Universität Bremen

Zentrum für Technomathematik

Parametrische Sensitivitätsanalyse

Bachelorarbeit

vorgelegt von

Renke Schäfer

1. Gutachter: Prof. Dr. Christof Büskens, Universität Bremen
2. Gutachter: Dr. Matthias Knauer, Universität Bremen

27. August 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Bei der parametrischen Sensitiviätsanalyse werden Wirkungsbeziehungen aller in einem komplexen System auftretenden Parameter untersucht. Von Interesse sind die Auswirkungen auf das System bei Veränderung eines oder mehrerer Parameter. Anwendung findet die Sensitivitätsanalyse beispielsweise beim Ausfindigmachen von dem System stark beeinflussenden Modellungenauigkeiten, bei der Maximierung der Stabilität des Systems oder beim Auffinden einer bestmöglichen Auswahl von Parametern für ein komplexes System.

In dieser Arbeit sind die untersuchten komplexen Systeme Optimierungsprobleme und die parametrische Sensitivitätsanalyse somit ein Teilgebiet der angewandten Mathematik. Die Sensitivitätsanalyse wird in Form der Post-Optimalitätsanalyse betrachtet, die nach Berechnung einer optimalen Lösung des Optimierungsproblems ihre parametrischen Abhängigkeiten bestimmt.
Mit Hilfe von Sensitivitäten lassen sich in der Optimierung beispielsweise die folgenden Fragestellungen klären: Wie abhängig ist die optimale Bahn einer Mondrakete von ihrer Masse? Ist die Abhängigkeit so groß, dass die Masse beim Bau der Rakete auf enorme Genauigkeit hin untersucht werden muss? Oder lohnt es sich vielleicht die Masse zu verringern, um eine noch bessere Bahn zu erhalten? Wie stabil ist diese Bahn bzgl. Störungen bedingt durch einen Wettereinbruch?

Das Ziel dieser Arbeit ist die Entwicklung, Vorstellung und Analyse des in den Löser von nichtlinearen Optimierungsproblemen WORHP[1] integrierten Sensitivitätsanalysemoduls WORHP ZEN[2]. Damit lassen sich die Sensitiviäten teilweise als Nebenprodukt des Optimierungsprozesses von WORHP gewinnen.

Dazu werden in Kapitel 2 zunächst die mathematischen Grundlagen der Optimierung dargelegt, um darauf aufbauend die Theorie der parametrischen Sensitiviätsanalyse herzuleiten. Eine kurze Einleitung in den Löser von nichtlinearen Optimie-

[1] WORHP ist eine Abkürzung für „We optimize really huge problems“.
[2] Zen wird wie „Sen“(für Sensitivitätsanalyse) ausgesprochen.

rungsproblemen WORHP folgt in Kapitel 3. Diese dient dem Verständnis der Implementierung sowie Funktionsweise des ebenfalls in Kapitel 3 vorgestellten Sensitivitätsanalysemoduls WORHP ZEN. Die Analyse der Ergebnisse und Berechnungen von WORHP ZEN folgt in Kapitel 4 anhand eines Optimierungsproblems sowie eines Optimalsteuerungsproblems.

Kapitel 2

Theorie der nichtlinearen Optimierung

Die Optimierung als Teilbereich der angewandten Mathematik hat die optimale Bestimmung von Variablen eines Systems zum Ziel. Bei technischen, aber auch wirtschaftlichen Systemen wird in der Regel eine hohe Effizienz verlangt, die optimalste gewünscht. Dies könnte die Minimierung der Zeit, der Strecke, der Energie oder aber der Kosten bedeuten. Die Optimierung spielt daher eine signifikante Rolle bei der Lösung von Problemstellungen aus nahezu allen wissenschaftlichen Gebieten wie beispielsweise der Physik, der Meteorologie oder auch den Wirtschaftswissenschaften und der Statistik.

Eine gute Übersicht über die verschiedenen Aufgabenstellungen der Optimierung und mögliche Methoden für deren Lösung hat Antoniou et al. in [1] zusammengestellt. In dieser Arbeit wird die beschränkte, nichtlineare Optimierung betrachtet. Diese stellt auch die Grundlage für die Werke Büskens [2] und Knauer [13] dar, an denen sich dieses Kapitel orientiert.

Im vorliegendem Kapitel wird zunächst ein kurzer Überblick über die Grundlagen der nichtlinearen Optimierung gegeben. Der Fokus dieses Kapitels liegt jedoch auf der anschließenden parametrischen Sensitivitätsanalyse in Abschnitt 2.4.

2.1 Aufgabenstellung der nichtlinearen Optimierung

In der klassischen Aufgabenstellung der nichtlinearen Optimierung soll eine Optimierungsvariable $x \in \mathbb{R}^N$ so bestimmt werden, dass eine *Zielfunktion* $f : \mathbb{R}^N \to \mathbb{R}$ unter Einhaltung der *Ungleichungsnebenbedingungen* $g = (g_1, \dots, g_{N_g})^T : \mathbb{R}^N \to \mathbb{R}^{N_g}$

sowie *Gleichungsnebenbedingung* $h = (h_1, \dots, h_{N_h})^T : \mathbb{R}^N \to \mathbb{R}^{N_h}$ minimiert wird. Das *Standardproblem der nichtlinearen Optimierung* lautet

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f(x) \\ \text{unter} \quad & g_i(x) \leq 0, \quad i = 1, \dots, N_g \\ & h_j(x) = 0, \quad j = 1, \dots, N_h \end{aligned} \tag{NLP}$$

Offensichtlich ist die Betrachtung von Minimierungsproblemen der Zielfunktion ausreichend, da die Beziehung

$$\max f(x) = -\min\left(-f(x)\right)$$

gilt.

Erfüllt ein Punkt $x \in \mathbb{R}^N$ alle Nebenbedingungen, so heißt er *zulässig*. Alle diese Punkte werden in der *zulässige Menge*

$$\begin{aligned} \mathcal{S} := \{x \in \mathbb{R}^N \quad | \quad & g_i(x) \leq 0, \quad i = 1, \dots, N_g \\ & h_j(x) = 0, \quad j = 1, \dots, N_h\} \end{aligned}$$

zusammengefasst.

Eine Lösung $x^* \in \mathbb{R}^N$ des Optimierungsproblems NLP ist ein Punkt, der

i. ein *lokales Minimum*, d.h. es existiert eine Umgebung $V \in \mathbb{R}^N$ von x^* sodass

$$f(x^*) \leq f(x), \quad \text{für alle } x \in S \cap V$$

ii. ein *streng lokales Minimum*, d.h. es existiert eine Umgebung $V \in \mathbb{R}^N$ von x^* sodass

$$f(x^*) < f(x), \quad \text{für alle } x \in S \cap V, x \neq x^*$$

iii. oder ein *(streng) globales Minimum*, d.h. die Eigenschaften in i. bzw. ii. sind für alle Umgebungen $V \in \mathbb{R}^N$ von x^* erfüllt,

darstellt.

Des Weiteren soll die Menge der aktiven Indizes definiert werden, da sie in den späteren Betrachtungen eine signifikante Rolle spielt.

Definition 2.1 (aktive Indizes). *Sei x^* eine optimale Lösung von NLP. Dann wird*

$$\mathcal{I}(x^*) := \{i \in 1, \dots, N_g \quad | \quad g_i(x, p) = 0\}$$

die Menge der aktiven Indizes *genannt.*

Über die Menge der aktiven Indizes kann demnach die Relevanz einer Ungleichungsnebenbedingung an einer optimalen Lösung x^* abgelesen werden. Ist eine Ungleichungsnebenbedingung g_i inaktiv, d.h. $i \notin \mathcal{I}(x^*)$, so ist die optimale Lösung unabhängig von dieser Nebenbedinung.

2.2 Notwendige und hinreichende Optimalitätsbedingungen

In diesem Abschnitt sollen Bedingungen einer optimalen Lösung des NLP aufgezeigt werden. Dabei bedeutet die Ordnung n einer Bedingung, dass lediglich n-te Ableitungen vorkommen.

In der Literatur können viele solcher Bedingungen gefunden werden. So beschreibt Spellucci [16] unter anderem geometrisch motivierte Bedingungen bei denen die Ableitung der Zielfunktion in einem bestimmten Kegel liegen muss. Diese, aber auch viele weitere Optimalitätskriterien sind in der Praxis schwer zugänglich bzw. sogar unbrauchbar. Die bekanntesten Optimalitätsbedingungen sind die Fritz-John-Bedingungen (vgl. z.B. Jungnickel [12]). Zusammen mit den in [14] erstmalig veröffentlichten Karush-Kuhn-Tucker Bedingungen können sie zu einem mächtigen Werkzeug der Optimierung werden.

Auf zuletzt genannten Methoden beruhen die in dieser Arbeit aufgezeigten Optimalitätsbedingungen. Sie sind dabei auf die Kriterien reduziert, die eine direkte analytische Berechnung einer optimalen Lösung ermöglichen.

Sowohl die Optimalitätsbedingung erster, als auch die zweiter Ordnung verwenden die im Folgenden definierte Lagrange-Funktion.

Definition 2.2 (Lagrange-Funktion). *Für $\lambda \in \mathbb{R}^{N_g}$ und $\mu \in \mathbb{R}^{N_h}$ wird die Funktion*

$$L(x, \lambda, \mu) := f(x) + \lambda^T g(x) + \mu^T h(x)$$

die Lagrange-Funktion *genannt. Ferner heißen die Vektoren λ und μ* Lagrange-Multiplikatoren*.*

Definition 2.3 (Regularitätsbedingung der linearen Unabhängigkeit). *Ein Punkt $x^* \in S$ heißt* normal*, falls die Gradienten $\nabla g_i(x^*), i \in \mathcal{I}(x^*)$ sowie $\nabla h_j(x^*), j = 1, \dots, N_h$ linear unabhängig sind.*

Wird für die Nebenbedingungen des NLP die Regularitätsbedingung der linearen Unabhängigkeit 2.3 vorausgesetzt, kann die notwendige Optimalitätsbedingung erster Ordnung in der folgenden Form formuliert werden.

Satz 2.4 (Notwendige Karush-Kuhn-Tucker (KKT) Bedingungen). *Sei x^* ein zulässiger Punkt des NLP. Die Zielfunktion f sei differenzierbar und die Nebenbedingungen g sowie h seien stetig differenzierbar. Ist x^* normal und eine Minimalstelle von f, dann existieren die eindeutig bestimmten* Lagrange*-Multiplikatoren $\lambda \in \mathbb{R}^{N_g}$ und $\mu \in \mathbb{R}^{N_h}$, die die folgenden KKT-Bedingungen erfüllen:*

i. Vorzeichenbedingung:

$$\lambda_i \geq 0 \quad \textit{für } i = 1, \ldots, N_g$$

ii. Optimalitätsbedingung:

$$\nabla_x L(x^*, \lambda, \mu) = \nabla f(x^*) + \lambda^T \nabla g(x^*) + \mu^T \nabla h(x^*) = 0$$

iii. Komplementaritätsbedingung:

$$\lambda^T g(x^*) = 0$$

Beweis. Diese Aussage wird von JUNGNICKEL [12] bis auf die Eindeutigkeit der LAGRANGE-Multikplikatoren bewiesen. Ein Beweis der Eindeutigkeit ist daher im Folgenden angeführt.

Seien $\overline{\lambda}$ und $\overline{\mu}$ weitere LAGRANGE-Multiplikatoren, die die KKT-Bedingungen erfüllen. Es gilt

$$0 = \nabla f(x^*) + \lambda^T \nabla g(x^*) + \mu^T \nabla h(x^*) = \nabla f(x^*) + \overline{\lambda}^T \nabla g(x^*) + \overline{\mu}^T \nabla h(x^*)$$

Dann ist

$$(\lambda^T - \overline{\lambda}^T) \nabla g(x^*) + (\mu^T - \overline{\mu}^T) \nabla h(x^*) = 0$$

Für $i \notin \mathcal{I}(x^*)$ muss bereits $\lambda_i^T = \overline{\lambda_i}^T = 0$ gelten. Da x^* nach Voraussetzung normal ist, folgt mit der linearen Unabhängigkeit der Gradienten $\lambda_i^T - \overline{\lambda_i}^T = 0$ für $i \in \mathcal{I}(x^*)$ sowie $\mu^T - \overline{\mu}^T = 0$ und damit die Eindeutigkeit. □

Es lassen sich auch Bedingungen erster Ordnung mit schwächeren Voraussetzungen formulieren. GERDTS [10] bietet hier eine gute Übersicht. Allerdings muss dann teilweise auf die Garantie der Eindeutigkeit der LAGRANGE-Multiplikatoren verzichtet werden.

Erfüllt ein Punkt die notwendigen Bedingungen erster Ordnung, so wird er *kritischer Punkt* genannt. Kritische Punkte können Lösungen des Optimierungsproblems NLP sein, sind es aber nicht notwendigerweise. Um diese Fragestellung vollends zu klären, werden die Optimalitätsbedingungen zweiter Ordnung benötigt. In ihnen wird untersucht, ob die HESSE-Matrix der LAGRANGE-Funktion auf dem *kritischen Kegel*

$$\begin{aligned} \mathcal{C} := \{ v \in \mathbb{R}^N \quad | \quad & \nabla g_i(x^*) v \leq 0, \quad i \in \mathcal{I}(x^*), \quad \lambda_i = 0 \\ & \nabla g_i(x^*) v = 0, \quad i \in \mathcal{I}(x^*), \quad \lambda_i > 0 \\ & \nabla h_j(x^*) v = 0, \quad j \in \{1, \ldots, N_h\} \} \end{aligned}$$

positiv definit ist.

Satz 2.5 (Notwendige Bedingungen zweiter Ordnung). *Sei x^* ein zulässiger Punkt des NLP. Die Zielfunktion f und die Nebenbedingungen g sowie h seien zweimal stetig differenzierbar. Ist x^* normal und eine Minimalstelle von f, dann sind nach Satz 2.4 die* LAGRANGE*-Multiplikatoren λ und μ eindeutig bestimmt und es gilt:*

$$v^T \nabla_x^2 L(x^*, \lambda, \mu) v \geq 0, \quad \text{für alle } v \in \mathcal{C}$$

Beweis. Ein Beweis dieser Tatsache kann in FLETCHER [7] gefunden werden. □

Satz 2.6 (Hinreichende Bedingung zweiter Ordnung). *Sei x^* ein zulässiger und normaler Punkt des NLP. Die Zielfunktion f und die Nebenbedingungen g sowie h seien zweimal stetig differenzierbar. Existieren* LAGRANGE*-Multiplikatoren λ sowie μ, die die KKT-Bedingungen aus Satz 2.4 erfüllen und gilt*

$$v^T \nabla_x^2 L(x^*, \lambda, \mu) v > 0, \quad \text{für alle } v \in \mathcal{C} \setminus \{0\}$$

dann ist x^* eine streng lokale Minimalstelle des NLP.

Beweis. Der Beweis dieses Satzes befindet sich ebenfalls in FLETCHER [7]. □

2.3 Aufgabenstellung der gestörten nichtlinearen Optimierung

In der praktischen Anwendung sind die Zielfunktion f als auch die Nebenbedingungen g und h des nichtlinearen Optimierungsproblems NLP oft von Parametern abhängig. Dabei sind möglichst viele Informationen über die Abhängigkeit der optimalen Lösung von diesen Parametern wünschenswert. Deshalb werden in diesem Abschnitt zunächst ein parametergestörtes Optimierungsproblem definiert und wichtige Eigenschaften aufgezeigt, um darauf aufbauend die parametrische Sensitivitätsanalyse in Abschnitt 2.4 herzuleiten.

Die Störungen bzw. die Parameter bzgl. derer das System einer Sensitivitätsanalyse unterzogen werden soll, werden im Folgenden *Störparameter* $p \in \mathbb{R}^{N_p}$ genannt. Alle Systemgleichungen können von diesem Störparameter abhängen, d.h. es werden die *Zielfunktion* $f : \mathbb{R}^N \times \mathbb{R}^{N_p} \to \mathbb{R}$ und die *Nebenbedingungen* $g = (g_1, \ldots, g_{N_g})^T : \mathbb{R}^N \times \mathbb{R}^{N_p} \to \mathbb{R}^{N_g}$ sowie $h = (h_1, \ldots, h_{N_h})^T : \mathbb{R}^N \times \mathbb{R}^{N_p} \to \mathbb{R}^{N_h}$ betrachtet.

Das *Standardproblem der gestörten nichtlinearen Optimierung* lautet dann:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f(x,p) \\ \text{unter} \quad & g_i(x,p) \leq 0, \quad i = 1, \ldots, N_g \\ & h_j(x,p) = 0, \quad j = 1, \ldots, N_h \end{aligned} \tag{NLP(p)}$$

Bei Betrachtung eines Referenzwertes $p = p_0$ wird das Standardproblem NLP(p) als *ungestört* oder *nominell* bezeichnet.

Die Definition der *Menge der zulässigen Punkte* ändert sich entsprechend zu

$$\begin{aligned} S(p) := \{x \in \mathbb{R}^N \quad | \quad & g_i(x,p) \leq 0, \quad i = 1, \ldots, N_g \\ & h_j(x,p) = 0, \quad j = 1, \ldots, N_h\} \end{aligned}$$

und analog sei nun

$$\mathcal{I}(x^*, p) := \{i \in 1, \ldots, N_g \quad | \quad g_i(x,p) = 0\}$$

die *Menge der aktiven Indizes*.

Da für ungestörte Probleme die Systemgleichungen zu $f(x) := f(x, p_0)$, $g(x) := g(x, p_0)$ und $h(x) := h(x, p_0)$ definiert werden können, bleibt die mathematische Theorie über die notwendigen und hinreichenden Optimalitätsbedingungen aus Abschnitt 2.2 gültig.

2.4 Parametrische Sensitivitätsanalyse

Um die Auswirkungen eines Parameters an der optimalen Lösung eines gestörten, nichtlinearen Optimierungsproblems NLP(p) beschreiben zu können, sollen in diesem Abschnitt zunächst Formeln für eben diese Sensitivitäten hergeleitet werden. Dies werden totale Ableitungen aller Systemgleichungen des NLP(p) sein. Abschließend werden die Sensitivitäten benutzt, um eine Möglichkeit der direkten, aber genäherten Berechnung der optimalen Lösung eines gestörten Problems aufzuzeigen. Diese Echtzeit-Optimierung findet in der Anwendung häufig Verwendung.

So zeigt beispielsweise Nikolayzik [15], wie mit Hilfe der Sensitivitätsanalyse innerhalb eines numerischen Verfahrens für die Lösung von nichtlinearen Optimierungsproblemen eine Nachkorrektur zur Verbesserung der Ergebnisse implementiert werden kann.

In einem angewandteren Beispiel beschreibt Knauer [13], wie die Sensitivitätsanalyse u.a. genutzt werden kann, um bei der Bahnberechnung eines Industrieroboters auftretende Störungen zu kompensieren.

Eine umfassende Betrachtung der Theorie der Sensitivitätsanalyse kann in Büskens [2] gefunden werden. Büskens weitet das Thema der Sensitivitäsanalyse dabei auch auf optimale Steuerungsprozesse aus.

2.4.1 Sensitivitäten der optimalen Lösung

Eine Aussage über die Existenz einer optimalen Lösung und deren Eigenschaften unter Einfluss eines Störparameters p liefert der folgende zentrale Satz der Sensitivitätsanalyse.

Satz 2.7 (Sensitivitätssatz). *Sei $x^* \in R^N$ ein zulässiger Punkt des NLP(p) zum Referenzwert $p_0 \in \mathbb{R}^{N_p}$, welcher zusammen mit den* LAGRANGE-*Multiplikatoren $\lambda^* \in \mathbb{R}^{N_g}$ und $\mu^* \in \mathbb{R}^{N_h}$ die hinreichende Bedingung zweiter Ordnung aus Satz 2.6 erfüllt. Zusätzlich sei $\lambda_i^* > 0$ für alle $i \in \mathcal{I}(x^*, p_0)$. Weiter seien f,g sowie h zweimal stetig differenzierbar bzgl. x in einer Umgebung von x^*. Ebenso seien $\nabla_x f$, $\nabla_x g$ sowie $\nabla_x h$ und die Funktionen g und h stetig differenzierbar bzgl. p in einer Umgebung von p_0.*

Dann existieren eine Umgebung $P \subset \mathbb{R}^{N_p}$ von p_0 und stetig differenzierbare Funktionen $x : P \to \mathbb{R}^N$, $\lambda : P \to \mathbb{R}^{N_g}$ sowie $\mu : P \to \mathbb{R}^{N_h}$ mit

i. $x(p_0) = x^*$, $\lambda(p_0) = \lambda^*$, $\mu(p_0) = \mu^*$

ii. Die Menge der aktiven Indizes bleibt unverändert, d.h.

$$\mathcal{I}(x(p), p) \equiv \mathcal{I}(x^*, p_0), \quad \text{für alle } p \in P$$

iii. Der Punkt $x(p)$ bleibt normal, d.h. für alle $p \in P$ sind die Gradienten

$$\nabla_x g_i(x(p), p), i \in \mathcal{I}(x(p), p) \quad \text{sowie} \quad \nabla_x h_j(x(p), p), j = 1, \dots, N_h$$

linear unabhängig.

iv. Für alle $p \in P$ erfüllt der Punkt $x(p)$ zusammen mit $\lambda(p)$ und $\mu(p)$ die hinreichenden Bedingungen zweiter Ordnung aus Satz 2.6 mit $\lambda_i(p) > 0$ für alle $i \in \mathcal{I}(x(p), p)$. Das bedeutet $x(p)$ ist ein streng lokales Minimum des NLP(p) mit den LAGRANGE-*Multiplikatoren $\lambda(p)$ und $\mu(p)$.*

Beweis. Diese Aussage wird unter anderem in SPELLUCCI [16] bewiesen. □

Werden direkt nur die aktiven Nebenbedingungen des NLP(p) betrachtet, können die Ergebnisse und Folgerungen des Sensitivitätssatzes in etwas kompakterer Form angegeben werden. Einen Beweis des Satzes bei dieser Herangehensweise liefert BÜSKENS [2].

Während des Beweises des Sensitivitätssatzes 2.7 werden Gleichungen für die Berechnung der Sensitivitäten der optimalen Lösung entwickelt. Da diese die Grundlage der Sensitivitätsanalyse bilden, soll deren Herleitung im Folgenden kurz skizziert werden.

Für die Lösung des gestörten Problems der nichtlinearen Optimierung müssen die notwendigen Bedingungen erster Ordnung aus Satz 2.4 erfüllt sein, die zusammenfassend als

$$\begin{aligned} K(x,\lambda,\mu,p) &:= \begin{pmatrix} \nabla_x L(x,\lambda,\mu,p) \\ \Delta \cdot g(x,p) \\ h(x,p) \end{pmatrix} \\ &= \begin{pmatrix} \nabla_x f(x,p) + \lambda^T \nabla_x g(x,p) + \mu^T \nabla_x h(x,p) \\ \Delta \cdot g(x,p) \\ h(x,p) \end{pmatrix} = 0 \end{aligned}$$

geschrieben werden können, wobei $\Delta := \operatorname{diag}(\lambda_1^*, \ldots, \lambda_{N_g}^*)$ ist. Der Vereinfachung der Darstellung halber sollen im Weiteren die abkürzenden Schreibweisen

$$\begin{aligned} L &:= L(x^*, \lambda^*, \mu^*, p_0) \\ f &:= f(x^*, p_0) \\ g &:= g(x^*, p_0) \\ h &:= h(x^*, p_0) \end{aligned} \tag{2.1}$$

als Auswertung der Funktionen in der optimalen Lösung (x^*, λ^*, μ^*) des nominellen Optimierungsproblems definiert werden. Wird K nun partiell nach den Argumenten (x, λ, μ) in der optimalen Lösung abgeleitet folgt

$$\frac{\partial K}{\partial(x,\lambda,\mu)}(x^*, \lambda^*, \mu^*, p_0) = \begin{pmatrix} \nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta \nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0 \end{pmatrix} \tag{2.2}$$

mit $\Gamma := \operatorname{diag}(g_1, \ldots, g_{N_g})$. Die Matrix aus Gleichung (2.2) wird im Folgenden *KKT-Matrix* genannt. In dem Beweis des Sensitivitätssatzes 2.7 wird bei Spellucci [16] die Invertierbarkeit der KKT-Matrix gezeigt. Damit sowie unter den Voraussetzungen des Sensitivitätssatzes 2.7 folgen dann unter Zuhilfenahme des Satzes über implizite Funktionen die Gleichungen für die Sensitivitäten der optimalen Lösung.

Satz 2.8 (Satz über implizite Funktionen). *Seien $x^* \in \mathbb{R}^N$ und $p_0 \in \mathbb{R}^{N_p}$ gegeben. Die Funktion $F : \mathbb{R}^N \times \mathbb{R}^{N_p} \to \mathbb{R}^N$ mit $F(x^*, p_0) = 0$ sei im Punkt (x^*, p_0) differenzierbar und die* Jacobi-*Matrix $\nabla_x F(x^*, p_0)$ sei invertierbar.*

Dann gibt es offene Umgebungen $U \subset \mathbb{R}^N$ von x^* und $P \subset \mathbb{R}^{N_p}$ von p_0 sowie eine differenzierbare Funktion $x : P \to U$ mit $F(x(p), p) = 0$ für alle $p \in P$.

Ist $(x, p) \in U \times P$ ein Punkt mit $F(x, p) = 0$, so folgt $x = x(p)$. Ferner gilt für die Jacobi-*Matrix von x*

$$\nabla_p x(p_0) = -\left(\nabla_x F(x^*, p_0)\right)^{-1} \nabla_p F(x^*, p_0).$$

Beweis. Ein Beweis findet sich in dem Standardwerk der Analysis Forster [8]. □

Folgerung 2.9 (Sensitivitäten der optimalen Lösung). *Seien die Voraussetzungen des Sensitivitätssatzes 2.7 erfüllt. Dann gilt*

$$\begin{pmatrix} \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) \\ \frac{\mathrm{d}\lambda}{\mathrm{d}p}(p_0) \\ \frac{\mathrm{d}\mu}{\mathrm{d}p}(p_0) \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta\nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \nabla_{xp} L \\ \Delta\nabla_p g \\ \nabla_p h \end{pmatrix}$$

In dieser Form lassen sich die Sensitivitäten der optimalen Lösung direkt ohne das Wissen der expliziten Abhängigkeit der Lösung x^* vom Störparameter $p \in \mathbb{R}^{N_p}$ berechnen.

2.4.2 Sensitivitäten von Zugehörigkeitsfunktionen

Neben der Sensitivitäten der optimalen Lösung sind auch die daraus resultierenden Auswirkungen auf die Zielfunktion, die Lagrange-Funktion sowie die Nebenbedingung von Interesse. Dazu werden diese Funktionen zunächst als Zugehörigkeitsfunktion des Optimierungsproblems NLP(p) betrachtet, deren Sensitivität berechnet und mit diesem Resultat die Sensitivitäten der oben genannten Funktionen hergeleitet.

Definition 2.10 (Zugehörigkeitsfunktion). *Sei $\tilde{z} : \mathbb{R}^N \times \mathbb{R}^{N_g} \times \mathbb{R}^{N_h} \times \mathbb{R}^{N_p} \to \mathbb{R}^{N_z}$ eine einmal stetig differenzierbare Funktion. Weiter seien die Voraussetzungen des Sensitivitätssatzes 2.7 erfüllt, d.h. es existiert die Umgebung $P \subset \mathbb{R}^{N_p}$ in der die Funktionen $x : P \to \mathbb{R}^N$, $\lambda : P \to \mathbb{R}^{N_g}$ sowie $\mu : P \to \mathbb{R}^{N_h}$ definiert sind. Dann heißt*

$$z(p) := \tilde{z}(x(p), \lambda(p), \mu(p), p)$$

eine zu NLP(p) zugehörige Funktion.

Unter Anwendung der Kettenregel und der Folgerung 2.9 gilt für die Sensitivitäten der Zugehörigkeitsfunktion der folgende Satz.

Satz 2.11 (Sensitivität von Zugehörigkeitsfunktionen). *Sei $z : \mathbb{R}^{N_p} \to \mathbb{R}^{N_z}$ eine Zugehörigkeitsfunktion. Weiter seien die Voraussetzungen des Sensitivitätssatzes 2.7 für das NLP(p) erfüllt. Dann gilt*

$$\frac{\mathrm{d}z}{\mathrm{d}p}(p_0) = - \begin{pmatrix} \nabla_x z & \nabla_\lambda z & \nabla_\mu z \end{pmatrix} \begin{pmatrix} \nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta\nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \nabla_{xp} L \\ \Delta\nabla_p g \\ \nabla_p h \end{pmatrix} + \nabla_p z$$

Mit diesem Resultat lassen sich nun die Sensitivitäten der Nebenbedingungen sowie der Zielfunktion angeben.

Folgerung 2.12 (Sensitivität der Nebenbedingungen). *Seien die Voraussetzungen des Sensitivitätssatzes 2.7 erfüllt. Dann folgt mit den Abkürzungen aus Gleichung (2.1)*

$$\begin{aligned}\frac{\mathrm{d}g}{\mathrm{d}p}(p_0) &= \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p g \\ \frac{\mathrm{d}h}{\mathrm{d}p}(p_0) &= \nabla_x h \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p h = 0\end{aligned}$$

Insbesondere gilt für die g_i mit $i \in \mathcal{I}(x^*, p_0)$: $\frac{\mathrm{d}g_i}{\mathrm{d}p}(p_0) = 0$

Beweis. Mit Satz 2.11 folgt

$$\begin{aligned}\frac{\mathrm{d}g}{\mathrm{d}p}(p_0) &= -\begin{pmatrix}\nabla_x g & 0 & 0\end{pmatrix} \begin{pmatrix}\nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta\nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0\end{pmatrix}^{-1} \begin{pmatrix}\nabla_{xp} L \\ \Delta\nabla_p g \\ \nabla_p h\end{pmatrix} + \nabla_p g \\ &= \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p g\end{aligned}$$

Für g_i mit $i \in \mathcal{I}(x^*, p_0)$ folgt aus der Folgerung 2.9 durch Anwenden der KKT-Matrix

$$\lambda^* \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) = -\lambda^* \nabla_p g$$

und damit

$$\frac{\mathrm{d}g}{\mathrm{d}p}(p_0) = -\nabla_p g + \nabla_p g = 0$$

Die Aussage für die Gleichungsnebenbedingungen h folgt analog. □

Folgerung 2.13 (Sensitivität der Zielfunktion). *Seien die Voraussetzungen des Sensitivitätssatzes 2.7 erfüllt. Ferner sei f bzgl. p stetig differenzierbar. Dann folgt mit den Abkürzungen aus Gleichung (2.1)*

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}p}(p_0) &= \nabla_x f \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p f \\ &= \nabla_p L\end{aligned}$$

Beweis. Die erste Gleichung folgt direkt unter Anwendung des Satzes 2.11. Weiter gilt auf Grund der notwendigen Optimalitätsbedingung erster Ordnung aus Satz 2.4:

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}p}(p_0) &= \nabla_x f \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p f \\ &= -\lambda^{*T} \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) - \mu^{*T} \nabla_x h \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p f\end{aligned}$$

Mit der Folgerung 2.12 und der Komplementaritätsbedingung aus Satz 2.4 folgt dann

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}p}(p_0) &= -\lambda^{*T}(-\nabla_p g) - \mu^{*T}(-\nabla_p h) + \nabla_p f \\ &= \nabla_p(f + \lambda^{*T} g + \mu^{*T} h) \\ &= \nabla_p L\end{aligned}$$

□

Für die Zielfunktion lassen sich sogar Sensitivitätsableitungen zweiter Ordnung angeben.

Folgerung 2.14 (Sensitivität zweiter Ordnung der Zielfunktion). *Seien die Voraussetzungen des Sensitivitätssatzes 2.7 erfüllt. Ferner seien f, g sowie h bzgl. p zweimal stetig differenzierbar. Dann folgt mit den Abkürzungen aus Gleichung (2.1)*

$$\begin{aligned}\frac{\mathrm{d}^2 f}{\mathrm{d}p^2}(p_0) &= \frac{\mathrm{d}x}{\mathrm{d}p}(p_0)^T \nabla_{xp} L + \frac{\mathrm{d}\lambda}{\mathrm{d}p}(p_0)^T \nabla_p g + \frac{\mathrm{d}\mu}{\mathrm{d}p}(p_0)^T \nabla_p h + \nabla_p^2 L \\ &= 2\frac{\mathrm{d}x}{\mathrm{d}p}(p_0)^T \nabla_{xp} L + \frac{\mathrm{d}x}{\mathrm{d}p}(p_0)^T \nabla_x^2 L \frac{\mathrm{d}x}{\mathrm{d}p}(p_0) + \nabla_p^2 L\end{aligned}$$

Beweis. Ein Beweis dieser Tatsache kann in Büskens [2] oder Knauer [13] gefunden werden. Zwar wird der Beweis dort nur unter Betrachtung der aktiven Nebenbedingungen geführt, jedoch funktioniert er mit den in dem Beweis der Folgerung 2.13 getätigten Umformungen und dem Auswerten der KKT-Matrix in Folgerung 2.9 komplett analog. □

2.4.3 Lineare Störungen in Zielfunktion und Nebenbedingungen

Die Betrachtung linearer Störparametern $p \in \mathbb{R}^{N_p}$ in der Zielfunktion sowie konstanter Störungen in den Nebenbedingungen liefern wichtige Erkenntnisse für die Effizienzsteigerung späterer Algorithmen. Bei dieser Art Störung kann bei der Berechnung der Sensitivitäten auf zusätzliche Ableitungsberechnungen verzichtet werden.

Zunächst werden lineare Störungen $r \in \mathbb{R}^N$ in der Zielfunktion betrachtet, d.h.

$$\begin{aligned}\min_{x\in\mathbb{R}^N} \quad & f(x,p) + r^T x \\ \text{unter} \quad & g_i(x,p) \le 0, \quad i = 1,\dots,N_g \\ & h_j(x,p) = 0, \quad j = 1,\dots,N_h\end{aligned} \tag{2.3}$$

Damit ergibt sich direkt $\nabla_{xr} L = E_N$ und $\nabla_r g = \nabla_r h = 0$ mit der Einheitsmatrix E wodurch sich die Sensitivitätsgleichungen nach den Folgerungen 2.9, 2.12, 2.13 und 2.14 vereinfachen lassen.

Folgerung 2.15 (Sensitivitäten bei linearen Störungen in der Zielfunktion). *Sei ein NLP(p) mit linearen Störungen in der Zielfunktion nach Gleichung 2.3 und Referenzparameter $r_0 \in \mathbb{R}^N$ gegeben. Dann lauten die Sensitivitäten nach r mit den abkürzenden Schreibweise nach 2.1*

i. für die optimale Lösung:

$$\begin{pmatrix} \frac{\mathrm{d}x}{\mathrm{d}r}(r_0) \\ \frac{\mathrm{d}\lambda}{\mathrm{d}r}(r_0) \\ \frac{\mathrm{d}\mu}{\mathrm{d}r}(r_0) \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta \nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} E_N \\ 0 \\ 0 \end{pmatrix}$$

ii. für die Nebenbedingungen:

$$\frac{\mathrm{d}g}{\mathrm{d}r}(r_0) = \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}r}(r_0)$$

iii. für die Zielfunktion:

$$\frac{\mathrm{d}f}{\mathrm{d}r}(r_0) = \nabla_r L = x^*$$

iv. zweiter Ordnung für die Zielfunktion:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}r^2}(r_0) = \frac{\mathrm{d}x}{\mathrm{d}r}(r_0)$$

Beweis. Die Folgerungen ergeben sich direkt durch Einsetzen in die Gleichungen der Sensitivitäten im allgemeinen Fall. □

Offensichtlich sind die Sensitivitäten nicht vom Störparameter r abhängig. Ferner bedarf es keiner weiteren Ableitungsberechnung, da der Gradient $\nabla_x g$ bereits für die Aufstellung der KKT-Matrix benötigt wird.

Bei konstanten Störungen in den Nebenbedingungen, d.h.

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f(x,p) \\ \text{unter} \quad & g_i(x,p) + q_i \leq 0, \quad i = 1, \ldots, N_g \\ & h_j(x,p) + q_{N_g+j} = 0, \quad j = 1, \ldots, N_h \end{aligned} \tag{2.4}$$

ergibt sich direkt $\nabla_{xq} L = 0$ und $\nabla_q (g,h)^T = E_{N_g+N_h}$. Damit lassen sich die Sensitivitätsgleichungen nach den Folgerungen 2.9, 2.12, 2.13 und 2.14 ebenfalls simplifizieren.

Folgerung 2.16 (Sensitivitäten bei konstanten Störungen in den Nebenbedingungen). *Sei ein NLP(p) mit konstanten Störungen in den Nebenbedingungen nach Gleichung 2.4 und Referenzparameter $q_0 \in \mathbb{R}^{N_g+N_h}$ gegeben. Dann lauten die Sensitivitäten nach q mit den abkürzenden Schreibweise nach 2.1*

i. für die optimale Lösung:

$$\begin{pmatrix} \frac{\mathrm{d}x}{\mathrm{d}q}(q_0) \\ \frac{\mathrm{d}\lambda}{\mathrm{d}q}(q_0) \\ \frac{\mathrm{d}\mu}{\mathrm{d}q}(q_0) \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L & \nabla_x g^T & \nabla_x h^T \\ \Delta \nabla_x g & \Gamma & 0 \\ \nabla_x h & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 \\ \Delta E_{N_g} & 0 \\ 0 & E_{N_h} \end{pmatrix}$$

ii. für die Nebenbedingungen:

$$\frac{\mathrm{d}g}{\mathrm{d}q}(q_0) = \nabla_x g \frac{\mathrm{d}x}{\mathrm{d}q}(q_0) + E_{N_g}$$

iii. für die Zielfunktion:

$$\frac{\mathrm{d}f}{\mathrm{d}q}(q_0) = \nabla_q L = \begin{pmatrix} \lambda^* \\ \mu^* \end{pmatrix}$$

iv. zweiter Ordnung für die Zielfunktion:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}q^2}(q_0) = \begin{pmatrix} \frac{\mathrm{d}\lambda}{\mathrm{d}q}(q_0) \\ \frac{\mathrm{d}\mu}{\mathrm{d}q}(q_0) \end{pmatrix}$$

Beweis. Die Aussage folgt direkt durch Einsetzen, bzw. (iv) auch durch (iii). □

Die Sensitiviäten sind folglich analog zu oben nicht vom Störparameter q abhängig.

2.4.4 Echtzeitapproximation

Mit Hilfe der Ergebnisse der Sensitivitätsanalyse lassen sich ausgehend von der Lösung des Optimierungsproblems NLP(p) Schätzwerte für die Lösung des gleichen Problems bei leicht veränderten, d.h. gestörten Parametern berechnen. Dafür ist eine neue umfassende Lösung des gestörten Optimierungsproblems NLP(p) nicht nötig.

Ist die Störung p in der aus dem Sensitivitätssatz 2.7 vorausgesetzten Umgebung P, dann bleibt die Menge der aktiven Indizes erhalten. Die Funktionen $x(p), \lambda(p), \mu(p)$

und $f(p)$ lassen sich mit einer TAYLOR-Entwicklung erster Ordnung approximieren mit

$$\begin{aligned}
x(p) &\approx x^*(p_0) + \frac{\mathrm{d}x}{\mathrm{d}p}(p_0)(p-p_0) \\
\lambda(p) &\approx \lambda^*(p_0) + \frac{\mathrm{d}\lambda}{\mathrm{d}p}(p_0)(p-p_0) \\
\mu(p) &\approx \mu^*(p_0) + \frac{\mathrm{d}\mu}{\mathrm{d}p}(p_0)(p-p_0) \\
f(p) &\approx f(x^*, p_0) + \frac{\mathrm{d}f}{\mathrm{d}p}(p_0)(p-p_0)
\end{aligned}$$

sowie

$$f(p) \approx f(x^*, p_0) + \frac{\mathrm{d}f}{\mathrm{d}p}(p_0)(p-p_0) + \frac{1}{2}(p-p_0)^T \frac{\mathrm{d}^2 f}{\mathrm{d}p^2}(p_0)(p-p_0)$$

Die Lösung des gestörten Problems kann so sehr effizient, aber genähert berechnet werden. Daher stellt die Sensitivitätsanalyse ein wichtiges Werkzeug der Echtzeit-Optimierung dar. Allerdings kann aufgrund der Näherung die Einhaltung der Nebenbedingungen nicht garantiert werden. Sei z.B. $x(p)$ die mit der TAYLOR-Entwicklung approximierte Lösung, dann gilt im Allgemeinen

$$g_i(x(p), p) \neq 0, \quad i \in \mathcal{I}(x^*, p) \quad \text{sowie} \quad h_j(x(p), p) \neq 0.$$

Kapitel 3

Der NLP-Solver WORHP mit WORHP ZEN

In diesem Kapitel soll die Implementierung des in den NLP-Solver WORHP integrierten parametrischen Sensitivitätsanalysemoduls WORHP ZEN vorgestellt werden. Dazu wird zunächst ein Überblick über die für WORHP ZEN wichtigen Komponenten von WORHP gegeben und diese werden dann in den Zusammenhang zu den theoretischen Ergebnissen aus Kapitel 2 gestellt. Die Vorstellung von WORHP dient außerdem dem Verständnis der Implementierungen der mit WORHP berechneten, numerischen Beispiele in Kapitel 4. Anschließend werden die Arbeitsweise, die Eigenschaften sowie die Benutzerschnittstelle des WORHP ZEN Moduls dargestellt.

Eine umfangreiche Zusammenfassung der Spezifikationen von WORHP kann in BÜSKENS ET AL. [6], [5] sowie [4] nachgelesen werden. Im letztgenannten Report wird darüber hinaus ein Beispiel einer Implementierung eines Optimalitätsproblems betrachtet. Der detaillierten Erklärung der Benutzung von WORHP dient der User Guide [17].

3.1 Überblick über die Grundlagen von WORHP

WORHP ist eine Bibliothek zum numerischen Lösen von nichtlinearen Optimierungsproblemen. Dabei ist WORHP darauf zugeschnitten mit mehreren Millionen Optimierungsvariablen und Nebenbedingung effizient arbeiten zu können. Solche Anzahlen können insbesondere bei der Diskretisierung von optimalen Steuerprozessen entstehen. WORHP ist in der Lage über 99% der Testsets von CUTEr[1] sowie COPS 3.0[2]

[1] CUTEr ist eine Testumgebung für Optimierungs- und lineare Algebra Solver. Mehr Informationen unter http://www.cuter.rl.ac.uk/ (Stand 29.06.2012)

[2] COPS 3.0 ist eine Testumgebung mit hochdimensionalen, beschränkten, nichtlinearen Optimierungsproblemen. Mehr Informationen unter http://www.mcs.anl.gov/~more/cops/ (Stand 29.06.2012)

zu lösen. Entwickelt wird WORHP in erster Linie von Prof. Dr. Christof Büskens und Dennis Wassel von der Universität Bremen sowie Prof. Dr. Matthias Gerdts von der Universität der Bundeswehr München. Die verwendete Programmiersprache ist im Kern FORTRAN 95, in der daher auch WORHP ZEN geschrieben ist. Der Rechteinhaber von WORHP ist das Steinbeis Forschungszentrum für Optimierung, Steuerung und Regelung.

3.1.1 Von WORHP verwendete Standardform

Das Ziel des Programms WORHP ist es nichtlineare Optimierungsaufgaben der Form

$$\begin{aligned} \min_{x\in\mathbb{R}^N} \quad & f(x) \\ \text{unter} \quad & \begin{pmatrix} l \\ L \end{pmatrix} \leq \begin{pmatrix} x \\ g(x) \end{pmatrix} \leq \begin{pmatrix} u \\ U \end{pmatrix} \end{aligned} \tag{3.1}$$

mit $l, u \in \mathbb{R}^N$ und $L, U \in \mathbb{R}^{N_g}$ numerisch zu lösen. Dabei sind analog zu NLP $f : \mathbb{R}^N \to \mathbb{R}$ die Zielfunktion und $g : \mathbb{R}^N \to \mathbb{R}^{N_g}$ die Nebenbedingungen.

Im Unterschied zu der Standardform NLP aus Kapitel 2.1 werden hier zusätzlich sogenannte *Boxschranken* l, u der Optimierungsvariable betrachtet. Offensichtlich kann diese Form jedoch direkt in die Standardform NLP überführt werden. Darüber hinaus wird bei WORHP formal nicht zwischen Gleichungs- und Ungleichungsnebenbedingungen unterschieden. Ob eine Nebenbedingung einer Ungleichung oder einer Gleichung entspricht, wird über die Schranken L und U festgesetzt. Gilt $L = U$, so handelt es sich um eine Gleichungsnebenbedingung.

3.1.2 Sequentielle quadratische Programmierung

Zur Lösung der Optimierungsprobleme verwendet WORHP einen SQP[3] Algorithmus, der eine spezielle Form eines Abstiegsrichtungsverfahren darstellt. Dazu wird iterativ in einem Punkt $x^{[k]}$ eine Suchrichtung $d^{[k]}$ für die optimale Lösung berechnet und sich so schrittweise dem Optimum genähert. Das Optimierungsproblem NLP wird dabei durch ein quadratisches Optimierungsproblem approximiert. Dieses lautet im k-ten Schritt

$$\begin{aligned} \min_{d^{[k]}\in\mathbb{R}^N} \quad & \frac{1}{2} d^{[k]^T} \nabla_x^2 L(x^{[k]}, \lambda^{[k]}, \mu^{[k]}) d^{[k]} + \nabla_x f(x^{[k]}) d^{[k]} \\ \text{unter} \quad & g(x^{[k]}) + \nabla_x g(x^{[k]}) d^{[k]} \leq 0 \\ & h(x^{[k]}) + \nabla_x h(x^{[k]}) d^{[k]} = 0 \end{aligned} \tag{3.2}$$

[3]SQP steht für sequentielle quadratische Programmierung

Für das quadratische Optimierungsproblem 3.2 folgt mit Hilfe der notwendigen Bedingungen erster Ordnung aus Satz 2.4

$$
\begin{aligned}
&\nabla_x^2 L(x^{[k]}, \lambda^{[k]}, \mu^{[k]}) d^{[k]} + \nabla_x f(x^{[k]}) + \nabla_x g(x^{[k]}) \lambda^{[k]} + \nabla_x h(x^{[k]}) \mu^{[k]} = 0 \\
&g_i(x^{[k]}) + \nabla_x g_i(x^{[k]}) d^{[k]} = 0 \quad \text{für } i \in \mathcal{I}(x^{[k]}) \\
&h(x^{[k]}) + \nabla_x h(x^{[k]}) d^{[k]} = 0
\end{aligned}
$$

bzw. umformuliert in Matrixschreibweise

$$
\begin{pmatrix} \nabla_x^2 L^{[k]} & \nabla_x g^{[k]^T} & \nabla_x h^{[k]^T} \\ \Delta \nabla_x g^{[k]} & \Lambda & 0 \\ \nabla_x h^{[k]} & 0 & 0 \end{pmatrix} \begin{pmatrix} d^{[k]} \\ \lambda^{[k]} \\ \mu^{[k]} \end{pmatrix} = - \begin{pmatrix} \nabla_x f^{[k]} \\ \Delta g^{[k]} \\ h^{[k]} \end{pmatrix} \tag{3.3}
$$

mit $\Delta := \operatorname{diag}(\lambda_1^{[k]}, \ldots, \lambda_{N_g}^{[k]})$ sowie $\Lambda := \operatorname{diag}(\chi_{\mathcal{I}(x^{[k]})^C}(1), \ldots, \chi_{\mathcal{I}(x^{[k]})^C}(N_g))$. Dabei ist χ die charakteristische Funktion. Außerdem wurden die abkürzenden Schreibweisen $L^{[k]} := L(x^{[k]}, \lambda^{[k]}, \mu^{[k]})$, $f^{[k]} := f(x^{[k]})$, $g^{[k]} := g(x^{[k]})$ sowie $h^{[k]} := h(x^{[k]})$ benutzt.

Nur im Falle von positiver Definitheit der HESSE-Matrix der LAGRANGE-Funktion $\nabla_x^2 L(x^{[k]}, \lambda^{[k]}, \mu^{[k]})$ ist $d^{[k]}$ eine Abstiegsrichtung und kann als Suchrichtung des nichtlinearen Optimierungsproblems dienen.

Details zum SQP-Algorithmus und der dazugehörigen mathematischen Theorie können in SPELLUCCI [16] oder GERDTS [10] gefunden werden.

Die Matrix des Systems 3.3 sowie die aus der parametrischen Sensitivitätsanalyse bekannte KKT-Matrix 2.2 führen bei identischer rechten Seite auf das gleiche lineare Gleichungssystem. Dieser Zusammenhang kann bei der Berechnung der parametrischen Sensitivitätsanalyse gewinnbringend verwendet werden (vgl. Abschnitt 3.2.1).

3.1.3 Speicherformat für dünn besetzte Matrizen

Um hochdimensionale Optimierungsprobleme effizient berechnen zu können, müssen der Speicherplatzbedarf minimiert sowie vorhandene Strukturen erkannt und genutzt werden. Dem wird WORHP mit dem Datentyp `WorhpMatrix` gerecht, in dem jede von WORHP benutzte Matrix gespeichert ist.

Hochdimensionale Optimierungsprobleme, insbesondere diejenigen, die durch die Diskretisierung von optimalen Steuerungsprozessen entstehen, sind sehr oft dünn besetzt, d.h. viele Einträge sind konstant Null. Deshalb werden in `WorhpMatrix` nur die von Null verschiedenen Werte abgespeichert. Bei internen Prozessen arbeitet WORHP mit dem *Coordinate Storage (CS) Format*. Dabei werden die von Null verschiedenen Einträge einer Matrix A durch den Vektor (A_{ij}, i, j) repräsentiert und all diese Triplets in den Vektoren `val`, `row` und `col` nach der Vorschrift

$(\texttt{val}_k, \texttt{row}_k, \texttt{col}_k) = (A_{ij}, i, j)_k$ mit $k = 1, \ldots, N_{nz}$ gespeichert. Der Wert N_{nz} bezeichnet die Anzahl von Null verschiedenen Werten. Folglich müssen in diesem Format N_{nz} double und $2N_{nz}$ integer Werte gespeichert werden.

Darüber hinaus stellt WORHP einige Bedingungen an die Sortierung der Einträge sowie die Ausnutzung der Struktur der HESSE-Matrix der LAGRANGE-Funktion. Diese sind für das weitere Verständnis dieser Arbeit nicht erforderlich, können aber bei Bedarf im User Guide [17] nachgelesen werden.

3.1.4 Benutzerschnittstellen von WORHP

WORHP bietet drei verschiedene Benutzerschnittstellen: *Full Feature Interface*, *Basic Feature Interface* und *Legacy Interface*.

Die beiden erst genannten machen Gebrauch vom *Unified Solver Interface*, bei dem sich sämtliche Funktionalitäten von WORHP mit den vier Datenstrukturen `OptVar`, `Workspace`, `Params` und `Control` bedienen lassen. Eine Liste mit für WORHP ZEN signifikanten Variablen der vier Datenstrukturen ist im Anhang A aufgeführt.

Das Full Feature Interface ist das funktionsreichste und bietet beispielsweise die Möglichkeit der *Reverse Communication*. Dafür sind die einzelnen Stufen des Lösungsprozesses des SQP-Verfahrens in *Stages* unterteilt. Die Hauptiterationsschleife des Prozesses wird in das von den UserInnen geschriebene Programm ausgelagert. Damit entsteht für die NutzerInnen die Möglichkeit zwischen den Stages Daten zu analysieren oder eigene bereitzustellen. Die Reverse Communication wird von WORHP u.a. dafür verwendet von den UserInnen analytisch berechnete Ableitung dem System zu übergeben oder die UserInnen zur Bildschirmausgabe des Ergebnisses der aktuellen Stage aufzufordern. In den *UserActions* werden die von WORHP aufgeforderten Aktionen geflaggt und können so von den NutzerInnen abgefragt werden.

Eine Auflistung der UserActions ergänzt die Arbeit in Anhang A. Als Beispiele zur Verwendung der Reverse Communication dienen die Quelltexte im Anhang C.

3.2 Das WORHP ZEN Modul

WORHP ZEN kommt dem Wunsch nach WORHP um eine parametrische Sensitivitätsanalyse zu ergänzen. Dabei ist das Ziel von WORHP ZEN möglichst viele Informationen, die bereits durch den SQP Prozess von WORHP anfallen, hinsichtlich der Effizienzsteigerung zu nutzen. Der Aufbau des Moduls ermöglicht sowohl die Nutzung für die klassische Post-Optimalitätsanalyse sowie die Eingliederungen in beliebige andere Prozesse, die Sensitivitätsableitungen benötigen, wie beispielsweise dem Korrekturschritt aus NIKOLAYZIK [15].

Die Post-Optimalitätsanalyse mit WORHP ZEN ist derzeitig für das Full Feature Interface von WORHP optimiert. Eine Nutzung mit dem Basic Feature Interface ist prinzipiell möglich. Die gesamten WORHP ZEN Routinen machen Gebrauch des Unified Solver Interface.

3.2.1 Berechnungen von WORHP ZEN

WORHP ZEN ist in der Lage die Sensitivitätsableitungen der optimalen Lösung, die der Nebenbedingungen, der Zielfunktion sowie die Sensitivitätsableitung zweiter Ordnung der Zielfunktion zu bestimmen. Grundlage für diese Berechnungen sind die Folgerungen 2.9, 2.12, 2.13 und 2.14.

WORHP ZEN nutzt gewinnbringend die KKT-Matrix des QP Problems 3.3, für deren Lösung WORHP mittels LEQSOL[4] eine LU Zerlegung berechnen und speichern lässt. Mittels dieser Zerlegung lässt sich das lineare Gleichungssystem für die Sensitivitätsableitungen der optimalen Lösung in 2.9 direkt durch Vorwärts- und Rückwärtseinsetzen berechnen. Dafür wird nach BÜSKENS [3] prinzipiell nur ein Aufwand von $\mathcal{O}((N + N_g + N_h)^2)$ benötigt. Daraus folgt jedoch, dass für einen Aufruf von WORHP ZEN mindestens ein SQP Schritt getätigt worden sein muss.

Die einzelnen Sensitivitätsableitungen werden mit den folgenden FORTRAN Routinen berechnet.

Listing 3.1: Routinen für die Berechnung der Sensitivitätsableitungen

```
CALL ZenDxDmu(opt, work, par, cnt)    ! optimale Loesung
CALL ZenDg(opt, work, par, cnt)       ! Nebenbedingungen
CALL ZenDf(opt, work, par, cnt)       ! Zielfunktion
CALL ZenDf2(opt, work, par, cnt)      ! Zielfunktion 2. Ordnung
```

Bei der Reihenfolge der Aufrufe dieser Routinen muss auf die untereinander herrschenden Abhängigkeiten geachtet werden. So ist beispielsweise die Sensitivitätsableitung der Nebenbedingungen von der der optimalen Lösung abhängig (vgl. Folgerung 2.12). Darüber hinaus müssen notwendige Ableitungen wie z.B. $\nabla_p L$ vorher bereit stehen.

Die Ableitungen $\nabla_p g$, $\nabla_p L$ und $\nabla_{xp} L$ können entweder per Reverse Communication von den NutzerInnen oder per Finiter Differenzen von WORHP ZEN berechnet werden. Dazu wird ein zentraler Differenzenquotient verwendet mit einer standardmäßigen Genauigkeit von 10^{-5}. Langfristig ist die Integration dieser Routinen in das WORHP Modul FIDIF geplant. Eine finite Differenzenmethode für $\nabla_{pp} L$ steht derzeitig nicht zur Verfügung. Eine Bereitstellung von den NutzerInnen ist unumgänglich.

[4]LEQSOL steht für Linear Equation Solver und ist im QP Solver QPSOL integriert. Für Mehr Informationen siehe GERDTS [11].

Die Echtzeit-Optimierung nach Abschnitt 2.4.4 stellt WORHP ZEN in der Routine

Listing 3.2: Routine für die Echtzeit-Optimierung

```
CALL ZenUpdateSolution(opt, work, par, cnt, p, r, q, x, mu, f,
   FALSE)
```

bereit, der neben dem Unified Interface auch die Störungen und die Pointer auf die approximierten Lösungen übergeben werden. Zusätzlich wird die Verwendung der zweiten Sensitivitätsableitung der Zielfunktion spezifiziert (vgl. Anhang C.2).

Die Verwendung der Skalierung sowie dem BFGS Update Verfahren[5] von WORHP (vgl. den User Guide [17]) können derzeitig zu falschen Sensitivitätsableitungen von WORHP ZEN führen und sollten daher ausgeschaltet werden. Die Implementierung der Rückskalierung von WORHP ZEN ist jedoch in Planung.

3.2.2 Benutzerschnittstelle von WORHP ZEN

Die EndnutzerInnen werden WORHP ZEN in der Form der Post-Optimalitätsanalyse nutzen wollen. Diese ist als Stage `Zen_Post_Optimum` dem SQP Prozess angehängt. Sie wird aufgerufen, falls eine `optimalSolution` gefunden wurde und mindestens eine Sensitivitätsableitung zu ermitteln ist. Diese Stage kümmert sich um die gesamte Reverse Communication, das Aufrufen der Finiten Differenzen Methoden sowie der Routinen für die Sensitivitätsableitungsberechnung.

Für den nichtlinearen Störparameter p wurde die Datenstruktur `OptVar` um den Vektor `P` mit der Dimension `K` ergänzt (vgl. auch Anhang B). Dimension und Referenzwert sind zu Beginn von den NutzerInnen zu setzen. Sensitivitätsableitungen bzgl. Parametern, die linearen Störungen in der Zielfunktion oder konstanten Störungen in den Nebenbedingungen entsprechen, werden nicht explizit deklariert. WORHP ZEN berechnet all diese Ableitungen automatisch nach den Ergebnissen aus Abschnitt 2.4.3.

Sensitivitätsableitung	Parameter	Matrix
$\frac{\mathrm{d}x}{\mathrm{d}p}(p_0)$	ZenDxDmu	ZenDX
$\frac{\mathrm{d}\mu}{\mathrm{d}p}(p_0)$	ZenDxDmu	ZenDMu
$\frac{\mathrm{d}f}{\mathrm{d}p}(p_0)$	ZenDf	ZenDF
$\frac{\mathrm{d}^2 f}{\mathrm{d}p^2}(p_0)$	ZenDf2	ZenDF2
$\frac{\mathrm{d}g}{\mathrm{d}p}(p_0)$	ZenDg	ZenDG

Tabelle 3.1: WORHP ZEN Zusammenhänge zwischen Matrizen der Sensitivitätsableitungen und ihren Parametern zum Ein- und Ausschalten der Berechnung

[5] BFGS steht für Broyden-Fletcher-Goldfarb-Shanno

Mittels logischer Parameter wie `ZenDf` als Teil der Datenstruktur `Params` kann festgelegt werden, welche Sensitivitätsableitungen berechnet werden sollen. WORHP ZEN speichert diese in `WorhpMatrix` Matrizen im `Workspace`. Eine komplette Übersicht befindet sich in Tabelle 3.1. Der Aufbau der Matrizen sieht vor, dass die ersten N_p Spalten den Sensitivitäten bzgl. der nichtlinearen Parameter `P` aus dem `OptVar` entsprechen. Es folgen `opt%N` Spalten mit Sensitivitäten bzgl. der linearen Störungen in der Zielnfunktion und `opt%M` Spalten für die konstanten Störungen in den Nebenbedingungen.

Wird vom Nutzer gewünscht, Ableitungen bzgl. des Störparameters für die Berechnungen per Reverse Communication bereitzustellen, so kann er dies über logische Parameter festlegen, die zugehörigen UserActions abfragen und die entsprechenden Matrizen beschreiben. Tabelle 3.2 dient als Übersicht über diese Zusammenhänge.

Ableitung	Parameter	UserAction	Matrix
$\nabla_p g$	UserZenDGp	evalZenDGp	ZenDGp
$\nabla_p L$	UserZenDLp	evalZenDLp	ZenDLp
$\nabla_{xp} L$	UserZenDLxp	evalZenDLxp	ZenDLxp

Tabelle 3.2: WORHP ZEN UserAction Zusammenhänge zwischen den zu setzenden Parametern, der abzufragenden UserAction und der zu beschreibenden Matrix

Kapitel 4

Numerische Ergebnisse

In diesem Kapitel soll die Funktionsfähigkeit und die Bedienbarkeit des WORHP ZEN Moduls demonstriert sowie die Gültigkeit der berechneten Sensitivitätsableitungen exemplarisch gezeigt werden. Ergebnisse werden dabei als gültig berachtet, wenn sie mit einer vorher definierten Genauigkeit (1% relativer Fehler) mit den analytisch berechneten Werten übereinstimmen.

In einem ersten Beispiel wird die u.a. in FLETCHER [7] und GERDTS [10] betrachtete Funktion von ROSENBROCK als bekannte Testfunktion für NLP-Solver näher untersucht. Ein zweites Beispiel betrachtet ein diskretisiertes optimales Steuerungsproblem und ist der Website des ZETEM [9] entnommen. Es dient vor allem dem Aufzeigen des Nutzens der parametrischen Sensitivitätsanalyse in der Praxis. Hier werden die Möglichkeiten der Echtzeit-Optimierung aufgezeigt und deren Ergebnisse validiert.

Die numerischen Ergebnisse wurden unter UBUNTU PRECISE PANGOLIN (12.04 LTS) auf einem 32-Bit System mit 4 GB Arbeitsspeicher und dem INTEL CORE 2 DUO T8100 (2.10 GHZ) Prozessor gewonnen. Es wurde WORHP in der Entwicklerversion 1.0-r1880 mit zusätzlichem WORHP ZEN Modul (vgl. Kapitel 3) verwendet. Die Beispiele sind jeweils in FORTRAN 95 programmiert und mit GFORTRAN 4.6.3 kompiliert worden. Die von WORHP verwendeten Löser für lineare Gleichungen sind MA57 3.6 und SUPERLU 4.0.

4.1 Minimierung der Funktion von Rosenbrock

Die Funktion von ROSENBROCK ist eine bekannte Testfunktion für Optimierungsalgorithmen. Sie wurde 1960 von HOWARD H. ROSENBROCK formuliert und ist definiert als

$$
\begin{aligned}
f : \mathbb{R}^2 &\longrightarrow \mathbb{R} \\
(x_1, x_2) &\longmapsto 100(x_2 - x_1^2)^2 + (1 - x_1)^2
\end{aligned}
$$

Ihr eindeutig bestimmtes, globales Minimum liegt bei $x^* = (1,1)^T$ mit $f(x^*) = 0$.

4.1.1 Problembeschreibung

In dieser Testumgebung sollen die beiden Ganzzahlwerte $(100, 1)$ in f als Parameter $p \in \mathbb{R}^2$ interpretiert werden. Darüber hinaus soll der zulässige Bereich mittels zweier Ungleichungsnebenbedingungen eingeschränkt werden um ebenfalls Sensitivitäten linearer sowie nichtlinearer Störungen in den Nebenbedingungen berechnen zu können.

Das hier betrachtete gestörte nichtlineare Optimierungsproblem lautet

$$
\begin{aligned}
\min_{x \in \mathbb{R}} \quad & f(x,p) = p_1(x_2 - x_1^2)^2 + (p_2 - x_1)^2 \\
\text{unter} \quad & g_1(x,p) = x_1 + x_2 + q_1 \leq 0 \\
& g_2(x,p) = -p_1 x_1^2 + x_2^2 + q_2 \leq 0
\end{aligned}
\tag{4.1}
$$

mit den Referenzparametern $p_0 = (100, 1)^T$ und $q_0 = (-\frac{299}{400}, -50)^T$. In Abbildung 4.1 ist der Funktionsverlauf der Funktion von Rosenbrock sowie der zulässige Bereich unter Verwendung der Referenzparameter aufgezeigt.

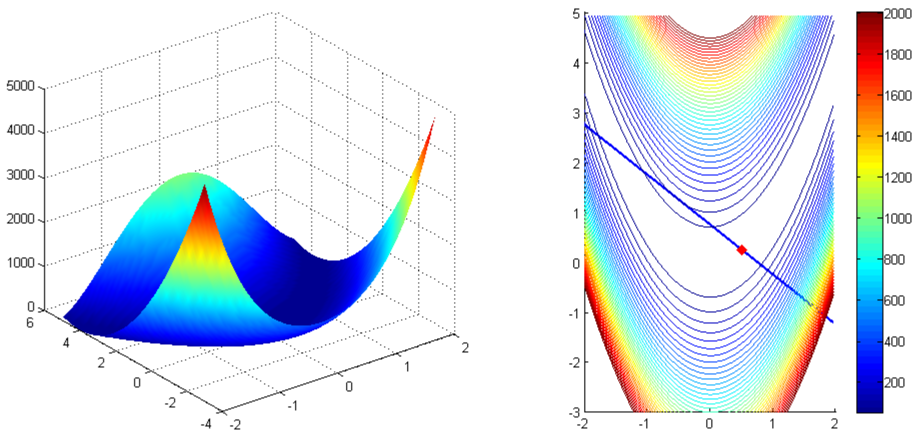

Abbildung 4.1: 3D-Plot und Höhenlinien der Funktion von Rosenbrock mit der ersten Nebenbedingung (blau, dick) und optimaler Lösung (rot, dick)

Nach Definition 2.3 sind die zulässigen Punkte bis auf die Gerade $100x_1 + x_2 = 0$ normal. Es kann jedoch leicht gezeigt werden, dass das Optimum nicht auf dieser

Geraden liegen kann. Somit folgt mit der notwendigen Optimalitätsbedingung erster Ordnung aus Satz 2.4

$$\begin{aligned} 0 &= \nabla_x L(x^*, \lambda^*, p) \\ &= \begin{pmatrix} -4p_1 x_1^* x_2^* + 4p_1 (x_1^*)^3 - 2p_2 + 2x_1^* \\ 2p_1 x_2^* - 2p_1 (x_1^*)^2 \end{pmatrix} + \lambda_1^* \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \lambda_2^* \begin{pmatrix} -2p_1 x_1^* \\ 2x_2^* \end{pmatrix} \end{aligned}$$

Das auftretende Gleichungssystem kann unter Zuhilfenahme der Vorzeichen- und Komplementaritätsbedingung aus Satz 2.4 zu

$$x^* = \left(\frac{1}{2}, \frac{99}{400}\right)^T \qquad \lambda^* = \left(\frac{1}{2}, 0\right)^T$$

gelöst werden.

Mit der KKT-Matrix nach Gleichung 2.2

$$\begin{pmatrix} \nabla_x^2 L & \nabla_x g^T \\ \Delta \nabla_x g & \Gamma \end{pmatrix} = \begin{pmatrix} 203 & -200 & 1 & -100 \\ -200 & 200 & 1 & \frac{99}{200} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1919999990199}{25600000000} \end{pmatrix}$$

und den Ableitungen

$$\begin{aligned} \nabla_{xp} L &= \begin{pmatrix} -4x_1^* x_2^* + 4x_1^{*3} + 2\lambda_2^* x_1^* & -2 \\ 2x_2^* - 2x_1^{*2} & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{200} & -2 \\ -\frac{1}{200} & 0 \end{pmatrix} \\ \nabla_p G &= \begin{pmatrix} 0 & 0 \\ -x_1^{*2} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ -\frac{1}{4} & 0 \end{pmatrix} \end{aligned}$$

können die parametrischen Sensitivitätsableitungen nach den Folgerungen aus Kapitel 2.4 bestimmt werden.

4.1.2 Auswertung

Im Folgenden findet ein Vergleich der analytisch und der von WORHP ZEN berechneten Sensitivitätsableitungen statt. Der Quellcode zu dieser Berechnung ist der Arbeit in Abschnitt C.1 angehängt. Dabei wurden die Standardparameter von WORHP verwendet. Lediglich die Skalierung des QP Problems ist deaktiviert.

In Tabelle 4.1 sind die Sensitivitäten der optimalen Lösung $(x^*, \mu^*)^T$, in Tabelle 4.2 die der Nebenbedingungen, in Tabelle 4.3 die der Zielfunktion und in Tabelle 4.4 die Sensitivitäten zweiter Ordnung der Zielfunktion aufgelistet.

Zusammengefasst liegen die relativen Fehler aller von WORHP ZEN berechneten Werte deutlich unter 1%. Die Fehler sind besonders klein bei der Sensitivitätsableitung der Zielfunktion. Dies resultiert aus der direkten Berechnung der Werte per

Sensitivität von	bzgl.	analytisch exakt	gerundet	WORHP ZEN	Fehler absolut	relativ
x_1	p_1	$-1/80300$	$-1.245330012 \cdot 10^{-5}$	$-1.245298145 \cdot 10^{-5}$	$3 \cdot 10^{-10}$	0.0026%
	p_2	$2/803$	$2.490660025 \cdot 10^{-3}$	$2.490596079 \cdot 10^{-3}$	$6 \cdot 10^{-8}$	0.0026%
	r_1	$-1/803$	$-1.245330013 \cdot 10^{-3}$	$-1.245298039 \cdot 10^{-3}$	$3 \cdot 10^{-8}$	0.0026%
	r_2	$1/803$	$1.245330013 \cdot 10^{-3}$	$1.245298035 \cdot 10^{-3}$	$3 \cdot 10^{-8}$	0.0026%
	q_1	$-400/803$	$-4.981320050 \cdot 10^{-1}$	$-4.981234967 \cdot 10^{-1}$	$9 \cdot 10^{-6}$	0.0017%
	q_2	0	0	$4.452704581 \cdot 10^{-20}$	$4 \cdot 10^{-20}$	-
x_2	p_1	$1/80300$	$1.245330012 \cdot 10^{-5}$	$1.245298145 \cdot 10^{-5}$	$3 \cdot 10^{-10}$	0.0026%
	p_2	$-2/803$	$-2.490660025 \cdot 10^{-3}$	$-2.490596071 \cdot 10^{-3}$	$6 \cdot 10^{-8}$	0.0026%
	r_1	$1/803$	$1.245330013 \cdot 10^{-3}$	$1.245298035 \cdot 10^{-3}$	$3 \cdot 10^{-8}$	0.0026%
	r_2	$-1/803$	$-1.245330013 \cdot 10^{-3}$	$-1.245298039 \cdot 10^{-3}$	$3 \cdot 10^{-8}$	0.0026%
	q_1	$-403/803$	$-5.018679950 \cdot 10^{-1}$	$-5.018765033 \cdot 10^{-1}$	$9 \cdot 10^{-6}$	0.0017%
	q_2	0	0	$-4.452704568 \cdot 10^{-20}$	$4 \cdot 10^{-20}$	-
μ_1	p_1	$3/160600$	$1.867995019 \cdot 10^{-5}$	$1.876502925 \cdot 10^{-5}$	$9 \cdot 10^{-8}$	0.4555%
	p_2	$800/803$	$9.962640100 \cdot 10^{-1}$	$9.9624699336 \cdot 10^{-1}$	$2 \cdot 10^{-5}$	0.0017%
	r_1	$-400/803$	$-4.981320050 \cdot 10^{-1}$	$-4.981234967 \cdot 10^{-1}$	$9 \cdot 10^{-6}$	0.0017%
	r_2	$-403/803$	$-5.018679950 \cdot 10^{-1}$	$-5.018765033 \cdot 10^{-1}$	$9 \cdot 10^{-6}$	0.0017%
	q_1	$600/803$	$7.471980075 \cdot 10^{-1}$	$7.488912955 \cdot 10^{-1}$	$2 \cdot 10^{-3}$	0.2266%
	q_2	0	0	$1.763485941 \cdot 10^{-17}$	$2 \cdot 10^{-17}$	-
μ_2	p_1	0	0	0	0	-
	p_2	0	0	0	0	-
	r_1	0	0	0	0	-
	r_2	0	0	0	0	-
	q_1	0	0	0	0	-
	q_2	0	0	0	0	-

Tabelle 4.1: Minimierung der Funktion von Rosenbrock: analytisch und von WORHP ZEN berechnete Sensitiviäten der optimalen Lösung

Sensitivität von	bzgl.	analytisch exakt	gerundet	WORHP ZEN	Fehler absolut	relativ
g_1	p_1	0	0	0	0	-
	p_2	0	0	0	0	-
	r_1	0	0	0	0	-
	r_2	0	0	0	0	-
	q_1	0	0	0	0	-
	q_2	0	0	0	0	-
g_2	p_1	$-4035099/16060000$	$-2.512514944 \cdot 10^{-1}$	$-2.487485391 \cdot 10^{-1}$	$3 \cdot 10^{-3}$	0.9962%
	p_2	$-20099/80300$	$-2.502988792 \cdot 10^{-1}$	$-2.502924536 \cdot 10^{-1}$	$5 \cdot 10^{-6}$	0.0026%
	r_1	$20099/160600$	$1.251494396 \cdot 10^{-1}$	$1.251462268 \cdot 10^{-1}$	$3 \cdot 10^{-6}$	0.0026%
	r_2	$-20099/160600$	$-1.251494396 \cdot 10^{-1}$	$-1.251462264 \cdot 10^{-1}$	$3 \cdot 10^{-6}$	0.0026%
	q_1	$7960103/160600$	$4.956477584 \cdot 10^{1}$	$4.956392093 \cdot 10^{1}$	$9 \cdot 10^{-4}$	0.0017%
	q_2	1	1	1	0	0.0000%

Tabelle 4.2: Minimierung der Funktion von Rosenbrock: analytisch und von WORHP ZEN berechnete Sensitiviäten der Nebenbedingungen

Sensitivität		analytisch		Worhp Zen	Fehler	
von	bzgl.	exakt	gerundet		absolut	relativ
f	p_1	1/160000	$6.250000000 \cdot 10^{-6}$	$6.250000517 \cdot 10^{-6}$	$5 \cdot 10^{-13}$	$< 0.0001\%$
	p_2	1	1	$9.999999974 \cdot 10^{-1}$	$6 \cdot 10^{-9}$	$< 0.0001\%$
	r_1	1/2	$5.000000000 \cdot 10^{-1}$	$5.000000013 \cdot 10^{-1}$	$1 \cdot 10^{-9}$	$< 0.0001\%$
	r_2	99/400	$2.4750000 \cdot 10^{-1}$	$2.475000011 \cdot 10^{-1}$	$1 \cdot 10^{-9}$	$< 0.0001\%$
	q_1	1/2	$5.000000000 \cdot 10^{-1}$	$5.000000276 \cdot 10^{-1}$	$3 \cdot 10^{-8}$	$< 0.0001\%$
	q_2	0	0	0	0	-

Tabelle 4.3: Minimierung der Funktion von Rosenbrock: analytisch und von Worhp Zen berechnete Sensitiviäten der Zielfunktion

Sensitivität		analytisch		Worhp Zen	Fehler	
von	bzgl.	exakt	gerundet		absolut	relativ
$\frac{\mathrm{d}f}{\mathrm{d}p_1}$	p_1	$-1/8030000$	$-1.245330012 \cdot 10^{-7}$	$-1.245298252 \cdot 10^{-7}$	$3 \cdot 10^{-12}$	0.0026%
	p_2	1/40150	$2.490660025 \cdot 10^{-5}$	$2.490596289 \cdot 10^{-5}$	$6 \cdot 10^{-10}$	0.0026%
	r_1	$-1/80300$	$-1.245330012 \cdot 10^{-5}$	$-1.245298145 \cdot 10^{-5}$	$3 \cdot 10^{-10}$	0.0026%
	r_2	1/80300	$1.245330012 \cdot 10^{-5}$	$1.245298145 \cdot 10^{-5}$	$3 \cdot 10^{-10}$	0.0026%
	q_1	3/160600	$1.867995019 \cdot 10^{-5}$	$1.876502925 \cdot 10^{-5}$	$8 \cdot 10^{-8}$	0.4555%
	q_2	0	0	$4.452704958 \cdot 10^{-22}$	$4 \cdot 10^{-22}$	-
$\frac{\mathrm{d}f}{\mathrm{d}p_2}$	p_1	1/40150	$2.490660025 \cdot 10^{-5}$	$2.490596289 \cdot 10^{-5}$	$6 \cdot 10^{-10}$	0.0026%
	p_2	$-4/803$	$-4.981320050 \cdot 10^{3}$	$-4.981192158 \cdot 10^{-3}$	$1 \cdot 10^{-7}$	0.0026%
	r_1	2/803	$2.490660025 \cdot 10^{-3}$	$2.490596079 \cdot 10^{-3}$	$6 \cdot 10^{-8}$	0.0026%
	r_2	$-2/803$	$-2.490660025 \cdot 10^{-3}$	$-2.490596071 \cdot 10^{-3}$	$6 \cdot 10^{-8}$	0.0026%
	q_1	800/803	$9.962640100 \cdot 10^{-1}$	$9.962469934 \cdot 10^{-1}$	$2 \cdot 10^{-5}$	0.0017%
	q_2	0	0	$-8.905409164 \cdot 10^{-20}$	$9 \cdot 10^{-20}$	-

Tabelle 4.4: Minimierung der Funktion von Rosenbrock: analytisch und von Worhp Zen berechnete zweite Sensitiviäten der Zielfunktion (ohne Anteil der linearen Störung, da dieser Tabelle 4.1 entspricht)

finiter Differenzen der Lagrange-Funktion mit einer Genauigkeit von 10^{-5} im nichtlinearen Fall (vgl. Folgerung 2.13). Im linearen Fall können die von Worhp berechneten Werte der optimalen Lösung mit einer Genauigkeit von 10^{-6} übernommen werden (vgl. Abschnitt 2.4.3).

Auffällig an den Tabellen 4.1, 4.2, 4.3 und 4.4 ist die große Anzahl an Null gesetzten Werten, die allein durch die Menge der aktiven Indizes bedingt sind. Bisher ist Worhp Zen zwar in der Lage mit dem aus Abschnitt 3.1.3 bekannten Speicherformat für dünn besetzte Matrizen zu rechnen, speichert die Null-Werte aber explizit ab. Die Matrizen der Sensitivitätsableitungen sind dicht besetzt. Mit Tabelle 4.5 wird deutlich, dass die Speicherung von 34.62% der Werte gespart werden kann.

Ein weiteres Einsparungspotenzial an Speicherbedarf bietet die zweite Sensitivitätsableitung der Zielfunktion, die bei linearen Störungen nach Abschnitt 2.4.3 der Sensitivitätsableitung der optimalen Lösung entspricht. Diese Einsparung würde 66.67% Werte (weitere 41.67%) von `ZenDf2` betreffen.

	Anzahl allgemein	im Beispiel	Proz. Anteil
ZenDx	$(N_g + N_h - \lvert\mathcal{I}(x^*, p_0)\rvert) \cdot N$	2	16.67%
ZenDmu	$(N_g + N_h - \lvert\mathcal{I}(x^*, p_0)\rvert)$ $\cdot (N + N_g + N_h + N_p + \lvert\mathcal{I}(x^*, p_0)\rvert)$	7	58.33%
ZenDg	$\lvert\mathcal{I}(x^*, p_0)\rvert \cdot (N + N_g + N_h + N_p)$	6	50.00%
ZenDf	$(N_g + N_h - \lvert\mathcal{I}(x^*, p_0)\rvert)$	1	16.67%
ZenDf2	$(N_g + N_h - \lvert\mathcal{I}(x^*, p_0)\rvert)$ $\cdot (2N + 2N_p + N_g + N_h + \lvert\mathcal{I}(x^*, p_0)\rvert)$	11	30.56%
gesamt		27	34.62%

Tabelle 4.5: Minimierung der Funktion von Rosenbrock: Anzahl und prozentualer Anteil an Null-Elementen in den Matrizen der Sensitivitätsableitungen

Eine Zeitmessung des WORHP ZEN Moduls findet in diesem Beispiel nicht statt, da die gesamte Ausführungszeit kleiner ist als die vom System bedingten Zeitschwankungen.

4.2 Lösung eines Optimalsteuerungsproblems

Exemplarisch für ein Optimalsteuerungsproblem wird das Splineproblem betrachtet, das auch als Minimum-Energy-Problem bekannt ist. Veranschaulichen lässt sich dieses Problem mit einem Stab in der (t, y)-Ebene, der an den Punkten $(0, 0)$ und $(1, 0)$ so befestigt ist, dass ihm eine Biegung aufgezwungen wird. Stab und t-Achse sollen sich an den Punkten mit den Winkeln 45° und −45° schneiden (vgl. linken Graphen der optimalen Lösung in Abbildung 4.2). Es ist bekannt, dass der Stab eine Form mit minimaler Biegeenergie annimmt. Zusätzlich soll die Krümmung des Stabes hier auf den Betrag 6 begrenzt werden.

4.2.1 Problembeschreibung

Mathematisch wird das Splineproblem folgendermaßen formuliert: Es werden Funktionen $x : \mathbb{R} \to \mathbb{R}^3$ und $u : \mathbb{R} \to \mathbb{R}$ gesucht, die das System von Differentialgleichungen

$$\dot{x}_1 = x_2 \qquad \dot{x}_2 = u \qquad \dot{x}_3 = u^2$$

mit den Anfangs- und Endwerten

$$x_1(0) = 0 \qquad x_2(0) = 1 \qquad x_3(0) = 0 \qquad x_1(1) = 0 \qquad x_2(1) = 1$$

unter Einhaltung der Einschränkung

$$u(t) \in [-6, 6] \quad \forall t \in [0, 1]$$

erfüllen und gleichzeitig $x_3(1)$ minimieren. Dabei steht $x_1(t)$ für die Position des Stabes, $x_2(t)$ für dessen Steigung und $x_3(t)$ für die bis zum Punkt t benötigte Biegeenergie. Die Steuerungsvariable u nimmt Einfluss auf die Krümmung des Stabes. Da in dieser Arbeit die rein mathematische Beschreibung des Problems im Vordergrund steht, wird im Folgenden auf Einheiten verzichtet.

Optimalsteuerungsprobleme lassen sich durch Diskretisierung in Optimierungsprobleme überführen. Mit Anwendung des Euler-Verfahrens zur approximativen Beschreibung der Differentialgleichungen lautet das als Optimierungsproblem formulierte Splineproblem dann:

$$\begin{aligned}
\min_{x\in\mathbb{R}} \quad & x_3^{(N_t)} \\
\text{unter} \quad & x_1^{(i+1)} = x_1^{(i)} + h x_2^{(i)} \\
& x_2^{(i+1)} = x_2^{(i)} + h u^{(i)} \qquad i = 0, \ldots, N_t - 1 \\
& x_3^{(i+1)} = x_3^{(i)} + h {u^{(i)}}^2 \\
& x_1^{(0)} = 0, \qquad x_2^{(0)} = 1, \qquad x_3^{(0)} = 0 \\
& x_1^{(N_t)} = 0, \qquad x_2^{(N_t)} = 1 \\
& u^{(i)} \leq 6 \qquad u^{(i)} \geq -6 \qquad i = 0, \ldots, N_t
\end{aligned} \tag{4.2}$$

Es wurde in 4.2 eine Diskretisierung mit $N_t + 1$ Stützstellen und einer Schrittweite von $h = \frac{1}{N_t}$ gewählt.

Details zur Theorie der Optimalsteuerungsprobleme und deren Approximation durch Optimierungsprobleme können in Büskens [2] nachgelesen werden.

4.2.2 Auswertung

WORHP berechnet den Zielfunktionswert der optimalen Lösung zu $f = 12.000$. Der Verlauf der Funktionen $x(t)$ und $u(t)$ ist in Abbildung 4.2 veranschaulicht.

Im Folgenden wird die optimale Lösung gestört, mit Hilfe von WORHP ZEN eine approximierte Lösung mittels der Echtzeit-Optimierung aus Abschnitt 2.4.4 berechnet und mit den optimalen Lösung (berechnet mit WORHP) verglichen. Hintergrund ist der Wunsch, ein gestörtes, hochdimensionales Optimierungsproblem effizient lösen zu können ohne zwingend einen kompletten SQP-Prozess zu durchlaufen. Es werden folgende zwei Störungen betrachtet

$$\begin{aligned}
&\text{Störung 1:} \quad && x_1^{(0)} = 0 + 0.1 \\
&\text{Störung 2:} \quad && x_1^{(0)} = 0 + 0.1 \qquad x_3^{(0)} = 0 + 0.1 \qquad x_1^{(N_t)} = 0 - 0.1
\end{aligned} \tag{4.3}$$

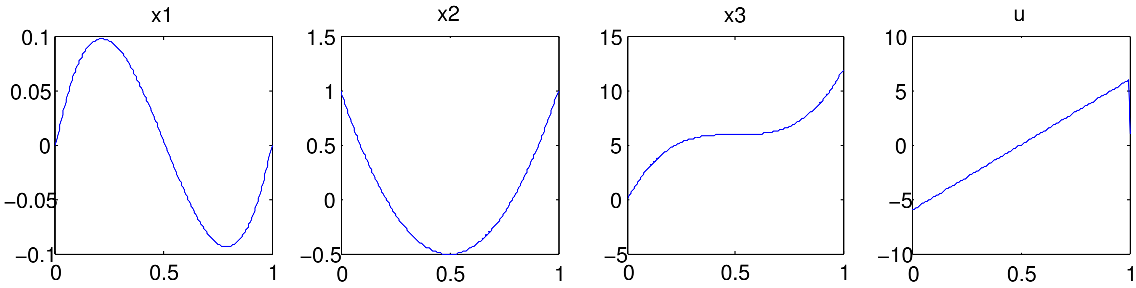

Abbildung 4.2: Splineproblem: Optimale Lösung

d.h. die Randbedingungen des Differentialgleichungssystems werden variiert. Bei den folgenden Auswertungen wird eine Diskretisierung mit 200 Stützstellen gewählt. Der Quellcode ist der Arbeit in Abschnitt C.2 angehängt.

In den Abbildungen 4.3 und 4.5 sind die Kurvenverläufe der optimalen und der durch WORHP ZEN approximierten Lösungen sowie in den Abbildungen 4.4 und 4.6 deren Differenz veranschaulicht. Zusätzlich befinden sich in den Tabellen 4.6 sowie 4.7 Auflistungen der absoluten und relativen Fehler der durch Diskretisierung entstandenen Vektoren x_1, x_2, x_3 und u.

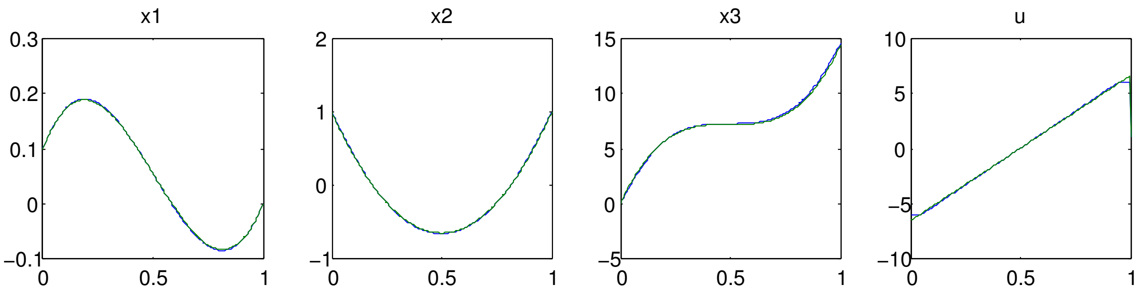

Abbildung 4.3: Splineproblem unter Einfluss von Störung 1: optimale Lösung von WORHP (blau), approximierte Lösung von WORHP ZEN (grün))

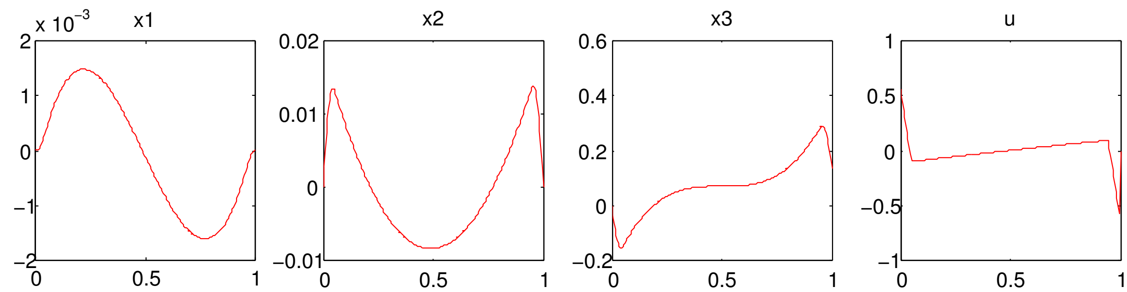

Abbildung 4.4: Approximationsfehler beim Splineproblem unter Störung 1: Jeweils Differenz der optimalen Lösung und der approximierten von WORHP ZEN

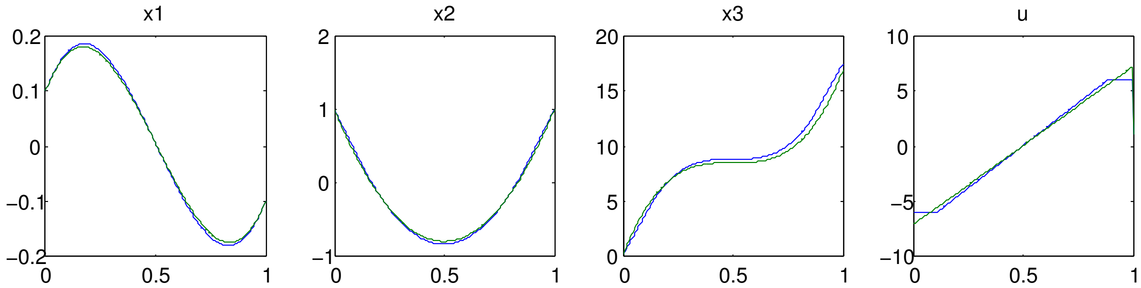

Abbildung 4.5: Splineproblem unter Einfluss von Störung 2: optimale Lösung von WORHP (blau), approximierte Lösung von WORHP ZEN (grün)

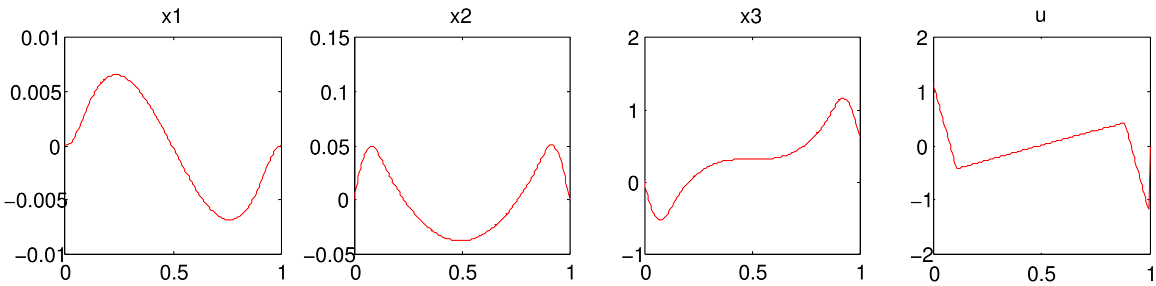

Abbildung 4.6: Approximationsfehler beim Splineproblem unter Störung 2: Jeweils Differenz der optimalen Lösung und der approximierten von WORHP ZEN

	x_1	x_2	x_3	u
absoluter Fehler	0.0155	0.1002	1.6984	1.6558
relativer Fehler	0.95%	1.38%	1.54%	3.08%

Tabelle 4.6: Normierter Fehler (in der euklidischen Norm) bei Approximation des Splineproblems durch WORHP ZEN bei Störung 1

	x_1	x_2	x_3	u
absoluter Fehler	0.0652	0.4263	7.4642	5.0873
relativer Fehler	3.32%	5.02%	5.55%	8.64%

Tabelle 4.7: Normierter Fehler (in der euklidischen Norm) bei Approximation des Splineproblems durch WORHP ZEN bei Störung 2

Die von WORHP ZEN berechnete Lösung entspricht weitestgehend der optimalen. Deutliche Unterschiede sind bei der Krümmung u sichtbar, die durch die von WORHP ZEN ignorierten Boxschranken entstehen. Dies ist ein Beispiel für die Überlegungen aus Abschnitt 2.4.3, dass die approximierte Lösung im Allgemeinen die Nebenbedingungen verletzt. Dadurch entstehen die relativen Fehler von bis zu 8.64%. KNAUER [13] beschreibt zur Minimierung dieser Fehler ein von BÜSKENS vorgeschlagenes Nachkorrekturverfahren.

	$f(p)$ ohne `ZenDf2`	$f(p)$ mit `ZenDf2`	$x_3^{(N_t)}(p)$	WORHP
Störung 1	14.40036512	14.52175370	14.40035227	14.53407727
Störung 2	16.90042698	17.38635004	16.90040193	17.51007122

Tabelle 4.8: Echtzeitoptimierung des Splineproblems: WORHP ZEN Approximationen der Zielfunktion im Vergleich

	$f(p)$ ohne `ZenDf2`		$f(p)$ mit `ZenDf2`		$x_3^{(N_t)}(p)$	
	absolut	relativ	absolut	relativ	absolut	relativ
Störung 1	0.13371215	0.92%	0.012323570	0.08%	0.13372500	0.92%
Störung 2	0.60964424	3.48%	0.123721180	0.71%	0.60966929	3.48%

Tabelle 4.9: Echtzeitoptimierung des Splineproblems: Fehler der WORHP ZEN Approximationen der Zielfunktion

Insgesamt darf die gute Übereinstimmung der Kurven in den Abbildungen 4.3 und 4.5 sowie die relativ geringen Fehler als ein Indiz der Gültigkeit der Sensitivitätsableitungen von WORHP ZEN betrachtet werden. Gleichheit kann generell aufgrund der TAYLOR-Approximation nicht erwartet werden.

Besonders gute Approximationen lassen sich unter Zuhilfenahme der zweiten Sensitivitätsableitung der Zielfunktion `ZenDf2` erzielen. Den Tabellen 4.8 und 4.9 ist zu entnehmen, dass dadurch die Fehler der Approximationen beider Störungen deutlich auf unter 1% verkleinert werden. Erwartungsgemäß nach Definition des Splineproblems 4.2 sind die TAYLOR-Approximation der Zielfunktion $f(p)$ und die Approximation des Elements der optimalen Lösung $x_3^{(n)}(p)$ nahezu identisch. Dies lässt auf eine ähnlich gute Qualität der Sensitivitätsableitung der optimalen Lösung und der Zielfunktion schließen.

Die Zeitmessung des Splineproblems in Abhängigkeit der Anzahl der Stützstellen (siehe Abbildung 4.7) zeigt, wie in Abschnitt 3.2.1 beschrieben, die effiziente Bestimmung der Sensitivitätsableitungen der optimalen Lösung $(x^*, \mu^*)^T$. Bei 1000 Stützstellen, d.h. 4000 Optimierungsvariablen, brauchte WORHP ZEN ca. 8 Sekunden für die Berechnung von `ZenDx` und `ZenDmu`. WORHP benötigte die doppelte Zeit um in 8 Iterationen die optimale Lösung zu finden. Abbildung 4.7 veranschaulicht jedoch auch die ineffiziente Bestimmung der restlichen Sensitivitätsableitungen. Besonders `ZenDf2` und `ZenDg` können aufgrund der Matrixmultiplikationen in den Folgerungen 2.12 und 2.14 nur sehr aufwändig berechnet werden.

Die Echtzeit-Optimierung von WORHP ZEN benötigte für beide Störungen bei 1000 Stützstellen unter zusätzlicher Verwendung der zweiten Sensitivitätsableitung der Zielfunktion `ZenDf2` ca. 4 Sekunden. Die Sparsity des Vektors $p-p_0$ wird von WORHP ZEN derzeit nicht ausgenutzt. Die Matrix-Vektor-Multiplikation für die optimale

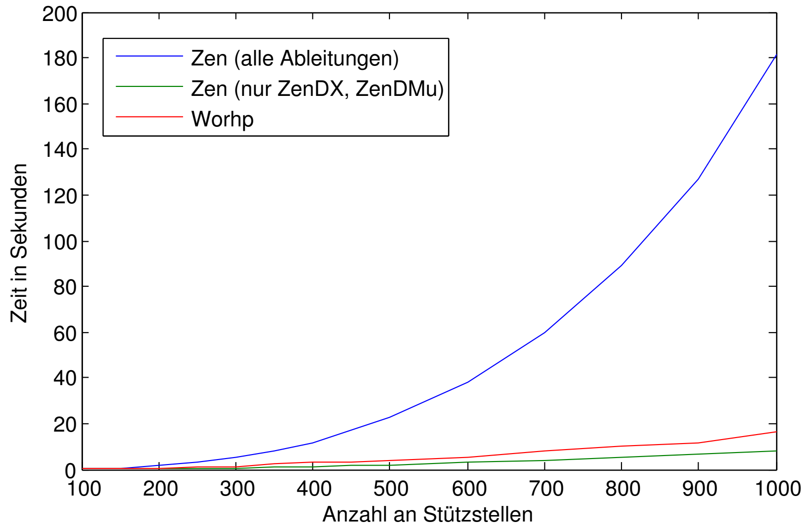

Abbildung 4.7: Zeitmessung beim Splineproblem

Lösung, bzw. Vektor-Vektor-Multiplikation für die Zielfunktion, in der TAYLOR-Approximation nach Abschnitt 2.4.4 sind bei einer Störung von 3 Werten allerdings nicht notwendig. Hier sind weitere Effizienzsteigerungen möglich.

Kapitel 5

Zusammenfassung und Ausblick

Die Zielsetzung dieser Arbeit bestand in der Entwicklung, Vorstellung und Analyse des parametrischen Sensitivitätsanalysemoduls WORHP ZEN. In Kapitel 3 wurde beschrieben, dass WORHP ZEN eine direkte Umsetzung der in Kapitel 2 dargestellten mathematischen Theorie der Optimierung ist. Kapitel 4 zeigte exemplarisch die zuverlässige Berechnung der Sensitiviäten beliebiger Störungen durch WORHP ZEN. Die ermittelten relativen Fehler der Sensitivitäten lagen alle deutlich unter 1%. Selbst bei der Approximation einer optimaler Lösung durch WORHP ZEN von leicht gestörten Systemen konnten ähnliche Fehlerwerte erzielt werden. Eine Zeitmessung belegte darüber hinaus die effiziente Berechnung der Sensitivitäten der optimalen Lösung.

Kapitel 4 zeigte demgegenüber aber auch weiteren Entwicklungsbedarf. So könnte die weitreichende Ausnutzung der Information über die Menge der aktiven Indizes signifikante Effizienzsteigerungen sowie Minimierungen des Speicherplatzbedarfes bedeuten. Derzeit werden Sensitivitäten berechnet, von denen aus den theoretischen Überlegungen im Vorhinein klar ist, dass sie den Wert Null besitzen. Außerdem sollte bei der hochdimensionalen zweiten Sensitivitätsableitung der Zielfunktion die redundante Abspeicherung der Sensitivitäten der optimalen Lösung für den linearen Fall vermieden werden.
Eine Verbesserung der Echtzeit-Optimierung von WORHP ZEN könnte durch einen zusätzlichen Nachkorrekturschritt erzielt werden.

Darüber hinaus ist eine tiefere Integration in WORHP wünschenswert damit beispielsweise das Zusammenspiel von WORHP ZEN und der Skalierung sowie dem BFGS Update Verfahren keine Fehlerwerte in den Sensitivitäten produziert. Um unnötige Code Redundanzen zu vermeiden, müssten bisher von WORHP ZEN durchgeführte Ableitugnsberechnungen von dem WORHP Modul FIDIF geleistet werden und der Nachkorrekturschritt aus NIKOLAYZIK [15] Verwendung von WORHP ZEN machen.

Abgesehen von diesen kleinen Optimierungsmöglichkeiten, sind die Ergebnisse von WORHP ZEN sehr zufriedenstellend.

Anhang A

Ausgewählte Variablen und Parameter von WORHP

In diesem Anhang soll eine Auswahl der wichtigsten Variablen und Parameter von WORHP hinsichtlich der Entwicklung eines parametrischen Sensitivitätsanalysemoduls wie WORHP ZEN sowie deren Benutzung bereit gestellt werden. Das bedeutet, dass in der folgenden Übersicht insbesondere alle Variablen und Parameter vorkommen, die in den Programmcodes der Beispiele 4.1 und 4.2 vorkommen.

Variablen des Datentypes `OptVar`:

`n [integer]` ≥ 0 . . . .
    Dimension der Optimierungsvariable.

`m [integer]` ≥ 0 . . . .
    Anzahl der Nebenbedingungen.

`F [double]` *0.0*
    Auswertung der Zielfunktion im aktuellen Punkt x.

`X [double(n)]` . . . .
    Optimierungsvariable des Optimierungsproblems.

`XL [double(n)]` . . . .
    Untere Boxschranke der Optimierungsvariable.

`XU [double(n)]` . . . .
    Obere Boxschranke der Optimierungsvariable.

`G [double(m)]` . . . .
    Auswertung der Nebenbedingungen im aktuellen Punkt x. Dieser Vektor wird nur initialisiert, wenn $m > 0$ gilt.

`GL` `[dobule(m)]` ....
Untere Schranke für die Nebenbedingungen.

`GU` `[double(m)]` ....
Obere Schranke für die Nebenbedingungen.

`Mu` `[double(m)]` ....
LAGRANGE-Multiplikatoren für die Nebenbedingungen.

Variablen des Datentyps `Workspace`:

`DF` `[WorhpMatrix]` ....
Gradient der Zielfunktion bzgl. der Optimierungsvariable ausgewertet im aktuellen Punkt x.

`DG` `[WorhpMatrix]` ....
JACOBI-Matrix der Nebenbedingungen bzgl. der Optimierungsvariable ausgewertet im aktuellen Punkt x.

`DL` `[WorhpMatrix]` ....
Gradient der LAGRANGE-Funktion bzgl. der Optimierungsvariable ausgewertet im aktuellen Punkt x.

`HM` `[WorhpMatrix]` ....
HESSE-Matrix der LAGRANGE-Funktion bzgl. der Optimierungsvariable ausgewertet im aktuellen Punkt x.

`ScaleObj` `[double]` *1.0*
Skalierungswert der Zielfunktion.

`IWMT` `[size_t]` *0*
Integer Workspace, in dem alle temporären Variablen vom Typ *Integer* gespeichert sind.

`RWMT` `[size_t]` *0*
Real Workspace, in dem alle temporären Variablen vom Typ *Double* gespeichert sind.

`scalecona` `[rwmt_index]` ....
Skalierungswert für die Gleichungsnebenbedinungen.

`scaleconc` `[rwmt_index]` ....
Skalierungswert für die Ungleichungsnebenbedingungen.

`Calls` `[counter]` ....
Zähler für die Aufrufe der Hauptschleife von WORHP .

`RelaxNVar` `[integer]` *1*
Anzahl der verwendeten Relaxierungsvariablen beim quadratischen Subproblem[1].

Variablen des Datentyps `Control`:

`Timer` `[TimerType]` ....
Timer des gesamten WORHP Prozesses mit einer Auflösung von 40ms.

`UserAction` `[boolean(NUserAction)]` .... *(true,false,false,...)*
Vektor, der angibt, welche User Action auszuführen sind. Zu den User Actions zählen u.a.:

callWorhp Gibt an, ob die Hauptschleife von Worhp aufgerufen werden soll. Stets auf *true*.

evalF Verlangt die Auswertung der Zielfunktion.

evalG Verlangt die Auswertung der Nebenbedingungen.

evalDF Verlangt die Auswertung des Gradienten der Zielfunktion.

evalDG Verlangt die Auswertung der JACOBI-Matrix der Nebenbedingungen.

evalHM Verlangt die Auswertung der HESSE-Matrix der LAGRANGE-Funktion.

evalZenDG Verlangt die Auswertung der JACOBI-Matrix der Nebenbedingungen bzgl. des nichtlinearen Störparameters p.

evalZenDL Verlangt die Auswertung der HESSE-Matrix der LAGRANGE-Funktion bzgl. x und des nichtlinearen Störparameters p.

Konstanten des Datentyps `Params`:

`Infty` `[double]` .... *1.0e+20*
Grenze ab der eine Schranke als unendlich betrachtet wird.

`Timeout` `[double]` > 0 .... *300.0*
Zeitliche Begrenzung des WORHP Prozesses.

[1]Genauere Informationen zur in WORHP verwendeten Relaxierung können in [17] gefunden werden

`TolComp` `[double]` > 0 .... *1.0e-6*
Genauigkeit für die Einhaltung der Nebenbedingungen.

`TolOpti` `[double]` > 0 .... *1.0e-6*
Genauigkeit für das Erreichen der optimalen Lösung.

`MaxIter` `[integer]` > 0 .... *500*
Maximale Ausführung an Iterationen um eine Endlosschleife zu verhindern.

`NLPprint` `[integer]` *2*
Stellt den Umfang der Ausgabe ein.

`UserDF` `[boolean]` .... *true*
Schalter um einzustellen ob die Ableitung der Zielfunktion vom User bereitgestellt wird, oder von WORHP numerisch berechnet werden soll.

`UserDG` `[boolean]` .... *true*
Schalter um einzustellen ob die JACOBI-Matrix der Nebenbedingungen vom User bereitgestellt wird, oder von WORHP numerisch berechnet werden soll.

`UserHM` `[boolean]` .... *true*
Schalter um einzustellen ob die HESSE-Matrix der LAGRANGE-Funktion vom User bereitgestellt wird, oder von WORHP numerisch berechnet werden soll.

Anhang B

Variablen und Parameter von Worhp Zen

In diesem Anhang werden alle zum Worhp Zen Modul gehörenden Variablen und Parameter aufgelistet und ihre Funktion erläutert.

Variablen im Datentype `OptVar`:

`k [integer]` ≥ 0 ....
    Dimension des nichtlinearen Störparameter p.

`P [double(k)]` ....
    Referenzparameter der nichtlinearen Störung p.

Variablen im Datentyp `Workspace`:

`ZenDX [WorhpMatrix]` ....
    Sensitivitätsableitung der optimalen Lösung x^*.

`ZenDMu [WorhpMatrix]` ....
    Sensitivitätsableitung der Lagrange-Multiplikatoren in der optimalen Lösung.

`ZenDF [WorhpMatrix]` ....
    Sensitivitätsableitung der Zielfunktion.

`ZenDF2 [WorhpMatrix]` ....
    Zweite Sensitivitätsableitung der Zielfunktion.

`ZenDG` [WorhpMatrix] ....
Sensitivitätsableitung der Nebenbedingungen.

`ZenDGp` [WorhpMatrix] ....
Ableitung der Nebenbedingungen nach dem nichtlinearen Störparameter p. Diese kann vom User bereit gestellt werden.

`ZenDLp` [WorhpMatrix] ....
Ableitung der LAGRANGE-Funktion nach dem nichtlinearen Störparameter p. Diese kann vom User bereit gestellt werden.

`ZenDLxp` [WorhpMatrix] ....
HESSE-Matrix der LAGRANGE-Funktion nach x und dem nichtlinearen Störparameter p. Diese kann vom User bereit gestellt werden.

`ZenPRQ` [rwmt_index] ....
Vektor, der die nichtlinearen Störparameter mit den linearen und konstanten in der Form $(p^T, r^T, q^T)^T$ vereint. Wird für die Echtzeit-Optimierung von WORHP ZEN benötigt.

Variablen im Datentyp `Control`:

`ZenRCcounter1` [counter] ....
Counter um die Reverse Communication zu handhaben. Der Counter 1 ist dafür zuständig innerhalb der Post-Optimalitätsanalyse zwischen den möglichen Punkten Initialisierung, Berechnung von Ableitungen und Ermittlung der Sensitivitätsableitungen zu unterscheiden.

`ZenRCcounter2` [counter] ....
Counter um die Reverse Communication zu handhaben. Der Counter 2 kommt bei der Finiten Differenzen Methode zum Einsatz.

Konstanten im Datentyp `Params`:

`UserZenDGp` [boolean] .... *true*
Schalter um einzustellen ob die Ableitung der Nebenbedingungen nach der nichtlinearen Störung p vom User bereit gestellt wird, oder von WORHP ZEN per Finiter Differenzen berechnet werden muss.

`UserZenDLp` [boolean] .... *true*
Schalter um einzustellen ob die Ableitung der LAGRANGE-Funktion nach der nichtlinearen Störung p vom User bereit gestellt wird, oder von WORHP ZEN per Finiter Differenzen berechnet werden muss.

`UserZenDLxp` `[boolean]` .... *true*

Schalter um einzustellen ob die Hesse-Matrix der Lagrange-Funktion nach x und der nichtlinearen Störung p vom User bereit gestellt wird, oder von Worhp Zen per Finiter Differenzen berechnet werden muss.

`ZenDxDmu` `[boolean]` .... *true*

Schalter um die Berechnung der Sensitivitätsableitung der optimalen Lösung ein-/auszuschalten.

`ZenDf` `[boolean]` .... *true*

Schalter um die Berechnung der Sensitivitätsableitung der Zielfunktion ein-/auszuschalten.

`ZenDf2` `[boolean]` .... *true*

Schalter um die Berechnung der zweiten Sensitivitätsableitung der Zielfunktion ein-/auszuschalten.

`ZenDg` `[boolean]` .... *true*

Schalter um die Berechnung der Sensitivitätsableitung der Nebenbedingungen ein-/auszuschalten.

Anhang C

Quelltexte

C.1 Minimierung der Funktion von Rosenbrock

In diesem Abschnitt kann der Quelltext für die Berechnung der Minimierung der Funktion von Rosenbrock nach Abschnitt 4.1 nachgeschlagen werden. Es wurden bis auf Ausschalten der Skalierung die WORHP Standardparameter verwendet. Darüber hinaus wurden die Ableitungen `ZenDGp`, `ZenDLp` sowie `ZenDLxp` mittels Finiter Differenzen berechnet und nicht vom User bereitgestellt. Dies könnte aber in analoger Form zur Bereitsstellung der HESSE-Matrix oder dem Gradienten der Zielfunktion geschehen.

Listing C.1: Minimierung der Funktion von Rosenbrock

```
!-----------------------------------------------
!
! Minimize Rosenbrock's Function
!
! Subject to two constraints changing the usal
! optimal solution to (1/2, 99/400). Afterwards
! a sensitivity analysis will be performed by
! WORHP ZEN.
!
! @author Renke Schaefer
!-----------------------------------------------
PROGRAM Main

! WORHP workspace access macros
#include "../core/macros.h"

  ! WORHP user interface module
  USE Worhp_User

  ! WORHP data structures
```

```
IMPLICIT NONE
TYPE (OptVar)          :: opt
TYPE (Workspace)       :: wsp
TYPE (Params)          :: par
TYPE (Control)         :: cnt

INTEGER(WORHP_INT)     :: status

! Data structure pointers
REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: X, XL, XU, Lambda
REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: G, GL, GU, Mu
REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: P

! Matrix pointers
REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: DFval, DGval, HMval
INTEGER(WORHP_INT), DIMENSION(:), POINTER :: HMrow, HMcol

! Dimensions of variables, constraints and perturbation
opt%N = 2
opt%M = 2
opt%K = 2

! WORHP parameters
CALL InitParams(status, par)
par%NLPprint     = 2
par%UserDF       = TRUE
par%UserDG       = TRUE
par%UserHM       = TRUE
par%UserZenDGp   = FALSE
par%UserZenDLxp  = FALSE
par%UserZenDLp   = FALSE
par%ScaledQP     = TRUE
par%ZenDxDMu     = TRUE
par%ZenDf        = TRUE
par%ZenDf2       = TRUE
par%ZenDg        = TRUE
par%ZenFullStorage = TRUE

wsp%DF%nnz = WorhpMatrix_Init_Dense
wsp%DG%nnz = WorhpMatrix_Init_Dense
wsp%HM%nnz = WorhpMatrix_Init_Dense

! WORHP data structure initialisation routine.
CALL WorhpInit(opt, wsp, par, cnt)
IF (cnt%status /= FirstCall) THEN
   PRINT *, "example: Initialisation failed."
   STOP
END IF

! For C interoperability, some arrays have to be C-pointers.
```

```
! Convert them to Fortran pointers to access them (after
   WorhpInit)
CALL C_F_POINTER(opt%X,      X,      [opt%N])
CALL C_F_POINTER(opt%XL,     XL,     [opt%N])
CALL C_F_POINTER(opt%XU,     XU,     [opt%N])
CALL C_F_POINTER(opt%Lambda, Lambda, [opt%N])

CALL C_F_POINTER(opt%G,      G,      [opt%M])
CALL C_F_POINTER(opt%GL,     GL,     [opt%M])
CALL C_F_POINTER(opt%GU,     GU,     [opt%M])
CALL C_F_POINTER(opt%Mu,     Mu,     [opt%M])

CALL C_F_POINTER(opt%P,      P,      [opt%K])

CALL C_F_POINTER(wsp%DF%val,    DFval,  [wsp%DF%nnz])
CALL C_F_POINTER(wsp%DG%val,    DGval,  [wsp%DG%nnz])
CALL C_F_POINTER(wsp%HM%val,    HMval,  [wsp%HM%nnz])

! Initial guess for X, Lambda and Mu
X      = TWO
Lambda = ZERO
Mu     = ZERO

! Reference value for perturbation
P      = [100*ONE, ONE]

! Set lower and upper bounds for variables and constraints
XL = -par%Infty
XU = +par%Infty
GL = -par%Infty
GU = ZERO

! WORHP Reverse Communication loop.
DO WHILE (cnt%status < terminateSuccess .AND. cnt%status >
   terminateError)

   ! WORHP's main routine.
   IF (GetUserAction(cnt, callWorhp)) THEN
      CALL Worhp(opt, wsp, par, cnt)
   END IF

   ! Show iteration output.
   IF (GetUserAction(cnt, iterOutput)) THEN
      CALL IterationOutput(opt, wsp, par, cnt)
      CALL DoneUserAction(cnt, iterOutput)
   ENDIF

   ! Evaluate the objective function.
   IF (GetUserAction(cnt, evalF)) THEN
      CALL UserF(opt, wsp, par, cnt)
```

```
         CALL DoneUserAction(cnt, evalF)
      ENDIF

      ! Evaluate the constraints.
      IF (GetUserAction(cnt, evalG)) THEN
         CALL UserG(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalG)
      ENDIF

      ! Evaluate the gradient of the objective function.
      IF (GetUserAction(cnt, evalDF)) THEN
         CALL UserDF(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalDF)
      ENDIF

      ! Evaluate the Jacobian of the constraints.
      IF (GetUserAction(cnt, evalDG)) THEN
         CALL UserDG(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalDG)
      ENDIF

      ! Evaluate the Hessian matrix of the Lagrange function (L = f
         + mu*g)
      IF (GetUserAction(cnt, evalHM)) THEN
         CALL UserHM(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalHM)
      ENDIF

      ! Use finite differences with RC to determine derivatives
      IF (GetUserAction(cnt, fidif)) THEN
         CALL WorhpFidif(opt, wsp, par, cnt)
      ENDIF
   END DO

   ! Translate the WORHP status flag into a meaningful message.
   CALL StatusMsg(opt, wsp, par, cnt)

   CALL ZenStatusMsg(opt, wsp, par, cnt)

   PRINT *,''
   PRINT *,'Optimal solution: '
   PRINT *,'X: ', X
   PRINT *,'Mu: ', Mu

   !CALL PrintWorhpMatrix(wsp%ZenDX)
   !CALL PrintWorhpMatrix(wsp%ZenDMu)

   ! Deallocate all data structures.
   CALL WorhpFree(opt, wsp, par, cnt)

```

```
CONTAINS

  ! Update F
  SUBROUTINE UserF(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt
    opt%F = wsp%ScaleObj * (P(1) * (X(2) - X(1)**2)**2 + (P(2) - X
        (1))**2)
  END SUBROUTINE UserF

  ! Update G
  SUBROUTINE UserG(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt
    G = [X(1) + X(2) - 299.0/400.0, -P(1)*X(1)**2 + X(2)**2 - 50]
  END SUBROUTINE UserG

  ! Update DF
  SUBROUTINE UserDF(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt
    DFval = wsp%ScaleObj * [-4*P(1)*X(1)*X(2)+4*P(1)*X(1)**3-2*P(2)
        +2*X(1), 2*P(1)*X(2)-2*P(1)*X(1)**2]
  END SUBROUTINE UserDF

  ! Update DG
  SUBROUTINE UserDG(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt
```

```
    DGval = [ONE, -2*P(1)*X(1), ONE, 2*X(2)]
  END SUBROUTINE UserDG

  ! Update HM
  SUBROUTINE UserHM(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt

    ! HMval may be reallocated during first WORHP call
    CALL C_F_POINTER(wsp%HM%val, HMval,  [wsp%HM%nnz])

    ! Only scale the F part of HM
    HMval = [-wsp%ScaleObj*4*P(1)*X(1), wsp%ScaleObj*(-4*P(1)*X(2)
        +12*P(1)*X(1)**2+2)-2*Mu(2)*P(1), wsp%ScaleObj*2*P(1) + 2*Mu
        (2)]
  END SUBROUTINE UserHM

END PROGRAM Main
```

C.2 Lösung eines Optimalsteuerungsproblems

In diesem Abschnitt befindet sich der Quelltext für die Berechnung des Splineproblems aus Abschnitt 4.2. Bis auf das Ausschalten der Skalierung wurden die Standardparameter von WORHP verwendet. Die Ableitungen `ZenDGp`, `ZenDLp` sowie `ZenDLxp` wurden analog zum Optimierungsproblem in Abschnitt C.1 per Finiter Differenzen berechnet.

Listing C.2: Lösung eines Optimalsteuerungsproblems

```
!---------------------------------------------
!
! Optimal Control Problem
!
! The Spline Problem or Minimum Energy Problem
! is implemented. After performing a sensitivity
! analysis perturbed systems will be approximated
! by WORHP ZEN.
!
! @author Renke Schaefer
!---------------------------------------------
PROGRAM Main
```

```
! WORHP workspace access macros
#include "../core/macros.h"

  ! WORHP user interface module
  USE Worhp_User

  ! WORHP data structures
  IMPLICIT NONE
  TYPE (OptVar)          :: opt
  TYPE (Workspace)       :: wsp
  TYPE (Params)          :: par
  TYPE (Control)         :: cnt

  INTEGER(WORHP_INT)     :: status, Nt, i
  REAL(WORHP_DOUBLE)     :: h

  ! Data structure pointers
  REAL(WORHP_DOUBLE), POINTER :: Fnew
  REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: X, XL, XU, Lambda
  REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: G, GL, GU, Mu
  REAL(WORHP_DOUBLE), DIMENSION(:), POINTER :: P, Q, R, Xnew, Munew

  ! Matrix pointers
  REAL(WORHP_DOUBLE), DIMENSION(:), POINTER   :: DFval, HMval
  REAL(WORHP_DOUBLE), DIMENSION(:,:), POINTER :: DGval
  INTEGER(WORHP_INT), DIMENSION(:), POINTER   :: HMrow, HMcol

  ! Specifications for the discretization
  Nt = 200
  h = ONE/(Nt - 1)

  ! Dimensions of variables, constraints and perturbation
  opt%N = 4 * Nt
  opt%M = 3 * (Nt-1) +5

  ! WORHP parameters
  CALL InitParams(status, par)
  par%NLPprint     = 2
  par%ScaledQP     = TRUE
  par%ScaledObj    = FALSE
  par%UserDF       = TRUE
  par%UserDG       = TRUE
  par%UserHM       = TRUE
  par%UserZenDGp   = FALSE
  par%UserZenDLxp  = FALSE
  par%UserZenDLp   = FALSE
  par%ZenDxDMu     = TRUE
  par%ZenDf        = TRUE
  par%ZenDf2       = TRUE
  par%ZenDg        = TRUE
```

```
par%ZenFullStorage = FALSE

wsp%DF%nnz = WorhpMatrix_Init_Dense
wsp%DG%nnz = WorhpMatrix_Init_Dense
wsp%HM%nnz = opt%N

! WORHP data structure initialisation routine.
CALL WorhpInit(opt, wsp, par, cnt)
IF (cnt%status /= FirstCall) THEN
   PRINT *, "example: Initialisation failed."
   STOP
END IF

! For C interoperability, some arrays have to be C-pointers.
! Convert them to Fortran pointers to access them (after
   WorhpInit)
CALL C_F_POINTER(opt%X,      X,      [opt%N])
CALL C_F_POINTER(opt%XL,     XL,     [opt%N])
CALL C_F_POINTER(opt%XU,     XU,     [opt%N])
CALL C_F_POINTER(opt%Lambda, Lambda, [opt%N])

CALL C_F_POINTER(opt%G,      G,      [opt%M])
CALL C_F_POINTER(opt%GL,     GL,     [opt%M])
CALL C_F_POINTER(opt%GU,     GU,     [opt%M])
CALL C_F_POINTER(opt%Mu,     Mu,     [opt%M])

CALL C_F_POINTER(opt%P,      P,      [opt%K])

CALL C_F_POINTER(wsp%DF%val,    DFval,  [wsp%DF%nnz])
CALL C_F_POINTER(wsp%DG%val,    DGval,  [3*(Nt-1)+5,4*Nt])
CALL C_F_POINTER(wsp%HM%val,    HMval,  [wsp%HM%nnz])

! Initial guess for X, Lambda and Mu
X      = ONE
Lambda = ZERO
Mu     = ZERO

! Set lower and upper bounds for variables and constraints
XL = -par%Infty
XU = +par%Infty
DO i = 1, Nt
  XL(4*i) = -6*ONE
  XU(4*i) = 6*ONE
END DO
GL = ZERO
GU = ZERO

! Define HM as diagonal
IF (wsp%HM%NeedStructure) THEN
   CALL C_F_POINTER(wsp%HM%row, HMrow,  [wsp%HM%nnz])
```

```
      CALL C_F_POINTER(wsp%HM%col, HMcol,  [wsp%HM%nnz])
      DO i = 1,opt%N
        HMrow(i) = i
        HMcol(i) = i
      END DO
   END IF

   ! WORHP Reverse Communication loop.
   DO WHILE (cnt%status < terminateSuccess .AND. cnt%status >
      terminateError)

      ! WORHP's main routine.
      IF (GetUserAction(cnt, callWorhp)) THEN
         CALL Worhp(opt, wsp, par, cnt)
      END IF

      ! Show iteration output.
      IF (GetUserAction(cnt, iterOutput)) THEN
         CALL IterationOutput(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, iterOutput)
      ENDIF

      ! Evaluate the objective function.
      IF (GetUserAction(cnt, evalF)) THEN
         CALL UserF(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalF)
      ENDIF

      ! Evaluate the constraints.
      IF (GetUserAction(cnt, evalG)) THEN
         CALL UserG(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalG)
      ENDIF

      ! Evaluate the gradient of the objective function.
      IF (GetUserAction(cnt, evalDF)) THEN
         CALL UserDF(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalDF)
      ENDIF

      ! Evaluate the Jacobian of the constraints.
      IF (GetUserAction(cnt, evalDG)) THEN
         CALL UserDG(opt, wsp, par, cnt)
         CALL DoneUserAction(cnt, evalDG)
      ENDIF

      ! Evaluate the Hessian matrix of the Lagrange function (L = f
         + mu*g)
      IF (GetUserAction(cnt, evalHM)) THEN
         CALL UserHM(opt, wsp, par, cnt)
```

```
        CALL DoneUserAction(cnt, evalHM)
     ENDIF

     ! Use finite differences with RC to determine derivatives
     IF (GetUserAction(cnt, fidif)) THEN
        CALL WorhpFidif(opt, wsp, par, cnt)
     ENDIF
  END DO

  ! Translate the WORHP status flag into a meaningful message.
  CALL StatusMsg(opt, wsp, par, cnt)

  CALL ZenStatusMsg(opt, wsp, par, cnt)

  !PRINT *,' '

  ! write results of optimal solution to file
  OPEN(20, file='./bsc3_optimalSolution.txt', iostat=status, status
     ='replace', action='write')
  IF (status == 0) THEN
    WRITE(20,*) '#, x1, x2, x3, u'
    DO i = 0, Nt-1
      WRITE (20,*) i+1, ', ', X(4*i+1), ', ', X(4*i+2), ', ', X(4*i
         +3), ', ', X(4*i+4)
    END DO
    !PRINT *,'write results to file: OK'
  END IF
  CLOSE(20)

  ! perturbe the system and approximate optimal solution using
  ! WORHP ZEN.
  PRINT *,' '
  PRINT *,'Update Solution using Wohrp Zen: '
  PRINT *,'Disturb Q(3*(Nt-1)+1)'
  ALLOCATE(R(4 * Nt))
  ALLOCATE(Q(3 * (Nt-1) + 5))
  ALLOCATE(Xnew(4 * Nt))
  ALLOCATE(Munew(3 * (Nt-1) + 5))
  ALLOCATE(Fnew)
  R = ZERO
  Q = ZERO
  Q(3*(Nt-1)+1) = -ONE/10
  CALL ZenUpdateSolution(opt, wsp, par, cnt, deltaQ = Q, Xnew =
     Xnew, Fnew = Fnew, order = 2)
  PRINT *,'new objective function: ', Fnew

  ! write results of approximated optimal solution to file
  OPEN(20, file='./bsc3_x10m_zen.txt', iostat=status, status='
     replace', action='write')
  IF (status == 0) THEN
```

```
    WRITE(20,*) '#, x1, x2, x3, u'
    DO i = 0, Nt-1
      WRITE (20,*) i+1, ', ', Xnew(4*i+1), ', ', Xnew(4*i+2), ', ',
          Xnew(4*i+3), ', ', Xnew(4*i+4)
    END DO
    PRINT *,'write results to file: OK'
  END IF
  CLOSE(20)

  ! perturbe the system and approximate optimal solution using
  ! WORHP ZEN.
  PRINT *,' '
  PRINT *,'Disturb Q(3*(Nt-1)+1), Q(3*(Nt-1)+3), Q(3*(Nt-1)+4)'
  R = ZERO
  Q = ZERO
  Q(3*(Nt-1)+1) = -ONE/10
  Q(3*(Nt-1)+3) = -ONE/10
  Q(3*(Nt-1)+4) = ONE/10
  CALL ZenUpdateSolution(opt, wsp, par, cnt, deltaQ = Q, Xnew =
     Xnew, Fnew = Fnew, order = 2)
  PRINT *,'new objective function: ', Fnew

  ! write results of approximated optimal solution to file
  OPEN(20, file='./bsc3_x10m_x30_x1nt_zen.txt', iostat=status,
     status='replace', action='write')
  IF (status == 0) THEN
    WRITE(20,*) '#, x1, x2, x3, u'
    DO i = 0, Nt-1
      WRITE (20,*) i+1, ', ', Xnew(4*i+1), ', ', Xnew(4*i+2), ', ',
          Xnew(4*i+3), ', ', Xnew(4*i+4)
    END DO
    PRINT *,'write results to file: OK'
  END IF
  CLOSE(20)

  ! Deallocate all data structures.
  CALL WorhpFree(opt, wsp, par, cnt)


CONTAINS

  ! Update F
  SUBROUTINE UserF(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt
```

```
    opt%F = wsp%ScaleObj * X(4*(Nt-1)+3)
  END SUBROUTINE UserF

  ! Update G
  SUBROUTINE UserG(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)         :: opt
    TYPE (Workspace)      :: wsp
    TYPE (Params)         :: par
    TYPE (Control)        :: cnt
    INTENT (INOUT)        :: opt, wsp
    INTENT (IN)           :: par, cnt

    DO i = 0,Nt-2
      G(3*i+1) = -X(4*(i+1)+1) + X(4*i+1) + h*X(4*i+2)
      G(3*i+2) = -X(4*(i+1)+2) + X(4*i+2) + h*X(4*i+4)
      G(3*i+3) = -X(4*(i+1)+3) + X(4*i+3) + h*X(4*i+4)**2
    END DO
    G(3*(Nt-1)+1) = X(1)
    G(3*(Nt-1)+2) = X(2) - 1
    G(3*(Nt-1)+3) = X(3)
    G(3*(Nt-1)+4) = X(4*(Nt-1)+1)
    G(3*(Nt-1)+5) = X(4*(Nt-1)+2) - 1
  END SUBROUTINE UserG

  ! Update DF
  SUBROUTINE UserDF(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)         :: opt
    TYPE (Workspace)      :: wsp
    TYPE (Params)         :: par
    TYPE (Control)        :: cnt
    INTENT (INOUT)        :: opt, wsp
    INTENT (IN)           :: par, cnt

    DFval              = ZERO
    DFval(4*(Nt-1)+3)  = wsp%ScaleObj
  END SUBROUTINE UserDF

  ! Update DG
  SUBROUTINE UserDG(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)         :: opt
    TYPE (Workspace)      :: wsp
    TYPE (Params)         :: par
    TYPE (Control)        :: cnt
    INTENT (INOUT)        :: opt, wsp
    INTENT (IN)           :: par, cnt

    DGval = ZERO
```

```
    DO i = 0,Nt-2
      DGval(3*i+1,4*i+1)      = ONE
      DGval(3*i+1,4*(i+1)+1)  = -ONE
      DGval(3*i+1,4*i+2)      = h
      DGval(3*i+2,4*i+2)      = ONE
      DGval(3*i+2,4*(i+1)+2)  = -ONE
      DGval(3*i+2,4*i+4)      = h
      DGval(3*i+3,4*i+3)      = ONE
      DGval(3*i+3,4*(i+1)+3)  = -ONE
      DGval(3*i+3,4*i+4)      = TWO*h*X(4*i+4)
    END DO
    DGval(3*(Nt-1)+1,1)  = ONE
    DGval(3*(Nt-1)+2,2)  = ONE
    DGval(3*(Nt-1)+3,3)  = ONE
    DGval(3*(Nt-1)+4,4*(Nt-1)+1)  = ONE
    DGval(3*(Nt-1)+5,4*(Nt-1)+2)  = ONE
  END SUBROUTINE UserDG

  ! Update HM
  SUBROUTINE UserHM(opt, wsp, par, cnt)
    IMPLICIT NONE
    TYPE (OptVar)        :: opt
    TYPE (Workspace)     :: wsp
    TYPE (Params)        :: par
    TYPE (Control)       :: cnt
    INTENT (INOUT)       :: opt, wsp
    INTENT (IN)          :: par, cnt

    ! HMval may be reallocated during first WORHP call
    CALL C_F_POINTER(wsp%HM%val, HMval,  [wsp%HM%nnz])

    HMval = ZERO
    DO i = 0,Nt-2
      HMval(4*i+4) = Mu(3*i+3)*TWO*h
    END DO
  END SUBROUTINE UserHM

END PROGRAM Main
```

Literaturverzeichnis

[1] A. Antoniou and W.-S. Lu. *Practical optimization: algorithms and engineering applications*. Springer, New York, 2007.

[2] C. Büskens. *Optimierungsmethoden und Sensitivitätsanalyse für optimale Steuerprozesse mit Steuer- und Zustands-Beschränkungen*. Dissertation, Münster, 1998.

[3] C. Büskens. *Numerische Mathematik I*. Universität Bremen, 2004. Vorlesungsskript.

[4] C. Büskens, T. Nikolayzik, and M. Gerdts. *Nonlinear large-scale Optimization with WORHP*. Berichte aus der Technomathematik 10-08, Zentrum für Technomathematik, Universität Bremen, 2010.

[5] C. Büskens, T. Nikolayzik, and D. Wassel. *Interface Control Document*. Berichte aus der Technomathematik 08-01, Zentrum für Technomathematik, Universität Bremen, 2008.

[6] C. Büskens, T. Nikolayzik, D. Wassel, and P. Kalmbach. *Solver Development Strategy*. Berichte aus der Technomathematik 08-02, Zentrum für Technomathematik, Universität Bremen, 2008.

[7] R. Fletcher. *Practical methods of optimization*. A Wiley-Interscience publication. Wiley, Chichester [u.a.], 2. ed., reprinted edition, 1999.

[8] O. Forster. *Analysis 2: Differentialrechnung im* $\mathbb{R}^n$*, gewöhnliche Differentialgleichungen*. Vieweg-Studium, Grundkurs Mathematik. Vieweg, Wiesbaden, 7., verb. aufl. edition, 2006.

[9] Zentrum für Technomathematik (ZeTeM). *Steuerungs-Lab*. `http://www.math.uni-bremen.de/zetem/o2c/steuerung`, Juni 2012.

[10] M. Gerdts. *Optimierung*. University of Birmingham, 2008. Vorlesungsskript.

[11] M. Gerdts. *User's Guide QP Solver*. Universität der Bundeswehr München, 2011.

[12] D. Jungnickel. *Optimierungsmethoden: eine Einführung.* Springer-Lehrbuch. Springer, Berlin [u.a.], 2. aufl. edition, 2008.

[13] M. Knauer. *Sensitivitätsanalyse verschiedener Gütekriterien bei der optimalen Bahnplanung von Industrierobotern.* Diplomarbeit, Bayreuth, 2001.

[14] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950*, pages 481 – 492, Berkeley and Los Angeles, 1951. University of California Press.

[15] T. Nikolayzik. *Korrekturverfahren zur numerischen Lösung nichtlinearer Optimierungsprobleme mittels Methoden der parametrischen Sensitivitätsanalyse.* PhD thesis, 2012.

[16] P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung.* Internationale Schriftenreihe zur numerischen Mathematik, Lehrbuch. Birkhäuser, Basel [u.a.], 1993.

[17] Steinbeis Forschungszentrum Optimierung, Steuerung und Regelung, Grasberg. *User's Guide to WORHP 1.0*, 2012.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und ausschließlich die angegebenen Quellen und Hilfsmittel verwendet habe.

Bremen, den 27. August 2012