FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Lösung restringierter nichtlinearer Optimierungsprobleme in Maple

Oliver Bücker

FZJ-ZAM-IB-2004-03

Juni 2004

(letzte Änderung: 14.06.2004)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Die Modellierung von Daten, insbesondere mit nichtlinearen Modellfunktionen, ist in vielen Bereichen von Wissenschaft und Technik eine regelmäßig wiederkehrende Aufgabe. In manchen Bereichen ist bei der Suche nach der Lösung darauf zu achten, dass diese gewissen Nebenbedingungen unterliegt. Ein Kennzeichen solcher restringierter Ausgleichsrechnungen ist, dass Lösungen nur durch Iterationsverfahren bestimmt werden können.

Ausreißer in den Eingabedaten beeinträchtigen erheblich die Güte der Anpassungsfunktion. Sie sollten automatisch erkannt und bei der Datenanalyse unberücksichtigt bleiben.

Bisher wurden zur Lösung dieser Problemstellungen Bibliotheksprogramme, wie etwa aus den NAG-Bibliotheken, verwendet. Hierzu waren Kenntnisse der jeweiligen Programmiersprache nötig. Zudem war die Implementierung von benötigten Ableitungen schwierig und fehleranfällig. Heutzutage werden für die mathematische Modellierung integrierte mathematische Softwaresysteme, wie z.B. Maple, eingesetzt. In der derzeitigen Version des Maple-Systems wird die Lösung solcher Probleme jedoch nicht unterstützt.

Im Rahmen dieser Diplomarbeit entstand ein Programmpaket, das zur Lösung restringierter nichtlinearer Optimierungsprobleme sowie zur automatischen Ausreißererkennung symbolische und numerische Methoden kombiniert. Zusätzlich wurde eine grafische Oberfläche entwickelt, mit der die Lösungen auf einfache und intuitive Weise berechnet werden können.

Abstract

Modeling of data is a regular task in many areas of science and engineering. In some areas it has to be considered that there are constraints in the search for the solution. If the model function is nonlinear, a solution cannot be computed directly. Therefore an iterative method must be chosen.

Outliers in the input data may have a big influence on the quality of the fitting function. They should be identified automatically and should be excluded from the data analysis process.

In the past, routines from libraries were used to calculate solutions for these problems. The knowledge of a specific programming language was needed to use theses routines. Furthermore, the implementation of required derivatives was difficult and error-prone. Nowadays, integrated mathematical software systems like Maple are used for mathematical modeling, but the current Maple version has no functionality to solve such problems.

In the context of this diploma thesis a program package has been developed for solving constrained nonlinear fitting problems by combining symbolic and numerical methods. In addition, a graphical user interface to this package enables an easy and intuitive access to the package.

Kapitel 1

Einleitung

Die Modellierung von Daten, insbesondere die mit nichtlinearen Modellfunktionen, ist nicht nur im Forschungszentrum Jülich, sondern auch in vielen anderen Forschungsbereichen eine immer wieder auftretende Problemstellung. Die Lösung des nichtlinearen Ausgleichsproblems kann nicht, wie beispielsweise beim linearen, über eine explizite Rechenvorschrift in einem Schritt bestimmt werden. Hierzu werden die verschiedensten iterativen Algorithmen genutzt. In manchen wissenschaftlichen Bereichen ist bei der Suche nach der Lösung darauf zu achten, dass diese gewissen Restriktionen unterliegt. Im Kapitel 2 wird ein Verfahren zur Lösung solcher restringierter nichtlinearer Approximationsprobleme vorgestellt.

Ausreißer in den Eingabedaten beeinträchtigen erheblich die Güte der Anpassungsfunktion. Sie sollten erkannt und bei der Datenanalyse unberücksichtigt bleiben. Möglichkeiten zur automatischen Ausreißererkennung werden im Kapitel 3 vorgestellt.

In der Vergangenheit wurden zur Lösung nichtlinearer Optimierungsaufgaben unter Nebenbedingungen Bibliotheksroutinen, wie sie die NAGfl90-Bibliothek [11] anbietet, genutzt. Eine visuelle Kontrolle der Ergebnisse war aber nur über zusätzliche Grafikprogramme möglich. Heutzutage geht man immer mehr dazu über, integrierte interaktive Softwaresysteme, wie Maple [16], zur Lösung solcher Problemstellungen einzusetzen. Die Vorteile dieser Programme liegen dabei in den Programmiermöglichkeiten, die flexibel und problemorientiert gestaltet werden können, und den umfangreichen grafischen Ausgabemöglichkeiten.

Wie in Kapitel 2 und 3 zu erkennen ist, werden zur Realisierung der vorgestellten Algorithmen Ableitungen benötigt. Hierbei kann die Benutzung numerischer Ableitungen problematisch werden und die Angabe von exakten Ableitungen eventuell sehr aufwändig und fehleranfällig sein. Diese Problematik und deren Lösung über die automatische Differentiation werden in Kapitel 4 beschrieben. Mit der automatischen Differentiation ist es möglich, effizient Ableitungen von Funktionen und komplexen Programmen zu bilden.

In seinem Ursprung war Maple früher ein reines Computeralgebrasystem, mit dem symbolische Berechnungen durchgeführt werden konnten. Inzwischen hat sich dieses Programmpaket weiterentwickelt und bietet nun die Möglichkeit externe Programme in C, Fortran oder Java über ein Interface in das Maple-System einzubinden. Diese Funktionalität sowie die Programmiermöglichkeiten über Prozeduren und Module werden in Kapitel 5 vorgestellt. Seit Release 8 steht auch ein Tool zur Erstellung von Oberflächen zur Verfügung. Mit diesen Oberflächen, den sogenannten Maplets, ist es dem Endbenutzer möglich, auf einfache und intuitive Art und Weise mit dem System zu kommunizieren. Hierbei muss der Benutzer keine Maple-Kenntnisse haben. Mit einem Überblick über die Erstellung und Funktionalität solcher Maplet-Anwendungen schließt das Kapitel 5.

Die Benutzung von Maple im Gegensatz zu reinen C- oder Fortran- Programmen, stellt eine wesentliche Vereinfachung dar. Leider bietet Maple in der aktuellen Version keine Funktionen zur Lösung von restringierten nichtlinearen Optimierungsaufgaben an. Aus diesem Grund ist im Zuge der vor-

liegenden Diplomarbeit eine Erweiterung des Nonlinear-Fitting-Paketes mit dem Namen `NLFit`, das bereits Grundlage einer Diplomarbeit [19] war, entstanden. Hierzu wurden die in den Kapiteln 4 und 5 vorgestellten Differentiationsmethoden und Programmiertechniken genutzt. Das Einbinden einer externen Bibliotheksfunktion hat zum einen den Vorteil, dass man auf robuste und ausreichend getestete Software zurückgreifen kann, zum anderen können Implementierungen direkt in Maple nicht die Effizienz von übersetzten externen Routinen erreichen. In Kapitel 6 wird beschrieben, wie man auf einfache Art und Weise nichtlineare Optimierungsprobleme unter Nebenbedingungen lösen und ggf. Ausreißer in den Eingabedaten automatisch herausfiltern kann. Die Funktionen können nicht nur über die Oberfläche genutzt werden, sondern auch auf Worksheet-Ebene. Dies bietet dem Benutzer die Möglichkeit, die Algorithmen auch noch in älteren Maple-Versionen nutzen zu können. Die erforderlichen Prozeduren und deren Parameter werden ebenfalls in Kapitel 6 vorgestellt.

Im Kapitel 7 werden Optimierungswerkzeuge aus anderen mathematischen Softwareprodukten verglichen und deren Vor- und Nachteile in Bezug auf das im Rahmen dieser Arbeit entstandene Programmpaket erläutert.

Kapitel 2

Restringierte nichtlineare Optimierung

2.1 Motivation

Bei der nichtrestringierten Approximation wird versucht die Fehlerquadrate zwischen Datenpunkten und Modellfunktion zu minimieren. Ziel ist es, die gegebene Modellfunktion möglichst exakt an die Daten anzunähern. Bei einer solchen Annäherung wird allerdings der wissenschaftliche Hintergrund der anzunähernden Funktion nicht mit in die Betrachtung einbezogen. Aus naturwissenschaftlichen Gründen kann es sein, dass einige Parameter bestimmten Restriktionen unterliegen und nicht *frei* sind, wie z.B. dass ein Parameter nur positive Werte annehmen darf. Im Folgenden werden diese Einschränkungen der Parameter als **Nebenbedingungen** bezeichnet und es wird ein Verfahren zur Lösung solcher Optimierungsprobleme vorgestellt. Grundlage der Überlegungen sind die Publikationen [1], [2] und [3] sowie [5], [6], [8] und [9].

2.2 Nichtlineare Optimierung

Unter einem Optimierungsproblem versteht man die Minimierung bzw. Maximierung einer Funktion. Die Funktionen besitzt die Grundform $f : \mathbb{R}^p \to \mathbb{R}$ und gesucht wird

$$\min f(\boldsymbol{x}) \text{ im } \mathbb{R}^p \qquad \text{bzw.} \qquad \max f(\boldsymbol{x}) \text{ im } \mathbb{R}^p \tag{2.1}$$

Solche Aufgaben werden im Folgenden als *nichtrestringierte Optimierungsprobleme* bezeichnet. Unterschieden werden *lokale Extremwerte* und *globale Extremwerte*.

Die Stelle $\boldsymbol{x}_0 \in \mathbb{R}^p$ kann als *lokale Minimalstelle* bezeichnet werden, wenn es eine ε-Kugel $K(\boldsymbol{x}_0, \varepsilon)$ derart gibt, dass für alle $\boldsymbol{x} \in K(\boldsymbol{x}_0, \varepsilon) \cap \mathbb{R}^p$ gilt

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0)$$

Für eine lokale Maximalstelle muss gleiches für $f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)$ gelten. Natürlich muss eine *lokale Extremstelle* nicht notwendigerweise eine *globale Extremstelle* sein. Was eine *globale Extremstelle* ist, wird in folgender Definition festgelegt.

Definition 2.2.1. *Eine Funktion vom Typ $f : \mathbb{R}^p \to \mathbb{R}$ besitzt im Punkt $\boldsymbol{x}^\star \in \mathbb{R}^p$ eine globale Maximalstelle bzw. Minimalstelle genau dann, wenn für alle $\boldsymbol{x} \in \mathbb{R}^p$ gilt*

$$f(\boldsymbol{x}) \leq f(\boldsymbol{x}^\star) \qquad \textit{bzw.} \quad f(\boldsymbol{x}) \geq f(\boldsymbol{x}^\star)$$

Darüber hinaus nennt man ein globales Extremum strikt, wenn für alle $\boldsymbol{x} \in \mathbb{R}^p$ mit $\boldsymbol{x}^\star \neq \boldsymbol{x}$ gilt: $f(\boldsymbol{x}) < f(\boldsymbol{x}^\star)$, bzw. $f(\boldsymbol{x}) > f(\boldsymbol{x}^\star)$.

Zur Lösung der *nichtrestringierten Optimierungsprobleme* gibt es verschiedene Verfahren:

- direkte Suchverfahren
  - Hooke- und Jeeves-Verfahren
  - Methode von Nelder und Mead
- Abstiegsmethoden
- Liniensuchverfahren
- quasi-Newton-Verfahren

Hier wird nicht weiter auf die verschiedenen Verfahren eingegangen. Informationen über diese Verfahren findet man in [7].

In dem dieser Arbeit zugrunde liegenden Programmpaket wird das Minimum einer Funktion f, die eine Modellfunktion M beinhaltet, gesucht. Es wird diejenige Funktion M gesucht, die die gegebenen Daten $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ möglichst optimal approximiert. Die Modellfunktion M ist gegeben und es müssen die Parameter $a_1, \ldots, a_p$ für diese Funktion bestimmt werden. Minimiert werden soll die Summe der quadratischen Abstände zwischen den Datenpunkten und der Modellfunktion. Gesucht wird somit

$$\min_{\boldsymbol{a}\in\mathbb{R}^p} \quad f(\boldsymbol{a}) \quad = \quad \frac{1}{2}\sum_{i=1}^{n}(M(\boldsymbol{a}, \boldsymbol{x}_i) - y_i)^2 \tag{2.2}$$

Für die Betrachtung des Optimierungsproblems kann man sich also auf die Bestimmung des Minimalwertes beschränken, da sich die Annäherung der Modellfunktion an Datenpunkte in ein solches Minimalwertproblem überführen lässt.

Ein Beispiel für (2.1) besteht in der Bestimmung der optimalen Verteilung von Ressourcen, $r_1, r_2, \ldots, r_p$, die sich gegenseitig beeinflussen und einem speziellen Gesetz unterliegen. Im Allgemeinen sind Ressourcen nicht unbegrenzt. Dieser Umstand bedeutet für die mathematische Realisierung des Optimierungsproblems, dass in einer Teilmenge $\Omega \subset \mathbb{R}^p$ das Minimum der Zielfunktion gesucht werden muss und möglicherweise gewisse zusätzliche Gleichungs- und Ungleichungsbedingungen erfüllt sein sollten.

Für unsere Optimierung müssen also Nebenbedingungen mit in die Betrachtung einbezogen werden. Wenn es diese Restriktionen gibt, nennt man das zu lösende Problem *Optimierungsproblem unter Nebenbedingungen* oder *restringiertes Optimierungsproblem*. Es gilt also für die Zielfunktion f

$$\min_{\boldsymbol{a}\in\Omega} f(\boldsymbol{a}) \quad \text{mit} \quad \Omega \subset \mathbb{R}^p \tag{2.3}$$

Die folgenden Kapitel befassen sich mit Optimierungsproblemen, bei denen Ω durch Nebenbedingungen in Gleichungsform $g(\boldsymbol{a}) = 0$ und/oder Nebenbedingungen in Ungleichungsform $g(\boldsymbol{a}) \leq 0$ charakterisiert wird. Hierbei ist $g : \mathbb{R}^p \to \mathbb{R}^m$, $m \leq p$, eine gegebene Funktion, *Kostenfunktional* genannt, und die Bedingungen $g(\boldsymbol{a}) = 0$ bzw. $g(\boldsymbol{a}) \leq 0$ werden elementweise als $g_i(\boldsymbol{a}) = 0$ für $i = 1, \ldots, m$ und $g_i(\boldsymbol{a}) \leq 0$ für $i = m+1, \ldots, m+k$ verstanden.

2.3 Restringierte Optimierung

Allgemein kann die restringierte Optimierung wie folgt formuliert werden. Gegeben $f : \mathbb{R}^p \to \mathbb{R}$, gesucht wird

$$\min f(\boldsymbol{a}) \quad \text{mit} \quad \boldsymbol{a} \in \Omega \subset \mathbb{R}^p \tag{2.4}$$

Wenn f stetig ist und Ω eine abgeschlossene und beschränkte Menge, so ist die Existenz der Lösung des Problems (2.4) durch den Satz von Weierstraß gesichert.

Satz 2.3.1 (Satz von Weierstraß). *Gegeben sei eine Funktion $f : \Omega \to \mathbb{R}$, wobei $\Omega \subset \mathbb{R}^p$ kompakt ist. Dann besitzt $f(x)$ in Ω ein absolutes Maximum max und ein absolutes Minimum min, d.h. es existiert in Ω wenigstens ein Punkt c und wenigstens ein Punkt d, so dass $\forall x \in \Omega$ gilt:*

$$min = f(d) \leq f(x) \leq f(c) = max$$

Sollte Ω nicht nur abgeschlossen und beschränkt sein, sondern auch eine konvexe Menge bilden, so gelten bestimmte Optimalitätsbedingungen:

Definition 2.3.2. *Eine Funktion $f : \mathbb{R}^p \to \mathbb{R}$ wird als konvex bezeichnet genau dann, wenn $\forall x, y \in \mathbb{R}^p$ mit $\alpha \in [0, 1]$ gilt:*

$$f\left[\alpha x + (1-\alpha)y\right] \quad \leq \quad \alpha f(x) + (1-\alpha) f(y) \tag{2.5}$$

Gilt in der Ungleichung (2.5) sogar $<$ statt $\leq$ für alle $0 < \alpha < 1$, dann nennt man f strikt konvex.

Anschaulich bedeutet die Definition (2.3.2), dass die Verbindungslinie zwischen beliebigen Punktepaaren (x, y), die sich im Bild von f befinden, oberhalb der Funktion f liegen müssen.

Satz 2.3.3. *Sei $\Omega \subset \mathbb{R}^p$ eine konvexe Menge, $\boldsymbol{a}^\star \in \Omega$ und $f \in C^1\left(B(\boldsymbol{a}^\star; R)\right)$ für ein geeignetes $R > 0$.*

1. *Ist $\boldsymbol{a}^\star$ eine lokale Minimalstelle von f, so gilt*

$$\nabla f(\boldsymbol{a}^\star)^T (\boldsymbol{a} - \boldsymbol{a}^\star) \quad \geq \quad 0, \qquad \forall \boldsymbol{a} \in \Omega \tag{2.6}$$

2. *Ist darüber hinaus f auf Ω konvex und erfüllt f die Bedingung (2.6), so ist $\boldsymbol{a}^\star$ eine globale Minimalstelle von f.*

Satz 2.3.4. *Sei $\Omega \subset \mathbb{R}^p$ eine abgeschlossene und konvexe Menge und f eine streng konvexe Funktion in Ω. Dann gibt es eine eindeutig bestimmte lokale Minimalstelle $\boldsymbol{a}^\star \in \Omega$.*

Ein Spezialfall für die Optimierung (2.4) ist, wenn die Menge Ω nur über eine Gleichung bestimmt wird. Es wird zunächst angenommen, dass die Nebenbedingungen g nur aus einer Gleichung $g(\boldsymbol{a}) = 0$ bestehen. Gegeben sei $f : \mathbb{R}^p \to \mathbb{R}$, gesucht wird

$$\min f(\boldsymbol{a}) \quad \text{unter der Nebenbedingung} \quad g(\boldsymbol{a}) = 0 \tag{2.7}$$

wobei $g : \mathbb{R}^p \to \mathbb{R}^m$, mit $m \leq p$, eine gegebene Funktion der Veränderlichen $g_1, \ldots, g_m$ ist.

Definition 2.3.5. *Ein Punkt $\boldsymbol{a}^\star \in \mathbb{R}^p$, für den $g(\boldsymbol{a}^\star) = 0$ gilt, heißt regulär, wenn die Spaltenvektoren der Jacobi-Matrix $J_g(\boldsymbol{a}^\star)$ linear unabhängig sind, wobei $g_i \in C^1\left(B(\boldsymbol{a}^\star; R)\right)$ für geeignetes $R > 0$ und $i = 1, \ldots, m$ angenommen wird.*

Ziel ist es nun das restringierte Problem (2.7) in ein freies Minimierungsproblem umzuwandeln. Auf dieses Problem kann man dann die auf Seite 5 genannten Verfahren zur Lösung von nichtrestringierten Optimierungsproblemen anwenden.

Zur Umwandlung der restringierten in eine freie Optimierung, muss zunächst die *Lagrange-Funktion* eingeführt werden: $\mathcal{L} : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}$

$$\mathcal{L}(\boldsymbol{a}, \boldsymbol{\lambda}) = f(\boldsymbol{a}) + \boldsymbol{\lambda}^T g(\boldsymbol{a})$$

Der Vektor $\boldsymbol{\lambda} \in \mathbb{R}^m$ wird *Lagrange-Multiplikator* genannt. Im Folgenden wird mit $J_{\mathcal{L}}$ die zu $\mathcal{L}$ gehörende Jacobi-Matrix bezüglich der partiellen Ableitungen nach $a_1, \ldots, a_p$ bezeichnet. Die Verbindung zwischen (2.7) und (2.1) wird dann ausgedrückt durch das folgende Resultat:

Satz 2.3.6. *Sei $\boldsymbol{a}^\star$ eine reguläre lokale Minimalstelle für (2.7) und es wird weiterhin angenommen, dass für ein geeignetes $R > 0$, $f, g_i \in C^1\left(B(\boldsymbol{a}^\star; R)\right)$, für $i = 1, \ldots, m$ gilt, dann existiert ein eindeutig bestimmter Vektor $\boldsymbol{\lambda}^\star \in \mathbb{R}^m$, so dass $J_{\mathcal{L}}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star) = 0$. Wird umgekehrt angenommen, dass $\boldsymbol{a}^\star \in \mathbb{R}^p$ der Beziehung $g(\boldsymbol{a}^\star) = 0$ genügt und dass $f, g_i \in C^2\left(B(\boldsymbol{a}^\star; R)\right)$ für ein geeignetes $R > 0$ und $i = 1, \ldots, m$ gilt. Sei $H_{\mathcal{L}}$ die Matrix der Einträge $\frac{\partial^2 \mathcal{L}}{\partial a_i \partial a_j}$ für $i, j = 1, \ldots, p$. Existiert ein Vektor $\boldsymbol{\lambda}^\star \in \mathbb{R}^n$ derart, dass $J_{\mathcal{L}}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star) = 0$ und*

$$z^T H_{\mathcal{L}}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star) z > 0 \quad \forall z \neq 0 \qquad \textit{mit} \qquad \nabla g(\boldsymbol{a}^\star)^T z = 0$$

gilt, dann ist $\boldsymbol{a}^\star$ eine strenge lokale Minimalstelle von (2.7).

Nachdem nun Gleichungsnebenbedingungen bei Optimierungsproblemen möglich sind, müssen in die Menge der möglichen Nebenbedingungen auch noch die Ungleichungen mit aufgenommen werden, d.h.: gegeben $f : \mathbb{R}^p \to \mathbb{R}$

$$\begin{aligned} &\min f(\boldsymbol{a}) \text{ unter den Bedingungen} \\ &g_i(\boldsymbol{a}) = 0 \quad \text{für} \quad i = 1, \ldots, m \\ &g_j(\boldsymbol{a}) \leq 0 \quad \text{für} \quad j = m+1, \ldots, m+k \end{aligned} \tag{2.8}$$

wobei $g_i : \mathbb{R}^p \to \mathbb{R}$ mit $m \leq p$ für $i = 1, \ldots, m$ und $g_j : \mathbb{R}^p \to \mathbb{R}$ für $j = m+1, \ldots, m+k$ gegebene Funktionen sind. Natürlich können bei Nebenbedingungen in Ungleichungsform nicht einfach die Ergebnisse der Nebenbedingungen in Gleichungsform angewandt werden. Es ist aber möglich durch kleine Veränderungen die Ergebnisse auf Ungleichungen zuzuschneiden. Man muss die Lösung von (2.8) auf die Minimierung einer geeigneten Lagrange-Funktion überführen. Die Definition 2.3.5 lautet somit:

Definition 2.3.7. *Angenommen, dass $g_i \in C^1\left(B(\boldsymbol{a}^\star; R)\right)$ für $i = 1, \ldots, m+k$ und einem geeignetem $R > 0$ gilt. Ein Punkt $\boldsymbol{a}^\star \in \mathbb{R}^p$, für den $g_i(\boldsymbol{a}^\star) = 0$ mit $i = 1, \ldots, m$ und $g_j(\boldsymbol{a}^\star) \leq 0$ mit $j = m+1, \ldots, m+k$ gilt, heißt dann regulär, wenn die Vektoren $\nabla g_i(\boldsymbol{a}^\star)$, $i = 1, \ldots, m$ gemeinsam mit den Vektoren $\nabla g_r(\boldsymbol{a}^\star)$, wobei $r \in \{j : j \in \{m+1, \ldots, m+k\}\ ,\ g_j(\boldsymbol{a}) = 0\}$ eine Menge linear unabhängiger Vektoren bilden.*

Des weiteren gilt der Satz 2.3.6 weiterhin, vorausgesetzt man ersetzt die Lagrange-Funktion $\mathcal{L}$ durch

$$\tilde{\mathcal{L}}(\boldsymbol{a}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{a}) + \boldsymbol{\lambda}^T \boldsymbol{h}_1(\boldsymbol{a}) + \boldsymbol{\mu}^T \boldsymbol{h}_2(\boldsymbol{a})$$

Unter $\boldsymbol{h}_1(\boldsymbol{a})$ ist ein Vektor bestehend aus den Nebenbedingungen $g_i(\boldsymbol{a}) = 0$ für $i = 1, \ldots, m$ zu verstehen und $\boldsymbol{h}_2(\boldsymbol{a})$ ist ein Vektor bestehend aus den übrigen Nebenbedingungen $g_j(\boldsymbol{a}) \leq 0$ für $j = m+1, \ldots, m+k$. Es müssen aber noch weitere Annahmen über die Nebenbedingungen getroffen werden.

Der Einfachheit halber formuliert man die notwendige Optimalitätsbedingung des Problems (2.8):

Satz 2.3.8. *Sei $\boldsymbol{a}^\star$ eine reguläre lokale Minimalstelle für (2.8) und angenommen, dass $f, g_i, \in C^1\left(B(a^\star, R)\right)$ für ein geeignetes $R > 0$ und $i = 1, \ldots, m+k$ gilt, dann gibt es nur zwei Vektoren $\boldsymbol{\lambda}^\star \in \mathbb{R}^m$ und $\boldsymbol{\mu}^\star \in \mathbb{R}^k$, so dass $J_{\tilde{\mathcal{L}}}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) = 0$ mit $\mu_j^\star \geq 0$ und $\mu_j^\star g_j(\boldsymbol{a}^\star) = 0$ $\forall j = m+1, \ldots, m+k$.*

2.4 Kuhn-Tucker-Bedingungen für die nichtlineare Optimierung

Jedes Iterationsverfahren benötigt eine Abbruchbedingung. Es werden also für die Suche nach der globalen Lösung eines restringierten Optimierungsproblems Bedingungen benötigt, mit denen überprüft werden kann, ob es sich bei dem aktuellen Punkt um eine globale Lösung handelt oder nicht. In diesem Abschnitt werden die *Kuhn-Tucker-Bedingungen* vorgestellt, die im Allgemeinen die Existenz einer lokalen Lösung für das nichtlineare Optimierungsproblem sichern. Unter geeigneten Annahmen sichern sie auch die Existenz einer globalen Lösung. Gegeben ist das allgemeine

nichtlineare Optimierungsproblem:

$$\begin{array}{c} \text{sei } f : \mathbb{R}^p \to \mathbb{R}, \text{ gesucht wird} \\ \min f(\boldsymbol{a}), \text{ unter den Nebenbedingungen} \\ g_i(\boldsymbol{a}) = b_i \quad i = 1, \ldots, m \\ g_i(\boldsymbol{a}) \leq b_i \quad i = m+1, \ldots, m+k \end{array} \tag{2.9}$$

Ein Vektor $\boldsymbol{a}$, der alle Nebenbedingungen aus (2.9) erfüllt, heißt *zulässige Lösung* von (2.9). Die Menge der zulässigen Lösungen heißt *zulässiger Bereich*. Im Folgenden gilt, dass $f, g_i \in C^1(\mathbb{R}^p)$ für $i = 1, \ldots, m+k$. Mit (2.9) ergibt sich folgende Lagrange-Funktion:

$$\mathcal{L}(\boldsymbol{a}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad = \quad f(\boldsymbol{a}) - \sum_{i=1}^{m} \lambda_i \left(b_i - g_i(\boldsymbol{a})\right) - \sum_{i=m+1}^{m+k} \mu_i \left(b_i - g_i(\boldsymbol{a})\right) \tag{2.10}$$

Für die Kuhn-Tucker-Bedingungen wird die folgende Definition benötigt:

Definition 2.4.1 (Tangentialraum). *Sei $J = \{i \ : \ i \in \{m+1, \ldots, m+k\} \ , \ g_i(\boldsymbol{a}) = b_i\}$ und $\tilde{\boldsymbol{g}} = (g_i, g_j)^t$ mit $i = 1, \ldots, m$ und $j \in J$, dann heißt $T(\boldsymbol{a}^\star)$ Tangentialraum, wenn gilt*

$$T(\boldsymbol{a}^\star) \quad = \quad \{\boldsymbol{q} \ : \ \boldsymbol{q} \in \mathbb{R}^p \ , \ \tilde{\boldsymbol{g}}_{\boldsymbol{a}}(\boldsymbol{a}^\star)\boldsymbol{q} = 0\} \tag{2.11}$$

Mit diesen Voraussetzungen gilt folgendes Resultat:

Satz 2.4.2 (Notwendige Kuhn-Tucker-Bedingungen [NB]). *Hat f ein restringiertes lokales Minimum im Punkt $\boldsymbol{a} = \boldsymbol{a}^\star$ und ist $\boldsymbol{a}^\star$ regulär, dann existieren notwendig die Vektoren $\boldsymbol{\lambda}^\star \in \mathbb{R}^m$ und $\boldsymbol{\mu}^\star \in \mathbb{R}^k$ mit:*

- *NB 1. Ordnung:*

  $$\nabla_{\boldsymbol{a}} \mathcal{L}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) = 0$$

  wobei $i = 1, \ldots, m, \quad j = m+1, \ldots, m+k$ und $\boldsymbol{\mu}^\star \geq 0$. Desweiteren gilt die Komplementaritätsbedingung

  $$\boldsymbol{\mu}^{\star t} \boldsymbol{h}(\boldsymbol{a}^\star) = 0$$

  wobei $\boldsymbol{h}$ den Vektor der Ungleichungsnebenbedingungen bildet.

- *NB 2. Ordnung:*

  $$\boldsymbol{q}^t \nabla^2_{\boldsymbol{a}} \mathcal{L}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star)\boldsymbol{q} \geq 0 \quad \forall \boldsymbol{q} \in T(\boldsymbol{a}^\star)$$

  was bedeutet, dass die Hesse-Matrix auf dem Tangentialraum positiv definit ist.

Für die hinreichenden Kuhn-Tucker-Bedingungen wird die Menge

$$T^+(\boldsymbol{a}^\star) = \left\{\boldsymbol{q} \ : \ \boldsymbol{q} \in \mathbb{R}^p \ , \ \nabla_{\boldsymbol{a}} g_i(\boldsymbol{a}^\star)\boldsymbol{q} = 0, \ i = 1, \ldots, m \ , \ \nabla_{\boldsymbol{a}} g_j(\boldsymbol{a}^\star)\boldsymbol{q} = 0, \forall j \in J \text{ mit } \boldsymbol{\mu}_j^\star > 0\right\}$$

definiert.

Satz 2.4.3 (Hinreichende Kuhn-Tucker-Bedingungen [HB]). *Sei $\boldsymbol{a}^\star$ ein restringiertes lokales Minimum der Funktion f, für das die NB 1. Ordnung gelte. Wenn weiterhin*

$$\boldsymbol{q}^t \nabla^2_{\boldsymbol{a}} \mathcal{L}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star)\boldsymbol{q} > 0 \quad \forall \boldsymbol{q} \in T^+(\boldsymbol{a}^\star) \text{ mit } \boldsymbol{q} \neq \boldsymbol{0}$$

erfüllt ist, liegt ein striktes lokales Minimum vor.

Alle Kuhn-Tucker-Bedingungen gelten nur unter der Voraussetzung, dass die Vektoren $\boldsymbol{\lambda}^\star$ und $\boldsymbol{\mu}^\star$ auch tatsächlich existieren. Um zu gewährleisten, dass sie existieren, werden im folgenden Satz geometrische Bedingungen an den zulässigen Bereich gestellt:

Satz 2.4.4. *Wenn*

- *die Funktion f in (2.9) auf dem zulässigen Bereich konvex ist,*
- *der Punkt $(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star)$ allen notwendigen Kuhn-Tucker-Bedingungen genügt,*
- *die Funktionen g_i mit $i = 1, \ldots, m$ konvex sind, für die $\lambda_i^\star > 0$ gilt und*
- *die Funktionen g_j mit $j = m+1, \ldots, m+k$ konvex sind, für die $\mu_j^\star > 0$ gilt,*

dann ist $f(\boldsymbol{a}^\star)$ die restringierte globale Maximalstelle von f für das Problem (2.9).

2.5 Die Penalty-Methode

Die Überprüfung, ob eine Lösung global ist oder nicht, wurde im letzten Kapitel vorgestellt. Nun muss eine Iteration gefunden werden, mit der man eine Lösung finden kann. Da kann die Penalty-Methode genutzt werden.

Die grundlegende Idee dieser Strafmethode besteht darin, das restringierte Problem in ein nichtrestringiertes Problem zu überführen. Hierzu sind die Nebenbedingungen teilweise oder vollständig zu eliminieren. Das neue Problem ist durch das Vorhandensein eines Parameters α charakterisiert, der ein Maß für die Genauigkeit liefert, mit der die Nebenbedingungen tatsächlich erfüllt sind.

Im Folgenden wird das restringierte Problem (2.7) betrachtet. Voraussetzung für die Penalty-Methode ist, dass sich mindestens eine optimale Lösung $\boldsymbol{a}^\star$ im zulässigen Bereich $\Omega \subset \mathbb{R}^p$ befindet. Man kann also die Suche nach diesem Optimum auf Ω beschränken. Das Problem hat somit folgende Form:

$$\min \mathcal{L}_\alpha(\boldsymbol{a}) = f(\boldsymbol{a}) + \frac{1}{2}\alpha\|\boldsymbol{h}(\boldsymbol{a})\|_2^2 \quad \text{für} \quad \boldsymbol{a} \in \Omega \tag{2.12}$$

wobei $\boldsymbol{h}(\boldsymbol{a})$ der Vektor der Gleichungsnebenbedingungen $g_i(\boldsymbol{a}) = 0$ für $i = 1, \ldots, m$ ist. Die Funktion $\mathcal{L}_\alpha : \mathbb{R}^p \to \mathbb{R}$ heißt die *Penalty-Lagrange-Funktion*, und α ist der *Penalty-Parameter*. Wenn die Nebenbedingungen exakt erfüllt sind, man sich also im zulässigen Bereich befindet, ist die Minimierung von f zur Minimierung von $\mathcal{L}_\alpha$ äquivalent. Die Penalty-Methode bildet ein iteratives Verfahren zur Lösung von (2.12). Für $i = 0, 1, \ldots$ bis zur Konvergenz ist die Folge von Problemen

$$\min \mathcal{L}_{\alpha_i}(\boldsymbol{a}) \qquad \text{mit } \boldsymbol{a} \in \Omega \tag{2.13}$$

zu lösen. Hierbei ist $\{\alpha_i\}$ eine monoton wachsende Folge positiver Penalty-Parameter, so dass für $i \to \infty$ die Folge $\alpha_i \to \infty$ konvergiert. In jedem Schritt des Penalty-Verfahrens muss, nachdem α_i gewählt wurde, ein Minimierungsproblem in Bezug auf die Variable $\boldsymbol{a}$ gelöst werden. Dies führt zu einer Folge von Werten $\boldsymbol{a}_i^\star$, den Lösungen von (2.13). Wenn so vorgegangen wird, strebt die Zielfunktion $\mathcal{L}_{\alpha_i}(\boldsymbol{a}_i^\star)$ gegen unendlich, es sei den $\boldsymbol{h}(\boldsymbol{a}^\star)$ geht gegen 0.

Die Minimierungsaufgaben können wiederum mit den auf Seite 5 genannten Verfahren gelöst werden. Damit eine Konvergenz der Penalty-Methode gesichert ist, benötigt man folgenden Satz:

Satz 2.5.1. *Es werde vorausgesetzt, dass $f : \mathbb{R}^p \to \mathbb{R}$ und $h : \mathbb{R}^p \to \mathbb{R}^m$ mit $m \leq p$ stetige Funktionen auf einer abgeschlossenen Menge $\Omega \in \mathbb{R}^p$ sind und dass die Folge von Penalty-Parametern $\alpha_i > 0$ monoton divergiert. Wenn dann $\boldsymbol{a}_i^\star$ eine globale Minimalstelle des Problems (2.13) im Schritt i ist, konvergiert für $i \to \infty$ die Folge $\boldsymbol{a}_i^\star$ gegen den Punkt $\boldsymbol{a}^\star$, der eine globale Minimalstelle von f in Ω ist und der Nebenbedingung $\boldsymbol{h}(\boldsymbol{a}^\star) = 0$ genügt.*

Hinsichtlich der Auswahl des Parameters α_i kann gezeigt werden, dass für große Werte von α_i das Minimierungsproblem (2.13) schlecht konditioniert ist. Eine Lösung des Problems ist sehr schwierig, es sei denn, der Anfangswert liegt besonders nahe bei $\boldsymbol{a}^\star$. Andererseits darf die Folge α_i nicht zu langsam wachsen, da dies die Gesamtkonvergenz der Methode negativ beeinflussen würde.

Üblicherweise wird in der Praxis ein nicht zu großer Wert α_0 ausgewählt und dann $\alpha_i = \beta\alpha_{i-1}$ für $i > 0$ gesetzt, wobei β eine ganze Zahl zwischen 4 und 10 ist. Der Startwert, der für die zur Lösung des Minimierungsproblems (2.13) verwendeten numerischen Methode benötigt wird, kann gleich der letzten berechneten Iterierten α_i gesetzt werden.

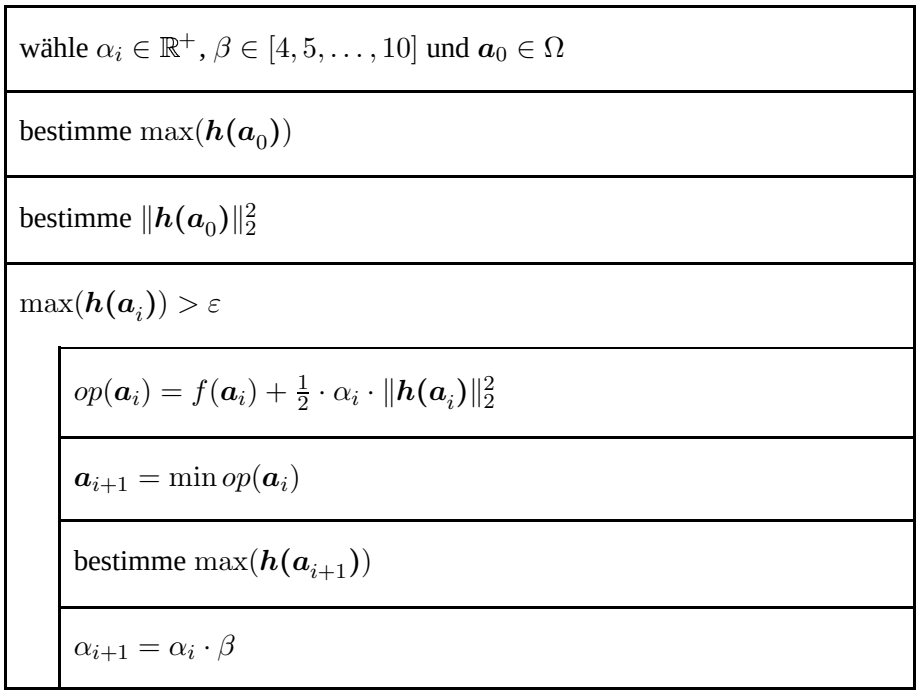

Abbildung 2.1: Nassi-Shneidermann-Diagramm zur Penalty-Methode

2.6 Die Methode der Lagrangeschen Multiplikatoren

Die Penalty-Methode hat den Nachteil, dass der Penalty-Parameter gegen ∞ laufen muss, damit das Iterationsverfahren konvergiert. Es gibt verschiedene Varianten von Strafmethoden, die versuchen, diesen Nachteil zu umgehen. Eine verwendet (anstelle von $\mathcal{L}_\alpha(\boldsymbol{a})$ in (2.12)) die *modifizierte Lagrange-Funktion* $\mathcal{G}_\alpha : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}$, die durch

$$\mathcal{G}_\alpha(\boldsymbol{a}, \boldsymbol{\lambda}) \quad = \quad f(\boldsymbol{a}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{a}) + \frac{1}{2}\alpha \|\boldsymbol{h}(\boldsymbol{a})\|_2^2 \tag{2.14}$$

gegeben ist. $\boldsymbol{\lambda} \in \mathbb{R}^m$ ist ein Lagrange-Multiplikator und die Lösung $\boldsymbol{a}^\star$ des Problems (2.7) ist auch Lösung von (2.14). Die hier vorgestellte Methode hat aber gegenüber (2.12) den Vorteil, dass ein weiterer Freiheitsgrad $\boldsymbol{\lambda}$ mit in die Berechnung aufgenommen wird. Diese Strafmethode angewendet auf (2.14) lautet:
Für $i = 0, 1, \ldots$ löse die Folge von Problemen

$$\min \mathcal{G}_{\alpha_i}(\boldsymbol{a}, \boldsymbol{\lambda}_i) \qquad \text{für } \boldsymbol{a} \in \Omega \tag{2.15}$$

bis zur Konvergenz, wobei $\{\boldsymbol{\lambda}_i\}$ eine beschränkte Folge von unbekannten Vektoren im $\mathbb{R}^m$ ist, und die Parameter α_i wie in Kapitel 2.5 definiert sind.

Satz (2.5.1) gilt auch für die Methode (2.15), vorausgesetzt die Multiplikatoren werden als beschränkt angenommen. Man beachte, dass die Existenz einer Minimalstelle von (2.15) nicht garantiert ist, selbst in dem Fall nicht, in dem f eine eindeutige globale Minimalstelle hat (siehe Beispiel 2.6.1). Dieser Umstand kann durch Addition weiterer nichtquadratischer Terme zur modifizierten Lagrange-Funktion (z.B. der Form $\|\boldsymbol{h}\|_2^q$ mit einem möglichst großen q) überwunden werden.

Beispiel 2.6.1. *Es ist die Minimalstelle von $f(x) = -x^4$ unter der Nebenbedingung $x = 0$ zu bestimmen. Dieses Problem besitzt die eindeutige Lösung $x^\star = 0$. Betrachtet man aber die modifizierte Lagrange-Funktion*

$$\mathcal{L}_{\alpha_k} = -x^4 + \boldsymbol{\lambda}_k x + \frac{1}{2}\alpha_k x^2$$

so stellt man fest, dass sie für jedes $\alpha_k \neq 0$ nicht länger ein Minimum in $x = 0$ besitzt, obwohl sie dort verschwindet.

Was die Wahl der Multiplikatoren betrifft, wird die Folge von Vektoren $\boldsymbol{\lambda}_i$ typischerweise durch die Formel

$$\boldsymbol{\lambda}_{i+1} = \boldsymbol{\lambda}_i + \alpha_i \boldsymbol{h}(\boldsymbol{a}^{(i)})$$

bestimmt, wobei $\boldsymbol{\lambda}_0$ ein gegebener Wert ist. Die Folge α_i wird *a priori* gesetzt oder während der Laufzeit modifiziert. Bezüglich der Konvergenzeigenschaften der Methode der Lagrange-Multiplikatoren gilt das folgende Resultat.

Satz 2.6.2. *Angenommen, dass $\boldsymbol{a}^\star$ eine reguläre, lokale strenge Minimalstelle von (2.7) ist und es gelte*

- *$f, h_j \in C^2(B(\boldsymbol{a}^\star; R))$ mit $j = 1, \ldots, m$ und für ein geeignetes $R > 0$;*
- *das Paar $(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star)$ genügt $z^T H_{\mathcal{G}_0}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star) z > 0$, $\forall z \neq 0$, so dass $J_h(\boldsymbol{a}^\star)^T z = 0$;*
- *$\exists\, \bar{\alpha} > 0$, so dass $H_{\mathcal{G}_{\bar{\alpha}}}(\boldsymbol{a}^\star, \boldsymbol{\lambda}^\star) > 0$.*

Dann gibt es drei positive Skalare δ, γ und $\mathcal{S}$ derart, dass für jedes Paar $(\boldsymbol{\lambda}, \alpha) \in V$ mit $V = \{(\boldsymbol{\lambda}, \alpha) \in \mathbb{R}^{m+1} : \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^\star\|_2 < \delta\alpha, \alpha \geq \bar{\alpha}\}$ das Problem

$$\min \mathcal{G}_\alpha(\boldsymbol{a}, \boldsymbol{\lambda}), \quad \text{mit } \boldsymbol{a} \in B(\boldsymbol{a}^\star; \gamma)$$

eine eindeutig bestimmte Lösung $\boldsymbol{a}(\boldsymbol{\lambda}, \alpha)$ besitzt, die differenzierbar in Bezug auf ihre Argumente ist. Darüber hinaus gilt $\forall (\boldsymbol{\lambda}, \alpha) \in V$

$$\|\boldsymbol{a}(\boldsymbol{\lambda}, \alpha) - \boldsymbol{a}^\star\|_2 \leq \mathcal{S} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^\star\|_2$$

Unter zusätzlichen Annahmen kann bewiesen werden, dass die Methode der Lagrange-Multiplikatoren konvergiert. Die Konvergenz der Methode ist superlinear, wenn $\alpha_i \to \infty$ für $i \to \infty$ gilt, denn

$$\lim_{i \to \infty} \frac{\|\boldsymbol{\lambda}_{i+1} - \boldsymbol{\lambda}^\star\|_2}{\|\boldsymbol{\lambda}_i - \boldsymbol{\lambda}^\star\|_2} = 0$$

Sollte α_i eine obere Schranke besitzen, so konvergiert die Methode linear.

Ein Vorteil der in diesem Kapitel vorgestellten Methode gegenüber der Strafmethode aus dem Kapitel 2.5 ist, dass die Folge α_i nicht mehr gegen Unendlich laufen muss, wodurch die Schlechtkonditioniertheit des Problems (2.15) begrenzt wird. Ein zweiter Vorteil dieser Methode ist, dass die Konvergenz der Methode unabhängig vom Wachstumsgrad des Strafparameters ist, was zu einer beträchtlichen Reduktion der numerischen Kosten führt.

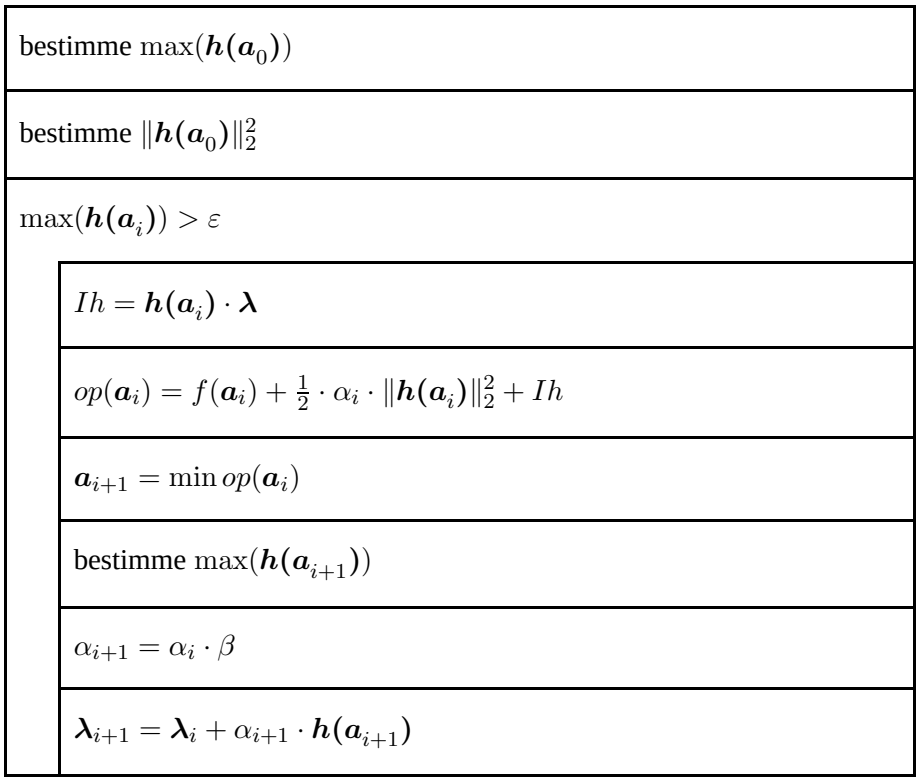

Abbildung 2.2: Nassi-Shneidermann-Diagramm zur Methode der Lagrange-Multiplikatoren

2.7 Methode zur Lösung von Optimierungsaufgaben mit Ungleichungsnebenbedingungen

Die beiden bisher beschriebenen Verfahren zur Lösung restringierter Optimierungsprobleme können nur bei Optimierungsaufgaben mit Gleichungsnebenbedingungen eingesetzt werden. Restriktionen können aber auch Ungleichungen enthalten. In diesem Fall kann das im Folgenden beschriebene Iterationsverfahren genutzt werden. Die hier vorgestellte Art von Verfahren zur Lösung des Minimierungsproblems

$$\begin{aligned} &\min_{\boldsymbol{a} \in \mathbb{R}^p} \quad f(\boldsymbol{a}) = \tfrac{1}{2} \sum_{i=1}^{n} (M(\boldsymbol{a}, \boldsymbol{x}_i) - y_i)^2 \\ &\text{mit} \quad \boldsymbol{l} := \begin{pmatrix} \boldsymbol{l}_B \\ \boldsymbol{l}_L \\ \boldsymbol{l}_N \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{a} \\ A_L \boldsymbol{a} \\ c(\boldsymbol{a}) \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{u}_B \\ \boldsymbol{u}_L \\ \boldsymbol{u}_N \end{pmatrix} =: \boldsymbol{u} \end{aligned} \tag{2.16}$$

wird auch sequentielles quadratisches Programm (SQP) genannt. Hauptbestandteil solcher SQP-Methoden sind Haupt- und Unteriterationen.

Vor Beginn der Hauptiteration wird zunächst, ausgehend von den durch den Benutzer gegebenen Startwerten, ein Punkt gesucht, der den Grenzen $\boldsymbol{l}_B \leq \boldsymbol{a} \leq \boldsymbol{u}_B$ und linearen Nebenbedingungen $\boldsymbol{l}_L \leq A_L \boldsymbol{a} \leq \boldsymbol{u}_L$ genügt. Sollte bei dieser Suche keine mögliche Lösung gefunden werden, bricht das Verfahren ab, da in diesem Fall das Minimierungsproblem (2.16) keine Lösung besitzt. Nach einer erfolgreichen Suche bleiben die Begrenzungen und linearen Nebenbedingungen in den nachfolgenden Iterationen immer erfüllt.

Die nun startende Hauptiteration generiert eine Sequenz von Iterationswerten, die gegen einen Punkt konvergiert, der den Kuhn-Tucker-Bedingungen genügt. In jedem Iterationsschritt verbes-

sert man den möglichen Lösungspunkt über

$$\bar{\boldsymbol{a}} \quad = \quad \boldsymbol{a} + \mu \boldsymbol{q} \tag{2.17}$$

wobei μ die nicht negative Schrittweite ist. Mit $\boldsymbol{q}$ wird die Suchrichtung bezeichnet, die über die Lösung eines Unterproblems bestimmt wird.

Aus Vereinfachungsgründen wird zunächst die Schrittweite auf $\mu = 1$ gesetzt. Die Funktion f soll in der nächsten Iteration kleiner sein als in dieser, was bedeutet, dass $f(\boldsymbol{a} + \boldsymbol{q}) < f(\boldsymbol{a})$ gelten soll. Aus diesem Grund wird die Funktion $f(\boldsymbol{a} + \boldsymbol{q})$ in einer Taylor-Reihe um den Punkt $\boldsymbol{a}$ entwickelt, und die erhaltene Summe über $\boldsymbol{q}$ minimiert (2.19).

$$\min_{\boldsymbol{q} \in \mathbb{R}^p} \quad f(\boldsymbol{a}) + (\nabla f)^T(\boldsymbol{a})\boldsymbol{q} + \boldsymbol{q}^T(Hf)(\boldsymbol{a})\boldsymbol{q} \tag{2.18}$$

$$\Leftrightarrow \quad \min_{\boldsymbol{q} \in \mathbb{R}^p} \quad \left\{ \begin{array}{ll} & \frac{1}{2} \cdot \sum_{i=1}^{n} \left(M\left(\boldsymbol{a}, \boldsymbol{x_i}\right) - y_i\right)^2 \\ + & \left(\sum_{i=1}^{n} \left(M\left(\boldsymbol{a}, \boldsymbol{x_i}\right) - y_i\right) \cdot \frac{\partial M(\boldsymbol{a}, \boldsymbol{x_i})}{\partial \boldsymbol{a}} \right)^T \cdot \boldsymbol{q} \\ + & \frac{1}{2} \cdot \boldsymbol{q}^T \cdot \left(\sum_{i=1}^{n} \left(M\left(\boldsymbol{a}, \boldsymbol{x_i}\right) - y_i\right) \cdot \frac{\partial^2 M(\boldsymbol{a}, \boldsymbol{x_i})}{\partial \boldsymbol{a}^2} + \left(\frac{\partial M(\boldsymbol{a}, \boldsymbol{x_i})}{\partial \boldsymbol{a}} \right)^2 \right) \cdot \boldsymbol{q} \end{array} \right. \tag{2.19}$$

$$\Leftrightarrow \quad \min_{\boldsymbol{q} \in \mathbb{R}^p} \quad f(\boldsymbol{a}) + g(\boldsymbol{a})^T \cdot \boldsymbol{q} + \frac{1}{2}\boldsymbol{q}^T H \boldsymbol{q} \tag{2.20}$$

$$\Leftrightarrow \quad \min_{\boldsymbol{q} \in \mathbb{R}^p} \quad g(\boldsymbol{a})^T \cdot \boldsymbol{q} + \frac{1}{2}\boldsymbol{q}^T H \boldsymbol{q} \tag{2.21}$$

Nachdem in (2.20) die Minimierungsaufgabe (2.19) zusammengefasst wurde, wird deutlich, dass $f(\boldsymbol{a})$ keine Auswirkung auf das Minimierungsproblem hat und wegfallen kann. Das nun in (2.21) entstandene Unterproblem ist ein Quadratisches Programm (QP) und kann gelöst werden. Wenn die Matrix H positiv definit ist, besitzt dieses Problem ein eindeutiges globales Minimum (siehe Abschnitt 2.8.2). Die Lösung dieses QP's basiert ebenfalls auf einem iterativen Verfahren, das als Unteriteration bezeichnet wird. Zur gewünschten Lösung von (2.21) müssen die Nebenbedingungen von (2.16) angepasst werden. Die untere Grenze $\boldsymbol{l}$ in (2.16) kann in 3 Bereiche aufgeteilt werden. In den Bereich für die direkten Grenzen der Parameter ($\boldsymbol{l}_B$), in den für die linearen Nebenbedingungen ($\boldsymbol{l}_L$) und den für die nichtlinearen ($\boldsymbol{l}_N$). Der Vektor $\bar{\boldsymbol{l}}$ im Unterproblem (2.21) besitzt dann folgende Definition:

$$\begin{aligned} \bar{\boldsymbol{l}}_B &= \boldsymbol{l}_B - \boldsymbol{a} \\ \bar{\boldsymbol{l}}_L &= \boldsymbol{l}_L - A_L \boldsymbol{a} \\ \bar{\boldsymbol{l}}_N &= \boldsymbol{l}_N - c \end{aligned}$$

$\boldsymbol{c}$ ist hierbei der Vektor der nichtlinearen Nebenbedingungen ausgewertet im aktuellen Folgenelement $\boldsymbol{a}$. Der Vektor $\bar{\boldsymbol{u}}$ ist auf gleiche Weise aus $\boldsymbol{u}$ zu erzeugen.

Nachdem das quadratische Unterproblem wie in Abschnitt 2.8.2 beschrieben gelöst wurde, ist die Unteriteration abgeschlossen. Die Hauptiteration muss nun passend zur Suchrichtung $\boldsymbol{p}$ die Schrittweite μ bestimmen. Dies geschieht über die in Abschnitt 2.5 beschriebene Penalty-Methode, die die *Qualität* jeder Iteration bestimmt.

Das Iterationsverfahren zur Lösung restringierter, nichtlinearer Optimierungsaufgaben läuft mit den bis hier beschriebenen Schritten. Eine mögliche Effizienzsteigerung wird dadurch erzielt, dass die Matrix H nicht in jedem Iterationsschritt der Hauptiteration neu bestimmt wird, sondern über die Updatefunktion

$$\bar{H} \quad = \quad H - \frac{1}{\boldsymbol{s}^T H \boldsymbol{s}} H \boldsymbol{s}\boldsymbol{s}^T H + \frac{1}{\boldsymbol{y}^T \boldsymbol{s}} \boldsymbol{y}\boldsymbol{y}^T \tag{2.22}$$

aktualisiert wird.

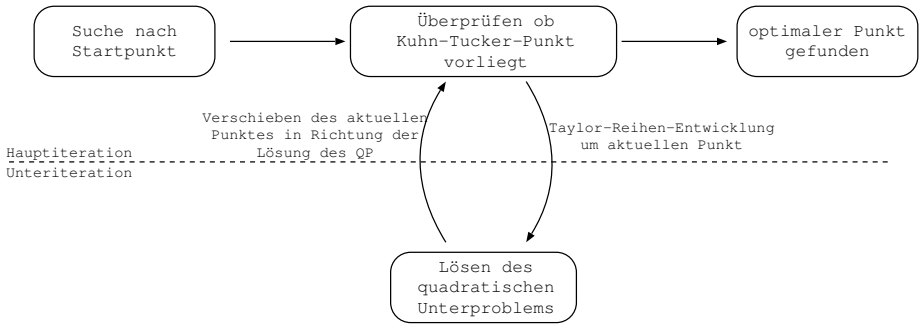

Abbildung 2.3: Übersicht des Iterationsverfahrens

2.8 Lösen quadratischer Optimierungsprobleme

Durch das im letzten Kapitel vorgestellte Iterationsverfahren wurde das allgemeine Optimierungsproblem auf ein quadratisches reduziert. Desweiteren wurden die nichtlinearen Nebenbedingungen in lineare umgewandelt. Verfahren zur Lösung solcher Unterprobleme werden in diesem Kapitel vorgestellt.

Neben bekannten linearen Programmen gibt es noch weitere Optimierungsprobleme, die über eine endliche Zahl von Iterationsschritten gelöst werden können. Diese Art von Programmen werden *quadratische Programme* genannt. Hierfür muss die zu optimierende Funktion $q(\boldsymbol{x})$ quadratisch und die Nebenbedingungen linear sein. Die Probleme, für die eine Lösung $\boldsymbol{x}^\star$ gesucht wird, hat somit die Struktur

$$\begin{array}{lrcll} \min\limits_{\boldsymbol{x}} & q(\boldsymbol{x}) & = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{g}^T\boldsymbol{x} & \\ \text{mit} & \boldsymbol{a}_i^T\boldsymbol{x} & = & \boldsymbol{b}_i, & i \in D \\ & \boldsymbol{a}_i^T\boldsymbol{x} & \geq & \boldsymbol{b}_j, & j \in E \end{array} \tag{2.23}$$

wobei D die Indexmenge der Gleichungsnebenbedingungen und E die Indexmenge der Ungleichungsnebenbedingungen ist. Für die folgende Betrachtung muss gelten, dass die Matrix G symmetrisch ist. Genauso wie in linearen Programmen könnte es sein, dass das Problem nicht lösbar oder die Lösung unbegrenzt ist. Diese Möglichkeiten sind aber bereits vor der quadratischen Programmierung abgefangen worden, so dass man davon ausgehen kann, dass immer eine Lösung $\boldsymbol{x}^\star$ existiert. In den Fällen, in denen die Hesse-Matrix G positiv semidefinit ist, existiert immer eine globale Lösung $\boldsymbol{x}^\star$. Sollte die Matrix sogar positiv definit sein, ist $\boldsymbol{x}^\star$ eindeutig. Diese beiden Ergebnisse folgen aus der Konvexität bzw. strikten Konvexität (siehe Definition 2.3.2) von $q(\boldsymbol{x})$.

2.8.1 Lösen quadratischer Optimierungsprobleme mit Gleichungsnebenbedingungen

Dieses Kapitel befasst sich mit der Bestimmung der Lösung $\boldsymbol{x}^\star$ des Optimierungsproblems mit Gleichungsnebenbedingungen. Die Lösung dieser Aufgabe könnte auch über die in Kapitel 2.5 und 2.6 vorgestellten Strafverfahren berechnet werden. Für quadratische Optimierungsprobleme ist die folgende Methode jedoch effizienter.

Angenommen, die Matrix $G \in \mathbb{R}^{p\times p}$ und die beiden Vektoren $\boldsymbol{x}$ und $\boldsymbol{g}$ aus $\mathbb{R}^p$ sind gegeben.

$$\begin{array}{lrcl} \min\limits_{\boldsymbol{x}} & q(\boldsymbol{x}) & = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{g}^T\boldsymbol{x} \\ \text{mit} & A^T\boldsymbol{x} & = & \boldsymbol{b} \end{array} \tag{2.24}$$

Es werde vorausgesetzt, dass es $m \leq p$ Nebenbedingungen gibt, so dass $\boldsymbol{b} \in \mathbb{R}^m$ ist und dass die Spalten der Matrix $A \in \mathbb{R}^{p\times m}$ die Vektoren $\boldsymbol{a}_i$, $i \in D$ aus (2.23) sind. Desweiteren sollte A vollen Rang m besitzen. Hat die Matrix A keinen vollen Rang, so gibt es linear abhängige Nebenbedingungen. In diesem Fall kann man diese Nebenbedingungen fallen lassen. Diese Voraussetzungen

garantieren die Eindeutigkeit der Lagrange-Multiplikatoren $\boldsymbol{\lambda}^\star$ und deren Anzahl, was beispielsweise für die *Methode des aktiven Datensatzes* in Kapitel 2.8.2 auf Seite 16 von Interesse ist.

Eine Methode zur Lösung von (2.24) ist, dass man die Nebenbedingungen nutzt, um Variablen zu elemenieren. Es werde angenommen, dass die Aufteilungen

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix}, \qquad A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}, \qquad \boldsymbol{g} = \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \end{pmatrix}, \qquad G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$$

gelten, und dass $\boldsymbol{x}_1 \in \mathbb{R}^m$, $\boldsymbol{x}_2 \in \mathbb{R}^{p-m}$, $A_1 \in \mathbb{R}^{m\times m}$, $A_2 \in \mathbb{R}^{m\times(p-m)}$ sowie dass A_1 regulär ist. Dann können die Gleichungen in (2.24) als $A_1^T\boldsymbol{x}_1 + A_2^T\boldsymbol{x}_2 = \boldsymbol{b}$ geschrieben werden. Dieses lineare Gleichungssystem kann einfach gelöst werden, beispielsweise über die *Gauß-Elimination*. $\boldsymbol{x}_1$ kann somit über $\boldsymbol{x}_2$ ausgedrückt werden:

$$\boldsymbol{x}_1 \;=\; A_1^{-T}\left(\boldsymbol{b} - A_2^T\boldsymbol{x}_2\right) \tag{2.25}$$

Eingesetzt in $q(\boldsymbol{x})$ ergibt sich somit das Problem: $\min \psi(x_2)$, $\boldsymbol{x}_2 \in \mathbb{R}^{p-m}$ wobei $\psi(\boldsymbol{x}_2)$ die quadratische Funktion

$$\begin{aligned} \psi(\boldsymbol{x}_2) \;=\;& \tfrac{1}{2}\boldsymbol{x}_2^T\left(G_{22} - G_{21}A_1^{-T}A_2^T - A_2A_1^{-1}G_{12} + A_2A_1^{-1}G_{11}A_1^{-T}A_2^T\right)\boldsymbol{x}_2 \\ +\;& \boldsymbol{x}_2^T\left(G_{21} - A_2A_1^{-1}G_{11}\right)A_1^{-T}\boldsymbol{b} + \tfrac{1}{2}\boldsymbol{b}^TA_1^{-1}G_{11}A_1^{-T}\boldsymbol{b} \\ +\;& \boldsymbol{x}_2^T\left(\boldsymbol{g}_2 - A_2A_1^{-1}\boldsymbol{g}_1\right) + \boldsymbol{g}_1^TA_1^{-T}\boldsymbol{b} \end{aligned} \tag{2.26}$$

ist.

Es existiert ein eindeutiger Minimierer $\boldsymbol{x}_2^\star$, wenn die Hesse-Matrix $\nabla^2\psi$ positiv definit ist, wobei $\boldsymbol{x}_2^\star$ durch Lösen des linearen Gleichungssystems $\nabla\psi(\boldsymbol{x}_2) = 0$ erhalten wird. Anschließend kann $\boldsymbol{x}_1^\star$ durch Einsetzen von $\boldsymbol{x}_2^\star$ in die Gleichung (2.25) bestimmt werden. Der Lagrange-Multiplikator-Vektor $\boldsymbol{\lambda}^\star$ wird definiert über $\boldsymbol{g}^\star = A\boldsymbol{\lambda}^\star$ wobei $\boldsymbol{g}^\star = \nabla q(\boldsymbol{x}^\star)$ ist. Sie können durch Lösen des ersten Teils dieser Gleichung $g_1^\star = A_1\boldsymbol{\lambda}^\star$ bestimmt werden. Über die Definition von $q(\boldsymbol{x})$ in (2.24), in der $\boldsymbol{g}^\star = \boldsymbol{g} + G\boldsymbol{x}^\star$ gilt, kann eine explizite Definition für $\boldsymbol{\lambda}^\star$ bestimmt werden:

$$\boldsymbol{\lambda}^\star \;=\; A_1^{-1}(\boldsymbol{g}_1 + G_{11}\boldsymbol{x}_1^\star + G_{12}\boldsymbol{x}_2^\star) \tag{2.27}$$

Beispiel 2.8.1. *Es soll das Problem*

$$\begin{aligned} \min_x \quad & q(x) = x_1^2 + x_2^2 + x_3^2 \\ \textit{mit} \quad & x_1 + 2x_2 - x_3 = 4 \\ \textit{und} \quad & x_1 - x_2 + x_3 = -2 \end{aligned} \tag{2.28}$$

gelöst werden. Um x_3 aus dem Problem zu entfernen werden die Nebenbedingungen umgeschrieben.

$$\begin{aligned} x_1 + 2x_2 \;&=\; 4 + x_3 \\ x_1 - x_2 \;&=\; -1 - x_3 \end{aligned}$$

Diese Gleichungen können beispielsweise mit dem Gauß-Verfahren gelöst werden.

$$\begin{aligned} x_1 \;&=\; -\tfrac{1}{3}x_3 \\ x_2 \;&=\; 2 + \tfrac{2}{3}x_3 \end{aligned} \tag{2.29}$$

Die Variablen aus (2.25) lauten somit:

$$\tilde{\boldsymbol{x}}_1 = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \qquad \tilde{\boldsymbol{x}}_2 = \begin{pmatrix} x_3 \end{pmatrix}, \qquad A_1 = \begin{pmatrix} 1 & 1 \\ 2 & -1 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} -1 & 1 \end{pmatrix}$$

Setzt man nun (2.29) in (2.28) ein, ergibt sich

$$\psi(x_3) \;=\; \frac{14}{9}x_3^2 + \frac{8}{3}x_3 + 4 \tag{2.30}$$

Entsprechend zu (2.26) erhält man also

$$G_{11} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \qquad G_{12} = G_{21}^T = 0, \qquad G_{22} = \begin{bmatrix} 2 \end{bmatrix}, \qquad g = 0$$

Die Hesse-Matrix in (2.30) ist $\left[\frac{28}{9}\right]$ *und ist somit positiv definit. Dies bedeutet, dass nun* $\nabla\psi = 0$ *gesetzt werden muss. Es ergibt sich* $x_3^\star = -\frac{6}{7}$*. Durch die Rücksubstitution in (2.28) erhält man* $x_1^\star = \frac{2}{7}$ *und* $x_2^\star = \frac{10}{7}$*. Für das System* $g^\star = A\lambda^\star$ *ergibt sich*

$$\frac{2}{7}\begin{pmatrix} 2 \\ 10 \\ -6 \end{pmatrix} \;=\; \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ -1 & 1 \end{bmatrix}\begin{pmatrix} \lambda_1^\star \\ \lambda_2^\star \end{pmatrix}$$

Löst man dieses Gleichungssystem, ergibt sich eine eindeutige Lösung für $\lambda_1^\star = \frac{8}{7}$ *und* $\lambda_2^\star = -\frac{4}{7}$*.*

2.8.2 Methode des aktiven Datensatzes

Die meisten quadratischen Optimierungsprobleme beinhalten Nebenbedingungen in Ungleichungsform wie in (2.23). Dieser Abschnitt beschreibt, wie man die Methode zur Lösung von Optimierungsaufgaben mit Gleichungsnebenbedingungen über die *Methode des aktiven Datensatzes* generalisieren kann, so dass man hiermit Optimierungsaufgaben mit Ungleichungsnebenbedingungen lösen kann.

Satz 2.8.2. *Wenn die Matrix G in (2.23) positiv definit ist, besitzt das quadratische Optimierungsproblem ein eindeutiges globales Minimum.*

In der Methode des aktiven Datensatzes werden bestimmte Nebenbedingungen, die im *aktiven Datensatz* $\mathcal{A}$ liegen, als Gleichungsnebenbedingungen angesehen, wobei die übrigen Nebenbedingungen kurzzeitig nicht betrachtet werden. Die Methode gleicht nun die Menge der aktiven Datensätze so lange ab, bis sie die korrekte Menge an aktiven Nebenbedingungen gefunden hat, die das Optimierungsproblem (2.23) löst. Da die Optimierungsfunktion quadratisch ist, wird es m ($0 \leq m \leq p$) aktive Nebenbedingungen geben. Dies steht im Gegensatz zur linearen Programmierung, wo $m = p$ ist.

Satz 2.8.3. *In jedem Iterationsschritt von (2.23) gibt es eine mögliche Lösung* $\boldsymbol{x}^{(j)}$*, die das Gleichungssystem*

$$\boldsymbol{a}_i^T\boldsymbol{x}^{(j)} \;=\; \boldsymbol{b}_i \qquad i \in \mathcal{A}$$

löst. $\mathcal{A}$ *bildet die Indexmenge der aktiven Nebenbedingungen. Diese aktiven Nebenbedingungen werden als Gleichungen angenommen.*

Jede Iteration versucht die Lösung eines Optimierungsproblems mit Gleichungsnebenbedingungen (OG), die nur aus aktiven Nebenbedingungen bestehen, zu lösen. Hierzu wird $\boldsymbol{x}^{(j)}$ verschoben und der Korrekturfaktor $\boldsymbol{\delta}^{(j)}$ gesucht, der

$$\begin{array}{rcl} \min_{\boldsymbol{\delta}} \frac{\partial g(\boldsymbol{x}^{(j)}+\boldsymbol{\delta})}{\partial \boldsymbol{x}^{(j)}} & = & \frac{1}{2}\boldsymbol{\delta}^T G\boldsymbol{\delta} + \boldsymbol{\delta}^T\left(\boldsymbol{g} + G\boldsymbol{x}^{(j)}\right) \\ \text{mit} \quad \boldsymbol{a}_i^T\boldsymbol{\delta} & = & \boldsymbol{0}\,, \qquad i \in \mathcal{A} \end{array} \tag{2.31}$$

löst. Diese Art von Optimierungsaufgabe kann wie in Kapitel 2.8.1 beschrieben gelöst werden. Sollte $\boldsymbol{x}^{(j)} + \boldsymbol{\delta}^{(j)}$ auch den Nebenbedingungen, die nicht in $\mathcal{A}$ liegen, genügen, wird für die nächste Iteration $\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)} + \boldsymbol{\delta}^{(j)}$ gesetzt. Sollte dies nicht gelten, wird eine lineare Suche in Richtung

von $\boldsymbol{\delta}^{(j)}$ durchgeführt, um den bestmöglichen Punkt zu finden. Dies erreicht man, indem man die Lösung aus (2.31) als Suchrichtung $\boldsymbol{s}^{(j)}$ definiert und die Schrittweite $\alpha^{(j)}$ berechnet über

$$\alpha^{(j)} = \min\Big(1, \min_{\substack{i\,:\,i \notin \mathcal{A},\\ \boldsymbol{a}_i^T \boldsymbol{s}^{(j)} < 0}} \frac{\boldsymbol{b}_i - \boldsymbol{a}_i^T \boldsymbol{x}^{(j)}}{\boldsymbol{a}_i^T \boldsymbol{s}^{(j)}}\Big) \tag{2.32}$$

um damit $\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)} + \alpha^{(j)} \boldsymbol{s}^{(j)}$ zu bestimmen. Wenn in (2.32) $\alpha^{(j)} < 1$ ist, wird eine weitere Nebenbedingung aktiv. Der Index q, der das Minimum in (2.32) bestimmt, wird dann mit in die Menge $\mathcal{A}$ aufgenommen.

Ist $\boldsymbol{x}^{(j)}$ die Lösung des aktuellen Optimierungsproblems mit Gleichungsnebenbedingungen (es gilt also $\boldsymbol{\delta} = \mathbf{0}$), so ist es möglich, Lagrange-Multiplikatoren $\boldsymbol{\lambda}^{(j)}$ für die aktiven Nebenbedingungen zu bestimmen. Wenn $\boldsymbol{\lambda}_i \geq 0$, $i \in E$ ist, erfüllen die Vektoren $\boldsymbol{x}^{(j)}$ und $\boldsymbol{\lambda}^{(j)}$ alle Kuhn-Tucker-Bedingungen.

Es muss also ein Test durchgeführt werden, der festlegt, ob $\boldsymbol{\lambda}_i^{(j)} \geq 0 \; \forall i \in \mathcal{A}^{(j)} \cap E$. Sollte dies der Fall sein, sind die Kuhn-Tucker-Bedingungen erfüllt. Sollte $q(x)$ konvex sein, wovon ausgegangen wird, reichen diese Bedingungen aus, um ein globales Ergebnis erwarten zu können. Ansonsten existiert ein Index q mit $q \in \mathcal{A}^{(j)} \cap E$, für den gilt $\boldsymbol{\lambda}_q^{(j)} < 0$. In diesem Fall wird $q(x)$ darüber reduziert, dass die q-te Nebenbedingung inaktiv wird. Das q muss aus der Menge $\mathcal{A}$ genommen werden, und der Algorithmus kann mit dem Lösen des Optimierungsproblems mit nun neuen Gleichungsnebenbedingungen fortfahren. Existiert mehr als ein Index q, für den gilt $\boldsymbol{\lambda}_q^{(j)} < 0$, so wird

$$q = \min_{i \in \mathcal{A} \cap E} \boldsymbol{\lambda}_i^{(j)} \tag{2.33}$$

genommen. Die Abbildung 2.4 gibt einen Überblick über den Algorithmus.

Beispiel 2.8.4. *Ein quadratisches Optimierungsproblem besitzt die Begrenzungen $x \geq 0$ und die Nebenbedingung $\boldsymbol{a}^T x \geq b$. Im Punkt $x^{(1)}$ sind die beiden Begrenzungen $x_1 \geq 0$ und $x_2 \geq 0$ aktiv und $x^{(1)}$ löst das OG (2.31) direkt. Bei der Bestimmung der Lagrange-Multiplikatoren wird festgestellt, dass die Nebenbedingung $x_2 \geq 0$ einen negativen Lagrange-Multiplikator besitzt. Aus diesem Grund wird die Nebenbedingung wieder inaktiv gesetzt, so dass nun nur noch $x_1 \geq 0$ aktiv ist. Die Lösung des nun neu entstandenen OGs ist $x^{(2)}$, wobei $\delta = x^{(2)} - x^{(1)}$ ist. $x^{(2)}$ ist ein möglicher Punkt, und somit wird $x^{(2)}$ als nächste Lösungsmöglichkeit angenommen. Nachdem ein weiteres Mal die Lagrange-Multiplikatoren bestimmt wurden, wird festgestellt, dass nun die Nebenbedingung $x_1 \geq 0$ einen negativen Multiplikator besitzt. Die Nebenbedingung wird inaktiv gesetzt. Die Lösung des nun wiederum neu entstandenen OGs ist $x^\star$, was aber keine mögliche Lösung unter den gegebenen Nebenbedingungen ist. Somit wird die Suchrichtung über $s^{(2)} = x^\star - x^{(2)}$ bestimmt. Bei der Suche entlang dieser Richtung (2.32) wird $x^{(3)}$ als bestmögliche Lösung gefunden. Hiermit wird die Nebenbedingung $a^T x \geq b$ aktiv. Da $x^{(3)}$ das aktuelle OG nicht löst, können auch keine Lagrange-Multiplikatoren bestimmt werden, aber stattdessen wird das eigentliche OG gelöst, und es ergibt sich $x^{(4)}$ als nächste Annäherung an die globale Lösung. Die Bestimmung der Lagrange-Multiplikatoren ergibt, dass $x^{(4)}$ die Lösung des Optimierungsproblems unter Ungleichungsnebenbedingungen ist, und der Algorithmus bricht ab.*

Gegeben sind $\boldsymbol{x}_1$ und $\mathcal{A}$

$\boldsymbol{\delta} = \mathbf{0}$ löst nicht (2.31)

Y

N

Bestimmen der Lagrange-Multiplikatoren $\boldsymbol{\lambda}^{(j)}$

$q = \min_{i \in \mathcal{A} \cap E} \boldsymbol{\lambda}_i^{(j)}$

$\boldsymbol{\lambda}_q^{(j)} \geq 0$

Y

N

Abbruch des Algorithmus mit dem Ergebnis $\boldsymbol{x}^\star = \boldsymbol{x}^{(j)}$

Entfernen von q aus der Menge $\mathcal{A}$

Lösen von (2.31), um $\boldsymbol{s}^{(j)}$ zu bestimmen

Bestimmen von $\alpha^{(j)}$ durch Lösen von (2.32)

$\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)} + \alpha^{(j)} \boldsymbol{s}^{(j)}$

Sollte $\alpha^{(j)} < 1$ sein, füge p zu $\mathcal{A}$ hinzu

Solange $\alpha^{(j)} \neq 0$

Abbildung 2.4: Nassi-Shneidermann-Diagramm zur Methode des aktiven Datensatzes

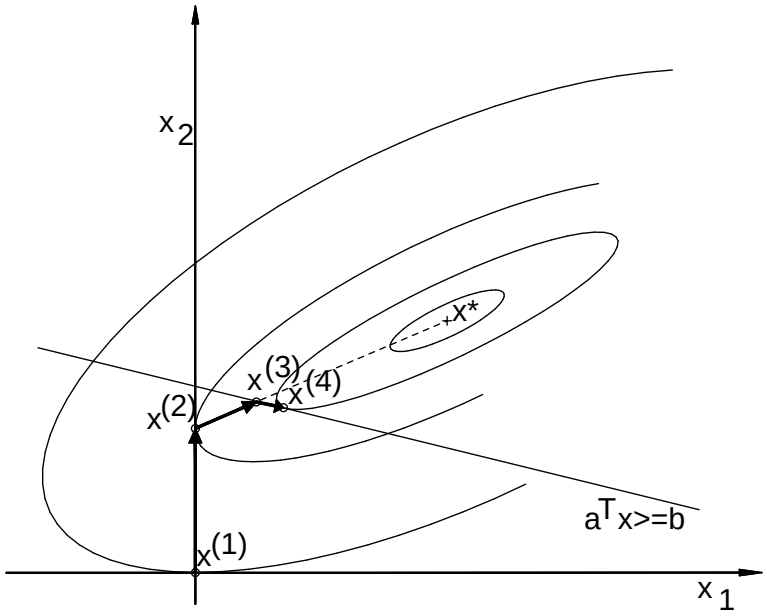

Abbildung 2.5: Grafische Darstellung des Beispiels 2.8.4

Kapitel 3

Ausreißererkennung

3.1 Motivation

Bei einer Datenanalyse werden alle gegebenen Datenpunkte mit in die Berechnung aufgenommen. Hierbei wäre es von Vorteil, wenn man diejenigen Datenpunkte, die aus der groben Struktur der gesamten Punkte herausfallen, eliminieren könnte, um bessere Ergebnisse erzielen zu können. Diese Punkte werden im Folgenden als **„Ausreißer“** bezeichnet. Solche Ausreißer können zu Fehlinterpretationen der Daten führen.
Zur Veranschaulichung wird im Folgenden eine Least-Squares-Approximation auf einer Menge von Daten durchgeführt. Der Datensatz hat folgende Struktur:

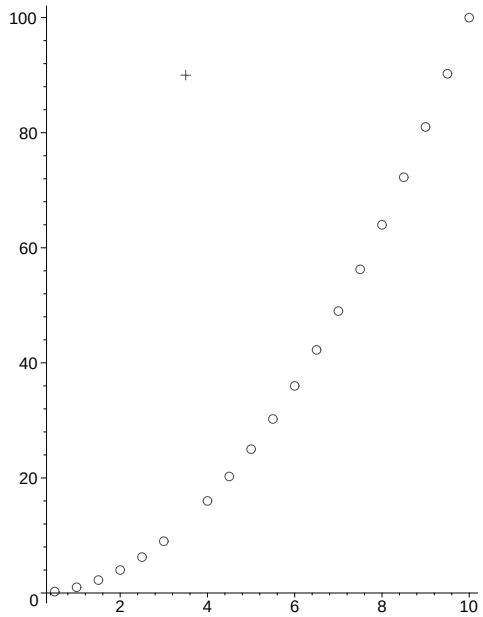

Abbildung 3.1: Datensatz mit Ausreißer

In der Abbildung 3.1 ist zu erkennen, dass der siebte Datenpunkt ein Ausreißer sein muss. Er fällt aus der globalen Struktur der restlichen Punkte heraus. Führt man nun eine Least-Squares-Approximation mit der Modellfunktion $f(x) = a \cdot x^2 + b \cdot x + c$ durch, so wirkt sich die lokale Ungenauigkeit des Punktes auf die globalen Kenngrößen der Least-Squares-Approximation aus, was in Abbildung 3.2 auch zu erkennen ist.

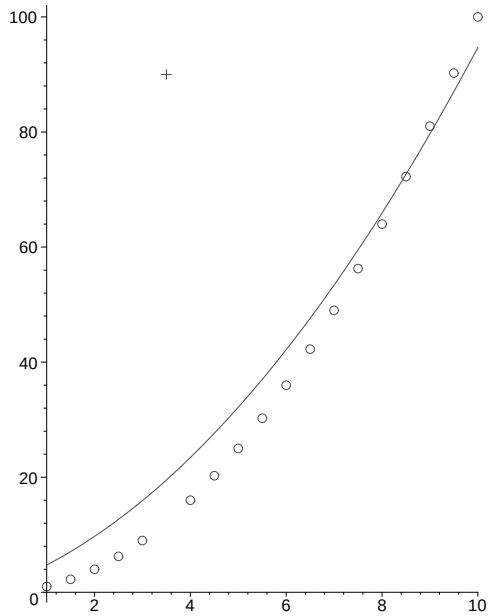

Abbildung 3.2: Least-Squares-Analyse für einem Datensatz mit einem Ausreißer

Findet man nun ein Verfahren, welches diesen Ausreißer erkennt, so könnte man ihn eliminieren. Wie man in der Abbildung 3.3 erkennen kann, wird damit die Qualität der Approximation um ein Vielfaches gesteigert.

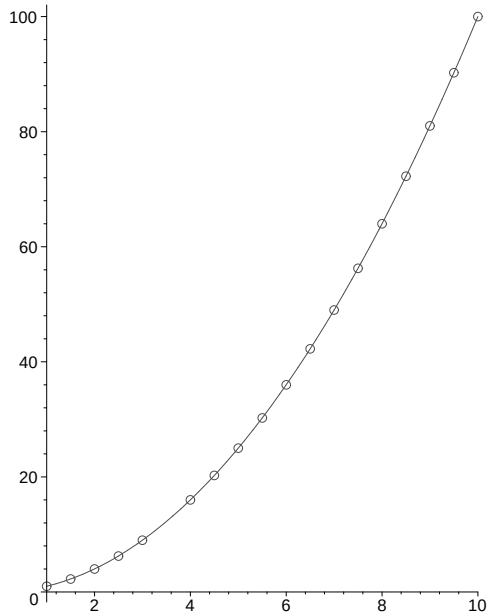

Abbildung 3.3: Least-Squares-Analyse für einem Datensatz ohne Ausreißer

In diesem Beispiel wird deutlich, dass Ausreißer einen erheblichen Einfluss auf die Analyse der Daten haben können. Dies liegt im Wesentlichen daran, dass die meisten Analysen für jeden Datenpunkt eine gleiche Gewichtung annehmen und somit führen große Fehler für wenige Punkte zu schlechten Approximationen.

Die Hauptschwierigkeit bestehen nun darin, die lokalen Ausreißer mathematisch zu erkennen und zu beurteilen, ob deren Abweichungen als signifikant zu betrachten sind, oder ob es sich bei deren Schwankungen um übliche Ungenauigkeiten handelt. Die im Folgenden beschriebenen Methoden und Algorithmen basieren auf [13], [12] und [14].

3.2 A priori Ausreißerbestimmung

Ein erster Ansatz zur Erkennung von Ausreißern vor einer Approximation könnte der euklidische Abstand jedes einzelnen Datenpunktes $\boldsymbol{z}_i = (x_{i,1}, \ldots, x_{i,l}, y_i)$ zum Datenzentrum $\overline{\boldsymbol{z}} = (\overline{x}_1, \ldots, \overline{x}_l, \overline{y})$ sein.

Definition 3.2.1 (Euklidischer Abstand). *Der **Euklidische Abstand** $d(\boldsymbol{a}, \boldsymbol{b})$ zweier Punkte $\boldsymbol{a} = (a_1, \ldots, a_r)$ und $\boldsymbol{b} = (b_1, \ldots, b_r)$ aus $\mathbb{R}^r$ ist definiert als*

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(a_1 - b_1)^2 + \ldots + (a_r - b_r)^2}$$

Ist $\boldsymbol{z}_i = (x_{i,1}, \ldots, x_{i,l}, y_i),\ i = 1, \ldots, n$ einer der Datenpunkte und sind $\overline{x}_1, \ldots, \overline{x}_l, \overline{y}$ die arithmetischen Mittel, so misst

$$d(\boldsymbol{z}_i, \overline{\boldsymbol{z}}) = \sqrt{(x_{i,1} - \overline{x}_1)^2 + \ldots + (x_{i,l} - \overline{x}_l)^2 + (y_i - \overline{y})^2}$$

den euklidischen Abstand der Beobachtung $\boldsymbol{z}_i = (x_{i,1}, \ldots, x_{i,l}, y_i)$ vom Datenzentrum $\overline{\boldsymbol{z}} = (\overline{x}_1, \ldots, \overline{x}_l, \overline{y})$.

Dass diese Art der Ausreißerfindung nicht optimal ist, zeigt sich schon in Abbildung 3.1. Hier ist der Euklidische Abstand des Ausreißers zum Datenzentrum geringer als der Abstand manch anderer Datenpunkte, die nicht als Ausreißer zu deklarieren sind. Die Ausrichtung der Punktwolke ist hier nicht beachtet worden. Will man auch die Ausrichtung der Datenpunkte mit in die Bestimmung einfließen lassen, könnte die sogenannte „Mahalanobis-Distanz“ $\Delta(\boldsymbol{z}_i, \overline{\boldsymbol{z}})$ zum Erfolg verhelfen. Hierbei wird dem Euklidischen Abstand eine Gewichtung über die inverse Kovarianzmatrix beigefügt. Dabei kann man die Streuungen in Richtung der unterschiedlichen Koordinaten-Achsen mit berücksichtigen.

Definition 3.2.2 (Mahalanobis-Distanz). *Gegeben sind die Datenpunkte $\boldsymbol{z}_i = (x_{i,1}, \ldots, x_{i,l}, y_i)$ für $i = 1, \ldots, n$ und die zugehörigen arithmetischen Mittel $\overline{x}_1, \ldots, \overline{x}_l, \overline{y}$. Unter $x_{i,l+1}$ ist y_i zu verstehen. Die **quadrierte Mahalanobis-Distanz** eines Datenpunktes $\boldsymbol{z}_i = (x_{1,i}, \ldots, x_{l+1,i})$ zum Datenzentrum $\overline{\boldsymbol{z}} = (\overline{x}_1, \ldots, \overline{x}_{l+1})$ ist definiert als*

$$\Delta^2(\boldsymbol{z}_i, \overline{\boldsymbol{z}}) = \begin{pmatrix} x_{i,1} - \overline{x}_1 \\ \vdots \\ x_{i,l+1} - \overline{x}_{l+1} \end{pmatrix}^T \cdot \begin{pmatrix} a_{1,1} & \ldots & a_{1,l+1} \\ \vdots & \ddots & \vdots \\ a_{l+1,1} & \ldots & a_{l+1,l+1} \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_{i,1} - \overline{x}_1 \\ \vdots \\ x_{i,l+1} - \overline{x}_{l+1} \end{pmatrix}$$

wobei für $a_{i,j}$ gilt

$$a_{i,j} = \begin{cases} \frac{1}{l} \cdot \sum_{k=1}^{l+1} (x_{i,k} - \overline{x}_i)^2 & \text{für} \quad i = j \\ \frac{1}{l} \cdot \left(\sum_{k=1}^{l+1} x_{i,k} x_{j,k} - l \overline{x_i}\, \overline{x_j} \right) & \text{für} \quad i \neq j \end{cases}$$

Eine Standardisierung der Daten auf Mittelwert 0 und Varianz 1 vereinfacht die Berechnung der Mahalanobis-Distanz. Geometrisch lässt sich der Übergang vom Euklidischen Abstand zur Mahalanobis-Distanz so interpretieren, dass man die erste Achse (X_1-Achse) des Koordinatensystems in Richtung der stärksten Streuung des Datensatzes dreht. In dieses gedrehte Koordinatensystem legt man nun um den Datensatz keinen Kreis, sondern eine Ellipse. Dass dieser Ansatz der Ausreißererkennung nicht immer von Vorteil ist, ist in Abbildung 3.4 zu erkennen.

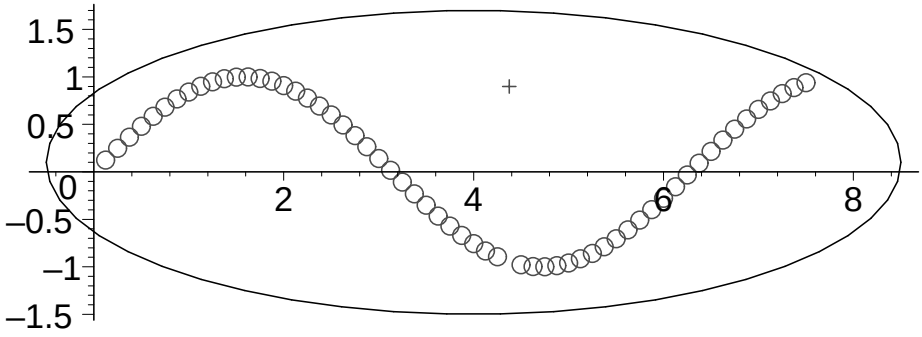

Abbildung 3.4: Datensatz mit einem nicht *außerhalb* der Ellipse liegenden Ausreißer

Der mit einem Kreuz markierte Datenpunkt liegt eindeutig nicht in der globalen Struktur aller anderen Datenpunkte. Er kann aber auch nicht von den anderen Punkten durch Einfügen einer Ellipse getrennt werden. Es wird die Approximation des kompletten Datensatzes benötigt, um eine Aussage treffen zu können, ob ein Punkt Ausreißer ist oder nicht.

3.3 A posteriori Ausreißerbestimmung

Nach einer Approximation haben alle Datenpunkte die ermittelte Modellfunktion beeinflusst. Ziel ist es nun, aus diesen Einflüssen Rückschlüsse auf die Lage der einzelnen Datenpunkte zu ziehen und demnach gegebenenfalls Ausreißer aus der Datenmenge herauszunehmen.

3.3.1 Lineare Modelle

Aus Vereinfachungsgründen werden zunächst nur lineare Modelle der Form

$$\boldsymbol{y} \;=\; X \cdot \boldsymbol{a} \tag{3.1}$$

mit

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{pmatrix} \qquad \boldsymbol{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_p \end{pmatrix} \tag{3.2}$$

in Betracht gezogen. Die Matrix X wird als *Designmatrix* oder auch *Regressormatrix* bezeichnet. Falls X vollen Rang hat, lautet der durch die Optimierung entstandene Fehler:

$$\begin{aligned} d_i^2 &= Var(y_i - M(\boldsymbol{a}, x_i)) \\ &= \sigma^2 \cdot \Big(1 - \underbrace{x_i^T \left(X^T X\right)^{-1} x_i}_{h_{ii}}\Big) \end{aligned} \tag{3.3}$$

Satz 3.3.1 (Hat-Matrix). *Die Matrix H wird als Hat-Matrix bezeichnet, sofern für sie gilt:*

$$H \;=\; X \left(X^T X\right)^{-1} X^T \tag{3.4}$$

Die Diagonalelemente werden mit h_{ii} bezeichnet.
Bemerkung 3.3.2. *Die Matrix H besitzt die Eigenschaft eine Orthogonalprojektionsmatrix zu sein, d.h.:*

$$H^T = H \quad , \qquad H^2 = H$$

Aufgrund von Bemerkung 3.3.2 gilt:

$$rang(H) \quad = \quad tr(H) \quad = \quad \sum_{i=1}^{n} h_{ii} \quad = \quad p$$

Für jedes h_{ii} sollte also idealerweise gelten: $h_{ii} \approx \frac{p}{n}$

Definition 3.3.3 (Hebelpunkte). *Ein Datenpunkt x_i wird als **Hebelpunkt** oder **high leverage point** bezeichnet, wenn gilt:*

$$h_{ii} \quad > \quad 2 \cdot \frac{p}{n} \tag{3.5}$$

Ist ein Datum ein Hebelpunkt, so ist dies ein Indiz für einen Ausreißer.

Ist ein Datenpunkt kein *Hebelpunkt*, kann er immer noch aus der Struktur der restlichen Datenpunkte herausfallen. Aus diesem Grund müssen weitere Überlegungen zur automatischen Ausreißererkennung angestellt werden.

Im Folgenden wird davon ausgegangen, dass die x-Werte eines Datums auf jeden Fall korrekt sind. Um nun Datenpunkte zu identifizieren, die einen starken Einfluss auf die Lage der Regressionsgeraden ausüben, muss überprüft werden, ob es sich bei y_i um einen *„unüblichen“* Wert handelt. Eine Möglichkeit sich diesem Problem zu nähern, ist über die Residuen. Man muss dabei aber bedenken, dass *Hebelpunkte* zu tendenziell kleineren Residuen führen. Große Hebelwirkung hat gerade zur Folge, dass die Regressionskurve näher an den entsprechenden Datenpunkt *„herangezogen“* wird.

Für eine *optimale* Approximation sind die Varianzen der Residuen r_i immer gleich, also $Var(r_i) = \sigma^2$. Es lässt sich nun zeigen, dass die Residuen $\hat{r}_i$ zwischen den Datenpunkten und der Kleinsten-Fehlerquadrat-Approximation unterschiedliche Varianzen besitzen, und zwar $Var(\hat{r}_i) = \sigma^2(1-h_{ii})$. Daher sollten die Residuen bezüglich der Kleinsten-Fehlerquadrat-Approximation mit Hilfe ihrer Varianzen standardisiert werden:

$$\hat{r}_i^{\circ} \quad = \quad \frac{\hat{r}_i}{\hat{\sigma}\sqrt{1-h_{ii}}} \tag{3.6}$$

In 3.6 sind $\hat{\sigma}$ und $\hat{r}_i$ nicht unabhängig voneinander, was bedeutet, dass hohe $\hat{r}_i$ zu grossen $\hat{\sigma}$ führen. Diese Eigenschaft wirkt sich ungünstig auf die Verteilung der $\hat{r}_i^{\circ}$ aus, denn sie sind somit nicht mehr t-verteilt. Deshalb geht man zu den studentisierten Residuen über.

Definition 3.3.4 (studentisierte Residuen). *Als studentisierte Residuen (studentized residuals oder auch studentized deleted residuals) werden diejenigen Residuen bezeichnet, die mit Hilfe ihrer Varianz standardisiert wurden, wobei $\hat{\sigma}$ durch $\hat{\sigma}_{(-i)}$ ersetzt wird. $\hat{\sigma}_{(-i)}$ ist eine Annäherung von σ unter Ausschluss von i, was dazu führt, dass $\hat{r}_i^{\star}$ t-verteilt ist.*

$$\hat{r}_i^{\star} = \frac{\hat{r}_i}{\hat{\sigma}_{(-i)}\sqrt{1-h_{ii}}} \quad \sim \quad t(n-l-2) \tag{3.7}$$

n ist hierbei die Anzahl der Datenpunkte und l die Dimension des x-Vektors eines Datenpunktes.

Da die studentisierten Residuen t-verteilt sind, lassen sich Ausreißer mit einem einfachen Signifikanztest identifizieren, und zwar als Werte mit

$$|\hat{r}_i^\star| \quad > \quad t_{\left(\frac{1-\alpha}{2n}\right)}(n-l-2) \tag{3.8}$$

wobei α dem Signifikanzniveau entspricht. Die Division durch n trägt der Tatsache Rechnung, dass implizit n simultane Tests durchgeführt werden. Da große Quantile oftmals in Tabellen der t-Verteilung fehlen und es in Computerprogrammen zu aufwändig wäre, diese immer wieder korrekt zu bestimmen nutzt man Bemerkung 3.3.5.

Bemerkung 3.3.5. *Besitzt ein Datum ein studentisiertes Residuum $\hat{r}_i^\star$, für dass gilt*

$$|\hat{r}_i^\star| \quad > \quad 2 \tag{3.9}$$

so ist dies ein Indiz für einen Ausreißer.

3.3.2 Nichtlineare Modelle

Für den Fall, dass die zugrundeliegende Modellfunktion nicht linear ist, können die Erkenntnisse aus Abschnitt 3.3.1 ebenfalls genutzt werden. Hierzu muss nur die Modellfunktion M linearisiert werden, so dass sich die Designmatrix X wie in (3.10) gezeigt verändert.

$$X \quad = \quad \begin{pmatrix} \frac{\partial M(a_1,\ldots,a_p,x_1)}{\partial a_1} & \cdots & \frac{\partial M(a_1,\ldots,a_p,x_1)}{\partial a_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial M(a_1,\ldots,a_p,x_n)}{\partial a_1} & \cdots & \frac{\partial M(a_1,\ldots,a_p,x_n)}{\partial a_p} \end{pmatrix} \tag{3.10}$$

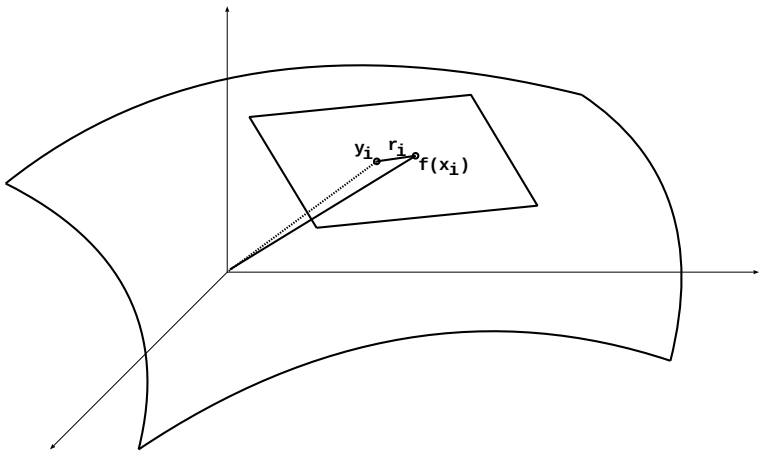

Abbildung 3.5: Grafische Darstellung der Linearisierung

3.4 Fazit

In diesem Kapitel wurden drei Verfahren zur automatischen Ausreißererkennung vorgestellt.

1. Mahalanobis-Distanzen (a priori)
2. Hebelpunkte (a posteriori)
3. studentisierte Residuen (a posteriori)

Man kann nicht von einem optimalen Verfahren sprechen, denn jedes Verfahren besitzt seine Vor- und Nachteile. Erst die Kombination der drei Verfahren führt zum gewünschten Erfolg. Bereits vor einer Approximation kann die Mahalanobis-Distanz genutzt werden, um erste Ausreißer zu erkennen. Leider achtet sie aber nur bedingt auf die Struktur der Datenpunkte. Die beiden übrigen Verfahren sind dazu in der Lage, benötigen aber vor ihrer Nutzung eine Approximation. Im Programmpaket zu dieser Arbeit sind alle drei Verfahren implementiert.

Kapitel 4

Methoden der Differentiation

4.1 Einleitung

Zur Anwendung der in Kapitel 2 und 3 vorgestellten Verfahren werden Methoden zur Differentation benötigt. Im Computeralgebrasystem Maple stehen dem Benutzer zwei verschiedene Arten der Differentiation zur Verfügung, die in diesem Kapitel vorgestellt werden. Zum einen ist das die *symbolische Differentiation*, die sicher die bekanntere der beiden Differentiationsarten ist, zum anderen die *automatische Differentiation*. Sie ist eine Verallgemeinerung der *symbolischen Differentiation*, denn mit ihr ist es nicht nur möglich Ausdrücke (Formeln) abzuleiten, sondern auch Prozeduren, bzw. allgemeine Programme. In den folgenden Abschnitten wird beispielhaft die Funktion

$$f(x_1, x_2) \quad = \quad \cos(x_1) + x_1 \cdot x_2 + e^{x_1} \tag{4.1}$$

auf beide Arten differenziert.

4.2 Symbolische Differentiation

Die symbolische Differentiation kann als klassische Differentiationsmethode bezeichnet werden. Sie basiert auf den Differentiationsregeln der Analysis. Es wird die zu differenzierende Formel (4.1) als Graph (siehe Abbildung 4.1) interpretiert, um auf diesen die Differentiationsregeln anzuwenden.

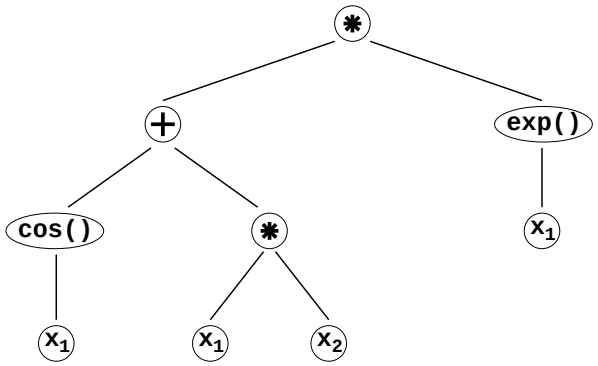

Abbildung 4.1: Graph zur *symbolischen Differentation* von Formel (4.1)

Das Ergebnis der symbolischen Ableitung ist wieder eine Funktion. In Maple kann die *symbolische Differentiation* mit der Prozedur `diff` durchgeführt werden. Die symbolischen Ableitungen nach x_1 bzw. x_2 ergeben:

```
> diff( f(x[1],x[2]), x[1]);
```
$$(-sin(x_1) + x_2)e^{x_1} + (cos(x_1) + x_1x_2)e^{x_1}$$
```
> diff( f(x[1],x[2]), x[2]);
```
$$x_1e^{x_1}$$

4.3 Automatische Differentiation

Die *automatische Differentiation*, auch algorithmische Differentiation oder Differentiation von Algorithmen genannt, versucht, im Gegensatz zur *symbolischen*, nicht nur symbolische Ausdrücke sondern Programme abzuleiten. Ziel ist es also zu einem Programm P ein Programm ∂P zu finden, das die Ableitung von P exakt berechnet.

Der wesentliche Unterschied der *automatischen Differentiation* zur *symbolischen* ist, dass die *symbolische Differentation*, basierend auf Graphen, Formeln ableiten kann, während die *automatische* Funktionen als Prozeduren betrachtet und geeignete Ableitungsprozeduren bestimmt.

Die Compiler der meisten Programmiersprachen sind nicht in der Lage, komplexe mathematische Ausdrücke auszuwerten. Deshalb zerlegen sie die Funktion in elementare Ausdrücke und werten diese dann aus. Anschließend werden die Teilergebnisse wieder zusammengefügt.

Gleiches macht Maple bei der *automatischen Differentiation*. Zunächst wird die Funktion bzw. das Programm in eine Prozedur elementarer Ausdrücke aufgesplittet. Dies würde für unsere Beispielfunktion (4.1) bedeuten:

```
f := proc(x_1,x_2)
        local t1, t2, t3, t4, t5, t6;
        t1 := cos(x_1);
        t2 := x_1 * x_2;
        t3 := exp(x_1);
        t4 := t1 + t2;
        t5 := t3 * t4;
      end proc;
```

Abbildung 4.2: Prozedur mit Elementarausdrücken zur Beispielfunktion (4.1)

Die *automatische Differentiation* ist zu weit mehr in der Lage, als hier in diesem Beispiel beschrieben. Sie kann nicht nur Elementarfunktionen $\{sin, cos, exp, \sqrt{}, \ldots\}$ ableiten, sondern sie besitzt auch die Möglichkeit komplexere Programme mit Schleifen und Abfragen zu differenzieren. In dieser Arbeit wird sich aber auf die Differentation von Elementarfunktionen beschränkt. Für die *automatische Differentiation* existieren die beiden Methoden *forward mode* und *reverse mode*. Diese werden nun kurz vorgestellt.

4.3.1 forward mode

Beim *forward mode* der automatischen Differentiation wird in der Prozedur f jeder Anweisung $u := g(v_1, \ldots, v_n)$ eine Anweisung $u_x := gp(v_1, \ldots, v_n)$ vorangestellt. $gp(v_1, \ldots, v_n)$ ist hierbei die formale Ableitung von $g(v_1, \ldots, v_n)$ bezüglich einer Variablen und $v_1, \ldots, v_n$ sind die lokalen Variablen oder formalen Parameter. Der letzten Anweisung wird nicht die Ableitung vorangestellt, sondern sie wird durch die Ableitung ersetzt. Sollten bei dieser Vorgehensweise unnötige Berechnungen entstehen, werden diese zur Effizienzsteigerung gestrichen. Besitzt f also m Anweisungen, so erhält man für ihre Ableitungsprozedur bis zu $2 \cdot m - 1$ Anweisungen. In Maple wird diese Differentiationsprozedur mit dem Operator `D` erzeugt.

```
> Df1 := D[1](f);
```

$$
\begin{aligned}
&Df1 := \mathbf{proc}(x\_1,\ x\_2)\\
&\mathbf{local}\ t1,\ t2,\ t3,\ t4,\ t1x\_1,\ t2x\_1,\ t3x\_1,\ t4x\_1;\\
&\quad t1x\_1 := -\sin(x\_1)\,;\\
&\quad t1 := \cos(x\_1)\,;\\
&\quad t2x\_1 := x\_2\,;\\
&\quad t2 := x\_1 * x\_2\,;\\
&\quad t3x\_1 := \exp(x\_1)\,;\\
&\quad t3 := t3x\_1\,;\\
&\quad t4x\_1 := t1x\_1 + t2x\_1\,;\\
&\quad t4 := t1 + t2\,;\\
&\quad t3x\_1 * t4 + t3 * t4x\_1\\
&\mathbf{end\ proc}
\end{aligned}
$$

Abbildung 4.3: Prozedur für die Ableitung von f nach x_1

Das die gerade erzeugte Prozedur korrekt ist, kann überprüft werden, indem man von der mit dem Kommando `diff` erzeugten Ableitung die mit dem Operator `D` erzeugte Ableitung abzieht.

```
> diff(f(x_1,x_2),x_1)-Df1(x_1,x_2);
```

$$0$$

4.3.2 reverse mode

Der *forward mode* Algorithmus bekommt immer eine Eingangsvariable und berechnet Schritt für Schritt die partiellen Ableitungen der einzelnen Elementarausdrücke. Der *reverse mode* geht den umgekehrten Weg. Zunächst durchläuft er zur Bestimmung der Funktionswerte alle Elementarfunktionen. Im zweiten Schritt wird die Prozedur noch einmal in umgekehrter Richtung (rückwärts) durchlaufen, um die partiellen Ableitungen bezüglich der Ergebnisvariablen y_i zu erstellen. Hierfür werden die Elementarfunktionen t_i durch $\bar{t}_i$ mit $\bar{t}_i = \frac{\partial y}{\partial t_i}$ ersetzt.

Der Vorteil des *reverse mode* gegenüber dem *forward mode* ist also, dass nicht nur die Ableitung bezüglich **einer** sondern bezüglich **aller** Variablen berechnen werden kann. Im Allgemeinen werden mit ihm auch effizientere Programme erzeugt. Er kann aber nicht auf Prozeduren mit unbestimmten Schleifendurchläufen angewandt werden, was beim *forward mode* möglich ist.

Beide Verfahren sind in der Funktion `GRADIENT` aus dem Paket `codegen` implementiert. Um den Effizienzunterschied zwischen den beiden Verfahren zu verdeutlichen, wird die Prozedur (Abbildung 4.2) mit `GRADIENT` nach beiden Verfahren differenziert. Im Gegensatz zum `D`-Operator berechnet die `GRADIENT`-Funktion immer die Ableitungen nach allen Variablen.

forward mode:

```
> f_forward := codegen:-GRADIENT(f,mode=forward);
```

$f\_forward := \mathbf{proc}(x\_1,\ x\_2)$
local $dt1,\ dt2,\ dt3,\ dt4,\ t1,\ t2,\ t3,\ t4,\ t5,\ t6;$
$dt1 := \mathrm{array}(1..2);$
$dt2 := \mathrm{array}(1..2);$
$dt3 := \mathrm{array}(1..2);$
$dt4 := \mathrm{array}(1..2);$
$dt1_1 := -\sin(x\_1);$
$dt1_2 := 0;$
$t1 := \cos(x\_1);$
$dt2_1 := x\_2;$
$dt2_2 := x\_1;$
$t2 := x\_1 * x\_2;$
$dt3_1 := \exp(x\_1);$
$dt3_2 := 0;$
$t3 := \exp(x\_1);$
$dt4_1 := dt1_1 + dt2_1;$
$dt4_2 := dt1_2 + dt2_2;$
$t4 := t1 + t2;$
return $t4 * dt3_1 + t3 * dt4_1,\ t4 * dt3_2 + t3 * dt4_2$
end proc

Abbildung 4.4: Ableitung der Beispielfunktion (4.1) mit *forward mode*

reverse mode:

```
> f_reverse := codegen:-GRADIENT(f,mode=reverse);
```

$f\_reverse := \mathbf{proc}(x\_1,\ x\_2)$
local $dfr0,\ t1,\ t2,\ t3,\ t4;$
$t1 := \cos(x\_1);$
$t2 := x\_1 * x\_2;$
$t3 := \exp(x\_1);$
$t4 := t1 + t2;$
$dfr0 := \mathrm{array}(1..4);$
$dfr0_4 := t3;$
$dfr0_3 := t4;$
$dfr0_2 := dfr0_4;$
$dfr0_1 := dfr0_4;$
return $dfr0_3 * \exp(x\_1) + dfr0_2 * x\_2 - dfr0_1 * \sin(x\_1),\ dfr0_2 * x\_1$
end proc

Abbildung 4.5: Ableitung der Beispielfunktion (4.1) mit *reverse mode*

Dass beide Verfahren zum selben Ergebnis führen zeigt Abbildung 4.6.

```
> f_forward(x_1,x_2);
```
$$e^{x\_1}\,(\cos(x\_1)+x\_1\;x\_2)+e^{x\_1}\,(-\sin(x\_1)+x\_2),\;e^{x\_1}\;x\_1$$
```
> f_reverse(x_1,x_2);
```
$$e^{x\_1}\,(\cos(x\_1)+x\_1\;x\_2)+e^{x\_1}\;x\_2-e^{x\_1}\sin(x\_1),\;e^{x\_1}\;x\_1$$
```
> expand(f_forward(x_1,x_2)-f_reverse(x_1,x_2));
```
$$0$$

Abbildung 4.6: Ergebnisvergleich von *forward mode*- und *reverse mode*-Methode

Die Rechenkosten der erzeugten Ableitungsprozeduren können mit der Funktion `cost`, die sich ebenfalls im `codegen` Paket befindet, ermittelt werden.

```
> codegen:-cost(f_forward);
```
$$14\;storage+16\;assignments+16\;subscripts+4\;functions+6\;additions+5\;multiplications$$
```
> codegen:-cost(f_reverse);
```
$$8\;storage+9\;assignments+10\;subscripts+4\;functions+3\;additions+5\;multiplications$$

Abbildung 4.7: Kostenvergleich von *forward mode*- und *reverse mode*-Methode

Es ist zu erkennen, dass die *reserve mode*-Methode effizientere Ableitungsprozeduren erzeugt.

4.4 Fazit

Im Programmpaket zu dieser Arbeit wird die symbolische Differentiation angewandt. Die bei der Berechnung von restringierten nichtlinearen Ausgleichsproblemen zu differenzierenden Ausdrücke haben die Gestalt von *einfachen* Funktionen. Hierbei besitzt die automatische Differentiation keine Vorteile gegenüber der symbolischen. Die Implementierung der symbolischen Differentiation ist ein wenig einfacher, was die Entscheidung für diese Art der Ableitungsberechnung gegeben hat.

Kapitel 5

Programmiertechniken in Maple

5.1 Einleitung

Im folgenden Kapitel werden die Programmiertechniken erläutert, die zur Erstellung des Programmpaketes genutzt wurden. Zunächst wird das Prozedur- und Modulkonzept vorgestellt. Diese beiden Konzepte waren notwendig, um das Programmpaket effizient und übersichtlich zu programmieren. Desweiteren wird gezeigt, wie in Maple Quellcode für Fortran, C oder andere Programmiersprachen generiert werden kann. Diese Funktionalität wird benötigt, um Quellcode für externe Routinen zu erzeugen. Nachdem diese Routinen compiliert wurden, können sie in das Maple-System integriert und dann wie jede andere Maple-Funktion genutzt werden. Diese Einbindung wird ebenfalls beschrieben. Abschließend wird noch ein Einblick in die GUI-Oberfläche von Maple, Maplets genannt, gegeben.

5.2 Prozeduren

Wie in jeder höheren Programmiersprache ist es auch in Maple möglich, Sequenzen von Befehlen zusammenzufassen. Eine solche Zusammenfassung wird in Maple **Prozedur** genannt. Eine Prozedur wird wie folgt definiert ([16]):

```
Prozedurname := proc(Parameter)
                    local L;
                    global G;
                    options O;
                    description str;
                    Prozedurkörper
                end proc;
```

Abbildung 5.1: Syntax einer Prozedur

Jede Prozedur kann einen eigenen Namen besitzen, den *Prozedurnamen*. Über diesen Namen wird später auf die Prozedur zugegriffen. Über die Liste der *Parameter* können im Programmablauf der Prozedur Werte (call-by-value) übergeben werden. Im *Prozedurkörper* werden dann alle gewünschten Anweisungen nacheinander aufgelistet. Über *L* werden alle Variablen deklariert, die nur lokal in der Prozedur existieren sollen. Variablen, die global erreichbar sein sollen, müssen über *G* angegeben werden. Über *O* können der Prozedur verschiedene Optionen übergeben werden und mit *str* kann man die Prozedur beschreiben. Die Bezeichner *L*, *G*, *O* und *str* sind nur optional und müssen bei den Prozeduren nicht angegeben werden. Funktionen können in Maple über die Pfeil-Notation definiert werden.

```
f := x -> sin(x);
```

Intern werden diese Funktionen allerdings als Prozeduren realisiert. Diese Funktion entspricht folgender Prozedur:

```
f := proc(x)
        sin(x);
     end proc;
```

Ein Beispiel für eine Maple-Prozedur könnte lauten:

```
f := proc(x::name, n::posint)
        local i, erg;
        global y;
        option Copyright;
        description summation ;
        erg := sum(x[i] + y[i], i=1..n );
     end proc;
```

Abbildung 5.2: Ein Beispiel für eine Maple-Prozedur

5.3 Das Modulkonzept

Um mehrere Prozeduren zusammenfassen zu können, nutzt man **Module**. Diese Blockbildung findet hauptsächlich in folgenden Gebieten Anwendung:

- Einkapselung von Daten
- Definition von Paketen
- Modellierung von Objekten
- Generische Programmierung

Die Syntax eines Moduls besitzt folgende Struktur ([17]):

```
Modulname := module(Parameter)
                local L;
                export E;
                global G;
                options O;
                description str;
                Modulkörper
             end module;
```

Abbildung 5.3: Syntax eines Moduls

Die Syntax ähnelt weitgehend der Syntax der Prozeduren. Einziger Unterschied ist die optionale Angabe von *E*. Hierüber können einzelne Prozeduren aus dem Modulkörper für die Umgebung bekannt gemacht werden. Alle nicht zu exportierenden Prozeduren sind nur im Modul selbst bekannt. Ein einfaches Beispiel eines Moduls:

```
Generator := module()
                description "Stringgenerator";
                export genstring;
                local counter;

                counter := 0;

                genstring := proc()
                               counter:=counter+1;
                               string||counter;
                             end proc;
             end module;
```

```
Generator :- genstring();
```

Abbildung 5.4: Ein Beispiel für ein Maple-Modul

5.4 Generierung von Quellcode

Maple bietet die Möglichkeit, Ausdrücke, Prozeduren und Module in eine andere Programmiersprache zu übersetzen. Dieses Feature kann genutzt werden, um in Maple programmierte Algorithmen in ein bereits existierendes Programm einer anderen Programmiersprache einzufügen, oder einfach um eine schnellere Ausführung von Algorithmen zu erlangen. Es existieren Schnittstellen zu C, Fortran, Java, Matlab und VisualBasic. Außerdem besteht die Möglichkeit, sich Schnittstellen für andere Programmiersprachen selbst zu definieren. Das Programmpaket zur Generierung lautet `CodeGeneration` und besitzt folgende Syntax:

```
CodeGeneration[L]( expression, options )
```

Der Platzhalter *L* steht für die gewünschte Programmiersprache, in der *expression* generiert werden soll. Für *expression* dürfen aber nicht nur einfache Ausdrücke stehen, sondern auch Funktionen, Prozeduren, Module, etc.. Über *options* kann man gewünschte Nebenbedingungen, die bei der Codegenerierung eingehalten werden sollen, mit übergeben.

Um einen Eindruck über dieses Feature von Maple zu bekommen, folgt ein kleines Beispiel. Es soll eine stückweise definierte Funktion f in Fortran generiert werden.

```
f := proc(x::numeric)
        if x>0 then cos(x) else x^3+1 end if;
     end proc:

CodeGeneration[Fortran](f);
```

Abbildung 5.5: Beispiel für eine Quellcodegenerierung

Dabei erhält man folgenden Fortran-Quellcode:

```
      doubleprecision function f (x)
        doubleprecision x
        if (0.0D0 .lt. x) then
          f = cos(x)
          return
        else
          f = x ** 3 + 0.1D1
          return
        end if
      end
```

Abbildung 5.6: Ergebnis des Beispiels in Abbildung 5.5

5.5 Einbinden von externen Routinen

Das Maple-System ermöglicht es dem Nutzer, externe C-, Fortran- und Java-Routinen einzubinden.

Die Anbindung externer Routinen hat den Vorteil, dass man sich zum einen die Algorithmen nicht selber erarbeiten muss, und zum anderen, dass eine Performance-Steigerung möglich ist (siehe [15]).

Als Basis für die Kommunikation zwischen externen Routinen und Maple dienen **Shared Libraries**. Diese Bibliotheksdateien sind zwar nicht plattformunabhängig, jedoch ist es mit dem `system`-Kommando möglich, diese Dateien aus einem Maple-Worksheet heraus zu erzeugen.

Beim Einbinden einer externen Routine ist darauf zu achten, dass Maple die gewünschte Shared Library auch findet. Hierzu muss man der globalen Shell-Variable `LD_LIBRARY_PATH` das Verzeichnis, in dem sich die gewünschte Datei befindet, mit übergeben. Dies sollte auf Shell-Ebene erfolgen.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:Verzeichnis
```

Zum Einbinden nutzt man dann die Funktion `define_external`, die folgende Syntax besitzt:

```
define_external(FunctionName,
                LANGUAGE,
                arg1::type1,
                    ⋮
                argN::typeN,
                options,
                'LIB'=LibraryName);
```

Abbildung 5.7: Aufruf-Syntax der `define_external`-Funktion

Für den Platzhalter *FunctionName* muss man den Namen der einzubindenden Funktion einsetzen. Über *LANGUAGE* wird die Sprache angegeben, in der die Funktion geschrieben ist und über *LibraryName* der Name der Shared-Library-Datei, in der sich die gewünschte Funktion befindet. Die Angaben der Argumente *arg1* bis *argN* und deren Typen *type1* bis *typeN* sind optional und müssen der einzubindenden Funktion angepasst werden. Ein kleines Beispiel, wie eine Fortran-Routine eingebunden wird, lautet:

```
function addieren( zahl1, zahl2 ) result(erg)
  integer, intent(in) :: zahl1
  integer, intent(in) :: zahl2
  integer :: erg
  erg = zahl1 + zahl2
end function addieren
```

Abbildung 5.8: Ein externes Fortran-Programm

Für diese Funktion muss nun zunächst eine Shared Library erzeugt werden. Auf einem Linux-System könnte dies beispielsweise mit dem Kommando **f90 -o fortran.so -shared fortran.f90** geschehen. Nun kann die Funktion in das Maple-System mit der Funktion `define_external` eingebunden werden:

```
define_external(addieren,
                FORTRAN,
                zahl1::integer[4],
                zahl2::integer[4],
                RETURN::integer[4],
                'LIB'="fortran.so");
```

Abbildung 5.9: Beispiel zur Einbindung von externen Routinen

Jetzt kann die externe Routine wie eine Maple-Funktion angesprochen werden:

```
addieren(17,4);
                21
```

5.6 Erzeugen von Maplet-Anwendungen

Das Paket `Maplets` bietet dem Anwender die Möglichkeit, eigene grafische Benutzeroberflächen für Maple-Programme zu erzeugen. Die Maplets sind kein Ersatz für die Worksheet-Oberfläche. Vielmehr bieten sie dem Programmierer, der sich mit dem Maple-System auskennt, die Möglichkeit Anwendungen mit Oberflächen zu erstellen, um diese anderen Benutzern zur Verfügung zu stellen. Eine solche Vorgehensweise kann erforderlich sein,

- wenn sich die Benutzer nicht mit dem Maple-System auseinandersetzen möchten,
- weil der gewünschte Algorithmus vielleicht sehr komplex ist,
- weil die Benutzer nicht mit der Maple-Syntax vertraut sind, aber dennoch Maple-Funktionalitäten nutzen wollen.

Das `Maplets`-Paket ist zwar neu, aber dennoch bereits zu umfangreich, um alle Einzelheiten hier zu erklären. Deshalb wird an einem kleinen Beispiel die wesentliche Struktur der Maplet-Programmierung dargestellt.

```
use Maplets[Elements] in
  maplet := Maplet(
              BoxColumn(
                  BoxRow("Is this your first Maplet?"),
                  BoxRow(
                     Button("Yes", Shutdown("Yes")),
                     Button("No",  Shutdown("No"))
                        )
                      )
                   );
end use:
Maplets[Display](maplet);
```

Abbildung 5.10: Beispiel eine Maplet-Anwendung

Alle „Bauteile“ die benötigt werden, um eine Maplet-Anwendung zu erstellen befinden sich in dem Paket `Maplets[Elements]`. Hierzu gehören unter anderem die Funktionen `Maplet`, `BoxColumn` und `BoxRow` sowie `Button` und `Shutdown`. Mit der Funktion `Maplet` erstellt man sich die Oberfläche, in der alle weiteren Funktionen angewendet werden können. Um das Maplet zu strukturieren, kann man mit den Funktionen `BoxRow` und `BoxColumn` die Oberfläche in Zeilen und Spalten aufteilen. In den beiden Funktionen können nun, durch Kommata getrennt, alle Elemente folgen, die in der entsprechenden Zeile / Spalte erscheinen sollen. In diesem Beispiel wurden in eine Zeile ein String *„Is this your first Maplet?“* und in eine zweite Zeile zwei Knöpfe eingefügt. Die Buttons werden über die Funktion `Button` erzeugt. Diese Funktion bekommt als erstes Argument einen String, der auf dem Knopf erscheinen soll, und als zweites eine Aktion, die beim Betätigen des Buttons ausgeführt werden soll. Bei beiden Buttons wird das Maplet mit der Funktion `Shutdown` beendet. Nur der Rückgabeparameter, eine Zeichenkette, ist unterschiedlich.

Nach der Definition der Maplet-Anwendung kann man sie mit der Funktion `Maplets[Display]` anzeigen lassen.

Abbildung 5.11: Beispiel einer einfachen Maplet-Anwendung

5.6.1 Dynamische Maplets

Leider ist es bei der Maplet-Programmierung nicht möglich, auf die Inhalte von Variablen zu reagieren, da `if`-Abfragen oder `for`-Schleifen in der Funktion `Maplet` nicht ausgeführt werden können. Man kann sich aber behelfen. In Maple müssen Variablen nicht mit ihrem Typ deklariert werden, was für die Erstellung von dynamischen Maplets von Vorteil ist. Man speichert sich zunächst die gewünschten Sequenzen auf eine Variable. Hierbei befindet man sich außerhalb der `Maplet`-Funktion und kann deshalb alle in Maple bekannten Routinen benutzen. In der `Maplet`-Funktion muss jetzt nur noch die Variable mit der gewünschten Sequenz eingefügt werden.

Es folgt ein kurzes Beispiel:

```
Punkt := ["13","17","23"]:

use Maplets[Elements] in
  if nops(Punkt) > 0 then
     Spalte:=BoxColumn(seq("x"||i||"="||(Punkt[i]),
                           i=1..nops(Punkt)));
  else
     Spalte := "Keine gültiger Punkt vorhanden";
  end if;
  maplet := Maplet(
             BoxRow(
                Spalte,
                Button("OK", Shutdown())
                  )
                ):
end use:
Maplets[Display](maplet);
```

Abbildung 5.12: Beispiel für eine dynamische Maplet-Programmierung

Bei der Ausführung erhält man folgendes Maplet:

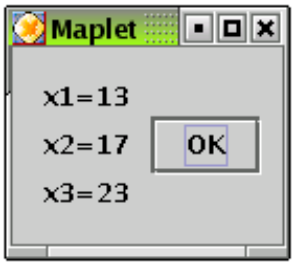

Abbildung 5.13: Ergebnis des dynamischen Maplets aus Abbildung 5.12

Kapitel 6

Funktionalität des Programmpaketes

6.1 Überblick

Das dieser Diplomarbeit zugrundeliegende Programmpaket ist aus den in den vorherigen Kapiteln beschriebenen Bausteinen erstellt worden. Einen Überblick über die verwendeten Elemente gibt die Abbildung 6.1.

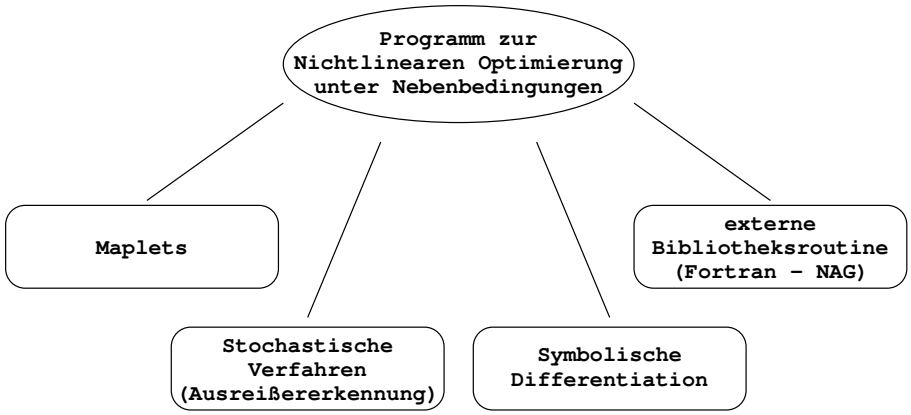

Abbildung 6.1: Im Programmpaket verwendete Elemente

Zur Lösung des numerischen Verfahrens der nichtlinearen Optimierung unter Nebenbedingungen wird eine Bibliotheksfunktion aus der NAGfl90 Library genutzt. Wie man eine solche Bibliotheksfunktion in Maple integrieren kann, ist im Kapitel 5 vorgestellt worden. Die Bibliotheksfunktion benötigt partielle Ableitungen der Modellfunktion. Hierfür wurden im Kapitel 4 zwei Arten von Ableitungsmethoden vorgestellt, die symbolische und die automatische Differentiation. Die Entscheidung in dieser Arbeit die symbolische Differentiation zu nutzen, beruht auf der Tatsache, dass es sich bei den zu approximierenden Modellfunktionen meist um „einfache“ Funktionen handelt. Es kann also sowohl die symbolische Differentiation mit der `diff`-Funktion also auch die automatische Differentiation mit dem `D`-Operator genutzt werden. Jedoch wurde die `diff`-Funktion im Kern von Maple implementiert, was in diesem Fall zu Effizienzvorteilen gegenüber dem `D`-Operator führt.

Das Programmpaket gliedert sich in zwei wesentliche Teilbereiche. Zum einen in die stochastischen / numerischen Verfahren und zum anderen in die darüberliegende grafische Oberfläche, wobei die Verfahren auch unabhängig von der grafischen Oberfläche genutzt werden können (siehe Abschnitt 6.3).

Da die nichtlineare Optimierung mit Hilfe von externen Routinen implementiert wurde, werden im Folgenden die Interaktionen zwischen Maple und dem externen Programm beschrieben.

Es wird per CodeGenerator das Hauptprogramm erzeugt, welches die Bibliothek einbindet. Diese

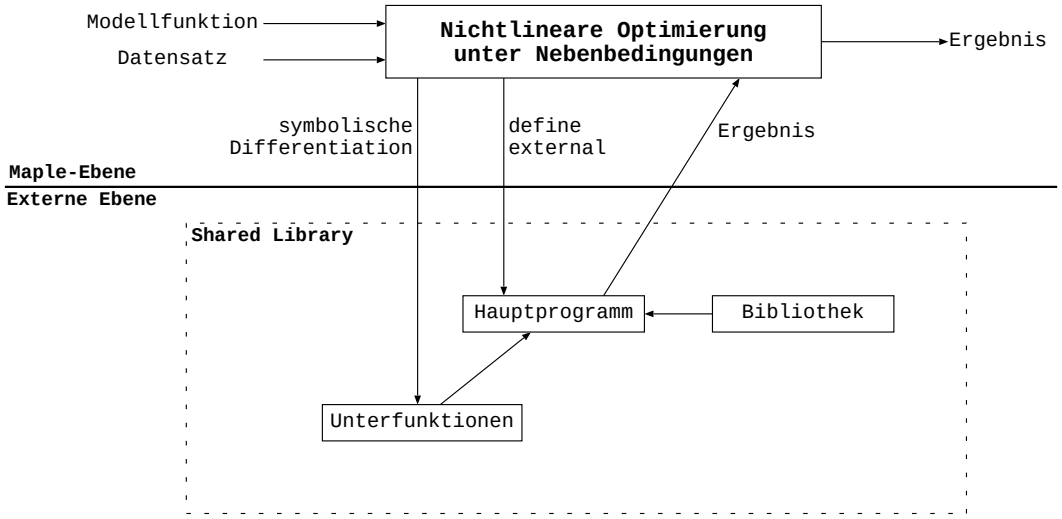

Abbildung 6.2: Einbindung des numerischen Verfahrens in Maple

Bibliothek benötigt noch weitere Unterfunktionen, die unter anderem die partiellen Ableitungen der Modellfunktion beinhalten. Die partiellen Ableitungen werden über die symbolische Differentiation erzeugt und die Unterfunktionen dann wieder über den CodeGenerator erstellt. Über das `system`-Kommando wird dieses externe Programm übersetzt und eine Shared Library erzeugt. Anschließend kann diese Library über `define_external` in das Maple-Programm eingebunden werden. Von nun an kann das externe Programm wie eine Maple-Funktion genutzt werden. Die Abbildung 6.2 gibt nochmals einen Überblick über diese Vorgehensweise.

Im Modul `NonLinearFit` befindet sich die Startoberfläche, von der aus man alle Funktionen des Programmpaketes ansteuern kann. Die Hilfsroutinen sind gesondert in dem Modul `NonLinearFitTools` gespeichert. Alle weiteren Routinen werden aufgeteilt in die Funktionen zur grafischen Oberfläche und die numerischen Algorithmen. Die Oberflächenfunktionen befinden sich in den Modulen `NLF_###` und die Funktionen, die die Algorithmen enthalten, befinden sich in `NLF_###Function`. Alle Module sind in den gleichnamigen Dateien gespeichert. Diese Aufteilung hat den Vorteil, dass die verschiedenen Fitting-Routinen voneinander abgekapselt sind, und dass bei der Nutzung im Worksheet nicht unnötige Funktionen mit eingebunden werden müssen. Die Oberfläche zur restringierten nichtlinearen Optimierung befinden sich im Modul `NLF_ApproxConstraint`. Die dazugehörigen numerischen Funktionen sind im Modul `NLF_ApproxConstraintFunction` zu finden. Funktionen, die in Abhängigkeit zum Betriebssystem stehen, wie das Einbinden externer Routinen, werden in den Modulen `NonLinearFitLinux` separat abgespeichert. Das Forschungszentrum Jülich besitzt keine Windows-Lizenz für die NAGfl90-Library, weshalb die Implementierung für Windows noch nicht erfolgt ist. Da die Ausreißererkennung für alle Fitting-Routinen zur Verfügung stehen soll, ist diese in eigene Module abgelegt worden. Das Oberflächenmodul lautet `NLF_Exclude` und die dazugehörigen Funktionen liegen in `NLF_ExcludeFunction`.

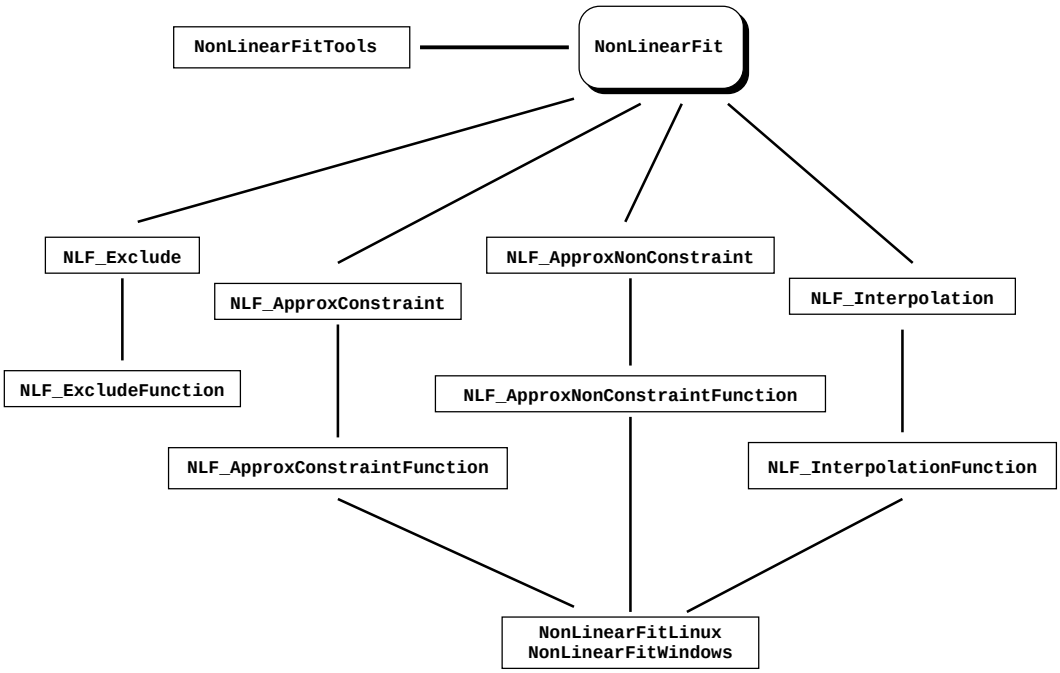

Abbildung 6.3: Die Hauptmodule und deren Abhängigkeiten

6.2 Die grafische Oberfläche

Das Startpanel der grafischen Oberfläche befindet sich in dem Modul `NonLinearFit` und kann über die Funktion `NonLinearFitMaplet` angesprochen werden. In Abbildung 6.4 ist das Eingangsfenster dargestellt.

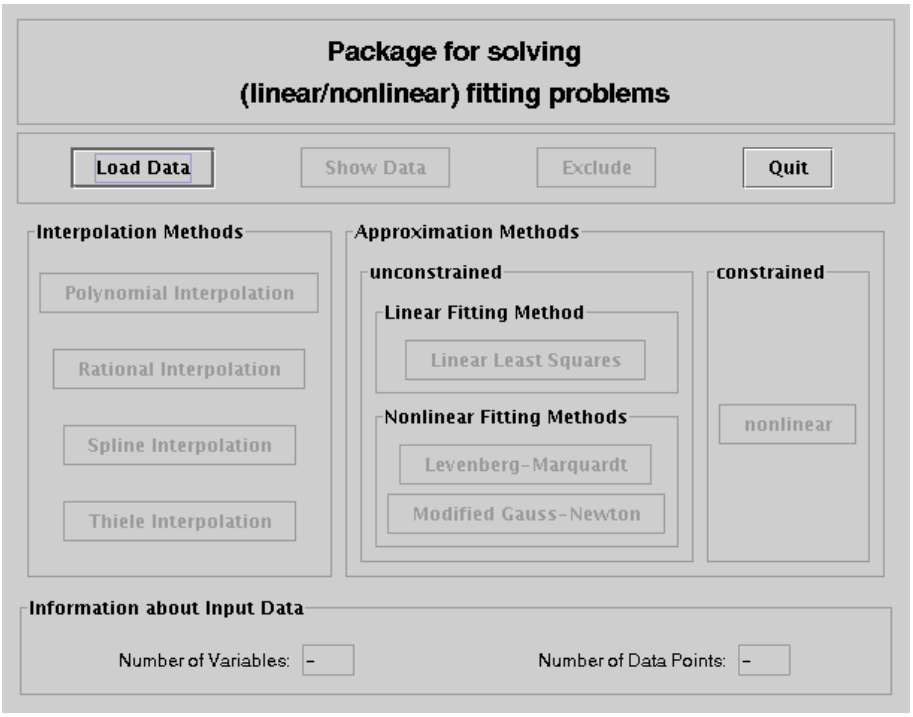

Abbildung 6.4: Die Startoberfläche des Programmpaketes

In der obersten Zeile des Fensters findet man die Operationen, die nicht in direktem Zusammenhang mit einem Fit durch Datenpunkte stehen. Hierzu gehört das *Laden* und *Anzeigen von Daten*, sowie das *Beenden* des Programmpaketes. Des Weiteren ist es möglich, über den Knopf *Exclude* ein weiteres Fenster zu starten, mit dem man einzelne Datenpunkte von der Datenanalyse ausschließen kann (siehe 6.2.2).

Im mittleren Teil des Fensters sind die mathematischen Verfahren aufgeführt, mit denen man die Daten aus den eingelesenen Dateien verarbeiten kann. Im linken Abschnitt dieses Bereiches befinden sich die Interpolationsmethoden:

1. Polynomiale Interpolation
2. Rationale Interpolation
3. Interpolation mit Splines
4. Thiele Interpolation

Der rechts liegende Abschnitt beinhaltet die Approximationsmethoden. Hierzu gehören

1. Lineare Ausgleichsrechnung ohne Nebenbedingungen
2. Nichtlineare Ausgleichsrechnung ohne Nebenbedingungen
   (a) Levenberg-Marquardt-Verfahren
   (b) Modifiziertes Gauß-Newton-Verfahren
3. Nichtlineare Ausgleichsrechnung unter Nebenbedingungen

Die nichtlineare Ausgleichsrechnung unter Nebenbedingungen bildet den Schwerpunkt dieser Arbeit. In Abschnitt 6.2.1 wird näher auf die Handhabung dieser Methode eingegangen.

Die letzte Zeile des Eingangspanels enthält Informationen (*Anzahl der Variablen* und *Anzahl der Datenpunkte*) über den eingelesenen Datensatz.

Vor der Nutzung der genannten Verfahren muss zunächst ein Datensatz aus einer Datei eingelesen werden. Welche Struktur eine solche Eingabedatei besitzen muss, wird in Anhang B beschrieben.

6.2.1 Restringierte nichtlineare Ausgleichsrechnung

Zur Demonstration des Verfahrens der nichtlinearen Ausgleichsrechnung unter Nebenbedingungen wird der Datensatz A.1 verwendet. Die Modellfunktion, die möglichst optimal an den Datensatz angenähert werden soll, hat die Gestalt:

$$M(x) \quad = \quad a_1 * (x_1 + a_2)^2 + a_3$$

Die Ergebnisse der Rechnung mit grafischer Darstellung der Ergebnisfunktion zeigt Abbildung 6.5.

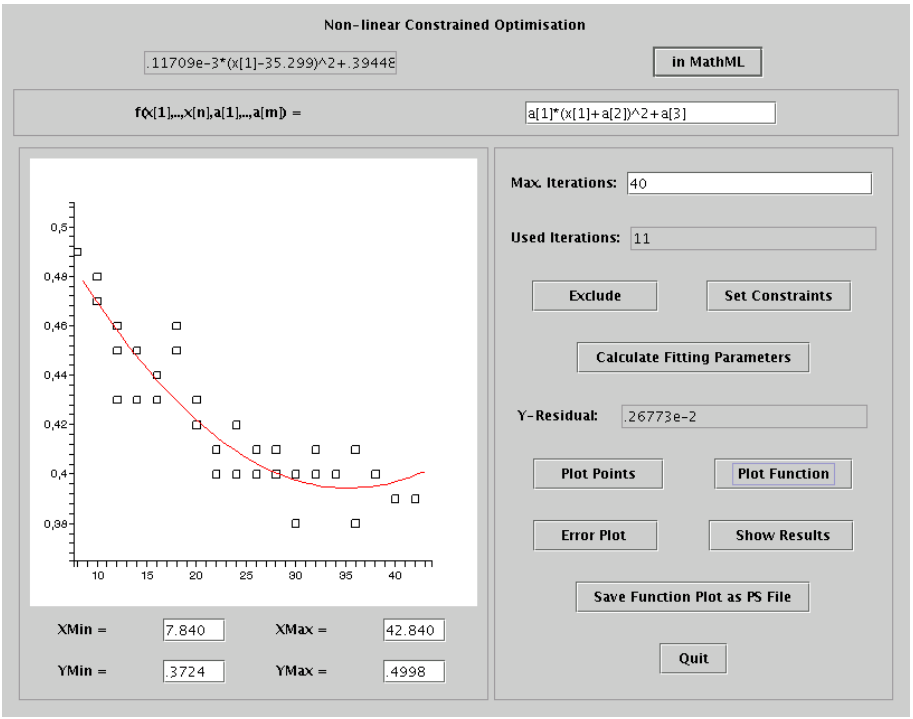

Abbildung 6.5: Nichtlineare Optimierung unter Nebenbedingungen

Um Ergebnisse wie in Abbildung 6.5 zu erhalten, müssen zunächst die gewünschten Nebenbedingungen sowie die Startwerte für die Parametersuche angegeben werden.

Festlegen der Nebenbedingungen

In dem vorliegenden Programmpaket gibt es drei Arten von Nebenbedingungen, die angegeben werden können:

1. begrenzende Nebenbedingungen (Intervalle)
2. lineare Nebenbedingungen
3. nichtlineare Nebenbedingungen

Indem man den Button *Set Constraints* in Abbildung 6.5 betätigt, erscheint ein weiteres Fenster (siehe Abbildung 6.6), in dem es möglich ist, die gewünschten Nebenbedingungen einzugeben.

In den Eingabefeldern auf der linken Seite können den Parametern Intervalle vorgegeben werden, in denen sich die Parameter nach der Approximation befinden sollen. Werden keine Begrenzungen in die entsprechenden Eingabefelder eingetragen, wird der Bereich automatisch auf $[-\infty, \infty]$ festgelegt. Im mittleren Block der Eingabemöglichkeiten können lineare Nebenbedingungen angegeben werden. Das bedeutet, dass die Parameter $a_1, \ldots, a_m$ in den gewünschten Ungleichungen nur linear

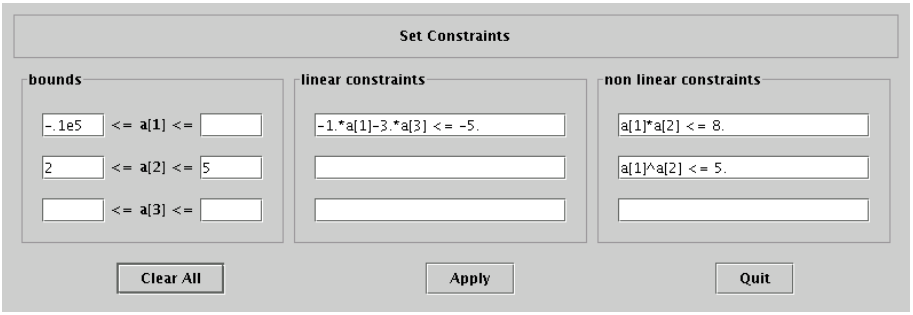

Abbildung 6.6: Panel zum Festlegen der Nebenbedingungen

auftreten dürfen. Sollten nichtlineare Nebenbedingungen vorliegen, also Ungleichungen in denen $a_1, \ldots, a_m$ nicht linear auftreten, können diese im rechten Eingabeblock definiert werden.

In der untersten Zeile des Eingabefensters befinden sich drei Steuerungsknöpfe. Mit *Clear All* werden alle Nebenbedingungen aus den Eingabefeldern gelöscht. Über den Knopf *Quit* kann man das Fenster wieder verlassen, und über den mittleren Knopf *Apply* werden die Nebenbedingungen aktiviert und damit im Optimierungsverfahren verwendet.

Angabe der Parameter

Um die Startwerte für das Optimierungsverfahren festzulegen, gibt es ebenfalls mehrere Möglichkeiten:

1. direkte Angabe des Startwertes
2. Angabe der Startwerte über ein Gitter

Sind dem Benutzer Näherungen für die Parameter bekannt, so können diese als Startwerte verwendet werden. Anderenfalls kann man ein Gitter angeben, in dem die Startwerte liegen sollen, und es wird der Startwert mit der geringsten Fehlerquadratsumme für die Durchführung des Optimierungsverfahrens genommen. Die Anzahl der Parameter in diesem Fenster ist gleich der Anzahl der Parameter in der Modellfunktion.

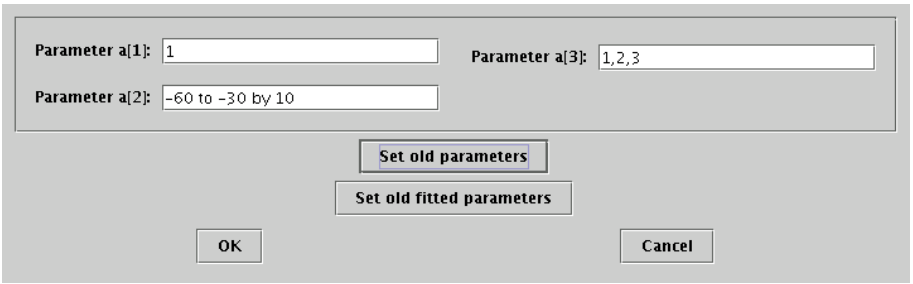

Abbildung 6.7: Die Eingabemaske für Parameter

6.2.2 Ausreißererkennung

Wie bereits erwähnt, ist in dem vorliegenden Programmpaket auch eine automatische Ausreißererkennung implementiert. Drei verschiedene Verfahren stehen dem Benutzer zur Verfügung:

1. Bestimmung über die Mahalanobis-Distanz
2. Bestimmung über die Hebelpunkte
3. Bestimmung über die studentisierten Residuen

Über die Mahalanobis-Distanz ist es möglich, a priori Ausreißer zu identifizieren und dann zu eliminieren. Betätigt man in Abbildung 6.5 den Button *Exclude*, erhält man das folgende Fenster.

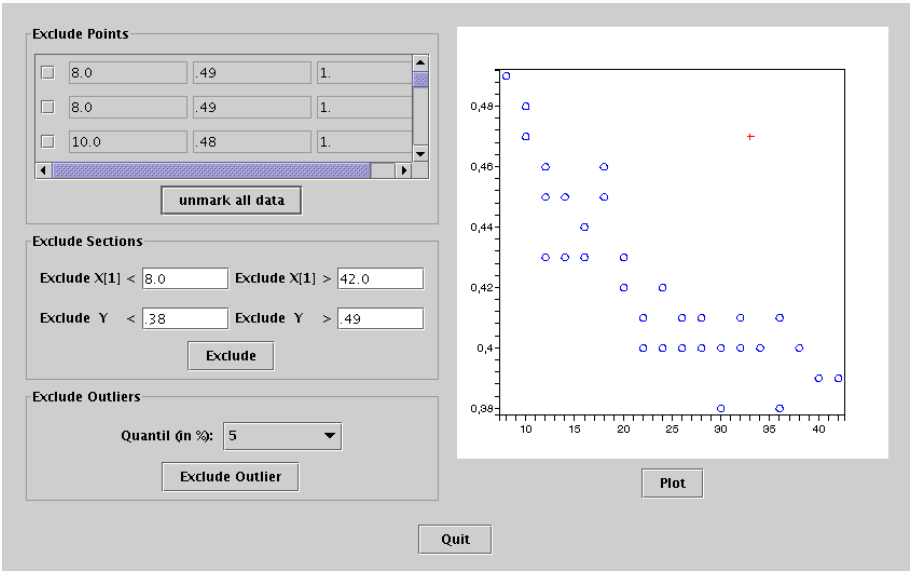

Abbildung 6.8: Ausreißererkennung vor einem Fit

Rechts im Fenster werden die Ergebnisse des Ausschlusses von Datenpunkten visualisiert. Auf der linken Seite befinden sich drei verschiedene Arten von Ausschlussverfahren von Datenpunkten. Im oberen Feld kann man einzelne Punkte von Hand aus dem Datensatz entfernen. Im mittleren Feld besteht die Möglichkeit, bestimmte Bereiche von Datenpunkten aus der Optimierungsaufgabe auszuschließen. Im untersten Feld kann man das Quantil angeben (1%- oder 5%-Quantil), mit dem man Ausreißer über die Mahalanobis-Distanz erkennen möchte.

Ausreißer a posteriori zu erkennen, bietet das Programmpaket in dem in Abbildung 6.5 abgebildeten Fenster. Ein Anklicken des Buttons *Exclude* öffnet hier das Fenster in Abbildung 6.9.

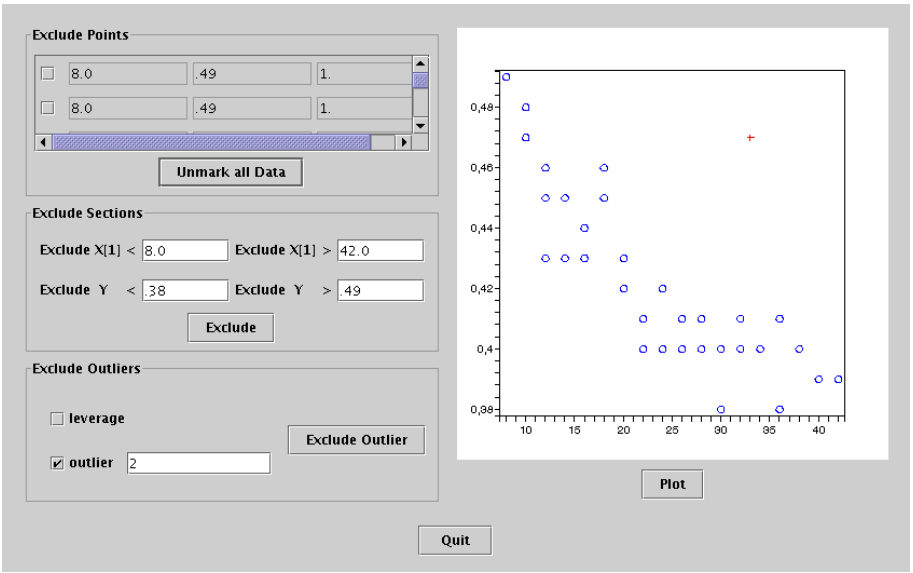

Abbildung 6.9: Ausreißererkennung nach einem Fit

Das Fenster ist äquivalent zu dem in Abbildung 6.8. Einziger Unterschied ist, dass man hier nicht versucht über die Mahalanobis-Distanz Ausreißer zu bestimmen, sondern hier kann man die *leverage-* und *outlier-Bestimmung* einzeln anwählen und ausführen lassen. Durch Ankreuzen der *leverage-Bestimmung* können wie in Abschnitt 3.3 beschrieben alle Hebelpunkte aus dem Datensatz entfernt werden. Über die *outlier-Bestimmung* werden die Datenpunkte mit einem studentisierten Residuum größer 2 aus dem Datensatz herausgenommen. Das Wiedereinbeziehen von bereits entfernten Ausreißern ist nur über die beiden gerade genannten Fenster möglich.

6.3 Verwendung im Worksheet

Die Routinen für die nichtlineare Ausgleichsrechnung unter Nebenbedingungen und für die Ausreißererkennung können nicht nur, wie im letzten Kapitel beschrieben, über die grafische Oberfläche genutzt werden, sondern auch direkt in einem Worksheet. Zur Nutzung der Routine zur Ausgleichsrechnung im Maple-System muss das Modul `Compute_SCO_Functions` eingebunden werden. Um die Ausreißererkennung zu nutzen, muss das Modul `NLF_Exclude_Functions` geladen werden. In diesen beiden Modulen befinden sich jeweils alle benötigten Routinen. Bei den nichtlinearen, restringierten Optimierungsaufgaben werden des Weiteren alle noch benötigten Untermodule eingebunden.

6.3.1 Restringierte nichtlineare Ausgleichsrechnung

Für die Ausgleichsrechnung steht nach dem Einbinden der benötigten Module die Funktion `ComputeKuhnTucker` zur Verfügung. Im Folgenden wird kurz darauf eingegangen, wie der Algorithmus von Maple aus aufgerufen werden kann.

```
ComputeKuhnTucker(  inXValues, inYValues, inF,
                    inParameterStart,
                    inBounds, inLinConst, inNonLinConst,
                    inMaxIterations,
                    outRes, outFittedParameters,
                    outUsedIterations  )
```

Abbildung 6.10: Aufruf-Syntax der Funktion `ComputeKuhnTucker`

Eine Beschreibung der formalen Parameter folgt in Tabelle 6.1.

Parameter	Bedeutung
`inXValues`	Liste der (mehrdimensionalen) X-Werte
`inYValues`	Liste der (eindimensionalen) Y-Werte
`inF`	Modellfunktion $M(x_1, \ldots, x_l, a_1, \ldots, a_p)$
`inParameterStart`	Liste mit den Startwerten für die Parameter $a_1, \ldots, a_p$
`inBounds`	Liste bestehend aus zwei weiteren Listen. Zum einen die Liste der unteren Grenzen und zum anderen die Liste der oberen Grenzen. $[[bl[1], .., bl[B]], [bu[1], .., bu[B]]]$ z.B.: $bl[k] <= inParameterStart[k] <= bu[k]$; Sollte ein Parameter nicht begrenzt sein, so müssen die Grenzen auf $infinity$ / $-infinity$ gesetzt werden.
`inLinConst`	Liste bestehend aus 4 Elementen. Zunächst die Anzahl der linearen Nebenbedingungen L und eine Liste bestehend aus L Listen, die jeweils die Koeffizienten der linearen Nebenbedingungen enthalten. Die beiden letzten Listen beinhalten, äquivalent zu `inBounds`, die unteren und oberen Schranken der Nebenbedingungen.
`inNonLinConst`	Liste bestehend aus 4 Elementen. Zunächst die Anzahl der nichtlinearen Nebenbedingungen N und eine Liste bestehend aus N nichtlinearen Ausdrücken, z.B.: $a_1 a_2$. Die beiden letzten Listen beinhalten, äquivalent zu `inBounds`, die unteren und oberen Schranken der Nebenbedingungen.
`inMaxIterations`	Anzahl der Iterationen, die maximal ausgeführt werden sollen
`outRes`	berechnetes Residuum
`outFittedParameters`	Vom Iterationsverfahren berechnete Parameter
`outUsedIterations`	Ausgeführte Iterationen bis zum Erreichen des Abbruchkriteriums

Tabelle 6.1: Bedeutung der formalen Parameter der Prozedur `ComputeKuhnTucker`

Wie die Prozedur `ComputeKuhnTucker` aus dem Modul `Compute_SCO_Functions` in der Praxis genutzt wird, zeigt folgender Maple-Code:

```
> with( Compute_ACO_Functions );
```

$$[ComputeKuhnTucker,\ Constraints,\ ReadConstraints,\ find\_a]$$

```
> # Model function
  Modelfunction := a[1] *(x[1]+a[2])^2 + a[3];
```

$$Modelfunction := a_1 \left(x_1 + a_2\right)^2 + a_3$$

```
> # List of x values
  XValues := [[ 8.0],[ 8.0],[10.0],[10.0],[10.0],[10.0],[12.0],
  [12.0],[12.0],[12.0],[14.0],[14.0],[14.0],[16.0],[16.0],[16.0],
  [18.0],[18.0],[20.0],[20.0],[20.0],[22.0],[22.0],[22.0],[24.0],
  [24.0],[24.0],[26.0],[26.0],[26.0],[28.0],[28.0],[30.0],[30.0],
  [30.0],[32.0],[32.0],[34.0],[36.0],[36.0],[38.0],[38.0],[40.0],
  [42.0] ]:

> # List of y values
  YValues := [ 0.49, 0.49, 0.48, 0.47, 0.48, 0.47, 0.46, 0.46,
  0.45, 0.43, 0.45, 0.43, 0.43, 0.44, 0.43, 0.43, 0.46, 0.45,
  0.42, 0.42, 0.43, 0.41, 0.41, 0.40, 0.42, 0.40, 0.40, 0.41,
  0.40, 0.41, 0.41, 0.40, 0.40, 0.40, 0.38, 0.41, 0.40, 0.40,
  0.41, 0.38, 0.40, 0.40, 0.39, 0.39 ]:
```

```
> # Initial values of parameters
  ParameterStart := [1,-40,1];
```

$$ParameterStart := [1, -40, 1]$$

```
> # Path to sourcecode
  PathToSource := "/home/buecker/Diplom/fitmaplet/sources";
```

$$PathToSource := \text{“/home/buecker/Diplom/fitmaplet/sources”}$$

```
> # Bounds
  Bounds := [[-infinity,-infinity,0],[infinity,infinity,0.39]];
  Linconst := [1,[0,1,1],[-infinity],[39]];
  NonLinConst := [1,[a[2]*a[3]],[-15],[-15]];
```

$$Bounds := [[-\infty, -\infty, 0], [\infty, \infty, 0.39]]$$

$$Linconst := [1, [0, 1, 1], [-\infty], [39]]$$

$$NonLinConst := [1, [a_2\, a_3], [-15], [-15]]$$

```
> # Number of iterations
  MaxIterations := 40;
```

$$MaxIterations := 40$$

```
> f := unapply(evalf(ComputeKuhnTucker(XValues, YValues,
      Modelfunction, ParameterStart, Bounds, Linconst,
      NonLinConst, MaxIterations, 'Residuum',
      'FittedParameters', 'UsedIterations',
      PathToSource), 5),x[1]);
```

$$f := x\_1 \rightarrow 0.000097100\,(x\_1 - 38.462)^2 + 0.39000$$

Um eine grafische Darstellung des Ergebnisses zu erhalten, muss folgende Funktion aufgerufen werden. Abbildung 6.11 zeigt den Output dieses Aufrufes.

```
> plots[display]( {plot(f(x),x=7..43)},
                  {plot( [seq( [XValues[i][1],YValues[i]],
                               i=1..44)],
                  style=point, symbol=circle )} );
```

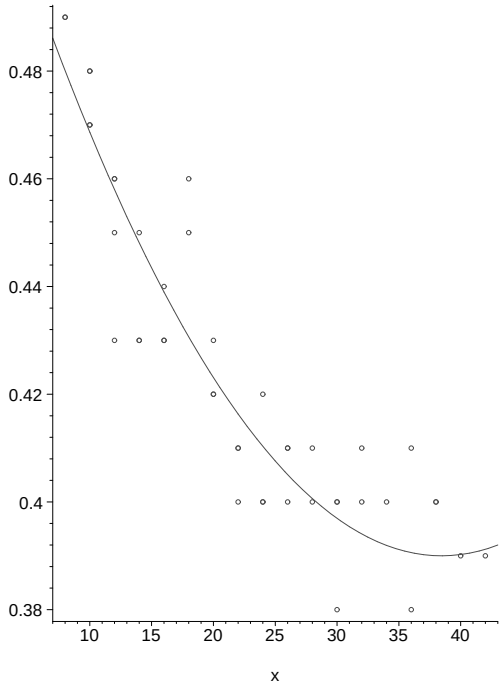

Abbildung 6.11: Grafische Darstellung der Datenpunkte und der optimierten Modellfunktion

6.3.2 Ausreißererkennung

Zur automatischen Bestimmung von Ausreißern in der gegebenen Datenmenge stehen nach dem Einbinden des Moduls `NLF_Exclude_Functions` die Funktion `ComputeExcludeMahalanobis` und das Untermodul `OriginalData` zur Verfügung. Beide werden zur Lösung des Problems benötigt. Die Hauptfunktion ist hierbei `ComputeExcludeMahalanobis`, deren Aufruf in Abbildung 6.12 angegeben ist.

```
ComputeExcludeMahalanobis ( quantil )
```

Abbildung 6.12: Aufruf-Syntax zur a priori Erkennung von Ausreißern innerhalb eines Maple-Worksheets

Einziges Argument, ist die echt positive Zahl *quantil*. Dieser Wert in Prozent gibt an, über welches Quantil die Ausreißer erkannt werden sollen. Die zu bearbeitenden Daten werden nicht direkt über diese Funktion, sondern über die Funktion `InitData` aus dem Modul `OriginalData` eingelesen. Diese besitzt folgende Struktur:

```
InitData ( XValues, YValues, Weight )
```

Abbildung 6.13: Aufruf-Syntax der `Initdata`-Funktion in einem Maple-Worksheet

Die einzelnen Parameter werden in Tabelle 6.2 beschrieben.

Parameter	Bedeutung
`XValues`	Liste der (mehrdimensionalen) X-Werte
`YValues`	Liste der (eindimensionalen) Y-Werte
`Weight`	Liste mit den Gewichtungen der einzelnen Punkte

Tabelle 6.2: Bedeutung der Parameter zur Initialisierungsfunktion

Nach der Bestimmung der Ausreißer kann über die Funktion `GetExcludeArr`, ebenfalls aus dem Modul `OriginalData`, abgefragt werden, ob ein bestimmter Punkt Ausreißer ist oder nicht.

```
GetExcludeArr ( Nr )
```

Abbildung 6.14: Aufruf-Syntax der `GetExcludeArr`-Funktion in einem Maple-Worksheet

Mit *Nr* wird die Nummer des Datenpunktes bezeichnet, der überprüft werden soll. Wie diese Ausreißerbestimmung in der Praxis funktioniert, ist im folgenden Maple-Code beschrieben.

```
> with( NLF_Exclude_Functions );
```

$$[ComputeExcludeMahalanobis,\ OriginalData]$$

```
> # List of x values
  XValues := [[ 8.0],[ 8.0],[10.0],[10.0],[10.0],[10.0],[12.0],
  [12.0],[12.0],[12.0],[14.0],[14.0],[14.0],[16.0],[16.0],[16.0],
  [18.0],[18.0],[20.0],[20.0],[20.0],[22.0],[22.0],[22.0],[24.0],
  [24.0],[24.0],[26.0],[26.0],[26.0],[28.0],[28.0],[30.0],[30.0],
  [30.0],[32.0],[32.0],[34.0],[36.0],[36.0],[38.0],[38.0],[40.0],
  [42.0],[33.0] ]:

> # List of y values
  YValues := [ 0.49, 0.49, 0.48, 0.47, 0.48, 0.47, 0.46, 0.46,
  0.45, 0.43, 0.45, 0.43, 0.43, 0.44, 0.43, 0.43, 0.46, 0.45,
  0.42, 0.42, 0.43, 0.41, 0.41, 0.40, 0.42, 0.40, 0.40, 0.41,
  0.40, 0.41, 0.41, 0.40, 0.40, 0.40, 0.38, 0.41, 0.40, 0.40,
  0.41, 0.38, 0.40, 0.40, 0.39, 0.39, 0.47 ]:
```

```
> # List of weights
  Weights := [ seq(1,i=1..44) ];
```

$$Weights := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

```
> # Initialise Data
  OriginalData:-InitData(XValues, YValues, Weights):
> # Exclude Outliers by the Mahalanobismethod
  ComputeExcludeMahalanobis(5);
```

Die grafische Auswertung der Ausreißererkennung kann über die folgenden Funktionen dargestellt werden.

```
> Decide := (i,fatr::boolean) ->
              if OriginalData:-GetExcludeArr(i)=fatr then
                [OriginalData:-GetDataX(i,1),
                 OriginalData:-GetDataY(i)];
              end if:
  DsplData := plot( {seq(Decide(i,false),
                   i=1..OriginalData:-GetQuantity() ) },
                  style=point, symbol=circle, color=blue ):
  DsplExcl := plot( {seq(Decide(i,true),
                   i=1..OriginalData:-GetQuantity() ) },
                  style=point, symbol=cross, color=red ):
  plots[display](DsplData, DsplExcl);
```

Das Ergebnis ist in folgender Abbildung zu erkennen.

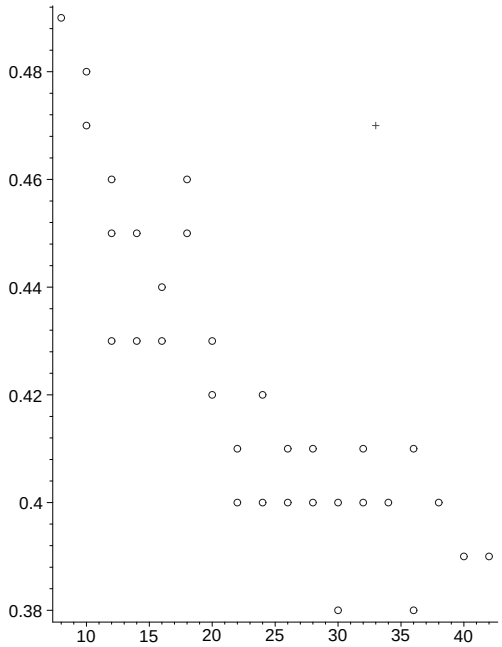

Abbildung 6.15: Grafische Darstellung der Datenpunkte inklusive des Ausreißers

Kapitel 7

Vergleich mit anderen Programmpaketen

Neben dem hier vorgestellten Programmpaket gibt es andere mathematische Softwaresysteme, die ebenfalls Tools bereitstellen, mit denen man vergleichbare Problemstellungen lösen kann. Stellvertretend werden hier die Prozedur `NLIN` aus dem Statistikpaket *SAS - Statistical Analysis System* [12] und die *Curve Fitting Toolbox* in *MATLAB* [10] mit dem vorliegenden Programmpaket verglichen. Für diesen Vergleich wird der Datensatz A.2 mit der Modellfunktion

$$M(x) \quad = \quad a_1 + (a_2 - a_1) \cdot e^{-\left(\frac{x-a_4}{a_6}\right)^4} + (a_3 - a_2) \cdot e^{-\left(\frac{x-a_4}{a_5}\right)^4}$$

herangezogen. Die Startwerte der Parameter a_1 bis a_6 lauten:

Parameter	Startwert
a_1	3.465
a_2	208.150
a_3	-547.340
a_4	1.782
a_5	1.644
a_6	6.012

Tabelle 7.1: Startwerte der Parameter für den Vergleich der Programmpakete

Als Nebenbedingungen sollen folgende zwei Ungleichungen erfüllt sein:

$$\begin{aligned} a_3 &< -570 \\ a_1 &< 6 \end{aligned}$$

7.1 Die `NLIN`-Prozedur aus dem *SAS*-Paket

Die von *SAS* angebotene `NLIN`-Prozedur hat für *SAS*-Benutzer eine sehr intuitive Syntax. Über Schlüsselworte, wie *MODEL, PARMS, BOUNDS, ...*, kann die gewünschte Aufgabenstellung der Prozedur übergeben werden. Im Interactive-Mode kann die Prozedur, wie Abbildung 7.1 zeigt, über den *Program Editor* eingegeben und mit der Funktionstaste `F3` ausgeführt werden.

```
proc nlin method=marquardt;
parms a1=3.465 a2=208.15 a3=-547.34
      a4=1.782 a5=1.644 a6=6.012;
model y = a1+(a2-a1)*exp(-((x-a4)/a6)**4)+
            (a3-a2)*exp(-((x-a4)/a5)**4);
bounds a3<-570;
bounds a1<6;
run;
```

Abbildung 7.1: Beispiel für die Handhabung der NLIN-Prozedur in *SAS*

Nachdem die Prozedur erfolgreich ausgeführt worden ist, erscheint im *Output*-Fenster folgendes Ergebnis:

The NLIN Procedure
Dependent Variable y
Method: Marquardt

Iterative Phase

Iter	a1	a2	a3	a4	a5	a6	Sum of Squares
0	3.4650	208.2	-570.0	1.7820	1.6440	6.0120	137.3
1	3.4650	208.2	-570.0	1.7820	1.6440	6.0120	137.3
2	5.1897	208.2	-570.0	1.7765	1.6444	6.0040	126.4
3	5.4564	208.3	-570.0	1.7700	1.6497	6.0051	126.2
4	5.2566	208.4	-570.0	1.7576	1.6594	6.0193	126.2
5	5.1713	208.5	-570.0	1.7526	1.6634	6.0251	126.2
6	5.1559	208.5	-570.0	1.7516	1.6641	6.0262	126.2
7	5.1535	208.5	-570.0	1.7515	1.6642	6.0264	126.2
8	5.1531	208.5	-570.0	1.7514	1.6643	6.0264	126.2

NOTE: Convergence criterion met.

Die Option `method` von NLIN zeigt einen der wesentlichen Vorteile zum Programmpaket dieser Arbeit. Es ist möglich, verschiedene Verfahren zur Lösung des Problems zu nutzen. Das sind:

1. Methode des steilsten Abstiegs (`gradient`)
2. Newton-Methode (`newton`)
3. Modifizierte Gauß-Newton-Methode (`gauss`)
4. Marquardt-Methode (`marquardt`)
5. Falsche Positions Methode (`dud`)

Im Programmpaket NLFit, dass in dieser Arbeit beschrieben wird, gibt es nur Auswahlmöglichkeiten für nicht restringierte Optimierungsprobleme. Für die restringierte Optimierung steht dem Benutzer bisher nur ein Verfahren zur Verfügung. Ein Nachteil von NLIN zu NLFit ist, dass keine Oberfläche zur Verfügung steht. Für Benutzer, die keine oder nur wenig Erfahrung mit dem *SAS*-System haben, ist es schwierig dieses zu handhaben. Ein weiterer wesentlicher Nachteil bildet die Tatsache, dass nur begrenzende Nebenbedingungen mit in die Problemstellung aufgenommen werden können. In NLFit ist es möglich, neben den begrenzenden Nebenbedingungen auch noch lineare und nichtlineare Nebenbedingungen mit in die Betrachtung einzubeziehen.

7.2 Die *Curve Fitting Toolbox* zu *MATLAB*

Die Firma *MathWorks* bietet für das von ihnen vertriebene Softwaresystem *MATLAB* unter anderem die *Curve Fitting Toolbox* an. Mit dieser Toolbox können Benutzer über eine grafische Oberfläche alle Funktionen nutzen, die *MATLAB* zur Anpassung von Daten zur Verfügung stellt. Die Toolbox stellt Funktionen für die Aufbereitung von Daten, sowie das Erstellen, Vergleichen und Analysieren von Ausgleichsproblemen zur Verfügung.

Die Oberfläche wird über das Kommando `cftool` gestartet. Zunächst müssen über den Knopf `Data...` Daten, die in *MATLAB* selber bereits bekannt sein müssen, geladen werden. Dies ist im Vergleich zu `NLFit` ein Nachteil, da es in der *Curve Fitting Toolbox* nicht möglich ist, Daten direkt aus einer Datei zu lesen.

Über den Knopf `Fitting...` öffnet sich ein Fenster, in dem man eine Optimierungsaufgabe eingeben und lösen kann.

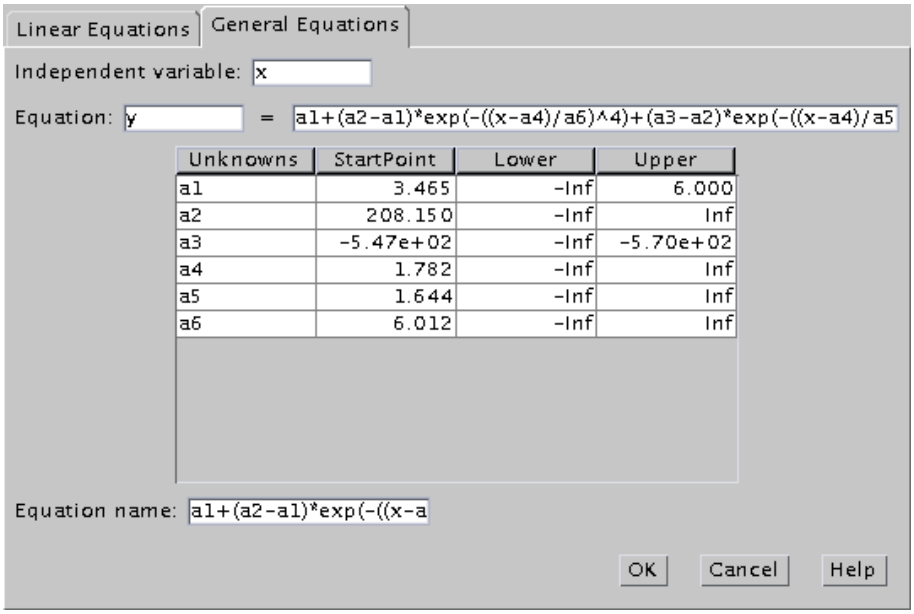

Abbildung 7.2: Eingabemaske für Modellfunktion und Parameter in der *Curve Fitting Toolbox* von *MATLAB*

Startet man die Optimierungsaufgabe mit den in Abbildung 7.2 angegebenen Grenzen, erhält man als Approximationsfunktion $M(x) = 6 - 855 * e^{-(x-1)^4}$, was aufgrund der Summe der Fehlerquadrate von 626068.687 als Lösung ausgeschlossen werden kann. Nimmt man nun die Begrenzung $a3 < -570$ weg, erhält man das folgende, korrekte Ergebnis mit einer Summe der Fehlerquadrate von 126.18889.

```
General model:
  fittedmodel1(x) = a1+(a2-a1)*exp(-((x-a4)/a6)^4)+(a3-a2)*exp(-((x-a4)/a5)^4)
Coefficients (with 95% confidence bounds):
  a1 =       5.258  (-6.254, 16.77)
  a2 =       208.5  (202.9, 214)
  a3 =      -562.4  (-1123, -1.809)
  a4 =       1.756  (1.224, 2.289)
  a5 =       1.662  (1.315, 2.009)
  a6 =        6.02  (5.371, 6.67)
```

Die *Curve Fitting Toolbox* hat zur `NLIN`-Prozedur von *SAS* den Vorteil der Benutzeroberfläche. Somit ist der Zugang zu den Funktionen für unerfahrene Nutzer leichter. Ein Nachteil dieser Toolbox ist, dass die zu verarbeitenden Daten vor dem Start der Toolbox in *MATLAB* definiert sein müssen. Hierfür muss sich der Nutzer mit dem *MATLAB*-System auskennen.

Wesentliche Nachteile zu `NLFit` sind zum einen, dass die Begrenzung von Parametern zwar möglich

ist, aber weder lineare noch nichtlineare Nebenbedingungen mit in die Berechnung einfließen können. Zum anderen können auch keine mehrdimensionalen Ausgleichsprobleme mit der *Curve Fitting Toolbox* gelöst werden.

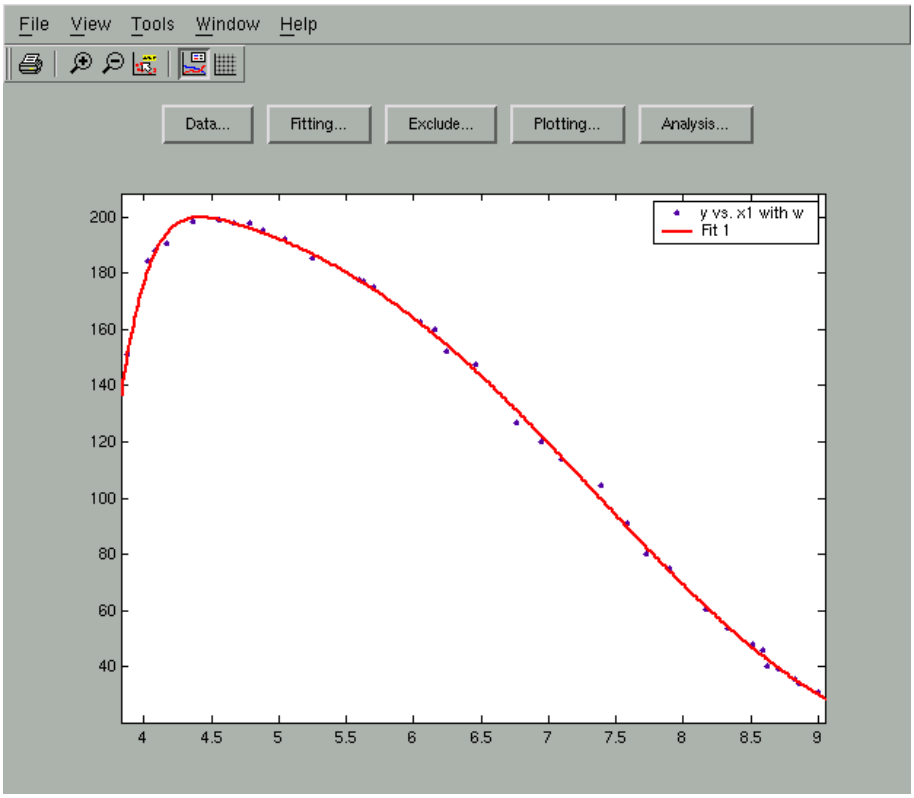

Abbildung 7.3: Grafische Ergebnisdarstellung in der *Curve Fitting Toolbox* von *MATLAB*

Anschließend kann man sich über den Knopf `Analysis...` Auswertungen bezüglich des erzeugten Fits anzeigen lassen. Hierzu gehören

1. die Bestimmung der Approximationsfunktion an gewünschten Punkten,
2. das Auswerten der ersten und zweiten Ableitung an bestimmten Punkten und
3. die Berechnung des Integrals über einen vorgegebenen Bereich.

Diese Auswertungsmöglichkeiten sind in `NLFit` noch nicht implementiert worden.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit wurden Algorithmen zur Lösung von restringierten nichtlinearen Ausgleichsproblemen und zur automatischen Ausreißererkennung sowie deren Implementierung im integrierten mathematischen Softwaresystem Maple vorgestellt.

Für diese Algorithmen werden Ableitungen der Modellfunktion benötigt. Deshalb wurden Methoden zur exakten Differentiation vorgestellt. Im Programmpaket zu dieser Arbeit wird die symbolische Differentiation angewandt.

Es wurden die wichtigsten Programmiertechniken vorgestellt, die für die Erstellung des Programmpaketes benötigt wurden. Zentrale Punkte waren hierbei das Erstellen von externen Routinen und deren Einbindung in das Maple-System sowie die Erstellung von grafischen Oberflächen, den Maplets, in Maple.

Um einen Überblick über das erzeugte Programmpaket zu bekommen, wurde dessen prinzipeller Einsatz sowohl unter der grafischen Benutzeroberfläche als auch im Worksheet aufgezeigt. Die Funktionalität wurde an einem exemplarischen Beispiel erläutert.

Auch in anderen mathematischen Softwaresystemen gibt es Programme, die ähnliche Aufgabenstellungen bearbeiten können. Um einen Vergleich zwischen diesen Tools und dem hier präsentierten Programmpaket zu bekommen, wurden abschließend deren Vor- und Nachteile aufgezeigt.

Das Tool befindet sich noch in der Entwicklungsphase und ist somit noch nicht ausgereift. In Zukunft sollten noch folgende Punkte implementiert werden:

1. Hinzufügen weiterer Algorithmen, vor allem im Bereich der nichtlinearen Optimierung unter Nebenbedingungen,
2. Anpassung von Kurven und Oberflächen mit kubischen bzw. bikubischen Splines,
3. Auswählen von Ausreißern durch Anklicken der entsprechenden Punkte im Plot,
4. Vereinfachung der Eingabemöglichkeit von Nebenbedingungen,
5. Erweiterung grafischer Ausgabemöglichkeiten, wie z.B. die Markierung der Bereiche, in denen bei der nichtlinearen Optimierung unter Nebenbedingungen das Optimum gesucht wird,
6. Implementierung von aufwändigen, in Maple geschriebenen Programmteilen, in einer anderen Programmiersprache.

Der erste Punkt dieser Auflistung zielt speziell auf das Implementieren von Open-Source-Routinen. Derzeit basiert die restringierte nichtlineare Optimierung auf einem Programm der NAGfl90-Bibliothek, die lizensiert werden muss.

Beim sechsten Punkt sind insbesondere zeitintensive Maple-Routinen gemeint, wie sie z.B. derzeit für das Bestimmen von optimalen Startwerten verwendet werden. Eine Implementierung solcher rekursiver Funktionen in externen Programmiersprachen kann eine erhebliche Effizienzerhöhung mit sich bringen.

Alle weiteren Punkte ergeben sich durch den Vergleich der *MATLAB Curve Fitting Toolbox* und der *SAS* `NLIN`-Prozedur in Kapitel 7. Aber auch andere Maplet-Anwendungen in Maple sind Ideengeber für die Weiterentwicklung dieses Tools.

Eine Portierung des Programmpaketes auf andere Betriebssysteme wie z.B. Windows ist wünschenswert. Hierzu muss nur das Modul `NonLinearFitLinux` portiert werden, da alle anderen Module betriebssystemunabhängig sind. Ebenso sollte die Portierung auf zukünftige Maple-Versionen berücksichtigt werden.

Anhang A

Verwendete Datensätze

Nr.	x_1	y	ω	Nr.	x_1	y	ω
1	8.0	0.49	1	23	22.0	0.41	1
2	8.0	0.49	1	24	22.0	0.40	1
3	10.0	0.48	1	25	24.0	0.42	1
4	10.0	0.47	1	26	24.0	0.40	1
5	10.0	0.48	1	27	24.0	0.40	1
6	10.0	0.47	1	28	26.0	0.41	1
7	12.0	0.46	1	29	26.0	0.40	1
8	12.0	0.46	1	30	26.0	0.41	1
9	12.0	0.45	1	31	28.0	0.41	1
10	12.0	0.43	1	32	28.0	0.40	1
11	14.0	0.45	1	33	30.0	0.40	1
12	14.0	0.43	1	34	30.0	0.40	1
13	14.0	0.43	1	35	30.0	0.38	1
14	16.0	0.44	1	36	32.0	0.41	1
15	16.0	0.43	1	37	32.0	0.40	1
16	16.0	0.43	1	38	34.0	0.40	1
17	18.0	0.46	1	39	36.0	0.41	1
18	18.0	0.45	1	40	36.0	0.38	1
19	20.0	0.42	1	41	38.0	0.40	1
20	20.0	0.42	1	42	38.0	0.40	1
21	20.0	0.43	1	43	40.0	0.39	1
22	22.0	0.41	1	44	42.0	0.39	1

Tabelle A.1: Beispieldatensatz aus der NAG-Dokumentation

Nr.	x_1	y	ω	Nr.	x_1	y	ω
1	3.88	151.0	1	18	6.46	147.5	1
2	4.03	184.0	1	19	6.76	126.5	1
3	4.08	188.0	1	20	6.95	120.0	1
4	4.17	190.5	1	21	7.10	114.0	1
5	4.36	198.0	1	22	7.39	104.5	1
6	4.56	198.5	1	23	7.59	91.0	1
7	4.67	197.8	1	24	7.73	80.0	1
8	4.79	197.5	1	25	7.90	75.0	1
9	4.88	195.0	1	26	8.17	60.5	1
10	5.04	192.0	1	27	8.33	53.5	1
11	5.25	185.5	1	28	8.52	48.0	1
12	5.60	177.5	1	29	8.59	46.0	1
13	5.63	177.0	1	30	8.62	40.0	1
14	5.71	175.0	1	31	8.71	39.0	1
15	6.05	162.5	1	32	8.83	35.5	1
16	6.16	160.0	1	33	8.86	34.0	1
17	6.24	152.2	1	34	9.00	31.0	1

Tabelle A.2: Beispieldatensatz aus der SAS-Dokumentation

Anhang B

Struktur der Eingabedateien

Damit Eingabedaten korrekt eingelesen und somit auch verarbeitet werden können, müssen diese eine bestimmte Stuktur besitzen. Der Aufbau einer Eingabedatei sollte mit dem in Abbildung B.1 vorgestellten Format übereinstimmen.

```
l
x_11  x_12   ...   x_1l  Y_1  W_1
x_21  x_22   ...   x_2l  Y_2  W_2
 ⋮     ⋮     ⋱     ⋮     ⋮    ⋮
x_n1  x_n2   ...   x_nl  Y_n  W_n
```

Abbildung B.1: Dateiformat einer Eingabedatei

In der ersten Zeile der Eingabedatei steht die Anzahl `l` der unabhängigen Variablen. Die darauffolgenden `n` Zeilen beinhalten jeweils zunächst die Koordinaten des `l`-dimensionalen x-Vektors. Hinter dem x-Vektor folgt die zugehörige y-Koordinate sowie die Gewichtung ω dieses Datenpunktes.

Anhang C

Abkürzungen und Symbole

$\boldsymbol{a}$	Vektor der Parameter
p	Dimension des Vektors $\boldsymbol{a}$
$(\boldsymbol{x_i}, y_i)$	der i-te Datenpunkt
n	Anzahl der Datenpunkte
l	Anzahl der unabhängigen Variablen (Dimension des gesamten Datenpunktes -1)
M	Modellfunktion
f	die zu minimierende Funktion
m	Anzahl der Gleichungsnebenbedingungen
k	Anzahl der Ungleichungsnebenbedingungen
B	Anzahl der begrenzenden Nebenbedingungen
L	Anzahl der linearen Nebenbedingungen
N	Anzahl der nichtlinearen Nebenbedingungen

Literaturverzeichnis

[1] J. E. Dennis Jr., R. B. Schnabel (1983)
Numerical Methods for Unconstrained Optimization and Nonlinear Equations
Prentice-Hall

[2] J. E. Dennis Jr., J. J. Moré (1977)
Quasi-Newton methods, motivation and theory
SIAM Rev. 19 46-89

[3] J. E. Dennis Jr., R. B. Schnabel (1981)
A new derivation of symmetric positive-definite secant updates
Nonlinear Programming 4 (ed O. L. Mangasarian, R. R. Meyer and S. M. Robinson)
Academic Press 167-199

[4] R. Fletcher (1981)
Practical Methods of Optimization - Volume 2
John Wiley & Sons (ISBN 0-471-27828-9)

[5] P. E. Gill , W. Murray, M. H. Wright (1981)
Some theoretical properties of an augmented Lagrangian merit function
Report SOL 86-6R Department of Operation Research, Stanford University

[6] P. E. Gill, W. Murray, M. A. Saunders, M. H. Wright (1998)
User's Guide For NPSOL (Version 5.0)
Report NA 98-2, Department of Mathematics, University of California, San Diego
`ftp://www.scicomp.ucsd.edu/pub/peg/reports/npdoc.ps`

[7] A. Quateroni, R. Sacco, F. Saleri (2000)
Numerical Mathematics
Springer Verlag (ISBN 0-387-98959-5)

[8] M. J. D. Powell (1974)
Introduction to constrained optimization *Numerical Methods for Constrained Optimization*
Academic Press 1-28

[9] H. G. Bock (2000)
Algorithmische Optimierung I
(Vorlesungsskript)

[10] *Dokumentation der MATLAB Curve Fitting Toolbox*
`http://www.mathworks.com/products/curvefitting/`

[11] *Online-Dokumentation der NAGfl90 Library*
`http://www.nag.com/numeric/fn/fndescription.asp`

[12] SAS Institute Inc. (1989)
SAS/STAT User's Guide, Version 6, Fourth Edition, Volume 2
(ISBN 1-55544-376-1)

[13] Dr. C. Kredler (2003)
Lineare Modelle mit Anwendungen
(Vorlesungsskript)
`http://www-m1.ma.tum.de/nbu/linmod/`

[14] Norbert Henze (2000)
Stochastik für Einsteiger - Eine Einführung in die faszinierende Welt des Zufalls
Vieweg (ISBN 3-528-26894-8)

[15] Johannes Grotendorst (2003)
ComputerMathematik mit Maple
(ISBN 3-89336-325-4)

[16] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn,
S. M. Vorkoetter, J. McCarron, P. DeMarco (2003)
Maple 9 - Introductory Programming Guide
(ISBN 1-894511-43-3)

[17] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn,
S. M. Vorkoetter, J. McCarron, P. DeMarco (2003)
Maple 9 - Advanced Programming Guide
(ISBN 1-894511-44-1)

[18] Bronstein, Semendjajew, Musiol, Mühlig (1999)
Taschenbuch der Mathematik
Verlag Harri Deutsch AG Thun (ISBN 3-8171-2014-1)

[19] René Külheim (2003)
Lösung nichtlinearer Ausgleichsprobleme
mit symbolisch-numerischen Rechenverfahren in Maple
Interner Bericht
(FZJ-ZAM-IB-2003-07)