_3(3\pi|2.475)$ und haben den Funktionswert $f(x_1, x_2) = 0.397887$. Die Funktionsgleichung lautet:

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - g) \cdot cos(x_1) + e$$

Den Parametern sind folgende Werte zugeordnet: $a = 1$, $b = \frac{5,1}{4 \cdot \pi^2}$, $c = \frac{5}{\pi}$, $d = 6$, $e = 10$ und $g = \frac{1}{8 \cdot \pi}$.

Mathematica findet mit dem Startpunkt $P_S(0.870404|0.952376)$ nur eines der Optima und zwar den Punkt $P(3.14159|2.275)$.

Eingabe: $NMinimize[(y - 5.1/(4\pi^2) \cdot x^2 + 5/\pi \cdot x - 6)^2 + 10 \cdot (1 - 1/(8\pi))cos[x] + 10, \{x, y\}]$

Ausgabe: $\{0.397887, \{x- > 3.14159, y- > 2.275\}\}$

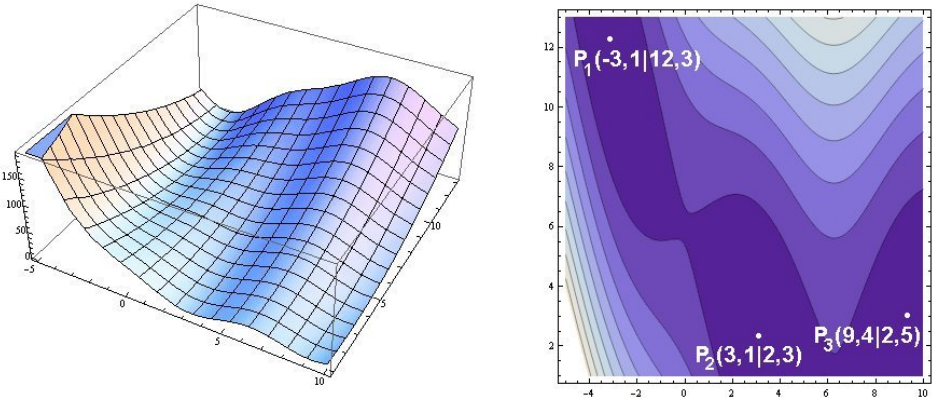

Abbildung 6.13: Branin-Funktion

Abbildung 6.13 zeigt den Suchverlauf bei der Branin-Funktion mit dem Startpunkt $P_S(0|0)$.
Wie aus der Tabelle ersichtlich findet der Nelder/Mead-Algorithmus bei Verwendung von Startpunkten,

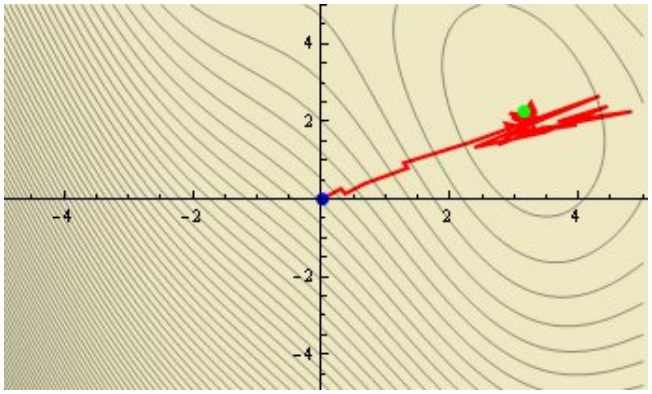

Abbildung 6.14: Suchverlauf der Branin-Funktion

die außerhalb des üblicherweise betrachteten Intervalls von $-5 \leq x_1 \leq 10$ und $0 \leq x_2 \leq 15$ liegen, auch andere Optima als den Punkt $P_2(3.14159|2.275)$. Mit dem Startpunkt $P_S(20|20)$ findet er sogar einen bisher nicht betrachteten $P(15.708|12.875)$. Getestet wurde in dieser Arbeit bis $P_S(500|500)$, bei dem immer noch neue Optima gefunden wurden.

Der Nelder/Mead-Neumann-Algorithmus würde sich hier analog zum Nelder/Mead-Algorithmus verhalten, wenn die Parameter defaultmäßig eingestellt blieben. Verändert man aber die Expansionskonstante β von 3 auf 100, erhält man bei einem Berechnungslauf wesentlich mehr Optima als beim Nelder/Mead-Algorithmus. Wenn also von einer Funktion entferntere Optima gesucht sind, sollte die Konstante β einen höheren Wert haben.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(0\|0)$	$P(3.14159\|2.275)$	0.397887	50
Nelder/Mead	$P_S(3\|3)$	$P(3.14159\|2.275)$	0.397887	50
Nelder/Mead	$P_S(-3\|-3)$	$P(9.42478\|2.475)$	0.397887	50
Nelder/Mead	$P_S(20\|20)$	$P(15.708\|12.875)$	0.397887	50

Die Stärke vom Nelder/Mead-Neumann-Algorithmus besteht darin, dass er mehrere Optimalpunkte vom gleichen Startpunkt in einem Lauf findet.

Verfahren	*Startpunkt*	*Ergebnis*	*Funktionswert*	*Anzahl*
Nelder/Mead	$P_S(30\|30)$	$P(21.9911\|33.475)$	0.397887	50
Nelder/Mead	$P_S(50\|50)$	$P(28.2743\|64.275)$	0.397887	50
Nelder/Mead	$P_S(70\|70)$	$P_1(34.5575\|105.275)$	0.397887	36
		$P_2(-28.2743\|154.275)$	0.397887	14
Nelder/Mead	$P_S(100\|100)$	$P_1(-34.5575\|215.275)$	0.397887	28
		$P_2(40.8407\|156.475)$	0.397887	22
Nelder/Mead	$P_S(200\|200)$	$P_1(-47.1239\|367.875)$	0.397887	28
		$P_2(53.4071\|289.476)$	0.397887	22
Nelder/Mead	$P_S(300\|300)$	$P_1(65.9735\|463.277)$	0.397887	21
		$P_2(-59.6925\|561.315)$	0.397887	27
Nelder/Mead	$P_S(400\|400)$	$P_1(78.539\|677.86)$	0.397887	15
		$P_2(-72.26\|795.606)$	0.397887	35
Nelder/Mead	$P_S(500\|500)$	$P_1(-78.5398\|927.874)$	0.397887	35
		$P_2(84.823\|800.475)$	0.397887	15
Nelder/Mead-Neumann	$P_S(0\|0)$	$P(3.14159\|2.275)$	0.397887	47
		$P_2(15.708\|12.875)$	0.397887	2
		$P_3(28.2743\|64.275)$	0.397887	1
Nelder/Mead-Neumann	$P_S(3\|3)$	$P(3.14159\|2.275)$	0.397887	50
Nelder/Mead-Neumann	$P_S(-3\|-3)$	$P_1(3.14159\|2.275)$	0.397887	48
		$P_2(15.708\|12.875)$	0.397887	2
Nelder/Mead-Neumann	$P_S(20\|20)$	$P_1(-3.14159\|12.275)$	0.397887	2
		$P_2(-9.42478\|32.475)$	0.397887	7
		$P_3(-15.708\|62.875)$	0.397887	1
		$P_4(9.42478\|2.475)$	0.397887	16
		$P_5(15.708\|12.875)$	0.397887	16
		$P_6(-21.9911\|103.475)$	0.397887	8

6.1.8 50-dimensionale-Funktion

Das Verfahren von Nelder/Mead-Neumann soll nun auch anhand höher dimensionaler Beispiele getesetet werden. Dabei ist grundsätzlich das bekannte Problem vom „Fluch der Dimension“ zu berücksichti-

gen. Denn mit zunehmender Dimensionalität eines Problems steigt auch der Rechenaufwand bzw. die Rechenzeit bei der Abarbeitung numerischer Verfahren.

Im folgenden Beispiel wird das Verfahren auf eine 50-dimensionale Funktion angewendet. Der Funktionswert des globalen Minimums liegt bei $f(x) = 0,\ x \in \mathbb{R}^{50}$ [SCH95].

$$f(x) = 1 + 6x_1^2 - \cos(12x_1) + 590 \sum_{i=2}^{50} (x_i - x_{i-1})^2$$

Die Testläufe für diese Funktionsgleichung werden durchgeführt mit verschiedenen Startpunkten, β-Werten, Nelder/Mead-Iterationen und Neumann-Iterationen. Mit dem Startpunkt

$$P_S(5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5|5)$$

wurde in den ersten Testläufen gerechnet. Die Expansionskonstante β wurde auf den Wert 200 gesetzt, damit die Expansion weit rausgeht, weil die Funktion 50-dimensional ist.

Eingabeparameter:	$\beta = 200$, Neumann-Iterationen bis zu 60
Ergebnis der Läufe:	Funktionswert $f(x) = 6.02048$ bei 323000 Nelder/Mead-Iterationen letzter ermittelter Punkt $P_L(0.945011|0.945001|0.945065|0.945058|0.945169|$ $0.945195|0.945177|0.94513|0.945242|0.945184|0.945088|0.945026|0.944712|$ $0.944304|0.944018|0.943636|0.943508|0.943299|0.943142|0.943281|0.943325|$ $0.943448|0.943415|0.943658|0.943624|0.94362|0.943623|0.943649|0.943568|$ $0.943587|0.943616|0.943605|0.943617|0.943747|0.943727|0.943726|0.943726|$ $0.943764|0.944055|0.944297|0.944562|0.944832|0.944849|0.945007|0.945109|$ $0.944973|0.944836|0.944842|0.944838|0.944891)$

Wird die Expansionskonstante weiter erhöht, ergeben die Berechnungen immer den Funktionswert $f(x) = 6.02048$ mit den Punkten des Simplexes ähnlich dem im zuletzt gezeigten Testlauf. Änderungen an der Anzahl der Neumann-Iterationen hatten keine Auswirkungen, weil die Testläufe ergaben, dass bei so hochdimensionalen Funktionen kaum der Neumann-Algorithmus aufgerufen wird. Versuche, erneut beim letzten Simplex anzusetzen und einen β-Wert von 50000 mit 50000 Neumann-Iterationen zu wählen, brachten keine Verbesserung des Simplexes.

Mit dem Startpunkt

$$P_S(0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|$$
$$0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5|0.5)$$

verfing sich der Lauf bei dem Funktionswert $f(x) = 1.51639$. Der zuletzt ermittelte Punkt nach 1000 Durchläufen lag bei

$P_L(0.482216|0.48227|0.482472|0.482684|0.482931|0.483076|0.48318|0.483216|0.483208|$
$0.483224|0.483174|0.48308|0.482952|0.482805|0.482753|0.482683|0.482688|0.482633|$
$0.48254|0.482395|0.482249|0.482131|0.481976|0.481852|0.48169|0.481538|0.481425|$
$0.481368|0.481316|0.481334|0.481345|0.481418|0.481538|0.481697|0.481739|0.481881|$
$0.481972|0.482123|0.482198|0.482273|0.482356|0.482446|0.482554|0.48267|0.482802|$
$0.482933|0.48304|0.483163|0.483256|0.483303)$

Im folgenden Lauf wurde der Startpunkt

$$P_S(0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2| \\ 0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2|0.2)$$

gewählt. Errechnet wurde folgender zuletzt ermittelte Punkt, der sehr nahe beim globalen Optimum liegt und den Funktionswert $f(x) = 0.000013808$ hat:

$P_L(0.0000456554|0.0000598754|0.0000650295|0.0000658702|0.0000685479|0.0000485486|$
$0.000041814|0.0000503829|0.0000504418|0.0000395892|0.000025181|0.0000165724|$
$4.38995 \cdot 10^{-6}| - 0.0000103005| - 0.000023939| - 0.0000492421| - 0.0000597659|$
$-0.0000908837| - 0.000114154| - 0.000134544| - 0.000171832| - 0.000210253| - 0.00024144|$
$-0.000262192| - 0.000294413| - 0.000324274| - 0.000365929| - 0.000388308| - 0.000414599|$
$-0.000434984| - 0.000457309| - 0.000475329| - 0.000483248| - 0.000493664| - 0.00049285|$
$-0.000493149| - 0.000496784| - 0.000497164| - 0.00048381| - 0.000469567| - 0.000444802|$
$-0.000430117| - 0.000398079| - 0.000369907| - 0.000335927| - 0.00029711| - 0.000257336|$
$-0.000239601| - 0.000226024| - 0.000221706)$

Ein Testlauf mit dem Startpunkt

$$P_S(0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0)$$

ergab den Funktionswert $f(x) = 0$ mit dem zuletzt ermittelten Punkt

$$P_L(0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0)$$

Bei Verwenden der NMinimize-Funktion von Mathematica wurde gleich das globale Optimum gefunden, weil der Startpunkt von NMinimize nahe dem Nullpunkt liegt.

Um nun aber auch auf die globale Lösung von $f(x) = 0$ mit einem weiter entfernten Startpunkt

zu kommen, wird zur Bildung des Startsimplexes nicht der bisher verwendete Einheitsvektor (je Komponente) addiert, sondern dieser zunächst mit dem Faktor 10 multipliziert. Ausgegangen wird von dem Punkt

$$P_S(20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20|20)$$

Die Anzahl der Nelder-Iterationen wird auf 10000000 und die Anzahl der Neumann-Iterationen beim ersten Testlauf auf maximal 10, um die Eigenschaften vom Nelder/Mead-Algorithmus auszunutzen, und beim zweiten auf maximal 400, um den Neumann-Algorithmus stärker zu gewichten, festgesetzt. Nach 70000 Nelder/Mead-Neumann-Iterationen hat der erste Lauf den Funktionswert $f(x) = 1.53745$ und der zweite den Funktionswert $f(x) = 1.21776$ mit folgenden P_L:

$P_{L1}(0.481313|0.481184|0.481086|0.481002|0.480809|0.480571|0.480146|0.479605|$
$0.478974|0.478297|0.477615|0.476956|0.476322|0.475578|0.474713|0.473741|0.472777|$
$0.471662|0.47041|0.46919|0.468116|0.466983|0.46584|0.464666|0.463585|0.462555|$
$0.461527|0.460403|0.459146|0.458045|0.456871|0.455685|0.454505|0.453348|0.452274|$
$0.451247|0.450222|0.449202|0.44826|0.447397|0.446594|0.445988|0.44539|0.444809|$
$0.444259|0.443917|0.443713|0.443502|0.443332|0.443395)$

$P_{L2}(-0.0112674| - 0.0135535| - 0.0186087| - 0.0238486| - 0.0284535| - 0.0344945|$
$-0.0415091| - 0.0497673| - 0.0582865| - 0.0676448| - 0.076059| - 0.0844995|$
$-0.0930987| - 0.101749| - 0.110374| - 0.118321| - 0.125885| - 0.133778| - 0.142633|$
$-0.151186| - 0.160616| - 0.170127| - 0.17926| - 0.188672| - 0.196239| - 0.204482|$
$-0.212871| - 0.222272| - 0.231726| - 0.240951| - 0.250189| - 0.259088| - 0.267497|$
$-0.275009| - 0.281324| - 0.287398| - 0.293229| - 0.298437| - 0.304682| - 0.308469|$
$-0.310943| - 0.313978| - 0.316458| - 0.317362| - 0.318463| - 0.318431| - 0.317687|$
$-0.317603| - 0.317583| - 0.317625)$

Aus diesen Läufen lässt sich schlussfolgern, dass bei einem größeren Simplex der Neumann-Algorithmus öfter aufgerufen wird und somit wie in diesem Beispiel zum Ergebnis führt. Wird der zweite Testlauf fortgeführt ergibt sich nach 425000 Nelder/Mead-Neumann-Iterationen ein Funktionswert von $f(x) = 0.000203751$ und der zuletzt ermittelte Punkt liegt dann bei:

$P_{L3}(-0.0000917924| - 0.000106955| - 0.000125188| - 0.000139848| - 0.000160016|$
$-0.000189654| - 0.000218909| - 0.000247886| - 0.000284206| - 0.000321915|$
$-0.000358035| - 0.000404003| - 0.000459929| - 0.000523462| - 0.000583475|$
$-0.000635614| - 0.000691199| - 0.000764764| - 0.000854507| - 0.000950303|$

$$-0.00105935| - 0.00117121| - 0.00128689| - 0.0014177| - 0.00153722| - 0.00166532|$$
$$-0.00180197| - 0.00191933| - 0.00203676| - 0.00216001| - 0.0022819| - 0.00240911|$$
$$-0.00253773| - 0.00265377| - 0.00277| - 0.00288988| - 0.00299154| - 0.00307256|$$
$$-0.00315989| - 0.00324076| - 0.00331983| - 0.00338811| - 0.00345205| - 00351081|$$
$$-0.00356464| - 0.00361107| - 0.00366956| - 0.00369392| - 0.00371742| - 0.00371987)$$

An dieser Stelle kann der Lauf abgebrochen werden, weil zu erkennen ist, dass es zum Ziel führt.

6.2 Beispiel aus dem Wasserwesen

Für die vorzustellenden Optimierungsläufe wurden folgende Parameterwerte gewählt: $\beta = 6$ und maximal 20 Neumann-Aufrufe je Iteration. Der Algorithmus wurde für die Beispiele aufgrund der aus der Vielzahl an Läufen der 50-dimensionalen Funktion (vgl. letzter Abschnitt) gewonnenen Erkenntnisse durch einen Simplex-Reset bei Erfüllung einer festgelegten Bedingung erweitert. Der Simplex-Reset kann auch als Restart des Algorithmus mit dem aktuellen Optimalpunkt als Startpunkt gesehen werden. Ein solcher Restart erschien bedingt durch die große Iterationenzahl gewinnbringend zu sein. Die zu optimierende Funktion ist auf Basis einer nichtlinearen Regression festgelegt. Zuletzt sei noch angemerkt, dass dies das erste Beispiel ist, bei dem a-priori nicht bekannt ist, wohin die Suche nach dem Optimalpunkt gehen soll, d.h. wo das globale Optimum zu finden ist.

6.2.1 Die Regressionsanalyse zur Parameterbestimmung

Eine Regression beschreibt den Zusammenhang zwischen einer Zielgröße η und einer oder mehreren Ausgangsvariablen $\zeta_1, \zeta_2, ..., \zeta_n$. Das allgemeine Modell ist wie folgt definiert:

$$\eta = g(x_1, x_2, ..., x_m; \zeta_1, \zeta_2, ..., \zeta_n)$$

Dabei ist $g : \mathbb{R}^n \to \mathbb{R}$ eine geeignete Funktion, die von den Ausgangsvariablen $\zeta_1, \zeta_2, ..., \zeta_n$ und von den $m \in \mathbb{N}$ zu bestimmenden Modellparametern $x_1, x_2, ..., x_m$ abhängt. Um die Parameter festzulegen, werden Messwerte herangezogen.

Gegeben seien $k \in \mathbb{N}$ Messwerte $(\zeta_{1j}, \ldots, \zeta_{nj}, \eta_j)$, $j = 1, ..., k$. Soll beispielsweise für $n = 1$ ein linearer Zusammenhang zwischen den ζ- und den η-Werten bestimmt werden, ist der Ansatz

$$\hat{\eta}(\zeta) := g(x_1, x_2; \zeta) := x_1\zeta + x_2$$

zu wählen. Abbildung 6.15 zeigt schematisch eine mögliche Gerade entlang vier fiktiv gewählter Messpunkte.

Das Modell passt eine Funktion an die Messpunkte an. Dies ist mit Fehlern verbunden. Der Fehler an

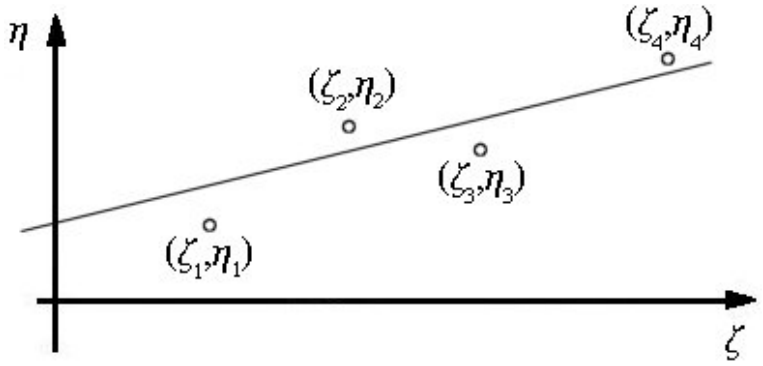

Abbildung 6.15: Lineare Regression, [ALT02]

den Messpunkten beträgt $\epsilon_j := \eta_j - \hat{\eta}_j$. Weil also aufgrund von Messfehlern in der Regel nicht alle Messwerte exakt auf einer Geraden liegen, möchte man $x = (x_1, x_2)^T \in \mathbb{R}^2$ so bestimmen, dass die zugehörige Gerade möglichst optimal zu den Messwerten passt, der Gesamtfehler $\epsilon := \epsilon_1 + \ldots + \epsilon_k$ minimal ist [ALT02]. Eine Möglichkeit der Anpassung besteht in der Methode der kleinsten Quadrate.

In der linearen Regression werden also Funktionen betrachtet, die linear sind in den Variablen. In der nichtlinearen Regression werden Funktionen untersucht, die sich nicht als lineare Funktionen in den Variablen schreiben lassen. Es bestehen im Prinzip unbeschränkte Möglichkeiten, das Modell anzusetzen. Wird die Methode der kleinsten Quadrate gewählt, gilt es, die Funktion

$$f : \mathbb{R}^n \to \mathbb{R}, \quad (x_1, \ldots, x_n) \mapsto \sum_{j=1}^{k} \left(g(x_1, \ldots, x_m; \zeta_{1j}, \ldots, \zeta_{nj}) - \eta_j\right)^2$$

zu minimieren. Das ist gleichbedeutend damit, den Gesamtfehler ϵ möglichst klein zu halten.

6.2.2 Anpassung von Infiltrationsversuchen

Anhand eines in der Praxis auftretenden Beispiels soll das Nelder/Mead-Neumann-Verfahren ein letztes Mal in dieser Arbeit überprüft werden. Es geht dabei um die Anpassung einer unbekannten Funktion an gegebene Datenpunkte, die aus einer Vielzahl von Messungen stammen. Es handelt sich um ein Beispiel aus dem Bereich des Wasserwesens. Der Einfluss der Beschaffenheit eines Bodens auf die Versickerungseigenschaften von Wasser soll untersucht werden. Abbildung 6.16 zeigt den Zusammenhang zwischen der gemessenen Infiltrationsrate η und der gemessenen kumulativen Niederschlagsenergie ζ des Niederschlages aus mehreren Experimenten. Dabei wurden insgesamt $k = 5929$ Messwerte (ζ, η) erhoben, die Einzelmessungen basierten jeweils auf ca. 50 Messwerten.

Anhand der Abbildung ist zu erkennen, dass hierbei in jedem Fall eine nichtlineare Funktion anzupassen ist. Die Anpassung der Funktion soll mittels einer nichtlinearen Regression erfolgen. Im folgenden sollen zwei Modellansätze untersucht werden. Im ersten wird eine Exponentialfunktion mit drei Modellparametern und einer Variablen angesetzt. Auf Basis dieser Funktion werden die drei Parameter ihrerseits

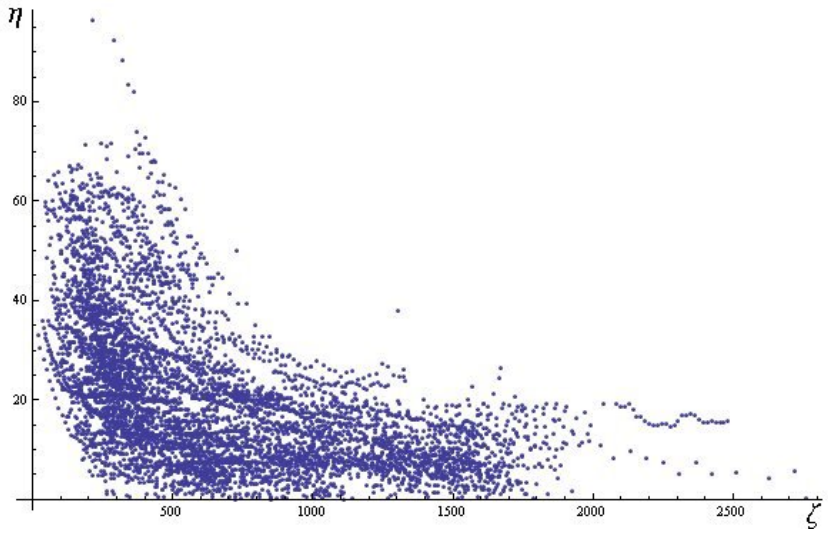

Abbildung 6.16: Punktewolke der Messdaten

durch Funktionen mit weiteren Parametern und Variablen festgelegt, so dass insgesamt im zweiten Modell eine Funktion mit neun Modellparametern und vier Variablen betrachtet wird. Sämtlichen Variablen liegen 5929 Messwerte zugrunde.

6.2.2.1 Modell mit drei Parametern und einer Variablen

Die potentielle Infiltrationsrate η $[mm/h]$ soll durch folgende Parameter beschrieben werden: Anfangsinfiltrationsrate x_1 $[mm/h]$, Neigung zur Krustenbildung x_2 $[m^2/J]$ und Endinfiltrationsrate x_3 [mm/h]. Die Modellvariable ist die kumulative Niederschlagsenergie ζ $[J/m^2]$. Ein mögliches mathematisches Modell dafür stellt folgende Gleichung dar:

$$\hat{\eta}(\zeta) := (x_1 - x_3) \cdot e^{-x_2 \cdot \zeta} + x_3$$

Um eine Regressionsgerade über die 5929 gemessenen Werte legen zu können, wird mittels Mathematica mit dem implementierten Nelder/Mead-Neumann-Algorithmus folgende Funktion nach dem Prinzip der kleinsten Quadrate minimiert [ZIM08]:

$$f(x_1, x_2, x_3) = \sum_{j=1}^{5929} \left((x_1 - x_3) \cdot e^{-x_2 \cdot \zeta_j} + x_3 - \eta_j\right)^2$$

Die Variablen x_1 bis x_3 sind dabei die zu optimierenden Parameter des Modells.

Eine frühere Anpassung desselben Modells an die Messungen eines einzelnen Experimentes haben Parameterwerte ergeben, die nun als Startwerte dienen sollen. Die Startwerte liegen somit dadurch bedingt bei $x_1 = 251.98$, $x_2 = 0.01$ und $x_3 = 11.28$. Der Funktionswert beim Start der Berechnung liegt bei $f(x_1, x_2, x_3) = 4681810$. Nach 84 Iterationen des Algorithmus ergibt sich $f(x_1, x_2, x_3) = 807408$ mit

den Parameterwerten $x_1 = 48.6234$, $x_2 = 0.00243391$ und $x_3 = 8.4132$.

Die sich ergebende Regressionsgerade $\hat{\eta}(\zeta) = 40.2102 \cdot e^{-0.0024 \cdot \zeta} + 8.4132$ ist in der Abbildung 6.17 rot markiert.

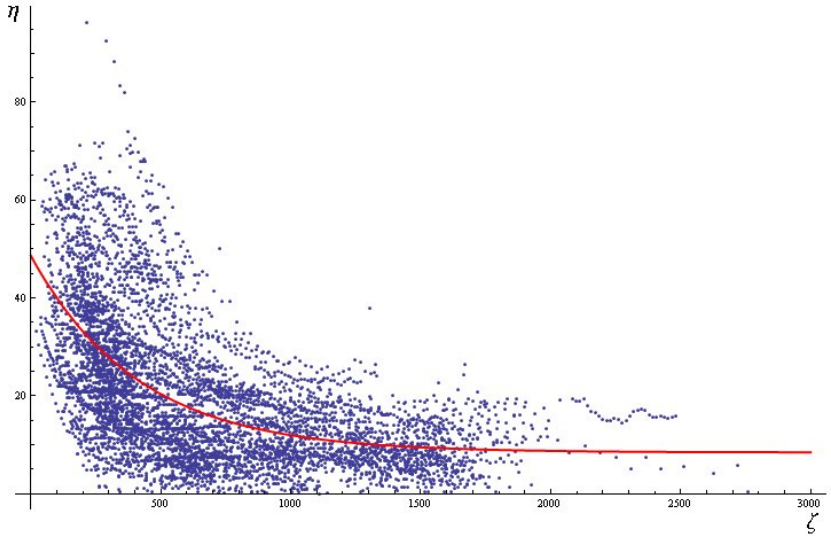

Abbildung 6.17: Nichtlineare Regression mit drei Parametern

Durch das Einbeziehen weiterer Bodenparametervariablen lässt sich möglicherweise eine noch bessere Anpassung einer Funktion an die Messwerte erzielen.

6.2.2.2 Modell mit neun Parametern und vier Variablen

Das eben betrachtete Beispiel soll nun differenzierter betrachtet werden. Deshalb wird es nun mit neun statt nur drei Parametern und vier anstelle der einen Variablen beschrieben. Dabei sollen nun die eben verwendeten Parameter $\hat{x}_1, \hat{x}_2$ und $\hat{x}_3$ ihrerseits funktional beschrieben werden durch

$$\begin{aligned} \hat{x}_1 &:= x_1 \cdot \zeta_2^{x_2}, \\ \hat{x}_2 &:= x_3 \cdot \zeta_3 + x_4 \cdot \zeta_2 + x_5, \\ \hat{x}_3 &:= x_6 \cdot \zeta_2^{x_7} \cdot \zeta_3^{x_8} \cdot \zeta_4^{x_9}. \end{aligned}$$

Es sind damit die neun Modellparameter x_1 bis x_9 empirisch zu bestimmen. Die Variablen des Modells sind durch die Tage seit Bearbeitung des Bodens ζ_2, den geometrischen Mittelwert der Korngrößenverteilung ζ_3 $[\mu m]$, den auftretenden Sandanteil ζ_4 und die kumulative Niederschlagsenergie ζ_1 wie oben gegeben. Die kumulative Niederschlagsenergie ζ wird jetzt also durch ζ_1 dargestellt. Das Gesamtmodell lautet dann

$$\begin{aligned} \hat{\eta}(\zeta_1, \zeta_2, \zeta_3, \zeta_4) &:= (x_1 \cdot \zeta_2^{x_2} - (x_3 \cdot \zeta_3 + x_4 \cdot \zeta_2 + x_5)) \cdot \\ &\quad \cdot e^{-x_6 \cdot \zeta_2^{x_7} \cdot \zeta_3^{x_8} \cdot \zeta_4^{x_9} \cdot \zeta_1} + x_3 \cdot \zeta_3 + x_4 \cdot \zeta_2 + x_5. \end{aligned}$$

Mittels Mathematica wird mit dem Nelder/Mead-Neumann-Algorithmus folgende Funktion minimiert:

$$\begin{aligned} f(x_1, x_2, ..., x_9) &= \sum_{j=1}^{5929} [(x_1 \cdot \zeta_{2j}^{x_2} - (x_3 \cdot \zeta_{3j} + x_4 \cdot \zeta_{2j} + x_5)) \cdot \\ &\quad \cdot e^{-x_6 \cdot \zeta_{2j}^{x_7} \cdot \zeta_{3j}^{x_8} \cdot \zeta_{4j}^{x_9} \cdot \zeta_{1j}} + x_3 \cdot \zeta_{3j} + x_4 \cdot \zeta_{2j} + x_5 - \eta_j]^2 \end{aligned}$$

Mit den Startwerten $x_1 = 61.18$, $x_2 = 0.4$, $x_3 = 0.35$, $x_4 = -0.23$, $x_5 = 10$, $x_6 = 0.025$, $x_7 = -0.1$, $x_8 = -0.1$, $x_9 = -0.1$, die wie im letzten Abschnitt aus einer früheren Anpassung stammen, und dem Funktionswert $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) = 4758980$ starten die Berechnungen. Nach 25000 Läufen ergibt sich dann für $x_1 = 54.9717$, $x_2 = -0.00151116$, $x_3 = 0.0629739$, $x_4 = -0.121298$, $x_5 = 9.62608$, $x_6 = 0.0100247$, $x_7 = -1.00096$, $x_8 = 0.68163$, $x_9 = -0.191601$ und der Funktionswert $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$=663115.

Da es keinen festen Wert für die Variablen ζ_2, ζ_3 und ζ_4 gibt, soll aus den Messwerten der Median genommen werden, da dieser nicht ausreißerempfindlich ist. Das Modell soll nun zurückgeführt werden auf das Modell des letzten Abschnittes, um den Zusammenhang zwischen der Infiltrationsrate η und der kumulativen Niederschlagsenergie ζ (bzw. ζ_1) zu ermitteln. Die Parameter $\hat{x}_1$, $\hat{x}_2$ und $\hat{x}_3$ errechnen sich wie folgt [ZIM08]:

$$\begin{aligned} \hat{x}_1 &:= x_1 \cdot \underbrace{\text{Median}(\zeta_2)}_{34.4194}^{x_2} = 54.6789, \\ \hat{x}_2 &:= x_3 \cdot \underbrace{\text{Median}(\zeta_3)}_{31.8875} + x_4 \cdot \underbrace{\text{Median}(\zeta_2)}_{34.4194} + x_5 = 0.00177909, \\ \hat{x}_3 &:= x_6 \cdot \underbrace{\text{Median}(\zeta_2)}_{34.4194}^{x_7} \cdot \underbrace{\text{Median}(\zeta_3)}_{31.8875}^{x_8} \cdot \underbrace{\text{Median}(\zeta_3)}_{26.5096}^{x_9} = 7.12075 \end{aligned}$$

Die sich ergebende Regressionsgerade $\hat{\eta}(\zeta) = 47.5582 \cdot e^{-0.0018 \cdot \zeta} + 7.1208$ ist in der folgenden Abbildung 6.18 zu sehen.

Einen Vergleich der beiden Modelle zeigt die Abbildung 6.19, wobei die rote Regressionslinie das Modell aus dem letzten Abschnitt zeigt. Das Modell mit den neun Parametern liefert auf Basis der Methode der kleinsten Quadrate eine bessere Anpassung an die Daten.

Zur Illustration des Optimierungsprozesses betrachte man die Abbildung 6.20. Dort ist eine orange gezeichnete Regressionslinie zu sehen. Diese Linie entspricht der Startlösung. Im weiteren Verlauf (dargestellt durch die roten Linien) zeigte sich im betrachteten Bereich für ζ nahezu eine Linearisierung des Modells, was sich durch einen sehr kleinen Wert für x_3 (bisweilen galt dabei $0 < |x_3| < 10^{-10}$) und damit verbunden einen nur geringen Einfluss der Exponentialfunktion auf den Funktionswert bemerkbar machte. Erst im hinteren Drittel der Iterationen verstärkte sich dieser Einfluss wieder und endete schließlich in den angegebenen Werten, zu sehen in der grünen Linie.

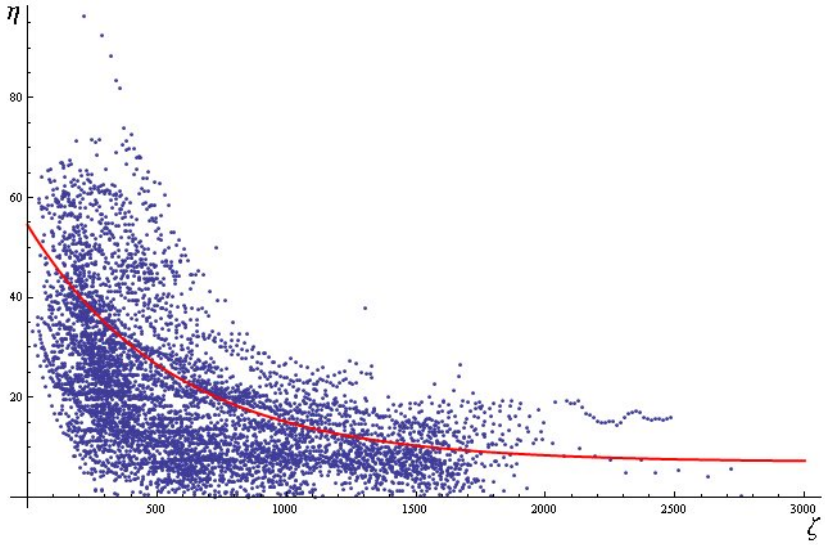

Abbildung 6.18: Nichtlineare Regression mit neun Parametern

Eine anschließende lokale Optimierung mit dem Newton-Verfahren brachte in keinem der Fälle eine weitere Verbesserung der Modellanpassung. Somit wurde bei diesen Ergebnissen die Untersuchung beendet.

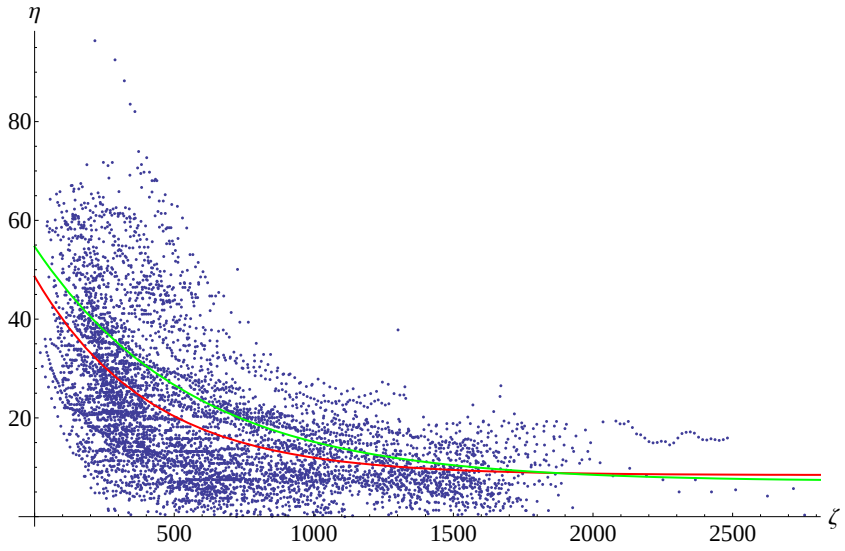

Abbildung 6.19: Vergleich der beiden Regressionslinien

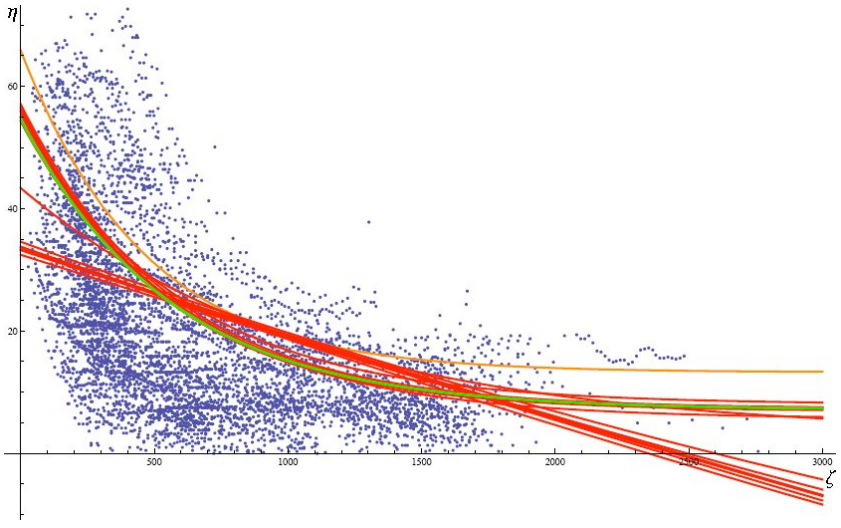

Abbildung 6.20: Iterative Verbesserung gegenüber der Startlösung

7 Ausblick

Auch wenn das wesentliche Ziel der Arbeit, ein ableitungsfreies Verfahren zur globalen Optimierung zu entwickeln, das an die zu optimierende Funktion nichts voraussetzt, erreicht wurde, ist die Fragestellung noch nicht erschöpfend bearbeitet. So könnte der in Kapitel 4.5.4 beschriebene Neumann-Algorithmus nicht nur auf den Nelder/Mead-Algorithmus aus Kapitel 3.4 aufbauen, sondern seperat zur Suche nach dem globalen Optimum verwendet werden. Nachdem mit dem Neumann-Algorithmus aber nur im Zweidimensionalen gerechnet werden kann, zu optimierende Funktionen aber auch bis zu 50-dimensional sein können, würde sich hier eine Intervallbetrachtung der zu optimierenden Funktion anbieten. Die Auswahl der betrachteten Intervalle kann durch das in Kapitel 2.3.4 beschriebene Clusterverfahren bezüglich der Wahrscheinlichkeit eines globalen Optimums optimiert werden.

Eine andere Möglichkeit der Optimierung des in Kapitel 5 beschriebenen Nelder/Mead-Neumann-Algorithmus ist das Rechnen mit einem modifizierten Startpunkt. Beispielsweise rechnet das Programm Mathematica mit der Funktion *NMinimize* Startpunkte, die im Intervall von [0,1] vom Startpunkt, von dem ursprünglich ausgegangen wird, entfernt sind. Die in dieser Arbeit diesbezüglich durchgeführten Tests haben aber keine Verbesserung der Effizienz zeigen können. Auch konnten durch diese Modifikation keine weiteren Optima gefunden werden.

Eine sinnvollere Herangehensweise wäre deshalb anstatt den Startpunkt zu verschieben, die Lage des ersten Simplexes zu verbessern. Das Beispiel der Abbildung 6.3 macht dies deutlich. Wie zu sehen ist, findet der Algorithmus das globale Optimum $P_1(3|2)$, in dessen Richtung das Startdreieck gerichtet ist. Würde es nach unten links ausgerichtet sein, würde sicherlich das Optimum $P_3(-3.8|-3.3)$ gefunden werden. Sinnvoll wäre somit eine Ausrichtung des Startdreiecks bei mehreren Läufen jeweils in verschiedene Richtungen.

Eine zusätzliche Modifikation des Expansionskonstanten β sollte abhängig von der zu optimierenden Funktion durchgeführt werden. Wie das Beispiel der 50-dimensionalen Funktion im Kapitel 6.1.8 zeigt, ist bei mehrdimensionalen Funktionen eine höhere Expansionskonstante zu wählen als bei weniger dimensionalen Funktionen. Außerdem ist bei Mehrdimensionalität die Größe des Startsimplexes entscheidend. Diese kann durch Multiplikation des Einheitsvektors modifiziert werden. Welche Größe für welche Dimensionalität sinnvoll ist, könnte durch weitere Läufe mit verschiedenen Funktionen berechnet werden.

Abbildungsverzeichnis

Literaturverzeichnis

[@CBO] http://www.computerbase.de/lexikon/Optimierung_(Mathematik)

[@FHE] http://www.it.fht-esslingen.de/∼schmidt/vorlesungen/kryptologie/seminar/ws9798/html/pseudo/pseudo-2.html

[@SCH] http://www.schlauweb.de/Konvexkombination

[@WID] http://de.wikipedia.org/wiki/Downhill-Simplex-Verfahren

[@WIM] http://de.wikipedia.org/wiki/Metropolisalgorithmus

[@WIS] http://de.wikipedia.org/wiki/Simplex-Verfahren

[@WIZ] http://de.wikipedia.org/wiki/Zufallszahl

[ALT02] Alt W.: „Nichtlineare Optimierung“, Vieweg Verlag, 2002

[BAC94] Bacher J.: „Clusteranalyse“, Oldenbourg, 1994

[BOE04] Boeker F.: „Diskrete Verteilungen“, Uni Göttingen, 2004

[BOR01] Borgwardt K.: „Optimierung, Operations Research, Spieltheorie“, Birkhäuser, 2001

[BRA72] Branin F.K.: „A widely convergent method for finding multiple solutions of simultaneous nonlinear equations“, IBM J. Res. Develop., pp. 504-522, 1972

[DOM95] Domschke W.: „Einführung in Operations Research“, Springer, 1995

[DOM01] Domschke W.: „Generierung von Wahrscheinlichkeitsverteilungen“, TU Darmstadt, 2001

[GER07] Gerken M.: „Entwurf optoelektronischer Bauelemente“, Universität Karlsruhe, 2007

[GOE06] Görl T.: „Methoden der dimensionalen Analyse und Typenbildung“, Uni Potsdam, 2006

[HIM72] Himmelblau D.: „Applied nonlinear programming“, McGraw-Hill, 1972

[HOT04] Hotho A.: „Knowledge Discovery“, Uni Kassel, 2004

[HOR95] Horst R.: „Introduction to Global Optimization“, Kluwer Academic Publishers, 1995

[JAN03] Janzing D.: „Algorithmentechnik“, Uni Karlsruhe, 2003

[KOE07] Köller F.: „Operations Research“, Uni Hannover, 2007

[KUE08] Kühn R.: „Zufallszahlen“, Uni Heidelberg, 2008

[LAW00] Law A.M.: „Simulation Modeling and Analysis“, Mcgraw-Hill Professional, 2000

[MAD07] Maddah B.: „Simulation“, American University of Beirut, 2007

[NEL65] Nelder J., Mead R.: „A Simplex Method for Function Minimization“, Computer Journal, 7; 308-313, 1965

[RIE05] Riedle M.: „Risikotheorie“, Humboldt-Universität Berlin, 2005

[RIA05] Rieger A.: „Zur Parameteridentifikation komplexer Materialmodelle auf der Basis realer und virtueller Testdaten“, Uni Stuttgart, 2005

[ROS60] Rosenbrock H.: „An Automatic Method for Finding the Greatest or Least Value of a Function“, Computer J. 3, 1960

[RUD08] Rudolph A.: „Nichtlineare Optimierung“, Universität der Bundeswehr München, 2008

[SCH95] Schäffler S.: „Global Optimization Using Stochastic Integration“, S. Roderer Verlag Regensburg, 1995

[THE04] Theus M.: „Multivariate Statistische Verfahren“, Uni Augsburg, 2004

[VOL00] Volkmann M.: „Clusteranalyse“, Uni Karlsruhe, 2000

[WEB07] Weber H.: „Nichtlineare Optimierung“, FH Wiesbaden, 2007

[WEI08] Weise T.: „Global Optimization Algorithms“, Uni Kassel, 2008

[WEI95] Weiß M.: „Nichtlineare Gleichungen“, FH Augsburg, 1995

[WIE07] Wiethoff A.: „Verifzierte globale Optimierung auf Parallelrechnern“, Uni Karlsruhe, 1997

[ZIM03] Zimmer C.: „Die Dimensionalität von Entscheidungen“, Uni Konstanz, 2003

[ZIM08] Zimmermann A.: „Influence of tillage practices on infiltration and runoff processes in agricultural catchment areas“, Universität der Bundeswehr München, 2008