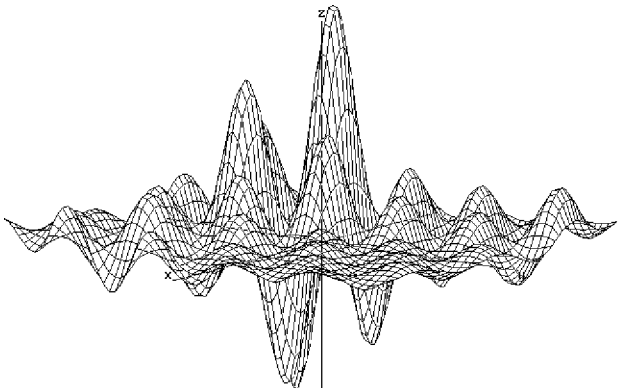
Automatische Ergebnisverifikation bei globalen Optimierungsproblemen

Dissertation

von

Dietmar Ratz



Karlsruhe 1992

Automatische Ergebnisverifikation bei globalen Optimierungsproblemen

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Mathematik der
Universität Karlsruhe (TH)

genehmigte

Dissertation

von

Dipl.-Math. techn. Dietmar Ratz

aus Karlsruhe

Tag der mündlichen Prüfung:	1. Juli 1992
Referent:	Prof. Dr. U. Kulisch
Korreferent:	Prof. Dr. E. Kaucher

Für

Claudia, Sebastian und Miriam

Vorwort

Die vorliegende Arbeit[1] enstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik der Universität Karlsruhe (TH). An dieser Stelle möchte ich deshalb ganz besonders meinem Referenten, Herrn Prof. Dr. U. Kulisch danken, der mir die Möglichkeit gegeben hat, an seinem Institut zu arbeiten und diese Arbeit anzufertigen. Ein herzlicher Dank geht auch an Herrn Prof. Dr. E. Kaucher für die Übernahme des Korreferats sowie für sein großes Interesse und seine ständige Bereitschaft zur Diskussion während der Entstehung der Arbeit.

Mein Dank gilt auch den Mitarbeiterinnen und Mitarbeitern des Instituts für Angewandte Mathematik, die durch ihre langjährige gute Zusammenarbeit und ihre Hilfsbereitschaft ihren Teil zur Entstehung dieser Arbeit beitrugen. Dies gilt insbesondere für Herrn Dr. R. Klatte, der mich auf die „Spur“ der globalen Optimierung ansetzte und so manchen guten Ratschlag zur rechten Zeit parat hatte, sowie für die Herren Dipl.-Math. S. Geörg, Dipl.-Math. techn. R. Hammer, Dipl.-Math. oec. M. Hocks und Dr. G. Schumacher, deren ständige Gesprächs- und Diskussionsbereitschaft das Gelingen der Arbeit förderten.

Für das Korrekturlesen des Manuskripts danke ich Herrn Dr. P. Jany, der – trotz seiner angeblichen „grenzenlosen mathematischen Ignoranz“ – wertvolle Verbesserungsvorschläge einbrachte.

Ein besonders herzliches Dankeschön geht an meine Frau Claudia, die mit viel Geduld und Einfühlungsvermögen dafür sorgte, daß unsere geliebten „Nervensägen“ Sebastian und Miriam an meinen arbeitsreichen Abenden und Wochenenden einen gewissen Lärmpegel nicht gar zu sehr überschritten haben.

Schließlich möchte ich mich auch bei meinen Eltern für ihre stete Unterstützung auf meinem Lebens- und Bildungsweg recht herzlich bedanken.

[1] Die Graphik auf dem Einband zeigt die Testfunktion W5 aus Kapitel 4 (Seite 144).

Inhaltsverzeichnis

Algorithmenverzeichnis

Abbildungsverzeichnis

Einleitung

Viele Probleme aus dem Bereich technisch-wissenschaftlicher Anwendungen können als globale nichtlineare Optimierungsprobleme formuliert werden. Im Gegensatz zur lokalen Optimierung, bei der eine optimale Lösung in der Nähe eines vorgegebenen Punktes gesucht wird, verlangt die globale Optimierung das Auffinden des „besten“ lokalen Optimums. Während das Gebiet der lokalen nichtlinearen Optimierung bereits seit vielen Jahren erforscht wird und entsprechend zahlreiche Theorien, numerische Verfahren und Veröffentlichungen vorliegen, ist das Gebiet der globalen Optimierung bisher noch wenig untersucht worden. Wie wichtig eine zukünftige intensive Forschung auf diesem Gebiet sein wird unterstreicht die Tatsache, daß die meisten sogenannten *real-world*-Probleme von globalem und nicht etwa lokalem Charakter sind (vgl. z. B. [49] und [57]). Im allgemeinen heißt das, daß eine große Zahl lokaler Minima existiert, für die ein globales Optimierungsverfahren zwar auf bekannte lokale Verfahren zurückgreifen kann, daß aber zusätzlich globale Information benötigt wird. Ein Verfahren zur globalen Optimierung konvergiert nämlich nur dann *garantiert* gegen das globale Minimum, wenn die Iterierten das gesamte Optimierungsgebiet vollständig abdecken und somit dicht darin liegen (vgl. Satz 1.3.5 in Abschnitt 1.3).

Ein exzellentes Hilfsmittel zur Ermittlung globaler Informationen über Teilbereiche des Optimierungsgebietes ist die *Intervallrechnung*, denn Intervalle repräsentieren i. a. unendlich viele Punkte, nämlich alle Punkte, die auf und zwischen den beiden Intervallgrenzen liegen. Unter Verwendung einer einzigen intervallarithmetischen Funktionsauswertung können beispielsweise Aussagen über die Lage der Funktionswerte für alle Punkte innerhalb eines Intervalls gemacht werden. Dabei sind alle bei der praktischen Durchführung auf dem Rechner auftretenden Rundungsfehler mit erfaßbar, so daß garantierte Fehlerschranken automatisch mitgeliefert werden. Intervallverfahren sind somit in der Lage, Einschließungen für *alle* Lösungen eines Problems, z. B. eines globalen Optimierungsproblems, zu berechnen.

Dabei kann prinzipiell eine beliebige Genauigkeit erreicht und nicht nur – wie von vielen „Intervall-Laien“ fälschlich angenommen – ein unrealistisch großes Intervall berechnet werden. Die Grundlage für die Durchführung solcher *Einschließungs-* oder auch *Verifikationsverfahren* auf einem Rechner bilden die Arbeiten über eine mathematisch formulierte Rechnerarithmetik in den üblichen reellen Räumen und Intervallräumen des numerischen Rechnens von Kulisch und Miranker ([31], [33], [35]). Implementierungen solcher Arithmetiken wurden im Rahmen von Spracherweiterungen von FORTRAN, PASCAL und C am Institut für Angewandte Mathematik der Universität Karlsruhe entwickelt.

Diese Arbeit beschäftigt sich mit einem auf erweiterter Intervallarithmetik beruhenden Verifikationsverfahren zur globalen Optimierung ohne Nebenbedingungen, dessen grundlegende Form auf den von Hansen [13] entwickelten Algorithmus bzw. auf die verwandten Algorithmen von Moore [38] und Skelboe [55] zurückgeht. Es werden zahlreiche Modifikationen sowohl des grundlegenden Algorithmus als auch der effizienzsteigernden, zusätzlich angewendeten Methoden entwickelt. Schwerpunktmäßig wird dabei der in erweiterter Intervallarithmetik auszuführende Intervall-Newton-Schritt bzw. Intervall-Gauß-Seidel-Schritt behandelt, für den unter Verwendung spezieller Aufteilungstechniken und Präkonditionierer je nach Anwendungsfall deutliche Verbesserungen erzielt werden konnten. Es wird auch eine spezielle Methode zur Randbehandlung entwickelt, was in der Literatur bisher vernachlässigt wurde. Neben einer hohen, steuerbaren Genauigkeit für die *garantierten* Einschließungen des Minimums und der globalen Minimalstellen, ermöglicht das Verfahren eine *automatische Verifikation* der Eindeutigkeit der Lösung innerhalb der berechneten Schranken. Dazu wird eine Methode angegeben, die es ermöglicht, diesen Nachweis auf dem Rechner im Rahmen der numerischen Durchführung des Verfahrens zu erbringen. In seiner Gesamtheit bietet das in dieser Arbeit entwickelte Verfahren eine große Palette an Verfahrensvarianten, die abhängig vom jeweils behandelten Optimierungsproblem ausgewählt und kombiniert werden können. Dies schafft die Voraussetzung für die Lösung zahlreicher Standardprobleme, die mit anderen Verfahren nicht, nur mit einem vielfach höheren Aufwand oder nicht verifiziert gelöst werden konnten. Es werden dabei stets verifizierte, hochgenaue Einschließungen für *alle* globalen Minimalstellen und für den Wert des globalen Minimums berechnet. Dies gilt sowohl für den Fall einer großen Zahl lokaler Minima als auch für den Fall mehrerer globaler Minimalstellen, wobei sehr dicht beieinanderliegende globale Minimalstellen nur dann in verschiedene Ergebnisquader eingeschlossen werden können, wenn das Datenformat auf dem Rechner genügend Mantissenstellen zur Verfügung

stellt. In den meisten Fällen ist die Effizienz des Verfahrens vergleichbar mit der von klassischen Verfahren, die jedoch in der Regel keinerlei Fehlerschranken mitliefern.

Die Darstellung der Algorithmen in dieser Arbeit erfolgt stets unter besonderer Beachtung einer einfachen Implementierung auf dem Rechner, um die Verfahren auch in der Praxis anwendbar zu machen. Diesem praktischen Aspekt ist in der bekannten Literatur kaum Beachtung geschenkt worden, was einer weiteren Verbreitung von Verfahren mit automatischer Ergebnisverifikation sicherlich nicht förderlich war. Im Rahmen dieser Arbeit entstand das Programmpaket GO zur globalen Optimierung ohne Nebenbedingungen, das eine interaktive Schnittstelle für die Funktionseingabe aufweist und darauf aufbauend Funktionsauswertungen mit automatischer Differentiation (d. h. mit automatischer Berechnung des Gradienten und der Hessematrix) ermöglicht (vgl. [47], [9]). Aufgrund der Implementierung in PASCAL–XSC ([29], [30]) kann dieses Programmpaket auf den verschiedensten Rechnertypen genutzt werden.

Übersicht

Im ersten Kapitel dieser Arbeit werden zunächst einige grundlegende Bezeichnungen, Definitionen und Sätze zusammengestellt und die für die Präsentation der Algorithmen gewählte Darstellungsform erläutert. Danach wird ein kurzer Abriß über die Grundlagen der nichtlinearen lokalen und globalen Optimierung mit einer Zusammenstellung der wichtigsten Begriffe gegeben. Die sich anschließenden Abschnitte über Rechnerarithmetik und Intervallarithmetik bilden die Grundlage für die Berechnung garantierter Ergebnisse und damit für die in der Arbeit vorgestellten Algorithmen. Außerdem erfolgt eine Einführung in die im Verfahren eingesetzte erweiterte Intervallarithmetik und deren Anwendung im Rahmen des Intervall-Newton-Verfahrens. Der letzte Abschnitt von Kapitel 1 enthält einen knapp gehaltenen Überblick über die Methode der automatischen Differentiation, die es ermöglicht, Gradienten und Hessematrizen – ohne explizite Kenntnis der entsprechenden formalen Darstellungen – automatisch bei der Funktionsauswertung mitzuberechnen.

Kapitel 2 beschäftigt sich mit dem eigentlichen Thema der Arbeit, der globalen Optimierung mit automatischer Ergebnisverifikation. Nach einer kurzen Einführung werden zunächst der grundlegende Algorithmus in seiner originalen Form vorgestellt und diverse Modifikationen entwickelt. Nach der Entwicklung und theoretischen Behandlung spezieller Formen des Mono-

tonietests und des Konkavitätstests mit Randbehandlung, wird ausführlich auf den Intervall-Newton-Schritt eingegangen. Für diesen werden zahlreiche Varianten entwickelt, in denen zum einen spezielle, auf die erweiterte Intervallarithmetik zurückgehende Aufteilungstechniken, die eine bessere Ausnutzung des Rechenaufwandes ermöglichen, und zum anderen optimale Präkonditionierer nach Kearfott [25] zum Einsatz kommen. Im Anschluß daran werden die Randbehandlung und der Verifikationsschritt zum Nachweis der Eindeutigkeit globaler Minimalstellen innerhalb der berechneten Einschließungen beschrieben. Danach wird der Algorithmus in seiner vollständigen Form formuliert, und es werden abschließend einige weitere Modifikationsmöglichkeiten zusammengestellt.

Die praktische Durchführung des Optimierungsverfahrens und seine Implementierung in PASCAL–XSC sind Thema von Kapitel 3, in dem auf die Anwendung der Maschinenintervallarithmetik und auf die programmtechnische Organisation des vollständigen Algorithmus eingegangen wird. Daneben wird ein kurzer Einblick in die verwendeten Datenstrukturen und Operationen gegeben und der Einsatz von maximal genauen Ausdrucksauswertungen erläutert. Den Abschluß des Kapitels bildet eine Programmform des vollständigen Algorithmus.

In Kapitel 4 wird das Verfahren auf zahlreiche aus der Literatur bekannte Testprobleme angewendet. Dabei werden Aufwandsvergleiche für die verschiedenen Varianten des Algorithmus betrachtet und die Verbesserungen des Verfahrens gegenüber den bekannten Standardverfahren demonstriert. Kapitel 5 schließt die Arbeit mit einer kurzen Zusammenfassung und einem Ausblick auf zukünftige Arbeiten ab.

Kapitel 1

Einige Grundlagen

1.1 Grundlegende Bezeichnungen, Definitionen und Sätze

In der vorliegenden Arbeit bezeichnet stets

$\mathbb{R}$	die Menge der reellen Zahlen,
$\mathbb{R}^n$	die Menge der n-dimensionalen Vektoren über $\mathbb{R}$,
$\mathbb{R}^{n\times m}$	die Menge der $n \times m$-Matrizen über $\mathbb{R}$,
$\mathbb{P}\mathbb{R}$	die Potenzmenge über den reellen Zahlen,
$\mathbb{P}\mathbb{R}^n$	die Potenzmenge über den Vektoren und
$\mathbb{P}\mathbb{R}^{n\times m}$	die Potenzmenge über den Matrizen.

Für einen Vektor $x \in \mathbb{R}^n$ bzw. eine Matrix $M \in \mathbb{R}^{n\times n}$ werden die Komponenten durch *untere* Indizes und Aufzählungen oder Folgen von Vektoren bzw. Matrizen durch *obere* Indizes gekennzeichnet. Es bezeichnet somit

$x_i \in \mathbb{R}$	die i-te Komponente des Vektors x,
$M_i \in \mathbb{R}^n$	die i-te Zeile der Matrix M,
$M_{*i} \in \mathbb{R}^n$	die i-te Spalte der Matrix M,
$M_{ij} \in \mathbb{R}$	die j-te Komponente der i-ten Zeile der Matrix M,
$x^k \in \mathbb{R}^n$	den k-ten Vektor in einer Aufzählung oder Folge und
$M^k \in \mathbb{R}^{n\times n}$	die k-te Matrix in einer Aufzählung oder Folge.

Für M_{ij} wird auch die Notation $M_{i,j}$ verwendet.

Im allgemeinen werden reelle Zahlen und Vektoren mit Kleinbuchstaben, reelle Matrizen mit Großbuchstaben bezeichnet. Beim Übergang zu Intervallen und Intervallvektoren (vgl. Abschnitt 1.5) werden Großbuchstaben auch für skalare und vektorielle Werte verwendet. Die jeweilige Bedeutung geht jedoch stets aus dem Kontext hervor, wie auch bei der Verwendung der 0 als reelle Null, als Nullvektor oder als Nullmatrix. Mengen werden zur späteren Unterscheidung von Intervallen durch kalligraphische Symbole bezeichnet (Beispiel: $\mathcal{M} \in I\!P I\!R$).

Für die inneren und äußeren Multiplikationen von reellen Zahlen, Vektoren und Matrizen verwenden wir stets das Symbol „$\cdot$“, das wie üblich auch weggelassen werden darf. Wir bezeichnen somit beispielsweise das Skalarprodukt zweier Vektoren $a, b \in I\!R^n$ durch

$$a \cdot b = ab = \sum_{i=1}^{n} a_i \cdot b_i.$$

Für eine reelle Funktion $f : D \subseteq I\!R^n \to I\!R$ mit $f \in C^2(D)$ bezeichnet

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix}$$

den *Gradienten* von f und

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix}$$

die *Hessematrix* von f. Für $g : D \subseteq I\!R^n \to I\!R^n$ mit $g \in C^1(D)$ bezeichnet

$$J_g(x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1}(x) & \frac{\partial g_1}{\partial x_2}(x) & \dots & \frac{\partial g_1}{\partial x_n}(x) \\ \frac{\partial g_2}{\partial x_1}(x) & \frac{\partial g_2}{\partial x_2}(x) & \dots & \frac{\partial g_2}{\partial x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g_n}{\partial x_1}(x) & \frac{\partial g_n}{\partial x_2}(x) & \dots & \frac{\partial g_n}{\partial x_n}(x) \end{pmatrix}$$

die *Jacobimatrix* von g.

Nach dem Satz von Schwarz ist die Hessematrix symmetrisch, und es gilt außerdem mit $g(x) = \nabla f(x)$ die Beziehung $J_g(x) = \nabla^2 f(x)$.

Definition 1.1.1 *Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix und $x \in \mathbb{R}^n$, dann heißt der Ausdruck*

$$Q(A,x) = \sum_{i,j=1}^{n} A_{ij} x_i x_j$$

quadratische Form von A und x. Die Matrix A heißt dann

positiv definit, wenn gilt $Q(A,x) > 0$, für alle $x \neq 0$,

positiv semidefinit, wenn gilt $Q(A,x) \geq 0$, für alle $x \neq 0$.

Satz 1.1.1 *Eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ ist genau dann positiv definit (bzw. positiv semidefinit), wenn alle ihre Eigenwerte positiv (bzw. nichtnegativ) sind.*

Beweis: Siehe z. B. [27]. □

Satz 1.1.2 *Ist die symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ positiv definit (bzw. positiv semidefinit), so gilt $A_{ii} > 0$ (bzw. $A_{ii} \geq 0$) für alle $i = 1, 2, \ldots, n$.*

Beweis: Ist A positiv definit (bzw. positiv semidefinit), so führt die Annahme $A_{ii} \leq 0$ (bzw. $A_{ii} < 0$) für ein i mit $x := e^i$ (i-ter Einheitsvektor) wegen

$$Q(A,x) = \sum_{j,k=1}^{n} A_{jk} x_j x_k = A_{ii}$$

sofort zu einem Widerspruch. □

Für $r \in \mathbb{R}$, $x \in \mathbb{R}^n$ und $M \in \mathbb{R}^{n\times n}$ verwenden wir die Bezeichnungen

$$\begin{aligned} |r| & & & \text{(Betrag)}, \\ \|x\| & := & \sqrt{\sum_{i=1}^{n} |x_i|^2} & \text{(euklidische Norm)}, \\ \|x\|_\infty & := & \max_i |x_i| & \text{(Maximumnorm)}, \end{aligned}$$

$$\|M\| := \sqrt{\sum_{i,j=1}^{n} |M_{ij}|^2} \quad \text{(Schur-Norm)},$$

$$\|M\|_\infty := \max_i \sum_{j=1}^{n} |M_{ij}| \quad \text{(Zeilensummennorm) und}$$

$$\rho(M) := \max\{|\lambda| \mid \lambda \text{ Eigenwert von } M\} \quad \text{(Spektralradius)}.$$

1.2 Darstellung der Algorithmen

Zur Beschreibung der Algorithmen in dieser Arbeit verwenden wir eine PASCAL-ähnliche Notation. Aus Gründen der Übersichtlichkeit werden einzelne Teilschritte durchnumeriert und jeweils durch einen Punkt abgeschlossen, während mehrere darin vorkommende Anweisungen durch einen Strichpunkt abgetrennt werden. In den Teilschritten bzw. Anweisungen können Unteralgorithmen in der Prozedurform

Algorithmusname (*Argumentliste*)

oder in der Funktionsform

Variable := Algorithmusname (*Argumentliste*)

auftreten. Der kontrollierte, vorzeitige Abbruch eines Unteralgorithmus unter Beibehaltung der bis dahin berechneten Werte wird durch die Anweisung

return

beschrieben. Die Argumente der Argumentliste können nach der Durchführung eines Unteralgorithmus mit veränderten Werten zurückgegeben werden. Im Rahmen der vier möglichen Schleifenkonstrukte

for *Laufindex* := *Untergrenze* **to** *Obergrenze* **do**
    Anweisung(en)

for all *Objektbeschreibung* **do**
    Anweisung(en)

repeat
    Anweisung(en)
until *Abbruchbedingung*

while *Durchlaufbedingung* **do**
Anweisung(en)

sind die Anweisungen

$\textbf{exit}_{Laufindex\text{-loop}}$, $\textbf{exit}_{while\text{-loop}}$ und $\textbf{exit}_{repeat\text{-loop}}$

zum vorzeitigen Ende der jeweiligen Schleife und die Anweisung

$\textbf{next}_{Laufindex}$

für den Abbruch des aktuellen und den Übergang zum nächsten Schleifendurchlauf zugelassen. Verzweigungen sind durch

if *Bedingung* **then**
Anweisung(en)
else
Anweisung(en)

möglich, wobei der **else**-Teil optional ist.

1.3 Nichtlineare Optimierung

In diesem Abschnitt wollen wir die nachfolgend verwendeten Bezeichnungen für das allgemeine Optimierungsproblem festlegen und einige Definitionen und Sätze zusammenstellen, die sowohl das lokale als auch das globale nichtlineare Optimierungsproblem betreffen.

Das allgemeine Optimierungsproblem

Das allgemeine nichlineare Optimierungsproblem ist gegeben durch

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.1}$$

und durch die Angabe von *Nebenbedingungen*

$$\begin{array}{rcll} g_i(x) & \leq & 0, & i = 1, \ldots, k \\ h_i(x) & = & 0, & i = k+1, \ldots, m \end{array} \tag{1.2}$$

mit $m - k \leq n$ oder in Vektorschreibweise durch

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{mit} \quad g(x) \leq 0,\ h(x) = 0. \tag{1.3}$$

Dabei ist mindestens eine der Funktionen $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^k$ und $h : \mathbb{R}^n \to \mathbb{R}^{m-k}$ nichlinear. Maximierungsprobleme werden durch Negation von f behandelt.

Definition 1.3.1 *Ein Punkt $x \in \mathbb{R}^n$ mit $g(x) \leq 0$ und $h(x) = 0$ heißt zulässiger Punkt.*

Eine Nebenbedingung $g_i(x) \leq 0$ heißt aktiv, falls gilt $g_i(x) = 0$.

Ein zulässiger Punkt x^* heißt lokale Minimalstelle (bzw. strenge lokale Minimalstelle), wenn es ein $\varepsilon > 0$ gibt, so daß gilt $f(x^*) \leq f(x)$ (bzw. $f(x^*) < f(x)$) für alle x mit $||x - x^*|| < \varepsilon$. $f(x^*)$ heißt dann lokales Minimum (bzw. strenges lokales Minimum).

Ein zulässiger Punkt x^* heißt globale Minimalstelle, wenn gilt $f(x^*) \leq f(x)$ für alle zulässigen Punkte x. $f(x^*)$ heißt dann globales Minimum.

Optimalitätsbedingungen

Eine allgemeine, notwendige Optimalitätsbedingung für das Problem (1.3) ist das *John-Kriterium*. Besser bekannt als dieses Kriterium sind jedoch die *Kuhn-Tucker-Bedingungen*.

Definition 1.3.2 *Es seien Vektoren $u = (u_0, \ldots, u_k)^T \in \mathbb{R}^{k+1}$ und $v = (v_{k+1}, \ldots, v_m)^T \in \mathbb{R}^{m-k}$ gegeben, dann heißt die Funktion*

$$\psi(x, u, v) = u_0 \cdot f(x) + (u_1, \ldots, u_k) \cdot g(x) + v^T \cdot h(x)$$

verallgemeinerte Lagrange-Funktion zum Problem (1.3).

Satz 1.3.1 (John-Kriterium) *Sind $f, g, h \in C^1$ und ist x^* eine lokale Minimalstelle von (1.3), dann existieren Vektoren $u = (u_0, \ldots, u_k)^T \in \mathbb{R}^{k+1}$ und $v = (v_{k+1}, \ldots, v_m)^T \in \mathbb{R}^{m-k}$, so daß gilt:*

$$\left.\begin{array}{rcl} \nabla \psi(x^*, u, v) & = & 0 \\ (u_1, \ldots, u_k) \cdot g(x^*) & = & 0 \\ u & \geq & 0 \\ \begin{pmatrix} u \\ v \end{pmatrix} & \neq & 0 \end{array}\right\} \tag{1.4}$$

Handelt es sich um ein Minimierungsproblem ohne Nebenbedingungen, so reduziert sich (1.4) zu

$$\nabla f(x^*) = 0. \tag{1.5}$$

Beweis: Siehe z. B. [60]. □

Definition 1.3.3 *Ist x zulässiger Punkt von (1.3) dann heißt die Menge $A(x) = \{i : g_i(x) = 0 \ \wedge \ 1 \leq i \leq k\}$ aktive Indexmenge von x.*
Der zulässige Punkt x erfüllt die Regularitätsbedingung, wenn die Menge der Gradienten $\nabla g_i(x), \nabla h_j(x)$ mit $i \in A(x), j = k+1, \ldots, m$ linear unabhängig ist.

Definition 1.3.4 *Es seien die Vektoren $u = (u_1, \ldots, u_k)^T \in \mathbb{R}^k$ und $v = (v_{k+1}, \ldots, v_m)^T \in \mathbb{R}^{m-k}$ gegeben, dann heißt die Funktion*

$$L(x, u, v) = f(x) + u^T \cdot g(x) + v^T \cdot h(x)$$

Lagrange-Funktion zum Problem (1.3).

Satz 1.3.2 (Kuhn-Tucker-Bedingungen) *Sind $f, g, h \in C^1$, ist x^* eine lokale Minimalstelle von (1.3) und erfüllt x^* die Regularitätsbedingung, dann existieren Vektoren $u = (u_1, \ldots, u_k)^T \in \mathbb{R}^k$ und $v = (v_{k+1}, \ldots, v_m)^T \in \mathbb{R}^{m-k}$, so daß gilt:*

$$\left.\begin{array}{rcl} \nabla L(x^*, u, v) & = & 0 \\ u^T g(x^*) & = & 0 \\ u & \geq & 0 \end{array}\right\} \tag{1.6}$$

Beweis: Siehe z. B. [60]. □

Ihre Verwendung in numerischen Optimierungsverfahren finden diese Optimalitätsbedingungen darin, daß zunächst das Problem (1.4) oder (1.6) für x, u, v mit zulässigem x gelöst wird und die gefundene Lösung auf ihre Tauglichkeit als Minimierer von (1.3) untersucht wird.

Optimierung ohne Nebenbedingungen

Für $k = m = 0$ in (1.3) ergibt sich das Problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.7}$$

ohne Nebenbedingungen, das üblicherweise als

$$\min_{x \in X} f(x) \tag{1.8}$$

mit $X = [a, b]$ und $a, b \in \mathbb{R}^n$ gestellt wird, was einer speziellen Nebenbedingung der Form $a \leq x \leq b$ entspricht. Eine hinreichende Bedingung für eine lokale Minimalstelle x^* von (1.7) bzw. (1.8) liefern die nachfolgenden Sätze.

Definition 1.3.5 *Sei $D \subseteq \mathbb{R}^n$ konvex, dann heißt $f : \mathbb{R}^n \to \mathbb{R}$ konvex in D, wenn für alle $x, y \in D$ und für alle $\lambda \in [0,1]$ gilt*

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

Die Funktion f heißt streng konvex in D, wenn für alle $x, y \in D$ und für alle $\lambda \in [0,1]$ gilt

$$f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda) f(y).$$

Satz 1.3.3 *Ist $f \in C^2$ und ist die Hessematrix $\nabla^2 f(x)$ positiv semidefinit (bzw. positiv definit) in der konvexen Menge $D \subseteq \mathbb{R}^n$, so ist f konvex (bzw. streng konvex) in D.*

Beweis: Siehe z. B. [60]. □

Satz 1.3.4 *Ist $f \in C^2$, gilt für x^* die Beziehung (1.5) und ist f konvex (bzw. streng konvex) in x^*, dann ist dies hinreichend dafür, daß x^* eine lokale (bzw. eine strenge lokale) Minimalstelle von f ist.*

Beweis: Siehe z. B. [60]. □

Wir wollen hier keine Beschreibung der zahlreichen klassischen Verfahren der lokalen nichtlinearen Optimierung geben und verweisen hierzu auf die umfangreiche Literatur auf diesem Gebiet.

Globale Optimierung

Das Ziel der globalen Optimierung ist es, nicht nur ein lokales Minimum der Funktion f zu finden, sondern ihr globales Minimum, also das kleinste lokale Minimum. Im Unterschied zur lokalen Minimierung kann kein allgemeines Kriterium für das tatsächliche Erreichen des globalen Minimums angegeben werden. Darüber hinaus muß das globale Minimum nicht notwendigerweise ein lokales Minimum sein, da es auch auf dem durch die Nebenbedingungen spezifizierten Rand des zulässigen Bereichs liegen kann.
Im folgenden bezeichnet

- f^* das globale Minimum von f und
- $\mathcal{X}^*$ die Menge aller globalen Minimalstellen.

Es ist somit für die Problemstellung (1.8)

$$f^* = \min\{f(x) \mid x \in X\} \quad \text{und} \quad \mathcal{X}^* = \{x^* \in X \mid f(x^*) = f^*\}.$$

Wir wollen auch hier wieder auf die Beschreibung gängiger klassischer Verfahren für die globale Optimierung verzichten und verweisen speziell auf den weitreichenden Überblick in [57]. Zu erwähnen ist, daß die meisten Verfahren aus der Kombination einer globalen und einer lokalen Strategie bestehen und daß eine Konvergenzaussage für die Folge der Testpunkte, die von einem Verfahren zur Annäherung an die globale(n) Minimalstelle(n) generiert wird, durch den nachfolgenden Satz gegeben ist.

Satz 1.3.5 *Ein globaler Optimierungsalgorithmus konvergiert genau dann gegen das globale Minimum einer stetigen Funktion f, wenn die Folge der durch den Algorithmus generierten Testpunkte überall dicht im kompakten Minimierungsbereich X liegt.*

Beweis: Siehe [57]. □

In [57] wird außerdem die Bedeutung von Heuristiken für die Konstruktion globaler Optimierungsverfahren unterstrichen. Als grobe Klassifizierung der globalen Optimierungsverfahren wir dort die Unterteilung in

- Verfahren mit garantierter Genauigkeit,
- direkte Verfahren und
- indirekte Verfahren

vorgenommen. Dies unterstreicht schon die Bedeutung der in dieser Arbeit behandelten Methode, die der erstgenannten Klasse angehört. Durch den Einsatz der Intervallarithmetik erlaubt sie sowohl ein *kontinuierliches Abdecken* des Minimierungsbereichs X als auch ein Erfassen sämtlicher bei der numerischen Durchführung auf dem Rechner auftretender Rundungsfehler. In den nächsten Abschnitten wollen wir einige Grundlagen für die Entwicklung eines solchen Verfahrens zusammenstellen.

1.4 Rechnerarithmetik

Da auf einer Rechenanlage nur endlich viele Zahlen darstellbar sind, muß die Menge der reellen Zahlen auf eine Teilmenge, die sogenannten *Gleitkommazahlen* oder auch *Maschinenzahlen*, abgebildet werden. Entsprechendes gilt

für die Vektoren und Matrizen über den reellen Zahlen. Außerdem müssen auch die exakten arithmetischen Grundoperationen $+$, $-$, $\cdot$ und $/$ für die inneren und äußeren Verknüpfungen der reellen Räume auf dem Rechner durch die sogenannten *Gleitkommaoperationen* oder auch *Maschinenoperationen* approximiert werden. Kontrolliertes wissenschaftliches Rechnen auf einer Rechenanlage erfordert daher eine mathematisch exakte Definition der Rechnerarithmetik. Diese wurde von Kulisch und Miranker in [31], [33] und [35] angegeben. In diesem Abschnitt wollen wir kurz die wichtigsten Begriffe und Schreibweisen aus diesem Bereich erläutern.

Es bezeichnen stets

R	die Menge der Maschinenzahlen,
R^n	die Menge der n-dimensionalen Vektoren über R und
$R^{n\times m}$	die Menge der $n \times m$-Matrizen über R.

Die reellen Räume werden gemäß

$$\begin{array}{lcl} \mathbb{R} & \to & R \\ \mathbb{R}^n & \to & R^n \\ \mathbb{R}^{n\times n} & \to & R^{n\times n} \end{array}$$

auf die Maschinenräume abgebildet. Im folgenden sei S ein Raum der linken Spalte und T der zugehörige Raum der rechten Spalte.

Definition 1.4.1 *Die Abbildung $\square : S \to T$ mit den Eigenschaften*

(R1) $\square a = a$ *für alle $a \in T$,* *(Projektion)*

(R2) $a \le b \;\Rightarrow\; \square a \le \square b$ *für alle $a, b \in S$,* *(Monotonie)*

heißt Rundung. Eine Rundung heißt antisymmetrisch, wenn gilt

(R3) $\square(-a) = -\square a$ *für alle $a \in S$.*

Eine Rundung heißt nach unten gerichtet (nach oben gerichtet), wenn gilt

(R4) $\square(a) \le a$ $(\square(a) \ge a)$ *für alle $a \in S$.*

Alle inneren und äußeren Verknüpfungen $+$, $-$, $\cdot$ und $/$ in S werden durch die Gleitpunktverknüpfungen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ in T approximiert, die über das Prinzip des Semimorphismus definiert sind.

Definition 1.4.2 *Eine antisymmetrische Rundung $\square : S \to T$ heißt Semimorphismus, wenn alle inneren und äußeren Verknüpfungen in T durch*

$$\text{(RG)} \quad a \boxtimes b := \square(a \circ b) \quad \text{für alle } a, b \in T \text{ und } \circ \in \{+, -, \cdot, /\}$$

definiert sind.

Die dadurch erklärten Verknüpfungen für Elemente aus T werden stets zunächst in S ausgeführt und das Ergebnis erst dann wieder nach T gerundet. Semimorphe Verknüpfungen sind damit von *maximaler Genauigkeit*, in dem Sinne, daß zwischen dem in S berechneten Verknüpfungsergebnis $a \circ b$ und seiner Approximation $a \boxtimes b$ in T kein weiteres Element aus T liegt (vgl. [31], [33]). Für die Produkträume ist dies komponentenweise zu verstehen.

Beispiel 1.4.1 Das Skalarprodukt zweier Gleitpunktvektoren $a, b \in R^n$ wird über den Semimorphismus definiert durch

$$a \boxdot b := \square(a \cdot b) = \square \left(\sum_{i=1}^{n} a_i \cdot b_i \right).$$

Bemerkung: Nachfolgend verwenden wir das zuvor als allgemeines Rundungssymbol gebrauchte Zeichen $\square$ für die antisymmetrische Rundung zur nächstgelegenen Maschinenzahl. Die nach unten bzw. nach oben gerichteten Rundungen zur nächstkleineren bzw. nächstgrößeren Maschinenzahl bezeichnen wir mit $\bigtriangledown$ bzw. $\bigtriangleup$. Für sie gilt die Identität

$$\bigtriangledown(-x) = - \bigtriangleup x \quad \text{für alle } x \in S.$$

Wir verwenden die Bezeichnung $f_{\square}(x)$ für die Maschinenauswertung einer Funktion f, d. h. einer Berechnung des Funktionswertes unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$. Den Begriff maximale Genauigkeit verwenden wir auch im Zusammenhang mit Maschinenauswertungen $s_{\square}$ von Funktionen $s : I\!R \to I\!R$, die durch

$$s_{\square}(x) := \square(s(x))$$

ebenfalls über den Semimorphismus definiert sind. Den auf der Rechenanlage zur Verfügung stehenden Satz maximal genauer Funktionen (Standardfunktionen) $s : R \to R$ bezeichnen wir im folgenden mit

$$\mathcal{SF} = \{\exp, \ln, \sin, \cos, \ldots\}.$$

1.5 Intervallarithmetik, Maschinenintervallarithmetik

Das wesentliche Hilfsmittel zur Berechnung von Schranken für die Lösung eines numerischen Problems und damit zur Gewinnung von garantierten Aussagen ist die Intervallrechnung. Eine ausführliche Definition und Darstellung der Intervallrechnung findet sich in [1], [2] oder [38], eine gelungene Einführung z. B. in [36]. Die Behandlung der zugehörigen Maschinenintervallarithmetik findet sich in [31] und [33]. Im folgenden geben wir eine Zusammenstellung der wichtigsten Begriffe und Eigenschaften.

Definition 1.5.1 *Die Menge* $A := [\underline{A}, \overline{A}] := \{x \in I\!R \mid \underline{A} \leq x \leq \overline{A}\}$ *mit* $\underline{A}, \overline{A} \in I\!R$ *heißt Intervall.* $\underline{A} = \inf A$ *heißt Infimum oder Unterschranke von* A, $\overline{A} = \sup A$ *heißt Supremum oder Oberschranke von* A. *Ein Intervall* A *heißt Punktintervall, wenn gilt* $\underline{A} = \overline{A}$.

Es bezeichnet im folgenden stets

$I\!I\!R$	die Menge der Intervalle,
$I\!I\!R^n$	die Menge der n-dimensionalen Vektoren über $I\!I\!R$ und
$I\!I\!R^{n\times m}$	die Menge der $n \times m$-Matrizen über $I\!I\!R$.

Zur Indizierung von Intervallen, Intervallvektoren und Intervallmatrizen verwenden wir die in Abschnitt 1.1 beschriebene Notation. Für $A \in I\!I\!R$, $X \in I\!I\!R^n$ und $M \in I\!I\!R^{n\times n}$ verwenden wir außerdem die Bezeichnungen

$$
\begin{array}{lcll}
m(A) & := & \frac{1}{2}(\underline{A} + \overline{A}) & \text{(Mittelpunkt)}, \\
d(A) & := & \overline{A} - \underline{A} & \text{(Durchmesser)}, \\
d(X) & := & \max\limits_{1\leq i\leq n} d(X_i) & \text{(Durchmesser(maximum))}, \\
|A| & := & \max\{|a| \mid a \in A\} & \text{(Betrag)}, \\
\langle A\rangle & := & \min\{|a| \mid a \in A\} & \text{(Betragsminimum)}, \\
\|x\|_\infty & := & \max\limits_i |x_i| & \text{(Maximumnorm) und} \\
\|M\|_\infty & := & \max\limits_i \sum\limits_{j=1}^{n} |M_{ij}| & \text{(Zeilensummennorm)}.
\end{array}
$$

Dabei sind Mittelpunkt und Betrag für Vektoren und Matrizen jeweils komponentenweise definiert. Für $A, B \in I\!I\!R$ gelten die Beziehungen

$$d(A \pm B) \quad = \quad d(A) + d(B),$$

$$\begin{aligned} d(A \cdot B) &\leq d(A) \cdot |B| + d(B) \cdot |A|, \\ |A + B| &\leq |A| + |B|, \\ |A \cdot B| &= |A| \cdot |B|. \end{aligned}$$

Die Beweise finden sich z. B. in [1]. Als weitere Schreibweisen für $A \in I\!I\!R$ und $X \in I\!I\!R^n$ benötigen wir

$$\begin{aligned} \overset{\circ}{A} &:= \{a \in A \mid \underline{A} < a < \overline{A}\} && \text{(Inneres von } A\text{)}, \\ \overset{\circ}{X} &:= \{x \in X \mid \underline{X_i} < x_i < \overline{X_i} \text{ für alle } i\} && \text{(Inneres von } X\text{)}, \\ \partial A &:= \{\underline{A}, \overline{A}\} && \text{(Rand von } A\text{)}, \\ \partial X &:= \{x \in X \mid x_i \in \partial X_i \text{ für ein } i\} && \text{(Rand von } X\text{)}. \end{aligned}$$

Vergleiche und Teilmengenbeziehungen werden mengentheoretisch interpretiert und sind für Vektoren und Matrizen genau dann erfüllt, wenn sie für alle ihre Komponenten erfüllt sind. Für $A, B \in I\!I\!R$ gilt

$$\begin{aligned} A = B &\iff \underline{A} = \underline{B} \ \wedge \ \overline{A} = \overline{B}, \\ A \subseteq B &\iff \underline{A} \geq \underline{B} \ \wedge \ \overline{A} \leq \overline{B}, \\ A \overset{\circ}{\subset} B &\iff \underline{A} > \underline{B} \ \wedge \ \overline{A} < \overline{B} \iff A \subseteq \overset{\circ}{B}\ . \end{aligned}$$

Die Verbandsoperationen $\cap$ und $\underline{\cup}$ für $A, B \in I\!I\!R$ sind definiert durch

$$\begin{aligned} A \cap B &:= [\max\{\underline{A}, \underline{B}\}, \min\{\overline{A}, \overline{B}\}], && \text{(Schnitt)} \\ A \mathbin{\underline{\cup}} B &:= [\min\{\underline{A}, \underline{B}\}, \max\{\overline{A}, \overline{B}\}], && \text{(Intervallhülle)} \end{aligned}$$

wobei $A \cap B$ nur definiert ist, falls $\max\{\underline{A}, \underline{B}\} \leq \min\{\overline{A}, \overline{B}\}$ erfüllt ist. Die intervallarithmetischen Operationen werden definiert durch

$$A \circ B := \{a \circ b \mid a \in A, b \in B\},$$

wobei $A, B \in I\!I\!R$ und $\circ \in \{+, -, \cdot, /\}$. Die explizite Berechnung dieser Intervalloperationen kann durch

$$\begin{aligned} A \ + \ B &= [\underline{A} + \underline{B},\ \overline{A} + \overline{B}], \\ A \ - \ B &= [\underline{A} - \overline{B},\ \overline{A} - \underline{B}], \\ A \ \cdot \ B &= [\min\{\underline{A}\,\underline{B}, \underline{A}\overline{B}, \overline{A}\underline{B}, \overline{A}\,\overline{B}\},\ \max\{\underline{A}\,\underline{B}, \underline{A}\overline{B}, \overline{A}\underline{B}, \overline{A}\,\overline{B}\}], \\ A \ / \ B &= [\underline{A}, \overline{A}] \cdot [1/\overline{B},\ 1/\underline{B}] \text{ für } 0 \notin B \end{aligned}$$

erfolgen. Dabei kann die Bildung des Minimums bzw. des Maximums in der Multiplikation durch vorherige Untersuchung der Intervallgrenzen in den meisten Fällen vermieden werden. Im Hinblick auf die in Abschnitt 1.6 beschriebenen Erweiterungen geben wir noch eine entsprechende Formulierung mit Fallunterscheidung für die Division A/B falls $0 \notin B$:

$$A/B = \begin{cases} [\overline{A}/\underline{B}, \underline{A}/\overline{B}] & \text{für } \overline{A} \leq 0 \ \wedge \ \overline{B} < 0 \\ [\underline{A}/\underline{B}, \overline{A}/\overline{B}] & \text{für } \overline{A} \leq 0 \ \wedge \ 0 < \underline{B} \\ [\overline{A}/\overline{B}, \underline{A}/\overline{B}] & \text{für } \underline{A} < 0 < \overline{A} \ \wedge \ \overline{B} < 0 \\ [\underline{A}/\underline{B}, \overline{A}/\underline{B}] & \text{für } \underline{A} < 0 < \overline{A} \ \wedge \ 0 < \underline{B} \\ [\overline{A}/\overline{B}, \underline{A}/\underline{B}] & \text{für } 0 \leq \underline{A} \ \wedge \ \overline{B} < 0 \\ [\underline{A}/\overline{B}, \overline{A}/\underline{B}] & \text{für } 0 \leq \underline{A} \ \wedge \ 0 < \underline{B} \end{cases} \tag{1.9}$$

Addition und Multiplikation sind kommutativ und assoziativ, es gilt jedoch nur die sogenannte *Subdistributivität*

$$A \cdot (B + C) \subseteq A \cdot B + A \cdot C$$

für Intervalle $A, B, C \in I\mathbb{R}$. Eine weitere, zentrale Eigenschaft der Intervalloperationen ist die *Inklusionsisotonie*

$$\begin{aligned} a \in A \ \wedge \ b \in B \quad &\Longrightarrow \quad a \circ b \in A \circ B \\ A \subseteq C \ \wedge \ B \subseteq D \quad &\Longrightarrow \quad A \circ B \subseteq C \circ D \end{aligned}$$

für alle $\circ \in \{+, -, \cdot, /\}$ mit $a, b \in \mathbb{R}$ und $A, B, C, D \in I\mathbb{R}$.

Mit Hilfe der Intervallarithmetik ist es möglich, den Wertebereich einer Funktion einzuschließen. In diesem Zusammenhang verwenden wir für Funktionen $f : D \to \mathbb{R}$ mit $D \subseteq \mathbb{R}$ oder $D \subseteq \mathbb{R}^n$ die Bezeichnungen

- $f(X)$ für den *Wertebereich* von f,
- $F(X)$ für die *Intervallauswertung*, also für die Auswertung der *Intervallerweiterung* oder auch *Einschließungsfunktion* F von f,
- $\overline{F_X}$ für $\sup F(X)$ und
- $\underline{F_X}$ für $\inf F(X)$,

wobei für $f(X)$ und $F(X)$ stets die Beziehung

$$f(X) \subseteq F(X)$$

gilt. Entsprechende Bezeichnungen gelten auch für vektorwertige Funktionen. Zur *natürlichen Intervallerweiterung* einer Funktion f kommt man, indem man alle auftretenden Variablen durch entsprechende Intervalle und alle Operationen durch die zugehörigen Intervalloperationen ersetzt. In Teilausdrücken können dabei auch sogenannte Standardintervallfunktionen $s \in \mathcal{SF}$ vorkommen, für die gilt

$$s(X) = S(X).$$

Für Punktintervalle bzw. Punktintervallvektoren als Argumente von f gilt stets

$$f(c) = F(c).$$

Auch für die Intervallerweiterungen gilt die Inklusionsisotonie

$$\begin{aligned} a \in A \quad &\Longrightarrow \quad f(a) \in F(A) \\ A \subseteq B \quad &\Longrightarrow \quad F(A) \subseteq F(B). \end{aligned}$$

Aufgrund der Tatsache, daß die natürliche Intervallauswertung üblicherweise große Überschätzungen mit sich bringt, werden häufig auch *zentrierte Formen* oder *Mittelwertsformen* verwendet (vgl. auch [48]). Die Mittelwertsform wird aus dem Mittelwertsatz abgeleitet und ist für $f : D \subseteq I\!R \to I\!R$ durch

$$F^c(X) = f(c) + F'(X) \cdot (X - c)$$

und für $f : D \subseteq I\!R^n \to I\!R$ durch

$$F^c(X) = f(c) + \boldsymbol{\nabla} F(X) \cdot (X - c)$$

definiert, wobei $c \in X$ ist und meistens $c = m(X)$ verwendet wird.

Beim Rechnen mit Intervallen auf einer Rechenanlage muß die Menge der reellen Intervalle auf die Menge der *Maschinenintervalle* abgebildet werden. Dazu werden die Intervallgrenzen durch gerichtete Rundungen auf Maschinenzahlen abgebildet. Man beachte jedoch, daß ein Intervall auch nach diesem Übergang auf ein Maschinenintervall

$$A := [\underline{A}, \overline{A}] := \{x \in I\!R \mid \underline{A} \leq x \leq \overline{A};\ \underline{A}, \overline{A} \in R\}$$

sämtliche *reelle* Werte zwischen den Grenzen repräsentiert, also nach wie vor das gesamte Kontinuum abdeckt. Entsprechendes gilt für die Vektoren und Matrizen über den reellen Intervallen. Somit müssen auch die exakten

arithmetischen Grundoperationen $+$, $-$, $\cdot$ und $/$ für die inneren und äußeren Verknüpfungen der reellen Intervallräume auf dem Rechner durch die sogenannten *Maschinenintervalloperationen* approximiert werden.
Es bezeichnen stets

IR	die Menge der Maschinenintervalle,
IR^n	die Menge der n-dimensionalen Vektoren über IR und
$IR^{n\times m}$	die Menge der $n \times m$-Matrizen über IR.

Die reellen Intervallräume werden gemäß

$$\begin{array}{lcl} I\!\!R & \to & IR \\ I\!\!R^n & \to & IR^n \\ I\!\!R^{n\times n} & \to & IR^{n\times n} \end{array}$$

auf die Maschinenräume abgebildet. Im folgenden sei IS ein Raum der linken Spalte und IT der zugehörige Raum der rechten Spalte.

Definition 1.5.2 *Die Abbildung $\diamondsuit : IS \to IT$ mit den Eigenschaften*

(R1) $\diamondsuit A = A$ *für alle $A \in IT$,*

(R2) $A \subseteq B \;\Rightarrow\; \diamondsuit A \subseteq \diamondsuit B$ *für alle $A, B \in IS$,*

(R3) $\diamondsuit(-A) = -\diamondsuit A$ *für alle $A \in IS$,*

(R4) $A \subseteq \diamondsuit A$ *für alle $A \in IS$,*

heißt antisymmetrische, nach außen gerichtete Rundung oder auch Intervallrundung.

Bemerkung: Die Rundung $\diamondsuit$ ist eindeutig bestimmt (vgl. [31], [33]).

Alle inneren und äußeren Verknüpfungen $+$, $-$, $\cdot$ und $/$ in IS werden durch die Maschinenintervallverknüpfungen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ in IT approximiert, die über das Prinzip des Semimorphismus durch

(RG) $A \circledcirc B := \diamondsuit(A \circ B)$ für alle $A, B \in IT$ und $\circ \in \{+, -, \cdot, /\}$

definiert sind. Wir verwenden die Bezeichnung $F_\diamondsuit(X)$ für die Maschinenintervallauswertung einer Intervallfunktion F unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$. Maschinenintervallfunktionen $S_\diamondsuit$ der

Intervallfunktionen $S : I\!R \to I\!R$ mit $S \in \mathcal{SF}$ (Standardfunktionen) sind über den Semimorphismus definiert durch

$$S_\diamond(X) := \diamondsuit(S(X)).$$

Die Inklusionsisotonie gilt für die Maschinenenoperationen in der Form

$$\begin{aligned} a \in A \ \wedge \ b \in B \quad &\Longrightarrow \quad a \circ b \in A \circ B \subseteq A \diamondsuit B \\ A \subseteq C \ \wedge \ B \subseteq D \quad &\Longrightarrow \quad A \diamondsuit B \subseteq C \diamondsuit D \end{aligned}$$

für alle $\circ \in \{+, -, \cdot, /\}$ mit $a, b \in R$ und $A, B, C, D \in I\,R$ und für die Maschinenintervallauswertungen in der Form

$$\begin{aligned} a \in A \quad &\Longrightarrow \quad f(a) \in F(A) \subseteq F_\diamond(A) \\ A \subseteq B \quad &\Longrightarrow \quad F_\diamond(A) \subseteq F_\diamond(B). \end{aligned}$$

Mittelpunkt und Durchmesser werden auf dem Rechner als

$$m_\square(X) = \square(m(X)) \quad \text{und} \quad d_\triangle(X) = \triangle(d(X))$$

definiert, wobei zu beachten ist, daß m und d damit zwar maximal genau sind, daß jedoch im allgemeinen gilt

$$m_\square(X) \neq m(X) \quad \text{bzw.} \quad d_\triangle(X) \neq d(X).$$

1.6 Erweiterte Intervallarithmetik

In dieser Arbeit werden wir im Rahmen eines Intervall-Newton-Schrittes (Abschnitte 1.7 und 2.5) Intervallausdrücke der Form

$$r - A/B$$

mit $r \in I\!R$ und $A, B \in I\,I\!R$ verwenden, in denen ein Intervall B auftritt, das möglicherweise die 0 enthält. Die Behandlung dieses Falles erfordert den Einsatz einer erweiterten Intervallarithmetik. In der Literatur finden sich verschiedene Ansätze für eine solche Arithmetik, wir wollen jedoch hier lediglich auf die eingeschränkte Erweiterung der in unserem Fall benötigten Operationen $-$ und $/$ eingehen (vgl. auch [13], [20] oder [39]). Im folgenden verwenden wir die Bezeichnungen

$$\begin{aligned} I\!I\!R_\infty &:= I\!I\!R \cup \{(-\infty, \rho] \mid \rho \in I\!R\} \cup \{[\lambda, \infty) \mid \lambda \in I\!R\} \cup \{(-\infty, \infty)\}, \\ I\,R_\infty &:= I\,R \cup \{(-\infty, \rho] \mid \rho \in R\} \cup \{[\lambda, \infty) \mid \lambda \in R\} \cup \{(-\infty, \infty)\}. \end{aligned}$$

Definition 1.6.1 *Ein Paar* $W = (W^1 \,|\, W^2) := W^1 \cup W^2$ *von Intervallen* $W^1, W^2 \in I\!I\!R_\infty \cup \{\emptyset\}$ *heißt erweitertes Intervallpaar, wenn entweder gilt* $\sup W^1 < \inf W^2$ *oder* $W^2 = \emptyset$.

Der Durchmesser eines erweiterten Intervallpaares $W = (W^1 \mid W^2)$ *mit endlichen Komponenten* $W^1, W^2 \in I\!I\!R \cup \{\emptyset\}$ *ist definiert durch*

$$d(W) = d(W^1) + d(W^2), \quad \textit{mit } d(\emptyset) = 0.$$

Seien $A, B \in I\!I\!R$, dann definieren wir den erweiterten Intervallquotienten $W = A/B$ für $0 \in B$ durch

$$W := \begin{cases} (-\infty, \infty) & \text{für } \underline{B} = \overline{B} = 0 \\ (-\infty, \infty) & \text{für } \underline{A} = \overline{A} = 0 \\ (-\infty, \infty) & \text{für } \underline{A} < 0 < \overline{A} \\ [\overline{A}/\underline{B}, \infty) & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ \underline{B} < \overline{B} = 0 \\ (-\infty, \overline{A}/\overline{B}] \cup [\overline{A}/\underline{B}, \infty) & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ \underline{B} < 0 < \overline{B} \\ (-\infty, \overline{A}/\overline{B}] & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ 0 = \underline{B} < \overline{B} \\ (-\infty, \underline{A}/\underline{B}] & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ \underline{B} < \overline{B} = 0 \\ (-\infty, \underline{A}/\underline{B}] \cup [\underline{A}/\overline{B}, \infty) & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ \underline{B} < 0 < \overline{B} \\ [\underline{A}/\overline{B}, \infty) & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ 0 = \underline{B} < \overline{B} \end{cases}$$

und für $0 \notin B$ durch (1.9). Darüber hinaus definieren wir die Differenz $U = r - W$ einer reellen Zahl r und eines erweiterten Intervallpaares W durch

$$U = \begin{cases} (-\infty, \infty) & \text{für } W = (-\infty, \infty) \\ (-\infty, r - \lambda] & \text{für } W = [\lambda, \infty) \\ [r - \rho, \infty) & \text{für } W = (-\infty, \rho] \\ (-\infty, r - \lambda] \cup [r - \rho, \infty) & \text{für } W = (-\infty, \rho] \cup [\lambda, \infty) \\ [r - \rho, r - \lambda] & \text{für } W = [\lambda, \rho] \end{cases}$$

Von den Verbandsoperationen benötigen wir für unsere Anwendungen außerdem die Schnittbildung eines erweiterten Intervallpaares mit einem Standard-Intervall, die wir für $W^1, W^2 \in I\!I\!R_\infty$ und $X \in I\!I\!R$ durch

$$W \cap X := (W^1 \cap X \mid W^2 \cap X)$$

definieren. Beim Übergang auf eine Rechenanlage verlangen wir auch von den erweiterten Intervalloperationen die Definition über den Semimorphis-

mus, so daß für die Maschinenoperation $\diamondsuit\!\!\!/$ und $\diamondsuit\!\!\!\!/$ gilt

$$\begin{aligned} A \diamondsuit\!\!\!/ B &= \diamondsuit(A/B), \\ r \diamondsuit\!\!\!\!/ U &= \diamondsuit(r-U), \end{aligned}$$

wobei $A, B \in IR$, $r \in R$ und U ein erweitertes Intervallpaar mit $U^1, U^2 \in IR_\infty$ ist. Die Rundung für erweiterte Intervallpaare definieren wir durch

$$\diamondsuit(W) = \begin{cases} (-\infty,\infty) & \text{für } W = (-\infty,\infty) \\ [\bigtriangledown(\lambda),\infty] & \text{für } W = [\lambda,\infty) \\ (-\infty,\bigtriangleup(\rho)] & \text{für } W = (-\infty,\rho] \\ (-\infty,\bigtriangleup(\rho)] \cup [\bigtriangledown(\lambda),\infty) & \text{für } W = (-\infty,\rho] \cup [\lambda,\infty) \\ [\bigtriangledown(\lambda),\bigtriangleup(\rho)] & \text{für } W = [\lambda,\rho] \end{cases}$$

für $W^1, W^2 \in I\mathbb{R}_\infty$, also $\lambda, \rho \in \mathbb{R}$.

1.7 Erweitertes Intervall-Newton-Verfahren

Zum besseren Verständnis der in Abschnitt 2.5 beschriebenen Verfahren wollen wir hier kurz auf das erweiterte Intervall-Newton-Verfahren im $\mathbb{R}^1$ eingehen und damit auch die unmittelbare Anwendung der erweiterten Intervallarithmetik demonstrieren. Das herkömmliche Intervall-Newton-Verfahren (vgl. z. B. [1], [2] oder [36]) zur Bestimmung der Nullstelle x^* der stetig differenzierbaren Funktion $f : \mathbb{R} \to \mathbb{R}$ ist ausgehend von einer Starteinschließung X^0 mit $0 \notin F'(X^0)$ definiert durch

$$X^{k+1} := \left(m(X^k) - \frac{f(m(X^k))}{F'(X^k)} \right) \cap X^k, \quad \text{für } k = 0, 1, 2, \ldots$$

Das Verfahren kann wegen der verwendeten Schnittbildung nicht divergieren, es ist jedoch aufgrund der Voraussetzung $0 \notin F'(X^0)$ nicht anwendbar, wenn im Startintervall X^0 mehrere Nullstellen von f liegen. Unter Verwendung der erweiterten Intervalloperationen aus dem letzten Abschnitt läßt sich dieser Mangel beseitigen, und es ist möglich, *alle* Nullstellen im Startintervall zu bestimmen. Die dabei durch die Division möglicherweise entstehenden erweiterten Intervallpaare werden durch die Schnittbildung auf ein Paar von Intervallen aus $I\mathbb{R}$ reduziert, die im Anschluß jeweils getrennt weiter iteriert werden können.

Der rekursiv formulierte Algorithmus 1.1 beschreibt das erweiterte Intervall-Newton-Verfahren im $I\!R^1$ unter Verwendung eines erweiterten Intervallpaares $W = (W^1 \,|\, W^2)$. Ist X die Starteinschließung für die Nullstellen der Funktion f, und gilt für die Lösungsmenge L zunächst $L = \{\,\}$, so liefert Newton (X, eps, L) als Ergebnis die Menge

$$L = \{X \in I\!I\!R \mid 0 \in F(X) \ \wedge \ d(X) < eps\}.$$

Algorithmus 1.1: Newton (X, eps, L)
Erweitertes Intervall-Newton-Verfahren

1. **if** $0 \notin F(X)$ **then return**.
2. $c := m(X)$; $z := f(c)$; $G := F'(X)$.
3. $W := (c - z/G) \cap X$.
4. **if** $W^1 = W^2 = \emptyset$ **then return**.
5. **if** $W^1 = X$ **then**
   $W^1 := [\underline{X}, c]$; $W^2 := [c, \overline{X}]$.
6. **if** $W^1 \neq \emptyset$ **then**
   if $d(W^1) < eps$ **then** $L := L \cup \{W^1\}$
   else Newton (W^1, eps, L).
7. **if** $W^2 \neq \emptyset$ **then**
   if $d(W^2) < eps$ **then** $L := L \cup \{W^2\}$
   else Newton (W^2, eps, L).

Ähnlich wie das klassische Newton-Verfahren kann auch das Intervall-Newton-Verfahren geometrisch dadurch interpretiert werden, daß in jedem Iterationsschritt an der Stelle $m(X)$ zwei Geraden an den Graphen der Funktion f angelegt werden. Es handelt sich dabei um die Geraden mit der Steigung $\underline{F'(X)}$ bzw. $\overline{F'(X)}$, also mit der kleinsten bzw. größten Steigung von f im Intervall X. Ihre Schnittpunkte mit der x-Achse entprechen den Intervallgrenzen der neuen Iterierten vor der Intervallschnittbildung mit der alten Iterierten X. Anhand von zwei Skizzen wollen wir diese geometrische Interpretation für den Fall $0 \notin F'(X)$ und für den Fall $0 \in F'(X)$ veranschaulichen.

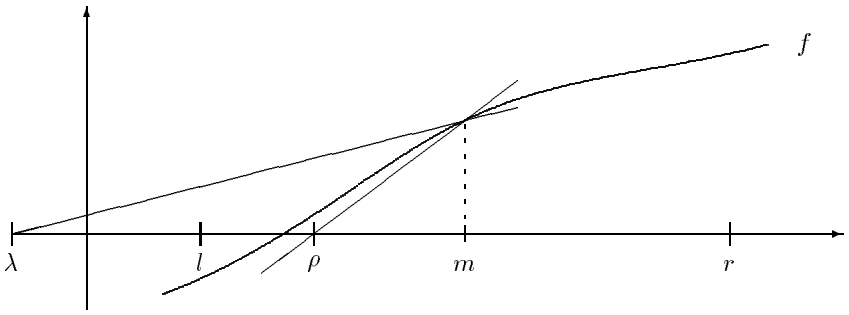


Abbildung 1.1: Intervall-Newton-Schritt mit $0 \notin F'(X)$

In Abbildung 1.1 ist für $X = [l, r]$ der Fall $0 \notin F'(X)$ skizziert. Die Gerade mit der kleinsten Steigung schneidet die x-Achse in λ, die Gerade mit der größten Steigung schneidet sie in ρ. In einem Newton-Schritt wird also zunächst das Intervall

$$Y := m - f(m)/F'(X) = [\lambda, \rho]$$

berechnet, das in der Skizze links vom Mittelpunkt liegt. Die im Anschluß daran erfolgende Schnittbildung liefert somit

$$X := Y \cap X = [\lambda, \rho] \cap [l, r] = [l, \rho]$$

als neue Iterierte.

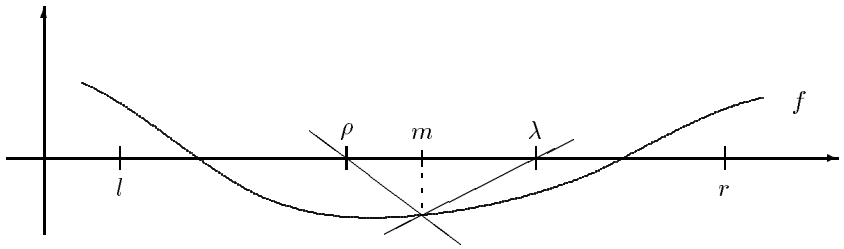


Abbildung 1.2: Intervall-Newton-Schritt mit $0 \in F'(X)$

In Abbildung 1.2 ist für $X = [l, r]$ der Fall $0 \in F'(X)$ skizziert. Die Gerade mit der kleinsten Steigung schneidet die x-Achse in ρ, die Gerade mit der größten Steigung schneidet sie in λ. In einem (erweiterten) Newton-Schritt wird also zunächst das erweiterte Intervallpaar

$$W := m - f(m)/F'(X) = (-\infty, \rho] \cup [\lambda, \infty)$$

berechnet und die anschließende Schnittbildung liefert

$$X := W \cap X = ((-\infty, \rho] \cup [\lambda, \infty)) \cap [l, r] = [l, \rho] \cup [\lambda, r],$$

also ein erweitertes Intervallpaar als neue Iterierte, deren Teilintervalle jeweils getrennt weiter zu iterieren wären.

Beispiel 1.7.1 Wir wollen abschließend noch eine Iteration nach Algorithmus 1.1 anhand der Funktion $f(x) = x^3 - 9x$ und dem Startintervall $[-5, 6]$ demonstrieren. Dazu geben wir nachfolgend das Ausgabeprotokoll einer entsprechenden Implementierung in PASCAL–XSC ([29], [30]) an. Darin kennzeichnet jedes Einrücken einen rekursiven Abstieg, und entsprechend sind die Intervalle, die mit der gleichen Spalte beginnend ausgegeben wurden, jeweils bei der erweiterten Division als Teilintervalle entstanden. Die Ausgabe `x = leer` steht für $x = \emptyset$. Die Iteration wurde abgebrochen, wenn der Durchmesser der jeweiligen Iterierten kleiner als 10^{-4} war.

```
x = [-5.000000000000000E+000, 6.000000000000000E+000]
  x = [-5.000000000000000E+000, 1.388888888888890E-002]
    x = [-5.000000000000000E+000,-2.598242630986108E+000]
      x = [-3.486365976874436E+000,-2.598242630986108E+000]
        x = [-3.013988939000460E+000,-2.973194901539887E+000]
          x = [-3.000154610537952E+000,-2.999891157776577E+000]
            x = [-3.000000003275874E+000,-2.999999997247139E+000]
    x = [-1.721683669064834E+000, 1.388888888888890E-002]
      x = [-6.917904286909904E-002, 1.388888888888890E-002]
        x = [-2.347528653315334E-006, 4.181990106690362E-005]
  x = [ 5.441919191919191E-001, 6.000000000000000E+000]
    x = [ 5.441919191919191E-001, 3.215689917901795E+000]
      x = [ 5.441919191919191E-001, 6.131789422418120E-001]
        x = leer
      x = [ 2.346539307642797E+000, 3.215689917901795E+000]
        x = [ 2.940919763903090E+000, 3.215689917901795E+000]
          x = [ 2.991850025436403E+000, 3.011773503665626E+000]
            x = [ 2.999983478176529E+000, 3.000019501050983E+000]
    x = [ 3.960520179512462E+000, 6.000000000000000E+000]
      x = [ 3.960520179512462E+000, 4.185280259525685E+000]
        x = leer

Die Nullstellen liegen in:
  x = [-3.000000003275874E+000,-2.999999997247139E+000]
  x = [-2.347528653315334E-006, 4.181990106690362E-005]
  x = [ 2.999983478176529E+000, 3.000019501050983E+000]
```

1.8 Differentiationsarithmetik

Im Rahmen unseres Optimierungsverfahrens müssen ständig die Werte der Ableitungen, also des Gradienten und der Hessematrix, der Zielfunktion berechnet werden, für die im allgemeinen keine expliziten formalen Darstellungen zur Verfügung stehen. Im Bereich des wissenschaftlichen Rechnens sind zur Lösung dieser Aufgabe drei Verfahren gebräuchlich: die numerische Differentiation, die symbolische Differentiation und die *automatische* oder auch *rekursive Differentiation*. Während bei der numerischen Differentiation die Werte der Ableitungen durch Differenzenquotienten approximiert werden, ermittelt die symbolische Differentiation zunächst durch Anwendung der Differentiationsregeln explizite, im allgemeinen recht umfangreiche Darstellungen für die Ableitungen, die dann für verschiedene Argumente ausgewertet werden können. Unter Verwendung einer sogenannten Differentiationsarithmetik kann auf dem Rechner die automatische Differentiation angewendet werden, bei der parallel zur Anwendung der Differentiationsregeln auch das Einsetzen der Argumente erfolgt, so daß lediglich Zahlenwerte und keine symbolischen Formeln bearbeitet werden müssen. Die Berechnung der Ableitungen erfolgt automatisch mit der Berechnung eines Funktionswertes.

Innerhalb der automatischen Differentiation unterscheidet man die sogenannte *Vorwärtsmethode*, die ausführlich z. B. in [47] behandelt wird, und die sogenannte *Rückwärtsmethode*, auch schnelle automatische Differentiation genannt. Letztere wird auf das Lösen spezieller Gleichungssysteme zurückgeführt und bringt einerseits einen deutlichen Laufzeitgewinn mit sich, erfordert andererseits jedoch einen erhöhten Speicherbedarf und eine aufwendigere Programmierung. Einen tieferen Einblick in die Rückwärtsmethode bietet z. B. [9]. Wir wollen im folgenden kurz die prinzipielle Methode der automatischen Differentiation anhand der in der Vorwärtsmethode verwendeten Differentiationsarithmetik erläutern. Wir behandeln dabei die Gradientenarithmetik und die Hessematrixarithmetik für Funktionen $f : I\!R^n \to I\!R$ mit dem Ziel, jeweils Intervalleinschließungen für die berechneten Werte zu erhalten. Somit werden alle Operationen unter Verwendung von Intervallarithmetik ausgeführt.

Die Gradientenarithmetik ist eine Arithmetik für Paare der Form

$$U = (u, u^g), \quad \text{mit } u \in I\!I\!R \text{ und } u^g \in I\!I\!R^n,$$

die jeweils mit u den Funktionswert der Funktion $U(x)$ und mit $u^g = \nabla U(x)$ den Wert des Gradienten an der festen Stelle $x \in I\!I\!R^n$ repräsentieren. Für die konstante Funktion $U(x) = c$ ist somit $U = (u, u^g) = (c, 0)$, und für die

für die Funktion $U(x) = x_i$ definiert man $U = (u, u^g) = (x_i, e^i)$, wobei e^i den i-ten Einheitsvektor bezeichnet.

Die Definition der arithmetischen Operationen für zwei Paare $U = (u, u^g)$ und $V = (v, v^g)$ erfolgt durch

$$\begin{aligned} U &+ V = (u+v,\ u^g + v^g), \\ U &- V = (u-v,\ u^g - v^g), \\ U &\cdot V = (u \cdot v,\ v \cdot u^g + u \cdot v^g), \\ U &/ V = (u/v,\ (v \cdot u^g - u \cdot v^g)/v^2), \quad v \neq 0. \end{aligned}$$

Für eine Elementarfunktion $S \in \mathcal{SF}$ mit $S : I\!R \to I\!R$ und $U = (u, u^g)$ definiert man

$$S(U) = (S(u),\ S'(u) \cdot u^g).$$

Dabei ist S' die explizit bekannte Ableitung der Funktion S.

Die Hessematrixarithmetik ist eine Arithmetik für Tripel der Form

$$U = (u, u^g, u^h), \quad \text{mit } u \in I\!R,\ u^g \in I\!R^n \text{ und } u^h \in I\!R^{n \times n},$$

die jeweils mit u den Funktionswert der Funktion $U(x)$, mit $u^g = \nabla U(x)$ den Wert des Gradienten und mit $u^h = \nabla^2 U(x)$ den Wert der Hessematrix an der festen Stelle $x \in I\!R^n$ repräsentieren. Für die konstante Funktion $U(x) = c$ ist somit $U = (u, u^g, u^h) = (c, 0, 0)$, und für die Funktion $U(x) = x_i$ definiert man $U = (u, u^g, u^h) = (x_i, e^i, 0)$.

Die arithmetischen Operationen für zwei Tripel $U = (u, u^g, u^h)$ und $V = (v, v^g, v^h)$ sind in den Komponenten w und w^g von $W = U \circ V$ für $\circ \in \{+, -, \cdot, /\}$ definiert wie in der Gradientenarithmetik, während sich die dritten Komponenten gemäß

$$\begin{aligned} W = U + V &\Longrightarrow w^h = u^h + v^h, \\ W = U - V &\Longrightarrow w^h = u^h - v^h, \\ W = U \cdot V &\Longrightarrow w^h = u \cdot v^h + u^g \cdot (v^g)^T + v^g \cdot (u^g)^T v \cdot u^h, \\ W = U / V &\Longrightarrow w^h = (u^h - w^g \cdot (v^g)^T - v^g \cdot (w^g)^T - w \cdot v^h)/v, \quad v \neq 0 \end{aligned}$$

berechnen. Für eine Elementarfunktion $S \in \mathcal{SF}$ mit $S : I\!R \to I\!R$ und $U = (u, u^g, u^h)$ definiert man

$$S(U) = (S(u),\ S'(u) \cdot u^g,\ S'(u) \cdot u^h + S''(u) \cdot u^g \cdot (u^g)^T).$$

Dabei sind S' und S'' die explizit bekannten ersten und zweiten Ableitungen der Funktion S.

In der praktischen Anwendung hat sich die automatische Differentiation bisher noch nicht durchsetzen können, was hauptsächlich darauf zurückzuführen ist, daß sich die Handhabung der Arithmetik ohne eine entsprechende programmiersprachliche Unterstützung äußerst schwierig gestaltet. Der Einsatz von Sprachen mit Operatorkonzept (z. B. PASCAL–XSC [29], [30]) erlaubt jedoch eine komfortable Programmierung und Anwendung der Differentiationsarithmetik (vgl. auch Abschnitt 3.2.2.2).

Kapitel 2

Globale Optimierung mit Ergebnisverifikation

Nachdem nun in den vorangegangenen Kapiteln das nötige Rüstzeug für die Entwicklung von Intervallverfahren zur globalen Optimierung bereitgestellt wurde, können wir uns in diesem Kapitel diesem eigentlichen Thema der Arbeit widmen.

Zunächst wollen wir uns nochmals unsere Zielsetzung im Hinblick auf die globale Optimierung ohne Nebenbedingungen vor Augen bringen und einen Überblick über die ersten Arbeiten und Ansätze auf diesem Gebiet geben. Anschließend wenden wir uns dann dem grundlegenden Algorithmus und den effizienzsteigernden zusätzlichen Verfahren zu.

Ein Schwerpunkt wird dabei auf dem aus der Literatur (z. B. [1], [2] oder [39]) bekannten Intervall-Newton-Verfahren liegen, das in verschiedenen Modifikationen eine wichtige Rolle in dem im folgenden entwickelten Verfahren spielen wird. Außerdem werden wir ausführlich auf die Problematik der Randbehandlung und die angewandten Methoden zur Ergebnisverifikation bzw. zum Nachweis der Eindeutigkeit der Lösung eingehen. Gerade diese Fragestellungen wurden bisher in der uns bekannten Literatur nur knapp oder überhaupt nicht behandelt.

Schließlich werden wir den vollständigen Algorithmus zur Lösung des globalen Optimierungsproblems mit automatischer Ergebnisverifikation entwickeln und das Kapitel mit einigen Hinweisen für mögliche Modifikationen dieses Verfahrens abschließen.

Ein wichtiges Augenmerk soll im folgenden stets darauf liegen, die Algorithmen in einer leicht auf den Rechner übertragbaren Form darzustellen.

2.1 Zielsetzung und Einführung

Das Intervallverfahren zur Lösung des globalen Optimierungsproblems

$$\min_{x \in X} f(x),$$

wobei $X \in I\mathbb{R}^n$, mit dem wir uns in den folgenden Abschnitten beschäftigen wollen, stellt eine Methode zur Bestimmung von garantierten Schranken – also von Einschließungen – für alle globalen Minimalstellen $x^* \in X$ und für den Wert des globalen Minimums $f^* = f(x^*)$ der Funtion $f : \mathbb{R}^n \to \mathbb{R}$ auf dem Rechner dar. Außerdem soll das Verfahren, sofern möglich, die Eindeutigkeit der Minimalstelle innerhalb der berechneten Einschließung automatisch nachweisen. Die einzige Einschränkung, die im Hinblick auf das Auffinden und eindeutige Einschließen aller Minimalstellen gemacht werden muß, ergibt sich allein aus der Ausstattung der Zielmaschine, auf der der Algorithmus implementiert werden soll. Unter Ausstattung verstehen wir in diesem Zusammenhang die Speicherkapazität sowie das interne Datenformat, speziell die Anzahl der signifikanten Mantissenstellen.

Als Grundlage zur Einschließung der globalen Minimalstellen werden wir ein spezielles Bisektions- oder auch Halbierungsverfahren verwenden. Dieses Verfahren ist als reines Näherungsverfahren wegen der nur linearen Konvergenz anderen Verfahren weit unterlegen, es kann jedoch im Zusammenhang mit Intervallarithmetik als echtes Einschließungsverfahren auch praktisch angewendet werden. Ein erster erfolgreicher Ansatz eines solchen Verfahrens zur Bestimmung des Minimalpunktes einer streng konvexen Funktion wurde bereits 1972 von Dussel [7] beschrieben.

Ein erster Algorithmus für die globale Optimierung (der Moore-Skelboe-Algorithmus) geht auf Skelboe [55] zurück, der ein von Moore [38] eingeführtes Verfahren zur Berechnung des Wertebereichs von Funktionen unter Zuhilfenahme von Intervallarithmetik mit einem *Branch-and-Bound*-Prinzip kombinierte. Dieses Verfahren zielte lediglich darauf ab, das globale Minimum f^* zu berechnen. Ein ähnliches Verfahren wurde auch von Ichida und Fujii [18] entwickelt.

Die vorliegende Arbeit verwendet den von Hansen 1979 für den eindimensionalen Fall [12] und 1980 für den mehrdimensionalen Fall [13] entwickelten Algorithmus als Ansatz. Auch dieser Algorithmus arbeitet nach dem Branch-and-Bound-Prinzip. Das bedeutet, daß das Optimierungsgebiet X nicht gleichmäßig nach Minimierern durchsucht, sondern in Teilbereiche (*branches*) aufgeteilt wird. In Abhängigkeit von den für f berechneten Schranken auf einem Teilbereich kann dann entschieden werden, ob dieser

Bereich weiter bearbeitet werden muß oder als Lösungsgebiet ausscheidet. Die wichtigsten Bestandteile des Optimierungsalgorithmus sind

- der grundlegende Algorithmus (Bisektionsverfahren),
- die beschleunigenden Phasen
  1. Mittelpunktstest,
  2. Monotonietest,
  3. Konkavitätstest,
  4. Intervall-Newton-Schritt,
- die Randbehandlung und
- der Verifikationsschritt.

Während der Grundalgorithmus abgesehen von der Stetigkeit keine Anforderungen an die Funktion f stellt, werden wir im Hinblick auf die beschleunigenden Phasen sowie auf den Eindeutigkeitstest mindestens $f \in C^2(X)$ fordern. Die einzelnen Teile des Verfahrens werden wir in den folgenden Abschnitten ausführlich behandeln, wobei wir sowohl bekannte Formen vorstellen als auch Modifikationen entwickeln werden.

2.2 Grundlegender Bisektions-Algorithmus

Für die einfache Beschreibung der Algorithmen in diesem und in den nachfolgenden Abschnitten wollen wir zunächst noch einige Begriffe und abkürzende Schreibweisen einführen.

2.2.1 Bezeichnungen

- Einen Intervallvektor $Y \in I\!I\!R^n$ bzw. Teilbereiche eines solchen Intervallvektors nennen wir auch kurz einen (n-dimensionalen) *Quader* oder eine *Box*. Dabei sprechen wir von einem *degenerierten* Quader Y wenn für den Durchmesser einer Komponente Y_i gilt $d(Y_i) = 0$, also wenn Y_i ein Punktintervall ist.

- Mit $\mathsf{Branch}\,(Y, i, V^1, V^2)$ wollen wir die Bisektion von Y in die Quader V^1 und V^2 bezüglich der i-ten Komponente laut Algorithmus 2.1 bezeichnen.

Algorithmus 2.1: Branch (Y, i, V^1, V^2)
1. $V^1 := Y$. 2. $V^2 := Y$. 3. $\overline{V_i^1} := m(Y_i)$. 4. $\underline{V_i^2} := m(Y_i)$.

- Zur Anordnung der durch die Branch-and-Bound-Methode entstehenden Teilquader verwenden wir eine geordnete *Liste*, die wir mit dem Buchstaben L bezeichnen. Für eine solche Liste L und ein Listenelement E seien die folgenden Operationen definiert:

Operation	Bedeutung
$L := \{\,\}$	Initialisierung mit der leeren Liste
$L := E$	Initialisierung mit nur einem Element
$L := L + E$	Einhängen des Elements E in L
$L := L - E$	Aushängen des Elements E aus L
$E :=$ Head (L)	Zuweisung des 1. Elements von L an E

  Zu beachten ist dabei, daß die Operation + verschiedene Varianten in Abhängigkeit der für die Listenorganisation festgelegten Ordnung realisieren kann.

- Als Elemente der Liste verwenden wir Paare $E = (Y, r)$ bestehend aus einem Intervallvektor $Y \in I\mathbb{R}^n$ und einer reellen Zahl $r \in \mathbb{R}$. Dabei verwenden wir im folgenden (mit Ausnahme von Abschnitt 2.7) stets

$$r = \underline{F_Y} = \underline{F(Y)} = \inf F(Y).$$

2.2.2 Der Hansen-Algorithmus

Die Struktur unseres grundlegenden Algorithmus wollen wir am sogenannten Hansen-Algorithmus, wie er in [49] dargestellt ist, verdeutlichen. Wir werden jedoch den für die Konvergenz des Verfahrens notwendigen Eliminationsschritt für das jeweils halbierte Intervall, der in [49] nicht angegeben wurde, ergänzen. Den dort verwendeten Mittelpunktstest werden wir erst in Abschnitt 2.2.3 mit in den Algorithmus einbeziehen. Außerdem wollen wir einige der wichtigsten von Moore, Ratschek und Rokne (1988) hergeleiteten Konvergenzeigenschaften des Hansen-Algorithmus aus [41] und [49] zitieren.

Algorithmus 2.2 zur Bestimmung des globalen Minimums von f in X wollen wir zwar als Hansen-Algorithmus bezeichnen, seine Struktur unterscheidet sich jedoch nicht von der des Moore-Skelboe- oder des Ichida-Fujii-Algorithmus (jeweils ohne Mittelpunktstest).

Algorithmus 2.2:
Hansen-Algorithmus ohne Mittelpunktstest

1. $Y := X$.
2. Berechne $\underline{F_Y} := \inf F(Y)$.
3. Initialisiere $L := (Y, \underline{F_Y})$.
4. Wähle $k \in \{i : d(Y_i) = d(Y)\}$.
5. Branch (Y, k, V^1, V^2).
6. $L := L - (Y, \underline{F_Y})$.
7. Berechne $\underline{F_{V^1}} := \inf F(V^1)$ und $\underline{F_{V^2}} := \inf F(V^2)$.
8. $L := L + (V^1, \underline{F_{V^1}}) + (V^2, \underline{F_{V^2}})$.
9. $(Y, \underline{F_Y}) :=$ Head (L).
10. **if not** genau genug **then goto** 4.

Bei Schritt 6 in Algorithmus 2.2 handelt es sich dabei um den in [49, Algor. 3] fehlenden Eliminationsschritt.

Der Unterschied zwischen den Varianten von Skelboe, Ichida-Fujii und Hansen liegt allein in der jeweils verwendeten Listenanordnung. Während die erstgenannten Verfahren die Listenelemente monoton steigend bezüglich ihrer $\underline{F_Y}$-Komponente anordnen, verwendet Hansen [13] eine Ordnung bezüglich des Alters oder bezüglich des Durchmessers $d(Y)$ der Quader. Im ersten Fall bedeutet dies, daß der Operator + einen neuen Quader stets ans Ende der Liste anhängt. Im zweiten Fall fügt + den neuen Quader so in L ein, daß die Quaderdurchmesser zum Ende der Liste hin monoton fallen. In beiden Fällen ergibt sich eine gleichmäßige Unterteilung aller Quader in der Liste. Genau diese Tatsache ermöglicht auch in Verbindung mit dem im nächsten Abschnitt beschriebenen Mittelpunktstest eine Konvergenzaussage, die über die für den Moore-Skelboe-Algorithmus oder den Ichida-Fujii-Algorithmus gemachte hinausgeht, bei denen die Listenelemente monoton steigend bezüglich der $\underline{F_Y}$-Komponente angeordnet werden.

Wir wollen nun kurz mit einem Satz aus [49], den wir ohne Beweis angeben wollen, auf die Konvergenzgeschwindigkeit dieses Basisalgorithmus eingehen. Dabei bezeichne $(Y^k, \underline{F_{Y^k}})$ das Listenelement der Liste L nach k Iterationen, für das gilt

$$\underline{F_{Y^k}} \leq \underline{F_Z} \quad \text{für alle } (Z, \underline{F_Z}) \in L.$$

Satz 2.2.1 *Sei F die isotone Intervallerweiterung von f, und es gelte $d(F(Y)) - d(f(Y)) \leq c(d(Y))^\alpha$ für alle $Y \in I\!I\!R^n$ und $c, \alpha > 0$. Dann gilt für $(Y^k, \underline{F_{Y^k}})$*

$$f^* - \underline{F_{Y^k}} = O\left(k^{-\alpha/n}\right).$$

Wie man leicht sieht, gilt für jeden Iterationsschritt in Algorithmus 2.2, daß jeweils eine Box halbiert und aus der Liste entfernt wird, an deren Stelle aber ihre Teile in die Liste aufgenommen werden. Somit decken die Elemente der Liste zu jedem Zeitpunkt den gesamten Startbereich X ab, und die Anzahl der Listenelemente nimmt mit jeder Iteration zu.

2.2.3 Der Mittelpunktstest

Der Mittelpunktstest wird zur Verkleinerung der Anzahl der Quader in der Liste benutzt, um damit dem zuletzt geschilderten Sachverhalt entgegenzutreten. Dies kann unter Verwendung einer bekannten Oberschranke $\widetilde{f}$ für das Minimum f^* durch Algorithmus 2.3 in jedem Iterationsschritt von Algorithmus 2.2 erfolgen.

Algorithmus 2.3: MidTest $(L, \widetilde{f})$
Mittelpunktstest

for all $(Z, \underline{F_Z}) \in L$ **do**
  if $\widetilde{f} < \underline{F_Z}$ **then**
    $L := L - (Z, \underline{F_Z})$;

Der Name Mittelpunktstest rührt daher, daß die Oberschranke $\widetilde{f}$ üblicherweise durch eine Funktionsauswertung im Mittelpunkt $c = m(Y)$ bestimmt wird. Man bestimmt zunächst ein Element $(Y, \underline{F_Y}) \in L$ so, daß gilt $\underline{F_Y} \leq \underline{F_Z}$ für alle $(Z, \underline{F_Z}) \in L$ und berechnet anschließend $\widetilde{f} := \sup F(c)$. Dabei muß c nicht notwendigerweise als der Mittelpunkt von Y gewählt werden, vielmehr ist jedes $c \in Y$ möglich.

Lemma 2.2.2 *Für jedes $c \in Y$ löscht der mit $\widetilde{f} = \sup F(c)$ durchgeführte Mittelpunktstest keine Elemente aus L, die globale Minimalstellen enthalten.*

Beweis: Ein Element $(Z, \underline{F_Z})$ aus L wird vom Mittelpunktstest gelöscht, wenn gilt

$$\widetilde{f} < \underline{F_Z} = \inf f(Z).$$

Somit können Minimalstellen nur gelöscht werden, falls gilt $\widetilde{f} < f^*$, was wegen

$$f^* = f(x^*) \leq f(c) \leq \sup F(c) = \widetilde{f}$$

für kein $c \in Y$ erfüllt sein kann. □

Abbildung 2.1 verdeutlicht nochmals die Arbeitsweise des Mittelpunktstests, der im skizzierten Fall ausgehend von $Y = Y^2$ die Quader Y^3, Y^4, Y^7 und Y^8 löscht.

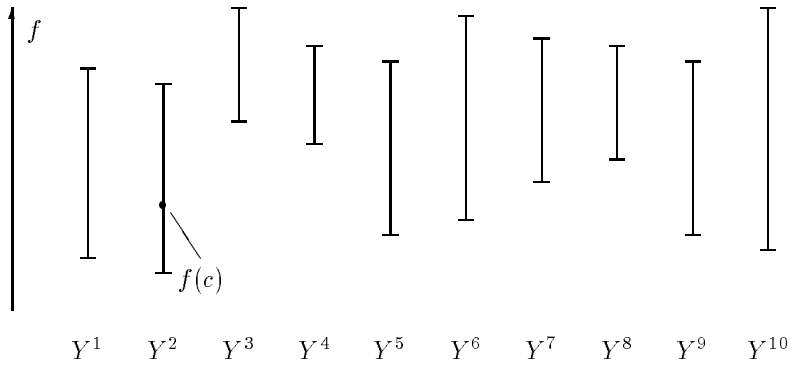


Abbildung 2.1: Mittelpunktstest

Unter Verwendung des Mittelpunktstests können wir die eigentliche Form des Hansen-Algorithmus (Algorithmus 2.4) formulieren.

Algorithmus 2.4:
Hansen-Algorithmus

1. $Y := X$.
2. Berechne $\underline{F_Y} := \inf F(Y)$.
3. Initialisiere $L := (Y, \underline{F_Y})$.
4. Wähle $k \in \{i : d(Y_i) = d(Y)\}$.
5. Branch (Y, k, V^1, V^2).
6. $L := L - (Y, \underline{F_Y})$.
7. Berechne $\underline{F_{V^1}} := \inf F(V^1)$ und $\underline{F_{V^2}} := \inf F(V^2)$.
8. $L := L + (V^1, \underline{F_{V^1}}) + (V^2, \underline{F_{V^2}})$.
9. Wähle $(Y, \underline{F_Y}) \in L$ so, daß gilt $\underline{F_Y} \leq \underline{F_Z}$ für alle $(Z, \underline{F_Z}) \in L$ und berechne $\widetilde{f} := \sup F(m(Y))$.
10. MidTest $(L, \widetilde{f})$.
11. $(Y, \underline{F_Y}) :=$ Head (L).
12. **if not** genau genug **then goto** 4.

Wiederum bleibt anzumerken, daß die Operation + die Listenelemente nach dem Alter der Elemente oder nach dem Durchmesser der Elemente ordnet. Verwendet man die Ordnung bezüglich der $\underline{F_Y}$-Komponenten, so entspricht dies dem Ichida-Fujii-Algorithmus.

2.2.4 Konvergenzeigenschaften

In diesem Abschnitt wollen wir die wichtigsten Resultate bezüglich der Konvergenzeigenschaften von Algorithmus 2.4 aus [49] zusammenfassen. Zunächst geben wir einen schwachen Konvergenzsatz für den Algorithmus mit der Listenanordnung bezüglich der $\underline{F_Y}$-Komponenten (Ichida-Fujii) und erst danach den stärkeren Satz für den Hansen-Algorithmus.

Im folgenden bezeichne L^k die k-te im Algorithmus gebildete Liste und $\mathcal{U}^k \in I\!P\,I\!R^n$ die mengentheoretische Vereinigung aller Quader $Y \in L^k$.

Satz 2.2.3 *Es seien (L^k) und $(\mathcal{U}^k)$ die durch den Ichida-Fujii-Algorithmus generierten Folgen. Ferner gelte für die Intervallerweiterung F von f, daß $d(F(Y)) \longrightarrow 0$ wenn $d(Y) \longrightarrow 0$. Dann existiert eine Menge $D \supseteq \mathcal{X}^*$, so daß $\mathcal{U}^k \supseteq D$ für alle k und $\mathcal{U}^k \longrightarrow D$ für $k \longrightarrow \infty$. Für die Folge $(\mathcal{U}^k)$ gilt*

$\mathcal{U}^i \supseteq \mathcal{U}^{i+1}$ *für* $i = 1, 2, \ldots$, *so daß*

$$D = \bigcap_{k=1}^{\infty} \mathcal{U}^k.$$

Beweis: Da ohne den Mittelpunktstest stets die Beziehung $\mathcal{U}^k = X$ gelten würde und aufgrund von Lemma 2.2.2 keine Box eliminiert wird, die eine globale Minimalstelle enthält, folgt sofort, daß $\mathcal{X}^* \subseteq \mathcal{U}^k$. Damit existiert aber auch ein $D \supseteq \mathcal{X}^*$ mit $D \subseteq \mathcal{U}^k$. Aufgrund der Elimination durch den Mittelpunktstest gilt $\mathcal{U}^i \supseteq \mathcal{U}^{i+1}$ und wegen der Konvergenzeigenschaft des Durchmessers folgt sofort die Konvergenz der $\mathcal{U}^k$ gegen D. □

Bemerkung: Die Menge D kann die Menge der Minimalstellen $\mathcal{X}^*$ trotz des Eliminationsprozesses tatsächlich überschätzen. Theoretisch ist es möglich, daß es mit der $\underline{F_Y}$-Ordnung eine Box Z in der Liste gibt, die zwar nicht am Kopf der Liste steht, die aber die Beziehung $f^* = \underline{F_Z}$ erfüllt. In diesem Fall würde Z praktisch nie zum Kopfelement der Liste und damit auch nicht halbiert werden. Auch der Mittelpunktstest würde diese Box nie aus L eliminieren.

Mir der von Hansen vorgeschlagenen Listenanordnung ist dieses Problem beseitigt, und wir erhalten den nachfolgenden Satz.

Satz 2.2.4 *Es seien* (L^k) *und* $(\mathcal{U}^k)$ *die durch den Hansen-Algorithmus generierten Folgen. Ferner gelte für die Intervallerweiterung* F *von* f, *daß* $d(F(Y)) \longrightarrow 0$ *wenn* $d(Y) \longrightarrow 0$. *Dann gilt* $\mathcal{U}^k \supseteq \mathcal{X}^*$ *für alle* k *und* $\mathcal{U}^k \longrightarrow \mathcal{X}^*$ *für* $k \longrightarrow \infty$. *Für die Folge* $(\mathcal{U}^k)$ *gilt* $\mathcal{U}^i \supseteq \mathcal{U}^{i+1}$ *für* $i = 1, 2, \ldots$, *so daß*

$$\mathcal{X}^* = \bigcap_{k=1}^{\infty} \mathcal{U}^k.$$

Beweis: Nach Satz 2.2.3 folgt sofort $\mathcal{X}^* \subseteq \bigcap_{k=1}^{\infty} \mathcal{U}^k$. Zu zeigen bleibt also $\bigcap_{k=1}^{\infty} \mathcal{U}^k \subseteq \mathcal{X}^*$.

Bezeichnet d_k den maximalen Durchmesser der Quader in der Liste L^k, so gilt $d_k \longrightarrow 0$ für $k \longrightarrow \infty$. Sei nun $x \in \mathcal{U}^k$ für alle k. Somit existiert in L^k ein Element $(Z^k, \underline{F_{Z^k}})$ mit $x \in Z^k$, für das gilt:

$$f(m(Y^k)) \geq \underline{F_{Z^k}}, \tag{2.1}$$

wobei $(Y^k, \underline{F_{Y^k}})$ das Element mit kleinstem $\underline{F_Y}$-Wert ist. Mit $Z^k \longrightarrow x$ gilt auch $d(F(Z^k)) \longrightarrow 0$ für $k \longrightarrow \infty$. Wegen $f(Z^k) \longrightarrow f(x)$ folgt

$f(x) \in f(Z^k) \subseteq F(Z^k)$ und

$$F(Z^k) \longrightarrow f(x) \quad \text{für} \quad k \longrightarrow \infty. \tag{2.2}$$

Mit (2.1) und (2.2) gilt

$$\underline{F_{Y^k}} \leq \underline{F_{Z^k}} \leq f(m(Y^k)) \in F(Y^k),$$

und wegen $F(Y^k) \longrightarrow f^*$ und $\underline{F_{Y^k}} \longrightarrow f^*$ ergibt sich $\underline{F_{Z^k}} \longrightarrow f^*$. Mit $d(F(Z^k)) \longrightarrow 0$ ergibt sich schließlich $F(Z^k) \longrightarrow f^*$. □

Auf die Frage der Abbruchbedingung wollen wir im nächsten Abschnitt im Rahmen einiger Modifikationen eingehen.

2.2.5 Modifikationen des Grundalgorithmus

In diesem Abschnitt wollen wir einige Modifikationen vorstellen, die schließlich zum grundlegenden Algorithmus 2.7 führen, wie wir ihn für unseren vollständigen globalen Optimierungsalgorithmus verwenden wollen.

2.2.5.1 Listenordnung, Abbruchbedingung und Mittelpunktstest

Zunächst sollte bemerkt werden, daß die Listenordnungsvariante von Skelboe bzw. Ichida-Fujii nicht unbedingt verworfen werden sollte, da mit einer geringfügigen Modifikation des Algorithmus auch mit dieser Ordnung dafür gesorgt werden kann, daß es nicht zu dem im letzten Abschnitt erläuterten schlechtesten (theoretischen) Fall kommt. Die Modifikation besteht darin, mit einer zusätzlichen Liste $\widehat{L}$ zu arbeiten und alle führenden Listenelemente von L, die bereits eine bestimmte Genauigkeitsanforderung erfüllen, in diese zweite Liste umzuspeichern.

In unserem Algorithmus wollen wir dieses Verfahren anwenden und entsprechend eine etwas andere Ordnung als im Hansen-Algorithmus verwenden. Dazu definieren wir für den Operator $+$, daß nach dem Einfügen eines neuen Listenelements $(Y, \underline{F_Y})$ gilt

- $(Y, \underline{F_Y})$ ist letztes Element der Liste *oder*
- $\underline{F_W} \leq \underline{F_Y} < \underline{F_Z}$, wobei $(W, \underline{F_W})$ der Vorgänger und $(Z, \underline{F_Z})$ der Nachfolger von $(Y, \underline{F_Y})$ in der Liste ist.

Damit werden grundsätzlich alle Elemente $(Y, \underline{F_Y})$ so in die Liste eingetragen, daß die $\underline{F_Y}$-Komponenten monoton steigen, und es wird gleichzeitig

dafür gesorgt, daß ein neues Element hinter allen bereits existierenden Elementen mit gleicher $\underline{F_Y}$-Komponente eingefügt wird.

Die Verwendung dieser Ordnung bringt den Vorteil mit sich, daß der Aufwand für die Bestimmung des Elements $(Y, \underline{F_Y})$ mit dem kleinsten $\underline{F_Y}$ zur Berechnung von $\widetilde{f}$ gespart und der Aufwand für die Durchführung von $\mathsf{MidTest}\,(L, \widetilde{f})$ aufgrund der vorsortierten Liste verringert werden kann.

Eine Genauigkeitsanforderung für das Umspeichern von Quadern erhalten wir leicht aus der üblichen Abbruchbedingung für den Algorithmus. Im Gegensatz zu der von Hansen ([13], [14]) verwendeten absoluten Genauigkeitsanforderung verwenden wir in unserem Algorithmus eine relative Toleranzangabe, die sich sowohl auf den Durchmesser des Quaders selbst als auch auf den Durchmesser des entsprechenden Funktionsintervalls bezieht.

Definition 2.2.1 *Sei $A \in I\!I\!R$ und $Y \in I\!I\!R^n$. Dann heißt d_{rel} mit*

$$d_{\mathrm{rel}}(A) = \begin{cases} d(A)/\langle A \rangle & \text{für } 0 \notin A \\ d(A) & \text{für } 0 \in A \end{cases}$$

und mit $d_{\mathrm{rel}}(Y) = \max\limits_{1 \le i \le n} d_{\mathrm{rel}}(Y_i)$ relativer Durchmesser von A bzw. Y.

Algorithmus 2.5: $\mathsf{NextY}\,(L, \widehat{L}, \widetilde{f}, \varepsilon, newy)$
Neues Y und Mittelpunktstest

1. **while** $L \neq \{\,\}$ **do**
   (a) $(Y, \underline{F_Y}) := \mathsf{Head}\,(L)$.
   (b) $L := L - (Y, \underline{F_Y})$.
   (c) $c := m(Y)$; $\overline{F_c} := \sup(F(c))$.
   (d) **if** $\overline{F_c} < \widetilde{f}$ **then** $\widetilde{f} := \overline{F_c}$.
   (e) $\mathsf{MidTest}\,(L, \widetilde{f})$.
   (f) **if** $d_{\mathrm{rel}}(Y) < \varepsilon$ **and** $d_{\mathrm{rel}}(F(Y)) < \varepsilon$ **then**
   $newy := false$; $\widehat{L} := \widehat{L} + (Y, \underline{F_Y})$.
   else
   $newy := true$; **exit**$_{while\text{-}\mathrm{loop}}$.
2. $\mathsf{NextY} := Y$.

Zusammenfassend kommen wir somit zum Algorithmus 2.5, der mit $(Y, \underline{F_Y}) := \mathsf{NextY}\,(L, \widehat{L}, \widetilde{f}, \varepsilon, newy)$ sowohl den Mittelpunktstest durchführt, als auch entweder die nächste zu behandelnde Box Y liefert oder durch $newy = false$ anzeigt, daß keine solche Box mehr existiert. Den Mittelpunktstest verwenden wir auch in abgewandelter Form, indem wir eine Box V nur dann in die Liste aufnehmen, wenn für die aktuelle garantierte Oberschranke $\widetilde{f}$ des globalen Minimums gilt $\underline{F_V} \leq \widetilde{f}$.

2.2.5.2 Auswahl der Halbierungsrichtung

Die Halbierung der Quader zielt zunächst einmal darauf ab, eine verbesserte Einschließung für den Wertebereich der Funktion zu erhalten. In diesem Zusammenhang ist die von Hansen verwendete Wahl der Komponente mit größtem Durchmesser als Halbierungsrichtung nicht unbedingt optimal.

Beispiel 2.2.1 Betrachtet man die Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ mit $f(x) = x_1 \cdot x_2$ und ein Ausgangsintervall X mit den Komponenten $X_1 = [100, 200]$ und $X_2 = [0, 1]$ so würde die nach Hansen gewählte Halbierung die Intervallvektoren

$$V^1 = \begin{pmatrix} [100, 150] \\ [0, 1] \end{pmatrix} \quad \text{und} \quad V^2 = \begin{pmatrix} [150, 200] \\ [0, 1] \end{pmatrix}$$

und damit $F(V^1) = [0, 150]$ und $F(V^2) = [0, 200]$ ergeben. Viel günstiger wäre jedoch die Teilung in der zweiten Komponente, so daß sich

$$V^1 = \begin{pmatrix} [100, 200] \\ [0, 0.5] \end{pmatrix} \quad \text{und} \quad V^2 = \begin{pmatrix} [100, 200] \\ [0.5, 1] \end{pmatrix}$$

und damit $F(V^1) = [0, 100]$ und $F(V^2) = [50, 200]$ ergeben. Somit wäre diese zweite Variante vorzuziehen, da sie den kleineren Durchmesser für die Funktionsintervalle mit sich bringt.

Zu einer allgemeinen Vorgehensweise für unseren Halbierungsschritt kommen wir, wenn wir uns von der bekannten Formel für die Fehlerfortpflanzung

$$f(\tilde{x}) - f(x) \doteq \boldsymbol{\nabla} f(x) \cdot (\tilde{x} - x)$$

mit $f : \mathbb{R}^n \to \mathbb{R}$ (vgl. etwa [56]) inspirieren lassen. Das Zeichen „$\doteq$“ deutet dabei an, daß diese Formel nur in erster Näherung gilt. Diese Formel bringt zum Ausdruck, daß die i-te Komponente des Gradienten die Empfindlichkeit mißt, mit der das Ergebnis auf Änderungen in x_i reagiert. Ganz entsprechend kann man als erste Näherung für unsere Intervallauswertung der Funktion f auch die *Mittelwertform* oder *spezielle zentrierte Form*

$$F(X) - F(c) \doteq \boldsymbol{\nabla} F(X) \cdot (X - c)$$

mit $c = m(X)$ verwenden. Für den Durchmesser von $F(X)$ erhalten wir

$$\begin{aligned} d(F(X)) &\approx d(F(X) - F(c)) \\ &\approx d\left(\nabla F(X) \cdot (X - c)\right) \\ &= d\left(\sum_{i=1}^{n} \frac{\partial F(X)}{\partial x_i} \cdot (X_i - c_i)\right) \\ &= \sum_{i=1}^{n} d\left(\frac{\partial F(X)}{\partial x_i} \cdot (X_i - c_i)\right). \end{aligned}$$

Somit sollte für die Halbierung von X die Komponente i gewählt werden, für die der Durchmesser $d_i = d(\frac{\partial F(X)}{\partial x_i} \cdot (X_i - c_i))$ am größten ist.

Beispiel 2.2.2 Wenden wir dieses Verfahren auf Beispiel 2.2.1 an, so erhalten wir zunächst

$$\nabla F(X) = \begin{pmatrix} X_2 \\ X_1 \end{pmatrix} = \begin{pmatrix} [0,1] \\ [100,200] \end{pmatrix} \quad \text{und} \quad X - c = \begin{pmatrix} [-50,50] \\ [-0.5,0.5] \end{pmatrix}.$$

Damit ergibt sich

$$\begin{aligned} d_1 &= d([0,1] \cdot [-50,50]) = d([-50,50]) = 100 \quad \text{und} \\ d_2 &= d([100,200] \cdot [-0.5,0.5]) = d([-100,100]) = 200, \end{aligned}$$

wonach die zweite Komponente als Halbierungsrichtung zu bevorzugen ist.

Zur Bestimmung der optimalen Komponente $k = \mathsf{OptComp}\,(Y)$ zur Halbierung unserer aktuellen Box Y verwenden wir somit Algorithmus 2.6.

Algorithmus 2.6: $\mathsf{OptComp}\,(Y)$
Optimale Halbierungsrichtung

1. $G := \nabla F(Y); \quad c := m(Y).$
2. **for** $i := 1$ **to** n **do**
   $d_i := d(G_i \cdot (Y_i - c_i)).$
3. $k := 1.$
4. **for** $i := 2$ **to** n **do**
   if $d_i > d_k$ **then**
   $k := i.$
5. $\mathsf{OptComp} := k.$

2.2.6 Verwendeter Grundalgorithmus

Unter Verwendung aller beschriebenen Modifikationen können wir jetzt mit Algorithmus 2.7 den grundlegenden Algorithmus angeben, auf dem unser Optimierungsverfahren aufbaut. Eingabedaten sind jeweils f, X und die Toleranz ε. Als Operator + für das Einhängen der Paare $(Y, \underline{F_Y})$ bzw. $(V^i, \underline{F_{V^i}})$ in die Liste verwenden wir den in Abschnitt 2.2.5.1 definierten Operator.

Algorithmus 2.7:
Verwendeter Grundalgorithmus

1. $Y := X$; $\widetilde{f} := \sup(F(m(Y)))$.
2. $L := \{\}$; $\widehat{L} := \{\}$.
3. **repeat**
   (a) $k :=$ OptComp(Y).
   (b) Branch (Y, k, V^1, V^2).
   (c) **for** $i := 1$ **to** 2 **do**
   if $\underline{F_{V^i}} \leq \widetilde{f}$ **then** $L := L + (V^i, \underline{F_{V^i}})$.
   (d) $Y :=$ NextY $(L, \widehat{L}, \widetilde{f}, \varepsilon, newy)$.

   until not *newy*;

Aufgrund der langsamen Konvergenz dieses Grundalgorithmus (vgl. auch Satz 2.2.1) ist es notwendig, diejenigen Quader, die garantiert kein globales Minimum enthalten können, frühzeitig zu eliminieren oder gar nicht erst in die Liste aufzunehmen.

2.3 Der Monotonietest

Der Monotonietest wird verwendet, um festzustellen, ob die zu minimierende Funktion f, für die wir jetzt $f \in C^1(X)$ fordern müssen, auf einem ganzen Quader $Y \subseteq X$ streng monoton ist. Ist dies der Fall, so braucht Y nicht mehr weiter auf eine Minimalstelle untersucht zu werden, da diese nicht in Y liegen kann, außer wenn der den kleinsten Funktionswert enthaltende Rand von Y mit dem Rand von X zusammenfällt. In diesem Fall kann aber

Y zumindest auf diesen Rand, der möglicherweise eine globale Minimalstelle enthält, reduziert werden.

Wir wollen nun mit Algorithmus 2.8 zunächst den Monotonietest in der aus [13] oder [49] bekannten Form betrachten, die nur auf strenge Monotonie überprüft. Wir bezeichnen diese Form mit MonoTest (X, Y, del), wobei $del = true$ anzeigt, daß die Box Y nicht mehr weiterbearbeitet werden muß, also gelöscht werden kann.

Algorithmus 2.8: MonoTest (X, Y, del)
Monotonietest

1. $G := \nabla F(Y)$.
2. $del := false$.
3. **for** $i := 1$ **to** n **do**
   (a) **if** $0 < \underline{G_i}$ **then**
   if $\underline{X_i} = \underline{Y_i}$ **then** $Y_i := [\underline{Y_i}, \underline{Y_i}]$
   else $del := true$; **exit**$_{i\text{-loop}}$.
   (b) **else if** $\overline{G_i} < 0$ **then**
   if $\overline{X_i} = \overline{Y_i}$ **then** $Y_i := [\overline{Y_i}, \overline{Y_i}]$
   else $del := true$; **exit**$_{i\text{-loop}}$.

Wir wollen nun zeigen, daß der Monotonietest in dieser Form keine Minimalstellen aus der Liste entfernt.

Satz 2.3.1 *Sei $f : \mathbb{R}^n \to \mathbb{R}$ aus $C^1(X)$. Für jeden Quader $Y \subseteq X$ mit $x^* \in Y$ für eine globale Minimalstelle $x^* \in X$ der Funktion f gilt auch nach Ausführung des Monotonietests* MonoTest (X, Y, del) *(Algorithmus 2.8) die Beziehung $x^* \in Y$.*

Beweis: Angenommen Y sei ein Quader mit $x^* \in Y$, für den nach Ausführung des Monotonietests gilt $del = true$ oder $x^* \notin Y^M$, wenn Y^M der reduzierte Ergebnisquader nach dem Test ist. Somit muß für mindestens ein i mit $1 \leq i \leq n$ entweder

$$0 < \underline{G_i} \tag{2.3}$$

oder

$$\overline{G_i} < 0 \tag{2.4}$$

gelten.

Fall 1: x^* ist stationäre Stelle. Es gilt also

$$0 = \nabla f(x^*) \in \nabla f(Y) \subseteq \nabla F(Y) = G \tag{2.5}$$

und damit $0 \in G_i$ bzw. $\underline{G_i} \leq 0 \leq \overline{G_i}$ für alle i im Widerspruch zu (2.3) und (2.4).

Fall 2: x^* ist Randminimum mit $\nabla f(x^*) \neq 0$. Für mindestens ein i gilt also $\frac{\partial f}{\partial x_i}(x^*) \neq 0$. Sei $\mathcal{I} = \{i : \frac{\partial f}{\partial x_i}(x^*) \neq 0\}$ und $\mathcal{J} = \{1, 2, \ldots, n\} \setminus \mathcal{I}$, dann gilt für alle $j \in \mathcal{J}$

$$0 = \frac{\partial f}{\partial x_j}(x^*) \in \frac{\partial f}{\partial x_j}(Y) \subseteq G_j,$$

und somit kann Y nur bezüglich einer Komponente $i \in \mathcal{I}$ reduziert oder als zu löschen gekennzeichnet worden sein.
Sei nun $i \in \mathcal{I}$ und $x_i^* = \underline{Y_i} = \underline{X_i}$ und es gelte $Y_i^M = \overline{Y_i} = \overline{X_i}$ oder $del = true$ für dieses i, was nur durch den Fall (b) des Tests erzielt worden sein kann. Dann muß aber gelten $\overline{G_i} < 0$ und somit

$$\frac{\partial f}{\partial x_i}(x^*) \leq \overline{\frac{\partial f}{\partial x_i}(Y)} \leq \overline{G_i} < 0,$$

so daß x^* nicht der kleinste Funktionswert bezüglich der i-ten Komponente sein kann, im Widerspruch zu $x^* = \underline{Y_i} = \underline{X_i}$.
Sei nun $i \in \mathcal{I}$ und $x_i^* = \overline{Y_i} = \overline{X_i}$ und es gelte $Y_i^M = \underline{Y_i} = \underline{X_i}$ oder $del = true$ für dieses i, was nur durch den Fall (a) des Tests erzielt worden sein kann. Dann muß aber gelten $0 < \underline{G_i}$ und somit

$$0 < \underline{G_i} \leq \overline{\frac{\partial f}{\partial x_i}(Y)} \leq \frac{\partial f}{\partial x_i}(x^*),$$

so daß x^* nicht der kleinste Funktionswert bezüglich der i-ten Komponente sein kann, im Widerspruch zu $x^* = \overline{Y_i} = \overline{X_i}$. □

Abbildung 2.2 verdeutlicht die Arbeitsweise des Monotonietests im $\mathbb{R}^1$ für vier Teilbereiche von X. Im skizzierten Fall wird Y^1 auf die Untergrenze reduziert, Y^2 bleibt unverändert, während Y^3 und Y^4 gelöscht werden.

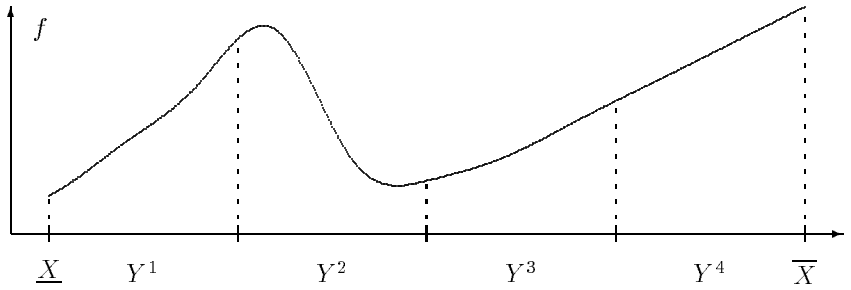


Abbildung 2.2: Monotonietest

Algorithmus 2.9: MonoTest2 (X, Y, del)
Neuer Monotonietest

1. $G := \nabla F(Y)$.
2. $del := false$.
3. **for** $i := 1$ **to** n **do**
   (a) **if** $0 < \underline{G_i}$ **then**
   if $\underline{X_i} = \underline{Y_i}$ **then** $Y_i := [\underline{Y_i}, \underline{Y_i}]$
   else $del := true$; **exit**$_{i\text{-loop}}$.
   (b) **else if** $\overline{G_i} < 0$ **then**
   if $\overline{X_i} = \overline{Y_i}$ **then** $Y_i := [\overline{Y_i}, \overline{Y_i}]$
   else $del := true$; **exit**$_{i\text{-loop}}$.
   (c) **else if** $\underline{G_i} = 0$ **then**
   $Y_i := [\underline{Y_i}, \underline{Y_i}]$.
   (d) **else if** $\overline{G_i} = 0$ **then**
   $Y_i := [\overline{Y_i}, \overline{Y_i}]$.

Der strenge Monotonietest nach Algorithmus 2.8 hat den Nachteil, daß auch sämtliche stationäre Punkte, die keine globale Minimalstelle darstellen, *nicht* gelöscht werden. Wir wollen den Test nun so weiterentwickeln, daß er diesen Fall von schwacher Monotonie behandelt, indem er solche Quader reduziert, ohne dabei einen Quader, der eine globale Minimalstelle enthält, zu löschen.

Wir wollen zunächst voraussetzen, daß die Funktion f in keiner Richtung parallel zu einer Koordinatenachse stückweise konstant ist. Mit Algorithmus 2.9 geben wir diese weiterentwickelte Form des Monotonietests an. Wir bezeichnen ihn mit MonoTest2 (X, Y, del), wobei $del = true$ wie in Algorithmus 2.8 anzeigt, daß die Box Y nicht mehr weiterbearbeitet werden muß, also gelöscht werden kann. Wir wollen nun zeigen, daß der Monotonietest auch in dieser Form keine Minimalstellen aus der Liste entfernt.

Satz 2.3.2 *Sei $f : \mathbb{R}^n \to \mathbb{R}$ aus $C^1(X)$ und in keiner Richtung parallel zu einer Koordinatenachse stückweise konstant. Für jeden Quader $Y \subseteq X$, für den gilt $x^* \in Y$ für eine globale Minimalstelle $x^* \in X$ von f auf dem Bereich X, gilt dann auch nach Ausführung des Monotonietests* MonoTest2 (X, Y, del) *(Algorithmus 2.9) die Beziehung $x^* \in Y$.*

Beweis: Aufgrund von Satz 2.3.1, der auf den aus Algorithmus 2.8 übernommenen Teil von Algorithmus 2.9 zutrifft, müssen wir den Beweis nur für die Erweiterungen (c) und (d) führen.
Angenommen Y sei ein Quader mit $x^* \in Y$, für den nach Ausführung des Monotonietests gilt $x^* \notin Y^M$, wenn Y^M der reduzierte Ergebnisquader nach dem Test ist. Somit muß für mindestens ein i mit $1 \leq i \leq n$ entweder

$$\underline{G_i} = 0 \ \wedge \ x_i^* > Y_i^M = \underline{Y_i} \tag{2.6}$$

oder

$$\overline{G_i} = 0 \ \wedge \ x_i^* < Y_i^M = \overline{Y_i} \tag{2.7}$$

gelten. Sei im folgenden e^i der i-te Einheitsvektor. Ist (2.6) erfüllt, so gibt es wegen $x_i^* > \underline{Y_i}$ ein $\delta > 0$, für das

$$x^* - \delta e^i \in Y \quad \text{und} \quad \frac{\partial f}{\partial x_i}(x^* - \delta e^i) < 0$$

erfüllt ist. Dies steht aber wegen

$$0 > \frac{\partial f}{\partial x_i}(x^* - \delta e^i) \geq \underline{\frac{\partial f}{\partial x_i}(Y)} \geq \underline{G_i}$$

im Widerspruch zu $\underline{G_i} = 0$. Ist (2.7) erfüllt, so gibt es wegen $x_i^* < \overline{Y_i}$ ein

$\delta > 0$, für das

$$x^* + \delta e^i \in Y \quad \text{und} \quad \frac{\partial f}{\partial x_i}(x^* + \delta e^i) > 0$$

erfüllt ist. Dies steht aber wegen

$$0 < \frac{\partial f}{\partial x_i}(x^* + \delta e^i) \leq \overline{\frac{\partial f}{\partial x_i}(Y)} \leq \overline{G_i}$$

im Widerspruch zu $\overline{G_i} = 0$. □

Wir wollen nun noch kurz untersuchen, ob wir die Einschränkung auf Funktionen, die in keiner Richtung parallel zu einer Koordinatenachse stückweise konstant sind, aufheben können, indem wir jeweils eine zusätzliche Bedingung im (c)- bzw. im (d)-Zweig der zweiten Form des Monotonietests überprüfen. Prinzipiell kann ein Quader Y auch bezüglich der i-ten Koordinate reduziert werden, wenn f längs dieser Achse stückweise konstant ist. Voraussetzung dafür ist aber, daß für den konstanten Wert r, den f auf diesem Teilstück annimmt, $r > f(x^*)$ gilt. Da wir diese Bedingung aber nicht nachprüfen können, müssen wir zumindest sicherstellen, daß durch die Reduktion von Y keine Kandidaten für globale Minimalstellen verlorengehen. Dies kann zum Beispiel dadurch geschehen, daß wir zunächst einmal die Reduktion und für das neue Y eine weitere Gradientenauswertung $G := \nabla F(Y)$ durchführen. Gilt anschließend $\underline{G_i} > 0$ für den Fall (c) bzw. $\overline{G_i} < 0$ für den Fall (d), so ist dies hinreichend dafür, daß diese Reduktion keine globalen Minimalstellen eliminiert hat. Andernfalls muß die Reduktion bezüglich der i-ten Komponente wieder rückgängig gemacht werden.

In Abbildung 2.3 geben wir für dieses Verfahren drei Fallbeispiele im $\mathbb{R}^1$. Während im ersten Fall eine Reduktion auf den linken Rand möglich ist, läßt sich Fall 2 durch die oben genannten Bedingungen leider nicht positiv entscheiden, da die Ableitung am linken Rand den Wert 0 hat, so daß eine Reduktion wieder rückgängig gemacht werden würde. Im dritten Fall wird eine Reduktion richtigerweise nicht erlaubt, da die Ableitung auf einem ganzen Teilstück von Y den Wert 0 annimt.

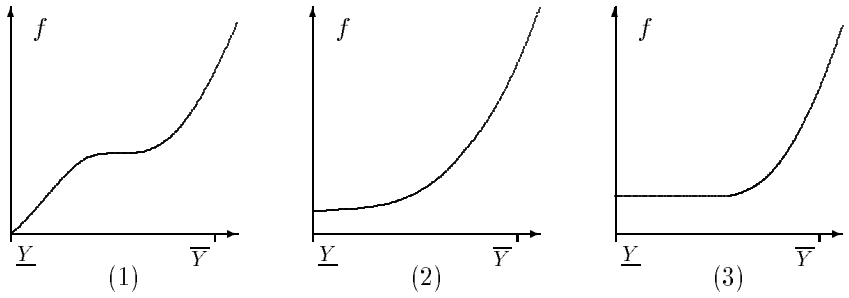


Abbildung 2.3: Drei Fallbeispiele zum Monotonietest

Diese Methodik erfordert natürlich einen erhöhten Aufwand für den Monotonietest, da für jedes i, das die Bedingung für Fall (c) oder (d) erfüllt, eine neuerliche Gradientenauswertung erforderlich wird. Andererseits kann das neu berechnete G im weiteren Verlauf des Tests aber auch zusätzliche erfolgreiche Tests für die Komponenten $j > i$ ermöglichen, da mit kleiner werdendem Intervalldurchmesser auch die Gradientenauswertung besser und die Überschätzung geringer wird. So wäre es je nach Bauart der Funktion auch sinnvoll, grundsätzlich nach jeder Reduktion in den Zweigen (a) und (b) des Monotonietests eine neue Gradientenauswertung durchzuführen, um damit mehr erfolgreiche Tests für die weiteren Komponenten zu ermöglichen.

2.4 Der Konkavitätstest

Der Konkavitätstest ist eigentlich ein Test auf „Nicht-Konvexität“, und er erhielt seinen Namen in der Literatur wohl zur Vereinfachung der Sprechweise. Er wird verwendet, um festzustellen ob die zu minimierende Funktion f, für die wir jetzt $f \in C^2(X)$ fordern müssen, auf einem ganzen Quader $Y \subseteq X$ *nicht konvex* ist. Ist dies der Fall, so braucht Y nicht mehr weiter auf eine Minimalstelle untersucht zu werden, außer wenn der Rand von Y mit dem Rand von X zusammenfällt. In diesem Fall kann Y, wie auch im Monotonietest, zumindest auf den Rand, der möglicherweise eine globale Minimalstelle enthält, reduziert werden.

Nach Satz 1.3.3 ist f konvex bzw. streng konvex in Y, wenn für alle $y \in Y$ gilt, daß die Hessematrix $H = \nabla^2 f(y)$ positiv semi-definit bzw. positiv definit ist. Ferner haben wir durch Satz 1.1.2 ein notwendiges Kriterium für die positive Definitheit bzw. Semi-Definitheit einer Matrix mit der Nichtnegativität ihrer Diagonalelemente gegeben. Mittels einer Inter-

vallauswertung der Hessematrix über Y können wir somit feststellen, ob f in Y konvex sein kann.

In [13] bzw. [59] beschreibt Hansen einen Konkavitätstest nur mit Löschen und ohne Randbetrachtungen, während Ratschek und Rokne [49] den Konkavitätstest nicht in ihren Algorithmus einbeziehen. Mit Algorithmus 2.10 beschreiben wir einen Konkavitätstest mit komponentenweiser Reduktion. Wir bezeichnen ihn mit ConcTest $(X, Y, L, \widetilde{f}, del)$, wobei $del = true$ wie bereits im Monotonietest anzeigt, daß die Box Y nicht mehr weiterbearbeitet werden muß, also gelöscht werden kann.

Besondere Beachtung müssen wir dabei dem Fall schenken, in dem eine Komponente von Y auf beide Randpunkte reduziert werden kann. Während wir eine einfache Reduktion noch problemlos wie beim Monotonietest durchführen können, müssen wir die doppelte Reduktion aufgrund der nachfolgenden Kombinationsmöglichkeiten separat behandeln. Dabei hat es sich als sinnvoll erwiesen, diese beiden Fälle unter Zuhilfenahme einer Indexmenge $\mathcal{I}$ und eines Algorithmus 2.11 Reduce $(Y, \mathcal{I}, L, \widetilde{f})$ in einer zusätzlichen Reduktionsschleife getrennt zu bearbeiten.

Algorithmus 2.10: ConcTest $(X, Y, L, \widetilde{f}, del)$
Konkavitätstest

1. $H := \nabla^2 F(Y)$; $del := false$; $\mathcal{I} := \{\}$.
2. **for** $i := 1$ **to** n **do**
   (a) **if** $\overline{H_{ii}} < 0$ **then**
   if $\underline{X_i} = \underline{Y_i}$ **and** $\overline{X_i} = \overline{Y_i}$ **then** $\mathcal{I} := \mathcal{I} \cup \{i\}$
   else if $\underline{X_i} = \underline{Y_i}$ **then** $Y_i := [\underline{Y_i}, \underline{Y_i}]$
   else if $\overline{X_i} = \overline{Y_i}$ **then** $Y_i := [\overline{Y_i}, \overline{Y_i}]$
   else $del := true$; **exit**$_{i\text{-loop}}$.
3. **if not** del **then** Reduce $(Y, \mathcal{I}, L, \widetilde{f})$.

In Algorithmus 2.11 verwenden wir Listen L' und L'' mit Intervallvektoren als Elemente, für die die Operationen aus Abschnitt 2.2.1 definiert seien. Für die Operation + müssen wir keine besondere Regelung treffen. Die Listen dienen der Zwischenspeicherung der Vektoren, die bei den Kombinationen der Komponentenreduktionen entstehen. Nachdem alle Reduktionen durchgeführt sind, werden die Vektoren in die eigentliche Liste L unseres Optimierungsalgorithmus eingehängt. Aus diesem Grund benötigen wir

auch den zusätzlichen Parameter $\widetilde{f}$, da wir einen Quader nur in die Liste aufnehmen, wenn die Unterschranke der Intervallfunktionsauswertung nicht bereits größer als $\widetilde{f}$ ist (vgl. auch Abschnitt 2.2.5.1).

Algorithmus 2.11: Reduce $(Y, \mathcal{I}, L, \widetilde{f})$
Doppelreduktion

1. $L' := \{Y\}$.
2. **for** $i \in \mathcal{I}$ **do**
   (a) $L'' := L'$; $L' := \{\}$.
   (b) **while** $L'' \neq \{\}$ **do**
      i. $Z :=$ Head (L''); $L'' := L'' - Z$.
      ii. $A := Z_i$.
      iii. $Z_i := [\underline{A}, \underline{A}]$; $L' := L' + Z$.
      iv. $Z_i := [\overline{A}, \overline{A}]$; $L' := L' + Z$.
3. **while** $L' \neq \{\}$ **do**
   (a) $Z :=$ Head (L'); $L' := L' - Z$.
   (b) **if** $\underline{F_Z} \leq \widetilde{f}$ **then** $L := L + (Z, \underline{F_Z})$.

Beispiel 2.4.1 Wenden wir Algorithmus 2.11 auf

$$Y = \begin{pmatrix} [-1,1] \\ [-2,2] \\ [-3,3] \\ [-4,4] \end{pmatrix} \quad \text{und} \quad \mathcal{I} = \{2,3\}$$

an, so ergibt sich L' zunächst zu

$$L' = \left\{ \begin{pmatrix} [-1,1] \\ [-2,-2] \\ [-3,3] \\ [-4,4] \end{pmatrix}, \begin{pmatrix} [-1,1] \\ [2,2] \\ [-3,3] \\ [-4,4] \end{pmatrix} \right\}$$

und anschließend zu

$$L' = \left\{ \begin{pmatrix} [-1,1] \\ [-2,-2] \\ [-3,-3] \\ [-4,4] \end{pmatrix}, \begin{pmatrix} [-1,1] \\ [-2,-2] \\ [3,3] \\ [-4,4] \end{pmatrix}, \begin{pmatrix} [-1,1] \\ [2,2] \\ [-3,-3] \\ [-4,4] \end{pmatrix}, \begin{pmatrix} [-1,1] \\ [2,2] \\ [3,3] \\ [-4,4] \end{pmatrix} \right\}.$$

Wir wollen nun zeigen, daß der Konkavitätstest in Form von Algorithmus 2.10 keine Minimalstellen aus der Liste entfernt.

Satz 2.4.1 *Sei $f : \mathbb{R}^n \to \mathbb{R}$ aus $C^2(X)$. Für jeden Quader $Y \subseteq X$, für den gilt $x^* \in Y$ für eine globale Minimalstelle $x^* \in X$ von f auf dem Bereich X, gilt nach Ausführung des Konkavitätstests nach Algorithmus 2.8 die Beziehung $x^* \in Y$ bzw. $x^* \in Z$ für $Z \in L' \subseteq L$.*

Beweis: Angenommen Y sei ein Quader mit $x^* \in Y$, für den nach Ausführung des Konkavitätstests gilt $del = true$ oder

$$x^* \notin Y^C \ \wedge \ x^* \notin Z \quad \text{für alle } Z \in L', \tag{2.8}$$

wenn Y^C der reduzierte Ergebnisquader nach dem Test ist. Somit muß für mindestens ein i mit $1 \leq i \leq n$

$$\overline{H_{ii}} < 0 \tag{2.9}$$

gelten.

Fall 1: x^* ist stationäre Stelle. Die Hessematrix $\nabla^2 f(x^*)$ ist also positiv definit oder positiv semidefinit. Nach Satz 1.1.2 müssen somit alle ihre Diagonalelemente nichtnegativ sein, was wegen

$$0 \leq \frac{\partial^2 f}{\partial x_i^2}(x^*) \leq \overline{\frac{\partial^2 f}{\partial x_i^2}(Y)} \leq \overline{H_{ii}}$$

im Widerspruch zu (2.9) steht.

Fall 2: x^* ist Randminimum. In diesem Fall wird, wenn (2.9) für ein i erfüllt ist, Y_i stets auf alle mit X_i gemeinsamen Ränder reduziert, was sofort einen Widerspruch zu (2.8) ergibt.

□

Abbildung 2.4 verdeutlicht die Arbeitsweise des Konkavitätstests im $\mathbb{R}^1$ für vier Teilbereiche von X. Im skizzierten Fall wird Y^1 auf die Untergrenze reduziert, Y^2 bleibt unverändert, Y^3 kann gelöscht werden und Y^4 wird nach oben reduziert.

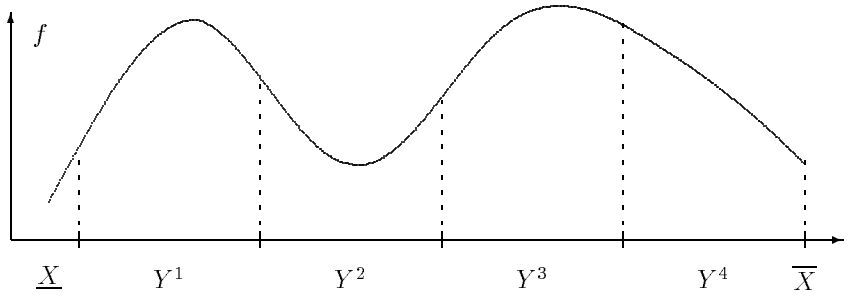


Abbildung 2.4: Konkavitätstest

Auch beim Konkavitätstest könnte prinzipiell mit erhöhtem Aufwand versucht werden, mehr erfolgreiche Tests für einzelne Komponenten zu ermöglichen, indem nach jeder Reduktion eine neuerliche Hessematrixauswertung durchgeführt wird. Aufgrund der kleiner gewordenen Intervalldurchmesser kann dies je nach Bauart der Funktion f zusätzliche erfolgreiche Tests für die weiteren Komponenten ermöglichen. Dieses Vorgehen ist jedoch bei einer mehrfachen Auswertung der Hessematrix zu aufwendig.

Andererseits ist der Konkavitätstest in Form von Algorithmus 2.10 ein sinnvolles Mittel, um die Quader vorzeitig aus der Liste zu eliminieren bzw. ihre Komponentendurchmesser zu verkleinern. Der Empfehlung in [8], den Konkavitätstest aufgrund des hohen Aufwands nicht zu verwenden, können wir entgegenhalten, daß im Hinblick auf den sich anschließenden Intervall-Newton-Schritt (Abschnitt 2.5) der Aufwand für die Auswertung der Hessematrix auf jeden Fall betrieben werden muß. Ein erfolgreicher Test mit $del = true$ erspart die Durchführung des Newton-Schrittes, ein negativer Test bedeutet lediglich die n Vergleiche als Aufwand.

2.5 Der Intervall-Newton-Schritt

In diesem Abschnitt werden wir ausführlich auf unser wesentliches Hilfsmittel zur Beschleunigung des Algorithmus und zur Verifikation von Existenz und Eindeutigkeit eingehen, den Intervall-Newton-Schritt. Er wird zum einen, ähnlich wie Mittelpunkts-, Monotonie- und Konkavitätstest, dazu eingesetzt, bestimmte Quader frühzeitig aus der Liste zu eliminieren, wenn garantiert werden kann, daß diese keine globale Minimalstelle enthalten, andererseits ermöglicht er aber auch die Verkleinerung der aktuellen Quader.

In Abschnitt 1.7 haben wir uns bereits mit dem eindimensionalen Intervall-Newton-Verfahren beschäftigt, das wir im folgenden für den mehrdimensionalen Fall einsetzen wollen. In unserem speziellen Fall der Anwendung auf globale Optimierungsprobleme werden wir das Newton-Verfahren bzw. die Newton-ähnlichen Verfahren auf den Gradienten $g = \nabla f$ der zu minimierenden Funktion f anwenden, um Nullstellen von g zu berechnen bzw. um Einschließungen solcher Nullstellen zu verbessern. Dabei müssen wir nunmehr $f \in C^2(X)$ fordern. Wir behandeln also das Problem

$$\nabla f(x) = g(x) = 0, \quad \text{für } x \in X \text{ und } g : X \subset I\!R^n \to I\!R^n.$$

Aufgrund der Tatsache, daß die Nullstellen von g nicht notwendigerweise globale Minimalstellen von f sind, werden wir keine echte Newton-Iterationen verwenden, sondern vielmehr im Rahmen einer Iteration unseres grundlegenden Algorithmus jeweils für den zu behandelnden Quader einen Intervall-Newton-Schritt durchführen. Da ein Teil der Quader zwischenzeitlich bereits durch die in den letzten Abschnitten beschriebenen Tests eliminiert wird, kann somit der Aufwand für die Newton-Iteration in der Nähe solcher Punkte gespart werden.

Wir werden zunächst die verschiedenen Varianten des Verfahrens, die ausführlich in [1], [2], [38] oder [44] behandelt werden, beschreiben und einige wesentliche Eigenschaften zusammenfassen. Anschließend werden wir die für unseren Algorithmus verwendeten modifizierten Methoden ausführlich behandeln.

2.5.1 Grundlegende Formen des Newton-Schritts

Zum Intervall-Newton-Verfahren kommt man, wie beim klassischen Newton-Verfahren, durch eine Linearisierung bzw. durch eine komponentenweise Expandierung der Funktion $g : X \subset I\!R^n \to I\!R^n$ mit Hilfe des Mittelwertsatzes. Mit $c, x^* \in X$, $\widetilde{H} \in I\!R^{n \times n}$ und x^* Nullstelle von g ist

$$0 = g(x^*) = g(c) + \widetilde{H} \cdot (x^* - c).$$

Ausgehend von einer Einschließung Y der Nullstelle x^* und $\widetilde{H} \in H(Y)$ kann nun mit dem Intervall-Newton-Verfahren iterativ das entsprechende Intervallgleichungssystem

$$H(Y) \cdot (c - Y^N) = g(c) \tag{2.10}$$

gelöst, d. h. eine Einschließung der Lösung Y^N des Systems bestimmt, und damit die Einschließung Y durch $Y := Y^N \cap Y$ verbessert werden. Dabei

ist $H = \nabla^2 F = J_G$ die Hessematrix der Funktion F bzw. die Jacobimatrix von G, Y die letzte Iterierte des Verfahrens und $c \in Y$ beliebig. Zu den verschiedenen Intervall-Newton-Verfahren gelangt man durch verschiedene Möglichkeiten zur Berechnung von Y^N aus (2.10).

2.5.1.1 Direkter Newton-Schritt

Der eigentliche, direkte Intervall-Newton-Schritt ist durch die Vorschrift

$$\left.\begin{array}{rcl} \boldsymbol{N}(Y) & := & m(Y) - (H(Y))^{-1} \cdot g(m(Y)) \\ Y^N & := & \boldsymbol{N}(Y) \cap Y \end{array}\right\} \tag{2.11}$$

gegeben, wobei $c = m(Y)$ gewählt wurde. Aufgrund der notwendig werdenden Berechnung von $(H(Y))^{-1}$, also der Intervalleinschließung der Inversen aller reellen Matrizen in $H(Y)$, in diesem Newton-Schritt, darf $H(Y)$ nur reguläre Matrizen enthalten. Damit darf g höchstens eine Nullstelle in Y haben. Unter diesen Voraussetzungen können wir einen ersten Satz angeben, der die zentralen Eigenschaften des Intervall-Newton-Schrittes zusammenfaßt (vgl. z. B. [44]).

Satz 2.5.1 *Sei $g : X \subset \mathbb{R}^n \to \mathbb{R}^n$ aus $C^2(\mathbb{R}^n)$ und die Intervall-Jacobimatrix $H(Y) = J_G(Y)$ enthalte nur reguläre Matrizen, dann gilt für den Intervall-Newton-Schritt (2.11):*

(a) Ist x^* Nullstelle von g mit $x^* \in Y$, dann gilt $x^* \in \boldsymbol{N}(Y)$ und somit auch $x^* \in Y^N$.

(b) Ist $\boldsymbol{N}(Y) \cap Y = \emptyset$, dann enthält Y keine Nullstelle von g.

(c) Gilt $\boldsymbol{N}(Y) \overset{\circ}{\subset} Y$, dann besitzt g in Y und damit auch in $\boldsymbol{N}(Y)$ eine eindeutige Nullstelle.

Beweis: Siehe zum Beispiel [44]. □

Der Intervall-Newton-Schritt (2.11) kann aufgrund der notwendigen Invertierung der Matrizen in $H(Y)$ nur verwendet werden, wenn $H(Y)$ nichtsingulär ist. Diesbezügliche Abhilfe schaffen hier die Verfahren von Krawczyk [28] bzw. Rump [52] oder eine nichtlineare Version des Intervall-Gauß-Seidel-Schrittes, die ohne die Invertierung von Intervallmatrizen auskommen. Die Intervall-Gauß-Seidel-Iteration wurde bereits von Alefeld [1] beschrieben und von Hansen und Sengupta [16] in Verbindung mit erweiterter Intervallarithmetik behandelt. Eine Modifikation des Krawczyk-Verfahrens in Verbindung mit der Gauß-Seidel-Iteration stammt von Shearer und Wolfe [54]. Ihre Anwendung im Hinblick auf die Optimierung wollen wir in Abschnitt 2.9 erläutern.

2.5.1.2 Präkonditionierung

Bevor wir im folgenden kurz auf die Methode von Krawczyk eingehen und uns anschließend ausführlich mit dem Gauß-Seidel-Schritt beschäftigen, den wir in unserem Verfahren zur globalen Optimierung einsetzen werden, wollen wir den Begriff der Präkonditionierung erläutern. Diese Thematik werden wir in Abschnitt 2.5.4 auch speziell im Hinblick auf den Gauß-Seidel-Schritt ausführlich behandeln.

Präkonditionerung bedeutet, daß das Intervallgleichungssystem (2.10) mit einer *Präkonditionierermatrix* R multipliziert wird, so daß das System

$$R \cdot H(Y) \cdot (c - Y^N) = R \cdot g(c) \tag{2.12}$$

zu lösen ist. Sehr oft wird $R = (m(H(Y)))^{-1}$ als Präkonditionierermatrix verwendet, um damit das System fast diagonaldominant zu machen (vgl. [15]).

2.5.1.3 Der Krawczyk-Operator

Der Krawczyk-Operator $\boldsymbol{K}(Y)$ hat die Form

$$\boldsymbol{K}(Y) := c - R \cdot g(c) + (I - R \cdot H(Y)) \cdot (Y - c), \tag{2.13}$$

wobei I die Einheitsmatrix bezeichnet. Wiederum berechnet sich Y^N durch

$$Y^N := \boldsymbol{K}(Y) \cap Y.$$

Ganz analog zu Satz 2.5.1 können auch für $\boldsymbol{K}(Y)$ die entsprechenden Eigenschaften nachgewiesen werden.

Satz 2.5.2 *Sei $g : X \subset \mathbb{R}^n \to \mathbb{R}^n$ aus $C^2(\mathbb{R}^n)$ und $H(Y) = J_G(Y)$, dann gilt für den Krawczyk-Schritt (2.13):*

(a) Ist x^* Nullstelle von g mit $x^* \in Y$, dann gilt $x^* \in \boldsymbol{K}(Y)$.

(b) Ist $\boldsymbol{K}(Y) \cap Y = \emptyset$, dann enthält Y keine Nullstelle von g.

(c) Gilt $\boldsymbol{K}(Y) \overset{\circ}{\subset} Y$, dann besitzt g in Y und damit auch in $\boldsymbol{K}(Y)$ eine eindeutige Nullstelle und $H(Y)$ ist nichtsingulär.

Beweis: Siehe zum Beispiel [44]. □

2.5.1.4 Der Gauß-Seidel-Schritt

Der Intervall-Gauß-Seidel-Schritt entsteht durch die Aufspaltung der Matrix $A = R \cdot H(Y)$ in ihren Diagonalanteil und den Rest sowie durch die Auflösung des entsprechenden Systems nach dem Einzelschrittverfahren. Er ist mit $c = m(Y)$ und $b = R \cdot g(c)$ gegeben durch

$$\left.\begin{array}{l} Z_i := c_i - \left(b_i + \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}(Y_j - c_j) \right) \Big/ A_{ii} \\ Y_i := Z_i \cap Y_i \end{array}\right\} i = 1, \ldots, n. \qquad (2.14)$$

Gilt $Y_i = \emptyset$ nach der Schnittbildung, so kann die Berechnung der Y_i abgebrochen werden und $Y := \emptyset$ gesetzt werden. Zu beachten ist, daß die Division durch das Diagonalelement A_{ii} in erweiterter Intervallarithmetik durchgeführt werden muß, falls gilt $0 \in A_{ii}$. In diesem Fall ist Z_i bzw. Y_i als abkürzende Schreibweise für die Paarschreibweise $(Z_i^1 \,|\, Z_i^2)$ bzw. $(Y_i^1 \,|\, Y_i^2)$ zu sehen, die wir in Abschnitt 1.6 eingeführt haben. Die daraus eventuell resultierende Aufteilung der Z_i und entsprechend der Y_i in zwei Intervalle hat jedoch keinerlei nachteilige Konsequenzen auf die Existenz- und Eindeutigkeitsaussagen, denn auch der Intervall-Gauß-Seidel-Schritt besitzt die Eigenschaften des Newton-Schrittes und des Krawczyk-Schrittes.

Satz 2.5.3 *Sei $g : \mathbb{R}^n \to \mathbb{R}^n$ aus $C^2(\mathbb{R}^n)$ und $H(Y) = J_G(Y)$, dann gilt für das Ergebnis $Y^N = \boldsymbol{GS}(Y)$ eines Intervall-Gauß-Seidel-Schrittes (2.14):*

(a) Ist x^* Nullstelle von g mit $x^* \in Y$, dann gilt $x^* \in Y^N$.

(b) Ist $Y^N = \emptyset$, dann enthält Y keine Nullstelle von g.

(c) Gilt $Y^N \overset{\circ}{\subset} Y$, dann besitzt g in Y und damit auch in Y^N eine eindeutige Nullstelle und $H(Y)$ ist nichtsingulär.

Beweis: Siehe zum Beispiel [44]. □

Der Gauß-Seidel-Schritt wird bevorzugt, da er beim Auftreten eines leeren Schnitts in der Berechnung der Y_i schon vorzeitig abgebrochen werden kann. Außerdem liefert er in der Regel engere Einschließungen als der Krawczyk-Schritt, so daß auch die Bedingung für den Nachweis der Eindeutigkeit leichter zu erfüllen ist. Eine allgemeine Beziehung zwischen Gauß-Seidel- und Krawczyk-Schritt wird durch den nachfolgenden Satz gegeben.

Satz 2.5.4 *Seien* $\boldsymbol{GS}(Y)$ *und* $\boldsymbol{K}(Y)$ *die Ergebnisse des Gauß-Seidel-Schrittes und des Krawczyk-Schrittes, dann gilt:*

$$\boldsymbol{GS}(Y) \subseteq \boldsymbol{K}(Y).$$

Beweis: Siehe zum Beispiel [44]. □

2.5.2 Der Gauß-Seidel-Schritt nach Hansen

Wir wollen uns nun zunächst der algorithmischen Beschreibung des Gauß-Seidel-Schrittes (2.14) zuwenden, wie er in [13] verwendet wird. Er liefert als Ergebnis höchstens zwei Quader für die weitere Bearbeitung innerhalb des grundlegenden Optimierungsalgorithmus. Die Aufteilung geschieht dabei in derjenigen Komponente von Y, in der aufgrund der Division durch A_{ii} mit $0 \in A_{ii}$ die größte Lücke entsteht. Hansen bearbeitet die Fälle $0 \notin A_{ii}$ und $0 \in A_{ii}$ deshalb getrennt. Als Präkonditionierermatrix wird die inverse Mittelpunktsmatrix der Intervall-Hessematrix über Y verwendet. Diese Form des Gauß-Seidel-Schrittes, die wir mit $\mathsf{NewtonStepGS}\,(Y, V, p, \mathit{mode})$ bezeichnen wollen, wird durch Algorithmus 2.13 beschrieben.

Um den Algorithmus möglichst übersichtlich zu halten, führen wir nachfolgend mit Algorithmus 2.12 eine Kurznotation für die Berechnung von $Y_i = (W_i^1 \mid W_i^2)$ entprechend der Vorschrift in (2.14) ein. Dabei liefert $\mathsf{IStepGS}\,(Y, A, b, c, i, W_i^1, W_i^2, p)$ als Ergebnis $(W_i^1 \mid W_i^2)$, wobei, entsprechend der Definition der Division (vgl. Abschnitt 1.6), $W_i^2 = \emptyset$ durch $p = 1$ und $W_i^1 = W_i^2 = \emptyset$ durch $p = 0$ angezeigt wird.

Algorithmus 2.12: $\mathsf{IStepGS}\,(Y, A, b, c, i, W_i^1, W_i^2, p)$
Gauß-Seidel-Schritt für i-te Komponente

1. $$(W_i^1 \,|\, W_i^2) := \left(c_i - \left(b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}\,(Y_j - c_j) \right) \Big/ A_{ii} \right) \cap Y_i.$$
2. **if** $W_i^1 = W_i^2 = \emptyset$ **then** $p := 0$
   else if $W_i^2 = \emptyset$ **then** $p := 1$
   else $p := 2$.

Darüber hinaus benötigen wir in Algorithmus 2.13 ein Maß für die nach der erweiterten Intervalldivision und der Schnittbildung entstehende Lücke.

Definition 2.5.1 *Sei $A = (A^1 \,|\, A^2)$ mit $A^1, A^2 \in I\!R$, dann heißt $d_\sqcup$ mit*

$$d_\sqcup(A) = \underline{A^2} - \overline{A^1} = \inf(A^2) - \sup(A^1)$$

Lückendurchmesser von A.

Algorithmus 2.13: NewtonStepGS $(Y, V, p, mode)$
Gauß-Seidel-Schritt nach Hansen

1. $H := \boldsymbol{\nabla}^2 F(Y)$; $M := m(H)$; $c := m(Y)$; $g := \boldsymbol{\nabla} f(c)$.
2. **if** $mode_{\text{precon}} = 0$ **or** M singulär **then** $R := I$
   else $R := M^{-1}$.
3. $A := R \cdot H$; $b := R \cdot g$; $k := 0$; $w := 0$.
4. **for** $i := 1$ **to** n **do**
   if $0 \notin A_{ii}$ **then**
   (a) IStepGS $(Y, A, b, c, i, W_i^1, W_i^2, p)$.
   (b) **if** $p = 0$ **then return**.
   (c) $Y_i := W_i^1$.
5. **for** $i := 1$ **to** n **do**
   if $0 \in A_{ii}$ **then**
   (a) IStepGS $(Y, A, b, c, i, W_i^1, W_i^2, p)$.
   (b) **if** $p = 0$ **then return**.
   (c) **if** $p = 1$ **then** $Y_i := W_i^1$
   else if $d_\sqcup(W_i) \geq w$ **then** $w := d_\sqcup(W_i)$; $k := i$.
6. **if** $k > 0$ **then**
   (a) $p := 2$; $V := Y$.
   (b) $Y_k := W_k^1$; $V_k := W_k^2$.

In Schritt 4 von Algorithmus 2.13 werden zunächst alle Komponenten behandelt, für die $0 \notin A_{ii}$ gilt, danach in Schritt 5 die restlichen Komponenten. Auch hier kann Y sofort verbessert werden, wenn $p = 1$ erfüllt ist, also keine Lücke aufgetreten ist. Für $p = 2$ wird die größte Lücke w und deren Index k

gespeichert. In beiden Teilschritten kann die Bearbeitung des Algorithmus komplett abgebrochen werden, wenn ein leerer Schnitt, also $p = 0$ im i-ten Teilschritt aufgetreten ist. Die eigentliche Aufspaltung von Y bezüglich der größten Lücke erfolgt in Schritt 6, indem Y in V kopiert wird und jeweils Y_k und V_k mit den entsprechenden Teilintervallen belegt werden.

Der Wert des Ergebnisparameters $p \in \{0, 1, 2\}$ entspricht der Anzahl der Quader nach dem Gauß-Seidel-Schritt, wobei $p = 0$ andeutet, daß ein leerer Schnitt aufgetreten ist und Y damit nicht mehr weiter behandelt werden muß. Y und V sind die Ergebnisquader, mit $V = \emptyset$ für $p = 1$ bzw. $Y = V = \emptyset$ für $p = 0$. Der Parameter *mode* ist eine vektorielle Notation für verschiedene Schalter, die in unserem Optimierungsalgorithmus zur Auswahl bestimmter Bearbeitungsmodi verwendet werden. Wir werden sie ausführlich in Abschnitt 2.5.8 bzw. 2.8 beschreiben. Im Zusammenhang mit Algorithmus 2.13 zeigt $mode_{\text{precon}}$ an, welche Art der Präkonditionierung vorgenommen werden soll. Unsere Resultate in Kapitel 4 werden zeigen, daß auch ohne Präkonditionierung in vielen Fällen gute Resultate erzielt werden können, so daß es sinnvoll ist, einen solchen Schalter einzuführen.

2.5.3 Ein modifizierter Gauß-Seidel-Schritt

Algorithmus 2.13 hat im Hinblick auf die Aufteilung der Komponentenintervalle einen großen Nachteil. In Schritt 5 werden zwar sämtliche Schritte, die eine Nulldivision beinhalten und damit eine Aufteilung des entsprechenden Intervalls verursachen, durchgeführt, am Ende wird jedoch in Schritt 6 nur *eine* der so entstandenen Lücken verwendet. Dies hat zur Folge, daß ein Großteil bereits erhaltenener Informationen praktisch weggeworfen wird. Dies ist vor allem deshalb als Nachteil anzusehen, weil der für den Newton-Schritt getriebene Aufwand zur Berechnung von Hessematrix, Präkonditionierer und Gauß-Seidel-Schritt schließlich nur für eine einzige Aufteilung benutzt wird (obwohl mehrere möglich wären) und die weiteren Aufteilungen erst in späteren Iterationsschritten – nach erneuter Berechnung der notwendigen Hessematrizen und Präkonditionierer – vorgenommen wird.

Andererseits wäre es kaum sinnvoll, sämtliche Aufteilungen zu verwenden und alle Quader, die durch die entsprechenden Kombinationen der einzelnen Intervallteile in den Komponenten entstehen, weiter zu behandeln, da dies eine zu große Zahl von Quadern für die weitere Behandlung im Gesamtalgorithmus und somit auch eine deutliche Erhöhung der Anzahl der Funktions- und Ableitungsauswertungen mit sich bringen würde. Dieser Schluß wird auch in [13] und [49] gezogen, wo Algorithmus 2.13 aus diesem Grund favorisiert wird.

Wir wollen nun eine Modifikation des Gauß-Seidel-Schritts entwickeln, in der wir für die Aufteilung der Intervalle im Gauß-Seidel-Schritt eine Strategie verfolgen, die einen Mittelweg zwischen dieser vollständigen Ausnutzung des betriebenen Aufwands und der minimalen Ausnutzung nach Algorithmus 2.13 einschlägt. Zur Verdeutlichung unseres neuen Verfahrens wollen wir uns jedoch zunächst die beiden zuerst genannten, extremen Methoden anhand eines Beispieles nochmals verdeutlichen.

Beispiel 2.5.1 Gegeben sei ein Quader $Y \in I\mathbb{R}^3$, für den durch den Gauß-Seidel-Schritt aufgrund von $0 \in A_{ii}$ für $i = 1, 2, 3$ in jeder Komponente eine Lücke entstehen würde, also

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} \to \begin{pmatrix} (W_1^1 \,|\, W_1^2) \\ (W_2^1 \,|\, W_2^2) \\ (W_3^1 \,|\, W_3^2) \end{pmatrix}.$$

Der Gauß-Seidel-Schritt nach Hansen würde somit, wenn in der zweiten Komponente die größte Lücke entstünde, die Quader

$$Y := \begin{pmatrix} Y_1 \\ W_2^1 \\ Y_3 \end{pmatrix} \quad \text{und} \quad V := \begin{pmatrix} Y_1 \\ W_2^2 \\ Y_3 \end{pmatrix}$$

als Ergebnis liefern. Würde man hingegen aus den drei Aufspaltungen sämtliche möglichen Kombinationen bilden, so würden die acht Quader

$$V^1 := \begin{pmatrix} W_1^1 \\ W_2^1 \\ W_3^1 \end{pmatrix}, \quad V^2 := \begin{pmatrix} W_1^2 \\ W_2^1 \\ W_3^1 \end{pmatrix}, \quad V^3 := \begin{pmatrix} W_1^1 \\ W_2^2 \\ W_3^1 \end{pmatrix}, \quad V^4 := \begin{pmatrix} W_1^2 \\ W_2^2 \\ W_3^1 \end{pmatrix},$$

$$V^5 := \begin{pmatrix} W_1^1 \\ W_2^1 \\ W_3^2 \end{pmatrix}, \quad V^6 := \begin{pmatrix} W_1^2 \\ W_2^1 \\ W_3^2 \end{pmatrix}, \quad V^7 := \begin{pmatrix} W_1^1 \\ W_2^2 \\ W_3^2 \end{pmatrix}, \quad V^8 := \begin{pmatrix} W_1^2 \\ W_2^2 \\ W_3^2 \end{pmatrix}$$

entstehen, die somit in der Arbeitsliste L des Optimierungsalgorithmus abgespeichert werden müßten.

Allgemein liefert die zweite Vorgehensweise bei m Intervallaufspaltungen in den Komponenten jeweils 2^m Ergebnisvektoren und somit ein exponentielles Anwachsen unserer Arbeitsliste.

Mit unserem modifizierten Verfahren wollen wir erreichen, daß die Anzahl der neuen Quader linear mit der Anzahl der Komponentenaufteilungen wächst. Unser Algorithmus innerhalb des Gauß-Seidel-Schrittes wird deshalb für die Berechnung der Y_i für den Fall $0 \in A_{ii}$ folgende grundlegende Struktur haben:

1. Berechne $(W_i^1 \,|\, W_i^2)$.
2. Setze $Y_i := W_i^2$.
3. Speichere Y.
4. Setze $Y_i := W_i^1$.

Nach der Berechnung einer Intervallaufteilung wird also jeweils ein Teilintervall zur Bildung und Speicherung eines neuen Quaders benutzt, während durch das andere Teilintervall die entsprechende Komponente des Vektors Y für das weitere Verfahren aktualisiert wird. Bei m Intervallaufteilungen entstehen somit insgesamt $m+1$ Quader, von denen m abgespeichert werden.

Beispiel 2.5.2 Wenden wir dieses Verfahren auf Beispiel 2.5.1 an, so entstehen jetzt nur noch vier Quader entprechend dem nachfolgend angedeuteten Schema.

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} \rightarrow \begin{pmatrix} W_1^1 \\ Y_2 \\ Y_3 \end{pmatrix} \rightarrow \begin{pmatrix} W_1^1 \\ W_2^1 \\ Y_3 \end{pmatrix} \rightarrow \begin{pmatrix} W_1^1 \\ W_2^1 \\ W_3^1 \end{pmatrix}$$

$$\downarrow \qquad\qquad \downarrow \qquad\qquad \downarrow$$

$$\begin{pmatrix} W_1^2 \\ Y_2 \\ Y_3 \end{pmatrix} \qquad \begin{pmatrix} W_1^1 \\ W_2^2 \\ Y_3 \end{pmatrix} \qquad \begin{pmatrix} W_1^1 \\ W_2^1 \\ W_3^2 \end{pmatrix}$$

Ausgehend von Y entstehen in der Waagerechten jeweils die aktualisierten Versionen von Y, die aus der Verwendung des ersten Teilintervalls W_i^1 gebildet werden. Jeweils senkrecht darunter werden die abzuspeichernden Quader unter Verwendung des zweiten Teilintervalles W_i^2 gebildet. Zu beachten ist dabei, daß die hier auftretenden W_i^k nicht notwendigerweise mit denen aus Beispiel 2.5.1 übereinstimmen müssen, da diese in unserem modifizierten Verfahren in Abhängigkeit von bereits veränderten Komponenten Y_i berechnet werden.

Mit Algorithmus 2.14 geben wir die vollständige algorithmische Beschreibung unseres Verfahrens mit der eben skizzierten speziellen Aufteilungsstrategie bezeichnet mit NewtonStepMGS $(Y, L, \widetilde{f}, p, mode)$ an. Da wir die bei den Aufteilungen entstehenden zusätzlichen Quader abspeichern wollen, benötigen wir die zusätzlichen Parameter L und $\widetilde{f}$, den letzteren zur Durchführung des abgewandelten Mittelpunktstests vor dem Listeneinhängen.

Algorithmus 2.14: NewtonStepMGS $(Y, L, \widetilde{f}, p, mode)$
Modifizierter Gauß-Seidel-Schritt

1. $H := \nabla^2 F(Y)$; $M := m(H)$; $c := m(Y)$; $g := \nabla f(c)$.
2. **if** $mode_{precon} = 0$ **or** M singulär **then** $R := I$
   else $R := M^{-1}$.
3. $A := R \cdot H$; $b := R \cdot g$.
4. **for** $i := 1$ **to** n **do**
   if $0 \notin A_{ii}$ **then**
   (a) IStepGS $(Y, A, b, c, i, W_i^1, W_i^2, p)$.
   (b) **if** $p = 0$ **then return**.
   (c) $Y_i := W_i^1$.
5. **for** $i := 1$ **to** n **do**
   if $0 \in A_{ii}$ **then**
   (a) IStepGS $(Y, A, b, c, i, W_i^1, W_i^2, p)$.
   (b) **if** $p = 0$ **then return**.
   (c) **if** $p = 2$ **then**
      i. $Y_i := W_i^2$; $\underline{F_Y} := \inf F(Y)$; $p := 1$.
      ii. **if** $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
   (d) $Y_i := W_i^1$.

In Schritt 4 von Algorithmus 2.14 werden zunächst wie in Algorithmus 2.13 alle Komponenten behandelt, für die $0 \notin A_{ii}$ gilt, danach in Schritt 5 die restlichen Komponenten. Dabei wird Y_i sowohl für $p = 1$ als auch für $p = 2$ aktualisiert. Für diesen zweiten Fall wird jedoch zunächst das Restintervall zur Bildung eines neuen Quaders verwendet, der in L eingehängt wird, falls er der Mittelpunktsbedingung genügt. Es wird außerdem der Ergebnisparameter p auf 1 gesetzt, da Algorithmus 2.14 in der oben beschriebenen Form stets nur einen Ergebnisquader Y mit $Y = \emptyset$ für $p = 0$ liefert. Der Parameter *mode* hat die gleiche Funktion wie in Algorithmus 2.13.

Die strikte Trennung der beiden Phasen für $0 \notin A_{ii}$ und $0 \in A_{ii}$ kann durchaus auch aufgehoben werden. Dies hätte allerdings zur Folge, daß die Komponentendurchmesser der jeweils abzuspeichernden Quader möglicher-

weise größer sind, als bei obigem Verfahren. Auch könnten durchaus in Schritt 5 von Algorithmus 2.14 nochmals die Komponenten aktualisiert werden, für die $0 \notin A_{ii}$ gilt. Diese Überlegungen sind jedoch rein heuristischer Natur, und die Resultate derartiger Varianten von Algorithmus 2.14 hängen im allgemeinen stark von der Bauart der zu minimierenden Funktion f und deren unmittelbaren Einfluß auf die Intervallauswertungen der Funktion, des Gradienten und der Hessematrix ab, was auch die Testresultate in Kapitel 4 verdeutlichen.

2.5.4 Allgemeine Definition von Präkonditionierern

Die weiteren Varianten des Intervall-Newton-Schrittes, mit denen wir uns in den nachfolgenden Abschnitten beschäftigen werden, machen Gebrauch von speziellen Präkonditionierern. Aus diesem Grund erläutern wir nun zunächst die allgemeine Definition und Klassifizierung von Präkonditionierern sowie einige ihrer zentralen Eigenschaften. Die grundlegenden theoretischen und auch praktischen Untersuchungen bezüglich der Wahl von optimalen Präkonditionierern im Hinblick auf das Intervall-Gauß-Seidel-Verfahren stammen von Kearfott [23], Hu [17] und Novoa [45]. Aus diesem Grund wollen wir uns im folgenden an ihre in [25] zusammengefaßten Begriffsbildungen halten.

2.5.4.1 Zur Notwendigkeit spezieller Präkonditionierer

In den meisten Arbeiten zum Thema Intervall-Newton-Verfahren wird üblicherweise die inverse Mittelpunktsmatrix der Jacobimatrix von G als Präkonditionierermatrix verwendet, mit dem Ziel, die Intervallmatrix A des Verfahrens der Einheitsmatrix anzunähern. Kearfott weist jedoch in [23] darauf hin, daß diese Art der Präkonditionierung nicht immer funktioniert. Die Mittelpunktsmatrix kann etwa nicht invertierbar oder sehr schlecht konditioniert sein, und es ist sogar möglich, daß selbst im gut konditionierten Fall keine besseren Ergebnisse erzielt werden als mit anderen Präkonditionierern. Unter einem besseren Ergebnis verstehen wir dabei einen kleineren Ergebnisquader. Wir wollen diesen Sachverhalt an einem einfachen Beispiel verdeutlichen.

Beispiel 2.5.3 Wir wollen einen Gauß-Seidel-Schritt für das System

$$H \cdot (c - Y) = g(c)$$

mit

$$H = \begin{pmatrix} [1.8, 2.2] & [2, 4] \\ [3.8, 4.2] & [4, 6] \end{pmatrix}, \quad Y = \begin{pmatrix} [1, 2] \\ [1, 2] \end{pmatrix}, \quad c = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}, \quad \text{und} \quad g(c) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

in dreistelliger Intervallarithmetik durchführen. Als Präkonditionierermatrix verwenden wir zunächst die inverse Mittelpunktsmatrix

$$R = m(H)^{-1} = \begin{pmatrix} -2.5 & 1.5 \\ 2.0 & -1.0 \end{pmatrix}.$$

Somit ist

$$A = R \cdot H = \begin{pmatrix} [0.2, 1.8] & [-4, 4] \\ [-0.6, 0.6] & [-2, 4] \end{pmatrix}, \quad \text{und} \quad b = R \cdot g(c) = \begin{pmatrix} -2 \\ 2 \end{pmatrix}.$$

Ein Gauß-Seidel-Schritt für die erste Komponente ergibt

$$\begin{aligned} W_1 &= c_1 - (b_1 + A_{12}(Y_2 - c_2))/A_{11} \\ &= 1.5 - (-2 + [-4, 4] \cdot [-0.5, 0.5])/[0.2, 1.8] \\ &= 1.5 - [-20, 0] \\ &= [1.5, 21.5] \end{aligned}$$

und damit $d(W_1) = 20$. Wählen wir hingegen

$$R = \begin{pmatrix} 0 & 0.264 \\ 0 & 0.25 \end{pmatrix}$$

als Präkonditionierermatrix, so ist

$$A = R \cdot H = \begin{pmatrix} [1, 1.11] & [1.05, 1.59] \\ [0.95, 1.05] & [1, 1.5] \end{pmatrix}, \quad \text{und} \quad b = R \cdot g(c) = \begin{pmatrix} 0.264 \\ 0.25 \end{pmatrix}.$$

Der Gauß-Seidel-Schritt für die erste Komponente ergibt dann

$$\begin{aligned} W_1 &= c_1 - (b_1 + A_{12}(Y_2 - c_2))/A_{11} \\ &= 1.5 - (0.264 + [1.05, 1.59] \cdot [-0.5, 0.5])/[1, 1.11] \\ &= 1.5 - [-0.531, 1.06] \\ &= [0.44, 2.04] \end{aligned}$$

und damit $d(W_1) = 1.6$. Diese zweite Präkonditionierermatrix liefert somit einen deutlich kleineren Durchmesser.

Es gilt also, eine Methode zur Bestimmung möglichst optimaler Präkonditionierer zu finden. Da für die Berechnung der i-ten Komponente Y_i im Gauß-Seidel-Schritt nur die Werte der i-ten Zeile der Matrix A benötigt werden, kann man die Berechnung optimaler Präkonditionierer zeilenweise durchführen. Algorithmus 2.12 muß deshalb geringfügig modifiziert werden, um diese Präkonditionierung mitzubearbeiten. Die neue Form, die wir mit IStepPGS $(Y, H, g, c, R_i, i, W_i^1, W_i^2, p)$ bezeichnen, wird durch Algorithmus 2.15 beschrieben und liefert die Ergebnisse W_i^1, W_i^2 und p, wie in Algorithmus 2.12 beschrieben.

Algorithmus 2.15: IStepPGS $(Y, H, g, c, R_i, i, W_i^1, W_i^2, p)$
Gauß-Seidel-Schritt für i-te Komponente mit R_i-Übergabe

1. $A_i := R_i \cdot H; \quad b_i := R_i \cdot g.$
2. $$(W_i^1 \,|\, W_i^2) := \left(c_i - \left(b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}(Y_j - c_j) \right) \Big/ A_{ii} \right) \cap Y_i.$$
3. **if** $W_i^1 = W_i^2 = \emptyset$ **then** $p := 0$
   else if $W_i^2 = \emptyset$ **then** $p := 1$
   else $p := 2$.

In den nachfolgenden Abschnitten verwenden wir für die entsprechenden Vektoren und Matrizen aus dem Gauß-Seidel-Schritt stets die Bezeichnungen aus Algorithmus 2.15.

2.5.4.2 Definition und Klassifizierung von Präkonditionierern

Wir wollen nun einige Bezeichnungen und wichtige Eigenschaften zusammenstellen und erläutern (vgl. dazu auch [25]).

Definition 2.5.2 *Eine Zeile R_i der Matrix $R \in \mathbb{R}^{n \times n}$, die zur Multiplikation von Gleichung (2.10) verwendet wird, heißt Präkonditionierer-Zeile oder auch kurz Präkonditionierer.*

Lemma 2.5.5 *Gilt für zwei Präkonditionierer $R_i^1, R_i^2 \in \mathbb{R}^n$ die Beziehung $R_i^1 = \lambda R_i^2$ mit $0 \neq \lambda \in \mathbb{R}$, so liefert Algorithmus 2.15 für beide das gleiche Ergebnis.*

Beweis: Der Faktor λ steckt in b_i und A_i, so daß er sowohl im Zähler als auch im Nenner des Quotienten in Algorithmus 2.15 vorkommt. □

Definition 2.5.3 *Zwei Präkonditionierer $R_i^1, R_i^2 \in \mathbb{R}^n$ heißen äquivalent, wenn gilt $R_i^1 = \lambda R_i^2$ für $0 \neq \lambda \in \mathbb{R}$.*

Definition 2.5.4 *R_i heißt C-Präkonditionierer, wenn gilt $0 \notin A_{ii}$.*
R_i heißt S-Präkonditionierer, wenn gilt $0 \in A_{ii}$ und $0 \neq A_{ii}$.

Das C in Definition 2.5.4 steht für *contracting*, das S für *splitting*. Die beiden Kürzel weisen somit auf das Ergebnis bei der Division im Gauß-Seidel-Schritt hin, das für C-Präkonditionierer aus einem einzelnen Intervall und für S-Präkonditionierer aus einem Paar von Intervallen besteht. Trotz der Äquivalenz der Präkonditionierer im $\mathbb{R}^1$, können wir im Eindimensionalen zumindest beispielhaft ihre Wirkung demonstrieren.

Beispiel 2.5.4 Wir betrachten die Funktion $g(y) = y^2 - 4$. Verwenden wir $Y = [1, 2.5]$, so ist die 1 ein C-Präkonditionierer. Es ist

$$\begin{aligned} W &= m(Y) - g(m(Y))/g'(Y) \\ &= 1.75 + 0.9375/[2,5] \\ &= [1.9375, 2.21875] \end{aligned}$$

und damit $d(W) = d(W \cap Y) < d(Y)$.

Verwenden wir $Y = [-3, 3]$, so ist die 1 ein S-Präkonditionierer. Es ist

$$\begin{aligned} W &= m(Y) - g(m(Y))/g'(Y) \\ &= 0 + 4/[-6,6] \\ &= [-\infty, -\frac{2}{3}] \cup [\frac{2}{3}, \infty] \end{aligned}$$

und damit $d(W \cap Y) = d([-3, -\frac{2}{3}]) + d([\frac{2}{3}, 3]) < d(Y)$.

Dieses Beispiel deutet bereits auf das Ziel für Dimensionen $n > 1$ hin. Es sollen nämlich jeweils C- bzw. S-Präkonditionierer R_i verwendet werden, die bewirken, daß $d(W_i \cap Y_i) < d(Y_i)$ gilt. Im folgenden geben wir die Definitionen für in diesem Sinne optimale Präkonditionierer an.

Definition 2.5.5 *Ein C-Präkonditionierer R_i heißt*

D-optimal, wenn er $d(W_i)$ minimiert,
L-optimal, wenn er $\underline{W_i}$ maximiert,
R-optimal, wenn er $\overline{W_i}$ minimiert.

Bemerkung: Wir weichen hier mit der Bezeichnung D-optimal von der Notation in [25] ab, wo ein W für *width* anstelle von D für Durchmesser verwendet wird.

In Abbildung 2.5 sind mit den Buchstaben D, L und R der Durchmesser und die Punkte des Intervalls W_i gekennzeichnet, deren Optimierung in Definition 2.5.5 zur Klassifizierung der C-Präkonditionierer herangezogen wird.

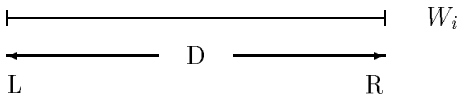


Abbildung 2.5: Optimale C-Präkonditionierer

Definition 2.5.6 *Sei* $U = [\underline{U}, \overline{U}] := -(b_i + \sum_{j=1, j\neq i}^{n} A_{ij}(Y_j - c_j))$ *der Zähler im Quotienten des Gauß-Seidel-Schrittes für die i-te Komponente und* $W_i = (W_i^1 \,|\, W_i^2)$*, dann heißt ein S-Präkonditionierer* R_i

D-optimal, wenn er $d_\sqcup(W_i)$ *maximiert (für* $\underline{U} > 0$ *und* $\underline{A_{ii}} < 0 < \overline{A_{ii}}$*),*
L-optimal, wenn er $\overline{W_i^1}$ *minimiert (für* $\underline{U} > 0$ *und* $\underline{A_{ii}} < 0 \leq \overline{A_{ii}}$*),*
R-optimal, wenn er $\underline{W_i^2}$ *maximiert (für* $\underline{U} > 0$ *und* $\underline{A_{ii}} \leq 0 < \overline{A_{ii}}$*).*

Mit Abbildung 2.6, in der mit den Buchstaben D, L und R der Durchmesser und die Punkte des Intervalls W_i gekennzeichnet sind, deren Optimierung in Definition 2.5.5 zur Klassifizierung herangezogen wird, wollen wir auch im Falle der S-Präkonditionierer die Definition nochmals verdeutlichen.

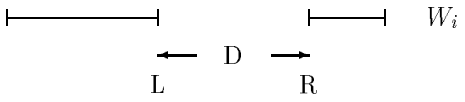


Abbildung 2.6: Optimale S-Präkonditionierer

Bevor wir uns einem Beispiel für verschiedene C-Präkonditionierer zuwenden, wollen wir der Vollständigkeit halber noch eine weitere Definition angeben.

Definition 2.5.7 *Ein Präkonditionierer* R_i *heißt pivotisierender Präkonditionierer, wenn gilt* $R_i = \lambda \cdot e^k$ *mit* $\lambda \neq 0$*, wobei* e^k *den k-ten Einheitsvektor bezeichnet.*

Beispiel 2.5.5 Anhand der Berechnung der ersten Komponente durch den Gauß-Seidel-Schritt für die fast-lineare Funktion von Brown mit Dimension 5

wollen wir die Verwendung verschiedener C-Präkonditionierer und des inversen Mittelpunkts-Präkonditionierers demonstrieren. Es sei

$$g(y) = \begin{pmatrix} 2y_1 + y_2 + y_3 + y_4 + y_5 - 6 \\ y_1 + 2y_2 + y_3 + y_4 + y_5 - 6 \\ y_1 + y_2 + 2y_3 + y_4 + y_5 - 6 \\ y_1 + y_2 + y_3 + 2y_4 + y_5 - 6 \\ y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5 - 1 \end{pmatrix} \quad \text{und} \quad Y = \begin{pmatrix} [0,2] \\ [0.5,1.1] \\ [0.8,1.2] \\ [0.9,1.5] \\ [-2,2] \end{pmatrix}.$$

Dann ist

$$c = m(Y) = \begin{pmatrix} 1 \\ 0.8 \\ 1 \\ 1.2 \\ 0 \end{pmatrix}, \quad g(c) = \begin{pmatrix} -1 \\ -1.2 \\ -1 \\ -0.8 \\ -1 \end{pmatrix}, \quad \text{und}$$

$$H = J_g(Y) = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ [-3.96,3.96] & [-7.2,7.2] & [-6,6] & [-5.28,5.28] & [0,3.96] \end{pmatrix}.$$

Mit L-optimalem C-Präkonditionierer $R_1 = (1,0,0,-1,0)$ ergibt sich

$Y_1 \cap W_1 = [0.9, 1.5]$.

Mit R-optimalem C-Präkonditionierer $R_1 = (1,-1,0,0,0)$ ergibt sich

$Y_1 \cap W_1 = [0.5, 1.1]$.

Mit D-optimalem C-Präkonditionierer $R_1 = (1,0,-1,0,0)$ ergibt sich

$Y_1 \cap W_1 = [0.8, 1.2]$.

Mit dem Präkonditionierer $R_1 \approx (0.8,-0.2,-0.2,-0.2,-0.101)$, der ersten Zeile von $R = m(H)^{-1}$, ergibt sich

$Y_1 \cap W_1 \approx [-0.3542, 2.684]$.

Dieses Beispiel zeigt bereits, daß das beste Ergebnis durch Verwendung *aller* Präkonditionierer und Schneiden der Ergebnisse erzielt werden könnte, was allerdings ein zu aufwendiges Verfahren wäre. Allgemein gilt

Lemma 2.5.6 *Seien $W_i^{C_L}$, $W_i^{C_R}$, $W_i^{S_L}$ bzw. $W_i^{S_R}$ die unter Verwendung eines L-optimalen C-Präkonditionierers, eines R-optimalen C-Präkonditionierers, eines L-optimalen S-Präkonditionierers bzw. eines R-optimalen S-Präkonditionierers berechneten Ergebnisse von Algorithmus 2.15, so gilt für das mit beliebigem Präkonditionierer berechnete Ergebnis W_i*

$$d(W_i^{C_L} \cap W_i^{C_R} \cap W_i^{S_L} \cap W_i^{S_R} \cap Y_i) \leq d(W_i \cap Y_i).$$

Beweis: Siehe [17]. □

2.5.4.3 Existenzaussagen

Die folgenden Sätze geben notwendige und hinreichende Bedingungen für die Existenz von C- bzw. R-Präkonditionierern. Für die Matrizen H, R und A gelte jeweils $A, R, H \in \mathbb{R}^{n\times n}$. Außerdem bezeichne $e^k \in \mathbb{R}^n$ den k-ten Einheitsvektor.

Satz 2.5.7 *Es existiert genau dann ein C-Präkonditionierer R_i, wenn mindestens ein Element der i-ten Spalte H_{*i} von $H = J_g(Y)$ nicht die Null enthält.*

Beweis: „$\Rightarrow$“: Es existiere ein C-Präkonditionierer R_i, und wir nehmen an, daß gilt $0 \in H_{ji}$ für $1 \le j \le n$. Dann ist aber $A_{ii} = R_i \cdot H_{*i}$, und es gilt $0 \in A_{ii}$ im Widerspruch zur Definition des C-Präkonditionierers.

„$\Leftarrow$“: Sei k beliebig mit $1 \le k \le n$ und $0 \notin H_{ki}$. Dann ist e^k ein C-Präkonditionierer, denn $A_{ii} = R_i \cdot H_{*i} = e^k \cdot H_{*i} = H_{ki}$ und damit $0 \notin A_{ii}$. □

Satz 2.5.8 *Es existiert genau dann ein S-Präkonditionierer R_i, wenn die i-te Spalte H_{*i} von $H = J_g(Y)$ eine der folgenden Bedingungen erfüllt:*

1. *Mindestens ein Element der Spalte enthält die Null, ist aber ungleich Null.*
2. *Mindestens zwei Elemente der Spalte enthalten die Null nicht, und mindestens eines davon hat einen nichtverschwindenden Durchmesser.*

Beweis: „$\Rightarrow$“: Es existiere ein S-Präkonditionierer R_i, und wir nehmen an, daß weder 1) noch 2) gilt. Also ist

a) $H_{ji} = 0$ für alle j mit $0 \in H_{ji}$,

und entweder

b1) es gibt nur ein k mit $0 \notin H_{ki}$ oder
b2) $d(H_{ji}) = 0$ für alle $1 \le j \le n$.

Gilt a) für den Index k und b1), so folgt sofort

$$0 \notin A_{ii} \text{ für } R_{ik} \neq 0 \quad \text{und} \quad 0 = A_{ii} \text{ für } R_{ik} = 0,$$

so daß R_i kein S-Präkonditionierer sein kann.
Gilt a) und b2), so folgt sofort daß auch $d(A_{ii}) = d(R_i \cdot H_{*i}) = 0$ gilt, so daß R_i kein S-Präkonditionierer sein kann.

„⇐“: Gilt $0 \in H_{ki} \neq 0$, dann ist e^k ein S-Präkonditionierer, da $0 \in A_{ii} = R_i \cdot H_{*i} = e^k \cdot H_{*i} = H_{ki} \neq 0$.
Gilt $0 \notin H_{ki}$, $0 \notin H_{li}$ und $d(H_{ki}) \neq 0$, dann existiert eine reelle Konstante $\lambda \neq 0$ mit

$$0 = m(H_{ki}) + \lambda m(H_{li}) \in H_{ki} + \lambda H_{li}.$$

Wegen $d(H_{ki}) \neq 0$ gilt auch $d(H_{ki} + \lambda H_{li}) \neq 0$. Somit ist $R_i = e^k + \lambda e^l$ ein S-Präkonditionierer. □

2.5.5 Berechnung spezieller Präkonditionierer

In diesem Abschnitt beschäftigen wir uns nun speziell mit der Berechnung von D-optimalen C-Präkonditionierern und von pivotisierenden S-Präkonditionierern. In [25] wurde eine spezielle Kombination beider Verfahren im Rahmen des Intervall-Gauß-Seidel-Verfahrens als gute Methode zur Lösung allgemeiner nichtlinearer Gleichungssysteme angegeben. In einer modifizierten Form werden wir diese Methode in unserem globalen Optimierungsverfahren einsetzen, den entsprechenden Algorithmus werden wir in Abschnitt 2.5.6 behandeln. Im folgenden verwenden wir die in [25] eingeführte abkürzende Notation

$$j' = \begin{cases} j & \text{für } j < i \\ j+1 & \text{für } j \geq i \end{cases} \tag{2.15}$$

für den Index j in Abhängigkeit von i. Außerdem seien A, b, c, W_i und

$$U = [\underline{U}, \overline{U}] := -(b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}(Y_j - c_j))$$

stets wie in Algorithmus 2.15 definiert.

2.5.5.1 D-optimale C-Präkonditionierer

Satz 2.5.9 *Exisitiert ein C-Präkonditionierer R_i, der*

$$\min_{\substack{R_k \\ \underline{A_{kk}}=1}} d(U) \tag{2.16}$$

löst, so ist R_i ein D-optimaler C-Präkonditionierer.

Beweis: Siehe [23]. □

Satz 2.5.10 *Ist $v \in \mathbb{R}^{3n-1}$ eine optimale Lösung von*

$$\left.\begin{array}{l} \min \sum_{j=1}^{n-1} d(Y_{j'}) \cdot v_j \quad \text{unter den Nebenbedingungen} \\ v_j \geq -\sum_{l=1}^{n} v_{l+n-1} \cdot \underline{H_{lj'}} + \sum_{l=1}^{n} v_{l+2n-1} \cdot \overline{H_{lj'}}, \quad j = 1, \ldots, n-1 \\ v_j \geq +\sum_{l=1}^{n} v_{l+n-1} \cdot \overline{H_{lj'}} - \sum_{l=1}^{n} v_{l+2n-1} \cdot \underline{H_{lj'}}, \quad j = 1, \ldots, n-1 \\ 1 = +\sum_{l=1}^{n} v_{l+n-1} \cdot \underline{H_{li}} - \sum_{l=1}^{n} v_{l+2n-1} \cdot \overline{H_{li}} \\ v_j \geq 0, \quad j = 1, \ldots, 3n-1 \end{array}\right\} \quad (2.17)$$

und $R_i \in \mathbb{R}^n$ definiert durch

$$R_{il} := v_{l+n-1} - v_{l+2n-1}, \quad l = 1, \ldots, n,$$

dann löst der C-Präkonditionierer R_i das Optimierungsproblem (2.16).

Beweis: Siehe [45]. □

Zur Bestimmung des D-optimalen C-Präkonditionierers muß also mit (2.17) ein lineares Optimierungsproblem gelöst werden. In [45] wird eine speziell für die Anwendung des Simplex-Verfahrens einfachere Formulierung von (2.17) hergeleitet, die mit weniger Aufwand gelöst werden kann.

Satz 2.5.11 *Ist $v \in \mathbb{R}^{3n-1}$ eine optimale Lösung von*

$$\left.\begin{array}{l} \min \sum_{j=1}^{n-1} \left(-\sum_{l=1}^{n} v_{l+n-1} \cdot \underline{H_{lj'}} + \sum_{l=1}^{n} v_{l+2n-1} \cdot \overline{H_{lj'}} + v_j \right) d(Y_{j'}) \\ \text{unter den Nebenbedingungen} \\ v_j \geq \sum_{l=1}^{n} (v_{l+n-1} - v_{l+2n-1}) \cdot \left(\underline{H_{lj'}} + \overline{H_{lj'}} \right), \quad j = 1, \ldots, n-1 \\ 1 = \sum_{l=1}^{n} v_{l+n-1} \cdot \underline{H_{li}} - \sum_{l=1}^{n} v_{l+2n-1} \cdot \overline{H_{li}} \\ v_j \geq 0, \quad j = 1, \ldots, 3n-1 \end{array}\right\} \quad (2.18)$$

und $R_i \in \mathbb{R}^n$ definiert durch

$$R_{il} := v_{l+n-1} - v_{l+2n-1}, \quad l = 1, \ldots, n,$$

dann löst der C-Präkonditionierer R_i das Optimierungsproblem (2.16).

Beweis: Siehe [45]. □

Auch diese Darstellung ermöglicht noch keine einfache Formulierung eines Algorithmus, so daß wir zunächst noch eine Form entwickeln müssen, die dieser Anforderung gerecht wird. Dabei wollen wir auch von der Tatsache Gebrauch machen, daß unser Gauß-Seidel-Schritt auf ein Optimierungsproblem angewendet werden soll, das heißt, daß die Matrix $H = J_G(Y) = \nabla^2 F(Y)$ symmetrisch ist.

Satz 2.5.12 *Seien $v, \widetilde{c} \in \mathbb{R}^{4n-2}$, $\widetilde{A} \in \mathbb{R}^{n \times 4n-2}$ und $\widetilde{b} \in \mathbb{R}^n$. Es sei $H \in I\,\mathbb{R}^{n \times n}$ symmetrisch, und es bezeichne $e^k \in \mathbb{R}^{n-1}$ den k-ten $(n-1)$-dimensionalen Einheitsvektor. Außerdem seien $S \in \mathbb{R}^{n \times n}$ und $p, q \in \mathbb{R}^n$ definiert durch*

$$S = \underline{H} + \overline{H} = 2m(H) \ , \tag{2.19}$$

$$p_l = \sum_{\substack{k=1 \\ k \neq i}}^{n} \underline{H_{lk}} \cdot d(Y_k), \quad l = 1, \ldots, n \ , \tag{2.20}$$

$$q_l = \sum_{\substack{k=1 \\ k \neq i}}^{n} \overline{H_{lk}} \cdot d(Y_k), \quad l = 1, \ldots, n \ . \tag{2.21}$$

Ist v eine Lösung des linearen Optimierungsproblems

$$\left.\begin{array}{l} \min \widetilde{c} \cdot v \\ \widetilde{A} \cdot v = \widetilde{b} \\ v \geq 0 \end{array}\right\} \tag{2.22}$$

mit

$$\widetilde{A}_j = \begin{cases} (\,-e^{jT}\,,\ S_j\ ,\ -S_j\ ,\ e^{jT}\,) & \text{für } j = 1, \ldots, i-1 \\ (\,-e^{jT}\,, S_{j+1}\,, -S_{j+1}\,, e^{jT}\,) & \text{für } j = i, \ldots, n-1 \\ (\ \ 0\ \ ,\ \underline{H_i}\ ,\ -\overline{H_i}\ ,\ 0\ \ ) & \text{für } j = n \end{cases} \tag{2.23}$$

$$\widetilde{c}_j = \begin{cases} d(Y_j) & \text{für } j = 1, \ldots, i-1 \\ d(Y_{j+1}) & \text{für } j = i, \ldots, n-1 \\ -p_{j-n+1} & \text{für } j = n, \ldots, 2n-1 \\ q_{j-2n+1} & \text{für } j = 2n, \ldots, 3n-1 \\ 0 & \text{für } j = 3n, \ldots, 4n-2 \end{cases} \tag{2.24}$$

und $\widetilde{b} = (0, \ldots, 0, 1)^T$*, und ist* $R_i \in \mathbb{R}^n$ *definiert durch*

$$R_{il} := v_{l+n-1} - v_{l+2n-1}, \quad l = 1, \ldots, n,$$

dann löst der C-Präkonditionierer R_i *das Optimierungsproblem (2.16).*

Beweis: Ist $\Phi(v)$ die zu minimierende Funktion in der Darstellung (2.18), so gilt wegen (2.15) und mit (2.20) bzw. (2.21)

$$\begin{aligned}
\Phi(v) &= -\sum_{l=1}^{n} v_{l+n-1} \sum_{j=1}^{n-1} \underline{H_{lj'}} d(Y_{j'}) + \sum_{l=1}^{n} v_{l+2n-1} \sum_{j=1}^{n-1} \overline{H_{lj'}} d(Y_{j'}) \\
&\quad + \sum_{j=1}^{n-1} v_j d(Y_{j'}) \\
&= -\sum_{l=1}^{n} v_{l+n-1} \sum_{\substack{j=1 \\ j \neq i}}^{n} \underline{H_{lj}} d(Y_j) + \sum_{l=1}^{n} v_{l+2n-1} \sum_{\substack{j=1 \\ j \neq i}}^{n} \overline{H_{lj}} d(Y_j) \\
&\quad + \sum_{j=1}^{i-1} v_j d(Y_j) + \sum_{j=i}^{n-1} v_j d(Y_{j+1}) \\
&= -\sum_{l=1}^{n} v_{l+n-1} p_l + \sum_{l=1}^{n} v_{l+2n-1} q_l + \sum_{j=1}^{i-1} v_j d(Y_j) \\
&\quad + \sum_{j=i}^{n-1} v_j d(Y_{j+1}) \\
&= +\sum_{j=1}^{i-1} v_j d(Y_j) + \sum_{j=i}^{n-1} v_j d(Y_{j+1}) - \sum_{j=n}^{2n-1} v_j p_{j-n+1} \\
&\quad + \sum_{j=2n}^{3n-1} v_j q_{j-2n+1} + \sum_{j=3n}^{4n-2} v_j \cdot 0 \\
&= \widetilde{c} \cdot v
\end{aligned}$$

und wir erhalten $\widetilde{c}$ wie in (2.24). Führen wir nun die Schlupfvariablen $v_{3n-1+j},\ j = 1,\ldots,n-1$ auch in den Ungleichungen in (2.18) ein, so folgt mit S aus (2.19) und mit der Symmetrie von S

$$\begin{aligned} 0 &= -v_j + \sum_{l=1}^{n} v_{l+n-1} S_{lj'} - \sum_{l=1}^{n} v_{l+2n-1} S_{lj'} + v_{3n-1+j} \\ &= -v_j + \sum_{l=1}^{n} v_{l+n-1} S_{j'l} - \sum_{l=1}^{n} v_{l+2n-1} S_{j'l} + v_{3n-1+j} \end{aligned}$$

für $j = 1,\ldots,n-1$. Unter Verwendung von (2.15) erhalten wir somit

$$0 = -v_j + \sum_{l=1}^{n} v_{l+n-1} S_{jl} - \sum_{l=1}^{n} v_{l+2n-1} S_{jl} + v_{3n-1+j} =: \widetilde{A}_j \cdot v$$

für $j = 1,\ldots,i-1$ und

$$0 = -v_j + \sum_{l=1}^{n} v_{l+n-1} S_{j+1,l} - \sum_{l=1}^{n} v_{l+2n-1} S_{j+1,l} + v_{3n-1+j} =: \widetilde{A}_j \cdot v$$

für $j = i,\ldots,n-1$. Damit haben wir die Zeilen $\widetilde{A}_j$ der Darstellung (2.23) und $b_j = 0$ jeweils für $1,\ldots,n-1$.
Aus der Nebenbedingungs-Gleichung in (2.18) erhalten wir aufgrund der Symmetrie von H

$$\begin{aligned} 1 &= \sum_{l=1}^{n} v_{l+n-1} \underline{H_{li}} - \sum_{l=1}^{n} v_{l+2n-1} \overline{H_{li}} \\ &= \sum_{l=1}^{n} v_{l+n-1} \underline{H_{il}} - \sum_{l=1}^{n} v_{l+2n-1} \overline{H_{il}} \\ &=: \widetilde{A}_n \cdot v \end{aligned}$$

und damit die Darstellung (2.23) für $\widetilde{A}_n$ und $b_n = 1$. Somit ist (2.22) eine äquivalente Darstellung für (2.18), und mit Satz 2.5.11 folgt die Behauptung für R_i. □

Satz 2.5.12 können wir nun als Grundlage für die Beschreibung eines implementierbaren Algorithmus DCPreCon (Y, H, i, R_i) zur Berechnung eines D-optimalen C-Präkonditionierers R_i für die i-te Zeile im Gauß-Seidel-Schritt

verwenden. Diesen Algorithmus 2.16 werden wir im Rahmen eines Gauß-Seidel-Schrittes mit spezieller Präkonditionierung zusammen mit pivotisierenden S-Präkonditionierern verwenden. Die in Algorithmus 2.16 verwendete Notation SolveLP $(\widetilde{A}, \widetilde{b}, \widetilde{c}, v)$ soll die Durchführung eines beliebigen Algorithmus zur Berechnung der Lösung v des durch $\widetilde{A}$, $\widetilde{b}$ und $\widetilde{c}$ definierten linearen Optimierungsproblems (2.22) symbolisieren.

Algorithmus 2.16: DCPreCon (Y, H, i, R_i)
Bestimmung eines D-optimalen C-Präkonditionierers

1. $S := \underline{H} + \overline{H}$; $\widetilde{A} := 0$; $\widetilde{b} := e^n$; $\widetilde{c} := 0$.
2. **for** $l := 1$ **to** n **do**
$$p_l := \sum_{\substack{k=1 \\ k \neq i}}^{n} \underline{H_{lk}} \cdot d(Y_k); \quad q_l := \sum_{\substack{k=1 \\ k \neq i}}^{n} \overline{H_{lk}} \cdot d(Y_k).$$
3. **for** $j := 1$ **to** $n-1$ **do**
   (a) $\widetilde{A}_{j,j} := -1$; $\widetilde{A}_{j,3n+j-1} := 1$.
   (b) **if** $j < i$ **then** $t := j$ **else** $t := j+1$.
   (c) $\widetilde{c}_j := d(Y_t)$.
   (d) **for** $k := n$ **to** $2n-1$ **do**
   $\widetilde{A}_{j,k} := S_{t,k-n+1}$; $\widetilde{A}_{j,k+n} := -S_{t,k-n+1}$.
4. **for** $k := n$ **to** $2n-1$ **do**
   (a) $\widetilde{A}_{n,k} := \underline{H}_{i,k-n+1}$; $\widetilde{A}_{n,k+n} := -\overline{H}_{i,k-n+1}$.
   (b) $\widetilde{c}_k := -p_{k-n+1}$; $\widetilde{c}_{k+n} := q_{k-n+1}$.
5. SolveLP $(\widetilde{A}, \widetilde{b}, \widetilde{c}, v)$.
6. **for** $l := 1$ **to** n **do** $R_{i,l} := v_{l+n-1} - v_{l+2n-1}$.

2.5.5.2 Pivotisierende S-Präkonditionierer

Prinzipiell können auch S-Präkonditionierer ähnlich wie C-Präkonditionierer berechnet werden (vgl. z. B. [17] oder [25]), am einfachsten jedoch können pivotisierende Präkonditionierer bestimmt werden. Dabei wäre es günstig, alle pivotisierenden Präkonditionierer zu verwenden, was aber zu aufwendig wäre. Eine einfache Alternative hierzu ist das Auflösen der Gleichung

$$H \cdot (Y - c) = -g \tag{2.25}$$

in jeder Zeile für jede Variable. Betrachtet man die m-te Zeile von Gleichung (2.25), so lautet diese

$$H_{m1}(Y_1 - c_1) + H_{m2}(Y_2 - c_2) + \ldots + H_{mn}(Y_n - c_n) + g_m = 0$$

und aufgelöst nach Y_i

$$\begin{aligned} Y_i &= c_i - \left(\sum_{\substack{j=1 \\ j \neq i}}^{n} H_{mj} \cdot (Y_j - c_j) + g_m \right) \Big/ H_{mi} \\ &= c_i - (S_{mi} + g_m) \,/ H_{mi} \end{aligned}$$

mit $S_{mi} = \sum_{\substack{j=1 \\ j \neq i}}^{n} T_{mj}$ und $T_{mj} = H_{mj} \cdot (Y_j - c_j)$.

Führt man diese Vorschrift iterativ für m und i durch, so tritt jedes Element der Matrix H einmal im Nenner auf, was einer Verwendung aller pivotisierenden Präkonditionierer im Gauß-Seidel-Schritt gleichkommt (vgl. auch [43]). Problematisch erweist sich in diesem Zusammenhang allerdings der große Aufwand für die Berechnung der Summe S_{mi}, der prinzipiell dadurch vermindert werden könnte, daß man die S_{mi} nicht durch die komplette Summation, sondern durch

$$S_{m,i+1} := S_{m,i} + T_{m,i} - T_{m,i+1}$$

berechnet. Aufgrund der Intervalloperationen führt dies aber zu unerwünschten Aufblähungseffekten.

Beispiel 2.5.6 Mit $T_{mj} = [-1, 1],\ j = 1, \ldots 3$ und $S_{m2} = T_{m1} + T_{m3} = [-2, 2]$ folgt

$$S_{m3} := S_{m2} + T_{m2} - T_{m3} = [-2, 2] + [-1, 1] - [-1, 1] = [-4, 4],$$

richtig wäre aber

$$S_{m3} = T_{m1} + T_{m2} = [-1, 1] + [-1, 1] = [-2, 2].$$

Unter Verwendung einer speziellen Intervallsubtraktion (vgl. [25], [43] und [45]) kann dieser Effekt jedoch vermieden werden. Für $A, B, X \in I\!I\!R$ definiert man

$$B - A := [\underline{B} - \underline{A}, \overline{B} - \overline{A}], \quad \text{falls gilt} \quad B = X + A.$$

Man verwendet somit die Information, daß B als Summe von A und einem Intervall X entstanden ist.

Mit diesen Vorüberlegungen können wir nun einen Algorithmus angeben, der dieses Auflösen aller Gleichungen für alle Variablen realisiert. Wie bereits beim Gauß-Seidel-Schritt mit inverser Mittelpunkts-Präkonditionierung werden wir eine Version NewtonStepMPivot $(Y, H, c, g, L, \widetilde{f}, p)$ mit dem in Abschnitt 2.5.3 entwickelten modifizierten Aufspaltungsverfahren angeben, was durch das M in der Algorithmusbezeichnung gekennzeichnet wird. Der Algorithmus baut auf dem von Kearfott und Hu in [25, Algorithmus 4.3] vorgeschlagenen Algorithmus auf, der jedoch in der dort angegebenen Form nicht korrekt arbeitet.

Algorithmus 2.17: NewtonStepMPivot $(Y, H, c, g, L, \widetilde{f}, p)$
Verwendung aller pivotisierenden Präkonditionierer

1. **for** $m := 1$ **to** n **do**
   (a) **for** $j := 2$ **to** n **do** $T_j := H_{mj} \cdot (Y_j - c_j)$.
   (b) $\Lambda := g_m + \sum_{j=2}^{n} T_j$.
   (c) **for** $i := 1$ **to** n **do**
      i. $(W_i^1 \,|\, W_i^2) := (c_i - \Lambda / H_{mi}) \cap Y_i$.
      ii. **if** $W_i^1 = W_i^2 = \emptyset$ **then** $p := 0$; **return**.
      iii. **if** $W_i^2 \neq \emptyset$ **then**
         A. $Y_i := W_i^2$; $\underline{F_Y} := \inf F(Y)$.
         B. **if** $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
      iv. $Y_i := W_i^1$
      v. **if** $i < n$ **then**
         A. $T_i := H_{mi} \cdot (Y_i - c_i)$.
         B. $\underline{\Lambda} := \underline{\Lambda} + \underline{T_i} - \underline{T_{i+1}}$.
         C. $\overline{\Lambda} := \overline{\Lambda} + \overline{T_i} - \overline{T_{i+1}}$.

In [25, Algorithmus 4.3] wird die i-Schleife (Schritt (c) in Algorithmus 2.17) fälschlicherweise bereits vor der speziellen Subtraktion abgebrochen. Außerdem fehlt die unbedingt notwendige Abfrage in Schritt v. Weiterhin soll laut [25] in der speziellen Subtraktion der alte Wert für T_i und der neu berechnete Wert für T_{i+1} verwendet werden. Da in der Summe Λ jedoch der alte Wert T_{i+1} auftritt, muß dieser auch wieder abgezogen werden. Schritt

(c)v.A. in Algorithmus 2.17 ermöglicht uns jedoch für T_i in der anschließenden Berechnung von Λ einen neuen Wert mit möglicherweise kleinerem Durchmesser zu verwenden.

2.5.6 Speziell präkonditionierter Gauß-Seidel-Schritt

Die in den beiden letzten Abschnitten vorgestellten Verfahren zur Berechnung und Anwendung von D-optimalen C-Präkonditionierern und pivotisierenden Präkonditionierern lassen sich nun zu einem speziellen Gauß-Seidel-Schritt zusammenfassen.

Algorithmus 2.18: NewtonStepSPGS $(Y, L, \widetilde{f}, p)$
Speziell präkonditionierter Gauß-Seidel-Schritt

1. $H := \boldsymbol{\nabla}^2 F(Y)$; $c := m(Y)$; $g := \boldsymbol{\nabla} f(c)$.
2. $Z := Y$; $\delta := 10^{-6}$; $\tau := 0.8$.
3. NewtonStepMPivot $(Y, H, c, g, L, \widetilde{f}, p)$.
4. **if** $p = 0$ **then return**.
5. **for** $i := 1$ **to** n **do**
   if $d(Y_i) \geq \delta |Y_i|$ **and** $d(Y_i) \geq \tau d(Z_i)$ **then**
   (a) DCPreCon (Y, H, i, R_i).
   (b) IStepPGS $(Y, H, g, c, R_i, i, W_i^1, W_i^2, p)$.
   (c) **if** $p = 0$ **then return**.
   (d) **if** $p = 2$ **then**
      i. $Y_i := W_i^2$; $\underline{F_Y} := \inf F(Y)$; $p := 1$.
      ii. **if** $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
   (e) $Y_i := W_i^1$.

Dieser führt zunächst das modifizierte Verfahren unter Verwendung der pivotisierenden Präkonditionierer durch. Danach wird für die Komponenten Y_i, deren Durchmesser (bzw. deren Werte Z_i vor dem pivotisierenden Teilschritt) die Bedingungen

$$d(Y_i) \geq \delta |Y_i| \quad \text{und} \quad d(Y_i) \geq \tau d(Z_i)$$

erfüllen, ein Gauß-Seidel-Schritt unter Einsatz eines D-optimalen C-Präkonditionierers durchgeführt. Mit $\delta = 10^{-6}$ und $\tau = 0.8$ verwenden wir die in

[25] vorgeschlagenen Werte. Somit bringt Schritt 5 dieses Algorithmus nur im ungünstigsten Fall die Lösung von n linearen Optimierungsproblemen der Gestalt (2.22) für jeweils $4n-2$ Variablen mit sich.

2.5.7 Behandlung von Singularitäten der Hessematrix

Die verschiedenen Modifikationen des Intervall-Newton-Schritts, mit denen wir uns in den letzten Abschnitten beschäftigt haben, bezeichnet man üblicherweise dann als effizient, wenn für den Quader Y^N nach der Durchführung des Schrittes gilt

$$d(Y_i^N) < d(Y_i), \quad \text{für } 1 \leq i \leq n. \tag{2.26}$$

Andernfalls tritt eine Verbesserung der Einschließung Y bzw. Y^N erst durch die innerhalb unseres grundlegenden Algorithmus erfolgende Bisektion ein. Speziell wenn die Matrix $H = \boldsymbol{\nabla}^2 F(Y) = J_G(Y)$ in der Nähe des Minimums bzw. der Nullstelle von $g = \boldsymbol{\nabla} f$ schlecht konditioniert oder sogar singulär ist, gilt (2.26) im allgemeinen nicht. Damit würde sich das globale Optimierungsverfahren trotz des Intervall-Newton-Schrittes auf ein Bisektionsverfahren mit Mittelpunkts-, Monotonie- und Konkavitätstest reduzieren.

Um diesen Mangel zu beseitigen, wollen wir ein Verfahren einsetzen, das versucht, die Singularitäten der Matrix H abzuspalten. Dabei machen wir von der Tatsache Gebrauch, daß das klassische, punktmäßige Newton-Verfahren auch für mehrfache bzw. singuläre Nullstellen konvergiert, wenn auch langsam (vgl. [10]). Wenn also die Bedingung (2.26) nach der Durchführung eines Intervall-Newton-Schritts für kein i erfüllt ist, so führen wir, ausgehend vom Mittelpunkt des Quaders Y, eine klassische Newton-Iteration gemäß

$$\begin{aligned} y^0 &:= m(Y) \\ y^k &:= y^{k-1} - (J_g(y^{k-1}))^{-1} \cdot g(y^{k-1}), \quad k = 1, \ldots, K \end{aligned}$$

durch. Wenn diese durchführbar ist und die Folge der y^k innerhalb von Y konvergiert, so können wir den Quader Y zerlegen in die Umgebung um y^K und die Restquader, die durch komponentenweises, sukzessives „Herausschneiden“ der Umgebungsquader entstehen. Die Methodik zur Bestimmung der Restquader ähnelt der speziellen Aufteilung im modifizierten Gauß-Seidel-Schritt aus Abschnitt 2.5.3 und wird durch Algorithmus 2.19 beschrieben. Dabei verwenden wir für die Durchführung eines geeigneten reellen Newton-Verfahrens zur Anwendung auf singuläre Nullstellen die Notation RSNewton $(Y, c, conv)$. Wenn das Verfahren mit c einen Punkt zur

Abspaltung liefert, wird dies durch $conv = true$ angezeigt. Der Parameter für die Berechnung des Umgebungsquaders um c wurde aufgrund der Testergebnisse zu $\mu = 10^{-4}$ gewählt.

Algorithmus 2.19: CutBox $(Y, L, \widetilde{f})$
Abspaltung von Singularitäten der Hessematrix

1. RSNewton $(Y, c, conv)$.
2. **if not** $conv$ **then return**.
3. $\mu := 10^{-4}$; $Z := ([1-\mu, 1+\mu] \cdot c) \cap Y$.
4. **for** $i := 1$ **to** n **do**
   (a) **if** $\underline{Y_i} \neq \underline{Z_i}$ **then**
      i. $u := \overline{Y_i}$; $\overline{Y_i} := \underline{Z_i}$.
      ii. **if** $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
      iii. $\overline{Y_i} := u$.
   (b) **if** $\overline{Y_i} \neq \overline{Z_i}$ **then**
      i. $\underline{Y_i} := \overline{Z_i}$.
      ii. **if** $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
   (c) $Y_i := Z_i$.

Algorithmus 2.19 liefert mit $Y = Z$ den Umgebungsquader um die potentielle Singularität von H zurück, während die Restquader in die Arbeitsliste L abgespeichert werden. Zum besseren Verständnis des Verfahrens sind in Abbildung 2.7 für den Fall $Y \in I\!I\!R^2$ die entstehenden Restquader Y^i, $i = 1, \ldots, 4$ und der Umgebungsquader Z skizziert.

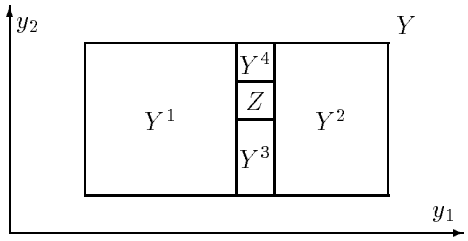


Abbildung 2.7: Aufspaltung von Y in Z und Y^1 bis Y^4

2.5.8 Zusammenfassung der verwendeten Varianten

Zum Abschluß des Abschnitts über den Intervall-Newton-Schritt geben wir nun noch den im Rahmen unseres globalen Optimierungsverfahrens eingesetzten zusammenfassenden Algorithmus 2.20 an, der abhängig vom Parameter *mode* eine der in den letzten Abschnitten beschriebenen Formen des Newton-Schrittes durchführt und im Anschluß daran eventuell versucht, Singularitäten der Hessematrix abzuspalten.

Algorithmus 2.20: NewtonStep $(Y, V^1, V^2, L, \widetilde{f}, p, mode)$
Gesamtform des Intervall-Newton-Schritts

1. $V^1 := Y$.
2. **if** $mode_{\text{precon}} = 2$ **then**
   NewtonStepSPGS $(V^1, L, \widetilde{f}, p)$
   else if $mode_{\text{multigaps}} = 0$ **then**
   NewtonStepGS $(V^1, V^2, p, mode)$
   else
   NewtonStepMGS $(V^1, L, \widetilde{f}, p, mode)$.
3. **if** $mode_{\text{cutbox}} \neq 0$ **and** $V^1 = Y$ **then**
   CutBox $(V^1, L, \widetilde{f})$.

Der Quader Y wird von Algorithmus 2.20 nicht verändert, da dieser für die im nachfolgenden Abschnitt beschriebenen Randbehandlungen noch benötigt wird. CutBox wird nur aufgerufen, wenn keine Komponente von Y verbessert werden konnte. Als Ergebnis von NewtonStep wird in Abhängigkeit von *mode*

- V^1 und V^2 (mit $p = 2$),
- nur V^1 (mit $p = 1$) oder
- kein Quader (mit $p = 0$)

geliefert. In Tabelle 2.1 werden die Werte der Komponenten von *mode* erläutert, die im Intervall-Newton-Schritt eine Bedeutung haben.

$mode_{\text{precon}}$	$= 0$	Keine Präkonditionierung ($R = I$)
$mode_{\text{precon}}$	$= 1$	Präkonditionierung mit $R = (m(H))^{-1}$
$mode_{\text{precon}}$	$= 2$	Pivotis. und D-optimale C-Präkonditionierung
$mode_{\text{multigaps}}$	$= 0$	Aufteilung nur in einer Komponente
$mode_{\text{multigaps}}$	$= 1$	Modifizierte Aufteilungstechnik
$mode_{\text{cutbox}}$	$= 0$	Ohne Abspaltung von Singularitäten
$mode_{\text{cutbox}}$	$= 1$	Mit Abspaltung von Singularitäten

Tabelle 2.1: Bedeutung der Komponenten von *mode*

2.6 Randbehandlung

Während im Monotonietest und im Konkavitätstest die Randpunkte des aktuellen Quaders Y, die auch auf dem Rand des Startquaders X liegen, jeweils gesondert behandelt werden, haben wir bis jetzt für den Intervall-Newton-Schritt noch keine solchen Überlegungen angestellt. Abbildung 2.8 zeigt für $X, Y \in I\!I\!R$ einen Fall, in dem der Newton-Schritt angewendet auf $g = f'$ ausgehend von Y ein Intervall $Y^N \subset Y$ liefert, das die Minimalstelle $\underline{X} = \underline{Y}$ nicht mehr enthält. Somit muß diese getrennt behandelt werden.

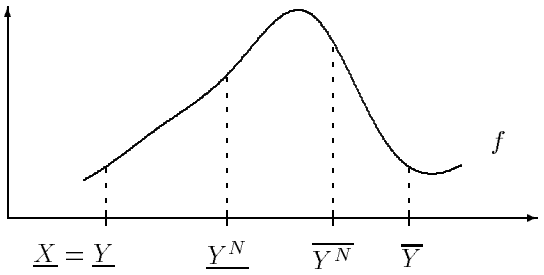


Abbildung 2.8: Randvernachlässigung durch den Newton-Schritt

Im folgenden wollen wir uns mit einigen Arten der Randbehandlung beschäftigen und schließlich einen Algorithmus entwickeln, der in unserem globalen Optimierungsverfahren zum Einsatz kommt.

2.6.1 Getrennte Behandlung des Randes

Während sich im Fall $n = 1$ die Behandlung der Randpunkte noch recht einfach durch die getrennte Bearbeitung der beiden Randpunkte von X, die als Punktintervalle in die Arbeitsliste mit aufgenommen werden, durchführen läßt, wird ein entsprechendes Verfahren für $n > 1$ recht aufwendig. In diesem Fall müssen nicht nur die $2n$ $(n–1)$-dimensionalen Ränder behandelt werden, sondern auch wiederum deren Randquader, da auch die $(n–1)$-dimensionalen Ränder selbst wieder durch das Newton-Verfahren reduziert werden können. Für die Randbehandlung durch separates Bearbeiten aller Randbereiche müssen somit alle k-dimensionalen Ränder mit $k = 0, \ldots, n-1$ gebildet und für die spätere Bearbeitung in der Liste L abgespeichert werden. Um den Aufwand für eine solche Randbehandlung abschätzen zu können, brauchen wir einige Vorbereitungen.

Definition 2.6.1 *Sei $0 \leq k < n$ und $Y \in I\!I\!R^n$ ein n-dimensionaler Quader mit $\underline{Y_i} < \overline{Y_i}$ für $1 \leq i \leq n$. Dann heißt $Z \in I\!I\!R^n$ k-degenerierter Randquader von Y, wenn für die Komponenten Z_i von Z gilt*

- *genau k Komponenten erfüllen die Bedingung $Z_i = Y_i$,*
- *genau $n-k$ Komponenten erfüllen die Bedingung $\underline{Z_i} = \overline{Z_i} = \lambda_i$ mit $\lambda_i \in \{\underline{Y_i}, \overline{Y_i}\}$.*

Satz 2.6.1 *Jeder n-dimensionale Quader $Y \in I\!I\!R^n$ besitzt $\binom{n}{k} 2^{n-k}$ paarweise verschiedene k-degenerierte Randquader Z.*

Beweis: Aufgrund der Definition eines k-degenerierten Randquaders Z gilt $Z_i = Y_i$ für genau k Komponenten von Z, und für die Auswahl dieser k Komponenten von Y gibt es genau $\binom{n}{k}$ Möglichkeiten. Für jede dieser Möglichkeiten wiederum gibt es genau $n-k$ Komponenten, für die entweder $\underline{Z_i} = \overline{Z_i} = \underline{Y_i}$ oder $\underline{Z_i} = \overline{Z_i} = \overline{Y_i}$ gilt. Für die Belegung dieser $n-k$ Komponenten von Z mit den Werten von $\underline{Y_i}$ bzw. $\overline{Y_i}$ gibt es somit 2^{n-k} Möglichkeiten. Damit gibt es insgesamt $\binom{n}{k} 2^{n-k}$ verschiedene k-degenerierte Randquader von Y. □

Wenn wir also den Intervall-Newton-Schritt auf die Quader in der Liste L unseres Algorithmus stets ohne Beachtung von Randverlusten anwenden würden, müßten wir, zusätzlich zum Startquader X selbst, alle k-degenerierten Randquader von X für $k = 0, \ldots, n-1$ mit in die Liste L aufnehmen. Damit wäre die Anzahl α der Quader in L zu Beginn des Verfahrens gegeben durch

$$\begin{aligned} \alpha(L) &= 1 + \sum_{k=0}^{n-1} \binom{n}{k} 2^{n-k} = \sum_{k=0}^{n} \binom{n}{k} 2^{n-k} \\ &= \sum_{k=0}^{n} \binom{n}{k} 2^{n-k} 1^k = (2+1)^n = 3^n. \end{aligned}$$

Die Anzahl der Quader in L zu Beginn des Algorithmus würde somit exponentiell mit der Dimension n wachsen. Aus diesem Grund ist diese Methode der Randbehandlung nicht zu bevorzugen.

2.6.2 Intervall-Newton-Schritt mit Randbehandlung

Eine ganz andere Art der Randbehandlung wurde bereits in [13] vorgeschlagen und in ähnlicher Form auch in [42] verwendet. Dort wird der Newton-Schritt für einen Quader Y, der Randbereiche von X enthält, in einer abgewandelten Form durchgeführt, die als Ergebnis den kleinsten Quader, der sowohl das eigentliche Ergebnis des Newton-Schrittes als auch alle Randbereiche von X, die auch in Y liegen, liefert. Für einen solchen Newton-Schritt mit Randbehandlung entsteht das Ergebnis Y^{NR} somit nach folgender Vorschrift:

1. Berechne $Y^R := Y \cap \partial X \in I\!P I\!R^n$.
2. Berechne Y^N durch einen üblichen Newton-Schritt.
3. Berechne $Y^{NR} := Y^N \cup\!\!\!\!\underline{}\; Y^R \in I\!I\!R^n$.

Dieses Verfahren reduziert allerdings die Effizienz des Newton-Schrittes. Im Extremfall gilt sogar $Y^{NR} = Y$, so daß der Newton-Schritt keinerlei Vorteile bringt.

2.6.3 Spezielle Randbehandlung jeweils nach einem Intervall-Newton-Schritt

In unserem globalen Optimierungsalgorithmus wollen wir ein Verfahren einsetzen, das die Randbehandlung jeweils nach einem Newton-Schritt

durchführt. Dabei werden die Randbereiche von X, die im Quader Y enthalten sind und durch den Newton-Schritt eliminiert werden, als degenerierte Quader in die Liste L aufgenommen. Um sowohl für die Standardform mit zwei Ergebnisquadern als auch für die modifizierten Formen mit einem Ergebnisquader und zusätzlichen in die Liste aufgenommenen Quadern eine vollständige Randbearbeitung möglich zu machen, werden wir unser Verfahren stets auf den Ergebnisquader V^1 anwenden. Dies hat zwar zur Folge, daß eventuell Randquader entstehen, die eigentlich noch Teilquader von V^2 oder von Quadern der aktuellen Liste L sind, und damit gar nicht bearbeitet werden müßten, verringert jedoch drastisch die Anzahl der für die Randquadererzeugung notwendigen Operationen. Unser Verfahren wird nach folgendem Schema arbeiten:

1. Berechne $\mathcal{Z} = Y \cap \partial X \in I\!P I\!R^n$.
2. Berechne $Z^i \in I\!R^n$, für $i = 1, \ldots, m$ mit $\bigcup_{i=1}^{m} Z^i = \mathcal{Z}$.
3. Berechne $Z^i := Z^i \backslash (Z^i \cap V^1)$, für $i = 1, \ldots, m$.
4. Füge die Z^i in die Liste L ein.

Wir fügen also alle Randquader von Y, die nicht mehr in V^1 enthalten sind, als degenerierte Quader in die Arbeitsliste L unseres Verfahrens ein. Dabei werden für einen n-dimensionalen Quader Y nur $(n-1)$-degenerierte Randquader gebildet, während für einen k-degenerierten Randquader auch wieder k-degenerierte Randquader in die Liste aufgenommen werden können.

In Abbildung 2.9 sind für den Fall $n = 2$ die eindimensionalen Randquader des skizzierten Quaders Y, die in die Liste aufgenommen werden, durch die Ziffern 1 bis 6 gekennzeichnet. Wäre Y für den Fall $n = 3$ bereits ein 2-degenerierter Randquader, so müßten die durch A und B gekennzeichneten 2-degenerierten Randquader in die Liste aufgenommen werden.

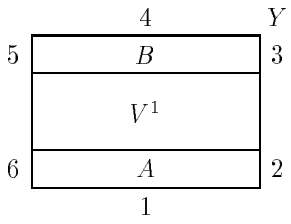


Abbildung 2.9: Randbehandlung für Y und V^1

Wir wollen nun eine algorithmische Beschreibung unseres Verfahrens geben. Da wir, ähnlich wie bei der Behandlung von Singularitäten im letzten Abschnitt, auch hier wieder mehrere Quader aus einem Quader „herausschneiden“ müssen, benötigen wir einen Teilschritt (Algorithmus 2.21) , der dieses realisiert. Im Unterschied zu Algorithmus 2.19 wird das Herausschneiden jedoch nicht bezüglich aller Komponenten erfolgen, da zum Zeitpunkt des Aufrufs innerhalb der Randbehandlung bereits eine Komponente auf ihren Rand reduziert wurde.

Algorithmus 2.21: CutEdge $(Y, V, i, L, \widetilde{f})$
Abspaltung von Randquadern

1. $Z := Y$.
2. **for** $k := 1$ **to** n **do**
   if $k \neq i$ **then**
   (a) **if** $\underline{Z_k} \neq \underline{V_k}$ **then**
      i. $u := \overline{Z_k}$; $\overline{Z_k} := \underline{V_k}$.
      ii. **if** $\underline{F_Z} \leq \widetilde{f}$ **then** $L := L + (Z, \underline{F_Z})$.
      iii. $\overline{Z_k} := u$.
   (b) **if** $\overline{Z_k} \neq \overline{V_k}$ **then**
      i. $\underline{Z_k} := \overline{V_k}$.
      ii. **if** $\underline{F_Z} \leq \widetilde{f}$ **then** $L := L + (Z, \underline{F_Z})$.
   (c) $Z_k := V_k$.

In Algorithmus 2.21 werden die zu bearbeitenden Randbereiche in Quader aufgeteilt und in L abgespeichert. Dabei darf Y selbst nicht verändert werden, da die weitere Randbehandlung immer auf dem Ausgangsquader Y erfolgt. Aus diesem Grund wird mit dem Hilfsquader Z gearbeitet. Außerdem wird die Zerlegung für die Komponente i nicht durchgeführt, da diese beim Eintritt in CutEdge bereits aus einem Punktintervall besteht.

Der vollständige Randbehandlungs-Algorithmus wird durch Algorithmus 2.22 beschrieben. In ihm wird jede Komponente Y_i von Y auf ihre beiden Randpunkte reduziert und die dabei entstehenden Randquader werden entweder durch CutEdge aufgeteilt, falls sie mit V^1 gemeinsame Punkte haben, oder vollständig in die Liste L aufgenommen.

Algorithmus 2.22: HandleEdges $(X, Y, V, L, \widetilde{f}, p)$
Randbehandlung für Y

1. **for** $i := 1$ **to** n **do**
   (a) **if** $\underline{Y_i} = \underline{X_i}$ **then**
      i. $u := \overline{Y_i}$; $\overline{Y_i} := \underline{Y_i}$.
      ii. **if** $Y \cap V \neq \emptyset$ **and** $Y \cap V \neq Y$ **then**
         CutEdge $(Y, V, i, L, \widetilde{f})$.
      else if $p = 0$ **or** $Y \cap V = \emptyset$ **then**
         if $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
      iii. $\overline{Y_i} := u$.
   (b) **if** $\overline{Y_i} = \overline{X_i}$ **then**
      i. $u := \underline{Y_i}$; $\underline{Y_i} := \overline{Y_i}$.
      ii. **if** $Y \cap V \neq \emptyset$ **and** $Y \cap V \neq Y$ **then**
         CutEdge $(Y, V, i, L, \widetilde{f})$.
      else if $p = 0$ **or** $Y \cap V = \emptyset$ **then**
         if $\underline{F_Y} \leq \widetilde{f}$ **then** $L := L + (Y, \underline{F_Y})$.
      iii. $\underline{Y_i} := u$.

Bemerkung: Unter Verwendung dieses speziellen Randbehandlungsalgorithmus haben wir auch den in [13] geschilderten Fall abgedeckt, für den der Intervall-Newton-Schritt die mit X gemeinsamen Randpunkte eines $(n-1)$-degenerierten Quaders Y eliminiert, die nicht auf dem $(n-2)$-degenerierten Rand des Quaders X, wohl aber auf dem Rand von X liegen.

2.7 Der Verifikationsschritt

Als Grundaussage nach Durchführung unseres globalen Optimierungsalgorithmus können wir festhalten, daß die Ergebnisliste $\widehat{L}$ nur Quader enthält, die im Rahmen der vorgegebenen Toleranzangabe und der Genauigkeit der verwendeten Maschinenintervallarithmetik mindestens eine globale Minimalstelle der Zielfunktion f enthalten können. In diesem Abschnitt wollen wir uns nun mit der Frage der Eindeutigkeit einer Minimalstelle innerhalb der berechneten Einschließungsintervalle bzw. Quader der Ergebnisliste $\widehat{L}$ beschäftigen. D. h. wir wollen möglichst für jeden Quader in $\widehat{L}$ nachweisen, daß er genau eine Minimalstelle enthält. Wir werden zunächst die allgemeine Aufgabenstellung erläutern und auf die spezielle Problematik der

Nullstellen-Verifikation im Zusammenhang mit Toleranzangaben, Randgebieten und singulären Nullstellen eingehen. Danach behandeln wir dann den speziellen Fall für globale Minimalstellen. Dabei werden wir eine auf Rump zurückgehende Idee aufgreifen, die den verifizierten Nachweis der positiven Definitheit der Hessematrix ermöglicht.

2.7.1 Verifikation und Epsilon-Aufblähung

In Abschnitt 2.5 haben wir bereits die Fähigkeit des Intervall-Newton-Schrittes erwähnt, die Eindeutigkeit der Nullstelle in Y zu verifizieren. Voraussetzung dafür ist die Erfüllung einer Inklusionsbedingung

$$Y^N \overset{\circ}{\subset} Y \tag{2.27}$$

für den Ergebnisquader Y^N eines Intervall-Newton- oder auch Intervall-Gauß-Seidel-Schrittes angewendet auf Y. Diese Eigenschaft ermöglicht bereits innerhalb des Optimierungsverfahrens durch den Intervall-Newton-Schritt die Feststellung, daß eine eindeutige Nullstelle von $g = \nabla f$ eingeschlossen wurde. Allerdings kann es aus verschiedenen Gründen vorkommen, daß die Bedingung (2.27) für die Quader in der Liste $\widehat{L}$ nicht erfüllt werden kann.

Zunächst einmal können die Quader in $\widehat{L}$ noch mehrere Nullstellen enthalten, so daß die Inklusionsbedingung gar nicht erfüllbar ist. Dies wird dann der Fall sein, wenn die Nullstellen so dicht zusammenliegen, daß unsere im grundlegenden Algorithmus verwendete Toleranzangabe für die Quader in $\widehat{L}$ keine Trennung mehr erwirkt oder der Abstand der Nullstellen unterhalb der relativen Genauigkeit der jeweiligen Rechenanlage liegt. Andererseits ist es auch möglich, daß die Bedingung (2.27) nicht erfüllt wird, obwohl bereits eine eindeutige Nullstelle von g eingeschlosssen wurde. Mögliche Gründe dafür sind:

1. Die Nullstelle ist zwar eindeutig aber singulär (vgl. auch [10]).
2. Bei der intervallmäßigen Berechnung von Y^N treten zu große Überschätzungen auf.
3. Die Nullstelle liegt auf dem Rand von Y.
4. Der Quader Y hat einen zu kleinen Durchmesser oder ist sogar degeneriert.

Aus diesem Grund müssen wir die Quader in der Ergebnisliste einer Nachbehandlung unterziehen.

Im ersten Fall ist je nach Ordnung der Nullstelle ein Nachweis der Eindeutigkeit durch das Erfüllen der Inklusionsbedingung auch durch eine Nachbehandlung nicht möglich. Im zweiten Fall kann versucht werden, durch Verbesserung der Intervallauswertungen doch noch eine Einschließung zu erzielen. Für den dritten und vierten Fall kann man die sogenannte Epsilonaufblähung (vgl. z. B. [51], [52]) einsetzen, die den Quader Y in geringem Maße aufbläht, um damit das Erfüllen der Inklusionsbedingung (2.27) möglich zu machen.

Definition 2.7.1 *Sei $A \in I\mathbb{R}$, $Y \in I\mathbb{R}^n$ und $0 \neq \varepsilon \in \mathbb{R}$. Ferner sei $\eta \in \mathbb{R}$ die kleinste, auf der verwendeten Rechenanlage darstellbare, positive Zahl. Dann heißt*

$$\mathsf{Blow}\,(A,\varepsilon) := \begin{cases} A + [-\varepsilon,\varepsilon] \cdot d(A) & \text{für } d(A) \neq 0 \\ A + [-\eta,\eta] & \text{für } d(A) = 0 \end{cases}$$

Epsilonaufblähung von A und

$$\mathsf{Blow}\,(Y,\varepsilon) := \begin{pmatrix} \mathsf{Blow}\,(Y_1,\varepsilon) \\ \vdots \\ \mathsf{Blow}\,(Y_n,\varepsilon) \end{pmatrix}$$

Epsilonaufblähung von Y.

Ist nun für $Z = \mathsf{Blow}\,(Y,\varepsilon)$ die Bedingung

$$Z^N \overset{\circ}{\subset} Z \tag{2.28}$$

erfüllt, so enthält Z bzw. Z^N eine eindeutige Nullstelle von g.

Problematisch im Zusammenhang mit der Epsilonaufblähung erweist sich die Tatsache, daß in Z Punkte enthalten sind, die nicht in Y liegen. Sofern diese Punkte nämlich auch außerhalb des Startquaders X liegen, haben wir mit Bedingung (2.28) allein noch nicht nachgewiesen, daß die Nullstelle auch in X bzw. im ursprünglichen Y liegt. Dies gilt erst dann, wenn zusätzlich die Bedingung $Z^N \subseteq X$ erfüllt werden kann. Andernfalls kann keine Eindeutigkeitsaussage gemacht werden.

2.7.2 Eindeutigkeitsnachweis für Minimalstellen

Um für einen Quader Y nachzuweisen, daß er eine eindeutige Minimalstelle einschließt, müssen wir noch weitere Überlegungen anstellen. Dabei müssen wir verschiedene Arten von Minimalstellen unterscheiden.

Handelt es sich bei Y um einen 0-degenerierten Quader, also um ein Punktintervall, so ist die Eindeutigkeit der eingeschlossenen Minimalstelle trivial. Als problematisch gestaltet sich der Nachweis für eine Minimalstelle auf dem Rand, die keine stationäre Stelle, also keine Nullstelle von g ist, die aber durch Monotonie- und Konkavitätstest nicht auf einen Randpunkt reduziert werden konnte. In diesem Fall können wir keine Eindeutigkeitsaussage machen.

Der wesentliche, in der Praxis am häufigsten vorkommende Fall ist die Behandlung eines stationären Punktes. In diesem Fall kann durch Nachweis von (2.27) bzw. (2.28) zunächst einmal der Einschluß einer eindeutigen stationären Stelle in Y bzw. Z nachgewiesen werden. Dies genügt jedoch nicht, um eine eindeutige *Minimalstelle* in Y nachzuweisen. Abbildung 2.10 verdeutlicht dies für $n = 1$.

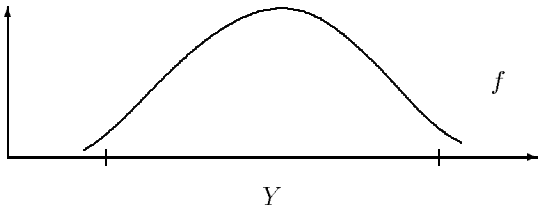


Abbildung 2.10: Zur Verifikation der Eindeutigkeit

Für den Nachweis, daß in einem Quader Y eine eindeutige Minimalstelle eingeschlossen wird, müssen wir zusätzlich zeigen, daß die Hessematrix von f positiv definit

ist. Ein entsprechendes Kriterium, das in [51] für reelle Matrizen angegeben wurde, wollen wir nun leicht modifizieren und auf Intervallmatrizen ausdehnen.

Satz 2.7.1 *Sei $A \in \mathbb{R}^{n\times n}$ symmetrisch und B definiert durch $B := I - \frac{1}{\kappa}A$ mit $\|A\|_\infty \leq \kappa \in \mathbb{R}$. Dann folgt aus $\rho(B) < 1$, daß A positiv definit ist.*

Beweis: Wegen der Symmetrie von A sind alle Eigenwerte von A reell. Sei nun λ, x ein beliebiges Eigenwert/Eigenvektorpaar von A mit $\lambda \in \mathbb{R}$, also $Ax = \lambda x$. Sei ferner $\mu \in \mathbb{R}$ ein Eigenwert von B zum Eigenvektor x, dann gilt

$$\mu x = Bx = (I - \frac{1}{\kappa}A)x = (1 - \frac{\lambda}{\kappa})x.$$

Wegen $|\mu| \leq \rho(B) < 1$ gilt $|1 - \frac{\lambda}{\kappa}| < 1$, und wegen $\rho(A) \leq \|A\|_\infty \leq \kappa$ folgt $\frac{\lambda}{\kappa} \leq 1$ und somit $1 - \frac{\lambda}{\kappa} < 1$, also $\lambda > 0$. □

Lemma 2.7.2 *Sei $H \in I\mathbb{R}^{n\times n}$ und S definiert durch $S := I - \frac{1}{\kappa}H$ mit $\|H\|_\infty \leq \kappa \in \mathbb{R}$. Dann folgt aus $\rho(B) < 1$ für alle $B \in S$, daß alle symmetrischen $A \in H$ positiv definit sind.*

Beweis: Für alle symmetrischen $A \in H$ gilt aufgrund der Inklusionsisotonie

$$B := I - \frac{1}{\kappa}A \in I - \frac{1}{\kappa}H,$$

und wegen $\|A\|_\infty \leq \|H\|_\infty \leq \kappa$ für alle $A \in H$, erfüllen A und B somit die Voraussetzungen von Satz 2.7.1. □

Satz 2.7.3 *Sei $S \in I\mathbb{R}^{n\times n}$ und $Z \in I\mathbb{R}^n$, dann folgt aus $S \cdot Z \overset{\circ}{\subset} Z$, daß $\rho(B) < 1$ für alle $B \in S$.*

Beweis: Siehe [3] bzw. [53]. □

Aufgrund dieser theoretischen Überlegungen können wir nun Algorithmus 2.23 angeben, der überprüft, ob alle symmetrischen Matrizen in einer Intervallmatrix positiv definit sind und dies durch *posdef* = *true* anzeigt.

Algorithmus 2.23: TestPosDef $(H, \varepsilon, posdef)$
Test auf positive Definitheit

1. $\kappa := \|H\|_\infty$; $S := I - \frac{1}{\kappa}H$; $k_{\max} := 10$.
2. **for** $i := 1$ **to** n **do** $Z_i := [-1, 1]$.
3. **for** $k := 1$ **to** $k_{\max}$ **do**
   (a) $U :=$ Blow (Z, ε).
   (b) $Z := S \cdot U$.
   (c) **if** $Z \overset{\circ}{\subset} U$ **then**
   posdef := *true*; **return**.
4. *posdef* := *false*.

Zu beachten ist, daß nach Ausführung von TestPosDef mit *posdef* = *false* lediglich angezeigt wird, daß der Test nicht erfolgreich war. Daraus können wir jedoch nicht schließen, daß in H keine positiv definiten Matrizen enthalten sind.

Den Nachweis der Eindeutigkeit für einen Quader der Ergebnisliste $\widehat{L}$ werden wir unter Verwendung von Algorithmus 2.23 und der vorangegangenen Überlegungen durch den nachfolgenden Algorithmus 2.24 erbringen. Als Kurznotation für die Durchführung eines Intervall-Newton-Schrittes vor dem Test der Inklusionsbedingung (2.28) verwenden wir hier der Einfachheit halber $\mathsf{N}(Z)$.

Algorithmus 2.24: $\mathsf{TestUnique}\,(Y, \varepsilon, \mathit{unique})$
Eindeutigkeitstest

1. **if** $d(Y) = 0$ **then** $\mathit{unique} := \mathit{true}$; **return**
   else $\mathit{unique} := \mathit{false}$.
2. $Z := \mathsf{Blow}(Y)$; $Y := \mathsf{N}(Z)$.
3. **if** $Y \overset{\circ}{\subset} Z$ **then**
   (a) $H := \boldsymbol{\nabla}^2 F(Y)$.
   (b) $\mathsf{TestPosDef}\,(H, \varepsilon, \mathit{posdef})$.
4. **if** $\mathit{posdef} = \mathit{true}$ **then** $\mathit{unique} := \mathit{true}$.
   else $\mathit{unique} := \mathit{false}$.

Bemerkung: In der praktischen Durchführung bzw. Implementierung ist das Erfüllen der Inklusionsbedingung meistens bereits aus den Intervall-Newton-Schritten des Optimierungsverfahrens gesichert, so daß in diesen Fällen auf die Durchführung von Schritt 2 in Algorithmus 2.24 verzichtet werden kann.

2.7.3 Reduzierung der Quader-Anzahl

Im Zusammenhang mit der Toleranzangabe für die Funktionsauswertungen und die Quaderdurchmesser tritt aufgrund der Intervallüberschätzungen und der daraus resultierenden fortgesetzten Bisektion das Phänomen auf, daß die Ergebnisliste $\widehat{L}$ eine sehr große Zahl von Quadern enthalten kann. Dabei können entweder mehrere dieser Quader die gleiche Minimalstelle enthalten (wenn diese auf dem Rand der Quader liegt) oder mehrere dieser Quader decken ein „größeres“ Gebiet um eine Minimalstelle ab, es kann aber nicht entschieden werden ob diese Quader überhaupt die Minimalstelle enthalten. Aus diesem Grund läßt sich aus der Anzahl der Quader in $\widehat{L}$ nicht auf die Anzahl der globalen Minimalstellen schließen.

Wir wollen deshalb im abschließenden Verifikationsschritt unseres globalen Optimierungsalgorithmus eine Methode anwenden, die vor dem Test auf Eindeutigkeit zunächst die Anzahl der Quader reduziert, indem, grob gesprochen, alle nichtdisjunkten Quader in $\widehat{L}$ zu jeweils einem neuen Quader vereinigt werden. Diese Vereinigung wird dabei als Potenzmengenoperation durchgeführt und nicht als Intervallhüllenbildung. Erst am Ende dieses Vereinigungsprozesses wird schließlich wieder ein echter Quader gebildet. Diese Vorgehensweise vermeidet eine zu starke Aufblähung der Vereinigungsquader.

Algorithmus 2.25: CompressVerify $(\widehat{L}, \varepsilon)$
Verifikationsschritt mit Reduzierung der Quaderanzahl

1. $L := \widehat{L}$; $i := 0$; $\widehat{L} := \{\}$.
2. **repeat**
   (a) $i := i + 1$; $Y :=$ Head (L); $L := L - Y$.
   (b) $L^i := Y$; $\mathcal{U}^i := Y$.
   (c) **repeat**
      i. $L^0 := L$; $L := \{\}$; $ready := true$.
      ii. **repeat**
         A. $Y :=$ Head (L^0); $L^0 := L^0 - Y$.
         B. **if** $Y \cap \mathcal{U}^i \neq \emptyset$ **then**
            $L^i := L^i + Y$; $\mathcal{U}^i := \mathcal{U}^i \cup Y$; $ready := false$
         else
            $L := L + Y$.
      until $L^0 = \{\}$.
   until $L = \{\}$ **or** $ready$.
   until $L = \{\}$.
3. **for** $k := 1$ **to** i **do**
   (a) $U^k :=$ Intval $(\mathcal{U}^k)$.
   (b) TestUnique $(U^k, \varepsilon, unique)$.
   (c) **if** $unique$ **then**
      Improve (U^k, L^k); $\widehat{L} := \widehat{L} + (U^k, 1)$
   else
      $\widehat{L} := \widehat{L} + (U^k, 0)$.

Mit Algorithmus 2.25 beschreiben wir das Verfahren, das sowohl diese Reduzierung der Quaderanzahl als auch den Eindeutigkeitstest für die verbleibenden Quader durchführt. Da wir die zweite Komponente r der Paare (Y, r) in der Ergebnisliste $\widehat{L}$ nicht mehr benötigen, werden wir dort den Wert 1 abspeichern, wenn die Eindeutigkeit garantiert ist bzw. den Wert 0, wenn keine Aussage gemacht werden kann. In den in Algorithmus 2.25 verwendeten Arbeitslisten L und L^i mit $i = 0, 1, \ldots$ werden keine Paare (Y, r), sondern nur Quader abgespeichert. Die verwendete Operation + hängt ein neues Element stets am Ende der Liste ein. Intval bezeichnet im folgenden die Bildung der Intervallhülle des Arguments.

Die durch Improve angedeutete Verbesserung der Einschließungen U^k kann nach erfolgreichem Eindeutigkeitstest durch Schnittbildung der Teilquader Y^i aus L^k, für die ein separater Eindeutigkeitstest erfolgreich war, erreicht werden. Dazu wird mit $\mathcal{I} = \{i : Y^i \in L^k \wedge d(Y^i) \neq 0 \wedge \mathsf{N}(Y^i) \overset{\circ}{\subset} Y^i\}$

$$U^k := U^k \cap Y^i \quad \text{für alle } i \in \mathcal{I}$$

gebildet. Die Begründung dafür liefert das nachfolgende Lemma, auf eine algorithmische Beschreibung dieser Methode wollen wir jedoch verzichten.

Lemma 2.7.4 *Seien $U^k, Y^1, Y^2 \in I\!R^n$ und $Y^1, Y^2 \in L^k$ mit $d(Y^1) \neq 0$ und $d(Y^2) \neq 0$. Enthalten U^k, Y^1 und Y^2 jeweils eine durch Algorithmus 2.24 nachgewiesene eindeutige Minimalstelle x^U, x^1 und x^2, so gilt $x^U = x^1 = x^2 \in Y^1 \cap Y^2$.*

Beweis: Aufgrund der Konstruktionsvorschrift ist $U^k = \bigcup_{Y_i \in L^k} Y^i$. Die Annahme $x^U \neq x^i$ führt für $i = 1, 2$ wegen der Eindeutigkeit der Minimalstelle in U sofort auf einen Widerspruch, da $x^i \in Y^i \subseteq U$. Daraus folgt sofort $x^U \in Y^1 \cap Y^2$. □

2.8 Der vollständige Algorithmus

Unter Verwendung aller beschriebenen Teilalgorithmen können wir jetzt mit Algorithmus 2.26 den vollständigen Algorithmus für unser globales Optimierungsverfahren angeben. Eingabedaten sind zunächst f, X und die Toleranz ε. Außerdem wird der Steuerparameter *mode* benötigt, der neben den in Abschnitt 2.5.8 beschriebenen Komponenten noch die Komponente $mode_{\text{level}}$ besitzt. Mit ihr wird durch den Wert 1 angezeigt, daß dem Algorithmus

bereits eine Näherung für die Oberschranke $\widetilde{f}$ des globalen Minimums mitgegeben wird (vgl. auch [6]). Für $mode_{\text{level}} = 0$ muß $\widetilde{f}$ durch eine Mittelpunktsauswertung bestimmt werden. Als Operator + für das Einhängen der Paare $(Y, \underline{F_Y})$ bzw. $(V^i, \underline{F_{V^i}})$ in die Liste verwenden wir den in Abschnitt 2.2.5.1 definierten Operator.

Die Einschließungsquader für alle globalen Minimalstellen sind am Ende des Verfahrens in der Liste $\widehat{L}$ abgespeichert, wobei der Wert in der zweiten Komponente eines Listenelements (Y, r) jeweils angibt, ob die Eindeutigkeit innerhalb von Y verifiziert werden konnte ($r = 1$) oder nicht ($r = 0$). Ist die Liste $\widehat{L}$ nach Durchführung des Algorithmus leer, so wurde eine Näherung $\widetilde{f} < f^*$ verwendet.

Algorithmus 2.26:
Vollständiger Algorithmus

1. $Y := X$; $L := \{\}$; $\widehat{L} := \{\}$.
2. **if** $mode_{\text{level}} = 0$ **then** $\widetilde{f} := \sup(F(m(Y)))$.
3. **repeat**
   (a) $k :=$ OptComp (Y).
   (b) Branch (Y, k, U^1, U^2).
   (c) **for** $i := 1$ **to** 2 **do**
      i. MonoTest (X, U^i, del).
      ii. **if** del **or** $\widetilde{f} < \underline{F_{U^i}}$ **then next**$_i$.
      iii. ConcTest $(X, U^i, L, \widetilde{f}, del)$.
      iv. **if** del **or** $\widetilde{f} < \underline{F_{U^i}}$ **then next**$_i$.
      v. NewtonStep $(U^i, V^1, V^2, L, \widetilde{f}, p, mode)$.
      vi. HandleEdges $(X, U^i, V^1, L, \widetilde{f}, p)$.
      vii. **for** $j := 1$ **to** p **do**
         A. MonoTest (X, V^j, del).
         B. **if** del **or** $\widetilde{f} < \underline{F_{V^j}}$ **then next**$_j$.
         C. $L := L + (V^j, \underline{F_{V^j}})$.
   (d) $Y :=$ NextY $(L, \widehat{L}, \widetilde{f}, \varepsilon, newy)$.

   until not $newy$;
4. CompressVerify $(\widehat{L}, \varepsilon)$.

Die Durchführung des Monotonietests vor dem ersten Überprüfen des Funktionsintervalls wie auch der zusätzliche Monotonietest nach dem Intervall-Newton-Schritt sind im Hinblick auf die praktische Durchführung sinnvoll, da die Intervallauswertung des Gradienten auch bei Verwendung von zentrierten Formen (vgl. Abschnitt 1.5) zur Berechnung des Funktionsintervalls verwendet wird. Dazu muß jedoch der Wert des Gradienten auch im Rahmenalgorithmus bekannt sein und nicht nur in MonoTest (vgl. dazu Abschnitt 3.1.5).

2.9 Weitere Modifikationen des Verfahrens

Zum Abschluß dieses Kapitels wollen wir noch einige Ansätze für weitere Modifikationen unseres Verfahrens geben. Wir verzichten jedoch auf eine algorithmische Beschreibung der modifizierten Schritte. Der Vollständigkeit halber wollen wir in diesem Rahmen auch kurz auf einige aus der Literatur bekannte Variationen von artverwandten Optimierungsverfahren eingehen. Für eine eingehende Studie verweisen wir jedoch auf die jeweils zitierte Literatur.

2.9.1 Zur Bisektion

2.9.1.1 Varianten für OptComp

Der in Abschnitt 2.2.5.2 entwickelte Algorithmus OptComp zur Bestimmung der zu halbierenden Komponente des Quaders Y verwendet den Gradienten von F über Y. Eine Variante für OptComp könnte die in [24] im Rahmen von Nullstellenproblemen benutzte Methode unter Verwendung der Jacobimatrix (in unserem Falle der Hessematrix $H = \boldsymbol{\nabla}^2 F(Y)$) einsetzen. Dann ist die optimale Komponente k so zu bestimmen, daß gilt

$$\sigma_k = \max_{1 \le j \le n} \sigma_j \tag{2.29}$$

wobei

$$\sigma_j = d(Y_j) \cdot \max_{1 \le i \le n} \{|\underline{H_{ij}}|, |\overline{H_{ij}}|\}. \tag{2.30}$$

Diese Vorgehensweise erfordert jedoch aufgrund der Berechnung der Hessematrix einen größeren Aufwand als unsere Methode.

Als sinnvoll erwies sich auch im Rahmen unseres Verfahrens die in [24] beschriebene Methode, die Auswahl der optimalen Komponente auch vom Ausgang des letzten Newton-Schrittes für den aktuellen Quader abhängig

zu machen. Dazu wird innerhalb des Gauß-Seidel-Schrittes (IStepGS) jeweils vor der Schnittbildung komponentenweise die Inklusionsbedingung $W_i^1 \subset Y_i$ überprüft. Hat Y bezüglich der i-ten Komponente diesen Test erfüllt, so wird diese Komponente nicht mehr zur Halbierung herangezogen, da in dieser Koordinatenrichtung der Intervall-Newton-Schritt als konvergent angesehen werden kann (vgl. [24, Satz 2.3]).

2.9.1.2 Adaptive Generierung von Teilquadern

Zu Beginn unseres Optimierungsverfahrens, kann es je nach Struktur der Zielfunktion f und aufgrund der noch großen Quaderdurchmesser vorkommen, daß der Intervall-Newton-Schritt keinen nennenswerten Vorteil erbringt, das heißt die Quaderdurchmesser verringern sich eigentlich nur durch die Bisektion. Wenn dieser Fall eintritt, stellt der Intervall-Newton-Schritt eigentlich einen unnötigen Aufwand dar. Aus diesem Grund schlägt Eriksson in [8] eine initialisierende und eine adaptive Generierung von mehrfachen Bisektionen vor. Die dort erläuterte Heuristik schlägt als Initialisierung vor dem Eintritt in die eigentliche Iteration die Generierung von $N_{\text{init}} = m^n$ Teilquadern mit

$$m = \left\{ \begin{array}{ll} \min(k,10) & \text{für } 1 \leq n \leq 3 \\ \min(k,4) & \text{für } 4 \leq n \leq 5 \\ 2 & \text{sonst} \end{array} \right\} \quad \text{und} \quad k = \log_{10}\left(\frac{d_{\text{rel}}(X)}{\varepsilon}\right)$$

vor. Entsprechend wird eine adaptive Bisektion mit der Generierung von $N_{\text{adapt}} = m^n$ Quadern unmittelbar nach einem „nicht erfolgreichen“ Intervall-Newton-Schritt vorgeschlagen. In diesem Fall wird m zu

$$m = \max\left(\min\left(a + \log_{10}\left(\frac{d(\boldsymbol{K}(Y))}{d(Y)}\right), 6\right), 2\right)$$

berechnet, wobei $\boldsymbol{K}(Y)$ den Krawczyk-Operator ohne Schnittbildung bezeichnet. Der Parameter $a \geq 0$ wird, ausgehend vom Wert 0, jeweils um 0.25 erhöht, wenn der Krawczyk-Schritt effizient war, andernfalls um 0.25 erniedrigt.

2.9.2 Zum Mittelpunktstest

Wird Algorithmus 2.26 ohne eine Startnäherung $\widetilde{f}$ für die Oberschranke des globalen Minimums durchgeführt, muß dafür gesorgt werden, daß die sukzessiv berechneten Werte für $\widetilde{f}$ schnell gegen den tatsächlichen Wert

des Minimums streben. Jede Funktionsauswertung innerhalb des Grundalgorithmus und auch innerhalb der Teilalgorithmen sollte deshalb dazu benutzt werden, eventuell den Wert von $\widetilde{f}$ zu verbessern. Hier können zum Beispiel auch klassische lokale Optimierungsmethoden zum Einsatz kommen, die ausgehend von $c = m(Y)$ versuchen, zu einer Stelle mit kleinerem Funktionswert zu gelangen.

In diesem Zusammenhang sei auch das Verfahren von Jansson [19] genannt, das eine spezielle, heuristische Branch-and-Bound-Methode mit einer Abstiegsmethode kombiniert. Dieses Verfahren, das Näherungen für die globalen Minimalstellen sowie Schranken für den Wert des globalen Minimums berechnet, können wir jedoch nicht näher erläutern, da zum Zeitpunkt des Entstehens dieser Arbeit noch keine algorithmische Beschreibung vorlag.

Eine einfache Modifikation zur Bestimmung eines neuen Wertes $\widetilde{f}$ wäre beispielsweise die Verwendung von mehreren Funktionsauswertungen (anstelle von nur einer Mittelpunktsauswertung) für Punkte aus dem aktuellen Quader. Noch besser wäre es, wenn nicht nur für einzelne Punkte, sondern für ganze Bereiche garantierte Angaben über die dort angenommenen Funktionswerte gemacht werden könnten. Genau dies kann mit Hilfe der sogenannten Inneneinschließung erreicht werden, mit der wir uns deshalb kurz beschäftigen wollen.

2.9.2.1 Inneneinschließungen

Eine Inneneinschließung $f_{\mathbb{I}}(Y)$ für den Wertebereich $f(Y)$ der Funktion $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ auf dem Intervallvektor Y ist ein Intervall, für das gilt

$$f_{\mathbb{I}}(Y) \subseteq f(Y).$$

Zu einer Inneneinschließung kommt man über die Restgliedform der Ordnung 2 (vgl. [5]). Hat f die Darstellung

$$f(y) = q(y) + r(y)$$

mit $q, r \in C^1(D)$, so gilt nach [5, Satz 4] mit Hilfe des Wertebereichs von q und der Intervallauswertung R von r die Beziehung

$$f(Y) \subseteq q(Y) + R(Y).$$

Man definiert dann

$$f_{\mathbb{I}}(Y) := \left[\underline{q(Y)} + \overline{R(Y)}, \overline{q(Y)} + \underline{R(Y)}\right]. \tag{2.31}$$

Mit $\underline{f_{\mathrm{II}}(Y)}$ haben wir somit eine neue garantierte Oberschranke für den Wert des Funktionsminimums.

Mit $g(y) = \nabla f(y)$, $m_g = m(g(Y))$ und $c, \xi \in Y$ kommt man konstruktiv zu einer Inneneinschließung über die aus dem Mittelwertsatz folgende Beziehung

$$\begin{aligned} f(y) &= f(c) + g(\xi) \cdot (y - c) \\ &= f(c) + (g(\xi) + m_g - m_g) \cdot (y - c) \\ &= \underbrace{f(c) + m_g \cdot (y - c)}_{=: q(y)} + \underbrace{(g(\xi) - m_g) \cdot (y - c)}_{=: r(y)}. \end{aligned}$$

Damit ergibt sich

$$q(Y) = f(c) + m_g \cdot (Y - c) \quad \text{und} \quad R(Y) = (G(Y) - m_g) \cdot (Y - c).$$

Bei der praktischen Berechnung einer Inneneinschließung unter Verwendung von Maschinenintervallarithmetik ist zu beachten, daß die Addition in (2.31) mit nach innen gerichteter Rundung ausgeführt werden muß. Da $q(Y)$ auf der Maschine nicht exakt berechnet werden kann, muß auch diese Intervallfunktionsauswertung mit Rundung nach innen erfolgen. Zusätzliche Probleme im Hinblick auf eine exakte Berechnung von $q(Y)$ bereitet die darin vorkommende Funktionsauswertung $f(c)$. Abhilfe schafft die Modifikation

$$q(Y) := m_g \cdot (Y - c) \quad \text{und} \quad R(Y) := f(c) + (G(Y) - m_g) \cdot (Y - c).$$

In Kapitel 3 werden wir diese Problematik nochmals aufgreifen.

Aufgrund der Tatsache, daß einerseits zu Beginn unseres Optimierungsverfahrens große Quader Y behandelt werden müssen, andererseits aber Inneneinschließungen nur für kleine Quader effizient sind, wollen wir noch kurz eine Alternative vorstellen. Um Inneneinschließungen doch sinnvoll verwenden zu können, berechnen wir diese nicht für den Quader Y selbst, sondern für einen Quader $V \subseteq Y$, den wir mit $0 \leq \ell \leq 1$ und $c = m(Y)$ aus

$$V := c + [-\ell, \ell] \cdot (Y - c)$$

erhalten. Aufgrund der Testergebnisse sollte ℓ deutlich kleiner als 1 gewählt werden.

2.9.3 Zum Intervall-Newton-Schritt

Wir wollen hier noch kurz auf einige Varianten des Intervall-Newton-Schrittes eingehen, die die im globalen Optimierungsverfahren zum Ein-

satz kommende Palette von Newton-Schritten erweitern können. Alle vorgestellten Varianten kommen in ihrer ursprünglichen Form aus dem Bereich der Lösung nichtlinearer Gleichungssysteme, ihre Teilschritte können jedoch problemlos auf die Optimierungsproblematik angewendet werden. In den folgenden Abschnitten sei wie bisher $g = \nabla f$, $J_g = \nabla^2 f$, $R = (m(J_G(Y)))^{-1}$, $A = R \cdot J_G(Y)$, $c = m(Y)$ und $b = R \cdot J_g(c)$.

2.9.3.1 Die Methode von Hansen und Greenberg

In [15] wird der übliche Gauß-Seidel-Schritt nach Hansen (vgl. Algorithmus 2.13) um zwei zusätzliche Teilschritte erweitert. Dabei handelt es sich um eine lokale Punktiteration und um eine Gauß-Elimination mit LU-Zerlegung. Die lokale Punktiteration wird nach Durchführung des üblichen Gauß-Seidel-Verfahrens (2.14) aus Abschnitt 2.5.1.4 angewendet. Dabei wird ausgehend von $y^0 = m(Y)$ ein Quasi-Newton-Verfahren in der Form

$$y^{k+1} := y^k - R \cdot g(y^k) \quad k = 0, 1, \ldots$$

durchgeführt. Als Abbruchbedingung für die Iteration wird

$$||g(y^{k+1})|| < 10^{-3} \quad \text{oder} \quad ||g(y^{k+1})|| > \frac{1}{2}||g(y^k)||$$

verwendet. Falls ein y^{k+1} außerhalb von Y liegt wird mit

$$y^{k+1} := (y^k + \lambda(y^{k+1} - y^k)) \cap \partial Y$$

für $0 \leq \lambda < 1$ der Schnittpunkt der Verbindungsgeraden von y^{k+1} und y^k mit dem Rand von Y benutzt.

Gilt nach dem Ende der lokalen Iteration $||g(y^{k+1})|| \geq 10^{-3}$, so wird nochmals das Gauß-Seidel-Verfahren in vereinfachter Form, nämlich nur für die Komponenten i, für die $0 \notin A_{ii}$ gilt, durchgeführt. Andernfalls schließt sich die Gauß-Elimination mit LU-Zerlegung an. Das heißt, es werden zunächst Intervallmatrizen $L, U \in I\!R^{n \times n}$ mit $A \subseteq LU$ berechnet, mit deren Hilfe das System

$$L \cdot U \cdot (y - Y) = b$$

mit $y = y^{k+1}$ gelöst wird. Dabei wird keine erweiterte Intervalldivision eingesetzt, und das Verfahren muß demnach abgebrochen werden, wenn eine Division durch ein die Null enthaltendes Intervall auftritt. Sollte dieser Fall auftreten, wird ebenfalls zum vereinfachten Gauß-Seidel-Schritt übergegangen.

2.9.3.2 Der symmetrische Operator

Als Alternative zum üblichen Krawczyk-Operator $\boldsymbol{K}$ bzw. zum entsprechenden Gauß-Seidel-Verfahren oder auch Hansen-Operator $\boldsymbol{H}$ verwendet Mohd [42] den symmetrischen Operator $\boldsymbol{S}$ (vgl. auch [54]) zur Durchführung eines Intervall-Newton-Schrittes. Mit $M = I - A$ sind diese Operatoren definiert durch

$$\begin{aligned}
\boldsymbol{K}(Y) &:= c - b + M \cdot (Y - c), \\
\boldsymbol{H}_i(Y) &:= c_i - b_i + \sum_{j=1}^{i-1} M_{ij}(\boldsymbol{H}'_j(Y) - c_j) + \sum_{j=i}^{n} M_{ij}(Y_j - c_j) \\
\boldsymbol{H}'_i(Y) &:= \boldsymbol{H}_i(Y) \cap Y_i, \quad i = 1, \ldots, n, \\
\boldsymbol{S}_i(Y) &:= c_i - b_i + \sum_{j=1}^{i} M_{ij}(\boldsymbol{H}'_j(Y) - c_j) + \sum_{j=i+1}^{n} M_{ij}(\boldsymbol{S}'_j(Y) - c_j) \\
\boldsymbol{S}'_i(Y) &:= \boldsymbol{S}_i(Y) \cap \boldsymbol{H}'_i(Y), \quad i = n, \ldots, 1.
\end{aligned}$$

Mohd bevorzugt den symmetrischen Operator aufgrund der folgenden Eigenschaften:

1. Es gilt: $\boldsymbol{S}(Y) \subseteq \boldsymbol{H}(Y) \subseteq \boldsymbol{K}(Y)$.
2. Gilt $d(\boldsymbol{S}(Y)_i) < d(\boldsymbol{H}'(Y)_i)$ für $i = 1, \ldots, n$, dann existiert genau ein $y^* \in \boldsymbol{S}(Y)$ mit $g(y^*) = 0$.

Die Beweise finden sich in [54].

2.9.3.3 Gauß-Seidel-Schritt mit Vorsortierung

Die Durchführung des Gauß-Seidel-Schrittes (2.14) in Verbindung mit den D-optimalen C-Präkonditionierern liefert unterschiedliche Ergebnisse, wenn man die Reihenfolge der Komponenten vertauscht. In [17, Satz 4.1.2] wird festgestellt, daß der Durchmesser $d(Y_i)$ mit den Durchmessern $d(Y_j)$ für $j \neq i$ wächst. Aus diesem Grund ist es sinnvoll, die Komponenten von Y absteigend nach ihren Durchmesser geordnet zu behandeln, also stets die Komponente mit dem größten Durchmesser zuerst zu reduzieren.

2.9.3.4 Gauß-Seidel-Schritt mit Trisektion

Im Zusammenhang mit der Behandlung singulärer Nullstellen schlägt Kearfott in [24] die Verwendung einer Trisektion der Box Y innerhalb des

Intervall-Newton-Schrittes vor. Nach Berechnung einer Näherung $\widetilde{y}$ für die singuläre Nullstelle werden Y^1, Y^2 und Y^3 nach folgendem Schema berechnet:

1. Berechne die optimale Komponente k gemäß (2.29) und (2.30).
2. $Y^1 := Y; \quad Y^2 := Y; \quad Y^3 := Y.$
3. $Y_k^1 := \left[\underline{Y_k}, \widetilde{y}_k - \varepsilon_{\max}\underline{Y_k}\right].$
4. $Y_k^2 := \left[\widetilde{y}_k + \varepsilon_{\max}\overline{Y_k}, \overline{Y_k}\right].$
5. $Y_k^3 := \left[\widetilde{y}_k - \varepsilon_{\max}\underline{Y_k}, \widetilde{y}_k + \varepsilon_{\max}\overline{Y_k}\right].$

Dabei ist der Parameter $\varepsilon_{\max}$ so zu wählen, daß

$$\underline{Y_k} \leq \widetilde{y}_k - \varepsilon_{\max}\underline{Y_k} \leq \widetilde{y}_k + \varepsilon_{\max}\overline{Y_k} \leq \overline{Y_k}$$

erfüllt ist.

Kapitel 3

Praktische Durchführung und Implementierung

In diesem Kapitel werden wir uns mit einigen Details der praktischen Durchführung unseres globalen Optimierungsverfahrens und seiner Implementierung auf dem Rechner beschäftigen. Im ersten Abschnitt werden wir auf den Übergang zur Maschinen(intervall)arithmetik, die damit verbundenen Modifikationen und Besonderheiten, den Einsatz hochgenauer Ausdrucksauswertung und die Ablauforganisation im Hinblick auf eine effiziente Ausnutzung von Funktionsauswertungen eingehen. Im zweiten Abschnitt beschreiben wir zunächst die in der Implementierung verwendeten Datenstrukturen und Operationen einschließlich interner Speicherverwaltung und erläutern den interaktiven Gebrauch des Optimierungsprogramms.

3.1 Praktische Durchführung des Optimierungsverfahrens

Beim Übertragen unseres in Kapitel 2 beschriebenen Algorithmus und seiner Teilalgorithmen auf einen Rechner müssen wir der Tatsache, daß nunmehr alle arithmetischen Operationen nicht mehr in den üblichen reellen Räumen bzw. Intervallräumen, sondern in den entsprechenden Maschinenräumen bzw. Maschinenintervallräumen durchgeführt werden, besondere Beachtung schenken. Auch für Eingabedaten muß dieser Übergang beachtet werden. Die dabei auftretenden Rundungsfehler müssen zum einen stets vollständig erfaßt werden, um garantierte Aussagen machen zu können, zum anderen

durch spezielle Techniken teilweise vermieden werden. Außerdem muß bei einer programmtechnischen Realisierung der Algorithmen so verfahren werden, daß der Aufwand für Funktions-, Gradienten- oder Hessematrixauswertungen gering gehalten wird. Das bedeutet, daß durch geeignete Organisation der internen Daten mehrfache Auswertungen vermieden werden müssen.

3.1.1 Erweiterte Intervallarithmetik

Für die Anwendung der erweiterten Intervalldivision

$$Z = (Z^1 \,|\, Z^2) := A/B$$

mit $A, B \in I\!I\!R$, $Z^1, Z^2 \in I\!I\!R_\infty$ und $0 \in B$ im Rahmen des Gauß-Seidel-Schrittes bieten sich bei der praktischen Durchführung grundsätzlich zwei Alternativen an, nämlich

- das Abfangen sämtlicher Nulldivisionen vor der eigentlichen Operation mit anschließender Sonderbehandlung für die Intervallgrenzen und
- die explizite Verwendung von entsprechend definierten Operationen.

Während die erste Möglichkeit vor allem bei Verwendung von Implementierungssprachen ohne vordefinierte Intervallarithmetik zur Anwendung kommen wird, kann die zweite Möglichkeit bei einer Implementierung in wissenschaftlichen Programmiersprachen mit Operatorkonzept ohne großen Aufwand eingebettet werden. Bei beiden Varianten ist auf die korrekte Ausführung der Rundung in den Intervallgrenzen zu achten (vgl. Abschnitt 1.6). Ähnliches gilt auch für die Subtraktion $(W^1 \,|\, W^2) := a - (Z^1 \,|\, Z^2)$ mit $a \in I\!R$ und die Schnittbildung $(W^1 \,|\, W^2) := Y \cap (Z^1 \,|\, Z^2)$ mit $Y \in I\!I\!R$ und $W^1, W^2 \in I\!I\!R_\infty$.

Prinzipiell müßten bei der Realisierung dieser Operationen fünf verschiedene Fälle für $Z = (Z^1 \,|\, Z^2)$ unterscheiden werden. Mit $\lambda, \rho \in I\!R$ sind dies:

Fall	Z^1	Z^2
(1)	$[\lambda, \rho]$	$\emptyset$
(2)	$[\lambda, \infty)$	$\emptyset$
(3)	$(-\infty, \rho]$	$\emptyset$
(4)	$(-\infty, \rho]$	$[\lambda, \infty)$
(5)	$(-\infty, \infty)$	$\emptyset$

Während im Fall (1) stets $\lambda \leq \rho$ gilt, haben wir für Fall (4) $\rho < \lambda$. Zur Vereinfachung bei der praktischen Durchführung können wir aber in Fall

(4) auch $\rho = \lambda = 0$ zulassen, so daß wir den Fall (5) automatisch mit abgedeckt haben. Die Bezeichnungen λ und ρ deuten hier auf die linke und rechte Grenze eines Intervalls hin, und für die Rundung können wir somit feststellen, daß die λ-Werte jeweils mit Rundung nach unten und die ρ-Werte jeweils mit Rundung nach oben berechnet werden müssen.

Bei Verwendung von Darstellung (4) mit $\lambda = \rho = 0$ für den Fall (5) kann die praktische Durchführung des Gauß-Seidel-Schrittes durch unterschiedliche Bearbeitung der Schnittbildung variieren. Wird nämlich dabei die tatsächliche Darstellung (5) zugrundegelegt, entsteht nur ein einzelnes Intervall. Behandelt man hingegen grundsätzlich den Fall (4), so ergibt sich

$$\begin{aligned}(W^1 \,|\, W^2) &= Y \cap (m(Y) - A/B) \\ &= Y \cap (m(Y) - ((-\infty, 0] \mid [0, \infty))) \\ &= Y \cap ((-\infty, m(Y)] \mid [m(Y), \infty)) \\ &= ([\underline{Y}, m(Y)] \mid [m(Y), \overline{Y}]),\end{aligned}$$

also eine Halbierung des Intervalls $Y = W^1 \cup W^2$.

3.1.2 Differentiationsarithmetik

Für die Durchführung unseres Verfahrens benötigen wir Intervallauswertungen der Zielfunktion f, ihres Gradienten $g = \nabla f$ und ihrer Hessematrix $h = \nabla^2 f$. Die entsprechenden Funktionsvorschriften müßten somit prinzipiell als Eingangsdaten für unseren Algorithmus vorhanden sein. Eine explizite Formulierung dieser Vorschriften für den Gradienten und die Hessematrix ist jedoch im allgemeinen sehr fehleranfällig und aufwendig. Verwenden wir hingegen eine Differentiationsarithmetik, so muß lediglich die Funktion f angegeben werden, während die Werte für g und h bei der Berechnung eines Funktionswertes automatisch mitgeliefert werden. Umgekehrt wird bei Berechnung eines Gradienten bzw. einer Hessematriz unter Verwendung von automatischer Differentiation der Funktionswert bzw. der Funktionswert und der Gradient praktisch „umsonst“ mitgeliefert.

In Abschnitt 1.8 haben wir uns bereits mit der automatischen Differentiation (Vorwärtsmethode) beschäftigt, wie sie im aktuellen Stand unserer Implementierung verwendet wird. Im Hinblick auf eine deutliche Beschleunigung der Berechnungen bietet sich die Verwendung der *schnellen automatischen Differentiation* (Rückwärtsmethode) an, die einen geringeren Aufwand erfordert. Beide Formen können die Symmetrieeigenschaft der Hessematrix ausnutzen und bei direkter Berechnung eines Produkts von Gradient oder Hessematrix und einem Vektor den Aufwand noch weiter verringern.

Für die spezielle Anwendung der automatischen Differentiation in unserem Optimierungsverfahren ist es notwendig, daß die Auswertung einer Funktion f nicht grundsätzlich mit automatischer Berechnung des Gradienten und der Hessematrix abläuft, sondern auch die Funktionswertberechnung allein bzw. die Berechnung des Funktionswertes und des Gradienten ohne Hessematrix durchgeführt werden kann. Unter dieser Forderung bietet es sich an, die Funktionsvorschrift nicht in dreifacher Ausfertigung für drei verschiedene Auswertungsarithmetiken, sondern in einer Art symbolischer Form an das Verfahren zu übergeben, so daß verschiedene Auswertungen mit dieser Darstellung möglich werden. Auf eine entsprechende Implementierung werden wir in Abschnitt 3.2.2.4 eingehen.

3.1.3 Behandlung von Eingabedaten und Konstanten

Bei der Eingabe der Funktion f und des Startbereichs X kann je nach verwendeter Arithmetik ein Konvertierungsfehler beim Übergang von reellen Konstanten in das interne Zahlenformat auftreten. Speziell bei der Verwendung einer Binärarithmetik tritt aufgrund der dezimalen Spezifikation des Optimierungsproblems dieser Effekt häufig auf. Verwendet man keine kontrollierte Rundung, so kann es vorkommen, daß ein Problem behandelt wird, das nicht mit dem tatsächlichen Problem übereinstimmt. Ein typischer Vertreter für diesen Fall wäre die Funktion $f(x) = (x - 0.1)^2$ mit $x \in \mathbb{R}$, da der Wert 0.1 im Binärraster nicht exakt darstellbar ist und üblicherweise zur nächstgelegenen Maschinenzahl gerundet wird. Die korrespondierende Intervallfunktion $F(X) = (X - 0.1)^2$ mit $X \in I\mathbb{R}$ darf dann nämlich für $X \in IR$ auf dem Rechner *nicht* als

$$F_\diamond(X) = (X \diamondminus \square(0.1))^2$$

definiert werden. Will man garantierte Aussagen über das globale Minimum und die Minimalstellen machen, so müssen alle auftretenden dezimalen Werte in das kleinste einschließende Maschinenintervall eingeschlossen werden. Für unsere Beispielfunktion bedeutet dies, daß sie als

$$F_\diamond(X) = (X \diamondminus \diamondsuit(0.1))^2$$

definiert werden muß. Aber auch Werte, die intern exakt dargestellt werden können, müssen stets zum (Punkt-) Intervall gewandelt werden, oder es muß anderweitig sichergestellt werden, daß alle mit diesen Werten arbeitenden Operationen intervallmäßig durchgeführt werden. Für die relle

Funktion $f(x) = (x - 1/3)^2$ heißt das beispielsweise, daß stets

$$F_\diamond(X) = (X \diamondminus 1 \diamondslash 3)^2 = (X \diamondminus \diamond(1) \diamondslash \diamond(3))^2$$

als Maschinenintervallfunktion verwendet werden muß. Allgemein muß eine von Konstanten $c_1, \ldots, c_m \in I\!R$ abhängige Funktion $f(X; c_1, \ldots, c_m)$ stets in die Maschninenintervallfunktion $F_\diamond(\diamond(X); \diamond(c_1), \ldots, \diamond(c_m))$ überführt werden. Ähnliches gilt auch für die Eingabe des Startquaders X, die stets mit nach außen gerichteter Rundung zum engsteinschließenden Maschinenintervall erfolgen muß.

3.1.4 Algorithmen in Maschinenintervallarithmetik

Grundsätzlich sind alle Algorithmen in Kapitel 2 direkt auf den Rechner übertragbar, sofern dort inklusionsisotone arithmetische Operatoren $\diamondsuit$ mit

$$a \circ b \in A \circ B \subseteq A \diamondsuit B \quad \forall a \in A, b \in B, \circ \in \{+, -, \cdot, /\}$$

und inklusionsisotone Standardfunktionen $S_\diamond$ mit

$$s(a) \in S(A) \subseteq S_\diamond(A) \quad \forall a \in A, s \in \mathcal{SF}$$

für $a, b \in R$ und $A, B \in IR$ bereitgestellt werden, die dann an die Stelle der in den Algorithmen verwendeten Intervalloperationen $\circ$ und S treten. Im folgenden wollen wir auf die Realisierung einiger dieser Teilalgorithmen unseres Verfahrens gezielt eingehen.

3.1.4.1 Grundlegender Algorithmus, Mittelpunktstest

Innerhalb des grundlegenden Algorithmus verwenden wir zwei Arten von Funktionsauswertungen. Zum einen ist dies die Intervallauswertung über dem jeweils aktuell behandelten Quader zur Bestimmung einer Unterschranke für den Funktionswert, zum anderen ist dies die Funktionsauswertung im Mittelpunkt zur Durchführung des Mittelpunktstests. Bei der praktischen Durchführung wird der Wert $\underline{F_Y}$ für einen Quader $Y \in IR$ stets als

$$\underline{F_Y} = \inf F_\diamond(Y)$$

berechnet. Damit ist sichergestellt, daß alle Quader, die potentielle Kandidaten für eine globale Minimalstelle sind, auch in die Liste aufgenommen werden, da nur die Quader Y *nicht* in die Liste kommen, die

$$\widetilde{f} < \underline{F_Y} = \inf F_\diamond(Y) \leq \inf F(Y)$$

erfüllen.

Da die Berechnung des reellen Wertes $\widetilde{f} = f(m(Y))$ zur Durchführung des Mittelpunktstests im allgemeinen nicht möglich ist, wurde bereits in der theoretischen Beschreibung des grundlegenden Algorithmus $\widetilde{f} = \sup F(m(Y))$ zur Berechnung einer garantierten Oberschranke verwendet. In der praktischen Durchführung wird diese Vorschrift zu

$$\widetilde{f} = \sup F_\diamond(m_\square(Y)).$$

Die Berechnung des Mittelpunkts erfolgt dabei bereits gerundet. Wir verwenden deshalb für die Maschinenauswertung die Darstellung $m_\square(Y)$.

3.1.4.2 Monotonietest

In der praktischen Durchführung des Monotonietests wird der Intervallgradient G durch

$$G = \boldsymbol{\nabla} F_\diamond(Y)$$

berechnet. Ein Quader Y wird somit nur dann gelöscht oder reduziert, wenn für ein $i \in \{1, \ldots, n\}$

$$0 < \underline{G_i} = \inf\left(\frac{\partial F_\diamond}{\partial x_i}(Y)\right) \leq \inf \frac{\partial F}{\partial x_i}(Y)$$

oder

$$0 > \overline{G_i} = \sup\left(\frac{\partial F_\diamond}{\partial x_i}(Y)\right) \geq \sup \frac{\partial F}{\partial x_i}(Y)$$

erfüllt ist. Damit gelten Satz 2.3.1 und Satz 2.3.2 auch für die Anwendung des Tests auf dem Rechner.

3.1.4.3 Konkavitätstest

In der praktischen Durchführung des Konkavitätstests wird die Intervall-Hessematrix H durch

$$H = \boldsymbol{\nabla}^2 F_\diamond(Y)$$

berechnet. Ein Quader Y wird somit nur dann gelöscht oder reduziert, wenn für ein $i \in \{1, \ldots, n\}$

$$0 > \overline{H_{ii}} = \sup\left(\frac{\partial^2 F_\diamond}{\partial x_i^2}(Y)\right) \geq \sup \frac{\partial^2 F}{\partial x_i^2}(Y)$$

erfüllt ist. Damit gilt Satz 2.4.1 auch für die Anwendung des Tests auf dem Rechner.

3.1.4.4 Intervall-Newton-Schritt

Für die praktische Durchführung der drei in Abschnitt 2.5.1 beschriebenen grundlegenden Formen des Intervall-Newton-Schritts sind Operatoren $\boldsymbol{N}_\diamond$, $\boldsymbol{K}_\diamond$ und $\boldsymbol{GS}_\diamond$ zu verwenden, auf deren Definition wir kurz eingehen wollen. Im folgenden sei stets

$$\begin{aligned} &Y \in IR^n, \\ &c = m_\square(Y) \in R^n, \\ &H = \boldsymbol{\nabla}^2 F_\diamond(Y) \in IR^{n\times n}, \\ &G = \boldsymbol{\nabla} F_\diamond(c) \in IR^n. \end{aligned} \tag{3.1}$$

Wir verwenden also sowohl für die Intervall-Hessematrix über Y als auch für die Gradienten in c Maschinenintervallauswertungen, um jeweils alle Rundungsfehler zu erfassen. Unter der Voraussetzung, daß H nur reguläre Matrizen enthält, können wir den direkten Newton-Operator als

$$\boldsymbol{N}_\diamond(Y) = c \mathbin{\diamondminus} \diamondsuit(H^{-1}) \mathbin{\diamondcdot} G \tag{3.2}$$

definieren. Für den Krawczyk-Operator und für den Gauß-Seidel-Operator benötigen wir eine Näherungsinverse $R \approx (m(H))^{-1}$, auf deren Berechnung wir noch weiter unten eingehen wollen. Damit können wir dann $\boldsymbol{K}_\diamond(Y)$ als

$$\boldsymbol{K}_\diamond(Y) := c \mathbin{\diamondminus} R \mathbin{\diamondcdot} G \mathbin{\diamondplus} \diamondsuit(I - R \cdot H) \mathbin{\diamondcdot} (Y \mathbin{\diamondminus} c) \tag{3.3}$$

definieren. Zu beachten ist, daß der Skalarproduktausdruck $I - R \cdot H$ mit nur einer einzigen Rundung pro Komponente berechnet wird. Ähnlich können wir beim Gauß-Seidel-Operator verfahren, den wir unter Verwendung von

$$A = R \mathbin{\diamondcdot} H \tag{3.4}$$

$$b = R \mathbin{\diamondcdot} G \tag{3.5}$$

durch

$$\left.\begin{aligned} Z &:= Y \mathbin{\diamondminus} c \\ W_i &:= c_i \mathbin{\diamondminus} \diamondsuit\left(b_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij} \cdot Z_j \right) \mathbin{\diamondslash} A_{ii} \\ W_i &:= W_i \cap Y_i \\ Z_i &:= W_i \mathbin{\diamondminus} c_i \end{aligned}\right\} i = 1, \ldots, n \tag{3.6}$$

$$\boldsymbol{GS}_\diamond(Y) := W$$

definieren können, wobei entweder W_i als Intervallpaar angesehen werden muß oder bei Auftreten von Nulldivisionen entsprechende Sonderbehandlungen in Verbindung mit erweiterter Intervallarithmetik ergänzt werden müssen. Auch hier wird die komplette Summe

$$b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} \cdot Z_j$$

mit nur einer abschließenden Rundung zum engsteinschließenden Intervall berechnet. Die Verwendung von PASCAL–XSC ([29], [30]) als Implementierungssprache ermöglich eine einfache Überführung eines solchen Ausdrucks in Programmtext (vgl. auch Abschnitt 3.2.3).

Die praktische Durchführung des Gauß-Seidel-Schrittes nach Hansen (Algorithmus 2.13) sowie unseres modifizierten Gauß-Seidel-Schrittes (Algorithmus 2.14) kann also unter Verwendung von (3.1) und (3.6) in IStepGS (Algorithmus 2.12) erfolgen, denn die durch (3.2), (3.3) und (3.6) definierten Newton-Schritt-Operatoren erfüllen die Beziehungen

$$\boldsymbol{N}(Y) \subseteq \boldsymbol{N}_\diamond(Y), \tag{3.7}$$

$$\boldsymbol{K}(Y) \subseteq \boldsymbol{K}_\diamond(Y), \tag{3.8}$$

$$\boldsymbol{GS}(Y) \subseteq \boldsymbol{GS}_\diamond(Y), \tag{3.9}$$

und es gelten somit die nachfolgenden Korollare.

Korollar 3.1.1 *Enthält H nur reguläre Matrizen, dann gilt für $g = \nabla f$ sowie für Y und $\boldsymbol{N}_\diamond$ aus (3.2):*

(a) Ist x^* Nullstelle von g mit $x^* \in Y$, dann gilt $x^* \in \boldsymbol{N}_\diamond(Y)$.

(b) Ist $\boldsymbol{N}_\diamond(Y) \cap Y = \emptyset$, dann enthält Y keine Nullstelle von g.

(c) Gilt $\boldsymbol{N}_\diamond(Y) \overset{\circ}{\subset} Y$, dann besitzt g in Y und damit auch in $\boldsymbol{N}_\diamond(Y)$ eine eindeutige Nullstelle.

Beweis: Die Behauptungen folgen unmittelbar aus Satz 2.5.1 und der Beziehung (3.7). □

Korollar 3.1.2 *Für $g = \nabla f$ sowie Y, H und $\boldsymbol{K}_\diamond$ aus (3.3) gilt:*

(a) Ist x^* Nullstelle von g mit $x^* \in Y$, dann gilt $x^* \in \boldsymbol{K}_\diamond(Y)$.

(b) Ist $\boldsymbol{K}_\diamond(Y) \cap Y = \emptyset$, dann enthält Y keine Nullstelle von g.

(c) Gilt $\boldsymbol{K}_\diamond(Y) \overset{\circ}{\subset} Y$*, dann besitzt* g *in* Y *und damit auch in* $\boldsymbol{K}_\diamond(Y)$ *eine eindeutige Nullstelle und* H *ist nichtsingulär.*

Beweis: Die Behauptungen folgen unmittelbar aus Satz 2.5.2 und der Beziehung (3.8). □

Korollar 3.1.3 *Für* $g = \boldsymbol{\nabla} f$ *und* Y, H *und* $\boldsymbol{GS}_\diamond$ *aus (3.6) gilt:*

(a) Ist x^* *Nullstelle von* g *mit* $x^* \in Y$*, dann gilt* $x^* \in \boldsymbol{GS}_\diamond(Y)$.

(b) Ist $\boldsymbol{GS}_\diamond(Y) \cap Y = \emptyset$*, dann enthält* Y *keine Nullstelle von* g.

(c) Gilt $\boldsymbol{GS}_\diamond(Y) \overset{\circ}{\subset} Y$*, dann besitzt* g *in* Y *und damit auch in* $\boldsymbol{GS}_\diamond(Y)$ *eine eindeutige Nullstelle und* H *ist nichtsingulär.*

Beweis: Die Behauptungen folgen unmittelbar aus Satz 2.5.3 und der Beziehung (3.9). □

Bemerkung: Insbesondere Teil (c) stellt jeweils ein auf dem Rechner nachprüfbares Kriterium für die Existenz und Eindeutigkeit einer Nullstelle von g dar (vgl. auch Abschnitt 3.1.4.5).

Zur Berechnung der Näherungsinversen $R \approx M^{-1}$ mit $M = m_\square(H)$ verwenden wir den Gauß-Jordan-Algorithmus mit Spaltenpivotisierung (vgl. z. B. [56]). Der entsprechende Algorithmus wird üblicherweise abgebrochen (wegen vermuteter Singularität von M), falls nach der Pivotisierung für ein j eine Division

$$\frac{\lambda}{m_{jj}}$$

mit $\lambda \in R$ und $m_{jj} = 0$ auftritt. Darüber hinaus ist es für die praktische Durchführung notwendig, ein eventuelles Erreichen des Überlaufbereichs der Maschinenarithmetik abzufangen. Der Algorithmus muß also auch abgebrochen werden, wenn gilt

$$\frac{\lambda}{|m_{jj}|} > r_{\max},$$

wobei $r_{\max}$ die betragsmäßig größte darstellbare Maschinenzahl bezeichnet. Ein solcher kontrollierter Abbruch unter Vermeidung eines Laufzeitfehlers wegen Überlauf bei der praktischen Durchführung des Optimierungsverfahrens kann entweder durch eine vollständige IEEE-Ausnahmebehandlung (vgl. [46]) oder durch das Überprüfen der Bedingung

$$t := \frac{|m_{jj}|}{\lambda} < \frac{1}{r_{\max}}$$

erfolgen. Letzteres ist möglich, da $\frac{1}{r_{\max}}$ noch nicht im Unterlaufbereich liegt und da für Werte t im Unterlaufbereich stets $\square t = 0$ gilt (vgl. [46]).

Bei der Berechnung eines D-optimalen C-Präkonditionierers R_i nach Algorithmus 2.16 können wir die Vektoren $p, q \in R^n$ durch jeweils eine maximal genaue Matrix-Vektor-Multiplikation berechnen. Dazu definieren wir $D \in R^n$ durch

$$D_k := \begin{cases} d_{\triangle}(Y_k) & \text{für } k \neq i \\ 0 & \text{für } k = i \end{cases},$$

und berechnen p und q durch

$$p := \underline{H} \boxdot D \quad \text{und} \quad q := \overline{H} \boxdot D.$$

Auch bei der Verwendung aller pivotisierenden Präkonditionierer in Algorithmus 2.17 werden wir wo immer möglich Rundungen vermeiden. Bei entsprechender Formulierung des Maschinenalgorithmus ist dies sogar für die spezielle Subtraktion möglich, indem der Wert von $\Lambda \in \mathbb{IR}$ exakt, also ohne Rundung, berechnet und gespeichert wird. Unter Verwendung eines langen Akkumulators (vgl. [32], [33]) ist dies implementierungstechnisch möglich. Wir werden in Abschnitt 3.2.3 auf die Realisierung eingehen.

Für festes m wird in NewtonStepMPivot (Algorithmus 2.17) folgendermaßen verfahren.

$$\begin{array}{rcl}
Z & := & Y \diamondminus c \\
T_j & := & H_{mj} \cdot Z_j, \quad j = 2, \ldots, n \\
\Lambda & := & G_m + \sum_{j=2}^{n} T_j \\
W_i & := & c_i \diamondminus \diamondsuit(\Lambda) \diamondslash H_{mi} \\
W_i & := & W_i \cap Y_i \\
Z_i & := & W_i \diamondminus c_i \\
T_i & := & H_{mi} \cdot Z_i \\
\underline{\Lambda} & := & \underline{\Lambda} + \underline{T_i} - \underline{T_{i+1}} \quad \left.\right\} \text{falls } i < n \\
\overline{\Lambda} & := & \overline{\Lambda} + \overline{T_i} - \overline{T_{i+1}} \\
Y & := & W
\end{array} \quad \left.\right\} i = 1, \ldots, n \qquad (3.10)$$

Auch hier müssen natürlich entsprechende Sonderbehandlungen ergänzt werden, falls die erweiterte Intervallarithmetik zum Einsatz kommen muß.

Die praktische Durchführung des Gauß-Seidel-Schrittes mit spezieller Präkonditionierung (Algorithmus 2.18) kann somit unter Verwendung von

(3.1) und (3.10) erfolgen, wobei jeweils nach Berechnung des Präkonditionierers R_i mit je einer Zeile aus (3.4) und (3.5) die Vorschrift (3.6) auch in ISTEPPGS (Algorithmus 2.15) zur Anwendung kommt.

3.1.4.5 Verifikationsschritt

Im letzten Abschnitt haben wir bereits mit dem Korollar 3.1.3 ein auf dem Rechner nachprüfbares Kriterium für die Existenz und Eindeutigkeit einer Nullstelle von $g = \nabla f$ angegeben. Dies ist auch für den Nachweis der positiven Definitheit aller Matrizen in $H = \nabla^2 F_\diamond(Y)$ durch Lemma 2.7.2 und unter Verwendung des nachfolgenden Korollars möglich.

Korollar 3.1.4 *Sei $S \in I\!R^{n\times n}$ und $Z \in I\!R^n$, dann folgt aus $S \diamondsuit Z \overset{\circ}{\subset} Z$, daß $\rho(B) < 1$ für alle $B \in S$.*

Beweis: Wegen $S \cdot Z \subseteq S \diamondsuit Z \overset{\circ}{\subset} Z$ folgt die Behauptung sofort mit Satz 2.7.3. □

In Algorithmus 2.23 wird die Intervallmatrix S mit Hilfe von $\kappa \in R$ mit $\|H\|_\infty \leq \kappa$ durch

$$\tau := \nabla\left(\frac{1}{\kappa}\right) \quad \text{und} \quad S := \diamondsuit(I - \tau \cdot H)$$

berechnet. Ist die Inklusionsbedingung in Schritt 4(b) von Algorithmus 2.23 nicht unmittelbar erfüllt, so kann der Wert von ε für die Berechnung der Epsilonaufblähung im Rahmen der Iteration auch verändert werden, um dadurch schneller zu einem Einschluß zu kommen (vgl. auch [26]).

3.1.4.6 Inneneinschließungen

Die praktische Berechnung von Inneneinschließungen $f_{\mathbb{I}}(Y)$ muß für $c = m_\square(Y)$, $G = \nabla F_\diamond(Y)$ und $m_g = m_\square(G)$ durch

$$\begin{aligned} Q &:= \diamondsuit_{\mathbb{I}}(m_g \cdot (Y - c)), \\ R &:= (G \diamondslash m_g) \diamondsuit (Y \diamondslash c), \\ \underline{f_{\mathbb{I}}(Y)} &:= \triangle(\underline{Q} + \overline{R}), \\ \overline{f_{\mathbb{I}}(Y)} &:= \nabla(\overline{Q} + \underline{R}), \end{aligned}$$

mit $Q, R \in I\!R$ erfolgen, um auch bei Verwendung von Maschinenarithmetik die Beziehung $f_{\mathbb{I}}(Y) \subseteq f(Y)$ garantieren zu können. Dabei bezeichnet $\diamondsuit_{\mathbb{I}}$

die Intervallrundung nach innen. Außerdem kann die Inneneinschließung nur verwendet werden, wenn die Bedingung

$$\underline{f_{\mathbb{I}}(Y)} \leq \overline{f_{\mathbb{I}}(Y)}$$

erfüllt ist. Für die Verwendung als erweiterter Mittelpunktstest ist jedoch lediglich der Wert $\underline{f_{\mathbb{I}}(Y)}$ als potentieller neuer Wert für $\widetilde{f}$ von Interesse.

3.1.5 Progammtechnische Organisation des vollständigen Algorithmus

In Kapitel 2 haben wir alle Algorithmen im Hinblick auf den Rechenaufwand unabhängig von einer tatsächlichen Realisierung auf dem Rechner formuliert. Aus diesem Grund erfolgen einige Funktions-, Gradienten-, oder Hessematrix-Auswertungen mehrfach, d. h. in mehreren Teilalgorithmen für das gleiche Argument. Die praktische Durchführung des Verfahrens muß jedoch programmtechnisch so organisiert sein, daß der Auswertungsaufwand gering gehalten wird. Dies kann dadurch erreicht werden, daß alle Intervallauswertungen im Grundalgorithmus ausgeführt und die entsprechenden Größen an die einzelnen Teilalgorithmen weitergereicht werden. Die Berechnung des Gradienten, die bei Verwendung von zentrierten Formen notwendig wird, muß beispielsweise nicht auch noch zusätzlich im Monotonietest durchgeführt werden.

Algorithmus 3.1 beschreibt nun unseren Rahmenalgorithmus unter Berücksichtigung dieser Anforderungen. Man beachte, daß aufgrund der notwendig werdenden Übergabe zusätzlicher Argumente an die Teilalgorithmen einige Parameterlisten verändert wurden. Wir wollen noch kurz auf die Modifikationen einiger Schritte eingehen.

Schritt 1.	Die Werte c und F_c, die für die Funktionsauswertung über zentrierte Formen benötigt werden, werden beim Start einmal berechnet. Innerhalb der Iteration in Schritt 3 werden diese jeweils in NextY berechnet.
Schritt 3. (a)	Der in OptComp benötigte und jeweils in NextY neu berechnete Mittelpunktsvektor c wird mit übergeben.
Schritt 3. (c) i.	Es wird eine Maschinenintervallauswertung des Gradienten berechnet, die sowohl für den nachfolgenden Monotonietest als auch für die eventuelle Berechnung der zentrierten Form (Schritt 3. (c) iii.) benötigt wird.

Algorithmus 3.1:
Praktische Organisation des vollständigen Algorithmus

1. $Y := X$; $c := m_\square(Y)$; $F_c := F_\diamond(c)$; $L := \{\}$; $\widehat{L} := \{\}$.
2. **if** $mode_{\text{level}} = 0$ **then** $\widetilde{f} := \sup(F_c)$.
3. **repeat**
   (a) $k :=$ OptComp (Y, c).
   (b) Branch (Y, k, U^1, U^2).
   (c) **for** $i := 1$ **to** 2 **do**
      i. $G_U := \boldsymbol{\nabla} F_\diamond(U^i)$.
      ii. MonoTest (X, U^i, G_U, del, red).
      iii. **if** del **then next**$_i$
         else if red **then** $F_{U^i} := F_\diamond(U^i)$
         else $F_{U^i} := (F_c \oplus\!\!\!\!\diamond\; (U^i \diamond\!\!\!\!\ominus\; c) \diamond\!\!\!\!\cdot\; G_U) \cap F_\diamond(U^i)$.
      iv. **if** $\widetilde{f} < \underline{F_{U^i}}$ **then next**$_i$
         else $H := \boldsymbol{\nabla}^2 F_\diamond(U^i)$.
      v. ConcTest $(X, U^i, L, \widetilde{f}, H, del)$.
      vi. **if** del **then next**$_i$.
      vii. NewtonStep $(U^i, V^1, V^2, L, \widetilde{f}, p, H, mode)$.
      viii. HandleEdges $(X, U^i, V^1, L, \widetilde{f}, p)$.
      ix. **for** $j := 1$ **to** p **do**
         A. $G_V := \boldsymbol{\nabla} F_\diamond(V^j)$.
         B. MonoTest (X, V^j, G_V, del, red).
         C. **if** del **then next**$_j$
            else if red **then** $F_{V^j} := F_\diamond(V^j)$
            else $c_V := m_\square(V^j)$;
               $F_{V^j} := (F_\diamond(c_V) \oplus\!\!\!\!\diamond\; (V^j \diamond\!\!\!\!\ominus\; c_V) \diamond\!\!\!\!\cdot\; G_V) \cap F_\diamond(V^j)$.
         D. **if** $\widetilde{f} < \underline{F_{V^j}}$ **then next**$_j$.
         E. $L := L + (V^j, \underline{F_{V^j}})$.
   (d) $Y :=$ NextY $(L, \widehat{L}, \widetilde{f}, \varepsilon, c, F_c, newy)$.

   until not $newy$;
4. CompressVerify $(\widehat{L}, \varepsilon)$.

Schritt 3. (c) ii. In MonoTest nimmt G_U den Platz von G ein (vgl. Algorithmus 2.8). Wird der Quader U^i reduziert, so wird dies durch $red = true$ angezeigt.

Schritt 3. (c) iii. Wenn U^i gelöscht werden kann, kann der nächste i-Wert behandelt werden. Wenn in MonoTest eine Reduktion stattgefunden hat, wird das Funktionsintervall ohne Verwendung einer zentrierten Form berechnet, da sonst eine erneute Gradientenauswertung notwendig wäre. In allen anderen Fällen wird F_{U^i} unter Verwendung der zentrierten Form berechnet. Dabei kann sowohl c als auch F_c verwendet werden, da für c auch nach der Halbierung von Y gilt $c \in U^i$ für $i = 1, 2$.

Schritt 3. (c) iv. Liegt das Funktionsintervall F_{U^i} bereits oberhalb von $\widetilde{f}$, so kann zum nächsten i-Wert übergegangen werden. Andernfalls wird eine Maschinenintervallauswertung der Hessematrix berechnet, die sowohl für den Konkavitätstest als auch für den Intervall-Newton-Schritt verwendet wird.

Schritt 3. (c) v. In ConcTest nimmt H den Platz des eigentlich lokal zu berechnenden H ein (vgl. Algorithmus 2.10).

Schritt 3. (c) vii. Die Hessematrix H wird auch an NewtonStep und die darin verwendeten Teilalgorithmen weitergegeben (vgl. Algorithmus 2.20).

Schritt 3. (c) ix. A. Es wird eine Maschinenintervallauswertung des Gradienten berechnet, die sowohl für den nachfolgenden Monotonietest, als auch für die eventuelle Berechnung der zentrierten Form (Schritt 3. (c) ix. C.) benötigt wird.

Schritt 3. (c) ix. B. In MonoTest nimmt G_V den Platz von G ein (vgl. Algorithmus 2.8). Wird der Quader V^j reduziert, so wird dies durch $red = true$ angezeigt.

Schritt 3. (c) ix. C. Wenn V^j gelöscht werden kann, kann der nächste j-Wert behandelt werden. Wenn in MonoTest eine Reduktion stattgefunden hat, wird das Funktionsintervall ohne Verwendung einer zentrierten Form berechnet, da sonst eine erneute Gradientenauswertung notwendig wäre. In allen anderen Fällen wird F_{V^j} unter Verwendung der zentrierten Form berechnet. Dazu muß der Mittelpunkt c_V und

das Funktionsintervall $F_\diamond(c_V)$ berechnet werden, da im allgemeinen nicht mehr $c \in V^j$ gilt.

Schritt 3. (d) Die in NextY berechneten Werte c und F_c werden auch an den Rahmenalgorithmus übergeben, wo sie für die Berechnung der zentrierten Formen verwendet werden.

Bemerkung: Die Schnittbildung der zentrierten Form mit einer herkömmlichen Funktionsauswertung, wie sie in Schritt 3. (c) iii. und 3. (c) ix. C. verwendet wird, bietet sich gerade in Verbindung mit automatischer Differentiation an, da bei der Berechnung des Gradienten der Funktionswert ebenfalls mitgeliefert wird.

3.2 Implementierung in PASCAL–XSC

Durch die Implementierung des in Kapitel 2 beschriebenen globalen Optimierungsverfahrens in der Programmiersprache PASCAL–XSC ([29], [30]), ergibt sich mit der Portabilität des entsprechenden Compilers die Möglichkeit, das Verfahren auf einer großen Palette von Rechnern in identischer Form zu verwenden. Damit haben wir die Möglichkeit, unter Verwendung von leistungsfähigen Großrechnern auch „große“ Probleme, wie sie in der Praxis vorkommen, zu behandeln. Wir wollen im folgenden kurz die wichtigsten Merkmale von PASCAL–XSC erläutern und im Anschluß daran näher auf die Implementierung des Verfahrens eingehen. Eine Beschreibung zur interaktiven Benutzung des Programms findet sich im Anhang.

Um den Rahmen dieser Arbeit nicht zu sprengen, verzichten wir auf die vollständige Wiedergabe des Quellcodes, dessen voller Umfang (für Differentiationsarithmetik, erweiterte Intervallarithmetik, Listenverwaltung, Funktionseingabe und -auswertung, Optimierungsalgorithmus und Menüsteuerung) etwa 7000 Programmzeilen beträgt. Vielmehr geben wir nur die wichtigsten Teile oder beispielhalfte Auszüge an, die Einblick in die Implementierung geben sollen. Wie bereits eingangs in Kapitel 2 erwähnt, sind alle dort angegebenen Algorithmen in Verbindung mit den Details zur pratischen Durchführung (Kapitel 3) nahezu direkt auf den Rechner übertragbar.

3.2.1 Der Einsatz von PASCAL–XSC

Die Programmiersprache PASCAL–XSC, eine Erweiterung von PASCAL für wissenschaftliches Rechnen, wurde mit dem Ziel entwickelt, ein mächtiges Werkzeug für die numerische Lösung wissenschaftlicher Probleme zur

Verfügung zu stellen. Die neuen Sprachkonzepte von PASCAL–XSC erlauben einen einfachen Zugriff auf die zugrundegelegte, mathematisch exakt erklärte und implementierte Rechnerarithmetik in den üblichen Räumen des numerischen Rechnens (siehe [31], [33], [35]). Über Standard-PASCAL hinaus umfaßt PASCAL–XSC die folgenden für unsere speziellen Anwendungen wichtigen Konzepte:

- Universelles Operatorkonzept (benutzerdefinierte Operatoren)
- Operatoren und Funktionen mit beliebigem Ergebnistyp
- Überladen von Prozeduren, Funktionen, Operatoren und Zuweisungen
- Modulkonzept
- Dynamische Felder
- Standarddatentypen *interval, ivector, imatrix, rvector* usw.
- Kontrollierte Rundung
- Hochgenaue Arithmetik und Standardfunktionen für alle Standarddatentypen
- Exakte Ausdrucksauswertung (#-Ausdrücke)

Während die reelle Arithmetik im Sprachkern verankert ist, werden Intervallarithmetik, Vektorarithmetiken (reell und intervallmäßig) und Matrixarithmetiken (reell und intervallmäßig) in Form von Modulen bereitgestellt. Sowohl die Maschinenoperationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ als auch die Maschinenintervalloperationen $\oplus$, $\ominus$, $\odot$ und $\oslash$ können sprachlich durch die üblichen Operatoren `+`, `-`, `*` und `/` angesprochen werden. Die Auswahl der Operation (reell oder intervallmäßig) erfolgt abhängig vom Typ der Operanden. Für eine ausführliche Beschreibung der Sprache verweisen wir auf [29] und [30].

Der aktuelle Sprachumfang von PASCAL–XSC bzw. des Compilers erlaubt größtenteils eine komfortable Programmierung der für unser Verfahren benötigten Hilfsmittel (Datenstrukturen, Operationen, Arithmetiken), die über die bereits vorhandenen Konzepte hinausgehen. Leider stellen sich jedoch auch einige implementierungstechnische Probleme aufgrund von syntaktischen bzw. semantischen Einschränkungen der Sprache. Auf einige Details der Implementierung wollen wir im folgenden eingehen.

3.2.2 Datenstrukturen und Operationen

3.2.2.1 Erweiterte Intervallarithmetik

Die erweiterten Intervalle werden durch einen *record*-Typ unter Verwendung einer Fallunterscheidung definiert:

```
type
  Fall       = (endlich, plusue, minusue, doppelt);
  p_interval = record
                 art      : Fall;
                 inf, sup : real;
               end;
```

Ist `Y` eine Variable vom Typ `p_interval`, so gilt

$$
\begin{array}{ll}
\texttt{Y} = [\ \texttt{inf}\ ,\ \texttt{sup}\ ] & \text{für } \texttt{art} = \texttt{endlich}, \\
\texttt{Y} = (\ -\infty\ ,\ \texttt{sup}\ ] & \text{für } \texttt{art} = \texttt{minusue}, \\
\texttt{Y} = [\ \texttt{inf}\ ,\ \infty\ ) & \text{für } \texttt{art} = \texttt{plusue}, \\
\texttt{Y} = (\ -\infty\ ,\ \texttt{sup}\ ] \cup [\ \texttt{inf}\ ,\ \infty\ ) & \text{für } \texttt{art} = \texttt{doppelt},
\end{array}
$$

wobei $\texttt{Y} = (\ -\infty\ ,\ \infty\ )$ durch $\texttt{art} = \texttt{doppelt}$ mit $\texttt{inf} = \texttt{sup} = 0$ realisiert wird. In unserer speziellen Anwendung werden für diesen Datentyp nur die in Abschnitt 1.6 beschriebenen Operationen

$$
\begin{array}{llllll}
\texttt{div} & \text{für} & \texttt{i} & / & \texttt{i} & \to\ \texttt{p} \\
\texttt{-} & \text{für} & \texttt{r} & - & \texttt{p} & \to\ \texttt{p} \\
\texttt{**} & \text{für} & \texttt{p} & \cap & \texttt{i} & \to\ \texttt{i1} \cup \texttt{i2}
\end{array}
$$

mit `i`, `i1` und `i2` vom Typ `interval`, `pi` vom Typ `p_interval` und `r` vom Typ `real` benötigt, deren Definition wir nachfolgend angeben.

```
operator div (a, b: interval) pdiv: p_interval;
  { Division zweier Intervalle (0 in b ist zugelassen) }
  var
    u: interval;
    h: p_interval;
  begin
    if (0 in b) then
      if ((a.inf < 0) and (0 < a.sup)) or (a=0) or (b=0)  then
        begin
          h.art := doppelt;
          h.sup := 0;
          h.inf := 0;
        end
      else if (a.sup <= 0) then
        if (b.sup = 0) then
          begin
            h.art := plusue;
            h.inf := a.sup /< b.inf;
          end
        else if (b.inf < 0) and (b.sup > 0) then
          begin
            h.art := doppelt;
```

```
            h.sup := a.sup /> b.sup;
            h.inf := a.sup /< b.inf;
          end
        else  { b.inf = 0 }
          begin
            h.art := minusue;
            h.sup := a.sup /> b.sup;
          end
      else  { a.inf >= 0 }
        if (b.sup = 0) then
          begin
            h.art := minusue;
            h.sup := a.inf /> b.inf;
          end
        else if (b.inf < 0) and (b.sup > 0) then
          begin
            h.art := doppelt;
            h.sup := a.inf /> b.inf;
            h.inf := a.inf /< b.sup;
          end
        else  { b.inf = 0 }
          begin
            h.art := plusue;
            h.inf := a.inf /< b.sup;
          end
    else  { not (0 in b) }
      begin
        u     := a / b;
        h.art := endlich;
        h.inf := u.inf;
        h.sup := u.sup;
      end;
    pdiv := h;
  end;
```

Da in PASCAL–XSC Operatoren nicht am Ergebnistyp, sondern nur an den Operandentypen unterschieden werden, wurde nicht das Operatorsymbol /, sondern der Operator `div` überladen, der standardmäßig nur für *integer*-Operanden definiert ist.

```
operator - (a: real; b: p_interval) pmin: p_interval;
  { Subtraktion real - p_interval }
  var h: p_interval;
  begin
    case b.art of
      endlich: begin
                 h.art := endlich;
                 h.inf := a -< b.sup;
                 h.sup := a -> b.inf;
```

```
            end;
      plusue : begin
                 h.art := minusue;
                 h.sup := a -> b.inf;
               end;
      minusue: begin
                 h.art := plusue;
                 h.inf := a -< b.sup;
               end;
      doppelt: begin
                 h.art := doppelt;
                 h.inf := a -< b.sup;
                 h.sup := a -> b.inf;
                 if (h.inf < h.sup) then
                   h.inf := h.sup;
               end;
    end;
    pmin := h;
  end;
```

Zur Darstellung eines „leeren“ Intervalls als Ergebnis des nachfolgenden Schnitt-Operators verwenden wir eine globale Variable `leer` vom Typ `interval`, deren Grenzen beispielsweise durch die konstanten Werte

$$\texttt{leer.inf} := 1 \quad \text{und} \quad \texttt{leer.sup} := -1$$

belegt werden.

```
operator ** (Y: p_interval; var X: interval) Schnitt: interval;
  { Schnittbildung für p_interval und interval }
  var Schnitt2: interval;
  begin
    Schnitt  := leer;
    Schnitt2 := leer;
    case Y.art of
      endlich: if not (X >< intval(Y.inf, Y.sup)) then
                 Schnitt := X ** intval(Y.inf, Y.sup);
      plusue : if (X.sup >= Y.inf) then
               begin
                 Schnitt := intval(max(X.inf, Y.inf), X.sup);
               end;
      minusue: if (Y.sup >= X.inf) then
               begin
                 Schnitt := intval(X.inf, min(X.sup, Y.sup));
               end;
      doppelt: if ((X.inf <= Y.sup) and (Y.inf <= X.sup)) then
                 begin
                   Schnitt  := intval(X.inf, Y.sup);
                   Schnitt2 := intval(Y.inf, X.sup);
```

```
            end
          else if (Y.inf <= X.sup) then
            Schnitt := intval(max(X.inf, Y.inf), X.sup)
          else if (X.inf <= Y.sup) then
            Schnitt :=  intval(X.inf, min(X.sup, Y.sup));
    end;
    X := Schnitt2;  { alter wert von X wird ueberschrieben }
  end;
```

Zu Beachten ist, daß der so definierte Operator ein Teilintervall als Operatorergebnis und ein Teilintervall über die Variable `X` zurückliefert, deren alter Wert somit verloren geht.

3.2.2.2 Differentiationsarithmetik

Die hier vorgestellte Realisierung der Operationen mit automatischer Differentiation arbeitet im Vorwärtsmodus. Im Hinblick auf Funktionsauswertungen mit gleichzeitiger Berechnung des Gradienten oder mit gleichzeitiger Berechnung des Gradienten *und* der Hessematrix bieten sich zwei Möglichkeiten an. Entweder man realisiert je eine Differentiationsarithmetik für die beiden Auswertungsvarianten, also eine Gradienten- und eine Hessematrixarithmetik, oder man realisiert nur eine Arithmetik, die die Operationen zur Berechnung der Hessematrix nur dann durchführt, wenn ein bestimmtes Flag gesetzt ist. Der Nachteil der zweiten Variante besteht darin, daß auch wenn nur Gradienten berechnet werden sollen stets der volle Speicherbereich für die entsprechende Datenstruktur einschließlich Hessematrix benötigt wird.

Wir wollen nun kurz auf die Datenstruktur (für beide Varianten) eingehen und anschließend beispielhaft die Implementierung von Operatoren und Standardfunktionen für die zweite Variante demonstrieren. Für das Optimierungsprogramm wurden unter Verwendung des Basistyps *interval* jeweils die Operatoren `+`, `-`, `*`, `/` und `power` (mit `power (x,y)` $= x^y$) sowie die Standardfunktionen `sqr`, `sqrt`, `exp`, `ln`, `sin`, `cos`, `tan`, `cot`, `arcsin`, `arccos`, `arctan`, `arccot`, `sinh`, `cosh`, `tanh`, `coth`, `arsinh`, `arcosh`, `artanh` und `arcoth` implementiert. Als Datenstruktur in PASCAL würde sich in beiden Fällen eine *record*-Struktur mit einer Funktions-, Gradienten- und gegebenenfalls Hessematrix-Komponente anbieten. Aus Effizienzgründen sollten außerdem für den Gradienten und die Hessematrix dynamische Felder eingesetzt werden, die es erlauben, die Dimension der verwendeten Felder zur Laufzeit des Programms zu ändern. Leider hat das dynamische Konzept von PASCAL–XSC (zur Zeit noch) den Nachteil, daß dynamische Felder nicht als Komponententyp eines *record*-Typs erlaubt sind. Aus diesem Grund

müssen die Datentypen jeweils aus einem einzigen dynamischen Feldtyp bestehen.

Die Typdefinition für eine Gradientenarithmetik erfolgt beispielsweise durch

```
type  fg_type = dynamic array [*] of interval;
```

Behandelt man Funktionen $f : I\!R^n \to I\!R$, so wird ein Funktionswert (inklusive Gradient) durch eine Variable `f` vom Typ `fg_type` mit

```
var  f : fg_type[0..n];
```

dargestellt. Die Komponente `f[0]` enthält den Funktionswert, die Komponenten `f[1]` bis `f[n]` den Gradienten, also `f[j]` enthält den Wert von $\frac{\partial f}{\partial x_j}(x)$ für festes $x \in I\!R^n$. Eine konstante Funktion $f(x) = c$ wird durch

$$\texttt{f[0]} = c \quad \text{und} \quad \texttt{f[j]} = 0 \quad \text{für } \texttt{j} = \texttt{1}, \ldots, \texttt{n},$$

die Funktion $f(x) = x_k$ durch

$$\texttt{f[0]} = x_k, \quad \texttt{f[k]} = 1 \quad \text{und} \quad \texttt{f[j]} = 0 \quad \text{für } \texttt{j} \neq \texttt{k}$$

dargestellt.

Ganz entsprechend erfolgt die Typdefinition für eine Hessematrixarithmetik durch

```
type  fgh_type = dynamic array [*,*] of interval;
```

Für Funktionen $f : I\!R^n \to I\!R$ wird ein Funktionswert (inklusive Gradient und Hessematrix) durch eine Variable `f` vom Typ `fgh_type` mit

```
var  f : fgh_type[0..n,0..n];
```

dargestellt. Die Komponente `f[0,0]` enthält den Funktionswert, die Komponenten `f[0,1]` bis `f[0,n]` den Gradienten und die Komponenten `f[1,1]`, `f[1,2]` bis `f[n,n]` die Hessematrix, während die Komponenten `f[i,0]` für $\texttt{i} \geq 1$ nicht verwendet werden. Somit enthält `f[0,i]` den Wert von $\frac{\partial f}{\partial x_i}(x)$ und `f[i,j]` enthält den Wert von $\frac{\partial^2 f}{x_i x_j}(x)$ für festes $x \in I\!R^n$. Eine Konstante Funktion $f(x) = c$ wird durch

$$\texttt{f[0,0]} = c \quad \text{und} \quad \texttt{f[i,j]} = 0 \quad \text{für } (\texttt{i},\texttt{j}) \neq (0,0)$$

und die Funktion $f(x) = x_k$ wird durch

$$\texttt{f[0,0]} = x_k, \quad \texttt{f[0,k]} = 1 \quad \text{und} \quad \texttt{f[i,j]} = 0 \quad \text{für } (\texttt{i},\texttt{j}) \notin \{(0,0),(0,\texttt{k})\}$$

dargestellt. Die Berechnung der Hessematrixelemente wird unter Ausnutzung der Symmetrieeigenschaft nur für die Elemente auf der Diagonalen und im unteren Dreiecksteil durchgeführt. Die nicht belegten Matrixelemente erhalten ihren Wert erst nach Beendigung einer Funktionswertberechnung.

Als Auszug aus der Implementierung der Operatoren und Standardfunktionen mit automatischer Differentiation geben wir im folgenden die Realisierung für * und sqr an. Unter Verwendung von #-Ausdrücken wurden, wo immer möglich, maximal genaue Operatoren implementiert.

```
operator * (var a,b: fgh_type) mul: fgh_type[0..ub(a),0..ub(a)];
  var
    i, j: integer;
  begin
    mul[0,0]:= a[0,0]*b[0,0];
    for i:=1 to ub(a) do
    begin
      mul[0,i]:= ## (b[0,0]*a[0,i] + a[0,0]*b[0,i]);
      if Hesse then
        for j:=1 to i do
          mul[i,j]:= ## (b[0,0]*a[i,j] + a[0,i]*b[0,j] +
                         b[0,i]*a[0,j] + a[0,0]*b[i,j] );
    end;
  end;

function sqr (var a: fgh_type) : fgh_type[0..ub(a),0..ub(a)];
  var
    i, j: integer;
    h1  : interval;
  begin
    sqr[0,0]:= sqr(a[0,0]);
    h1      := 2*a[0,0];
    for i:=1 to ub(a) do
    begin
      sqr[0,i]:= h1*a[0,i];
      if Hesse then
        for j:=1 to i do
          sqr[i,j]:= ## (h1*a[i,j] + a[0,i]*a[0,j]
                                   + a[0,i]*a[0,j]);
    end;
  end;
```

3.2.2.3 Listenverwaltung

Ähnlich wie bei der Realisierung von Differentiationsarithmetiken würde sich bei der Implementierung der Listenverwaltung eine verkettete Liste von *record*-Elementen anbieten, in denen jeweils ein Wert vom Typ *ivector* (der Quader Y) und ein Wert vom Typ *real* (der Wert $\underline{F_Y}$) gespeichert

wird. Allerdings stellt sich auch hier wieder das Problem, daß ein dynamischer Feldtyp nicht als Komponente in einer *record*-Datenstruktur zugelassen ist. Aus diesem Grund wird in den Listenelementen nicht der Quader Y selbst, sondern seine Adresse innerhalb eines für den Algorithmus global zur Verfügung stehenden und jeweils dynamisch allokierten Speicherfeldes `VecReg` vom Typ

```
type
  RegType = dynamic array [*] of ivector; { Speicherfeld }
```

abgespeichert. Als Datenstruktur wird somit

```
type
  Zeiger  = ^Element;
  Element = record
              adr : integer; { Adresse v. Quader Y in VecReg }
              fyi : real;    { inf ( F(Y) )                  }
              next: Zeiger;  { Zeiger auf naechstes Element  }
            end;
```

verwendet. Für die Begrenzung des Speicherbedarfs der verwendeten Listen ist es notwendig, daß bei der Erzeugung neuer Elemente auf eine interne Speicherverwaltung zurückgegriffen wird. Diese ermöglicht es, nicht mehr benötigte, frühere Elemente der Liste wiederzuverwenden. Solche Elemente werden beim Aushängen aus der Liste stets in eine Hilfsliste `FreeList` vom Typ `Zeiger` eingehängt und von dort bei Bedarf wieder angefordert. Die Implementierung weist aus diesem Grunde drei Prozeduren `NewZ`, `Free` und `FreeAll` auf, die die Erzeugung eines neuen Listenelementes, die Freigabe eines solchen und die Freigabe einer kompletten Liste realisieren.

```
var
  FreeZeiger  : Zeiger;     { Freie Speicherstellen in VecReg }
  RegTypeCtr  : integer;    { Adressenzaehler                 }

procedure NewZ (var z: Zeiger);
  { Neues Element generieren oder aus der FreeList aushaengen }
  begin
    if FreeZeiger = nil then
      begin
        new (z);
        RegTypeCntr := succ (RegTypeCntr);
        z^.adr      := RegTypeCntr;
      end
    else
      begin
        z          := FreeZeiger;
        FreeZeiger := FreeZeiger^.next;
```

```
      end;
    z^.next := nil;
  end;

procedure Free (z: Zeiger);
  { z in die FreeList einhaengen }
  begin
    if z <> nil then
    begin
      z^.next    := FreeZeiger;
      FreeZeiger := z;
    end;
  end;

procedure FreeAll (Kette: Zeiger);
  { Komplette Kette in die FreeList einhaengen }
  var
    H: Zeiger;
  begin
    if Kette <> nil then
    begin
      H:= Kette;
      while H^.next <> nil do
         H:= H^.next;
      H^.next    := FreeZeiger;
      FreeZeiger := Kette;
    end;
  end;
```

Die Belegung eines Listenelements mit dem Paar $(Y, \underline{F_Y})$ erfolgt durch die Funktion `Paar`, der Zugriff auf die Komponenten eines Paares bzw. Elementes durch die Funktionen `fyi` und `vec`.

```
function Paar (var y: ivector; fyi: real) : Zeiger;
  var
    i: integer;
    z: Zeiger;
  begin
    NewZ (z);
    z^.fyi         := fyi;
    VecReg[z^.adr] := y;
    Paar           := z;
  end;

function fyi (z: Zeiger) : real;
  begin
    fyi := z^.fyi;
  end;
```

```
function vec (z: Zeiger) : ivector[1..ub(VecReg,2)];
  begin
    vec := VecReg[z^.adr];
  end;
```

Durch den Operator + wird ein Element in eine Liste bezüglich der in Kapitel 2 definierten Listenordnung eingehängt.

```
operator + (Liste, Neues: Zeiger) enter : Zeiger;
  { Einhaengen eines neuen Paares in die Liste }
  var
    H: Zeiger;
    a: integer;
    f: real;
  begin
    if (Liste = nil) then
      enter := Neues
    else
      begin
        H := Liste;
        while (H^.next <> nil) and (H^.fyi < Neues^.fyi) do
          H := H^.next;
        if (H^.fyi < Neues^.fyi) then
          H^.next := Neues                { Anhaengen am Ende }
        else
          begin
            while (H^.next <> nil)
              and (VecReg[H^.adr] <> VecReg[Neues^.adr])
              and (H^.fyi = Neues^.fyi) do
                H := H^.next;
            if (VecReg[H^.adr] <> VecReg[Neues^.adr]) then
              if (H^.fyi = Neues^.fyi) then
                H^.next := Neues          { Anhaengen am Ende }
              else
                begin
                  a       := Neues^.adr;  { Einhaengen vor H    }
                  f       := Neues^.fyi;  { notwendig!          }
                  Neues^  := H^;          { Dies erfolgt durch  }
                  H^.next := Neues;       { Einhaengen hinter H }
                  H^.adr  := a;           { und Vertauschen der }
                  H^.fyi  := f;           { Element-Inhalte     }
                end
            else  { Neues wird nicht eingehaengt }
              Free (Neues);
          end;
        enter := Liste;
      end
  end;
```

3.2.2.4 Funktionseingabe und -auswertung

Wie bereits erwähnt, ist es im Zusammenhang mit der Verwendung von Differentiationsarithmetiken sinnvoll, Funktionen in einer symbolischen Notation einzulesen und speichern zu können, so daß eine Auswertung mit oder ohne Differentiation durchgeführt werden kann. Eine solche Vorgehensweise bringt außerdem den Vorteil mit sich, daß nicht für jedes neue Problem eine Recompilierung des entsprechenden Programms notwendig wird, da die Funktion durch eine interaktive Eingabe spezifiziert werden kann. Als interne Datenstruktur wurde in unserer Implementierung eine Postfixform verwendet, wie sie in ähnlicher Form auch in Formelauswerterpaketen zum Einsatz kommt. Diese wird unter Verwendung eines Aufzählungstyps `Token`, auf den die reservierten Wörter, die Operatoren, die Operanden und die Konstanten abgebildet werden, und eines Aufzählungstyps `SymbolArten`, auf den die verschiedenartigen Symbole abgebildet werden, folgendermaßen definiert:

```
SymbolListe = ^Symbol;
Symbol      = record
                Next   : SymbolListe;
                Name   : string;
                Art    : SymbolArten;
                Nummer,
                Dimens : integer;
              end;

Postfixform = ^Element;
Element     = record
                Next      : Postfixform;
                Operation : Token;
                IdIndex   : integer;
              end;
```

Eine Routine, die eine Funktionseingabe in *string*-Form in diese Postfixform wandelt, ermöglicht nun eine interaktive Eingabe von Funktionen. Es wird dabei stets eine Form generiert, die es ermöglicht, jeweils Intervalleinschließungen zu berechnen. Alle reellen Konstanten werden somit durch ihr engsteinschließendes Intervall dargestellt.

Die Syntax für Funktionsausdrücke bei interaktiver Eingabe entspricht der Syntax von PASCAL-Ausdrücken, wobei nur die in der Differentiationsarithmetik implementierten Operatoren und Funktionen zugelassen sind. Darüber hinaus können auch Summen- und Produktschleifen der Form

```
sum (i, l, u, A(i));   und   prod (i, l, u, A(i));
```

zur Darstellung von

$$\sum_{i=l}^{u} A(i) \quad \text{und} \quad \prod_{i=l}^{u} A(i)$$

verwendet werden. Dabei sind `l` und `u` *integer*-Werte, und `A(i)` ist ein beliebiger, von `i` abhängender Ausdruck. Die vollständige Syntax für die interaktive Eingabe wird im Anhang beschrieben, wo auch der Gebrauch unseres Optimierungsprogramms demonstriert wird.

Die intervallmäßige Auswertung der in Postfixform vorliegenden Funktion für das Argument `X` kann nun durch die drei überladenen Prozeduren

```
procedure IAuswertung (var X : ivector; var FX: interval);

procedure IAuswertung (var Y : ivector; var FX: interval;
                                        var GX: ivector);

procedure IAuswertung (var Y : ivector; var FX: interval;
                       var GX: ivector; var HX: imatrix);
```

erfolgen. In der ersten Form wird lediglich der Funktionswert `FX` berechnet, in der zweiten Form zusätzlich der Gradient `GX`, in der dritten Form darüber hinaus auch die Hessematrix `HX`.

3.2.3 Maximal genaue Auswertung von Ausdrücken

Für die maximal genaue Auswertung von skalarproduktartigen Ausdrücken bietet PASCAL–XSC mit den #-Ausdrücken (vgl. auch [11]) ein mächtiges Konzept, das es ermöglicht, die in Abschnitt 3.1.4 beschriebenen Methoden innerhalb des Intervall-Newton-Schrittes zu implementieren. Im folgenden geben wir die entsprechenden Programmstücke an, die stets die Operationen und Funktionen aus den Standardmodulen `I_ARI`, `MV_ARI` und `MVI_ARI` verwenden (siehe [29] und [30]). Für den Quader `Y` seien `c`, `FY`, `GY`, `H`, `FC`, `G`, `R`, `A`, `b` und die Einheitsmatrix `I` stets mit

```
var
  Y, GY, G, b : ivector[1..n];
  c           : rvector[1..n];
  H, A        : imatrix[1..n,1..n];
  R, I        : rmatrix[1..n,1..n];
  FY, FC      : interval;
```

vereinbart und deren Werte durch die Anweisungen

```
c := mid (Y);
IAuswertung (Y, FY, GY, H);
IAuswertung (intval(c), FC, G);
R := mid (H);
R := minv (R);
I := id (R);
A := R * H;
b := R * G;
```

berechnet, wobei `n` die Dimension des Quaders `Y` bezeichnet und `minv` eine Näherungsinverse berechnet.

Der Krawczyk-Operator läßt sich entsprechend der Definition (3.3) direkt durch die Anweisung

```
K := c - R * G + ## (I - R * H) * (Y - c);
```

formulieren. Der Gauß-Seidel-Operator kann unter Verwendung von

```
var
  i, j, p : integer;
  Z, Y2   : ivector[1..n];
  W       : p_interval;
```

und den Operatoren für die erweiterte Intervallarithmetik aus Abschnitt 3.2.2.1 durch die Sequenz

```
Z  := Y - c;
Y2 := Y;
i  := 0;
p  := 1;
while (i < n) and (p <> 0) do
begin
  i := i + 1;
  W := c[i] - ## ( b[i] + (for j:=1 to i-1 sum
                              ( A[i,j]*Z[j] ))
                        + (for j:=i+1 to n sum
                              ( A[i,j]*Z[j] )) ) div A[i,i];
  Y[i] := W ** Y2[i];
  p := ord (Y[i] <> leer) + ord (Y2[i] <> leer);
  if p <> 0 then
    Z[i] := Y[i] - c[i];
end;
```

formuliert werden, wobei in dieser Form die Fälle $0 \in A_{ii}$ und $0 \notin A_{ii}$ nicht getrennt behandelt werden (vgl. auch Algorithmus 2.13).

Die Implementierung des praktischen Algorithmus für die pivotisierenden Präkonditionierer laut (3.10) mit der speziellen Subtraktion erfordert einen etwas größeren Aufwand, da es in der verwendeten Version 2.01 von

PASCAL–XSC noch keinen *interval-dotprecision* Datentyp gibt (vgl. auch [11]), der den exakten Wert von Λ speichern kann. Ober- und Untergrenze müssen deshalb getrennt behandelt werden. Für festes m erfolgt die Implementierung von (3.10) unter Verwendung von

```
var
  i, j, p      : integer;
  Z, Y2        : ivector[1..n];
  ASHm         : rvector[1..n];
  W            : p_interval;
  L_inf, L_sup,
  T_i, T_ip1   : dotprecision;
  Lambda       : interval;
```

und der Funktion

```
function abssup (A: ivector) : rvector[lb(A)..ub(A)];
  var i : integer;
  begin
    abssup[i] := sup ( abs (A[i]) );
  end;
```

durch die Sequenz

```
Z    := ## (Y - 0.5*inf(Y) - 0.5*sup(Y));
Y2   := Y;
i    := 0;
p    := 1;
ASHm := abssup (H[m]);
T_ip1:= # (ASHm[2] * Z[2].sup);
L_inf:= # (G[m].inf - T_ip1 - (for j=3 to n sum
                                 ( ASHm[j] * Z[j].sup )));
L_sup:= # (G[m].sup + T_ip1 + (for j=3 to n sum
                                 ( ASHm[j] * Z[j].sup )));
while (i < n) and (p <> 0) do
begin
  i     := i + 1;
  Lambda:= intval (#< (L_inf), #> (L_sup));
  W     := c[i] - Lambda div H[m,i];
  Y[i]  := W ** Y2[i];
  p     := ord (Y[i] <> leer) + ord (Y2[i] <> leer);
  if p <> 0 then
  begin
    Z[i] := Y[i] - c[i];
    if i < n then
    begin
      T_i   := # ( ASHm[i] * Z[i].sup );
      L_inf := # ( L_inf - T_i + T_ip1 );
      L_sup := # ( L_sup + T_i - T_ip1 );
```

```
        T_ip1 := T_i;
      end;
    end;
  end;
```

Die Variablen `T_i` und `T_ip1` speichern jeweils die exakte Obergrenze der Intervalle T_i und T_{i+1} aus (3.10), mit deren Hilfe `Lambda` $= \diamondsuit(\Lambda)$ für jedes i mit nur einer einzigen Rundung berechnet werden kann. Wir machen dabei von der Tatsache Gebrauch, daß für $A, B \in I\!I\!R$ und $b \in I\!R$ mit $A = [\underline{A}, \overline{A}]$ und $B = [-b, b]$ gilt

$$A \cdot B = |A| \cdot B = [-|A| \cdot b, |A| \cdot b].$$

Das Ergebnis ist somit wieder ein symmetrisches Intervall. In unserem Fall sind die Z_j und damit die T_j in (3.10) symmetrische Intervalle, zumindest theoretisch. Für die Praxis gilt dies im allgemeinen Fall nicht, da $c = m(Y)$ nicht exakt bestimmt werden kann. Um für die Intervalle Z_j bzw. `Z[j]` auch auf dem Rechner die Symmetrie sicherzustellen, müssen wir bei der Berechnung von `Z` nicht den gerundeten Mittelpunkt `c` sondern den exakten Mittelpunkt verwenden, denn es gilt dann

Lemma 3.2.1 *Ist $X \in IR$ und $Y = \diamondsuit(X - \frac{1}{2}\underline{X} - \frac{1}{2}\overline{X})$, so gilt $\underline{Y} = -\overline{Y}$.*

Beweis: Für $r \in I\!R$ gilt $\triangle(-r) = - \bigtriangledown (r)$, und für Y gilt somit

$$\begin{aligned} Y &= \diamondsuit(X - \frac{1}{2}\underline{X} - \frac{1}{2}\overline{X}) \\ &= [\bigtriangledown(\underline{X} - \frac{1}{2}\underline{X} - \frac{1}{2}\overline{X}), \triangle(\overline{X} - \frac{1}{2}\underline{X} - \frac{1}{2}\overline{X})] \\ &= [\bigtriangledown(\frac{1}{2}\underline{X} - \frac{1}{2}\overline{X}), \triangle(-\frac{1}{2}\underline{X} + \frac{1}{2}\overline{X})] \\ &= [\bigtriangledown(\frac{1}{2}\underline{X} - \frac{1}{2}\overline{X}), - \bigtriangledown (\frac{1}{2}\underline{X} - \frac{1}{2}\overline{X})], \end{aligned}$$

also $\underline{Y} = -\overline{Y}$. □

Durch den Einsatz eines #-Ausdrucks (siehe erste Zeile der angegebenen Progammsequenz) ist die Verwendung des exakten Mittelpunkts auf dem Rechner möglich.

3.2.4 Algorithmus 3.1 in Programmform

In diesem letzten Abschnitt zur Implementierung unseres Verfahrens wollen wir mit der Prozedur `algo` die Programmform von Algorithmus 2.26 bzw. Algorithmus 3.1 angeben. Benötigt werden dabei die Arithmetikmodule `i_ari` und `mvi_ari`, die dynamische Listenverwaltung auf der Basis des Datentyps `Zeiger` und die Routinen `IAuswertung` für die Postfixform-Auswertung mit automatischer Differentiation. Ein Zugriff auf die Postfixform erfolgt dabei nur innerhalb der Prozeduren `IAuswertung`, so daß diese nicht als lokale Variable auftritt. Darüber hinaus werden in `algo` die folgenden Datentypen benötigt:

```
type
  kind       = (level, precon, multigaps, cutbox);
  modevec    = array [kind] of integer;
  BranchType = dynamic array [*] of ivector;
```

Für die verwendeten Teilalgorithmen bzw. Unterprogramme geben wir nachfolgend die Schnittstellen und als Kommentar deren Funktion an.

```
function OptComp (var Y: ivector; var c: rvector) : integer;
  { Bestimmung der optimalen Komponente }
  { (Algorithmus 2.6)                   }

procedure Branch (var Y: ivector; k: integer;
                  var U1, U2 : ivector);
  { Aufteilung von Y entlang k in U1 und U2 }
  { (Algorithmus 2.1)                       }

procedure MonoTest (var X, U, G: ivector;
                    var del, red: boolean);
  { Monotonietest (Algorithmus 2.8) }

procedure ConcTest (var X, U: ivector; var L: Zeiger;
                    var fmax: real;
                    var H: imatrix; var del: boolean);
  { Konkavitaetstest (Algorithmus 2.10) }

procedure NewtonStep (var U, V1, V2: ivector; var L: Zeiger;
                      var fmax: real; var p: integer;
                      var H: imatrix; mode: modevec);
  { Intervall-Newton-Schritt (Algorithmus 2.20) }

procedure HandleEdges (var X, U, V: ivector; var L: Zeiger;
                       var fmax: real; p: integer);
  { Randbehandlung (Algorithmus 2.22) }
```

```
function NextY (var L, L_res: Zeiger; var fmax, eps: real;
                var c: rvector; var Fc: interval;
                var newy: boolean) : ivector[1..ub(c)];
  { Mittelpunktstest und Auswahl des naechsten Quaders }
  { (Algorithmus 2.5)                                  }

procedure CompressVerify (var L_res: Zeiger; eps: real);
  { Verifikation und Reduzierung der Quaderzahl }
  { (Algorithmus 2.25)                          }
```

Im folgenden nun die Prozedur `algo`, die Implementierung des globalen Optimierungsalgorithmus.

```
procedure algo (var X: ivector; fmax, eps: real; mode: modevec;
                var L_res: Zeiger);
  { Globales Optimierungsverfahren (Algorithmus 2.26 bzw. 3.1) }
  { Eingangsdaten:   X       : Startquader                     }
  {                  fmax    : Naeherung fuer Minimum          }
  {                  eps     : relative Toleranzangabe         }
  {                  mode    : Optionen fuer Verfahren         }
  { Ausgangsdaten:   L_res   : Liste der Minimalstellen        }
  var
    L          : Zeiger;                    { Arbeitsliste     }
    Y,                                      { aktueller Quader }
    G          : ivector[1..ub(X)];         { Gradient         }
    H          : imatrix[1..ub(X),1..ub(X)];{ Hessematrix      }
    c, cV      : rvector[1..ub(X)];         { m (Y), m (V[j])  }
    Fc,FcV,
    FU, FV     : interval;                  { Funktionswerte   }
    U, V       : BranchType[1..2,1..ub(X)]; { Teilquader von Y }
    k, i, j, p : integer;
    del, red,
    newy       : boolean;
  begin
    L     := nil;
    L_res := nil;
    Y     := X;
    c     := mid (Y);
    IAuswertung (intval(c), Fc);
    if mode[level] = 0 then
      fmax := sup (Fc);
    repeat
      k := OptComp (Y, c);
      Branch (Y, k, U[1], U[2]);
      for i:=1 to 2 do
      begin
        IAuswertung (U[i], FU, G);
        MonoTest (X, U[i], G, del, red);
        if not del then
        begin
```

```
        if red then
          IAuswertung (U[i], FU)
        else
          FU := (Fc + (U[i] - c) * G) ** FU;
        if fmax >= inf (FU) then
        begin
          IAuswertung (U[i], FU, G, H);
          ConcTest (X, U[i], L, fmax, H, del);
          if not del then
          begin
            NewtonStep (U[i], V[1], V[2], L, fmax, p, H, mode);
            HandleEdges (X, U[i], V[1], L, fmax, p);
            for j:=1 to p do
            begin
              IAuswertung (V[j], FV, G);
              MonoTest (X, V[j], G, del, red);
              if not del then
              begin
                if red then
                  IAuswertung (V[j], FV)
                else
                  begin
                    cV := mid (V[j]);
                    IAuswertung (intval(cV), FcV);
                    FV := (FcV + (V[j] - cV) * G) ** FV;
                    if fmax >= inf (FV) then
                      L := L + Paar (V[j], inf(FV));
                  end;
              end;
            end; { for j ... }
          end; { if not del ... }
        end; { if fmax ... }
      end; { if not del ... }
    end; { for i ... }
    Y := NextY (L, L_res, fmax, eps, c, Fc, newy);
  until not newy;
  CompressVerify (L_res, eps);
end;
```

Kapitel 4

Numerische Beispiele und Tests

Zum Test des in Kapitel 2 beschriebenen Algorithmus wurde eine Vielzahl aus der Literatur bekannte Testfunktionen verwendet, die in vielen Fällen *real-world*-Probleme darstellen (vgl. [57]). Im folgenden werden wir anhand von drei Testgruppen das Verhalten des Algorithmus in den verschiedenen Arbeitsmodi beleuchten. Bei der ersten Gruppe handelt es sich um einen in der Literatur (vgl. z. B. [6], [57]) gebräuchlichen Satz von Standardtestfunktionen, die oft zum Vergleich von verschiedenen Algorithmen herangezogen werden. Die zweite Gruppe besteht aus einer Reihe von Testproblemen, die von Walster, Hansen und Sengupta in [59] verwendet wurden. In der dritten Gruppe finden sich schließlich noch einige Funktionen, die während der Entwicklungsphase unseres Algorithmus verwendet wurden, um dessen Arbeitsweise zu kontrollieren, und einige weitere Testfunktionen aus der Literatur. Für jede Testfunktion geben wir zu Beginn der entsprechenden Abschnitte ihre Definition, den Startbereich und die mit einer relativen Toleranzangabe von $\varepsilon = 10^{-12}$ berechneten verifizierten Einschließungen für die globalen Minimalstellen und für das Minimum an.

Für die Aufwandsvergleiche der Algorithmus-Varianten untereinander und mit entsprechenden Testresultaten aus der Literatur wurde mit einer relativen Toleranzangabe von $\varepsilon = 10^{-2}$ gearbeitet. Alle Testfunktionen wurden mit sämtlichen Algorithmus-Varianten behandelt. Dabei wurden sämtliche Daten für den jeweils notwendigen Aufwand protokolliert.

Die Aufwandsübersichten für die einzelnen Funktionen und Arbeitsmodi werden in einer tabellarischen Form angegeben. Dabei werden die folgenden Abkürzungen für die Bedeutung der einzelnen Tabellenzeilen verwendet:

mode Arbeitsmodus des Algorithmus,

IT Anzahl der Iterationen,

FA Anzahl der Funktionsauswertungen,

GA Anzahl der Gradientenauswertungen,

HA Anzahl der Hessematrixauswertungen,

LE maximale Anzahl von Elementen (Quader) in der Arbeitsliste und

ZE Laufzeit in STU-Zeiteinheiten.

Da der Algorithmus in der Implementierung von der Differentiationsarithmetik Gebrauch macht, wird bei jeder Gradientenberechnung auch ein Funktionswert und bei jeder Hessematrix auch ein Gradient und ein Funktionswert berechnet. Es werden jedoch dabei nicht alle Zähler erhöht, sondern nur der Zähler für die höchste Ableitung. Für eine Gradientenauswertung erhöht sich damit nur GA und nicht auch noch FA. Bei der STU-Zeiteinheit handelt es sich um die sogenannte *Standard Time Unit* (die Berechnungszeit für 1000 reelle Auswertungen der Shekel-5-Funktion (**S5**)), die als eine nahezu von Compiler und Rechner unabhängige Zeitmeßkonstante verwendet wird. Die Kennzeichnung der Arbeitsmodi (*mode*) erfolgt in den Tabellen durch die Angabe von vier Ziffern, die den *mode*-Komponenten

level precon multigaps cutbox

in dieser Reihenfolge entsprechen. Die Ziffernkombination 0210 steht also beispielsweise für die Algorithmusvariante *ohne* Eingabe einer Minimumsnäherung ($mode_{\mathrm{level}} = 0$), *mit* pivotisierender und D-optimaler C-Präkonditionierung ($mode_{\mathrm{precon}} = 2$), *mit* spezieller Aufspaltungstechnik ($mode_{\mathrm{multigaps}} = 1$) und *ohne* Abspaltung von Singularitäten ($mode_{\mathrm{cutbox}} = 0$), entsprechend den Werten in Tabelle 2.1 in Abschnitt 2.5.8. Im Falle von $mode_{\mathrm{level}} = 1$ wurde $\widetilde{f} > f^*$ jeweils so gewählt, daß $d_{\mathrm{rel}}([f^*, \widetilde{f}]) \approx 0.1$ erfüllt war.

Testgruppe 1

Diese Gruppe besteht aus den folgenden Standardtestfunktionen ([6], [57]):

S5, S7, S10: Die Shekel-Funktionen ($x \in I\!R^4$):

$$f_{\mathrm{S}m}(x) = -\sum_{i=1}^{m} \frac{1}{(x - A_i)(x - A_i)^T + c_i},$$

für $m = 5$, $m = 7$ und $m = 10$. Dabei ist

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix} \quad \text{und} \quad c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}.$$

Startbereich: $0 \le x_i \le 10$, $i = 1, \ldots, 4$.

Minimalstelle:
(**S5**)

```
[ 4.000037152818980E+000, 4.000037152821031E+000]
[ 4.000133276591354E+000, 4.000133276591888E+000]
[ 4.000037152819639E+000, 4.000037152819693E+000]
[ 4.000133276591559E+000, 4.000133276591561E+000]
```

Minimum:

```
[-1.015319967905829E+001,-1.015319967905816E+001]
```

Minimalstelle:
(**S7**)

```
[ 4.000572916185218E+000, 4.000572916186515E+000]
[ 4.000689366185151E+000, 4.000689366185395E+000]
[ 3.999489708859148E+000, 3.999489708859153E+000]
[ 3.999606158858631E+000, 3.999606158858633E+000]
```

Minimum:

```
[-1.040294056681887E+001,-1.040294056681845E+001]
```

Minimalstelle:
(**S10**)

```
[ 4.000746531591439E+000, 4.000746531592502E+000]
[ 4.000592934138421E+000, 4.000592934138670E+000]
[ 3.999663398040321E+000, 3.999663398040324E+000]
[ 3.999509800586807E+000, 3.999509800586809E+000]
```

Minimum:

```
[-1.053640981669226E+001,-1.053640981669182E+001]
```

H3: Die Hartman-Funktion der Dimension 3 ($x \in \mathbb{R}^3$):

$$f_{\mathrm{H3}}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right).$$

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \quad \text{und}$$

$$P = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4378 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}.$$

Startbereich: $0 \le x_i \le 1$, $i = 1, \ldots, 3$.

Minimalstelle:
```
[ 1.145248868047914E-001, 1.145248868047942E-001]
[ 5.555230190395223E-001, 5.555230190395230E-001]
[ 8.525997844999939E-001, 8.525997844999945E-001]
```

Minimum:
```
[-3.861305797100195E+000,-3.861305797100181E+000]
```

H6: Die Hartman-Funktion der Dimension 6 ($x \in \mathbb{R}^6$):

$$f_{\mathrm{H6}}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right).$$

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \quad \text{und}$$

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

Startbereich: $0 \le x_i \le 1$, $i = 1, \ldots, 6$.

Minimalstelle:
```
[ 2.016895110066914E-001, 2.016895110067208E-001]
[ 1.500106918234515E-001, 1.500106918234638E-001]
[ 4.768739742218904E-001, 4.768739742219030E-001]
[ 2.753324304940550E-001, 2.753324304940572E-001]
[ 3.116516166001127E-001, 3.116516166001138E-001]
[ 6.573005340656198E-001, 6.573005340656208E-001]
```

Minimum:
```
[-3.322368011415553E+000,-3.322368011415477E+000]
```

BR: Die Funktion von Branin ($x \in I\!R^2$):

$$f_{\mathrm{BR}}(x) = \left(\frac{5}{\pi}x_1 - \frac{5.1}{4\pi^2}x_1^2 + x_2 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10.$$

Startbereich: $-5 \le x_1 \le 10,\ 0 \le x_2 \le 15$.

Minimalstelle(n):
```
1. [ 3.141592653589792E+000, 3.141592653589795E+000]
   [ 2.274999999999996E+000, 2.275000000000005E+000]
2. [ 9.424777960769374E+000, 9.424777960769387E+000]
   [ 2.474999999999978E+000, 2.475000000000021E+000]
3. [-3.141592653589798E+000,-3.141592653589789E+000]
   [ 1.227499999999998E+001, 1.227500000000002E+001]
```

Minimum:
```
[ 3.978873577297381E-001, 3.978873577297435E-001]
```

SHCB: Die *Six-Hump-Camel-Back-Function* ($x \in I\!R^2$):

$$f_{\mathrm{SHCB}}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4.$$

Startbereich: $-2.5 \le x_i \le 2.5$.

Minimalstelle(n):
```
1. [-8.984201310032836E-002,-8.984201310031070E-002]
   [ 7.126564030207390E-001, 7.126564030207405E-001]
2. [ 8.984201310031189E-002, 8.984201310032657E-002]
   [-7.126564030207403E-001,-7.126564030207390E-001]
```

Minimum:
```
[-1.031628453489896E+000,-1.031628453489858E+000]
```

Aufwandsvergleiche

Für die Funktionen der Gruppe 1 wurden alle Varianten unseres Algorithmus getestet, die nachfolgenden Tabellen enthalten jedoch keine Resultate mit $mode_{\mathrm{cutbox}} = 1$, da diese sich für diese Gruppe von Funktionen nicht wesentlich von den aufgelisteten Resultaten unterscheiden.

S5										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	24	15	27	16	18	24	15	27	16	18
FA	25	95	28	99	316	25	65	28	69	226
GA	132	68	145	70	73	108	58	121	60	63
HA	29	19	31	19	19	21	14	23	14	14
LE	18	38	18	38	53	5	4	7	5	15
ZE	1.8	1.1	2.0	1.2	1.4	1.3	0.8	1.7	1.0	1.2

S7										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	60	14	79	15	17	60	14	79	15	17
FA	61	84	80	87	305	61	55	80	58	204
GA	298	68	366	70	72	259	56	327	58	60
HA	60	20	70	20	19	47	14	57	14	13
LE	38	28	44	28	44	18	5	23	6	20
ZE	5.0	1.4	6.4	1.7	2.0	4.2	1.2	5.7	1.3	1.4

S10										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	63	15	83	17	20	63	15	83	17	20
FA	64	99	84	104	338	64	63	84	68	218
GA	346	72	413	76	80	277	58	347	62	66
HA	74	21	83	21	20	51	14	61	14	13
LE	66	31	67	31	42	20	6	26	8	21
ZE	8.2	2.2	10.2	2.4	3.1	6.4	1.7	8.3	1.8	2.3

Die Tests der drei Shekel-Funktionen verdeutlichen, daß die Durchführung des Gauß-Seidel-Schrittes ohne Präkonditionierer sehr wohl seine Berechtigung haben kann, denn in Verbindung mit der in Abschnitt 2.5.3 entwickelten speziellen Aufteilungstechnik, die eine deutliche Effizienzsteigerung erbringt, fordert diese Variante einen noch geringeren Aufwand als die Verwendung von optimalen Präkonditionierern. Letztere sind allerdings der inversen Mittelpunktspräkonditionierung schon deutlich überlegen. Alle Varianten erfahren durch die Hinzunahme einer Anfangsnäherung für die Oberschranke des Minimums eine Effizienzsteigerung.

H3										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	43	23	46	30	51	43	23	46	30	51
FA	47	65	50	87	187	47	62	50	83	180
GA	194	88	200	101	144	191	86	197	98	142
HA	39	22	39	22	26	38	21	38	20	25
LE	19	13	19	17	28	19	10	19	13	28
ZE	4.1	2.3	4.4	2.7	4.3	4.0	2.2	4.3	2.6	4.2

H6										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	4504	321	21218	519	1273	4504	321	21218	519	1273
FA	4607	1491	22003	2229	10229	4607	1491	22003	2229	10229
GA	18682	1087	81264	1613	3184	18640	1080	81148	1601	3160
HA	3269	223	13334	289	379	3255	218	13124	265	368
LE	1652	274	5818	386	662	1543	244	5456	363	602
ZE	772	57.3	3641	86.6	199	771	57.0	3567	85.1	183

Die Tests der Hartman-Funktionen unterstreichen die enorme Effizienzsteigerung unserer speziellen Aufteilungstechnik. Es zeigt sich auch, daß die Verwendung der üblichen Präkonditionierung den Aufwand um eine Vielfaches erhöht, die Verwendung optimaler Präkonditionierer hingegen Verbesserungen einbringt. Die Hinzunahme einer Anfangsnäherung für das Minimum bringt in diesen Fällen kaum Vorteile.

BR										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	18	18	13	14	15	18	18	13	14	15
FA	29	33	20	30	38	29	32	20	29	35
GA	74	74	56	57	64	73	71	53	55	61
HA	19	20	14	16	18	18	18	13	14	16
LE	8	11	8	8	9	7	7	6	7	7
ZE	0.7	0.8	0.6	0.7	0.7	0.7	0.7	0.6	0.7	0.7

Für die Funktion von Branin erweist sich die spezielle Aufteilungstechnik als etwas aufwendiger, denn für diese liefert die Variante mit inverser Mittelpunktspräkonditionierung mit und ohne Minimumsnäherung die besten Resultate.

SHCB										
mode	0000	0010	0100	0110	0210	1000	1010	1100	1110	1210
IT	65	62	81	75	67	65	62	81	75	67
FA	75	166	89	190	173	75	166	89	190	173
GA	340	233	406	273	245	340	233	406	271	245
HA	102	61	120	71	63	100	61	118	70	63
LE	33	37	39	45	36	30	33	39	39	33
ZE	1.2	0.9	1.7	1.2	0.9	1.2	0.9	1.7	1.2	0.9

Für die SHCB-Funktion erweisen sich die Verwendung der speziellen Aufteilungstechnik ohne Präkonditionierung und die Verwendung optimaler

Präkonditionierer etwa als gleich effizient. Die Hinzunahme einer Minimumsnäherung bringt lediglich eine Verminderung der Listenelemente, aber keine Verringerung des Aufwands.

Wir wollen für diese Gruppe von Standardtestfunktionen die durchschnittlichen Laufzeiten der Varianten unseres neuen Verifikationsverfahrens unter Verwendung der modifizierten Aufteilungstechnik mit den Laufzeiten von traditionellen Näherungsverfahren ohne Ergebnisverifikation, wie sie in [57, Seite 190] protokolliert wurden, vergleichen. Die in der nachfolgenden Tabelle zusammengefaßten Zahlen machen deutlich, daß das neue Verfahren in seiner Effizienz durchaus mit den Näherungsverfahren vergleichbar ist und diesen in den meisten Fällen sogar deutlich überlegen ist.

Vergleich der durchschnittlichen Standardzeiteinheiten						
Funktion	**BR**	**S5**	**S7**	**S10**	**H3**	**H6**
Verifikations-Verfahren	0.7	1.2	1.5	2.3	3.1	111
Näherungs-Verfahren	6.2	15.4	19.4	22.2	11.5	29.1

Testgruppe 2

In diese Gruppe wurden die nachfolgenden Testfunktionen aus [59] (mit der dort verwendeten Numerierung) aufgenommen.

W4: $(x \in \mathbb{R}^2)$: $f_{\mathrm{W4}}(x) = \sum_{i=1}^{5} i\cos((i-1)x_1 + i) \sum_{j=1}^{5} j\cos((j+1)x_2 + j)$.

Startbereich: $-10 \le x_i \le 10$, $i = 1, 2$.

Minimalstelle(n):
```
1. [-1.306707703621444E+000,-1.306707703621129E+000]
   [-7.708313735499354E+000,-7.708313735499341E+000]
   ⋮
9. [ 4.976477603558285E+000, 4.976477603558287E+000]
   [ 4.858056878859824E+000, 4.858056878859827E+000]
```

Minimum:
```
[-1.765417931367463E+002,-1.765417931367449E+002]
```

W5: $(x \in \mathbb{R}^2)$: $f_{\mathrm{W5}}(x) = f_{\mathrm{W4}}(x) + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2$.

Startbereich: $-10 \le x_i \le 10$, $i = 1, 2$.

Minimalstelle:
```
[-1.306853009753580E+000,-1.306853009753564E+000]
[-1.424845041560682E+000,-1.424845041560681E+000]
```

Minimum:
```
[-1.761375780016305E+002,-1.761375780016283E+002]
```

W10: $(x \in \mathbb{R}^{10})$: $f_{\mathrm{W}10}(x) = \sum_{i=1}^{9} y_i^2(1+10\sin^2(\pi(1+y_{i+1}))) + \sin^2(\pi(1+y_1)) + y_{10}^2$, mit $y_i = (x_i-1)/4$, $i = 1,\ldots,10$.

Startbereich: $-10 \le x_i \le 10$, $i = 1,\ldots,10$.

Minimalstelle:
```
[ 9.999999999999998E-001, 1.000000000000001E+000]
                  ⋮                       ⋮
[ 9.999999999999998E-001, 1.000000000000001E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 6.566893418535897E-030]
```

W16: $(x \in \mathbb{R}^2)$: $f_{\mathrm{W}16}(x) = \sum_{i=1}^{2}\left(\sum_{j=1}^{i} x_j\right)^2 = x_1^2 + (x_1+x_2)^2$.

Startbereich: $-10000 \le x_i \le 10000$, $i = 1,2$.

Minimalstelle:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

W17: $(x \in \mathbb{R}^2)$: $f_{\mathrm{W}17}(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$.

Startbereich: $0 \le x_i \le 9$, $i = 1,2$.

Minimalstelle:
```
[ 2.999999999999997E+000, 3.000000000000003E+000]
[ 4.999999999999993E-001, 5.000000000000007E-001]
```

Minimum:
```
[ 0.000000000000000E+000, 9.173281362314180E-029]
```

W18: $(x \in \mathbb{R}^3)$: $f_{\mathrm{W}18}(x) = \sum_{i=1}^{3}\left(x_1 - x_i^2\right)^2 + (x_i - 1)^2\big)$.

Startbereich: $-10 \le x_i \le 10$, $i = 1,2,3$.

Minimalstelle:
```
[ 9.999999999999710E-001, 1.000000000000481E+000]
[ 9.999999999999882E-001, 1.000000000000193E+000]
[ 9.999999999999882E-001, 1.000000000000193E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 1.795174634647188E-024]
```

W25: $(x \in \mathbb{R}^5)$: $f_{\mathrm{W}25}(x) = \sum_{i=1}^{5} x_i^{10}$.

Startbereich: $-1 \le x_i \le 2$, $i = 1,\ldots,5$.

Minimalstelle:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
                  ⋮                       ⋮
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

W27: $(x \in \mathbb{R}^{30})$: $f_{\mathrm{W27}}(x) = \sum_{i=1}^{30} x_i^{10}$.

Startbereich: $-1 \le x_i \le 2$, $i = 1, \ldots, 30$.

Minimalstelle:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
                  ⋮                         ⋮
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

W29: Rosenbrock $(x \in \mathbb{R}^2)$: $f_{\mathrm{W29}}(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$.

Startbereich: $-5 \le x_i \le 5$, $i = 1, 2$.

Minimalstelle:
```
[ 9.999999999998603E-001, 1.000000000000133E+000]
[ 9.999999999997205E-001, 1.000000000000266E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 2.969873293021112E-023]
```

Aufwandsvergleiche

Auch für die Funktionen der Gruppe 2 wurden alle Varianten unseres Algorithmus getestet. Die nachfolgenden Tabellen enthalten jedoch meistens nur die entscheidenden Auszüge aus den Gesamtresultaten.

	W4		**W5**					**W10**		
mode	0000	0110	0000	0010	0100	0110	0210	0000	0100	0210
IT	216	223	56	63	58	63	64	29	27	178
FA	225	799	57	224	59	239	383	32	34	3008
GA	1161	1026	311	286	318	300	290	169	169	441
HA	251	290	67	80	68	87	82	40	39	59
LE	67	192	31	74	31	93	78	28	23	545
ZE	100	102	11.8	12.3	12.3	13.3	14.5	20	20	122

Für die Funktionen **W4** und **W5** liefern die Varianten mit Präkonditionierung oder mit spezieller Aufteilungstechnik keine Verbesserungen, sondern eher eine Aufwandserhöhung. Die Funktion **W10** (mit 10^{10} lokalen Minima innerhalb des Startbereichs!) ist ein krasses Beispiel dafür, daß die optimalen Präkonditionierer nicht immer „optimal“ sein müssen. Für sie liefert die Standardpräkonditionierung die besten Ergebnisse.

	W16			**W17**			**W18**		**W25**	**W27**
mode	0000	0100	0210	0000	0110	0210	0000	0111	bel.	bel.
IT	41	3	5	403	113	125	11	9	4	9
FA	89	12	18	1345	320	334	15	33	17	68
GA	162	10	18	2236	515	610	67	40	32	36
HA	40	2	4	793	170	185	19	11	8	9
LE	4	4	4	80	52	45	7	7	2	2
ZE	0.3	0.1	0.2	8.1	2.2	2.2	0.4	0.3	0.1	7.5

Für **W16** und **W17** zeigt sich die Standard-Präkonditionierung der optimalen Präkonditionierung leicht überlegen, beide sind jedoch der nichtpräkonditionierten Form vorzuziehen. Bei **W18** vermindert die Anwendung des Teilalgorithmus 2.19 (CutBox) die Anzahl der Hessematrix- und Gradientenauswertungen drastisch. **W25** und **W27** werden durch alle Varianten gleich behandelt.

	W29							
mode	0000	0001	0100	0101	0110	0210	0211	1211
IT	20	8	17	6	17	14	8	8
FA	24	11	22	9	40	38	27	23
GA	99	37	101	39	84	67	28	27
HA	27	9	33	9	33	21	7	6
LE	7	3	14	3	14	15	5	3
ZE	0.2	0.1	0.3	0.2	0.3	0.2	0.1	0.1

Die Funktion von Rosenbrock (**W29**) verdeutlicht die Effizienzsteigerungen durch den Einsatz der optimalen Präkonditionierer und der Abspaltungstechnik zur Behandlung von Singularitäten in der Hessematrix. Die Anwendung der Abspaltungstechnik bringt auch bereits ohne eine Präkonditionierung eine deutliche Aufwandsverminderung.

Testgruppe 3

In diese Gruppe wurden die nachfolgenden Testfunktionen aufgenommen.

Han: Hansen ($x \in \mathbb{R}^1$): $f_{\mathrm{Han}}(x) = 24x^4 - 142x^3 + 303x^2 - 276x + 93$.

Startbereich: $0 \le x \le 3$.

Minimalstelle: `[ 1.999999999999998E+000, 2.000000000000000E+000]`

Minimum: `[ 9.999999999972715E-001, 1.000000000002615E+000]`

TZ1: f_1 aus [57] ($x \in I\!R^1$): $f_{\mathrm{TZ1}}(x) = \sin x + \sin \frac{10x}{3} + \ln x - 0.84x$.

Startbereich: $2.7 \leq x \leq 7.5$.

Minimalstelle: `[ 5.199778371061005E+000, 5.199778371061008E+000]`

Minimum: `[-4.601307546494399E+000,-4.601307546494392E+000]`

TZ4: f_4 aus [57] ($x \in I\!R^1$): $f_{\mathrm{TZ4}}(x) = (x + \sin x)e^{-x^2}$.

Startbereich: $-10 \leq x \leq 10$.

Minimalstelle: `[-6.795786600198818E-001,-6.795786600198813E-001]`

Minimum: `[-8.242393984760775E-001,-8.242393984760758E-001]`

TZ5: f_5 aus [57] ($x \in I\!R^1$): $f_{\mathrm{TZ5}}(x) = -\sum_{i=1}^{10} \frac{1}{(k_i(x-a_i))^2 + c_i}$.

$$a = \begin{pmatrix} 3.040 \\ 1.098 \\ 0.674 \\ 3.537 \\ 6.173 \\ 8.679 \\ 4.503 \\ 3.328 \\ 6.937 \\ 0.700 \end{pmatrix}, \quad k = \begin{pmatrix} 2.983 \\ 2.378 \\ 2.439 \\ 1.168 \\ 2.406 \\ 1.236 \\ 2.868 \\ 1.378 \\ 2.348 \\ 2.268 \end{pmatrix}, \quad c = \begin{pmatrix} 0.192 \\ 0.140 \\ 0.127 \\ 0.132 \\ 0.125 \\ 0.189 \\ 0.187 \\ 0.171 \\ 0.188 \\ 0.176 \end{pmatrix}.$$

Startbereich: $0 \leq x \leq 10$.

Minimalstelle: `[ 6.858609265769486E-001, 6.858609265769490E-001]`

Minimum: `[-1.459265202569392E+001,-1.459265202569388E+001]`

Shu: Shubert ($x \in I\!R^1$): $f_{\mathrm{Shu}}(x) = -\sum_{k=1}^{5} k \sin((k+1)x + k)$.

Startbereich: $-10 \leq x \leq 10$.

Minimalstelle(n):
1. `[-6.774576143438998E+000,-6.774576143438839E+000]`
2. `[ 5.791794470920271E+000, 5.791794470920273E+000]`
3. `[-4.913908362593147E-001,-4.913908362593144E-001]`

Minimum: `[-1.203124944216715E+001,-1.203124944216713E+001]`

R: Rastrigin ($x \in I\!R^2$): $f_{\mathrm{R}}(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2$

Startbereich: $-1 \leq x \leq 1$, $i = 1, 2$.

Minimalstelle: `[ 0.000000000000000E+000, 0.000000000000000E+000]`
`[ 0.000000000000000E+000, 0.000000000000000E+000]`

Minimum: `[-2.000000000000000E+000,-2.000000000000000E+000]`

TR1: ($x \in I\!R^2$): $f_{\mathrm{TR1}}(x) = (x_1^2 + 2x_2^2)\exp(-x_1^2 - x_2^2)$.

Startbereich: $-2 \le x \le 3$, $i = 1, 2$.

Minimalstelle:
```
[-8.077935669463161E-028, 8.077935669463161E-028]
[-2.019483917365791E-028, 2.019483917365791E-028]
```

Minimum:
```
[ 0.000000000000000E+000, 7.340967526498341E-055]
```

TR2: ($x \in I\!R^2$): $f_{\mathrm{TR2}}(x) = \sin x_1 + \cos x_2$.

Startbereich: $1 \le x_1 \le 2$, $-1 \le x_2 \le 1$.

Minimalstelle(n):
```
1. [ 1.000000000000000E+000, 1.000000000000000E+000]
   [-1.000000000000000E+000,-1.000000000000000E+000]
2. [ 1.000000000000000E+000, 1.000000000000000E+000]
   [ 1.000000000000000E+000, 1.000000000000000E+000]
```

Minimum:
```
[ 1.381773290676036E+000, 1.381773290676037E+000]
```

TR3: ($x \in I\!R^2$): $f_{\mathrm{TR3}}(x) = (\sin x_1)^2 + (\cos x_2)^2$.

Startbereich: $-3.2 \le x_1 \le 3.2$, $-4.8 \le x_2 \le 4.8$.

Minimalstelle(n):
```
 1. [-3.141592653589794E+000,-3.141592653589793E+000]
    [-4.712388980384691E+000,-4.712388980384689E+000]
 ⋮
12. [ 3.141592653589793E+000, 3.141592653589794E+000]
    [ 4.712388980384689E+000, 4.712388980384691E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 5.997363809597702E-031]
```

G5: Griewank ($x \in I\!R^5$): $f_{\mathrm{G5}}(x) = \sum_{i=1}^{5} \frac{x_i^2}{400} - \prod_{i=1}^{5} \cos(\frac{x_i}{\sqrt{i}}) + 1$.

Startbereich: $-500 \le x_i \le 600$, $i = 1, \ldots, 5$.

Minimalstelle:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
                ⋮                         ⋮
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

Minimum:
```
[ 0.000000000000000E+000, 0.000000000000000E+000]
```

GP: Goldstein und Price ($x \in I\!R^2$):

$$f_{\mathrm{GP}}(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)).$$

Startbereich: $-2 \le x_i \le 2$, $i = 1, 2$.

Minimalstelle:
```
[-2.480242682842661E-024, 2.224426648044042E-025]
[-1.000000000000243E+000,-9.999999999999836E-001]
```

Minimum:
```
[ 2.999999999999096E+000, 3.000000000000355E+000]
```

Aufwandsvergleiche

	Han	**TZ1**	**TZ4**	**TZ5**	**Shu**		**R**	
mode	0000	0000	0000	0000	0000	0001	0000	0010
IT	20	6	6	8	27	22	7	6
FA	53	8	7	9	30	26	17	26
GA	109	26	35	45	131	95	39	26
HA	37	6	10	11	27	18	9	7
LE	10	3	4	6	16	16	4	4
ZE	0.2	0.4	0.3	0.2	3.4	4.4	0.3	0.2

Die Daten für die Funktionen **Han**, **TZ1**, **TZ4** und **TZ5** sind stellvertretend für die Resultate aller Varianten angegeben. **Shu** bzw. **R** demonstrieren nochmals die Vorteile von Algorithmus 2.19 bzw. von Algorithmus 2.14.

	TR1					**TR2**	**TR3**		
mode	0000	0010	0100	0101	0210	0000	0000	0010	0100
IT	6	5	6	6	160	1	25	39	25
FA	7	17	7	7	559	9	63	128	63
GA	31	22	31	31	603	2	152	170	152
HA	7	6	7	7	161	2	38	46	38
LE	3	4	3	3	63	3	18	31	18
ZE	0.2	0.2	0.2	0.3	2.4	0.1	2.0	2.5	2.0

Der Aufwand für **TR1** erhöht sich drastisch beim Einsatz optimaler Präkonditionierer. Spezielle Aufteilung und die Standardpräkonditionierung sind zu bevorzugen. **TR2** ist ein Fallbeispiel für Randminima, **TR3** zeigt erstmals eine Aufwandserhöhung durch die spezielle Aufteilungstechnik.

	G5				**GP**		
mode	0000	0010	0100	0110	0010	0011	0210
IT	61	26	61	25	5178	5199	8615
FA	62	247	62	249	9004	9029	17935
GA	404	140	404	136	22002	22106	30376
HA	96	44	96	43	8583	8616	10165
LE	68	87	68	87	1019	1029	1861
ZE	13.5	7.1	13.9	7.0	178	202	341

Für die Griewank-Funktion vermindert sich durch die spezielle Aufteilungstechnik die Zahl der Hessmatrix- und Gradientenauswertungen, und es tritt nur eine geringe Erhöhung der Zahl der Funktionsauswertungen ein. Für die Funktion von Goldstein und Price wirken sich die speziellen Präkonditionierer nachteilig aus.

Kapitel 5

Zusammenfassung und Ausblick

In dieser Arbeit wurden zahlreiche Modifikationen eines Intervallverfahrens zur globalen Optimierung ohne Nebenbedingungen entwickelt, dessen grundlegende Form auf den Hansen-Algorithmus zurückgeht. Dabei lag der Schwerpunkt auf dem in erweiterter Intervallarithmetik auszuführenden Intervall-Gauß-Seidel-Schritt, für den eine spezielle Aufteilungstechnik entwickelt und optimale Präkonditionierer nach Kearfott eingesetzt wurden. Daneben wurden Methoden für die Randbehandlung, für die Behandlung von Singularitäten der Hessematrix und für die Verifikation der Eindeutigkeit einer globalen Minimalstelle innerhalb der berechneten Schranken hergeleitet. Die theoretischen Aussagen auf der Grundlage der Intervallarithmetik für all diese Methoden lassen sich auf eine mathematisch exakt definierte Maschinenintervallarithmetik übertragen. Aus diesem Grund bildete auch die Behandlung der praktischen Durchführung und der Implementierung des Verfahrens einen wesentlichen Aspekt dieser Arbeit, dem durch leicht auf den Rechner übertragbare Formulierungen der Algorithmen und durch die Ausführungen in Kapitel 3 Rechnung getragen wurde.

Durch die Implementierung des Programmpakets GO in der Sprache PASCAL–XSC war es möglich, umfangreiche Tests und Studien auf verschiedenen Rechnertypen durchzuführen. Zahlreiche Tests haben gezeigt, daß das Verfahren sehr effizient arbeitet und stets verifizierte, hochgenaue Einschließungen für *alle* globalen Minimalstellen und für den Wert des globalen Minimums berechnet. Dies gilt sowohl für den Fall einer großen Zahl lokaler Minima als auch für den Fall mehrerer globaler Minimalstellen. Es muß lediglich die Einschränkung gemacht werden, daß sehr dicht beieinanderliegende globale Minimalstellen nur dann in verschiedene Ergebnisqua-

der eingeschlossen werden können, wenn die Toleranzangabe genügend klein und das Gleitkommaformat auf dem Rechner genügend Mantissenstellen zur Verfügung stellt. In den meisten Fällen ist das vorgestellte Verfahren vom Aufwand her vergleichbar mit klassischen Verfahren, in vielen Fällen ist das verifizierende Verfahren sogar weniger aufwendig. Dabei ist nochmals zu betonen, daß Näherungsverfahren keinerlei Fehlerschranken mitliefern oder Garantieaussagen machen können.

Die Testresultate in Kapitel 4 haben gezeigt, daß vor allem die spezielle Aufteilungstechnik in den meisten Fällen eine wesentliche Effizienzsteigerung mit sich bringt. Außerdem wurde deutlich, daß die optimalen Präkonditionierer in einigen Fällen dem herkömmlichen Mittelpunktspräkonditionierer überlegen sind, daß aber auch der Gauß-Seidel-Schritt ohne Präkonditionierung durchaus seine Einsatzberechtigung hat und somit keine Variante stets zu bevorzugen ist. In diesem Zusammenhang gilt es hervorzuheben, daß es für den praktischen Einsatz eines globalen Optimierungsverfahrens mit Ergebnisverifikation kein einzelnes, allumfassendes Verfahren geben kann, das sämtliche Probleme mit zufriedenstellendem Aufwand löst. Vielmehr wird eine geeignete Kombination diverser Verfahren (darunter klassische Verfahren *und* Intervallverfahren) das Ziel weiterer Entwicklungen sein.

Vorteile für die Behandlung von Problemen mit singulären Hessematrizen erbrachte die Verwendung der Abspaltungstechnik (CutBox), die ebenfalls von klassischen Methoden Gebrauch macht. Hier bietet sich, wie auch bei der speziellen Aufteilungstechnik, die Möglichkeit, die heuristisch gewählten Strategien durch adaptive Verfahren der jeweiligen Problemklasse anzupassen. Die Tests in Kapitel 4 haben auch gezeigt, daß in den meisten Fällen das Verfahren durch die Kenntnis einer Näherung für den Wert des globalen Minimums beschleunigt werden kann. Dies weist auch schon auf eine zusätzliche Anwendungsmöglichkeit für das Verfahren hin, nämlich die Verifikation einer durch ein klassisches Näherungsverfahren berechneten globalen Minimalstelle. Eine solche verifizierende Nachbehandlung kann durch den Einsatz eines Abspaltungsverfahrens, ähnlich dem bei der Singularitätenbehandlung angewendeten Verfahren, bereits mit einer engen Einschließung für die Näherung gestartet und damit sehr effizient durchgeführt werden.

Neben der weiteren Effizienzsteigerung unseres Verfahrens, beispielsweise durch den Einsatz der schnellen automatischen Differentiation (vgl. [9]), wird die Ausweitung des Einsatzgebietes auf Optimierungsprobleme mit Nebenbedingungen wesentliches Ziel für zukünftige Entwicklungen sein. Dabei kann ein Weg z. B. über die Lösung der Optimalitätsbedingungen für die Lagrange-Funktion gehen, so daß prinzipiell ähnliche Methoden eingesetzt

werden können wie bei der Optimierung ohne Nebenbedingungen. Auch hier wird es unbedingt notwendig sein, klassische Verfahren mit in eine sinnvolle Entwicklung von Intervallverfahren einzubeziehen (vgl. auch [49]).

Aufgrund der offensichtlichen „Parallelität“ des in dieser Arbeit behandelten Verfahrens, ist sicherlich die Übertragung des Algorithmus auf Parallelrechner ebenfalls ein Ansatzpunkt für weitere Arbeiten. Allein die Verteilung des Optimierungsgebietes auf einzelne Prozessoren, kann bereits eine deutliche Effizienzsteigerung erbringen. Allerdings ist in diesem Zusammenhang zu beachten, daß die Bearbeitungszeiten für die Teilgebiete stark differieren können. Bei der Parallelisierung des Verfahrens muß deswegen vor allem auf eine gleichmäßige Ausnutzung der Einzelprozessoren geachtet werden (vgl. auch [8]).

Anhang A

Das Programmpaket GO

Wir geben hier noch kurz einige Informationen zur interaktiven Verwendung des im Rahmen der Arbeit entstandenen Programmpakets GO. Zur Vereinfachung von Tests wurde mit einer menügesteuerten Benutzeroberfläche und unter Verwendung der in Abschnitt 3.2.2.4 beschriebenen internen Funktionsdarstellung in Verbindung mit einer Differentiationsarithmetik die Möglichkeit geschaffen, den Algorithmus auf verschiedene Funktionen anzuwenden, ohne dazu Teile des Programms neu compilieren zu müssen.

A.1 Benutzeroberfläche

Das Programm GO besitzt eine portable Benutzeroberfläche, die sowohl auf PC's als auch auf Workstations und Großrechnern verwendet werden kann. Das nach dem Start erscheinende Hauptmenü hat den folgenden Aufbau:

```
-------------- Global Optimization - Main Menu ----------------

 Please, select:                                  Work file: ---

        S     Select optimization problem input file

        E     Edit selected file

        W     Work on specified problem

        H     display Help information

        Q     Quit program

 Select  ==> _
```

Mit der Option `S` wird eine Eingabedatei spezifiziert, dabei kann auch der Platzhalter `*` verwendet werden, um eine Liste der existierenden Dateien zu erhalten. Der Name der ausgewählten Arbeitsdatei wird jeweils oben rechts angegeben. Bei Bedarf kann diese unter Verwendung der Option `E` editiert werden. Mit der Option `W` erfolgt das Einlesen der Eingabedatei und der Eintritt in das eigentliche Arbeitsmenü. Informationen zur Bedienung des Programms können mit der Option `H` abgerufen werden. Durch die Option `Q` wird das Programm verlassen.

Das Arbeitsmenü hat den folgenden Aufbau:

```
------------- Global Optimization - Work Menu -----------------

 Please, select:                              Work file: camel-6

        D     Display optimization problem

        C     Compute inclusions for all minimizers

        R     display Results

        S     Save results on file

        N     New initial region

        P     new Parameters

        M     change Mode of optimization algorithm

        Q     Quit work menu

 Select  ==> _
```

Die Option `D` des Arbeitsmenüs erlaubt die Anzeige der aktuellen Problemspezifikation, die zunächst der Spezifikation aus der Eingabedatei entspricht. Die dort festgelegten Werte für den Startbereich und für die Parameter können jedoch auch noch innerhalb des Arbeitsmenüs durch die Optionen `N` und `P` verändert werden, ohne daß die Eingabedatei erneut gelesen werden muß. Das Optimierungsverfahren wird durch die Option `C` gestartet, die Anzeige der berechneten Ergebnisse erfolgt durch die Option `R`. Zur Speicherung der Resultate in einer Datei kann die Option `S` angewählt werden, die auch beim Verlassen des Arbeitsmenüs mit der Option `Q` automatisch als Sicherungsabfrage aktiviert wird. Die Option `M` erlaubt es, den Arbeitsmodus des Optimierungsverfahrens zu verändern.

A.2 Eingabesyntax

Eine Eingabedatei zur Spezifikation eines Optimierungsproblems für das Programm GO hat folgenden syntaktischen Aufbau:

```
problem
   Name ;
parameters
   Definition ; ... ; Definition;
variables
   Definition ; ... ; Definition;
function
   Ausdruck ;
tolerance
   real-Konstante .
```

Dabei können die Zeilen 1 bis 4 entfallen. `Name` steht hier und im folgenden stets für einen üblichen PASCAL-Bezeichner. Für `Definition` ist eine Skalardefinition

```
Name := Ausdruck
```

oder eine Vektordefinition

```
Name := (Ausdruck, ..., Ausdruck)
```

mit implizit, durch die Anzahl der Ausdrücke, festgelegter Dimension, möglich. Die Ausdrücke dienen dabei zur Initialisierung der Variablen und Parameter. Die Initialwerte der Parameter bleiben beim Ablauf des Optimierungsalgorithmus konstant, die Initialwerte der Variablen definieren lediglich den Startbereich. Ein `Ausdruck` hat die syntaktische Form

```
Operand  Operator  Operand
```

wobei auch ein vorgestelltes monadisches `+` oder `-` erlaubt ist. Als `Operator` kann `+`, `-`, `*`, `/` und `^` verwendet werden. Letzterer zur Kennzeichnung der Exponentiation. Als `Operand` ist eine `Konstante`, eine `Variable`, ein `Parameter`, ein `Ausdruck`, ein geklammerter `Ausdruck`, eine `Summen-Schleife` oder eine `Produkt-Schleife` zugelassen. Für eine `Variable` oder einen `Parameter` kann entweder ein `Name` (bei skalaren Größen) oder

```
Name [ integer-Konstante ]
```

(bei Vektor-Größen) verwendet werden. Innerhalb einer `Summen-Schleife`

```
sum (Index-Name, integer-Konstante, integer-Konstante, Ausdruck)
```

oder einer `Produkt-Schleife`

prod `(Index-Name, integer-Konstante, integer-Konstante, Ausdruck)`

kann auch

`Name [ Index-Name ]`

als `Variable` oder `Parameter` verwendet werden. Der `Index-Name` ist innerhalb der Schleifen implizit vereinbart, alle anderen in Ausdrücken auftretenden Variablen oder Parameter müssen bei ihrer Verwendung bereits erklärt und initialisiert sein.

Im folgenden noch ein Beispiel für eine Eingabedatei:

```
--------- Top of File ---------
   problem
       Six_hump_camel_back_function ;

   parameters
       b := 3 ;

   variables
       x := ( [-2.5,2.5],
              [-2.5,2.5] ) ;

   function
          4 * x[1]^2 - 2.1 * x[1]^4 +     x[1]^6 / b  +
       x[1] * x[2]   -   4 * x[2]^2 + 4 * x[2]^4 ;

   tolerance
       1e-9 .
--------- End of File ---------
```

Literaturverzeichnis

[1] Alefeld, G. und Herzberger, J.: *Einführung in die Intervallrechnung*. Bibliographisches Institut, Mannheim, 1974.

[2] Alefeld, G. und Herzberger, J.: *Introduction to Interval Computations*. Academic Press, New York, 1983.

[3] Böhm, H., Rump, S. M. und Schumacher, G.: *E-Methods for Nonlinear Problems*. In [21], 59–80, 1987.

[4] Chatterji, S. D., Fuchssteiner, B., Kulisch, U., Liedl, R. und Purkert, W. (Hrsg.): *Jahrbuch Überblicke Mathematik 1992*. Vieweg, Braunschweig, 1992.

[5] Cornelius, H. und Lohner, R.: *Computing the Range of Values of Real Functions with Accuracy Higher than Second Order*. Computing **33**, 331–347, 1984.

[6] Csendes, T.: *Test Results of Interval Methods for Global Optimization*. In [22], 417–424, 1991.

[7] Dussel, R.: *Einschließung des Minimalpunktes einer streng konvexen Funktion auf einem n-dimensionalen Quader*. Dissertation, Universität Karlsruhe, 1972.

[8] Eriksson, J: *Parallel Global Optimization Using Interval Analysis*. Licentiate Thesis, University of Umeå, Sweden, 1991.

[9] Fischer, H.-C.: *Schnelle automatische Differentiation, Einschließungsmethoden und Anwendungen*. Dissertation, Universität Karlsruhe, 1990.

[10] Frommer, A: *Newton-Verfahren und nicht einfache Nullstellen*. In [4], 27–44, 1992.

[11] Hammer, R.: *Maximal genaue Berechnung von Skalarproduktausdrücken und hochgenaue Auswertung von Programmteilen.* Dissertation, Universität Karlsruhe, 1992.

[12] Hansen, E.: *Global Optimization Using Interval Analysis – The One-Dimensional Case.* Journal of Optimization Theory and Applications **29**, 331–344, 1979.

[13] Hansen, E.: *Global Optimization Using Interval Analysis – The Multi-Dimensional Case.* Numerische Mathematik **34**, 247–270, 1980.

[14] Hansen, E.: *An Overview of Global Optimization Using Interval Analysis.* In [40], 289–307, 1988.

[15] Hansen, E. und Greenberg, R.: *An Interval Newton Method.* Applied Mathematics and Computations **12**, 89–98, 1983.

[16] Hansen, E. und Sengupta, S.: *Bounding Solutions of Systems of Equations using Interval Analysis.* BIT **21**, 203–211, 1981.

[17] Hu, C.: *Optimal Preconditioners for the Interval-Newton-Method.* Ph. D. Dissertation, University of Southwestern Louisiana, 1990.

[18] Ichida, K. und Fujii, Y.: *An Interval Arithmetic Method for Global Optimization.* Computing **23**, 85–97, 1979.

[19] Jansson, C.: *A Global Optimization Method Using Interval Arithmetic.* In: *Computer Arithmetic and Scientific Computation.* Proceedings of the SCAN 91, North-Holland, Elsevier, Amsterdam, erscheint 1992.

[20] Kaucher, E.: *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Räume.* Dissertation, Universität Karlsruhe, 1973.

[21] Kaucher, E., Kulisch, U. und Ullrich, Ch. (Hrsg.): *Computer Arithmetic – Scientific Computation and Programming Languages.* Teubner, Stuttgart, 1987.

[22] Kaucher, E., Markov, S. M. und Mayer, G. (Hrsg.): *Computer Arithmetic, Scientific Computation and Mathematical Modelling.* IMACS Annals on Computing and Applied Mathematics **12**, J.C. Baltzer AG, Basel, 1991.

[23] Kearfott, R. B.: *Preconditioners for the Interval Gauss-Seidel Method.* SIAM Journal of Numerical Analysis **27**, 1990.

[24] Kearfott, R. B.: *Interval Newton / Generalized Bisection When There are Singularities near Roots.* Annals of Operations Research **27**, 181–196, 1990.

[25] Kearfott, R. B., Hu, C. und Novoa, M.: *A Review of Preconditioners for the Interval Gauss-Seidel Method*. Interval Computations **1**, Institute for New Technologies, St. Petersburg, 1991.

[26] König, S.: *On the Inflation Parameter Used in Self-Validating Methods*. In [58], 1990.

[27] Kosmol, P.: *Methoden zur numerischen Behandlung nichtlinearer Gleichungen und Optimierungsaufgaben*. Teubner, Stuttgart, 1989.

[28] Krawczyk, R.: *Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken*. Computing **4**, 187–201, 1969.

[29] Klatte, R., Kulisch, U., Neaga, M., Ratz, D. und Ullrich, Ch.: *PASCAL–XSC – Sprachbeschreibung mit Beispielen*. Springer-Verlag, Heidelberg, 1991.

[30] Klatte, R., Kulisch, U., Neaga, M., Ratz, D. und Ullrich, Ch.: *PASCAL–XSC – Language Reference with Examples*. Springer-Verlag, Heidelberg, 1992.

[31] Kulisch, U.: *Grundlagen des Numerischen Rechnens – Mathematische Begründung der Rechnerarithmetik*. Reihe Informatik, Band 19, Bibliographisches Institut, Mannheim, 1976.

[32] Kulisch, U. (Hrsg.): *Wissenschaftliches Rechnen mit Ergebnisverifikation – Eine Einführung*. Akademie Verlag, Ost-Berlin, Vieweg, Wiesbaden, 1989.

[33] Kulisch, U. und Miranker, W. L.: *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.

[34] Kulisch, U. und Miranker, W. L. (Hrsg.): *A New Approach to Scientific Computation*. Academic Press, New York, 1983.

[35] Kulisch, U. und Miranker, W. L.: *The Arithmetic of the Digital Computer: A New Approach*. SIAM Review **28**, No. 1, 1–140, 1986.

[36] Mayer, G.: *Grundbegriffe der Intervallrechnung*. In [32], 101–118, 1989.

[37] Miranker, W. L. und Toupin, R. A. (Hrsg.): *Accurate Scientific Computations*. Lecture Notes in Computer Science, No. 235, Springer-Verlag, Berlin, 1986.

[38] Moore, R. E.: *Interval Analysis*. Prentice Hall, Engelwood Cliffs, New Jersey, 1966.

[39] Moore, R. E.: *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, Pensylvania, 1979.

[40] Moore, R. E. (Hrsg.): *Reliability in Computing: The Role of Interval Methods in Scientific Computations.* Academic Press, New York, 1988.

[41] Moore, R. E. und Ratschek, H.: *Inclusion Functions and Global Optimization II.* Mathematical Programming **41**, 341–356, 1988.

[42] Mohd, I. B.: *An Interval Global Optimization Algorithm for a Class of Functions with Several Variables.* Journal of Computational and Applied Mathematics **31**, 373–382, 1990.

[43] Neumaier A.: *The Enclosure of Solutions of Parameter-Dependent Systems of Equations.* In [40], 269–286, 1988.

[44] Neumaier A.: *Interval Methods for Systems of Equations.* Cambridge University Press, Cambridge, 1990.

[45] Novoa, M.: *Linear Programming Preconditioners for the Interval Gauss-Seidel Method.* Ph. D. Dissertation, University of Southwestern Louisiana, erscheint 1992.

[46] Numerik Software GmbH: *PASCAL–XSC – A PASCAL Extension for Scientific Computation.* User's Guide, Baden-Baden, 1991.

[47] Rall, L. B.: *Automatic Differentiation, Techniques and Applications.* Lecture Notes in Computer Science, No. 120, Springer-Verlag, Berlin, 1981.

[48] Ratschek, H. und Rokne, J.: *Computer Methods for the Range of Functions.* Ellis Horwood Limited, 1984.

[49] Ratschek, H. und Rokne, J.: *New Computer Methods for Global Optimization.* Ellis Horwood Limited, 1988.

[50] Ratz, D: *An Inclusion Algorithm for Global Optimization in a Portable PASCAL–XSC Implementation.* In: *Computer Arithmetic and Scientific Computation.* Proceedings of the SCAN 91, North-Holland, Elsevier, Amsterdam, erscheint 1992.

[51] Rump, S. M.: *Kleine Fehlerschranken bei Matrixproblemen.* Dissertation, Universität Karlsruhe, 1980.

[52] Rump, S. M.: *Solving Algebraic Problems with High Accuracy.* In [34], 51–120, 1983.

[53] Rump, S. M.: *New Results on Verified Inclusions.* In [37], 31–69, 1986.

[54] Shearer J. und Wolfe, M.: *An Improved Form of the Krawczyk-Moore Algorithm.* Applied Mathematics and Computations **17**, 229–239, 1985.

[55] Skelboe, S.: *Computation of Rational Interval Functions.* BIT **4**, 87–95, 1974.

[56] Stoer, J.: *Einführung in die Numerische Mathematik I.* Springer-Verlag, Berlin, Heidelberg, 1983.

[57] Törn, A. und Žilinskas, A.: *Global Optimization.* Lecture Notes in Computer Science, No. 350, Springer-Verlag, Berlin, 1989.

[58] Ullrich, Ch. (Hrsg.): *Contributions to Computer Arithmetic and Self-Validating Numerical Methods.* J. C. Baltzer AG, Scientific Publishing Co., IMACS, Basel, 1990.

[59] Walster, G., Hansen, E. und Sengupta, S.: *Test Results for a Global Optimization Algorithm.* In: Boggs, P., Byrd, R. und Schnabel, R. (Hrsg.): *Numerical Optimization 1984.* SIAM, Philadelphia, 272–287, 1985.

[60] Wolfe, M. A.: *Numerical Methods for Unconstrained Optimization – An Introduction.* Van Nostrand Reinhold, New York, 1978.

Stichwortverzeichnis

Lebenslauf

Dietmar Walter Ratz

geboren am	4. Dezember 1960 in Karlsruhe, Baden-Württemberg
Eltern	Kurt Ratz und Else Ratz, geb. Götz
Schulbildung	1967 – 1971 Grundschule in Stutensee-Friedrichstal 1971 – 1980 Goethe-Gymnasium in Karlsruhe
Reifeprüfung	20. Mai 1980
Wehrdienst	Juli 1980 bis Juni 1982
Studium	der Technomathematik mit den Nebenfächern Physik und Angewandte Informatik an der Universität Karlsruhe vom Wintersemester 1982/83 bis August 1988
Beurlaubung	im Sommersemester 1986 wegen eines Industriesemesters
Diplomprüfung	im Studiengang Technomathematik am 30. August 1988
Berufstätigkeit	• Oktober 1984 bis März 1985 als wissenschaftliche Hilfskraft am Mathematischen Institut II der Universität Karlsruhe • April 1984 bis März 1986 und Oktober 1986 bis September 1988 als wissenschaftliche Hilfskraft am Institut für Angewandte Mathematik der Universität Karlsruhe • April 1986 bis September 1986 als Praktikant/Werkstudent bei der SIEMENS AG, Karlsruhe • seit Oktober 1988 als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik der Universität Karlsruhe
Eheschließung	am 9. August 1985 mit Claudia Irene Ratz, geb. Jany
Kinder	Sebastian Patrick Ratz, geboren am 25. Mai 1987 Miriam Denise Ratz, geboren am 15. April 1990