Innere-Punkt-Methoden und automatische Ergebnisverifikation in der Linearen Optimierung

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Mathematik der
Universität Karlsruhe (TH)

genehmigte

Dissertation

von

Dipl.-Math. oec. Matthias Hocks

aus Berlin

Tag der mündlichen Prüfung:	18. Januar 1995
Referent:	Prof. Dr. U. Kulisch
Korreferent:	H.-Doz. Dr. W. Krämer

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik der Universität Karlsruhe. Das Thema entsprang der Suche nach numerischen Anwendungen aus dem Bereich der Betriebswirtschaftslehre, bei denen Verifikationsmethoden bisher weitgehend unbekannt aber dennoch in kritischen Fällen von großer Bedeutung sind. Die *Lineare Optimierung* mit ihrer derzeit rasanten Entwicklung bot hier ausgezeichnete Ansatzpunkte.

Für die Möglichkeit, dieses Thema zu bearbeiten, und für die Übernahme des Referats danke ich meinem Institutsleiter Herrn Prof. Dr. U. Kulisch. Mein Dank gilt ebenfalls Herrn H.-Doz. Dr. Walter Krämer für die Übernahme des Korreferats und für die hilfreichen Kommentare in der Endphase der Entstehung dieser Arbeit, sowie Herrn Prof. Dr. K. Neumann für das große Interesse, das er dem Inhalt meiner Arbeit entgegenbrachte. Besonders erwähnen möchte ich an dieser Stelle die fruchtbaren Gespräche zu allen (globalen) Fragen der Optimierung mit meinem Kollegen Herrn Dr. Dietmar Ratz. Ich danke außerdem ihm und Herrn Dr. Rolf Hammer für das (urlaubs-)zeitraubende Korrekturlesen des Manuskripts. Die angenehme Arbeitsatmosphäre, speziell in der Institutsaußenstelle „Engesserstraße 2“, haben die Fertigstellung der Arbeit ebenso positiv beeinflußt wie die fachlichen und TEXnischen Diskussionen bei gemeinsamen Projekten. Für die Unterstützung und die Organisation der institutsinternen Rahmenbedingungen gebührt mein Dank Herrn Dr. R. Klatte.

Bei meiner Frau Martina bedanke ich mich für die ständige Unterstützung auf dem Weg zur Promotion und für die fachlich kompetenten Ergänzungen einer Mathematikerin, die nicht durch „Verifikationspropaganda“ vorbelastet ist.

Zu guter Letzt danke ich auch meinen Eltern, die alle notwendigen Voraussetzungen während meines Ausbildungsweges geschaffen haben, und die Idee, nach dem Diplom an der Universität „noch weiter zu machen“, schon früh gefördert haben.

Inhaltsverzeichnis

Algorithmenverzeichnis

Abbildungsverzeichnis

Einleitung

Die lineare Optimierung, die das Problem der Minimierung oder Maximierung einer linearen Funktion unter Berücksichtigung einer endlichen Anzahl linearer Nebenbedingungen behandelt, ist ein wichtiges Anwendungsgebiet der Mathematik. Sie findet Verwendung in vielen Teilen der numerischen Mathematik, wie zum Beispiel bei der Behandlung bestimmter Typen von Anfangs- und Randwertaufgaben bei gewöhnlichen und partiellen Differentialgleichungen, bei Approximationsaufgaben oder in der Spieltheorie. Auch in wichtigen Anwendungsgebieten wie der Volks- und Betriebswirtschaftslehre wird sie eingesetzt. Gerade im Bereich betriebswirtschaftlicher Problemstellungen hat sich ein eigenständiges Forschungsgebiet von inzwischen großer wirtschaftlicher Bedeutung entwickelt, das unter dem Begriff Operations Research zusammengefaßt wird und in dem die lineare Optimierung eine zentrale Rolle einnimmt.

Aufgrund der wachsenden Leistungsfähigkeit moderner Rechenanlagen ist die Behandlung umfangreicher Optimierungsprobleme, die ohne dieses technische Hilfsmittel nicht lösbar wären, eine Routineaufgabe der Numerik geworden. Die Entwicklung moderner Algorithmen zielte daher bislang stets auf eine Steigerung der Effizienz und die Begrenzung des numerischen Aufwands. Dies führte nach umfangreichen Forschungen und Untersuchungen Mitte der 80er Jahre zu Innere-Punkt-Methoden, die wegen ihres polynomial begrenzten Rechenaufwandes den bis dahin konkurrenzlosen Simplex-Algorithmus in vielen Fällen ersetzen. Die vorliegende Arbeit basiert sowohl auf dem heute als „Stand der Technik“ zu bezeichnenden asymptotischen Iterationsverfahren der Innere-Punkt-Methode als auch auf der Technik des direkten Simplex-Verfahrens. Hierbei ist es gelungen, eine Synthese der Vorzüge beider Verfahren zu realisieren. Das asymptotische Verfahren liefert mit geringem numerischen Aufwand eine gute Näherungslösung. Das direkte Verfahren startet im Anschluß an aussichtsreicher Stelle, so daß der Aufwand begrenzt bleibt, und bestimmt die optimale Lösung in Form eines Einschlusses. Im Vordergrund steht hierbei die Problematik der Kontrolle

numerisch ermittelter Lösungen. Die Tatsache, daß bei der Ausführung eines numerischen Algorithmus auf einem Rechner die tatsächliche reelle Operation durch sogenannte Gleitpunktoperationen ersetzt wird, macht den Einsatz einer nach mathematischen Gesichtspunkten definierten Rechnerarithmetik notwendig. Hierdurch werden Aussagen über die Qualität der berechneten Ergebnisse auch nach vielen Millionen Rechenoperationen möglich. Unter Qualität des Ergebnisses wird hierbei der Nachweis der Existenz und gegebenenfalls auch Eindeutigkeit einer berechneten Lösung sowie deren Optimalität und Zulässigkeit, die das Erfüllen aller Nebenbedingungen aus der Problemstellung bedeutet, verstanden. Dies ist mit Hilfe intervallanalytischer Prinzipien, der Fixpunkttheorie und einer programmiersprachlichen Umgebung, die, wie PASCAL–XSC, eine mathematisch präzise definierte Rechnerarithmetik zugänglich macht, auch auf dem Rechner nachprüfbar. Die Notwendigkeit der automatischen Ergebnisverifikation gerade bei „großen“ Anwendungsproblemen, bei denen Zulässigkeit und Optimalität der Lösung nicht mehr durch einfache Plausibilitätsprüfungen abgeschätzt werden können, gewinnt immer stärker an Bedeutung, wenn numerische Verfahren, beispielsweise zur Behandlung sicherheitskritischer Optimierungsmodelle, in der Praxis eingesetzt werden. Diese Lücke im Bereich der Kontrolle numerischer Berechnungen, die in der Reihe der existierenden Verfahren zur Lösung linearer Optimierungsaufgaben besteht, wird mit dem in dieser Arbeit vorgestellten Verfahren erstmals geschlossen.

In den folgenden einleitenden Abschnitten wird eine allgemeine Motivation und ein Überblick über die historische Entwicklung der Lösungsmethoden im Gebiet der linearen Optimierung gegeben. Dabei wird die Zielsetzung der Arbeit definiert und der Zusammenhang zu verwandten Forschungsgebieten und Veröffentlichungen aufgezeigt.

Die für das selbstverifizierende Lösungsverfahren notwendigen mathematischen Grundlagen der Rechnerarithmetik und der Intervallrechnung sowie die allgemeinen Definitionen und Sätze der Optimierungstheorie sind im ersten Kapitel zusammengestellt. Hier werden auch die Theorien des Barriere-Verfahrens, der Methode der Lagrangeschen Multiplikatoren und des Newton-Verfahrens als die wesentlichen Komponenten der Innere-Punkt-Methode vollständig hergeleitet und anhand einzelner Beispiele erläutert.

Das eigentliche Thema der Arbeit, die automatische Ergebnisverifikation in der linearen Optimierung, behandelt Kapitel 2. Es wird, ausgehend vom Karmarkar-Algorithmus in seiner ursprünglichen Form, die äquivalente primale Innere-Punkt-Methode entwickelt. Der Übergang zur numerisch

stabileren und aus Verifikationssicht besser kontrollierbaren primal-dualen Innere-Punkt-Methode, die sich auf den Aussagen der Dualitätssätze der linearen Optimierung begründet, ist vollständig dargestellt. Auf der Grundlage der Theorie des Simplex-Algorithmus wird ein Verfahren vorgestellt, das, ausgehend vom Resultat der primal-dualen Innere-Punkt-Methode, die Optimalität der zugehörigen Basis-Lösung verifiziert und gegebenenfalls benachbarte, ebenfalls optimale Basis-Lösungen bestimmt.

Nach einer Präsentation der für die Ergebnisverifikation notwendigen Änderungen und Ergänzungen einzelner Verfahrenskomponenten folgt, als zentrales Resultat dieser Arbeit, die vollständige algorithmische Darstellung der theoretischen Herleitung. Das kombinierte Lösungsverfahren liefert garantierte Aussagen über die Existenz einer Lösung des Optimierungsproblems, sowie über die Zulässigkeit einer gefundenen Lösung und gegebenenfalls über deren Eindeutigkeit. Es bestimmt weiterhin einen Einschluß des optimalen Zielfunktionswertes und Einschließungen aller (numerisch) optimalen Basis-Lösungen.

Eine Implementierung des Verfahrens ist in der wissenschaftlichen Programmiersprache PASCAL–XSC angegeben. Durch das in die Sprache integrierte allgemeine Operatorkonzept ist es möglich, die nach dem Semimorphismus-Prinzip definierten Grundverknüpfungen für reelle Gleitpunktzahlen und den darüber definierten Vektoren und Matrizen, sowie die entsprechenden Intervalldatentypen über die üblichen mathematischen Symbole anzusprechen. Dies ermöglicht eine nahezu unveränderte Übertragung der mathematischen Notation des Verfahrens in ein modular aufgebautes Programm zur Lösung linearer Optimierungsaufgaben mit automatischer Ergebnisverifikation.

Lineare Optimierungsprobleme und Anwendungsgebiete

Zahlreiche Fragestellungen aus Technik, Naturwissenschaften und Wirtschaftswissenschaften führen auf Optimierungsaufgaben, die man zumindest näherungsweise durch ein mathematisches Modell beschreiben kann. Ein lineares Optimierungsproblem, auch lineares Programm (LP) genannt, behandelt die Aufgabenstellung, eine lineare Zielfunktion ($f : \mathbb{R}^n \rightarrow \mathbb{R}$) unter Berücksichtigung endlich vieler linearer Nebenbedingungen, die in Gleichungs- und Ungleichungsform vorliegen, zu minimieren oder maximieren.

Ein typisches Beispiel hierfür stammt aus der Produktionsplanung. In

einem Betrieb können drei verschiedene Produkte P1, P2 und P3 unter Verwendung von vier Ressourcen R1, R2, R3 und R4 (Rohstoffe, Maschinen etc.) hergestellt werden. Die zur Herstellung einer Einheit eines Produktes notwendigen Ressourcen, die Verfügbarkeit der jeweiligen Ressourcen sowie der mögliche Gewinn je Einheit eines Produktes sind in der folgenden Tabelle angegeben:

Ressourcen	P1	P2	P3	Kapazität
R1	1	3	2	30
R2	4	2	1	25
R3	3	4	3	45
R4	2	3	5	50
Gewinn	5	8	4	

Man versucht nun die Mengen x_1, x_2, x_3 der herzustellenden Produkte so zu bestimmen, daß der Gewinn maximal wird. Das führt zu der Aufgabe:

$$\begin{aligned}
&\text{maximiere} && f(x) = 5x_1 + 8x_2 + 4x_3 \\
&\text{unter den Nebenbedingungen} && x_1 + 3x_2 + 2x_3 \leq 30 \\
& && 4x_1 + 2x_2 + x_3 \leq 25 \\
& && 3x_1 + 4x_2 + 4x_3 \leq 45 \\
& && 2x_1 + 3x_2 + 5x_3 \leq 50 \\
&\text{und} && x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0.
\end{aligned}$$

Diese Klasse von Optimierungsaufgaben hat besonders breiten Eingang in viele Anwendungen aus den Wirtschafts- und Ingenieurwissenschaften gefunden (siehe [51] und [53]). Die Ursache dafür ist, daß sehr viele konkrete Probleme in diese Aufgabenklasse fallen, die dann einheitlich mit einem sehr effizienten Berechnungsverfahren behandelt werden können. Es besteht auch ein Zusammenhang zwischen linearen Optimierungsproblemen und Optimierungsaufgaben, die nicht-lineare Terme in der Zielfunktion oder den Nebenbedingungen zulassen. Oft kann ein allgemeines Optimierungsproblem durch sukzessive Lösung linearer Optimierungsprobleme gelöst werden. Für manche schwierige Aufgabenstellung kann die lineare Programmierung zur Berechnung einer Näherungslösung verwandt werden.

Bei vielen wissenschaftlichen Beobachtungen ist es notwendig, zu einer Funktion, die nur durch Meßwerte oder eine Kurve gegeben ist, einen analytischen Ausdruck zu finden, der diese Funktion approximiert. Ein ähnliches Problem kann sich auch bei einer Funktion ergeben, die durch eine Formel

gegeben ist, wenn diese Formel entweder zu kompliziert oder für die geforderten Zwecke ungeeignet ist (zum Beispiel wenn eine Funktion integriert werden soll, ihr Integral aber nicht durch elementare Funktionen darstellbar ist).

Sei beispielsweise auf dem reellen Intervall $[a,b]$ eine stetige Funktion $f : [a,b] \to \mathbb{R}$ und ein von m stetigen Funktionen $v_1, \ldots, v_m : [a,b] \to \mathbb{R}$ erzeugter Funktionenteilraum $V = \{v = \sum_{i=1}^{m} \alpha_i v_i \mid \alpha_i \in \mathbb{R}\}$ gegeben (z.B. mit $v_i = x^i$ die Menge der Polynome von maximalem Grade m). Gesucht wird eine Funktion $v \in V$, die f möglichst gut annähert, wobei dies in der folgenden Bedeutung gemeint ist:

$$\min_{\alpha \in \mathbb{R}^m} \quad f(\alpha) := \max_{a \le t \le b} |f(t) - \sum_{i=1}^{m} \alpha_i v_i(t)|. \tag{0.1}$$

Bei (0.1) spricht man von der *besten Tschebyschew-Approximation*.

Man betrachte zunächst die diskrete Tschebyschew-Approximation

$$\min_{\alpha \in \mathbb{R}^m} \quad f(\alpha) := \max_{j \in \{1,\ldots,n\}} |f_j - \sum_{i=1}^{m} \alpha_i v_{ij}|, \tag{0.2}$$

wobei v_{ij} die j-te Komponente des Vektors $v_i \in \mathbb{R}^n$ bezeichne. Nun ist V ein von den m Vektoren $v_1, \ldots, v_m$ erzeugter Teilraum des $\mathbb{R}^n$. Führt man dann den Wert $f(\alpha)$ als eine neue Variable ein, d.h. $\alpha_{n+1} := f(\alpha)$, so ist (0.2) äquivalent zu der folgenden linearen Optimierungsaufgabe:

$$\text{minimiere} \qquad \alpha_{n+1} \tag{0.3a}$$

$$\text{unter den Nebenbedingungen} \quad -\alpha_{n+1} \le f_j - \sum_{i=1}^{m} \alpha_i v_{ij} \le \alpha_{n+1} \tag{0.3b}$$

$$\text{für} \quad j \in \{1, \ldots, n\}$$

Um jetzt eine Näherungsaufgabe für (0.1) zu bekommen, kann man n Punkte $t_1, \ldots, t_n$ in $[a,b]$ wählen und die Aufgabe (0.3) mit

$$f_j := f(t_j) \quad \text{und} \quad v_{ij} := v_i(t_j) \tag{0.4}$$

benutzen. Da es möglich ist, die Tschebyschew-Approximation (0.1) auf die Lösung eines allgemeinen (nicht-linearen) Gleichungssystems zurückzuführen, kann die durch (0.3) und (0.4) erhaltene Näherungslösung als Startwert bei der numerischen Behandlung dieses Gleichungssystems verwendet werden.

Neben den genannten, eher theoretischen Problemstellungen treten in der Praxis oft ähnliche Optimierungsprobleme auf. In den Wirtschaftswissenschaften sind dies neben der Produktionsplanung auch Fragen der Preispolitik und Personalplanung. Ständig müssen hierbei Entscheidungen über die Preisgestaltung getroffen werden, wobei die Einflüsse von Nachfrage, langfristiger Produktionsentwicklung und gesamtökonomischen Einflüssen in zum Teil sehr komplexen Modellen abgebildet werden. Auch am Kapitalmarkt gewinnen rechnergestützte Optimierungsverfahren, zum Beispiel bei der Zusammenstellung des Portfolios eines Aktienfonds, an Bedeutung. Modelle, die auf lineare Optimierungsprobleme führen, finden sich auch bei Werkstoffanalysen und der Qualitätssicherung. Bei der Überprüfung der Reinheit eines pharmazeutischen Produktes, die für eine Zulassung durch die Aufsichtsbehörden garantiert sein muß, werden die (möglicherweise toxischen) Reaktionsmöglichkeiten der im Produkt nachgewiesenen chemischen Nebenprodukte abgeschätzt. Typische Problemstellungen ergeben sich auch im Bereich des Transportwesens, wo etwa die Routenplanung und der Personaleinsatz in einem Speditionsunternehmen „alltägliche“ Optimierungsaufgaben sind. Auch die Bestückung eines Flugzeuges und die Wechselwirkung zwischen Flugstrecke, Gewicht, Wettereinflüssen und Kerosinverbrauch lassen sich in dieser Weise modellieren, um eine Minimierung der Kosten zu erreichen [53].

Überblick über die Entwicklung der Lösungsmethoden

In den Jahren vor 1984 wurde die Frage nach dem Lösungsalgorithmus für lineare Optimierungsprobleme nahezu ausnahmslos mit einem Hinweis auf die 1947 von *George Dantzig* entwickelte *Simplex-Methode* beantwortet [12].

Wie weithin bekannt ist, stellt die *Simplex-Methode* ein iteratives Verfahren dar, das auf einer fundamentalen Eigenschaft aller linearen Optimierungsprobleme beruht:

> Eine optimale Lösung eines linearen Optimierungsproblems liegt in einer Ecke des durch die linearen Nebenbedingungen definierten konvexen zulässigen Bereiches.

Ausgehend von einer Startecke bewegt sich der Simplex-Algorithmus in jedem Iterationsschritt zu einer benachbarten Ecke, wobei der Wert der Zielfunktion f verbessert wird. Dieser Vorgang wird solange wiederholt, bis keine Verbesserung mehr möglich ist. Die zuletzt bestimmte Ecke stellt

eine optimale Lösung dar. Aufgrund der Tatsache, daß die Anzahl der Ecken beschränkt und endlich ist, kann (unter schwachen Voraussetzungen) die Konvergenz des Verfahrens garantiert werden.

Auch wenn alternative Strategien von Zeit zu Zeit vorgestellt und getestet wurden, so erreichten diese Techniken bis in die jüngere Vergangenheit den Simplex-Algorithmus in puncto Zuverlässigkeit und Geschwindigkeit nicht. Aus diesem Grund blieb der Simplex-Algorithmus unumstritten die Nummer 1 unter den Lösungsmethoden für lineare Optimierungsprobleme, auch wenn hier die Problematik der theoretischen Komplexität dieses Verfahrens nicht unerwähnt bleiben darf.

Eine derartige Fixierung auf die Simplex-Methode hatte unterschiedliche Auswirkungen auf das Forschungsgebiet *Lineare Optimierung*. Aus vornehmlich historischen Gründen umgibt sich der Simplex-Algorithmus mit einer Vielzahl an Spezialausdrücken wie zum Beispiel *zulässige Basis-Lösung* oder *Tableau*, die wenig Ähnlichkeit zu Begriffen der allgemeinen Optimierungstheorie haben, und sogar der Name Simplex-Algorithmus ist mehrfach belegt. Dies liegt unter anderem an der unbestreitbaren militärischen Bedeutung des Forschungsgebietes und sollte Anfang der 40er Jahre beim Kriegsgegner gezielt Verwirrung stiften [18]. Die lineare Optimierung wurde daher von vielen als ein von der allgemeinen Optimierung vollständig unabhängiges Gebiet betrachtet, was wiederum dazu führte, daß *neue* Entwicklungen in der linearen Programmierung (fast) ausschließlich Abwandlungen der Simplex-Methode waren.

Im Gegensatz dazu zeichnete sich das Gebiet der allgemeinen, nichtlinearen Optimierung durch eine konstante Entwicklung neuer Methoden und Strategien aus, wobei im Laufe der Zeit verschiedene Lösungsmethoden bevorzugt wurden. In den 60er Jahren wurden beschränkte Optimierungsprobleme meist in unbeschränkte Teilprobleme überführt, wobei insbesondere *Strafkostenverfahren* wie das *Penalty-* oder *Barriere-Verfahren* zum Einsatz kamen, die beide die Strategie verfolgen, eine zusammengesetzte Funktion, die sowohl die Zielfunktion als auch den Einfluß der Nebenbedingungen widerspiegelt, zu minimieren. Klassische Barriere-Verfahren, angewandt auf Nebenbedingungen in Ungleichungsform, führen auf eine zusammengesetzte Funktion, die eine unüberwindbare positive Singularität (Barriere) am Rand des zulässigen Bereiches enthält. Hierdurch wird in jedem Iterationsschritt eine strikte Zulässigkeit der Iterierten während des Approximationsprozesses garantiert. In den 70er Jahren wurden nicht-lineare beschränkte Optimierungsprobleme vorzugsweise mit Hilfe sequentieller quadratischer Programmierungsmethoden, die eine Folge beschränkter Teilpro-

bleme auf der Basis *Lagrangescher Funktionen* behandeln, gelöst.

Obwohl Barriere-Methoden in den 60er Jahren vielfach eingesetzt und eingehend analysiert wurden, haben sie in den 70er Jahren stark an Popularität eingebüßt, nicht zuletzt wegen einerseits inhärent schlechter Kondition des numerischen Verfahrens sowie andererseits deutlich geringerer Effizienz im Vergleich zu alternativen Lösungsstrategien.

Aus heutiger Sicht stellt sich die Situation seit 1984 sowohl für lineare als auch für allgemeine Problemstellungen in der Optimierung wesentlich anders dar. Dies ist nicht zuletzt auf die unbefriedigende theoretische Komplexität des Simplex-Algorithmus zurückzuführen.

Bei der Lösung kleiner und mittlerer Probleme erweist sich die Simplex-Methode unverändert als extrem effizient, wobei die Anzahl der Iterationen allgemein nur ein kleines Vielfaches (2-3) der Problemgröße beträgt. Da die Zahl der Ecken des zulässigen Bereiches, der durch ein (LP) definiert wird, endlich ist, konvergiert das Verfahren in der Regel – allerdings kann die Anzahl der Ecken exponentiell mit der Dimension des Problems wachsen. Das bekannteste Beispiel in diesem Zusammenhang ist der „twisted cube“ (Klee/Minty [33]), der ein lineares Optimierungsproblem mit n Unbekannten und $2n$ Nebenbedingungen repräsentiert. Die Standardversion des Simplex-Algorithmus mit Pivotisierung passiert in diesem Fall alle 2^n Ecken des zulässigen Bereiches. Der „worst case“ Aufwand des Simplex-Verfahrens, das heißt die maximale Anzahl arithmetischer Operationen, die zur Lösung eines allgemeinen (LP) notwendig sind, hängt dementsprechend exponentiell von der Dimension des Problems ab. Der große Unterschied zwischen in praktischen Anwendungen festgestelltem $\boldsymbol{O}(3n)$ und worst case Aufwand $\boldsymbol{O}(2^n)$ ist allerdings ein bisher ungelöstes Phänomen. Ob zum Beispiel eine (bisher unentdeckte) Pivotisierungsstrategie die Komplexität des Simplex-Algorithmus verringern könnte, ist eine offene Frage.

Seit in den 60er und 70er Jahren Komplexitätsuntersuchungen an Bedeutung gewonnen haben, besteht eine weitgehende Übereinstimmung, daß ein Algorithmus nur dann als *schnell* bezeichnet wird, wenn sein Aufwand polynomial beschränkt ist. Dies bedeutet, daß die Zahl der Operationen, die zur Lösung der Problemstellung benötigt werden, durch ein Polynom in n nach oben beschränkt werden kann. Die Simplex-Methode erfüllt diese Bedingung eindeutig nicht. Obwohl sich das Verfahren in der Praxis auch bei Problemen großer Dimensionen erfolgreich einsetzen ließ und läßt, blieb die Suche nach einem beweisbar polynomial beschränkten Algorithmus ein zentrales Thema.

1979 veröffentlichte *Leonid Khachian* [30] den ersten Algorithmus zur

Lösung linearer Optimierungsprobleme, dessen Aufwand polynomial begrenzt ist. Khachians *Ellipsoid-Methode* beruht auf bekannten Techniken der allgemeinen, nicht-linearen Optimierung, die unter anderem auf Shor, Yudin und Nemirovsky [48] zurückgehen. Beachtenswert ist die Tatsache, daß Khachians Theorie unabhängig von den kombinatorischen Eigenschaften linearer Optimierungsprobleme ist. Statt dessen wird eine Folge von Ellipsoiden konstruiert, wobei jedes Folgenelement die optimale Lösung enthält und gleichzeitig echt im Inneren des vorhergehenden Folgenelementes liegt. Die Ellipsoid-Methode generiert verbesserte Iterierte im Sinne einer sukzessiven monotonen Verkleinerung der Umgebung der Lösung. (Die Iterierten, die durch die Simplex-Methode generiert werden, führen ebenfalls zu einer verbesserten Näherung der Lösung, indem der Zielfunktionswert verbessert wird. Es wird allerdings keine Information über die Nähe zur tatsächlichen Lösung gegeben.)

Wesentlich in Bezug auf den polynomialen Aufwand der Ellipsoid-Methode sind die sogenannten äußeren und inneren Schranken der optimalen Lösung. Die äußere Schranke garantiert ein einschließendes Anfangsellipsoid zu Beginn des Verfahrens. Die innere Schranke, die eine geforderte Nähe zur exakten Lösung garantiert, wird durch die Größe des zuletzt bestimmten Ellipsoids definiert.

Trotz der Eigenschaft, daß der Aufwand des Verfahrens polynomial begrenzt ist, erwies sich die Ellipsoid-Methode in praktischen Anwendungen als wenig erfolgreich. Der Aufwand zur Lösung praktischer Optimierungsprobleme lag fast immer in der Größenordnung des „worst case“ und der ist, wenn auch polynomial begrenzt, sehr groß. Dementsprechend behielt der Simplex-Algorithmus seine Vorrangstellung in allen praktischen Laufzeitvergleichen. Die Entwicklung der Ellipsoid-Methode führte auf die unerwartete Anomalie, daß ein Algorithmus, dessen Aufwand theoretisch polynomial begrenzt ist, im Vergleich mit einem Algorithmus mit exponentiellem „worst case“ Aufwand in Bezug auf die Geschwindigkeit in der Praxis schlechter abschneidet. Die Frage nach einem Lösungsalgorithmus für lineare Optimierungsprobleme, dessen Aufwand polynomial begrenzt ist *und* der in der Praxis effizient einsetzbar ist, blieb daher offen.

Die Ankündigung *Narendra Karmarkars* [27] 1984, eine neue *Innere-Punkt-Methode* entwickelt zu haben, deren Aufwand polynomial begrenzt ist und die im praktischen Vergleich mit dem Simplex-Algorithmus konkurrenzfähig und oft sogar überlegen ist, hat zu immensem öffentlichen Interesse geführt. In Praxis und Forschung wurden daraufhin zahlreiche Untersuchungen durchgeführt, die die vorteilhaften Eigenschaften des Karmarkar-

Algorithmus auch für sehr große Anwendungen bestätigten [36]. Als äußerst problematisch erwies sich zunächst die theoretische Untersuchung des Algorithmus auf Grund seiner komplizierten Darstellung und seiner scheinbar geringen Verwandtschaft zu bekannten Verfahren. Dies verzögerte eine marktfähige Implementierung mit universeller Einsetzbarkeit bis 1989. Erst als es Gill u.a. [19] gelang, eine Äquivalenz zwischen *Karmarkar-Algorithmus* und den klassischen Barriere-Methoden zu zeigen, was zu einer wesentlichen Vereinfachung der Herleitung, Motivation und des Verständnisses führte, fand eine intensive Forschung im Bereich der Grundlagen der linearen und allgemeinen Optimierung statt. Im Gegensatz zum Simplex-Algorithmus sind Innere-Punkt-Methoden ganz offensichtlich auch auf allgemeine Problemstellungen anwendbar, denn Barriere- und Penalty-Verfahren sowie die Methoden der Lagrangeschen Multiplikatoren wurden gerade für diese Zwecke entwickelt. Es ist inzwischen gelungen, quadratische, nicht-linear konvexe und zum Teil kombinatorische Probleme erfolgreich mit Innere-Punkt-Methoden zu lösen [28].

Die zentrale Strategie der *Innere-Punkt-Methode* ist die Konstruktion einer Folge kontinuierlich parametrisierter, unbeschränkter Optimierungsprobleme, deren Lösungen asymptotisch gegen die exakte Lösung konvergieren. Das Verfahren läßt sich ganz grob wie folgt beschreiben.

1. Wähle einen Parameter $\mu > 0$.

2. Konstruiere ein (zum Ausgangsproblem asymptotisch äquivalentes) *unrestringiertes* Optimierungsproblem bezüglich des Parameters μ.

3. Löse das unrestringierte Optimierungsproblem unter Verwendung klassischer (allgemeiner) Verfahren.

4. Berechne den Zielfunktionswert.

5. Prüfe, ob der Zielfunktionswert hinreichend nahe am Optimum liegt.
   Falls nein: verkleinere den Parameter μ und gehe zu 2;
   sonst: Lösung gefunden.

Ausgehend von einem (strikt) zulässigen Startpunkt im Innern des zulässigen Lösungsbereiches nähert sich der Iterationsprozeß in jedem Schritt des Karmarkar-Algorithmus der exakten Lösung an, während der Parameter μ gegen 0 geht. Auf Grund von Barriere-Termen wird ein Verlassen des zulässigen Bereiches verhindert. Kurze und übersichtliche Beschreibungen des Verfahrens finden sich in [45], [55] und [63].

Anforderungen an aktuelle Algorithmen und Implementierungen

Bei komplexen Problemstellungen, die mathematisch formuliert und numerisch gelöst werden können, ist der Einsatz leistungsfähiger Prozessoren und moderner Rechnerarchitekturen Stand der Technik. Bei der Entwicklung neuer Rechnergenerationen stehen seit langem Rechengeschwindigkeit und Speicherkapazität als Fortschrittsziele an erster Stelle. Die beeindruckenden Erfolge auf diesem Gebiet ermöglichen die Bearbeitung immenser Datenmengen, wie sie beispielsweise bei Optimierungsmodellen in der Produktionsplanung großer Industrieunternehmen auftreten, mit akzeptablem Zeitaufwand. Vielfach werden Modelle entsprechender Dimension, das heißt mehrere tausend Unbekannte und einige tausend Nebenbedingungen, bei Simulationen und Parameterstudien eingesetzt. Berechnungen dieser Art werden in der Regel unter Verwendung der von der Hardware-Industrie zur Verfügung gestellten Gleitpunktarithmetik durchgeführt. Diese Technik, bei der die tatsächlichen reellen Operationen durch Gleitpunktoperationen ersetzt werden, birgt aber bei jeder ausgeführten Rechenoperation die Gefahr von Rundungsfehlern und Auslöschungen in sich, die das Berechnungsergebnis zum Teil erheblich verfälschen können. Im Falle linearer Optimierungsprobleme kann dies zu Lösungen führen, die Nebenbedingungen verletzen oder die weit entfernt von der tatsächlich optimalen Lösung liegen. Es ist auch möglich, daß ein Ergebnis für eine Optimierungsaufgabe berechnet wird, obwohl sie gar keine zulässige Lösung besitzt oder obwohl ihre Zielfunktion unbeschränkt ist (siehe Kapitel 4). Solch fehlerhafte Ergebnisse können mittels herkömmlicher Fehlerrechnung wegen der unübersehbaren Anzahl an Rechenoperationen weder kontrolliert noch durch simple Plausibilitätsuntersuchungen überprüft werden. Zur Überwindung dieser Problematik muß der Rechner in die Lage versetzt werden, automatisch die Kontrolle seiner Berechnungen durchzuführen. Dies wird durch die Definition einer Rechnerarithmetik nach mathematischen Gesichtspunkten ermöglicht, bei der die Grundoperationen für reelle Gleitpunktzahlen nach dem Semimorphismus-Prinzip bestimmt werden. Diese wurde von Kulisch und Miranker in [39], [41] und [42] angegeben.

Auf der Basis einer präzise definierten Rechnerarithmetik ist die Implementierung einer Intervallarithmetik möglich, die als zentrales Hilfsmittel der Ergebnisverifikation dient. Die Realisierung dieser Hilfsmittel ist in die wissenschaftlichen Programmiersprachen PASCAL–XSC [31], C–XSC [32] und ACRITH–XSC [61] integriert. Durch ein allgemeines Operatorkonzept

ist es möglich, die reellen Grundoperationen sowohl für Skalare als auch für Vektoren und Matrizen sowie die entsprechenden Intervalldatentypen mit den üblichen mathematischen Symbolen anzusprechen.

Neben diesen hardware-nahen Hilfsmitteln der Arithmetik und der Programmiersprache gilt in der Numerik ein wesentliches Augenmerk der algorithmischen Seite der Problemlösung komplexer Aufgabenstellungen. Es wird allgemein verlangt, daß der Rechenaufwand für die Lösung eines Problems abhängig von dessen Dimension polynomial begrenzt bleibt, um in der Praxis mit vertretbaren Rechenzeiten arbeiten zu können. Da gerade auf diesem Gebiet der Simplex-Algorithmus wegen des exponentiellen Anstiegs seines theoretischen Rechenaufwandes teilweise unbefriedigende Ergebnisse liefert, führten zahlreiche Forschungsprojekte auf die Innere-Punkt-Methoden, für die ein nur polynomiales Anwachsen des Rechenaufwandes mit der Problemdimension beweisbar ist. Wenig Engagement gab es demgegenüber auf dem Gebiet der Kontrolle der Berechnungsergebnisse, der sogenannten automatischen Ergebnisverifikation. Die von einem (schnellen und effizienten) Lösungsalgorithmus gelieferten Berechnungsergebnisse mußten aufgrund der Unüberschaubarkeit der ausgegebenen Datenmengen quasi „geglaubt“ werden und konnten nur durch Vergleichsrechnungen mit leicht veränderten Anfangswerten und Randbedingungen „überprüft“ werden. Daß derartige „Plausibilitätsprüfungen“ keine Garantie über die Korrektheit der Berechnungsergebnisse liefern kann, liegt auf der Hand. Mitte der 80er Jahre wurden daher von Rump [57] und Jansson ([24], [25]) Untersuchungen durchgeführt, die zu einem a-posteriori Verfahren zur Verifikation der Berechnungsergebnisse des Simplex-Algorithmus führten. Eine präzise Beschreibung und eine frei verfügbare Implementierung des Verfahrens findet sich in [20]. Auch auf dem verwandten Gebiet der globalen Optimierung hat es in den letzten Jahren eine Vielzahl von Aktivitäten und Lösungsstrategien zur automatischen Ergebnisverifikation gegeben, die zum Teil sogar im Vergleich des Rechenaufwandes herkömmlichen Näherungsverfahren überlegen sind. Veröffentlichungen hierzu stammen unter anderem von Csendes [11], Jansson [26] und Ratz [54].

Für die Innere-Punkt-Methode, die den „state of the art“ Lösungsalgorithmus für große lineare Optimierungsprobleme darstellt, ist die Verifikation der Berechnungsergebnisse allerdings noch eine offene Frage. Zur Schließung dieser Wissenslücke leistet die vorliegende Arbeit einen Beitrag. Es wird eine geschlossene Zusammenstellung der mathematischen Grundlagen der linearen Optimierung (Optimalitätsbedingungen, konvexe Optimierung) gegeben. Darauf aufbauend werden die drei wesentlichen Komponen-

ten der Innere-Punkt-Methode — das Barriere-Verfahren, die Methode der Lagrangeschen Multiplikatoren und das Newton-Verfahren — vollständig hergeleitet und mit einzelnen Beispielen beschrieben. Unter Verwendung dieser Komponenten folgt eine kompakte und übersichtliche Darstellung der primal-dualen Innere-Punkt-Methode und eine detaillierte Beschreibung der für die Verifikation der Berechnungsergebnisse notwendigen Ergänzungen des Algorithmus. Um den Nachteil des asymptotischen Verfahrens, das immer nur eine Näherung, nicht aber einen Einschluß der exakten Lösung liefern kann, zu überwinden, werden in einem anschließenden direkten Verifikationsschritt alle optimalen Basis-Lösungen bestimmt. Zusammenfassend wird ein vollständiger selbst-verifizierender Lösungsalgorithmus für lineare Optimierungsprobleme präsentiert und bis zur praktischen Einsetzbarkeit implementiert. Der Algorithmus garantiert Aussagen sowohl über die Existenz einer Lösung, als auch über deren Optimalität und Zulässigkeit. Es werden Einschließungen für den optimalen Zielfunktionswert und für gegebenenfalls alle optimalen Basis-Lösungen bestimmt. Falls möglich, wird die Eindeutigkeit der bestimmten optimalen Lösung verifiziert. Zur automatischen Überprüfung dieser Aussagen auf dem Rechner werden die Hilfsmittel der Rechner- und Intervallarithmetik eingesetzt und die Prinzipien der Fixpunkttheorie ausgenutzt. Durch die Implementierung in PASCAL–XSC kann einerseits die mathematische Notation der entwickelten Algorithmen beibehalten werden, und es ist andererseits möglich, das entstandene Programm auf alle gängigen Rechnerplattformen zu übertragen. Eine ausführbare Version der Verfahrens für Personal Computer und Workstations wird auf einem allgemein zugänglichen FTP–Server der Universität Karlsruhe zur Verfügung gestellt.

Kapitel 1

Mathematische und arithmetische Grundlagen

1.1 Bezeichnungen und Darstellungen

In der vorliegenden Arbeit bezeichnet stets

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

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

$x_i \in \mathbb{R}$	die i-te Komponente des Vektors x,
$x^{(k)} \in \mathbb{R}^n$	den k-ten Vektor in einer Aufzählung oder Folge,
$e \in \mathbb{R}^n$	den n-dimensionalen Einsvektor, d.h. $e_i = 1$,
$a_{i*} \in \mathbb{R}^n$	die i-te Zeile der Matrix A,
$a_{*i} \in \mathbb{R}^n$	die i-te Spalte der Matrix A,
$a_{ij} \in \mathbb{R}$	die j-te Komponente der i-ten Zeile der Matrix A,
$A^T \in \mathbb{R}^{m\times n}$	die Transponierte der Matrix A,
$A^{-1} \in \mathbb{R}^{n\times n}$	die Inverse der Matrix A und
$diag(x) = X \in \mathbb{R}^{n\times n}$	die $(n \times n)$-dimensionale Diagonalmatrix mit $X_{ii} = x_i$.

Für die inneren und äußeren Multiplikationen von reellen Zahlen, Vektoren und Matrizen wird stets das Symbol „$\cdot$“, das wie üblich auch weggelassen werden darf, verwandt. Das Skalarprodukt zweier Vektoren $a, b \in \mathbb{R}^n$ wird somit beispielsweise bezeichnet durch

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

Für eine reelle Funktion $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ 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 *Hesse-Matrix* von f.

Für eine reelle Funktion $g : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ mit $g \in C^1(D)$ bezeichnet

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

die *Jacobi-Matrix* von g.

Nach dem Satz von Schwarz ist die Hesse-Matrix für $m = n$ symmetrisch, und es gilt außerdem mit $g(x) = \nabla f(x)$ die Beziehung $J_g(x) = \nabla^2 f(x)$ [60].

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 $x \neq 0$.

Spezielle Bezeichnungen der Optimierungstheorie sind

(LP)	das (primale) lineare Optimierungsproblem in Standardform,
(GrundLP)	das (primale) lineare Optimierungsproblem in Grundform,
(DP)	das duale lineare Optimierungsproblem,
(ECP)	das durch Gleichungen beschränkte Optimierungsproblem,
(ICP)	das durch Ungleichungen beschränkte Optimierungsproblem,
$B(x \mid \mu)$	die Barriere-Funktion bzgl. des Barriere-Parameters μ,
$L(x,\lambda)$	die Lagrangesche Funktion,
$\mathcal{B}$ bzw. $\mathcal{N}$	die Menge der Basis- bzw. Nichtbasis-Indizes,
β_i bzw. ν_i	Basis- bzw. Nichtbasis-Index,
$\mathcal{A} \setminus \mathcal{B}$	die Menge $\mathcal{A}$ ohne die Menge $\mathcal{B}$ und
A_B	die Teilmatrix der Matrix A bestehend aus den Spaltenvektoren $(a_{*,\beta_1}, \ldots, a_{*,\beta_m})$.

1.2 Rechnerarithmetik

Da auf einer Rechenanlage nur endlich viele Zahlen darstellbar sind, muß die Menge der reellen Zahlen auf eine Teilmenge, die sogenannten *Gleitpunktzahlen* 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 *Gleitpunktoperationen* 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 [39], [40], [41] und [42] angegeben. In diesem Abschnitt werden kurz die wichtigsten Begriffe und Schreibweisen aus diesem Bereich erläutert.

Es bezeichnen stets

R die Menge der Maschinenzahlen,
VR die Menge der n-dimensionalen Vektoren über R und
MR die Menge der $(m \times n)$-Matrizen über R.

Die reellen Räume werden so auf die Maschinenräume abgebildet, daß gilt

$$\begin{array}{rcl} I\!R & \supset & R \\ VI\!R & \supset & VR \\ MI\!R & \supset & MR \end{array}$$

Abbildung 1.1: Die Räume des numerischen Rechnens (Teil 1)

Im folgenden sei S ein Raum der linken Spalte und T der zugehörige Raum der rechten Spalte aus Abbildung 1.1.

Definition 1.2.1 Die Abbildung $\square : S \to T$ mit den Eigenschaften

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

(R2) $a \leq b \Rightarrow \square a \leq \square b$ 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) $\square a \leq a$ $(\square a \geq a)$ 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 1.2.2 Eine antisymmetrische Rundung $\square : S \to T$ heißt *Semimorphismus*, wenn alle inneren und äußeren Verknüpfungen in T durch

(RG) $a \circledcirc b := \square(a \circ b)$ für alle $a, b \in T$ 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 $a \boxdot b$ in T kein weiteres Element aus T liegt (vgl. [39], [41]). Für die Produkträume ist dies komponentenweise zu verstehen.

Beispiel 1.2.1 Das Skalarprodukt zweier Gleitpunktvektoren $a, b \in VR$ 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 wird das zuvor als allgemeines Rundungssymbol gebrauchte Zeichen $\square$ für die antisymmetrische Rundung zur nächstgelegenen Maschinenzahl verwandt. Die nach unten bzw. nach oben gerichteten Rundungen zur nächstkleineren bzw. nächstgrößeren Maschinenzahl wird mit $\bigtriangledown$ bzw. $\bigtriangleup$ bezeichnet. Für sie gilt die Identität

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

Die Bezeichnung $f_\square(x)$ wird für die Maschinenauswertung einer Funktion f, d.h. einer Berechnung des Funktionswertes unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$, verwendet. Der Begriff maximale Genauigkeit wird auch im Zusammenhang mit Maschinenauswertungen $f_\square$ von Funktionen $f : I\!R \to I\!R$, die durch

$$f_\square(x) := \square(f(x))$$

ebenfalls über den Semimorphismus definiert sind, verwandt.

1.3 Intervallarithmetik

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 [2], [3] oder [47], eine Einführung z.B. in [20]. Die Behandlung der zugehörigen Maschinenintervallarithmetik wird in [39] und [41] beschrieben. Im folgenden wird eine Zusammenstellung der wichtigsten Begriffe und Eigenschaften gegeben.

Definition 1.3.1 Die Menge $[a] := [\underline{a}, \overline{a}] := \{x \in \mathbb{R} \mid \underline{a} \leq x \leq \overline{a}\}$ mit $\underline{a}, \overline{a} \in \mathbb{R}$ heißt *reelles 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 $\underline{a} = \overline{a}$.

Bezeichnet $I\mathbb{R}$ die Menge der Intervalle über $\mathbb{R}$, so kann man weiter in Analogie zu den Räumen über den reellen Vektoren $V\mathbb{R}$, sowie den reellen Matrizen $M\mathbb{R}$ die Räume $VI\mathbb{R}$ bzw. $MI\mathbb{R}$ der Vektoren bzw. der Matrizen über den reellen Intervallen einführen. Entsprechendes gilt wiederum für die Komplexifizierungen $I\mathbb{C}$, $VI\mathbb{C}$ und $MI\mathbb{C}$ dieser Räume. In der Abbildung 1.2 ist der Zusammenhang zwischen den Räumen der reellen Intervalle und denen der Maschinenintervalle noch einmal graphisch dargestellt.

$$\begin{array}{rcl} I\mathbb{R} & \supset & IR \\ VI\mathbb{R} & \supset & VIR \\ MI\mathbb{R} & \supset & MIR \end{array}$$

Abbildung 1.2: Die Räume des numerischen Rechnens (Teil 2)

Vergleiche und Teilmengenbeziehungen werden mengentheoretisch interpretiert und sind für Vektoren und Matrizen genau dann erfüllt, wenn sie für alle ihre Komponenten erfüllt sind. Für $[a], [b] \in I\mathbb{R}$ gilt

$$[a] = [b] \iff \underline{a} = \underline{b} \ \wedge \ \overline{a} = \overline{b}, \tag{1.1}$$

$$[a] \subseteq [b] \iff \underline{a} \geq \underline{b} \ \wedge \ \overline{a} \leq \overline{b}, \tag{1.2}$$

$$[a] \overset{\circ}{\subset} [b] \iff \underline{a} > \underline{b} \ \wedge \ \overline{a} < \overline{b}. \tag{1.3}$$

Die Verbandsoperationen $\cap$ und $\underline{\cup}$ für $[a], [b] \in I\mathbb{R}$ sind definiert durch

$$\begin{array}{lcll} [a] \cap [b] & := & [\max\{\underline{a}, \underline{b}\}, \min\{\overline{a}, \overline{b}\}], & \text{(Schnitt)} \\ [a] \underline{\cup} [b] & := & [\min\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}], & \text{(Intervallhülle)} \end{array}$$

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

$$[a] \circ [b] := \{a \circ b \mid a \in [a], \, b \in [b]\}, \tag{1.4}$$

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

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

erfolgen. Dabei kann die Bildung des Minimums bzw. des Maximums in der Multiplikation durch vorherige Untersuchung der Intervallgrenzen in den meisten Fällen vermieden werden.

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

$$[a] \cdot ([b] + [c]) \subseteq [a] \cdot [b] + [a] \cdot [c] \tag{1.5}$$

für Intervalle $[a],[b],[c] \in I\!I\!R$. 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 $a,b \in I\!R$ und $[a],[b],[c],[d] \in I\!I\!R$.

Für $[a] \in I\!I\!R$ kann man die folgenden Hilfsgrößen definieren:

$$\begin{aligned}
m([a]) &:= \frac{1}{2}(\underline{a}+\overline{a}) && \text{(Mittelpunkt)},\\
d([a]) &:= \overline{a}-\underline{a} && \text{(Durchmesser)},\\
|[a]| &:= \max\{|x| \mid x \in [a]\} && \text{(Betrag)}.
\end{aligned}$$

Mittelpunkt, Durchmesser und Betrag sind für Vektoren und Matrizen jeweils komponentenweise definiert. Für $[a],[b] \in I\!I\!R$ gelten die Beziehungen

$$d([a] \pm [b]) = d([a]) + d([b]), \tag{1.6}$$

$$d([a] \cdot [b]) \leq d([a]) \cdot |[b]| + d([b]) \cdot |[a]|, \tag{1.7}$$

$$|[a] + [b]| \leq |[a]| + |[b]|, \tag{1.8}$$

$$|[a] \cdot [b]| = |[a]| \cdot |[b]|. \tag{1.9}$$

Die Beweise finden sich z.B. in [2].

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

$f([x])$ für den *Wertebereich* von f, d.h. $\{f(x) \mid x \in [x]\}$, und

$f_{[]}([x])$ für die *Intervallauswertung* der *Intervallerweiterung* oder auch *Einschließungsfunktion* $f_{[]}$ von f,

wobei für $f([x])$ und $f_{[]}([x])$ stets die Beziehung

$$f([x]) \subseteq f_{[]}([x])$$

gilt. Entsprechende Bezeichnungen gelten auch für vektorwertige Funktionen. Die *natürliche Intervallerweiterung* einer Funktion f erhält man, indem man alle auftretenden Variablen durch entsprechende Intervalle und alle Operationen durch die zugehörigen Intervalloperationen ersetzt. Für Punktintervalle bzw. Punktintervallvektoren als Argumente von f gilt stets

$$f(c) = f_{[]}(c)$$

Auch für die Intervallerweiterungen gilt die Inklusionsisotonie

$$\begin{aligned} a \in [a] &\implies f(a) \in f_{[]}([a]) \\ [a] \subseteq [b] &\implies f_{[]}([a]) \subseteq f_{[]}([b]). \end{aligned}$$

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

$$[a] := [\underline{a}, \overline{a}] \subseteq [\bigtriangledown\underline{a}, \bigtriangleup\overline{a}] := \{x \in I\!R \mid \bigtriangledown\underline{a} \leq x \leq \bigtriangleup\overline{a};\ \bigtriangledown\underline{a}, \bigtriangleup\overline{a} \in R\}$$

sämtliche reellen 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.

Im folgenden sei IS ein Raum der linken Spalte und IT der zugehörige Raum der rechten Spalte aus Abbildung 1.2.

Definition 1.3.2 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. Die Rundung $\diamondsuit$ ist eindeutig bestimmt (vgl. [39], [41]).

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

$$\text{(RG)} \quad [a] \diamondsuit [b] := \diamondsuit([a] \circ [b]) \quad \begin{array}{l}\text{für alle } [a], [b] \in IT \\ \text{und } \circ \in \{+, -, \cdot, /\}\end{array}$$

definiert sind.

Die Inklusionsisotonie gilt für die Maschinenoperationen in der Form

$$\begin{array}{rcl} a \in [a] \;\wedge\; b \in [b] & \Longrightarrow & a \circ b \in [a] \circ [b] \subseteq [a] \diamondsuit [b] \\ [a] \subseteq [c] \;\wedge\; [b] \subseteq [d] & \Longrightarrow & [a] \diamondsuit [b] \subseteq [c] \diamondsuit [d] \end{array}$$

für alle $\circ \in \{+, -, \cdot, /\}$ mit $a, b \in T$ und $[a], [b], [c], [d] \in IT$.

Die Bezeichnung $f_\diamondsuit([x])$ wird für die intervallmäßige Maschinenauswertung einer Funktion f, d.h. einer Berechnung einer Obermenge des Funktionswertebereiches auf $[x]$ unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$, verwendet. Der Begriff maximale Genauigkeit wird auch im Zusammenhang mit Maschinenauswertungen $f_\diamondsuit$ von Funktionen $f : I\!I\!R \to I\!I\!R$, die durch

$$f_\diamondsuit([x]) := \diamondsuit(f([x]))$$

ebenfalls über den Semimorphismus definiert sind, verwandt.

1.4 Allgemeine Optimierungsprobleme

Ein Optimierungsproblem behandelt die Aufgabenstellung, eine Funktion $f : I\!R^n \to I\!R$ bezüglich einer Teilmenge M des $I\!R^n$ zu optimieren. Die Menge M wird durch endlich viele Nebenbedingungen in Gleichungs- oder Ungleichungsform beschrieben.

Definition 1.4.1 Gegeben seien eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ und eine Menge $M \subseteq \mathbb{R}^n$. Das *allgemeine Optimierungsproblem* stellt man in der Form

$$\min_{x \in M} f(x) \tag{1.10}$$

dar. Man sagt auch, daß die Funktion f unter der *Nebenbedingung* $x \in M$ zu minimieren ist.

Die zu minimierende Funktion f des Optimierungsproblems (1.10) heißt *Zielfunktion*. Die Menge M wird *zulässiger Bereich* genannt. Ein Punkt $x \in \mathbb{R}^n$ heißt *zulässiger Punkt* von (1.10), wenn $x \in M$ gilt. Ein Punkt $\hat{x} \in M$ mit $f(\hat{x}) = \min f(x)$ für alle $x \in M$ heißt *optimale Lösung* von (1.10).

Definition 1.4.2 Im Gebiet der Optimierungstheorie unterscheidet man zwischen *unrestringierten Optimierungsproblemen* (ohne Nebenbedingungen), bei denen der zulässige Bereich der gesamte $\mathbb{R}^n$ ist, und *restringierten Optimierungsproblemen* (mit Nebenbedingungen), deren zulässiger Bereich eine echte Teilmenge des $\mathbb{R}^n$ ist. In der Regel wird der zulässige Bereich M durch Ungleichungen der Form

$$g_i(x) \geq 0 \quad (i = 1, \ldots, m) \tag{1.11}$$

mit $g_i : \mathbb{R}^n \to \mathbb{R}$ beschrieben, d.h. $M := \{x \in \mathbb{R}^n \mid g_i(x) \geq 0, i = 1, \ldots, m\}$.

Definition 1.4.3 Das durch (1.10) beschriebene Problem heißt *nicht-lineares Optimierungsproblem*, falls mindestens eine der Funktionen aus (1.11) oder f selbst nicht-linear ist.

Definition 1.4.4 Unter einem *linearen Optimierungsproblem*, auch lineares Programm (LP) genannt, versteht man die Aufgabenstellung, eine lineare Funktion unter Berücksichtigung endlich vieler linearer Restriktionen zu optimieren. Dargestellt wird dies in der *Standardform*:

$$\begin{array}{lrcl} & \text{minimiere} \quad f(x) & = & c^T x \\ \text{(LP)} \qquad & \text{unter den Nebenbedingungen} \quad Ax & = & b \\ & \text{und} \quad x & \geq & \vec{0}, \end{array}$$

wobei $x \in \mathbb{R}^n$ der Variablenvektor der zu optimierenden *Zielfunktion* $f(x) = c^T x$ ist, $c \in \mathbb{R}^n$. Die *Nebenbedingungen* werden durch eine $(m \times n)$

Matrix A und einen m-dimensionalen Vektor b bestimmt. $x \geq \vec{0}$ heißt *Vorzeichenbedingung* oder *Nichtnegativitätsbedingung*, wobei $\vec{0}$ für den Nullvektor steht und die Relation „$\geq$“ komponentenweise gilt. Nebenbedingung und Vorzeichenbedingung werden unter dem Begriff *Restriktionen* zusammengefaßt. Bezeichnet die Menge

$$M := \{x \in I\!R^n \mid Ax = b, x \geq \vec{0}\}$$

den *zulässigen Bereich*, dann läßt sich (LP) alternativ als

$$\min_{x \in M} c^T x$$

schreiben (vgl. (1.10)).

Es gilt allgemein für ein *korrekt gestelltes* (LP), daß A vollen Rang hat und daß

$$m < n$$

ist, d.h. die Zeilenzahl der Matrix A ist kleiner als die Spaltenzahl. Aufgrund der Theorie der linearen Gleichungssysteme müßte nämlich im Fall $m \geq n$ (mindestens) eine der folgenden Aussagen zutreffen:

- Durch $Ax = b$ ist x eindeutig bestimmt und die Lösung des Gleichungssystems ist dann gleichzeitig die Lösung des Optimierungsproblems (LP), falls $x \geq \vec{0}$ gilt. Sonst hat (LP) keine Lösung.
- Die Gleichung $Ax = b$ ist unverträglich, d.h. (LP) hat keine Lösung.
- Einige Gleichungen sind linear abhängig und damit redundant.

Das Gleichungssystem der Nebenbedingungen $Ax = b$ eines (LP) ist somit unterbestimmt. Gerade dies eröffnet einen großen und eben auch interessanten Spielraum für die Wahl des Lösungsvektors x, der die Zielfunktion optimieren soll.

Optimierungsprobleme können in unterschiedlicher Form vorliegen, es ist jedoch stets möglich, sie in die Standardform (LP) zu überführen. Folgende Abweichungen sind möglich:

- Gegeben ist ein lineares Optimierungsproblem in seiner *Grundform* (GrundLP)

$$\begin{array}{lrcl} & \text{minimiere} \quad f(x) & = & c^T x \\ \text{(GrundLP)} \qquad & \text{unter den Nebenbedingungen} \quad Ax & \leq & b \\ & \text{und} \quad x & \geq & \vec{0}, \end{array}$$

wobei die Nebenbedingungen in Ungleichungsform vorliegen.

Durch die Einführung von m zusätzlichen nicht-negativen Variablen $s_1, \ldots, s_m$, den sogenannten *Schlupfvariablen*, wird das Optimierungsproblem erweitert. Für jedes i wird die Ungleichung $A_{i*} \cdot x \leq b_i$ durch die Schlupfvariable s_i in eine Gleichungsnebenbedingung $A_{i*} \cdot x + s_i = b_i$ umgewandelt. Wird gleichzeitig die Zielfunktion $f(x) = c^T x$ nicht verändert, d.h. die Schlupfvariablen tragen nichts zur Zielfunktion bei, so kann das (GrundLP) als ein äquivalentes Optimierungsproblem in Standardform formuliert werden:

$$\begin{array}{lrcl} & \text{minimiere} & f(\tilde{x}) & = & \tilde{c}^T \tilde{x} \\ \text{(LP)} & \text{unter den Nebenbedingungen} & \tilde{A}\tilde{x} & = & b \\ & \text{und} & \tilde{x} & \geq & \vec{0}, \end{array}$$

mit $\tilde{x} = (x_1, \ldots, x_n, s_1, \ldots s_m)$, $\tilde{c} = (c_1, \ldots, c_n, 0, \ldots, 0)$ und $\tilde{A} = (A, I_m)$, wobei I_m die m-dimensionale Einheitsmatrix bezeichne.

- Gesucht ist das Maximum einer Zielfunktion $f(x)$. Durch einen Übergang zu $-f(x)$ wird die Standardform erreicht.

- Die Vorzeichenbedingungen $x \geq \vec{0}$ können wegfallen oder schon in den Nebenbedingungen enthalten sein.

- Liegt eine Ungleichung der Form $g_i(x) \leq 0$ vor, dann läßt sich dies offenbar durch Multiplikation mit -1 auf die Standardform $-g_i(x) \geq 0$ zurückführen.

- Eine Gleichung $g_i(x) = 0$ ist gleichbedeutend mit den zwei Ungleichungen $g_i(x) \geq 0$ und $-g_i(x) \geq 0$.

Diese Standardform dient einer kurzen und einheitlichen Formulierung. Bei der Verwendung üblicher Näherungsverfahren ist es aber nicht sinnvoll, vor Anwendung eines Lösungsverfahrens Gleichungen in Ungleichungen umzuwandeln, da nach [22] Gleichungsrestriktionen numerisch meistens einfacher zu behandeln sind als Ungleichungen. Beim Einsatz von Intervallmethoden wie sie in späteren Kapiteln angegeben werden, erweist es sich allerdings als sinnvoll und z.T. sogar notwendig, Nebenbedingungen, die in Gleichungsform vorliegen, in Ungleichungen umzuwandeln, um beispielsweise auch auf dem Rechner zulässige Punkte eines (LP) bestimmen zu können.

1.5 Optimalitätsbedingungen

Es wird zunächst ganz allgemein die Minimierung einer zweimal stetig differenzierbaren Funktion $f : \mathbb{R}^n \to \mathbb{R}$ über einer Teilmenge $M \subset \mathbb{R}^n$ betrachtet. Man unterscheidet hierbei zwischen notwendigen und hinreichenden Optimalitätsbedingungen für die Existenz eines lokalen Minimums.

Im weiteren Verlauf dieses Abschnitts werden speziell konvexe allgemeine Optimierungsprobleme betrachtet, die die Eigenschaft haben, daß jedes lokale Minimum auch ein globales Minimum ist. Desweiteren wird der Satz von Kuhn und Tucker angegeben, der eine wichtige Bedeutung für die konvexe Programmierung erlangt hat.

1.5.1 Notwendige und hinreichende Optimalitätsbedingungen

Definition 1.5.1 Sei $M \subset \mathbb{R}^n$ eine offene Menge und $f : M \to \mathbb{R}$ eine Funktion. Ein Punkt $\hat{x} \in M$ heißt *lokaler Minimalpunkt* (bzw. *lokaler Maximalpunkt*) von f, falls eine Umgebung $U \subset M$ von $\hat{x}$ existiert, so daß

$$f(\hat{x}) \leq f(x) \quad (\text{bzw. } f(\hat{x}) \geq f(x))$$

für alle $x \in U$.

Ein Punkt $\hat{x} \in M$ heißt *globaler Minimalpunkt* (bzw. *globaler Maximalpunkt*) von f, falls

$$f(\hat{x}) \leq f(x) \quad (\text{bzw. } f(\hat{x}) \geq f(x))$$

für alle $x \in M$.

Will man zur Lösung allgemeiner Optimierungsprobleme sukzessive in Richtung abnehmender Zielfunktionswerte fortschreiten (Variation entlang einer Strecke), um Minimalpunkte zu finden, so ist man natürlich nur an Richtungen interessiert, längs derer man wenigstens ein Stück vorangehen kann, ohne den zulässigen Bereich zu verlassen. Dies führt auf den Begriff einer zulässigen Richtung.

Definition 1.5.2 Sei $x \in M$ ein zulässiger Punkt. Ein Vektor $\Delta x \in \mathbb{R}^n$ heißt *zulässige Richtung* in x, falls es eine reelle Zahl $\mathcal{T} > 0$ gibt, so daß für alle τ mit $0 \leq \tau \leq \mathcal{T}$ der Punkt $x + \tau\,\Delta x$ zulässig ist, d.h. $x + \tau\,\Delta x \in M$. Die *Menge aller zulässigen Richtungen in* $x \in M$ bezeichnet man mit $Z(x)$.

Satz 1.5.1 (Notwendige Optimalitätsbedingung) *: Sei $M \subseteq \mathbb{R}^n$ und $f : M \to \mathbb{R}$ stetig partiell differenzierbar. Ist $\hat{x} \in M$ lokaler Minimalpunkt von f, dann gilt für alle zulässigen Richtungen $\Delta\hat{x} \in Z(\hat{x})$*

$$\Delta\hat{x}^T \cdot \nabla f(\hat{x}) \geq 0. \tag{1.12}$$

Beweis: Für jedes τ mit $0 \leq \tau \leq \mathcal{T}$ ist der Punkt $x(\tau) = \hat{x} + \tau\Delta\hat{x} \in M$. Sei für $0 \leq \tau \leq \mathcal{T}$ die Funktion $g(\tau) = f(x(\tau))$ definiert, dann hat g ein lokales Minimum in $\tau = 0$, da $\hat{x}$ lokaler Minimalpunkt von f ist. Die Zwischenwertform liefert

$$g(\tau) - g(0) = g'(0) \cdot \tau + \mathbf{o}(\tau). \tag{1.13}$$

Wäre $g'(0) < 0$, dann wäre für einen hinreichend kleinen Wert $\tau > 0$ die rechte Seite von (1.13) negativ und somit $g(\tau) - g(0) < 0$. Dies widerspräche der Aussage, daß g ein lokales Minimum in $\tau = 0$ besitzt. Es folgt daher, daß $g'(0) = \Delta\hat{x}^T \cdot \nabla f(\hat{x}) \geq 0$. □

Satz 1.5.1 besagt, daß es keine zulässige Richtung in $\hat{x}$ gibt, in der der Zielfunktionswert $f(x)$ verkleinert werden kann. Ein Punkt $\hat{x} \in M$, der die Bedingung (1.12) erfüllt, heißt *stationärer Punkt* von f auf M.

Für einen inneren Punkt $\dot{x} \in M$, für den es eine ε–Umgebung gibt, die vollständig in M liegt, ist $Z(\dot{x}) = \mathbb{R}^n$. (1.12) ist dann nur zu erfüllen, wenn $\nabla f(\dot{x}) = 0$ ist. Damit folgt unmittelbar aus Satz 1.5.1:

Satz 1.5.2 (Notwendige Optimalitätsbedingung erster Ordnung) *Sei $M \subseteq \mathbb{R}^n$ offen und $f : M \to \mathbb{R}$ stetig partiell differenzierbar. Ist $\hat{x}$ innerer Punkt von M sowie lokaler Minimalpunkt von f auf M. Dann gilt*

$$\nabla f(\hat{x}) = 0.$$

Dieser Satz ist insbesondere für allgemeine Optimierungsprobleme ohne Nebenbedingungen von Bedeutung, da in diesem Fall $M = \mathbb{R}^n$ ist und jeder Punkt aus M somit innerer Punkt von M ist.

Satz 1.5.3 (Hinreichende Optimalitätsbedingung) *Es sei $M \subseteq \mathbb{R}^n$ und $f : M \to \mathbb{R}$ zweimal stetig differenzierbar. $\hat{x}$ sei ein innerer Punkt von M. Ist $\nabla f(\hat{x}) = 0$ und $\nabla^2 f(\hat{x})$ positiv definit, dann ist $\hat{x}$ lokaler Minimalpunkt von f auf M.*

Ist $\nabla f(\hat{x}) = 0$ sowie $\nabla^2 f(\hat{x})$ nur positiv semidefinit, so stellt dies lediglich die notwendige Optimalitätsbedingung zweiter Ordnung dafür dar, daß ein innerer Punkt $\hat{x}$ von M lokaler Minimalpunkt von f auf M ist. Zum Beweis von Satz 1.5.3 und der letzten Aussage vergleiche etwa [22].

1.5.2 Konvexe Optimierungsprobleme

Die Idee der Variation entlang einer Strecke, die es Euler erlaubte, „Curven zu finden, denen eine Eigenschaft im höchsten oder geringsten Grade zukommt“ (siehe [13]), führte zu einer abstrakten notwendigen Bedingung für Minimallösungen einer Funktion, die auf einem Vektorraum definiert ist. Der erst Anfang des 20. Jahrhunderts gefundene Begriff einer konvexen Funktion sorgt zusammen mit der Idee der Variation für einen einfachen und eleganten Zugang zur Optimierungstheorie.

Es ist bei der Lösung allgemeiner Optimierungsprobleme sehr nützlich, wenn jeder lokale Minimalpunkt der Zielfunktion auch globaler Minimalpunkt ist. Die wichtigste Funktionenklasse, die diese Anforderung erfüllt, ist die Klasse der konvexen Funktionen, die auch bei der Formulierung linearer Optimierungsprobleme auftritt.

Definition 1.5.3 Eine nicht-leere Teilmenge $K \subseteq \mathbb{R}^n$ heißt *konvex*, wenn für alle τ mit $0 \leq \tau \leq 1$ und für alle $x_1, x_2 \in K$ gilt

$$\tau\, x_1 + (1-\tau)x_2 \in K.$$

Definition 1.5.4 K sei eine konvexe Teilmenge des $\mathbb{R}^n$. Eine auf K definierte Funktion $f : K \to \mathbb{R}$ heißt *konvex*, wenn für je zwei Punkte $x_1, x_2 \in K$ und alle τ mit $0 \leq \tau \leq 1$ gilt

$$f(\tau\, x_1 + (1-\tau)x_2) \;\leq\; \tau\, f(x_1) + (1-\tau) f(x_2). \tag{1.14}$$

f heißt *konkav*, wenn gilt

$$f(\tau\, x_1 + (1-\tau)x_2) \;\geq\; \tau\, f(x_1) + (1-\tau) f(x_2). \tag{1.15}$$

Definition 1.5.5 $f : K \to \mathbb{R}$ heißt *streng konvex* (bzw. *streng konkav*), wenn in (1.14) (bzw. (1.15)) für $x_1 \neq x_2$ echte Ungleichungen stehen.

Korollar 1.5.4 *Eine lineare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ ist auf jeder konvexen Teilmenge des $\mathbb{R}^n$ sowohl konvex als auch konkav.*

Aus der Definition über konvexe und konkave Funktionen folgt unmittelbar, daß eine Funktion $f : K \to \mathbb{R}$ konvex ist, wenn die Funktion $-f$ konkav ist und umgekehrt. Die Maximierung einer konkaven Funktion kann deshalb stets auf die Minimierung einer konvexen Funktion zurückgeführt werden.

Es werden nun einige Sätze über konvexe Mengen und Funktionen angegeben, deren Beweise sich unmittelbar aus den Definitionen dieser Begriffe ergeben.

Satz 1.5.5 *Seien $K \subseteq \mathbb{R}^n$ eine konvexe Menge und $f_1, \ldots, f_m : K \to \mathbb{R}$ konvexe Funktionen. Dann ist auch jede nicht-negative Linearkombination der Funktionen $f_1, \ldots, f_m$*

$$f := \sum_{i=1}^{m} \tau_i f_i \quad mit\ \tau_i \geq 0 \quad (i = 1, \ldots, m)$$

eine konvexe Funktion.

Satz 1.5.6 *Seien $K \subseteq \mathbb{R}^n$ eine konvexe Menge und $f : K \to \mathbb{R}$ eine konvexe Funktion sowie $a \in \mathbb{R}$. Dann sind die abgeschlossene Menge $\{x \in K \,|\, f(x) \leq a\}$ und die offene Menge $\{x \in K \,|\, f(x) < a\}$ konvex.*

Satz 1.5.7 *Sind $K_1, \ldots, K_m \subseteq \mathbb{R}^n$ konvexe Mengen, so ist auch deren Durchschnitt $K_1 \cap K_2 \cap \ldots \cap K_m$ konvex.*

Aus den Sätzen 1.5.6 und 1.5.7 folgt, daß der zulässige Bereich M, gegeben durch die Ungleichungen $g_i(x) \leq 0$ $(i = 1, \ldots, m)$, konvex ist, wenn die Funktionen $g_1, \ldots, g_m$ konvex sind.

Definition 1.5.6 Das in (1.10) beschriebene allgemeine Optimierungsproblem heißt *konvex*, wenn der zulässige Bereich M und die Zielfunktion f konvex sind.

Da, wie schon eingangs erwähnt, die meisten Restriktionen von allgemeinen Optimierungsproblemen durch Gleichungen und Ungleichungen beschrieben werden, ist es nun wichtig festzustellen, wann gewisse Funktionen konvex sind. Es werden nun zwei wichtige Kriterien für die Konvexität einer Funktion angegeben, zum Beweis siehe etwa [17].

Satz 1.5.8 (Subgradientenungleichung) *Sei $K \subseteq \mathbb{R}^n$ eine konvexe Menge und $f : K \to \mathbb{R}$ einmal stetig differenzierbar. f ist auf K genau dann konvex, wenn für alle $x_1, x_2 \in K$ gilt*

$$f(x_2) - f(x_1) \geq (x_2 - x_1)^T \nabla f(x_1).$$

Satz 1.5.9 *Sei $K \subseteq \mathbb{R}^n$ eine konvexe Menge, die innere Punkte enthalte, und sei $f : K \to \mathbb{R}$ zweimal stetig differenzierbar. f ist auf K genau dann konvex (streng konvex), wenn die Hesse-Matrix $\nabla^2 f(x)$ für alle $x \in K$ positiv semidefinit (positiv definit) ist.*

Man kann nun den folgenden grundlegenden Satz für konvexe allgemeine Optimierungsprobleme angeben:

Satz 1.5.10 *Sei $K \subseteq \mathbb{R}^n$ eine konvexe Menge und $f : K \to \mathbb{R}$ konvex. Dann gilt:*
(a) Die Menge $\widehat{K}$ aller globalen Minimalpunkte von f auf K ist konvex.
(b) Jeder lokale Minimalpunkt von f auf K ist auch globaler Minimalpunkt.

Zum Beweis siehe Neumann [50].

Bemerkung: Ist die Funktion $f : K \to \mathbb{R}$ in Satz 1.5.10 streng konvex, so wird das globale Minimum von f auf K in höchstens einem Punkt $\hat{x} \in K$ angenommen.

Satz 1.5.11 (Notwendige und hinreichende Optimalitätsbedingung für konvexe Funktionen) *Sei $K \subseteq \mathbb{R}^n$ eine konvexe Menge, $f : K \to \mathbb{R}$ eine konvexe, stetig differenzierbare Funktion und $\hat{x} \in K$. Gilt*

$$\Delta\hat{x}^T \nabla f(\hat{x}) \geq 0 \tag{1.16}$$

für alle zulässigen Richtungen $\Delta\hat{x} \in Z(\hat{x})$, dann ist $\hat{x}$ globaler Minimalpunkt von f auf K.

Beweis: Sei $\hat{x}$ ein Minimalpunkt von f auf K. Für alle $\Delta\hat{x} \in Z(\hat{x})$ und $0 < \tau \leq 1$ ist $\hat{x} + \tau\,\Delta\hat{x} \in Z(\hat{x})$ und somit

$$\frac{f(\hat{x} + \tau\,\Delta\hat{x}) - f(\hat{x})}{\tau} \geq 0$$

Der Grenzübergang mit $\tau \longrightarrow 0$ ist wegen der Monotonie des Differenzenquotienten konvexer Funktionen erlaubt und liefert (1.16). Andererseits folgt aus (1.16) mit Satz 1.5.8 für alle $\Delta\hat{x} \in Z(\hat{x})$

$$f(\hat{x} + \Delta\hat{x}) - f(\hat{x}) \geq \Delta\hat{x}^T \nabla f(\hat{x}) \geq 0,$$

d.h. $\hat{x}$ ist Minimalpunkt von f auf K. □

Der Satz 1.5.11 besagt anschaulich, daß ein Punkt $\hat{x} \in K$ genau dann ein globaler Minimalpunkt von f auf K ist, wenn es keine zulässige Richtung in $\hat{x}$ gibt, in der der Zielfunktionswert $f(x)$ verkleinert werden kann. Für allgemeine restringierte Optimierungsprobleme ist die Optimalitätsbedingung (1.16) sehr unhandlich, da sich die Menge der zulässigen Richtungen $Z(\hat{x})$ im allgemeinen nicht in einfacher Weise durch die Nebenbedingungen ausdrücken läßt (siehe etwa [22]).

Korollar 1.5.12 *Für ein unrestringiertes allgemeines Optimierungsproblem vereinfacht sich die Optimalitätsbedingung (1.16) zu* $\nabla f(\hat{x}) = 0$.

Dies ist eine Motivation für die Idee der Innere-Punkt-Methoden, die Lösung eines restringierten allgemeinen Optimierungsproblems durch eine Folge von Lösungen unrestringierter Optimierungsprobleme (vgl. Definition 1.4.2) zu approximieren.

1.6 Barriere-Verfahren

Die ursprüngliche Motivation (um 1960) für ein Strafkostenverfahren wie das Barriere- oder Penalty-Verfahren war die Tatsache, daß keine numerisch effektiven Algorithmen zur Lösung restringierter Optimierungsprobleme der Form (1.10) (vgl. Definition 1.4.2) verfügbar waren, während für unrestringierte Probleme gute Techniken entwickelt waren. Es ist auch heute noch ein wesentlicher Vorteil der Strafkostenverfahren, daß sie restringierte Optimierungsprobleme mit Hilfe von unrestringierten Problemen lösen, die im allgemeinen wesentlich einfacher zu behandeln sind.

Dabei verfolgen die Strafkostenverfahren zwei oftmals in Konflikt zueinander stehende Ziele, nämlich die Zielfunktion zu minimieren und die Restriktionen zu erfüllen.

Man betrachte das restringierte allgemeine Optimierungsproblem (1.10)

$$\min_{x \in M} f(x).$$

Trivialerweise kann man (1.10) mit Hilfe einer Indikatorfunktion

$$\delta_M(x) := \begin{cases} 0 & \text{für } x \in M \\ \infty & \text{für } x \notin M \end{cases} \tag{1.17}$$

von M als unrestringiertes Optimierungsproblem formulieren, denn

$$\min_{x \in M} f(x) \Leftrightarrow \min_{x \in \mathbb{R}^n} \{f(x) + \delta_M(x)\}.$$

Durch die Addition des Terms (1.17) zur Zielfunktion $f(x)$ wird das Verlassen des zulässigen Bereiches M mit einer „unendlichen“ Strafe belegt.

Numerisch ist $f(x)+\delta_M(x)$ wegen der Unstetigkeit von $\delta_M(x)$ am Rande von M natürlich nicht zu gebrauchen, auch dann nicht, wenn man ∞ in (1.17) durch eine große positive Zahl ersetzt.

Man versucht daher bei den Strafkostenverfahren $\delta_M(x)$ mit Hilfe einer mindestens stetigen, möglichst auch differenzierbaren Funktion zu approximieren. Für die Wahl dieser Funktionen werden in der Literatur eine Vielzahl an Vorschlägen gemacht (vgl. [8], [15]).

Ziel eines Barriere-Verfahrens ist die Bestimmung der optimalen Lösung eines restringierten Optimierungsproblems. Das Verfahren startet an einem inneren Punkt des zulässigen Bereichs M und bewegt sich mit Hilfe eines Gradienten-Verfahrens in Richtung der optimalen Lösung. Hierbei wird das Verlassen des zulässigen Bereiches M durch eine geeignete Straffunktion oder einem sogenannten Barriere-Term verhindert. Eine wichtige Voraussetzung für den Einsatz von Barriere-Methoden ist deshalb, daß der zulässige Bereich M, der im allgemeinen durch Ungleichungen (1.11)

$$h_i(x) \geq 0 \quad (i = 1, \ldots, m)$$

gegeben ist, innere Punkte besitzt. Im weiteren wird nun der zulässige Bereich M als konvex vorausgesetzt.

Definition 1.6.1 Mit

$$\overset{\circ}{M} := \{x \in I\!R^n \,|\, h_i(x) > 0 \quad (i = 1, \ldots, m)\}$$

wird die nicht-leere *Menge der inneren Punkte* von M bezeichnet. Die Punkte $x \in \overset{\circ}{M}$ heißen *strikt zulässig*.

Definition 1.6.2 Eine Funktion $b : \overset{\circ}{M} \to I\!R$ heißt *Barriere-Term* bezüglich M, falls gilt

$$b(x) \text{ ist stetig in } \overset{\circ}{M}, \tag{1.18a}$$

$$b(x) \geq 0 \quad \forall x \in \overset{\circ}{M}, \tag{1.18b}$$

$$b(x) \to \infty, \text{ falls } x \text{ sich dem Rand von } M \text{ nähert.} \tag{1.18c}$$

Sind die Funktionen h_i $(i = 1, \ldots, m)$ stetig, so wird

$$b(x) := \sum_{i=1}^{m} \frac{1}{h_i(x)}$$

als der klassische *inverse Barriere-Term* bezeichnet.

Definition 1.6.3 Eine Funktion $B : \overset{\circ}{M} \times (0, \infty) \to I\!R$ mit einem *Barriere-Parameter* $\mu > 0$ definiert als

$$B(x \mid \mu) := f(x) + \mu \cdot b(x)$$

heißt *Barriere-Funktion*, falls $b(x)$ ein Barriere-Term ist.

Wählt man nun eine streng monoton fallende Nullfolge positiver Zahlen $(\mu^{(l)})_{l \in I\!N}$, so erhält man für jedes $l \in I\!N$ das aus dem ursprünglichen restringierten Optimierungsproblem (1.10) abgeleitete Problem

$$\min_{x \in I\!R^n} B(x \mid \mu^{(l)}) := f(x) + \mu^{(l)} \cdot b(x), \tag{1.19}$$

dessen Minimalstelle mit $\widehat{x}^{(l)}$ bezeichnet sei. Startet man zur Lösung von (1.19) für $l = 1$ mit einer strikt zulässigen Anfangslösung $x^{(0)} \in \overset{\circ}{M}$ und verwendet dann als Ausgangslösung für (1.19) mit $l > 1$ stets die im vorhergehenden Iterationsschritt berechnete optimale Lösung $\widehat{x}^{(l-1)}$, so verhindert die Eigenschaft (1.18c) des Barriere-Terms $b(x)$ automatisch ein Verlassen des zulässigen Bereiches M.

Man kann also zur Lösung von (1.19) ein Lösungsverfahren für unrestringierte Optimierungsprobleme verwenden und erhält somit eine Punktfolge $(\widehat{x}^{(l)})_{l \in I\!N}$, deren Häufungspunkt ein globaler Minimalpunkt des Ausgangsproblems $\min_{x \in M} f(x)$ ist. Ist das Optimierungsproblem nicht konvex, so liefert ein Lösungsverfahren für unrestringierte Minimierungsprobleme, wie schon weiter oben erwähnt, oftmals nur ein lokales Minimum. Folglich ist der Häufungspunkt der Punktfolge $(\widehat{x}^{(l)})_{l \in I\!N}$ im allgemeinen auch nur ein lokaler Minimalpunkt.

Die Eigenschaften (1.18) der Definition 1.6.2 gestatten einen Konvergenzbeweis für das Verfahren der Barriere-Methoden. Wesentlich für das Verfahren sind hierbei insbesondere die Eigenschaften (1.18a) und (1.18c). Oftmals wird die Eigenschaft (1.18b) auch etwas modifiziert angegeben. So wird etwa die Logarithmus-Funktion

$$b(x) := -\sum_{i=1}^{m} \ln(h_i(x))$$

bevorzugt als Barriere-Term benutzt, da sie aufgrund der Singularität bei Null die wesentlichen Eigenschaften (1.18a) und (1.18c) der Definition 1.6.2 erfüllt.

Auf diese Weise kann ein durch Ungleichungen beschränktes Minimierungsproblem in eine Folge von unbeschränkten Minimierungsproblemen

überführt werden, deren Minima gegen die Minimalstelle des Ausgangsproblems streben.

Strategie: Ein Optimierungsproblem mit Ungleichungen als Nebenbedingungen (inequality constrained problem ICP)

$$\text{(ICP)} \qquad \begin{array}{rl} \text{minimiere} & f(x) \\ \text{unter den Nebenbedingungen} & h_i(x) \;\geq\; 0, \quad (i = 1, \ldots, m) \end{array}$$

wird durch eine Folge von Optimierungsproblemen mit einer Barriere-Funktion als Zielfunktion

$$\min_{x \in I\!R^n} B(x \mid \mu^{(l)}) = f(x) - \mu^{(l)} \sum_{i=1}^{m} \ln(h_i(x)) \tag{1.20}$$

ersetzt, die durch positive Barriere-Parameter $\mu^{(l)}$ gewichtet werden und deren Minimum unter Verwendung eines Minimierungsverfahrens für unrestringierte Optimierungsprobleme bestimmt werden kann.

Satz 1.6.1 (Fiacco/McCormick) *Bezeichnet $\widehat{x}^{(l)}$ die Minimalstelle der Barriere-Funktion $B(x \mid \mu^{(l)})$ und $\widehat{x}$ das Minimum des (ICP), so gilt:*

$$\widehat{x}^{(l)} \longrightarrow \widehat{x} \quad \textit{für} \quad \mu^{(l)} \longrightarrow 0.$$

Zum Beweis siehe Fiacco und McCormick [14].

Zur Illustration der Barriere-Methode betrachte man folgendes

Beispiel 1.6.1

$$\begin{array}{rrcll} \text{minimiere} & f(x) & = & 2x & \\ \text{unter den Nebenbedingungen} & h_1(x) = x - 1 & \geq & 0 & \text{und} \\ & h_2(x) = 3 - x & \geq & 0 & \end{array}$$

Die entsprechend (1.20) konstruierte Barriere-Funktion lautet

$$B(x \mid \mu) = f(x) - \mu \sum_{i=1}^{2} \ln(h_i(x)) = 2x - \mu \left(\ln(x-1) + \ln(3-x) \right).$$

Für die Parameterwerte $\mu = 0.5$, 0.25 und 0.1 nähern sich die minimalen Funktionswerte und die Minimalstellen der entsprechenden Barriere-Funktionen $B(x \mid 0.5)$, $B(x \mid 0.25)$ und $B(x \mid 0.1)$ der optimalen Lösung des Ausgangsproblems $f(1) = 2$ an (siehe Abbildung 1.3).

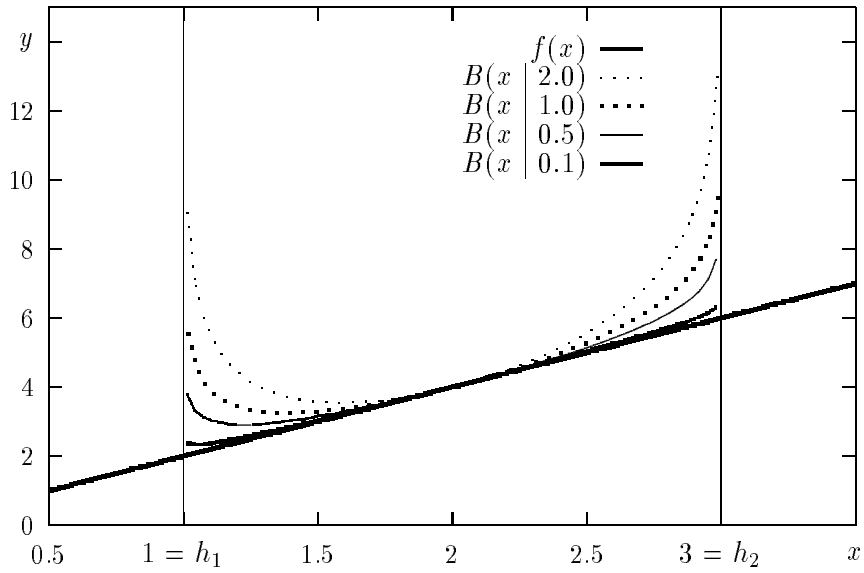

Abbildung 1.3: Die *Barriere-Funktion* für verschiedene Parameter μ und die *Zielfunktion* $f(x) = 2x$ aus Beispiel 1.6.1.

Algorithmische Beschreibung

Algorithmisch läßt sich die Barriere-Methode durch Algorithmus 1.6.1 beschreiben:

Algorithmus 1.6.1: BarriereMethod $(f, h, x^{(0)}, x^{(l)})$

1. {Initialisierung.}
   $l := -1$; $l_{\max} := 100$; $\varepsilon := 10^{-10}$; $\delta := 10^{-50}$; *wähle* $\mu^{(0)} > 0$;
2. {Prüfe, ob $x^{(0)}$ ein strikt zulässiger Punkt ist, d.h. $h_i(x^{(0)}) > 0 \; \forall \; i$.}
   CheckFeasibility $(h, x^{(0)})$;
3. **repeat**
   (a) $l := l + 1$;
   (b) **if** $l > 0$ **then** wähle $0 < \mu^{(l)} < \mu^{(l-1)}$;
   (c) {Bilde die (neue) Barriere-Funktion.}
   $$B(x \mid \mu^{(l)}) := f(x) - \mu^{(l)} \sum_{i=1}^{m} \ln(h_i(x));$$
   (d) {Bestimme $\hat{x}^{(l)}$, das Minimum von $B(x \mid \mu^{(l)})$.}
   $\hat{x}^{(l)} := \min\limits_{x \in \mathbb{R}^n} B(x \mid \mu^{(l)})$;

until $(\|\widehat{x}^{(l)} - \widehat{x}^{(l-1)}\| < \varepsilon)$ **or** $(\mu^{(l)} < \delta)$ **or** $(l = l_{\max})$;

4. **return** $\widehat{x}^{(l)}$;

Bemerkung: Zur Bestimmung des Minimums $\widehat{x}^{(l)}$ von $B(x \mid \mu^{(l)})$ in Schritt 3d muß die Nullstelle des Gradienten von $B(x \mid \mu^{(l)})$ ermittelt werden.

$$\nabla_x B(\widehat{x}^{(l)} \mid \mu^{(l)}) = \nabla f(\widehat{x}^{(l)}) - \mu^{(l)} \sum_{i=1}^{n} \frac{1}{h_i(\widehat{x}^{(l)})} \nabla h_i = 0$$

Um die Barriere-Methode anwenden zu können, ist es notwendig, eine zulässige Anfangslösung $x^{(0)} \in \overset{\circ}{M}$ zu finden. Dies kann aber bei manchen Optimierungsproblemen zu einem gesonderten Problem führen.

1.7 Methode der Lagrangeschen Multiplikatoren

Die folgende Idee von J. L. Lagrange (1788) hat eine fundamentale Bedeutung in der Optimierungstheorie erlangt [34].

Sei U eine beliebige Menge und seien $f, g : U \to \mathbb{R}$ beliebige Funktionen. Die Suche nach einer Minimallösung von f auf U unter der Nebenbedingung $x \in M$ mit $M := \{x \in U \mid g(x) = 0\}$ kann durch folgenden Vorgang ersetzt werden:

Man finde ein $\lambda \in \mathbb{R}$ derart, daß das Minimum $\widehat{x} \in U$ der unrestringierten Funktion $f(x) - \lambda\, g(x)$ in M liegt, d.h. daß $\widehat{x}$ die Gleichung $g(\widehat{x}) = 0$ erfüllt. Offenbar gilt dann für alle $x \in M$ die Ungleichung

$$f(\widehat{x}) = f(\widehat{x}) - \lambda\, g(\widehat{x}) \leq f(x) - \lambda\, g(x) = f(x).$$

Dieser Ansatz läßt sich unmittelbar auf mehrere Nebenbedingungen übertragen und führt zu folgender hinreichenden Bedingung für Lösungen restringierter Optimierungsprobleme (vgl. Definition 1.4.2).

Lemma 1.7.1 (Lagrangesches Lemma) *Sei $f : U \to \mathbb{R}$, $g = (g_1, \ldots, g_m) : U \to \mathbb{R}^m$ und $M = \{x \in U \mid g(x) = 0\}$ ein Optimierungsproblem mit Gleichungen als Nebenbedingungen (equality constrained problem ECP)*

$$(ECP) \qquad \begin{array}{rll} \textit{minimiere} & f(x) & \\ \textit{unter den Nebenbedingungen} & g_j(x) = 0, & (j = 1, \ldots, m) \end{array}$$

Sei der Lagrangesche Multiplikator $\lambda := (\lambda_1, \ldots, \lambda_m)^T \in \mathbb{R}^m$ *ein* m*-dimensionaler Vektor derart, daß ein* $\hat{x} \in M$ *eine Minimallösung der Lagrangeschen Funktion*

$$L(x, \lambda) := f(x) - \sum_{j=1}^{m} \lambda_j g_j(x)$$

auf U *ist.*

Dann ist $\hat{x}$ *eine Minimallösung von* f *auf* M*.*

Beweis: Für $x \in M$ gilt:

$$f(\hat{x}) = f(\hat{x}) - \lambda^T g(\hat{x}) \leq f(x) - \lambda^T g(x) = f(x).$$

□

Das Lagrangesche Lemma liefert eine allgemeine hinreichende Bedingung für Minimallösungen restringierter Aufgaben, die auch in Funktionenräumen verwendbar ist. Lagrange hat seine Methode bereits bei Variationsaufgaben benutzt. Bei der Verwendung des Lagrangeschen Lemmas im $\mathbb{R}^m$ beachte man die folgende

Bemerkung: Sei U eine Teilmenge des $\mathbb{R}^n$ und seien $f : U \to \mathbb{R}$, $g = (g_1, \ldots, g_m) : U \to \mathbb{R}^m$ differenzierbar. Nach dem Lagrangeschen Lemma gilt es, ein $\lambda \in \mathbb{R}^m$ so zu finden, daß für einen Punkt $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ gilt:

(i) x erfüllt die geforderten Nebenbedingungen, d.h.

$$g_j(x) = 0 \quad \text{für} \quad j \in \{1, \ldots, m\}. \tag{1.21}$$

(ii) x ist eine globale Minimallösung der Funktion

$$L(x, \lambda) = f(x) - \sum_{j=1}^{m} \lambda_j g_j(x).$$

Eine notwendige Bedingung für das Erfüllen der Bedingung (ii) ist nach Satz 1.5.2 $\nabla L(x, \lambda) = 0$. Das führt zu den zusätzlichen n Gleichungen

$$\frac{\partial f}{\partial x_i}(x) - \sum_{j=1}^{m} \lambda_j \frac{\partial g}{\partial x_i}(x) = 0 \quad \text{für} \quad i \in \{1, \ldots, n\}. \tag{1.22}$$

Mit (1.21) und (1.22) bekommt man ein System von $(n+m)$ Gleichungen (i.a. nicht-linear) für die $(n+m)$ Unbekannten $(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m)$. Aber durch das Auffinden einer Lösung $(\widehat{x}_1, \ldots, \widehat{x}_n, \widehat{\lambda}_1, \ldots, \widehat{\lambda}_m)$ von (1.21) und (1.22) erhält man im Falle $\widehat{x} \in U$ nur einen Kandidaten für die Minimallösung von f auf $M := \{x \in U \mid g(x) = 0\}$.

Es bleibt noch zu prüfen, ob $(\widehat{x}_1, \ldots, \widehat{x}_n)$ eine Minimallösung der Funktion $L(x, \widehat{\lambda}) := f(x) - \sum_{j=1}^{m} \widehat{\lambda}_j g_j(x)$ auf $\mathbb{R}^n$ ist.

Ist die zu dem berechneten $\widehat{\lambda}$ gehörige Funktion $L(x, \widehat{\lambda})$ konvex, dann ist (siehe Korollar 1.5.12) $\widehat{x}$ eine Minimallösung von $L(x, \widehat{\lambda})$ und mit dem Lagrangeschen Lemma auch eine Minimallösung von f auf M. Bei zweimal stetig differenzierbaren Funktionen garantiert die positive Definitheit von $\nabla^2 L(\widehat{x}, \widehat{\lambda})$, daß $\widehat{x}$ eine globale Minimalstelle von f auf M ist (siehe Satz 1.5.11).

Satz 1.7.2 (Lagrangesche Multiplikatorenregel) *Sei $U \subset \mathbb{R}^n$ eine offene Umgebung von $\widehat{x}$ und seien $f : U \to \mathbb{R}$ und $g = (g_1, \ldots, g_m) : U \to \mathbb{R}^m$ aus (ECP) stetig differenzierbare Funktionen, wobei $m < n$ ist. Hat die Funktion f auf M an der Stelle $\widehat{x}$ ein lokales Extremum und hat die Jacobi-Matrix $\boldsymbol{J}_g(\widehat{x})$ vollen Rang, so existiert ein $\widehat{\lambda}$ derart, daß gilt*

$$\nabla L(\widehat{x}, \widehat{\lambda}) = \begin{pmatrix} \nabla_x L(\widehat{x}, \widehat{\lambda}) \\ \nabla_\lambda L(\widehat{x}, \widehat{\lambda}) \end{pmatrix} = \begin{pmatrix} \nabla f(\widehat{x}) - \left(\boldsymbol{J}_g(\widehat{x}) \right)^T \widehat{\lambda} \\ g(\widehat{x}) \end{pmatrix} = 0$$

Zum Beweis siehe Walter [60].

Lineare Nebenbedingungen

Definition 1.7.1 Der hier zu behandelnde Spezialfall eines Optimierungsproblems mit linearen Nebenbedingungen in Gleichungsform (linear equality constrained problem LECP) ist definiert als

$$\text{(LECP)} \qquad \begin{array}{lll} & \text{minimiere} & f(x) \\ \text{unter den Nebenbedingungen} & g(x) = Ax - b & = \quad 0, \end{array}$$

wobei A eine $(m \times n)$-Matrix mit vollem Rang ist. Zu beachten ist, daß die Matrix A die Jacobi-Matrix $\boldsymbol{J}_g(x)$ der linearen Nebenbedingungen $g(x) = Ax - b = 0$ ist.

$N \in \mathbb{R}^{(n-m)\times n}$ bezeichne eine Matrix, deren Spalten eine Basis des Nullraumes von A bilden, d.h. eine Basis des Unterraumes für dessen Vektoren p gilt: $Ap = 0$.

Lemma 1.7.3 *Notwendige Bedingungen dafür, daß $\widehat{x}$ eine lokale Lösung von (LECP) ist, sind*

$$A\widehat{x} - b = 0 \qquad \textit{(Zulässigkeit)} \tag{1.23a}$$

$$\nabla f(\widehat{x}) - A^T\widehat{\lambda} = 0 \quad \textit{für ein bel. } \widehat{\lambda} \qquad \textit{(Optimalität 1. Ordnung)} \tag{1.23b}$$

$$N^T \cdot \nabla^2 f(\widehat{x}) \cdot N \geq 0 \qquad \textit{(Optimalität 2. Ordnung)} \tag{1.23c}$$

Hinreichende Optimalitätsbedingung für $\widehat{x}$ ist, daß (1.23a) und (1.23b) gelten und daß $N^T \cdot \nabla^2 f(\widehat{x}) \cdot N$ positiv definit ist.

Beweis: Die Aussage des Lemmas folgt direkt aus der Anwendung des Satzes 1.7.2 auf das (LECP). □

Die Optimalitätsbedingung 1. Ordnung besagt, daß der Gradient von f an einem optimalen Punkt als Linearkombination der Spalten von A^T dargestellt werden kann, d.h. $\nabla f(\widehat{x})$ liegt im Bildraum von A^T. Der *Lagrangesche Multiplikator* $\widehat{\lambda}$ definiert die Koeffizienten dieser Linearkombination und ist eindeutig, falls A vollen Rang besitzt (was bei der allgemeinen Definition eines LP vorausgesetzt wird).

Die *Lagrangesche Funktion* für das (LECP) lautet

$$L(x, \lambda) = f(x) - \lambda^T \cdot (Ax - b),$$

wobei der *Lagrangesche Multiplikator* λ ein m-dimensionaler Vektor ist.

Die Optimalitätsbedingung $\nabla f(\widehat{x}) - A^T\widehat{\lambda} = 0$ (1.23b) läßt sich in der Form interpretieren, daß der Gradient der Lagrangeschen Funktion bezüglich x verschwindet, wenn $\lambda = \widehat{\lambda}$. Diese Bedingung ist außerdem äquivalent zu der Aussage $N^T \nabla f(\widehat{x}) = 0$, d.h. die Projektion des Gradienten $\nabla f(\widehat{x})$ in den Nullraum von A verschwindet. Die Optimalitätsbedingung 1. Ordnung (1.23b) ist daher analog zu der Bedingung $\nabla f(\widehat{x}) = 0$ im unrestringierten Fall zu sehen.

Die gesuchten Werte $\widehat{x}$ und $\widehat{\lambda}$, die die Zulässigkeitsbedingung (1.23a) und die Optimalitätsbedingung (1.23b) erfüllen, können durch das Lösen des Gleichungssystems aus $(n + m)$ Gleichungen

$$\nabla L(x, \lambda) = \begin{pmatrix} \nabla_x L(x, \lambda) \\ \nabla_\lambda L(x, \lambda) \end{pmatrix} = \begin{pmatrix} \nabla f(x) - A^T\lambda \\ Ax - b \end{pmatrix} = 0$$

unter Verwendung des Newton-Verfahrens (siehe Abschnitt 1.9.1) bestimmt werden. Die Gleichungen besagen anschaulich, daß sowohl der Gradient der Lagrangeschen Funktion bezüglich x als auch der Vektor der Nebenbedingungen $Ax - b$ gleich Null sein sollen.

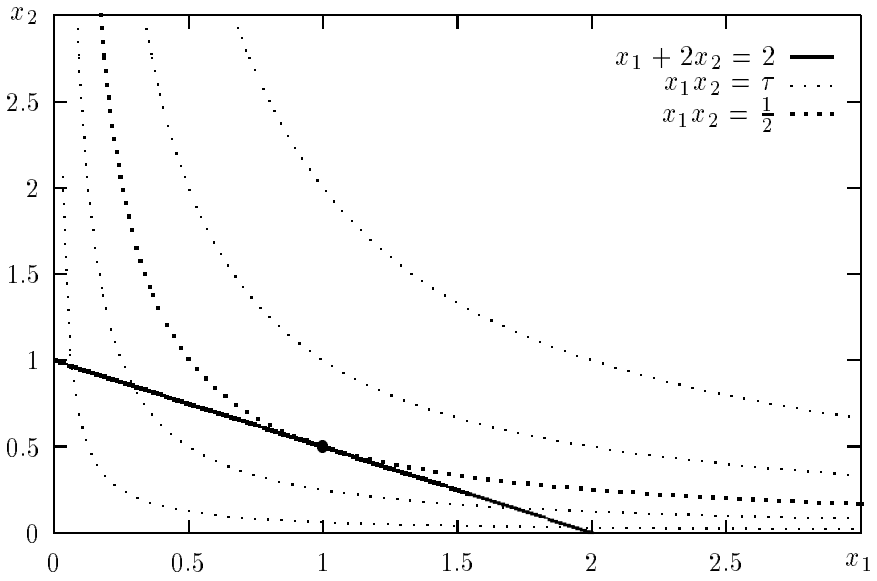

Abbildung 1.4: Zur Lagrangeschen Multiplikatorenregel: Höhenlinien der Zielfunktion $f(x)$ für verschiedene Funktionswerte τ und die Nebenbedingung $g(x) = 2$ zu Beispiel 1.7.1.

Beispiel 1.7.1 Bestimme den maximalen Funktionswert von $f(x) = x_1 x_2$ unter der Nebenbedingung $x_1 + 2x_2 = 2$, $x \geq 0$ (vgl. Abbildung 1.4).

Wendet man die Methode der *Lagrangeschen Multiplikatoren* auf das Problem

$$\begin{array}{rrcl} \text{minimiere} & f(x) & = & x_1 x_2 \\ \text{unter den Nebenbedingungen} & g(x) = x_1 + 2x_2 - 2 & = & 0 \end{array}$$

an, so lautet die resultierende *Lagrangesche Funktion*:

$$L(x, \lambda) = x_1 x_2 - \lambda \cdot (x_1 + 2x_2 - 2).$$

Zur Lösung des Problems bestimmt man eine Nullstelle des Gradienten von $L(x, \lambda)$. Dafür ergibt sich das Gleichungssystem

$$\nabla L(x, \lambda) = \begin{pmatrix} \nabla_x L(x, \lambda) \\ \nabla_\lambda L(x, \lambda) \end{pmatrix} = \begin{pmatrix} \nabla f(x) - \Big(\boldsymbol{J}_g(x) \Big)^T \lambda \\ g(x) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

und man erhält

$$\nabla L(\widehat{x}, \widehat{\lambda}) = \begin{pmatrix} & \widehat{x}_2 & - & \widehat{\lambda} & \\ \widehat{x}_1 & & - & 2\widehat{\lambda} & \\ \widehat{x}_1 & + \; 2\widehat{x}_2 & & & -2 \end{pmatrix} = 0.$$

Damit ist $\widehat{x}_2 = \widehat{\lambda}$ und $\widehat{x}_1 = 2\widehat{\lambda}$.
Durch Substitution in $\widehat{x}_1 + 2\widehat{x}_2 - 2 = 0$ folgt $2\widehat{\lambda} + 2\widehat{\lambda} = 2 \Rightarrow \widehat{\lambda} = \frac{1}{2}$
und damit $\begin{pmatrix} \widehat{x}_1 \\ \widehat{x}_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}$ und somit $f_{\max} = f\begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} = \frac{1}{2}$.

1.8 Kuhn-Tucker-Bedingungen

Die Kuhn-Tucker-Bedingungen ermöglichen eine Verallgemeinerung der klassischen Multiplikatorenmethode von Lagrange zur Bestimmung von Extrema unter Nebenbedingungen für den Fall, daß die Nebenbedingungen nicht nur Gleichungen, sondern auch Ungleichungen enthalten. Hierbei betrachtet man das allgemeine Optimierungsproblem in der Standardform (1.10), wobei der zulässige Bereich durch die Ungleichungen

$$g_j(x) \geq 0 \quad (j = 1, \ldots, m)$$

gegeben ist.

Gegeben seien die Lagrangesche Funktion $L : \mathbb{R}^{n+m} \to \mathbb{R}$ mit dem Vektor der Lagrangeschen Multiplikatoren $\lambda := (\lambda_1, \ldots, \lambda_m)^T$ als

$$L(x, \lambda) := f(x) - \sum_{j=1}^{m} \lambda_j g_j(x) = f(x) - \lambda^T g(x),$$

wobei $g = (g_1, \ldots, g_m) : \mathbb{R}^n \to \mathbb{R}^m$.

Nun wird der Begriff des Sattelpunktes einer Lagrangesche Funktion eingeführt.

Definition 1.8.1 Ein Punkt $(\widehat{x}, \widehat{\lambda})^T \in \mathbb{R}^{n+m}$ heißt *Sattelpunkt* von L, wenn

$$L(\widehat{x}, \lambda) \leq L(\widehat{x}, \widehat{\lambda}) \leq L(x, \widehat{\lambda}) \quad \forall x \in \mathbb{R}^n, \lambda \in \mathbb{R}^m_+ \tag{1.24}$$

mit $\widehat{\lambda} \geq 0$ gilt.

Man kann nun folgenden Satz zeigen, zum Beweis siehe Neumann [51]:

Satz 1.8.1 *Ist $(\widehat{x}, \widehat{\lambda})$ mit $\widehat{\lambda} \geq 0$ Sattelpunkt von L, so ist $\widehat{x}$ eine optimale Lösung des restringierten allgemeinen (nicht-linearen) Optimierungsproblems.*

Für die Umkehrung des Satzes 1.8.1 benötigt man die Konvexität des Optimierungsproblems sowie die folgende sogenannte *Slaterbedingung*:

Ist g_j nicht-linear, so gibt es ein $x' \in \mathbb{R}^n$
mit $g_j(x') < 0$ $(j = 1, \ldots, m)$.

Bei nicht-linearen Funktionen $g_1, \ldots, g_m$ impliziert die Slaterbedingung, daß der zulässige Bereich M innere Punkte enthält.

Satz 1.8.2 (Satz von Kuhn und Tucker) *Ist die Slaterbedingung erfüllt, so ist $\widehat{x}$ genau dann optimale Lösung des konvexen allgemeinen Optimierungsproblems, wenn L einen Sattelpunkt $(\widehat{x}, \widehat{\lambda})$ mit $\widehat{\lambda} \geq 0$ besitzt.*

Zum Beweis siehe [51].

Die Sattelpunktbedingung (1.24) stellt eine globale Bedingung für die Lagrangesche Funktion dar, die im allgemeinen schwer nachprüfbar ist. Sind die Funktionen $f, g_1, \ldots, g_m$ stetig differenzierbar und konvex, so läßt sich die Sattelpunktbedingung in Satz 1.8.2 durch äquivalente lokale Bedingungen ersetzen, zum Beweis siehe [22]:

Satz 1.8.3 *Ist die Slaterbedingung erfüllt und sind die Funktionen $f, g_1, \ldots, g_m$ stetig differenzierbar und konvex, so ist $\widehat{x}$ genau dann optimale Lösung des beschränkten allgemeinen Optimierungsproblems, wenn die folgenden Kuhn-Tucker-Bedingungen gelten:*

$$\begin{aligned} \nabla f(\widehat{x}) - \sum_{j=1}^{m} \widehat{\lambda}_j \nabla g_j(\widehat{x}) &= 0 \\ \sum_{j=1}^{m} \widehat{\lambda}_j g_j(\widehat{x}) &= 0 \\ g_j(\widehat{x}) &\geq 0 \quad (j = 1, \ldots, m) \\ \widehat{\lambda} &\geq 0. \end{aligned}$$

Beispiel 1.8.1 Man betrachte das allgemeine (nicht-lineare) Optimierungsproblem

$$\begin{aligned} &\text{minimiere} && (x_1 - 1)^2 + x_2^2 \\ &\text{unter der Nebenbedingung} && x_1 + x_2 \geq 2. \end{aligned}$$

Die Funktionen f und g mit

$$f(x_1, x_2) := (x_1 - 1)^2 + x_2^2 \quad \text{und} \quad g(x_1, x_2) := -x_1 - x_2 + 2$$

sind auf $\mathbb{R}^2$ konvex. Mit

$$\nabla f(x) = \begin{pmatrix} 2x_1 - 2 \\ 2x_2 \end{pmatrix} \quad \text{und} \quad \nabla g(x) = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

lauten die Kuhn-Tucker-Bedingungen

$$\begin{aligned} 2(\widehat{x}_1 - 1) - \widehat{\lambda} &= 0 \\ 2\widehat{x}_2 - \widehat{\lambda} &= 0 \\ \widehat{\lambda}(-\widehat{x}_1 - \widehat{x}_2 + 2) &= 0 \\ \widehat{x}_1 + \widehat{x}_2 - 2 &\geq 0 \\ \widehat{\lambda} &\geq 0. \end{aligned}$$

Für $\lambda = 0$ folgt aus den ersten beiden Gleichungen $x_1 = 1$, $x_2 = 0$. Dieser Punkt erfüllt jedoch nicht die Restriktion $x_1 + x_2 - 2 \geq 0$. Für $\lambda > 0$ folgt aus der dritten Gleichung $x_2 = 2 - x_1$ und die beiden ersten Gleichungen liefern $x_2 = x_1 - 1$. Hieraus erhält man $x_1 = 1.5$ und $x_2 = 0.5$. Dieser Punkt erfüllt die Restriktion $x_1 + x_2 - 2 \geq 0$. $\widehat{x} = (1.5, 0.5)^T$ und $\widehat{\lambda} = 1$ ist also die optimale Lösung.

1.9 Newton-Verfahren und Verifikation der Nullstelle

Sei $F : \mathbb{R}^n \to \mathbb{R}^n$ eine stetig differenzierbare Funktion und bezeichne $\boldsymbol{J}_F$ die dazugehörige Jacobi-Matrix. Das Problem der Nullstellenbestimmung für die Funktion F,

$$F(x) = 0, \tag{1.25}$$

kann mit Hilfe eines Iterationsverfahrens der Form

$$x^{(k+1)} := x^{(k)} + \alpha^{(k)} \Delta x^{(k)}$$

gelöst werden. Der Skalar $\alpha^{(k)}$ ist ein positiver Parameter für die Schrittweitensteuerung.

Definition 1.9.1 Ein spezielles Verfahren dieser Art stellt das *Newton-Verfahren*

$$x^{(k+1)} := x^{(k)} - \alpha^{(k)} \left(\boldsymbol{J}_F(x^{(k)}) \right)^{-1} F(x^{(k)}) \tag{1.26}$$

dar, wobei $\left(\boldsymbol{J}_F(x^{(k)}) \right)^{-1}$ die Inverse der Jacobi-Matrix bezeichnet.

Daß das Newton-Verfahren vorteilhafte Konvergenzeigenschaften in der Nähe einer Nullstelle besitzt, zeigt der folgende

Satz 1.9.1 (Konvergenz des Newton-Verfahrens) *Gegeben sei eine Abbildung $F : \mathbb{R}^n \to \mathbb{R}^n$, ein $\varepsilon > 0$ und ein $\widehat{x}$ mit $F(\widehat{x}) = 0$, so daß F in einer ε-Umgebung $U(\widehat{x}, \varepsilon)$ von $\widehat{x}$ differenzierbar ist. Weiterhin sei die Jacobi-Matrix $\boldsymbol{J}_F(\widehat{x})$ invertierbar.*
Dann existiert ein $\delta > 0$, so daß für ein $x^{(0)} \in U(\widehat{x}, \delta)$ mit der Iteration

$$x^{(k+1)} = x^{(k)} - \left(\boldsymbol{J}_F(x^{(k)})\right)^{-1} F(x^{(k)})$$

eine Folge $\{x^{(k)}\}$ wohldefiniert ist und gegen $\widehat{x}$ konvergiert. Es gilt für alle k: $x^{(k)} \in U(\widehat{x}, \delta)$.

Falls $x^{(k)} \neq \widehat{x}$ für alle k, dann gilt darüber hinaus

$$\lim_{k \to \infty} \frac{|x^{(k+1)} - \widehat{x}|}{|x^{(k)} - \widehat{x}|^2} = C, \quad C \geq 0 \text{ konstant},$$

d.h. die Folge $\{|x^{(k)} - \widehat{x}|\}$ konvergiert quadratisch.

Zum Beweis siehe [6].

Die Eigenschaft des Newton-Verfahrens, die Nullstelle einer differenzierbaren Funktion zu bestimmen, wird ausgenutzt, um das Verfahren auf ein unbeschränktes Optimierungsproblem anzuwenden. Nach Satz 1.5.2 lautet die notwendige Optimalitätsbedingung 1. Ordnung für eine Minimalstelle $\widehat{x}$ der Funktion f:

$$\boldsymbol{\nabla} f(\widehat{x}) = 0 \tag{1.27}$$

Sei nun f eine auf $\mathbb{R}^n$ zweimal stetig differenzierbare Funktion, d.h. $f \in C^2$. Für den Fall, daß f nur auf einer Teilmenge $M \subseteq \mathbb{R}^n$ differenzierbar ist, gilt das folgende analog. Um die Minimalstelle einer zweimal stetig differenzierbaren Funktion $f(x)$ zu bestimmen, ersetzt man die Funktion $F(x)$ in der Iterationsvorschrift des Newton-Verfahrens (1.26) durch $\boldsymbol{\nabla} f(x)$.

Definition 1.9.2 Allgemein hat das *Newton-Verfahren für reellwertige n-dimensionale unrestringierte Optimierungsprobleme* dann die Form

$$x^{(k+1)} := x^{(k)} - \alpha^{(k)} \left(\boldsymbol{\nabla}^2 f(x^{(k)})\right)^{-1} \boldsymbol{\nabla} f(x^{(k)}),$$

wobei vorausgesetzt wird, daß die Inversen der Hesse-Matrizen $\left(\boldsymbol{\nabla}^2 f(x^{(k)})\right)^{-1}$ existieren und daß die *Newton-Korrektur*

$$\Delta x^{(k)} = -\left(\boldsymbol{\nabla}^2 f(x^{(k)})\right)^{-1} \boldsymbol{\nabla} f(x^{(k)})$$

eine Abstiegsrichtung (d.h. $\Delta x^{(k)} \nabla f(x^{(k)}) < 0$) ist. Die Newton-Korrektur wird als Lösung des linearen Gleichungssystems

$$\nabla^2 f(x^{(k)}) \Delta x^{(k)} = -\nabla f(x^{(k)})$$

bestimmt. Allgemein wird eine solche Methode zur Nullstellenbestimmung als *Gradienten-Methode* bezeichnet.

Algorithmische Beschreibung

Der Newton-Algorithmus kann in der folgenden Form dargestellt werden:

Algorithmus 1.9.1: NewtonMethod $(f, x^{(0)}, x^{(k+1)}, Err)$

1. {Initialisierung.}
   $k := -1$; $k_{\max} := 5$; $Err :=$ „No Error“;
2. **repeat**
   (a) $k := k + 1$; $approx :=$ **false** ;
   (b) {Bestimme die Newton-Korrektur Δx aus Definition 1.9.2.}
   lss_aprx $(\nabla^2 f(x^{(k)}), -\nabla f(x^{(k)}), \Delta x, Err)$;
   (c) {Bestimme die neue Iterierte $x^{(k+1)}$}
   $x^{(k+1)} := x^{(k)} + \alpha \cdot \Delta x$;
   (d) {Überprüfe den Grad der Verbesserung der Näherung.}
   if $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ **then** $approx :=$ **true** ;

   until $approx$ **or** $k = k_{\max}$ **or** $Err <>$ „No Error“;
3. **return** $x^{(k+1)}$, Err;

Die in Schritt 2b eingesetzte Prozedur lss_aprx löst ein lineares Gleichungssystem mit einem beliebigen üblichen Näherungsverfahren. Zur Beschreibung sei auf den PASCAL–XSC User's Guide [52] verwiesen.

1.9.1 Anwendung des Newton-Verfahrens auf die Lagrangesche Funktion

Gesucht sei das Minimum der Lagrangeschen Funktion (vgl. Abschnitt 1.7)

$$L(x, \lambda) = f(x) - \lambda^T \cdot g(x),$$

d.h. die Lösung für

$$\nabla L(x,\lambda) = \begin{pmatrix} \nabla_x L(x,\lambda) \\ \nabla_\lambda L(x,\lambda) \end{pmatrix} = \begin{pmatrix} \nabla f(x) - \boldsymbol{J}_g(x)\lambda \\ g(x) \end{pmatrix} = 0. \tag{1.28}$$

Die Newton-Iteration zur Lösung dieses Systems wird definiert durch:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \Delta x^{(k)}, \\ \lambda^{(k+1)} &= \lambda^{(k)} + \Delta\lambda^{(k)}, \end{aligned}$$

wobei $(\Delta x^{(k)}, \Delta\lambda^{(k)}) \in I\!R^{n+m}$ durch die Lösung des Gleichungssystems

$$\nabla^2 L(x^{(k)},\lambda^{(k)}) \begin{pmatrix} \Delta x^{(k)} \\ \Delta\lambda^{(k)} \end{pmatrix} = -\nabla L(x^{(k)},\lambda^{(k)}) \tag{1.29}$$

bestimmt wird. Man erhält also:

$$\nabla^2 L(x^{(k)},\lambda^{(k)}) = \begin{pmatrix} H^{(k)} & M^{(k)T} \\ M^{(k)} & 0 \end{pmatrix}, \quad \nabla L(x^{(k)},\lambda^{(k)}) = \begin{pmatrix} \nabla_x L(x^{(k)},\lambda^{(k)}) \\ g(x^{(k)}) \end{pmatrix}$$

mit

$$H^{(k)} = \nabla^2_{xx} L(x^{(k)},\lambda^{(k)}), \quad M^{(k)} = \boldsymbol{J}_g(x^{(k)}).$$

Das Gleichungssystem (1.29) hat dann die Form:

$$\begin{pmatrix} H^{(k)} & M^{(k)T} \\ M^{(k)} & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta x^{(k)} \\ \Delta\lambda^{(k)} \end{pmatrix} = -\begin{pmatrix} \nabla_x L(x^{(k)},\lambda^{(k)}) \\ g(x^{(k)}) \end{pmatrix}. \tag{1.30}$$

1.9.2 Verifikation der Nullstelle des Gradienten

Da die exakte Lösung des Nullstellenproblems (1.27) i.a. weder berechenbar noch im Maschinenzahlenformat darstellbar ist, ist die numerische Bestimmung einer Einschließung der gesuchten Lösung von Interesse. Die im folgenden dargestellte a-posteriori Methode geht von einer Näherungslösung $\widetilde{x}$ des Problems (1.27) aus und überprüft, ob eine Umgebung $[x]$ von $\widetilde{x}$ eine Nullstelle enthält. Anstelle eines Bisektionsverfahren als a-posteriori Methode, wird hier ein Verfahren verwandt, das auf intervallanalytischen Prinzipien und der Fixpunkt-Theorie basiert. Grundlegende Angaben zu dieser Thematik finden sich in [16] und [37].

Unter der Voraussetzung, daß $f : I\!R^n \to I\!R^n$ auf $[x] \in I\,I\!R^n$ zweimal stetig differenzierbar ist, liefert die Zwischenwertform, daß für alle $x_1, x_2 \in [x]$ gilt:

es existiert ein $\xi = (\xi_1, \ldots, \xi_n)$ mit $\xi_i \in [x]$, so daß

$$\boldsymbol{\nabla} f(x_1) - \boldsymbol{\nabla} f(x_2) = \boldsymbol{\nabla}^2 f(\xi) \cdot (x_1 - x_2)$$

Für den folgenden Einschließungssatz benötigt man das

Lemma 1.9.2 *Seien $[x], [z] \in I\!I\!R^n$ Intervallvektoren und sei $[B] \in I\!I\!R^{n \times n}$ eine Intervallmatrix. Falls gilt*

$$[z] + [B][x] \overset{\circ}{\subset} [x],$$

dann gilt für alle Punktmatrizen $B \in [B]$, daß der Spektralradius $\rho(B) < 1$ ist.

Zum Beweis siehe [7].

Eine wesentliche Rolle bei Verifikations- und Einschließungsverfahren spielt der

Satz 1.9.3 (Fixpunktsatz von Brouwer) *Jede stetige Selbstabbildung $f : I\!R^n \to I\!R^n$ einer nichtleeren, konvexen, kompakten Menge $[x] \subseteq I\!R^n$ hat mindestens einen Fixpunkt in $[x]$.*

Zum Beweis siehe Heuser [21], S. 603 ff.

Wir wollen nun einen Satz angeben, der sowohl die Existenz als auch die Eindeutigkeit einer Lösung des Nullstellenproblems (1.27) garantiert und dessen Voraussetzungen auf einem Rechner automatisch überprüft werden können.

Satz 1.9.4 (1. Einschließungssatz) *Seien ein Intervallvektor $[x] \in I\!I\!R^n$ und ein Vektor $\widetilde{x} \in [x]$ gegeben und sei die Funktion $f : [x] \to I\!R^n$ zweimal stetig differenzierbar. Wenn gilt*

$$N([x]) := \widetilde{x} - R \cdot \boldsymbol{\nabla} f(\widetilde{x}) + \Big(I - R \cdot \boldsymbol{\nabla}^2 f([x]) \Big) \cdot [x] \overset{\circ}{\subset} [x] \qquad (1.31)$$

für eine gegebene Matrix R und eine Intervallauswertung der Hesse-Matrix $\boldsymbol{\nabla}^2 f$ über ganz $[x]$, dann folgt

1. *die Matrix R und alle Punkt-Hesse-Matrizen $\boldsymbol{\nabla}^2 f(\xi)$ mit $\xi \in [x]$ sind nicht-singulär und*
2. *der Gradient $\boldsymbol{\nabla} f$ hat genau eine Nullstelle $\widehat{x}$ in $[x]$.*

Beweis: Angenommen $n(x) := x - R \cdot \nabla f(x)$ sei auf $[x]$ stetig. Da f zweimal stetig differenzierbar ist, existiert nach dem Zwischenwertsatz für die Vektoren $x, \tilde{x} \in [x]$ eine Punktmatrix $\nabla^2 f(\xi)$ mit $\xi \in [x]$, für die gilt

$$\begin{aligned} n(x) &= x - R\left(\nabla f(\tilde{x}) + \nabla^2 f(\xi) \cdot (x - \tilde{x}) \right) \\ &= \tilde{x} - R \cdot \nabla f(\tilde{x}) + \left(I - R \cdot \nabla^2 f(\xi) \right) \cdot (x - \tilde{x}) \quad \in \quad N([x]). \end{aligned}$$

Satz 1.9.3 liefert hierfür die Existenz eines Fixpunktes $\hat{x}$ der Funktion n in $[x]$. Mit Lemma 1.9.2 folgt weiterhin, daß alle Punktmatrizen $B \in \left(I - R \cdot \nabla^2 f([x]) \right)$ nicht-singulär sind. Aus der Betrachtung der Eigenwerte aller Matrizen B folgt sofort, daß auch die Matrix R sowie alle Punktmatrizen $\nabla^2 f(\xi) \in \nabla^2 f([x])$ mit $\xi \in [x]$ nicht-singulär sind. Somit folgt aus

$$\hat{x} = n(\hat{x}) = \hat{x} - R \cdot \nabla f(\hat{x}),$$

daß der Gradient ∇f an der Stelle $\hat{x}$ eine Nullstelle hat. Sei $\hat{y}$ eine weitere Nullstelle von ∇f in $[x]$. Dann existiert nach dem Zwischenwertsatz eine Punktmatrix $\nabla^2 f(\psi) \in \nabla^2 f([x])$ für die gilt $\nabla f(\hat{y}) = \nabla f(\hat{x}) + \nabla^2 f(\psi)(\hat{x} - \hat{y})$. Aus der Nicht-Singularität von $\nabla^2 f(\psi)$ folgt aber $\hat{x} = \hat{y}$. □

Nach den Voraussetzungen des Einschließungssatzes 1.9.4 kann die Matrix $R \in I\!R^{n \times n}$ beliebig gewählt werden. Es ist allerdings offensichtlich, daß die Bedingung (1.31) um so eher erfüllt sein wird, je näher R an der Inversen der Hesse-Matrix $\left(\nabla^2 f(\tilde{x}) \right)^{-1}$ liegt.

Daß der Satz 1.9.4 auch auf dem Rechner anwendbar ist, zeigt das

Korollar 1.9.5 (Defekteinschließung) *Seien $[u]$ aus dem Raum der Maschinenintervallvektoren VIR und $\tilde{x} \in [u]$ aus dem Raum der Maschinenvektoren VR gegeben und sei $f : \tilde{x} \oplus (\vec{0} \cup [u]) \to I\!R^n$ zweimal stetig differenzierbar. Wenn gilt*

$$N_\diamond([u]) := -R \circledast \nabla f_\diamond(\tilde{x}) + \diamondsuit \left(I - R \cdot \nabla^2 f_\diamond(\tilde{x} \oplus (\vec{0} \cup [u])) \right) \circledast [u] \overset{\circ}{\subset} [u] \tag{1.32}$$

für eine gegebene Matrix R und eine Maschinenintervallauswertung der Hesse-Matrix $\nabla^2 f_\diamond$ über ganz $\tilde{x} \oplus (\vec{0} \cup [u])$, dann folgt

1. *die Matrix R und alle Punkt-Hesse-Matrizen $\nabla^2 f(\xi) \in \nabla^2 f_\diamond(\tilde{x} \oplus (\vec{0} \cup [u]))$ sind nicht-singulär und*
2. *der Gradient ∇f hat genau eine Nullstelle $\hat{x}$ in $\tilde{x} \oplus [u]$.*

Beweis: Im Unterschied zu Satz 1.9.4 muß der Intervallvektor $[x] \in I\!I\!R^n$ durch $\widetilde{x} \oplus [u]$ ersetzt werden. Außerdem ist zu berücksichtigen, daß Satz 1.9.4 weiterhin gilt, wenn $N([x])$ durch eine Obermenge $N_\diamond([x])$ ersetzt wird (vgl. Abschnitt 1.3), solange die Bedingung $N_\diamond([x]) \overset{\circ}{\subset} [x]$ erfüllt bleibt. □

Bemerkung:

1. Wenn es gelingt, die Bedingung (1.32) auf einem Rechner nachzuprüfen, kann ein Verfahren angeben werden, das die Existenz und die Eindeutigkeit der Nullstelle des Gradienten ∇f garantiert. Zudem wird als Ergebnis des Verfahrens ein Intervall $\widetilde{x} \oplus [u]$ angegeben, in dem die Lösung $\widehat{x}$ liegt, d.h. man hat mit dem Durchmesser des Intervalls gleichzeitig sogar eine garantierte, verifizierte Fehlerabschätzung der berechneten Lösung zur Verfügung.

2. Der sogenannte Verifikationsschritt, d.h. die Berechnung von $N_\diamond([u])$ und die Überprüfung der Inklusionsbedingung (1.32) wird in einer Schleife durchgeführt, da die Inklusionsbedingung, die auch als Abbruchbedingung für diese Schleife herangezogen wird, durchaus erst nach einigen Iterationen erfüllt sein kann.

3. In den Schritten 1 und 2d des folgenden Algorithmus wird das im Iterationsprozeß befindliche Intervall etwas vergrößert, was im weiteren als *ε-Aufblähung* bezeichnet wird. Die ε-Aufblähung für ein reelles Gleitpunktintervall $[x] \in IR$ ist als

$$[x] \bowtie \varepsilon := \begin{cases} [x] + [-\varepsilon, \varepsilon] \cdot d([x]) & \text{falls } d([x]) \neq 0 \\ [x] + [-x_{\min}, x_{\min}] & \text{sonst} \end{cases}$$

   definiert, wobei ε eine positive reelle Zahl ist, $d([x])$ den Durchmesser des Intervalls $[x]$ bezeichnet und $x_{\min}$ die kleinste positive darstellbare Maschinenzahl des zugrundeliegenden Maschinenzahlsystems R ist. Da die Startnäherung i.a. nicht die exakte Lösung des Optimierungsproblems enthält und der Iterationsprozeß sich möglicherweise der Lösung immer stärker annähert ohne sie tatsächlich zu erreichen, kann zu dem Hilfsmittel der ε-Aufblähung gegriffen werden, um die Erfolgsaussichten des Verfahrens zu erhöhen. Vor dem Start eines neuen Iterationsschrittes wird die aktuelle Iterierte etwas aufgebläht, so daß der Algorithmus

$$[u]^{(k+1)} := N_\diamond([u]^{(k)})$$

abgewandelt wird in

$$[u]^{(k)} := [u]^{(k)} \bowtie \varepsilon$$
$$[u]^{(k+1)} := N_{\diamond}([u]^{(k)})$$

Zur Illustration der Prinzips siehe auch [20], Seiten 47 und 51 ff.

Algorithmische Beschreibung

Das Vorstehende führt auf das folgende Verfahren zur Verifikation der Nullstelle der Gradienten einer zweimal stetig differenzierbaren Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ und kann in der hier angegebenen algorithmischen Form dargestellt werden:

Algorithmus 1.9.2: Verification $(f, \tilde{x}, [x], unique)$

1. {Initialisierung.}
   $k := -1$; $k_{\max} := 5$; $Err :=$ "No Error";
   $[u]^{(0)} := [\vec{0}] \bowtie \varepsilon$; $[h] := R \circledast \nabla f_{\diamond}(\tilde{x})$;

2. **repeat**

   (a) $k := k + 1$;

   (b) {Bestimme den Verifikationsschritt aus Definition 1.32.}
   $$[u]^{(k+1)} := -[h] + \diamondsuit \Big(I - R \cdot \nabla^2 f_{\diamond}(\tilde{x} \oplus (\vec{0} \cup [u]^{(k)})) \Big) \circledast [u]^{(k)};$$

   (c) {Überprüfe die Existenz und Eindeutigkeit der Nullstelle.}
   $unique := [u]^{(k+1)} \overset{\circ}{\subset} [u]^{(k)}$;

   (d) **if not** *unique*
   {Führe eine ε-Aufblähung durch, um die Inkusion im}
   {nächsten Schritt zu erreichen. }
   $[u]^{(k+1)} := [u]^{(k+1)} \bowtie \varepsilon$;

   until *unique* **or** $k = k_{\max}$;

3. **return** $[x] := \tilde{x} + [u]^{(k+1)}$; *unique*;

1.10 Terminologie der Theorie des Simplex-Algorithmus

Die in Kapitel 2 folgende Diskussion der Innere-Punkt-Methode als Lösungsverfahren für lineare Optimierungsprobleme wird zeigen, daß dieses asymptotische Verfahren nicht in der Lage ist, mit einer endlichen Anzahl an Iterationsschritten die exakte Lösung eines (LP) zu bestimmen. Durch das Mitverwenden der Strategie der Barriere-Methode wird, ausgehend von einem strikt zulässigen (inneren) Startpunkt, in jedem Iterationsschritt ein neuer, ebenfalls strikt zulässiger Punkt generiert. Ein zentraler Satz der linearen Optimierungstheorie (s.u.) besagt aber, daß im Falle der Existenz einer optimalen Lösung eines gegebenen

$$\begin{array}{lrcl} & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & \text{unter den Nebenbedingungen} & Ax & = & b \\ & \text{und} & x & \geq & \vec{0}, \end{array}$$

($A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c, x \in \mathbb{R}^n$) ein optimaler Lösungsvektor höchstens m von Null verschiedene Komponenten besitzt. Geometrisch betrachtet entspricht jeder zulässige Punkt mit maximal m von Null verschiedenen Komponenten einer Ecke des durch die Nebenbedingungen des (LP) definierten konvexen Polyeders $M := \{x \in \mathbb{R}^n \mid Ax = b, x \geq \vec{0}\}$. Hierdurch wird klar, daß die exakte Lösung des (LP) immer ein Randpunkt ist. Im Gegensatz zur unbeschränkten Iterationszahl bei Innere-Punkt-Verfahren steht nun die Tatsache, daß die Anzahl dieser Eckpunkte $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ beträgt und damit endlich ist. Auf diesen Aussagen basiert das von Dantzig entwickelte Simplex-Verfahren, das die Eckpunkte nacheinander auf Optimalität hin überprüft. Für eine vollständige Erläuterung dieser Zusammenhänge sei auf die einführenden Werke von Collatz und Wetterling [10] oder Morlock und Neumann [51] verwiesen.

Lösungen des Gleichungssystems $Ax = b$, bei denen genau $n - m$ Komponenten des Vektors x gleich Null sind, können durch eine Zerlegung der rechteckigen Matrix A, die aus den Spaltenvektoren $(a_{*,1}, \ldots, a_{*,n})$ besteht, in zwei Teilmatrizen A_B und A_N beschrieben werden. $A_B = (a_{*,\beta_1}, \ldots, a_{*,\beta_m})$ ist dabei eine nicht-singuläre $m \times m$ Teilmatrix mit den Spaltenvektoren a_{*,β_i} der Matrix A und $A_N = (a_{*,\nu_1}, \ldots, a_{*,\nu_{n-m}})$ ist die $m \times (n - m)$ Teilmatrix bestehend aus den übrigen Spalten der Matrix A.

Definition 1.10.1 Die Indexmenge $\mathcal{B} = \{\beta_1, \ldots, \beta_m\} \subseteq \{1, \ldots, n\}$, die die nicht-singuläre quadratische Matrix A_B bestimmt, heißt *Basis-Indexmenge*

des (LP). Die Indexmenge $\mathcal{N} := \{\nu_1, \ldots, \nu_{n-m}\} := \{1, \ldots, n\} \setminus \mathcal{B}$ wird als *Nicht-Basis-Indexmenge* bezeichnet.

Für eine gegebene Zerlegung $\mathcal{B}$ und $\mathcal{N}$ kann das lineare Gleichungssystem $Ax = b$ in der Form

$$A_B \cdot x_B + A_N \cdot x_N = b \tag{1.33}$$

dargestellt werden, wobei $x = (x_B, x_N)^{\mathrm{T}}$ der analog zerlegte Lösungsvektor ist. Multipliziert man die Gleichung (1.33) mit der Inversen A_B^{-1}, so erhält man

$$x_B = A_B^{-1} \cdot b - H \cdot x_N \quad \text{mit} \quad H := A_B^{-1} \cdot A_N.$$

Zerlegt man $c = (c_B, c_N)^{\mathrm{T}}$ entsprechend, so kann die Zielfunktion in der Form

$$\begin{aligned} c^{\mathrm{T}}x = c_B^{\mathrm{T}} \cdot x_B + c_N^{\mathrm{T}} \cdot x_N &= c_B^{\mathrm{T}} \cdot (A_B^{-1} \cdot b - H \cdot x_N) + c_N^{\mathrm{T}} \cdot x_N \\ &= c_B^{\mathrm{T}} \cdot A_B^{-1} \cdot b - (c_B^{\mathrm{T}} \cdot H - c_N^{\mathrm{T}}) \cdot x_N. \end{aligned}$$

angegeben werden. Daraus folgt für $z = (z_B, z_N)^{\mathrm{T}}$

$$c^{\mathrm{T}}x = c_B^{\mathrm{T}} \cdot A_B^{-1} b - z_N^{\mathrm{T}} \cdot x_N \quad \text{mit} \quad z_N := H^{\mathrm{T}} \cdot c_B - c_N. \tag{1.34}$$

Das heißt, daß sowohl die m Variablen x_B als auch der Wert der Zielfunktion durch die $n - m$ Komponenten des Vektors x_N bestimmt sind.

Definition 1.10.2 Eine *Basis-Lösung* (bezüglich der Basis-Indexmenge $\mathcal{B}$) ist eine Lösung x bezüglich der Nebenbedingungen $Ax = b$ aus (LP), deren $n - m$ Komponenten x_N gleich Null sind.

$$x := (x_B, x_N)^{\mathrm{T}} \quad \text{mit} \quad x_B := A_B^{-1} \cdot b \quad \text{und} \quad x_N := 0.$$

Gilt zusätzlich $x \geq 0$ (d.h. $x_B := A_B^{-1} b \geq 0$), dann heißt $x \in M$ eine *zulässige Basis-Lösung*. Der Wert der Zielfunktion ist dann $f = c^{\mathrm{T}}x = c_B^{\mathrm{T}} A_B^{-1} b$. Der Vektor z_N aus (1.34) heißt der *Vektor der reduzierten* oder *relativen Kosten*, da er angibt, wie stark die Zielfunktion von x_N abhängt.
Liefert $f(x)$ eine Minimallösung für das (LP), so heißt x eine *zulässige optimale Basis-Lösung*.

Es gilt nun der oben erwähnte

Satz 1.10.1 (Fundamentalsatz der linearen Optimierung) *Gegeben sei ein lineares Optimierungsproblem (LP) in Standardform, wobei A eine $(m \times n)$ Matrix vom Rang m ist. Dann gilt:*

(i) Falls es eine zulässige Lösung gibt, so existiert auch eine zulässige Basis-Lösung.

(ii) Falls es eine zulässige optimale Lösung gibt, so existiert auch eine zulässige optimale Basis-Lösung.

Zum Beweis siehe Luenberger [44].

Kapitel 2

Innere-Punkt-Methoden und Ergebnisverifikation

2.1 Karmarkar-Algorithmus

2.1.1 Beschreibung

Das 1984 von Karmarkar entwickelte Projektionsverfahren [27], auch *Karmarkar-Algorithmus* genannt, basiert auf einem linearen Optimierungsproblem der Form

$$\text{(KarP)} \qquad \begin{array}{rrcl} & \text{minimiere} & f(x) & = & c^T x \\ & \text{unter den Nebenbedingungen} & Ax & = & 0, \\ & & e^T x & = & 1, \\ & & x & \geq & 0, \end{array}$$

wobei A eine reelle $(m \times n)$-Matrix ist, c und $x \in \mathbb{R}^n$ und e ein n-dimensionaler Vektor mit 1 in allen Komponenten ist. Weiterhin nimmt Karmarkar an, daß für die optimale Lösung $\hat{x}$ von (KarP) der Zielfunktionswert $f(\hat{x}) = 0$ ist.

Da (KarP) nicht der Standardform eines linearen Optimierungsproblems (LP) (vgl. Definition 1.4.4) entspricht, zeigte Karmarkar, daß jedes (LP) in die Form (KarP) transformiert werden kann [27].

Zur folgenden Kurzbeschreibung vergleiche z.B. Richter [55]. Der Karmarkar-Algorithmus startet an einer Anfangsnäherung $x^{(0)}$ der Lösung von (KarP), für die $Ax^{(0)} = 0$, $e^T x^{(0)} = 1$ und $x^{(0)} > 0$ gilt. Der Algorithmus bewegt sich mit Hilfe von projektiven Transformationen zu einer neuen Näherung $x^{(1)}$.

Sei $T(x) := \frac{(X^{(0)})^{-1}x}{e^T(X^{(0)})^{-1}x}$ mit der Diagonalmatrix $X^{(0)}$, deren Diagonalelemente die Komponenten von $x^{(0)}$ sind, eine projektive Transformation, die $x^{(0)}$ in $\frac{1}{n} \cdot e$ abbildet. Die Matrix B sei definiert als $\begin{bmatrix} AX^{(0)} \\ e^T \end{bmatrix}$ und der Vektor $\delta \in \mathbb{R}^n$ sei definiert als

$$\delta = \alpha \left(I - B^T(BB^T)^{-1}B \right) X^{(0)}c, \tag{2.1}$$

wobei der positive Parameter α die Schrittweite steuert. Ein neuer Punkt $\xi \in \mathbb{R}^n$ wird dann bestimmt durch

$$\xi = \frac{1}{n}e + \delta$$

und die neue Näherung $x^{(1)}$ erhält man durch die Inverse der projektiven Transformation $T^{-1}(\xi)$, die definiert ist als

$$x^{(1)} = T^{-1}(\xi) = \frac{X^{(0)}\xi}{e^T X^{(0)}\xi}.$$

Karmarkar zeigt, daß bei einer geeigneten Wahl von α in (2.1) in $\boldsymbol{O}(n^{3,5}K)$ Iterationen eine Näherungslösung $\tilde{x}$ bestimmt wird, für die $f(\tilde{x}) < 2^{-K}f(x^{(0)})$ gilt, wobei

$$K = \sum_{i=0}^{m}\sum_{j=0}^{n} \lceil \log(|a_{ij}| + 1) + 1 \rceil, \tag{2.2}$$

mit $a_{i0} = b_i$ und $a_{0j} = c_j$, der Zahl der notwendigen Bits entspricht, die für die Darstellung der Komponenten von A, b und c benötigt werden [27]. Die Bezeichnung $\lceil \cdot \rceil$ steht für den ganzzahligen Anteil des eingeklammerten reellen Ausdrucks.

Um die Komplexität seines Algorithmus zu beweisen, führte Karmarkar die Potentialfunktion

$$p(x) = \ln(f(x)) - \sum_{j=1}^{n} \ln(x_j)$$

ein. Er zeigt, daß die Potentialfunktion bei einer geeigneten Wahl des Parameters α zumindestens um einen konstanten Wert in jedem Iterationsschritt reduziert wird. Dies liefert die gewünschte Aussage über die Komplexität des Verfahrens (vgl. [4]). Allerdings erwies sich die von Karmarkar in seiner

ersten Veröffentlichung vorgeschlagene Wahl des Parameters α in der Praxis (und insbesondere im Vergleich zum Simplex-Algorithmus) als ungeeignet, da sie in allen Fällen zur maximal möglichen Anzahl an Iterationen führte.

Wesentlich ist die Tatsache, daß das Projektionsverfahren von Karmarkar als ein beschränktes Newton-Verfahren bezüglich der Potentialfunktion $p(x)$ hergeleitet werden kann. Dies führt zu den in den folgenden Abschnitten beschriebenen Herleitungen des Karmarkar-Algorithmus mit Hilfe allgemeiner logarithmischer Barriere-Methoden.

Bei der Verwendung des ursprünglichen Algorithmus stellte insbesondere die Konvertierung eines linearen Optimierungsproblems in Standardform (LP) in die homogene Form Karmarkars (KarP) eine Schwierigkeit dar. Karmarkars Vorschlag führte auf eine Verdoppelung der Dimension der Nebenbedingungsmatrix A.

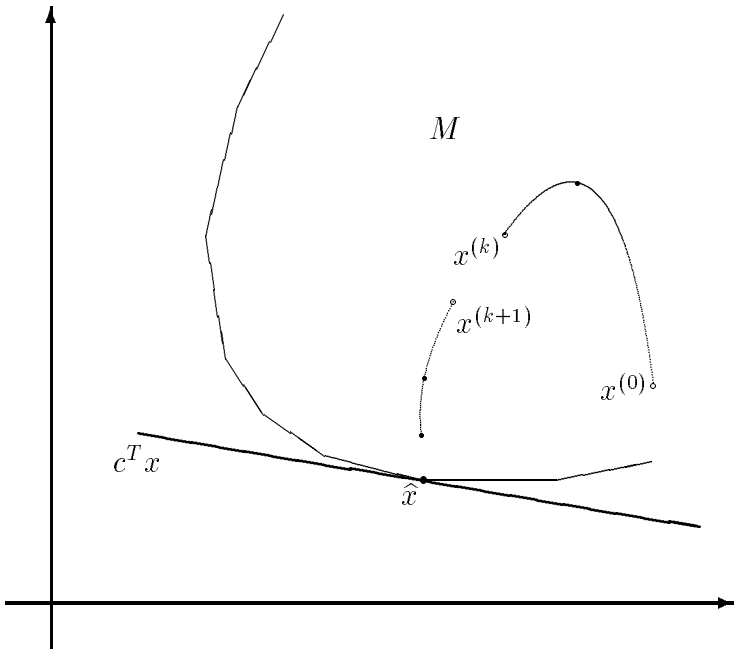

Abbildung 2.1: Graphische Darstellung des *Karmarkar-Algorithmus*

2.1.2 Illustration

Der Iterationsprozeß des Karmarkar-Algorithmus führt durch das Innere des zulässigen Bereiches M (vgl. Abbildung 2.1). Ausgehend von einem strikt zulässigen Punkt $x^{(0)} \in \overset{\circ}{M}$ springt der Iterationsprozeß für einen i.a. rela-

tiv großen Startwert des Barriere-Parameters in Richtung des analytischen Zentrums des zulässigen Bereiches und nähert sich dann asymptotisch der optimalen Lösung $\hat{x}$.

2.1.3 Aufwandsabschätzung

Der *Simplex-Algorithmus* besticht unverändert durch seine Effizienz und Zuverlässigkeit bei kleinen, z.T. aber auch bei mittleren und größeren Anwendungen, den sogenannten „real-world"-Problemen. Zur Problemlösung wird hierbei häufig nur ein kleines Vielfaches (2-3) der Problemdimension an Iterationsschritten benötigt. Da die Zahl der Ecken eines beliebigen (LP) endlich ist, konvergiert das Simplex-Verfahren unter relativ schwachen Voraussetzungen garantiert. Nachteilig wirkt sich hingegen die Tatsache aus, daß die Zahl der Ecken exponentiell mit der Dimension des Problems wächst. Ein typisches Beispiel ist der „gedrehte Quader", ein Beispiel von Klee und Minty [33], der ein (LP) mit n Unbekannten und $2n$ Ungleichungen als Nebenbedingungen bestimmt. Der Standard-Algorithmus sucht im Laufe des Iterationsprozesses alle 2^n Ecken auf. Die „worst-case" Komplexität des Simplex-Algorithmus ist somit exponentiell abhängig von der Dimension des Problems. Daß dieser Fall in der Praxis nur sehr selten eintritt, ist ein bislang ungeklärtes Phänomen.

Der *Karmarkar-Algorithmus* stellt nach einer Reihe von weniger erfolgreichen Versuchen die erste Lösungsstrategie dar, deren Komplexität polynomial begrenzt ist und die auch im praktischen Einsatz mit dem Simplex-Algorithmus konkurrieren kann. Im Vergleich erreicht diese *Innere-Punkt-Methode* einen Beschleunigungsfaktor von ca. 50 gegenüber dem traditionellen Simplex-Algorithmus bei zahlreichen komplexen Anwendungen.

Bei Problemgrößen kleiner Dimension ($n < 100$) behält das Simplex-Verfahren noch in den meisten Fällen einen Vorteil, da der Rechenaufwand häufig nur bei $\boldsymbol{O}(3n)$ liegt, während der Karmarkar-Algorithmus durch den problemabhängigen Wert K aus (2.2), der i.a. wesentlich größer als $3n$ ist, bestimmt wird. Bei mittleren Problemdimensionen ($100 \leq n < 10000$) kehren sich diese Verhältnisse im Durchschnitt bereits um, da hier der Aufwand, den der Simplex-Algorithmus benötigt, in der Regel quadratisch ansteigt, wohingegen der Aufwand des Karmarkar-Algorithmus mit $\boldsymbol{O}(K)$ quasi unverändert bleibt. Bei großen linearen Optimierungsproblemen ($n \geq 10000$) erweist sich Karmarkar als deutlich überlegen mit einem Aufwand von $\boldsymbol{O}(n^{3,5}K)$ gegenüber beobachteten $\boldsymbol{O}(n^4)$ bis $\boldsymbol{O}(n^5)$ für den Simplex. Diese Vergleiche werden in Abbildung 2.2 nochmals zusammengestellt.

Vergleichkriterium	**Simplex**	**Karmarkar**
Autor	G. Dantzig (1947)	N. Karmarkar (1984)
Aufwand bzgl. Problem-dimension n	Durchschnitt	Durchschnitt
$n < 100$	$\boldsymbol{O}(3n)$	$\boldsymbol{O}(K)$
$100 \leq n < 10.000$	$\boldsymbol{O}(n^2)$	$\boldsymbol{O}(K)$
$n \geq 10.000$	$\boldsymbol{O}(n^4)$	$\boldsymbol{O}(n^3K)$
worst case	$\boldsymbol{O}(2^n)$	$\boldsymbol{O}(n^{3,5}K)$
Konvergenz	i.a. garantiert	asymptotisch
behandelbare Problemklassen	linear	linear, (nicht-linear) konvex, kombinatorisch

Abbildung 2.2: Vergleich von Simplex- und Karmarkar-Algorithmus
Mit K aus (2.2)

Auf Grund der Äquivalenz zwischen dem Karmarkar-Algorithmus und den klassischen Barriere-Verfahren ist es möglich, dieses Verfahren, im Gegensatz zum Simplex-Algorithmus, auch auf nichtlineare Optimierungsprobleme anzuwenden. Der zentrale Gedanke der Innere-Punkt-Methoden ist die Konstruktion einer stetig von einem Parameter abhängenden Folge von Näherungslösungen, die asymptotisch gegen die exakte Lösung konvergiert. Während sich der Parameter seinem Grenzwert nähert, beschreibt die Folge der Näherungen einen glatten Graphen, dessen geometrische Eigenschaften analysierbar sind.

2.2 Primale Innere-Punkt-Methode

Die *primale Innere-Punkt-Methode* bezeichnet eine Kombination aus Barriere-Verfahren, Lagrangescher Multiplikatorenregel und Newton-Verfahren, die einen zum Karmarkar-Algorithmus äquivalenten Lösungsansatz für lineare Optimierungsprobleme darstellt.

Man geht von einem linearen Optimierungsproblem in Standardform

$$\begin{array}{lrcl} & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & \text{unter den Nebenbedingungen} & Ax & = & b, \\ & & x & \geq & 0 \end{array}$$

aus, das folgende Bedingungen erfüllt:

(i) es existieren innere Punkte $x \in \overset{o}{M}$, d.h. die Menge M der Punkte x, für die gilt: $Ax = b$ und $x > 0$, ist nicht leer, und

(ii) $A \in I\!R^{m \times n}$ hat vollen Rang.

Mit Hilfe der *Barriere-Methode* (vgl. Abschnitt 1.6) wird das (für $\mu \longrightarrow 0$ äquivalente) Problem *ohne* Ungleichungen in den Nebenbedingungen gebildet. Die Nicht-Negativitätsbedingungen $x \geq 0$ aus (LP) werden durch den Barriere-Term $-\mu \sum_{i=1}^{n} \ln(x_i)$ in die Zielfunktion $f(x)$ aufgenommen und man erhält das nur noch durch lineare Nebenbedingungen in Gleichungsform beschränkte (LECP)

$$\begin{aligned} \text{minimiere} \quad & B(x \mid \mu) = c^T x - \mu \sum_{i=1}^{n} \ln(x_i) \\ \text{unter den Nebenbedingungen} \quad & Ax = b. \end{aligned} \tag{2.3}$$

Die Nebenbedingungen in Gleichungsform $Ax = b \Leftrightarrow Ax - b = 0$ werden eliminiert, indem man die unbeschränkte *Lagrangesche Funktion* für das Problem (2.3) konstruiert (vgl. Abschnitt 1.7)

$$L(x, y \mid \mu) = c^T x - \mu \sum_{i=1}^{n} \ln(x_i) - y^T (Ax - b), \tag{2.4}$$

wobei $y \in I\!R^m$ der Vektor der Lagrangeschen Multiplikatoren ist. Nach Lemma 1.7.1 gilt, daß für eine Minimalstelle $(\hat{x}, \hat{y})$ von (2.4) der Vektor $\hat{x}$ Minimallösung des (LECP) (2.3) und damit hier auch des (LP) ist. Die Optimalitätsbedingungen 1. Ordnung für den hier zu behandelnden Spezialfall eines Optimierungsproblems mit linearen Nebenbedingungen in Gleichungsform (2.3) lautet dann nach Lemma 1.7.3

$$\nabla L(x, y \mid \mu) = \begin{pmatrix} \nabla_x L \\ \nabla_y L \end{pmatrix} = \begin{pmatrix} c - \mu X^{-1} e - A^T y \\ -Ax + b \end{pmatrix} = 0. \tag{2.5}$$

wobei X eine Diagonalmatrix ist, deren Diagonalelemente die Variablen x_i sind und $e = 1$ ein entsprechend dimensionierter Vektor ist.

Ausgehend von der Hessematrix

$$\nabla^2 L(x, y \mid \mu) = \begin{pmatrix} \nabla^2_{xx} L & (\nabla^2_{xy} L)^T \\ \nabla^2_{yx} L & 0 \end{pmatrix}$$

mit

$$\begin{aligned} \nabla^2_{xx} L(x,y \mid \mu) &= \mu X^{-2}, \quad \text{und} \\ \nabla^2_{xy} L(x,y \mid \mu) &= \nabla^2_{yx} L(x,y \mid \mu) = A. \end{aligned}$$

erhält man ein $(m+n)$-dimensionales lineares Gleichungssystem zur Bestimmung der Newton-Korrektur (vgl. Anwendung des Newton-Verfahrens auf die Lagrangesche Funktion (1.30))

$$\begin{pmatrix} \mu (X^{(k)})^{-2} & A^T \\ A & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta x^{(k)} \\ \Delta y^{(k)} \end{pmatrix} = - \begin{pmatrix} c - \mu (X^{(k)})^{-1} e - A^T y^{(k)} \\ -Ax^{(k)} + b \end{pmatrix}.$$

Unter der Annahme, daß $x^{(k)}$ ein innerer Punkt des zulässigen Bereiches ist, d.h. $x^{(k)} > 0$ und $Ax^{(k)} = b$ für $(k = 0, 1, \ldots)$, kann dies umgeformt werden in

$$\begin{pmatrix} \mu (X^{(k)})^{-2} & A^T \\ A & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta x^{(k)} \\ \Delta y^{(k)} \end{pmatrix} = - \begin{pmatrix} c - \mu (X^{(k)})^{-1} e - A^T y^{(k)} \\ 0 \end{pmatrix}.$$

Führt man $y^{(k+1)} := y^{(k)} + \Delta y^{(k)}$ als neue Variable ein, so kann das Gleichungssystem in die Form

$$\begin{pmatrix} \mu (X^{(k)})^{-2} & A^T \\ A & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta x^{(k)} \\ -y^{(k+1)} \end{pmatrix} = \begin{pmatrix} -c + \mu (X^{(k)})^{-1} e \\ 0 \end{pmatrix} \tag{2.6}$$

gebracht werden.

Da die Hessematrix $\nabla^2_{xx} L = \mu (X^{(k)})^{-2}$ wegen der Voraussetzung $x^{(k)} > 0$ für $(k = 0, 1, \ldots)$ nicht-singulär ist, kann die 1. Teilgleichung aus (2.6) mit $(\nabla^2_{xx} L)^{-1} = \frac{1}{\mu}(X^{(k)})^2$ multipliziert werden, so daß man die Gleichungen

$$\Delta x^{(k)} = \frac{1}{\mu}(X^{(k)})^2 \left(A^T y^{(k+1)} - c + \mu (X^{(k)})^{-1} e \right) \quad \text{bzw.} \tag{2.7a}$$

$$\Delta x^{(k)} = x^{(k)} + \frac{1}{\mu}(X^{(k)})^2 (A^T y^{(k+1)} - c) \tag{2.7b}$$

erhält. Der Lagrangesche Multiplikator $y^{(k+1)}$ wird bestimmt, indem die 1. Teilgleichung aus (2.6) mit $A(X^{(k)})^2$ multipliziert wird und die Voraussetzung $Ax^{(k+1)} = A(x^{(k)} + \Delta x^{(k)}) = b \Rightarrow A\Delta x^{(k)} = 0$ für $(k = 0, 1, \ldots)$ ausgenutzt wird:

$$A(X^{(k)})^2 A^T y^{(k+1)} = A(X^{(k)})^2 c - \mu A X^{(k)} e = A X^{(k)} (X^{(k)} c - \mu e). \tag{2.8}$$

Da A nach Voraussetzung (ii) vollen Rang hat und $x^{(k)} \neq 0$ ist, ist die Matrix $A(X^{(k)})^2 A^T$ positiv definit [64]. Löst man (2.8) nach $y^{(k+1)}$ auf und setzt dies in (2.7a) ein, so erhält man

$$\Delta x^{(k)} = -\frac{1}{\mu} X^{(k)} P X^{(k)} c + X^{(k)} P e \tag{2.9}$$

mit

$$P = I - X^{(k)} A^T \left(A X^{(k)} X^{(k)} A^T \right)^{-1} A X^{(k)}, \tag{2.10}$$

so daß die neue Iterierte $x^{(k+1)}$ bestimmt wird als

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} \Delta x^{(k)} \tag{2.11}$$

für einen geeigneten Schrittweitenparameter $\alpha^{(k)}$. Entsprechend der in Abschnitt 1.6 beschriebenen Barriere-Methode wird für den nächsten Iterationsschritt der Barriere-Parameter $\mu^{(k)}$ auf $\mu^{(k+1)} = \tau \mu^{(k)}$, mit $0 < \tau < 1$, reduziert und so der Algorithmus fortgeführt.

Die Äquivalenz zwischen der Newton-Korrektur (2.9) zur Bestimmung der Abstiegsrichtung und dem Richtungsvektor δ des Karmarkar-Algorithmus (2.1) wird von Gill et al. [19] mit folgendem Satz gezeigt.

Satz 2.2.1 *Wird der nachfolgend beschriebene primale Newton-Barriere-Algorithmus (vgl. Algorithmus 2.2.1) mit der Abstiegsrichtung (2.9) auf ein lineares Optimierungsproblem in Karmarkars Form (KarP) angewandt, so existiert für jeden Iterationsschritt ein μ, so daß (2.1) und (2.9) identisch sind.*

Mit Hilfe dieses Satzes ist der Nachweis gegeben, daß der Karmarkar-Algorithmus ein Spezialfall der allgemeinen logarithmischen Barriere-Methoden ist, die in diesem Abschnitt als die primale Innere-Punkt-Methode bezeichnet wurde.

Algorithmische Beschreibung

Eine algorithmische Darstellung der primalen Innere-Punkt-Methode beinhaltet eine „äußere“ und eine „innere“ Iteration. In der äußeren Iteration (Iterationsindex l) wird der Barriere-Parameter μ sukzessive verkleinert, während in der inneren Iteration (Index k) der Newton-Algorithmus verwandt wird, um das Minimum des aktuellen Barriere-Problems, d.h. der Lagrangeschen Barriere-Funktion (2.4), anzunähern.

Algorithmus 2.2.1: PrimalNewtonBarriereMethod $(c, A, b, x^{(0)}, \hat{x}^{(l)})$

1. {Initialisierung.}
   $l := -1$; $l_{\max} := 100$; $\varepsilon := 10^{-10}$; $\delta := 10^{-50}$; *wähle* $\mu^{(0)} > 0$;
2. {Prüfe, ob $x^{(0)}$ ein strikt zulässiger Punkt ist, d.h.}
   $\{x^{(0)} > 0 \quad \text{und} \quad Ax^{(0)} = b. \qquad \}$
   CheckFeasibility $(A, b, x^{(0)})$;
3. {Starte die Barriere-Iteration (vgl. Algorithmus 1.6.1).}
   repeat {Barriere-Iteration mit Index l.}
   (a) $l := l + 1$; $k := -1$;
   (b) **if** $l > 0$ **then** wähle $0 < \mu^{(l)} < \mu^{(l-1)}$;
   (c) {Bilde die (neue) Lagrangesche Barriere-Funktion.}
   $L(x, y \mid \mu^{(l)}) := c^T x - \mu^{(l)} \sum_{i=1}^{n} \ln(x_i) - y^T(Ax - b)$;
   (d) {Bestimme $(\hat{x}^{(l)}, \hat{y}^{(l)})^T$, das Minimum von $L(x, y \mid \mu^{(l)})$ mit}
   {dem Newton-Verfahren (vgl. Algorithmus 1.9.1). }
   NewtonMethod $(L, (x^{(l,0)}, y^{(l,0)}), (x^{(l,k+1)}, y^{(l,k+1)}), Err)$;
   (e) $\hat{x}^{(l)} := x^{(k+1,l)}$; $\hat{y}^{(l)} := y^{(k+1,l)}$;
   until $(\|\hat{x}^{(l)} - \hat{x}^{(l-1)}\| < \varepsilon)$ **or** $(\mu^{(l)} < \delta)$ **or** $l = l_{\max}$;
4. **return** $\hat{x}^{(l)}$;

Für die in Schritt 2 verwandte Prozedur CheckFeasibility zur Überprüfung der Zulässigkeit der Startlösung wird auf die Beschreibung in Abschnitt 2.4.1 verwiesen.

2.3 Primal-duale Innere-Punkt-Methode

2.3.1 Dualitätssätze

Einer Minimierungsaufgabe der linearen Optimierung (LP) kann eine duale Maximierungsaufgabe zugeordnet werden [10]. Der Zusammenhang zwischen primalem und dualem Problem kann für die Bestimmung eines numerisch überprüfbaren Kriteriums für die Qualität der Lösung genutzt werden. Gegeben seien das primale Optimierungsproblem

$$\begin{array}{lrcl} & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & \text{unter den Nebenbedingungen} & Ax & = & b, \\ & & x & \geq & 0, \end{array}$$

und das dazu duale Optimierungsproblem

$$\text{(DP')} \qquad \begin{array}{rrcl} & \text{maximiere} & g(y) & = & b^T y \\ & \text{unter den Nebenbedingungen} & A^T y & \leq & c, \end{array}$$

Es ist zu beachten, daß (DP') nicht vorzeichenbeschränkt ist.

Definition 2.3.1 Durch die Einführung von Schlupfvariablen z_i, $(i = 1, \ldots, n)$ in (DP') erhält man das zu (DP') äquivalente *duale Optimierungsproblem* in Standardform

$$\text{(DP)} \qquad \begin{array}{rrcl} & \text{maximiere} & g(y) & = & b^T y \\ & \text{unter den Nebenbedingungen} & A^T y + z & = & c, \\ & & z & \geq & 0. \end{array}$$

Es gelten nun die folgenden *Dualitätssätze*.

Satz 2.3.1 *Ist x zulässiger Punkt des (LP) und y zulässiger Punkt des (DP), so gilt:*

$$f(x) \geq g(y).$$

Satz 2.3.2 *Das (LP) besitzt genau dann eine endliche Lösung, wenn das (DP) eine Lösung besitzt. Die Extremwerte beider Probleme sind (wenn sie existieren) identisch, d.h. $f(\hat{x}) = g(\hat{y})$.*

Satz 2.3.3 *Besitzen beide Optimierungsprobleme, das (LP) und das (DP), zulässige Punkte, so besitzen auch beide Probleme Optimallösungen.*

Zum Beweis siehe Collatz und Wetterling [10].

Aus den Sätzen 2.3.1 und 2.3.3 folgt, daß man eine beidseitige Einschließung des Optimalwertes $f(\hat{x}) = g(\hat{y})$ durch das Paar aus primalem und dualem Optimierungsproblem erhält.

$$g(y^{(k)}) \leq g(\hat{\mu}) = f(\hat{x}) \leq f(x^{(k)}), \quad (k = 0, 1, \ldots)$$

Dies führt zu der Idee, (LP) und (DP) simultan zu lösen, da so in jedem Schritt des Iterationsprozesses eine Einschließung des tatsächlichen Optimalwertes geliefert wird, über deren Durchmesser eine Abschätzung der Qualität der Lösung möglich ist.

Definition 2.3.2 Der Abstand der Zielfunktionswerte eines zulässigen Punktes des primalen Problems $x^{(k)}$ und eines zulässigen Punktes des dualen Problems $y^{(k)}$ wird als

$$gap^{(k)} := f(x^{(k)}) - g(y^{(k)})$$

definiert und mit *Dualitätslücke* bezeichnet.

2.3.2 System aus primalem und dualem Optimierungsproblem

Betrachtet wird nun das Paar aus *primalem* und *dualem Programm* in Standardform:

$$\begin{array}{rrrcl} & & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & & \text{unter den Nebenbedingungen} & Ax & = & b, \\ & & & x & \geq & 0, \end{array}$$

und

$$\begin{array}{rrrcl} & & \text{maximiere} & g(y) & = & b^T y \\ \text{(DP)} & & \text{unter den Nebenbedingungen} & A^T y + z & = & c, \\ & & & z & \geq & 0, \end{array}$$

das folgende Bedingungen erfüllt:

(i) es existieren strikt innere Punkte für das LP, d.h. die Menge der Punkte x, für die gilt: $Ax = b$ und $x > 0$, ist nicht leer,

(ii) es existieren strikt innere Punkte für das DP, d.h. die Menge der Punkte (y, z), für die gilt: $A^T y + z = c$ und $z > 0$, ist nicht leer, und

(iii) $A \in I\!R^{m \times n}$ hat vollen Rang.

Die entsprechenden *logarithmischen Barriere-Funktionen* lauten (analog zum primalen Fall (2.3)):

$$\begin{array}{rl} \text{minimiere} & B_{LP}(x \mid \mu) = c^T x - \mu \sum_{i=1}^{n} \ln(x_i) \\ \text{unter den Nebenbedingungen} & Ax = b. \end{array}$$

und

$$\begin{array}{rl} \text{maximiere} & B_{DP}(y, z \mid \mu) = b^T y + \mu \sum_{i=1}^{n} \ln(z_i) \\ \text{unter den Nebenbedingungen} & A^T y + z = c. \end{array}$$

Die *unbeschränkten Lagrangeschen Funktionen* sind dann (analog zu (2.4))

$$L_{LP}(x, y \mid \mu) = c^T x - \mu \sum_{i=1}^{n} \ln(x_i) - y^T (Ax - b) \tag{2.12a}$$

$$L_{DP}(x, y, z \mid \mu) = b^T y + \mu \sum_{i=1}^{n} \ln(z_i) - x^T (A^T y + z - c). \tag{2.12b}$$

Es bezeichnen X und Z Diagonalmatrizen, deren Diagonalelemente die Komponenten der Vektoren x und z sind. Durch die Berechnung aller partieller Ableitungen, die gleich Null gesetzt werden, erhält man zunächst (vgl. (2.5))

$$\begin{aligned}
\nabla_x L_{LP}(x, y \mid \mu) &= c - \mu X^{-1} e - A^T y \\
\nabla_y L_{LP}(x, y \mid \mu) &= -Ax + b \\
\nabla_x L_{DP}(x, y, z \mid \mu) &= -(A^T y + z - c) \\
\nabla_y L_{DP}(x, y, z \mid \mu) &= b - Ax \\
\nabla_z L_{DP}(x, y, z \mid \mu) &= \mu Z^{-1} e - Xe.
\end{aligned}$$

Durch einige algebraische Umformungen von

$$\begin{aligned}
c - \mu X^{-1} e - A^T y &= 0 \quad \mid \nabla_x L_{DP} : -A^T y = z - c \\
c - \mu X^{-1} e + z - c &= 0 \\
\mu Z^{-1} e - Xe &= 0 \quad \mid \nabla_z L_{DP} \\
-\mu X^{-1} e + z &= 0
\end{aligned}$$

bzw. aufgrund von Dublizität ergeben sich drei Gleichungssysteme

$$Ax = b, \quad x > 0 \tag{2.13a}$$

$$A^T y + z = c, \quad z > 0 \tag{2.13b}$$

$$XZe = \mu e. \tag{2.13c}$$

Die Bedingungen (2.13a) und (2.13b) geben die übliche primale und duale Zulässigkeit an und (2.13c) entspricht für $\mu \longrightarrow 0$ der Dualitätslücke

$$\begin{aligned}
gap &= f(x) - g(y) \\
&= c^T x - b^T y \\
&= (A^T y + z)^T x - (Ax)^T y \\
&= z^T x.
\end{aligned}$$

Dieses Gesamtsystem entspricht der Optimalitätsbedingung 1. Ordnung (vgl. (1.28))

$$\nabla(L_{LP}, L_{DP}) = \begin{pmatrix} -Ax + b \\ -A^T y - z + c \\ XZe - \mu e \end{pmatrix} = 0 \tag{2.14}$$

deren Nullstelle mit dem Newton-Verfahren bestimmt werden kann.

Der Startpunkt $(x^{(0)}, y^{(0)}, z^{(0)})$ erfülle die Bedingungen $Ax^{(0)} = b$, $A^T y^{(0)} + z^{(0)} = c$, $x^{(0)} > 0$ und $z^{(0)} > 0$, d.h. er sei ein strikt zulässiger Punkt.

Mit der Hesse-Matrix

$$\nabla^2(L_{LP}, L_{DP}) = \begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix}, \tag{2.15}$$

die sofort angegeben werden kann, erhält man die *Newton-Iteration* (1.30) als

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} r_{LP} \\ r_{DP} \\ r_\mu \end{pmatrix}, \tag{2.16}$$

wobei

$$\begin{aligned} r_{LP} &= b - Ax, \\ r_{DP} &= -A^T y - z + c \text{ und} \\ r_\mu &= \mu e - XZe \end{aligned}$$

den *primalen* bzw. *dualen Residuenvektor* und die *Dualitätslücke* bezeichne. Aus der zweiten Zeile des Gleichungssystems (2.16) folgt

$$\Delta z = r_{DP} - A^T \Delta y.$$

Durch Substitution von Δz in der dritten Gleichung erhält man

$$\begin{aligned} Z\Delta x + X\Delta z &= r_\mu \\ Z\Delta x + X(r_{DP} - A^T\Delta y) &= \mu e - XZe \\ X^{-1}Z\Delta x + r_{DP} - A^T\Delta y &= \mu X^{-1}e - Ze \\ X^{-1}Z\Delta x - A^T\Delta y &= \mu X^{-1}e - z - r_{DP} \\ &= \mu X^{-1}e - z + A^T y + z - c \\ &= \mu X^{-1}e - c + A^T y \end{aligned}$$

und damit für Δx und Δy das System

$$\begin{pmatrix} A & 0 \\ X^{-1}Z & A^T \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta y \end{pmatrix} = \begin{pmatrix} r_{LP} \\ \mu X^{-1}e - c + A^T y \end{pmatrix}. \tag{2.17}$$

Da x und z echt positive Vektoren sind, ist die Matrix $X^{-1}Z$ eine reguläre Diagonalmatrix. Verwendet man $(X^{-1}Z)^{-1}$ zur Eliminierung von Δx in der zweiten Gleichung aus (2.17), so folgt

$$(AZ^{-1}XA^T)\Delta y = b - \mu AZ^{-1}e + AZ^{-1}Xr_{DP}. \tag{2.18}$$

Um (2.16) zu lösen, genügt es also, zunächst Δy durch das lineare Gleichungssystem (2.18) zu bestimmen und anschließend

$$\begin{aligned} \Delta z &= r_{DP} - A^T \Delta y \quad \text{und} \\ \Delta x &= Z^{-1}(r_\mu - X\Delta z) \end{aligned}$$

direkt zu berechnen. Dies hat insbesondere zur Folge, daß der Berechnungsaufwand für die einzelne Newton-Korrektur der primal-dualen Innere-Punkt-Methode gegenüber der primalen Methode praktisch unverändert bleibt.

Vergleicht man die Beschreibungen von Karmarkar-Algorithmus, primaler und primal-dualer Innere-Punkt-Methode, so ist zu erkennen, daß in allen Fällen eine Matrix von Typ ADA^T konstruiert wird, wobei D eine Diagonalmatrix ist. Die Komponenten der Matrix D unterscheiden sich zwar von Verfahren zu Verfahren, der Rechenaufwand eines Newton-Schrittes allerdings nur unwesentlich. Ausgehend von dieser Ähnlichkeit lassen sich sofort zwei Vorteile der primal-dualen Methode erkennen.

Zum einen ist für zulässige Vektoren x und y bzw. z die aktuelle Dualitätslücke $gap = c^T x - b^T y$ stets bekannt und es gilt

$$gap = c^T x - b^T y = (A^T y + z)^T x - (Ax)^T y = z^T x \geq 0, \tag{2.19}$$

da $A^T y + z = c$ aus (DP) und $Ax = b$ aus (LP) gilt, sowie Dualitätssatz 2.3.1 anwendbar ist. Dies liefert in jedem Iterationsschritt eine berechenbare Grenze für die Nähe von $c^T x$ zum optimalen Wert $c^T \hat{x}$. Sei $c^T x - b^T y = \beta$. Da nach der Aussage von Dualitätssatz 2.3.2 gilt $c^T \hat{x} = b^T \hat{y}$, erhält man

$$c^T x - c^T \hat{x} = \beta + b^T y - b^T \hat{y}.$$

Da weiterhin bekannt ist, daß die Zielfunktion $g(y)$ von (DP) durch $\hat{y}$ maximiert wird, gilt $b^T \hat{y} \geq b^T y$ für alle dual zulässigen Vektoren y, d.h. $b^T y - b^T \hat{y} \leq 0$.

Entsprechend gilt für alle primal zulässigen Vektoren x: $c^T x - c^T \hat{x} \geq 0$. Hieraus folgt, daß die numerisch exakt berechenbare Dualitätslücke β für ein Paar x, y eine obere Grenze für den Abstand zum optimalen Zielfunktionswert angibt:

$$0 \leq c^T x - c^T \hat{x} \leq \beta$$

Der zweite Vorteil der primal-dualen Methode ist die Möglichkeit, den Schrittweitenparameter α im Newton-Schritt für das primale (α_{LP}) und das

duale Problem (α_{DP}) getrennt zu wählen, d.h. der Newton-Schritt hat die Form

$$\begin{aligned} x^{(k+1)} &:= c^{(k)} + \alpha_{LP}^{(k)} \Delta x^{(k)} \\ y^{(k+1)} &:= y^{(k)} + \alpha_{DP}^{(k)} \Delta y^{(k)} \\ z^{(k+1)} &:= z^{(k)} + \alpha_{DP}^{(k)} \Delta z^{(k)}. \end{aligned}$$

Wie in Abschnitt 2.4.2 näher beschrieben wird, ist dadurch eine unabhängige Steuerung der Newton-Korrekturen bezüglich der primalen und dualen Näherungslösungen möglich, die das Verfahren erheblich beschleunigt.

2.4 Verifikation

2.4.1 Zulässigkeit des Startpunktes $x^{(0)}$ und der Ierierten

Sowohl die primale Innere-Punkt-Methode (vgl. Abschnitt 2.2) als auch die primal-duale Innere-Punkt-Methode (vgl. Abschnitt 2.3) setzen für die erfolgreiche Verwendung des Verfahrens die Existenz eines strikt zulässigen Startpunktes $x^{(0)}$ bzw. $(x^{(0)}, y^{(0)}, z^{(0)})$ voraus, d.h. eines Punktes, der echt im Innern des zulässigen Bereiches von (LP) bzw. (LP) und (DP) liegt. Dies bedeutet im primalen Fall (2.13a)

$$Ax = b \quad \text{und} \quad x > 0$$

sowie im dualen Fall (2.13b)

$$A^T y + z = c \quad \text{und} \quad z > 0.$$

Da auf einem Rechner im Rahmen der Maschinenzahlen i.a. keine Gleichheit ($Ax = b$) erreicht werden kann (siehe Beispiel 2.4.1), muß für den Nachweis der Zulässigkeit eines Punktes x zu dem äquivalenten Grundproblem (GrundLP) (vgl. Abschnitt 1.4) übergegangen werden. Hierzu wird davon ausgegangen, daß das im Algorithmus behandelte (LP) in seiner Standardform aus einem gegebenen Grundproblem (GrundLP) durch die Einführung von Schlupfvariablen $s_1, \ldots, s_m$ (vgl. Abschnitt 1.4) hervorgegangen ist:

$$\begin{array}{lrrl} & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & \text{unter den Nebenbedingungen} & Ax & = & b \\ & \text{und} & x & \geq & 0 \end{array}$$

mit $x = (\widetilde{x}_1, \ldots, \widetilde{x}_n, s_1, \ldots s_m)^T$, $c = (\widetilde{c}_1, \ldots, \widetilde{c}_n, 0, \ldots, 0)$ und $A = (\widetilde{A}, I_m)$, wobei I_m die m-dimensionale Einheitsmatrix bezeichne. Das duale Problem lautet dann

$$\begin{array}{lrcl} & \text{maximiere} & g(y) & = & b^T y \\ \text{(DP)} & \text{unter den Nebenbedingungen} & A^T y + z & = & c \\ & \text{und} & z & \geq & 0, \end{array}$$

wobei $z = (\widetilde{z}_1, \ldots, \widetilde{z}_n, t_1, \ldots, t_m)$ der duale Schlupfvariablenvektor sei. Wird nun das Grundproblem

$$\begin{array}{lrcl} & \text{minimiere} & f(\widetilde{x}) & = & \widetilde{c}^T \widetilde{x} \\ \text{(GrundLP)} & \text{unter den Nebenbedingungen} & \widetilde{A}\widetilde{x} & \leq & b \\ & \text{und} & \widetilde{x} & \geq & 0, \end{array}$$

für die Überprüfung der strikten Zulässigkeit eines Startpunktes $(x^{(0)}, y^{(0)}, z^{(0)})$ für das (LP) verwandt, so ist dies vollständig durch die Überprüfung von Ungleichungen möglich. Für den Nachweis der strikten Zulässigkeit im primalen Fall sind dann nämlich die Bedingungen:

$$\begin{array}{rl} x > 0 & \text{für den erweiterten Vektor } \tilde{x} \text{ und} \\ \widetilde{A}\widetilde{x} \leq b & \text{für das nicht-erweiterte Grundproblem} \end{array}$$

sowie für den dualen Fall

$$\begin{array}{rl} z > 0 & \text{für den erweiterten Vektor } \tilde{z} \text{ und} \\ \widetilde{A}^T y \leq \widetilde{c} & \text{für das nicht-erweiterte Grundproblem} \end{array}$$

gleichbedeutend mit (2.13a) und (2.13b).

Zur Erläuterung der Problematik wird folgendes Beispiel angegeben:

Beispiel 2.4.1 Geben sei das (LP)

$$\begin{array}{lrcl} & \text{minimiere} & f(x) & = & (1,1,1,0,0) \cdot x \\ \text{(LP)} & \text{u.d.NB} & \begin{pmatrix} 1 & 2 & 0 & 0.3 & 0 \\ 1 & 1 & 1 & 0 & 0.5 \end{pmatrix} \cdot x & = & \begin{pmatrix} 3 \\ 3 \end{pmatrix} \\ & \text{und} & x & \geq & 0. \end{array}$$

Ausgehend von einer 2-stelligen Dezimalarithmetik sind folgende Zwischenergebnisse möglich:

$$x^{(1)} = \begin{pmatrix} 1.7 \\ 1 \\ 1 \\ 1 \\ 0.6 \end{pmatrix} > 0 \quad \text{und} \quad \begin{pmatrix} 1 & 2 & 0 & 0.3 & 0 \\ 1 & 1 & 1 & 0 & 0.5 \end{pmatrix} \cdot x^{(1)} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

aber wegen $x_4^{(2)} = \square(2/3) = 0.67 \neq \frac{2}{3}$

$$x^{(2)} = \begin{pmatrix} 1.8 \\ 1 \\ 1 \\ 0.67 \\ 0.4 \end{pmatrix} > 0 \quad \text{und} \quad \begin{pmatrix} 1 & 2 & 0 & 0.3 & 0 \\ 1 & 1 & 1 & 0 & 0.5 \end{pmatrix} \cdot x^{(2)} \neq \begin{pmatrix} 3 \\ 3 \end{pmatrix}.$$

Dies liegt daran, daß die Zahl $\frac{2}{3}$ in einer 2-stelligen Dezimalarithmetik nicht darstellbar ist. Wird jedoch auf das Grundproblem

$$\text{(GrundLP)} \qquad \begin{array}{lrcl} \text{minimiere} & f(\tilde{x}) & = & (1,1,1) \cdot \tilde{x} \\ \text{u.d.NB.} & \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \tilde{x} & \leq & \begin{pmatrix} 3 \\ 3 \end{pmatrix} \\ \text{und} & \tilde{x} & \geq & 0. \end{array}$$

übergegangen, so können

$$x^{(1)} > 0 \Rightarrow \tilde{x}^{(1)} = \begin{pmatrix} 1.7 \\ 1 \\ 1 \end{pmatrix} > 0 \quad \text{und} \quad \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \tilde{x}^{(1)} \leq \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

und

$$x^{(2)} > 0 \Rightarrow \tilde{x}^{(2)} = \begin{pmatrix} 1.8 \\ 1 \\ 1 \end{pmatrix} > 0 \quad \text{und} \quad \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \tilde{x}^{(2)} \leq \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

als zulässige Punkte identifiziert werden.

Praktisch kann die Überprüfung der Zulässigkeit einer Iterierten auf dem Rechner nur garantiert werden, wenn eine Intervallarithmetik zur Verfügung steht, die alle auftretenden Rundungsfehler berücksichtigt. Mit Hilfe der gerichteten Rundung für arithmetische Operationen ist auch die Vergleichsoperation, wie sie zur Überprüfung der abgewandelten Gleichungsbedingungen $Ax \leq b$ und $A^T y \leq c$ benötigt werden, auf dem Rechner entscheidbar. Steht zusätzlich ein exaktes Skalarprodukt zur Verfügung, das ein Ergebnis mit nur einer Rundung liefert, so ist eine Minimierung der Überschätzung bei der Berechnung des Matrix-Vektor-Produktes möglich.

Diese Ungleichungsbedingungen sind somit auch auf dem Rechner im Rahmen der Maschinenzahlen erfüllbar und für die Verifikation der Zulässigkeit der Startlösung verwendbar. Die Prozedur zur Überprüfung der Zulässigkeit kann dann in der folgenden Form algorithmisch beschrieben werden:

Algorithmus 2.4.1: CheckFeasibility $(A, b, c, x, y, z, primal, dual)$

1. {Prüfe Nicht-Negativitätsbedingungen}
   $primal := x > 0$;
   $dual := z > 0$;
2. {Prüfe „Gleichungsbedingungen“}
   if *primal* **or** *dual* **then**
   begin {Bestimme Teilmatrix und Teilvektoren}
   $\widetilde{A} := A_{1..m,1..n-m}$; $\widetilde{x} := x_{1..n-m}$;
   $\widetilde{c} := c_{1..n-m}$;
   $primal := \widetilde{A}\widetilde{x} \leq b$;
   $dual := \widetilde{A}^T y \leq \widetilde{c}$;
   end
3. **return** *primal*, *dual*;

Da es ein Charakteristikum der Innere-Punkt-Methoden ist, daß der Iterationsprozeß vollständig im Innern des zulässigen Bereiches verläuft, müssen die oben angegebenen Bedingungen für die primale und duale Zulässigkeit auch in jedem Barriere-Schritt und sogar in jedem Newton-Schritt, der Teil des Barriere-Schrittes ist, erfüllt sein. D.h. die Prozedur CheckFeasibility wird nicht nur zur Überprüfung des Startpunktes $(x^{(0)}, y^{(0)}, z^{(0)})$ benötigt, sondern auch im Verlauf des Iterationsprozesses für alle $(x^{(k)}, y^{(k)}, z^{(k)})$, z.B. zur Steuerung der Schrittweite im Newton-Verfahren (vgl. Algorithmus 2.4.2).

2.4.2 Steuerung der Parameter α und μ

Um das Wechselspiel zwischen dem Parameter α für den Newton-Schritt (2.16) und dem Barriere-Parameter μ zu beschreiben, wird zunächst die primal-affine Methode dargestellt.

Definition 2.4.1 Die *primal-affine Methode* stellt eine Abwandlung der in Abschnitt 2.3 beschriebenen primalen logarithmischen Barriere-Methode für lineare Optimierungsprobleme dar, in der die Newton-Korrektur $\Delta x^{(k)}$ in (2.9) durch

$$\Delta x^{(k)} = -X^{(k)} P X^{(k)} c \tag{2.20}$$

ersetzt wird. Die Richtung des Vektors $\Delta x^{(k)}$ entspricht dem Grenzwert der Richtung der Newton-Korrektur aus (2.9) für $\mu^{(k)} \longrightarrow 0$.

Eine entscheidende Voraussetzung für Innere-Punkt-Methoden ist, daß der Iterationsprozeß im Inneren des zulässigen Bereiches stattfindet. Dies wird durch den Barriere-Parameter μ beeinflußt. Es ist naheliegend, daß eine Wahl von $\alpha^{(k)}$ in (2.11), die den Newton-Schritt möglichst nahe an den Rand des zulässigen Bereich führt, am effizientesten ist.

Geht man von dem Problem

$$\begin{aligned} \text{minimiere} \quad & f(x) &=& -\sum_{j=1}^{n} \ln(x_j) \\ \text{unter den Nebenbedingungen} \quad & Ax &=& b \end{aligned}$$

aus, dessen Optimalstelle das (analytische) Zentrum des zulässigen Bereichs ist, so führt eine Anwendung der Newton-Methode auf die Optimalitätsbedingung 1. Ordnung zu

$$\Delta x^{(k)} = X^{(k)} P e \tag{2.21}$$

mit P aus (2.10).

Dies zeigt, daß der Newton-Schritt (2.9) der primalen logarithmischen Barriere-Methode aus einem zentrierenden Term (2.21), der dem Erreichen des Randes entgegenwirkt, und einem affinen Term (2.20), der zur optimalen Lösung führt, besteht.Während für große $\mu^{(k)}$ der Iterationsprozeß aufgrund des zentrierenden Anteils durch das Innere des zulässigen Bereichs führt, bewegt sich der Prozeß für $\mu^{(k)} \longrightarrow 0$ in Richtung der optimalen Lösung.

2.4.2.1 Parameter α für die Newton-Iteration

Nach der Berechnung der Newton-Korrektur in (2.16) mit Δx in x-Richtung und $(\Delta y, \Delta z)^T$ in Richtung der y-z-Ebene ist der Erhalt der Zulässigkeit der neuen Iterierten

$$\begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \\ z^{(k+1)} \end{pmatrix} := \begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \end{pmatrix} + (\alpha_{LP}, \alpha_{DP}, \alpha_{DP}) \cdot \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$

(vgl. Definition 1.9.2) durch eine geeignete Wahl der Schrittweitenparameter α_{LP} und α_{DP} zu garantieren. Die Wahl der Parameter wird bestimmt durch den größtmöglichen Wert, bei dem der Rand des zulässigen Bereiches erreicht wird, d.h. bei dem eine der Komponenten von x oder z zu Null wird. Mit $\hat{\alpha}_{LP}$ und $\hat{\alpha}_{DP}$ seien diese größtmöglichen Werte für die Parameter bezeichnet:

$$\hat{\alpha}_{LP} := \min_i \left\{ -\frac{x_i^{(k)}}{\Delta x_i} \;\middle|\; \Delta x_i < 0 \right\} \quad \text{und} \tag{2.22a}$$

$$\hat{\alpha}_{DP} := \min_i \left\{ -\frac{z_i^{(k)}}{\Delta z_i} \;\middle|\; \Delta z_i < 0 \right\}. \tag{2.22b}$$

Um ein Verlassen des Inneren des zulässigen Bereiches zu verhindern, wird der tatsächliche Wert für die Schrittweiten allerdings kleiner gewählt:

$$\alpha_{LP} = \tau \hat{\alpha}_{LP} \quad \text{und} \quad \alpha_{DP} = \tau \hat{\alpha}_{DP}$$

mit $0 \leq \tau < 1$.

Die Überprüfung der Zulässigkeit aller Iterierten $(x^{(k)}, y^{(k)}, z^{(k)})$, $(k = 0, 1, \ldots)$ wird auch im Rahmen der Newton-Iteration (wie in Abschnitt 2.4.1 beschrieben) durch die Prozedur CheckFeasibility aus Algorithmus 2.4.1 übernommen. Im Laufe zahlreicher numerischer Tests des hier vorgestellten Verfahrens hat es sich als sinnvoll erwiesen, die Parameter $\hat{\alpha}_{LP}$ und $\hat{\alpha}_{DP}$ zunächst um einen Faktor $\tau = 0.9$ zu verringern, falls ein maximaler Newton-Schritt auf dem Rechner zum Verlassen des zulässigen Bereiches führt. Dieser Vorgang wird maximal fünf mal wiederholt, bevor α_{LP} bzw. α_{DP} gleich Null gesetzt wird. Falls weder in der x-Richtung des primalen Problems noch in der y-z-Ebene des dualen Problems ein zulässiger Newton-Schritt möglich ist, d.h. $\alpha_{LP} = \alpha_{DP} = 0$, so muß das gesamte Verfahren der primal-dualen Innere-Punkt-Methode abgebrochen werden, da eine weitere Verbesserung des Ergebnisses bei gleichzeitiger Garantie der Zulässigkeit der Iterierten nicht möglich ist.

Algorithmisch läßt sich das beschriebene Vorgehen folgendermaßen darstellen.

Algorithmus 2.4.2: FeasibleNewtonStep $(A, b, c, x^{(k)}, y^{(k)}, z^{(k)}, \Delta x, \Delta y, \Delta z, x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, Err)$

1. {Initialisierung.}
   $i_{\max} := 5$; $\tau := 0.9$; $Err :=$ „No Error“;
2. $x^{(k+1)} := x^{(k)} + \Delta x$; $y^{(k+1)} := y^{(k)} + \Delta y$; $z^{(k+1)} := z^{(k)} + \Delta z$;
3. {Prüfe die primale und duale Zulässigkeit der neuen Iterierten.}
   CheckFeasibility $(A, b, c, x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, primal, dual)$;
4. {Behandlung der primalen Zulässigkeit (vgl. (2.13a)).}
   if not *primal* **then**
   begin
   {Bestimme die maximal zulässige Newton-Korrektur.}
   $i := 0$; $\alpha_{LP} := \min\{-\frac{x_i^{(k)}}{\Delta x_i} \mid \Delta x_i < 0\}$;

while (**not** *primal*) **and** $(i \leq i_{\max})$ **do**
begin
$i := i + 1$;
$x^{(k+1)} := x^{(k)} + \alpha_{LP} \cdot \Delta x$;
CheckFeasibility $(A, b, c, x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, primal, dual)$;
{Verringere die Newton-Korrektur.}
$\alpha_{LP} := \tau \cdot \alpha_{LP}$;
end;
if not *primal* **then**
begin $\alpha_{LP} := 0$; $x^{(k+1)} := x^{(k)}$; **end**;
end;

5. {Behandlung der dualen Zulässigkeit (vgl. (2.13b)).}
if not *dual* **then**
begin
{Bestimme die maximal zulässige Newton-Korrektur.}
$i := 0$; $\alpha_{DP} := \min\{-\frac{z_i^{(k)}}{\Delta z_i} \mid \Delta z_i < 0\}$;
while (**not** *dual*) **and** $(i \leq i_{\max})$ **do**
begin
$i := i + 1$;
$y^{(k+1)} := y^{(k)} + \alpha_{DP} \cdot \Delta y$;
$z^{(k+1)} := z^{(k)} + \alpha_{DP} \cdot \Delta z$;
CheckFeasibility $(A, b, c, x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, primal, dual)$;
{Verringere die Newton-Korrektur.}
$\alpha_{DP} := \tau \cdot \alpha_{DP}$;
end;
if not *dual* **then**
begin $\alpha_{DP} := 0$; $y^{(k+1)} := y^{(k)}$; $z^{(k+1)} := z^{(k)}$; **end**;
end;

6. **if** $(\alpha_{LP} = 0)$ **and** $(\alpha_{DP} = 0)$
then *Err* := „No feasible Newton-step possible“

7. **return** $x^{(k+1)}$, $y^{(k+1)}$, $z^{(k+1)}$, *Err*;

2.4.2.2 Parameter μ für die Barriere-Iteration

Für den erfolgreichen Einsatz des Barriere-Verfahrens (vgl. Algorithmus 1.6.1 in Abschnitt 1.6) ist bei der Wahl des Barriere-Parameters $\mu^{(k)}$ lediglich zu beachten, daß es sich um eine streng monoton fallende Folge $\mu^{(k)} \longrightarrow 0$ handelt. Um allerdings zu garantieren, daß der Iterationsprozeß

der primal-dualen Innere-Punkt-Methode in jedem Schritt zu einer Verbesserung des Ergebnisses, d.h. zu einer Reduktion der Dualitätslücke führt, muß $(\mu^{(k)})_{k=0,1,\ldots}$ die im folgenden hergeleiteten Bedingungen erfüllen.

Bezeichne $gap(0)$ die aktuelle Dualitätslücke im k-ten Schritt der Iteration.

$$\begin{aligned} gap(0) &= c^T x^{(k)} - b^T y^{(k)} \\ &= (z^{(k)})^T x^{(k)}. \end{aligned}$$

Sei weiter $\alpha = \min\{\alpha_{LP}, \alpha_{DP}\} > 0$. Dann erhält man für den $(k+1)$-ten Schritt, der sich aus (2.16) ergibt

$$\begin{aligned} gap(\alpha) &= (z^{(k)} + \alpha\Delta z)^T (x^{(k)} + \alpha\Delta x) \\ &= (z^{(k)})^T x^{(k)} + \alpha \left((z^{(k)})^T \Delta x + (x^{(k)})^T \Delta z \right) + \alpha^2 \Delta z^T \Delta x. \end{aligned}$$

Die primale (2.13a) und duale (2.13b) Zulässigkeitsbedingung bedeuten, daß es $\Delta z = -A^T \Delta y$ und $A\Delta x = 0$ gibt, so daß $(\Delta z)^T \Delta x = 0$. Man erhält so

$$\begin{aligned} gap(\alpha) &= gap(0) + \alpha \left(\left(Z^{(k)} \Delta x \right)^T e + \left(X^{(k)} \Delta z \right)^T e \right) \\ &= gap(0) + \alpha \left(Z^{(k)} \Delta x + X^{(k)} \Delta z \right) e. \end{aligned}$$

Mit der Definition von r_μ aus (2.16) folgt

$$\begin{aligned} gap(\alpha) &= gap(0) + \alpha(\mu e - X^{(k)} Z^{(k)} e)^T e \\ &= gap(0) + \alpha(n\mu - gap(0)). \end{aligned} \tag{2.23}$$

Aus diesen Umformungen folgt das

Lemma 2.4.1 *Seien $gap(0)$ und $gap(\alpha)$ mit $\alpha = \min\{\alpha_{LP}, \alpha_{DP}\} > 0$ wie oben definiert und sei die Beziehung (2.16) für die Berechnung des Newton-Schrittes gegeben, dann erhält man aus (2.23)*

$$\mu < \frac{gap(0)}{n} \Longrightarrow gap(\alpha) < gap(0).$$

Für die praktische Anwendung der primal-dualen Innere-Punkt-Methode erwies es sich allerdings als sinnvoll, den Barriere-Parameter μ in größeren Schritten zu verringern (siehe [45]). Zur Berechnung des neuen Barriere-Parameters wird die Vorschrift

$$\mu^{(k+1)} := \frac{c^T x^{(k)} - b^T y^{(k)}}{D(n)} \quad \text{für} \quad k = 0, 1, \ldots$$

mit

$$D(n) = \begin{cases} n^2 & \text{falls} \quad n \leq 5000 \\ n\sqrt{n} & \text{falls} \quad n > 5000 \end{cases}$$

verwandt, die eine substantielle Verringerung der Dualitätslücke in jedem Barriere-Iterationsschritt garantiert.

Algorithmus 2.4.3: Getμ (b, c, x, y, z, Err)

1. {Initialisierung.}
   $n :=$ *ProblemDimension*; $\delta := 10^{-50}$; $Err :=$ "No Error";
2. {Wähle dimensionsabhängigen Divisor $D(n)$.}
   if $n \leq 5000$ **then** $D := n^2$ **else** $D := n\sqrt{n}$;
3. {Bestimme neuen Parameter μ.}
   $\mu_{new} := \frac{\text{DualityGap}\,(b,c,x,y,z)}{D}$;
4. {Überprüfe, ob $\mu_{new} \geq \delta$.}
   if $(\mu_{new} < \delta)$ **then** $Err :=$ „Wrong Parameter“;
5. **return** μ_{new}, Err;

2.4.3 Abwandlung des Newton-Verfahrens

Das in jedem Barriere-Schritt zu behandelnde Optimierungsproblem, d.h. die Nullstellenbestimmung für die Gradienten der Lagrangeschen Barriere-Funktionen, wird mit dem in Abschnitt 1.9 beschriebenen Newton-Verfahren gelöst. Wie in (2.16) beschrieben, führt die Berechnung der Newton-Korrektur auf die Lösung eines linearen Gleichungssystems, die mit Hilfe des Näherungslösers lss_aprx [52] bestimmt wird. Die im folgenden angegebene Beschreibung einer Abwandlung des Newton-Verfahrens nutzt zum einen die spezielle Form des linearen Gleichungssystems (2.16) aus, wodurch der Berechnungsaufwand für die Newton-Korrektur $(\Delta x, \Delta y, \Delta z)$ auf ein System der Größe $(m \times m)$ anstelle der vollen Systemgröße $(2n + m) \times (2n + m)$ reduziert wird. Die zweite Änderung bezieht sich auf die eigentliche Newton-Korrektur Δx aus Definition 1.9.1 bzw. 1.9.2, der, aufgespalten in Δx, Δy und Δz, im Ablauf der Prozedur 2.4.2 FeasibleNewtonStep bestimmt wird, um die Zulässigkeit der neuen Iterierten $(x^{(k+1)}, y^{(k+1)}, z^{(k+1)})$ zu gewährleisten.

Algorithmus 2.4.4: FeasibleNewtonMethod $(A, b, c, \mu, x^{(0)}, y^{(0)}, z^{(0)}, x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, Err)$

1. {Initialisierung.}
   $k := -1$; $k_{\max} := 5$; $\delta := 10^{-10}$; $Err :=$ „No Error“;

2. **repeat**

   (a) $k := k + 1$; $approx$:= **false** ;

   (b) {Bestimme die Residuenvektoren zu (2.16).}

   $$\begin{aligned} r_{LP} &:= \#*\left(b - Ax^{(k)} \right); \\ r_{DP} &:= \#*\left(A^T y^{(k)} + z^{(k)} - c \right); \\ r_{\mu} &:= \#*\left(\mu e - X^{(k)} Z^{(k)} e \right); \end{aligned}$$

   (c) {Bestimme die Newton-Korrektur $(\Delta x, \Delta y, \Delta z,)$ aus}
   {Definition 1.9.2. }
   $X := diag(x^{(k)})$; $Z := diag(z^{(k)})$;
   lss_aprx $\left(AZ^{-1}XA^T, b - \mu AZ^{-1}e - AZ^{-1}Xr_{DP}, \Delta y, Err \right)$;

   $$\begin{aligned} \Delta z &:= \#*\left(-r_{LP} - A^T \Delta y \right); \\ \Delta x &:= \#*\left(Z^{-1}(r_{\mu} - X\Delta z) \right); \end{aligned}$$

   (d) {Bestimme die neue zulässige Iterierte $(x^{(k+1)}, y^{(k+1)}, z^{(k+1)})$.}
   FeasibleNewtonStep $(A, b, c, x^{(k)}, y^{(k)}, z^{(k)},$
   $x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, Err)$;

   (e) {Überprüfe den Grad der Verbesserung der Nullstelle.}

   if $\left\| \begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \\ z^{(k+1)} \end{pmatrix} - \begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \end{pmatrix} \right\|_{\infty} < \delta$ **then** $approx$:= **true** ;

   until $approx$ **or** $k = k_{\max}$ **or** $Err <>$ „No Error“;

3. **return** $x^{(k+1)}$,$y^{(k+1)}$,$z^{(k+1)}$, Err;

In Schritt 2e wird die Maximum-Norm

$$\|x\|_{\infty} := \max_{i=1,\ldots,n} \mid x_i \mid \tag{2.24}$$

des Abstands zweier aufeinanderfolgender Iterierten $x^{(k)}$, $x^{(k+1)}$ als Abbruchkriterium für das Newton-Verfahren verwandt.

2.4.4 Verifikation der Existenz und Eindeutigkeit der Lösung

In jedem Barriere-Schritt wird ein neues Paar Lagrangescher Barriere-Funktionen L_{LP} und L_{DP} (2.12) gebildet. Ziel der primal-dualen Innere-Punkt-Methode ist es, in jedem Iterationsschritt mit Hilfe der rasch konvergierenden Newton-Iteration eine Nullstelle der Gradienten von L_{LP} und

L_{DP} zu bestimmen, da diese Optimalitätsbedingung 1. Ordnung nach Korollar 1.5.12 hinreichend für die Existenz einer Extremalstelle der aktuellen Näherungsfunktion ist. Um die Effizienz des Verfahrens zu erhalten, wird während des Iterationsprozesses im primal-dualen Innere-Punkt-Algorithmus das Minimum des vom Barriere-Parameter $\mu^{(l)}$ abhängigen Funktionenpaares $L_{LP}(x, y \mid \mu^{(l)})$ und $L_{DP}(x, y, z \mid \mu^{(l)})$ nur näherungsweise mit dem in Abschnitt 2.4.3 beschriebenen Algorithmus 2.4.4 FeasibleNewtonMethod berechnet.

Unter dem Gesichtspunkt der Verifikation des Berechnungsergebnisses des Verfahrens ist es notwendig, nach Abschluß des Iterationsprozesses die Existenz und Eindeutigkeit einer Nullstelle der Gradienten von L_{LP} und L_{DP} und damit des eigentlichen Optimierungsproblems (LP) nachzuweisen.

Bei einem fehlerfreien Ablauf des Iterationsverfahrens ist die Existenz einer Lösung des Optimierungsproblems (LP) durch den Nachweis eines primal zulässigen Punktes x und eines dual zulässigen Punktes (y, z) unter Ausnutzung der Aussage von Dualitätssatz 2.3.3 garantiert. Die Eindeutigkeit der optimalen Lösung wird mit Hilfe des Algorithmus 1.9.2 Verification aus Abschnitt 1.9 überprüft. Ist eine Verifikation der Eindeutigkeit möglich, so ist die Existenz weiterer optimaler Lösungen ausgeschlossen, d.h. insbesondere, daß nicht mehrere Ecken des zulässigen Bereiches optimale Lösungen sein können.

2.4.5 Einschluß des optimalen Zielfunktionswertes

Aufgrund des Dualitätssatzes 2.3.1 gilt für alle zulässigen Iterierten $([x]^{(l)}, [y]^{(l)}, [z]^{(l)})$, $(l = 0, 1, \ldots)$, daß die Dualitätslücke bestimmt wird als das Supremum von

$$f([x]^{(l)}) - g([y]^{(l)}) = c^T[x]^{(l)} - b^T[y]^{(l)} = ([z]^{(l)})^T[x]^{(l)}$$

(vgl. Definition 2.19), d.h.

$$gap = sup\left(([z]^{(l)})^T[x]^{(l)} \right).$$

Um auf dem Rechner eine garantierte obere Schranke für den Abstand zum exakten optimalen Funktionswert $f(\widehat{x})$ numerisch bestimmen zu können, werden erneut das exakte Skalarprodukt in Verbindung mit der gerichteten Rundung eingesetzt. Algorithmisch führt dies auf

Algorithmus 2.4.5: DualityGap $(b, c, [x], [y], [z])$

1. {Berechnung der Dualitätslücke, d.h. Bestimmung einer oberen}
   {Schranke für den Fehler des optimalen Funktionswertes. }
   $DualityGap := \sup\left(\#\#(c^T[x] - b^T[y]) \right);$
   {oder}
   $DualityGap := \sup\left([z] \circledast [x] \right);$
2. **return** $DualityGap$;

Daß an dieser Stelle die Intervall-Version des Algorithmus, die sich unmittelbar aus der reellwertigen Version ergibt, angegeben wird, liegt in der Tatsache begründet, daß nicht nur die Optimalitätslücke für Punktlösungen bestimmt werden soll, sondern auch eine Obergrenze der Dualitätslücke für intervallmäßige Einschließungen des Zielfunktionswertes, die durch den Verifikationsschritt aus Abschnitt 2.4.4 entstehen.

2.4.6 Primal-dualer Innere-Punkt-Algorithmus

Der vollständige, das Berechnungsergebnis verifizierende Algorithmus zur Bestimmung des optimalen Zielfunktionswertes eines linearen Optimierungsproblems (LP), das in der in Abschnitt 2.4.1 beschriebenen Standardform vorliegt, gliedert sich in mehrere Komponenten.

Zunächst wird die strikte Zulässigkeit einer Startlösung für das gegebene (LP) und das dazu korrespondierende duale Optimierungsproblem (DP) mit der Prozedur CheckFeasibility verifiziert ($(x^{(0)}, y^{(0)}, z^{(0)}) \in \overset{\circ}{M}$). Daß die Startlösung echt im Innern der zulässigen Bereiche von (LP) und (DP) liegt, ist eine notwendige Voraussetzung für den erfolgreichen Einsatz des Barriere-Verfahrens (vgl. Abschnitt 1.6).

Das eigentliche Innere-Punkt-Verfahren besteht aus einer *äußeren* Iteration, der Barriere-Iteration, und einer *inneren* Iteration, dem Newton-Verfahren, das in jedem Barriere-Schritt zur Bestimmung der Minimalstelle der aktuellen Barriere-Funktion verwandt wird.

Die Anwendung der Theorie der Lagrangeschen Multiplikatoren (Abschnitt 1.7) und des Barriere-Verfahrens (Abschnitt 1.6) auf ein lineares Optimierungsproblem (LP) führt auf ein Paar logarithmischer Lagrangescher Barriere-Funktionen L_{LP} und L_{DP} aus (2.12). Diese stellen ein unrestringiertes Optimierungsproblem dar, das (für einen Barriere-Parameter $\mu \longrightarrow 0$) äquivalent zum Grundproblem ist (vgl. Abschnitt 2.3.2). Das Abbruchkriterium für das Innere-Punkt-Verfahren ist die relative Größe der

Dualitätslücke, die aufgrund der Dualitätssätze aus Abschnitt 2.3.1 eine obere Schranke für den relativen Fehler des berechneten optimalen Zielfunktionswertes liefert. Mit der Forderung

$$\frac{c^T[x]^{(k)} - b^T[y]^{(k)}}{1 + \mid b^T[y]^{(k)} \mid} = \frac{\mathsf{DualityGap}\,(b, c, [x], [y], [z])}{1 + \mid b^T[y]^{(k)} \mid} < \varepsilon$$

und einem Wert von $\varepsilon = 10^{-10}$ werden 10 übereinstimmende Dezimalziffern zwischen primalem und dualem Zielfunktionswert garantiert, wobei durch die Verwendung der Intervallarithmetik und der Funktion DualityGap eine Verifikation dieses Einschlusses des Zielfunktionswertes möglich ist.

Um in jedem Iterationsschritt des Barriere-Verfahrens eine Verringerung der Dualitätslücke zu garantieren, wird der Barriere-Parameter $\mu^{(l)}$ gemäß der in Abschnitt 2.4.2.2 beschriebenen Strategie mit der Funktion Getμ bestimmt.

Die Minimalstelle des Funktionenpaares $L_{LP}(x, y \mid \mu^{(l)})$ und $L_{DP}(x, y, z \mid \mu^{(l)})$ wird nun näherungsweise mit dem in Abschnitt 2.4.3 beschriebenen Algorithmus 2.4.4 FeasibleNewtonMethod berechnet.

Der vollständige Algorithmus zur verifizierten Lösung linearer Optimierungsprobleme unter Verwendung der primal-dualen Innere-Punkt-Methode baut auf dem Prinzip des Algorithmus 2.2.1 PrimalNewtonBarriereMethod auf. Er nutzt dabei die in Abschnitt 2.3 beschriebenen Zusammenhänge zwischen primalem (LP) und dualem Problem (DP) sowie die in den Abschnitten 2.4.1 bis 2.4.5 eingeführten Hilfsmittel zur Kontrolle der Berechnungsergebnisse aus.

Die Eingabeparameter für den Lösungsalgorithmus sind die Daten des linearen Optimierungsproblems in der erweiterten Standardform (LP), die in Form einer $(m \times n)$-dimensionalen reellen Matrix A, eines m-dimensionalen Vektors b und eines n-dimensionalen Vektors c gegeben sind, wobei die Voraussetzungen, daß A vollen Rang hat und daß die Zahl der Unbekannten x_i größer ist als die Zahl der Nebenbedingungen, d.h. $m < n$, erfüllt sein müssen. Weiterer Eingabeparameter ist eine primal und dual zulässige Startlösung $(x^{(0)}, y^{(0)}, z^{(0)})$ mit $x^{(0)}, z^{(0)} \in I\!R^n$ und $y^{(0)} \in I\!R^m$.

Ist die Startlösung als innerer Punkt verifiziert, so startet die Lagrange-Barriere-Iteration bezüglich des Barriere-Parameters $\mu^{(0)}$. Formal werden nun die primale und duale Lagrangesche Barriere-Funktion bezüglich des neuen Barriere-Parameters $\mu^{(l)}$ gebildet (vgl. (2.12)). Für dieses Funktionenpaar, das ein unbeschränktes nicht-lineares Optimierungsproblem darstellt, wird die Nullstelle des Gradienten (Optimalitätsbedingung 1. Ord-

nung)

$$\nabla(L_{LP}, L_{DP}) = \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe - \mu e \end{pmatrix} = 0$$

mit Hilfe des Newton-Verfahrens bestimmt. Auf Grund der speziellen Gestalt des Nullstellenproblems führt die Berechnung der Newton-Korrektur auf das $((2n + m) \times (2n + m))$-dimensionale Gleichungssystem (2.16). Um die Zulässigkeit aller Iterierten im Newton-Verfahren zu gewährleisten, wird allerdings die abgewandelte Version des Newton-Algorithmus eingesetzt, die im vorangegangenen Abschnitt 2.4.3 FeasibleNewtonMethod angegeben wurde.

Falls die Minimalstelle $(x^{(l,k+1)}, y^{(l,k+1)}, z^{(l,k+1)})$ für das Funktionenpaar L_{LP} und L_{DP} numerisch ausreichend angenähert werden konnte, wird das Resultat der Newton-Iteration als neue Iterierte $(x^{(l+1)}, y^{(l+1)}, z^{(l+1)})$ des Barriere-Verfahrens übernommen.

Für diese aktuelle Näherungslösung wird eine Oberschranke für den relativen Fehler des optimalen Zielfunktionswertes unter Ausnutzung der Dualitätssätze der linearen Optimierung (vgl. Abschnitt 2.3.1) bestimmt. Die Dualitätslücke wird mit der Funktion DualityGap aus Abschnitt 2.4.5 berechnet, wobei exaktes Skalarprodukt und gerichtete Rundung eingesetzt werden, um eine Garantie über die Anzahl der richtigen Ziffern des berechneten Optimalwertes abgeben zu können.

Das Verfahren bricht ab, wenn die gewünschte relative Genauigkeit des Ergebnisses (Dualitätslücke $< \varepsilon$) garantiert werden kann, wenn eine vorgegebene maximale Anzahl an Iterationsschritten $l_{\max}$ überschritten wird oder wenn ein Fehler in einem Teilschritt des Verfahrens aufgetreten ist.

Ausgegeben werden — im Falle eines fehlerfreien Ablaufes des Iterationsverfahrens — eine zulässige Näherungslösung sowie ein Einschluß des optimalen Funktionswertes, die die gewünschten Genauigkeitsanforderungen erfüllen. Falls möglich, wird die Eindeutigkeit der primal bzw. dual zulässigen optimalen Lösungen garantiert. Ist ein Fehler während des Iterationsprozesses aufgetreten, so wird dieser in textueller Form als Ausgabeparameter *Msg* zurückgegeben.

Algorithmus 2.4.6: PrimalDualMethod ($A, b, c, x^{(0)}, y^{(0)}, z^{(0)}$, $x^{(l+1)}, y^{(l+1)}, z^{(l+1)}, [f]$, *unique*, *primal*, *dual*, *Msg*, *Err*)

1. {Initialisierung.}
   $l := -1$; $l_{\max} := 100$; $\varepsilon = 10^{-10}$; *Msg* := „ “; *Err* := „No Error“;

2. {Prüfe die strikte Zulässigkeit der Startlösung $(x^{(0)}, y^{(0)}, z^{(0)})$,}
   {d.h. $Ax^{(0)} = b$, $A^T y^{(0)} + z^{(0)} = c$, $x^{(0)} > 0$, und $z^{(0)} > 0$. }
   CheckFeasibility $(A, b, c, x^{(0)}, y^{(0)}, z^{(0)}, primal, dual)$;
   if not ($primal$ **and** $dual$) **then** Err := „Initial solution not feasible“
3. {Lagrange-Barriere-Iteration bzgl. Iterationsparameter μ.}
   if (Err = „No error“) **then**
   repeat
   (a) $l := l + 1$;
   (b) {Bestimme neuen Parameter μ.}
   $\mu^{(l)}$:= Getμ $(b, c, x^{(l)}, y^{(l)}, z^{(l)}, \mu^{(l-1)}, Err)$;
   (c) {Bilde die neuen Lagrange-Barriere-Funktionen.}
   $L_{LP}(x, y \mid \mu^{(l)}) := c^T x - \mu \sum_{i=1}^{n} \ln(x_i) - y^T(Ax - b)$;
   $L_{DP}(x, y, z \mid \mu^{(l)}) := b^T y + \mu \sum_{i=1}^{n} \ln(z_i) - x^T(A^T y + z - c)$;
   (d) {Bestimme das Minimum $\left(x^{(l,k+1)}, y^{(l,k+1)}, z^{(l,k+1)} \right)^T$ }
   {für L_{LP} (2.12a) und L_{DP} (2.12b), d.h. bestimme die}
   {Nullstelle von $\nabla(L_{LP}, L_{DP})$. }
   FeasibleNewtonMethod ($A, b, c, \mu^{(l)}, x^{(l,0)}, y^{(l,0)}, z^{(l,0)}$,
   $x^{(l,k+1)}, y^{(l,k+1)}, z^{(l,k+1)}, Err$);
   (e) {Bestimme die neue Iterierte.}
   if Err = “No Error” **then**
   begin $x^{(l+1)} := x^{(l,k+1)}$; $y^{(l+1)} := y^{(l,k+1)}$; $z^{(l+1)} := z^{(l,k+1)}$; **end**
   (f) {Bestimme die Dualitätslücke.}
   Gap := DualityGap $\left((b, c, x^{(l+1)}, y^{(l+1)}, z^{(l+1)} \right)$;
   until $Gap < \varepsilon$ **or** $l = l_{\max}$ **or** Err <> „No Error“
4. {Überprüfe, ob die ermittelte optimale zulässige Lösung }
   {eindeutig ist und ob ggf. auch eine Umgebung der Lösung}
   {zulässig ist und die Optimalitätsbedingung erfüllt. }
   $[f] := c^T x^{(l+1)} \cup b^T y^{(l+1)}$;
   Verification ($A, b, c, \mu^{(l)}, x^{(l+1)}, y^{(l+1)}, z^{(l+1)}$,
   $[x]^{(l+1)}, [y]^{(l+1)}, [z]^{(l+1)}, unique$);
   CheckFeasibility $(A, b, c, [x]^{(l+1)}, [y]^{(l+1)}, [z]^{(l+1)}, primal, dual)$;
   Gap := DualityGap $\left(b, c, [x]^{(l+1)}, [y]^{(l+1)}, [z]^{(l+1)} \right)$;
   if $unique$ **and** $primal$ **and** $dual$ **and** $Gap < \varepsilon$ **then**
   Msg := „Uniqueness of optimal solution verified.“
   else
   Msg := „Optimal solution probably not unique.“

5. **return** $x^{(l+1)}, y^{(l+1)}, z^{(l+1)}, [f], unique, primal, dual, Msg, Err$);

2.4.7 Einschluß aller optimalen Basis-Lösungen

Von Interesse sind bei der Lösung eines linearen Optimierungsproblems nicht nur ein Einschluß des optimalen Zielfunktionswertes sondern auch die optimale Basis-Indexmenge und die zulässige optimale Basis-Lösung. Der primal-duale Innere-Punkt-Algorithmus aus Abschnitt 2.4.6 bestimmt sehr effizient einen Einschluß des optimalen Zielfunktionswertes und eine zulässige Näherung des optimalen Lösungsvektors. Da der so bestimmte Vektor per Definition ein *innerer* Punkt des zulässigen Bereiches $M := \{x \in I\!R^n \mid Ax = b, x \geq \vec{0}\}$ ist – die optimale Basis-Lösung eines (LP) hingegen immer ein Eckpunkt vom M ist – kann diese Näherung, speziell in numerisch kritischen Fällen, deutlich „entfernt“ von der Optimalstelle liegen. Ist die Lösung eines (LP) nicht eindeutig, so ist dies sogar die Regel.

Definition 2.4.2 Es bezeichne f_{opt} den *optimalen Zielfunktionswert*, $\mathcal{V}_{opt}$ die *Menge der optimalen Basis-Indexmengen* und $\mathcal{X}_{opt}$ die *Menge der zulässigen optimalen Basis-Lösungen* eines linearen Optimierungsproblems (LP).

Es ist das Ziel, Einschließungen für f_{opt}, $\mathcal{V}_{opt}$ und $\mathcal{X}_{opt}$ zu berechnen. Der im folgenden angegebene Algorithmus 2.4.13 berechnet ein Intervall $[f]$, eine *Menge von Indexmengen* $\mathcal{V} = \{\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(s)}\}$ und eine Menge von Intervallvektoren $\mathcal{X} = \{[x]^{(1)}, \ldots, [x]^{(s)}\}$ mit $[x]^{(i)} \in I\!I\!R^n$, für die die drei Bedingungen

1. $f_{opt} \in [f]$
2. $\mathcal{B} \in \mathcal{V}_{opt} \Rightarrow \mathcal{B} \in \mathcal{V}$
3. $x \in \mathcal{X}_{opt} \Rightarrow$ es existiert ein $\mathcal{B}^{(i)} \in \mathcal{V}$ mit $x \in [x]^{(i)} \in \mathcal{X}$

erfüllt sind.

Die folgende *a-posteriori* Methode, die bereits von Hammer, Hocks, Kulisch und Ratz in [20] veröffentlicht ist, liefert entweder die genannten Einschließungen $[f]$, $\mathcal{V}$ und $\mathcal{X}$, welche die Bedingungen 1, 2 und 3 erfüllen, und garantiert somit die Existenz einer optimalen Lösung für das (LP), oder sie gibt eine Fehlermeldung zurück, die besagt, daß keine Einschließungen berechnet werden konnten. Das letztere ist der Fall, wenn die Menge der zulässigen Punkte M leer ist, wenn die Zielfunktion auf M unbeschränkt ist

oder wenn die Teilmatrix A_B bezüglich der optimalen Basis-Indexmenge $\mathcal{B}$ (numerisch) singulär ist.

Sei $\mathcal{B} \in \mathcal{V}$ eine Indexmenge $\{\beta_1, \ldots, \beta_m\} \subseteq \{1, \ldots, n\}$ und seien $[x_B]$, $[y]$, $[h]_{*,\nu}$ mit $\nu \in \mathcal{N} := \{\nu_1, \ldots, \nu_{n-m}\} := \{1, \ldots, n\} \setminus \mathcal{B}$ Intervallvektoren, die die Lösungen der linearen Gleichungssysteme

$$A_B \cdot x_B = b \tag{2.25a}$$
$$A_B^{\mathrm{T}} \cdot y = c_B \tag{2.25b}$$
$$A_B \cdot h_{*,\nu} = a_{*,\nu} \tag{2.25c}$$

einschließen. Diese Lösungseinschließungen werden mit dem selbstverifizierenden Gleichungssystemlöser lss des PASCAL–XSC Systems bestimmt (siehe PASCAL–XSC User's Guide [52]). Ist dieser Algorithmus erfolgreich, so ist die Nicht-Singularität der reellen Matrix A_B garantiert [56].

Es wird definiert

$$\begin{array}{rcl} [z] & := & ([z_B], [z_N])^{\mathrm{T}}, \quad [z_B] := 0 \\ [z_N] & := & A_N^{\mathrm{T}} \cdot [y] - c_N \\ [x] & := & ([x_B], [x_N])^{\mathrm{T}}, \quad [x_N] := 0 \\ [H] & := & ([h]_{*,\nu_1}, \ldots, [h]_{*,\nu_{m-n}}) \\ [f] & := & c_B^{\mathrm{T}} \cdot [x_B] \cap b^{\mathrm{T}} \cdot [y]. \end{array} \tag{2.26}$$

Definition 2.4.3 Das sogenannte *Intervalltableau* $[T]$ bezüglich einer Basis-Indexmenge $\mathcal{B} \in \mathcal{V}$ hat die Form

$$[T] := \left(\begin{array}{c|c} [f] & [z_N]^{\mathrm{T}} \\ \hline [x_B] & [H] \end{array}\right) = \left(\begin{array}{c|ccc} [f] & [z]_{\nu_1} & \cdots & [z]_{\nu_{n-m}} \\ \hline [x]_{\beta_1} & [h]_{\beta_1\nu_1} & \cdots & [h]_{\beta_1\nu_{n-m}} \\ \vdots & \vdots & & \vdots \\ [x]_{\beta_m} & [h]_{\beta_m\nu_1} & \cdots & [h]_{\beta_m\nu_{n-m}} \end{array}\right). \tag{2.27}$$

Für den selbsverifizierenden Lösungsalgorithmus sind die folgenden zwei Sätze, die von Krawczyk [38] und Jansson [24] bewiesen wurden, von zentraler Bedeutung.

Satz 2.4.2 *Sei $\mathcal{B} \in \mathcal{V}$ eine Basis-Indexmenge des (LP) und sei $[T]$ das entsprechende Intervalltableau.*

1. *Falls für die Intervallvektoren $[x_B]$ und $[z_N]$ aus (2.26) die Bedingungen*

$$\underline{x}_B > 0 \quad \textit{und} \quad \underline{z}_N > 0 \tag{2.28}$$

erfüllt sind, dann besitzt das (LP) eine eindeutige optimale Lösung, die in $[x]$ enthalten ist. Darüber hinaus gilt $\mathcal{V}_{opt} = \mathcal{B}$ und $f_{opt} \in [f]$.

2. *Falls* $\mathcal{B} \in \mathcal{V}_{opt}$, *dann gilt*

$$\overline{x}_B \geq 0 \quad \textit{und} \quad \overline{z}_N \geq 0. \tag{2.29}$$

Ist Bedingung (2.28) erfüllt, so heißt das (LP) *basisstabil*, d.h. das lineare Optimierungsproblem hat eine eindeutige Lösung. Die Ungleichungen aus (2.29) stellen eine notwendige Bedingung dafür dar, daß $\mathcal{B}$ eine optimale Basis-Indexmenge des (LP) ist. Diese Basis-Indexmengen sind allerdings nur von Interesse, wenn das (LP) nicht basisstabil ist.

Der zweite Satz gibt an, wie eine Einschließung der *Menge aller benachbarten Basis-Indexmengen* $\mathcal{L}$ für $\mathcal{B}$ bestimmt werden kann, wobei $\mathcal{B} \in \mathcal{V}_{opt}$ und die $\mathcal{B}' \in \mathcal{L}$ sich in genau einem Index von $\mathcal{B}$ unterscheiden. Jede benachbarte Basis-Indexmenge bestimmt eine neue benachbarte Ecke des zulässigen Bereiches M. Satz 2.4.3 liefert ein hinreichendes Kriterium für die Bestimmung einer Liste von Basis-Indexmengen, die eine echte Obermenge derjenigen benachbarten Basis-Indexmengen ist, die optimale Basis-Lösungen repräsentieren.

Satz 2.4.3 *$\mathcal{B}$ sei eine Basis-Indexmenge des (LP), für die $\overline{x}_B \geq 0$ und $\overline{z}_N \geq 0$ gilt. Sei $[T]$ das entsprechende Intervalltableau und sei $\widetilde{\mathcal{L}}$ definiert als die Menge aller Indexmengen $\mathcal{B}' = (\mathcal{B} \setminus \{\beta\}) \cup \{\nu\}$ mit $\beta \in \mathcal{B}$ und $\nu \in \mathcal{N}$, für die eine der folgenden Bedingungen erfüllt ist:*

$$0 \in [z]_\nu,\ \overline{h}_{\beta_\nu} > 0 \quad \textit{und} \quad \frac{\underline{x}_\beta}{\overline{h}_{\beta\nu}} \leq \min\left\{ \frac{\overline{x}_\beta}{\underline{h}_{\beta\nu}} \mid \underline{h}_{\beta\nu} > 0,\ \beta \in \mathcal{B} \right\} \tag{2.30}$$

$$0 \in [x]_\beta,\ \underline{h}_{\beta_\nu} < 0 \quad \textit{und} \quad \frac{\overline{z}_\nu}{\underline{h}_{\beta\nu}} \geq \max\left\{ \frac{\underline{z}_\nu}{\overline{h}_{\beta\nu}} \mid \overline{h}_{\beta\nu} < 0,\ \nu \in \mathcal{N} \right\}. \tag{2.31}$$

Falls $\mathcal{B} \in \mathcal{V}_{opt}$, dann gilt $\widetilde{\mathcal{L}} \supseteq \mathcal{L}$.

Der folgende Algorithmus 2.4.7 berechnet das Intervalltableau $[T]$ aus (2.27), wobei die selbstverifizierende Routine zur Lösung linearer Gleichungssysteme lss der PASCAL–XSC Systems verwandt wird. Scheitert die Prozedur lss, so wird eine Fehlermeldung zurückgegeben.

Algorithmus 2.4.7: ComputeTableau $(A, b, c, \mathcal{B}, \mathcal{N}, [T], Err)$

1. {Initialisierung.}
   Err := „No Error“;
2. {Bestimme die Lösung der Gleichungssysteme (2.25).}
   lss $(A_B, b, [x_B], Err)$; $[x_N] := 0$; $[z_B] := 0$;

lss $(A_B^T, c_B, [y], Err)$;
for $\nu \in \mathcal{N}$ **do**
lss $(A_B, a_{*,\nu}, [h]_{*,\nu}, Err)$;
if $Err \neq$ "No Error" {in allen lss Aufrufen} **then**
return $Err :=$ „Submatrix A_B probably singular“;

3. {Berechne die Komponenten des Intervalltableaus $[T]$.}
$[x] := ([x_B], [x_N])^T$;
$[z_N] := A_N^{\mathrm{T}} \cdot [y] - c_N$;
$[H] := ([h]_{\nu_1}, \ldots, [h]_{\nu_{m-n}})$;
$[f] := c_B^{\mathrm{T}} \cdot [x_B] \cap b^{\mathrm{T}} \cdot [y]$;
4. **return** $[T], Err$;

Mit Algorithmus 2.4.8 wird überprüft, ob das zur aktuellen Basis-Indexmenge $\mathcal{B}$ gehörende Intervalltableau $[T]$ eine basisstabile Lösung des (LP) repräsentiert. Wird die Bedingung (2.28) erfüllt, so folgt daraus die Eindeutigkeit der optimalen Basis-Lösung, d.h. $\mathcal{B} = \mathcal{V} \supseteq \mathcal{V}_{opt}$, $[x] = \mathcal{X} \supseteq \mathcal{X}_{opt}$ und $[f] \supseteq f_{opt}$. Weitere Berechnungen sind daraufhin nicht mehr nötig.

Algorithmus 2.4.8: BasisStable $([T])$

1. {Prüfe, ob $\underline{x}_B > 0$ (vgl. (2.28)).}
for $\beta \in \mathcal{B}$ **do**
if $\underline{x}_\beta \leq 0$ **then return** $IsStable :=$ *false*;
2. {Prüfe, ob $\underline{z}_N > 0$ (vgl. (2.28)).}
for $\nu \in \mathcal{N}$ **do**
if $\underline{z}_\nu \leq 0$ **then return** $IsStable :=$ *false*;
3. **return** $IsStable :=$ *true*;

Algorithmus 2.4.9 wird eingesetzt, um zu entscheiden, ob das zur aktuellen Basis-Indexmenge $\mathcal{B}$ gehörende Intervalltableau $[T]$ eine möglicherweise (d.h. numerisch) optimale Lösung des (LP) repräsentiert. Das bedeutet $\mathcal{B} \in \{\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(s)}\} = \mathcal{V} \supseteq \mathcal{V}_{opt}$, $[x] \in \mathcal{X} = \{[x]^{(1)}, \ldots, [x]^{(s)}\} \supseteq \mathcal{X}_{opt}$ und $[f] \supseteq f_{opt}$. Ist die Bedingung (2.29) erfüllt, dann werden $\mathcal{B}$, $[x]$ und $[f]$ in die Liste der optimalen Basis-Lösungen aufgenommen.

Algorithmus 2.4.9: PossiblyOptimalSolution $([T])$

1. {Prüfe, ob $\overline{x}_B >= 0$ (vgl. (2.29)).}
for $\beta \in \mathcal{B}$ **do**
if $\overline{x}_\beta < 0$ **then return** $IsOptimal :=$ *false*;

2. {Prüfe, ob $\overline{z}_N >= 0$ (vgl. (2.29)).}
   for $\nu \in \mathcal{N}$ **do**
   if $\overline{z}_\nu < 0$ **then return** $IsOptimal := false$;
3. **return** $IsOptimal := true$;

Mit den Algorithmen 2.4.10 und 2.4.11 wird überprüft, ob der zulässige Bereich M leer ist oder ob die Zielfunktion $c^T x$ auf M unbeschränkt ist. Hierzu wird das Intervalltableau $[T]$ untersucht. M ist möglicherweise (d.h. numerisch) leer, fall es keinen Index $\nu \in \mathcal{N}$ gibt, so daß $[h]_{\beta\nu} < 0$ für alle Indizes $\beta \in \mathcal{B}$ mit $0 \in [x]_\beta$ gilt. Die Zielfunktion ist möglicherweise (d.h. numerisch) unbeschränkt auf M, falls es keinen Index $\beta \in \mathcal{B}$ gibt, so daß $[h]_{\beta\nu} > 0$ für alle Indizes $\nu \in \mathcal{N}$ mit $0 \in [z]_\nu$ gilt.

Algorithmus 2.4.10: EmptySolutionSet $([T])$

1. {Initialisierung.}
   $IsEmpty := false$;
2. {Prüfe, ob für alle $\beta \in \mathcal{B}$ mit $0 \in [x]_\beta$ ein}
   {$\nu \in \mathcal{N}$ mit $\overline{h}_{\beta\nu} < 0$ existiert. }
   for $\beta \in \mathcal{B}$ **with** $0 \in [x]_\beta$ **do**
   while (**not** $IsEmpty$) **do**
   $IsEmpty := true$;
   for $\nu \in \mathcal{N}$ **do**
   if $\overline{h}_{\beta\nu} < 0$ **then**
   $IsEmpty := false$; **exit**$_{\text{for-loop}}$;
3. **return** $IsEmpty$;

Algorithmus 2.4.11: Unbounded $([T])$

1. {Initialisierung.}
   $IsUnbounded := false$;
2. {Prüfe, ob für alle $\nu \in \mathcal{N}$ mit $0 \in [z]_\nu$ ein}
   {$\beta \in \mathcal{B}$ mit $\underline{h}_{\beta\nu} > 0$ existiert. }
   for $\nu \in \mathcal{N}$ **with** $0 \in [z]_\nu$ **do**
   while (**not** $IsUnbounded$) **do**
   $IsUnbounded := true$;
   for $\beta \in \mathcal{B}$ **do**
   if $\underline{h}_{\beta\nu} > 0$ **then**
   $IsUnbounded := false$; **exit**$_{\text{for-loop}}$;
3. **return** $IsUnbounded$;

Für die aktuelle Basis-Indexmenge $\mathcal{B}$ bestimmt Algorithmus 2.4.12 die Menge der benachbarten Basis-Indexmengen $\mathcal{L}$, die mögliche Kandidaten für weitere optimale Basis-Indexmengen sind. Eine echte Obermenge aller tatsächlich möglichen Kandidaten wird durch eine Überprüfung der Bedingungen aus Satz 2.4.3 ermittelt.

Algorithmus 2.4.12: NeighboringList $([T], \mathcal{B}, \mathcal{N})$

1. {Initialisierung.}
   $\mathcal{L} := \emptyset$;
2. {Bestimme eine Liste von benachbarten Basis-Indexmengen, die}
   {mögliche Kandidaten für eine optimale Lösung sind }
   {(vgl. Satz 2.4.3). }

   (a) {Suche nach Kandidaten $\nu \in \mathcal{N}$ und $\beta \in \mathcal{B}$ durch eine }
   {Bestimmung des minimalen Koeffizienten aus $[x]_\beta/[h]_{\beta\nu}$}
   {(vgl. (2.30)). }
   for all $\nu \in \mathcal{N}$ **with** $0 \in z_\nu$ **do**
   {Bestimme das Minimum aller $\overline{x}_\beta/\underline{h}_{\beta\nu}$ aus Spalte ν.}
   $colmin := \min(\overline{x}_\beta/\underline{h}_{\beta\nu})$;
   {Bestimme Kandidaten $\beta \in \mathcal{B}$ für einen Indextausch.}
   if $\underline{x}_\beta/\overline{h}_{\beta\nu} \leq colmin$ **then**
   {Bestimme die neue Basis-Indexmenge.}
   $\mathcal{B}' := (\mathcal{B} \setminus \{\beta\}) \cup \{\nu\}$;
   $\mathcal{L} := \mathcal{L} + \mathcal{B}'$;

   (b) {Suche nach Kandidaten $\beta \in \mathcal{B}$ und $\nu \in \mathcal{N}$ durch eine }
   {Bestimmung des minimalen Koeffizienten aus $[z]_\nu/[h]_{\beta\nu}$}
   {(vgl. (2.31)). }
   for all $\beta \in \mathcal{B}$ **with** $0 \in x_\beta$ **do**
   {Bestimme das Maximum aller $\underline{z}_\nu/\overline{h}_{\beta\nu}$ aus Zeile β.}
   $rowmax := \max(\underline{z}_\nu/\overline{h}_{\beta\nu})$;
   {Bestimme Kandidaten $\nu \in \mathcal{N}$ für einen Indextausch.}
   if $\overline{z}_\nu/\underline{h}_{\beta\nu} \geq rowmax$ **then**
   {Bestimme die neue Basis-Indexmenge.}
   $\mathcal{B}' := (\mathcal{B} \setminus \{\beta\}) \cup \{\nu\}$;
   $\mathcal{L} := \mathcal{L} + \mathcal{B}'$;
3. **return** $\mathcal{L}$;

Der nun folgende Algorithmus EncloseBasicSolution bestimmt Einschließungen für den optimalen Zielfunktionswert f_{opt}, für die komplette Menge

der optimalen Basis-Indexmengen $\mathcal{V}_{opt}$ und die Menge der zulässigen optimalen Basis-Lösungen $\mathcal{X}_{opt}$ eines linearen Optimierungsproblems (LP). Wie bereits erwähnt, wird eine optimale Basis-Indexmenge $\mathcal{B}_{start}$ als Eingabeparameter des Algorithmus benötigt.

Während des Ablaufes des Algorithmus 2.4.13 wird in Schritt 2a eine Basis-Indexmenge $\mathcal{B}$ aus der Liste der Kandidaten für eine optimale Basis-Indexmenge $\mathcal{C}$ ausgewählt und gleichzeitig die Liste der bereits untersuchten Basis-Indexmengen $\mathcal{E}$ aktualisiert. In Schritt 2b wird das Intervalltableau (2.27) unter Verwendung von Algorithmus 2.4.7 bestimmt. Anschließend wird mit Algorithmus 2.4.8 überprüft, ob die aktuelle Basis-Indexmenge $\mathcal{B}$ eine basisstabile Lösung repräsentiert. Ist dies nicht der Fall, wird in Schritt 2d mit Hilfe der Algorithmen 2.4.10 und 2.4.11 entschieden, ob das gegebene (LP) unzulässig ist, d.h. M ist leer oder $c^T x$ ist auf M unbeschränkt. In Schritt 2e wird die Optimalität der aktuellen Lösung mit Algorithmus 2.4.9 verifiziert. Ist diese Verifikation möglich, so wird sie in der Liste der optimalen Basis-Lösungen $\mathcal{X}_{opt}$ gespeichert. Anschließend wird mit Algorithmus 2.4.12 die Liste der Kandidaten für eine optimale Basis-Indexmenge $\mathcal{L}$ bestimmt.

Algorithmus 2.4.13: EncloseBasicSolution $(A, b, c, \mathcal{B}_{start}, [f], \mathcal{V}, \mathcal{X}, No, Err)$

1. {Initialisierung.}
   $Err :=$ „No Error“; $k_{\max} := 100$;
   $k := 0$; $\mathcal{C} := \mathcal{B}_{start}$; $\mathcal{E} := \emptyset$; $\mathcal{V} := \emptyset$; $\mathcal{X} := \emptyset$; $No := 0$;

2. {Iteration.}
   repeat

   (a) {Bestimme die Indexmengen $\mathcal{B}$ und $\mathcal{N}$.}
   select $\mathcal{B} \in \mathcal{C}$; $\mathcal{C} := \mathcal{C} \setminus \mathcal{B}$; $\mathcal{E} := \mathcal{E} \cup \mathcal{B}$;
   $\mathcal{N} := \{1, \ldots, n\} \setminus \mathcal{B}$;

   (b) {Berechne das Intervalltableau $[T]$.}
   ComputeTableau $(A, b, c, \mathcal{B}, \mathcal{N}, [T], Err)$;
   if $Err \neq$ „No Error“ **then exit**$_{\text{repeat-loop}}$

   (c) {Prüfe, ob das Tableau basisstabil ist.}
   if BasisStable $([T])$ **then**
   {Speichere den optimalen Wert, die optimale Indexmenge}
   {und die optimale Lösung. }
   $[f] := c_B^T \cdot [x_B]$; $\mathcal{X} := ([x_B], [x_N])^{\mathrm{T}}$; $\mathcal{V} := \mathcal{B}$; $No := No + 1$;
   return $[f], \mathcal{V}, \mathcal{X}$;

(d) {Prüfe, ob der zulässige Bereich leer ist oder ob}
{die Zielfunktion unbeschränkt ist. }
if EmptySolutionSet ($[T]$) **then**
Err := „Set of feasible solutions is empty“;
exit$_{\text{repeat-loop}}$
if Unbounded ($[T]$) **then**
Err := „Objective function is unbounded“;
exit$_{\text{repeat-loop}}$

(e) {Prüfe, ob die aktuelle Lösung möglicherweise optimal ist.}
if PossiblyOptimalSolution ($[T]$) **then**
{Speichere den optimalen Wert, die optimale Indexmenge}
{und die optimale Lösung. }
$[f] := [f] \cup c_B^T \cdot [x_B]$; $\mathcal{X} := \mathcal{X} \cup ([x_B], [x_N])^{\mathrm{T}}$;
$\mathcal{V} := \mathcal{V} \cup \mathcal{B}$; $No := No + 1$;
{Bestimme die Menge der benachbarten Basis-Indexmengen.}
$\mathcal{L}$:= NeighboringList ($[T], \mathcal{B}, \mathcal{N}$);
$\mathcal{L} := (\mathcal{L} \setminus \mathcal{E}) \setminus \mathcal{C}$; $\mathcal{C} := \mathcal{C} \cup \mathcal{L}$;

(f) $k := k + 1$;

until $(\mathcal{C} = \emptyset)$ **or** $(k = k_{\max})$;

3. {Prüfe, ob Fehler aufgetreten sind.}
if $(k = k_{\max})$ **then** Err := „Maximum number of iterations exceeded“;
if $(No = 0)$ **then** Err := „Initial basic index set is not optimal“;

4. **return** $[f], \mathcal{V}, \mathcal{X}, Err$;

Der Algorithmus 2.4.13 bricht in den Schritten 2b oder 2d mit einer Fehlermeldung ab, falls $[T]$ nicht berechnet werden kann, oder wenn eine der Bedingungen aus Schritt 2d erfüllt ist.

$[T]$ kann nicht berechnet werden, wenn in Algorithmus 2.4.7 die Lösung der linearen Gleichungssysteme (2.25) scheitert. Das Auftreten dieses Fehlers ist ein Hinweis darauf, daß die aktuelle Teilmatrix A_B tatsächlich oder „numerisch“ singulär ist.

Die Abbruchbedingungen in Schritt 2d deuten auf einen leeren zulässigen Bereich M oder auf eine unbeschränkte Zielfunktion hin. In diesem Fall ist keine Berechnung von Einschließungen für f_{opt}, $\mathcal{V}_{opt}$ und $\mathcal{X}_{opt}$ möglich.

Die Basis-Indexmenge $\mathcal{B}_{start} \in \mathcal{V}$ wird durch eine Untersuchung des Resultats der primal-dualen Innere-Punkt-Methode in Algorithmus 2.4.6 bestimmt.

Gilt für die Intervallvektoren $[x_B]$ und $[y]$, die die exakten Lösungen der linearen Gleichungssysteme

$$A_B \cdot x_B = b, \quad A_B^{\mathrm{T}} \cdot y = c_B$$

einschließen, die Bedingung $\overline{x}_B \geq 0$ und $\overline{z} \geq 0$ wobei $[z] := A_N^{\mathrm{T}} \cdot [y] - c_N$, dann ist durch Satz 2.4.2 bewiesen, daß $\mathcal{B}$ (numerisch) eine optimale Basis-Indexmenge des (LP) ist. Damit gilt: $\mathcal{B}_{start} := \mathcal{B} \in \mathcal{V}$.

Algorithmus 2.4.14: DetermineBase $(x, y, z, \mathcal{B}_{start})$

1. {Initialisierung.}
   $m := upperbound(y)$; $n := upperbound(x)$;
   $\mathcal{B}_{start} := \emptyset$; $limit := \min(x) \cdot \max(x)$;
2. {Bestimme die Indizes, die Kandidaten für die Basis-Indexmenge}
   {$\mathcal{B}_{start}$ sind. }
   for $i := 1$ **to** n **do**
   if $x_i > limit$ **then**
   $\mathcal{B}_{start} := \mathcal{B}_{start} \cup i$;
3. {Prüfe Korrektheit von $\mathcal{B}_{start}$.}
   if $(size(\mathcal{B}_{start}) < m)$ **then**
   $Err :=$ „Wrong basic index set determined.“
   if not $(\overline{x}_B \geq 0$ **and** $\overline{z}_N \geq 0)$ **then**
   $Err :=$ „Wrong basic index set determined.“
4. **return** $\mathcal{B}_{start}$, Err;

2.4.8 Vollständiger Algorithmus

Durch die Kombination der Algorithmen PrimalDualMethod aus Abschnitt 2.4.6 und EncloseBasicSolution aus Abschnitt 2.4.7 ist die Möglichkeit gegeben, die Vorteile der Innere-Punkt-Methode bezüglich des Rechenaufwandes und die Vorteile des Simplex-Verfahrens bezüglich der Lösungseinschließung miteinander zu verbinden. Ausgehend von einer primal und dual zulässigen Startnäherung $x^{(0)}$, $y^{(0)}$ und $z^{(0)}$ für ein gegebenes (LP) in Standardform (vgl. Definition 1.4.4) wird durch die selbstverifizierende primal-duale Innere-Punkt-Methode eine garantiert zulässige Näherungslösung $\widetilde{x}$, $\widetilde{y}$ und $\widetilde{z}$ bestimmt, die den optimalen Zielfunktionswert $f_{opt} = c^T \widehat{x}$ auf mindestens 10 Dezimalstellen genau approximiert. Zusätzlich wird ein verifizierter Einschluß von f_{opt} ebenfalls auf 10 Dezimalstellen genau bestimmt. Als weitere Information liefert das Verfahren gegebenenfalls eine Verifikation der Eindeutigkeit der theoretischen optimalen Lösung $\widehat{x}$. Ausgehend von diesem

Resultat wird eine optimale Basis-Indexmenge $\mathcal{B}_{start}$ bestimmt. Durch den nun möglichen Wechsel auf das diskrete Simplex-Verfahren können neben einem Einschluß des optimalen Zielfunktionswertes auch Einschließungen der optimalen Basis-Lösung bzw. aller optimalen Basis-Lösungen bestimmt werden.

Der folgende Algorithmus faßt die Kombination der Lösungsverfahren zusammen.

Algorithmus 2.4.15: Combination

1. {Eingabe der Daten des (LP).}
   Input $(A, b, c, x^{(0)}, y^{(0)}, z^{(0)})$;
2. {Berechne einen Einschluß des optimalen Zielfunktionswertes f_{opt}}
   {und zulässige Näherungen $\widetilde{x}, \widetilde{y}, \widetilde{z}$ der optimalen Lösung mit }
   {der primal-dualen Innere-Punkt-Methode. }
   PrimalDualMethod ($A, b, c, x^{(0)}, y^{(0)}, z^{(0)}, \widetilde{x}, \widetilde{y}, \widetilde{z}, [f]$,
   $unique, primal, dual, Msg, Err$);
3. {Bestimme eine optimale Basis-Indexmenge $\mathcal{B}_{start}$.}
   DetermineBase $(\widetilde{x}, \widetilde{y}, \widetilde{z}, unique, \mathcal{B}_{start}, Err)$;
4. {Berechne Einschließungen für den optimalen Zielfunktionswert }
   {z_{opt}, die Menge aller optimalen Basis-Indexmengen $\mathcal{V}_{opt}$ und die}
   {Menge aller optimalen Basis-Lösungen $\mathcal{X}_{opt}$. }
   EncloseBasicSolution $(A, b, c, \mathcal{B}_{start}, [f], \mathcal{V}, \mathcal{X}, No, Err)$;
5. {Ausgabe der verifizierten Berechnungsergebnisse.}
   Output $(\widetilde{x}, \widetilde{y}, \widetilde{z}, [f], unique, \mathcal{V}, \mathcal{X}, Err)$;

Kapitel 3

Realisierung und Implementierung

Dieses Kapitel beschäftigt sich mit einigen Details der praktischen Durchführung der Innere-Punkte-Methode und ihrer Implementierung auf dem Rechner. Durch die Implementierung der in Kapitel 2 beschriebenen primal-dualen Innere-Punkt-Methode in der Programmiersprache PASCAL–XSC [31], ergibt sich mit der Portabilität des entsprechenden Compilers die Möglichkeit, das Verfahren auf einer großen Palette von Rechnern in identischer Form zu verwenden. Damit besteht die Möglichkeit, unter Verwendung von leistungsfähigen Großrechnern auch „große“ Probleme, wie sie in der Praxis vorkommen, zu behandeln. Im folgenden werden kurz die wichtigsten Merkmale von PASCAL–XSC erläutert und im Anschluß daran wird näher auf die Implementierung des Verfahrens eingegangen. Eine Beschreibung zur interaktiven Benutzung des Programms befindet sich in Kapitel 4.

Um den Rahmen dieser Arbeit nicht zu sprengen, wird auf die vollständige Wiedergabe des Quellcodes verzichtet, dessen voller Umfang etwa 4000 Programmzeilen beträgt. Der vollständige Quellcode wird, wie in der Einleitung angedeutet, auf einem FTP-Server der Universität Karlsruhe zur Verfügung gestellt. Hier werden vielmehr nur die wichtigsten Teile oder beispielhafte Auszüge angegeben, die Einblick in die Implementierung geben sollen. Wie bereits früher erwähnt, sind alle in Kapitel 2 angegebenen Algorithmen in Verbindung mit den Details zur praktischen Durchführung nahezu direkt auf den Rechner übertragbar.

3.1 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 [39], [41], [42]). Über Standard-PASCAL hinaus umfaßt PASCAL–XSC die folgenden für diese 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 wird auf [31] verwiesen.

Der aktuelle Sprachumfang von PASCAL–XSC bzw. des Compilers erlaubt größtenteils eine komfortable Programmierung der für das Verfahren benötigten Hilfsmittel (Datenstrukturen, Operationen, Arithmetiken), die über die bereits vorhandenen Konzepte hinausgehen. Auf einige Details der Implementierung wird im folgenden eingegangen.

3.2 Verifikationshilfsmittel

3.2.1 Zulässigkeitsprüfung

Das in Abschnitt 2.4.1 beschriebene Verfahren zur Überprüfung der Zulässigkeit eines Startpunktes bzw. einer Iterierten trägt der Problematik Rechnung, daß Rundungsfehler und Einschränkungen des darstellbaren Zahlenbereiches auf dem Rechner die Ergebnisse beeinflussen. Durch die Möglichkeit, Rechenoperationen in PASCAL–XSC mit den üblichen Operatoren und vorgegebener Rundungsrichtung nach oben $\triangle$ bzw. unten $\triangledown$ auszuführen, ist sowohl ein gesicherter Vergleich der Ergebnisse einer Maschinenzahloperation (z.B. $Ax \leq b$) als auch die Durchführung von Maschinenintervalloperationen (z.B. $A[x] \leq b$) möglich. Ein weiterer wesentlicher Vorteil der Spracherweiterung für wissenschaftliches Rechnen ist das Vorhandensein eines exakten Skalarproduktes. Die Zahl der notwendigen Rundungen für die Berechnung eines Matrix-Vektor-Produktes wird dabei auf eine Rundung je Lösungsvektorkomponente reduziert, was auch in numerisch kritischen Fällen noch eine korrekte Überprüfung der Relation $Ax \leq b$ erlaubt.

Die beiden Prozeduren `CheckFeasibility` (vgl. Algorithmus 2.4.1) für reelle Parameter und Intervallparameter verwenden die Module `mv_ari` für die reelle und `mvi_ari` für die Intervall-Matrix-Vektor-Arithmetik.

```
global
procedure CheckFeasibility(
                      A            : rmatrix;  { Procedure checks,  }
                      b, c         : rvector;  { if actual solution }
                      x, y, z      : rvector;  { is feasible        }
                    var primal, dual : boolean); {-------------------}
var
  A_   : rmatrix[lb(A,1)..ub(A,1),lb(A,2)..ub(A,2)-ub(A,1)];
  c_   : rvector[lb(c)..ub(c)-ub(A,1)];
  x_   : rvector[lb(x)..ub(x)-ub(A,1)];
  i, j : integer;

begin
  primal := positive(x);                      { Primal nonnegativity }
  dual   := positive(z);                      { Dual nonnegativity   }

  for i:=lb(A,1) to ub(A,1) do                { Determine sub-matrix }
    for j:=lb(A,2) to ub(A,2)-ub(A,1) do A_[i,j] := A[i,j];
  for i:=lb(c) to ub(c)-ub(A,1) do c_[i] := c[i];{ Determine sub-vec.}
  for i:=lb(x) to ub(x)-ub(A,1) do x_[i] := x[i];
                                             { Primal/dual feasibility }
  if not (A_*>x_ <= b)          then primal := FALSE;
  if not (transp(A)*>y <= c)    then dual := FALSE;
end;
```

3.2.2 Parameter

Die für das Iterationsverfahren der primal-dualen Innere-Punkt-Methode wichtigen Parameter α (siehe Abschnitt 2.4.2.1) und μ (siehe Abschnitt 2.4.2.2) werden als Funktionen realisiert. Die maximal zulässigen Parameter α_{LP} für die Newton-Korrektur des primalen Problems und α_{DP} des dualen Problems werden entsprechend den Vorschriften aus (2.22a) und (2.22b) bestimmt. Durch die Verwendung der gerichteten Rundung wird ein Überschätzen des Parameters, die eine Verletzung einer oder mehrerer Nebenbedingungen zur Folge hätte, vermieden.

```
global
function maxAlpha( x, dx : rvector): real;   { Determine max. Newton }
var                                          { step size             }
  i     : integer;
  Alpha : real;
begin
  Alpha := 1;
  for i:=lb(x) to ub(x) do
    if (dx[i] < 0) then
      if ( -x[i]/<dx[i] < Alpha) then Alpha := -x[i]/<dx[i];
  maxAlpha := Alpha;
end;
```

Der Barriere-Parameter μ wird Problemdimensionsabhängig bestimmt (vgl. Algorithmus 2.4.3). Die für die Berechnung der Dualitätslücke verwandte Prozedur `DualityGap` ist in einem getrennten Modul implementiert, das in Abschnitt 2.4.5 erläutert wird.

```
const delta = 1e-50;

global
function GetMu(    b, c    : rvector;    { Determine appropriate new }
                   x, y, z : rvector;    { barriere parameter        }
               var Err     : string) : real;
var
  n             : integer;
  mu_old, D, mu : real;

begin
  n := ub(x) - lb(x) + 1;
  if (n <= 5000) then D := sqr(n)
                 else D := sqrt(n);
  mu := DualityGap(b, c, x, y, z) / D;
  if (mu <= delta) then Err := 'Err: Wrong parameter mu';
  GetMu := mu;
end;
```

3.2.3 Spezielles Newton-Verfahren

Die in Abschnitt 2.4.3 beschriebene Abwandlung des allgemeinen Newton-Verfahrens, wird auf der Basis der Algorithmen 2.4.2 FeasibleNewtonStep und 2.4.4 FeasibleNewtonMethod implementiert. Als sinnvoller Wert hat sich $i_{\max} = 5$ für die maximale Anzahl der Newton-Iterationen ergeben, da dies auf Grund der quadratischen Konvergenz des Verfahrens in der Regel nicht überschritten wird und wegen des hohen Rechenaufwands eines einzelnen Iterationsschrittes praktischerweise nicht überschritten werden sollte. Abbruchkriterium für die Newton-Iteration ist $\delta = 10^{-10}$ als Vergleichsgröße für den Abstand zweier aufeinanderfolgender Iterierten bezüglich der Maximum-Norm (vgl. (2.24)).

Die Berechnung der Newton-Korrektur erfolgt in der Prozedur `FeasibleNewtonStep` unter Berücksichtigung der Nebenbedingung, den zulässigen Bereich der Optimierungsproblems (LP) nicht zu verlassen. Die Erfüllung dieser Bedingung wird durch die Berechnung der Schrittweitenparameter α_{LP} und α_{DP} (siehe Abschnitt 3.2.2) angestrebt und durch die in Abschnitt 2.4.1 beschriebenen Prozedur `CheckFeasibility` überprüft. Die Vorgabe des Verkleinerungsfaktors $\tau = 0.9$ für die Newton-Korrektur (wie in Abschnitt 2.4.2.1 angegeben) hat sich im praktischen Einsatz als sinnvoll erwiesen.

```
const
  i_max = 5;

procedure FeasibleNewtonStep(                  { Compute maximum      }
              A                  : rmatrix; { feasible Newton step }
              b, c               : rvector; { in primal and dual   }
              x_0, y_0, z_0      : rvector; { direction not        }
              dx, dy, dz         : rvector; { touching the boundary}
          var x_k, y_k, z_k      : rvector; { of the feasible      }
          var Err                : string); { region               }
                                            {-----------------------}
var
  alpha_LP, alpha_DP  : real;
  i                   : integer;
  primal, dual        : boolean;

begin

  x_k := x_0 + dx;
  y_k := y_0 + dy;
  z_k := z_0 + dz;

  CheckFeasibility(A, b, c, x_k, y_k, z_k,     { Check feasibility   }
                   primal, dual);              { of initial solution }

  if not(primal) then                                { Handle primal }
```

```
  begin                                                  { feasibility   }
    i := 0; alpha_LP := maxAlpha(x_k, dx);
    while not(primal) and (i < i_max) do
    begin
      i := i+1;
      x_k := x_0 + alpha_LP * dx;
      CheckFeasibility(A, b, c, x_k, y_k, z_k, primal, dual);
      alpha_LP := 0.9 * alpha_LP;                        { Shrink Newton }
                                                         { step size     }
    end;
    if not(primal) then
    begin
      alpha_LP := 0;
      x_k := x_0;
    end;
  end;

  { ... }

  end;

  if (alpha_LP = 0) and (alpha_DP = 0) then
    Err := 'Err: No feasible Newton step possible';

end; { procedure FeasibleNewtonStep }
```

Die soeben angegebene Prozedur `FeasibleNewtonStep` kommt als zentraler Teil des eigentlichen Newton-Verfahrens zum Einsatz, das in der folgenden Prozedur `FeasibleNewtonMethod` mit größtmöglicher Ähnlichkeit zur mathematischen Beschreibung von Algorithmus 2.4.4 programmtechnisch realisiert ist.

```
const
  maxNewtonSteps = 5;
  Epsilon        = 1E-10;

global
procedure FeasibleNewtonMethod(              { Modified interval     }
              A                  : rmatrix; { Newton methode to     }
              b, c               : rvector; { verify existence and  }
              mu                 : real;    { uniqueness of minimum }
                                            { of the actual         }
          var x_k, y_k, z_k      : rvector; { lagrangian barriere   }
          var Err                : string); { function              }
                                            {-----------------------}
var
  x_old, dx           : rvector[lb(x_k)..ub(x_k)];
  y_old, dy           : rvector[lb(y_k)..ub(y_k)];
  z_old, dz           : rvector[lb(z_k)..ub(z_k)];
  X                   : rmatrix[lb(x_k)..ub(x_k),lb(x_k)..ub(x_k)];
  Z,Z_1               : rmatrix[lb(z_k)..ub(z_k),lb(z_k)..ub(z_k)];
  r_P                 : rvector[lb(b)..ub(b)];
  r_D                 : rvector[lb(c)..ub(c)];
```

```
  e, r_mu              : rvector[lb(c)..ub(c)];
  approx, primal, dual : boolean;
  k, ErrCode           : integer;

begin

    k := 0;

    repeat                                              { Newton step }
      k := k + 1; approx := FALSE;                      {-------------}

      x_old := x_k; y_old := y_k; z_old := z_k;

      e := 1; X := diag(x_old); Z := diag(z_old);
      r_P  := b - A*x_old;
      r_D  := transp(A)*y_old + z_old - c;
      r_mu := mu*e - (X*Z)*e;
      Z_1 := Z; diag_minv(Z_1, ErrCode);{ Z_1 = approx. inverse of Z }
      if (ErrCode <> 0) then
        Err := 'Err: Inversion of diagonal matrix failed';

      if (Err = 'No Error') then
      begin                                  { Compute Newton correction }
        lss_aprx(A*(Z_1*X)*transp(A),    {---------------------------}
                 r_P - A*Z_1*r_mu - A*(X*Z_1)*r_D,
                 dy, ErrCode);
        if (ErrCode <> 0) then
          Err := 'Err: Linear system solver failed';
        dz := -r_D - transp(A)*dy;
        dx := Z_1*( r_mu - X*dz );
      end;

      if (Err = 'No Error') then
      FeasibleNewtonStep(A, b, c,            { Determine new iterate }
                         x_old, y_old, z_old,{----------------------}
                         dx, dy, dz,
                         x_k, y_k, z_k, Err);

      if (Err = 'No Error') then             { Determine accuracy of }
      if (MaxNorm(x_k - x_old) < Epsilon) and{ approximation         }
         (MaxNorm(y_k - y_old) < Epsilon) and{----------------------}
         (MaxNorm(z_k - z_old) < Epsilon)
        then approx := true;

    until approx or (Err <> 'No Error') or (k = maxNewtonSteps);

end; {FeasibleNewtonMethod}
```

3.2.4 Ergebnisverifikation

Im Rahmen der Ergebnisverifikation werden garantierte Aussagen über:

- Zulässigkeit des Startpunktes, aller Iterierten im Iterationsprozeß und der optimalen Lösung;
- Verifikation der Existenz einer optimalen Lösung;
- Einschluß des optimalen Zielfunktionswertes;
- Gegebenenfalls die Eindeutigkeit einer optimalen Lösung.

Die auf dem Rechner verifizierbare Zulässigkeitsprüfung wird, wie in Abschnitt 3.2.1 angegeben, mit Hilfe des Programmierwerkzeuges PASCAL–XSC und der hierdurch verwendbaren, präzise definierten Rechnerarithmetik realisiert.

Die Frage der Existenz einer Lösung für ein gegebenes lineares Optimierungsproblem kann durch den Nachweis der Voraussetzungen für den Dualitätssatz 2.3.3 ebenfalls durch die Prozedur `CheckFeasibility` beantwortet werden. Falls es möglich ist, die Zulässigkeit eines Punktes $x^{(k)}$ des primalen Problems (LP) und eines Punktes $(y^{(k)}, z^{(k)})$ des dualen Problems (DP) zu verifizieren, liefert Satz 2.3.3 sofort den Existenzbeweis für eine optimale Lösung des Optimierungsproblems.

Einen garantierten Einschluß für den optimalen Zielfunktionswert liefert der Dualitätssatz 2.3.1, nach dem für alle zulässigen Punkte $x^{(k)}$ des primalen Problems (LP) und $(y^{(k)}, z^{(k)})$ des dualen Problems (DP) die Beziehung $f(x^{(k)}) \geq g(y^{(k)})$ gilt. Implementiert ist diese Berechnung in Form des in Abschnitt 2.4.5 angegebenen Algorithmus DualityGap, der sowohl für reelle als auch für Intervallparameter zur Verfügung steht.

```
global
function DualityGap( b, c    : rvector;       { Determine duality gap}
                     x, y, z : rvector): real;{ for real arguments   }
begin                                         {-----------------------}
  DualityGap := #>(c*x - b*y);
end;

global
function DualityGap( b, c    : rvector;       { Determine duality gap}
                     x, y, z : ivector): real;{ for interval arg.    }
begin                                         {-----------------------}
  DualityGap := sup(##(c*x - b*y));
end;
```

Um ggf. die Eindeutigkeit der Lösung des linearen Optimierungsproblems, die insbesondere den Fall mehrere optimaler Eckpunkte des zulässigen Bereichs ausschließt, nachzuweisen, werden die Bedingungen des Brouwerschen Fixpunktsatzes 1.9.3 in Form einer Intervallversion des Newton-Verfahrens geprüft. Das in Abschnitt 1.9.2 vorgestellte Verfahren, welches in

Algorithmus 1.9.2 Verification zusammengefaßt wurde, ist in seiner mathematischen Formulierung fast unverändert in eine PASCAL–XSC Prozedur übertragen worden. Hierfür war es notwendig, die primalen und dualen Lösungsvektoren $\tilde{x}, \tilde{y}, \tilde{z}$ zu einem Vektor $\tilde{u}$ zusammenzufassen. Die Bestimmung des Gradienten und der Hesse-Matrix erfolgt bezüglich der Eingangsdaten $A, b, c, \mu, \tilde{x}, \tilde{y}$ und $\tilde{z}$ entsprechend den Herleitungen (2.14) und (2.15) aus Abschnitt 2.3.2. Die für die Realisierung des Inklusionsschrittes notwendige ε-Aufblähung wird mit einem in praktischen Tests [46] ermittelten ε-Wert von 10^{-5} durchgeführt.

```
global
procedure Verification(                          { Newton methode to     }
        A                          : rmatrix; { verify existence and  }
        b, c                       : rvector; { uniqueness of minimum }
        mu                         : real;    { of the actual         }
        x_tilde, y_tilde, z_tilde  : rvector; { lagrangian barriere   }
    var x_incl, y_incl, z_incl     : ivector; { function              }
    var unique                     : boolean; {-----------------------}
    var Err                        : string );

var
  u_tilde          : rvector[1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde)];
  u_incl, u_old, h : ivector[1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde)];
  R                : rmatrix[1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde),
                             1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde)];
  help,            : imatrix[1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde),
                             1..ub(x_tilde)+ub(y_tilde)+ub(z_tilde)];
  Basis            : rvector[1..ub(y_tilde)];
  EpsilonInflation,
  ErrCode          : integer;

begin

  DetermineBasis(x_tilde, y_tilde, z_tilde, unique, Basis, Err);

  if unique and (Err = 'No Error') then
  begin
              { Compose one vector out of primal and dual solutions }
    u_tilde := compose(x_tilde, y_tilde, z_tilde);
    EpsilonInflation := 0;

    R := hess(A,b,c, mu, u_tilde);     { Compute approximate inverse }
    minv(R, ErrCode);                  { of Hessian matrix           }
    if (ErrCode <> 0) then
      writeln('matrix inversion failed in verification');

    u_incl := blow(intval(null(u_incl)),eps);      { Perform epsilon }
                                                   { inflation       }
    h := R * grad(A, b, c, mu, u_tilde);

    repeat                                        { Verification step }
                                                  {-------------------}
```

```
      EpsilonInflation := EpsilonInflation + 1; unique := false;

      u_old := u_incl;

      help := hess(A,b,c, mu, u_tilde +* (u_tilde + u_incl) );
      u_incl := -h + ##( id(R) - R*help) * u_incl;

      if (u_incl in u_old)
        then unique := true                          { Verify uniqueness }
        else u_incl := blow(u_incl,eps); { Perform epsilon inflation }

    until unique or (EpsilonInflation = maxEpsilonInflations);

    decompose(u_incl, x_incl, y_incl, z_incl);{ Restore primal and   }
    x_incl := x_tilde + x_incl;               { dual solution vectors}
    y_incl := y_tilde + y_incl;
    z_incl := z_tilde + z_incl;

  end;

end; {procedure Verification}
```

3.3 Primal-duale Innere-Punkt-Methode

Das vollständige Innere-Punkt-Verfahren wird durch die Prozedur `PrimalDualMethod`, die in einem PASCAL–XSC Modul `PrimalDualIPM` vereinbart ist, realisiert. Um den modularen Aufbau dieses Programmteils zu erläutern, sind in Abbildung 3.1 in einer baumartigen Struktur die Beziehungen der Einzelmodule zueinander und zum Hauptmodul `PrimalDualIPM` angegeben.

Das `FeasibilityModule` stellt die Prozedur `CheckFeasibility` für reelle und Intervallparameter zur Verfügung. Eingangsparameter der in Abschnitt 3.2.1 beschriebenen Prozedur `CheckFeasibility` zur Zulässigkeitsprüfung sind die Daten des linearen Optimierungsproblems (LP) A, b, c und die reellen bzw. intervallmäßigen Daten über die primalen x und dualen (y, z) Punkte.

Im `VerificationModule` werden sowohl die Funktion `DualityGap` als auch die Prozedur `Verification` angegeben. Die in Abschnitt 3.2.2 näher beschriebene Funktion `DualityGap` zur Berechnung der Dualitätslücke (vgl. Definition 2.3.2) und damit eines Einschlusses des optimalen Zielfunktionswertes benötigt als Eingabeparameter die Koeffizienten der primalen und dualen Zielfunktion c und b sowie die Koeffizienten der aktuellen primal und dual zulässigen Punkte x und (y, z). Die Prozedur `Verification` überprüft, ausgehend von einer Näherung $\widetilde{x}, \widetilde{y}, \widetilde{z}$ die Eindeutigkeit einer

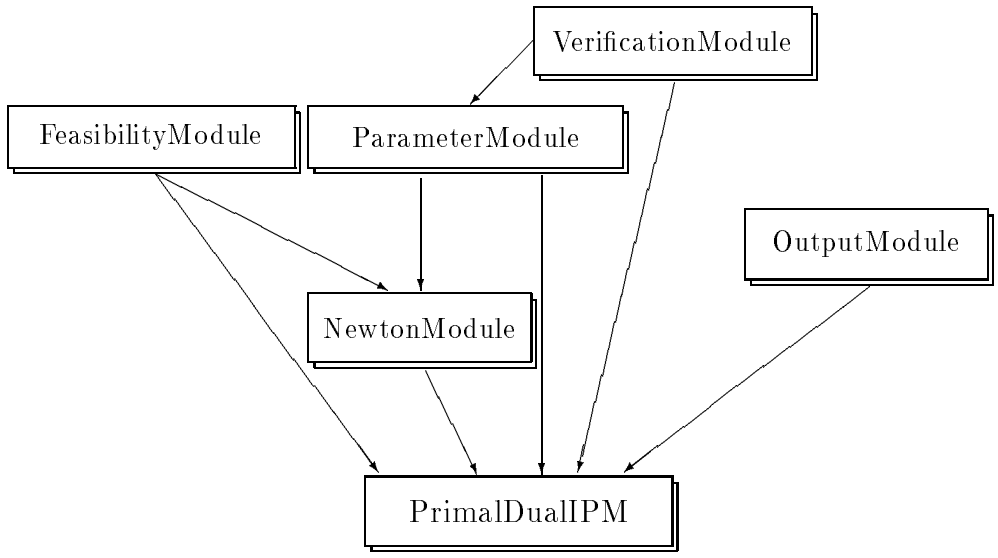

Abbildung 3.1: Der modulare Programmaufbau

Nullstelle für die gegebene Lagrangesche Barriere-Funktion, die durch die Daten des linearen Optimierungsproblems (LP) A, b, c und den aktuellen Barriere-Parameter μ bestimmt ist. Vergleiche hierzu Abschnitt 3.2.2.

Das `ParameterModule` stellt die Funktionen `maxAlpha` und `GetMu` bereit. `maxAlpha` wird zur Berechnung der Schrittweitenparameter α_{LP} und α_{DP} im Newton-Iterationsschritt eingesetzt. Für weitere Erläuterungen wird auf die Abschnitte 1.9, 2.4.2.1 und 3.2.3 verwiesen.

Eine modifizierte Version des Newton-Verfahrens wird in Form der Prozedur `FeasibleNewtonMethod` im `NewtonModule` vereinbart. Die Besonderheiten des Verfahrens sind dem Abschnitt 2.4.3 zu entnehmen. Eingangsparameter sind hierbei die Lagrangesche Barriere-Funktion, die durch die Daten des linearen Optimierungsproblems (LP) A, b, c und den aktuellen Barriere-Parameter μ bestimmt ist. Als Ergebnis wird eine Näherung der Nullstelle des Gradienten der Lagrangeschen Barriere-Funktion in den Vektoren $x^{(k)}, y^{(k)}, z^{(k)}$ zurückgegeben.

Das `OutputModule` steuert und formatiert die Ausgabe der Zwischenergebnisse und des Endergebnisses sowie die Präsentation der verifizierten

Aussagen über Zulässigkeit, Existenz und Eindeutigkeit der Lösung des linearen Optimierungsproblems (LP).

Die eigentliche Schnittstelle der selbstverifizierenden primal-dualen Innere-Punkt-Methode ist im Modul `PrimalDualIPM` implementiert. Die globale Prozedur `PrimalDualMethod` löst ein lineares Optimierungsproblem, das in Standardform vorliegt. Notwendige Eingabeparameter sind die Daten des (LP) A, b, c sowie primale und duale Startpunkte $x^{(0)}$ und $(y^{(0)}, z^{(0)})$. Nach einer Überprüfung der Zulässigkeit der Startpunkte wird das Barriere-Verfahren gestartet. Hierzu wird der aktuelle Barriere-Parameter μ bestimmt, die Nullstelle des Gradienten der Lagrangeschen Barriere-Funktion (bzgl. μ) bestimmt und die sich daraus ergebende Dualitätslücke berechnet. Falls das Barriere-Iterationsverfahren erfolgreich verlief, wird nachgewiesen, ob das gegebene Optimierungsproblem eine eindeutige optimale Lösung besitzt. Als Ausgabeparameter liefert die Prozedur neben einer *zulässigen* Näherung $x^{(l)}, y^{(l)}, z^{(l)}$ der optimalen Lösung, die den exakten optimalen Zielfunktionswert auf eine relative Genauigkeit von $\varepsilon = 10^{-10}$ approximiert, auch einen verifizierten Einschluß des exakten optimale Zielfunktionswertes. Durch den Nachweis der Zulässigkeit der primalen und dualen Näherungslösungen ist gleichzeitig auch die Existenz einer optimalen Lösung garantiert. Zusätzlich wird, falls möglich, die Eindeutigkeit der Lösung für das gegebene (LP) nachgewiesen und eine entsprechende Information ausgegeben. Die Implementierung des in Abschnitt 2.4.6 angegebenen Algorithmus 2.4.6 ist im folgenden angegeben.

```
const
  maxBarriereSteps  = 100;
  Epsilon           = 1E-10;

global procedure PrimalDualMethod(       { Primal-dual interior      }
       A                     : rmatrix; { point method for solving  }
       b, c, x_0, y_0, z_0   : rvector; { linear optimization       }
   var x_l, y_l, z_l         : rvector; { problems with verification}
   var x_incl, y_incl, z_incl : ivector; { of the optimality and     }
   var value                 : interval;{ feasibility of the result.}
   var unique                : boolean; { If possible, the          }
   var Msg                   : string;  { uniqueness of the optimal }
   var Err                   : string); { solution is guaranteed.   }
var                                      {----------------------------}
  l            : integer;
  mu, Gap      : real;
  primal, dual : boolean;

begin

  Err := 'No Error'; Msg := ''; l := 0;            { Initialisation }
  mu := 1E+5; unique := false;                     {----------------}
```

```
  x_1 := x_0; y_1 := y_0; z_1 := z_0;

  CheckFeasibility(A,b,c, x_0,y_0,z_0, primal,dual); { Check feasi-  }
  if not(primal and dual) then                       { bility of ini-}
  begin                                              { tial solution }
    if not(primal) then
      Err := 'Err: Initial solution primal infeasible';
    if not(dual)   then
      Err := 'Err: Initial solution dual infeasible';
    if (not primal) and (not dual) then
       Err := 'Err: Initial solution primal and dual infeasible';
  end;

  if (Err = 'No Error') then
  repeat                                             { Barriere step }
                                                     {---------------}
    l := l+1;
                                  { Determine new barriere parameter }
    if (l>1) then mu := GetMu(b,c, x_1,y_1,z_1, Err);

    { Contruct the new pair of Lagrangian barriere functions related }
    { to the barriere parameter mu                                   }
    if (Err = 'No Error') then
    FeasibleNewtonMethod(A,b,c, mu,  { Determine zero of the gradient}
                         x_1,y_1,z_1,{ of the pair of Lagrangian     }
                         Err);       { barriere functions using a    }
                                     { modified Newton method        }

    Gap := DualityGap(b,c, x_1,y_1,z_1);     { Determine duality gap }

    if (l > maxBarriereSteps) and (Gap > Epsilon) then
      Err := 'Err: Maximum number of barriere steps exceeded';

    ScreenOutput(l, A,b,c, x_1,y_1,z_1, Gap);

  until (Gap <= Epsilon) or (Err <> 'No Error');

  if (Err = 'No Error') then
  begin
    value := (c * x_1) +*  (b * y_1);      { Determine optimal value }

    Verification(A, b, c, mu, x_1, y_1, z_1,     { Verify uniqueness }
                 x_incl, y_incl, z_incl, unique);{ of the zero of    }
                                                 { the gradient      }
    if unique then Msg := ' Uniqueness of optimal solution verified.';
  end;

end; {procedure PrimalDualMethod}
```

3.4 Einschluß aller optimalen Basis-Lösungen

Die in Abschnitt 2.4.7 vorgestellten Algorithmen zur Besimmung von Einschließungen für den optimalen Zielfunktionswert z_{opt}, die Menge aller optimalen Basis-Indexmengen $\mathcal{V}_{opt}$ und die Menge aller optimalen Basis-Lösungen $\mathcal{X}_{opt}$ werden im Modul `EnclosureModule` zusammengefaßt. Auf die Implementierung des Datentyps einer linear verketteten Liste zur Speicherung der Mengenlisten V und X kann an dieser Stelle nicht eingegangen werden. Eine Dokumentation kann der Veröffentlichung [20] entnommen werden. Der Datentyp `BaseList` wird in einem separaten Modul `lop_ari` definiert.

Die Algorithmen ComputeTableau , BasisStable , PossiblyOptimalSolution , EmptySolutionSet , Unbounded , NeighboringList und die globale Prozedur EncloseBasicSolution werden mit Hilfe der wissenschaftlichen Programmiersprache PASCAL–XSC nahezu unverändert in den folgenden Quelltext übertragen.

```
{------------------------------------------------------------------------}
{ Compute the interval tableau                                           }
{                          ([z]   | [d]↑t    )                         }
{                  [T] := (------+----------)                           }
{                          ([x]_B | [H]      ).                        }
{------------------------------------------------------------------------}
procedure ComputeTableau(      A             : rmatrix;
                               b, c          : rvector;
                               Base, NonBase : IndexSet;
                          var TT             : imatrix;
                          var Err            : integer);
var
  A_B               : rmatrix[1..ub(A,1), 1..size(Base)];
  A_N               : rmatrix[1..ub(A,1), 1..size(NonBase)];
  c_B               : rvector[1..size(Base)];
  c_N               : rvector[1..size(NonBase)];
  xx_B, yy          : ivector[1..size(Base)];
  dd                : ivector[1..size(NonBase)];
  HH                : imatrix[1..size(Base), 1..size(NonBase)];
  zz                : interval;
  Index, local_Error : integer;
  i, j, m, n        : integer;

begin { procedure ComputeTableau }

  m := ub(A,1);
  n := ub(A,2);

  { Determine submatrices and subvectors according to index sets Base}
  { and NonBase                                                      }
```

```
  A_B := extract(A, Base);                    { Determine submatrices}
  A_N := extract(A, NonBase);                 { and subvectors accor-}
  c_B := extract(c, Base);                    { ding to B and N      }
  c_N := extract(c, NonBase);                 {----------------------}

  { Solve linear systems of equations }
  lss(A_B, b, xx_B, local_Error);

  if (local_Error = NoError) then
    lss(transp(A_B), c_B, yy, local_Error);

  Index := 0;
  if (local_Error = NoError) then
    for i:=1 to (n-m) do
      begin
        Index := NextIndex(Index,NonBase);
        if (local_Error=NoError) then
        lss(A_B, A[*,Index], HH[*,i], local_Error);
      end;

  if (local_Error = NoError) then
    begin
      dd      := transp(A_N) * yy - c_N;      { Compute components of }
      zz      := (c_B * xx_B) ** (b * yy);    { interval tableau      }
      TT[1,1] := zz;                          {-----------------------}
      for j:=2 to (n-m+1) do
        TT[1,j] := dd[j-1];
      for i:=2 to (m+1) do
        TT[i,1] := xx_B[i-1];
      for i:=2 to (m+1) do
        for j:=2 to (n-m+1) do
          TT[i,j] := HH[i-1,j-1];
    end
  else Err := SubmatrixSingular;

end; { procedure ComputeTableau }

{-------------------------------------------------------------------------}
{ Determine whether the interval tableau [T] represents a basis           }
{ stable solution, i.e. ([x]_B.inf > 0) and ([d].inf > 0).                }
{-------------------------------------------------------------------------}
function BasisStable(TT : imatrix) : boolean;
var IsStable : boolean;
    i,j      : integer;
begin
  IsStable := true; i := 1; j := 1;
  while ( (i < ub(TT,1)) and IsStable ) do
    begin
      i := i+1; { [x]_B = [T_i,1] (i = 2 .. m+1) }
      if (TT[i,1].inf <= 0) then IsStable := false;
    end;
  while ( (j < ub(TT,2)) and IsStable ) do
    begin
      j := j+1; { [d] = [T_1,j] (j = 2 .. n-m+1) }
      if (TT[1,j].inf <= 0) then IsStable := false;
```

```
    end;
  BasisStable := IsStable;
end;

{------------------------------------------------------------------------}
{ Determine whether the interval tableau [T] represents a possibly       }
{ optimal solution, i.e ([x]_B.sup >= 0) and ([d].sup >= 0)              }
{------------------------------------------------------------------------}
function PossiblyOptimalSolution(TT : imatrix) : boolean;
var IsOptimal : boolean;
    i,j       : integer;
begin
  IsOptimal := true; i := 1; j := 1;
  while ( (i < ub(TT,1)) and IsOptimal ) do
    begin
      i := i+1; { [x]_B = [T_i,1] (i = 2 .. m+1) }
      if not( TT[i,1].sup >= 0 ) then IsOptimal := false;
    end;
  while ( (j < ub(TT,2)) and IsOptimal ) do
    begin
      j := j+1; { [d] = [T_1,j] (j = 2 .. n-m+1) }
      if not( TT[1,j].sup >= 0 ) then IsOptimal := false;
    end;
  PossiblyOptimalSolution := IsOptimal;
end;

{------------------------------------------------------------------------}
{ Check whether the set of feasible solutions is empty, i.e. not for     }
{ all(beta in B) with (0 in [x]_beta) there exists a (nu in N) with      }
{ ([H]_beta,nu < 0)                                                      }
{------------------------------------------------------------------------}
function EmptySolutionSet(TT : imatrix) : boolean;
var IsEmpty : boolean;
    i,j     : integer;
begin
  IsEmpty := false; i := 1;
  while (not IsEmpty) and (i < ub(TT,1)) do
    begin
      i := i+1;
      if (0 in TT[i,1]) then
        begin
          IsEmpty := true;
          j := 1;
          while IsEmpty and (j < ub(TT,2)) do
            begin
              j := j+1;
              if (TT[i,j].sup < 0) then IsEmpty := false;
            end;
        end;
    end;
  EmptySolutionSet := IsEmpty;
end;

{------------------------------------------------------------------------}
{ Check whether the objective function is unbounded, i.e. not for all}
```

```
{ (nu in N) with (0 in [d]_nu) there exists a (beta in B) with      }
{ ([H]_beta,nu > 0)                                                }
{------------------------------------------------------------------}
function Unbounded(TT : imatrix) : boolean;
var IsUnbounded : boolean;
    i,j         : integer;
begin
  IsUnbounded := false; j := 1;
  while (not IsUnbounded) and (j < ub(TT,2)) do
    begin
      j := j+1;
      if (0 in TT[1,j]) then
        begin
          IsUnbounded := true;
          i := 1;
          while IsUnbounded and (i < ub(TT,1)) do
            begin
              i := i+1;
              if (TT[i,j].inf > 0) then IsUnbounded := false;
            end;
        end;
    end;
  Unbounded := IsUnbounded;
end;

{------------------------------------------------------------------}
{ Determine list of neighboring basic index sets L for index set Base}
{ that are good candidates for being optimal basic index sets.     }
{------------------------------------------------------------------}
function NeighboringList(TT : imatrix; Base, NonBase : IndexSet) :
                                                          BaseList;
var
  L                : BaseList;
  NewBase          : IndexSet;
  i, j, m, n       : integer;
  beta, nu, Counter : integer;
  colmin, rowmax   : real;
  xx, dd, HH       : interval;

begin
  m := size(Base); n := m + size(NonBase);          { Initialization }
  L := nil;                                         {----------------}

  colmin := maxReal;                    { Search for candidates (nu   }
  for j := 1 to (n-m) do                { in N) and (beta in B) by    }
    begin                               { determining of the minimum  }
      dd := TT[1,j+1];                  { of the quotients            }
      if (0 in dd) then                 { [x]_beta / [H]_beta,nu      }
        begin                           {-----------------------------}
          for i:=1 to m do                  { Determine minimum of    }
            begin                           {  [x]_beta / [H]_beta,nu }
              xx := TT[i+1,1];              { for column nu of [T]    }
              HH := TT[i+1,j+1];            {-------------------------}
              if ((HH.Inf > 0) and (xx.Sup/<HH.Inf < colmin)) then
                colmin := xx.Sup/<HH.Inf;
```

```
          end; { (beta in B) }
        for i:=1 to m do                { Determine candidate (beta}
          begin                         { in B) for exchange of    }
            xx := TT[i+1,1];            { indices                  }
            HH := TT[i+1,j+1];          {--------------------------}
            if (HH.Sup > 0) and (xx.Inf/<HH.Sup <= colmin) then
              begin                             { Determine new index }
                beta := GetIndex(i,Base);       { set and add it to L }
                nu   := GetIndex(j,NonBase); {----------------------}
                NewBase := Base - [beta] + [nu];
                insert(L, NewBase);
              end;
          end; { (beta in B) }
      end;
    end; { (nu in N) }

  rowmax := -maxReal;                   { Search for candidates (beta }
  for i := 1 to m do                    { in B) and (nu in N) by      }
    begin                               { determining of the maximum  }
      xx := TT[i+1,1];                  { of the quotients            }
      if (0 in xx) then                 { [d]_nu / [H]_beta,nu        }
        begin                           {-----------------------------}
          for j:=1 to (n-m) do                  { Determine maximum of  }
            begin                               { [d]_nu / [H]_beta,nu  }
              dd := TT[1,j+1];                  { for row beta of [T]   }
              HH := TT[i+1,j+1];                {----------------------}
              if ((HH.Sup < 0) and (dd.Inf/>HH.Sup > rowmax)) then
                rowmax := dd.Sup/>HH.Sup;
            end; { (nu in N) }
          for j:=1 to (n-m) do                  { Determine candidate   }
            begin                               { (nu in N) for exchange}
              dd := TT[1,j+1];                  { of indices            }
              HH := TT[i+1,j+1];                {----------------------}
              if (HH.Inf < 0) and (dd.Inf/>HH.Inf >= rowmax) then
                begin                               { Determine new index }
                  beta := GetIndex(i,Base);         { set and add it to L }
                  nu := GetIndex(j,NonBase);    {----------------------}
                  NewBase := Base - [beta] + [nu];
                  insert(L, NewBase);
                end;
            end; { (nu in N) }
      end;
    end; { (beta in B) }

  NeighboringList := L;
end;  { function NeighboringList }

{-------------------------------------------------------------------}
{ Recursive procedure to determine list of possible basic index sets,}
{ if the number of base indices is not m.                           }
{-------------------------------------------------------------------}
procedure CandidateList(  Base       : IndexSet;
                          NonBase    : IndexSet;
                          depth      : Integer ;
                      var CList      : BaseList);
```

```
var i,alpha    : integer;
    Used_alpha : IndexSet;

begin
  if (depth > 0) then
    begin
      if (depth > 1) then
        begin
          Used_alpha := [];
          for i := 1 to ((size(NonBase)-depth)+1) do
            begin
              alpha := GetIndex(i,NonBase);
              Used_alpha := Used_alpha + [alpha];
              CandidateList(Base + [alpha], NonBase - Used_alpha,
                            depth-1, CList);
            end;
        end
      else
        for i := 1 to size(NonBase) do
          begin
            alpha := Getindex(i, NonBase);
            Base := Base + [alpha];
            insert(CList, Base);
            Base := Base - [alpha];
          end;
    end;

  if (depth = 0) then insert(CList, Base);
end;

{------------------------------------------------------------------------}
{ Determination of enclosures for the solutions of a linear opti-        }
{ mization problem:          ( z = c↑t * x = max! )                      }
{                      (LP)  (     A * x = b      )                      }
{                            (      x >= 0        )                      }
{   with an initial optimal basic index set.                             }
{------------------------------------------------------------------------}
global procedure EncloseBasicSolution
                         (      A                       : rmatrix;
                                b, c, B_start_Vector    : rvector;
                           var  z                       : interval;
                           var  V                       : rmatrix;
                           var  X                       : imatrix;
                           var  No, Err                 : integer   );

var
  B_start           : IndexSet;
  Base, NonBase     : IndexSet;
  L, CList, E       : BaseList;
  TT                : imatrix[1..1+(ub(b)), 1..1+(ub(c)-ub(b))];
  k, m, n, i, maxNo : integer;
  depth             : integer;

begin { global procedure EncloseBasicSolution }
```

```
Err := NoError;

m := ub(A,1); n := ub(A,2);       { Determine dimension of problem }
                                  {--------------------------------}
if (n <= m) or (m < 1) or (n < 1) or (n > maxDim){ Check dimensions}
            or (ub(V,1) <> ub(X,1)) then         {-----------------}
  Err := WrongDimension
else { Err <> WrongDimension }
  begin
    depth := 0;
    B_start := []; E := nil; CList := nil;        { Initialization }
    k := 0; V := 0; X := 0; No := 0; maxNo := ub(V,1);

    for i:=1 to m do                  { Determination of start base }
      if (trunc(B_start_Vector[i])=-1) then depth := depth+1
      else  B_start := B_start +[trunc(B_start_Vector[i])] ;
    NonBase := [1..n]-B_start;
    CandidateList(B_start,NonBase,depth,CList);

    repeat { until empty(CList) }

      Base := select(CList);                   { Select Base out of}
      NonBase := [1..n] - Base;                { CList and update  }
      delete(CList, Base); insert(E, Base);    { examined list E   }
                                               {-------------------}
      ComputeTableau(A, b, c, Base, NonBase, TT, Err);

      if (Err = NoError) then

        if BasisStable(TT) then
          begin
            No          := No + 1;              { Store unique     }
            z       := TT[1,1];                 { optimal solution }
            X[No,*] := Solution(TT, Base, NonBase); {--------------}
            V[No,*] := Base;
          end

        else { not basis stable }
          begin
            if EmptySolutionSet(TT) then Err := SolutionSetIsEmpty
            else if Unbounded(TT)   then Err := FunctionUnbounded

            else if PossiblyOptimalSolution(TT)
                    and (Err = NoError) then
              begin
                No          := No + 1;             { Store optimal }
                if (No = 1) then z:= TT[1,1]       { solution      }
                            else z:= TT[1,1] +* z; {---------------}
                X[No,*] := Solution(TT, Base, NonBase);
                V[No,*] := Base;

                L := NeighboringList(TT,Base,NonBase); { Determine }
                remove(L, E); remove(L, CList);{list of neighboring}
                                               { basic index sets  }
                                               {-------------------}
```

```
                append(CList, L); { Compute new list of candidates }
              end;                {-------------------------------}
            end;

        k := k+1;

      until empty(CList) or (Err <> NoError)
            or (k = kmax) or (No = maxNo);

      if (Err = NoError) then                  { Determine error code }
        if (not empty(CList)) and (k = kmax) then {------------------}
                              Err := IterationError
        else if (No = 0) then Err := StartIndexSetNotOptimal
        else if (No = maxNo) and (not empty(CList)) then
                              Err := SolutionMatrixTooSmall;

      FreeAll(CList); FreeAll(E);

    end; { Err <> WrongDimension }

end;  { global procedure EncloseBasicSolution }

{-------------------------------------------------------------------------}
{ Module initialization part                                              }
{-------------------------------------------------------------------------}
begin
  Rplus := intval(0,maxReal);
end.
```

Kapitel 4

Numerische Tests und Anwendungsbeispiele

Zum Test des in dieser Arbeit vorgestellten Algorithmus wurden eine Vielzahl aus der Literatur bekannter Testfunktionen verwandt. Im folgenden wird das Verhalten des Algorithmus für verschiedene kritische Problemfälle betrachtet. Bei der ersten Gruppe handelt es sich um eine aus der Literatur (vgl. z.B. [20], [35], [58]) bzw. aus der öffentlich zugänglichen Datenbank *Netlib* [49] stammende Sammlung von Standardtestproblemen, die oft zum Vergleich von verschiedenen Algorithmen zur linearen Optimierung herangezogen werden. In der zweiten Gruppe finden sich einige Beispiele, die während der Entwicklungsphase des Algorithmus Verwendung fanden, um dessen Arbeitsweise zu kontrollieren. Für jedes Testbeispiel wird zu Beginn der entsprechenden Abschnitte die Definition des linearen Optimierungsproblems angegeben. Als Resultat folgen die Ergebnisse des Verfahrens, d.h.

1. Für die verifizierte Innere-Punkt-Methode Angaben über
   (a) die Zahl der benötigten Iterationsschritte,
   (b) die Zulässigkeit der primalen und dualen Näherungslösung,
   (c) die Existenz einer optimalen Lösung,
   (d) einen Einschluß des optimalen Zielfunktionswertes mit einer relativen Toleranz von $\varepsilon = 10^{-10}$,
   (e) (gegebenenfalls) die Eindeutigkeit einer optimalen Lösung und
   (f) eine Näherung $\widetilde{x}$ des optimalen Lösungsvektors $\widehat{x}$.

2. Für den anschließenden Simplex-Schritt
   (a) ein Einschluß des optimalen Zielfunktionswertes
   (b) die Anzahl der optimalen Basis-Indexmengen und
   (c) Einschließungen der zugehörigen Basis-Lösungsvektoren.

Eine typische Bildschirmeingabe und -ausgabe ist im Anhang angegeben.

4.1 Standardtestprobleme

Beispiel 4.1.1 Als erstes wird das in der Einleitung angegebene Optimierungsproblem aus der Produktionsplanung behandelt [35]. Die Standardform des zu einem Minimierungsproblem umformulierten (LP) lautet:

$$\text{minimiere} \quad f(x) = -5x_1 - 8x_2 - 4x_3$$

$$\text{u.d.NB.} \quad \begin{pmatrix} 1 & 3 & 2 & 1 & 0 & 0 & 0 \\ 4 & 2 & 1 & 0 & 1 & 0 & 0 \\ 3 & 4 & 3 & 0 & 0 & 1 & 0 \\ 2 & 3 & 5 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 30 \\ 25 \\ 45 \\ 50 \end{pmatrix}$$

$$\text{und} \quad x_i, s_i \geq 0$$

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode
   (a) Anzahl der Iterationsschritte: 21
   (b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig
   (c) (LP) besitzt endliche optimale Lösung: ja
   (d) Einschluß des optimalen Zielfunktionswertes: $[-83.^{49999999997}_{50000000004}]$
   (e) (LP) besitzt eine eindeutige optimale Lösung: ja
   (f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x} = \begin{pmatrix} 1.499999999998054 \\ 9.499999999994822 \\ 7.117387809820147 \cdot 10^{-12} \\ 3.558693904338395 \cdot 10^{-12} \\ 1.118446655712257 \cdot 10^{-11} \\ 2.500000000005568 \\ 1.849999999998408 \cdot 10^{1} \end{pmatrix}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

   (a) Einschluß des optimalen Zielfunktionswertes: $[-83.5]$

   (b) Anzahl der optimalen Basis-Indexmengen: 1

   (c) Einschließungen der optimalen Basis-Lösung(en)

$$[x] = \begin{pmatrix} [1.5] \\ [9.5] \\ [0.0] \\ [0.0] \\ [0.0] \\ [2.5] \\ [18.5] \end{pmatrix}$$

Dies bedeutet, daß der *maximale Gewinn* für das ursprüngliche Maximierungsproblem 83.5 Geldeinheiten beträgt, wobei von Produkt P1 1.5 Einheiten und von Produkt P2 9.5 Einheiten hergestellt werden, während das Produkt P3 nicht produziert wird. Die Resultate konnten exakt, d.h. als Punktintervalle bestimmt werden.

Um die Fähigkeiten des hier entwickelten Verfahrens zu demonstrieren, wird an dieser Stelle ein Transportproblem untersucht, das bereits von Jansen, Roos und Terlaky [23] für den Vergleich zahlreicher (z.T. kommerzieller) Programmpakete zur linearen Optimierung (LP-Pakete) verwandt wurde.

Beispiel 4.1.2 Man betrachte das unbalancierte Transportproblem für drei Unternehmen und drei Märkte.

$$\begin{array}{rll} \text{minimiere} & f(x) = \sum_{i=1}^{3} \sum_{j=1}^{3} c_{ij} x_{ij} & \\ \text{u.d.NB.} & \sum_{j=1}^{3} x_{ij} + s_i = a_i & (i = 1,2,3) \\ & \sum_{i=1}^{3} x_{ij} - d_j = b_j & (j = 1,2,3) \\ \text{und} & x_{ij}, s_i, d_j \geq 0 & (i,j = 1,2,3) \end{array}$$

Die Transportkosten von Unternehmen i zu Markt j betragen $c_{ij} = 1$, $(i,j = 1,2,3)$. Die vorhandenen Bestände bei Unternehmen i seien $a_1 = 2$, $a_2 = 6$ und $a_3 = 5$. Die vorgegebenen Absatzmengen auf den Märkten j betragen jeweils $b_j = 3$, $(j = 1,2,3)$.

In [23] sind für dieses Transportproblem die Ergebnisse von fünf bekannten LP-Paketen (CPLEX, LINDO, OSL, PC-PROG, XPRESS-MP) veröffentlicht, die um eigene Resultate ergänzt wurden. Die verwendeten

LP-Paket	Lösungsmethode	max. Problemgröße	Preis
CPLEX	Simplex- oder Innere-Punkt-Methode	?	645 US$
LINDO	Simplex-Methode	2000	90 US$
OSL	Simplex- oder Innere-Punkt-Methode	?	1660 US$
PC-PROG	Simplex-Methode	500	1795 hFl
XPRESS-MP	Simplex-Methode	?	?
lp_solve	Simplex-Methode	30000	frei
IPM	Innere-Punkt-Methode	?	?
LOQO	Innere-Punkt-Methode	systemabhängig	frei
LPverify	Simplex- und Innere-Punkt-Methode	systemabhängig	frei

Abbildung 4.1: Programmpakete zur linearen Optimierung

LP-Pakete und die jeweiligen Resultate sind im folgenden tabellarisch zusammengestellt.

Die Angaben der Abbildung 4.1 zu den eingesetzten LP-Paketen über Lösungsmethode, maximale Problemgröße und Preis sind den Informationen des öffentlich zugänglichen WIOR Gopher Server [62] des Instituts für Wirtschaftstheorie und Operations Research (WIOR) der Universität Karlsruhe entnommen.

LP-Paket	primale Lösung									duale Lösung					
	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}	x_{31}	x_{32}	x_{33}	y_1	y_2	y_3	y_4	y_5	y_6
CPLEX	0	2	0	2	1	3	1	0	0	0	0	0	1	1	1
LINDO	2	0	0	0	0	2	1	3	1	0	0	0	1	1	1
OSL	0	2	0	2	1	3	1	0	0	0	0	0	1	1	1
PC-PROG	0	0	0	0	3	1	3	0	2	0	0	0	1	1	1
XPRESS-MP	0	0	2	3	3	0	0	0	1	0	0	0	1	1	1
lp_solve	0	0	0	0	1	3	3	2	0						
IPM	0.62	0.62	0.62	1.45	1.45	1.45	0.93	0.93	0.93	0	0	0	1	1	1
LOQO										dual unzulässig					
LPverify	siehe Abbildung 4.3									0	0	0	1	1	1

Abbildung 4.2: Ergebnisse der LP-Pakete

In Abbildung 4.2 ist zu erkennen, daß für dieses recht einfache Optimierungsproblem kaum zwei LP-Pakete identische Resultate für die primale Lösung liefern.

Mit Hilfe des in dieser Arbeit vorgestellten Verfahrens ist eine verifizierte Bestimmung eines Einschlusses des optimalen Zielfunktionswertes, einer primal und dual zulässigen Näherungslösung und darüber hinaus aller optimalen Basis-Indexmengen mit den dazugehörigen Basis-Lösungen möglich. Ein Auszug der numerischen Resultate ist in Abbildung 4.3 zusammengestellt.

LPverify	primale Lösung									Schlupfvariablen					
	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}	x_{31}	x_{32}	x_{33}	s_1	s_2	s_3	d_1	d_2	d_3
Näherung	1.66	0.16	0.01	0.01	2.10	0.01	1.26	0.80	2.82	0.12	3.76	0.12	0	0	0
$[x]^{(B1)}$	[2]	[0]	[0]	[0]	[2]	[0]	[1]	[1]	[3]	[0]	[4]	[0]	[0]	[0]	[0]
$[x]^{(B2)}$	[2]	[0]	[0]	[0]	[3]	[0]	[1]	[0]	[3]	[0]	[3]	[1]	[0]	[0]	[0]
$[x]^{(B3)}$	[1]	[0]	[0]	[0]	[3]	[0]	[2]	[0]	[3]	[1]	[3]	[0]	[0]	[0]	[0]
$[x]^{(B4)}$	[2]	[0]	[0]	[1]	[1]	[0]	[0]	[2]	[3]	[0]	[4]	[0]	[0]	[0]	[0]
$[x]^{(B5)}$	[1]	[1]	[0]	[0]	[2]	[0]	[2]	[0]	[3]	[0]	[4]	[0]	[0]	[0]	[0]
$[x]^{(B6)}$	[2]	[0]	[0]	[1]	[3]	[0]	[0]	[0]	[3]	[0]	[2]	[2]	[0]	[0]	[0]
$[x]^{(B7)}$	[0]	[0]	[0]	[0]	[3]	[1]	[3]	[0]	[2]	[2]	[2]	[0]	[0]	[0]	[0]
$[x]^{(B8)}$	[0]	[0]	[0]	[1]	[3]	[0]	[2]	[0]	[3]	[2]	[2]	[0]	[0]	[0]	[0]
$[x]^{(B9)}$	[0]	[0]	[1]	[0]	[3]	[0]	[3]	[0]	[2]	[1]	[3]	[0]	[0]	[0]	[0]
$[x]^{(B10)}$	[2]	[0]	[0]	[1]	[0]	[1]	[0]	[3]	[2]	[0]	[4]	[0]	[0]	[0]	[0]
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$[x]^{(B17)}$	[0]	[0]	[2]	[3]	[3]	[0]	[0]	[0]	[1]	[0]	[0]	[4]	[0]	[0]	[0]
$[x]^{(B22)}$	[2]	[0]	[0]	[0]	[0]	[2]	[1]	[3]	[1]	[0]	[4]	[0]	[0]	[0]	[0]
$[x]^{(B58)}$	[0]	[2]	[0]	[2]	[1]	[3]	[1]	[0]	[0]	[0]	[0]	[4]	[0]	[0]	[0]
$[x]^{(B62)}$	[0]	[0]	[0]	[0]	[1]	[3]	[3]	[2]	[0]	[2]	[2]	[0]	[0]	[0]	[0]
$[x]^{(B66)}$	[0]	[2]	[0]	[3]	[0]	[3]	[0]	[1]	[0]	[0]	[0]	[4]	[0]	[0]	[0]

Abbildung 4.3: Ergebnisse von LPverify

Das Ergebnis zeigt, daß (fast) alle LP-Pakete eine richtige, d.h. optimale Lösung für Beispiel 4.1.2 liefern. Die Frage nach weiteren optimalen Lösungen wird allerdings nicht beantwortet. In der praktischen Anwendung ist es allerdings interessant, über eine mögliche Wahlfreiheit informiert zu sein. Diese bedeutet nämlich, daß weitere Nebenbedingungen, die bisher unberücksichtigt blieben, in die Formulierung des Optimierungsproblem aufgenommen werden können.

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

   (a) Anzahl der Iterationsschritte: 7

   (b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig

   (c) (LP) besitzt endliche optimale Lösung: ja

   (d) Einschluß des optimalen Zielfunktionswertes: $\begin{bmatrix}9.00000000009\\8.99999999999\end{bmatrix}$

   (e) (LP) besitzt eine eindeutige optimale Lösung: nein

(f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x} = \begin{pmatrix} 1.662592226802926 \\ 1.158172061040926 \cdot 10^{-1} \\ 9.967411158993619 \cdot 10^{-2} \\ 7.672977827561021 \cdot 10^{-2} \\ 2.089048300101242 \\ 7.714370407636476 \cdot 10^{-2} \\ 1.260677994950243 \\ 7.951344938234448 \cdot 10^{-1} \\ 2.823182184362477 \\ 1.219164555030455 \cdot 10^{-1} \\ 3.757078217546782 \\ 1.210053268638344 \cdot 10^{-1} \\ 2.877896394625857 \cdot 10^{-11} \\ 2.877895765381127 \cdot 10^{-11} \\ 2.877896389919158 \cdot 10^{-11} \end{pmatrix}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

(a) Einschluß des optimalen Zielfunktionswertes: [9.0]

(b) Anzahl der optimalen Basis-Indexmengen: 66

(c) Einschließungen der optimalen Basis-Lösung(en)
siehe Abbildung 4.3

Beispiel 4.1.3 Um die Funktionsweise des Verfahrens im Falle numerischer Instabilität zu demonstrieren, ist im folgenden ein lineares Optimierungsproblem angegeben, dessen Zielfunktionskoeffizienten in einem Größenordnungsbereich von 100 Zehnerpotenzen schwanken. Die auftretende Restriktionsmatrix A liegt in der Nähe einer singulären Matrix. Diese Konstellation hat zur Folge, daß herkömmliche Lösungsverfahren die Problemstellung entweder gar nicht lösen können oder eine unzulässige bzw. nicht-optimale Näherungslösung als Resultat liefern. Das lineare Optimierungsproblem lautet:

$$\begin{aligned} \text{minimiere} \quad f(x) &= -2.8x_1 - 1.7x_2 - 3.1x_3 - 2.7 \cdot 10^{-50}x_4 - 10^{-100}x_5 \\ \text{u.d.NB.} \quad Ax &= \begin{pmatrix} 2.8 \\ 2.8 \\ 2.8 \\ 2.8 \\ 2.8 \end{pmatrix} \\ \text{und} \quad x &\geq 0 \end{aligned}$$

wobei

$$A^T = \begin{pmatrix} 0.9879 & 1.8 \cdot 10^{-5} & 2.8 \cdot 10^{-10} & 0 & 10^{-10} \\ 2.8 \cdot 10^{-10} & 0.99 & 1.9 \cdot 10^{-15} & 0 & 0 \\ 1.4 \cdot 10^{-20} & 2.8 \cdot 10^{-10} & 0.9879 & 0 & 0 \\ 2.8 \cdot 10^{-10} & 2.8 \cdot 10^{-10} & 0 & 0.9879 & 0 \\ 2.8 \cdot 10^{-10} & 2.8 \cdot 10^{-10} & 0 & 0 & 0.9879 \\ 1.00123 & 2.8 \cdot 10^{-10} & 0 & 0 & 0 \\ 2.8 \cdot 10^{-10} & 1 & 0 & 0 & 10^{-10} \\ 3.8 \cdot 10^{-50} & 0 & 0.9879 & 0 & 10^{-10} \\ 2.8 \cdot 10^{-10} & 2.8 \cdot 10^{-10} & 0 & 0.9879 & 0 \\ 2.8 \cdot 10^{-10} & 2.8 \cdot 10^{-10} & 0 & 0 & 0.9879 \end{pmatrix}$$

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode
   (a) Anzahl der Iterationsschritte: 21
   (b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig
   (c) (LP) besitzt endliche optimale Lösung: ja
   (d) Einschluß des optimalen Zielfunktionswertes: $[-21.530333507\frac{1}{2}]$
   (e) (LP) besitzt eine eindeutige optimale Lösung: nein
   (f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x} = \begin{pmatrix} 2.834294966715412 \\ 2.828231293237656 \\ 2.834294968320602 \\ 1.417147484563215 \\ 1.417147484419765 \\ 2.730036004349582 \cdot 10^{-12} \\ 4.511580942991243 \cdot 10^{-12} \\ 2.499083611531827 \cdot 10^{-12} \\ 1.417147484563215 \\ 1.417147484419765 \end{pmatrix}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes
   (a) Einschluß des optimalen Zielfunktionswertes: $[-21.53033350712427\frac{7}{9}]$
   (b) Anzahl der optimalen Basis-Indexmengen: 4

(c) Einschließungen der optimalen Basis-Lösung(en)

$$[x]^{(B1)} = \begin{pmatrix} [2.8342949667181^{81}_{79}] \\ [2.828231293242214^{4}_{2}] \\ [2.834294968323102^{2}_{1}] \\ [2.834294969126430^{30}_{29}] \\ [2.834294968839529^{9}_{8}] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \end{pmatrix}$$

$$[x]^{(B2)} = \begin{pmatrix} [2.8342949667181^{81}_{79}] \\ [2.828231293242214^{4}_{2}] \\ [2.834294968323102^{2}_{1}] \\ [2.834294969126430^{30}_{29}] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [2.834294968839529^{9}_{8}] \end{pmatrix}$$

$$[x]^{(B3)} = \begin{pmatrix} [2.8342949667181^{81}_{79}] \\ [2.828231293242214^{4}_{2}] \\ [2.834294968323102^{2}_{1}] \\ [0] \\ [2.834294968839529^{9}_{8}] \\ [0] \\ [0] \\ [0] \\ [2.834294969126430^{30}_{29}] \\ [0] \end{pmatrix}$$

$$[x]^{(B4)} = \begin{pmatrix} [2.834294966718 1^{81}_{79}] \\ [2.828231293242 21^{4}_{2}] \\ [2.834294968323 10^{2}_{1}] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [2.834294969126 4^{30}_{29}] \\ [2.834294968839 52^{9}_{8}] \end{pmatrix}$$

Dies bedeutet, daß 4 optimale Basis-Lösungen existieren, für die die zugehörigen Zielfunktionswerte im Intervall $[-21.53033350712 42^{7}_{9}]$ liegen.

Beispiel 4.1.4 Das zur Illustration des Barriere-Verfahrens verwandte Optimierungsproblem (vgl. Beispiel 1.6.1 und Abbildung 1.3) [58] kann ebenfalls als lineares Optimierungsproblem formuliert werden.

$$\begin{aligned} \text{minimiere} \quad & f(x) = 2x \\ \text{u.d.NB.} \quad & \begin{pmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} \\ \text{und} \quad & x, s_i \geq 0 \end{aligned}$$

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode
   (a) Anzahl der Iterationsschritte: 34
   (b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig
   (c) (LP) besitzt endliche optimale Lösung: ja
   (d) Einschluß des optimalen Zielfunktionswertes: $\begin{bmatrix} 2.0000000001 \\ 1.9999999999 \end{bmatrix}$
   (e) (LP) besitzt eine eindeutige optimale Lösung: ja
   (f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x} = \begin{pmatrix} 1.000000000008328 \\ 8.328091653078450 \cdot 10^{-12} \\ 1.999999999991672 \end{pmatrix}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

   (a) Einschluß des optimalen Zielfunktionswertes: [2.0]

   (b) Anzahl der optimalen Basis-Indexmengen: 1

   (c) Einschließungen der optimalen Basis-Lösung(en)

$$[x] = \begin{pmatrix} [1.0] \\ [0.0] \\ [2.0] \end{pmatrix}$$

Dies bedeutet, daß der minimale Funktionswert 2 beträgt und daß der Lösungsvektor $x = 1$, $s_1 = 0$ und $s_2 = 2$ ist.

Beispiel 4.1.5 Für das folgende Beispiel aus [9] benötigt der Simplex-Algorithmus $2^n - 1$ Iterationsschritte

$$\begin{aligned} &\text{minimiere} \quad f(x) = -\tfrac{10}{16}x_1 - \tfrac{1}{20}x_2 - \tfrac{1}{25}x_3 \\ &\text{u.d.NB.} \quad \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ \frac{5}{2} & 1 & 0 & 0 & 1 & 0 \\ \frac{25}{8} & \frac{5}{2} & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 25 \end{pmatrix} \\ &\text{und} \quad x_i, s_i \geq 0 \end{aligned}$$

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

   (a) Anzahl der Iterationsschritte: 4

   (b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig

   (c) (LP) besitzt endliche optimale Lösung: ja

   (d) Einschluß des optimalen Zielfunktionswertes: $[-1.^{4999999999}_{5000000001}]$

   (e) (LP) besitzt eine eindeutige optimale Lösung: ja

   (f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x} = \begin{pmatrix} 9.999999999740720 \cdot 10^{-1} \\ 2.592962541471672 \cdot 10^{-10} \\ 2.187499999910866 \cdot 10^{1} \\ 2.592962532942723 \cdot 10^{-11} \\ 2.499999999805525 \\ 3.241203167619102 \cdot 10^{-10} \end{pmatrix}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

   (a) Einschluß des optimalen Zielfunktionswertes: $[-1.5]$

   (b) Anzahl der optimalen Basis-Indexmengen: 1

   (c) Einschließungen der optimalen Basis-Lösung(en)

$$[x] = \begin{pmatrix} [1.0] \\ [0.0] \\ [2.1875 \cdot 10^1] \\ [0.0] \\ [2.5] \\ [0.0] \end{pmatrix}$$

4.2 Spezielle Testprobleme

Um die Korrektheit des Verfahrens zu prüfen, werden im folgenden einige akademische Problemstellungen behandelt, deren (ggf. eindeutige) optimale Lösung bereits bekannt ist. Die Probleme wurden für verschiedene Zielfunktionen und Dimensionen gelöst, um die Relation zwischen Anzahl der Iterationen und Problemdimension veranschaulichen zu können.

Die Standardform des Testproblems lautet:

$$\begin{array}{rcl} \text{minimiere} & f(x) = c_1 x_1 \ldots c_k x_k & \\ & x_1 + s_1 = 2 & \\ \text{u.d.NB.} & \vdots \quad \vdots & \\ & x_k + s_k = 2 & \\ \text{und} & x_i, s_i \geq 0, & \end{array}$$

d.h. die Dimension setzt sich zusammen aus $n = 2k$ Unbekannten und $m = k$ Nebenbedingungen.

Für negative Zielfunktionskoeffizienten ist der exakte optimale Zielfunktionswert $2 \cdot \sum_{i=1}^{k} c_i$. Sind alle Koeffizienten echt kleiner als 0, so ist die optimale Lösung theoretisch eindeutig. Für positive c_i ist der optimale Zielfunktionswert 0.

Beispiel 4.2.1 Für die Dimension $k = 5 \Rightarrow n = 10$ und einer Zielfunktion mit $c_i = -2.8$ für $i = 1, \ldots, k$ liefert das verifizierende Optimierungsverfahren als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

   (a) Anzahl der Iterationsschritte: 18

   (b) Zulässigkeit der Lösungsvektoren x und (y, z):
   primal + dual zulässig

   (c) (LP) besitzt endliche optimale Lösung: ja

   (d) Einschluß des optimalen Zielfunktionswertes: $[-2.^{79999999999}_{80000000001} \cdot 10^1]$

   (e) (LP) besitzt eine eindeutige optimale Lösung: ja

   (f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x}_i = \begin{cases} 1.999999999998703 & \text{für} \quad i = 1, \ldots, 5 \\ 1.297000602223812 \cdot 10^{-12} & \text{für} \quad i = 6, \ldots, 10 \end{cases}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

   (a) Einschluß des optimalen Zielfunktionswertes:
   $[-2.^{799999999999999}_{800000000000000} \cdot 10^1]$

   (b) Anzahl der optimalen Basis-Indexmengen: 1

   (c) Einschließungen der optimalen Basis-Lösung(en)

$$[x]_i = \begin{cases} [2.0] & \text{für} \quad i = 1, \ldots, 5 \\ [0.0] & \text{für} \quad i = 6, \ldots, 10 \end{cases}$$

Der optimale Zielfunktionswert ist -28 und wird auf mindestens 10 Stellen genau eingeschlossen.

Beispiel 4.2.2 Für die Dimension $k = 5 \Rightarrow n = 10$ und einer Zielfunktion mit $c_i = -2.8$ für $i = 1, \ldots, k-1$ und $c_k = 0$ liefert das verifizierende Optimierungsverfahren als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

   (a) Anzahl der Iterationsschritte: 18

   (b) Zulässigkeit der Lösungsvektoren x und (y, z):
   primal + dual zulässig

   (c) (LP) besitzt endliche optimale Lösung: ja

   (d) Einschluß des optimalen Zielfunktionswertes: $[-22.^{3999999999}_{4000000001}]$

   (e) (LP) besitzt eine eindeutige optimale Lösung: nein

(f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\widetilde{x}_i = \begin{cases} 1.999999999998248 & \text{für} \quad i = 1, \ldots, 4 \\ 1.000000000000000 & \text{für} \quad i = 5, 10 \\ 1.751892568999259 \cdot 10^{-12} & \text{für} \quad i = 6, \ldots, 9 \end{cases}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

(a) Einschluß des optimalen Zielfunktionswertes: $[-22.^{39999999999999]}_{40000000000000}$

(b) Anzahl der optimalen Basis-Indexmengen: 2

(c) Einschließungen der optimalen Basis-Lösung(en)

$$[x]_i^{(B1)} = \begin{cases} [2.0] & \text{für} \quad i = 1, \ldots, 5 \\ [0.0] & \text{für} \quad i = 6, \ldots, 10 \end{cases}$$

und

$$[x]_i^{(B2)} = \begin{cases} [2.0] & \text{für} \quad i = 1, \ldots, 4, 10 \\ [0.0] & \text{für} \quad i = 5, \ldots, 9 \end{cases}$$

Der optimale Zielfunktionswert ist -22.4 und wird auf mindestens 10 Stellen genau eingeschlossen. Der optimale Lösungsvektor ist allerdings nicht eindeutig, da die letzte Komponenten x_5 bzw. s_5 frei zwischen 0 und 2 gewählt werden können.

Beispiel 4.2.3 Interessant ist auch die Beobachtung, daß die Dimension des Optimierungsproblems nur geringen Einfluß auf die Zahl der Iterationsschritte hat. Dies bestätigt die Komplexitätsbetrachtungen aus Abschnitt 2.1 eindrucksvoll. Für die Dimension $k = 100 \Rightarrow n = 200$ und einer Zielfunktion mit $c_i = -3$ für $i = 1, \ldots, k$ liefert das verifizierende Optimierungsverfahren als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

(a) Anzahl der Iterationsschritte: 14

(b) Zulässigkeit der Lösungsvektoren x und (y, z): primal + dual zulässig

(c) (LP) besitzt endliche optimale Lösung: ja

(d) Einschluß des optimalen Zielfunktionswertes: $[-^{5.99999999999997}_{6.00000000000004} \cdot 10^2]$

(e) (LP) besitzt eine eindeutige optimale Lösung: ja

(f) Zum Zielfunktionswert gehörige zulässige Näherungslösung

$$\begin{aligned} \widetilde{x}_1 = \ldots = \widetilde{x}_{100} &= 1.999999999999990 \quad \text{und} \\ \widetilde{s}_1 = \ldots = \widetilde{s}_{100} &= 9.923543470390585 \cdot 10^{-15} \end{aligned}$$

2. Verifizierte Resultate des anschließenden Simplex-Schrittes

(a) Einschluß des optimalen Zielfunktionswertes: $[-600]$

(b) Anzahl der optimalen Basis-Indexmengen: 1

(c) Einschließungen der optimalen Basis-Lösung(en)

$$\begin{aligned} \widetilde{x}_1 = \ldots = \widetilde{x}_{100} &= [2.0] \quad \text{und} \\ \widetilde{s}_1 = \ldots = \widetilde{s}_{100} &= [0.0] \end{aligned}$$

Der optimale Zielfunktionswert ist -600 und wird auf mindestens 10 Stellen genau eingeschlossen.

Beispiel 4.2.4 Die Überprüfung eines Optimierungsproblems auf seine Lösbarkeit hin wird durch das folgende Beispiel aus [20] demonstriert. Dieses Problem zeichnet sich durch die Eigenschaft aus, eine (durch die Nebenbedingungen) nicht beschränkte Zielfunktion zu haben. Dies bedeutet, daß keine dual zulässigen Punkte existieren. Aus der Umkehrung der Dualitätssätze 2.3.2 und 2.3.3 folgt, daß das (LP) keine endliche optimale Lösung besitzt.

$$\begin{aligned} \text{minimiere} \quad & f(x) = -50x_1 - 9x_2 - 10^{-40}x_3 \\ \text{u.d.NB.} \quad & \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 100 & 18 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 50 \\ 200 \\ 5000 \end{pmatrix} \\ \text{und} \quad & x_i \geq 0 \end{aligned}$$

Das verifizierende Optimierungsverfahren liefert als Ausgabe:

1. Verifizierte Resultate der primal-dualen Innere-Punkt-Methode

(a) Anzahl der Iterationsschritte: –

(b) Zulässigkeit der Lösungsvektoren x und (y, z):
Startlösung dual unzulässig

(c) (LP) besitzt endliche optimale Lösung: nein

(d) Einschluß des optimalen Zielfunktionswertes: –

(e) (LP) besitzt eine eindeutige optimale Lösung: nein

Anhang A

Das Programm LPverify

An dieser Stelle wird eine kurze Erläuterung zur Verwendung des Programms LPverify an Hand des Beispiels 4.1.5 gegeben. Der Aufruf des Programms zur verifizierten Lösung eines linearen Optimierungsproblems lautet

LPverify .

Gegeben ist ein (LP) in Standardform

$$\begin{array}{lrcl} & \text{minimiere} & f(x) & = & c^T x \\ \text{(LP)} & \text{unter den Nebenbedingungen} & Ax & = & b \\ & \text{und} & x & \geq & \vec{0}, \end{array}$$

wobei $x \in I\!R^n$ der Variablenvektor der zu optimierenden Zielfunktion $f(x) = c^T x$ ist, $c \in I\!R^n$. Die Nebenbedingungen werden durch eine $m \times n$ Matrix A und einen m-dimensionalen Vektor b bestimmt.

Eingabedaten für das Programm sind:

- Dimension des (LP): m, n
- Zielfunktionskoeffizienten: c
- Restriktionsmatrix: A
- rechte Seite der Restriktionen: b
- (zulässiger) primaler Startpunkt: $x^{(0)}$
- (zulässiger) dualer Startpunkt: $(y^{(0)}, z^{(0)})$

Die Bildschirmausgabe während des Programmablaufs hat die folgende Form:

```
+-------------------------------------------------------------------+
|           Linear Programming with Result Verification             |
|           ===========================================             |
|           Version 2.0  Date 20.10.94   Author M.Hocks             |
+-------------------------------------------------------------------+

Dimension (m,n): 3 6
c[1..n]  :     0.625  0.05  0.04 0  0  0

A[1..m,1..n] : 1      0     0    1  0  0
               2.5    1     0    0  1  0
               3.125  2.5   1    0  0  1

b[1..m]  :     1  5  25

x0[1..n] :  9.99999996e-01
            1.43159949e-07
            2.18749996e+01
            4.61114408e-09
            2.49999987e+00
            2.58442553e-08

y0[1..m] : -5.00000001e-01
           -8.92042081e-10
           -4.00000000e-02

z0[1..n] :  3.09140093e-09
            5.00000009e-02
            8.03857148e-11
            5.00000001e-01
            9.18481146e-10
            4.00000001e-02
-------------------------------------------------------------------
              Primal                        Dual
Iter   Obj Value   (Residual)     Obj Value   (Residual)   Gap
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1 -1.50000E+000 ( 2.9E-008)  -1.50000E+000 ( 1.4E-010)   1.6802E-008
  2 -1.50000E+000 ( 3.6E-015)  -1.50000E+000 ( 0.0E+000)   2.8004E-009
  3 -1.50000E+000 ( 3.6E-014)  -1.50000E+000 ( 6.9E-018)   4.6673E-010
  4 -1.50000E+000 ( 1.8E-015)  -1.50000E+000 ( 1.4E-017)   7.7788E-011

-------------------------------------------------------------------
Results of verified interior point method:
-------------------------------------------------------------------

 Optimal solution determined. No error occurred.
 The solutions x, y, and z are primal and dual feasible.
 The existence of an optimal solution is verified.
 The optimal value is enclosed in the interval:
    [      -1.5000000001E+000,      -1.4999999999E+000 ]
 Uniqueness of optimal solution verified.
```

```
------------------------------------------------------------------------
Results of verified simplex step:
------------------------------------------------------------------------

 The optimal value is enclosed in the interval:
    [ -1.500000000000001E+000, -1.500000000000000E+000 ]
 Number of optimal basic index sets : 1

For further results see file IPM.out.
```

Weitere Ergebnisdaten für die Beispiele 4.1.1, 4.1.4 und 4.2.3 werden angegeben. Hieran wird die Funktionsweise der Innere-Punkt-Methode deutlich sichtbar. Als Berechnungsergebnis werden für jeden Iterationsschritt (Iter) der aktuelle primale und duale Zielfunktionswert (Primal Obj Value und Dual Obj Value) mit ihren entsprechenden Abweichungen zu den Nebenbedingungen (Residual) angegeben. In der letzten Spalte steht der Wert der aktuellen Dualitätslücke (Gap), der den maximalen Abstand zum optimalen Zielfunktionswert angibt. Als Abbruchkriterium wird ein relativer Abstand von primalem und dualem Zielfunktionswert von weniger als 10^{-10} verwandt. Bei der Betrachtung des Iterationsprozesses ist gut zu erkennen, wie das Verfahren zunächst die Zentren der zulässigen Bereiche anstrebt ($\mu \gg 0$), was i.a. eine Vergrößerung der Dualitätslücke zur Folge hat. Anschließend bewegen sich primaler und dualer Zielfunktionswert aufeinander zu ($\mu \longrightarrow 0$), bis das Abbruchkriterium erfüllt ist. Während der Iteration ist es möglich, daß in einzelnen Schritten nur die primale oder duale Lösung verbessert wird, da beide unabhängig voneinander über die Schrittweitenparameter α_{LP} und α_{DP} gesteuert werden.

Die Ausgabe für Beispiel 4.1.1:

```
+------------------------------------------------------------------------+
|           Linear Programming with Result Verification                  |
|           ===========================================                  |
|           Version 2.0  Date 20.10.94   Author M.Hocks                  |
+------------------------------------------------------------------------+

------------------------------------------------------------------------
               Primal                          Dual
Iter   Obj Value   (Residual)     Obj Value   (Residual)   Gap
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1 -1.70000E+001 ( 3.9E+001)  -1.50000E+002 ( 6.0E+000)   1.3300E+002
  2 -7.07791E+001 ( 2.1E-014)  -1.50000E+002 ( 6.0E+000)   7.9221E+001
  3 -7.59192E+001 ( 4.2E-013)  -1.50000E+002 ( 6.0E+000)   7.4081E+001
  4 -7.67119E+001 ( 5.8E-013)  -1.50000E+002 ( 6.0E+000)   7.3288E+001
  5 -7.82116E+001 ( 5.8E-013)  -1.50000E+002 ( 6.0E+000)   7.1788E+001
  6 -7.89125E+001 ( 2.3E-013)  -1.50000E+002 ( 6.0E+000)   7.1087E+001
  7 -7.92673E+001 ( 3.6E-013)  -8.78904E+001 ( 0.0E+000)   8.6231E+000
  8 -7.89226E+001 ( 3.8E-013)  -8.42438E+001 ( 8.9E-016)   5.3213E+000
```

```
  9 -8.32716E+001 ( 3.4E-012)  -8.40318E+001 ( 4.4E-016)   7.6018E-001
 10 -8.34497E+001 ( 2.0E-012)  -8.35583E+001 ( 8.9E-016)   1.0860E-001
 11 -8.34933E+001 ( 5.2E-013)  -8.35088E+001 ( 8.9E-016)   1.5514E-002
 12 -8.34990E+001 ( 4.9E-013)  -8.35013E+001 ( 8.9E-016)   2.2163E-003
 13 -8.34999E+001 ( 7.6E-013)  -8.35002E+001 ( 8.9E-016)   3.1661E-004
 14 -8.35000E+001 ( 1.9E-012)  -8.35000E+001 ( 1.8E-015)   4.5230E-005
 15 -8.35000E+001 ( 6.7E-013)  -8.35000E+001 ( 8.9E-016)   6.4615E-006
 16 -8.35000E+001 ( 7.7E-013)  -8.35000E+001 ( 8.9E-016)   9.2307E-007
 17 -8.35000E+001 ( 9.2E-014)  -8.35000E+001 ( 1.8E-015)   1.3187E-007
 18 -8.35000E+001 ( 2.8E-013)  -8.35000E+001 ( 8.9E-016)   1.8839E-008
 19 -8.35000E+001 ( 2.8E-013)  -8.35000E+001 ( 0.0E+000)   2.6907E-009
 20 -8.35000E+001 ( 3.6E-013)  -8.35000E+001 ( 0.0E+000)   3.8363E-010
 21 -8.35000E+001 ( 3.7E-013)  -8.35000E+001 ( 8.9E-016)   5.3999E-011

------------------------------------------------------------------------
Results of verified interior point method:
------------------------------------------------------------------------

 Optimal solution determined. No error occurred.
 The solutions x, y, and z are primal and dual feasible.
 The existence of an optimal solution is verified.
 The optimal value is enclosed in the interval:
    [    -8.350000000004E+001,    -8.349999999997E+001 ]
 Uniqueness of optimal solution verified.


------------------------------------------------------------------------
Results of verified simplex step:
------------------------------------------------------------------------

 The optimal value is enclosed in the interval:
    [ -8.350000000000000E+001, -8.350000000000000E+001 ]
 Number of optimal basic index sets : 1

For further results see file IPM.out.
```

Die Ausgabe für Beispiel 4.1.2:

```
+------------------------------------------------------------------------+
|           Linear Programming with Result Verification                  |
|           ===========================================                  |
|           Version 2.0  Date 20.10.94   Author M.Hocks                  |
+------------------------------------------------------------------------+

------------------------------------------------------------------------
               Primal                        Dual
Iter   Obj Value   (Residual)     Obj Value   (Residual)   Gap
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1  9.30000E+000 ( 4.0E-001)   9.00000E+000 ( 1.0E+000)   3.0000E-001
  2  9.04149E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   4.1493E-002
  3  9.00231E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   2.3059E-003
  4  9.00004E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   3.6337E-005
  5  9.00000E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   4.8563E-007
  6  9.00000E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   6.4753E-009
  7  9.00000E+000 ( 8.9E-016)   9.00000E+000 ( 1.0E+000)   8.6336E-011
```

```
------------------------------------------------------------------------
Results of verified interior point method:
------------------------------------------------------------------------

 Optimal solution determined. No error occurred.
 The solutions x, y, and z are primal and dual feasible.
 The existence of an optimal solution is verified.
 The optimal value is enclosed in the interval:
    [      8.99999999999E+000,      9.00000000009E+000 ]
 Optimal solution probably not unique.


------------------------------------------------------------------------
Results of verified simplex step:
------------------------------------------------------------------------

 The optimal value is enclosed in the interval:
    [  9.000000000000000E+000,  9.000000000000000E+000 ]
 Number of optimal basic index sets : 66

For further results see file IPM.out.
```

Die Ausgabe für Beispiel 4.1.4:

```
+---------------------------------------------------------------------+
|           Linear Programming with Result Verification               |
|           ===========================================               |
|           Version 2.0  Date 20.10.94   Author M.Hocks               |
+---------------------------------------------------------------------+

------------------------------------------------------------------------
              Primal                          Dual
Iter   Obj Value   (Residual)     Obj Value   (Residual)   Gap
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1  4.00000E+000 ( 0.0E+000)  -2.77775E+005 ( 0.0E+000)   2.7778E+005
  2  4.13548E+000 ( 1.3E-015)  -9.25888E+004 ( 0.0E+000)   9.2593E+004
  3  4.23450E+000 ( 8.9E-016)  -3.08601E+004 ( 0.0E+000)   3.0864E+004
  4  4.29963E+000 ( 2.2E-016)  -1.02838E+004 ( 0.0E+000)   1.0288E+004
  5  4.34278E+000 ( 4.4E-016)  -3.42502E+003 ( 0.0E+000)   3.4294E+003
  6  4.37060E+000 ( 2.2E-016)  -1.13875E+003 ( 0.0E+000)   1.1431E+003
  7  4.38633E+000 ( 8.9E-016)  -3.76654E+002 ( 0.0E+000)   3.8104E+002
  8  4.38833E+000 ( 2.2E-016)  -1.22625E+002 ( 0.0E+000)   1.2701E+002
  9  4.36399E+000 ( 4.4E-016)  -3.79739E+001 ( 0.0E+000)   4.2338E+001
 10  4.26825E+000 ( 6.7E-016)  -9.84437E+000 ( 0.0E+000)   1.4113E+001
 11  3.94668E+000 ( 8.9E-016)  -7.57523E-001 ( 0.0E+000)   4.7042E+000
 12  2.95205E+000 ( 4.4E-016)   1.38398E+000 ( 0.0E+000)   1.5681E+000
 13  2.06299E+000 ( 0.0E+000)   1.54030E+000 ( 0.0E+000)   5.2269E-001
 14  2.05857E+000 ( 8.9E-016)   1.88434E+000 ( 0.0E+000)   1.7423E-001
 15  2.01912E+000 ( 2.2E-016)   1.96104E+000 ( 0.0E+000)   5.8077E-002
 16  2.00642E+000 ( 0.0E+000)   1.98706E+000 ( 0.0E+000)   1.9359E-002
 17  2.00215E+000 ( 0.0E+000)   1.99569E+000 ( 0.0E+000)   6.4530E-003
 18  2.00072E+000 ( 4.4E-016)   1.99857E+000 ( 0.0E+000)   2.1510E-003
 19  2.00024E+000 ( 0.0E+000)   1.99952E+000 ( 0.0E+000)   7.1700E-004
 20  2.00008E+000 ( 2.2E-016)   1.99984E+000 ( 0.0E+000)   2.3900E-004
```

```
 21  2.00003E+000 ( 0.0E+000)   1.99995E+000 ( 0.0E+000)   7.9666E-005
 22  2.00001E+000 ( 2.2E-016)   1.99998E+000 ( 0.0E+000)   2.6555E-005
 23  2.00000E+000 ( 1.1E-016)   1.99999E+000 ( 0.0E+000)   8.8518E-006
 24  2.00000E+000 ( 4.4E-016)   2.00000E+000 ( 4.4E-016)   2.9506E-006
 25  2.00000E+000 ( 2.2E-016)   2.00000E+000 ( 0.0E+000)   9.8353E-007
 26  2.00000E+000 ( 4.4E-016)   2.00000E+000 ( 2.2E-016)   3.2784E-007
 27  2.00000E+000 ( 2.2E-016)   2.00000E+000 ( 0.0E+000)   1.0928E-007
 28  2.00000E+000 ( 0.0E+000)   2.00000E+000 ( 0.0E+000)   3.6427E-008
 29  2.00000E+000 ( 4.4E-016)   2.00000E+000 ( 0.0E+000)   1.2142E-008
 30  2.00000E+000 ( 8.9E-016)   2.00000E+000 ( 0.0E+000)   4.0475E-009
 31  2.00000E+000 ( 1.1E-016)   2.00000E+000 ( 0.0E+000)   1.3492E-009
 32  2.00000E+000 ( 8.9E-016)   2.00000E+000 ( 0.0E+000)   4.4972E-010
 33  2.00000E+000 ( 1.1E-016)   2.00000E+000 ( 0.0E+000)   1.4991E-010
 34  2.00000E+000 ( 1.1E-016)   2.00000E+000 ( 0.0E+000)   4.9968E-011

------------------------------------------------------------------------
Results of verified interior point method:
------------------------------------------------------------------------

 Optimal solution determined. No error occurred.
 The solutions x, y, and z are primal and dual feasible.
 The existence of an optimal solution is verified.
 The optimal value is enclosed in the interval:
    [       1.9999999999E+000,       2.0000000001E+000 ]
 Uniqueness of optimal solution verified.


------------------------------------------------------------------------
Results of verified simplex step:
------------------------------------------------------------------------

 The optimal value is enclosed in the interval:
    [  2.000000000000000E+000,  2.000000000000000E+000 ]
 Number of optimal basic index sets : 1

For further results see file IPM.out.
```

Die Ausgabe für Beispiel 4.2.3:

```
+------------------------------------------------------------------------+
|           Linear Programming with Result Verification                  |
|           ===========================================                  |
|           Version 2.0  Date 20.10.94   Author M.Hocks                  |
+------------------------------------------------------------------------+

------------------------------------------------------------------------
               Primal                          Dual
Iter   Obj Value   (Residual)     Obj Value   (Residual)   Gap
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1 -4.80000E+002 ( 1.1E-011)  -2.00005E+007 ( 0.0E+000)   2.0000E+007
  2 -4.79103E+002 ( 0.0E+000)  -1.00479E+005 ( 0.0E+000)   1.0000E+005
  3 -4.78788E+002 ( 4.4E-016)  -9.78788E+002 ( 0.0E+000)   5.0000E+002
  4 -5.45940E+002 ( 0.0E+000)  -9.78788E+002 ( 0.0E+000)   4.3285E+002
  5 -5.77280E+002 ( 2.2E-016)  -9.78788E+002 ( 0.0E+000)   4.0151E+002
  6 -5.90416E+002 ( 4.4E-016)  -9.78788E+002 ( 0.0E+000)   3.8837E+002
```

```
  7 -5.95673E+002 ( 2.2E-016)  -9.78788E+002 ( 0.0E+000)   3.8312E+002
  8 -5.97737E+002 ( 0.0E+000)  -9.78788E+002 ( 0.0E+000)   3.8105E+002
  9 -5.98541E+002 ( 0.0E+000)  -6.00446E+002 ( 0.0E+000)   1.9053E+000
 10 -5.99994E+002 ( 4.4E-016)  -6.00004E+002 ( 0.0E+000)   9.5263E-003
 11 -6.00000E+002 ( 2.2E-016)  -6.00000E+002 ( 0.0E+000)   4.7631E-005
 12 -6.00000E+002 ( 0.0E+000)  -6.00000E+002 ( 0.0E+000)   2.3816E-007
 13 -6.00000E+002 ( 0.0E+000)  -6.00000E+002 ( 0.0E+000)   1.1908E-009
 14 -6.00000E+002 ( 0.0E+000)  -6.00000E+002 ( 0.0E+000)   6.0174E-012

------------------------------------------------------------------------
Results of verified interior point method:
------------------------------------------------------------------------

 Optimal solution determined. No error occurred.
 The solutions x, y, and z are primal and dual feasible.
 The existence of an optimal solution is verified.
 The optimal value is enclosed in the interval:
    [  -6.00000000000004E+002,  -5.99999999999997E+002 ]
 Optimal solution probably not unique.
  Interval solution not dual feasible.


------------------------------------------------------------------------
Results of verified simplex step:
------------------------------------------------------------------------

 The optimal value is enclosed in the interval:
    [ -6.000000000000000E+002, -6.000000000000000E+002 ]
 Number of optimal basic index sets : 1

For further results see file IPM.out.
```

Literaturverzeichnis

[1] Adams, E. und Kulisch, U. (Hrsg.): *Scientific Computing with Automatic Result Verification.* Academic Press, San Diego, 1993.

[2] Alefeld, G. und Herzberger, J.: *Einführung in die Intervallrechnung.* Bibliographisches Institut, Mannheim, 1974.

[3] Alefeld, G. und Herzberger, J.: *Introduction to Interval Computations.* Academic Press, New York, 1983.

[4] Anstreicher, K.: *The worst-case step in Karmarkar's algorithm.* Math. Operations Research **14**, 294 – 302, 1989.

[5] Atanassova, L. und Herzberger, J. (Hrsg.): *Computer Arithmetic and Enclosure Methods.* Proceedings of SCAN 91, North-Holland, Elsevier Science Publishers B. V., Amsterdam, 1992.

[6] Bertsekas, D.: *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, New York, 1982.

[7] Bauch, H., Jahn, K.-U., Oelschlägel, D., Süsse, H. und Wiebigke, V.: *Intervallmathematik.* Teubner, Leipzig, 1987.

[8] Carrol, C. W.: *The created response surface technique for optimizing non-linear restrained systems.* Math. Operations Research **9**, 169 – 184, 1961.

[9] Clausen, J.: *A Tutorial Note on the Complexity of the Simplex Algorithm.* In Krarup, J. and Walukiewicz (Eds.) Proceedings of DAPS-79, Intitute of Datalogy, University of Copenhagen, 1980.

[10] Collatz, L. und Wetterling, W.: *Optimierungsaufgaben.* Springer-Verlag, Heidelberg, 1966.

[11] Csendes, T.: *Test Results of Interval Methods for Global Optimization.* In [29], 417 – 424, 1991.

[12] Dantzig, G.: *Lineare Optimierung und Erweiterungen.* Springer-Verlag, Heidelberg, 1966.

[13] Euler, L.: *Methode Curven zu finden, denen eine Eigenschaft im höchsten oder geringsten Grade zukommt*. 1744. In Ostwald's Klassiker der exakten Wissenschaften, **46**, Wilhelm Engelmann, Leibzig, 1894.

[14] Fiacco, A. und McCormick, G.: *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York, 1968.

[15] Frisch, K. R.: *The logarithmic potential method for convex programming*. Memorandum May 13, University Institute of Economics, Oslo, 1955.

[16] Frommer, A: *Newton-Verfahren und nicht einfache Nullstellen*. In Chatterji, S. D., Fuchssteiner, B., Kulisch, U., Liedl, R. und Purkert, W. (Hrsg.): *Jahrbuch Überblicke Mathematik 1992*. Vieweg, Braunschweig, 27 – 44, 1992.

[17] Gal, T.: *Grundlagen des Operations Research*. Springer-Verlag, Heidelberg, 1989.

[18] Gass, S.: *Linear Programming*. McGraw–Hill Book Company, New York, 1969.

[19] Gill, P., Murray, W., Saunders, M., Tomlin, J. und Wright, M.: *On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method*. Mathematical Programming **36**, 183 – 209, 1986.

[20] Hammer, R., Hocks, M., Kulisch, U. und Ratz, D.: *Numerical Toolbox for Verified Computing I: Basic Numerical Problems*. Springer–Verlag, Heidelberg, 1993.

[21] Heuser, H.: *Lehrbuch der Analysis, Teil 1 + 2*. G. B. Teubner, Stuttgart, 1981.

[22] Horst, R.: *Nichtlineare Optimierung*. Carl Hanser Verlag, München, 1979.

[23] Jansen, B., Roos, C., Terlaky, T.: *An interior point approach to postoptimal and parametric analysis in linear programming*. Report 92-21, Faculteit der Technische Wiskunde en Informatica, TU Delft, 1992.

[24] Jansson, C.: *Zur Linearen Optimierung mit unscharfen Daten*. Dissertation, Universität Kaiserslautern, 1985.

[25] Jansson, C.: *A Self-Validating Method for Solving Linear Programming Problems with Interval Input Data*. In [43], 33 – 45, 1988.

[26] Jansson, C.: *A Global Optimization Method Using Interval Arithmetic.* In [5], 259 – 267, 1992.

[27] Karmarkar, N.: *A new polynomial-time algorithm for linear programming.* In Combinatorica **4**, 373 – 395, 1984.

[28] Karmarkar, N.: *Riemannian geometry underlying interior-point methods for linear programming.* In Mathematical Developments Arising from Linear Programming, 51 – 75 , American Mathematical Society, Providence, 1990.

[29] 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.

[30] Khachian, L.: *A polynomial algorithm in linear programming.* Dokl. Ak. Nauk SSSR **244**, 1093 – 1096, 1979.

[31] Klatte, R., Kulisch, U., Neaga, M., Ratz, D. und Ullrich, Ch.: *PASCAL–XSC – Sprachbeschreibung mit Beispielen.* Springer-Verlag, Heidelberg, 1991.

[32] Klatte, R., Kulisch, U., Lawo, C., Rauch, M. und Wiethoff, A.: *C–XSC, A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, Heidelberg, 1993.

[33] Klee, V. und Minty G.: *How good is the simplex algorithm?.* In Shisha, O.: *Inequalities III.* Academic Press, New York, 159 – 175, 1972.

[34] Kosmol, P.: *Methoden zur numerischen Behandlung nichtlinearer Gleichungen und Optimierungsaufgaben.* Teubner, Stuttgart, 1989.

[35] Kosmol, P.: *Optimierung und Approximation.* de Gruyter, Berlin, 1991.

[36] Kranich, E.: *Interior point methods for mathematical programming: a bibliography.* Report **171**, Universität Hagen, 1991. Zugriff via E-MAIL (netlibresearch.att.com).

[37] Krawczyk, R.: *Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken.* Computing **4**, 187 – 201, 1969.

[38] Krawczyk, R.: *Fehlerabschätzungen bei linearer Optimierung.* In: Nickel, K. (Hrsg.): *Interval Mathematics.* Lecture Notes in Computer Science, **29**, 215 – 227, Springer-Verlag, Wien, 1975.

[39] Kulisch, U.: *Grundlagen des Numerischen Rechnens – Mathematische Begründung der Rechnerarithmetik.* Reihe Informatik, Band **19**, Bibliographisches Institut, Mannheim, 1976.

[40] Kulisch, U. (Hrsg.): *Wissenschaftliches Rechnen mit Ergebnisverifikation – Eine Einführung.* Akademie Verlag, Ost-Berlin, Vieweg, Wiesbaden, 1989.

[41] Kulisch, U. und Miranker, W. L.: *Computer Arithmetic in Theory and Practice.* Academic Press, New York, 1981.

[42] Kulisch, U. und Miranker, W. L.: *The Arithmetic of the Digital Computer: A New Approach.* SIAM Review **28**, No. 1, 1 – 140, 1986.

[43] Kulisch, U. und Stetter, H. J. (Hrsg.): *Scientific Computation with Automatic Result Verification.* Computing Supplementum **6**, Springer-Verlag, Wien, 1988.

[44] Luenberger, D.G.: *Introduction to linear and nonlinear programming.* Addison-Wesley Publishing Company, Massachusettes, 1973.

[45] Marsten, R., Subramanian, R., Salzman, M., Lustig, I. und Shanno, D.: *Interior Point Methods for Linear Programming.* Interfaces **20**, 105 – 116, 1990.

[46] Mayer, G.: *Some Remarks on Two Interval-Arithmetic Modifications of the Newton Method.* Computing **48**, 125 – 128, 1992.

[47] Moore, R. E.: *Interval Analysis.* Prentice Hall, Engelwood Cliffs, New Jersey, 1966.

[48] Nemirovsky A.: *Self-Concordant Functions and Polynomial-Time Methods in Convex Programming.* USSR Academy of Science, Moskau, 1989.

[49] *Netlib Collection.* Elektronische Beispielsammlung. Zugriff via FTP oder XMosaic (ftp://netlib2.cs.utk.edu/lp/data).

[50] Neumann, K.: *Operations-Research-Verfahren, Band 1.* Carl Hanser Verlag, München, 1975.

[51] Neumann, K. und Morlock, M.: *Operations-Research.* Carl Hanser Verlag, München, 1993.

[52] Numerik Software GmbH: *PASCAL–XSC – A PASCAL Extension for Scientific Computation.* User's Guide, Baden-Baden, 1991.

[53] Pierre, D.: *Optimization Theory with Applications.* Wiley & Sons, New York, 1969.

[54] Ratz, D.: *Globale Optimierung mit automatischer Ergebnisverifikation.* Dissertation, Universität Karlsruhe, 1992.

[55] Richter, C: *Optimierungsverfahren und PASCAL–Programme.* Akademie-Verlag, Berlin, 1990.

[56] Rump, S. M.: *Kleine Fehlerschranken bei Matrixproblemen.* Dissertation, Universität Karlsruhe, 1980.

[57] Rump, S. M.: *Solving Algebraic Problems with High Accuracy.* In Kulisch, U. und Miranker, W. L. (Hrsg.): *A New Approach to Scientific Computation.* Academic Press, New York, 1983.

[58] Scales, L. E.: *Introduction to Non-linear Optimization*. MacMillan, London, 1985.

[59] Stoer, J.: *Einführung in die Numerische Mathematik I*. Springer-Verlag, Berlin, 1983.

[60] Walter, W.: *Analysis I + II*. Springer-Verlag, Berlin, Heidelberg 1990.

[61] Walter, W. V.: *ACRITH–XSC, A Fortran-like Language for Verified Scientific Computing.* In [1], 45 – 70, 1993.

[62] *WIOR Gopher Server*. Elektronische Datensammlung der Instituts für Wirtschaftstheorie und Operations Research. Zugriff via FTP oder XMosaic (gopher://wiorgopher.wiwi.uni-karlsruhe.de/1).

[63] Wright, M.: *Interior methods for constrained optimization.* In Acta Numerica, 341 – 407, 1991.

[64] Zurmühl, R., Falk, S.: *Matrizen und ihre Anwendungen*. Springer-Verlag, Berlin, 1984.

Stichwortverzeichnis

Lebenslauf

Matthias Hocks

geboren am	14. Juli 1964 in Berlin
Eltern	Dr. Peter Hocks und Renate Hocks, geb. Mittemeyer
Familienstand	verheiratet seit dem 29. Mai 1992 mit Martina Hocks, geb. Schmidt
Schulbildung	1970 – 1975 Renée-Sintenes-Grundschule in Berlin-Frohnau 1975 – 1982 Evangelische Schule in Berlin-Frohnau
Reifeprüfung	8. Dezember 1982
Studium	1983 – 1985 Mathematik mit dem Nebenfach Betriebswirtschaftslehre an der Technischen Universität Berlin mit abgeschlossener Diplom-Vorprüfung 1985 – 1990 Wirtschaftsmathematik mit den Nebenfächern Betriebswirtschaftslehre und Angewandte Informatik an der Universität Karlsruhe (TH)
Diplomprüfung	29. November 1990 im Studiengang Wirtschaftsmathematik
Berufstätigkeit	1984 – 1987 monatsweise als Praktikant/Werkstudent bei der Schering AG, Berlin September 1987 als Praktikant/Werkstudent bei der Schering Holdings Ltd., Cambridge, Großbritannien Mai 1987 – Dezember 1989 teilzeitbeschäftigt als Projektassistent bei der ADI Software GmbH, Karlsruhe seit Januar 1991 als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik der Universität Karlsruhe