Institut für Parallele und Verteilte Systeme

Abteilung Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Bachelorarbeit

Erklären und Visualisieren von NLP-Lösungen

Jochen Wimpff

Studiengang: B. Sc. Informatik

Prüfer: Prof. Dr. M. Toussaint

Betreuer: Prof. Dr. M. Toussaint/ Prof. Daniel Weiskopf

begonnen am: 06.05.2019

beendet am: 06.11.2019

Aufgabenstellung

"Constrained optimization has many applications, including in robotics and design. A core problem with formulating optimization problems (i.e. specifying the costs and constraints of an NLP), is that it is hard to understand the solutions that an optimizer comes up with. The KKT conditions are differentiable, and in principle solutions should be 'explainable'. This project explores fundamentally how optimization algorithms can not only output a solution, but also a (visual) explanation for this solution. That is, an explanation for why the found solution is a solution, or whatmake a certain configuration infeasible, where the costs come from, and how the solution would change if you modify parameters of the solution. The goal is to develop visualization techniques that allow the user to intuitively grasp solutions. In turn, this supports the user to more easily specify optimization problem."[1]

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Motivation und Ziele der Arbeit 5

1.2 Vorgehensweise und Aufbau der Arbeit 6

2 Grundlagen der Optimierungstheorie **7**

2.1 Optimierung ohne Nebenbedingungen 7

2.1.1 Notwendige Bedingung erster Ordnung 7

2.1.2 Notwendige Bedingung zweiter Ordnung 7

2.1.3 Hinreichende Bedingung zweiter Ordnung 8

2.2 Numerische Verfahren 10

2.2.1 Newton-Verfahren 10

2.2.2 Backtracking Line Search 11

2.3 Lösungsverfahren von Optimierungsproblemen mit Nebenbedingungen . . . 12

2.3.1 Lagrange Funktion und Lagrange Dualität 13

2.3.2 Karush-Kuhn-Tucker-Bedingungen 13

2.3.3 Fritz-John-Bedingungen 18

2.3.4 Notwendige Bedingung zweiter Ordnung 18

2.3.5 Hinreichende Bedingung zweiter Ordnung 19

2.3.6 Regularitätsbedingungen 20

2.4 Weitere Lösungsverfahren 22

2.4.1 Barriere Methoden 22

2.4.2 Penalty Methoden 22

2.4.3 Erweiterte Lagrange Methode 24

2.5 Sensitivitätsanalyse 28

3 Debuggingtool für NLP Lösungen **32**

3.1 Aufbau und Untersuchungsmöglichkeiten 33

3.1.1 Overview 34

3.1.2 Important Spots View 34

3.1.3 Cluster View 36

3.1.4 Sensitivity Analysis View 38

3.2 Untersuchung einer Lösung 39

4 Zusammenfassung und Ausblick **44**

Abbildungsverzeichnis

1 Teilausschnitt der Funktion $f(x) = x_1^2 + x_2^2$ 9
2 Plot von Konturlinien 16
3 Plot der Logarithmus-Barrierefunktion 23
4 Plot der Funktion $p(\mu) = [\underset{x}{min}\ x^2 + \mu(x-1)^2$ 25
5 Unterschied Logarithmus-Barrierefunktion und Squared Penalty 26
6 Alle Kosten und Strafen 42
7 Screenshot 1 Debuggingtool 42
8 Screenshot 2 Debuggingtool 43
9 Screenshot 3 Debuggingtool 43
10 Screenshot 4 Debuggingtool 44

1 Einleitung

Optimierungsprobleme unter Nebenbedingungen (constrained optimization) haben ein großes Anwendungsfeld. Im Design von Bauteilen werden sie beispielsweise verwendet, um Material zu sparen aber dennoch die Stabilität des Bauteils zu gewährleisten. Ein weiteres Anwendungsfeld findet sich in der Robotik, wo Optimierungsprobleme mit Nebenbedingungen für das Finden von optimalen Roboter-Trajektorien (Bewegungsbahn) ohne Kollisionen genutzt wird. Es gibt viele verschiedene Tools, die für solche Probleme optimiert wurden, um so schnell wie möglich eine Lösung zu berechnen. Ein Hauptproblem liegt allerdings darin, die Lösung, die basierend auf der Problemformulierung berechnet wurde, zu verstehen. Um bei diesem Verständnis zu helfen und somit für eine genauere und bessere Formulierung von Optimierungsproblemen unter Nebenbedingungen zu sorgen, wurde im Zuge dieser Arbeit eine Art Debuggingtool entwickelt. Dieses Tool erforscht grundlegend, wie man eine gegebene Lösung von (nichtlinearen) Optimierungsproblemen unter Nebenbedingungen untersuchen kann, um eine Vorstellung davon zu bekommen, warum der Optimierer zu dieser speziellen Lösung kommt.

1.1 Motivation und Ziele der Arbeit

Wie in der Einleitung schon angesprochen, ist es schwierig eine Lösung eines nichtlinearen Optimierungsproblems unter Nebenbedingungen (kurz NLP genannt) zu verstehen. Doch genau dieses Verständnis ist für die Formulierung von Optimierungsproblemen wichtig. Die korrekte Formulierung des Problems, also das Spezifizieren von Kostenfunktionen und Nebenbedingungen, ist essenziell für das Erreichen des gewünschten Ergebnisses. So kann die falsche Formulierung eines Optimierungsproblems für eine Roboter-Trajektorie zu unerwünschtem Verhalten des Roboters in der Lösung des Optimierers führen. Das Erkennen von unerwünschtem Verhalten in der Lösung in Form eines Videos, in dem der Roboter seine Bewegungsbahn abfährt ist recht einfach zu erkennen. Wieso der Optimierer aber zu dieser Lösung kommt ist häufg schwierig zu verstehen. Schon die schlechte Spezifizierung von nur einer Nebenbedingung reicht aus, dass es zu unerwünschtem Verhalten des Roboters kommen kann. Da sich die Constraints gegenseitig beeinflussen (siehe 2.4.3), ist es besonders schwierig genau den einen Constraint zu identifizieren, der für das unerwünschte Verhalten verantwortlich ist.

Da nichtlineare Optimierungsprobleme mit Nebenbedingungen (im Folgenden zur Einfachheit nur noch Optimierungsprobleme mit NB genannt) ein sehr großes Anwendungsfeld finden, ist auch die Entwicklung und Verbesserung von Tools und Algorithmen zum Lösen solcher Probleme sehr nachgefragt. Es gibt viele öffentlich verfügbare Tools/Algorithmen, die entwickelt wurden um Roboter-Trajektorien zu optimieren. TrajOpt [2] oder CHOMP [15] sind bekannte Beispiele hierfür. Die meisten Optimierer/Tools bieten neben

dem Optimieren nicht nur die Endlösung an, sondern können auch noch weitere Informationen anzeigen. So kann man sich beispielsweise bei TrajOpt nach jeder Iteration des Optimierers die aktuellen Kosten und Strafen anzeigen lassen und so den Optimierungsprozess verfolgen. Was jedoch noch keiner der Optimierer anbietet, ist eine Erklärung, warum die gefundene Lösung so aussieht, wie sie aussieht. Um dieser Frage auf den Grund zu gehen, wird in dieser Arbeit ein Tool vorgestellt, welches das Untersuchen von Kosten und Strafen einer Lösung gewährleistet. Da dieses Tool das erste seiner Art ist, wird mit ihm grundlegend erforscht, was man für wichtige und interessante Informationenen aus der gegebenen Lösung und den dazugehörigen Kosten/Strafen extrahieren kann.

1.2 Vorgehensweise und Aufbau der Arbeit

Da das Verstehen der Theorie ein wichtiger Bestandteil für die Untersuchung von NLP-Lösungen ist, findet sich in dieser Arbeit relativ viel Theorie wieder. Der genaue Aufbau der Arbeit sieht wie folgt aus.

Zuerst wird ein Blick auf die Grundlagen der nicht restringierten Optimierung geworfen. Hier wird erklärt, welche Bedingungen an einem Optimum erfüllt sein müssen und wie man anhand dieser Bedingungen Optima identifizieren kann.
Im nächsten Teil werden verschiedene numerische Verfahren vorgestellt, die in vielen Optimierungsprogrammen Anwendung finden. Diese können für alle Optimierungsprobleme genutzt werden.
Im dritten Teil werden die Bedingungen für Optimalpunkte bei Problemen mit NB präsentiert (u.a. die KKT-Bedingungen). Sie sind Erweiterungen der Optimalitätskriterien von Problemen ohne Nebenbedingungen. Zusätzlich wird die Lagrange Funktion und die Lagrange Dualität eingeführt und ihre Vorteile werden aufgezeigt. Da das Verstehen der in diesem Teil vorgestellten Formeln ein essentieller Bestandteil für das Lösen von Optimierungsproblemen mit NB darstellt, werden in diesem Teil häufig Beispiele zur Hilfestellung aufgeführt.
Im 4. Abschnitt werden neben der im zuvorigen Abschnitt vorgestellten Lagrange Methode noch drei weitere Methoden zum Lösen von Optimierungsproblemen mit NB vorgestellt. Auch ihre Unterschiede und Ähnlichkeiten werden in diesem Abschnitt disskutiert.
Im letzten Theorieabschnitt wird die Theorie der Sensitivitätsanalyse beschrieben. Mit der Sensitivitätsanalyse kann herausgefunden werden, wie sich eine Lösung ändern würde, wenn sich die Problemstellung minimal ändern würde.
Nachdem die Theorie erklärt wurde, wird das entwickelte Debuggintool zum Untersuchen von Lösungen solcher Optimierungsprobleme vorgestellt. Neben der Beschreibung des Aufbaus wird in einem weiteren Abschnitt die Funktionsweise und Effektivität des Tools anhand eines Beispiels präsentiert.
Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick im Bezug auf die

weitere Forschung ab.

2 Grundlagen der Optimierungstheorie

2.1 Optimierung ohne Nebenbedingungen

Ein Optimierungsproblem ohne Nebenbedingungen kann wie folgt beschrieben werden:

$$\min_{x \in \mathbb{R}^n} f(x) \qquad f : \mathbb{R}^n \to \mathbb{R} \tag{1}$$

In dieser Arbeit wird das Optimierungsproblem immer als Minimierungsproblem ($\min_x f(x)$) formuliert, da ein Maximierungsproblem ($\max_x f(x)$) immer in ein Minimierungsproblem umgeschrieben werden kann [3]:

$$\max_x f(x) = -\min_x [-f(x)]$$

2.1.1 Notwendige Bedingung erster Ordnung

Die notwendige Bedingung erster Ordnung für eine Minimum x^* ist:

$$\nabla f(x^*) = 0 \tag{2}$$

Wenn $\nabla f(x^*) \neq 0$ gilt, gibt es mindestens eine Richtung $d \in \mathbb{R}^n$, in die von x^* aus zu einem Punkt gelaufen werden kann, an dem $f(x^* + \epsilon d) < f(x^*)$ mit $\epsilon > 0$ gilt. Diese Behauptung ist wahr, da der Gradient immer die Richtung angibt, in der eiene Funktion am schnellsten steigt. Somit muss für eine Richtung d nur folgendes erfüllt sein, damit von x^* aus ein Punkt erreicht werden kann, der tiefer liegt als $f(x^*)$: $\nabla f(x^*)^T d < 0$

2.1.2 Notwendige Bedingung zweiter Ordnung

Neben der notwendigen Bedingung erster Ordnung gibt es noch eine zweite notwendige Bedingung für eine Minimum x^*. Diese ist die sogenannte notwendige Bedingung zweiter Ordnung:

$$d^T \nabla^2 f(x^*) d \geq 0 \qquad \forall d \in \mathbb{R}^n \tag{3}$$

Das heißt, die Hesse-Matrix von f am Punkt x^* muss positiv semidefinit sein.

2.1.3 Hinreichende Bedingung zweiter Ordnung

Die hinreichende Bedingung zweiter Ordnung ähnelt der notwendige Bedingung zweiter Ordnung und ist wie folgt definiert:

$$d^T \nabla^2 f(x^*) d > 0 \qquad \forall d \in \mathbb{R}^n \setminus \{0\} \tag{4}$$

Das bedeutet, dass die Hesse-Matrix von f am Punkt x^* positiv definit sein muss. Die Bedingungen zweiter Ordnung können über die Taylor–Formel bewiesen werden (siehe [4]).

Schaut man sich die Funktion $f(x) = x_1^2 + x_2^2$, so sieht man, dass die notwendigen und hinreichenden Bedingungen erster und zweiter Ordnung ((2), (3), (4)) an der Stelle $x^* = (0, 0)$ erfüllt sind:

$$\nabla f(x) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} \qquad \nabla f(x^*) = 0$$

$$\nabla^2 f(x) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad \nabla^2 f(x^*) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \text{ ist positiv (semi-)definit}$$

Das heißt, x^* ist ein lokales Minimum und kommt somit als globales Minimum von $f(x)$ und damit auch als Lösung des Minimierungsproblems in Frage. Wie man dem Plot der Funktion $f(x)$ entnehmen kann (s. Abb. 1), ist an der Stelle $x^* = (0, 0)$ tatsächlich das globale Minimum der Funktion $f(x)$. Folglich ist x^* das am besten minimierende x des Optimierungsproblems und somit ist die Lösung des Optimierungsproblems $f(x^*) = 0$.

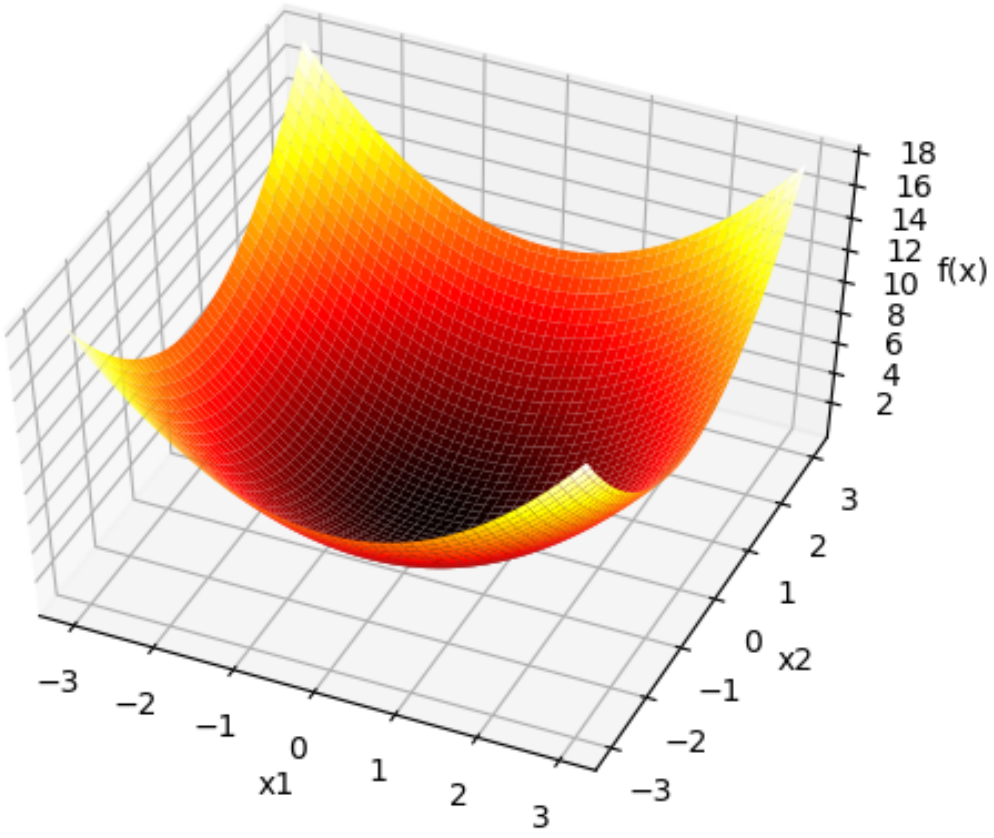

Abbildung 1: Teilausschnitt der Funktion $f(x) = x_1^2 + x_2^2$

Je dunkler der Rotton desto niedriger der Wert von $f(x)$; je heller der Gelbton desto höher ist er.

2.2 Numerische Verfahren

Numerische Verfahren sind Verfahren die für die näherungsweise Berechnung von Lösungen mit Hilfe von Computern entwickelt wurden. Sie helfen dabei Probleme zu simplifizieren und effizient Lösungen zu finden. Im Folgenden werden zwei Verfahren vorgestellt die bei der Berechnung von Lösungen von Optimierungsproblemen mit NB häufig verwendet werden. Es gibt noch viele weitere Verfahren und Abwandlung der hier vorgestellten Verfahren, aber diese werden in dieser Arbeit nicht weiter thematisiert.

2.2.1 Newton-Verfahren

Das Newton-Verfahren ist ein numerisches Verfahren zur Nullstellenbestimmung. Iterativ wird von einem Startwert x_0 aus der Wert von x_i (i ist am Anfang 0) immer weiter angepasst, sodass er letztendlich dem Wert der Nullstelle entspricht. Bei jeder Iteration wird die Tangente am Punkt $(x_i|f(x_i))$ berechnet und die Nullstelle dieser Tangente als neues x_{i+1} verwendet. Für die Tangentengleichung im i-ten Iterationsschritt gilt:

$$T(x) = f(x_i) + f'(x_i) * (x - x_i)$$

Für das x des nächsten Iterationsschritts ($= x_{i+1}$) wird nun die Nullstelle von $T(x_{i+1})$ gesucht, d.h.

$$T(x_{i+1}) = f(x_i) + f'(x_i) * (x_{i+1} - x_i) = 0$$

und nach x_{i+1} aufgelöst:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Als Algorithmus sieht das Newton-Verfahren folgendermaßen aus:

for i = 0 ,1 ,2 ,...:
$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

Für mehrdimensionale Funktionen funktioniert das Verfahren genau gleich, der Iterationsschritt in der Schlife sieht dann so aus:

$$x_{i+1} = x_i - [\nabla f(x_i)]^{-1} f(x_i)$$

Sollte die Funktion $f(x)$ nicht stetig differenzierbar oder die Jacobi-Matrix ($\nabla f(x_i)$ für $f \in \mathbb{R}^n \to \mathbb{R}^m$) nicht invertierbar sein, funktioniert das Newton Verfahren nicht mehr. Dafür gibt es Erweiterungen wie das Quasi-Newton-Verfahren oder das Gauss-Newton-Verfahren, mit welchen trotz dieser Einschränkungen die Nullstellen (näherungsweise) bestimmen werden können.

Das Newton-Verfahren kann auch zum Bestimmen einer Extremstelle verwendet werden [17]. Hierfür wird die Tangente der ersten Ableitung genutzt:

$$T(x_{i+1}) = f'(x_i) + f''(x_i) * (x_{i+1} - x_i) = 0$$

Nach x_{i+1} aufgelöst ergibt sich:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

bzw.

$$x_{i+1} = x_i - \nabla^2 f(x_i)^{-1} \nabla f(x_i)$$

mit der Hessematrix $\nabla^2 f(x_i)$ von f.

2.2.2 Backtracking Line Search

Backtracking Line Search ist ein iteratives Verfahren zur Tiefpunktbestimmung einer Funktion. Dabei wird von einem Startwert x und einer Suchrichtung d (beispielsweise der Gradient am Punkt x) aus diejenige Schrittgröße gesucht, für die in Richtung d eine hinreichend großer oder der größte Abstieg erfolgt. Mathematisch ausgedrückt wird also das Problem

$$\min_{\alpha \geq 0} f(x + \alpha * d)$$

gelöst [17]. Hierbei wird mit einem großen Wert für α angefangen, welcher immer weiter dekrementiert wird, bis eine Abbruchbedingung gilt. Eine Abbruchbedingung, die auf dem Gradienten der Funktion basiert, ist

$$f(x) + f(x + \alpha * d) \geq -\alpha * \nabla f(x)^T d * c \text{ mit } c \in (0, 1), d \text{ Eigenvektor},$$

bekannt unter dem Namen Armijo–Goldstein Bedingung [6]. Der Algorithmus sieht dann wie folgt aus:

$$\begin{aligned} &\text{while not } f(x) + f(x + \alpha * d) \geq -\alpha * \nabla f(x)^T d * c: \\ &\quad \alpha = \alpha * t \end{aligned}$$

mit dem Dekrementierungsparameter $t \in (0, 1)$.
Es gibt auch die Variante, bei der mit einem sehr kleinen Wert für α angefangen wird und diesen iterativ inkrementiert wird. Diese Variante wird Line Search genannt.

2.3 Lösungsverfahren von Optimierungsproblemen mit Nebenbedingungen

Allgemein kann ein Optimierungsproblem unter Nebenbedingungen wie folgt beschrieben werden:

$$\min_x f(x) \quad \text{unter den Bedingungen} \quad g(x) \leq 0, \quad h(x) = 0 \tag{5}$$

$$x \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}, \quad g : \mathbb{R}^n \to \mathbb{R}^m, \quad h : \mathbb{R}^n \to \mathbb{R}^l$$

$$g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{pmatrix}, \quad \forall j \in \{1, 2, ..., m\} : \quad g_j : \mathbb{R}^n \to \mathbb{R}$$

$$h(x) = \begin{pmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_l(x) \end{pmatrix}, \quad \forall k \in \{1, 2, ..., l\} : \quad h_k : \mathbb{R}^n \to \mathbb{R}$$

Wobei $f(x)$ die Kostenfunktion ist, die minimiert weden soll, $g_j(x)$ und $h_k(x)$ sind die Nebenbedingungen. Nebenbedingungen können in zwei unterschiedliche Kategorien eingeteilt werden - gilt $g_j(x) \leq 0$ so spricht man von einem Inequality Constraint (zur Verdeutlichung mit g dargestellt) und gilt $h_k(x) = 0$ so spricht man von einem Equality Constraint (mit h dargestellt). Das Optimierungsproblem ist lösbar falls gilt:

$$\exists \overline{x} \in \mathbb{R}^n : g_j(\overline{x}) \leq 0 \wedge h_k(\overline{x}) = 0, \quad \forall j \in \{1, 2, ..., m\}, \forall k \in \{1, 2, ..., l\}$$

Es muss also ein Punkt $\overline{x}$ existieren, der alle NB erfüllt (häufig wird $\overline{x}$ dann als feasible bezeichnet). Die Menge aller dieser Punkte, bildet die zulässige Menge $\mathcal{Z}$.

$$\mathcal{Z} = \{\overline{x} \in \mathbb{R}^n : g(\overline{x}) \leq 0 \wedge h(\overline{x}) = 0\} \tag{6}$$

Das Optimierungsproblem (5) kann dann umgeschrieben werden zu:

$$\min_{\overline{x} \in \mathcal{Z}} f(\overline{x}) \tag{7}$$

Schaut man sich erneut das obige Beispiel mit der Funktion $f(x) = x_1^2 + x_2^2$ an und erweitert es um die Nebenbedingung $g(x) = x_1 + 1 \leq 0$, so sieht man direkt, dass das zuvorige Minimum bei $x^* = (0, 0)$ nicht mehr zulässig ist, also $x^* \notin \mathcal{Z}$. Somit ist bei x^* nicht (mehr) die Lösung des Optimierungsproblems mit NB.

Wie man also sieht, ist eine Anpassung der notwendigen und hinreichenden Bedin-

gungen erster und zweiter Ordnung notwendig, wenn das Optimierungsproblem mit NB gelöst werden soll. Diese neuen/angepassten Bedingungen dürfen am Punkt $(x^*, f(x^*))$ nicht erfüllt sein. Wie man später sieht, wird dies auch der Fall sein. Doch bevor man einen Blick in die angepassten Bedingungen wirft, ist es hilfreich die Lagrange Funktion zu kennen.

2.3.1 Lagrange Funktion und Lagrange Dualität

Zu einem Optimierungsproblems unter NB wie in (5) berschrieben, kann die dazugehörige Lagrange Funktion wie folgt definiert werden [13]:

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \lambda^T * g(x) + \nu^T * h(x), \qquad \lambda \in \mathbb{R}^m,\ \nu \in \mathbb{R}^l \tag{8}$$

Ebenfalls kann die duale Funktion $q(\lambda, \nu)$ für das Optimierungsproblem:

$$q(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) \tag{9}$$

und damit auch das duale Problem definiert werden:

$$\max_{\lambda,\nu} q(\lambda, \nu) \quad \text{unter der Bedingung} \quad \lambda \geq 0 \tag{10}$$

Eine besondere Eigenschaft des dualen Problems ist, dass das es immer konvex ist (d.h. q ist immer konkav), auch wenn das primale Problem nicht konvex ist (5). Für zulässige $\overline{x}, \overline{\lambda}, \overline{\nu}$ wird $f(\overline{x}) - q(\overline{\lambda}, \overline{\nu})$ als Dualitätslücke des Punktes $(\overline{x}, \overline{\lambda}, \overline{\nu})$ bezeichnet. Man spricht von schwacher Dualität falls $f(x^*) - q(\lambda^*, \nu^*) \geq 0$ gilt und von starker Dualität falls $f(x^*) - q(\lambda^*, \nu^*) = 0$ gilt, wobei $f(x^*)$ bzw. $q(\lambda^*, \nu^*)$ die Werte der Optimallösungen der Probleme sind. Gilt starke Dualität, ist das Optimum des primalen Problems $f(x^*)$ und das Optimum des dualen Problems $q(\lambda^*, \nu^*)$ gleich, was bedeutet, dass man durch das Lösen des dualen Problems (konvexes Problem) auch das primale Problem löst. Gilt nur schwache Dualität ist das Lösen des dualen Problems dennoch interessant, da die Lösung des dualen Problems eine untere Schranke für das primale Problem angibt. Das Besondere an konvexen Problemen ist, dass jedes lokale Minimum auch ein globales Minimum ist und somit eine Lösung des (dualen) Optimierungsproblems. Es wäre also schön, eine Dualitätslücke von 0 zu haben.

2.3.2 Karush-Kuhn-Tucker-Bedingungen

Die Karush-Kuhn-Tucker-Bedingungen (kurz KKT-Bedingungen) sind **notwendige Optimalitätskriterien erster Ordnung** für eine Lösung eines Optimierungsproblems unter NB, vorausgesetzt das Optimierungsproblem erfüllt gewisse Regularitätsbedingungen. Für einen zulässigen Punkt $\overline{x} \in \mathcal{Z}$ kann man nach William Karush, Harold W. Kuhn und

Albert W. Tucker folgende Aussage treffen [12]:
Vorrausgesetzt

- der Punkt $\overline{x}$ ist ein lokales Minimum,
- f, g_j und h_k sind alle stetig differenzierbar,
- am Punkt $\overline{x}$ gelten gewisse Regularitätsbedingungen,

gilt:
Es existieren Vektoren $\overline{\lambda} \in \mathbb{R}^m$ und $\overline{\nu} \in \mathbb{R}^l$, so dass folgende Bedingungen gelten:

$$\nabla_x \mathcal{L}(\overline{x}, \overline{\lambda}, \overline{\nu}) = \nabla f(\overline{x}) + \overline{\lambda}^T * \nabla g(\overline{x}) + \overline{\nu}^T * \nabla h(\overline{x}) = 0 \tag{11}$$

$$g_j(\overline{x}) \leq 0, \qquad \forall j \in \{1, 2, ..., m\} \tag{12}$$

$$h_k(\overline{x}) = 0, \qquad \forall k \in \{1, 2, , ..., l\} \tag{13}$$

$$\overline{\lambda}_j \geq 0, \qquad \forall j \in \{1, 2, ..., m\} \tag{14}$$

$$\overline{\lambda}_j * g_j(\overline{x}) = 0, \qquad \forall j \in \{1, 2, ..., m\} \tag{15}$$

Einen solchen Punkt $(\overline{x}, \overline{\lambda}, \overline{\nu})$, für den die eben genannte Bedingungen gelten, nennt man auch KKT-Punkt, $\overline{\lambda}$ und $\overline{\nu}$ KKT-Multiplikatoren. Die Gleichungen (11) bis (15) werden KKT-Bedingungen genannt.
Die Gleichung (11) kann umgeschrieben werden zu:

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

D.h. am Optimum muss ein Gleichgewicht zwischen dem Kostengradienten und den Gradienten der aktiven NB vorliegen. Eine NB g_j bzw. h_k wird aktiv bei $\overline{x}$ genannt, wenn $g_j(\overline{x}) = 0$ bzw. $h_k(\overline{x}) = 0$.

Schaut man sich ein Beispiel an, wird diese Bedingung deutlich. Sei das Beispiel erneut mit der Funktion $f(x) = x_1^2 + x_2^2$ und der NB $g(x) = x_1 + 1 \leq 0$. Wie bereits bekannt, ist $x^* = (0, 0)$ nicht die Lösung des Optimierungsproblems, obwohl $\nabla f(x^*) = 0$ gilt. Schaut man sich die Stelle $x' = (-1, 0)$ (lokales Minimum für $x_1 \leq -1$) und den Gradienten der Lagrange Funktion an, so sieht man, dass die Gleichung (11) an dieser Stelle (für $\lambda' = 2 \geq 0$) eingehalten wird:

$$\nabla_x \mathcal{L}(x, \lambda, \nu) = \nabla f(x) + \lambda^T * \nabla g(x) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} + \lambda * \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\nabla_x \mathcal{L}(x', \lambda') = \begin{pmatrix} -2 \\ 0 \end{pmatrix} + 2 * \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Betrachtet man die Höhenlinie (alle Punkte einer Funktion, die den selben Funktionswert

haben) zu $g(x) = 0$ (also alle Punkte (x, g(x) = 0)), welche durch folgende Geradengleichung definiert werden kann,

$$w(x) = x' + x * \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

erkennt man sofort, dass der Gradient von g bei x' orthogonal auf w steht. Intuitiv macht dies auch Sinn, denn auf einer Höhenlinie liegen alle Punkte die für $g(x)$ den gleichen Wert haben. Will man nun so steil wie möglich von einem Punkt auf der Höhenlinie aus zu einem Punkt mit niedrigeren Kosten gehen, ist es am sinnvollsten, dies in eine orthogonale Richtung zur Höhenlinie zu machen. Da wenn man mit der Höhenlinie läuft (auch wenn es nur ein kleiner Schritt ist), man kein Gefälle hat. Würde man also von x' aus in die Richtung $\begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$ laufen, so würde man einen $\frac{\sqrt{2}}{2}$-langen Schritt orthogonal zu w, aber auch einen $\frac{\sqrt{2}}{2}$-langen Schritt auf w machen. Somit wäre diese Richtung nicht die steilste Richtung. Da der Gradient an einem Punkt immer in die Richtung zeigt, in der sich der Funktionswert am meisten erhöht (= steilste Richtung), steht der Gradient immer orthogonal auf der Höhenlinie. Für alle aktiven Constraints hat man automatisch eine feste (= 0) Höhenlinie $g(x) = 0$ bzw. $h(x) = 0$. Schaut man sich nun alle Höhenlinien zu $f(x) = a$, $a \in \mathbb{R}$ an, kann das Minimierungsproblem umformuliert werden:
Gesucht ist die Höhenlinie mit dem kleinsten Wert für a von $f(x)$, die sich mit allen Höhenlinien der aktiven Constraints im Punkt x^* im zulässigen Bereich trifft. In Abb. 2 ist dies genau bei $x' = (-1, 0)$ der Fall. Da für alle Höhenlinien gilt, dass die Gradienten orthogonal auf ihnen stehen, müssen in diesem Punkt alle Gradienten der aktiven Nebenbedingungen parallel zueinander sein. Diese Parallelität wird durch die Gleichung (11) beschrieben.

Auch die anderen KKT-Bedingungen ((12) - (15)) werden im Punkt x' eingehalten, wodurch das Minimum bestätigt wird. Dies kann auch im Plot gezeigt werden.

Die beiden Gleichungen (12) und (13) fordern die Zulässigkeit von $\overline{x}$ für das primale Problem und sind vergleichbar mit der zulässigen Menge, wie sie in (6) definiert ist. Die Menge aller zulässigen Punkte ist meist größer als die Menge aller KKT-Punkte, da nich an jedem zulässigen Punkt die KKT-Bedingungen eingehalten werden. Formel (14) fordert die Zulässigkeit von $\overline{\lambda}$ für das duale Problem. Die letzte Zeile (15) stellt die Komplementaritätsbedingung dar. Sie stellt die Verbindung der Optimalpunkte x^* und (λ^*, ν^*) des primalen und dualen Problems her. Aus der Komplementaritätsbedingung kann gefolgert werden, dass $\overline{\lambda}_j = 0$ gelten muss, falls $g_j(\overline{x}) < 0$ und $\overline{\lambda}_j \geq 0$, falls $g_j(\overline{x}) = 0$

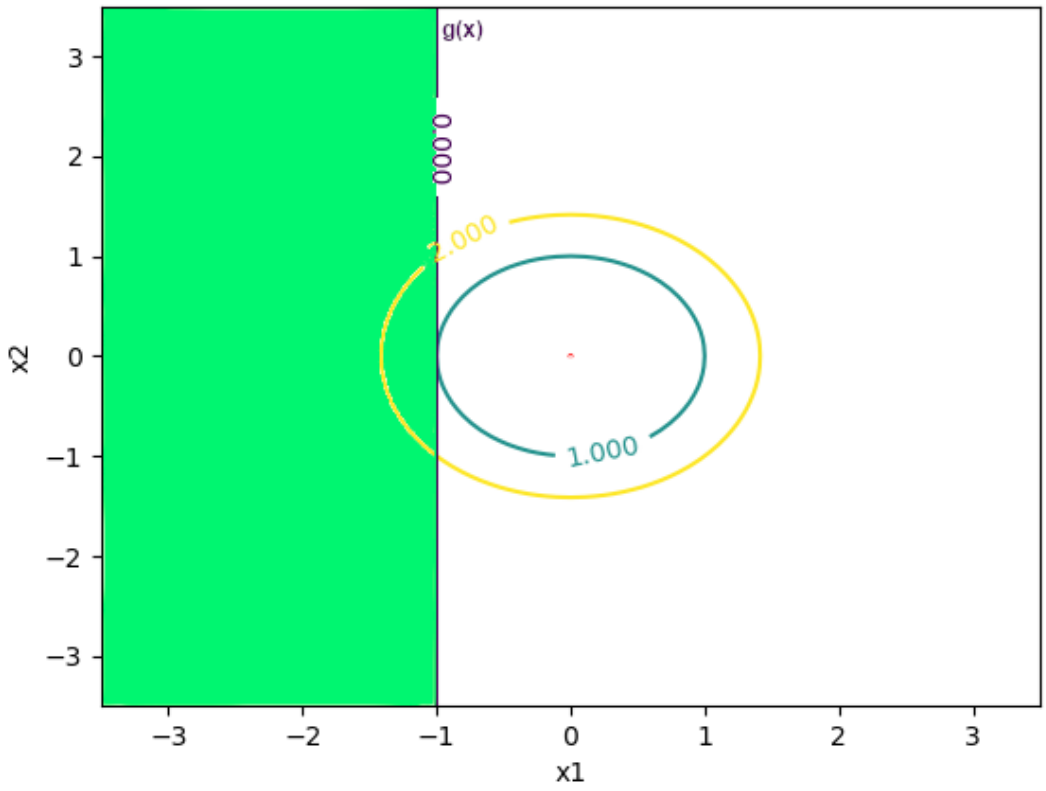

Abbildung 2: Plot von Konturlinien

Der Plot zeigt die Konturlinien $g(x) = x + 1 = 0$ (lila), $f(x) = x_1^2 + x_2^2 = 0$ (rot), $f(x) = 1$ (blau-grün) und $f(x) = 2$ (gelb). Der grüne Bereich ist der (zulässige) Bereich in dem $g(x) \leq 0$ gilt.

(g_j aktiv). Am Optimalpunkt (x^*, λ^*, ν^*) gilt für die Lagrange Funktion $\mathcal{L}$:

$$\mathcal{L}(x^*, \lambda^*, \nu^*) = f(x^*) + \underbrace{\lambda^{*\,T} * g(x^*)}_{=0,\text{ da (12),(15)}} + \underbrace{\nu^{*\,T} * h(x^*)}_{=0,\text{ da (13)}} = f(x^*) \tag{16}$$

Gilt starke Dualität, kann gezeigt werden, dass an einer Lösung $\overline{x}$ und $\overline{\lambda}, \overline{\nu}$ der beiden Probleme (primales und duales Problem) $(\overline{x}, \overline{\lambda}, \overline{\nu})$ immer ein KKT-Punkt ist [12]. Umgekehrt kann gezeigt werden, dass für einen KKT-Punkt $(\overline{x}, \overline{\lambda}, \overline{\nu})$, $\overline{x}$ und $\overline{\lambda}, \overline{\nu}$ Lösungen der jeweiligen Probleme sind [12]. Gilt also starke Dualität sind die Aussagen $(\overline{x}, \overline{\lambda}, \overline{\nu})$ ist ein KKT-Punkt und $\overline{x}$ und $\overline{\lambda}, \overline{\nu}$ sind Lösungen der Probleme äquivalent. Sollte also eine Bedingung die starke Dualität garantieren, so kann diese Bedingung als Regularitätsbedingung für die Verwendung der KKT-Bedingungen verwendet werden.

Schaut man sich $\underset{x}{min}\ \mathcal{L}$, $\underset{\lambda \geq 0}{max}\ \mathcal{L}$ und $\underset{\nu}{max}\ \mathcal{L}$ an, kann gezeigt werden, dass alle drei Min-/Maximierungen zusammen die KKT-Bedingungen erzwingen [17]:

$$\begin{aligned}
&\underset{x}{min}\ \mathcal{L} && \Longrightarrow \nabla_x \mathcal{L}(x, \lambda, \nu) = 0 \\
&\underset{\lambda \geq 0}{max}\ \mathcal{L} && \Longrightarrow \underset{\lambda \geq 0}{max}\ \mathcal{L}(x, \lambda, \nu) = \begin{cases} f(x) + \nu^T * h(x) & \text{if } g(x) \leq 0 \\ \infty & \text{else} \end{cases} \text{ da} \\
& && \quad \underset{\lambda \geq 0}{argmax}\ \mathcal{L}(x, \lambda, \nu) = \begin{cases} \lambda_j = 0 & \text{if } g_j(x) < 0 \\ \lambda_j \geq 0 & \text{else, } \nabla_{\lambda_j} \mathcal{L}(x, \lambda, \nu) = g_j(x) = 0 \end{cases} \\
& && \Longrightarrow (\lambda_j = 0 \wedge g_j(x) < 0) \vee g_j(x) = 0 \\
& && \Longrightarrow g(x) \leq 0 \text{ und } \lambda * g(x) = 0 \\
&\underset{\nu}{max}\ \mathcal{L} && \Longrightarrow \nabla_\nu \mathcal{L}(x, \lambda, \nu) = h(x) = 0
\end{aligned}$$

Wegen dieser Eigenschaften kann ein Optimierungsproblem unter NB auch mit der dazugehörigen Lagrange Funktion (8) beschrieben werden. Ähnlich wie in Gleichung (6), kann eine Menge mit möglichen Punkten $\overline{y} = (\overline{x}, \overline{\lambda}, \overline{\nu})$, die für das Optimum x^* in Frage kommen, definiert werden.

$$\mathcal{Z}_\mathcal{L} = \{\overline{y} \in \mathbb{R}^{n+m+l} : \mathcal{L}(\overline{y}) = \underset{x}{min}\ \mathcal{L}(x, \overline{\lambda}, \overline{\nu}) = \underset{\lambda \geq 0}{max}\ \mathcal{L}(\overline{x}, \lambda, \overline{\nu}) = \underset{\nu}{max}\ \mathcal{L}(\overline{x}, \overline{\lambda}, \nu)\}$$

Dabei gilt $\mathcal{Z}_\mathcal{L} \subseteq \mathcal{Z}$, da $\mathcal{Z}_\mathcal{L}$ nur KKT-Punkte enthält und $\mathcal{Z}$ alle zulässigen Punkte enthält. Das Optimum $f^* = f(x^*)$ ist dann ähnlich wie in (7) definiert als

$$f^* = \underset{\overline{y} \in \mathcal{Z}_\mathcal{L}}{min}\ \mathcal{L}(\overline{x}, \overline{\lambda}, \overline{\nu}) \underset{(16)}{=} \underset{(\overline{x}, \overline{\lambda}, \overline{\nu}) \in \mathcal{Z}_\mathcal{L}}{min}\ f(\overline{x})$$

2.3.3 Fritz-John-Bedingungen

Die Fritz-John-Bedingungen (kurz FJ-Bedingungen) sind ebenfalls **notwendige Optimalitätskriterien erster Ordnung** für eine Lösung eines Optimierungsproblems unter NB, ohne dass das Optimierungsproblem gewisse Regularitätsbedingungen erfüllen muss. Die FJ-Bedingungen sind eine Verallgemeinerung der KKT-Bedingungen und besagen [7]: Vorrausgesetzt

- der Punkt $\overline{x}$ ist ein lokales Minimum,
- f, g_j und h_k sind alle stetig differenzierbar

gilt:
Es existieren Vektoren $\overline{\lambda} \in \mathbb{R}^m, \overline{\nu} \in \mathbb{R}^l$ und eine Zahl $\overline{z} \in \mathbb{R}$, so dass folgende Bedingungen gelten:

$$\begin{aligned}
\overline{z} * \nabla f(\overline{x}) + \overline{\lambda}^T * \nabla g(\overline{x}) + \overline{\nu}^T * \nabla h(\overline{x}) &= 0 \\
g_j(\overline{x}) &\leq 0, \qquad \forall j \in \{1, 2, ..., m\} \\
h_k(\overline{x}) &= 0, \qquad \forall k \in \{1, 2, , ..., l\} \\
\overline{\lambda}_j &\geq 0, \qquad \forall j \in \{1, 2, ..., m\} \\
\overline{\lambda}_j * g_j(\overline{x}) &= 0, \qquad \forall j \in \{1, 2, ..., m\} \\
\overline{z} &\geq 0
\end{aligned}$$

Einen Punkt $(\overline{z}, \overline{x}, \overline{\lambda}, \overline{\nu}) \in \mathbb{R}^{1+n+m+l}$, für den die eben genannten Bedingungen gelten, heißt FJ-Punkt. Für einen FJ-Punkt mit $\overline{z} = 1$, also $(1, \overline{x}, \overline{\lambda}, \overline{\nu})$ ist $(\overline{x}, \overline{\lambda}, \overline{\nu})$ ein KKT-Punkt, da für $\overline{z} = 1$ die KKT-Bedingungen gleich den FJ-Bedingungen sind. Eine weitere Beobachtung ist, dass es unendlich viele FJ-Punkte gilt, falls $\overline{z} > 0$ gibt. Sei $(\overline{z}, \overline{x}, \overline{\lambda}, \overline{\nu})$ mit $\overline{z} > 0$ ein FJ-Punkt, dann ist auch $(\alpha * \overline{z}, \overline{x}, \alpha * \overline{\lambda}, \alpha * \overline{\nu})$ mit $\alpha > 0$ ein FJ-Punkt. Und für $\alpha * \overline{z} = 1$ ist der FJ-Punkt auch ein KKT-Punkt [7]. Über diese Beobachtung ergibt sich eine weitere Möglichkeit eine Regularitätsbedingung aufzustellen. Garantiert eine Bedingung an einem FJ-Punkt, dass $\overline{z} > 0$ gilt, kann diese Bedingung als Regularitätsbedingung genutzt werden.

2.3.4 Notwendige Bedingung zweiter Ordnung

Neben den notwendigen Bedingungen erster Ordnung (KKT/FJ-Bedingungen) gibt es, wie auch schon bei der Optimierung ohne NB, die notwendige Bedingung zweiter Ordnung. Wie bei den KKT-Bedingungen müssen einige Voraussetzungen gelten:

- Der Punkt $\overline{x}$ ist ein lokales Minimum,
- f, g_j und h_k sind alle zweimal stetig differenzierbar,

- am Punkt $\overline{x}$ gelten gewisse Regularitätsbedingungen,
- der Punkt $(\overline{x}, \overline{\lambda}, \overline{\nu})$ ist ein KKT-Punkt,

dann gilt:

$$d^T \nabla^2_{xx} L(\overline{x}, \overline{\lambda}, \overline{\nu}) d \geq 0 \qquad \forall d \in \mathcal{T}(\overline{x})$$

($\nabla^2_{xx} L(\overline{x}, \overline{\lambda}, \overline{\nu})$ muss positiv semidefinit auf $\mathcal{T}(\overline{x})$ sein) [9], [3]
Wobei $\mathcal{T}(\overline{x})$ der kritische Kegel für $\overline{x}$ ist, der wie folgt definiert ist:

$$\begin{aligned} \mathcal{T}(\overline{x}) = \{d \in \mathbb{R}^n |\ & \nabla g_j(\overline{x})^T d \leq 0, g_j(\overline{x}) = 0, \lambda_j = 0 \\ & \nabla g_j(\overline{x})^T d = 0, g_j(\overline{x}) = 0, \lambda_j > 0 \\ & \nabla h_k(\overline{x})^T d = 0, \forall k\} \end{aligned}$$

$\mathcal{T}(\overline{x})$ enthält die tangentialen Richtungen d von $\overline{x}$ aus, die in/an den zulässigen Bereich zeigen. Es existiert also ein $\epsilon > 0$ für jedes $d \in \mathcal{T}(\overline{x})$, für das $\overline{x} + \epsilon * d \in \mathcal{Z}$ gilt. Der Grund, wieso nur die Richtungen in den zulässigen Bereich untersucht werden müssen und nicht wie bei (3) in alle Richtungen, sind die NB. Ähnlich wie mit der zweiten Ableitung bei eindimensionalen Funktionen die Krümmung von Funktionen bestimmt werden kann, verrät die Definitheit der Hessematrix die Krümmung der Funktion in die untersuchten Richtungen (hier aller Richtungen die in den zulässigen Bereich zeigen). Krümmt sich für eine Richtung d' von x aus die Funktion nach unten (negative Krümmung), so kann an der Stelle x kein Minimum sein, da man mit einem Schritt in Richtung d' einen kleineren Funktionswert erreichen würde. Da aber ein Schritt in den unzulässigen Bereich nicht von Interesse für die Suche des Minimums ist, muss das Krümmungsverhalten nur in die Richtungen, die in den zulässigen Bereich zeigen, nicht negativ sein.

2.3.5 Hinreichende Bedingung zweiter Ordnung

Gelten die notwendigen Bedingungen erster und zweiter Ordnung an einem Punkt, so kann an diesm Punkt entweder ein Sattelpunkt oder ein lokaler Tiefpunkt/Minimum vorliegen. Um zu unterscheiden, welcher dieser beiden Punkte vorliegt, wird die hinreichende Bedingung zweiter Ordnung benötigt, welche besagt:
Gilt

$$d^T \nabla^2_{xx} L(x^*, \lambda^*, \nu^*) d > 0 \qquad \forall d \in \mathcal{T}(x^*) \setminus \{0\}$$

($\nabla^2_{xx} L(x^*, \lambda^*, \nu^*)$ ist positiv definit auf $\mathcal{T}(x^*)$) [3]
und es gelten die notwendigen Bedingungen zweiter Ordnung mit dessen Voraussetzungen,

so existiert eine Umgebung U von x^*, für die gilt:

$$f(x^*) \leq f(x) \qquad \forall x \in \mathcal{U}$$

Oder einfacher formuliert: Gilt eben Genanntes an einem Punkt x^*, so ist dieser Punkt ein lokales Minimum (also kein Sattelpunkt).

Zusammenfassend kann man also sagen,

- $\exists \lambda, \nu:$ (x, λ, ν) ist ein KKT-Punkt,
- bei x gelten gewisse Regularitätsbedingungen,
- $d^T \nabla^2_{xx} L(x^*, \lambda^*, \nu^*) d > 0 \qquad \forall d \in \mathcal{T}(x^*) \setminus \{0\}$,

so handelt es sich bei x um ein lokales Minimum und somit um eine mögliche Lösung des Optimierungsproblems.

2.3.6 Regularitätsbedingungen

Regularitätsbedingungen oder auch Constraint Qualification genannt, sind Bedingungen, die wenn sie erfüllt sind, für jeden zulässigen Punkt $\overline{x} \in \mathcal{Z}$, welcher ein lokales Minimum des Optimierungsproblems ist, die KKT-Bedingungen an diesem Punkt implizieren [9]. Es gibt viele verschiedene Constraint Qualifications, eine gute Übersicht bietet M. Solodov [16]. In [16] wird ebenfalls gezeigt in welcher Relation die einzelnen Regularitätsbedingungen zueinander stehen, also aus welcher CQ man andere CQs ableiten kann. Im Folgenden werden drei Regularitätsbedingungen präsentiert.

2.3.6.1 Slater-Bedingung

Die Slater-Bedingung oder auch Slater CQ (Slater Constraint Qualification) ist eine solche Regularitätsbedingung. Sie wird auch als Voraussetzung für starke Dualität bei der Lagrange Dualität genutzt und besagt [8]:
Sind alle h_k affin linear und alle g_j ebenfalls affin linear oder nicht affin linear aber konkav und existiert mindestens ein zulässiger Punkt $\overline{x}$ für den $g_j(\overline{x}) < 0$ für alle konkaven Funktionen g_j gilt, dann ist jeder lokale Minimalpunkt ein KKT-Punkt.
Den Beweis, dass die Slater-Bedingung eine Constraint Qualification ist und wie aus den gerade genannten Bedingungen die KKT-Bedingungen folgen, ist in [10] zu finden.

2.3.6.2 Linear Independence Constraint Qualification

Eine weitere Constraint Qualification, und wohl die bekannteste von allen, ist die Linear Independence constraint qualification oder kurz LICQ. Sie besagt:

Sind alle $\nabla h_k(x^*)$ und $\nabla g_j(x^*)$ der aktiven Constraints linear unabhängig und x^* ein lokales Minimum, dann sind auch die KKT-Bedingungen an diesem Punkt erfüllt. (Und sei (x^*, λ^*, ν^*) der KKT-Punkt zu x^*).
Es kann gezeigt werden, dass die KKT-Multiplikatoren λ^* und ν^* in diesem Fall eindeutig sind, es also keinen alternativen Wert für λ^* bzw. ν^* gibt, für die die KKT-Bedingungen ebenfalls gelten. Beweis durch Gegenannahme [3]:

$$\begin{aligned}
\text{GA: Sei } (x^*, \overline{\lambda}, \overline{\nu}) \text{ ein weiterer KKT-Punkt neben } (x^*, \lambda^*, \nu^*), \quad & \overline{\lambda} \neq \lambda^*, \overline{\nu} \neq \nu^* \\
\Rightarrow 0 &= \nabla f(x^*) + {\lambda^*}^T * \nabla g(x^*) + {\nu^*}^T * \nabla h(x^*) \\
&= \nabla f(x^*) + \overline{\lambda}^T * \nabla g(x^*) + \overline{\nu}^T * \nabla h(x^*) \\
\Rightarrow 0 &= ({\lambda^*}^T - \overline{\lambda}^T) * \nabla g(x^*) + ({\nu^*}^T - \overline{\nu}^T) * \nabla h(x^*)
\end{aligned}$$

Für alle nicht aktiven Constraints gilt laut der KKT-Bedingung (15) $\lambda_j^* = \overline{\lambda}_j = 0$ und wegen der linearen Unabhängigkeit von $\nabla h_k(x^*)$ und $\nabla g_j(x^*)$ der aktiven Constraints:

$$0 = \alpha * \nabla g(x^*) + \beta * \nabla h(x^*) \quad \Leftrightarrow \quad \alpha = 0 \wedge \beta = 0, \qquad \alpha \in \mathbb{R}^m, \beta \in \mathbb{R}^l$$

(Definition linearer Unabhängigkeit). Somit gilt ${\lambda^*}^T - \overline{\lambda}^T = 0 \Rightarrow \lambda^* = \overline{\lambda}$ und ${\nu^*}^T - \overline{\nu}^T \Rightarrow \nu^* = \overline{\nu}$. Folglich war die Gegenannahme falsch, d.h. die Eindeutigkeit der KKT-Multiplikatoren ist bei LICQ garantiert.
Lineare Unabhängigkeit ist zwar einfach zu kontrollieren, aber gleichzeitig relativ restriktiv. So gibt es viele Optimierungsprobleme unter NB, die die KKT-Bedingungen einhalten, nicht aber die LICQ. LICQ wird deshalb auch als eine starke Regularitätsbedingung bezeichnet, also eine Bedingung, die relativ schwierig einzuhalten ist. Eine schwächere Regularitätsbedingung ist die Mangasarian-Fromovitz Constraint Qualification [14], die im folgenden Kapitel beschrieben wird.

2.3.6.3 Mangasarian-Fromovitz Constraint Qualification

Die Mangasarian-Fromovitz Constraint Qualification (kurz MFCQ) besagt:
Sind alle $\nabla h_k(x^*)$ linear unabhängig, x^* ein lokales Minimum und existiert ein Vektor $d \in \mathbb{R}^n$ für den folgendes gilt:

$$\nabla h_k(x^*)^T d = 0 \text{ und } \nabla g_j(x^*)^T d < 0 \text{ für alle aktiven Inequality Constraints,}$$

dann sind die KKT-Bedingungen an diesem Punkt erfüllt [9].
Es kann gezeigt werden, dass LICQ $\Rightarrow$ MFCQ aber MFCQ $\not\Rightarrow$ LICQ [10]. Das heißt, gilt LICQ so gilt auch MFCQ, aber gilt MFCQ so muss nicht notwendigerweise auch LICQ gelten.

2.4 Weitere Lösungsverfahren

Um ein Optimierungsproblem unter NB zu lösen gibt es unterschiedliche Methoden. Eine schon vorgestellte Methode ist die Optimierung mit der Lagrange Funktion. Im Folgenden werden drei weitere Methoden dargestellt. Bei all diesen Methoden wird nicht $f(x)$ alleine minimiert, sonder ein Funktion in der zusätzlich zu $f(x)$ noch weitere Therme enthalten sind, die entweder ein Minimum von $f(x)$ an einem nicht zulässigem Punkt $x \notin Z$ unmöglich machen, oder dies zumindest erschweren. Mit dem Erschweren eines Minimums ist gemeinet, dass zusätzlichen Kosten, so genannte Strafen erhoben werden.

2.4.1 Barriere Methoden

Bei der (Log-)Barriere Methode wir folgende Funktion minimiert:

$$\min_x f(x) - \mu \sum_j log(-g_j(x)) \text{ unter der Bedingung } h(x) = 0 \tag{17}$$

Die Logarithmusfunktion (eigentlich $-log(-r)$) wird hier als Barrierefunktion bezeichnet. Allgemein muss für eine Barrierefunktion $b(x) \in \mathbb{R}^n \to \mathbb{R}$ folgendes gelten [11]:

- $b(x) \to \infty$,wenn $\lim_x max(\{g_j(x)\}) \to 0$

Diese Bedingung ist so zu lesen, dass sich eine unüberwindbare Barriere auftut, sobald die Grenze des zulässigen Bereichs erreicht wird (vgl. Abb. 5). Die Funktion $-log(-g_j(x))$ garantiert genau das. Für ein $\overline{x} \in \mathcal{Z}$ aus dem zulässigem Bereich strebt $-log(-g_j(x))$ gegen unendlich, falls $g_j(\overline{x})$ gegen 0 strebt. Für immer kleiner werdende Werte von μ wird die Barriere immer näher nach außen (Richtung unzulässigen Bereich) gedrückt und die Funktion im zulässigen Bereich wird immer flacher. Für $\mu \to 0$ nähert sich die Logarithmus-Barrierefunktion der Funktion $I(g_j(x)) = \begin{cases} 0 & \text{if } g_j(x) < 0 \\ \infty & \text{otherwise} \end{cases}$ an (vgl. Abb. 3). Diese Beobachtung bedeutet, dass für $\mu \to 0$ das Lösen der Funktion (17) äquivalent zum Lösen von (5) ist, wobei statt für alle $g_j(x) \leq 0$ zu fordern nun die Bedingung $g_j(x) < 0$ gefordert wird. Welche Barrierefunktion bei der Barriere Methoden gewählt wird, ist prinzipiell egal. Häufig werden Funktionen gewählt, die besondere Eigenschaften haben, wie die (mehrmalige) stetige Differenzierbarkeit, die beim numerischen Lösen des Problems hilfreich ist. Hier wurde nur die Logarithmus-Barrierefunktion vorgestellt, da sie die populärste aller Barrierefunktionen ist und schon früh verwendet wurde.

2.4.2 Penalty Methoden

Eine weitere Moglichkeit zum Lösen von Optimierungsproblemen mit NB bieten die Penalty Methoden. Sie erweitern die zu minimierende Funktion um einen Strafterm $b(x)$, der folgende Eigenschaften besitzt:

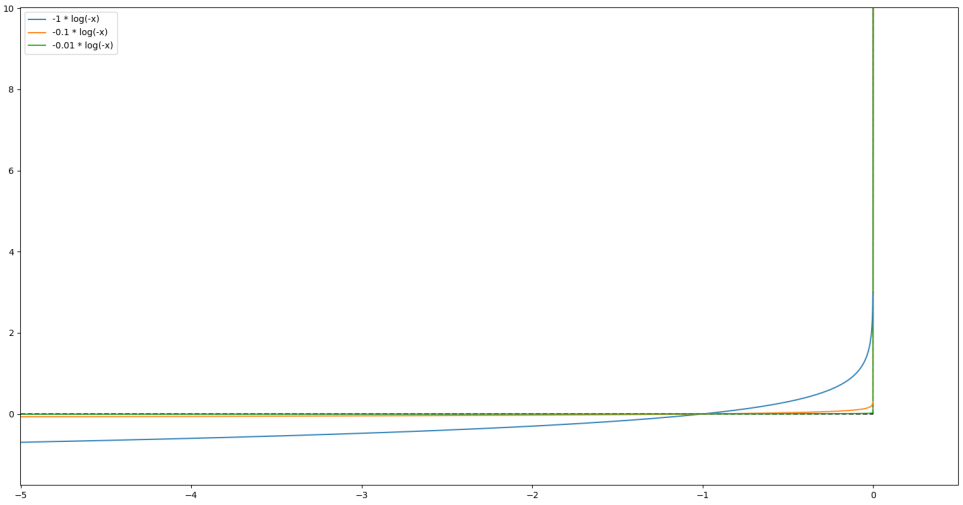

Abbildung 3: Plot der Logarithmus-Barrierefunktion

Hier ist die Logarithmus-Barrierefunktion mit unterschiedlichen μ-Werten abgebildet. Je kleiner der μ-Wert, desto später steigt die Funktion ins Unendliche und desto flacher (näher an 0) ist sie für alle negativ reellen Zahlen. (Original aus [17])

- $b(x) = 0$,wenn $g(x) \leq 0 \wedge h(x) = 0$
- $b(x) > 0$ andernfalls

Das folgende Problem wird bei der allgemeinen Penalty Methode gelöst:

$$\min_x f(x) + b(x)$$

Bei der Squared Penalty Methode sieht das zu lösende Problem folgendermaßen aus:

$$\min_x f(x) + \mu \sum_j I(g_j, x) * g_j(x)^2 + \mu \sum_k h_k(x)^2 \tag{18}$$

$$I(g_j, x) = \begin{cases} 0 & \text{if } g_j(x) \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

Es ist zu sehen, dass das Quadrieren von $h_k(x)$ und das Quadrieren von $g_j(x)$ in Verbindung mit $I(g_j, x)$ die obigen Regularien für Strafterme einhalten. Mit diesem Verfahren ist garantiert, dass für ein zulässiges Minimum x^* von f $f(x^*) + \mu \sum_j I(g_j, x^*) * g_j(x^*)^2 + \mu \sum_k h_k(x^*)^2 = f(x^*)$ gilt. Es ist jedoch nicht garantiert, dass mit dieser Methode ein zulässiges Minimum gefunden wird. Hierzu ein kleines Beispiel:

Sei $f(x) = x^2$ und die NB $h(x) = x - 1 = 0$ (Lösung ist trivialer Weise $f(1) = 1$)

gegeben. Formuliert man für dieses Problem die Formel für die Squared Penalty Methode, lautet diese wie folgt: $\min_x x^2 + (x-1)^2$ (mit $\mu = 1$). Diese Funktion hat bei $x = 0,5$ sein Minimum. Somit würde man beim Lösen des Problems mit der Squared Penalty Methode eine unzulässige Lösung (da $x = 0.5$ wegen $x - 1 = 0$ unzulässig ist) finden. Was man auch sehr schön beobachten kann, ist, dass ähnlich wie bei der Barriere Methode, die Wahl von μ entscheidend für die Genauigkeit der gefundenen Lösung ist (s. Abb. 4). Allerdings muss bei der (Squared) Penalty Methode, anders als bei der Log-Barriere Methode eine großer Wert für μ gewählt werden. Ist also das exakte Einhalten von Constraints nicht unbedingt notwendig, ist die Squared Penalty Methode eine gute Methode, um schnell und einfach eine Lösung des Problems zu berechnen.

Eine Methode, welche die Strafidee auch enthält, ist die erweiterte Lagrange Methode (s. nächster Abschnitt).
Um den Unterschied der (Squared) Penalty Methode und der (Log-) Barriere Methode aufzuzeigen, ist Abbildung 5 gedacht. Eine Barriere verursacht hohe Kosten/Strafen an den Rändern des zulässigen Bereichs, das heißt im zulässigen Bereich werden Strafen erhoben (egal wie klein man μ wählt). Bei einem Strafterm hingegen werden keine Strafen im zulässigen Bereich erhoben, erst außerhalb des zulässigen Bereichs ist dies der Fall. Folglich ist für ein im zulässigen Bereich gefundenes Minimum bei einer Barriere Methode nicht immer gewährleistet, dass es gleich dem Minimum von $f(x)$ ist, was bei einer Penalty Methode gewährleistet ist.

2.4.3 Erweiterte Lagrange Methode

Die erweiterte Lagrange Methode (englisch augmented Lagrangian method) ist eine Methode zum iterativen Lösen von Optimierungsproblemen mit NB. Anstatt $f(x)$ zu minimieren wird folgende Funktion minimiert:

$$\min_x f(x) + \mu \sum_{j=1}^{m} I(j) g_j(x)^2 + \sum_{j=1}^{m} \lambda_j g_j(x) + \mu \sum_{k=1}^{l} h_k(x)^2 + \sum_{k=1}^{l} \nu_k h_k(x) \tag{19}$$

$$\mu \geq 0, \quad I(j) = \begin{cases} 1 & \text{if } g_j(x) \geq 0 \vee \lambda_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

Die Terme $\mu \sum_{j=1}^{m} I(j) g_j(x)^2$ und $\mu \sum_{k=1}^{l} h_k(x)^2$ erheben zusätzliche Strafkosten, falls ein Constraint nicht eingehalten wird. Die Funktion beziehungsweise die Methode ist so konstruiert, dass die NB je nach Wahl von μ und der Anzahl an Iterationen beliebig genau erfüllt werden können. Bei jeder Iteration wird das x' berechnet, welches die Funktion (19) mit den aktuellen Parametern der Iteration (aktuelle μ, λ_j und ν_k - Werte) minimiert. Mit diesem x' werden anschließend die λ_j und ν_k - Werte für die nächste Iteration wie folgt

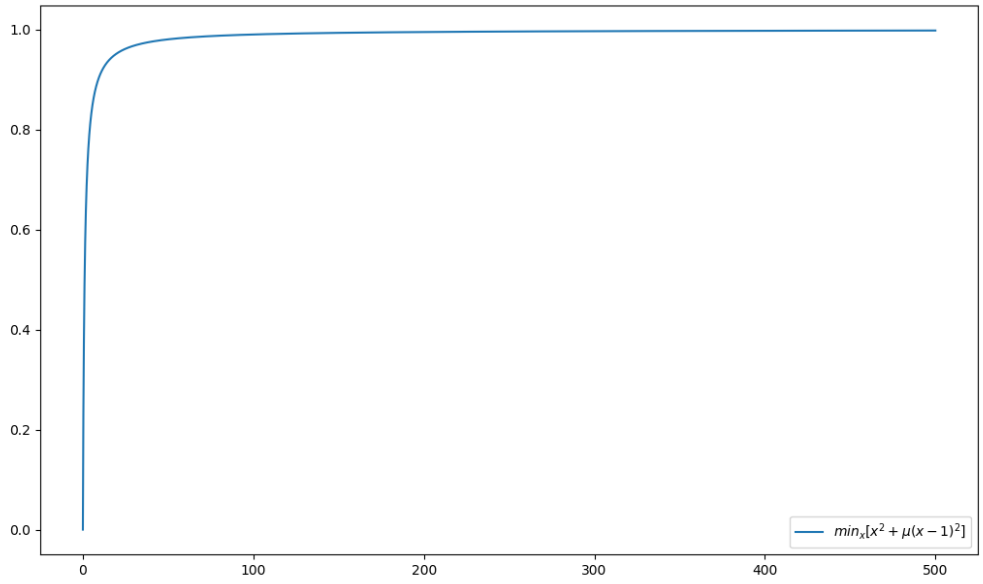

Abbildung 4: Plot der Funktion $p(\mu) = \min\limits_{x} x^2 + \mu(x-1)^2$

Es ist die Funktion $p(\mu) = \min\limits_{x} x^2 + \mu(x-1)^2$ geplottet. Wie man sehr schön sehen kann, schmiegt sich die Funktion $p(\mu)$ für immer größer werdende Werte für μ immer näher zu 1 an, welches das gesuchte Optimum des Problems ist.

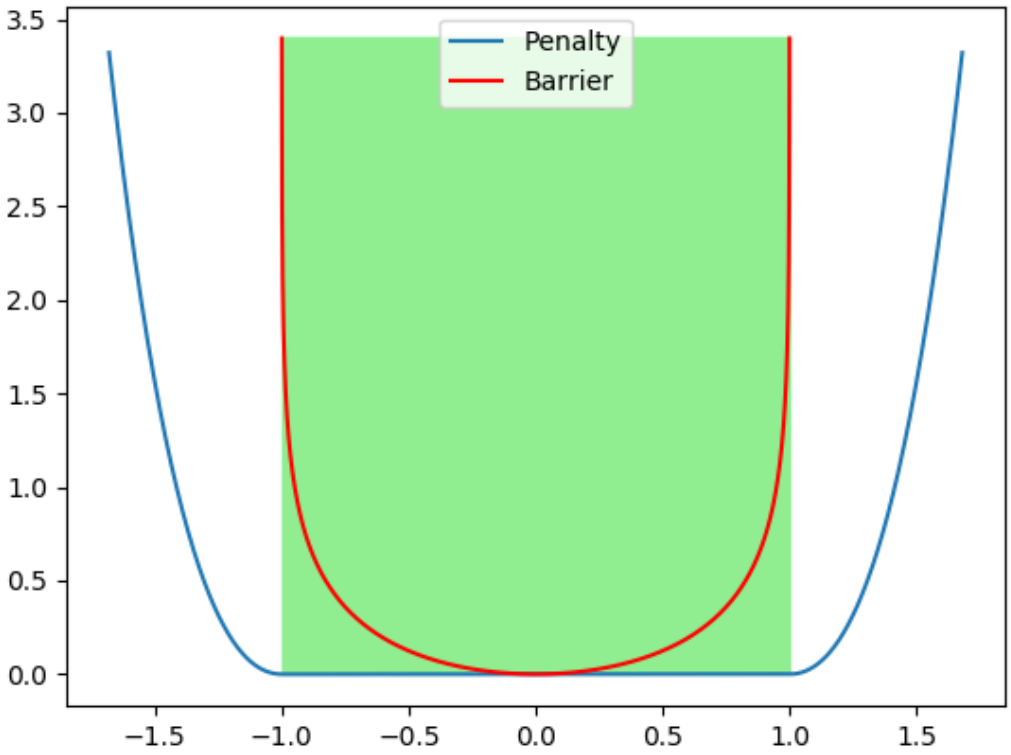

Abbildung 5: Unterschied Logarithmus-Barrierefunktion und Squared Penalty

Die Logarithmus-Barrierefunktion ($-log(-(x^2-1))$) ist in rot dargestellt, der quadrierte Strafterm ($(x^2-1)^2$ für $x^2-1>0$ sonst 0) in blau (für $g(x)=x^2-1\leq 0$). Der grüne Bereich ist der zulässige Bereich. (vgl. Original aus [13])

angepasst:

$$\lambda_j^{neu} = \max(\lambda_j^{alt} + 2\mu g_j(x'),\ 0) \qquad \nu_k^{neu} = \nu_k^{alt} + 2\mu h_k(x')$$

Durch dieses Anpassen generiert $\sum_{j=1}^{m} \lambda_j^{neu} g_j(x')$ in der nächsten Iteration genau den selben Gradienten (dieser Gradient wird auch häufig virtueller Gradient genannt) wie $\mu \sum_{j=1}^{m} I(j) g_j(x')^2 + \sum_{j=1}^{m} \lambda_j^{alt} g_j(x')$ in der zuvorigen Iteration (falls $2\mu g_j(x') \geq 0$) [17]. Selbiges gilt auch für die Gleichheitsbedingungen h_k (ohne gerade genannte Bedingung). Da der Gradient immer in die Richtung des größten Kostenanstiegs zeigt, wird beim Minimieren in der Regel in die entgegengesetzte Richtung des Gradienten gegangen. Durch das Einbauen des virtuellen Gradienten wird also beim Minimieren der Funktion (19) in diejenige Richtung gelaufen, in der $g_j(x)$ und $h_k(x)^2$ kleiner werden. Folglich schiebt das Minimieren den x-Wert in Richtung der Erfüllung der Constraints. Zur Vollständigkeit muss noch der Fall $2\mu g_j(x') < 0$ überprüft werden. Falls $2\mu g_j(x') < 0$ gilt, muss $g_j(x') < 0$ gelten, d.h. der Constraint wird in x' eingehalten und kein Verschieben von x' ist aus Sicht dieses Constraints notwendig. Somit ist $\lambda_j^{neu} = 0$ ein legitimer Schritt im Optimierungsverfahren. Anderst als bei der Penalty Methode, muss bei der erweiterten Lagrange Methode nicht $\mu \to \infty$ gefordert werden, um das exakte Einhalten der NB zu garantieren.

2.5 Sensitivitätsanalyse

Die Sensitivitätsanalyse (sensitivity analysis) beschäftigt sich mit der Frage, wie sehr sich die Lösung des Optimierungsproblems ändern würde, wenn die Eingabeparameter (die Formulierung des Problems) minimal geändert werden. Obiges Problem (5) kann in Abhängigkeit von ϵ beschrieben werden:

$$\min_x f(x,\epsilon) \quad \text{unter den Bedingungen} \quad g(x,\epsilon) \leq 0, \quad h(x,\epsilon) = 0$$

Dabei beschriebt ϵ die (minimale) Änderung der Formulierung des Problems. Häufig wird ϵ auch als Störparameter bezeichnet. Auch die dazugehörige Lagrange Funktion kann in Abhängigkeit des sog. Störparameters definiert werden:

$$\mathcal{L}(x,\lambda,\nu,\epsilon) = f(x,\epsilon) + \lambda^T * g(x,\epsilon) + \nu^T * h(x,\epsilon)$$

Es gilt:

$$\begin{aligned} \forall\, x \in \mathbb{R}^n: \quad & f(x,0) = f(x) \\ & g(x,0) = g(x) \\ & h(x,0) = h(x) \\ \Longrightarrow \forall\, x \in \mathbb{R}^n: \quad & \mathcal{L}(x,\lambda,\nu,0) = \mathcal{L}(x,\lambda,\nu) \end{aligned}$$

Da der Satz über implizite Funktionen in Kürze genutzt wird, hier die Aussage des Satzes [3], [5]:

- Sei $\mathcal{F}: U \subseteq \mathbb{R}^m \times V \subseteq \mathbb{R}^n \to \mathbb{R}^n$ eine stetig differnzierbare Abbildung mit $\mathcal{F}(x,y) = \begin{pmatrix} \mathcal{F}_1(x,y) \\ \vdots \\ \mathcal{F}_n(x,y) \end{pmatrix}$, U, V offene Mengen
- $(x_0, y_0) \in U \times V$ erfüllt $\mathcal{F}(x_0, y_0) = 0$ und
- $\nabla_y \mathcal{F}(x_0, y_0)$ ist invertierbar

so existieren offene Umgebungen $U_0 \in U, V_0 \in V$ von x_0, y_0 und eine eindeutige stetig differenzierbare Funktion $f: U_0 \to V_0$ mit:

$$\begin{aligned} f(x_0) = y_0 \qquad \text{und} \qquad & \forall x \in U_0, y \in V_0 : \mathcal{F}(x,y) = 0 \Leftrightarrow f(x) = y \\ \nabla_x f(x_0) = -\nabla_y F(x_0, f(x_0))^{-1} & \nabla_x F(x_0, f(x_0)). \end{aligned}$$

Seien $f(x,\epsilon), g(x,\epsilon), h(x,\epsilon)$ zweimal stetig differenzierbar, x^* das Optimum des Problems

mit $\epsilon = 0$, LICQ bei x^* erfüllt und (x^*, λ^*, ν^*) der KKT-Punkt zu x^*. Folglich muss bei x^* auf jeden Fall

$$\begin{aligned} \nabla_x L(x^*, \lambda^*, \nu^*) &= 0 \\ \lambda_j^* * g(x^*)_j &= 0 \qquad \forall j \in \{1, 2, ..., m\} \\ h(x^*) &= 0 \end{aligned}$$

gelten (s. KKT-Bedingungen). In einen Vektor geschrieben:

$$\begin{pmatrix} \nabla_x L(x^*, \lambda^*, \nu^*) \\ diag(\lambda^*) * g(x^*) \\ h(x^*) \end{pmatrix} = 0$$

Wir definieren die stetig differnzierbare Funktion $\mathcal{F}(\epsilon, z) : \mathbb{R}^{\mathbb{N}_\epsilon} \times \mathbb{R}^{n+m+l} \to \mathbb{R}^{n+m+l}$ als:

$$\mathcal{F}(\epsilon, z) = \mathcal{F}(\epsilon, (x, \lambda, \nu)) = \begin{pmatrix} \nabla_x L(x, \lambda, \nu, \epsilon) \\ diag(\lambda) * g(x, \epsilon) \\ h(x, \epsilon) \end{pmatrix}$$

Es gilt $\mathcal{F}(\epsilon_0, z^*) \overset{\epsilon_0 = 0, z^* = (x^*, \lambda^*, \nu^*)}{=} \mathcal{F}(0, (x^*, \lambda^*, \nu^*)) = 0$.
Die Jacobi-Matrix

$$\nabla_z \mathcal{F}(\epsilon_0, z^*) = \begin{pmatrix} \nabla_x^2 L(x^*, \lambda^*, \nu^*) & \nabla_x g(x^*)^T & \nabla_x h(x^*)^T \\ diag(\lambda^*) \nabla_x g(x^*) & diag(g(x^*)) & 0 \\ \nabla_x h(x^*) & 0 & 0 \end{pmatrix}$$

ist invertierbar, da LICQ und $\nabla_x^2 L(x^*, \lambda^*, \nu^*)$ positiv definit ist. Laut dem Satz über implizite Funktionen existiert dann eine stetig differenzierbare Funktion $z^*(\epsilon) : U_{\epsilon_0} \subseteq \mathbb{R}^{\mathbb{N}_\epsilon} \to U_{z^*} \subseteq \mathbb{R}^{n+m+l}$ mit $z^*(0) = z^*$ und $\forall \epsilon \in U_{\epsilon_0}, z \in U_{z^*} : \mathcal{F}(\epsilon, z) \Leftrightarrow z = z^*(\epsilon)$ und

$$\nabla_\epsilon z^*(\epsilon_0) = -\nabla_z F(\epsilon_0, z^*(\epsilon_0))^{-1} \nabla_\epsilon F(\epsilon_0, z^*(\epsilon_0)) = -\nabla_z F(0, z^*)^{-1} \nabla_\epsilon F(0, z^*) \tag{20}$$

Das Besondere hieran ist, dass $z^*(\epsilon)$ nicht bekannt ist und auch nicht spezifiziert werden müssen, $\nabla_\epsilon z^*(\epsilon_0)$ ist jedoch bekannt und von Iteresse. $\nabla_\epsilon z^*(\epsilon_0)$ macht eine Aussage darüber, wie sehr sich z, und somit x, λ und ν am Optimum (x^*, λ^*, ν^*) ändern, wenn sich der Wert von ϵ ändert. Dies ist die Sensitivität der optimalen Lösung in Abhängigkeit von ϵ.

$f(x,\epsilon), g(x,\epsilon)$ und $h(x,\epsilon)$ seien wie folgt definiert:

$$\begin{aligned} \epsilon \in \mathbb{R} \qquad f(x,\epsilon) &= f(x) + \epsilon * \hat{f}(x) \\ g(x,\epsilon) &= g(x) + \epsilon * \hat{g}(x) \\ h(x,\epsilon) &= h(x) + \epsilon * \hat{h}(x) \end{aligned}$$

Es ist leicht zu sehen, dass obige Bedingungen immer noch erfüllt sind und deshalb (20) ebenfalls noch gilt:

$$\nabla_\epsilon z^*(\epsilon_0) = -\nabla_z F(0,z^*)^{-1} \nabla_\epsilon F(0,z^*) \tag{21}$$

$$-\nabla_z F(0,z^*)\nabla_\epsilon z^*(\epsilon_0) = \underbrace{\begin{pmatrix} \nabla\hat{f}(x^*) + \lambda^{*\,T}\nabla\hat{g}(x^*) + \nu^{*\,T}\nabla\hat{h}(x^*) \\ diag(\lambda^*)\hat{g}(x^*) \\ \hat{h}(x^*) \end{pmatrix}}_{\text{veränderte Problemformulierung findet sich hier wieder}} \tag{22}$$

Für eine Approximation eines Wertes $z^*(\epsilon)$ kann die Taylor Funktion herangezogen werden (Therme höherer Ordnung werden vernachlässigt).

$$z^*(\epsilon) = z^*(\epsilon_0) + \nabla_\epsilon z^*(\epsilon_0) * (\epsilon - \epsilon_0) = z^* + \nabla_\epsilon z^*(0) * \epsilon$$

Umgeformt zu

$$\nabla_\epsilon z^*(0) = \frac{z^*(\epsilon) - z^*}{\epsilon} = \begin{pmatrix} \pi_x(z^*(\epsilon)) - x^* \\ \pi_\lambda(z^*(\epsilon)) - \lambda^* \\ \pi_\nu(z^*(\epsilon)) - \nu^* \end{pmatrix} = \begin{pmatrix} \triangle_\epsilon x \\ \triangle_\epsilon \lambda \\ \triangle_\epsilon \nu \end{pmatrix}$$

mit

$$\pi_w(z^*(\epsilon)) = \begin{pmatrix} diag(\{[w == x]\}^{dim(w)}) \\ diag(\{[w == \lambda]\}^{dim(w)}) \\ diag(\{[w == \nu]\}^{dim(w)}) \end{pmatrix}^T z^*(\epsilon), \qquad w \in \{x, \lambda, \nu\}$$

($[... == ...] \in \{0,1\}$, Wahrheitswert) und in (22) eingesetzt, ergibt sich:

$$-\nabla_z F(0,z^*) \begin{pmatrix} \triangle_\epsilon x \\ \triangle_\epsilon \lambda \\ \triangle_\epsilon \nu \end{pmatrix} = \begin{pmatrix} \nabla\hat{f}(x^*) + \lambda^{*\,T}\nabla\hat{g}(x^*) + \nu^{*\,T}\nabla\hat{h}(x^*) \\ diag(\lambda^*)\hat{g}(x^*) \\ \hat{h}(x^*) \end{pmatrix} \tag{23}$$

Diese Gleichung setzt die konkrete Veränderung der Lösung $(\triangle_\epsilon x, \triangle_\epsilon \lambda, \triangle_\epsilon \nu)$ mit der Veränderung des Problems am Optimum $(\hat{f}(x^*), \hat{g}(x^*), \hat{h}(x^*))$ in Relation. Es kann somit

die anfangs gestellte Frage, wie sehr sich die Lösung des Optimierungsproblems ändern würde, wenn man die Eingabeparameter, also die Formulierung des Problems minimal ändern würde, beantworten.

3 Debuggingtool für NLP Lösungen

Wie schon in der Einleitung angeschnitten, ist das Verstehen von Lösungen, die ein Optimierer berechnet, ein wichtiger Bestandteil für die passende Formulierung von Optimierungsproblemen unter NB. Hat man eine gute Intuition, welcher Parameter bei der Spezifizierung der NB einen wie großen Einfluss auf das Endergebnis hat, so ist die passende Formulierung von NB und somit ein wichtiger und großer Schritt für die Definition des gewollten Optimierungsproblems deutlich erleichtert. Auch die Fehlersuche bzw. die Suche nach den NB, die dem Optimierungsproblem ungewollte Eigenschaften verleihen, ist ebenfalls erleichtert. Solche Fehler in der Formulierung können zu ungewolltem Verhalten des Roboters (bei Trajektorienoptimierungen) führen (z.B. zu einer Kollision). Um dem Nutzer zu helfen eine solche Intuition aufzubauen und NLP Lösungen zu verstehen, wurde das Debuggingtool, welches in den nächsten Abschnitten vorgestellt wird, entwickelt. Der genutzte Optimierer basierte auf der erweiterten Lagrange Methode undwurde zum Optimieren von Roboter-Trajektorien eingesetzt.

Allgemein ist es bei Optimierunsproblemen interessant woher die Kosten bzw. Strafen kommen und wie hoch die Strafen einzelner Constraints sind. Sind die Kosten/Strafen an einer Stelle einer Roboter-Trajektorie besonders hoch, kann dies ein Indiz dafür sein, dass an dieser Stelle etwas Besonderes passiert ist (was so vielleicht nicht geplant war). Falls einzelne Constraints sehr hohe Strafen haben oder eine kleine Gruppe an Constraints die Gesamtkosten/-strafen maßgeblich mitbestimmen, führt eine genauere Untersuchung dieser Constraints und der Zeitschritte, wo diese Constraints besonders hohe Strafen haben meist zu einem besseren Verständnis der Lösung. Um die Untersuchung von Kosten und Strafen zu gewährleisten, wäre ein einfacher Plot der Kosten/Strafen über die gesamte Trajektorie prinzipiell ausreichend (vgl. Abb. 6). Da es aber bei komplexeren/größeren Optimierunsproblemen unter NB häufig viele Constraints gibt, würde ein Plot über alle Kosten/Strafen sehr unübersichtlich und das Extrahieren von wertvollen Informationen deutlich erschwert werden. Auch das Aufteilen der Constraints, in zum Beispiel Equality- und Inequality-Constraints, um die einzelnen Plots übersichtlicher zu machen, wäre keine zufriedenstellende Lösung des Problems, da sich Constraints gegenseitig beinflussen können und somit diese Informationen leicht übersehen werden können. Um solche großen und unübersichlichen Datenmengen leicht verständlich zu machen, braucht es eine intelligente Filterung der Kosten-/Strafwerte. Darauf aufbauend ist eine sinnvolle Sortierung und Gruppierung von Constraints möglich ist.

Bevor der Aufbau des Tools und dessen Funktionsweise beschrieben wird, sind noch folgende Dinge zu nennen: Bislang war zur Vereinfachung immer nur von Equality-/Inequality-Constraints und einer Kostenfunktion die Rede, deren Straf-/Kostenwerte sich über die gesamte Roboter Trajektorie ziehen. In der Praxis wird eine Nebenbedingung pro Zeit-

schritt der Trajektorie definiert. Hat die Trajektorie also beispielsweise 150 Zeitschritte und man möchte als NB, dass der Roboter an keiner der 150 Zeitschritte kollidiert, so muss man für jeden Zeitschritt eine (die gleiche) NB definieren, die eine Kollision bestraft bzw. verbietet. Bei manchen Constraints muss man auch mehrere NB pro Zeitschritt definieren, da jeder Zeitschritt durch eine Roboterkonfiguration repräsentiert wird und diese immer mehrdimensional ist. Selbiges gilt für Kostenfunktionen (wovon beliebig viele definieren werden können; deren Summe ist dann die letztendliche Kostenfunktion f). Um bei so vielen Nebenbedingungen und Kostenfunktionen den Überblick zu behalten, gilt folgende Notation ($B \in \{C, I, E\}$ C = Kostenfunktion, I = Inequality Constraint, E = Equality Constraint):

Notation	Beschreibung
B_{Name}	Eine Kostenfunktion/Constraint welcher sich über die gesamte Trajektorie erstreckt, besteht aus mehreren Features $B^{\text{Zeitschritt}}_{\text{Name}}$. $B_{\text{Name}} = \sum_{\text{Zeitschritte}} B^{\text{Zeitschritt}}_{\text{Name}}$
$B^{\text{Zeitschritt (als Array)}}_{\text{Name}}$	Eine Kostenfunktion/Constraint für einen definierten Zeitschritt, kann aus mehreren $B^{\text{Zeitschritt, Teil-ID}}_{\text{Name}}$ bestehen. Wird im Folgenden als Feature bezeichnet. $B^{\text{Zeitschritt}}_{\text{Name}} = \sum_{\text{Teil-IDs}} B^{\text{Zeitschritt, Teil-ID}}_{\text{Name}}$
$B^{\text{Zeitschritt, Teil-ID}}_{\text{Name}}$	Ein Teil eines Features.

Also wäre das Feature $E^{[39]}_{\text{QuaternionNorms}}$ ein Teil des Equality Constraint mit dem Namen QuaternionNorms, der die Eigenschaften des Constraints zum Zeitschritt 39 definiert. Den Strafwert dieses Features bzw. den Strafwert des QuaternionNorms Constraints zum Zeitschritt 39 könnte dann zum Beispiel in Abb. 6 bei x=39 ablesen werden.

3.1 Aufbau und Untersuchungsmöglichkeiten

Das Tool (zu finden unter [1]) hat vier verschiedene Sichten/Untersuchungsmöglichkeiten für eine Lösung eines Optimierungsproblems, die ein Optimierer berechnet hat. Die erste Ansicht (Overview) bietet eine Sortierung aller Kostenfunktionen und Constraints nach ihren Gesamtkosten über das Optimierungsproblem (B_{Name}). Zusätzlich beinhaltet das Tool noch neben diesem Ranking auch einen Plot aller Kosten/Strafen von Kostenfunktionen/Constraints, die zu mindestens einem Zeitschritt Kosten/Strafen ≥ 1 haben. Diese Ansicht soll einen Überblick über die Kosten und Strafwerte und deren ungefähre Verteilung geben. Die zweite Ansicht (Important Spots View) soll dabei helfen wichtige Zeitschritte/Features einzelner Constraints und Kostenfunktionen zu extrahieren, diese aber noch nicht in Relation zueinander zu setzen. Hiermit ist das Untersuchen einzelner Kostenfunktionen/Constraints möglich. Der Cluster View (die dritte Ansicht) gruppiert Constraints und Kostenfunktionen. Mit dieser Ansicht kann das Zusammenspiel von unterschiedlichen Kostenfunktionen und Constraints analysiert werden. Die letzte Ansicht bietet eine Sensitivitätsanalyse, um zu schauen, wie sich die Lösung verändern würde, falls die Formulierung des Optimierungsproblems minimal geändert wird.

3.1.1 Overview

Diese Sicht erlaubt dem Nutzer einen Überblick über die gesamten angefallenen Kosten/Strafen des Problems. Existiert eine Kostenfunktion oder ein Constraint, welcher zu mindestens einem Zeitschritt Kosten/Strafen ≥ 1 hat, wird ein Ausschnitt der Kosten/Strafen dieser Kostenfunktion/dieses Constraints in ein Diagramm geplottet. Gibt es mehrere solcher Kostenfunktionen/Constraints werden alle in dieses Diagramm eingezeichnet. Durch einen Doppelklick auf das Diagramm öffnet sich ein neues Fenster in dem das Diagramm genauer untersucht werden kann. Dass der Wert 1 einer Strafe eine nicht mögliche Konfiguration bezogen auf die Constraints anzeigt, ist eine Konvention bei der Formulierung eines Optimierungsproblems. Während kleinere Strafen ($\ll 1$) toleriert werden, sind hohe Strafen v.a. Strafen ≥ 1 ein Zeichen für die Nichteinhaltbarkeit einer Bedingung zu einem Zeitschritt. Somit sieht man anhand dieses Plots direkt an welchen Stellen große Problempunkte der Lösung liegen.

Neben diesem Plot gibt es in dieser Ansicht noch ein Balkendiagramm über alle Kosten und Strafen des gesamten Optimierungsproblems. An ihm kann abgelesen werden, welche Constraints/Kostenfunktionen wie viel Strafen/Kosten verursacht haben. Durch das Balkendiagramm werden die Kostenfunktionen und Constraints nicht nur sortiert, sondern auch in Relation gesetzt. Somit sieht der Nutzer, mit einem ihm vertrauten und deshalb sehr einfach und schnell interpretierbaren Diagramm, auf einen Blick welche Constraints oder Kostenfunktionen einen wie großen Einfluss auf die gesamten Kosten/Strafen des Problems haben. Hohe Strafen eines Constraints können bedeuten, dass das (genaue) Einhalten dieses Constraints sehr schwierig ist und dieser Constraint somit die letztendliche Lösung des Optimierungsproblems maßgeblich beeinflusst hat. Bei solchen Constraints kann eine genauere Untersuchung der Strafen sehr aufschlussreich sein. Eine wichtige Frage bei dieser Untersuchung ist, wo genau die Strafen in der Trajektorie auftreten. Um diese Frage zu beantworten gibt es den Important Spots View, welcher im Folgenden vorgestellt wird.

3.1.2 Important Spots View

Der Important Spots View hilft dabei interessante Zeitschritte von Constrains oder Kostenfunktionen leichter zu erkennen. Für jede Kostenfunktion und für jeden Constraint können die Kosten/Strafen über alle Roboterkonfigurationen als eine Funktion geplottet werden (s. Abb. 6). Interessante Punkte sind dort, wo die Funktion einen signifikanten Sprung macht. Wobei ein Sprung erst dann signifikant ist, wenn er (1) einen lokalen durchschnittlichen Sprungwert (=max(∅Steigung, ∅Gefälle) der Strafen/Kosten des Constraints/Kostenfunktion überschreitet und (2) größer als ein globaler Signifikantswert (=max(alle Kosten-/Strafwerte aller Zeitschritte, 1) $*$ 0.01) ist. Durch diese Signifikantswerte werden die Daten gefiltert und der Nutzer bekommt ein aufgeräumtes Bild der

Strafen-/Kostenverläufe, in dem die wichtigsten Constraints und Kostenfunktionen, um die Lösung des Problems genauer zu untersuchen, abgebildet sind. Doch nicht nur uninteressante Constraints und Kostenfunktionen werden herausgefiltert, es werden auch uniteressante Zeitschritte-/Intervalle herausgeschnitten, also Intervalle in denen keine Funktion einen signifikanten Sprung hat. Somit wird die zu betrachtende Datenmenge auf ein Minimum reduziert. Will man doch den gesamten Kosten/Strafenverlauf sehen, kann man dies durch Doppelklicken auf das Diagramm erreichen. Es öffnet sich ein zweites Fenster, in dem der gesamte Funktionsverlauf geplottet ist und in dem der gesamten Kosten-/Strafenverlauf noch einmal genauer unter die Lupe genommen werden kann. Wie schon beim Overview, sind auch hier die Kostenfunktionen und Constraints nach insgesamt verursachten Kosten/Strafen sortiert.

Der Grund, warum signifikante Sprünge untersucht werden, liegt in der Art der Optimierung, wie gleich näher erklärt wird. Allgemein wird bei der Optimierung versucht, die Gesamtkosten-/strafen so gering wie möglich zu halten. Verursachen mehrere Constraints an einem Zeitschritt Strafen, so wird beim Optimieren ein Kompromiss für die Konfiguration in diesem Zeitschritt gesucht, die global am günstigsten ist. Um deutlicher zu machen, warum der Kompromiss global (über die gesamte Robotertrajektorie) und nicht lokal (die Konfiguration, die, wenn man nur diesen einen Zeitschritt anschaut am wenigsten Kosten verursacht) am günstigsten sein muss, wird ein Beispiel betrachtet:

Es soll eine Robotertrajektorie optimiert werden, bei der das Ziel ist, den besten Weg von einem festen Punkt $P1$ zu einem leicht daneben liegenden Zielpunkt $P2$ zu finden. Dafür gibt es eine Kostenfunktion und zwei Constraints. Einen Constraint, welcher dafür sorgen soll, dass der Roboter am Punkt $P1$ startet und einen für das Einhalten der Zielbedingung, also am Ende am Punkt $P2$ zu sein. Die Kostenfunktion verursacht Kosten für jede Bewegung, je mehr Distanz zurrückgelegt wird, desto höher sind die Kosten (Kosten steigen quadratisch mit der Distanz). Weiter sind die Strafen, die durch das Nichteinhalten von Constraints verursacht werden, deutlich größer als die Kosten der Kostenfunktion. Schaut man sich nun jeden Zeitschritt einzeln an und minimiert lokal die Kosten, so würde der Roboter zu jedem Zeitpunkt außer dem letzten an Punkt $P1$ verweilen. Erst ganz am Ende würde sich der Roboter bewegen, da die Strafe, nicht am Punkt $P2$ zu sein höher ist, als die durch die Bewegung verursachten Kosten. Es ist aber recht leicht zu verstehen, dass das nicht die global optimale Lösung ist. Die wäre nämlich, jeden Zeitschritt einen kleinen Schritt in Richtung $P2$ zu machen und damit die Kosten immer klein zu halten, was letztendlich aufsummiert zu den geringsten Gesamtkosten führt.

Gäbe es also bei diesem Beispiel einen siginifikanten Sprung der Kosten, so wäre dies sehr interessant zu untersuchen. Bei solchen simplen Beispielen ist es recht einfach die Lösung zu verstehen und auch warum zu einem Zeitschritt welche Kosten verursacht werden. Aber bei größeren und komplexeren Problemen ist dies nicht mehr gegeben. Es

gilt aber immer noch, dass die Kosten und Strafen an einem Zeitpunkt nicht isoliert betrachtet werden dürfen, sondern dass man das Große und Ganze betrachten muss. Auch das Untersuchen von signifikanten Sprüngen ist bei komplexeren Problemen immer noch sinnvoll, um die Lösung besser zu verstehen.

Ein signifikanter Sprung bei einer Strafe/Kosten weist auf eine plötzlich erhöhte oder erniedrigte Schwierigkeit, einen guten Kompromiss zu finden hin. Es ist also schwieriger geworden alle oder zumindest manche Bedingungen für einen Zeitschritt einzuhalten. Dafür kann es mehrere Gründe geben. Es kann beispielsweis sein, dass auf einmal ein oder mehrere neue Constraints hinzukommen, die das Einhalten einer neue Bedingung forcieren sollen. Dadurch wird unter Umständen die Priorität, eine andere Bedingung einzuhalten abgeschwächt. In Folge der geringen Priorität, wird bei der Optimierung eine (kleine) Strafe durch das Nichteinhalten der Bedingung in Kauf genommen. Solche Verschiebungen der Prioritäten verschiedener Bedingungen können gewollt sein, da zum Beispiel das Vermeiden von Kollisionen als wichtiger angesehen wird, als das langsame Bewegen des Roboters. Aber eine Verschiebung der Prioritäten kann auch ungewollt sein und somit zu ungewolltem Verhalten des Roboters führen. Genau solche Punkte sind dann besonders wichtig zu finden, um zu verstehen, warum der Roboter in der Lösung ein ungewolltes Verhalten aufweist. Auch für die Korrektur der Problemformulierung ist es essenziell solche Punkte aufzuspüren und die einzelnen Constraints und ihre Gewichtungen zu identifizieren, um diese anzupassen. Wie man also sieht, ist die Untersuchung von signifikanten Sprüngen eine gute Herangehensweise, um eine Lösung die ein Optimierer berechnet hat, (besser) zu verstehen. Häufig ist auch ein Blick in die Lösung, also in unserem Fall auf das Video in dem der Roboter seine Bewegungsbahn abfährt, hilfreich um zu erkennen ob es sich bei dem signifikanten Sprung um einen gewollten oder um einen ungewollten Sprung bzw. Verschiebung der Prioritäten handelt. Es hilft auch dabei zu sehen, welche Constraints zu einem Zeitpunkt überhaupt Kosten verursachen könnten. Das beste Beispiel sind Constraints, die Regeln für die Interaktion des Roboters mit einem Objekt definieren. Interagiert der Roboter in der Lösung nur in gewissen Zeitintervallen mit dem Objekt, so kann ein solcher Constraint auch nur in diesen Zeitintervallen für das Nichteinhalten dieser Regeln Strafen verursachen.

3.1.3 Cluster View

Wie schon im vorigen Abschnitt angesprochen, ist die Optimierung eine Kompromisssuche. Das Abwägen von verschiedenen Kosten und Strafen führt letztendlich zur resultierenden Lösung. Auch die Bedeutsamkeit der Untersuchung von signifikanten Sprüngen wurde in diesem Abschnitt dargelegt. Bislang wurden die Sprünge aber nur einzeln betrachtet und nicht miteinander verglichen.

Im Cluster View werden Constraints und Kostenfunktionen basierend auf der Position ihrer signifikanten Sprünge gruppiert. Haben mehrere Constraints/Kostenfunktionen an

derselben Stelle einen signifikanten Sprung, so werden diese gruppiert. Diese Gruppenbildung hilft dabei, die Sprünge der einzelnen Kostenfunktionen und Constraints miteinander zu vergleichen. Wie schon beim Important Spots View, werden auch in dieser Ansicht nur die interessanten Intervalle geplottet und uniteressante Dinge herausgeschnitten. Interessante Intervalle sind dort, wo mindestens zwei Funktionen einen signifikanten Sprung der Kosten/Strafen haben. Um den Umgang mit den einzelnen Gruppierungen zu erleichtern, kann man die einzelnen Gruppen (=Cluster) nach den folgenden Kriterien sortieren:

Name	Bedeutung
Cost	Gesamt verursachte Kosten der Kostenfunktion/ des Constraints.
Size	Anzahl der Kostenfunktionen/Constraints die zusammen eine Gruppe bilden.
Quality	Anzahl der Punkte/Zeitschritte an der alle in der Gruppe beteiligten Kostenfunktionen/Constraints einen signifikanten Sprung/Abfall haben.

Neben der dazugewonnenen Möglichkeit die Sprünge zu vergleichen, ist hier auch zu sehen, ob viele Sprünge an der selben Stelle vorkommen oder ob sie breit verteilt sind. Je nach Sortierung können unterschiedliche Dinge besser oder schlechter begutachtet werden, weshalb im Folgenden die jeweiligen Kerngedanken hinter den Sortierungen vorgestellt werden. Die beiden Sortierungen nach Kosten (Cost) und Größe (Size) bieten sich an, um die Konfigurationen, bei denen mehrere Kostenfunktionen/Constraints signifikante Sprünge haben, nach der Bedeutsamkeit zu sortieren. Haben viele Kostenfunktionen/Constraints bei einer Konfiguration (also einem Zeitschritt in der Trajektorie) einen Sprung, bedeutet dies, dass der Kompromiss der für diesen Zeitschritt gefunden wurde trotzdem noch bei vielen Kostenfunktionen/Constraints überdurchschnittlich starke Kosten/Strafen verursacht. Man beachte, dass dieser Kompromiss global die beste Lösung ist. Das heißt, dass diese Konfiguration möglicherweise ein entscheidender Punkt ist, der erklärt, warum die gefundene Lösung so aussieht, wie sie aussieht. Ähnliches gilt auch für einzelne kleine Gruppen von Constraints, die zusammen relativ hohe Strafen verursacht haben.

Ein Blick auf die Qualität der Cluster (Quality) kann sehr informativ sein. Schlagen Kosten/Strafen von mehreren Kostenfunktionen/Constraints häufig zusammen aus, so deutet dies darauf hin, dass die Kostenfunktionen/Constraints sehr ähnliche Dinge bekosten/Bedingungen beschreiben. An einem solchem Ausschlagspunkt ist die Konfiguration der Lösung dann so definiert, dass diese sehr ähnlich bekosteten Dinge/ähnliche Bedingungen nicht ganz eingehalten sind und dadurch die Kostenfunktionen/Constraints Kosten/Strafen verursachen.

Findet man solche Gruppen von Kostenfunktionen oder Constraints, die häufig an den selben Punkten eine signifikante Kosten-/Strafenänderung aufweisen, kann die Untersuchung der Kurvenverläufe über die gesamte Trajektorie weitere Informationen hervorbringen. Gibt es Regionen in denen die Kurven auseinanderdriften oder haben sie über

die gesamte Bewegungsbahn fast immer identische Kosten/Strafwerte? Regionen in denen die Kurven auseinanderdriften, können verraten, worin sich die Constraints unterscheiden, die sonst eigentlich immer recht ähnliche Verhalten aufgewiesen haben. Zu wissen, welche Constraints sich worin unterscheiden und welche Constraints ungefähr ähnlich sind, kann bei der Interpretation der Lösung sehr hilfreich sein.

Durch das Untersuchen von Clustern kann also gut bestimmt werden, welche Constraints ähnliche Bedingungen haben bzw. welche ungefähr die selben Zile verfolgen. Man sieht auch, wann solche Ziele besonders schwierig einzuhalten waren und es deshalb an diesen Stellen zu (hohen) Strafkosten gekommen ist. Ein weiterer Vorteil solcher Gruppierungen - und eigentlich auch der Wichtigste - ist, dass die Gesamtkomplexität der Lösung damit reduziert werden kann. Man muss sich jetzt nicht mehr drei oder vier Constraints einzeln anschauen, sonder weiß, dass diese Constraints ungefähr auf dasselbe Ziel hinaus wollen, und kann die Gruppe von Constraints als einen großen Constraint betrachten.

Um die Untersuchung der Lösung noch einfacher zu gestalten, ist die Farbe, in der die Funktionen/Constraints geplottet werden, in jeder Ansicht gleich. Somit kann selbst beim Wechseln zwischen den verschiedenen Ansichten immer wieder sehr schnell die zu untersuchende Funktion/der Constraint gefunden werden. Genau dieser Wechsel zwischen verschiedenen Ansichten kann auch aufschlussreiche Dinge hervorbringen: Gibt es zum Besipiel einen Constraint, der (mehrere) interessante Punkte aufweist, aber dieser Constraint in keinem der Cluster vorkommt, zeigt dies, dass dieser Constraint nur in sehr geringem Maße ein ähnliches Ziel, wie andere Constraints und Kostenfunktionen verfolgt. Je nachdem, wie wichtig das Erreichen dieses Ziels im Vergleich zu den Anderen sein soll, ist eine entsprechende Gewichtung notwendig.

3.1.4 Sensitivity Analysis View

Der Sensitivity Analysis View besteht aus einer Reihe von Säulendiagrammen. In jedem Säulendiagramm (ein Säulendiagramm pro Constraint/Kostenfunktion) sind die approximierten Werte, wie sehr sich die Trajektorie ändern würde, wenn es die einzelnen Features des Constraints bzw. der Kostenfunktion nicht geben würde, abzulesen. Dabei spiegelt jede Säule ein Feature wieder. Die Säulendiagramme sind nach der maximalen Änderung sortiert. Das bedeutet, dass der Constraint/die Kostenfunktion, welche(r) die Lösung am meisten ändern würde, ganz oben platziert ist. Kostenfunktionen und Constraints, die nichts ändern würden, werden nicht angezeigt.

Je größer die Säule, desto größer ist der Einfluss dieses Features auf die gefundene Lösung und umgekehrt. Eine sehr interessante Betrachtung ist daher der Vergleich der interessanten Zeitschritte (Important Spots View), also der Strafen/Kosten der Features mit den dazugehörigen Sensitivitätswerten. Aber auch mit der Untersuchung von einzelnen Sensitivitätswerte können hilfreiche Informationen, die zum Verständnis der Lösung beitragen, gewonnen werden. Zum Beispiel sollte der Einfluss einer Endbedingung, also

einem Constraint, der definiert, was der Roboter am Ende der Bewegung erreichen/berühren sollte, relativ groß gegen Ende der Trajektorie sein. Ist dies nicht der Fall, kann dieser geringe Einfluss der Endbedingung ein Grund für das unerwünschte Verhalten des Roboter sein. Ebenfalls aufschlussreich ist der Blick auf die Sensitivitätswerte aller Kostenfunktionen und Constraints über die gesamte Trajektorie, da sie zeigt, wie sich die einzelnen Prioritäten/Einflüsse der einzelnen Constraints über die Zeit verändern. In Verbindung mit den Clustern kann recht genau herausgefunden werden, zu welchem Zeitpunkt welche Ziele besonders wichtig einzuhalten waren.

Eine neue Information, die diese Ansicht bringt, sind die Einflüsse von Constraints, die nie eine Strafe verursacht haben, also immer eingehalten wurden (schwarz dargestellte Constraints). Ein Grund, warum ein Constraint die ganze Zeit über eingahalten wird kann verschiedene Gründe haben: Der einfachste ist, dass die Bedingung, die der Constraint definiert, sehr einfach einzuhalten ist. Ist dies der Fall, hat dieser Constraint keinen großen Einfluss auf die Optimierung genommen und folglich nur sehr kleine Sensitivitätswerte. Ein anderer Grund für das dauerhafte Einhalten einer Bedingung ist, dass das Nichteinhalten so (exorbitant) große Kosten verursachen würde, dass bei der Optimierung darauf geachtet wurde, dass diese Bedingung auf jeden Fall immer eingehalten wird. Als Resultat muss der Constraint dann aber auch große Sensitivitätswerte haben.

3.2 Untersuchung einer Lösung

Im dargestellten Beispiel sollte der Roboter eine Box von einem Ort zu einem anderen Ort befördern. Die Box ist vom Roboter aber so weit entfernt, dass zum Unstellen/Berühren der Box ein Stab, den der Roboter aufheben muss, als Hilfsmittel benötigt wird (da der Roboter stationär ist). Dazu ist an der Oberseite der Box ein Haken/Ring angebracht. Um zu überprüfen wie hilfreich das Debuggingtool ist, wurden in diesem Besipiel extra ein paar Nebenbedingungen so umgeschrieben, dass die Lösung des Optimierungsproblems unzumutbare (infeasible) Konfigurationen besitzt und der Roboter ungewollte Dinge tut.

Der erste Blick auf die Kostenverteilung und den Plot, der alle Constraints und Kostenfunktionen anzeigt, die Strafen/Kosten größer gleich Eins haben (s. Abb. 7), verrät direkt, dass bei der Formulierung des Problems ein Fehler gemacht wurde. Es sind insgesamt drei Constraints, die Strafen ≥ 1 verursacht haben, und manche von ihnen sogar häufiger oder über einen längeren Zeitraum. Auch dass der Ursprung der meisten Strafen und Kosten hauptsächlcich von vier Constraints beherrscht wird und keiner Kostenfunktion, legt nahe, dass die Formulierung mancher Constraints nicht ideal war. Am Strafenverlauf des Inequality-Constraints PairCollision-box-table kann man erkennen, dass es zwischen den Zeitschritten 96 und 98 zu einer Kollision der Box mit dem Tisch gekommen sein muss. In diesem Zeitraum verursacht auch der Equality-Constraint

NewtonEuler_DampedVelocites-box extrem hohe Strafen.

Eine Ansicht weiter, im Important Spots View (s. Abb. 8), ist dies auch noch einmal zu erkennen. Ebenso sind die beiden weiteren Zeitpunkte, an denen der Constraint NewtonEuler_DampedVelocites-box Strafen über Eins auslöst festzustellen; unter anderem zum Zeitpunkt 79. Zu diesem Zeitpunkt gibt es viele Constraints, die einen plötzlichen Strafenanstieg haben. Schaut man sich an, welche Constraints dort Strafen verursachen, so fällt auf, dass die meisten Constraints für die Interaktion des Stabs mit dem Ring definiert sind (zu sehen an den Namen der Constraints, viele entahlten im Namen stick-ring4). Diese Beobachtung ist auch im Cluster View (s. Abb. 9) recht schnell gemacht, vorallem wenn man die Gruppen nach ihrer Größe (Size) sortiert hat. Beim weiteren Begutachten der Important Spots und Cluster fallen noch weitere Zeitschritte beziehungsweise Zeitintervalle auf, bei denen es hohe Strafen gibt. Zum Beispiel gibt es drei Funktionen (qItself#14, ZeroQVel-stick und F_Pose-stick), die um den Zeitschritt 39 einen Sprung aufweisen. Zwei von ihnen (ZeroQVel-stick und F_Pose-stick) lassen darauf schließen, dass hier das erste Mal der Strab berührt wurde. Auch der Sprung der dritten Funktion, der Kostenfunktion qItself#14, welche in Relation zur Beschleunigung des Roboters Kosten generiert, bekräftigt diese Vermutung. Da für das Berühren des Stabs abgebremst (negative Beschleunigung) und dannach wieder beschleunigt werden muss, besitzt der Kostenverlauf der Kostenfunktion qItself#14 zu diesen Zeitschritten einen Anstieg, welcher direkt danach wieder abfällt. Weiter kann man sehen, dass der Equality-Constraint LinAngVel-2-box seinen einzigen Strafausschlag bei Zeitschritt 160 besitzt. Dieser liegt mit $1,6$ auch weit über den tollerierten Strafwerten. Interessant an diesem Sprung ist, dass er im Cluster View nicht auftaucht. Folglich muss der Equality-Constraint der Einzige sein, der zu diesem Zeitschritt einen signifikanten Sprung aufweist. Um einen genaueren Einblick zu bekommen, lohnt sich der Blick in die letzte Ansicht (Sensitivity Analysis View, s. Abb. 10): Neben dem gerade genannten Equality-Constraint haben auch noch die beiden Constraints ZeroQVel-box und AboveBox:target:box relativ hohe Sensitivitätswerte zum Zeitschritt 160. Alle drei Constraints definieren Regeln für das Abstellen der Box beziehungsweise deren Endposition. Es muss also bei dem Platziern der Box an der Endposition ebenfalls etwas nicht wie geplant funktioniert haben.

Fasst man die bislang erlangten Informatioenen zusammen und rekonstruiert basierend auf diesen Informationen, wie die Lösung ungefähr aussehen muss, so würde man grob zu folgendem Ergebnis kommen:

Zeitschritt(e)	Was ist vermutlich in der Lösung passiert
0-38	Der Roboter bewegt seien Arm zu dem neben ihm liegenden Stab
39	Der Roboter berührt/greift den Stab
40-78	Der Roboter führt den Stab zum Ring über der Box
79	Kontakt Stab - Box
80-95	Bewegen der Box (Richtung Tisch)
96-99	Kollision Box - Tisch
100-159	Bewegen der Box Richtung Endposition
160	Abstellen der Box

Die andauernd hohen Strafen des Constraints NewtonEuler_DampedVelocites-box legen zusätzlich nahe, dass die Box immer recht schnell bewegt worden ist.
Und tatsächlich bestätigen sich die Vermutungen, die man aufgrund der Untersuchung der Strafen und Kosten machen konnte. Der Roboter führt in der Lösung bis zum Zeitschritt 39 seinen Arm zu dem Stab, greift ihn und nutzt ihn, um die Box zu bewegen. Die erste Berührung der Box (mit dem Stab) findet zum Zeitschritt 79 statt. Im folgenden fährt der Roboter eine Bahn ab, die vielleicht am besten mit einem liegenden S beschrieben werden kann. Er hebt zuerst die Box an und drückt diese danach wieder nach unten (Richtung Tisch, wo es dann auch zu Kollision kommt). Nach der Kollision passier noch einmal Ähnliches auf der zweiten Hälfte der S-Bewegungsbahn. Er drückt die Box wieder nach oben und schafft es, sie an der gewünschten Endposition zu platzieren. Die komplette S-Bewegung verläuft relativ schnell.

Wie also zu sehen ist, ist die grobe Rekonstruktion der Lösung alleine durch das Vorwissen, was ungefähr geschehen soll, und der Untersuchung der Kosten und Strafen möglich. Aber damit sind die Möglichkeiten des Tools noch nicht erschöpft. Die Fehlersuche beziehungsweise das Finden des Constraints, welcher zu dem unerwünschten Verhalten des Roboters geführt hat steht noch aus. Da schon bei der ersten Berührung der Box mit dem Stab hohe Strafen aufgetreten sind und das Bewegen der Box relativ umständlich gelöst wurde, liegt der Fehler wahrscheinlich an einem Constraint, der eine Regel für die Interaktion des Stabs mit der Box oder für die Interaktion des Stabs mit dem Roboter (z.B: das Greifen des Stabs), festlegt. Auch die Inkaufnahme der Kollision zu den Zeitschritten 96 bis 98 spricht für diese Vermutung. Und wie anfangs erwähnt, wurde dieses Beispiel extra manipuliert, das heißt, es ist bekannt welcher Constraint umformuliert wurde. Bei der (Um-)Formulierung des Problems wurde der Constraint, der die Interaktion (die Physik) des Stabs mit dem Ring auf der Box definiert, umgeschrieben. Er wurde so umgeschrieben, dass bei der Berührung der Rings mit dem Stab so viel Reibung entsteht, dass der Stab nicht durch den Ring gesteckt werden muss, um die Box anzuheben, sondern dass eine kleine Kontaktfläche dafür schon ausreicht.

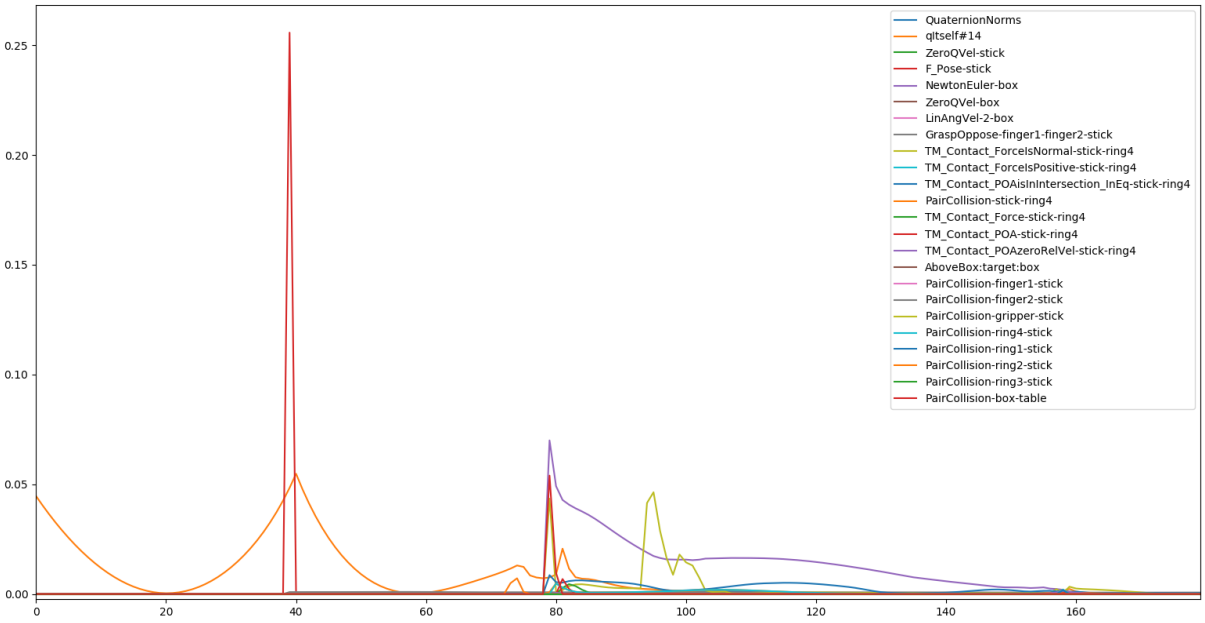

Abbildung 6: Alle Kosten und Strafen

Ein Plot über alle entstandenen Kosten/Strafen über die gesamte Roboter-Trajektorie. Die Höhe der Kosten/Strafen ist auf der y-Achse abzulesen, die Position/Konfiguration in der Trajektorie auf der x-Achse.

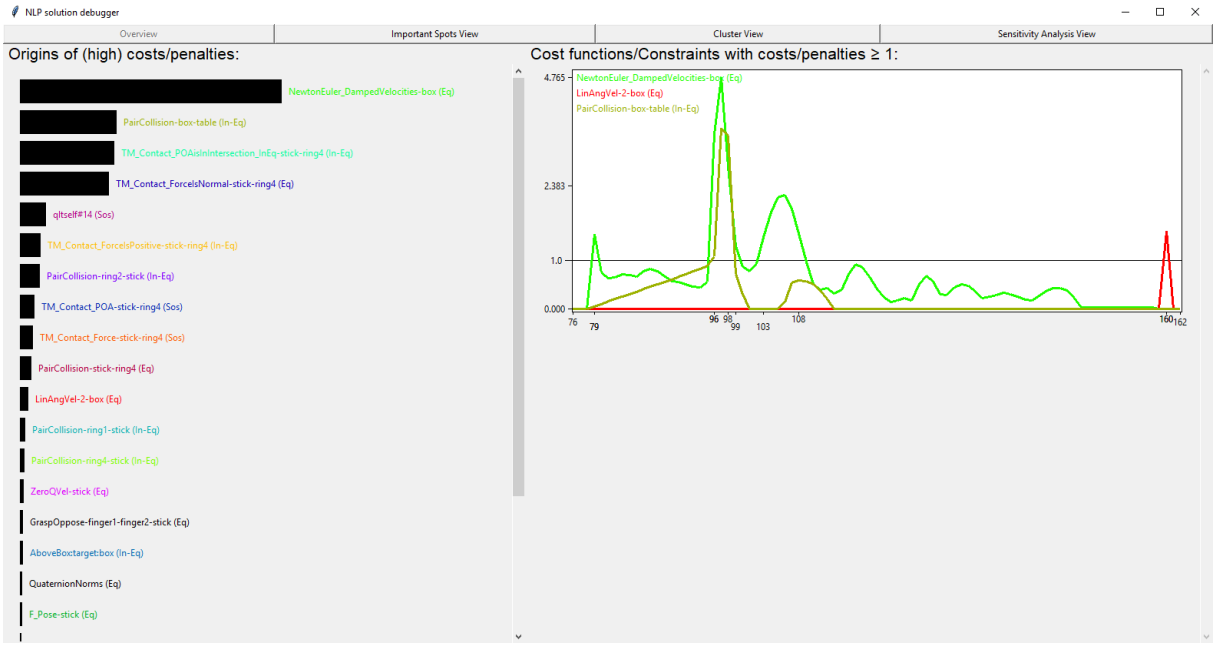

Abbildung 7: Screenshot 1 Debuggingtool

Ein Screenshot des Debuggingtools, auf dem der Overview zu sehen ist. Links das Balkendiagramm und rechts der Plot der Kostenfunktionen und Constraints, die Kosten/Strafen ≥ 1 verursacht haben.

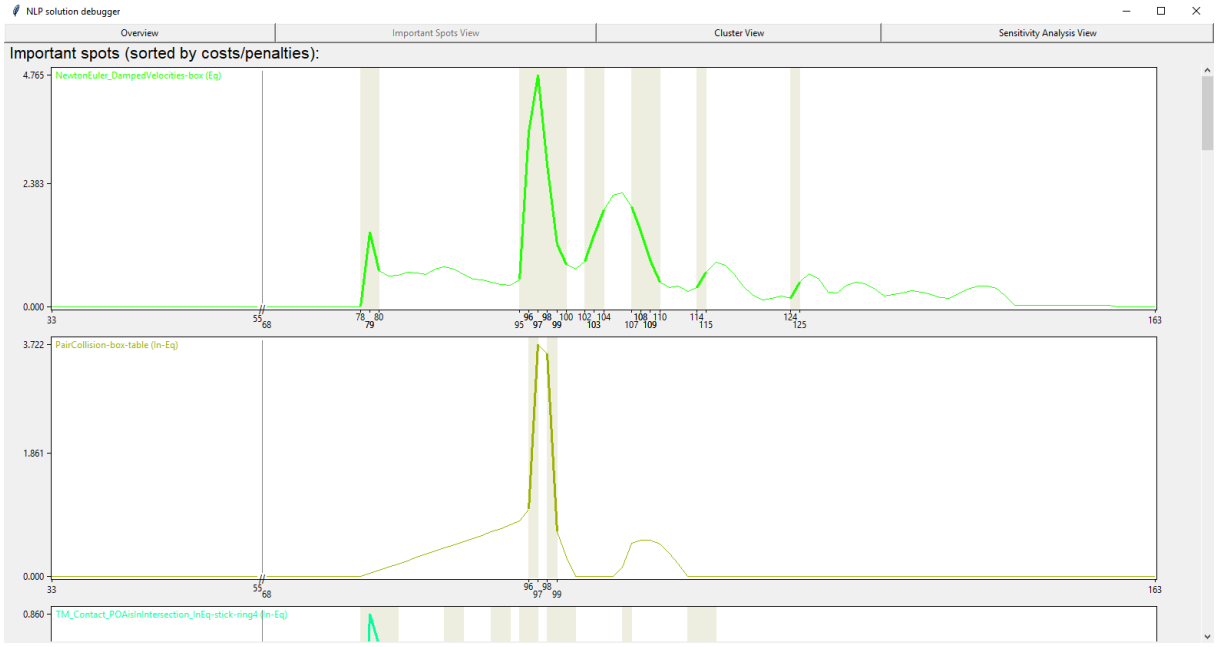

Abbildung 8: Screenshot 2 Debuggingtool

Ein Screenshot des Debuggingtools, auf welchem der Important Spots View zu sehen ist.

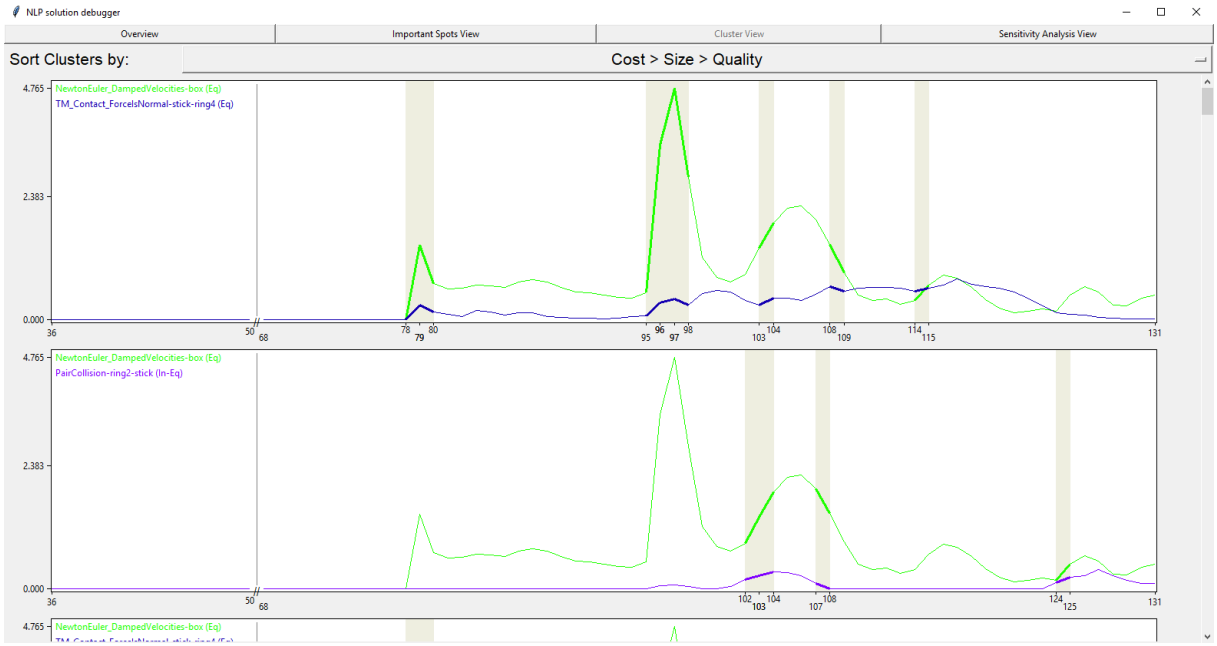

Abbildung 9: Screenshot 3 Debuggingtool

Ein Screenshot des Debuggingtools, auf welchem der Cluster View zu sehen ist.

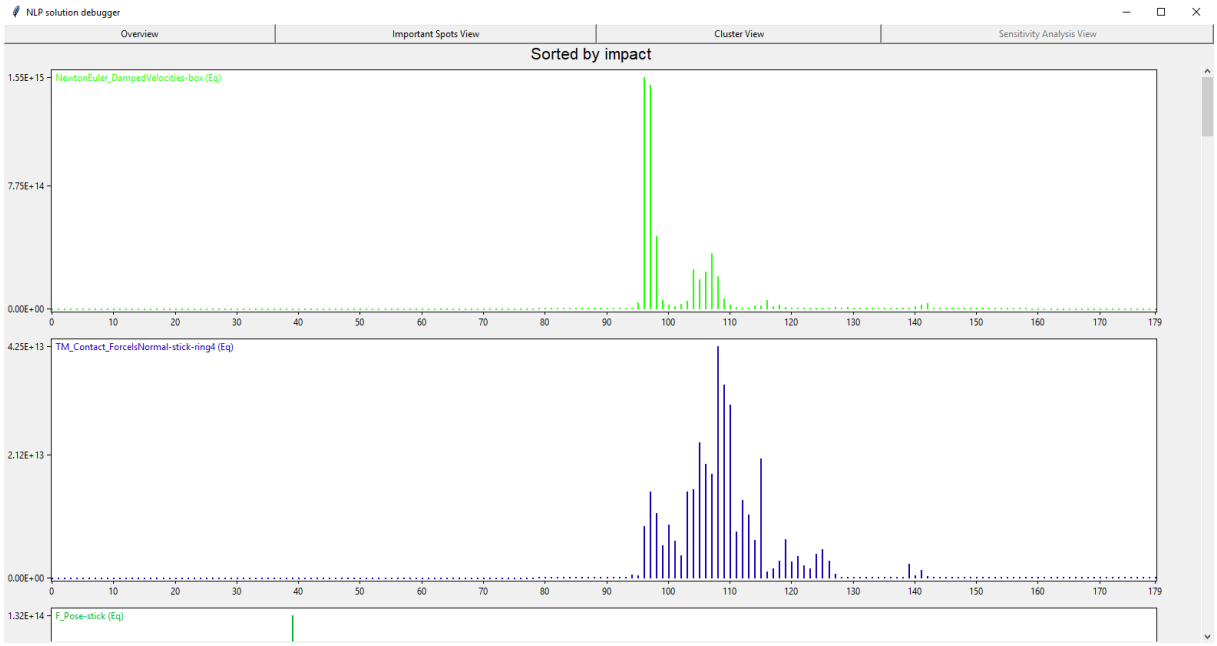

Abbildung 10: Screenshot 4 Debuggingtool

Ein Screenshot des Debuggingtools, auf welchem die Ansicht der Sensitivitätsanalyse zu sehen ist.

4 Zusammenfassung und Ausblick

Das Optimieren von Optimierungsproblemen mit Nebenbedingungen ist recht komplex und greift deshalb gerne auf numerische Verfahren zurück, die das Problem simplifizieren, zum Beispiel durch eine lineare Approximation. Mindestens genauso komplex ist auch das Verstehen einer Lösung, die ein Optimierer hervorbringt.

Das mit dieser Arbeit entwickelte Tool stellt für das Verstehen einer solchen Lösung ein Hilfsmittel dar. Es gibt dem Nutzer eine Auswahl an interessanten Punkten und Regionen der Lösung um Schritt für Schritt zu ergründen, warum der Optimierer zu dieser Lösung gekommen ist. Es werden die Gesamtkosten und Strafen in ihre Einzelteile zerlegt und somit sichtbar gemacht, wo schwere Konfigurationen der Lösung liegen und welche Constraints und Kostenfunktionen daran beteiligt sind. Zusätzlich zeigt die Sensitivitätsanalyse auf, wie groß der Einfluss einzelner Features auf die Lösung ist. Selbst eine grobe Rekonstruktion der Lösung ist ohne das Betrachten der Lösung (in Form eines Videos) mit Hilfe des Tools möglich.

Alles in Allem erlaubt das Tool mit seinen vorgestellten Untersuchungsmöglichkeiten dem Grund, warum die Lösung so aussieht, wie sie aussieht näher zu kommen und ein Gefühl dafür zu bekommen, wie stark ein Constraint die Lösung beeinflusst. Letzteres sind genau die Informationen, die essenziell sind, um eine Lösung erklären und verstehen zu können.

Da das Gebiet der Lösungsuntersuchung noch recht neu ist, ist noch viel Arbeit und Forschung notwendig um bessere Tools und neue Untersuchungsmöglichkeiten zu erarbeiten. Diese Arbeit ist ein erster Schritt um grundsätzlich zu schauen, welche (naheliegenden) Möglichkeiten der Untersuchung möglich sind. Welche Methode sich als die beste Methode herauskristallisiert oder ob die Kombination der Methoden ein wichtiger Bestandteil der Lösungsuntersuchung ist, wird sich herausstellen müssen. Eventuell ist es auch hilfreich, sich den gesamten Optimierungsprozess anzuschauen um zu verstehen, wie die gefundene Lösung entstanden ist. Hierdurch können vielleicht neue Informationen gewonnen werden, die Aufschluss über die Frage geben können, warum die gefundene Lösung die beste Lösung bei der vorgegebenen Problemformulierung ist.

Literatur

[1] Git Repo. https://animal.informatik.uni-stuttgart.de/student-projects/19-wimpff/wikis/home, 05.11.2019.

[2] Website. http://rll.berkeley.edu/trajopt/doc/sphinx_build/html/; abgerufen am 29.10.2019.

[3] Website. Online erhältlich unter http://www.math.uni-bremen.de/zetem/cms/media.php/309/bscarbeit_renke_schaefer.pdf; abgerufen am 27.10.2019.

[4] Website. Online erhältlich unter https://www.math.uni-hamburg.de/home/lauterbach/tuhh/analysisIII/folien5.pdf; abgerufen am 23.10.2019.

[5] Website. https://de.wikipedia.org/wiki/Satz_von_der_impliziten_Funktion; abgerufen am 29.10.2019.

[6] Backtracking line search. Website. https://en.wikipedia.org/wiki/Backtracking_line_search; abgerufen am 25.10.2019.

[7] Fritz-john-bedingungen. Website. https://de.wikipedia.org/wiki/Fritz-John-Bedingungen; abgerufen am 03.11.2019.

[8] Slater-bedingung. Website. https://de.wikipedia.org/wiki/Slater-Bedingung; abgerufen am 03.11.2019.

[9] R Andreani, M Martínez, and L Schuverdt. On second-order optimality conditions for nonlinear programming. *Optimization*, 56(5–6):529–542, Oct 2007. dummy note.

[10] RODRIGO G Eustaquio, ELIZABETH W Karas, and ADEMIR A Ribeiro. Constraint qualifications for nonlinear programming. *Federal University of Parana*, 2008.

[11] Robert M. Freund. Penalty and barrier methods for constrained optimization. Website, 2004. Online erhältlich unter https://pdfs.semanticscholar.org/a187/a63069e2478344cb1fe6aa71c849aecc3fcc.pdf; abgerufen am 23.10.2019.

[12] Ryan Tibshirani Geoff Gordon. Karush-kuhn-tucker conditions. Website. Online erhältlich unter http://nlp.chonbuk.ac.kr/PGM/slides_other/16-kkt.pdf; abgerufen am 21.10.2019.

[13] Matthias Gerdts. Optimierung. Website. Online erhältlich unter https://www.unibw.de/ingmathe/teaching/optimierung_me_msc.pdf/download; abgerufen am 23.10.2019.

[14] Olvi L Mangasarian and Stanley Fromovitz. The fritz john necessary optimality conditions in the presence of equality and inequality constraints. *Journal of Mathematical Analysis and applications*, 17(1):37–47, 1967.

[15] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. 2009.

[16] M. Solodov. *Constraint Qualifications*. 02 2010.

[17] Marc Toussaint. Introduction to optimization. Website, 2016. Online erhältlich unter `https://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/Lecture-Optimization.pdf`; abgerufen am 22.10.2019.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Datum Unterschrift