

VORLESUNGSSKRIPT

OPTIMIERUNG FÜR NICHTMATHEMATIKER

WINTERSEMESTER 2020/21
TU CHEMNITZ

Roland Herzog*

2021-01-11

*Technische Universität Chemnitz, Fakultät für Mathematik, 09107 Chemnitz
(roland.herzog@mathematik.tu-chemnitz.de, https://www.tu-chemnitz.de/mathematik/part_dgl/people/herzog).

Dieses Vorlesungsskript enthält Material u. a. aus den Büchern [Nocedal, Wright, 2006](#); [Alt, 2002](#); [Boyd, Vandenberghe, 2004](#); [Gerds, Lempio, 2011](#) sowie aus einer früheren Vorlesung von Prof. Dr. Christoph Helmberg, TU Chemnitz. Das Skript wurde im WS2012/13 von Prof. Dr. Roland Herzog erstellt und im WS2013/14 von Prof. Dr. Christoph Helmberg überarbeitet. Es wird ständig aktualisiert.

Material für: 15 Vorlesungen à 90 Minuten

Fehler und Kommentare bitte an: roland.herzog@mathematik.tu-chemnitz.de

Die verwendete MATLAB-Version in dieser Vorlesung ist: R2020b.

Inhaltsverzeichnis

o. Einführung	5
§ 1 Einführung	5
1. Freie Optimierung	9
§ 2 Ein erstes Beispiel	9
§ 3 Das Verfahren hinter <code>fminunc</code>	17
§ 4 Bereitstellen von Ableitungen	25
§ 5 Weitere Informationen zu <code>fminunc</code>	28
§ 6 Umwandlung in freie Optimierungsaufgaben	30
2. Parameteridentifikation und KQ-Aufgaben	35
§ 7 Ein erstes Beispiel	35
§ 8 Das Verfahren hinter <code>lsqcurvefit</code> und <code>lsqnonlin</code>	41
§ 9 Bereitstellen von Ableitungen	49
§ 10 Weitere Kleinste-Quadrate-Aufgaben	51
3. Netzwerk- und Transportprobleme	57
§ 11 Ein erstes Beispiel	57
§ 12 Das Verfahren hinter <code>linprog</code>	66
§ 13 Lagrange-Multipl., duales Simplex-V. und Schattenpreise	76
§ 14 Probleme des kostenminimalen Transports auf Graphen	86
§ 15 Ganzzahlige Lösungen	95
§ 16 Weitere Optimierungsaufgaben auf Graphen	97
4. Jetzt wird's beschränkt	114
§ 17 Ein erstes Beispiel	114
§ 18 Das Verfahren hinter <code>fmincon</code>	135
A. Mathematische Grundlagen	139
§ 19 Mengen	139
§ 20 Vektoren	139
§ 21 Matrizen	141

Kapitel 0 Einführung

Wir beginnen mit einem Zitat aus Nocedal, Wright, 2006:

People optimize. Investors seek to create portfolios that avoid excessive risk while achieving a high rate of return. Manufacturers aim for maximum efficiency in the design and operation of their production processes. Engineers adjust parameters to optimize the performance of their designs.

Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

§ 1 EINFÜHRUNG

Gegenstand dieser Vorlesung sind Optimierungsaufgaben der folgenden Gestalt:

$$\begin{aligned} \text{Minimiere} \quad & f(x), & x \in \Omega, \\ \text{sodass} \quad & g_i(x) \leq 0, & i = 1, \dots, m, \\ \text{und} \quad & h_j(x) = 0, & j = 1, \dots, p. \end{aligned}$$

Dabei geht es darum, eine Belegung gewisser **Optimierungsgrößen** oder **Entscheidungsvariablen** (kurz: **Variablen**) x zu finden, sodass die **Zielfunktion** oder **Kostenfunktion** $f(x)$ minimiert wird. Die Zielfunktion legt also fest, welche Entscheidungen besser und welche schlechter sind. Statt „Minimiere $f(x)$ “ schreibt man manchmal auch $f(x) \rightarrow \min$.

Die Variablen x sollen dabei einer **Grundmenge** Ω angehören. Dies wird bei uns bis auf wenige Ausnahmen (in Kapitel 3) immer

$$\Omega = \mathbb{R}^n$$

sein. Unsere Entscheidungen bestehen also aus n Einzelentscheidungen, die zu einem Vektor $x \in \mathbb{R}^n$ zusammengefasst werden.

Wichtiger Bestandteil einer Optimierungsaufgabe sind i. d. R. **Nebenbedingungen** (auch **Beschränkungen** oder **Restriktionen** genannt), die die Variablen x zu erfüllen haben. Diese können die Form von Ungleichungen $g_i(x) \leq 0$ bzw. von Gleichungen $h_j(x) = 0$ haben.

Es ist im Übrigen keine Einschränkung, dass wir uns nur mit *Minimierungsaufgaben* befassen: Sollte in einer Aufgabe die *Maximierung* von $f(x)$ gefragt sein, so können wir stattdessen die Funktion $-f(x)$ minimieren, siehe Abb. 1.1. Ganz analog können wir Ungleichungen der Form $g_i(x) \geq 0$ durch Multiplikation mit (-1) auf die Form $-g_i(x) \leq 0$ bringen.

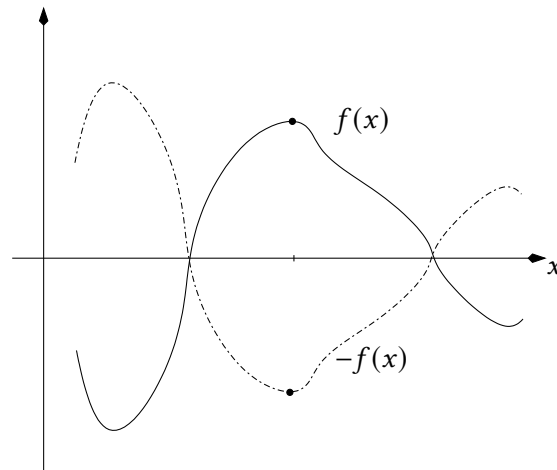


Abbildung 1.1.: Statt eine Funktion $f(x)$ (durchgezogene Linie) zu maximieren, können wir auch $-f(x)$ minimieren (gestrichelt).

Wir geben ein erstes Beispiel einer Optimierungsaufgabe:

Beispiel 1.1 (Transportproblem¹). Eine Firma besitzt zwei Fabriken F_1, F_2 und zwölf Verkaufsstellen $V_1, \dots, V_{12}$. Jede Fabrik F_i produziert pro Woche die Menge p_i eines bestimmten Produktes. Jede Verkaufsstelle V_j hat eine bekannte wöchentliche Nachfrage n_j nach diesem Produkt, die gedeckt werden muss. Die Kosten, eine Einheit des Produkts von Fabrik F_i zur Verkaufsstelle V_j zu transportieren, seien c_{ij} , und die von F_i nach V_j zu transportierende Menge sei x_{ij} . (Dies sind die Entscheidungsvariablen.) Wie muss die Produktion auf die Verkaufsstellen verteilt werden, um die Gesamtkosten des Transports zu minimieren und die Nachfrage zu decken?

$$\left. \begin{array}{ll} \text{Minimiere} & f(x) := \sum_{i=1}^2 \sum_{j=1}^{12} c_{ij} x_{ij} \quad (\text{Transportkosten}), \quad x \in \mathbb{R}^{2 \times 12} \\ \text{sodass} & \sum_{j=1}^{12} x_{ij} \leq p_i, \quad i = 1, 2 \quad (\text{Produktionsrestriktionen}) \\ \text{und} & \sum_{i=1}^2 x_{ij} \geq n_j, \quad j = 1, \dots, 12 \quad (\text{Bedarfsdeckung}) \\ \text{sowie} & x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \dots, 12. \end{array} \right\} \quad (1.1)$$

Diese Aufgabe hat folgende Merkmale:

¹aus Nocedal, Wright, 2006, S. 4

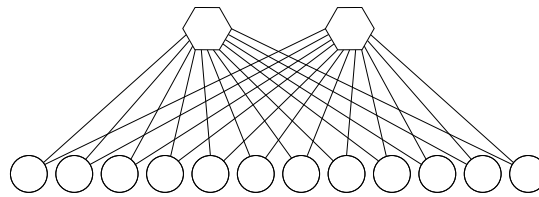


Abbildung 1.2.: Schematische Darstellung des Transportproblems ([Beispiel 1.1](#)). Die Transportwege zwischen den zwei Fabriken (oben) und den zwölf Verkaufsstellen (unten) sind als Kanten dargestellt. Gesucht sind die optimalen Transportmengen für alle 24 Kanten.

- (1) Die Optimierungsvariable x wurde als „Tabelle“, d. h. als Matrix $x \in \mathbb{R}^{2 \times 12}$ statt als Vektor $x \in \mathbb{R}^{24}$ angelegt. Dies ist aus Gründen der Modellierung praktisch. Zur Lösung der Aufgabe mit Hilfe entsprechender Software muss x aber i. d. R. als Vektor geschrieben werden. Dies zieht eine (einfache) Umformulierung der Zielfunktion und der Nebenbedingungen nach sich.
- (2) Die Zielfunktion und alle Nebenbedingungen sind (affin-)lineare² Funktionen der Entscheidungsvariablen x . Es handelt sich damit um eine **lineare Optimierungsaufgabe**, auch **lineares Programm (LP)** genannt, die in [Kapitel 3](#) besprochen werden.³
- (3) Die Ungleichungsnebenbedingungen sind ein wesentlicher Bestandteil der Aufgabenstellung ([1.1](#)). Lässt man etwa die Beschränkungen für die Bedarfsdeckung weg, so wird sich als Lösung $x_{ij} = 0$ für alle $i = 1, \dots, 2$, $j = 1, \dots, 12$ ergeben, also überhaupt kein Transport, der ja auch keine Kosten verursacht.
- (4) Die Aufgabe ([1.1](#)) hat keine Gleichungsnebenbedingungen.
- (5) Die Größen c_{ij} , p_i und n_j werden als bekannte Problemdata (Parameter) angenommen. Wenn wir die Aufgabe ([1.1](#)) lösen wollen, müssen wir konkrete Werte für diese Parameter vorgeben. Mit konkreten Werten spricht man von einer **Instanz** dieses Problems. Ändern wir die Parameter, so beschreibt das eine andere Instanz, und es wird sich i. A. auch die Lösung der Aufgabe ändern.

AUFBAU UND ZIELE DER LEHRVERANSTALTUNG

In dieser Lehrveranstaltung sollen Sie lernen,

- Optimierungsaufgaben zu modellieren,

²Genau genommen ist eine Funktion wie $f(x) = 2x$ linear, aber $g(x) = 2x + 1$ ist es nicht, sondern diese ist affin oder (affin-)linear (linear plus Konstante). In der Optimierung ist es jedoch üblich, auch affine Funktionen als linear zu bezeichnen.

³Eine vollständige formale Klassifikation von Optimierungsaufgaben nehmen wir in diesem Skript nicht vor. Die einzelnen Kapitel behandeln immer jeweils eine Problemklasse. Durch die Beschäftigung mit den Beispielen und Übungsaufgaben sollten im Anschluss an die Lehrveranstaltung die besprochenen Aufgabentypen wiedererkannt werden können. Hilfe zu einer viel weitergehenden Klassifikation von Optimierungsaufgaben und geeigneter Software zu deren Lösung (außerhalb der MATLAB-Optimierungs-Toolbox) bietet der [NEOS Solver Software Guide](#).

- diese mit geeigneter Optimierungssoftware zu lösen
- und die Lösungen zu überprüfen und zu interpretieren.

Dazu ist ein gewisses Grundverständnis der Theorie und gängiger Algorithmen für wichtige Problemklassen unerlässlich, das ebenfalls vermittelt wird.

Die Vorlesung arbeitet mit zahlreichen Fallbeispielen und ist nach Aufgabentypen gegliedert. Das richtige Erkennen des Typs einer Optimierungsaufgabe ist für die Auswahl geeigneter Algorithmen wichtig. Wir arbeiten in dieser Lehrveranstaltung mit den Algorithmen aus der Optimierungstoolbox von MATLAB. Diese Plattform bietet einen komfortablen und intuitiven Zugang zu vielen einsatzbereiten Verfahren.

Die Arbeit mit MATLAB können und sollen Sie im Rahmen dieser Lehrveranstaltung lernen. Neben dem Angebot betreuter Übungen am Computer können Sie auch die unter https://www.tu-chemnitz.de/mathematik/part_dgl/teaching/material/index.de.php#Matlab verlinkten Materialien verwenden, um Ihre MATLAB-Kenntnisse auszubauen. In der Vorlesung und Übung verwenden wir die MATLAB-Kommandozeile bzw. die grafische Oberfläche (GUI) von MATLAB sowie auch JUPYTER-Notebooks.

Die wichtigsten Grundlagen der Analysis und linearen Algebra sind im Anhang kompakt zusammengefasst.

Ende der Woche 1

Kapitel 1 Aller Anfang ist leicht (Freie Optimierung)

Obwohl die allermeisten praktischen Aufgabenstellungen erst durch Beschränkungen sinnvoll werden, geht es in diesem Kapitel zunächst um Optimierungsaufgaben ohne Beschränkungen:

$$\text{Minimiere } f(x), \quad x \in \mathbb{R}^n.$$

Weiterhin gehen wir davon aus, dass $f(x)$ eine glatte Funktion ist, genauer, dass $f(x)$ zweimal stetig differenzierbar ist.

§ 2 EIN ERSTES BEISPIEL

Beispiel 2.1 (Isolierung beim Hausbau). *Bei der Planung eines Wohnhauses soll die Wärmeisolierung ausgelegt werden. Ein Teil der Außenwand wird mit Material 1 gedämmt, der Rest mit Material 2. Die Wandstärken d_1 und d_2 können noch gewählt werden. Die Materialkosten sind dabei proportional zu d_1 bzw. d_2 . Der Gesamtwärmewiderstand des Hauses ist (wie bei der Parallelschaltung elektrischer Widerstände) proportional zu $(1/d_1 + 1/d_2)^{-1}$. Die Heizkosten sind demnach proportional zur Wärmeleitfähigkeit $(1/d_1 + 1/d_2)$.*

Der Wunsch nach der Minimierung der Summe aus Investitions- und Heizkosten (etwa innerhalb einer Spanne von 10 Jahren) wird durch die Zielfunktion

$$K(d_1, d_2) = \underbrace{c_0 \left(\frac{1}{d_1} + \frac{1}{d_2} \right)}_{\text{Heizkosten}} + \underbrace{c_1 d_1 + c_2 d_2}_{\text{Investitionskosten}} \quad (2.1)$$

ausgedrückt. Dabei sind die Konstanten als

$$c_0 = 20\,000 \text{ € cm}, \quad c_1 = 300 \frac{\text{€}}{\text{cm}}, \quad c_2 = 400 \frac{\text{€}}{\text{cm}}$$

gegeben. Bestimmen Sie die optimalen Wandstärken.¹

Wir geben nun an, wie man diese Aufgabe mit Hilfe des Löser `fminunc` für unbeschränkte (unconstrained) Optimierungsaufgaben² aus der MATLAB-Optimierungs-Toolbox lösen kann.

¹Die Implementierung wird in der Übung wiederholt. In der Übung wird die Lösung zudem zum Vergleich auch analytisch berechnet.

²Natürlich ist die Zielfunktion dort, wo $d_1 = 0$ oder $d_2 = 0$ ist, überhaupt nicht definiert. Für den Bereich $d_1 < 0$ oder $d_2 < 0$ ist sie zwar definiert, aber negative Wanddicken sind natürlich nicht sinnvoll. Dennoch können wir hier ohne Beschränkungen an d_1 und d_2 auskommen und das Beispiel als eine unbeschränkte Optimierungsaufgabe behandeln.

Dazu programmieren wir zunächst die Zielfunktion $K(d_1, d_2)$. Die Datei, nennen wir sie `Hausbau_Zielfunktion.m`, sollte etwa wie folgt aussehen:

```
function K = Hausbau_Zielfunktion(x,c0,c1,c2)

% Bedeutung der Variablen
% x(1) = d1 = Wandstaerke Material 1 [cm]
% x(2) = d2 = Wandstaerke Material 2 [cm]
d1 = x(1);
d2 = x(2);

% Berechnung der Zielfunktion
K = c0 * (1/d1 + 1/d2) + c1 * d1 + c2 * d2;
```

Zu beachten ist dabei, dass wir die Entscheidungsvariablen zu einem Vektor

$$x = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

zusammengefasst haben, da die Optimierungsverfahren in MATLAB stets nur mit *einer* Optimierungsvariablen (die aber ein Vektor des $\mathbb{R}^n$ sein darf) arbeiten. Außerdem bekommt die Zielfunktion die Werte der Parameter c_0 , c_1 und c_2 mitgeteilt. Diese sind keine Optimierungsvariablen, sondern Konstanten mit festen Werten.

Nun schreiben wir unser Hauptprogramm in eine Datei, etwa mit dem Namen `Hausbau.m`. Alternativ können die folgenden Zeilen auch direkt im MATLAB-Kommandofenster eingegeben werden. Zunächst werden die Problemparameter gesetzt (damit wird das Problem instantiiert):

```
% Setzen der Problemparameter
c0 = 20000; % Heizkostenfaktor [EUR * cm]
c1 = 300; % Kosten pro Wandstaerke Material 1 [EUR / cm]
c2 = 400; % Kosten pro Wandstaerke Material 2 [EUR / cm]
```

Nun wird die Zielfunktion gesetzt. Wir bezeichnen sie hier mit $f(x)$:

```
% Setzen der Zielfunktion
f = @(x) Hausbau_Zielfunktion(x,c0,c1,c2);
```

Dabei haben wir die Programmiertechnik einer sogenannten *anonymen Funktion* verwendet. Kurz gesagt ist nun $f(x)$ unsere Zielfunktion, wie wir sie in `Hausbau_Zielfunktion.m` implementiert haben, wobei jedoch die zum Zeitpunkt der Definition aktuellen Werte von c_0 , c_1 und c_2 fest eingesetzt werden. Dies ist eine elegante Möglichkeit, einer Funktion zusätzliche Parameter zu übergeben.³ Wir überprüfen an einem Beispiel, ob unsere Zielfunktion das Richtige tut. Bei $x = (d_1, d_2) = (10, 20)$ müsste sich (ohne Einheiten) ein Funktionswert von

$$K(10, 20) = 20\,000 \left(\frac{1}{10} + \frac{1}{20} \right) + 300 \cdot 10 + 400 \cdot 20 = 14\,000 :$$

ergeben:

³Eine andere Möglichkeit besteht in der Verwendung globaler Variablen.

```
% % Ueberpruefen der Zielfunktion
% x = [10;20];
% f(x)
```

Wir legen noch eine Startschätzung $x^{(0)}$ fest, hier $d_1 = 5$ und $d_2 = 5$, also $x^{(0)} = (5, 5)$:

Matlab/Hausbau.m

```
% Setzen einer Startschätzung
% Bedeutung der Variablen
% x(1) = d1 = Wandstaerke Material 1 [cm]
% x(2) = d2 = Wandstaerke Material 2 [cm]
x0 = [5;5];
```

Schließlich setzen wir noch einige Optionen für das Optimierungsverfahren. Mit diesen wählen wir einen von zwei in `fminunc`⁴ verfügbaren Algorithmen aus. Außerdem fordern wir einen detaillierten Bericht über den Fortschritt des Verfahrens an:

Matlab/Hausbau.m

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('fminunc','Algorithm','quasi-newton','Display',
    'iter-detailed');

% Aufruf des Optimierungsverfahrens
[xstar,fstar] = fminunc(f,x0,options);
```

Die Ausgabe sollte wie folgt aussehen:

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	11500		500
1	6	10901.6	0.002	256
2	9	10625.3	1	103
3	12	10563.9	1	35
4	15	10556.4	1	9.03
5	18	10555.9	1	4.04
6	21	10555.9	1	2.11
7	24	10555.8	1	0.36
8	27	10555.8	1	0.0536
9	30	10555.8	1	0.00284
10	33	10555.8	1	0.000121

Optimization completed: The first-order optimality measure, 2.412045e-07, is less than options.OptimalityTolerance = 1.000000e-06.

Optimization Metric		Options	
relative norm(gradient) =	2.41e-07	OptimalityTolerance =	1e-06 (default)

Diese Ausgabe werden wir weiter unten interpretieren.

⁴In MATLAB erhalten Sie die Dokumentation mittels `doc fminunc`.

Die von `fminunc` berechnete Lösung steht jetzt in der Variablen `xstar`:⁵

$$x^* = \begin{pmatrix} 8.1650 \\ 7.0711 \end{pmatrix}.$$

In der Notation der Aufgabe sind dies die berechneten Werte für die Wandstärken, also $d_1 = 8.1650$ und $d_2 = 7.0711$, jeweils in cm.⁶

Frage: Wie können wir überprüfen, ob dies tatsächlich eine Lösung der Aufgabe ist?

Definition 2.2 (Minimum in der freien Optimierung). *Wir betrachten eine freie (unrestringierte) Optimierungsaufgabe*

$$\text{Minimiere } f(x), \quad x \in \mathbb{R}^n. \quad (2.2)$$

(i) Ein Vektor $x^* \in \mathbb{R}^n$ heißt ein **globales Minimum**, wenn gilt:

$$f(x^*) \leq f(x) \quad \text{für alle } x \in \mathbb{R}^n.$$

(ii) Ein globales Minimum x^* heißt **strikt**, wenn gilt:⁷

$$f(x^*) < f(x) \quad \text{für alle } x \in \mathbb{R}^n, \quad x \neq x^*.$$

(iii) Ein Vektor $x^* \in \mathbb{R}^n$ heißt ein **lokales Minimum**, wenn es eine Kugel (Umgebung) $U(x^*)$ mit Mittelpunkt x^* gibt, sodass gilt:

$$f(x^*) \leq f(x) \quad \text{für alle } x \in U(x^*).$$

(iv) Ein lokales Minimum heißt **strikt**, wenn gilt:

$$f(x^*) < f(x) \quad \text{für alle } x \in U(x^*), \quad x \neq x^*.$$

Ein Minimum nennt man auch **Optimum**, **Minimalstelle** oder **Lösung** von (2.2). Ein globales Minimum ist natürlich auch ein lokales, aber nicht umgekehrt. Mit jeweils umgedrehten Relationszeichen ($\geq$ statt $\leq$ bzw. $>$ statt $<$) definiert man entsprechend die Begriffe (**striktes**) **lokales** bzw. **globales Maximum**.

Zum Erkennen lokaler Minima benutzt man sogenannte Optimalitätsbedingungen.

Satz 2.3 (Notwendige Optimalitätsbedingung 1. Ordnung). *Es sei x^* ein lokales Minimum der Funktion $f(x)$. Weiter sei $f(x)$ stetig partiell diffbar in einer Umgebung $U(x^*)$. Dann ist $\nabla f(x^*) = 0$ (der Nullvektor).*

⁵Wir verwenden – wie auch MATLAB – durchgängig die englische Schreibweise für Zahlen, schreiben also 1.0 statt 1,0.

⁶In der Übung werden wir auch die exakte Lösung per Hand berechnen.

⁷Das globale Minimum ist in diesem Fall natürlich eindeutig.

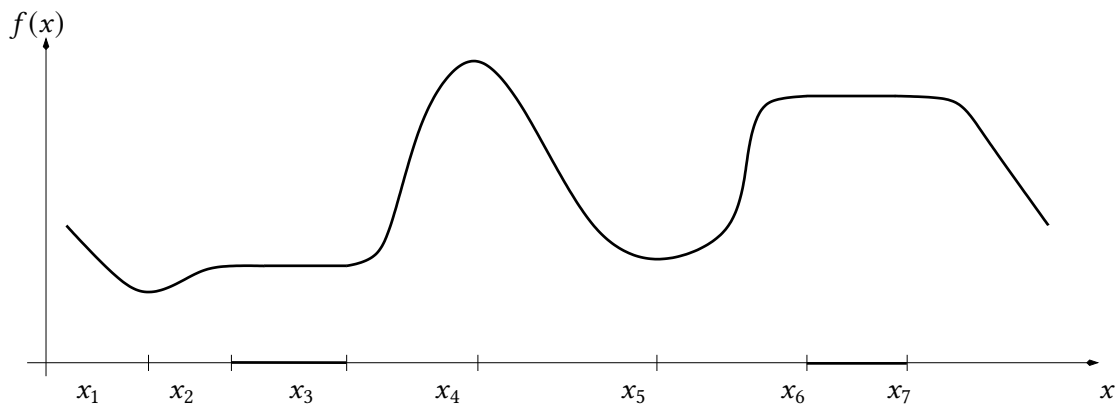


Abbildung 2.1.: Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ mit einigen besonderen Punkten: x_1 : striktes globales Minimum; x_2 : lokales Maximum; x_3 : lokales Minimum; alle Punkte im offenen Intervall (x_2, x_3) : gleichzeitig lokales Minimum und Maximum; x_4 : striktes globales Maximum; x_5 : striktes lokales Minimum; x_6 und x_7 : lokale Maxima; alle Punkte im offenen Intervall (x_6, x_7) : gleichzeitig lokales Minimum und Maximum

Ein Vektor $x \in \mathbb{R}^n$ mit $\nabla f(x) = 0$ heißt ein **stationärer Punkt** von $f(x)$. Der [Satz 2.3](#) sagt also aus, dass ein lokales Minimum immer auch ein stationärer Punkt ist. Allerdings ist längst nicht jeder stationäre Punkt auch ein lokales Minimum; er kann auch ein lokales Maximum sein (wie der Punkt $x = 0$ für die Funktion $f(x) = -x^2$) oder ein Sattelpunkt (wie der Punkt $x = 0$ für die Funktion $f(x) = x^3$).

Wir wollen bei der oben erhaltenen „Lösung“ von [Beispiel 2.1](#) überprüfen, ob dort tatsächlich der Gradient (die Ableitung) gleich dem Nullvektor ist. Dazu rufen wir das Optimierungsverfahren `fminunc` noch einmal mit einer erweiterten Liste von Rückgabeparametern auf, um mehr Informationen zu erhalten:

Matlab/Hausbau.m

```
% Aufruf des Verfahrens mit erweiterten Rueckgabewerten
[xstar, fstar, exitflag, output, grad, hessian] = fminunc(f, x0, options);
```

Die Variable `fstar` enthält den Wert der Zielfunktion $f(x^*)$, hier

$$f(x^*) = 1.0556e+04 = 1.0556 \cdot 10^4 = 10\,556.$$

Weiter enthält `grad` den Wert von $\nabla f(x^*)$ an der Stelle `xstar`:

$$\nabla f(x^*) = 10^{-3} \begin{pmatrix} 0.0598 \\ -0.1208 \end{pmatrix},$$

was ungefähr null (der Nullvektor) ist. Die Euklidische Norm (Länge) dieses Vektors, also $\|\nabla f(x^*)\|$, können wir mittels

```
norm(grad)
```

erfahren, sie beträgt $1.3483 \cdot 10^{-4}$, ist also recht klein. Als Abbruchbedingung verwendet `fminunc` jedoch stattdessen die sogenannte ∞ -Norm $\|\nabla f(x^*)\|_\infty$, das ist betragsmäßig der größte Eintrag des Vektors, also $1.2084 \cdot 10^{-4}$:

```
norm(grad, Inf)
```

Damit hat das Verfahren `fminunc` näherungsweise zumindest einen stationären Punkt der Zielfunktion $f(x)$ ausgerechnet. Wie oben bemerkt, wissen wir jedoch noch nicht, ob wir mit x^* vielleicht statt eines Minimums ein Maximum oder einen Sattelpunkt erwisch haben.

Zur weiteren Unterscheidung verwendet man Bedingungen 2. Ordnung, bei denen es auf die zweite Ableitung von $f(x)$, also auf die Hessematrix $\nabla^2 f(x)$ ankommt.

Satz 2.4 (Notwendige Optimalitätsbedingung 2. Ordnung). *Es sei x^* ein lokales Minimum der Funktion $f(x)$. Weiter sei $f(x)$ zweimal stetig diffbar in einer Umgebung $U(x^*)$. Dann ist die Hessematrix $\nabla^2 f(x^*)$ positiv semidefinit, alle ihre Eigenwerte sind also ≥ 0 .*⁸

Satz 2.5 (Hinreichende Optimalitätsbedingung 2. Ordnung). *Es sei $f(x)$ zweimal stetig diffbar in einer Umgebung $U(x^*)$ eines Punktes x^* , und es gelte:*

- (i) $\nabla f(x^*) = 0$ und
- (ii) $\nabla^2 f(x^*)$ ist positiv definit, alle ihre Eigenwerte sind also > 0 .⁸

Dann ist x^* ein striktes lokales Minimum von $f(x)$.

Mit dem obigen Aufruf von `fminunc` finden wir die Hessematrix $\nabla^2 f(x^*)$ an der Stelle `xstar` in der Variable `hessian`:⁹

$$\nabla^2 f(x^*) = \begin{pmatrix} 73.4578 & 0.0000 \\ 0.0000 & 113.0957 \end{pmatrix}.$$

Deren Eigenwerte sind offenbar beide positiv, dieses lässt sich ansonsten in MATLAB auch leicht mit dem Befehl

```
eig(hessian)
```

überprüfen. Damit greift hier der [Satz 2.5](#), und das Verfahren `fminunc` hat in der Tat mit x^* eine (ungefähre) lokale Minimalstelle für unser [Beispiel 2.1](#) gefunden.

Dies wollen wir schließlich noch grafisch bestätigen, indem wir einen Ausschnitt (des Funktionsgraphen) der Zielfunktion zeichnen. Dazu legen wir zunächst den Zeichenbereich fest, hier $d_1 \in [2, 20]$ und ebenso d_2 :

⁸Die Hessematrix ist eine symmetrische Matrix, also kann man die Definitheit über die Eigenwerte beschreiben, vgl. [Anhang 21](#) im Anhang.

⁹Dass die Hessematrix hier eine Diagonalmatrix ist, ist kein Zufall, sondern liegt an der speziellen Struktur der Zielfunktion in unserer Aufgabe in [Beispiel 2.1](#). Es gilt nämlich $K(d_1, d_2) = K_1(d_1) + K_2(d_2)$ mit $K_i(d_i) = c_0 \frac{1}{d_i} + c_i d_i$ für $i \in \{1, 2\}$. Da auch keine Nebenbedingungen vorliegen, über die d_1 und d_2 noch gekoppelt sein könnten, ist die Aufgabe **separabel**. Wir hätten also auch einfach über d_1 und d_2 getrennt optimieren können. Dies lohnt sich jedoch erst bei Aufgaben mit deutlich mehr als zwei Optimierungsvariablen.

```
% Zu zeichnenden Bereich fuer d1 und d2 festlegen
d1_vector = linspace(2,20,25);
d2_vector = d1_vector;
```

Anschließend berechnen wir eine Tabelle der Funktionswerte (in der Variable z) mittels zweier Schleifen:

Matlab/Hausbau.Zielfunktion_zeichnen.m

```
% Funktionswerte z = K(d1,d2) berechnen
for i1 = 1:length(d1_vector)
    d1 = d1_vector(i1);
    for i2 = 1:length(d2_vector)
        d2 = d2_vector(i2);
        z(i2,i1) = f([d1;d2]);
    end
end
```

Dies geht zwar in MATLAB auch wesentlich eleganter, aber das würde an dieser Stelle zu weit führen.¹⁰

Schließlich wird eine neue Grafikausgabe geöffnet und die berechneten Punkte mittels `surf` als Oberfläche gezeichnet:

Matlab/Hausbau.Zielfunktion_zeichnen.m

```
% Grafik erstellen
figure(1); clf
surf(d1_vector,d2_vector,z,'FaceAlpha',0.3);
xlabel('Dicke d_1');
ylabel('Dicke d_2');
zlabel('Zielfunktionswert K(d_1,d_2)');
title('Zielfunktion K(d_1,d_2) fuer das Hausbau-Beispiel');
```

Um die Lösung als Punkt in dasselbe Bild mit einzuzichnen, müssen wir vorher mit `hold on` festlegen, dass der bestehende Bildinhalt erhalten bleibt:

Matlab/Hausbau.Zielfunktion_zeichnen.m

```
% Loesung zusaetzlich einzeichnen
hold on
plot3(xstar(1),xstar(2),fstar,'ro','MarkerFaceColor','r','MarkerSize',10);
```

Das Ergebnis ist in [Abb. 2.2](#) zu sehen.¹¹ Obwohl wir natürlich nur den festgelegten Ausschnitt der Zielfunktion $K(d_1, d_2)$ sehen können, erscheint es plausibel, dass `fminunc` hier tatsächlich das globale(!) Minimum in [Beispiel 2.1](#) gefunden hat, und es scheint überhaupt keine weiteren lokalen Minima zu geben.

¹⁰Es sei insbesondere bemerkt, dass man `for`-Schleifen in MATLAB aus Geschwindigkeitsgründen weitestgehend vermeiden und stattdessen Vektoroperationen verwenden sollte.

¹¹Natürlich wird es bei mehr als zwei Optimierungsvariablen (also den erst wirklich interessanten Optimierungsaufgaben) nicht mehr möglich sein, die Lösung grafisch zu überprüfen oder diese gar grafisch (also ohne ein Optimierungsverfahren zu benutzen) zu bestimmen.

Dass dies in der Tat der Fall ist, kann man der Zielfunktion (2.1) sogar relativ leicht ansehen: Diese ist nämlich konvex, wie man aufgrund von Abb. 2.2 (zumindest im gezeigten Ausschnitt) schon vermuten und anhand der Hessematrix auch tatsächlich global nachweisen kann

$$\nabla^2 K(d_1, d_2) = 2 c_0 \begin{pmatrix} 1/d_1^3 & 0 \\ 0 & 1/d_2^3 \end{pmatrix}$$

überprüfen kann. Deren Eigenwerte sind im sinnvollen Bereich $d_1 > 0, d_2 > 0$ stets positiv, woraus die (strikte) Konvexität von $K(d_1, d_2)$ auf diesem Bereich folgt, siehe Anhang 21 im Anhang.

Satz 2.6 (Bedeutung konvexer Optimierungsaufgaben). *Es sei $f : \mathbb{R}^n \rightarrow \mathbb{R}$ stetig partiell diffbar und konvex. Dann ist jeder stationäre Punkt x^* (wo also $\nabla f(x^*) = 0$ gilt) bereits ein globales Minimum der Aufgabe (2.2). Insbesondere braucht man also nicht zwischen lokalen und globalen Minima zu unterscheiden.*

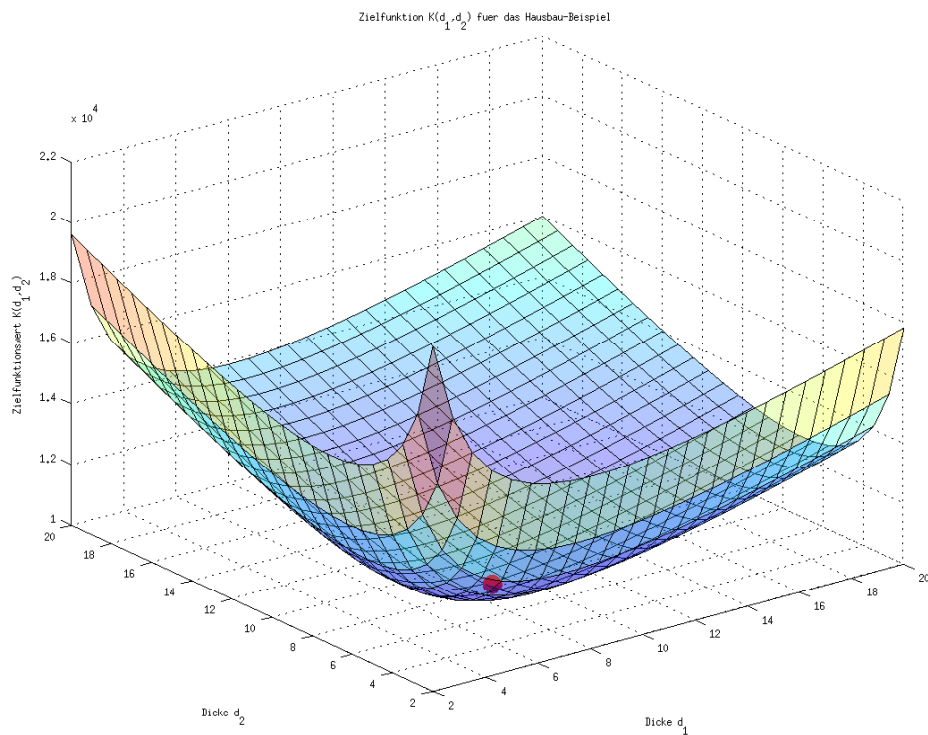


Abbildung 2.2.: Die Zielfunktion $K(d_1, d_2)$ aus Beispiel 2.1 mit eingezeichneter Lösung x^* .

§ 3 DAS VERFAHREN HINTER `fminunc`

Wir wollen nun das Optimierungsverfahren hinter `fminunc` kennenlernen. In der von uns durch Setzen der Option `'Algorithm', 'quasi-newton'` gewählten Variante (früher: `'LargeScale', 'off'`) ist dies ein sogenanntes **Quasi-Newton-Verfahren mit Liniensuche**. Dieses erzeugt eine Folge $\{x^{(k)}\}$ von Vektoren $x^{(k)} \in \mathbb{R}^n$, die i. d. R. gegen ein lokales Minimum der Zielfunktion $f(x)$ konvergiert, wie im Beispiel gesehen. In jedem Schritt (jeder Iteration) des Verfahrens wird

- (1) ein quadratisches Modell $q^{(k)}(d)$ der Zielfunktion aufgebaut,
- (2) das Minimum $d^{(k)} \in \mathbb{R}^n$ dieses Modells bestimmt,
- (3) eine geeignete Schrittweite $\alpha^{(k)} > 0$ gewählt
- (4) und die nächste Iterierte als

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

gesetzt.

Das quadratische Modell (quadratisches Polynom) der Zielfunktion orientiert sich am Taylorpolynom zweiter Ordnung, entwickelt an der Stelle $x^{(k)}$:

$$\begin{aligned} M^{(k)}(x) &:= f(x^{(k)}) && \text{Term nullter Ordnung (konstant)} \\ &+ \nabla f(x^{(k)})^\top (x - x^{(k)}) && \text{Term erster Ordnung (linear)} \\ &+ \frac{1}{2} (x - x^{(k)})^\top \nabla^2 f(x^{(k)}) (x - x^{(k)}) && \text{Term zweiter Ordnung (quadratisch).} \end{aligned}$$

Dieses Modell der Zielfunktion $f(x)$ ist allein durch die Daten

$$f(x^{(k)}) \in \mathbb{R}, \quad \nabla f(x^{(k)}) \in \mathbb{R}^n \quad \text{und} \quad \nabla^2 f(x^{(k)}) \in \mathbb{R}^{n \times n}$$

festgelegt. Es ist das beste quadratische Modell der Funktion $f(x)$ in der Nähe des Entwicklungspunktes $x^{(k)}$. Dabei ist jedoch die Berechnung der Hessematrix $\nabla^2 f(x^{(k)})$ in vielen Fällen aufwändig. Aus diesem Grund verwenden **Quasi-Newton-Verfahren** ein vereinfachtes quadratisches Modell der Zielfunktion $f(x)$:

$$m^{(k)}(x) := f(x^{(k)}) + \nabla f(x^{(k)})^\top (x - x^{(k)}) + \frac{1}{2} (x - x^{(k)})^\top B^{(k)} (x - x^{(k)}) \quad (3.1)$$

mit einer symmetrischen Matrix $B^{(k)}$. (Wie diese genau bestimmt wird, dazu später mehr.) Dieses vereinfachte Modell (3.1) gibt an der Stelle $x^{(k)}$ den Funktionswert und die erste Ableitung der Zielfunktion $f(x)$ noch richtig wieder:

$$\begin{aligned} m^{(k)}(x) &= f(x^{(k)}) + \nabla f(x^{(k)})^\top (x - x^{(k)}) + \frac{1}{2} (x - x^{(k)})^\top B^{(k)} (x - x^{(k)}) \\ \Rightarrow m^{(k)}(x^{(k)}) &= f(x^{(k)}) \\ \nabla m^{(k)}(x) &= \nabla f(x^{(k)}) + B^{(k)} (x - x^{(k)}) \\ \Rightarrow \nabla m^{(k)}(x^{(k)}) &= \nabla f(x^{(k)}). \end{aligned}$$

Die Hessematrix des vereinfachten Modells, also

$$\nabla^2 m^{(k)}(x) \equiv B^{(k)},$$

muss jedoch nicht mehr mit der Hessematrix der Zielfunktion $\nabla^2 f(x^{(k)})$ übereinstimmen. Man spricht hier von einer **Quasi-Newton-Approximation** der Hessematrix. Die Vereinfachung von $m^{(k)}(x)$ gegenüber $M^{(k)}(x)$ besteht also darin, dass man sich die Berechnung der „richtigen“ Hessematrix $\nabla^2 f(x^{(k)})$ spart und stattdessen die **Modell-Hessematrix** $B^{(k)}$ verwendet. Die Approximation von $f(x)$ durch die verschiedenen quadratische Modelle $M^{(0)}(x)$ und $m^{(0)}(x)$ in der Nähe eines Entwicklungspunktes $x^{(0)}$ ist in Abb. 3.1 gezeigt.

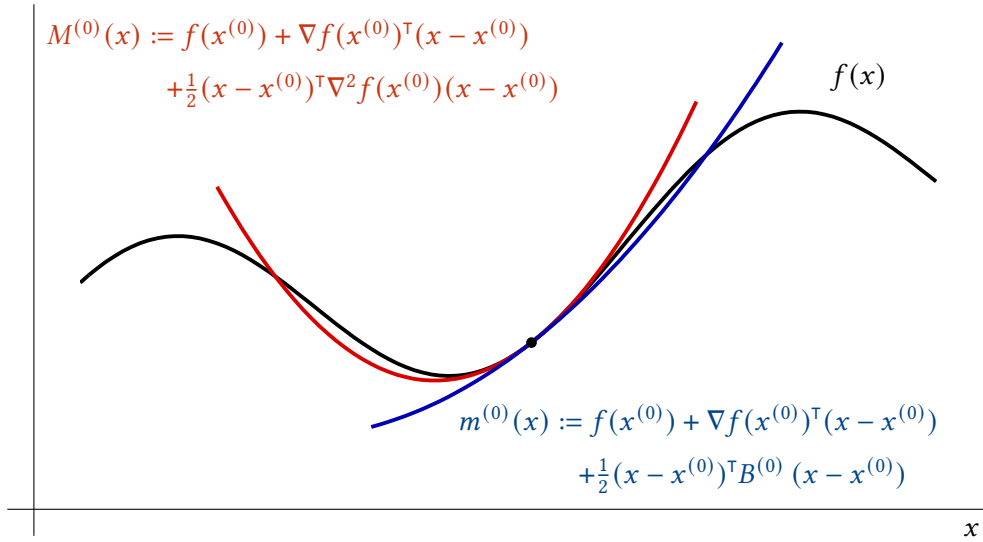


Abbildung 3.1.: Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ und ihr quadratisches Taylorpolynom $M^{(0)}(x)$ (in rot) mit Entwicklungspunkt $x^{(0)}$. Gezeigt wird außerdem ein vereinfachtes quadratisches Modell $m^{(0)}(x)$ der Funktion (blau), dessen zweite Ableitung $B^{(0)}$ nicht mit $\nabla^2 f(x^{(0)})$ übereinstimmt. Man sieht, dass das Taylorpolynom $M^{(0)}(x)$ die Funktion $f(x)$ besser annähert als das vereinfachte Modell $m^{(0)}(x)$, das dafür ohne die Berechnung der zweiten Ableitung $\nabla^2 f(x^{(0)})$ auskommt.

Zur Vereinfachung der Notation führen wir die **Richtung** $d \in \mathbb{R}^n$ ein und ersetzen $x - x^{(k)}$ durch d . (Der Vektor d zeigt also von $x^{(k)}$ nach x .) Das Modell mit dieser Ersetzung nennen wir $q^{(k)}(d)$:

$$\begin{aligned} q^{(k)}(d) &:= m^{(k)}(x^{(k)} + d) \\ &= f(x^{(k)}) + \nabla f(x^{(k)})^T d + \frac{1}{2} d^T B^{(k)} d. \end{aligned} \quad (3.2)$$

Im Teilschritt (2) soll das Minimum der quadratischen Modellfunktion $q^{(k)}(d)$ mit $d \in \mathbb{R}^n$ bestimmt werden. Die Optimalitätsbedingung 1. Ordnung (Satz 2.3) sagt aus, dass bei einem Minimum d^* von (3.2) auf jeden Fall

$$\nabla q^{(k)}(d^*) = 0$$

erfüllt sein muss, also

$$\nabla f(x^{(k)}) + B^{(k)} d^* = 0 \quad \text{bzw.} \quad B^{(k)} d^* = -\nabla f(x^{(k)}). \quad (3.3)$$

Dies ist ein lineares Gleichungssystem für die Unbekannte $d^* \in \mathbb{R}^n$. Wenn d^* eine Lösung von (3.3) ist, dann müssen wir anhand der zweiten Ableitung prüfen, ob d^* tatsächlich ein Minimum von $q^{(k)}(d)$ ist:¹²

$$\nabla^2 q^{(k)}(d) \equiv B^{(k)} \quad \Rightarrow \quad \nabla^2 q^{(k)}(d^*) = B^{(k)}.$$

Die hinreichenden Optimalitätsbedingungen 2. Ordnung (Satz 2.5) besagen nun: Wenn $B^{(k)}$ positiv definit ist, dann ist d^* tatsächlich das eindeutige globale Optimum von $q^{(k)}(d)$.

Diese Beobachtung ist der Grund dafür, dass Quasi-Newton-Verfahren mit Liniensuche die Hessematrix $B^{(k)}$ des Ersatzmodells $q^{(k)}(d)$ als **positiv definit** wählen. Dies hat außerdem noch die weiterreichende Konsequenz, dass das lineare Gleichungssystem (3.3) *eindeutig* lösbar ist. Die Lösung d^* ist also das *eindeutige* Minimum von $q^{(k)}(d)$ und damit natürlich automatisch das globale Minimum des Ersatzproblems aus Teilschritt (2).

Ab sofort wollen wir den Vektor d^* auch mit $d^{(k)}$ bezeichnen. Man überzeugt sich leicht davon, dass $d^{(k)}$ eine **Abstiegsrichtung** für unsere Zielfunktion $f(x)$ an der Stelle $x^{(k)}$ ist. Das heißt, dass die Richtungsableitung von $f(x)$ an der Stelle $x^{(k)}$ in die Richtung $d^{(k)}$ negativ ist:

$$\nabla f(x^{(k)})^\top d^{(k)} \stackrel{(3.3)}{=} -\nabla f(x^{(k)})^\top (B^{(k)})^{-1} \nabla f(x^{(k)}) < 0,$$

falls $\nabla f(x^{(k)}) \neq 0$. Die Ungleichung folgt aus der Tatsache, dass mit $B^{(k)}$ auch $(B^{(k)})^{-1}$ positiv definit ist, siehe [Anhang 21](#) im Anhang.

Die Negativität der Richtungsableitung besagt, dass wir zumindest für kleine Schrittweiten $\alpha > 0$ eine Reduktion in der Zielfunktion erreichen:

$$f(x^{(k)} + \alpha d^{(k)}) < f(x^{(k)}).$$

Es wäre für ein Optimierungsverfahren naheliegend, als nächste Iterierte $x^{(k+1)}$ das Minimum des quadratischen Modells $m^{(k)}(x)$ zu wählen bzw. — was auf dasselbe hinausläuft — als Schritt von $x^{(k)}$ zu $x^{(k+1)}$ das Minimum $d^{(k)}$ des quadratischen Modells $q^{(k)}(d)$ zu verwenden. Dieses naive Vorgehen entspricht dem Setzen von $x^{(k+1)} := x^{(k)} + d^{(k)}$. Einfache Beispiele zeigen jedoch, dass dann die Werte der Zielfunktion anwachsen können und das Verfahren evtl. nicht konvergiert.

Deshalb wird in Schritt (3) des Verfahrens noch eine sogenannte **Liniensuche** zur Bestimmung einer geeigneten Schrittweite $\alpha^{(k)}$ durchgeführt.¹³ Wir sehen uns dazu die Liniensuchfunktion

$$\alpha \mapsto \varphi(\alpha) := f(x^{(k)} + \alpha d^{(k)}), \quad \alpha \geq 0$$

an, also den „Schnitt“ durch die Funktion $f(x)$, ausgehend von der Stelle $x^{(k)}$ und in Richtung $d^{(k)}$, die ja schon berechnet wurde. Die Steigung dieser Funktion an einer Stelle α ist (nach Kettenregel)

$$\varphi'(\alpha) = \nabla f(x^{(k)} + \alpha d^{(k)})^\top d^{(k)}.$$

¹²Da $q^{(k)}(d)$ ein quadratisches Polynom in d ist, ist die zweite Ableitung (Hessematrix) natürlich konstant, also von der Stelle d unabhängig.

¹³Die Verwendung einer Liniensuche ist (neben der Verwendung von trust regions, siehe § 5) eine gängige Methode zur **Globalisierung** von Optimierungsverfahren. „Globalisierung“ bedeutet nicht, dass globale statt lokaler Minima gefunden werden, sondern, dass das Verfahren möglichst von beliebigen Startpunkten aus gegen einen für die Zielfunktion „interessanten“ Punkt x^* (zumindest ein stationärer Punkt, möglichst ein lokales Minimum) konvergieren soll.

An der Stelle $\alpha = 0$ ist die Steigung also

$$\varphi'(0) = \nabla f(x^{(k)})^\top d^{(k)} < 0,$$

da $d^{(k)}$ eine Abstiegsrichtung für $f(x)$ an der Stelle $x^{(k)}$ war.

Es ist den Aufwand nicht wert, ein lokales oder gar globales Minimum von $\varphi(\alpha)$ zu bestimmen, denn das ist recht aufwändig, und im nächsten Iterationsschritt wird ohnehin wieder eine neue Suchrichtung bestimmt. Deshalb geben sich praktische Optimierungsverfahren damit zufrieden, eine Schrittweite $\alpha > 0$ zu bestimmen, die bestimmte Bedingungen erfüllt. Die erste dieser Bedingungen ist die sogenannte **Armijo-Bedingung**:

$$f(x^{(k)} + \alpha d^{(k)}) \leq f(x^{(k)}) + \sigma \alpha \nabla f(x^{(k)})^\top d^{(k)} \quad \text{bzw.} \quad \varphi(\alpha) \leq \varphi(0) + \sigma \alpha \varphi'(0). \quad (3.4)$$

Dabei ist $0 < \sigma < 1$ ein gegebener Parameter.¹⁴ Die Armijo-Bedingung fordert also einen Abstieg $\varphi(\alpha) - \varphi(0)$ in den Funktionswerten, der mindestens proportional (mit dem Proportionalitätsfaktor σ) zur Schrittlänge α ist, siehe auch [Abb. 3.2](#). Im Zusammenhang mit unserem Quasi-Newton-Verfahren `fminunc` ist noch eine weitere Bedingung relevant, die sogenannte **Krümmungsbedingung**:

$$\nabla f(x^{(k)} + \alpha d^{(k)})^\top d^{(k)} \geq \tau \nabla f(x^{(k)})^\top d^{(k)} \quad \text{bzw.} \quad \varphi'(\alpha) \geq \tau \varphi'(0) \quad (3.5)$$

mit einem weiteren Parameter $\sigma < \tau < 1$. Diese Bedingung fordert, dass die Steigung der Liniensuchfunktion φ an der Stelle α größer ist als τ mal die anfängliche Steigung (bei $\alpha = 0$), also entweder negativ und näher bei null oder sogar positiv. Die Bedingung verhindert insbesondere, dass sehr kleine Schrittweiten $\alpha > 0$ akzeptiert werden.

„Robuste“ (auf den vielfachen praktischen Einsatz getrimmte) Optimierungsverfahren wie `fminunc` verwenden ausgefeilte Strategien, um eine für (3.4) und (3.5) zulässige Schrittweite $\alpha^{(k)}$ zu finden.¹⁵ Ziel muss es sein, möglichst wenige Werte für α ausprobieren zu müssen, denn: Für jeden Wert von α , der getestet wird, müssen $f(x^{(k)} + \alpha d^{(k)})$ und $\nabla f(x^{(k)} + \alpha d^{(k)})$ ausgerechnet werden.

Wenn die Schrittweite $\alpha^{(k)}$ schließlich bestimmt wurde, datieren wir unsere Iterierte auf:

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

Wir können nun mit diesem Wissen die Ausgabe von `fminunc` am obigen Beispiel interpretieren:¹⁶

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	11500		500
1	6	10901.6	0.002	256

¹⁴Die Parameter σ und τ sind beim Quasi-Newton-Verfahren von `fminunc` voreingestellt und können vom Benutzer nicht beeinflusst werden.

¹⁵Die Bedingungen (3.4) und (3.5) zusammen heißen die **Wolfe-(Powell)-Bedingungen**.

¹⁶Die Ausgabe wird in der Übung besprochen.

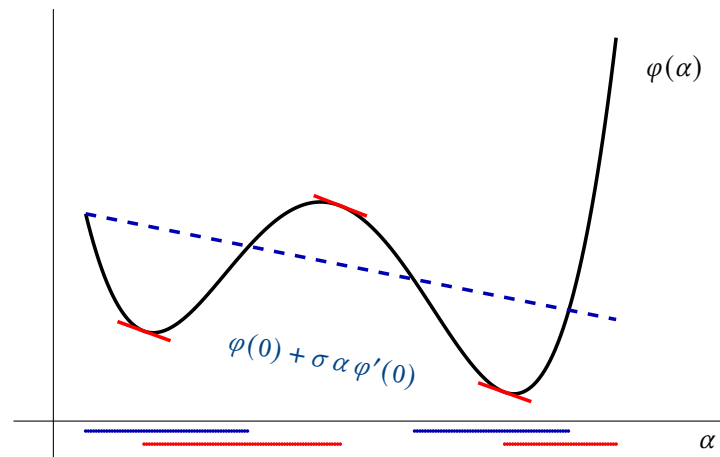


Abbildung 3.2.: Illustration der Schrittweiten $\alpha \geq 0$, die die Armijo-Bedingung (3.4) erfüllen (blau) bzw. die Krümmungsbedingung (3.5) (rot). Als Parameter sind hier $\sigma = 0.05$ und $\tau = 0.1$ gewählt.

2	9	10625.3	1	103
3	12	10563.9	1	35
4	15	10556.4	1	9.03
5	18	10555.9	1	4.04
6	21	10555.9	1	2.11
7	24	10555.8	1	0.36
8	27	10555.8	1	0.0536
9	30	10555.8	1	0.00284
10	33	10555.8	1	0.000121

Optimization completed: The first-order optimality measure, 2.412045e-07, is less than options.OptimalityTolerance = 1.000000e-06.

Optimization Metric	Options
relative norm(gradient) = 2.41e-07	OptimalityTolerance = 1e-06 (default)

- Die Spalte Iteration bezeichnet unseren Iterationszähler $k = 0, 1, \dots$
- In der Spalte Func-count finden wir die Gesamtanzahl der Auswertungen unserer Zielfunktion $f(x)$. Dies ist eine gängige „Währung“, um den Aufwand eines Verfahrens anzugeben. (Dass hier in unserem Beispiel immer Vielfache von drei auftreten, ist kein Zufall und wird in § 4 erklärt.)
- Die Spalte $f(x)$ zeigt die Werte $f(x^{(k)})$ der Zielfunktion an der jeweils aktuellen Iterierten. Man erkennt sehr schön, dass das Verfahren ein **Abstiegsverfahren** ist, d. h. die Werte sind monoton fallend. Ab $k = 7$ ändert sich die hier sichtbare erste Nachkommastelle der Zielfunktionswerte nicht mehr. Dennoch macht das Verfahren noch Fortschritte, wie man an der letzten Spalte sieht.
- In der Spalte Step-size wird die mittels Liniensuche bestimmte Schrittweite $\alpha^{(k)}$ ausgegeben. Werte von $\alpha^{(k)} = 1$ sind wünschenswert und deuten auf einen guten Fortschritt des Verfahrens hin.
- Schließlich finden sich in der letzten Spalte die Werte von $\|\nabla f(x^{(k)})\|_\infty$. Diese Werte

sind interessant, weil wir ja ein Minimum x^* von $f(x)$ suchen, an dem also die notwendige Bedingung 1. Ordnung (Satz 2.3) $\nabla f(x^*) = 0$ gilt. Deshalb ist die Spalte auch mit `First-order optimality` überschrieben.

Die Iterierten $x^{(k)}$ und die Suchrichtungen $d^{(k)}$ werden nicht ausgegeben. Dies sind ja auch Vektoren im $\mathbb{R}^n$, die möglicherweise sehr viel Platz bei der Ausgabe benötigen würden. Es gibt aber die Möglichkeit, die aktuellen Iterierten $x^{(k)}$ mit Hilfe einer `OutputFcn` bzw. `PlotFcns` problemspezifisch darzustellen.¹⁷

Das Verfahren `fminunc` stoppt nach Iteration 10 und gibt $x^* := x^{(10)}$ als Lösung zurück, weil $\|\nabla f(x^{(10)})\|_\infty$ hinreichend klein ist, nämlich

$$\|\nabla f(x^{(10)})\|_\infty = 0.000121 = 1.2084 \cdot 10^{-4}.$$

Die Iterierte $x^{(10)}$ ist also mit hinreichender Genauigkeit ein stationärer Punkt und – wie wir oben bereits mit Hilfe der Hessematrix überprüft haben – tatsächlich (ungefähr) ein lokales Minimum für Beispiel 2.1. Genauer wird hier das Verfahren deshalb gestoppt, weil eine relative Reduktion

$$\frac{\|\nabla f(x^{(10)})\|_\infty}{\|\nabla f(x^{(0)})\|_\infty} = \frac{0.000121}{500} = 2.42 \cdot 10^{-7}$$

in der Norm des Gradienten erreicht wurde, was unterhalb der Standard-Toleranz von 10^{-6} liegt. Der Grund des Abbruchs wird auch durch die Rückgabebvariable `exitflag` reflektiert, die hier den Wert 1 hat. Laut Dokumentation von `fminunc` bedeutet dies: `Magnitude of gradient small enough`.

Es bleibt noch zu klären, wie das Verfahren die Hessematrizen $B^{(k)}$ der Modellfunktionen $q^{(k)}(d)$ wählt. Wenn etwa der Schritt $x^{(k)} \leadsto x^{(k+1)}$ bereits gemacht wurde, stellt sich die Frage, wie die *nächste* Matrix $B^{(k+1)}$ aussehen soll. Dazu betrachten wir das quadratische Modell bei $x^{(k+1)}$, vgl. (3.1):

$$m^{(k+1)}(x) = f(x^{(k+1)}) + \nabla f(x^{(k+1)})^\top (x - x^{(k+1)}) + \frac{1}{2} (x - x^{(k+1)})^\top B^{(k+1)} (x - x^{(k+1)}) \quad (3.6)$$

wiederum mit einer symmetrischen Matrix $B^{(k+1)}$. Nach Konstruktion stimmt der Funktionswert und der Gradient dieses neuen Modells mit dem der Zielfunktion $f(x)$ an der Stelle $x^{(k+1)}$ automatisch überein. Wir fordern nun zusätzlich, dass der Gradient von $m^{(k+1)}(x)$ auch noch mit dem Gradienten von $f(x)$ an der *alten* Stelle $x^{(k)}$ übereinstimmt, also:

$$\begin{aligned} \nabla m^{(k+1)}(x^{(k)}) &= \nabla f(x^{(k)}) \\ \Leftrightarrow \nabla f(x^{(k+1)}) + B^{(k+1)} (x^{(k)} - x^{(k+1)}) &= \nabla f(x^{(k)}) \\ \Leftrightarrow B^{(k+1)} (x^{(k+1)} - x^{(k)}) &= \nabla f(x^{(k+1)}) - \nabla f(x^{(k)}). \end{aligned} \quad (3.7)$$

Diese Bedingung heißt die **Sekantenbedingung**, und sie wird von allen gängigen Quasi-Newton-Verfahren benutzt.

¹⁷In der Übung wird eine Plot-Funktion geschrieben, die den Iterationsverlauf in einen Plot der Zielfunktion einzeichnet.

Da auch die Sekantenbedingung (3.7) die symmetrische Matrix $B^{(k+1)}$ noch nicht eindeutig festlegt, gibt es verschiedene Quasi-Newton-Varianten. Eine der gängigsten, die auch die Quasi-Newton-Methode in `fminunc` verwendet, ist die **BFGS-Formel** (Broyden, Fletcher, Goldfarb, Shanno):

$$B_{\text{BFGS}}^{(k+1)} = B_{\text{BFGS}}^{(k)} - \frac{B_{\text{BFGS}}^{(k)} s^{(k)} (s^{(k)})^\top B_{\text{BFGS}}^{(k)}}{(s^{(k)})^\top B_{\text{BFGS}}^{(k)} s^{(k)}} + \rho^{(k)} y^{(k)} (y^{(k)})^\top, \quad (3.8)$$

wobei die Abkürzungen

$$s^{(k)} := x^{(k+1)} - x^{(k)}, \quad y^{(k)} := \nabla f(x^{(k+1)}) - \nabla f(x^{(k)}) \quad \text{und} \quad \rho^{(k)} := \frac{1}{(y^{(k)})^\top s^{(k)}}$$

verwendet wurden.

Es stellt sich die Frage, ob die so berechnete Matrix $B^{(k+1)}$ wie gewünscht positiv definit ist. Aus der Sekantenbedingung (3.7), also $B^{(k+1)} s^{(k)} = y^{(k)}$, sieht man sofort, dass dafür

$$(s^{(k)})^\top B^{(k+1)} s^{(k)} = (s^{(k)})^\top y^{(k)} > 0$$

gelten muss. In der Tat ist dies genau die Bedingung, die man braucht:

Lemma 3.1 (Positive Definitheit des BFGS-Updates). *Es sei $B_{\text{BFGS}}^{(k)}$ positiv definit. Wenn die Bedingung $(s^{(k)})^\top y^{(k)} > 0$ erfüllt ist, dann ist auch $B_{\text{BFGS}}^{(k+1)}$ wieder positiv definit.*

Beweis. Die Symmetrie von $B_{\text{BFGS}}^{(k+1)}$ ergibt sich sofort aus (3.8). Es sei nun $z \in \mathbb{R}^n \setminus \{0\}$ ein beliebiger Vektor ungleich dem Nullvektor. Wir multiplizieren z von links und rechts an die Matrix $B_{\text{BFGS}}^{(k+1)}$ heran, um auf positive Definitheit zu testen. Zur Vereinfachung der Notation lassen wir die Bezeichnung BFGS hier weg:

$$\begin{aligned} z^\top B^{(k+1)} z &= z^\top B^{(k)} z - \frac{z^\top B^{(k)} s^{(k)} (s^{(k)})^\top B^{(k)} z}{(s^{(k)})^\top B^{(k)} s^{(k)}} + \rho^{(k)} z^\top y^{(k)} (y^{(k)})^\top z \\ &\geq z^\top B^{(k)} z - \frac{\|z\|_{B^{(k)}}^2 \|s^{(k)}\|_{B^{(k)}}^2}{\|s^{(k)}\|_{B^{(k)}}^2} + \rho^{(k)} (z^\top y^{(k)})^2 \quad \text{Cauchy-Schwarz-Ungl.} \\ &= \|z\|_{B^{(k)}}^2 - \|z\|_{B^{(k)}}^2 + \rho^{(k)} (z^\top y^{(k)})^2 \geq 0, \end{aligned}$$

weil nach Voraussetzung $\rho^{(k)} = 1/((y^{(k)})^\top s^{(k)}) > 0$ gilt. Der letzte Ausdruck ist nur dann Null, wenn $z^\top y^{(k)} = 0$ gilt. Dann folgt aber aus $(s^{(k)})^\top y^{(k)} > 0$, dass z und $s^{(k)}$ linear unabhängig sind. Damit kann die Cauchy-Schwarz-Ungleichung (siehe [Anhang 20](#)) nicht mit Gleichheit erfüllt sein. $\square$

Die offenbar für die positive Definitheit wichtige Bedingung $(s^{(k)})^\top y^{(k)} > 0$ ist durch die

Erfüllung der Krümmungsbedingung (3.5) bei der Liniensuche sichergestellt, denn:

$$\begin{aligned}
 & \nabla f(x^{(k)} + \alpha^{(k)} d^{(k)})^\top d^{(k)} \geq \tau \nabla f(x^{(k)})^\top d^{(k)} \quad \Big| - \nabla f(x^{(k)})^\top d^{(k)} \\
 \Rightarrow & \quad (\underbrace{\nabla f(x^{(k)} + \alpha^{(k)} d^{(k)})}_{=x^{(k+1)}} - \nabla f(x^{(k)}))^\top d^{(k)} \geq (\tau - 1) \nabla f(x^{(k)})^\top d^{(k)} \quad \Big| \cdot \alpha^{(k)} \\
 \Rightarrow & \quad (y^{(k)})^\top \underbrace{(\alpha^{(k)} d^{(k)})}_{=s^{(k)}} \geq \underbrace{\alpha^{(k)}}_{>0} \underbrace{(\tau - 1)}_{<0} \underbrace{\nabla f(x^{(k)})^\top d^{(k)}}_{<0} > 0,
 \end{aligned}$$

denn $d^{(k)}$ ist – wie oben nachgerechnet – eine Abstiegsrichtung.

Wir fassen unsere Erkenntnisse über das Quasi-Newton-Verfahren in fminunc zusammen:

- (i) Ein Quasi-Newton-Verfahren erzeugt eine Folge quadratischer Modellfunktionen $m^{(k)}(x)$ bzw. $q^{(k)}(d)$ der Zielfunktion. Diese Modelle sind jeweils an der aktuellen Iterierten $x^{(k)}$ zentriert.
- (ii) Die quadratischen Modelle benutzen an Stelle der exakten Hessematrix der Zielfunktion bestimmte symmetrisch positiv definite Matrizen $B^{(k)}$ als Hessematrizen. Diese ergeben sich jeweils durch ein leicht zu berechnendes Update (hier die BFGS-Formel) aus der vorangegangenen Matrix.
- (iii) Das Verfahren wählt als anfängliche Hessematrix $B^{(0)}$ ein Vielfaches der Einheitsmatrix. Man kann diese jedoch auch benutzerdefiniert vorgeben.
- (iv) Durch dieses Vorgehen
 - spart man sich erstens die Berechnung zweiter Ableitungen der Zielfunktion $f(x)$
 - und zweitens stellt man sicher, dass die Matrizen $B^{(k)}$ positiv definit sind, wodurch die gemäß (3.3) berechnete Suchrichtung $d^{(k)}$ eine Abstiegsrichtung für die Zielfunktion ist.

Wir fügen also unserem Ablaufschema für ein Quasi-Newton-Verfahren vom Anfang von § 3 noch einen Schritt hinzu:

- (1) Wähle die neue Modell-Hessematrix $B^{(k+1)}$ mit Hilfe von $B^{(k)}$, $s^{(k)} = x^{(k+1)} - x^{(k)}$ und $y^{(k)} = \nabla f(x^{(k+1)}) - \nabla f(x^{(k)})$, z. B. über die BFGS-Formel (3.8).

In Wirklichkeit geht das Verfahren noch geschickter vor, um weitere Arbeit zu sparen. Für die Bestimmung der Suchrichtung $d^{(k)}$ muss in jeder Iteration das lineare Gleichungssystem (3.3) gelöst werden. Dies ist relativ aufwändig (Gauß-Verfahren) und kann erspart werden, wenn man statt mit $B^{(k)}$ mit der *inversen Matrix* $H^{(k)} := (B^{(k)})^{-1}$ arbeitet. Und in der Tat kann man auch diese Matrizen mit Hilfe einer Update-Formel (hier der **inversen BFGS-Formel**) erzeugen:¹⁸

$$H_{\text{BFGS}}^{(k+1)} = (I - \rho^{(k)} s^{(k)} (y^{(k)})^\top) H_{\text{BFGS}}^{(k)} (I - \rho^{(k)} y^{(k)} (s^{(k)})^\top) + \rho^{(k)} s^{(k)} (s^{(k)})^\top. \quad (3.9)$$

¹⁸Sofern $B_{\text{BFGS}}^{(0)}$ und $H_{\text{BFGS}}^{(0)}$ zueinander inverse Matrizen sind, so gilt das auch für alle später folgenden Matrizen $B_{\text{BFGS}}^{(k)}$ und $H_{\text{BFGS}}^{(k)}$.

Das [Lemma 3.1](#) über die positive Definitheit gilt unter denselben Voraussetzungen auch für diese Matrizen. Der Vorteil der Verwendung von $H^{(k)} = (B^{(k)})^{-1}$ liegt darin, dass jetzt statt

$$B^{(k)} d^{(k)} = -\nabla f(x^{(k)})$$

dieselbe Suchrichtung durch die wesentlich weniger aufwändige Matrix-Vektor-Multiplikation

$$d^{(k)} = -H^{(k)} \nabla f(x^{(k)})$$

berechnet werden kann.

Ende der Woche 3

§ 4 BEREITSTELLEN VON ABLEITUNGEN

Bei näherem Hinsehen fällt auf, dass wir bei der Lösung von [Beispiel 2.1](#) mittels `fminunc` überhaupt keine Ableitungen der Zielfunktion $f(x)$ programmiert haben, obwohl diese in den quadratischen Ersatzmodellen [\(3.2\)](#) benötigt werden. Der Widerspruch klärt sich auf, wenn man weiß, dass `fminunc` sich den Gradienten $\nabla f(x^{(k)})$ selbstständig berechnet, wenn der Benutzer ihn nicht bereitstellt. Dies geschieht durch eine **Differenzenapproximation (finite Differenzen)**

$$\frac{\partial f(x)}{\partial x_j} \approx \frac{f(x + h e_j) - f(x)}{h} \quad (4.1)$$

mit einer kleinen Zahl $h > 0$. Dabei ist

$$e_j = (0, \dots, 0, 1, 0, \dots, 0)^T$$

einer der Einheitsvektoren des $\mathbb{R}^n$ mit der „1“ an Position j für $1 \leq j \leq n$.

Dies erklärt, warum wir in der obigen Ausgabe von `fminunc` beim [Beispiel 2.1](#) jeweils Dreier-Schritte bei der Anzahl der Funktionsauswertungen `Func-count` beobachtet haben: In jeder Iteration werden an der aktuellen Stelle $x^{(k)}$ die Daten

$$f(x^{(k)}) \quad \text{und} \quad \nabla f(x^{(k)})$$

benötigt. Da im Beispiel $x \in \mathbb{R}^2$ ist (zwei Entscheidungsvariablen), werden dafür mittels Differenzenapproximation insgesamt drei Auswertungen von $f(x)$ benötigt, nämlich

$$f(x^{(k)}), \quad f(x^{(k)} + h e_1) \quad \text{und} \quad f(x^{(k)} + h e_2).$$

Während der Liniensuche könnten evtl. weitere Auswertungen von $f(x)$ hinzukommen, und zwar dann, wenn die erste Versuchsschrittweite α eine der Bedingungen [\(3.4\)](#) oder [\(3.5\)](#) nicht erfüllt. Im Beispiel war jedoch immer direkt der erste Versuch erfolgreich.

Das oben beschriebene Verhalten von `fminunc`, sich den Gradienten der Zielfunktion selbst durch eine Differenzenapproximation zu berechnen, ist zwar bequem, hat aber auch Nachteile:

- Die Differenzenapproximation hat nur begrenzte Genauigkeit. Diese beeinflusst aber die Genauigkeit, mit der `fminunc` (und auch jedes andere ableitungsbasierte Optimierungsverfahren) eine Lösung x^* der Optimierungsaufgabe (2.2) bestimmen kann. Die Genauigkeit ist umso geringer, je größer die Krümmung (zweite Ableitung) der Zielfunktion ist.¹⁹
- Der Aufwand für die Berechnung von $\nabla f(x)$ durch Differenzenquotienten ist proportional zur Anzahl n der Optimierungsvariablen $x \in \mathbb{R}^n$.

Wann immer möglich, sollte man daher neben der Zielfunktion auch deren Gradienten programmieren und dem Optimierungsverfahren zur Verfügung stellen. Dies wollen wir nun für das [Beispiel 2.1](#) nachholen. Prinzipiell gibt es dafür folgende Möglichkeiten:

- (1) Man berechnet die Ableitung mit der Hand und setzt dies in Computercode um.
- (2) Man verwendet die Symbolic-Math-Toolbox in MATLAB, die Funktionen auch symbolisch ableiten kann.
- (3) Man berechnet die Ableitung mit Hilfe eines Computeralgebrasystems (z. B. MATHEMATICA, MAPLE oder MAXIMA). Dies nennt man **symbolisches Differenzieren**. Die Formel für die Ableitung muss anschließend in Computercode umgesetzt werden. Dabei kann die Code-Export-Funktion dieser Programmsysteme sehr hilfreich sein.²⁰
- (4) Man nutzt **algorithmisches Differenzieren** (auch: **automatisches Differenzieren** oder kurz: **AD**) zur Berechnung der Ableitung. Dabei wird direkt der Computercode zur Berechnung der Zielfunktion „abgeleitet“.²¹

Da unsere Zielfunktion in [Beispiel 2.1](#) so einfach ist, verwenden wir hier Möglichkeit (1):

$$K(d_1, d_2) = c_0 \left(\frac{1}{d_1} + \frac{1}{d_2} \right) + c_1 d_1 + c_2 d_2, \quad (4.2a)$$

$$\frac{\partial K(d_1, d_2)}{\partial d_1} = c_0 \left(\frac{-1}{d_1^2} + 0 \right) + c_1 = 0, \quad (4.2b)$$

$$\frac{\partial K(d_1, d_2)}{\partial d_2} = c_0 \left(0 + \frac{-1}{d_2^2} \right) + 0 = c_2. \quad (4.2c)$$

Nun erweitern wir unseren Code zur Berechnung der Zielfunktion wie folgt. Die neue Datei nennen wir zur Unterscheidung `Hausbau_Zielfunktion_mit_Ableitung.m`:

```
function [K,gradK] = Hausbau_Zielfunktion(x,c0,c1,c2)

% Bedeutung der Variablen
% x(1) = d1 = Wandstaerke Material 1 [cm]
% x(2) = d2 = Wandstaerke Material 2 [cm]
```

¹⁹In der Übung wird dies an einem Beispiel gezeigt.

²⁰MAPLE kann zum Beispiel mit Hilfe von `with(CodeGeneration): Matlab(expression) MATLAB-Code erzeugen, der die expression berechnet.`

²¹Das gelingt etwa mit Hilfe von **ADiMat** sehr einfach: `admopts = admOptions('independents', [1]); [gradK,K] = admDiffRev(@Hausbau_Zielfunktion,1,x0,c0,c1,c2,admopts);` Dadurch wird unsere Zielfunktion mitsamt ihrer ersten Ableitung an der Stelle `x0` ausgewertet. Leider kann der so erhaltene Code nicht direkt in `fminunc` verwendet werden, da die Reihenfolge der Ausgabeargumente nicht kompatibel ist. Dies kann aber mit einem entsprechenden Wrapper leicht behoben werden, siehe `Hausbau_Zielfunktion_mit_Ableitung_ADiMat.m`

```
d1 = x(1);
d2 = x(2);

% Berechnung der Zielfunktion und ihres Gradienten
K = c0 * (1/d1 + 1/d2) + c1 * d1 + c2 * d2;
gradK(1) = c0 * (-1/d1^2 + 0) + c1 + 0;
gradK(2) = c0 * (0 + -1/d2^2) + 0 + c2;
```

Im Hauptprogramm Hausbau_mit_Ableitung.m sind gegenüber der früheren Version Hausbau.m folgende Änderungen vorzunehmen:

Matlab/Hausbau_mit_Ableitung.m

```
% Setzen der Zielfunktion
f = @(x) Hausbau_Zielfunktion_mit_Ableitung(x,c0,c1,c2);
```

Wir überprüfen direkt wieder an einem Beispiel, ob sie die korrekten Werte nun auch für die Ableitung liefert:

Matlab/Hausbau_mit_Ableitung.m

```
% % Ueberpruefen der Zielfunktion und ihres Gradienten
% x = [10;20];
% [K,gradK] = f(x)
```

Dies stimmt mit den Ergebnissen der Handrechnung überein:

$$K(10, 20) = 20\,000 \left(\frac{1}{10} + \frac{1}{20} \right) + 300 \cdot 10 + 400 \cdot 20 = 14\,000,$$

$$\frac{\partial K(10, 20)}{\partial d_1} = 20\,000 \left(\frac{-1}{10^2} + 0 \right) + 300 + 0 = 100,$$

$$\frac{\partial K(10, 20)}{\partial d_2} = 20\,000 \left(0 + \frac{-1}{20^2} \right) + 0 + 400 = 350.$$

Schließlich müssen wir fminunc vor dem Aufruf mittels SpecifyObjectiveGradient²² noch mitteilen, dass es ab sofort die Ableitung der Zielfunktion (engl.: *objective*) verwenden soll. Außerdem erlauben wir fminunc zur Sicherheit noch, diese Ableitung mit Hilfe der zuvor schon verwendeten Differenzenapproximation zu überprüfen: CheckGradients²³.

Matlab/Hausbau_mit_Ableitung.m

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('fminunc','Algorithm','quasi-newton','Display',
    'iter-detailed','SpecifyObjectiveGradient',true,'CheckGradients',
    true);

% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,grad,hessian] = fminunc(f,x0,options);
```

Wir erhalten folgende Ausgabe:

²²früher: GradObj on

²³früher: DerivativeCheck on

CheckGradients Information				
Objective function derivatives: Maximum relative difference between supplied and finite-difference derivatives = 2.03837e-08.				
CheckGradients successfully passed.				
Iteration	Func-count	f(x)	Step-size	First-order optimality
0	1	11500		500
1	2	10901.6	0.002	256
2	3	10625.3	1	103
3	4	10563.9	1	35
4	5	10556.4	1	9.03
5	6	10555.9	1	4.04
6	7	10555.9	1	2.11
7	8	10555.8	1	0.36
8	9	10555.8	1	0.0536
9	10	10555.8	1	0.00284
10	11	10555.8	1	0.000144
Optimization completed: The first-order optimality measure, 2.878942e-07, is less than options.OptimalityTolerance = 1.000000e-06.				
Optimization Metric		Options		
relative norm(gradient) =		2.88e-07	OptimalityTolerance =	
			1e-06 (default)	

Zunächst werden wir darüber informiert, dass unsere programmierte Ableitung den Test (Vergleich mit finiten Differenzen) bestanden hat. Anschließend sehen wir wieder den Verlauf des Verfahrens. Im Unterschied zum Lauf ohne Ableitungen fällt auf, dass nun unsere Zielfunktionsroutine Hausbau.Zielfunktion.mit.Ableitung.m, wie zu erwarten war, nur noch elf Mal aufgerufen wird. Wir erhalten in diesem Beispiel dieselbe Lösung (bis auf vier Nachkommastellen) wie zuvor ohne Programmierung der Ableitung, nämlich:

$$x^* = \begin{pmatrix} 8.1650 \\ 7.0711 \end{pmatrix}.$$

Das Bereitstellen der Ableitung lohnt sich jedoch deutlicher

- bei Aufgaben, deren Zielfunktion nichtlinearer ist als die in [Beispiel 2.1](#) oder bei denen die Zielfunktion in einzelne Richtungen stark unterschiedlich gekrümmt ist, denn darunter leidet die Genauigkeit²⁴ der Differenzenapproximation von $\nabla f(x)$,
- sowie bei Aufgaben, die wesentlich mehr Optimierungsvariablen haben.

§ 5 WEITERE INFORMATIONEN ZU fminunc

Unser anfängliches [Beispiel 2.1](#) ist insofern besonders, als dass es überhaupt nur einen stationären Punkt besitzt, der auch das globale Minimum der Aufgabe ist. Dies liegt an

²⁴siehe Übung

der Konvexität der Zielfunktion (Satz 2.6). Die meisten Optimierungsaufgaben besitzen aber mehrere (u. U. viele) lokale Minima. Optimierungsverfahren werden – von speziellen Ausnahmen einmal abgesehen – i. d. R. nur ein solches lokales Minimum finden. Gegen welches sie konvergieren, hängt dabei von der Startschätzung $x^{(0)}$ ab.

Wir illustrieren dies am Beispiel einer Funktion mit vielen lokalen Minima.

Beispiel 5.1 (Funktion mit vielen lokalen Minima²⁵).

$$\text{Minimiere } f(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2) - \cos(x_1) - \cos(x_2), \quad x = (x_1, x_2) \in \mathbb{R}^2. \quad (5.1)$$

Der Graph dieser Funktion ist in Abb. 5.1 zu sehen. Das globale Minimum liegt bei $x^* = (0, 0)$

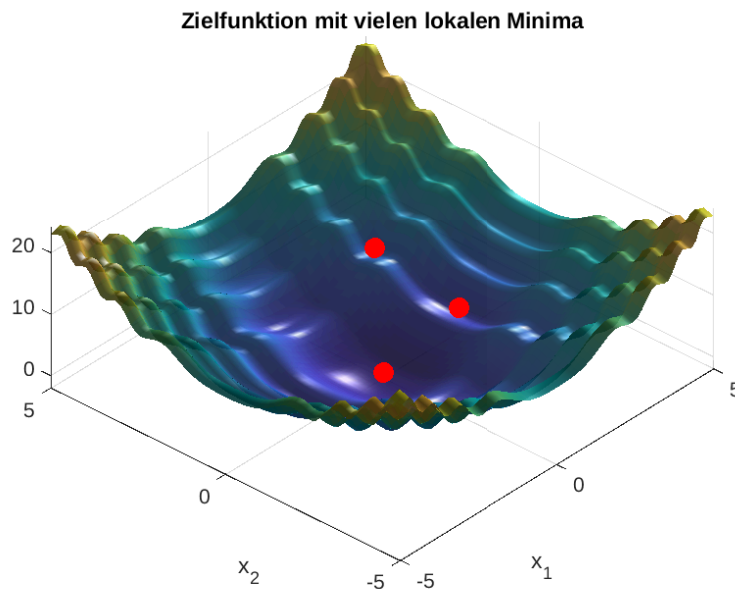


Abbildung 5.1.: Darstellung der Zielfunktion (5.1) mit einigen von `fminunc` berechneten lokalen Minima, ausgehend von verschiedenen Startwerten.

mit Funktionswert $f(x^*) = -2$. Wenn wir `fminunc` in verschiedenen Punkten $x^{(0)}$ starten, erhalten wir verschiedene Lösungen, die offenbar auch lokale Minima sein können.²⁶

Bisher haben wir nur die Quasi-Newton-Variante von `fminunc` kennengelernt. Im Gegensatz dazu ist die implementierte *Trust-Region*-Variante ein echtes **Newton-Verfahren**, d. h., dass die quadratischen Modelle $q^{(k)}(d)$ die tatsächliche Hessematrix $B^{(k)} = \nabla^2 f(x^{(k)})$ der Zielfunktion verwenden. Das ist zwar aufwändiger, verspricht aber insbesondere bei Aufgaben mit vielen Optimierungsvariablen (also $x \in \mathbb{R}^n$ mit großem n) schnellere Konvergenz. Bei Bedarf kann sich `fminunc` die Hessematrix mittels Differenzenapproximation selbst bestimmen.

²⁵aus Alt, 2002, Beispiel 1.1.2

²⁶In der Übung wird ein Beispiel für eine Funktion behandelt, die vier lokale Minima besitzt.

	Quasi-Newton-Variante (früher: <i>medium scale</i>)	Trust-Region-Variante (früher: <i>large scale</i>)
obligatorisch	$f(x)$	$f(x)$ und $\nabla f(x)$
optional	$\nabla f(x)$	$\nabla^2 f(x)$

Tabelle 5.1.: Unterschiede zwischen der Quasi-Newton- und der Trust-Region-Variante von `fminunc`

Der Benutzer muss jedoch bei Nutzung dieses Verfahrens immer zumindest den Gradienten der Zielfunktion mit implementieren.

Da die Hessematrix $\nabla^2 f(x^{(k)})$ nicht positiv definit sein muss, kann es passieren, dass die mittels (3.3) bestimmte Suchrichtung keine Abstiegsrichtung ist. Das bringt Schwierigkeiten bei der Liniensuche mit sich. Aus diesem Grund wird bei der *Trust-Region*-Variante die Suchrichtung anders bestimmt, und zwar durch Lösung eines sogenannten *Trust-Region*-Problems:

$$\begin{aligned} &\text{Minimiere} && q^{(k)}(d) \\ &\text{unter} && \|d\| \leq \Delta^{(k)}, \quad d \in \mathbb{R}^n. \end{aligned} \tag{5.2}$$

Die Beschränkung des Schrittes d auf eine Kugel vom Radius $\Delta^{(k)}$ steht dafür, dass wir dem quadratischen Modell $q^{(k)}(d)$ nur innerhalb dieses Bereiches (der *trust region*) vertrauen. Die Liniensuche entfällt dann.

Wir gehen auf die *Trust-Region*-Technik hier nicht weiter ein, stellen jedoch einige Unterschiede zwischen den beiden Varianten von `fminunc` noch einmal in [Tabelle 5.1](#) zusammen.

Weitere Beispiele für Optimierungsaufgaben ohne Beschränkung folgen in der Übung: Standortoptimierung und Rosenbrock-Funktion.

§ 6 UMWANDLUNG VON BESCHRÄNKTEN IN FREIE OPTIMIERUNGSAUFGABEN

Wie schon erwähnt, werden die meisten praktischen Optimierungsaufgaben erst durch Beschränkungen sinnvoll. Manchmal jedoch kann man vorhandene Gleichungsbeschränkungen auflösen und eliminieren, sodass man eine freie Optimierungsaufgabe erhält.²⁷

Beispiel 6.1 (Getränkedose). Ein Hersteller möchte Getränkedosen aus Blech herstellen, die 330 ml fassen sollen. Die Form der Dose soll zylindrisch sein. Der benötigte Überstand für die Bördelung (die Verbindung des Mantels mit dem Deckel bzw. Boden) wird vernachlässigt. Gesucht sind die Abmessungen, bei denen der Materialverbrauch minimal ist.

²⁷In der Übung wird dies am Beispiel des minimalen Abstands zwischen Punkt und Parabel demonstriert.

Die Abmessungen der zylinderförmigen Dose sind durch den Radius r und die Höhe h bestimmt. Der Materialverbrauch wird durch die Funktion

$$F(r, h) = \underbrace{2 \pi r^2}_{\text{Deckel, Boden}} + \underbrace{2 \pi r h}_{\text{Mantel}} \quad (6.1)$$

modelliert, die unsere zu minimierende Zielfunktion wird. Natürlich dürfen wir die Volumenbedingung nicht vergessen. Das Volumen der Dose beträgt

$$V(r, h) = \pi r^2 h.$$

Also lautet die zu beachtende Bedingung

$$\pi r^2 h = V_0 \quad (6.2)$$

mit der Konstanten $V_0 = 330 \text{ [cm}^3\text{]}$. Die gesamte Aufgabe lautet also

$$\begin{aligned} &\text{Minimiere} && 2 \pi r^2 + 2 \pi r h, && (r, h) \in \mathbb{R}^2 \\ &\text{unter} && \pi r^2 h = V_0. \end{aligned}$$

Dies ist eine Optimierungsaufgabe mit zwei Optimierungsvariablen $(r, h) \in \mathbb{R}^2$ und einer Gleichungsnebenbedingung, wie wir sie eigentlich erst in [Kapitel 4](#) behandeln werden.²⁸ Wir können aber hier leicht die Gleichungsnebenbedingung nach h auflösen:

$$h = \frac{V_0}{\pi r^2}.$$

Die Höhe h wird dadurch eine vom Radius r abhängige Größe, und die Gleichungsnebenbedingung ist damit erfüllt. Nun müssen wir diesen Ausdruck für h noch in die Zielfunktion einsetzen:

$$\begin{aligned} f(r) &:= F\left(r, \frac{V_0}{\pi r^2}\right) = 2 \pi r^2 + 2 \pi r \frac{V_0}{\pi r^2} \\ &= 2 \pi r^2 + 2 \frac{V_0}{r} \end{aligned} \quad (6.3)$$

und diese **reduzierte Zielfunktion** über die verbleibende Optimierungsvariable $r \in \mathbb{R}$ minimieren. Wir haben also schließlich die Aufgabe²⁹

$$\text{Minimiere} \quad 2 \pi r^2 + 2 \frac{V_0}{r}, \quad r \in \mathbb{R}$$

zu lösen.

Ein Blick auf den Funktionsgraphen ([Abb. 6.1](#)) lässt vermuten, dass $f(r)$ eine (sogar gleichmäßig) konvexe Funktion für $r > 0$ ist, und eine Untersuchung der zweiten Ableitung, die überall positiv ist, zeigt die Richtigkeit dieser Vermutung. Wir sind also hier auch wieder in dem angenehmen Fall, dass es nur ein globales Minimum geben kann und keine weiteren lokalen. Ist der Radius r^* dieses Minimum, so kann man die zugehörige Höhe h der Getränkedose aus

$$h^* = \frac{V_0}{\pi (r^*)^2}$$

²⁸Die eigentlich erforderlichen zusätzlichen Beschränkungen $r \geq 0$ und $h \geq 0$ ignorieren wir hier in der Hoffnung, dass sie in der Lösung automatisch erfüllt sein werden.

²⁹Hier bietet sich eine kurze Prüfung der Einheiten an, um ggf. falsche Umformungen zu erkennen. Die Bedingung, dass r positiv sein sollte, wird wieder ignoriert.

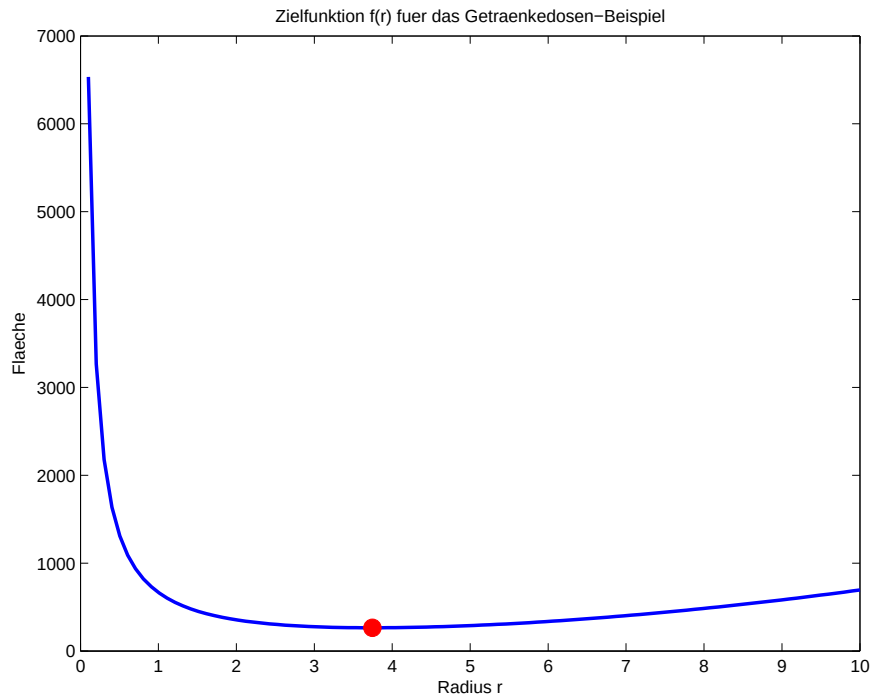


Abbildung 6.1.: Die reduzierte Zielfunktion $f(r)$ aus [Beispiel 6.1](#) mit eingezeichneter Lösung r^* .

zurückgewinnen. Das Paar (r^*, h^*) ist dann auch das globale Minimum für die ursprüngliche Aufgabe. Im Beispiel ergibt sich

$$r^* = 3.7449 \quad \text{und zugehörig} \quad h^* = 7.4899,$$

jeweils in cm. In [Abb. 6.2](#) ist die ursprüngliche (nicht reduzierte) Zielfunktion $F(r, h)$ dargestellt, zusammen mit dem Schnitt entlang der Beschränkung (6.2) und der Lösung (r^*, h^*) .

Diese Eliminationstechnik zur Rückführung von Optimierungsaufgaben mit Gleichungsbeschränkungen auf solche ohne Beschränkungen (in entsprechend weniger Variablen) muss man aber mit Vorsicht anwenden, wie das folgende Beispiel aus [Fletcher, 1987](#) zeigt:³⁰

Beispiel 6.2 (Vorsicht beim Eliminieren). *Die Aufgabe*

$$\begin{aligned} \text{Minimiere} \quad & x_1^2 + x_2^2, \quad x \in \mathbb{R}^2 \\ \text{unter} \quad & (x_1 - 1)^3 = x_2^2 \end{aligned} \tag{6.4}$$

hat das eindeutige globale Minimum $x^* = (1, 0)$, siehe [Abb. 6.3](#). Wir versuchen, x_2 zu eliminieren, und erhalten als reduzierte Zielfunktion

$$f(x_1) = x_1^2 + (x_1 - 1)^3.$$

Die Aufgabe, $f(x_1)$ über $x_1 \in \mathbb{R}$ zu minimieren, besitzt aber keine Lösung, da $f(x_1)$ nach unten nicht beschränkt ist: $f(x_1) \rightarrow -\infty$, wenn $x_1 \rightarrow -\infty$.

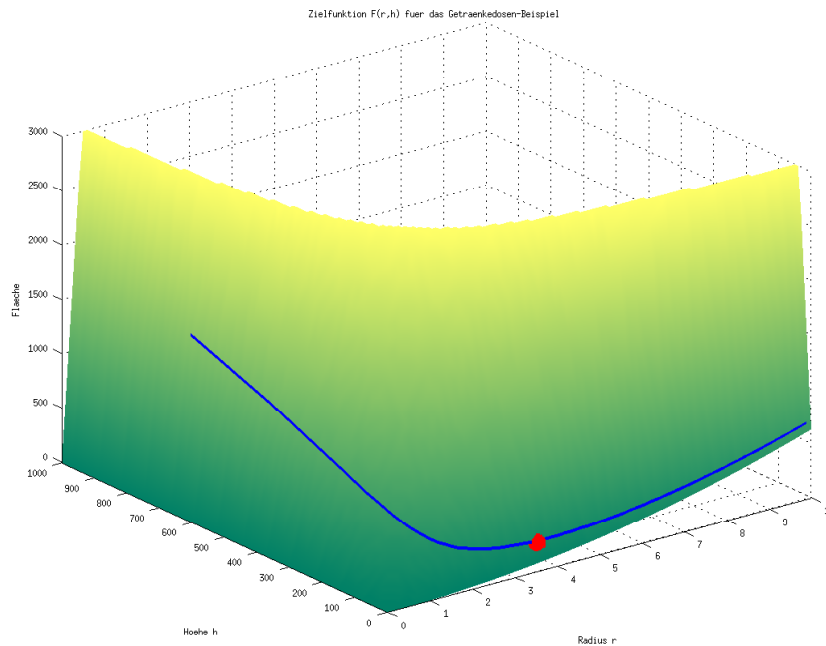


Abbildung 6.2.: Die Zielfunktion $F(r, h)$ aus [Beispiel 6.1](#) mit eingezeichneter Lösung (r^*, h^*) und dem Schnitt durch $F(r, h)$ entlang der Beschränkung (6.2).

Frage: Was haben wir hier übersehen?

Die Gleichungsnebenbedingung $(x_1 - 1)^3 = x_2^2$ enthält implizit die Bedingung $x_1 \geq 1$, die wir in der reduzierten Aufgabe vergessen haben zu stellen. Wir können hier also nicht wie in [Beispiel 6.1](#) auf eine unbeschränkte Aufgabe kommen, sondern müssten formulieren:

$$\begin{array}{ll} \text{Minimiere} & x_1^2 + (x_1 - 1)^3, \quad x_1 \in \mathbb{R} \\ \text{unter} & x_1 \geq 1. \end{array}$$

³⁰hier zitiert nach Nocedal, Wright, 2006, Example 15.2

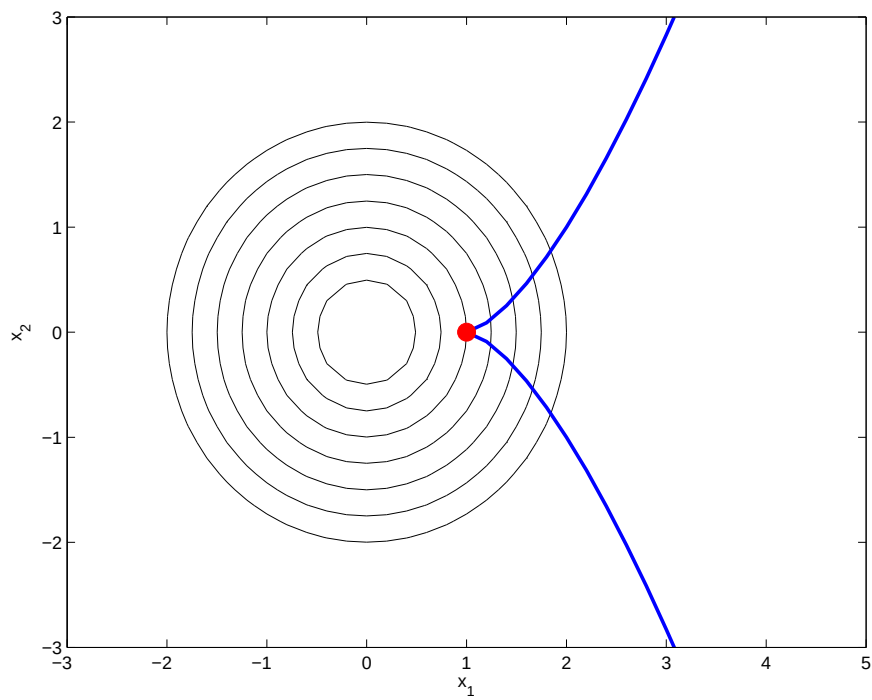


Abbildung 6.3.: Einige Höhenlinien der Zielfunktion aus [Beispiel 6.2](#) mit eingezeichneter Lösung $(x_1^*, x_2^*) = (1, 0)$ und der zulässigen Menge.

Ende der Woche 4

Kapitel 2 Wie gut ist mein Modell? (Parameteridentifikation und Kleinste-Quadrate-Aufgaben)

Eine Vielzahl von Aufgaben in den Ingenieur- und Wirtschaftswissenschaften ist von folgendem Typ: Gegeben ist ein funktionales Modell $g(\xi)$, welches einen Zusammenhang $\eta = g(\xi)$ zwischen den **unabhängigen** Variablen $\xi \in \mathbb{R}^r$ und einer davon **abhängigen** Größe $\eta \in \mathbb{R}$ modelliert. Dieses Modell enthält noch unbekannte **Parameter** $x \in \mathbb{R}^n$, wir schreiben also genauer:

$$\eta = g(x; \xi).$$

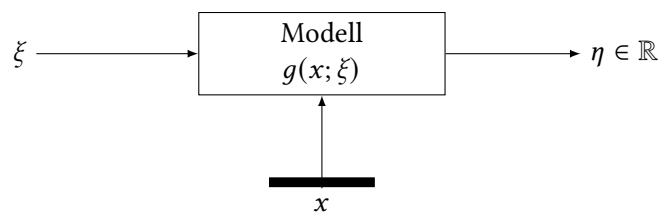


Abbildung 6.1.: Schematische Darstellung eines Modells mit Parametervektor $x \in \mathbb{R}^n$, der so einzustellen ist, dass die Modellvorhersage η möglichst gut mit den vorhandenen Messdaten übereinstimmt.

Die Aufgabe ist nun, anhand von vorliegenden Messdaten $(\xi_i, \eta_i) \in \mathbb{R}^r \times \mathbb{R}$, $i = 1, \dots, m$, den Parameter $x \in \mathbb{R}^n$ so zu wählen, dass das Modell den Messdaten möglichst gut entspricht. Das Auffinden des (in einem noch zu präzisierenden Sinne) besten Parametervektors $x \in \mathbb{R}^n$ wird als **Parameteridentifikation**, **Parameterschätzung**, **Modellkalibrierung**, **Ausgleichsrechnung**, **Datenassimilation** oder **Fitting** bezeichnet.

§ 7 EIN ERSTES BEISPIEL

Beispiel 7.1 (Kalibrierung einer Preis-Absatz-Funktion¹). Eine Sektkellerei möchte einen Rieslingsekt auf den Markt bringen. Für die Festlegung des Abgabepreises soll zunächst eine Preis-Absatz-Funktion ermittelt werden. Dazu wird in sechs Geschäften ein Testverkauf über jeweils einen Tag durchgeführt. Daraus erhält man folgende „Messdaten“:

¹Beispiel aus https://de.wikipedia.org/wiki/Einfache_lineare_Regression. Ein ähnlich gelagertes Beispiel mit einem einfachen Modell für den Bremsweg eines Autos wird in der Übung betrachtet.

Geschäft	1	2	3	4	5	6
Abgabepreis a in €	20	16	15	16	13	10
verkaufte Stückzahl S	0	3	7	4	6	10

Die Bezeichnungen der allgemeinen Aufgabe sind hier wie folgt belegt:

$$\begin{aligned}\xi &= \text{Abgabepreis } a && (\text{unabhängige Variable}) \in \mathbb{R} \\ \eta &= \text{verkaufte Stückzahl } S && (\text{abhängige Variable}) \in \mathbb{R}.\end{aligned}$$

Aus der Darstellung der $m = 6$ Datenpaare (Abb. 7.1) oder aus Vorwissen lässt sich die Vermutung gewinnen, dass ein linearer Zusammenhang zwischen S und a bestehen könnte:

$$S = g(x; a) := x_1 + x_2 a. \quad (7.1)$$

Dieser Zusammenhang ist unser Modell, das die noch unbekannten Parameter $x = (x_1, x_2) \in \mathbb{R}^2$ enthält.

Die Abb. 7.1 zeigt die vorliegenden Daten und einige mögliche (angesichts der Daten mehr oder weniger plausible) Preis-Absatz-Funktionen (7.1) zu verschiedenen Parametern x . Um

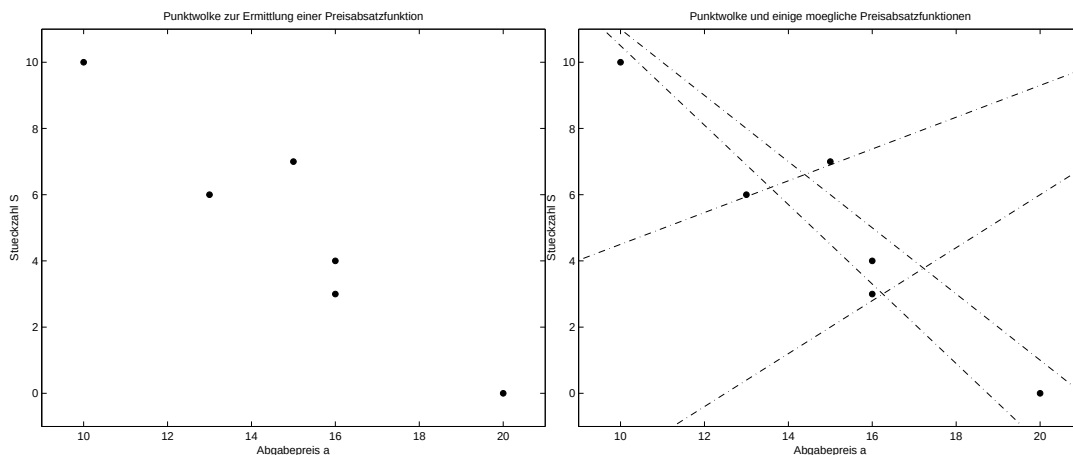


Abbildung 7.1.: Daten aus der Erhebung der Sektkellerei (Beispiel 7.1) und einige mögliche (mehr oder weniger sinnvolle) Preis-Absatz-Funktionen (7.1) zu verschiedenen Parametern x .

zu den *optimalen* Parametern x zu gelangen, also zu derjenigen Preis-Absatz-Funktion, bei der die Daten am besten „gefittet“ werden, benötigen wir ein Maß für die Güte des Fits. Dabei sollten die Abweichungen der Messwerte von der Modellfunktion eine Rolle spielen:

$$r_i(x) := \underbrace{g(x; a_i)}_{\text{Vorhersage des Modells}} - \underbrace{S_i}_{\text{Messwert}} \quad \text{für } i = 1, \dots, m. \quad (7.2)$$

Die Zahl $r_i(x)$ heißt das i -te **Residuum** (plural: **Residuen**). Ist $r_i(x)$ nahe bei null, so wird für die gewählten Modellparameter x das i -te Datenpaar (a_i, S_i) durch die Modellfunktion gut getroffen (vorhergesagt), siehe Abb. 7.2.

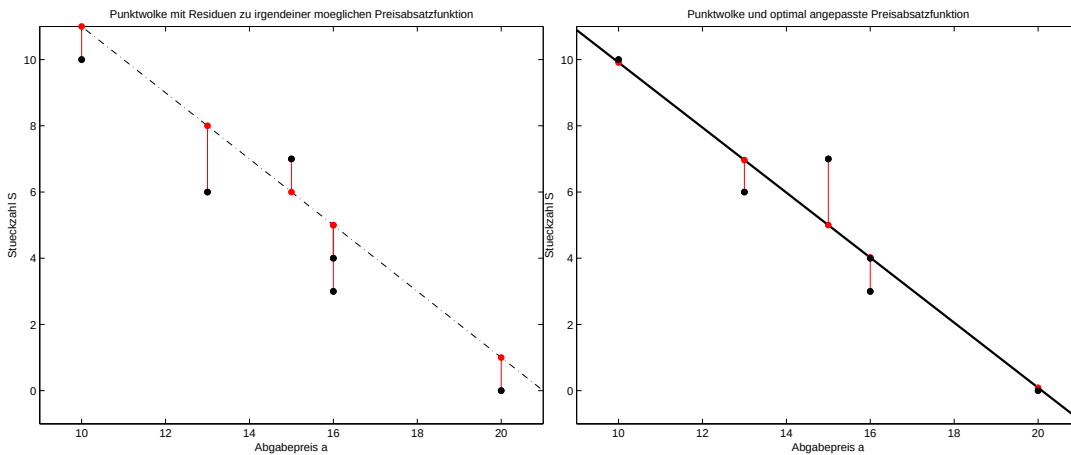


Abbildung 7.2.: Punktwolke zu [Beispiel 7.1](#) und die Residuen (in rot dargestellt) zu einer möglichen Preis-Absatz-Funktion (links). Die (im Sinne der kleinste Fehlerquadratsumme) *optimal* an die Daten angepasste Preis-Absatz-Funktion ist rechts zu sehen. Sie gehört zum Parameter x^* , der Lösung von (7.6).

Es wäre nun durchaus naheliegend, für eine Bewertung der Gesamtgüte der Vorhersage der Daten durch das Modell die Funktion²

$$f(x) := \sum_{i=1}^m |r_i(x)| = \sum_{i=1}^m |g(x; a_i) - S_i| = \left\| \begin{pmatrix} g(x; a_1) - S_1 \\ \vdots \\ g(x; a_m) - S_m \end{pmatrix} \right\|_1 \quad (7.3)$$

zu verwenden, also die Gesamtsumme der Abstände zwischen Messwerten und Modellvorhersage über alle Datenpunkte $i = 1, \dots, m$. Die Minimierung von $f(x)$ würde dann die im Sinne der kleinsten Fehlersumme optimalen Parameter x^* für die Modellfunktion liefern.

Obwohl (7.3) seine Berechtigung hat, ist es wesentlich gängiger, die Abweichungen der Modellfunktion von den Datenpunkten im Sinne der **kleinsten Fehlerquadratsumme** zu messen:³

$$\begin{aligned} f(x) &:= \sum_{i=1}^m |r_i(x)|^2 = \sum_{i=1}^m |g(x; a_i) - S_i|^2 = \sum_{i=1}^m (g(x; a_i) - S_i)^2 \\ &= \left\| \begin{pmatrix} g(x; a_1) - S_1 \\ \vdots \\ g(x; a_m) - S_m \end{pmatrix} \right\|_2^2. \end{aligned} \quad (7.4)$$

Während bei (7.3) jegliche Abweichungen proportional zu ihrer Größe eingehen, werden in (7.4) durch das Quadrieren kleine Abweichungen (Residuen) nur sehr wenig, große dagegen sehr stark gewichtet, vgl. [Abb. 7.3](#).

Wir wollen nun die im Sinne von (7.4) optimale Preis-Absatz-Funktion im [Beispiel 7.1](#) finden.

²Dies ist gerade die sogenannte 1-Norm des Vektors der Residuen.

³Eine Erklärung für die Popularität von (7.4) folgt später noch.

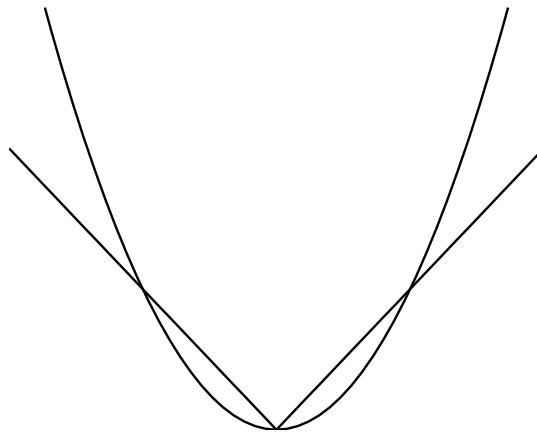


Abbildung 7.3.: Die Funktionen $r \mapsto |r|$ und $r \mapsto r^2$ im Vergleich. Dies illustriert die Funktion der kleinsten Fehlersumme (7.3), bei der die Residuen proportional zu ihrer absoluten Größe eingehen bzw. die Funktion der kleinsten Fehlerquadratsumme (7.4), bei der kleine Residuen wenig, große jedoch stark gewichtet werden.

Dazu formulieren wir die Optimierungsaufgabe:

$$\text{Minimiere } \sum_{i=1}^m (g(x; a_i) - S_i)^2, \quad x \in \mathbb{R}^2. \quad (7.5)$$

Ein solches Optimierungsproblem heißt eine **Kleinste-Quadrate-Aufgabe**. In unserem ersten [Beispiel 7.1](#) sollte ein linearer Zusammenhang (7.1) zwischen a und S angenommen werden. Also lautet die Aufgabe genauer:

$$\text{Minimiere } \sum_{i=1}^m (x_1 + x_2 a_i - S_i)^2, \quad x \in \mathbb{R}^2. \quad (7.6)$$

Zur Erinnerung: Die Werte (a_i, S_i) für $i = 1, \dots, m$ sind dabei gegebene Datenpaare (Messwerte).

Zur Lösung von Kleinste-Quadrate-(Parameteridentifikations-)Aufgaben wie (7.5) steht in der MATLAB-Optimierungs-Toolbox die Routine `lsqcurvefit` zur Verfügung.⁴ Dieser müssen wir lediglich die Modellfunktion, hier $g(x; a)$ aus Gleichung (7.1), zur Verfügung stellen sowie die Datenpaare.⁵ Wir implementieren die Modellfunktion etwa in einer Datei `Preisabsatzfunktion_Modellfunktion.m`:

```
function S = Preisabsatzfunktion_Modellfunktion(x,a)

% Bedeutung der Variablen (Modellparameter)
% x(1) = Offset
% x(2) = Steigung

% Berechnung der Modellfunktion
S = x(1) + x(2) * a;
```

⁴Die Abkürzung `lsq` steht für *least squares*, also kleinste Quadrate.

⁵**Achtung:** Es muss die Modellfunktion (7.1) programmiert werden, nicht etwa die Zielfunktion (7.6)!

Zur Abkürzung bezeichnen wir die Modellfunktion im Hauptprogramm namens Preisabsatzfunktion.m mit $g(x,a)$:

Matlab/Preisabsatzfunktion.m

```
% Setzen der Modellfunktion  $g(x_1, x_2; a) = x_1 + x_2 * a$ 
% Bedeutung der Variablen (Modellparameter)
%  $x(1)$  = Offset
%  $x(2)$  = Steigung
g = @(x,a) Preisabsatzfunktion_Modellfunktion(x,a);
```

Die Daten der Aufgabe, also die Tabelle mit den Datenpaaren, geben wir als Vektoren a und S ein:⁶

Matlab/Daten_Preisabsatzfunktion.m

```
% Abgabepreis
a = [20 16 15 16 13 10]';

% Stueckzahl
S = [0 3 7 4 6 10]';
```

Aus Abb. 7.1 kann man als grobe Schätzung die Funktion $g(x; a) = 20 - x$ als näherungsweise Preis-Absatz-Funktion ablesen. Wir setzen also $x^{(0)} = (20, -1)$ als Startschätzung an:

Matlab/Preisabsatzfunktion.m

```
% Setzen einer Startschätzung
x0 = [20; -1];
```

Wir prüfen kurz die korrekte Arbeitsweise unserer Modellfunktion:

Matlab/Preisabsatzfunktion.m

```
% % Ueberpruefen der Modellfunktion
%  $g(x_0, a)$ 
```

Schließlich setzen wir noch einige Optionen für das Optimierungsverfahren. Mit diesen wählen wir einen von zwei in `lsqcurvefit` verfügbaren Algorithmen, das sogenannte Levenberg-Marquardt-Verfahren (dazu später mehr) aus. Außerdem fordern wir detaillierte Informationen über die Aufgabe und den Fortschritt des Verfahrens an und rufen `lsqcurvefit` mit einer erweiterten Liste von Rückgabeparametern auf, um mehr Informationen zu erhalten:

Matlab/Preisabsatzfunktion.m

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('lsqcurvefit', 'Algorithm', 'Levenberg-Marquardt',
    'Diagnostics', 'on', 'Display', 'iter-detailed');

% Aufruf des Optimierungsverfahrens
[xstar, resnorm, residual, exitflag, output, lambda, jacobian] = lsqcurvefit
    (g, x0, a, S, [], [], options);
```

Die beiden leeren Argumente `[]` signalisieren, dass wir an die Modellparameter x in unserem Beispiel weder untere noch obere Beschränkungen stellen.

Die Ausgabe von `lsqcurvefit` sieht etwa wie folgt aus:

⁶Bei umfangreicheren Datensätzen würde man diese aus praktischen Gründen aus einer Datei einlesen.

Diagnostic Information					
Number of variables: 2					
Functions					
Objective:	lsqcurvefit/objective				
Gradient:	finite-differencing				
Number of lower bound constraints:	0				
Number of upper bound constraints:	0				
Algorithm selected					
levenberg-marquardt					

End diagnostic information					
Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	3	6	1	0.01	
1	6	5.98217	0.00257	0.001	0.257627
2	9	5.98214	1.08e-05	0.0001	0.0107793
3	12	5.98214	4.52e-09	1e-05	4.52722e-05
Optimization stopped because the relative sum of squares (r) is changing by less than options.FunctionTolerance = 1.000000e-06.					
Optimization Metric			Options		
relative change r =	8.16e-11		FunctionTolerance =	1e-06 (default)	

Wir werden diese Ausgabe wieder weiter unten kommentieren, sobald wir den Algorithmus hinter `lsqcurvefit` genauer kennen. Die von `lsqcurvefit` berechnete Lösung steht jetzt in der Variablen `xstar`:

$$x^* = \begin{pmatrix} 19.7321 \\ -0.9821 \end{pmatrix}.$$

Unsere gewählte Startschätzung $x^{(0)} = (20, -1)$ war also schon recht gut. In `residual` steht der Vektor der Residuen, hier⁷

$$r(x^*) = \begin{pmatrix} 0.0893 \\ 1.0179 \\ -2.0000 \\ 0.0179 \\ 0.9643 \\ -0.0893 \end{pmatrix},$$

und in `resnorm` der Wert $\|r(x^*)\|^2$, also der Wert der Kleinste-Quadrate-Zielfunktion (7.4) in der Lösung x^* , hier

$$\|r(x^*)\|^2 = 5.9821.$$

Selbst bei optimal angepasster Modellfunktion liegen also nicht alle Datenpunkte genau auf der Kurve, wie auch in [Abb. 7.2](#) (rechts) dargestellt. Dies ist der Normalfall, da in der Praxis

⁷Positive Werte bedeuten hier, dass die Modellfunktion *über* dem Messwert liegt, vgl. (7.2).

immer Messfehler auftreten und auch das Modell die Wirklichkeit oft nicht vollkommen korrekt beschreibt.

Ende der Woche 5

§ 8 DAS VERFAHREN HINTER `lsqcurvefit` UND `lsqnonlin`

Zur Beschreibung des Verfahrens kehren wir zunächst kurz zurück zur allgemeinen Notation vom Anfang von [Kapitel 2](#). Die Modellfunktion $g(x; \xi)$ – mit den noch unbekannten Modellparametern $x \in \mathbb{R}^n$ – stellt den Zusammenhang her zwischen den unabhängigen Variablen $\xi \in \mathbb{R}^r$ und den abhängigen Variablen $\eta \in \mathbb{R}$. Gegeben sind die Datenpaare $(\xi_i, \eta_i) \in \mathbb{R}^r \times \mathbb{R}$, $i = 1, \dots, m$. Daraus ergeben sich die Residuen⁸

$$r_i(x) := g(x; \xi_i) - \eta_i, \quad i = 1, \dots, m. \quad (8.1)$$

Gesucht ist ein globales (oder zumindest lokales) Minimum der Kleinste-Quadrate-Aufgabe

$$\text{Minimiere} \quad \sum_{i=1}^m r_i(x)^2, \quad x \in \mathbb{R}^n. \quad (8.2)$$

Zur Lösung der Optimierungsaufgabe (8.2) gibt es in der MATLAB-Optimierungs-Toolbox neben `lsqcurvefit` eine weitere Funktion: `lsqnonlin`. Genauer gesagt ist `lsqcurvefit` lediglich ein anderer Aufruf von `lsqnonlin`, der für Fitting-Aufgaben wie [Beispiel 7.1](#) praktisch ist, also wenn die Residuen in der Form (8.1) vorliegen.⁹ Der Unterschied zwischen beiden Routinen liegt lediglich in der Formulierung der Residuenfunktion, siehe auch [Tabelle 8.1](#).¹⁰ Wir besprechen später (in § 10) noch eine Aufgabe vom Typ (8.2), die keine Fitting-Aufgabe ist.

Beachte: Weder bei `lsqcurvefit` noch bei `lsqnonlin` muss der Benutzer die eigentliche Zielfunktion des Optimierungsproblems (8.2) programmieren. Die Routinen wissen ja, dass ein Kleinste-Quadrate-Problem gelöst werden soll.

Da `lsqcurvefit` und `lsqnonlin` im Grunde dieselben Aufgabe lösen, gehen wir zur Beschreibung des dahinterstehenden Algorithmus' jetzt von der Form (8.2) aus. Zur Abkürzung der

⁸Als Abweichung von diesem gewohnten Modell-Messpaare-Schema betrachten wir in der Übung eine Aufgabe, in der zwei Längen einer Pyramide bestimmt werden sollen. Da die Komponenten der Residuenfunktion dort alle unterschiedlich sind, muss entweder eine vektorwertige Modellfunktion in `lsqcurvefit` verwendet werden, oder die Residuenfunktion selbst wird modelliert und `lsqnonlin` verwendet.

⁹In der Übung wird eine Positionsbestimmung für ein Flugzeug mittels Funkfeuer modelliert. Hierbei ist es geschickter, die Residuen direkt zu berechnen und `lsqnonlin` zu verwenden.

¹⁰Nicht alle Kleinste-Quadrate-Aufgaben (8.2) besitzen Residuenfunktionen der Form (8.1), siehe etwa Flugzeugpeilung. Eine andere Aufgabe vom Typ ist die Bestimmung eines Zylinders (parametrisiert durch Aufpunkt, Mittelachsenrichtung und Radius), auf dem eine gegebene Menge Punkte möglichst gut liegen soll. Hierbei kann man als Residuen den Abstand der Punkte zur Mittelachse - Radius nehmen. Zusätzlich ist dabei ein 1D-Unterraum unbestimmt, da man den Aufpunkt beliebig in Richtung der Mittelachse verschieben kann, ohne die Residuen zu ändern. Schließlich kann man noch die Länge (und Orientierung) des Vektors der Mittelachse beliebig wählen, eine weitere 1D-Mannigfaltigkeit.

	lsqcurvefit	lsqnonlin
Residuenfunktionen	$r_i(x) = g(x; \xi_i) - \eta_i$	$r_i(x)$ beliebig
zu programmierende Funktion	Modellfunktion $g(x; \xi)$	Residuenfunktion $r(x)$
obligatorische Problem Daten	Datenpaare (ξ_i, η_i)	—

Tabelle 8.1.: `lsqcurvefit` löst Fitting-Aufgaben wie [Beispiel 7.1](#), `lsqnonlin` dagegen beliebige Kleinste-Quadrate-Aufgaben (8.2).

Notation ist es praktisch, die Residuen zu einem Residuenvektor zusammenzufassen:

$$r(x) := \begin{pmatrix} r_1(x) \\ \vdots \\ r_m(x) \end{pmatrix} \in \mathbb{R}^m.$$

Die vektorielle Residuenfunktion bildet $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ab. Dadurch können wir die Kleinste-Quadrate-Aufgabe (8.2) auch kurz schreiben als

$$\text{Minimiere } \|r(x)\|^2, \quad x \in \mathbb{R}^n. \quad (8.3)$$

Da es sich hier um einen Spezialfall einer freien Optimierungsaufgaben handelt, sind die Ergebnisse aus [Kapitel 1](#) anwendbar. Insbesondere können und werden Optimierungsverfahren wie `lsqnonlin` die Optimalitätsbedingungen 1. und 2. Ordnung ([Satz 2.3](#), [Satz 2.5](#)) benutzen, um eine Lösung von (8.3) zu erkennen und zu finden.

Um dies nachzuvollziehen, bestimmen wir die erste und zweite Ableitung der Zielfunktion aus (8.3) mit Hilfe der Kettenregel. Dabei wird die Ableitung der Residuenfunktion $r(x)$ vorkommen, also deren Jacobimatrix, die wir mit $r'(x)$ oder auch einfach mit $J(x)$ bezeichnen:

$$J(x) := \left[\frac{\partial}{\partial x_j} r_i(x) \right]_{ij} = \begin{bmatrix} \left| \frac{\partial}{\partial x_1} r(x) \right| & \cdots & \left| \frac{\partial}{\partial x_n} r(x) \right| \\ \vdots & & \vdots \\ \left| \frac{\partial}{\partial x_1} r_m(x) \right| & \cdots & \left| \frac{\partial}{\partial x_n} r_m(x) \right| \end{bmatrix} = \begin{bmatrix} -\nabla r_1(x)^\top - \\ \vdots \\ -\nabla r_m(x)^\top - \end{bmatrix} \in \mathbb{R}^{m \times n}. \quad (8.4)$$

Aus kosmetischen Gründen ist es praktisch, die Zielfunktion in (8.3) noch durch 2 zu teilen.¹¹ Auf die Lage der Minimierer hat dies natürlich keinen Einfluss. Die so erhaltene Zielfunktion leiten wir nun ab:

$$f(x) = \frac{1}{2} \|r(x)\|^2 = \frac{1}{2} r(x)^\top r(x) \quad \text{Zielfunktion} \quad (8.5a)$$

$$\nabla f(x) = J(x)^\top r(x) = \sum_{i=1}^m r_i(x) \nabla r_i(x) \quad \text{1. Ableitung} \quad (8.5b)$$

$$\nabla^2 f(x) = J(x)^\top J(x) + \sum_{i=1}^m r_i(x) \nabla^2 r_i(x) \quad \text{2. Ableitung.} \quad (8.5c)$$

¹¹Das Levenberg-Marquardt-Verfahren in `lsqnonlin` tut dies auch, ohne dass das jedoch in der Hilfe erwähnt wird.

Mit Hilfe dieser Daten könnte man, so wie es das Quasi-Newton-Verfahren in `fminunc` tut (siehe § 3), an einer Iterierten $x^{(k)}$ ein quadratisches Modell der Zielfunktion $f(x)$ aufstellen, durch Minimierung eine Suchrichtung $d^{(k)}$ bestimmen, dazu eine Schrittweite $\alpha^{(k)}$, dann den Schritt gehen usw. Die Zusatzinformation, dass es sich hier um ein Kleinste-Quadrate-Problem handelt, lässt sich jedoch algorithmisch noch besser ausnutzen. Wir beschreiben dies am Beispiel des sehr populären **Levenberg-Marquardt-Verfahrens**¹², das wir ja auch oben im Beispiel durch `optimoptions` ausgewählt hatten.

Das Levenberg-Marquardt-Verfahren baut an der Stelle $x^{(k)}$ ein bestimmtes quadratisches Modell auf, vgl. (3.2):

$$\begin{aligned} q_{\text{LM}}^{(k)}(d) &:= f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top B_{\text{LM}}^{(k)} d \\ &\stackrel{(8.5)}{=} \frac{1}{2} r(x^{(k)})^\top r(x^{(k)}) + r(x^{(k)})^\top J(x^{(k)}) d + \frac{1}{2} d^\top B_{\text{LM}}^{(k)} d. \end{aligned} \quad (8.6)$$

Im Unterschied zu einem Quasi-Newton-Verfahren wie `fminunc` wird dabei jedoch keine Quasi-Newton-Approximation von $\nabla^2 f(x^{(k)})$ als Modell-Hessematrix verwendet, sondern

$$B_{\text{LM}}^{(k)} := J(x^{(k)})^\top J(x^{(k)}) + \lambda^{(k)} I. \quad (8.7)$$

Dabei ist $\lambda^{(k)} > 0$ eine positive Zahl und I die Einheitsmatrix in $\mathbb{R}^{n \times n}$. Dies hat folgende Vorteile:

- (i) Man nutzt den „guten“ ersten Anteil $J(x)^\top J(x)$ der echten Hessematrix (8.5c). Dieser ist immerhin positiv semidefinit und lässt sich außerdem ohne zusätzlichen Aufwand bestimmen, denn $J(x)$ wird ohnehin schon für die erste Ableitung $\nabla f(x)$ benötigt.
- (ii) Den „schlechten“ Anteil $\sum_{i=1}^m r_i(x) \nabla^2 r_i(x)$ der Hessematrix (8.5c) lässt man weg. Dieser ist durch das Vorkommen der zweiten Ableitungen $\nabla^2 r_i(x)$ aufwändig zu berechnen, und außerdem könnte er die positive Definitheit zerstören. Weiterhin haben wir die Hoffnung, dass in einer Lösung x^* die Residuen $r_i(x^*)$ ohnehin betragsmäßig klein (nahe Null) sind, dass wir also einen guten Fit erhalten und dieser Term daher vernachlässigt werden kann.
- (iii) Stattdessen addiert man den Ersatzterm $\lambda^{(k)} I$. Dieser sorgt durch $\lambda^{(k)} > 0$ für die positive Definitheit der Modell-Hessematrix $B_{\text{LM}}^{(k)}$.

Wie schon beim Quasi-Newton-Verfahren `fminunc` ist man in der k -ten Iteration am Minimum des quadratischen Modells (8.6) interessiert. Wegen der positiven Definitheit von $B_{\text{LM}}^{(k)}$ ist dieses Minimum eindeutig und lässt sich, vgl. (3.3), über das lineare Gleichungssystem

$$B_{\text{LM}}^{(k)} d^{(k)} = -\nabla f(x^{(k)}) \quad (8.8)$$

bzw. ausgeschrieben, siehe (8.7) und (8.5b), über

$$[J(x^{(k)})^\top J(x^{(k)}) + \lambda^{(k)} I] d^{(k)} = -J(x^{(k)})^\top r(x^{(k)}) \quad (8.9)$$

berechnen. Die positive Definitheit von $B_{\text{LM}}^{(k)}$ und damit von $(B_{\text{LM}}^{(k)})^{-1}$ sorgt wieder dafür, dass $d^{(k)}$ eine Abstiegsrichtung für unsere Kleinste-Quadrate-Zielfunktion $f(x)$ an der Stelle $x^{(k)}$ ist:

$$\nabla f(x^{(k)})^\top d^{(k)} \stackrel{(8.8)}{=} -\nabla f(x^{(k)})^\top (B_{\text{LM}}^{(k)})^{-1} \nabla f(x^{(k)}) < 0,$$

¹²vorgeschlagen von Levenberg (1944), wiederentdeckt von Marquardt (1963)

falls $\nabla f(x^{(k)}) \neq 0$.

Wir könnten also jetzt wie beim Quasi-Newton-Verfahren (vgl. § 3) eine Liniensuche durchführen, um eine geeignete Schrittlänge $\alpha^{(k)}$ zu bestimmen und dann $x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}$ zu setzen. Das Levenberg-Marquardt-Verfahren geht jedoch etwas anders vor: Es verzichtet auf die Liniensuche, verwendet also immer $\alpha^{(k)} = 1$, und steuert stattdessen die Länge des Schrittes, also $\|d^{(k)}\|$, über den Parameter $\lambda^{(k)}$. Wir betrachten dazu die Bestimmungsgleichung (8.9) für $d^{(k)}$:

(i) Ist $\lambda^{(k)}$ sehr groß, so lautet (8.9) in etwa

$$\lambda^{(k)} d^{(k)} \approx -J(x^{(k)})^T r(x^{(k)}) \quad \text{bzw.} \quad d^{(k)} \approx -\frac{1}{\lambda^{(k)}} \underbrace{J(x^{(k)})^T r(x^{(k)})}_{\nabla f(x^{(k)})}. \quad (8.10)$$

Das bedeutet, der Schritt $d^{(k)}$ entspricht einem kleinen Schritt in Richtung des negativen Gradienten, also einem kleinen Schritt in Richtung des steilsten Abstiegs.

(ii) Ist $\lambda^{(k)}$ dagegen sehr klein, so lautet (8.9) in etwa

$$J(x^{(k)})^T J(x^{(k)}) d^{(k)} = -J(x^{(k)})^T r(x^{(k)}). \quad (8.11)$$

Das heißt, der Schritt $d^{(k)}$ entspricht einem sogenannten vereinfachten Newton-Schritt, wobei die Vereinfachung darin besteht, dass an Stelle der Hessematrix $\nabla^2 f(x^{(k)})$ nur der „gute“ erste Anteil in (8.5c) verwendet wird.

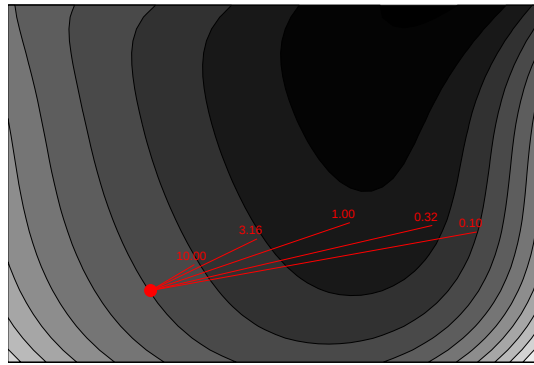


Abbildung 8.1.: Illustration des Einflusses von $\lambda^{(k)}$ auf den in (8.9) berechneten Schritt $d^{(k)}$ im Levenberg-Marquardt-Verfahren. Große Werte von $\lambda^{(k)}$ führen zu kurzen, gradientenähnlichen Schritten, siehe (8.10). Diese sind auch daran erkennbar, dass sie etwa senkrecht auf der Kontourlinie stehen. Kleine Werte von $\lambda^{(k)}$ führen dagegen zu vereinfachten Newton-Schritten (8.11), die i. A. in eine andere Richtung zeigen.

Durch die Wahl von $\lambda^{(k)}$ kann stufenlos zwischen dem „vorsichtigen“ Gradientenschritt (8.10) und dem „mutigeren“ vereinfachten Newton-Schritt (8.11) hin- und hergeschaltet werden, siehe auch Abb. 8.1. Deshalb wird $\lambda^{(k)}$ auch als **Dämpfungsparameter** bezeichnet.

Ob der gewählte Wert für $\lambda^{(k)}$ und der damit berechnete Schritt $d^{(k)}$ gut oder schlecht war,

wird mit Hilfe des sogenannten **Fortschrittsquotienten** entschieden:

$$\rho^{(k)} := \frac{f(x^{(k)}) - f(x^{(k)} + d^{(k)})}{q_{\text{LM}}^{(k)}(0) - q_{\text{LM}}^{(k)}(d^{(k)})}. \quad (8.12)$$

Dabei gibt der Zähler den Abstieg in der Kleinste-Quadrate-Zielfunktion $f(x)$ an, den man durch Ausführen des berechneten Schrittes von $x^{(k)}$ nach $x^{(k)} + d^{(k)}$ erhalten würde. Der Nenner gibt analog den Abstieg in der quadratischen Modellfunktion $q_{\text{LM}}^{(k)}(d)$ an. Durch den Quotienten erhält man einen Vergleich zwischen tatsächlichem und vorhergesagtem Abstieg:

$$\rho^{(k)} = \frac{\text{in der Zielfunktion } f(x) \text{ tatsächlich erreichter Abstieg}}{\text{durch die Modellfunktion } q_{\text{LM}}^{(k)}(d) \text{ vorhergesagter Abstieg}}.$$

Abhängig vom Wert von $\rho^{(k)}$ wird nun entschieden:¹³

- (i) Ist $\rho^{(k)} \geq 0.75$, dann bewerten wir den Schritt $x^{(k)} \rightsquigarrow x^{(k)} + d^{(k)}$ als sehr erfolgreich. Beim nächsten Mal können wir „mutiger“ sein, also setzen wir $\lambda^{(k+1)}$ kleiner als $\lambda^{(k)}$ an, z. B. $\lambda^{(k+1)} := \lambda^{(k)} / 3$.
- (ii) Ist $0.25 \leq \rho^{(k)} < 0.75$, dann ist der Schritt $x^{(k)} \rightsquigarrow x^{(k)} + d^{(k)}$ immerhin mäßig erfolgreich. Wir wählen $\lambda^{(k+1)} := \lambda^{(k)}$.
- (iii) Ist $\rho^{(k)} < 0.25$, dann ist der Schritt $x^{(k)} \rightsquigarrow x^{(k)} + d^{(k)}$ nicht sehr erfolgreich. Wir wollen beim nächsten Mal vorsichtiger sein und setzen $\lambda^{(k+1)}$ größer als $\lambda^{(k)}$, z. B. $\lambda^{(k+1)} := 2 \lambda^{(k)}$.

Ist sogar $\rho^{(k)} \leq 0$, dann würden wir in der Zielfunktion einen „negativen Abstieg“ erhalten, denn der Nenner in $\rho^{(k)}$ in (8.12) ist immer positiv. Die Zielfunktion würde sich also durch Ausführung des Schrittes $x^{(k)} \rightsquigarrow x^{(k)} + d^{(k)}$ wieder vergrößern. In diesem Fall verwerfen wir den Schritt und berechnen mit dem vergrößerten $\lambda^{(k+1)}$ einen neuen. Im Normalfall $\rho^{(k)} > 0$ wird der Schritt jedoch akzeptiert.

Das Levenberg-Marquardt-Verfahren geht grob zusammengefasst wie folgt vor: In jedem Schritt (jeder Iteration) des Verfahrens wird

- (1) das quadratische Modell $q_{\text{LM}}^{(k)}(d)$ der Zielfunktion aufgebaut, siehe (8.6) und (8.7), dazu werden $r(x^{(k)})$ und $J(x^{(k)})$ ausgewertet,
- (2) das eindeutige Minimum $d^{(k)} \in \mathbb{R}^n$ dieses Modells durch Lösen des linearen Gleichungssystems (8.8) bestimmt,
- (3) der Fortschrittsquotient $\rho^{(k)}$ in (8.12) ausgewertet,
- (4) der nächste Dämpfungsparemeter $\lambda^{(k+1)}$ gewählt
- (5) und die nächste Iterierte

$$\text{falls } \rho^{(k)} > 0: \quad x^{(k+1)} := x^{(k)} + d^{(k)} \quad (\text{Schritt wird akzeptiert})$$

$$\text{falls } \rho^{(k)} \leq 0: \quad x^{(k+1)} := x^{(k)} \quad (\text{Schritt wird verworfen})$$

gesetzt.

¹³Die angegebenen Zahlenwerte 0.75 und 0.25 und die Faktoren 2 bzw. 1/3 dienen nur der Illustration und können bei einem konkreten Levenberg-Marquardt-Verfahren anders gewählt sein.

Wir können mit diesem Wissen nun die Ausgabe von `lsqcurvefit` am obigen Beispiel interpretieren:

Diagnostic Information					
Number of variables: 2					
Functions					
Objective:	lsqcurvefit/objective				
Gradient:	finite-differencing				
Number of lower bound constraints:	0				
Number of upper bound constraints:	0				
Algorithm selected					
levenberg-marquardt					

End diagnostic information					
Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	3	6	1	0.01	
1	6	5.98217	0.00257	0.001	0.257627
2	9	5.98214	1.08e-05	0.0001	0.0107793
3	12	5.98214	4.52e-09	1e-05	4.52722e-05
Optimization stopped because the relative sum of squares (r) is changing by less than options.FunctionTolerance = 1.000000e-06.					
Optimization Metric			Options		
relative change r =	8.16e-11		FunctionTolerance =	1e-06 (default)	

- Die Spalte `Iteration` bezeichnet unseren Iterationszähler $k = 0, 1, \dots$
- In der Spalte `Func-count` finden wir die Gesamtanzahl der Auswertungen unserer Modellfunktion $g(x; \xi)$ bzw. der Residuenfunktion $r(x)$. Da wir bisher von diesen Funktionen keine Ableitung zur Verfügung gestellt haben, muss sich `lsqcurvefit` in jeder Iteration die Jacobimatrix $J(x^{(k)})$ durch eine Differenzenapproximation beschaffen, wie dies auch in unserem ersten Beispiel für `fminunc` der Fall war, vgl. § 4. Die j -te Spalte der Jacobimatrix (8.4) wird dabei näherungsweise durch

$$\frac{\partial r(x)}{\partial x_j} \approx \frac{r(x + h e_j) - r(x)}{h} \in \mathbb{R}^m \quad (8.13)$$

mit einer kleinen Zahl $h > 0$ bestimmt. Da wir im Beispiel zwei Modellparameter $x = (x_1, x_2) \in \mathbb{R}^2$ haben, werden neben $r(x^{(k)})$ pro Iteration zwei weitere Auswertungen der Residuenfunktion benötigt, daher ergeben sich wieder Vielfache von drei bei der Zählung.

- Die etwas ungenau bezeichnete Spalte `Residual` zeigt das Quadrat der Norm der Residuenfunktion $\|r(x^{(k)})\|^2$ an der jeweils aktuellen Iterierten. Man erkennt, dass das Levenberg-Marquardt-Verfahren ein **Abstiegsverfahren** ist, d. h. die Werte unserer

Zielfunktion $f(x^{(k)}) = (1/2)\|r(x^{(k)})\|^2$ und damit auch die Werte von $\|r(x^{(k)})\|$ sind monoton fallend.¹⁴ Ab $k = 2$ ändern sich die hier sichtbaren Nachkommastellen nicht mehr. Dennoch macht das Verfahren noch Fortschritte, wie man an der nächsten Spalte sieht.

- In der Spalte `First-Order optimality` werden die Werte von $\|\nabla f(x^{(k)})\| = \|J(x^{(k)})^\top r(x^{(k)})\|$ aufgeführt. Wir suchen ja ein Minimum x^* der Kleinste-Quadrate-Zielfunktion $f(x)$, an dem also die notwendige Bedingung 1. Ordnung (Satz 2.3) $\nabla f(x^*) = 0$ gilt.
- In der Spalte `Lambda` sieht man die Wahl von $\lambda^{(k)}$ in der jeweiligen Iteration. Offenbar waren hier alle Schritte, gemessen mit dem Fortschrittsquotienten $\rho^{(k)}$, sehr erfolgreich, sodass $\lambda^{(k)}$ in jedem Schritt verringert wurde, hier um den Faktor 10.
- Schließlich finden sich in der letzten Spalte die Werte von $\|d^{(k+1)}\|$, also die Schrittlänge des Schrittes von $x^{(k)}$ nach $x^{(k+1)}$.

Der Grund des Abbruchs wird auch durch die Rückgabewariable `exitflag` reflektiert, die hier den Wert 3 hat. Laut Dokumentation von `lsqcurvefit` bedeutet dies: `Change in RESNORM too small`, also im Beispiel:

$$\frac{f(x^{(2)}) - f(x^{(3)})}{f(x^{(3)})} = \frac{\|r(x^{(2)})\|^2 - \|r(x^{(3)})\|^2}{\|r(x^{(3)})\|^2} \leq 10^{-6}.$$

Wir wollen das mit `lsqcurvefit` erhaltene Ergebnis von [Beispiel 7.1](#) noch kurz diskutieren. In der Lösung x^* ist die Zielfunktion mit dem Wert $2 f(x^*) = \|r(x^*)\|^2 = 5.98214$ im Beispiel nicht wesentlich kleiner als am Startwert $x^{(0)}$, da dieser bereits recht gut gewählt war. In der Lösung ist der Residuenvektor (vgl. [Abb. 7.2](#) rechts)

$$r(x^*) = \begin{pmatrix} 0.0893 \\ 1.0179 \\ -2.0000 \\ 0.0179 \\ 0.9643 \\ -0.0893 \end{pmatrix}$$

noch relativ weit vom Nullvektor entfernt. Die gegebenen Datenpaare (a_i, S_i) lassen sich mit einer linearen Preis-Absatz-Funktion nicht allzu gut „fitten“. Dies kann daran liegen, dass eine lineare Funktion nicht sachgemäß ist, oder hier wohl eher daran, dass die Messwerte (hier das Kaufverhalten der Testkunden) natürlich zufälligen Schwankungen unterworfen ist.

Bemerkung 8.1 (zu Kleinste-Quadrate-Aufgaben (8.1)–(8.2)).

- (i) Um Parameter in Modellen anhand von Messdaten zuverlässig identifizieren zu können, müssen mehr Datenpaare (ξ_i, η_i) , $i = 1, \dots, m$ vorhanden sein, als Parameter $x \in \mathbb{R}^n$ verwendet werden. Es muss also $m > n$ gelten. Man erkennt dies auch daran, dass die Jacobimatrix (8.4) der Residuenfunktionen eine „schlanke“ Matrix ist (höher als breit). Im [Beispiel 7.1](#) war dies wegen $m = 6 > 2 = n$ erfüllt.

¹⁴Wir werfen ja den Schritt $d^{(k)}$, falls $\rho^{(k)} < 0$ sein sollte, falls also kein Abstieg in der Zielfunktion $f(x) = \frac{1}{2}\|r(x)\|^2$ vorliegt.

- (ii) Ob die Parameter zuverlässig identifiziert wurden oder diese eher unsicher sind, lässt sich mit Hilfe der Jacobimatrix $J(x^*)$ in der Lösung x^* bzw. genauer mit Hilfe der Singulärwerte $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$ von $J(x^*)$ feststellen. Kleine Singulärwerte stehen für eine schlechte Identifizierbarkeit und eine empfindliche Abhängigkeit der Lösung von Messfehlern.
- (iii) Unser [Beispiel 7.1](#) war ein sogenanntes **lineares Kleinste-Quadrate-Problem**, denn die in (7.1) definierte zugehörige Modellfunktion $g(x; a) = x_1 + x_2 a$ und damit auch der Vektor der Residuenfunktionen $r(x)$ ist eine (affin-)lineare Funktion der unbekannten Parameter x . Deshalb hätten wir auch die Routine `lsqlin` zur Lösung verwenden können.
- Bei linearen Kleinste-Quadrate-Aufgaben ist die zweite Ableitung der Residuenfunktionen gleich null ($\nabla^2 r_i(x) = 0$), sodass das Levenberg-Marquardt-Verfahren bei deren Weglassen (vgl. (8.7)) keinen Fehler macht und deshalb in der Regel schneller konvergiert als bei nichtlinearen Kleinste-Quadrate-Aufgaben. Außerdem sind lineare Kleinste-Quadrate-Problem konvexe Optimierungsaufgaben, sodass das Levenberg-Marquardt-Verfahren nicht in einem lokalen Minimum stehenbleiben kann, das nicht bereits auch ein globales Minimum ist.
- (iv) In unseren Beispielen in diesem Kapitel sind die Modellfunktionen $g(x; \xi)$ relativ einfach gehalten und lassen sich durch eine Formel direkt angeben, wie etwa in (7.1).¹⁵ In vielen praktischen Aufgabenstellungen ist jedoch die Modellfunktion nur implizit durch die Lösung einer Differentialgleichung gegeben, und die unabhängige Variable ξ hat die Bedeutung der Zeit. $g(x; t)$ entspricht dann der Lösung der Differentialgleichung, die von den Modellparametern x abhängt, zum Zeitpunkt t .¹⁶
- (v) Viele Verfahren für Kleinste-Quadrate-Aufgaben können noch zusätzliche Beschränkungen an die Parameter x berücksichtigen, siehe [Tabelle 8.2](#). Hierin ist z. B. die vektorwertige Ungleichung $\ell \leq x$ komponentenweise zu verstehen, also:

$$\ell \leq x \quad \Leftrightarrow \quad \ell_i \leq x_i \quad \text{für alle } i = 1, \dots, n.$$

¹⁵Dies ist auch in der Übung der Fall, wo das sogenannte Hollmon-Modell betrachtet wird. In der Übung wird es auf das Johnson-Cooke-Modell mit mehr Parametern erweitert, wobei die Wahl einer guten Startschätzung wichtiger wird.

¹⁶In der Übung wird dies anhand einer Schwingungsgleichung geübt.

Typ der Beschränkung		lsqnonlin		lsqlin	
		TRR	LM	TRR	LM
Box-Beschränkungen	$\ell \leq x \leq u$	ja	nein	ja	ja
lineare Gleichungen	$A_{\text{eq}} x = b_{\text{eq}}$	nein	nein	nein	ja
lineare Ungleichungen	$A_{\text{ineq}} x \leq b_{\text{ineq}}$	nein	nein	nein	ja

Tabelle 8.2.: Beschränkungen, die von den Routinen zur Lösung von nichtlinearen (lsqnonlin) bzw. linearen (lsqlin) Kleinste-Quadrate-Aufgaben behandelt werden können. Für diese gibt es je zwei algorithmische Varianten, die mit *trust-region-reflective* (früher auch: *large-scale*) und *Levenberg-Marquardt* (früher auch: *medium-scale*) bezeichnet werden. Die Angaben zu lsqnonlin gelten auch für lsqcurvefit.

Ende der Woche 6

§ 9 BEREITSTELLEN VON ABLEITUNGEN

Wir haben an der Ausgabe von lsqcurvefit oben bereits gesehen, dass sich das Verfahren die benötigte Jacobimatrix der Residuenfunktion durch eine Differenzenapproximation beschaffen muss, wenn wir sie nicht zur Verfügung stellen. Die damit verbundenen Nachteile (Aufwand und möglicherweise unzureichende Genauigkeit) sind dieselben wie auf Seite 25 im Zusammenhang mit fminunc beschrieben. Wir wollen deshalb hier noch die Programmierung der Ableitung der Residuenfunktionen nachholen.

Unsere Modellfunktion (7.1) im Beispiel 7.1 ist sehr einfach, deshalb können wir die Ableitung von Hand berechnen:¹⁷

$$r(x) = \begin{pmatrix} g(x; a_1) - S_1 \\ \vdots \\ g(x; a_m) - S_m \end{pmatrix} = \begin{pmatrix} x_1 + x_2 a_1 - S_1 \\ \vdots \\ x_1 + x_2 a_m - S_m \end{pmatrix} \quad (9.1a)$$

$$J(x) = \begin{bmatrix} | & | \\ \frac{\partial}{\partial x_1} r(x) & \frac{\partial}{\partial x_2} r(x) \\ | & | \end{bmatrix} = \begin{bmatrix} 1 & a_1 \\ 1 & a_2 \\ \vdots & \vdots \\ 1 & a_m \end{bmatrix}. \quad (9.1b)$$

Wir erweitern nun unseren Code zur Berechnung der Modellfunktion (7.1) um die zusätzliche Berechnung der Ableitung. Die neue Datei nennen wir zur Unterscheidung Preisabsatzfunktion_Modellfunktion_mit_Ableitung.m:

¹⁷Wie man hier sieht, brauchen wir übrigens nicht zwischen der Ableitung der Residuenfunktionen $r_i(x)$ und der Ableitung der Modellfunktion $g(x; a_i)$ zu unterscheiden, denn die Größen S_i sind als Messgrößen ja vom Parameter x unabhängig und deshalb beim Ableiten nach x als Konstanten zu betrachten.

Matlab/
Preisabsatzfunk
tion_Modellfunktio
n_mit_Ableitung.m

```
function [S,J] = Preisabsatzfunktion_Modellfunktion(x,a)

% Bedeutung der Variablen (Modellparameter)
% x(1) = Offset
% x(2) = Steigung

% Berechnung der Modellfunktion und ihrer Ableitung (Jacobimatrix)
S = x(1) + x(2) * a;
J(:,1) = ones(length(S),1);
J(:,2) = a;
```

Im Hauptprogramm, das wir nun `Preisabsatzfunktion_mit_Ableitung.m` nennen, sind folgende Änderungen vorzunehmen:

```
% Setzen der Modellfunktion g(x1,x2;a) = x1 + x2 * a
% Bedeutung der Variablen (Modellparameter)
% x(1) = Offset
% x(2) = Steigung
g = @(x,a) Preisabsatzfunktion_Modellfunktion(x,a);
```

Wir überprüfen beispielhaft an der Stelle `x0` die Ableitung:

```
% % Ueberpruefen der Modellfunktion und ihrer Jacobimatrix
% [S,J] = g(x0,a)
```

Schließlich müssen wir `lsqcurvefit` noch mitteilen, dass die Ableitung (Jacobimatrix) der Residuenfunktion ab sofort verwendet werden soll. Dies geschieht mittels der (leider irreführend benannten) Option `SpecifyObjectiveGradient`¹⁸. Außerdem erlauben wir zur Sicherheit wieder, diese Ableitung mit Hilfe der bisher intern verwendeten Differenzenapproximation zu überprüfen: `CheckGradients`¹⁹.

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('lsqcurvefit','Algorithm','Levenberg-Marquardt',
    'Diagnostics','on','Display','iter-detailed',
    'SpecifyObjectiveGradient',true,'CheckGradients',true);
```

Der Verlauf der Iterationen und die Ausgabe ist der obigen (wo wir noch keine Ableitungen programmiert hatten) sehr ähnlich und wird deshalb hier nicht angegeben. Wichtigster Unterschied ist, dass nun unsere Modellfunktionsroutine `Preisabsatzfunktion_Modellfunktion_mit_Ableitung.m` nur noch vier Mal (einmal pro Iteration) aufgerufen wird.

Wir holen noch kurz die Motivation nach, warum die Residuen wie in (7.4) oft im quadratischen Sinne minimiert werden. Dafür gibt es eine statistische Begründung: Die Messwerte η_i , $i = 1, \dots, m$, an die wir unser Modell $g(x; \xi)$ anpassen wollen, sind in der Regel fehlerbehaftet. Geht man davon aus, dass diese Messfehler ε_i , $i = 1, \dots, m$ voneinander unabhängig und normalverteilt mit Mittelwert null und Varianz eins ($\varepsilon_i \sim \mathcal{N}(0, 1)$) sind, dann erhält

¹⁸Die frühere Bezeichnung `Jacobian` on war treffender.

¹⁹früher: `DerivativeCheck` on

man zumindest im Falle einer linearen Kleinste-Quadrate-Aufgabe (8.3) durch deren Lösung gerade den *wahrscheinlichsten Wert*²⁰ des Parameters x unter den gegebenen Messdaten.

Etwas allgemeiner können die Messfehler auch verschiedene Varianzen besitzen: $\varepsilon_i \sim \mathcal{N}(0, \sigma_i^2)$. Dann erhält man durch die Lösung der **gewichtete Kleinste-Quadrate-Aufgabe**

$$\text{Minimiere } f(x) = \frac{1}{2} r(x)^\top \Sigma^{-1} r(x) = \frac{1}{2} \sum_{i=1}^m \frac{r_i(x)^2}{\sigma_i^2}, \quad x \in \mathbb{R}^n \quad (9.2)$$

zumindest wieder im Falle (affin-)linear von x abhängender Residuen $r(x)$ den wahrscheinlichsten Parameterwert. Die **Gewichtsmatrix** Σ enthält dabei gerade die Varianzen der Messfehler auf der Hauptdiagonale:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_m^2 \end{bmatrix}.$$

In der MATLAB-Optimierungs-Toolbox sind gewichtete Kleinste-Quadrate-Aufgaben nicht explizit vorgesehen, aber man kann sie durch Definition der Residuen als

$$\tilde{r}_i(x) := \frac{r_i(x)}{\sigma_i} = \frac{g(x; \xi_i) - \eta_i}{\sigma_i}$$

immer auf gewöhnliche (ungewichtete) Kleinste-Quadrate-Aufgaben zurückführen und (am einfachsten) durch `lsqnonl` in lösen lassen.

Bei anderen Wahrscheinlichkeitsverteilungen (keine Normalverteilung) der Messfehler führt die Forderung, den wahrscheinlichsten Parameterwert zu finden, zu anderen Gewichtungen der Residuen. Beispielsweise erhält man im Falle von impulsartigen Fehlern (engl.: *salt and pepper noise*) gerade eine Aufgabe vom Typ (7.3) der kleinsten Fehlersumme (ohne Quadrate). Diese ist robust gegenüber Ausreißern (engl.: *outlier*) und erfordert andere Optimierungsverfahren, die hier nicht besprochen werden. Etwas weitergehende Informationen dazu findet man z. B. in der MATLAB-Curve-Fitting-Toolbox und in [Boyd, Vandenberghe, 2004](#), Kapitel 6.

§ 10 WEITERE KLEINSTE-QUADRATE-AUFGABEN

Kleinste-Quadrate-Aufgaben, also Optimierungsaufgaben der Form

$$\text{Minimiere } f(x) = \frac{1}{2} \|r(x)\|^2 = \frac{1}{2} r(x)^\top r(x), \quad x \in \mathbb{R}^n \quad (10.1)$$

treten nicht nur in der Parameteridentifikation auf. Um das zu illustrieren, behandeln wir hier zumindest ein weiteres Beispiel solcher Aufgaben, und zwar aus der Bildverarbeitung.

²¹

²⁰genauer: denjenigen mit der höchsten Wahrscheinlichkeitsdichte

²¹Dieses Beispiel stellen wir den Studenten aus mehreren Gründen nicht zur Verfügung: Erstens benötigt man die MATLAB-Imaging-Toolbox zum Erzeugen der Unschärfe. Zweitens wird die Unschärfematrix U durch Bestimmung der Bilder der Einheitsvektoren erzeugt, was nicht sehr elegant ist, dann gespeichert und beim nächsten Lauf wieder eingelesen.

Beispiel 10.1 (Bildrekonstruktion). Gegeben ist ein Vektor $y \in \mathbb{R}^n$, der die Daten eines rechteckigen Bildes der Größe $n_1 \times n_2$ repräsentiert (etwa spaltenweise ausgelesen).²² Diese Bilddaten sind jedoch durch gewisse Umstände verfälscht worden. Ein gutes Beispiel dafür ist die Aufnahme einer Originalszene $x \in \mathbb{R}^n$ mit einer Kamera, deren Objektiv nicht ganz scharfgestellt war, siehe Abb. 10.2. Dieses Unschärfmachen (engl.: blurring) der Bilddaten wird dabei durch einen linearen Zusammenhang

$$y = U x \quad (10.2)$$

beschrieben. Dabei repräsentiert die Matrix U eine sogenannte Faltung (engl.: convolution) des Vektors x der Originaldaten mit einem Gauß-Kern. Diese bewirkt, dass ein Bildpunkt von y nicht nur aus dem Wert desselben Eintrages in x gebildet wird, sondern auch die Werte der Nachbarpunkte eingehen. Es werden also Bildpunkte lokal gemittelt, was zu der Unschärfe führt. Eine Illustration des Effekts ist in Abb. 10.1 dargestellt.

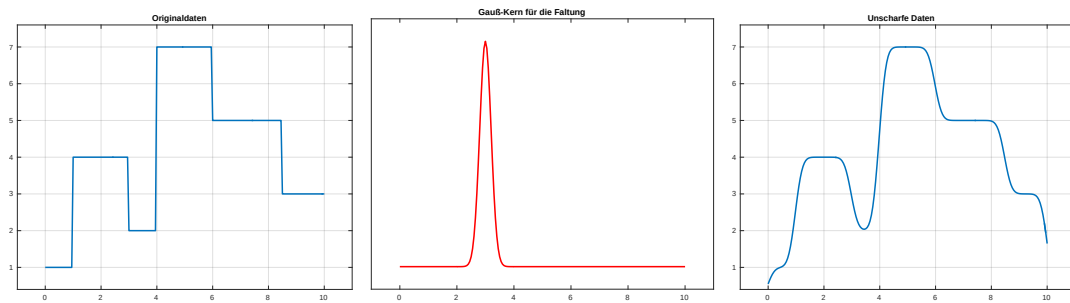


Abbildung 10.1.: Illustration des Entstehens von Unschärfe an eindimensionalen Daten. Die Originaldaten (links) werden mit einem Gauß-Kern (Mitte) gefaltet, d. h., diese Maske wird über die Originaldaten bewegt und dient als Gewichtung bei der Mittelwertbildung mit den Nachbardaten. Dadurch werden die Originaldaten unscharf (rechts).

Zusätzlich betrachten wir noch eine zufällige Störung ε der Bilddaten (Rauschen):

$$z = y + \varepsilon = U x + \varepsilon. \quad (10.3)$$

Es liegt nun der Wunsch nahe, aus dem unscharfen und verrauschten Bild z das i. A. unbekannte Originalbild x zu rekonstruieren. Die direkte Invertierung von (10.2) unter Vernachlässigung des unbekannten Rauschens, also die Berechnung von $x_{\text{invertiert}} = U^{-1}z$, ist nicht zu empfehlen, obwohl U hier eine invertierbare Matrix ist, denn: durch die große Konditionszahl von U wirkt sich das Rauschen (aber auch Rundungsfehler bei der Rechnung etc.) verheerend aus, siehe Abb. 10.3 Mitte. Man spricht hier von einem schlecht gestellten Problem.²³

Das wirkliche Originalbild wird man aus den Daten z niemals rekonstruieren können. Um jedoch eine brauchbare und plausible Rekonstruktion zu erhalten, formulieren wir eine passende Optimierungsaufgabe: Wir suchen ein Bild x , welches nach Anwendung

²²Der Einfachheit halber beschränken wir uns auf Graustufenbilder, aber eine Übertragung der Ideen auf Farbbilder ist kein Problem.

²³Als *schlecht gestellte Aufgaben* oder *inverse Probleme* bezeichnet man (anschaulich gesprochen) Situationen, bei denen es schwierig ist, von der Wirkung auf die Ursache rückzuschließen.



Abbildung 10.2.: Originalbild x (links) der Größe 256×256 Pixel, unscharfes Bild y (Mitte) und zusätzlich verrauschtes Bild z (rechts).

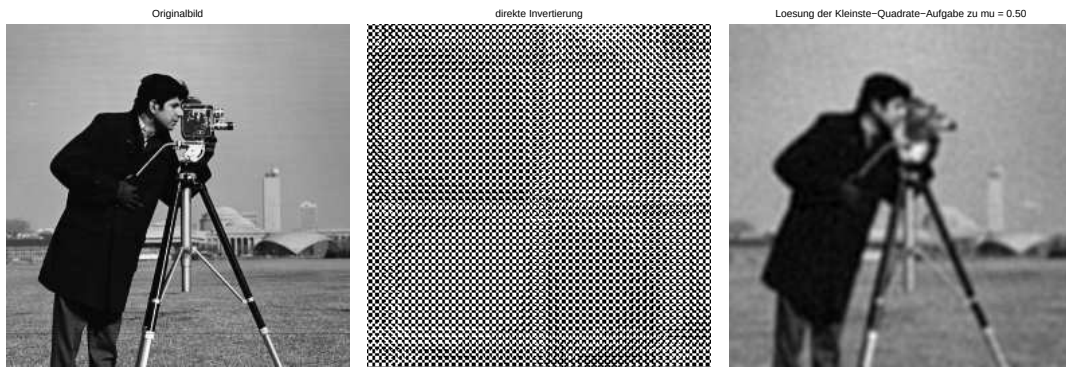


Abbildung 10.3.: Originalbild x (links) und durch Invertierung von (10.2), also $x_{\text{invertiert}} = U^{-1}z$ gewonnene unbrauchbare Rekonstruktion (Mitte). Rechts die Rekonstruktion durch Lösung von (10.4) zum Parameter $\mu = 0.5$.

der Unschärfeoperation U möglichst gut mit den gegebenen Daten z übereinstimmt. Diese Überlegung allein führt auf die Aufgabe

$$\text{Minimiere } \frac{1}{2} \|Ux - z\|^2, \quad x \in \mathbb{R}^n,$$

die wiederum genau $x_{\text{invertiert}} = U^{-1}z$ als eindeutige (und unbrauchbare) Lösung besitzt. Wir benötigen einen Term, der für eine *Stabilisierung* sorgt, insbesondere für eine Unempfindlichkeit gegenüber Rauschen. Die „Lösung“ $x_{\text{invertiert}}$ zeigt ein sprunghaftes Verhalten, also – anders als das Originalbild – wenig Zusammenhänge zwischen benachbarten Pixeln, vgl. Abb. 10.3 (Mitte). Hier setzen wir an und addieren noch einen weiteren Term zur Zielfunktion, der große Unterschiede (Differenzen) zwischen horizontal und vertikal benachbarten Pixeln „bestraft“. Sei dazu D_1 eine Matrix, sodass $D_1 x$ aus allen horizontalen Differenzen benachbarter Pixel im Bild besteht, und analog D_2 für die vertikalen Differenzen.

Damit können wir eine regularisierte Rekonstruktionsaufgabe wie folgt formulieren:

$$\text{Minimiere } \frac{1}{2} \|Ux - z\|^2 + \frac{\mu}{2} \|D_1 x\|^2 + \frac{\mu}{2} \|D_2 x\|^2, \quad x \in \mathbb{R}^n. \quad (10.4)$$

Dabei ist $\mu > 0$ ein Parameter, der die Gewichtung der verschiedenen Anteile in der Ziel-

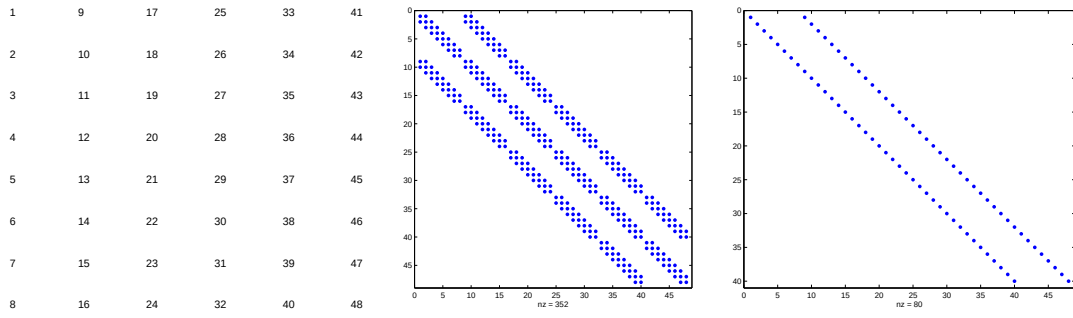


Abbildung 10.4.: Reihenfolge der Bilddaten in den Vektoren x , y , z usw. (links), Besetzungsstruktur der Faltungsmatrix U (bei einer Maskengröße von 3×3 Pixeln) (Mitte) sowie Besetzungsstruktur des horizontalen Differenzenoperators D_1 an einem Bild der Größe 8×6 (Höhe mal Breite).

funktion untereinander bestimmt.²⁴ Ein kurzer Blick auf (10.4) zeigt, dass dies wieder eine (lineare) Kleinste-Quadrate-Aufgabe (10.1) ist, wobei der Residuenvektor als

$$r(x) = \begin{pmatrix} Ux - z \\ \sqrt{\mu} D_1 x \\ \sqrt{\mu} D_2 x \end{pmatrix} = \begin{bmatrix} U \\ \sqrt{\mu} D_1 \\ \sqrt{\mu} D_2 \end{bmatrix} x - \begin{pmatrix} z \\ 0 \\ 0 \end{pmatrix} \quad (10.5)$$

definiert wird. Die Lösung der Aufgabe (10.4), die mit Hilfe der *large-scale*-Variante von `lsq` in berechnet wurde, wird in Abb. 10.3 (rechts) gezeigt, wobei als Parameter $\mu = 0.5$ gewählt wurde. Den Einfluss des Regularisierungsparameters μ zeigt die Studie in Abb. 10.5.

Es gibt natürlich wesentlich ausgefeiltere Techniken der Bildrekonstruktion. Beispielsweise wird durch unseren Ansatz (10.4) das Rauschen gemindert, aber es werden auch Kanten verwischt. Dies kann man durch eine geschicktere Formulierung der Zielfunktion vermeiden, die dann aber nicht mehr auf eine Kleinste-Quadrate-Aufgabe führt und deshalb hier nicht weiter behandelt wird. Für weiterführende Informationen zur mathematischen Bildverarbeitung verweisen wir z. B. auf Bredies, Lorenz, 2011.

²⁴Man bezeichnet die beiden letzten Terme in der Zielfunktion (10.4) als *Regularisierungs-* oder *Stabilisierungsterme*. In ihnen haben wir unser Vorwissen bzw. unsere Erwartung ausgedrückt, dass sich benachbarte Pixel nicht zu sehr in ihrem Wert (ihrer Helligkeit) unterscheiden werden. Man spricht auch von *A-Priori-Information*. Mehr über solche Regularisierungsverfahren für *schlecht gestellte Aufgaben* lernt man in Vorlesungen über *Inverse Probleme*.



Abbildung 10.5.: Unscharfes und verrauschtes Bild (oben links) und fünf verschiedene Rekonstruktionen durch Lösung von (10.4) zu den Parametern $\mu \in \{0.050, 0.158, 0.500, 1.581, 5.000\}$.

Ende der Woche 7

Kapitel 3 Netzwerk- und Transportprobleme (Lineare (ganzzahlige) Optimierung)

In diesem Kapitel geht es um lineare Optimierungsprobleme und insbesondere um Netzwerk- und Transportprobleme. Weitere Anwendungen finden Sie z. B. unter <http://de.wikipedia.org/wiki/Lineare.Optimierung>. Ein einfaches Beispiel für diese Aufgabenklasse hatten wir schon in [Beispiel 1.1](#) gesehen.

§ 11 EIN ERSTES BEISPIEL

Eine **lineare Optimierungsaufgabe**, auch **lineares Programm (LP)** genannt, hat folgende allgemeine Gestalt:

$$\begin{aligned} &\text{Minimiere} && c^T x, && x \in \mathbb{R}^n \\ &\text{unter} && A_{\text{ineq}} x \leq b_{\text{ineq}} \\ &\text{und} && A_{\text{eq}} x = b_{\text{eq}} \\ &\text{sowie} && \ell \leq x \leq u. \end{aligned} \tag{11.1}$$

Wir haben es also hier erstmals mit einem Optimierungsprobleme mit Beschränkungen zu tun. Jedoch sind die Zielfunktion und die Gleichungs- und Ungleichungsbeschränkungen alles (affin-)lineare Funktionen von x .¹ Der Vektor c in der Zielfunktion heißt **Kostenvektor**. Die Anzahl der Ungleichungen m_{ineq} bzw. Gleichungen m_{eq} lässt sich an der Länge des Vektors b_{ineq} bzw. b_{eq} oder auch an der Höhe der Matrizen A_{ineq} bzw. A_{eq} ablesen. In einer konkreten Aufgabe müssen auch nicht immer Ungleichungs-, Gleichungs- und Box-Beschränkungen auftreten.

Unser erstes Beispiel in diesem Kapitel hat noch nichts mit Netzwerken oder Transport zu tun. Solche Aufgaben folgen in [§ 14](#).

Beispiel 11.1 (Mozartproblem). *Eine Firma stellt Mozartkugeln und Mozarttaler her und benötigt dafür folgende Zutaten:*

	Marzipan	Nougat	Bitterschokolade	Verkaufspreis
Mozartkugeln	1	2	1	9
Mozarttaler	1	1	2	8
Lagerbestand	6	11	9	

¹Aus diesem Grund müssen wir uns in diesem Kapitel nicht um Ableitungen kümmern.

Gesucht ist die zu produzierende Menge an Mozartkugeln und -talern, sodass der Umsatz maximal wird.

Wir formulieren dies als Optimierungsaufgabe. Die unbekannten und zu optimierenden Größen sind

x_1 = zu produzierende Menge an Mozartkugeln,

x_2 = zu produzierende Menge an Mozarttalern,

die wir zu $x = (x_1, x_2) \in \mathbb{R}^2$ zusammenfassen. Wir stellen zunächst die Zielfunktion auf:

$$\text{Maximiere } 9x_1 + 8x_2, \quad x \in \mathbb{R}^2. \quad (11.2a)$$

Anschließend formulieren wir die Beschränkungen, die durch die Lagerbestände gegeben sind (Ressourcenbeschränkungen):

$$1x_1 + 1x_2 \leq 6 \quad \text{Marzipanbeschränkung,}$$

$$2x_1 + 1x_2 \leq 11 \quad \text{Nougatbeschränkung,} \quad (11.2b)$$

$$1x_1 + 2x_2 \leq 9 \quad \text{Bitterschokoladenbeschränkung.}$$

Schließlich ist es naheliegend, nur nicht-negative Produktionsmengen zuzulassen:

$$x_1 \geq 0, \quad x_2 \geq 0. \quad (11.2c)$$

Da unsere Aufgabe (11.2) nur zwei Optimierungsvariable (x_1, x_2) hat, können wir sie noch grafisch lösen. Zunächst stellen wir die **zulässige Menge** dar, also diejenigen Punkte $x = (x_1, x_2)$, die alle Ungleichungsbeschränkungen (11.2b) sowie die Vorzeichenbedingungen (11.2c) erfüllen, siehe Abb. 11.1 (links).

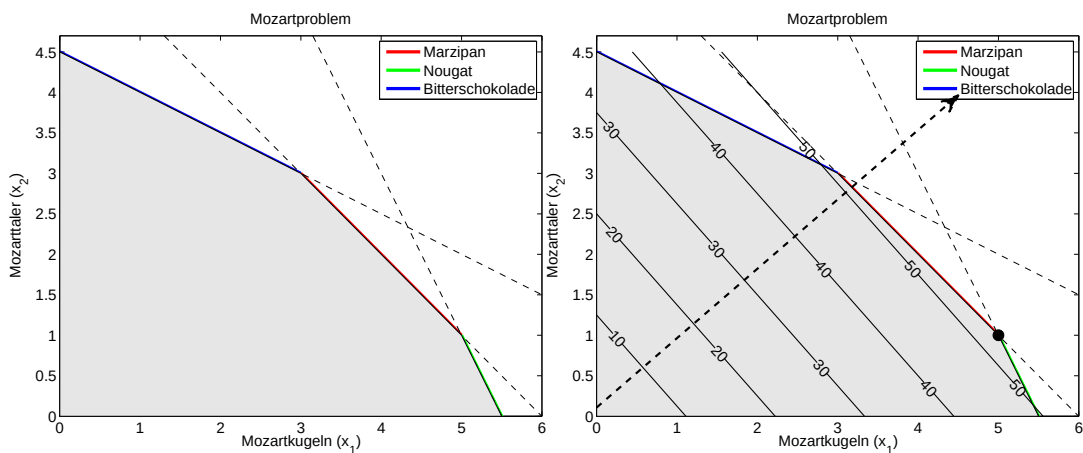


Abbildung 11.1.: Zulässige Menge (links) beim Mozartproblem (Beispiel 11.1) und grafische Lösung durch Einzeichnen des Kostenvektors c und einiger Höhenlinien der Zielfunktion (rechts).

Um zu bestimmen, wo in der zulässigen Menge die Lösung x^* liegt, müssen wir wissen, wo die Punkte mit kleinen bzw. großen Zielfunktionswerten liegen. Dazu zeichnen wir einige Höhenlinien (Linien gleicher Werte) der Zielfunktion ein, etwa

$$\{x \in \mathbb{R}^2 : c^T x = 10\}, \dots, \{x \in \mathbb{R}^2 : c^T x = 50\}.$$

Da unsere Zielfunktion (11.2a) ja linear ist, ist jede solche Höhenlinie eine Gerade:

$$\{x \in \mathbb{R}^2 : 9x_1 + 8x_2 = 10\}, \dots, \{x \in \mathbb{R}^2 : 9x_1 + 8x_2 = 50\}$$

usw. Zu deren Darstellung zeichnen wir zunächst den Kostenvektor unserer Aufgabe, also den Vektor

$$c = \begin{pmatrix} 9 \\ 8 \end{pmatrix}$$

ein. Die Höhenlinien stehen dann jeweils senkrecht auf c . Die Funktionswerte der Zielfunktion steigen in Richtung c und fallen in der Gegenrichtung $-c$. Da wir maximieren wollen, suchen wir von den zueinander parallelen Höhenlinie diejenige heraus, die am weitesten in Richtung des Kostenvektors c verschoben wurde und die die zulässige Menge gerade noch schneidet. Der Schnittpunkt dieser extremalen Höhenlinie mit der zulässige Menge ist die gesuchte Lösung, siehe Abb. 11.1. Im Beispiel ist dies der Punkt

$$x^* = \begin{pmatrix} 5 \\ 1 \end{pmatrix},$$

den wir aus der Zeichnung ablesen oder als Schnittpunkt der Geraden $2x_1 + x_2 = 11$ und $x_1 + x_2 = 6$ bestimmen können.² Aus der Zeichnung erkennt man weiterhin, dass in der Lösung x^* die Vorräte an Nougat und Marzipan aufgebraucht sein werden, während noch ein Lagerbestand an Bitterschokolade übrig bleibt. Der maximal mögliche (optimale) Umsatz schließlich beläuft sich auf

$$c^T x^* = 9 \cdot 5 + 8 \cdot 1 = 53.$$

Offenbar ist es typisch für lineare Optimierungsaufgaben, dass die zulässige Menge ein eckiges Gebilde darstellt (**Polyeder**) und dass sich eine Optimallösung in einer Ecke befindet. Dies ist kein Zufall und wird auch von numerischen Optimierungsverfahren ausgenutzt, wie wir unten noch genauer sehen werden.

Wir schreiben jetzt unsere Aufgabe (11.2) in einer Form, in der wir sie auch linprog, dem MATLAB-Löser für lineare Optimierungsaufgaben, übergeben können. Dazu müssen wir, wie bereits in § 1 beschrieben, die Maximierungs- in eine Minimierungsaufgabe umwandeln³ und die Beschränkungen in Matrix-Vektor-Form schreiben:

$$\left. \begin{array}{ll} \text{Minimiere} & \underbrace{\begin{pmatrix} -9 \\ -8 \end{pmatrix}^T}_{c} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad x = (x_1, x_2)^T \in \mathbb{R}^2 \\ \text{unter} & \underbrace{\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix}}_{A_{\text{ineq}}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \underbrace{\begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix}}_{b_{\text{ineq}}} \quad \begin{array}{l} \text{Marzipan} \\ \text{Nougat} \\ \text{Bitterschokolade} \end{array} \\ \text{sowie} & \underbrace{0}_{\ell} \leq x. \end{array} \right\} \quad (11.3)$$

²Hierfür muss also ein lineares Gleichungssystem der Größe 2×2 gelöst werden.

³Wir bezeichnen ab jetzt den Kostenvektor der *Minimierungsaufgabe* (11.3) als c . Bisher war c das Negative davon.

Beschränkungen der Art $x \leq u$ oder Gleichungsbeschränkungen $A_{\text{eq}} x = b_{\text{eq}}$ liegen bei dieser Aufgabe nicht vor. Das Symbol „0“ in der Ungleichung $0 \leq x$ kennzeichnet den Nullvektor.

Wir wollen nun unsere Aufgabe (11.3) mittels `linprog` lösen:

```
% Erstellen der Problembeschreibung
% Kostenvektor
c = [-9; -8];
% Ungleichungsbeschränkungen
Aineq = [1 1; 2 1; 1 2];
bineq = [6; 11; 9];
% Gleichungsbeschränkungen
Aeq = [];
beq = [];
% untere und obere Schranken
l = zeros(size(c));
u = [];
```

Aus den drei in `linprog` verfügbaren Algorithmen wählen wir durch Setzen der entsprechenden Optionen das (duale) **Simplex-Verfahren** aus:⁴

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('linprog','Algorithm','dual-simplex','
    Diagnostics','on','Display','iter');
```

Schließlich rufen wir `linprog` auf:

```
% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda] = linprog(c,Aineq,bineq,Aeq,beq,l
    ,u,options);
```

Die leere Klammer `[]` zeigt an, dass wir keine Startschätzung setzen, da diese vom Simplex-Verfahren in `linprog` ohnehin ignoriert wird. Die Ausgabe von `linprog` sieht etwa wie folgt aus:

```
-----
Diagnostic Information

Number of variables: 2

Number of linear inequality constraints:    3
Number of linear equality constraints:      0
Number of lower bound constraints:         2
Number of upper bound constraints:         0

Algorithm selected
    dual-simplex
```

⁴Das früher ebenfalls verfügbare primale Simplex-Verfahren ist in der MATLAB-Optimierungs-Toolbox mit R2016b weggefallen.

```

-----
End diagnostic information

LP preprocessing removed 0 inequalities, 0 equalities,
0 variables, and added 0 non-zero elements.

Iter      Time      Fval  Primal Infeas  Dual Infeas
  0         0    0.000000e+00  0.000000e+00  6.758116e+00
  1         0   -5.400000e+01  8.908987e-01  0.000000e+00
  2         0   -5.300000e+01  0.000000e+00  0.000000e+00

Optimal solution found.

```

Wir werden diese Ausgabe später wieder genauer kommentieren, nachdem wir die Grundlagen des Simplex-Verfahrens kennengelernt haben. Zunächst wollen wir einmal überprüfen, ob `linprog` richtig gerechnet hat: Das Ergebnis

$$x^* = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

und der Zielfunktionswert $fstar$ von -53 stimmen mit unserer grafischen Lösung überein.⁵

Bei Aufgaben mit mehr als zwei Optimierungsvariablen können wir die Lösung nicht mehr auf grafischem Wege bestimmen. Wie können wir dann sicher sein, dass `linprog` wirklich eine Lösung gefunden hat? Dazu benötigen wir die passenden Optimalitätsbedingungen für lineare Optimierungsaufgaben der Form (11.1). Wir erinnern daran, dass die Zahl $m_{ineq} \geq 0$ die Anzahl der Ungleichungsbeschränkungen in (11.1) kennzeichnet und $m_{eq} \geq 0$ die Anzahl der Gleichungsbeschränkungen. Die Beschränkungen sind auch der Grund dafür, dass die Optimalitätsbedingungen wesentlich komplizierter erscheinen als die Bedingungen für freie (unrestringierte) Optimierungsaufgaben.

Zunächst müssen wir klären, was überhaupt ein Minimum von (11.1) ist.⁶

Definition 11.2 (Minimum in der linearen Optimierung). *Wir betrachten eine lineare Optimierungsaufgabe (11.1). Die Menge*

$$X := \{x \in \mathbb{R}^n : A_{ineq} x \leq b_{ineq}, A_{eq} x = b_{eq}, \ell \leq x \leq u\} \quad (11.4)$$

heißt **zulässige Menge** von (11.1).

(i) Ein Vektor $x^* \in \mathbb{R}^n$ heißt ein (**globales**) **Minimum** von (11.1), wenn gilt:

$$c^T x^* \leq c^T x \quad \text{für alle } x \in X.$$

⁵Zu beachten ist natürlich, dass der Zielfunktionswert der ursprünglichen Maximierungsaufgabe (11.2) dann $+53$ beträgt, da wir ja in Vorbereitung auf die Nutzung von `linprog` zur Minimierung der negativen Zielfunktion übergegangen sind.

⁶Die bisherige Definition 2.2 bezieht sich nur auf freie Optimierungsaufgaben.

(ii) Ein globales Minimum x^* heißt **strikt**, wenn gilt:⁷

$$c^T x^* < c^T x \quad \text{für alle } x \in X, \quad x \neq x^*.$$

Ein Minimum nennt man auch wieder **Optimum, Minimalstelle** oder **(Optimal)lösung** von (11.1).

Wir geben nun die Optimalitätsbedingungen für (11.1) an.

Satz 11.3 (Notwendige und hinreichende Optimalitätsbedingung 1. Ordnung).

(i) Es sei $x^* \in \mathbb{R}^n$ ein (globales) Minimum von (11.1). Dann kann man dazu passende Vektoren (genannt **Lagrange-Multiplikatoren**)

$$\lambda_{ineq} \in \mathbb{R}^{m_{ineq}}, \quad \lambda_{eq} \in \mathbb{R}^{m_{eq}}, \quad \lambda_\ell \in \mathbb{R}^n \quad \text{und} \quad \lambda_u \in \mathbb{R}^n$$

finden, sodass das folgende System aus Gleichungen und Ungleichungen (die sogenannten **Karush-Kuhn-Tucker-Bedingungen**, kurz: **KKT-Bedingungen**) erfüllt ist:

$$\left. \begin{aligned} A_{ineq}^T \lambda_{ineq} + A_{eq}^T \lambda_{eq} - \lambda_\ell + \lambda_u + c &= 0, \\ \lambda_{ineq} &\geq 0, \quad A_{ineq} x^* - b_{ineq} \leq 0, \quad \lambda_{ineq}^T (A_{ineq} x^* - b_{ineq}) = 0, \\ \lambda_\ell &\geq 0, \quad \ell - x^* \leq 0, \quad \lambda_\ell^T (\ell - x^*) = 0, \\ \lambda_u &\geq 0, \quad x^* - u \leq 0, \quad \lambda_u^T (x^* - u) = 0, \\ A_{eq} x^* - b_{eq} &= 0. \end{aligned} \right\} \quad (11.5)$$

(ii) Sind umgekehrt $x^* \in \mathbb{R}^n$ und Vektoren

$$\lambda_{ineq} \in \mathbb{R}^{m_{ineq}}, \quad \lambda_{eq} \in \mathbb{R}^{m_{eq}}, \quad \lambda_\ell \in \mathbb{R}^n \quad \text{und} \quad \lambda_u \in \mathbb{R}^n$$

bekannt, die das System (11.5) erfüllen, dann ist x^* ein (globales) Minimum von (11.1).

Auf die Herleitung des Optimalitätssystems (11.5) und die Bedeutung der Lagrange-Multiplikatoren kommen wir in § 13 zu sprechen. Wir weisen hier zunächst nur darauf hin, dass es für jeden Typ von Beschränkung in (11.1) einen zugehörigen Multiplikator-Vektor der passenden Dimension gibt. Insgesamt hat man genau so viele Multiplikatoren, wie es Beschränkungen gibt. Diejenigen Multiplikatoren, die zu Ungleichungsbeschränkungen gehören, müssen nicht-negativ sein. Für die Multiplikatoren, die zu Gleichungsbeschränkungen gehören, gilt dagegen keine Vorzeichenbedingung.⁸

Bemerkung 11.4 (Eigenschaften linearer Optimierungsaufgaben). Es sei hier bemerkt, dass lineare Optimierungsaufgaben (11.1) konvexe Optimierungsaufgaben sind. Daraus folgen zwei wichtige Konsequenzen:

⁷Das globale Minimum ist in diesem Fall natürlich eindeutig.

⁸Das sieht man schon daran, dass man Gleichungsnebenbedingungen mit (-1) multiplizieren kann, ohne die Lösung x^* zu verändern. Die zugehörigen Multiplikatoren werden dann ebenfalls mit (-1) multipliziert, wechseln also das Vorzeichen.

- (i) Es gibt keine Unterscheidung zwischen lokalen und globalen Minima.
(Jedes lokale Minimum ist bereits ein globales.)
- (ii) Das System (11.5) ist notwendig und hinreichend für die Optimalität von x^* . Das bedeutet z. B., wenn man alle Lösungen von (11.5) berechnet hat, so hat man auch alle Lösungen von (11.1) gefunden. Man kann dabei weder Lösungen übersehen noch Punkte erhalten, die keine Lösungen sind.

Wir behandeln diese Thematik nochmal ausführlicher bei nichtlinearen Aufgaben im [Satz 17.10](#).

Wir wollen nun überprüfen, ob die mit `linprog` erhaltene Lösung tatsächlich das Optimalitätssystem (11.5) erfüllt. Da wir `linprog` mit einer erweiterten Liste von Rückgabeparametern aufgerufen haben,

Matlab/
Mozartproblem.m

```
% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda] = linprog(c,Aineq,bineq,Aeq,beq,l
,u,options);
```

haben wir in der Variablen `lambda` (einer MATLAB-*struct*) nun die Lagrange-Multiplikatoren zur Verfügung:

$$\begin{aligned} \text{lambda.ineqlin} &\hat{=} \lambda_{\text{ineq}}, & \text{lambda.lower} &\hat{=} \lambda_\ell, \\ \text{lambda.eqlin} &\hat{=} \lambda_{\text{eq}}, & \text{lambda.upper} &\hat{=} \lambda_u. \end{aligned}$$

In unserem [Beispiel 11.1](#) gab es keine Gleichungsbeschränkung $A_{\text{eq}} x = b_{\text{eq}}$ und auch keine obere Schranke $x \leq u$. Die entsprechenden Terme und Bedingungen sind im Optimalitätssystem (11.5) deshalb zu ignorieren.

Mit diesen Informationen können wir nun das System (11.5) schrittweise überprüfen. Zunächst die erste Zeile (ohne λ_{ineq} und λ_u): $A_{\text{ineq}}^T \lambda_{\text{ineq}} - \lambda_\ell + c = 0$

Matlab/Mozartproblem.KKT_pruefen.m

```
Aineq' * lambda.ineqlin - lambda.lower + c
```

ergibt den Vektor

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

die erste Zeile in (11.5) ist also erfüllt.

Für die zweite Zeile $\lambda_{\text{ineq}} \geq 0$, $A_{\text{ineq}} x^* - b_{\text{ineq}} \leq 0$ und $\lambda_{\text{ineq}}^T (A_{\text{ineq}} x^* - b_{\text{ineq}}) = 0$ lassen wir ausgeben:

Matlab/Mozartproblem.KKT_pruefen.m

```
[lambda.ineqlin, Aineq * xstar - bineq]
```

und erhalten als Antwort

$$\begin{pmatrix} 7 & 0 \\ 1 & 0 \\ 0 & -2 \end{pmatrix}.$$

Zunächst fällt auf, dass die erste Spalte, also λ_{ineq} , komponentenweise ≥ 0 ist. Gleichfalls ist die zweite Spalte $A_{\text{ineq}} x - b_{\text{ineq}}$, komponentenweise ≤ 0 . Schließlich ist das Skalarprodukt dieser beiden Vektoren gleich null:

$$7 \cdot 0 + 1 \cdot 0 + 0 \cdot (-2) = 0,$$

denn in jeder Zeile steht mindestens eine Null. Die zweite Zeile von (11.5) ist also ebenfalls erfüllt. Eine solche Situation zwischen zwei Vektoren $y, z \in \mathbb{R}^m$

$$y \geq 0, \quad z \leq 0, \quad y^T z = 0$$

nennt man ein **Komplementaritätssystem**.⁹ Der Name leitet sich aus folgender Beobachtung ab: Die Bedingung

$$y^T z = \sum_{i=1}^m \underbrace{y_i}_{\geq 0} \underbrace{z_i}_{\leq 0} = 0$$

kann nur erfüllt sein, wenn alle Summanden $y_i z_i = 0$ sind. Dies wiederum bedeutet, dass in jeder Komponente $i = 1, \dots, m$ entweder $y_i = 0$ oder $z_i = 0$ oder beide gleich null gelten muss. Wenn z. B. $y_i > 0$ ist, dann muss zwangsläufig $z_i = 0$ sein. In diesem Sinne sind die Vektoren y und z komponentenweise komplementär.

Überprüfen wir nun die dritte Zeile in (11.5), ein weiteres Komplementaritätssystem:

```
[lambda.lower, l - xstar]
```

ergibt

$$\begin{pmatrix} 0 & -5 \\ 0 & -1 \end{pmatrix}.$$

Also ist auch dieses Komplementaritätssystem erfüllt.

Die vierte Zeile schließlich ist hier nicht relevant und kann ignoriert werden, da wir keine obere Schranke für x gesetzt hatten. Für die fünfte Zeile (die Zulässigkeit von x^* bzgl. der Gleichungsnebenbedingungen, $A_{\text{eq}} x^* - b_{\text{eq}} = 0$) ist hier ebenfalls nichts zu prüfen, da unsere Aufgabe (11.3) ja keine Gleichungsnebenbedingungen besitzt. Also können wir schließen, dass linprog tatsächlich eine optimale Lösung gefunden hat. Diese ist hier im Beispiel sogar eindeutig.

Lineare Optimierungsaufgaben wie (11.1) sind vielseitig einsetzbar und treten in vielen Aufgabenstellungen auf. Einer solchen Aufgabenklasse, nämlich Transportproblemen auf Netzwerken und verwandten Aufgaben, widmen wir gleich mehrere Abschnitte (§ 14–§ 16). Auch Optimierungsaufgaben, die zunächst nicht so aussehen, lassen sich manchmal in lineare Programme überführen. Wir besprechen dazu zwei Beispiele.

Beispiel 11.5 (Umschreiben einer Optimierungsaufgabe mit $\|\cdot\|_\infty$ in der Zielfunktion). Die **Unendlich-Norm** eines Vektors $y \in \mathbb{R}^p$ ist erklärt durch

$$\|y\|_\infty := \max_{i=1, \dots, p} |y_i|.$$

⁹Die Vorzeichen von y und z sind dabei nicht entscheidend. Auch bei $y \geq 0, z \geq 0$ und $y^T z = 0$ spricht man von einem Komplementaritätssystem.

Als Optimierungsaufgabe können wir die Berechnung dieser Norm wie folgt formulieren:

$$\begin{aligned} &\text{Minimiere } t, \quad t \in \mathbb{R} \\ &\text{unter } -t e \leq y \leq t e. \end{aligned} \quad (11.6)$$

Dabei ist $e = (1, \dots, 1)^T \in \mathbb{R}^p$ ein Vektor aus lauter Einsen. Eine Illustration findet sich in Abb. 11.2.

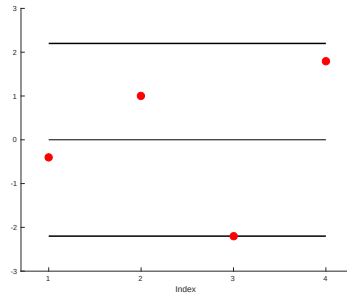


Abbildung 11.2.: Illustration der Bestimmung der Norm $\|y\|_\infty$ am Beispiel des Vektors $y = (-0.4, 1.0, -2.2, 1.8)^T$ gemäß (11.6).

Eine Optimierungsaufgabe mit der Norm $\|x\|_\infty$ der Optimierungsvariablen in der Zielfunktion und ansonsten linearen Nebenbedingungen wie in (11.1) kann deshalb als lineare Optimierungsaufgabe umformuliert werden. Das gilt auch für die etwas allgemeinere Zielfunktion $\|Cx - d\|_\infty$ mit einer Matrix $C \in \mathbb{R}^{p \times n}$ und $d \in \mathbb{R}^p$. Beispielsweise ist

$$\begin{aligned} &\text{Minimiere } \|Cx - d\|_\infty, \quad x \in \mathbb{R}^n \\ &\text{unter } Ax \leq b \end{aligned} \quad (11.7)$$

äquivalent zur linearen Optimierungsaufgabe

$$\begin{aligned} &\text{Minimiere } t, \quad (x, t) \in \mathbb{R}^n \times \mathbb{R} \\ &\text{unter } -t e \leq Cx - d \leq t e \\ &\text{und } Ax \leq b. \end{aligned} \quad (11.8)$$

Das Umschreiben erkauft man sich durch die Einführung einer zusätzlichen Optimierungsvariablen sowie weiteren Nebenbedingungen. Zur Lösung in linprog müssen natürlich (x, t) zu einer Optimierungsvariablen in $\mathbb{R}^{n+1}$ zusammengefasst werden.

Beispiel 11.6 (Umschreiben einer Optimierungsaufgabe mit $\|\cdot\|_1$ in der Zielfunktion). Die **1-Norm** eines Vektors $y \in \mathbb{R}^p$ ist erklärt durch

$$\|y\|_1 := \sum_{i=1}^p |y_i| = \sum_{i=1}^p y_i^+ + \sum_{i=1}^p y_i^-.$$

Dabei sind y^+ und y^- die (komponentenweise gebildeten) positiven bzw. negativen Teile von y :

$$y^+ := \max\{0, y\}, \quad y^- := -\min\{0, y\}.$$

Es ist leicht einzusehen, dass y_i^+ und y_i^- die Lösungen der folgenden Optimierungsaufgaben sind:

$$\begin{array}{ll} \text{Minimiere} & v_i, \quad v_i \in \mathbb{R} \\ \text{unter} & v_i \geq 0 \\ \text{und} & v_i \geq y_i. \end{array} \quad \text{bzw.} \quad \begin{array}{ll} \text{Minimiere} & u_i, \quad u_i \in \mathbb{R} \\ \text{unter} & u_i \geq 0 \\ \text{und} & u_i \geq -y_i. \end{array}$$

Damit erhalten wir, dass beispielsweise die Aufgabe

$$\begin{array}{ll} \text{Minimiere} & \|Cx - d\|_1, \quad x \in \mathbb{R}^n \\ \text{unter} & Ax \leq b \end{array} \quad (11.9)$$

mit einer Matrix $C \in \mathbb{R}^{p \times n}$ und $d \in \mathbb{R}^p$ äquivalent ist zur linearen Optimierungsaufgabe

$$\begin{array}{ll} \text{Minimiere} & e^T u + e^T v, \quad (x, u, v) \in \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^p \\ \text{unter} & -u \leq Cx - d \leq v \\ \text{und} & u \geq 0, \quad v \geq 0 \\ \text{sowie} & Ax \leq b. \end{array} \quad (11.10)$$

Auch hier benötigt man also wieder zusätzliche Optimierungsvariablen und Nebenbedingungen.

Ende der Woche 8

§ 12 DAS VERFAHREN HINTER linprog

Wir wollen nun das Verfahren hinter `linprog` kennenlernen, genauer: das **Simplex-Verfahren**. Dieses gibt es in einer sogenannten primalen und einer dualen Variante. Im Folgenden werden wir auf das **primale Simplex-Verfahren** im Detail eingehen, das einfacher zu erklären ist. Hingegen steht in der MATLAB-Optimierungs-Toolbox nur das duale Simplex-Verfahren zur Verfügung, das wir oben durch Setzen der Optionen

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('linprog','Algorithm','dual-simplex','
    Diagnostics','on','Display','iter');
```

ausgewählt hatten. Auf die Unterschiede kommen wir in § 13 zu sprechen.

Wir erklären das primale Simplex-Verfahren am Beispiel einer linearen Optimierungsaufgabe in **Normalform**:

$$\begin{array}{ll} \text{Minimiere} & c^T x, \quad x \in \mathbb{R}^n \\ \text{unter} & Ax = b \\ \text{sowie} & x \geq 0. \end{array} \quad (12.1)$$

Dies dient hier nur der einfacheren Illustration. Im Prinzip kann zwar jede lineare Optimierungsaufgabe der allgemeinen Form (11.1) durch bestimmte Manipulationen in Normalform

gebracht werden, dies geht jedoch einher mit der Einführung zusätzlicher Optimierungsvariablen. Die MATLAB-Implementierung von `linprog` kann direkt die allgemeine Aufgabenstellung (11.1) ohne Umwandlung in Normalform lösen bzw. führt diese Umwandlung intern (in etwas abgewandelter Form) durch.

Wir gehen weiter hier davon aus, dass wir in (12.1) $m \leq n$ Gleichungsnebenbedingungen haben, die widerspruchsfrei sind und keine Redundanzen aufweisen. Dies wird dadurch sichergestellt, dass die Matrix $A \in \mathbb{R}^{m \times n}$ vollen Rang besitzt, die m Zeilen von A also linear unabhängig sind.

Wir hatten schon beobachtet, dass die Optimallösungen einer linearen Optimierungsaufgabe in den Ecken der zulässigen Menge zu suchen sind. Dies gilt auch für Aufgaben in Normalform (12.1).

Die Hauptidee des (primalen) Simplex-Algorithmus ist einfach: Laufe von einer Ecke zu einer benachbarten Ecke, bis es keine bessere (d. h. mit kleinerem Zielfunktionswert) mehr gibt. Dazu muss das Verfahren in der Lage sein, Ecken zu berechnen. Da Ecken extreme Punkte der zulässigen Menge sind, ist es nicht verwunderlich, dass in einer Ecke einige Koordinaten $x_i = 0$ sind, während die restlichen $x_i \geq 0$ sind, siehe auch Abb. 12.1.

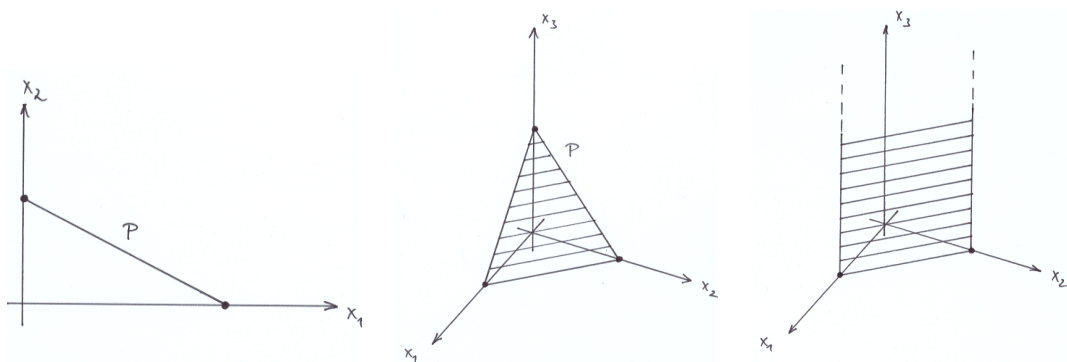


Abbildung 12.1.: Polyeder in Normalform und ihre Ecken: Strecke im $\mathbb{R}^2$ ($n = 2$ Variablen, $m = 1$ Gleichung) und Flächen im $\mathbb{R}^3$ ($n = 3$ Variablen, $m = 1$ Gleichung).

Genauer erhält man eine Ecke $x \in \mathbb{R}^n$ wie folgt:

- (i) Man wähle eine Indexmenge $B \subset \{1, \dots, n\}$ mit m Elementen, kurz: $|B| = m$.
- (ii) Die Menge B muss so gewählt sein, dass die zugehörigen Spalten der Matrix A linear unabhängig sind. Diese Untermatrix von A bezeichnen wir mit A_B . Sie muss also eine quadratische, invertierbare Matrix sein. Wir bezeichnen dann auch B als eine sogenannte **Basis** (Menge von **Basisindizes**) und die Matrix A_B als **Basismatrix**.
- (iii) Die restlichen Indizes bilden die Indexmenge $N = \{1, \dots, n\} \setminus B$ der Mächtigkeit $|N| = n - m$ (Anzahl der Elemente in N). Die Untermatrix aus den zugehörigen Spalten von A bezeichnen wir mit A_N . Die Menge N heißt **Nichtbasis** (Menge von **Nichtbasisindizes**) und A_N **Nichtbasismatrix**.

Ebenso wie die Spalten der Matrix $A = [A_B, A_N]$ partitionieren wir Vektoren $x \in \mathbb{R}^n$ in

$x = [x_B, x_N]$. Dann setzen wir

$$x_i = 0 \quad \text{für alle } i \in N, \quad \text{kurz: } x_N = 0,$$

und bestimmen die restlichen Koordinaten x_B aus dem linearen Gleichungssystem

$$A_B x_B + A_N \underbrace{x_N}_{=0} = b, \quad \text{also: } A_B x_B = b, \quad (12.2)$$

das eindeutig lösbar ist. Der so erhaltene Vektor $x \in \mathbb{R}^n$ ist genau dann eine Ecke der zulässigen Menge in (12.1), wenn $x_B \geq 0$ gilt. Umgekehrt können *alle Ecken* der zulässigen Menge auf diese Art und Weise gefunden werden. Man nennt deshalb die Ecken auch **(zulässige) Basisvektoren**.

Nach der obigen Konstruktion gibt es höchstens soviele Ecken, wie es m -elementige Teilmengen von $\{1, \dots, n\}$ gibt:

$$\binom{n}{m} = \frac{n!}{m! (n-m)!} \quad (\text{sprich: „}n \text{ über } m\text{“})$$

Dazu ein kurzes Beispiel:¹⁰

Beispiel 12.1 (Generierung von Ecken eines Polyeders in Normalform). *Wir verwenden $n = 3$ Variablen und $m = 2$ Gleichungen, die durch*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \text{und} \quad b = \begin{pmatrix} 7 \\ 20 \end{pmatrix}$$

beschrieben werden. Es gibt drei Möglichkeiten, die Mengen B und N zu wählen, also auch höchstens drei Ecken:

- (1) $B = \{1, 2\}$ und $N = \{3\}$: Die Komponente x_3 ist null, und $x_{1,2}$ erhalten wir durch Lösen von

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 20 \end{pmatrix}, \quad \text{also zusammen: } x = \begin{pmatrix} 1.6667 \\ 2.6667 \\ 0.0000 \end{pmatrix}.$$

Dieser Vektor ist eine der Ecken.

- (2) $B = \{1, 3\}$ und $N = \{2\}$: Die Komponente x_2 ist null, und $x_{1,3}$ erhalten wir durch Lösen von

$$\begin{bmatrix} 1 & 3 \\ 4 & 6 \end{bmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 20 \end{pmatrix}, \quad \text{also zusammen: } x = \begin{pmatrix} 3.0000 \\ 0.0000 \\ 1.3333 \end{pmatrix}.$$

Dieser Vektor ist eine weitere Ecke.

¹⁰Wir nehmen hier nicht das Mozartproblem (Beispiel 11.1) als Beispiel, da wir dieses erst durch Einführung sogenannter Schlupfvariablen in Normalform bringen müssten. Wir erläutern dies später noch, siehe Seite 72.

(3) $B = \{2, 3\}$ und $N = \{1\}$: Die Komponente x_1 ist null, und $x_{2,3}$ erhalten wir durch Lösen von

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 20 \end{pmatrix}, \quad \text{also zusammen: } x = \begin{pmatrix} 0.0000 \\ 6.0000 \\ -1.6667 \end{pmatrix}.$$

Dieser Vektor ist keine Ecke, da nicht $x \geq 0$ gilt.

Das Polyeder in diesem Beispiel besitzt also zwei Ecken und ist damit eine Strecke im $\mathbb{R}^3$, siehe Abb. 12.2.

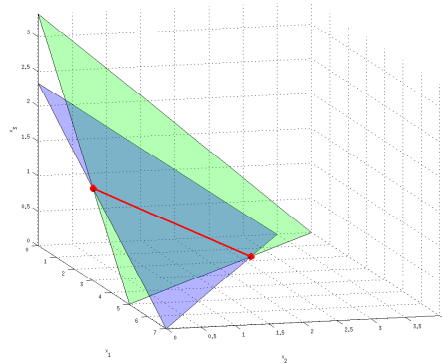


Abbildung 12.2.: Illustration des Polyeders aus Beispiel 12.1: Die Punkte, die die Gleichungsnebenbedingung $Ax = b$ erfüllen, liegen auf einer Geraden. Diese entsteht durch den Schnitt zweier Ebenen (grün und blau). Der Abschnitt der Gerade, der im Bereich $x \geq 0$ liegt, ist das gesuchte Polyeder: eine Strecke, die durch zwei Ecken begrenzt wird.

Nun aber zurück zur Beschreibung des primalen Simplex-Verfahrens für die Aufgabe (12.1) eines linearen Optimierungsproblems in Normalform. Das Verfahren startet immer in einer Ecke, sagen wir x .¹¹ Wir beschreiben nur einen Schritt des Verfahrens, der von der Ecke x zur neuen Ecke $\hat{x}$ führt. So sparen wir uns den Iterationsindex zu notieren.

Zur aktuellen Ecke x gehöre die Basis B und Nichtbasis N . Wir betrachten zunächst einen weiteren, beliebigen Vektor z in der zulässigen Menge, den wir ebenso partitionieren wie x , also $x = [x_B, x_N]$ und $z = [z_B, z_N]$. Auch den Kostenvektor schreiben wir als $c = [c_B, c_N]$. Um festzustellen, ob der Punkt z besser oder schlechter wäre als der aktuelle Punkt x , vergleichen wir deren Zielfunktionswerte:

$$\begin{aligned} c^T z &= c_B^T z_B + c_N^T z_N \\ &= c_B^T A_B^{-1} (b - A_N z_N) + c_N^T z_N \quad (\text{denn } A_B z_B + A_N z_N = b) \\ &= c_B^T x_B + (c_N^T - c_B^T A_B^{-1} A_N) z_N \quad (\text{denn } A_B x_B + A_N \underbrace{x_N}_{=0} = b) \\ &= c^T x + \underbrace{(c_N - A_N^T A_B^{-1} c_B)^T}_{=: \Delta_N} z_N. \end{aligned}$$

¹¹Wie sich das Verfahren diese erste Ecke beschaffen kann, dazu später mehr (Phase 1).

Den Vektor Δ_N bezeichnet man als die **reduzierten Kosten** zur Nichtbasis N .¹² Für den Vergleich der Funktionswerte gilt also:

$$\underbrace{c^T z}_{\text{Wert am Punkt } z} = \underbrace{c^T x}_{\text{Wert am Punkt } x} + \sum_{i \in N} \Delta_i z_i. \quad (12.3)$$

Hier sieht man schon Folgendes: Wenn die reduzierten Kosten

$$\Delta_N := c_N - A_N^T A_B^{-T} c_B \quad (12.4)$$

in allen Komponenten ≥ 0 sind, dann ist die aktuelle Ecke x bereits eine Optimallösung der Aufgabe (12.1), denn: Jeder zulässige Punkt z erfüllt natürlich $z_i \geq 0$, und damit ist sein Zielfunktionswert niemals kleiner als der von x , d. h., x ist ein globales Minimum.

Nehmen wir also an, dass Δ_N nicht in allen Komponenten ≥ 0 ist. Dann wollen wir zu einer benachbarten Ecke $\hat{x}$ – diese gehöre zur Basis $\hat{B}$ und Nichtbasis $\hat{N}$ – gehen, deren Funktionswert kleiner ist als der von x . „Benachbart“ bedeutet, dass nur ein Index zwischen B und N ausgetauscht wird, um die neuen Mengen $\hat{B}$ und $\hat{N}$ zu bilden.

Dazu wird ein Index $k \in N$ gewählt, der nach $\hat{B}$ wechselt, d. h., wir lassen $\hat{x}_k \geq 0$ werden. Die restlichen Einträge in $\hat{x}_N$ bleiben bei null. Aufgrund des obigen Vergleichs der Zielfunktionswerte in (12.3) wählt man den Index $k \in N$ so, dass $\Delta_k < 0$ ist. Diese Auswahl heißt **pricing**.

Wir wissen zunächst nicht, auf welchen (positiven) Wert wir $\hat{x}_k$ setzen sollen und machen daher den Ansatz

$$\hat{x}_k(t) := t \geq 0, \quad \hat{x}_i(t) := 0 \text{ für alle } i \in N \setminus \{k\}. \quad (12.5)$$

Dann ist der neue Funktionswert wegen (12.3) gleich

$$c^T \hat{x}(t) = c^T x + \Delta_N \hat{x}_N = c^T x + t \Delta_k \leq c^T x. \quad (12.6)$$

Die Gleichungsnebenbedingung $A \hat{x}(t) = b$ liefert

$$\begin{aligned} & A_B \hat{x}_B(t) + A_N \hat{x}_N(t) = b \\ \Leftrightarrow & A_B \hat{x}_B(t) + t a_k = b && (a_k \text{ ist die } k\text{-te Spalte von } A) \\ \Leftrightarrow & \hat{x}_B(t) = A_B^{-1} (b - t a_k) \\ \Leftrightarrow & \hat{x}_B(t) = x_B - t \underbrace{A_B^{-1} a_k}_{=: d_B}. \end{aligned} \quad (12.7)$$

Der Vektor d_B beschreibt die Richtung der Änderung beim Übergang von x_B zu $\hat{x}_B$. In Abhängigkeit von t wird $\hat{x}_N(t)$ durch (12.5) und $\hat{x}_B(t)$ durch (12.7) definiert.

Jetzt können wir die Frage klären, wie groß die Schrittweite $t \geq 0$ gewählt werden soll. Die Gleichung (12.6) sagt ja voraus, dass die Reduktion der Zielfunktion beim Übergang von x

¹²Die Bezeichnung A_N^T steht für $(A_N)^T$, also die Transponierte derjenigen Matrix A_N , die man aus den N -Spalten von A erhält. Eine ähnliche Notation nutzen wir für $A_B^{-T} = ((A_B)^T)^{-1} = ((A_B)^{-1})^T$.

zu $\hat{x}$ proportional zu t ist. Also möchte man $t \geq 0$ möglichst groß wählen. Andererseits ist $\hat{x}_B(t) \geq 0$ zu beachten, also:

$$t \geq 0 \quad \text{und} \quad \hat{x}_i(t) \stackrel{(12.7)}{=} x_i - t d_i \geq 0 \quad \text{für alle } i \in B. \quad (12.8)$$

Dies stellt eine Einschränkung nur in denjenigen Komponenten dar, für die $d_i > 0$ gilt. Die größtmögliche Schrittweite ist also bestimmt durch

$$\hat{t} := \min \left\{ \frac{x_i}{d_i} : i \in B, d_i > 0 \right\}. \quad (12.9)$$

Diese Berechnung nennt man den **Quotiententest** oder *ratio test*. Derjenige Index $r \in$

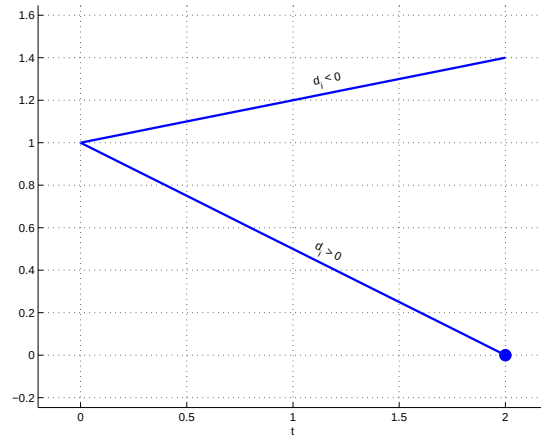


Abbildung 12.3.: Illustration der Begrenzung der Schrittweite t im Fall $d_i > 0$, siehe (12.8).

B , an dem das Minimum realisiert wird, wandert von der Basis B in die Nichtbasis $\hat{N}$.¹³ Dementsprechend ist dann auch $\hat{x}_r = 0$. Außerdem kann man sich davon überzeugen, dass auch die neue Basismatrix $A_{\hat{B}}$ wieder invertierbar ist, sodass wir wieder am Ausgangspunkt angekommen sind und die nächste Iteration starten können.

Was passiert aber, wenn im Quotiententest alle $d_i \leq 0$ sind und deshalb in (12.9) kein Minimum gebildet werden kann? Dann gibt es keine Einschränkung an die Schrittweite $t \geq 0$, und wir erhalten aus (12.6) eine beliebig große Reduktion des Wertes der Zielfunktion. In diesem Fall ist die Optimierungsaufgabe nicht lösbar, da die Werte der Zielfunktion über der zulässigen Menge nach unten unbeschränkt sind.¹⁴ Eine solche Situation ist in Abb. 12.5 (rechts) illustriert.

Wir fassen den kompletten (primalen) Simplex-Schritt, ausgehend von einem Basisvektor (Ecke) $x = [x_B, x_N]$ der Aufgabe (12.1) in Normalform, nochmals zusammen:

- (1) Berechne die reduzierten Kosten Δ_N , siehe (12.4).
- (2) Wenn $\Delta_N \geq 0$ ist, dann ist x eine Optimallösung der Aufgabe (12.1).
- (3) Andernfalls wähle einen Index $k \in N$ mit $\Delta_k < 0$ (*pricing*).

¹³Sollte das Minimum an mehreren Indizes gleichzeitig angenommen werden, so wählt man einfach einen davon aus.

¹⁴Dieses Ergebnis ist oft auf einen Modellierungsfehler in der Aufgabenstellung zurückzuführen.

- (4) Berechne d_B aus dem linearen Gleichungssystem $A_B d_B = a_k$.
- (5) Wenn $d_B \leq 0$ ist, dann hat die Aufgabe (12.1) keine Optimallösung, da die Zielfunktion über der zulässigen Menge nach unten unbeschränkt ist.
- (6) Andernfalls führe den Quotiententest (12.9) durch. Dabei sei $r \in B$ der Index bzw. einer der Indizes, an dem das Minimum auftritt.
- (7) Setze

$$\widehat{B} = B \cup \{k\} \setminus \{r\}, \quad \widehat{N} = N \cup \{r\} \setminus \{k\},$$

$$\widehat{x}_i = \begin{cases} x_i - \widehat{t} d_i, & \text{falls } i \in B \setminus \{r\}, \\ 0, & \text{falls } i = r, \\ \widehat{t}, & \text{falls } i = k, \\ 0, & \text{falls } i \in N \setminus \{k\}. \end{cases}$$

Zum besseren Verständnis wollen wir nun einen solchen Schritt des primalen Simplex-Verfahrens per „Handrechnung“ (natürlich mit Hilfe von MATLAB) nachvollziehen. Dazu verwenden wir unser [Beispiel 11.1](#). Die dazugehörige Minimierungsaufgabe (11.3) enthält Ungleichungen $A_{\text{ineq}} x \leq b_{\text{ineq}}$, daher müssen wir sie zunächst in Normalform bringen.¹⁵ Dies geschieht durch Einführung zusätzlicher Variablen (sogenannte **Schlupfvariablen**), von denen wir eine pro Ungleichung benötigen, hier (x_3, x_4, x_5) . Dann können wir die Ungleichungen

$$\begin{aligned} 1x_1 + 1x_2 &\leq 6 \\ 2x_1 + 1x_2 &\leq 11 \\ 1x_1 + 2x_2 &\leq 9 \end{aligned}$$

äquivalent umschreiben als

$$\begin{aligned} 1x_1 + 1x_2 + x_3 &= 6 \quad \text{mit } x_3 \geq 0, \\ 2x_1 + 1x_2 + x_4 &= 11 \quad \text{mit } x_4 \geq 0, \\ 1x_1 + 2x_2 + x_5 &= 9 \quad \text{mit } x_5 \geq 0. \end{aligned}$$

Unsere Aufgabe (11.3) lautet also in Normalform:¹⁶

$$\left. \begin{array}{l} \text{Minimiere} \quad \underbrace{\begin{pmatrix} -9 \\ -8 \\ 0 \\ 0 \\ 0 \end{pmatrix}^\top}_{c} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}, \quad x = (x_1, x_2, x_3, x_4, x_5)^\top \in \mathbb{R}^5 \\ \\ \text{unter} \quad \underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_b = \underbrace{\begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix}}_b \quad \begin{array}{l} \text{Marzipan} \\ \text{Nougat} \\ \text{Bitterschokolade} \end{array} \\ \\ \text{sowie} \quad x \geq 0. \end{array} \right\} \quad (12.10)$$

¹⁵Wie bereits erwähnt, findet diese Umwandlung in Normalform in `linprog` intern statt, wenn auch in etwas abgewandelter Form.

¹⁶Man beachte, dass die Schlupfvariablen (x_3, x_4, x_5) in der Zielfunktion nicht vorkommen und deshalb im Kostenvektor mit 0 gewichtet werden.

Abb. 12.4 zeigt die Ecken der zulässigen Menge mit den Werten der jeweiligen Schlupfvariablen. An diesem Bild können wir die nachfolgende Rechnung gut nachvollziehen.

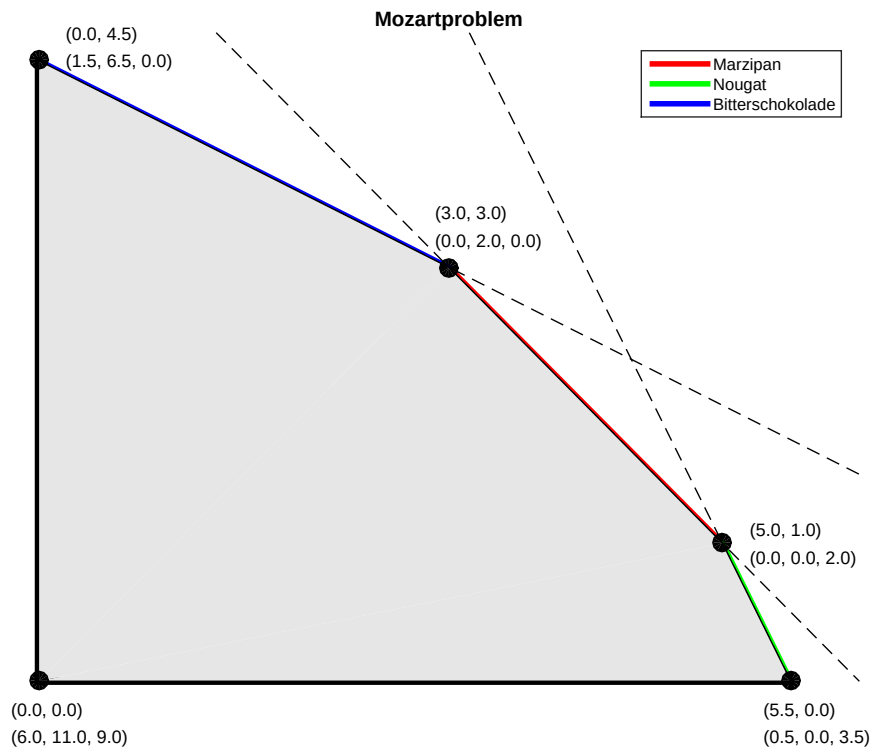


Abbildung 12.4.: Darstellung der Ecken der zulässigen Menge beim Mozartproblem (12.10) mitsamt den jeweiligen Werten der Schlupfvariablen.

Wir geben die Problemdata in MATLAB ein:

```
% Erstellen der Problembeschreibung in Normalform
A = [1 1 1 0 0; 2 1 0 1 0; 1 2 0 0 1];
b = [6; 11; 9];
c = [-9; -8; 0; 0; 0];
```

Matlab/Mozartproblem.Einzelschritt.m

Als Startvektor x wählen wir die linke untere Ecke, also $(x_1, x_2) = (0, 0)$ und passend dazu die Schlupfvariablen $(x_3, x_4, x_5) = b = (6, 11, 9)$:

```
% Setzen des Startvektors
x = [0; 0; b];
```

Matlab/Mozartproblem.Einzelschritt.m

Diese Ecke gehört zur Basis $B = \{3, 4, 5\}$ und Nichtbasis $N = \{1, 2\}$:

```
% Setzen der zugehörigen anfanglichen Basis und Nichtbasis
B = [3 4 5];
N = [1 2];
```

Matlab/Mozartproblem.Einzelschritt.m

Der Zielfunktionswert am Startpunkt

Matlab/Mozartproblem.Einzelschritt.m

```
% Bestimmung des Zielfunktionswertes
c' * x
```

ist natürlich null. In Schritt (1) berechnen wir die reduzierten Kosten Δ_N , siehe (12.4). Dabei ist es etwas einfacher, den ganzen Vektor $\Delta \in \mathbb{R}^n = \mathbb{R}^5$ zu berechnen und dann später nur die N -Komponenten weiter zu beachten:

```
% Schritt (1): Berechnung der reduzierten Kosten
% Schritt (2): Ueberpruefen der Vorzeichen (Test auf Optimalitaet)
Delta = c - A' * (A(:,B))' \ c(B)
Delta(N)
```

ergibt

$$\Delta = (-9, -8, 0, 0, 0)^T \quad \text{und damit} \quad \Delta_N = (-9, -8)^T.$$

Da also Δ_N noch negative Einträge hat, ist die aktuelle Ecke x noch keine Optimallösung unserer Aufgabe, siehe Schritt (2). In Schritt (3), dem *pricing*, wählen wir einen der beiden Indizes in N aus, hier willkürlich

```
% Schritt (3): Auswahl eines Index k aus N
k = 1
```

Wir wählen damit die Kante aus, die uns zur nächsten Ecke führen wird, hier ist es die Achse der ersten Variable, also die x_1 -Achse.

In Schritt (4) bestimmen wir die Richtung d_B . Ähnlich wie oben beim Vektor der reduzierten Kosten ist es praktisch, stattdessen einen Vektor $d \in \mathbb{R}^n = \mathbb{R}^5$ einzuführen und nur mit dessen B -Komponenten zu arbeiten:

```
% Schritt (4): Berechnung der Richtung d
d = zeros(size(x));
d(B) = A(:,B) \ A(:,k)
```

Die Abbruchbedingung in Schritt (5) trifft nicht zu, da

$$d = (0, 0, 1, 2, 1)^T \quad \text{und damit} \quad d_B = (1, 2, 1)^T$$

nicht ≤ 0 ist, also geht es in Schritt (6) weiter mit dem Quotiententest, vgl. (12.9): Wir berechnen der Einfachheit halber *alle* Quotienten x_i/d_i für $i = 1, \dots, n$ statt nur für diejenigen Indizes $i \in B$ mit $d_i > 0$. Dann setzen wir einfach diejenigen Einträge in q , die nicht an der Minimumbildung teilnehmen sollen, auf NaN (*not a number*).¹⁷

```
% Schritt (6): Durchfuehrung des Quotiententests
q = x ./ d;
q(find(d<=0)) = NaN;
[that,r] = min(q);
```

¹⁷Den Wert NaN haben ohnehin schon die Einträge in q , die in Folge einer Division durch null entstanden sind, insbesondere die Einträge $q(N)$

Die Variable `that` enthält jetzt den Wert $\hat{t}$ aus (12.9), hier $\hat{t} = 5.5$, und $r = 4$ ist derjenige Index, wo dieser Wert angenommen wird.¹⁸ In Schritt (7) müssen nur noch die Koordinaten der neuen Ecke $\hat{x}$ berechnet werden und die zugehörige Basis $\hat{B}$ und Nichtbasis $\hat{N}$:

Matlab/Mozartproblem.Einzelschritt.m

```
% Schritt (7): Setzen der naechsten Iterierten
% und der zugehoerigen Basis und Nichtbasis
xhat = zeros(size(x));
xhat(B) = x(B) - that * d(B);
xhat(k) = that
Bhat = setdiff(union(B,k),r)
Nhat = setdiff(union(N,r),k)
```

Es ist $\hat{x} = (5.5, 0, 0.5, 0, 3.5)^T$ zur Basis $\hat{B} = \{1, 3, 5\}$ und Nichtbasis $\hat{N} = \{2, 4\}$. Der zugehörige Wert der Zielfunktion ist -49.5 , wie

Matlab/Mozartproblem.Einzelschritt.m

```
% Bestimmung des neuen Zielfunktionswertes
c' * xhat
```

zeigt, und damit wie erwartet kleiner als am Startwert. In Abb. 12.4 haben wir jetzt die Ecke rechts unten erreicht. Mit einem weiteren Simplex-Schritt würden wir die Lösung $x^* = (5, 1, 0, 0, 2)^T$ (zu $B = \{1, 2, 5\}$ und $N = \{3, 4\}$) mit dem minimalen Funktionswert $c^T x^* = -53$ erreichen.

Es bleibt außerdem noch nachzutragen, wie das primale Simplex-Verfahren überhaupt die erste Ecke findet, die zum Start benötigt wird. Dazu schaltet das Verfahren intern eine sogenannte Phase 1 vor: Darin wird ein Hilfsproblem mit zusätzlichen Variablen gelöst.¹⁹ Wir betrachten wieder nur den Fall, dass das Ausgangsproblem in Normalform (12.1) vorliegt. Dann lautet das Hilfsproblem:

$$\begin{aligned} &\text{Minimiere} && e^T s, && (x, s) \in \mathbb{R}^n \times \mathbb{R}^m \\ &\text{unter} && Ax + s = b \\ &\text{sowie} && x \geq 0, \quad s \geq 0. \end{aligned} \tag{12.11}$$

Dabei ist $e = (1, \dots, 1)^T \in \mathbb{R}^m$ ein Vektor aus lauter Einsen. Da die Phase-1-Aufgabe (12.11) wiederum eine lineare Optimierungsaufgabe ist, lässt sich das Simplex-Verfahren zur Lösung anwenden. Nur hat man dieses Mal den Vorteil, dass man eine erste Ecke kennt, nämlich den Vektor $x = 0$ und $s = b \geq 0$.²⁰

Man kann zeigen, dass die Hilfsaufgabe (12.11) immer lösbar ist. Die berechnete Lösung bezeichnen wir mit (x^*, s^*) . Ist $s^* = 0$ (Nullvektor), so ist x^* eine zulässige Ecke des Ausgangsproblems (12.1). Damit kann das Simplex-Verfahren für die eigentliche Aufgabe (12.1) von dort aus gestartet werden, was man dann als Phase 2 bezeichnet. Ist dagegen $s^* \neq 0$, so besitzt das Ausgangsproblem (12.1) überhaupt keine zulässigen Punkte, also auch keine Optimallösung.²¹

¹⁸Bei mehreren kleinsten Werten in einem Vektor liefert `min` nur einen Index zurück, und zwar den niedrigsten.

¹⁹Beim obigen Beispiel des Mozartproblems (11.3) konnten wir aber sogar die Phase 1 überspringen, da die erste Ecke sofort aus der Abb. 12.4 abgelesen werden konnte.

²⁰Falls eine oder mehrere Komponenten von b negativ sind, so werden zunächst noch die entsprechenden Zeilen

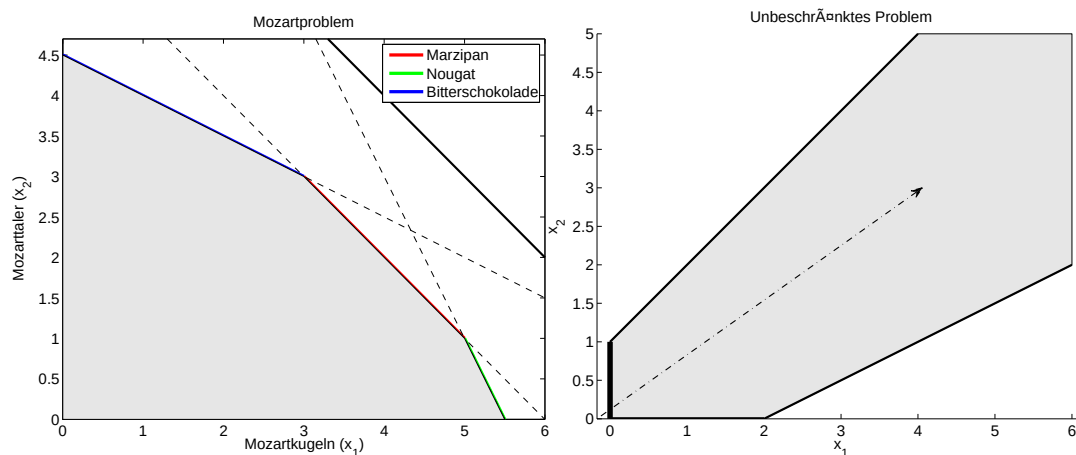


Abbildung 12.5.: Fügt man dem Mozartproblem (Beispiel 11.1) die Beschränkung $x_1 + x_2 \geq 8$ (Gesamtproduktionsmenge) hinzu, wird der zulässige Bereich leer (links). Mit den zur Verfügung stehenden Lagerbeständen ist das nicht zu schaffen. Die Aufgabe besitzt dann keinen zulässigen Punkt, was in Phase 1 festgestellt wird. Das rechte Bild illustriert folgende lineare Optimierungsaufgabe: Maximiere $4x_1 + 3x_2$ unter den Bedingungen $-\frac{1}{2}x_1 + x_2 \geq -1$, $-x_1 + x_2 \leq 1$, $x_1 \geq 0$ und $x_2 \geq 0$. Hier kann die Zielfunktion in der zulässigen Menge beliebig große Werte erreichen. Diese Situation wird im Quotiententest erkannt.

Ende der Woche 9

§ 13 LAGRANGE-MULTIPLIKATOREN, DUALE AUFGABEN, DAS DUALE SIMPLEX-VERFAHREN UND SCHATTENPREISE

Wir kehren zurück zum Optimalitätssystem (11.5). Es stellt sich die Frage, wie man auf ein solches Optimalitätssystem kommt und welche Bedeutung die Lagrange-Multiplikatoren λ_{ineq} , λ_{eq} , λ_l und λ_u haben. Der Einfachheit halber besprechen wir die Details nur für das vereinfachte lineare Optimierungsproblem in Normalform, vgl. (12.1):²²

$$\begin{aligned} &\text{Minimiere} && c^T x, && x \in \mathbb{R}^n \\ &\text{unter} && Ax = b \\ &\text{sowie} && x \geq 0. \end{aligned} \tag{13.1}$$

Dabei ist wie gehabt $A \in \mathbb{R}^{m \times n}$ eine Matrix und $b \in \mathbb{R}^m$ der Vektor der rechten Seite. Wir

von $Ax = b$ mit (-1) multipliziert.

²¹Dieses Ergebnis ist entweder auf einen Modellierungsfehler in der Aufgabenstellung zurückzuführen oder, etwa in Produktionsaufgaben, auf Anforderungen, die sich mit den vorhandenen Ressourcen nicht erfüllen lassen.

²²Der Vereinfachung der Notation wegen schreiben wir einfach A statt A_{eq} und b statt b_{eq} .

erwarten, dass wir aus (11.5) durch Weglassen der nicht vorhandenen Ungleichungsbedingungen $A_{\text{ineq}} x \leq b_{\text{ineq}}$ und $x \leq u$ und der zugehörigen Multiplikatoren sowie durch Einsetzen der unteren Schranke ($\ell = 0$) das passende Optimalitätssystem für (13.1) erhalten, also

$$\left. \begin{aligned} A_{\text{eq}}^T \lambda_{\text{eq}} - \lambda_{\ell} + c &= 0, \\ \lambda_{\ell} &\geq 0, \quad x \geq 0, \quad \lambda_{\ell}^T x = 0, \\ A_{\text{eq}} x - b_{\text{eq}} &= 0. \end{aligned} \right\} \quad (13.2)$$

Wir wollen nun untersuchen, wie man darauf kommt.

Tatsächlich ist es so, dass man in der linearen Optimierung immer automatisch mit dem Originalproblem, hier (13.1), ein sogenanntes **duales Problem** gleichzeitig mit löst, dessen Optimierungsvariablen die Lagrange-Multiplikatoren, hier $\lambda_{\text{eq}} \in \mathbb{R}^m$ und $\lambda_{\ell} \in \mathbb{R}^n$ sind. Wenn wir (13.2) verstehen wollen, müssen wir zunächst das duale Problem zu (13.1) verstehen. Dazu machen wir folgendes Gedankenexperiment und nehmen uns einen beliebigen zulässigen Punkt $x \in \mathbb{R}^n$ für (13.1), dieser erfüllt also $x \geq 0$ und $Ax = b$.²³ Damit erfüllt der Punkt auch jede Linearkombination der Gleichungen, beispielsweise zweimal die erste Gleichung (Zeile) von $Ax = b$ minus dreimal die zweite Gleichung von $Ax = b$.²⁴ Solche Linearkombinationen lassen sich durch Multiplikation mit einem Zeilenvektor von links ausdrücken. Es gilt also für unseren zulässigen Punkt x von (13.1), dass er auch

$$y^T Ax = y^T b$$

erfüllt, und zwar für jede beliebige Linearkombination der Zeilen von $Ax = b$, also für jeden beliebigen Vektor $y \in \mathbb{R}^m$. Man beachte, dass wir die obige Gleichung natürlich auch schreiben können in der Form (alles einmal transponieren)

$$x^T A^T y = b^T y$$

für jeden beliebigen Vektor $y \in \mathbb{R}^m$.

Natürlich bringt es nichts, einfach nur Linearkombinationen zu bilden, aber wir könnten ja die Wahlmöglichkeit in y vielleicht dazu nutzen, etwas zu optimieren, was nach Möglichkeit auch noch etwas mit unserer eigentlichen Aufgabenstellung (13.1) zu tun haben sollte. Dazu betrachten wir den Term $b^T y$ und schränken die (bisher beliebigen) Linearkombinationen dahingehend ein, dass wir für y die Bedingung $A^T y \leq c$ fordern. Dann gilt nämlich:

$$b^T y = \underbrace{x^T}_{\geq 0} \underbrace{A^T y}_{\leq c} \leq x^T c = c^T x.$$

Nochmal in Worten ausgedrückt: Achtet man darauf, dass der (ansonsten beliebige) Linearkombinationsvektor $y \in \mathbb{R}^m$ die Bedingung $A^T y \leq c$ erfüllt, dann ist der Wert von $b^T y$ immer eine untere Schranke für den Wert der Zielfunktion $c^T x$ in (13.1), und zwar für alle $x \in \mathbb{R}^n$, die für (13.1) zulässig sind. Das legt die Idee nahe, den Ausdruck $b^T y$ zu maximieren, um die größte solche untere Schranke zu finden, die also den Zielfunktionswert $c^T x$ von (13.1), den

²³Der Einfachheit halber schreiben wir nun $Ax = b$ statt $A_{\text{eq}} x = b_{\text{eq}}$.

²⁴For simplicity, we write $Ax = b$ instead of $A_{\text{eq}} x = b_{\text{eq}}$.

wir ja minimieren wollen, am stärksten nach unten einschränkt. Genau so definiert man das zu (13.1) **duale Problem**:²⁵

$$\begin{aligned} &\text{Maximiere} && b^T y, && y \in \mathbb{R}^m \\ &\text{unter} && A^T y \leq c. \end{aligned} \tag{13.3}$$

Natürlich kann man auch hier wieder Schlupfvariablen (sagen wir: $z \in \mathbb{R}^n$) einführen, um die Ungleichungen als Gleichungen zu formulieren:²⁶

$$\begin{aligned} &\text{Maximiere} && b^T y, && (y, z) \in \mathbb{R}^m \times \mathbb{R}^n \\ &\text{unter} && A^T y + z = c \\ &\text{und} && z \geq 0. \end{aligned} \tag{13.4}$$

Im Zusammenhang mit der dualen Aufgabe (13.3) oder äquivalent (13.4) bezeichnet man das Ausgangsproblem (13.1) auch als **primale Aufgabe**. Ein Vektor $x \in \mathbb{R}^n$ wird als **primal zulässig** bezeichnet, wenn er die Nebenbedingungen von (13.1) erfüllt. Ein Vektor $y \in \mathbb{R}^m$ bzw. ein Paar von Vektoren $(y, z) \in \mathbb{R}^m \times \mathbb{R}^n$ wird als **dual zulässig** bezeichnet, wenn die Nebenbedingungen von (13.3) bzw. (13.4) erfüllt sind.

Weil die Ungleichung $b^T y \leq c^T x$ natürlich nicht nur den Wert der primalen Zielfunktion $c^T x$ nach unten beschränkt, sondern umgekehrt auch den Wert der dualen Zielfunktion $b^T y$ nach oben, liegt hier eine symmetrische Situation vor.²⁷ Man spricht deshalb bei (13.1)/(13.3) bzw. (13.1)/(13.4) auch einfach von **primal-dualen Paaren** linearer Optimierungsaufgaben.

Wir halten das bisher Gesehene in einem Satz fest:

Satz 13.1 (Schwache Dualität in der linearen Optimierung). *Ist x primal zulässig für (12.1) und (y, z) dual zulässig für (13.4), dann gilt die Ungleichung $b^T y \leq c^T x$. Dies nennt man **schwache Dualität**.*

Natürlich ist damit die Frage nach den Optimalitätsbedingungen (13.2) und der Bedeutung der Lagrange-Multiplikatoren noch überhaupt nicht geklärt. Dazu rechnen wir einmal aus, wie groß die Lücke zwischen den jeweiligen Zielfunktionswerten für primal bzw. dual zulässige Punkte x bzw. (y, z) ist:

$$c^T x - b^T y \stackrel{Ax=b}{=} x^T c - x^T A^T y = x^T (c - A^T y) \stackrel{A^T y + z = c}{=} x^T z \stackrel{\substack{x \geq 0 \\ z \geq 0}}{\geq} 0.$$

Die Lücke hat also gerade den Wert $x^T z$. Hat man Vektoren x und (y, z) gefunden, bei denen die Lücke gleich null ist (also $b^T y = c^T x$ gilt), dann muss x optimal für das primale Problem (13.1) sein, denn kleiner als die Schranke $b^T y$ kann der Zielfunktionswert $c^T x$ ja nicht mehr werden. Umgekehrt ist dann auch (y, z) optimal für das duale Problem (13.4).

²⁵**Beachte:** Das duale Problem wird mit Hilfe derselben Daten (A, b, c) wie das primale Problem (13.1) formuliert. Diese gehen natürlich anders in die Aufgabenstellung ein.

²⁶Dennoch ist (13.4) noch kein LP in Normalform, da y keine Vorzeichenbedingung $y \geq 0$ erfüllen muss.

²⁷Die primale Aufgabe kann wieder verstanden werden als die duale Aufgabe zur dualen Aufgabe (13.3) oder (13.4).

Wir können nun eine erste Deutung der Lagrange-Multiplikatoren geben. Benennen wir nämlich einfach in der dualen Aufgabe die Variablen anders:

$$y \leftrightarrow -\lambda_{\text{eq}} \quad \text{und} \quad z \leftrightarrow \lambda_{\ell},$$

dann sehen die Zielfunktionen und die Nebenbedingungen in (13.1) und (13.4) wie folgt aus:

primale Zielfunktion	duale Zielfunktion
$c^T x$	$b^T (-\lambda_{\text{eq}})$
primale Zulässigkeit	duale Zulässigkeit
$A x = b$	$A^T (-\lambda_{\text{eq}}) + \lambda_{\ell} = c$
$x \geq 0$	$\lambda_{\ell} \geq 0$

Die Lagrange-Multiplikatoren sind also (bis auf das Vorzeichen) gerade die Optimierungsvariablen der dualen Aufgabe! Sie werden deshalb auch als **duale Variablen** bezeichnet. Und wie wir gerade gesehen haben, hängt die Optimalität von x (und gleichzeitig die Optimalität von (y, z)) davon ab, ob die Bedingung $x^T z = 0$ erfüllt ist bzw. nach Umbenennung:

$$x^T \lambda_{\ell} = 0.$$

Zusammen sind dies genau unsere Optimalitätsbedingungen aus (13.2)! Diese können wir also jetzt neu deuten. Sie bestehen aus den Bedingungen zur primalen und dualen Zulässigkeit, und für die Optimalität ist dann entscheidend die **Komplementaritätsbedingung** $x^T \lambda_{\ell} = 0$ zuständig, also die Übereinstimmung der primalen und dualen Zielfunktionswerte.

Damit ist die Herkunft des Optimalitätssystems (13.2) eines LP in Normalform (13.1) geklärt. Ganz analog kann man zeigen, dass zum allgemeinen LP (11.1) das Optimalitätssystem (11.5) gehört. Auch hier läuft die Betrachtung wieder darüber, dass man ein zu (11.1) passendes duales Problem formuliert, dessen Zielfunktion untere Schranken für die Zielfunktion in (11.1) liefert.

Bei näherem Hinsehen erkennt man, dass der Simplex-Algorithmus intern sogar die dualen Variablen (y, z) bereits mitführt: Bei der Berechnung der reduzierten Kosten wird die Zwischengröße

$$y := A_B^{-T} c_B$$

benötigt:

$$\Delta_N = c_N - A_N^T \underbrace{A_B^{-T} c_B}_y = c_N - A_N^T y.$$

Nach Definition sind daher die B -Zeilen der dualen Zulässigkeitsbedingung $A^T y \leq c$ (mit Gleichheit) erfüllt:

$$A_B^T y = c_B.$$

Wenn man nun die dualen Schlupfvariablen ebenfalls in $z = [z_B, z_N]$ aufteilt, so folgt

$$z_B := c_B - A_B^T y = 0.$$

Für die verbleibenden N -Zeilen gilt

$$z_N := c_N - A_N^T y \geq 0 \quad \Leftrightarrow \quad \Delta_N \geq 0.$$

Dies bedeutet, dass die Bedingung „Optimallösung gefunden, da $\Delta_N \geq 0$ “ in unserem Simplex-Verfahren gleichbedeutend damit ist, dass die (implizit mitgeführten) dualen Variablen (y, z) für die duale Aufgabe (13.4) zulässig (engl.: *feasible*) sind. Während der Simplex-Iterationen ist dabei immer nur die Bedingung $z \geq 0$ verletzt, denn für alle anderen Bedingungen im Optimalitätssystem (13.2) ist ja gesorgt:

primale Zulässigkeit	$Ax = b$	durch Konstruktion der Iterierten (Ecken)
primale Zulässigkeit	$x \geq 0$	durch Konstruktion der Iterierten (Ecken)
duale Zulässigkeit	$A^T y + z = c$	nach Definition von (y, z)
Komplementarität	$x^T z = 0$	gilt, wie man einfach nachrechnet:

$$x^T z = \underbrace{x_B^T z_B}_{=0} + \underbrace{x_N^T z_N}_{=0} = 0.$$

Sobald $z \geq 0$ gilt (also $\Delta_N \geq 0$), sind alle Bedingungen in (13.2) erfüllt, und das Simplex-Verfahren hat eine Lösung gefunden.

Der Vollständigkeit halber geben wir noch den starken Dualitätssatz der linearen Optimierung an:

Satz 13.2 (Starke Dualität der linearen Optimierung). *Das primale lineare Optimierungsproblem in Normalform (13.1) hat genau dann eine Optimallösung²⁸, wenn das zugehörige duale Problem (13.3) bzw. (13.4) eine Optimallösung besitzt. Die Werte der Zielfunktion sind dann gleich, und für jedes Paar aus primaler und dualer Optimallösung x und (y, z) ist die Komplementaritätsbedingung $x^T z = 0$ erfüllt.*

Der Satz gilt analog auch für lineare Optimierungsaufgaben, die nicht in Normalform vorliegen, etwa (11.1).

§ 13.1 DAS DUALE SIMPLEX-VERFAHREN

Statt dem oben ausführlich besprochenen primalen Simplex-Verfahren ist in `linprog` das sogenannte **duale Simplex-Verfahren** implementiert, das wir durch

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('linprog','Algorithm','dual-simplex','
    Diagnostics','on','Display','iter');
```

ausgewählt hatten. Diese Variante gilt in vielen Situationen als schneller als das primale Simplex-Verfahren. Außerdem lässt sie sich als Unteroutine bei der Lösung von linearen Programmen mit Ganzzahligkeitsnebenbedingungen verwenden, siehe (16.3).

Das duale Simplex-Verfahren verwendet im Grunde dieselben Techniken wie das primale Verfahren. Die wesentlichen Unterschiede, die die Eigenschaften der Iterierten des Verfahrens betreffen, sind in folgender Tabelle zusammengestellt.

²⁸ist also weder unbeschränkt noch unzulässig

	Eigenschaft	primales Simplex-Verfahren	duales Simplex-Verfahren
primale Zulässigkeit	$x_B \geq 0$	✓	erst in der Lösung
	$x_N = 0$	✓	✓
	$Ax = b$	✓	✓
duale Zulässigkeit	$z_B = 0$	✓	✓
	$z_N \geq 0$	erst in der Lösung	✓
	$A^T y + z = c$	✓	✓
Komplement.	$x^T z = 0$	✓	✓

Wir werfen mit diesem Wissen nochmals einen Blick auf die Ausgabe von `linprog` bei der Lösung des Mozartproblems ([Beispiel 11.1](#)):

```

-----
Diagnostic Information

Number of variables: 2

Number of linear inequality constraints: 3
Number of linear equality constraints: 0
Number of lower bound constraints: 2
Number of upper bound constraints: 0

Algorithm selected
dual-simplex

-----
End diagnostic information

LP preprocessing removed 0 inequalities, 0 equalities,
0 variables, and added 0 non-zero elements.

Iter      Time          Fval  Primal Infeas  Dual Infeas
  0         0      0.000000e+00  0.000000e+00  6.758116e+00
  1         0     -5.400000e+01  8.908987e-01  0.000000e+00
  2         0     -5.300000e+01  0.000000e+00  0.000000e+00

Optimal solution found.

```

Die duale Unzulässigkeit in Iteration 0, die ja eigentlich nicht vorkommen dürfte, rührt daher, dass zur Bestimmung einer dual zulässigen Startecke von `linprog` zunächst ein internes Hilfsproblem (analog zu Phase 1 beim primalen Simplex-Verfahren) gelöst werden muss, dessen Iterationen hier mit aufgeführt werden. Das eigentliche duale Simplex-Verfahren (Phase 2) startet hier dann in Iteration 1.

Aufgrund der Ähnlichkeit zum primalen Simplex-Verfahren gehen wir auf das duale Verfahren hier jetzt nicht näher ein. Für Details verweisen wir auf [Vanderbei, 2008](#), Kapitel 6.4,

Nocedal, Wright, 2006, Kapitel 13.6 und das Handbuch der MATLAB-Optimierungs-Toolbox.

§ 13.2 INTERPRETATION DER LAGRANGE-MULTIPLIKATOREN ALS SCHATTENPREISE

Wir besprechen nun noch eine weitere Bedeutung der Lagrange-Multiplikatoren am Beispiel der Aufgabe (13.1) in Normalform. Wir betrachten dazu, was bei einer Veränderung der rechten Seite b in $Ax = b$ passiert. Diese Fragestellung tritt auf, wenn z. B. beim Mozartproblem mit Schlupfvariablen (12.10) die Lagerbestände (Ressourcen) verändert werden, beispielsweise durch eine Berichtigung der Zahlen bei einer Inventur oder durch Verderben von Rohstoffen.

Als Ausgangspunkt starten wir mit einer primalen und dualen Optimallösung x bzw. (y, z) für die (unveränderten) Aufgaben (13.1) bzw. (13.4), die also die Optimalitätsbedingungen (13.2) erfüllen. x sei eine Ecke zur Basis B .

Wir betrachten das zugehörige primal-duale Paar unter einer Änderung von b in $b + t \Delta b$ ($t \in \mathbb{R}$):

$$\left. \begin{array}{ll} \text{Minimiere} & c^T x \\ \text{sodass} & Ax = b + t \Delta b \\ \text{und} & x \geq 0 \end{array} \right\} \quad \left\{ \begin{array}{ll} \text{Maximiere} & (b + t \Delta b)^T y \\ \text{sodass} & A^T y + z = c \\ \text{und} & z \geq 0. \end{array} \right. \quad (13.5)$$

Die duale Zulässigkeit von (y, z) bleibt also auch für das gestörte Problempaar erhalten, die Störung führt aber zu Änderungen in der primal zulässigen Menge. Die Optimallösung x wird sich also in Abhängigkeit der Störungsgröße t ändern müssen. Die geringstmögliche Anpassung, mit der man von x und (y, z) zu einer Lösung des gestörten Problempaares kommen könnte, wäre, die primale Basis B sowie (y, z) konstant zu halten und die erforderlichen Anpassungen an x_B durchzuführen:

$$x_B(t) = A_B^{-1}(b + t \Delta b) = x_B + t \underbrace{A_B^{-1} \Delta b}_{=: \Delta x_B}. \quad (13.6)$$

Können wir so bereits eine Lösung des gestörten Problempaares gefunden haben? Wir überprüfen einfach die Optimalitätsbedingungen (13.2)!

primale Zulässigkeit	$Ax = b + t \Delta b$	gilt nach Konstruktion
primale Zulässigkeit	$x \geq 0$	$x_B(t) \geq 0$ ist zu überprüfen!
duale Zulässigkeit	$A^T y + z = c$	gilt auch für die gestörte Aufgabe
duale Zulässigkeit	$z \geq 0$	gilt auch für die gestörte Aufgabe
Komplementarität	$x(t)^T z = 0$	wie man einfach nachrechnet:

$$x(t)^T z = x_B(t)^T \underbrace{z_B}_{=0} + x_N(t)^T \underbrace{z_N}_{=0} = 0.$$

Mit dieser geringstmöglichen Anpassung haben wir also tatsächlich ein primal-duales Lösungspaar des gestörten Aufgabenpaares gefunden, falls $x_B(t) \geq 0$ bleibt!²⁹ Den optimalen Wert der Zielfunktion der gestörten Lösung können wir (wegen der starken Dualität) nicht nur aus $c^T x(t)$ ablesen, sondern auch aus der dualen Aufgabe:

$$(b + t \Delta b)^T y$$

Der Unterschied zum nominalen Problem ($t = 0$) beträgt also gerade

$$t \Delta b^T y = \underbrace{t \Delta b^T}_{\text{Störung}} \underbrace{(-\lambda_{\text{eq}})}_{\text{Faktor}}.$$

Damit erkennen wir eine weitere Bedeutung des Lagrange-Multiplikators λ_{eq} , der zur Bedingung $Ax = b$ gehört: Er gibt den **Empfindlichkeitsfaktor** (die **Sensitivität**) an, mit der sich der optimale Zielfunktionswert einer linearen Optimierungsaufgabe ändert, wenn die rechte Seite b zu $b + t \Delta b$ verändert wird. Diese Erkenntnis gilt auch bei der allgemeinen (nicht in Normalform vorliegenden) linearen Optimierungsaufgabe (11.1), d. h., die Multiplikatoren λ_{ineq} , λ_{eq} , λ_ℓ und λ_u geben (mit negativem Vorzeichen) die Empfindlichkeit des optimalen Zielfunktionswertes bei Änderungen in den Problemdata b_{ineq} , b_{eq} , ℓ bzw. u an.³⁰ Die vom (primalen oder dualen) Simplex-Verfahren mitberechneten und von linprog auch zurückgegebenen Multiplikatoren sind also keinesfalls nur unnützer Ballast, sondern sie tragen wertvolle Information. Wir wollen dies nun am Beispiel des Mozartproblems (11.3) erläutern:

$$\left. \begin{array}{ll} \text{Minimiere} & \underbrace{\begin{pmatrix} -9 \\ -8 \end{pmatrix}^T}_{c} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad x = (x_1, x_2)^T \in \mathbb{R}^2 \\ \text{unter} & \underbrace{\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix}}_{A_{\text{ineq}}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \underbrace{\begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix}}_{b_{\text{ineq}}} \quad \begin{array}{l} \text{Marzipan} \\ \text{Nougat} \\ \text{Bitterschokolade} \end{array} \\ \text{sowie} & \underbrace{0}_{\ell} \leq x. \end{array} \right\} \quad (11.3)$$

Wir hatten als Lösung aus linprog den Vektor $x = (5, 1)$ erhalten und als Multiplikatoren

$$\lambda_{\text{ineq}} = \begin{pmatrix} 7 \\ 1 \\ 0 \end{pmatrix} \quad \text{und} \quad \lambda_\ell = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Wir wollen nun annehmen, dass sich der Lagerbestand an Marzipan von 6 auf 6.5 Einheiten erhöht hat. Die anderen Ressourcen bleiben gleich. Dies drücken wir z. B. durch den Vektor

$$\Delta b_{\text{ineq}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

²⁹Dies ist genau dann der Fall, wenn die Größe der Störung t ausreichend klein ist:

$$\max_{\substack{i \in B \\ \Delta x_i > 0}} \underbrace{\left\{ -\frac{x_i}{\Delta x_i} \right\}}_{\leq 0} \leq t \leq \min_{\substack{i \in B \\ \Delta x_i < 0}} \underbrace{\left\{ -\frac{x_i}{\Delta x_i} \right\}}_{\geq 0}.$$

³⁰geeignet kleine Änderungen der Problemdata vorausgesetzt

und Störungsgröße $t = 0.5$ aus. Unsere obigen Betrachtungen sagen voraus, dass sich der optimale Zielfunktionswert (bisher $c^T x = -9 \cdot 5 - 8 \cdot 1 = -53$) ändern wird um den Wert

$$\underbrace{t \Delta b_{\text{ineq}}^T}_{\text{Störung}} \underbrace{(-\lambda_{\text{ineq}})}_{\text{Faktor}} = 0.5 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} -7 \\ -1 \\ 0 \end{pmatrix} = -3.5.$$

Dies gilt zumindest dann, wenn die Größe der Störung nicht dazu führt, dass $x_B(t) \geq 0$ verletzt wird.³¹ Um dies zu überprüfen, berechnen wir also zunächst $x(t)$ gemäß (13.6):

```
% Setzen der Basis zur Loesung xstar
B = [1; 2; 5];
% Setzen der Matrix A fuer Normalform
A = [Aineq, eye(3)];
% Setzen der Loesung in Normalform
x = [xstar; bineq - Aineq * xstar];
% Setzen der Stoerung
t = 0.5;
Delta_bineq = [1; 0; 0];
% Berechnen der voraussichtlichen gestoerten Loesung
x_vor = x;
Delta_x = zeros(size(x));
Delta_x(B) = A(:,B) \ Delta_bineq;
x_vor(B) = x_vor(B) + t * Delta_x(B);
```

Das Ergebnis, $x_{\text{vor}} = (4.5, 2.0)$ mit Schlupfvariablen $(0, 0, 0.5)$, ist primal zulässig, wird also eine (sogar die eindeutige) Lösung der gestörten Aufgabe sein, und der zugehörige optimale Zielfunktionswert wird

$$-53 - 3.5 = -56.5$$

betragen. Wohlgemerkt können wir diese Aussagen treffen, ohne dass wir erneut die gestörte Aufgabe mit `linprog` lösen mussten! Dennoch tun wir dies nun zur Überprüfung:

```
% Aufruf des Optimierungsverfahrens
[x_gestoert, f_gestoert, exitflag, output] = linprog(c, Aineq, bineq+t*
    Delta_bineq, Aeq, beq, l, u, options);
```

Wir erhalten als Lösung tatsächlich $x_{\text{gestört}} = (4.5, 2.0)$ und als weitere Ausgabe:

```
-----
Diagnostic Information

Number of variables: 2
```

³¹Im Beispiel lautet (13.6) konkret: $x_B(t) = \begin{pmatrix} 5 \\ 1 \\ -2 \end{pmatrix} + t \begin{pmatrix} -1 \\ 2 \\ -3 \end{pmatrix}$, sodass Störungen $-1/2 \leq t \leq 2/3$ zulässig sind. Für zunehmende Störungsgröße $t \geq 0$ wird also als erstes der Überschuss (Schlupfvariable) an Bitterschokolade erschöpft sein, und zwar bei $t = 2/3$.

```

Number of linear inequality constraints: 3
Number of linear equality constraints: 0
Number of lower bound constraints: 2
Number of upper bound constraints: 0

```

```

Algorithm selected
dual-simplex

```

```

-----
End diagnostic information

```

```

LP preprocessing removed 0 inequalities, 0 equalities,
0 variables, and added 0 non-zero elements.

```

Iter	Time	Fval	Primal Infeas	Dual Infeas
0	0	0.000000e+00	0.000000e+00	6.758116e+00
1	0	-5.850000e+01	1.781797e+00	0.000000e+00
2	0	-5.650000e+01	0.000000e+00	0.000000e+00

```

Optimal solution found.

```

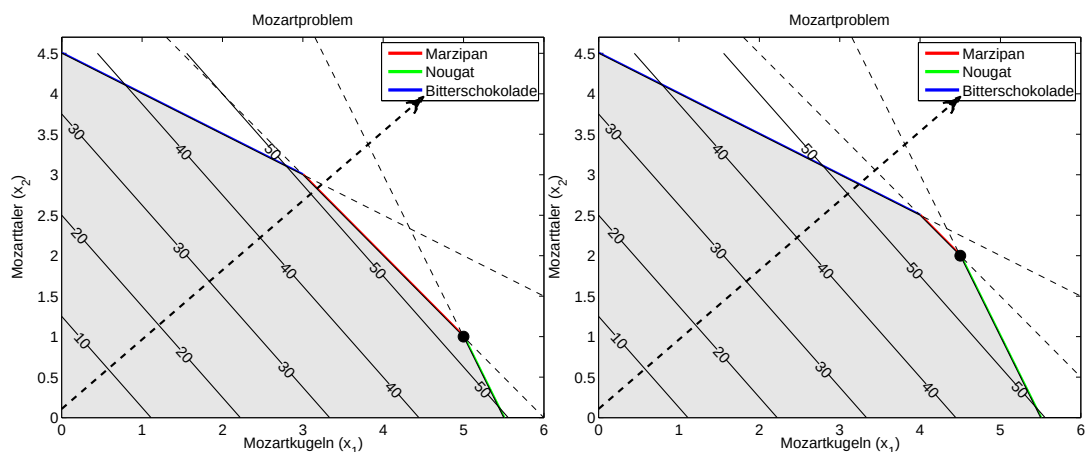


Abbildung 13.1: Optimallösung des nominalen Mozartproblem (links) und des gestörten Problems mit einem Marzipanbestand von 6.5 an Stelle von 6 Einheiten.

Wir halten also fest, dass uns der Lagrange-Multiplikator $\lambda_{\text{ineq}} = (7, 1, 0)$ verrät, dass wir bei einer Erhöhung des Lagerbestandes von Marzipan pro Einheit sieben Einheiten mehr Umsatz machen könnten, wenn wir die Produktionsmengen optimal an die neue Situation anpassen. Bei Nougat könnten wir immerhin noch einen zusätzlichen Umsatz von *einer* Einheit pro zusätzlicher Einheit Nougat machen. Bei einer Reduktion der Lagerbestände (also Δb_{ineq} mit negativem Vorzeichen) verringert sich der optimale Umsatz entsprechend.³² Die Frage, warum eine kleine Änderung des Bestandes an Bitterschokolade sich nicht auf den möglichen Umsatz auswirkt, sollte man mit Hilfe von Abb. 13.1 leicht selbst beantworten können.

³²Diese Aussagen gelten, wie schon mehrfach betont, nur für hinreichend kleinen Änderungen an den Lagerbeständen. An Abb. 13.1 sieht man das auch nochmal recht gut: Wichtig ist, dass die optimale Lösung unter den veränderten Bedingungen nicht auf einen anderen Teil des Randes springt, da sich in dem Moment die Empfindlichkeiten ändern.

In der Regel erreicht man eine Erhöhung der Lagerbestände nicht umsonst, sondern man kann sich überlegen, zusätzliche Ressourcen am Markt zu kaufen. Der Vektor der Multiplikatoren $\lambda_{\text{ineq}} = (7, 1, 0)$ gibt dabei an, dass der Zukauf von Marzipan nur dann lohnt, wenn der Preis dafür bei unter sieben Geldeinheiten pro Einheit Marzipan liegt. Ist das der Fall, dann sollte man Marzipan zukaufen, und zwar zunächst soviel, bis die angepasste Optimallösung die nächste Ecke erreicht (Basiswechsel). Dort kann man die Multiplikatoren erneut auswerten usw. Aufgrund dieser wirtschaftlichen Interpretation der Lagrange-Multiplikatoren nennt man sie auch **Schattenpreise**.

Ende der Woche 10

§ 14 PROBLEME DES KOSTENMINIMALEN TRANSPORTS AUF GRAPHEN

Wir beginnen wieder mit einem Beispiel.³³

Beispiel 14.1 (Transportproblem, kostenminimaler Fluss). *Ein Unternehmen verfügt über das in Abb. 14.1 dargestellte **Transportnetzwerk**. Dabei entsprechen die **Knoten** 1–3 den Produktionsstätten, 4–5 den Zwischenlagern und 6–9 den Verkaufsstätten. Die **Kanten** zwischen den Knoten entsprechen den möglichen Transportwegen. Die Kapazitäten der Produktionsstätten sowie die Bedarfe der Verkaufsstätten (für einen festen Zeitraum, z. B. einen Monat) sind bekannt.*

Es geht darum, den Transport von den Produktionsstätten über die Zwischenlager zu den Verkaufsstätten zu planen. Jeder Transportweg (Kante) ist dabei mit Transportkosten belegt, die proportional zu der Warenmenge sind, die über diesen Weg transportiert wird. Außerdem wird es üblicherweise Kapazitätsbeschränkungen auf jedem Transportweg geben. Gesucht ist nun eine Verteilung der Transportströme über die Kanten, die (unter Beachtung aller Restriktionen) die Gesamttransportkosten minimiert.

Eine solche Aufgabe nennt man auch ein Problem des **kostenminimalen Flusses**. Dies ist ein Beispiel für eine Optimierungsaufgabe auf Graphen. Andere Beispiele sind Maximalflussprobleme, Kürzeste-Wege-Probleme und Zuordnungsprobleme, siehe § 16. Um diese Aufgaben als (lineare) Optimierungsaufgabe zu formulieren und anschließend mit geeigneten Verfahren zu lösen, ist es sinnvoll, zunächst einige Begriffe einzuführen.

Definition 14.2 (Graphen).

- (i) Ein **gerichteter Graph** (engl.: directed graph, kurz: **Digraph**) besteht aus einer Menge V von Knoten (engl.: vertices) und einer Menge E von gerichteten Kanten (engl.: edges) zwischen Knoten.³⁴
- (ii) Eine gerichtete Kante $e \in E$ wird eindeutig durch ihren **Anfangsknoten** $x \in V$ und ihren **Endknoten** $y \in V$ gekennzeichnet, wobei $x \neq y$ ist. Wir schreiben auch $e = (x, y)$ für die Kante von x nach y .

³³Dies ist eine Weiterentwicklung unseres allerersten Beispiels einer Optimierungsaufgabe, Beispiel 1.1.

³⁴Manchmal werden Knoten auch mit *nodes* und Kanten mit *arcs* bezeichnet.

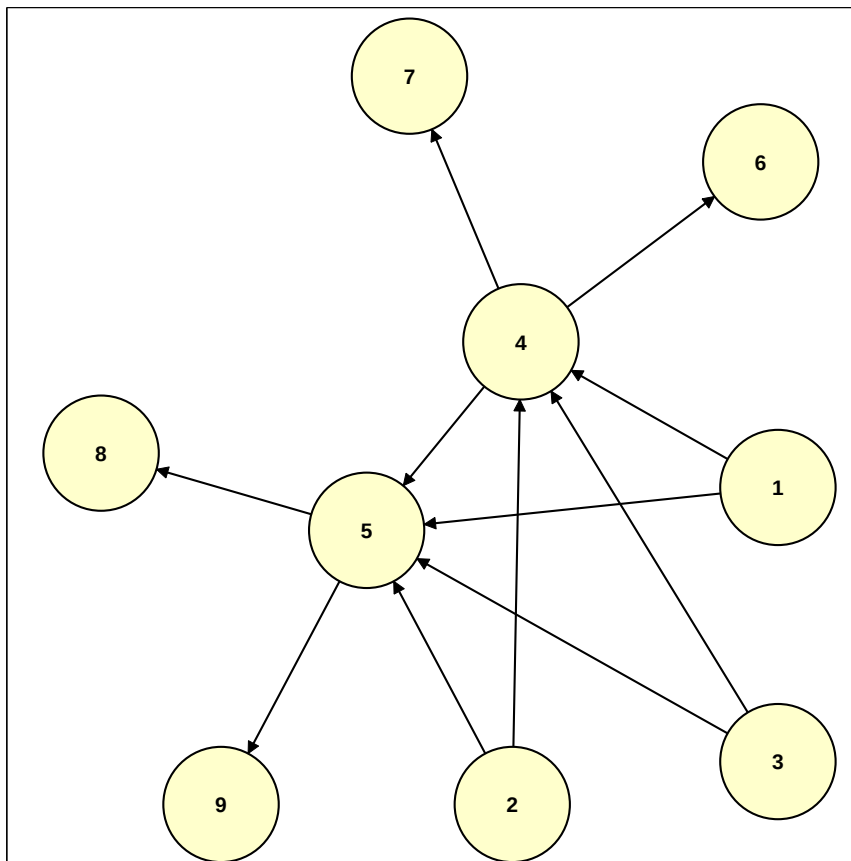


Abbildung 14.1.: Transportnetzwerk eines Unternehmens (siehe [Beispiel 14.1](#)) mit Produktionsstätten 1–3, Zwischenlagern 4–5 und Verkaufsstätten 6–9.

- (iii) Wir behandeln nur sogenannte **einfache gerichtete Graphen** (engl.: simple directed graphs), bei denen keine Schleifen (engl.: loops) und keine mehrfachen Kanten zwischen zwei Knoten auftreten.

Mit dieser Definition können wir unseren Graphen aus [Beispiel 14.1](#) beschreiben durch

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$E = \{(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5), (4, 6), (4, 7), (5, 8), (5, 9)\}.$$

Da wir es in dieser Vorlesung praktisch ausschließlich mit einfachen gerichteten Graphen zu tun haben werden, sprechen wir auch einfach von Graphen.

Die Auflistung von Knoten und Kanten hilft uns alleine noch nicht weiter bei der Formulierung von Transportaufgaben als Optimierungsprobleme. Dafür benötigen wir eine Beschreibung von Graphen, mit der man rechnen kann. Dies wird durch Adjazenz- und Inzidenzmatrizen erreicht.

Definition 14.3 (Adjazenzmatrix). Es sei ein Graph mit Knotenmenge V und Kantenmenge E gegeben. Die Knotenmenge habe die Mächtigkeit $|V| = m$, und die Knoten seien von 1 bis

m durchnummeriert, also ist $V = \{v_1, v_2, \dots, v_m\}$. Die zugehörige **Adjazenzmatrix** (engl.: adjacency matrix) $A^{ad} = (a_{ij}^{ad})$ hat die Dimension $m \times m$ und ist wie folgt definiert:

$$a_{ij}^{ad} = \begin{cases} 1, & \text{falls es die Kante von } v_i \text{ nach } v_j \text{ in der Kantenmenge } E \text{ gibt,} \\ 0, & \text{andernfalls.} \end{cases}$$

Dabei steht also die Nummer i der Zeile für den Anfangs- und die Nummer j der Spalte für den Endknoten. Im [Beispiel 14.1](#) haben wir folgende Adjazenzmatrix:

$$A^{ad} = \begin{bmatrix} \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \in \mathbb{R}^{9 \times 9}.$$

Diese beschreibt den Graphen eindeutig. Der besseren Lesbarkeit wegen sind Nullen hier durch $\cdot$ ersetzt worden.

Mit Adjazenzmatrizen kann man sehr gut *knotenorientierte* Aufgabenstellungen bearbeiten, z. B. die Frage, wieviele verschiedene Wege (Aneinanderreihung von Kanten) einer bestimmten Länge von v_i nach v_j es gibt. Die meisten wirtschaftswissenschaftlichen Aufgabenstellungen auf Graphen sind aber *kantenorientiert*. Bei unserem [Beispiel 14.1](#) ist z. B. eine Belegung der Kanten (Transportwege) gesucht. Bei solchen Fragestellungen hilft uns die Adjazenzmatrix nicht weiter.

Definition 14.4 (Inzidenzmatrix). Es sei ein Graph mit Knotenmenge V und Kantenmenge E gegeben. Die Knotenmenge habe die Mächtigkeit $|V| = m$, und die Knoten seien von 1 bis m durchnummeriert, also ist $V = \{v_1, v_2, \dots, v_m\}$. Die Kantenmenge habe die Mächtigkeit $|E| = n$, und die Kanten seien von 1 bis n durchnummeriert, also ist $E = \{e_1, e_2, \dots, e_n\}$. Die zugehörige **(Knoten-Kanten-)Inzidenzmatrix** (engl.: (node-edge-)incidence matrix) $A^{in} = (a_{ij}^{in})$ hat die Dimension $m \times n$ und ist wie folgt definiert:

$$a_{ij}^{in} = \begin{cases} -1, & \text{falls die Kante } e_j \text{ im Knoten } v_i \text{ startet,} \\ 1, & \text{falls die Kante } e_j \text{ im Knoten } v_i \text{ endet,} \\ 0, & \text{sonst.} \end{cases}$$

Dabei steht also die Nummer i der Zeile für die Knoten und die Nummer j der Spalte für die Kanten. Im [Beispiel 14.1](#) haben wir folgende Inzidenzmatrix, wenn wir die Kanten wie in

Abb. 14.2 nummerieren:

$$A^{\text{in}} = \begin{bmatrix} -1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & -1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & 1 & \cdot & -1 & -1 & -1 & \cdot & \cdot \\ \cdot & 1 & \cdot & 1 & \cdot & 1 & 1 & \cdot & \cdot & -1 & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix} \in \mathbb{R}^{9 \times 11}.$$

Auch die Inzidenzmatrix beschreibt den Graphen eindeutig. Der besseren Lesbarkeit wegen sind auch hier Nullen wieder durch $\cdot$ ersetzt worden. Da jede Kante (Spalte) genau einen Anfang (-1) und ein Ende ($+1$) hat, sind alle Spaltensummen gleich null.³⁵ Dies wird später noch wichtig werden.

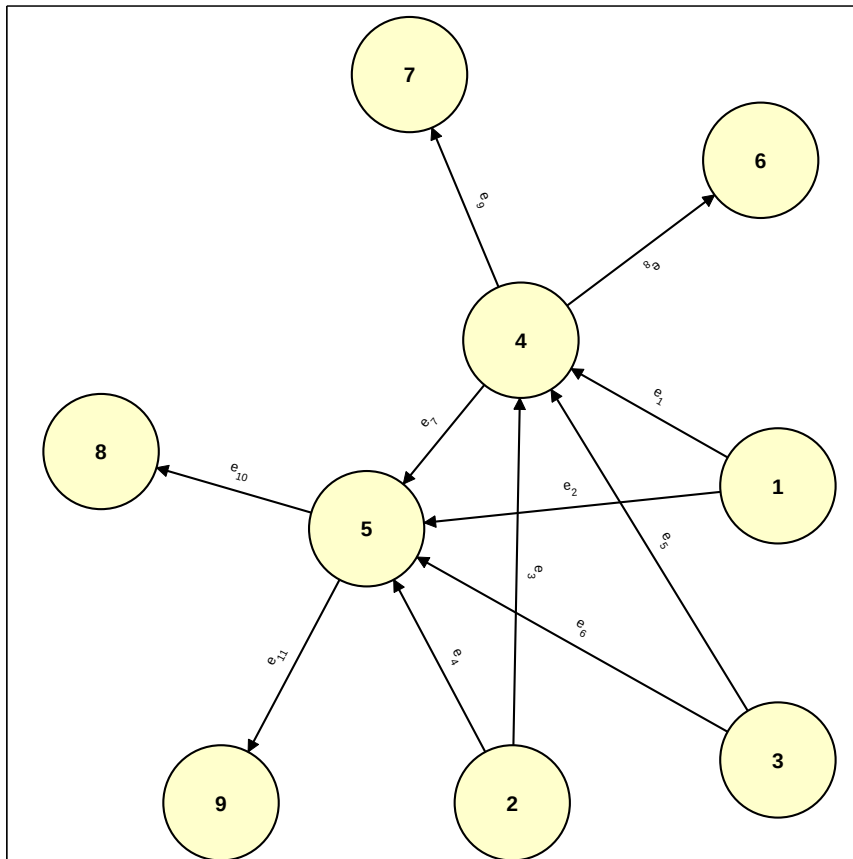


Abbildung 14.2.: Transportnetzwerk aus siehe [Beispiel 14.1](#) mit einer Nummerierung der Kanten $e_1, \dots, e_{11}$.

Achtung: Manchmal werden Inzidenzmatrizen transponiert zu unserer Definition definiert (Kanten entsprechen dann Zeilen, Knoten entsprechen Spalten) und / oder mit der Bedeutung

³⁵Mit anderen Worten, es gilt $e^T A^{\text{in}} = 0$ (Zeilenvektor), wobei $e = (1, \dots, 1)^T \in \mathbb{R}^m$ ist.

von ± 1 vertauscht. Für uns ist aber die hier gegebene kantenorientierte [Definition 14.4](#) praktisch.

Wir wollen nun sehen, wie man mit Inzidenzmatrizen rechnen kann. Diese bezeichnen ab sofort einfach mit A . Wir untersuchen an unserem Beispiel mit dem Graph aus [Abb. 14.2](#) die Bedeutung des Matrix-Vektor-Produkts Ax mit einem Vektor $x \in \mathbb{R}^{11}$. Der Vektor x gibt eine Belegung der Kanten (Transportwege) an, die wir als Warenströme über diese Kanten interpretieren. Zur Illustration sehen wir uns als Beispiel nur die Zeile 5 der Matrix an, also

$$(0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad -1 \quad -1) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{11} \end{pmatrix} = x_2 + x_4 + x_6 + x_7 - x_{10} - x_{11}.$$

Bei dem Ergebnis spielen also nur die Warenströme der an den Knoten 5 (eines der Zwischenlager) angrenzenden Kanten (mit den Nummern 2, 4, 6, 7, 10, 11) eine Rolle. Dabei werden die über die Kanten 2, 4, 6 und 7 *eingehenden* Warenströme positiv gezählt und die über die Kanten 10 und 11 *ausgehenden* negativ.

Das Matrix-Vektor-Produkt Ax gibt also offenbar den Vektor der **Knotenbilanzen** an, der sich bei der Belegung der Kanten (Transportwege) mit den Transportmengen ergibt, die im Vektor x eingetragen sind. Mit dieser Erkenntnis können wir unser [Beispiel 14.1](#) des **kostenminimalen Flusses** nun als (lineare) Optimierungsaufgabe formulieren: Die zu minimierenden Gesamtkosten aller Transportströme setzen sich als Summe der Kosten über die einzelnen Kanten zusammen:

$$\sum_{j=1}^n c_j x_j = c^T x.$$

Dabei sind c_j die gegebenen Transportkosten pro Einheit über die Kante j . Weiter sind die Bilanzen b_i aller Knoten $i = 1, \dots, m$ gegeben. Man unterscheidet

- **Bedarfsknoten** oder **Senken** ($b_i > 0$, engl.: *demand nodes*),
- **Angebotsknoten** oder **Quellen** ($b_i < 0$, engl.: *supply nodes*) und
- **Durchfluss-** oder **Umladeknoten** ($b_i = 0$, engl.: *transshipment nodes*).

Die Erfüllung aller Knotenbilanzen wird durch das lineare Gleichungssystem

$$Ax = b$$

ausgedrückt. Dieses heißen auch **Flusserhaltungsgleichungen**, da sie insbesondere bei Durchflussknoten sicherstellen, dass alles, was hinfließt, auch wieder hinausfließt. Zusätzlich ist zu beachten, dass die Transportmengen über die Kanten nicht negativ sein dürfen; dies würde einer Umkehrung der Flussrichtung entsprechen. Schließlich sind Kapazitätsbeschränkungen der einzelnen Transportwege (Kanten) einzuhalten:

$$0 \leq x_i \leq u_i \quad \text{für alle } i = 1, \dots, n.$$

Kompakt kann man eine Optimierungsaufgabe des kostenminimalen Flusses also wie folgt schreiben:

$$\begin{aligned} &\text{Minimiere} && c^T x, \quad x \in \mathbb{R}^n \\ &\text{unter} && A x = b \\ &\text{sowie} && 0 \leq x \leq u. \end{aligned} \tag{14.1}$$

Dabei gelten folgende Bezeichnungen:

A = Inzidenzmatrix in $\mathbb{R}^{m \times n}$, c = Vektor der Kantenkosten in $\mathbb{R}^n$,
 b = Vektor der Knotenbilanzen in $\mathbb{R}^m$, u = Vektor der Kantenkapazitäten in $\mathbb{R}^n$,
 m = Anzahl der Knoten, n = Anzahl der Kanten.

Noch ein Wort zu den Begrifflichkeiten:

Definition 14.5 (Transportnetzwerk, Flussnetzwerk, (zulässiger) Flussvektor). *Ein (einfacher gerichteter) Graph mit Inzidenzmatrix A , Kantenkapazitäten u und Knotenbilanzen b wird auch als **Transportnetzwerk** oder **Flussnetzwerk** bezeichnet. Eine Belegung x der Kanten, die die **Erhaltungsbedingung** $Ax = b$ erfüllt, heißt ein **Fluss** oder **Flussvektor** auf diesem Netzwerk. Ein **zulässiger Fluss** erfüllt zusätzlich die Kapazitätsbeschränkungen $0 \leq x \leq u$.*

Bei (14.1) handelt es sich um eine lineare Optimierungsaufgabe, vgl. (11.1). Wir können sie also etwa mit dem (dualen) Simplex-Verfahren in `linprog` lösen. Zunächst setzen wir die Problemdata:

Matlab/Daten_Transpo
rtproblem.m

```
% Aufstellen der Inzidenzmatrix
% fuer ein Netzwerk mit m = 9 Knoten und n = 11 Kanten
A = ...
[ -1 -1  0  0  0  0  0  0  0  0  0; ...
  0  0 -1 -1  0  0  0  0  0  0  0; ...
  0  0  0  0 -1 -1  0  0  0  0  0; ...
  1  0  1  0  1  0 -1 -1 -1  0  0; ...
  0  1  0  1  0  1  1  0  0 -1 -1; ...
  0  0  0  0  0  0  0  0  1  0  0; ...
  0  0  0  0  0  0  0  0  0  1  0; ...
  0  0  0  0  0  0  0  0  0  0  1];

m = size(A,1);
n = size(A,2);

% Setzen der Knotenbilanzen
b = [-100; -200; -300; 0; 0; 150; 150; 150; 150];

% Setzen der Kapazitaetsbeschraenkungen auf den Kanten
l = zeros(n,1);
u = 300 * ones(n,1);

% Setzen des Kostenvektors auf den Kanten
c = [0.8; 2.0; 2.5; 1.0; 1.2; 2.0; 1.0; 1.0; 1.0; 1.0; 1.0];
```

Mit folgendem Aufruf stellen wir den Graphen wie in Abb. 14.1 zur besseren Anschauung dar:³⁶

Matlab/
Transportproblem.m

```
% Darstellung des Graphen
h = Graph(A,h);
```

Wie bereits in § 11 wählen wir durch Setzen der entsprechenden Optionen das Simplex-Verfahren in linprog aus

Matlab/
Transportproblem.m

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('linprog','Algorithm','dual-simplex','
    Diagnostics','on','Display','iter');
```

und rufen linprog auf:

Matlab/
Transportproblem.m

```
% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda] = linprog(c,[],[],A,b,l,u,options
    );
```

Für die Beispieldaten

$$\begin{aligned} b &= (-100, -200, -300, 0, 0, 150, 150, 150, 150)^T, \\ c &= (0.8, 2.0, 2.5, 1.0, 1.2, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0)^T, \\ u &= (300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300)^T, \end{aligned}$$

die auch in Abb. 14.3 dargestellt sind, erhalten wir die Optimallösung³⁷

$$x^* = (100, 0, 0, 200, 200, 100, 0, 150, 150, 150, 150)^T.$$

Diese ist in Abb. 14.4 gezeigt. Die zugehörigen Transportkosten betragen $c^T x^* = 1320$. Der Vollständigkeit halber geben wir auch hier wieder die Ausgabe von linprog an.

```
-----
Diagnostic Information

Number of variables: 11

Number of linear inequality constraints:    0
Number of linear equality constraints:      9
Number of lower bound constraints:         11
Number of upper bound constraints:         11
```

³⁶Die erforderlichen MATLAB-Routinen sind auf der Homepage zur Vorlesung erhältlich, und die Handhabung wird in der Übung besprochen.

³⁷**Beachte:** Da die Matrix A der Nebenbedingung $Ax = b$ hier $m = 9$ Zeilen und $n = 11$ Spalten besitzt, hat die Nichtbasis im Simplex-Verfahren die Mächtigkeit $|N| = 2$. Jede Ecke und damit auch die von linprog gefundene Lösung x^* besitzt damit mindestens zwei Nulleinträge, d. h. Kanten, über die nichts transportiert wird.

Algorithm selected
dual-simplex

End diagnostic information

LP preprocessing removed 0 inequalities, 5 equalities,
5 variables, and 10 non-zero elements.

Iter	Time	Fval	Primal Infeas	Dual Infeas
0	0.002	8.400000e+02	5.099020e+02	0.000000e+00
3	0.002	1.320000e+03	0.000000e+00	0.000000e+00

Optimal solution found.

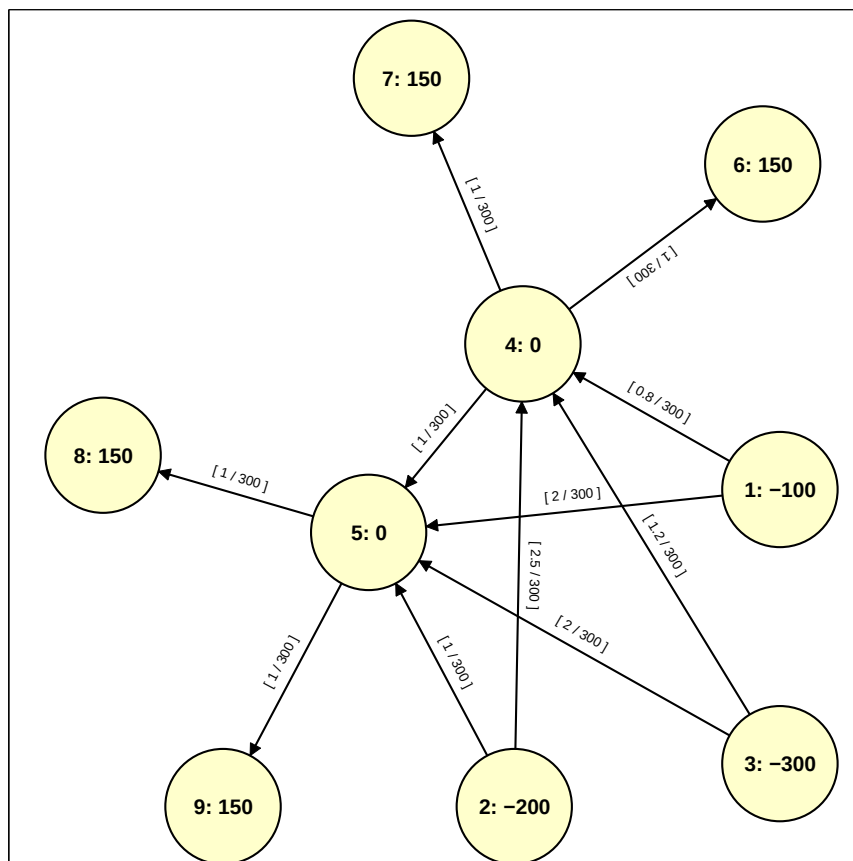


Abbildung 14.3.: Transportnetzwerk aus Abb. 14.1 mit Knotennummern: Knotenbilanzen b sowie $[\text{Kantenkosten } c / \text{Kantenkapazitäten } u]$.

Die interaktive Arbeitsweise mit MATLAB erlaubt es, schnell die Auswirkungen von Änderungen in den Daten auf den optimalen Transportplan zu studieren. Steigen etwa die Transportkosten auf der Kante e_6 (zwischen Produktionsstätte 3 und Zwischenlager 5) von 2.0 auf 3.0 pro Einheit des transportierten Gutes:

$$c(6) = 3$$

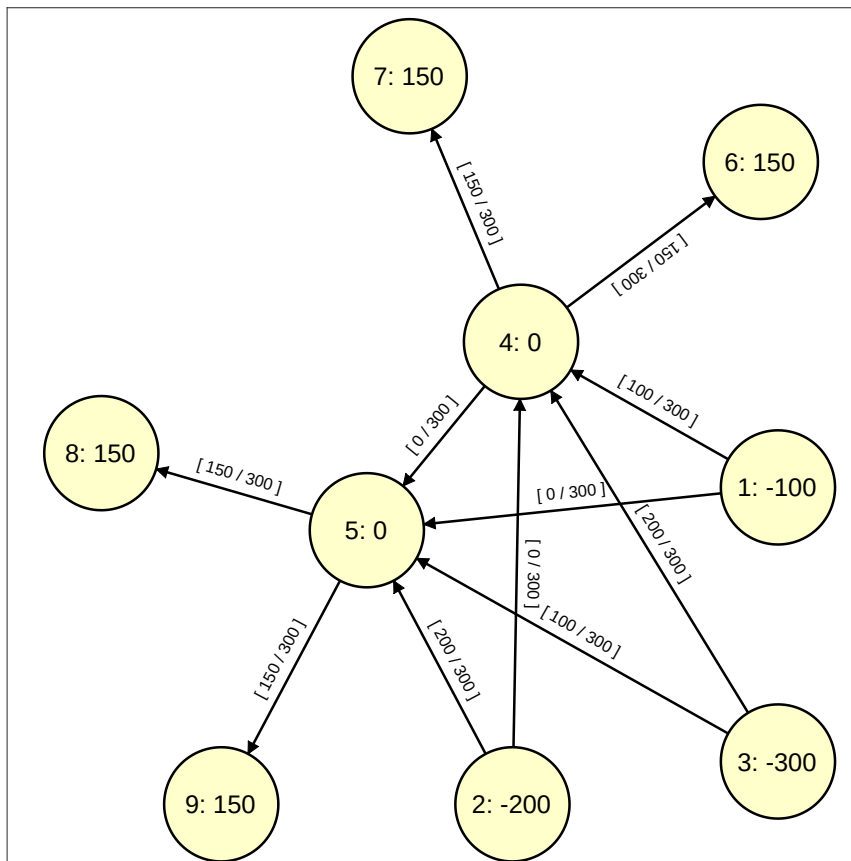


Abbildung 14.4.: Lösung der Aufgabe des kostenminimalen Flusses mit den im Text angegebenen Daten. Zu den Kanten sind jeweils [die optimalen Transportströme x^* / Kantenkapazitäten u] mit angegeben.

so ändert sich die optimale Lösung in

$$x^* = (100, 0, 0, 200, 300, 0, 100, 150, 150, 150)^T.$$

Es fließt jetzt also auch ein Transportstrom vom Zwischenlager 4 in Zwischenlager 5.

Bemerkung 14.6 (zu Aufgaben des kostenminimalen Flusses).

- (i) Wie oben bereits beobachtet, hat jede Inzidenzmatrix A eines (gerichteten) Graphen die Eigenschaften, dass alle ihre Spaltensummen null sind. Was bedeutet das für eine Aufgabe des kostenminimalen Flusses (14.1)? Bilden wir die Summe über alle Zeilen der Bilanzgleichung $Ax = b$ durch Multiplikation von links mit dem Vektor $e = (1, \dots, 1)^T \in \mathbb{R}^m$:

$$\underbrace{e^T A}_{=0} x = e^T b.$$

Die Bilanzgleichung ist also überhaupt nur dann durch irgendeinen Flussvektor x erfüllbar, wenn auch die Summe über die Einträge der rechten Seite null ist: $e^T b = \sum_{i=1}^m b_i = 0$.³⁸ Mit anderen Worten müssen sich die Knotenbilanzen (alle Angebote und Nachfragen)

³⁸Ein Transportnetzwerk besitzt also eine globale Erhaltungseigenschaft.

gerade gegenseitig aufheben. Für unsere Beispieldaten war dies der Fall, wie man leicht per Hand oder `sum(b)` in MATLAB nachrechnet.

- (ii) Sollte in einem Transportnetzwerk $e^T b < 0$ gelten, dann besitzt die Aufgabe des kostenminimalen Flusses keinen zulässigen Flussvektor (die zulässige Menge ist leer), also auch keine Lösung. Da dies auf ein **Überangebot** des zu transportierenden Gutes zurückzuführen ist, kann man sich wie folgt behelfen: Es wird ein zusätzlicher Knoten eingeführt, der einem künstlichen Abnehmer entspricht, dessen Bedarf gerade das Überangebot kompensiert. Diese künstliche Senke wird mit allen Angebotsknoten durch neue Kanten verbunden, und diese neuen Kanten werden beispielsweise mit null Kosten und unendlicher Kapazität ausgestattet.³⁹

Im Fall $e^T b > 0$ liegt dagegen ein **Mangel** an dem zu transportierenden Gut vor. Auch dann kann durch einen zusätzlichen Knoten mit einem Angebot in Höhe der Fehlmenge weitergearbeitet werden. Dieser Knoten wird mit allen Bedarfsknoten durch neue Kanten verbunden, und diese neuen Kanten werden wie oben mit null Kosten oder aber den realen Fehlmengenkosten und z. B. unendlicher Kapazität ausgestattet.⁴⁰

- (iii) Das Simplex-Verfahren, wie wir es in § 12 kennengelernt haben, ist nicht die effizienteste Lösungsmöglichkeit für Aufgaben kostenminimaler Flüsse auf Netzwerken. Es gibt dafür eine spezielle Variante, das **Netzwerk-Simplex-Verfahren**, siehe etwa Gerds, Lempio, 2011, Abschnitt 4.2. Diese nutzt aus, dass die Matrix A eine Inzidenzmatrix ist, die nur aus Einträgen $\{0, \pm 1\}$ besteht.⁴¹ Die beiden aufwändigsten Schritte, die Berechnung der reduzierten Kosten (12.4) und der Richtung d_B in (12.7), erfordern dabei statt der Lösung von linearen Gleichungssystemen nur Additionen und Subtraktionen von Vektoren. Das Netzwerk-Simplex-Verfahren ist jedoch in der MATLAB-Optimierungs-Toolbox nicht implementiert. Für nicht allzu große Aufgaben können wir aber auch einfach mit dem normalen Simplex-Verfahren in `linprog` arbeiten.

Ende der Woche 11

§ 15 GANZZAHLIGE LÖSUNGEN

Im obigen Beispiel 14.1 war der optimale Fluss x^* über jede Kante ganzzahlig, also $x^* \in \mathbb{Z}^n$, siehe etwa Abb. 14.4. Dies ist bei vielen Aufgabenstellungen auf Transportnetzwerken erwünscht oder sogar erforderlich, weil sich die verwendeten Transporteinheiten (Paletten, LKW etc.) oft nicht teilen lassen. Es stellt sich die Frage, ob die Ganzzahligkeit der Lösung Zufall ist oder Methode hat. In numerischen Experimenten mit veränderten Problem

³⁹Null Kosten entsprechen der Annahme, dass die über diese Kanten fließenden Transportmengen in Wirklichkeit kostenfrei am jeweiligen Produktionsort verbleiben. Sollten dafür Lagerkosten entstehen, so werden diese statt einer Null als Kantenkosten angesetzt. Dasselbe gilt, wenn die Überschussmengen tatsächlich abtransportiert werden, etwa in ein Lager. Statt unendlicher Kapazität kann natürlich auch eine Obergrenze angesetzt werden.

⁴⁰Dies bedeutet, dass wir die ursprüngliche Aufgabenstellung erweitern: Wir wollen die vorhandenen Angebotsmengen ausschöpfen, um den Bedarf so gut wie möglich zu decken, und zwar noch immer zu minimalen Transportkosten. Sollten Fehlmengenkosten entstehen, etwa durch Zukauf der Fehlmengen, so werden die Kosten dafür statt der Null als Kantenkosten bei den neuen Kanten angesetzt. Siehe Übung.

⁴¹Zu beachten ist auch, dass Inzidenzmatrizen wegen $e^T A = 0$ nicht vollen Zeilenrang haben.

zeigt sich, dass der vom Simplex-Verfahren berechnete optimale Fluss x^* in allen Fällen ganzzahlig ist. Voraussetzung ist lediglich, dass auch die Knotenbilanzen $b \in \mathbb{Z}^m$ und die Kantenkapazitäten $u \in \mathbb{Z}^n$ jeweils ganzzahlig sind. Dem wollen wir nun nachgehen.

Da das Simplex-Verfahren auf den Ecken der zulässigen Menge

$$\{x \in \mathbb{R}^n : Ax = b, 0 \leq x \leq u\} \quad (15.1)$$

arbeitet und eine davon als Lösung zurückgibt, geht es um die Frage, wann die Ecken dieses Polyeders *alle* ganzzahlig sind (genauer: alle nur ganzzahlige Koordinaten haben). Beim Mozartproblem war dies zum Beispiel nicht der Fall, siehe [Abb. 12.4](#).

Definition 15.1 (total unimodulare Matrix). *Eine Matrix A der Größe $m \times n$ heißt **total unimodular**, wenn für jede quadratische Teilmatrix $\hat{A}$ von A (beliebiger Größe zwischen 1×1 und $m \times m$ bzw. $n \times n$) gilt:*

$$\det \hat{A} \in \{0, 1, -1\}.$$

Das heißt natürlich auch, dass die Matrix A überhaupt nur aus den Einträgen $\{0, 1, -1\}$ bestehen kann.

Die Bedeutung total unimodularer Matrizen ergibt sich aus folgendem Satz:

Satz 15.2 (Bedeutung total unimodularer Matrizen). *Es sei A eine Matrix der Größe $m \times n$. Folgende Aussagen sind äquivalent:*

- (i) *A ist total unimodular.*
- (ii) *Für jeden Vektor $b \in \mathbb{Z}^m$ besitzt das Polyeder*

$$\{x \in \mathbb{R}^n : Ax \geq b, x \geq 0\}$$

nur ganzzahlige Ecken.

- (iii) *Für jeden Vektor $b \in \mathbb{Z}^m$ und jeden Vektor $u \in \mathbb{Z}^n$ besitzt das Polyeder*

$$\{x \in \mathbb{R}^n : Ax = b, 0 \leq x \leq u\}$$

nur ganzzahlige Ecken.

Diese Liste äquivalenter Aussagen könnte man noch weiterführen: Für jede einzelne Zeile von A kann man z. B. $\leq$, $\geq$ oder $=$ schreiben. Neben oberen Schranken dürfen auch ganzzahlige untere Schranken an x vorkommen.

Man kann zeigen, dass Inzidenzmatrizen A für (einfache, gerichtete) Graphen total unimodular sind. Dies erklärt, warum wir in numerischen Experimenten zu kostenminimalen Flussproblemen in Transportnetzwerken immer ganzzahlige Lösungen x^* erhalten, solange nur die Knotenbilanzen $b \in \mathbb{Z}^m$ und die Kantenkapazitäten $u \in \mathbb{Z}^n$ jeweils ganzzahlig sind: Alle Ecken der zulässigen Menge sind ganzzahlig und damit auch die Lösung, die das Simplex-Verfahren zurückgibt. Auf den Kostenvektor c kommt es dabei nicht an.

§ 16 WEITERE OPTIMIERUNGSAUFGABEN AUF GRAPHEN

Es gibt sehr viele Beispiele für Optimierungsaufgaben auf Graphen und Netzwerken. Diese sind überwiegend, aber nicht ausschließlich wirtschaftswissenschaftlichen Ursprungs.

Beispiel 16.1 (Kürzeste-Wege-Problem). Beim **Kürzeste-Wege-Problem** geht es darum, innerhalb eines (einfachen, gerichteten) Graphen den „kürzesten“ Weg zwischen zwei gegebenen Knoten v_A und v_B zu finden. Solche Aufgaben werden z. B. von Navigationssystemen, aber auch beim Routing von Datenverkehr im Internet gelöst. Die „Länge“ eines Weges ist nicht unbedingt wörtlich zu nehmen, sie kann auch „Zeitdauer“ oder allgemein Aufwand bedeuten.

Ein Kürzeste-Wege-Problem kann als Aufgabe eines kostenminimalen Flusses modelliert werden. Die Idee dazu ist, einen „Spielstein“ entlang der Kanten durch das Netzwerk zu bewegen. Da der Spielstein am Startknoten v_A eingesetzt und am Zielknoten v_B wieder entnommen wird, ist der Vektor der Knotenbilanzen

$$b = (0, \dots, 0, \underbrace{-1}_{\text{Position A}}, 0, \dots, 0, \underbrace{1}_{\text{Position B}}, 0, \dots, 0)^T.$$

Die Gleichungsnebenbedingung

$$Ax = b$$

stellt wieder die Einhaltung der Knotenbilanzen sicher.

Jede Kante im Netzwerk soll mit dem Wert $x_i = 1$ (benutzt) oder $x_i = 0$ (nicht benutzt) belegt werden. Das heißt, unser Transportgut (der Spielstein) wird nicht zerlegt, sondern passiert eine Kante entweder als Ganzes oder gar nicht. Die Bedingung $x_i \in \{0, 1\}$ für alle Kanten $i = 1, \dots, n$ (kurz: $x \in \{0, 1\}^n$) lässt sich garantieren, wenn man die Kantenkapazitäten einfach auf alle auf 1 setzt und wie gehabt fordert:

$$0 \leq x \leq u = (1, 1, \dots, 1)^T \in \mathbb{R}^n.$$

Da die Matrix A total unimodular und der Bilanzvektor b sowie die Vektoren der oberen und unteren Grenze ganzzahlig sind, sind nach [Satz 15.2](#) alle Ecken der zulässigen Menge ganzzahlig, und wegen der gerade genannten Ungleichung liegen alle Komponenten zusätzlich zwischen 0 und 1, sodass nur 0 oder 1 selbst übrig bleiben.⁴²

Schließlich soll die die Gesamt„länge“ des Weges minimiert werden, die sich wieder als

$$\sum_{j=1}^n c_j x_j = c^T x$$

ergibt, also aus der Summe der Längen der benutzten Kanten.

⁴²Wir können sogar noch die obere Schranke u bei der Modellierung weglassen, denn: Da das Gesamtangebot des Transportgutes bei 1 liegt (ein Spielstein) und über eine beliebige Kante natürlich nicht mehr als das gesamte Gut fließen kann, ist die obere Schranke überflüssig.

Ein Kürzeste-Wege-Problem ist also eine Aufgabe des kostenminimalen Flusses (14.1) mit einem speziellen Bilanzvektor b (genau eine Quelle und eine Senke mit Angebot bzw. Bedarf 1) und Kantenkapazitäten von 1. Ein Beispiel mit einem zufällig erzeugten Graphen ist in Abb. 16.1 zu sehen.

Bemerkung 16.2 (zu Kürzeste-Wege-Problemen, Beispiel 16.1). *Wie schon bei Aufgaben mit kostenminimalen Flüssen ist das Simplex-Verfahren auch für Kürzeste-Wege-Aufgaben nicht das effizienteste Optimierungsverfahren. Es gibt schnellere Verfahren, die die spezielle Problemstruktur besser ausnutzen, als wir es hier tun. Dazu zählen*

- *der Algorithmus von Dijkstra (falls alle Kantenkosten $c_j > 0$ sind),*
- *der Algorithmus von Floyd-Warshall, der auch mit Kantenkosten $c_j \leq 0$ arbeitet (solange keine Kreise negativer Gesamtlänge auftreten).*

Wir verweisen z. B. auf *Gerdts, Lempio, 2011*, Abschnitt 4.4.

Eine weitere Aufgabenklasse, die auf kostenminimale Flüsse führt, sind **Evakuierungsprobleme**.

Beispiel 16.3 (Evakuierungsproblem). *Bei Evakuierungsaufgaben geht es darum, abzuschätzen, wie schnell etwa ein Gebäude bei Gefahr evakuiert werden kann.⁴³ Wir legen im Beispiel das in Abb. 16.2 gezeigte einstöckige Gebäude zu Grunde und modellieren die Aufgabe als ein Flussproblem. Die zu evakuierenden Personen stellen das zu transportierende Gut dar. Die Räume bilden einen Teil der Knoten, nämlich die Quellen („Angebotsknoten“) im Problem. Wir müssen davon ausgehen, dass jeder Raum voll belegt ist (mit der gezeigten Personenzahl). Weitere Knoten legen wir etwa an Kreuzungspunkte in den Gängen, dieses sind Durchflusssknoten mit Nullbilanzen.⁴⁴ Die beiden Notausgänge bilden schließlich die Senken.*

Entsprechend dem Grundriss werden nun die Knoten durch gerichtete Kanten verbunden. Jede Kante entspricht einem Abschnitt eines möglichen Fluchtweges und trägt zwei Informationen: die Kapazität sowie die Durchquerungszeit (z. B. in Vielfachen von 5 s) für diesen Abschnitt.⁴⁵

Die Zielfunktion soll den Wunsch berücksichtigen, möglichst viele Personen in möglichst kurzer Zeit zu evakuieren. Sie wird weiter unten aufgestellt.

Die direkte Verwendung des Graphen aus Abb. 16.3 als Transportnetzwerk in einer kostenminimalen Flussaufgabe ist aus zwei Gründen nicht möglich: Erstens haben wir zwei Senken (Notausgänge), deren „Bedarfe“ jedoch nicht einzeln festgelegt sind, sondern nur in Summe.⁴⁶ Dieses Problem lässt sich aber sehr leicht beheben, indem wir die beiden Senken zu einer

⁴³Die Ergebnisse solcher Betrachtungen gehen auch in die Planung und Abnahme öffentlicher Gebäude ein.

⁴⁴Dies bedeutet, dass sich dort keine Personen sammeln können. Das könnte natürlich auch anders modelliert werden.

⁴⁵Durch Einführung zusätzlicher Zwischenknoten und zugehöriger Kanten könnte man die Durchquerungszeiten immer auch auf eins bringen.

⁴⁶Es ist ja unerheblich, durch welchen Notausgang jemand aus dem Gebäude gelangt.

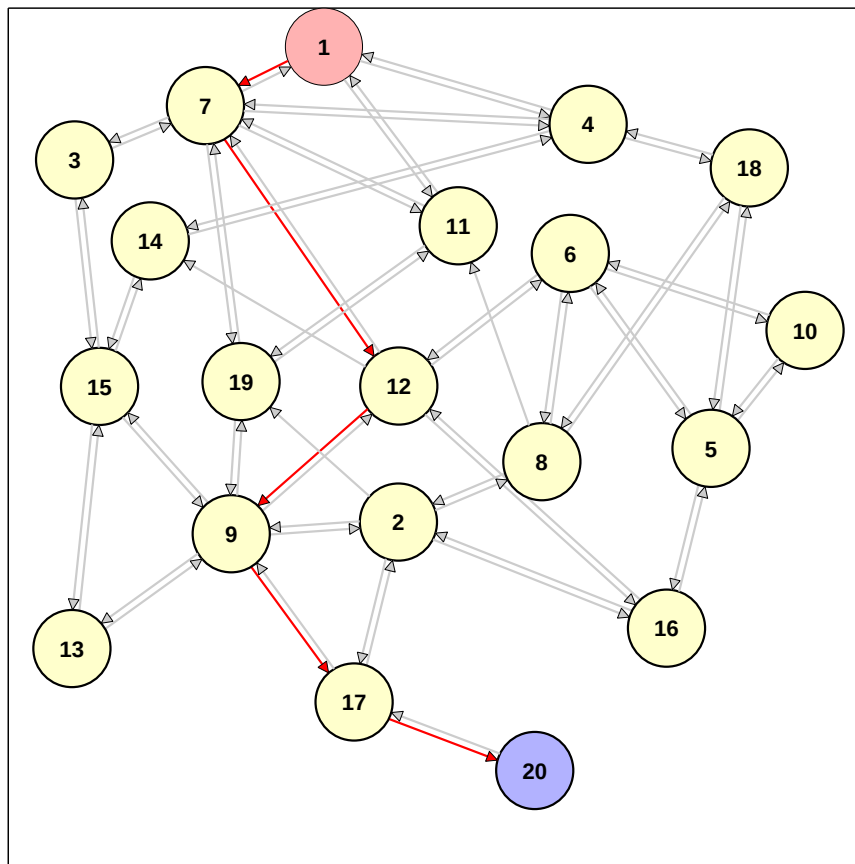


Abbildung 16.1.: Eine Lösung (1–7–12–9–17–20) des Problems des kürzesten Weges von Knoten 1 zu Knoten 20 in einem zufällig erzeugten Graphen. Alle Kanten haben hier die Länge (Kosten) $c_i = 1$. Doppelpfeile deuten an, dass zwischen zwei Knoten beide möglichen gegenläufigen Kanten existieren. Dies ist in diesem Beispiel (wie bei einem realen Straßennetzwerk) fast immer der Fall. Die rot markierte Lösung ist nicht eindeutig, es gibt mehrere kürzeste Weg von Knoten 1 nach 20.

zusammenfassen, deren „Bedarf“ der Gesamtanzahl der Personen im Gebäude entspricht, siehe Abb. 16.4.

Der zweite Grund ist viel gravierender: Ganz offensichtlich reicht ein Zeittakt nicht aus, um das gesamte Gebäude zu evakuieren.⁴⁷ Die Kapazitäten der Fluchtwege (Kanten) sind dafür einfach zu gering, und eine Formulierung auf Grundlage des Graphen aus Abb. 16.3 besäße schlicht und ergreifend keinen zulässigen Punkt.

Der Ausweg ist, dass wir zu einer *instationären* Problemformulierung über mehrere Zeitperioden hinweg kommen müssen.⁴⁸ Dazu werden die Knoten des stationären Graphen aus Abb. 16.4 kopiert, und zwar für jede Zeitschicht $t = 0, 1, \dots, T$ einmal. Die nun folgende

⁴⁷Allein das Verlassen des Zimmers mit den 40 Personen ist ja frühestens nach 21 Zeittakten beendet.

⁴⁸Unsere bisherigen Beispiele (Beispiel 14.1 und Beispiel 16.1) kamen immer mit einer einzigen Zeitperiode aus.

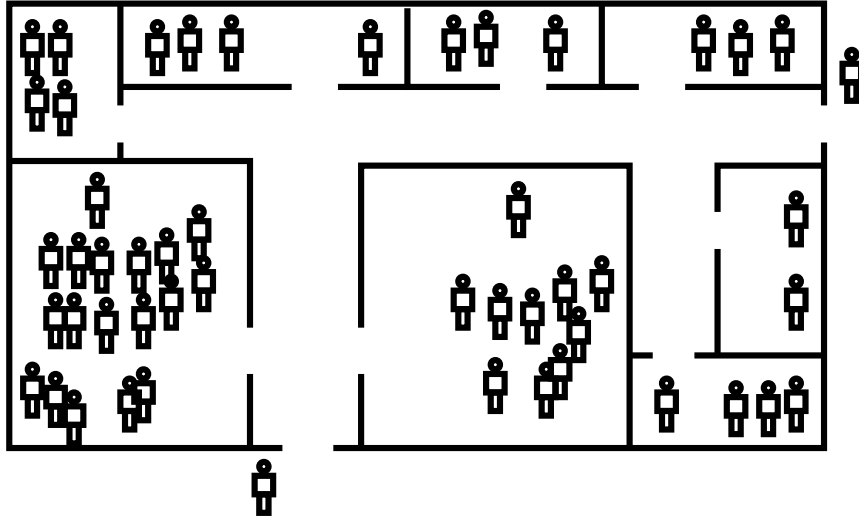


Abbildung 16.2.: Skizze eines (einstöckigen) Gebäudes. Gezeigt sind die Maximalbelegung der Räume, die Fluchtwege und die beiden Notausgänge.

Konstruktion ist in Abb. 16.5 dargestellt. Waren die Knoten des *stationären* Graphen mit

$$v_j, \quad j = 1, \dots, m_0$$

bezeichnet, so nennen wir die Knoten des *instationären* Graphen nun

$$v_j^t, \quad j = 1, \dots, m_0, \quad t = 0, \dots, T.$$

Die Kanten des instationären Graphen werden wie folgt erzeugt: Sei t eine der Zeitschichten. Aus jeder Kante $e_i = (v_j, v_k)$ des stationären Graphen erzeugen wir in dieser Zeitschicht eine Kante des instationären Graphen, und zwar wie folgt: Sei Δt_i die Durchquerungszeit der Kante e_i , dann erzeugen wir eine Kante⁴⁹

$$\text{vom Knoten } v_j^t \text{ zum Knoten } v_k^{t+\Delta t_i}$$

im instationären Graphen — allerdings nur dann, wenn $t + \Delta t_i \leq T$ ist, wenn man also noch vor der Endzeit T am anderen Ende dieser Kante ankommen würde. Die Kapazität der erzeugten Kante ist dieselbe wie die von e_i im stationären Graphen. Es gibt keine Kanten zwischen Knoten desselben Zeitniveaus (keine Teleportation).

Die so erzeugten Kanten modellieren die tatsächliche *Bewegung* von Personen durch das Gebäude. Es fehlen noch die Kanten, die einem *Abwarten* in einem Knoten entsprechen. Dazu ziehen wir einfach in jeder Zeitschicht für jeden Knoten v_j , $j = 1, \dots, m_0$ des stationären Graphen eine Kante

$$\text{vom Knoten } v_j^t \text{ zum Knoten } v_j^{t+1}$$

ein — allerdings wieder nur dann, wenn $t + 1 \leq T$ bleibt. Die Kapazität der erzeugten Wartekanten wird auf ∞ gesetzt. Es gibt also keine Beschränkung dafür, wieviele Leute zwischen zwei Zeitschichten am selben Punkt verbleiben.

⁴⁹Hier geht entscheidend ein, dass die Durchquerungszeiten immer ganzzahlig sind, ansonsten würde man ja zwischen zwei Zeitebenen herauskommen.

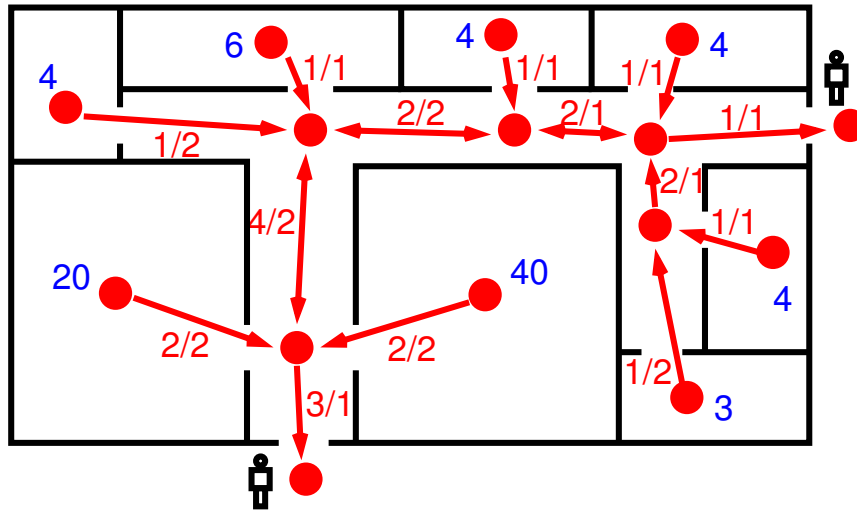


Abbildung 16.3.: Darstellung der Skizze aus Abb. 16.2 als (stationärer) Graph. Gezeigt sind die angenommenen Personenzahlen (in blau) in den einzelnen Räumen, die die Quellen darstellen. Die Kanten entsprechen Abschnitten auf den Fluchtwegen. Für jeden solchen Abschnitt ist die Kapazität sowie die Durchquerungszeit (z. B. in Vielfachen von 5 s) angegeben. Lesebeispiele: Durch den unteren Notausgang gelangen vom letzten Gangabschnitt davor höchstens drei Personen innerhalb eines Zeittaktes. Von dem mit 40 Personen belegten Raum können in jedem Zeittakt maximal zwei Personen heraus auf den Gang treten. Einmal unterwegs, benötigen Sie zwei Zeittakte, um den nächsten Knoten zu erreichen. Wie im Text erläutert, dient dieser Graph dient nur zur Illustration. Um das Evakuierungsproblem korrekt zu formulieren und zu lösen, muss man zu einer instationären Formulierung über mehrere Zeitperioden kommen.

Wir haben jetzt den instationären Graphen vollständig beschrieben, kennen also auch dessen Inzidenzmatrix A .⁵⁰ Weiterhin haben wir alle Kantenkapazitäten u angegeben. Für die komplette Formulierung als Problem des kostenminimalen Flusses (14.1) fehlt noch die Angabe der Knotenbilanzen b und der Kantenkosten c . Die Einträge im Vektor der Knotenbilanzen kennzeichnen wir wie die Knoten selbst mit b_j^t , wobei $j = 1, \dots, m_0$ und $t = 0, \dots, T$ laufen. Wir setzen

$$b_j^t = \begin{cases} \widehat{b}_j, & \text{falls } t = 0 \text{ ist und der Knoten } v_j \text{ im stationären Netzwerk eine Quelle} \\ & \text{(gefüllter Raum) der Stärke } \widehat{b}_j < 0 \text{ war,} \\ \widehat{b}_j, & \text{falls } t = T \text{ ist und der Knoten } v_j \text{ im stationären Netzwerk eine Senke} \\ & \text{(Sammelpunkt) der Stärke } \widehat{b}_j > 0 \text{ war,} \\ 0 & \text{sonst.} \end{cases}$$

Jetzt beschreibt jeder zulässige Fluss (vgl. Definition 14.5) eine funktionierende Evakuierungsstrategie, bei der sich am Ende (zum gesetzten Zeitpunkt $t = T$) alle Personen am

⁵⁰Diese Konstruktion des Graphen von Hand zu programmieren ist hier noch gut machbar, aber bei größeren Aufgaben unpraktisch. Es gibt dafür aber spezielle Modellierungssoftware wie etwa <http://www.zet-evakuierung.de>.

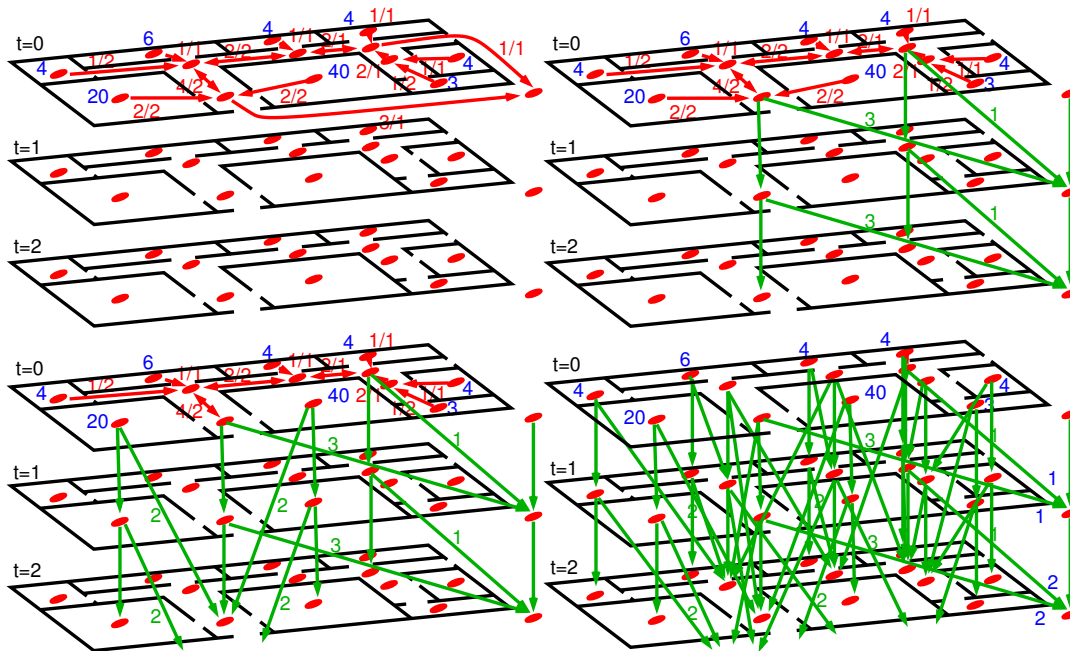


Abbildung 16.5.: Konstruktion des instationären aus dem stationären Graphen. Gezeigt sind nur die ersten drei Zeitebenen. Oben links: Kopie der Knoten in alle Zeitschichten. Rechts oben: Die von den Ausgängen zum Sammelpunkt führenden Kanten werden eingefügt, diese haben Kapazitäten 3 bzw. 1. Außerdem werden die Wartekanten an den Ausgängen und am Sammelpunkt erzeugt. Unten links: Zusätzlich werden hier gerade die aus zwei der Räume herausführenden Kanten eingefügt. Da diese Kanten eine Durchquerungszeit von 2 Zeittakten haben, erreichen die in $t = 0$ startenden Kanten ihre Zielknoten erst bei Zeitschicht $t = 2$. Unten rechts: Sichtbar sind bereits alle Kanten, die in den Zeitschichten $t = 0$ und $t = 1$ starten.

- (ii) Im Beispiel ist die Evakuierung zum Zeitpunkt $t = 23$ abgeschlossen. Die zwei überflüssigen Zeitebenen $t \in \{24, 25\}$ erzeugen zwar mehr Optimierungsvariablen, haben sonst aber keine Auswirkung auf der Ergebnis. Wählt man die Anzahl der Zeittakte jedoch zu klein, so gibt es keinen zulässigen Punkt.
- (iii) Da die Inzidenzmatrix des dem Problem zugrunde liegenden (instationären) Graphen wieder total unimodular ist und die Knotenbilanzen b und Kantenkapazitäten u ganzzahlig sind, sind auch alle Ecken des zulässigen Polyeders $\{x \in \mathbb{R}^n : Ax = b, 0 \leq x \leq u\}$ ganzzahlig, und das Simplex-Verfahren ermittelt eine optimale Lösung mit ganzzahligen Flussvektoren. Dies heißt ganz konkret, dass sich niemand auf der Flucht teilen muss!
- (iv) Wir haben positive Kosten nur auf den Kanten zum Sammelpunkt angesetzt und so die Gesamtverweildauer aller Personen im Gebäude als Zielfunktion verwendet. In weiteren Modellierungsszenarien könnte man z. B. die Rauchentwicklung bei bekanntem (angenommenem) Brandherd berücksichtigen und den Aufenthalt von Personen auf den angrenzenden Kanten mit weiteren Kosten beaufschlagen.
- (v) Mit Hilfe der Optimierung kann man viele Szenarien schnell durchspielen, etwa Einschrän-

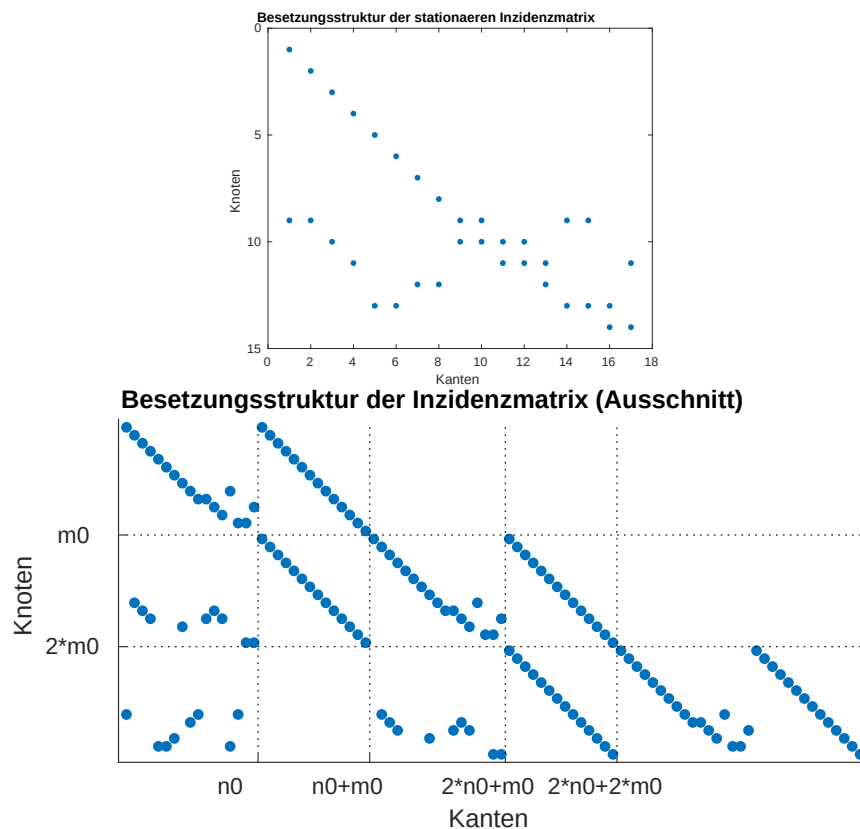


Abbildung 16.6.: Besetzungsstruktur der Inzidenzmatrix des *stationären* Graphen beim Evakuierungsproblem mit $m_0 = 14$ Knoten und $n_0 = 17$ Kanten (oben). Das untere Bild zeigt einen Ausschnitt (zu den ersten drei Zeitschichten) der Inzidenzmatrix des zugehörigen *instationären* Graphen. Die drei Blockzeilen entsprechen den Knoten in den Zeitschichten $t \in \{0, 1, 2\}$. Die Kanten der ersten Blockspalte sind die Bewegungskanten, die zur Zeitschicht $t = 0$ starten und an einem Knoten in $t = 1$ bzw. $t = 2$ enden. In der zweiten Blockspalte sieht man die Wartekanten von $t = 0$ nach $t = 1$. In der dritten und vierten Blockspalte folgen dann die Bewegungs- bzw. Wartekanten, die in der Zeitschicht $t = 1$ starten usw.

kungen von Fluchtwegkapazitäten durch Gegenstände auf den Gängen, Überbelegung der Räume, Einrichtung weiterer Notausgänge etc.

- (vi) *Das Evakuierungsproblem ist eine typische Optimierungsaufgabe auf einem Netzwerk mit mehreren Perioden. Solche Aufgaben treten auch bei den schon behandelten Transportproblemen (vgl. [Beispiel 1.1](#) und [Beispiel 14.1](#)) auf, wenn aus Kapazitätsgründen das gesamte Transportvolumen nicht in einem Zeittakt bewältigt werden kann.*
- (vii) *Unsere Modellierung des Evakuierungsproblems ist nur dann korrekt, wenn alle Personen „gleichartig“ sind und nicht unterschieden werden müssen.*

Alle bisherigen Aufgaben in diesem Kapitel ([Beispiel 14.1](#), [Beispiel 16.1](#) und [Beispiel 16.3](#)) führten letztlich auf Aufgaben mit kostenminimalen Flüssen. Die nun folgenden Beispiele zeigen noch andere Problemklassen von Optimierungsaufgaben auf Graphen.

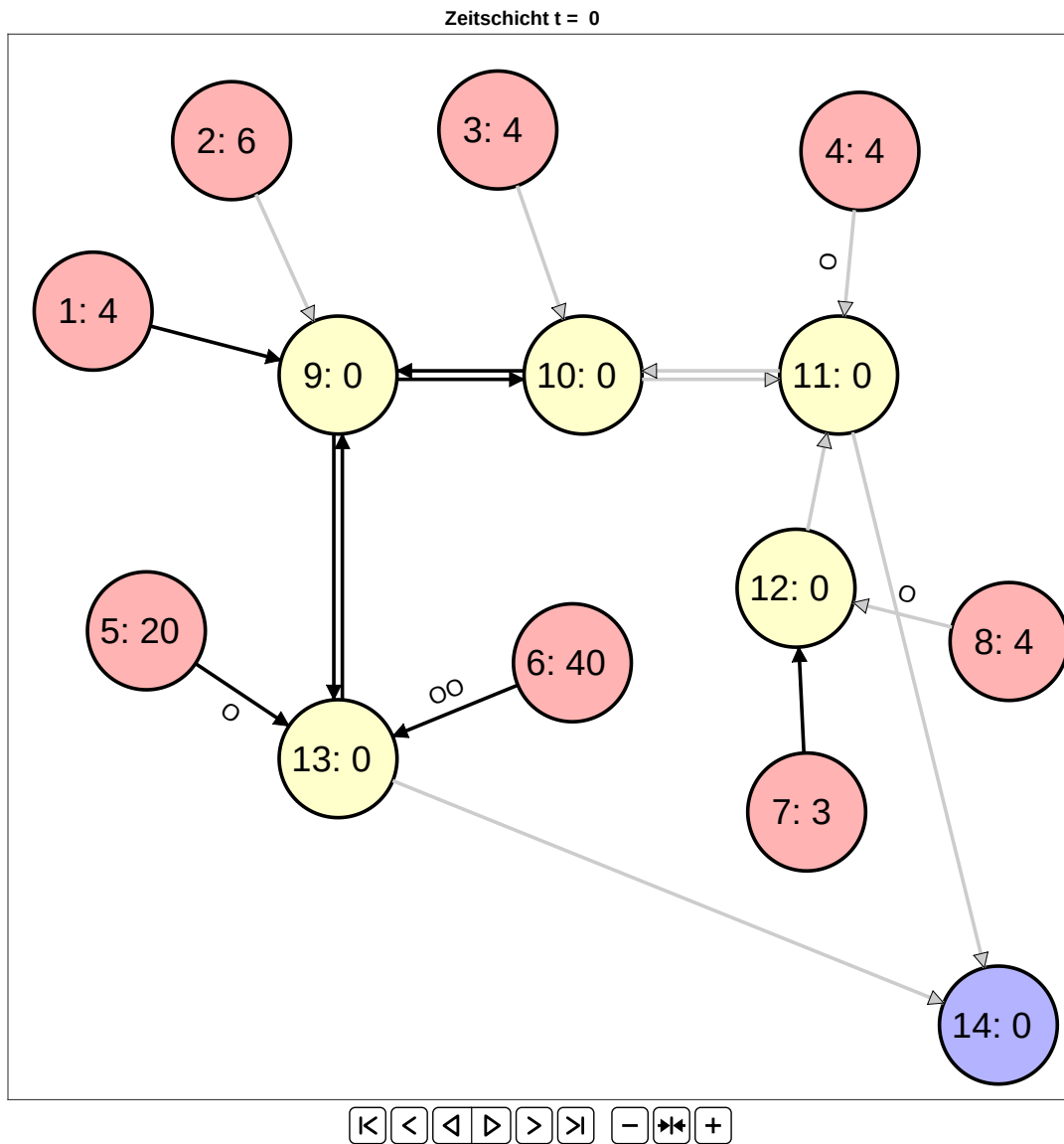


Abbildung 16.7.: Ein optimaler Evakuierungsverlauf (interaktiv) für [Beispiel 16.3](#). Gezeigt werden in jedem Knoten (Raum) die Knotennummer: aktuelle Personenanzahl. Die Symbole an den Kanten stellen die Personen dar, die im aktuellen Zeitschritt diese Kante gerade betreten haben. Die grauen Kanten haben eine Durchquerungszeit von 1, die schwarzen Kanten von 2.

Beispiel 16.5 (Transportproblem, maximaler Fluss⁵²). In einer Ölraffinerie soll soviel Öl wie möglich über das in [Abb. 16.8](#) abgebildete Netzwerk aus Pipelines transportiert werden. Der Ausgangsknoten / Quelle (engl.: source) wird oft mit s bezeichnet, der Zielknoten / Senke (engl.: target) mit t . Die maximale Durchsatzmenge der einzelnen Pipelines (Kanten) ist ebenfalls angegeben.

Bei dem so erhaltenen Graphen handelt es sich streng genommen nicht um ein Transportnetzwerk (vgl. [Definition 14.5](#)), da die Knotenbilanzen an Quelle und Senke unbekannt sind. (Sie sollen ja gerade optimiert werden.) Mit folgender Idee gelingt es aber, zu einer äquivalenten Aufgabe mit bekannten Knotenbilanzen zu kommen: Wir ziehen eine künstliche **Rückführungskante** zurück von der Senke zur Quelle ein. Damit werden alle Knoten zu Durchflussknoten, der Bilanzvektor ist $b = 0$.

Solche Aufgaben heißen Probleme des **maximalen Flusses**. Sie haben allgemein folgende Gestalt:

$$\begin{aligned} &\text{Maximiere} && x_{t,s}, && x \in \mathbb{R}^n \\ &\text{unter} && Ax = 0 \\ &\text{sowie} && 0 \leq x \leq u. \end{aligned} \tag{16.1}$$

Dabei ist $u_{t,s} = \infty$, d. h., auf der Rückführungskante gibt es keine Kapazitätsbeschränkung. Als Kostenvektor in der als *Minimierungsaufgabe* formulierten äquivalenten Variante von (16.1) wird

$$c = (0, \dots, 0, \underbrace{-1}_{\text{zur Kante } x_{t,s}})^T$$

gesetzt. Dadurch sorgen wir dafür, dass der (künstliche) Rückfluss von t zurück nach s maximal wird.

Die Lösung für [Beispiel 16.5](#) wird in [Abb. 16.9](#) gezeigt. Der maximale Fluss, den man von der Quelle s zur Senke t durch das Netzwerk aus Pipelines transportieren kann, beträgt 6 Einheiten.

Bemerkung 16.6 (zu Aufgaben des maximalen Flusses). Aufgaben mit mehreren Quellen und Senken können ebenfalls in der Form (16.1) formuliert werden. Dazu führt man zunächst eine künstliche „Super-Quelle“ s und eine „Super-Senke“ t ein. Diese werden durch neue Kanten ohne Kapazitätsrestriktionen mit den ursprünglich vorhandenen Quellen und Senken verbunden.

Auch für Aufgaben des maximalen Flusses gibt es spezielle Verfahren, etwa der **Algorithmus von Ford und Fulkerson**, siehe etwa [Gerdt, Lempio, 2011](#), Abschnitt 4.3.

Wir kommen nun zum letzten hier vorgestellten Typ eines Optimierungsproblems auf Graphen, dem Zuordnungsproblem.

Beispiel 16.7 (Taxiunternehmen). Ein Taxiunternehmen bekommt gleichzeitig drei telefonische Fahraufträge. Momentan sind vier Taxis frei. Gesucht ist die günstigste Zuteilung der Fahrten zu den Taxis, sodass die GesamtWartezeit für alle drei Kunden minimal ist.

⁵²aus [Gerdt, Lempio, 2011](#), Abschnitt 4.3

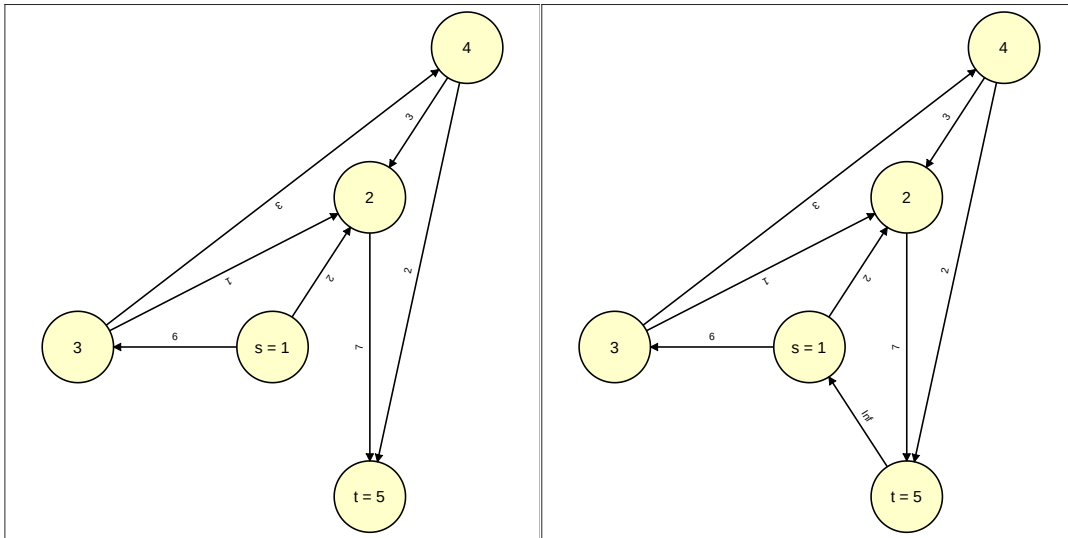


Abbildung 16.8.: Netzwerk aus Pipelines zu [Beispiel 16.5](#) (links). Dieses wird um die Rückführungskante ergänzt (rechts). Außerdem werden die Kapazitätsbeschränkungen u gezeigt.

Der Graph zum [Beispiel 16.7](#) ist in [Abb. 16.10](#) dargestellt. Das Besondere daran ist, dass die Kanten jeweils nur von den Taxis zu den Fahrgästen verlaufen. Es gibt also zwei Arten von Knoten (Taxis und Fahrgäste), und es gibt keine Kanten innerhalb einer Art (von Taxi zu Taxi oder Fahrgast zu Fahrgast). Ein solcher Graph heißt **bipartit**. Der Graph in [Abb. 16.10](#) ist sogar **vollständig bipartit**, da jede mögliche Kante auch tatsächlich vorkommt, weil jedes Taxi im Prinzip jeden Fahrauftrag übernehmen kann.

Jede Kante ist mit einem Gewicht (Nutzen) c_i versehen. Dieses ist umso höher, je „geeigneter“ der Fahrauftrag ist, also z. B. je näher sich der momentane Standort des Taxis am Standort des Kunden befindet. Jede Kante soll entweder mit dem Wert $x_i = 1$ (gewählt) oder $x_i = 0$ (nicht gewählt) belegt werden. Wir suchen eine Auswahl der Kanten, sodass die Summe der zugehörigen Nutzen maximal wird:

$$\sum_{i=1}^n c_j x_j = c^T x.$$

Dabei ist natürlich zu beachten, dass kein Taxi und auch kein Fahrgast mehrfach vergeben werden. Eine entsprechende Kantenauswahl, die dies berücksichtigt, nennt man ein **Matching**. Die Bedingungen, die ein Matching ausmachen, können mit Hilfe der Inzidenzmatrix

$$A = \begin{bmatrix} -1 & -1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -1 & -1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & -1 & -1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & -1 & -1 \\ \hline 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 \end{bmatrix}$$

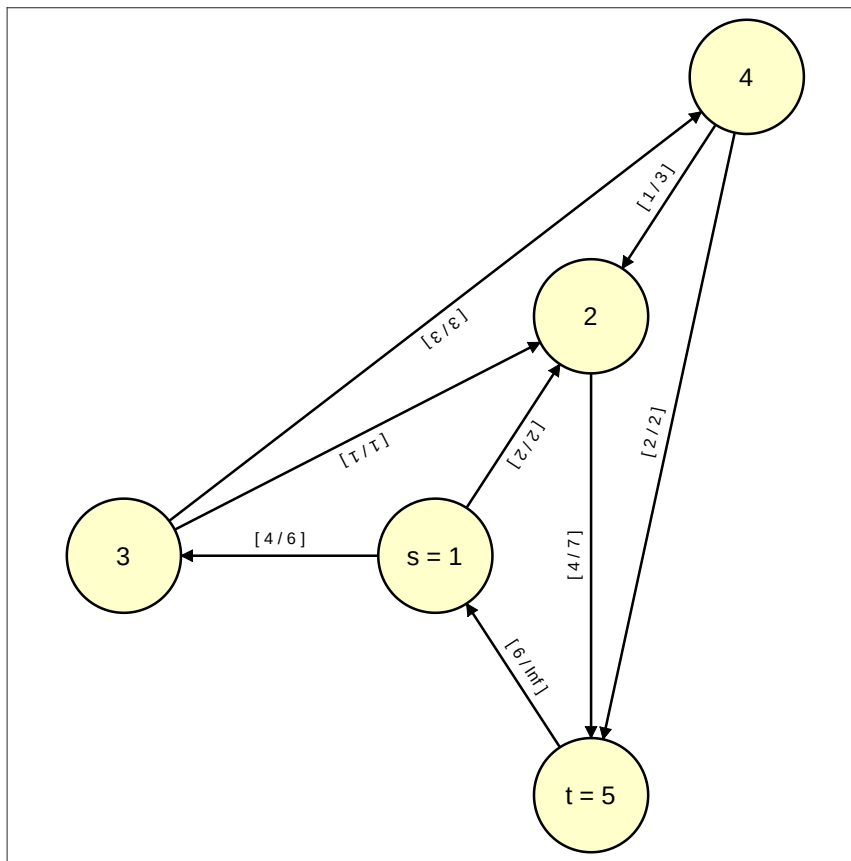


Abbildung 16.9.: Lösung für [Beispiel 16.5](#). Der maximale Fluss von der Quelle s zur Senke t beträgt 6, wie man an der Rückführungskante von t zurück nach s ablesen kann. Zu den Kanten sind jeweils [die optimalen Transportströme x^* / Kantenkapazitäten u] mit angegeben.

formuliert werden: Die Ungleichung

$$A^{1,2,3,4} x \geq -e \quad \text{oder äquivalent} \quad -A^{1,2,3,4} x \leq e$$

mit den ersten vier Zeilen von A sorgt dafür, dass kein Taxi mehrfach vergeben wird. Wie zuvor steht hier $e = (1, \dots, 1)^T$ für einen Vektor aus lauter Einsen passender Länge. Analog stellt

$$A^{5,6,7} x \leq e$$

sicher, dass kein Fahrauftrag mehrfach vergeben wird. Wir können alle Ungleichungen auch kompakt als

$$|A| x \leq e$$

schreiben, wobei $|A|$ *Eintrag für Eintrag* aus den Beträgen der Elemente von A besteht, also

hier⁵³

$$|A| = \begin{bmatrix} 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot \\ \hline 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \end{bmatrix}.$$

Wie beim Kürzeste-Wege-Problem (Beispiel 16.1) lässt sich die Bedingung $x_i \in \{0, 1\}$ für alle Kanten $i = 1, \dots, n$ einfach garantieren, indem man die Kantenkapazitäten auf $u_i = 1$ setzt:⁵⁴

$$0 \leq x \leq u = e.$$

Dies ist wieder so begründet, dass die Inzidenzmatrizen bipartiter (gerichteter oder ungerichteter) Graphen total unimodular und deshalb die Ecken des zulässigen Polyeders $\{x \in \mathbb{R}^n : |A| x \leq e, 0 \leq x \leq e\}$ alle ganzzahlig sind und Koordinaten in $\{0, 1\}$ haben.

Zusammenfassend erhalten wir die Optimierungsaufgabe

$$\begin{aligned} &\text{Maximiere} && c^T x, && x \in \mathbb{R}^n \\ &\text{unter} && |A| x \leq e \\ &\text{sowie} && 0 \leq x \leq e, \end{aligned} \tag{16.2}$$

die man allgemein als **(bipartites) Matching-** oder **Zuordnungsproblem** bezeichnet. Dabei ist $|A|$ die Inzidenzmatrix eines ungerichteten bipartiten Graphen. Im Fall, dass der Vektor c der Kantengewichte aus lauter Einsen besteht, spricht man von einem **ungewichteten** oder **kardinalitätsmaximalen** Matchingproblem. Es geht dann nur darum, die Anzahl der zugeordneten Paare (die Anzahl der Einsen im Flussvektor x) zu maximieren. Beim allgemeinen Vektor c der Kantengewichte liegt ein **gewichtetes** Matchingproblem vor.

Für unser Beispiel 16.7 wird in Abb. 16.10 die Lösung zum angegebenen Kostenvektor c gezeigt. Das optimale Matching umfasst die drei rot markierten Kanten, der Zielfunktionswert ist $3.8 + 5.3 + 3.4 = 12.5$.

Bemerkung 16.8 (zu Zuordnungsproblemen).

- (i) Für Zuordnungsprobleme gibt es viele Anwendungen, wie etwa die Zuordnung von Angestellten zu Arbeitsaufgaben oder Maschinen, die Zusammenstellung von Cockpit-Besatzungen in Flugzeugen oder in der Partnervermittlung („**Heiratsproblem**“).
- (ii) Ein ungewichtetes Zuordnungsproblem lässt sich auch als eine Aufgabe des maximalen Flusses formulieren. Ein gewichtetes Zuordnungsproblem mit bekannter Anzahl an Zuordnungen ist als kostenminimaler Fluss darstellbar.

⁵³ $|A|$ ist die Inzidenzmatrix des zugehörigen ungerichteten Graphen, bei dem die Richtungen der Kanten nicht unterschieden werden.

⁵⁴Auch hier kann man die obere Schranke weglassen, da die Matching-Bedingungen ohnehin verhindern, dass eine Kante mehrfach gewählt wird.

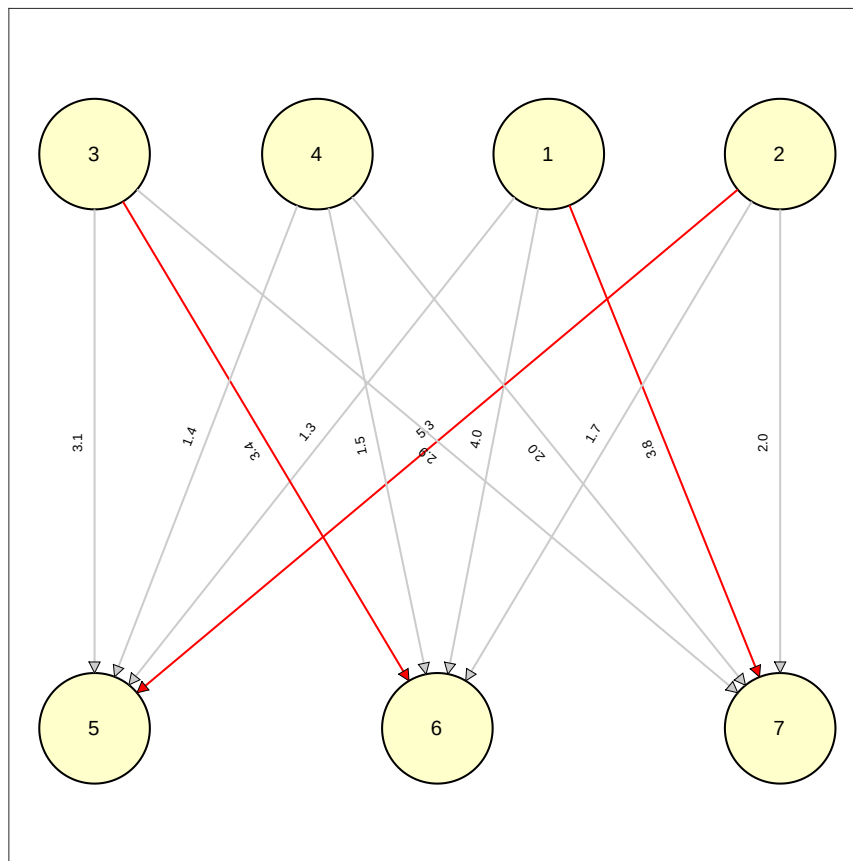


Abbildung 16.10.: Lösung für [Beispiel 16.7](#) zum dargestellten Gewichtsvektor c . Das optimale Matching besteht aus den drei rot markierten Kanten, und diese sind im Vektor $x^* \in \{0, 1\}^{12}$ durch die drei Einträge repräsentiert, die gleich Eins sind.

- (iii) Auch für Zuordnungsaufgaben gibt es spezielle Verfahren, etwa der **Algorithmus von Kuhn und Munkres** (auch **ungarische Methode** genannt).

Eine Warnung zum Abschluss dieses Kapitels: In allen unseren Beispielen in § 14 und § 16, die alle letztendlich auf lineare Optimierungsaufgaben mit total unimodularer Matrix A in den Gleichungs- bzw. Ungleichungsnebenbedingungen geführt haben, war die durch das Simplex-Verfahren erhaltene Lösung x^* immer *automatisch ganzzahlig*, wenn b und u es waren. Diese Eigenschaft der optimalen Lösung ist bei vielen Aufgabenstellungen auf Transportnetzwerken erwünscht oder sogar erforderlich, weil sich die verwendeten Transporteinheiten (Paletten, LKW etc.) nicht teilen lassen.

Wenn jedoch die Modellierung eine lineare Optimierungsaufgabe (11.1) ergibt, bei der die Matrix

$$A = \begin{bmatrix} A_{\text{ineq}} \\ A_{\text{eq}} \end{bmatrix}$$

nicht total unimodular ist, dann ist die Ganzzahligkeit der Lösung x^* im Simplex-Verfahren

nicht garantiert, sondern muss extra gefordert werden:

$$\begin{aligned}
 &\text{Minimiere} && c^\top x, && x \in \mathbb{Z}^n \\
 &\text{unter} && A_{\text{ineq}} x \leq b_{\text{ineq}} \\
 &\text{und} && A_{\text{eq}} x = b_{\text{eq}} \\
 &\text{sowie} && \ell \leq x \leq u.
 \end{aligned} \tag{16.3}$$

Eine solche Aufgabe heißt ein **ganzzahliges (lineares) Programm** (engl.: *integer (linear) program*), kurz: **IP** oder **ILP**. Sie ist erheblich schwerer zu lösen als die ansonsten gleiche Aufgabe mit der Grundmenge $x \in \mathbb{R}^n$ ⁵⁵. Sollen nur einige der Variablen ganzzahlig sein und andere nicht, so spricht man von einem **gemischt-ganzzahligen (linearen) Programm** (engl.: *mixed-integer (linear) program*), kurz: **MIP** oder **MILP**.

Beispiele für Transportprobleme, bei denen die Matrix A nicht total unimodular ist, sind beispielsweise **Mehrgütertransportprobleme** (**Mehrgüterflussprobleme**, engl.: *multi-commodity flow problems*).

Beispiel 16.9 (Mehrgütertransportproblem, kostenminimaler Fluss, vgl. [Beispiel 14.1](#)). Ein Unternehmen verfügt über das in [Abb. 16.11](#) dargestellte Transportnetzwerk. Es geht darum, den Transport von zwei verschiedenen Gütern von den jeweiligen Produktionsstätten zu den zugehörigen Verkaufsstätten zu planen.

Die Kapazitäten der Produktionsstätten sowie die Bedarfe der Verkaufsstätten an dem produzierten bzw. benötigten Gut sind (der Einfachheit halber) jeweils eins. Die beiden Güter teilen sich dabei die Transportwege, deren Gesamtkapazität jeweils auf eins beschränkt ist. Die Transportkosten für jede Kante sind wie in [Beispiel 14.1](#) proportional zu der darüber transportierten Warenmenge und können für die beiden Güter unterschiedlich sein. Gesucht ist nun diejenige Verteilung der Transportströme, die unter Beachtung der Kapazitätsrestriktionen die Gesamttransportkosten minimiert.

Zur Modellierung eines Mehrgüterflussproblems müssen wir den Graphen des Transportnetzwerks vervielfachen (bei zwei Gütern: verdoppeln), siehe [Abb. 16.11](#). Beide Kopien sind identisch und haben dieselbe Inzidenzmatrix A_0 . Wir bezeichnen mit $x^{(1)}$ den Flussvektor des ersten Gutes und mit $x^{(2)}$ den des zweiten. Die Knotenbilanzen werden mit $b^{(1)}$ und $b^{(2)}$ und die Kantenkosten mit $c^{(1)}$ und $c^{(2)}$ bezeichnet.

Wir erhalten folgende Optimierungsaufgabe des kostenminimalen **Zweigüterflusses** als Erweiterung der Aufgabe für ein einzelnes Transportgut ([14.1](#)):

⁵⁵Die Schwierigkeit solcher Aufgaben liegt in der kombinatorischen Komplexität der zulässigen Menge begründet. Diese Aufgaben fallen in den Bereich der *Diskreten Optimierung*.

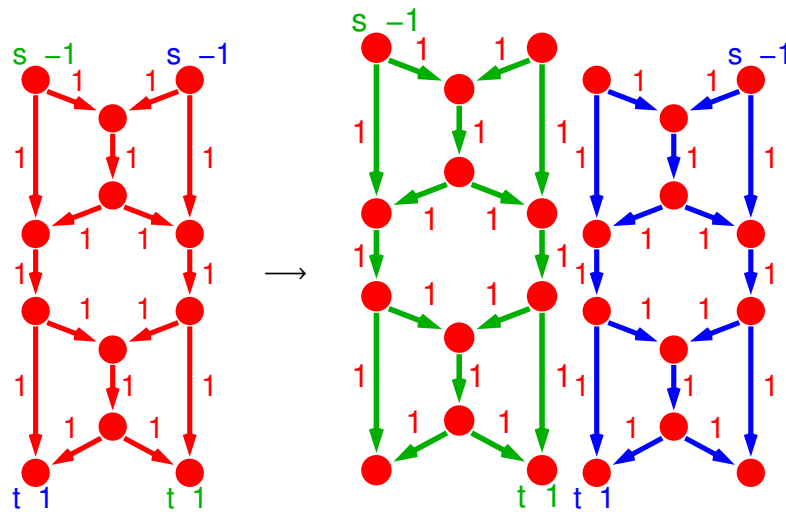


Abbildung 16.11.: Zwei Güter teilen sich ein Transportnetzwerk (links). Zur Modellierung des Problems des kostenminimalen Flusses wird der zugehörige Graph verdoppelt (rechts).

$$\begin{aligned}
 &\text{Minimiere} \quad (c^{(1)})^T x^{(1)} + (c^{(2)})^T x^{(2)}, \quad x^{(1)}, x^{(2)} \in \mathbb{R}^n \\
 &\text{unter} \quad \begin{cases} A_0 x^{(1)} = b^{(1)} & \text{Knotenbilanzen Transportgut 1} \\ A_0 x^{(2)} = b^{(2)} & \text{Knotenbilanzen Transportgut 2} \\ x^{(1)} + x^{(2)} \leq u & \text{Kapazitätsbeschränkung} \end{cases} \quad (16.4) \\
 &\text{sowie} \quad x^{(1)} \geq 0, \quad x^{(2)} \geq 0.
 \end{aligned}$$

Wenn es die gemeinsame Kapazitätsbeschränkung nicht gäbe, hätten wir es mit zwei einzelnen (entkoppelten) Eingüterflussproblemen zu tun.

Wenn wir die Aufgabe (16.4) wie gehabt mit dem Simplex-Verfahren in `linprog` lösen, ergeben sich als Lösung x^* die in Abb. 16.12 dargestellten Kantenflüsse. Genau genommen ist dies sogar überhaupt der einzige Flussvektor, der alle Gleichungs- und Ungleichungsbeschränkungen erfüllt, und damit natürlich gleichzeitig der mit den geringsten Kosten.

Die Ecken des zulässigen Polyeders sind hier offenbar nicht ganzzahlig, denn: In der Form (11.1) geschrieben lauten hier unsere Matrizen und Vektoren in den Gleichungs- und Ungleichungsbeschränkungen

$$\begin{aligned}
 A_{\text{eq}} &= \begin{bmatrix} A_0 & \cdot \\ \cdot & A_0 \end{bmatrix}, \quad b_{\text{eq}} = \begin{pmatrix} b^{(1)} \\ b^{(2)} \end{pmatrix} \\
 A_{\text{ineq}} &= \begin{bmatrix} I & I \end{bmatrix}, \quad b_{\text{ineq}} = u,
 \end{aligned}$$

und die Gesamtmatrix

$$A = \begin{bmatrix} A_{\text{eq}} \\ A_{\text{ineq}} \end{bmatrix} = \begin{bmatrix} A_0 & \cdot \\ \cdot & A_0 \\ I & I \end{bmatrix}$$

ist *nicht* total unimodular, obwohl dies auf die einzelnen Blöcke noch zutrifft.

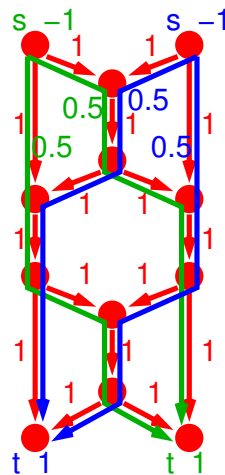


Abbildung 16.12.: Die eindeutige Lösung x^* des Zweigüterflussproblems (16.4) ist nicht ganzzahlig. Der Fluss eines jeden Gutes über jede Kante ist entweder 0 oder 0.5.

Sofern die Ganzzahligkeit der Transportströme nicht notwendig ist, kann man also die Zweigüterflussaufgabe (16.4) mit dem Simplex-Verfahren (oder seiner Variante, dem Netzwerk-Simplex-Verfahren) lösen. Andernfalls benötigt man spezielle Algorithmen für die **diskrete Optimierungsaufgabe** (ein **IP**) (16.3), in der die Ganzzahligkeit der Optimierungsvariablen x explizit gefordert wird ($x \in \mathbb{Z}^n$). Stichworte sind hier **Branch-and-bound**- und **Branch-and-cut-Verfahren**. In der MATLAB-Optimierungs-Toolbox ist ein Branch-and-bound-Verfahren (bintprog) für **binäre lineare Optimierungsaufgaben**, einem Spezialfall von (16.3) mit $x \in \{0, 1\}^n$, sowie (seit Version R2014a) ein Cut-and-branch-Verfahren (intlinprog) für **gemischt-ganzzahlige lineare Optimierungsaufgaben** enthalten.

Mit Mehrgüterflussproblemen modelliert man beispielsweise auch Aufgaben in Verkehrsnetzwerken (Straßen), wo Gruppen von Verkehrsteilnehmern von verschiedenen Ausgangspunkten (z. B. Vororten) zu verschiedenen Zielen (in der Stadt) fahren wollen. Jede vorkommende Kombination aus Ausgangs- und Zielknoten stellt ein eigenes „Transportgut“ da. Man teilt sich gemeinsam die Kapazität der Kanten (Straßen). Analog zum **Evakuierungsproblem 16.3** kann man auch hier zeitabhängige Varianten betrachten, die Quellen in der Zeit verteilen (um verschiedene Abfahrtszeiten zu modellieren) und die Dauer der Arbeitswege in der Zielfunktion bewerten.

Ende der Woche 12

Kapitel 4 Jetzt wird's beschränkt (Nichtlineare Optimierung unter Nebenbedingungen)

Wir beginnen dieses Kapitel mit einem kurzen Rückblick. In [Kapitel 1](#) hatten wir zunächst Optimierungsaufgaben ohne Beschränkungen kennengelernt und gelöst. Dasselbe trifft auch auf [Kapitel 2](#) zu, wobei die Zielfunktion dabei ein spezielles Aussehen (Kleinste-Quadrate-Struktur) hatte, was von den einschlägigen Lösungsverfahren ausgenutzt wird. In [Kapitel 3](#) kamen bei der Optimierung erstmals Gleichungs- und Ungleichungsbeschränkungen ins Spiel; jedoch waren diese, wie auch die Zielfunktion, (affin-)linear, also von besonders einfacher Gestalt, was natürlich wiederum von den Lösungsverfahren ausgenutzt wird, insbesondere vom Simplex-Verfahren.

§ 17 EIN ERSTES BEISPIEL

In diesem Kapitel geht es nun um allgemeine **nichtlineare Optimierungsaufgaben** mit Beschränkungen:

$$\begin{aligned} \text{Minimiere } & f(x), & x \in \mathbb{R}^n, \\ \text{sodass } & g_i(x) \leq 0, & i = 1, \dots, m, \\ \text{und } & h_j(x) = 0, & j = 1, \dots, p. \end{aligned} \tag{17.1}$$

Man spricht auch von einem **nichtlinearen Programm (NLP)**. Dabei ist $m \geq 0$ die Anzahl der Ungleichungs- und $p \geq 0$ die Anzahl der Gleichungsnebenbedingungen.¹ Alle Funktionen $f(x)$, $g_i(x)$ und $h_j(x)$ sollen als Generalvoraussetzung in diesem Kapitel mindestens einmal stetig partiell diffbar sein.

Die Zielfunktion $f(x)$ in der Aufgabe (17.1) darf also nicht über der ganzen Grundmenge $x \in \mathbb{R}^n$ minimiert werden, sondern es müssen Beschränkungen in Form von Gleichungen und Ungleichungen beachtet werden. In den allermeisten praktischen Aufgabenstellungen (von Kleinste-Quadrate-Aufgaben abgesehen) sind solche Beschränkungen wesentlich, d. h., die Optimierungsaufgaben werden sinnlos, wenn man Beschränkungen weglässt oder vergisst zu modellieren.²

¹Natürlich dürfen m oder p auch gleich Null sein, wenn keine Ungleichungen oder keine Gleichungen vorliegen.

Im Extremfall ist $m = p = 0$, und wir sind wieder bei der freien Optimierung ([Kapitel 1](#)).

²Es ist dann z. B. ein unendlich großer Gewinn möglich.

In der Aufgabe (17.1) sind zur Konkurrenz um den kleinsten Zielfunktionswert nur diejenigen $x \in \mathbb{R}^n$ zugelassen, die bezüglich *aller* Gleichungen und Ungleichungen **zulässig** sind:

$$X = \{x \in \mathbb{R}^n : g_i(x) \leq 0 \quad \text{für alle } i = 1, \dots, m, \\ h_j(x) = 0 \quad \text{für alle } j = 1, \dots, p\}. \quad (17.2)$$

Man bezeichnet X als die **zulässige Menge** der Aufgabe (17.1), und jedes $x \in X$ heißt ein **zulässiger Punkt**.

Im Prinzip kann es vorkommen, dass das System aus Gleichungs- und Ungleichungsbeschränkungen Widersprüche enthält, also gar nicht zu erfüllen ist ($X = \emptyset$). In der Regel liegt dann ein Modellierungsfehler vor. Lösungsverfahren sollten diese Situation natürlich möglichst erkennen und darauf hinweisen.

Beispiel 17.1 (Querlenker minimalen Gewichts³). In Abb. 17.1 ist der *Querlenker* eines Autos dargestellt. Seine Form soll in folgendem Sinne optimiert werden: Bei gegebener Mindeststeifigkeit unter Belastung in z -Richtung soll das Volumen (und damit das Gewicht) des Bauteils minimiert werden.

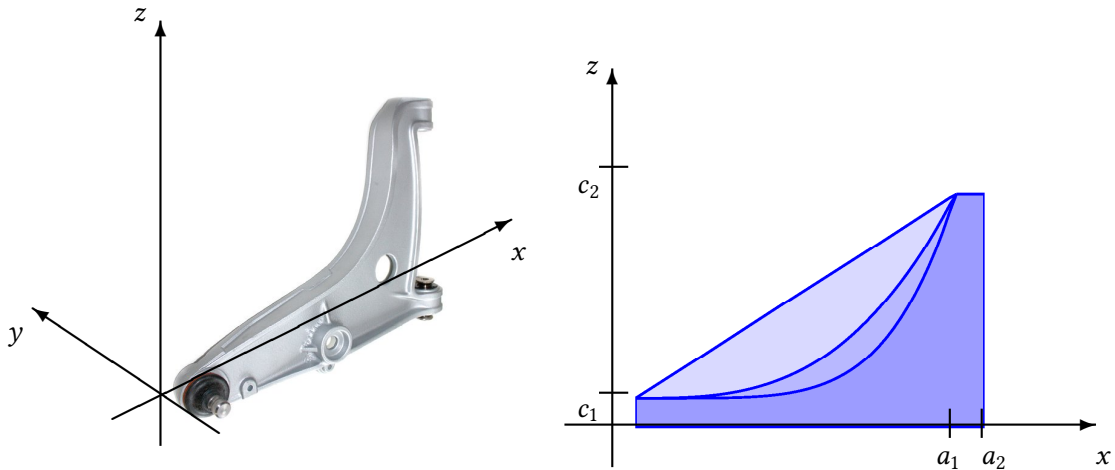


Abbildung 17.1: Querlenker (links) und schematische Darstellung des Querschnitts (rechts) für verschiedene Formparameter $m \in \{1, 3, 5\}$.

Wir müssen zunächst die Optimierungsaufgabe modellieren. Dazu wird die Form des Querlenkers durch den in Abb. 17.1 (rechts) dargestellten Querschnitt angenähert. Die obere Begrenzungskurve folgt dem angenommenen Verlauf

$$k(x; a_1, a_2, c_1, c_2, m) = \begin{cases} c_1 + (c_2 - c_1) \left(\frac{x}{a_1}\right)^m & \text{für } 0 \leq x \leq a_1, \\ c_2 & \text{für } a_1 \leq x \leq a_2. \end{cases}$$

Die grundsätzliche Form des Bauteils ist durch diese Kurve vorgegeben, und das genaue Aussehen wird durch die fünf Formparameter a_1 , a_2 , c_1 , c_2 und m festgelegt, wobei $a_{1,2} \geq 0$,

³Dieses Beispiel ist die Weiterentwicklung einer Idee von Herrn PD Dr.-Ing. Thorsten Halle (ehemals Professur Oberflächentechnik/Funktionswerkstoffe, jetzt an der Universität Magdeburg).

$c_{1,2} \geq 0$ und $m \geq 1$ eingehalten werden sollen. Dabei sind $a_{1,2}$ und $c_{1,2}$ Längenparameter mit der Einheit mm, und m ist eine dimensionslose Zahl.

Das zu minimierende Volumen des Querlenkers ist proportional zur Fläche A des Querschnitts, die sich zu

$$A(a_1, a_2, c_1, c_2, m) = \frac{a_1 (c_2 + c_1 m)}{m + 1} + (a_2 - a_1) c_2 \quad (17.3)$$

ergibt. Nun ist noch zu klären, was mit dem Begriff „Steifigkeit“ gemeint ist. Wir verstehen darunter die Biegesteifigkeit bzgl. Belastungen in z -Richtung. Diese ergibt sich als Produkt des (gegebenen) Elastizitätsmoduls E des verwendeten Werkstoffes (Einheit $\frac{\text{N}}{\text{mm}^2}$) und des axialen Flächenträgheitsmoments I_x (Einheit mm^4), das von unseren zu optimierenden Geometrieparametern abhängt. Das Flächenträgheitsmoment erhalten wir nach dem **Satz von Steiner** wie folgt:

$$I_x(a_1, a_2, c_1, c_2, m) = \int z^2 dA - [z_s(a_1, a_2, c_1, c_2, m)]^2 A(a_1, a_2, c_1, c_2, m).$$

Dabei ist⁴

$$\begin{aligned} z_s(a_1, a_2, c_1, c_2, m) &= \frac{1}{A(a_1, a_2, c_1, c_2, m)} \int z dA \\ &= \frac{1}{2 A(a_1, a_2, c_1, c_2, m)} \left[\frac{a_1 (2 c_1^2 m^2 + c_2^2 m + 2 c_2 c_1 m + c_2^2)}{(3 m + 1 + 2 m^2)} + c_2^2 (a_2 - a_1) \right] \end{aligned} \quad (17.4)$$

die z -Koordinate des Flächenschwerpunkts. Nach einiger Rechnung erhalten wir den Ausdruck

$$\begin{aligned} I_x(a_1, a_2, c_1, c_2, m) &= \frac{1}{3} \cdot \frac{a_1 (c_2^3 + 3 c_2^2 c_1 m + 6 c_2 c_1^2 m^2 + 3 c_2^2 c_1 m^2 + 6 c_1^3 m^3 + 3 c_2^3 m + 2 c_2^3 m^2)}{11 m^2 + 6 m + 6 m^3 + 1} \\ &\quad + \frac{1}{3} c_2^3 (a_2 - a_1) - [z_s(a_1, a_2, c_1, c_2, m)]^2 A(a_1, a_2, c_1, c_2, m) \end{aligned} \quad (17.5)$$

für das Flächenträgheitsmoment.

Konstruktionsbedingt geben wir noch untere und obere Schranken für die Optimierungsvariablen vor:

$$\begin{aligned} 180 &\leq a_1, & 30 &\leq c_1, & 1 &\leq m \leq 10, \\ 240 &\leq a_2, & 150 &\leq c_2 \end{aligned}$$

sowie die Bedingungen

$$\begin{aligned} a_2 &\geq a_1 + 20 & \text{oder auch} & & a_1 - a_2 &\leq -20, \\ c_2 &\geq c_1 + 20 & \text{oder auch} & & c_1 - c_2 &\leq -20. \end{aligned}$$

Schließlich soll das den Querlenker umgebende Rechteck gewisse Proportionen aufweisen:

$$\begin{aligned} a_2 &\leq 1.5 c_2 & \text{oder auch} & & a_2 - 1.5 c_2 &\leq 0, \\ c_2 &\leq 1.5 a_2 & \text{oder auch} & & c_2 - 1.5 a_2 &\leq 0. \end{aligned}$$

⁴Auch dies kann zur eigenen Übung (bei entsprechenden Kenntnissen der Technischen Mechanik) nachgerechnet werden. Alle Rechnungen wurden hier mit der Computer-Algebra-Software MAPLE realisiert.

Wir können nun unsere Optimierungsaufgabe aus [Beispiel 17.1](#) genau spezifizieren:

$$\begin{aligned}
 &\text{Minimiere } A(x), \quad x = (a_1, a_2, c_1, c_2, m) \in \mathbb{R}^5 \\
 &\text{sodass } I_x(x) \geq I_0 \\
 &\text{und } \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1.5 & 0 \\ 0 & -1.5 & 0 & 1 & 0 \end{bmatrix} x \leq \begin{pmatrix} -20 \\ -20 \\ 0 \\ 0 \end{pmatrix} \\
 &\text{und } \ell \leq x \leq u.
 \end{aligned} \tag{17.6}$$

Dabei wurde die Forderung nach einer Mindeststeifigkeit der Struktur in eine Forderung nach der Einhaltung eines gegebenen Mindestträgheitsmoments I_0 umgerechnet, das wir hier als $I_0 = 3 \times 10^7 \text{ mm}^4$ annehmen. Die angegebenen Schranken lauten in Vektorform

$$\ell = (180, 240, 30, 150, 1)^\top \quad \text{und} \quad u = (\infty, \infty, \infty, \infty, 10)^\top.$$

Wir lösen diese Optimierungsaufgabe nun mit Hilfe der MATLAB-Funktion `fmincon`. Diese Routine behandelt nichtlineare Optimierungsaufgaben der folgenden Gestalt:

$$\begin{aligned}
 &\text{Minimiere } f(x), \quad x \in \mathbb{R}^n, \\
 &\text{sodass } \begin{cases} c_{\text{eq}}(x) = 0, \\ c_{\text{ineq}}(x) \leq 0, \\ A_{\text{eq}} x = b_{\text{eq}}, \\ A_{\text{ineq}} x \leq b_{\text{ineq}}, \end{cases} \\
 &\text{sowie } \ell \leq x \leq u.
 \end{aligned} \tag{17.7}$$

Die zulässige Menge für eine solche Aufgabe ist natürlich

$$\begin{aligned}
 X = \{x \in \mathbb{R}^n : c_{\text{eq}}(x) = 0, c_{\text{ineq}}(x) \leq 0, \\
 A_{\text{eq}} x = b_{\text{eq}}, A_{\text{ineq}} x \leq b_{\text{ineq}}, \ell \leq x \leq u\}.
 \end{aligned} \tag{17.8}$$

Man kann – und sollte! – also diejenigen Gleichungs- und Ungleichungsbeschränkungen in der allgemeinen Aufgabe (17.1) separat angeben, die (affin-)linear sind, und ebenso die Box-Beschränkungen. Die anderen Gleichungs- und Ungleichungsbeschränkungen $c_{\text{eq}}(x) = 0$ und $c_{\text{ineq}}(x) \leq 0$ werden von `fmincon` als nichtlinear behandelt.⁵

Bei der Umsetzung in MATLAB setzen wir zunächst wieder alle Problem Daten:

Matlab/Querlenker.m

```
% Setzen der Zielfunktion
f = @(x) Querlenker_Flaeche(x);

% Setzen des Mindestflaechentraegheitsmoments und der nichtlinearen
  Beschraenkungen
```

⁵Der Vorteil der Unterscheidung zwischen linearen und nichtlinearen Beschränkungen liegt darin, dass `fmincon` nur für nichtlineare Funktionen in jeder Iteration des Verfahrens Ableitungen berechnen muss. Die Ableitung (Jacobimatrix) z. B. der Beschränkung $A_{\text{eq}} x - b_{\text{eq}} = 0$ ist einfach A_{eq} , was erstens von der aktuellen Stelle (Iterierten des Verfahrens) x_k unabhängig ist und zweitens vom Benutzer schon angegeben wird.

```

I0 = 3e7;
gh = @(x) Querlenker_Beschraenkungen(x,I0);

% Setzen der linearen Ungleichungsbeschaenkungen
Aineq = ...
[1, -1, 0, 0, 0; ...
0, 0, 1, -1, 0; ...
0, 1, 0, -1.5, 0; ...
0, -1.5, 0, 1, 0];
bineq = [-20; -20; 0; 0];

% Setzen der Box-Beschraenkungen
l = [180; 240; 30; 150; 1];
u = [inf; inf; inf; inf; 10];

% Setzen der unteren Schranke als Startschaeztung
x0 = l;

```

Matlab/Querlenker_F
laeche.m

Dabei setzt die Zielfunktion den in (17.3) berechneten Flächeninhalt um:

```

function A = Querlenker_Flaeche(x)

% Bedeutung der Variablen
% x(1) = a1 = Abmessung in x-Richtung
% x(2) = a2 = Abmessung in x-Richtung
% x(3) = c1 = Abmessung in z-Richtung
% x(4) = c2 = Abmessung in z-Richtung
% x(5) = m = Formparameter
a1 = x(1);
a2 = x(2);
c1 = x(3);
c2 = x(4);
m = x(5);

% Berechnung der Flaechen
A = a1 * (c2 + c1 * m) / (m + 1) + (a2 - a1) * c2;

```

Die Funktion Querlenker_Beschraenkungen realisiert die Funktionen der nichtlinearen Gleichungs- und Ungleichungsbeschränkungen $g(x)$ bzw. $h(x)$.⁶ In unserem Beispiel haben wir nur eine nichtlineare Ungleichungsbeschränkung: $I_0 - I_x(x) \leq 0$ und keine nichtlinearen Gleichungsbeschränkungen:

Matlab/Querlenker_B
eschraenkungen.m

```

function [g,h] = Querlenker_Beschraenkungen(x,I0)

% Berechnung der Ungleichungsbeschaenkungsfunktion
Ix = Querlenker_Flaechentraegheitsmoment(x);
g = I0 - Ix;

% Berechnung der Gleichungsbeschaenkungsfunktion
h = [];

```

⁶In Anlehnung an die Dokumentation von `fmincon` wurden diese Funktionen in (17.7) mit $c_{ineq}(x)$ bzw. $c_{eq}(x)$ bezeichnet.

Die eigentliche Arbeit zur Berechnung von $I_x(x)$ wie in (17.5) angegeben erfolgt in der hier nicht abgedruckten Routine `Querlenker_Flaechentraegheitsmoment.m`, die aber einfach zu implementieren ist.

Schließlich setzen wir einige Optionen für `fmincon`, insbesondere wählen wir aus den vier verschiedenen Algorithmen das sogenannte SQP-Verfahren aus, dazu später mehr:

Matlab/Querlenker.m

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('fmincon','Algorithm','sqp','Display','iter-
    detailed');

% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda,grad] = fmincon(f,x0,Aineq,bineq
   ,[],[],l,u,gh,options);
```

Die Ausgabe von `fmincon` sieht etwa wie folgt aus:

Iter	Func-count	Fval	Feasibility	Step Length	Norm of step	First-order optimality
0	6	2.520000e+04	1.500e+01	1.000e+00	0.000e+00	5.400e+03
1	12	2.085378e+04	0.000e+00	1.000e+00	6.986e+01	3.284e+03
2	18	1.448863e+04	0.000e+00	1.000e+00	4.255e+01	3.947e+02
3	24	1.391378e+04	0.000e+00	1.000e+00	1.362e+01	3.767e+02
4	30	1.356817e+04	0.000e+00	1.000e+00	1.099e+00	2.907e+02
5	36	1.333439e+04	0.000e+00	1.000e+00	1.288e+00	1.986e+01
6	42	1.333130e+04	0.000e+00	1.000e+00	7.724e-02	4.767e-02
7	48	1.333130e+04	0.000e+00	1.000e+00	3.047e-05	2.373e-05

Optimization completed: The relative first-order optimality measure, 8.514431e-08, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric		Options	
relative first-order optimality =	8.51e-08	OptimalityTolerance =	1e-06 (default)
relative max(constraint violation) =	0.00e+00	ConstraintTolerance =	1e-06 (default)

Die Abbruchbedingung des *relative first-order optimality measure*, die hier greift, ist die ausreichende Reduktion der in der Spalte *first-order optimality* aufgeführten Größe:⁷

$$\frac{5.400 \cdot 10^3}{1.376 \cdot 10^{-5}} = 2.548 \cdot 10^{-9} < 1.000 \cdot 10^{-6}.$$

Die *first-order optimality* misst die (Nicht-)Erfülltheit bestimmter Bedingungen des zur Aufgabe gehörenden Systems an Optimalitätsbedingungen, das später noch erläutert wird.

Nach unseren bisherigen Erfahrungen mit den Routinen aus der MATLAB-Optimierungs-Toolbox können wir hier schon vermuten, dass die Optimierung erfolgreich verlaufen ist.

⁷Der von `fmincon` angegebene Zahlenwert von $4.936 \cdot 10^{-8}$ ist dabei nicht exakt nachvollziehbar.

Wir erhalten als Lösung den Vektor

$$x^* = \begin{pmatrix} 220.0000 \\ 240.0000 \\ 30.0000 \\ 183.2825 \\ 10.0000 \end{pmatrix}.$$

Wir erkennen außerdem schon aus der obigen Ausgabe, dass im Laufe der Optimierung die Querschnittsfläche des Querlenkers von $25\,200\text{ mm}^2$ auf $13\,331\text{ mm}^2$ reduziert werden konnte:

```
Querlenker_Flaeche(x0)
Querlenker_Flaeche(xstar)
```

Das Mindestträgheitsmoment $I_0 = 3 \times 10^7\text{ mm}^4$ wird in der Lösung (mit Gleichheit) eingehalten, während die Struktur am Startpunkt noch steifer als nötig war:

```
Querlenker_Flaechentraegheitsmoment(x0)
Querlenker_Flaechentraegheitsmoment(xstar)
```

Die erhaltene Geometrie (im Vergleich mit der anfänglichen) wird in [Abb. 17.2](#) gezeigt.

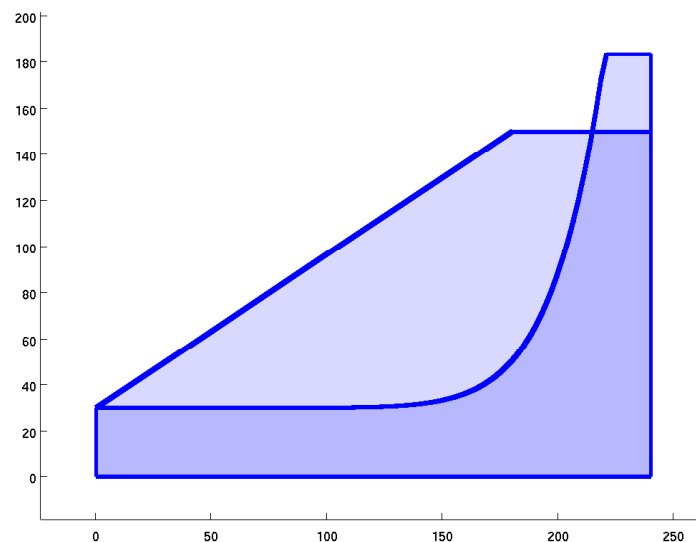


Abbildung 17.2.: Anfängliche und optimierte Form des Querlenkers aus [Beispiel 17.1](#).

Ende der Woche 13

Wie schon zuvor wollen wir auch hier überprüfen, ob dies tatsächlich eine Lösung der Aufgabe ist.

Definition 17.2 (Minimum in der beschränkten Optimierung, vgl. [Definition 2.2](#) und [Definition 11.2](#)). Wir betrachten eine beschränkte (restringierte) Optimierungsaufgabe (17.1) bzw. (17.7) mit zulässiger Menge X , siehe (17.2) bzw. (17.8).

(i) Ein Vektor $x^* \in X$ heißt **globales Minimum**, wenn gilt:

$$f(x^*) \leq f(x) \quad \text{für alle } x \in X.$$

(ii) Ein globales Minimum heißt **strikt**, wenn gilt:

$$f(x^*) < f(x) \quad \text{für alle } x \in X, \quad x \neq x^*.$$

(iii) Ein Vektor $x^* \in X$ heißt **lokales Minimum**, wenn es eine Kugel (Umgebung) $U(x^*)$ mit Mittelpunkt x^* gibt, sodass gilt:

$$f(x^*) \leq f(x) \quad \text{für alle } x \in X \cap U(x^*).$$

(iv) Ein lokales Minimum heißt **strikt**, wenn gilt:

$$f(x^*) < f(x) \quad \text{für alle } x \in X \cap U(x^*), \quad x \neq x^*.$$

Ein Minimum nennt man auch wieder **Optimum**, **Minimalstelle** oder **Lösung** von (17.1) bzw. (17.7).

[Abb. 17.3](#) illustriert den Begriff des lokalen und globalen Minimums.

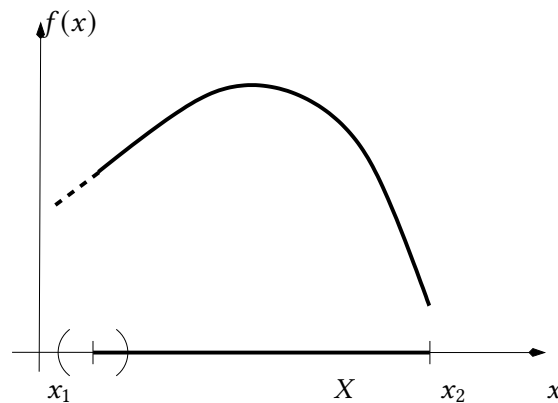


Abbildung 17.3.: Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ wird über der zulässigen Menge X (Intervall) minimiert. Der Punkt x_2 ist ihr striktes (eindeutiges) globales Minimum. Der Punkt x_1 ist ein weiteres lokales Minimum, da es in der Nähe (Umgebung) von x_1 innerhalb der zulässigen Menge X keine Punkte mit kleinerem Funktionswert gibt. Eine passende Umgebung $U(x_1)$ ist eingezeichnet. Man beachte, dass die Funktionswerte von $f(x)$ außerhalb der zulässigen Menge keine Rolle spielen.

Zum Erkennen lokaler Minima benutzt man wie bereits in allen früheren Kapiteln Optimalitätsbedingungen. Wie bereits im [Kapitel 3](#) (lineare Optimierung) sorgt die Gegenwart

von Beschränkungen dafür, dass die Optimalitätsbedingungen wesentlich komplizierter erscheinen als die Bedingungen für freie (unrestringierte) Optimierungsaufgaben. Bevor wir diese angeben, motivieren wir sie durch ein einfaches Beispiel, in dem nur eine einzige Nebenbedingung in Gleichungsform auftritt.

Beispiel 17.3 (Illustration notwendiger Optimalitätsbedingungen).

$$\begin{aligned} \text{Minimiere } f(x) &:= x_1^2 + x_2^2, \quad x = (x_1, x_2) \in \mathbb{R}^2 \\ \text{sodass } h(x) &:= x_2 - (x_1 + 1)^2 - 1 = 0. \end{aligned} \quad (17.9)$$

Die zulässige Menge besteht hier also aus dem Graphen einer Parabel:

$$X = \{x = (x_1, x_2) \in \mathbb{R}^2 : x_2 = (x_1 + 1)^2 + 1\}.$$

Diese ist in [Abb. 17.4](#) dargestellt, zusammen mit einigen Höhenlinien der Zielfunktion $f(x) = x_1^2 + x_2^2$.

Wir erkennen in diesem Beispiel grafisch die Lage des (hier eindeutigen und damit globalen) Minimums x^* . An dieser Stelle liegen offenbar die Gradienten der Zielfunktion $\nabla f(x^*)$ (blauer Vektor) und der Beschränkungsfunktion $\nabla h(x^*)$ (schwarzer Vektor) parallel. Mit anderen Worten, beide Vektoren sind linear abhängig, d. h., es gibt eine Zahl λ , sodass gilt:

$$\nabla f(x^*) + \lambda \nabla h(x^*) = 0. \quad (17.10)$$

Diese Beobachtung ist kein Zufall, sondern sie gilt (unter bestimmten Voraussetzungen, siehe [Satz 17.4](#)) für jedes lokale Minimum nichtlinearer Optimierungsaufgaben ([17.1](#)). Die Zahl λ heißt der zum Minimum x^* gehörende **Lagrange-Multiplikator** für die Beschränkung $h(x^*) = 0$.

Die Bedingung ([17.10](#)) lässt sich bequem mit Hilfe der **Lagrange-Funktion** zur Aufgabe ([17.9](#)) formulieren:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda h(x),$$

denn ([17.10](#)) ist nichts anderes als die Bedingung

$$\nabla_x \mathcal{L}(x^*, \lambda) = \nabla f(x^*) + \lambda \nabla h(x^*) = 0.$$

Für Optimierungsaufgaben der Gestalt ([17.1](#)) mit mehreren Gleichungen und Ungleichungen gilt i. W. dieselbe Beobachtung: Auch hier lässt sich in einem lokalen Minimum x^* der Gradient der Zielfunktion zusammen mit den Gradienten der Nebenbedingungen linear kombinieren, sodass sich der Nullvektor ergibt. Dies kann man wieder bequem mit Hilfe der **Lagrange-Funktion** formulieren. Für die Aufgabe ([17.1](#)) lautet diese:

$$\begin{aligned} \mathcal{L}(x, \mu, \lambda) &:= f(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^p \lambda_j h_j(x) \\ &= f(x) + \mu^T g(x) + \lambda^T h(x). \end{aligned} \quad (17.11)$$

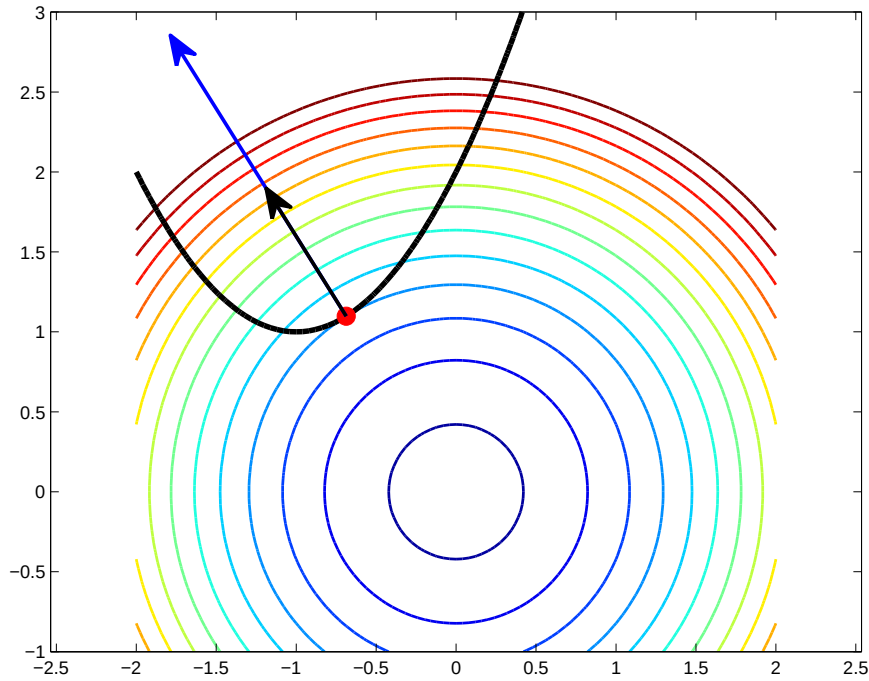


Abbildung 17.4.: Einige Höhenlinien der Zielfunktion $f(x)$ und die zulässige Menge (in schwarz), beschrieben durch $h(x) = 0$, am Beispiel der Aufgabe (17.9). In der Lösung x^* liegen die Gradienten der Zielfunktion $\nabla f(x^*)$ (blauer Vektor) und der Beschränkungsfunktion $\nabla h(x^*)$ (schwarzer Vektor) parallel.

Die Vektoren μ und λ nennt man **Lagrange-Multiplikatoren**, und sie haben ebenso viele Komponenten, wie es Ungleichungs- bzw. Gleichungsnebenbedingungen in der Aufgabe gibt.

Die Linearkombination zum Nullvektor lässt sich nun schreiben als

$$\begin{aligned}\nabla_x \mathcal{L}(x, \mu, \lambda) &:= \nabla f(x) + \sum_{i=1}^m \mu_i \nabla g_i(x) + \sum_{j=1}^p \lambda_j \nabla h_j(x) \\ &= \nabla f(x) + g'(x)^\top \mu + h'(x)^\top \lambda = 0.\end{aligned}\tag{17.12a}$$

Dabei sind $g'(x)$ und $h'(x)$ die Jacobimatrizen von $g(x)$ bzw. $h(x)$. Man kann sich an einem Beispiel (vgl. Abb. 17.5) überlegen, dass bei dieser Linearkombination nur diejenigen Ungleichungen $g_i(x) \leq 0$ eine Rolle spielen können, die an der Stelle x **aktiv** sind:

$$\mathcal{A}_g(x) = \{i = 1, \dots, m : g_i(x) = 0\}.$$

Die Gradienten $\nabla g_i(x)$ der an der Stelle x **inaktiven** Ungleichungen

$$\mathcal{I}_g(x) = \{i = 1, \dots, m : g_i(x) < 0\}$$

schaltet man in der Linearkombination (17.12a) dadurch aus, dass gefordert wird, dass die zugehörigen Lagrange-Multiplikatoren $\mu_i = 0$ sind für alle Indizes $i \in \mathcal{I}_g(x)$. Aus Kapitel 3 kennen wir außerdem schon die Forderung, dass Lagrange-Multiplikatoren zu Ungleichungen

grundsätzlich nicht-negativ sein müssen. Diese Forderungen (zusammen mit der Zulässigkeit des Vektors x bzgl. aller Ungleichungen) formulieren wir als

$$g_i(x) \leq 0, \quad \mu_i \geq 0, \quad \mu_i g_i(x) = 0 \quad \text{für alle } i = 1, \dots, m \quad (17.12b)$$

oder äquivalent als

$$g(x) \leq 0, \quad \mu \geq 0, \quad \mu^\top g(x) = 0. \quad (17.12b')$$

Die Ungleichungsbedingungen erfüllen also zusammen mit dem zugehörigen Multiplikator wieder ein **Komplementaritätssystem**, wie wir es schon aus [Kapitel 3](#) kennen, vgl. (11.5). Schließlich gilt natürlich auch

$$h_j(x) = 0 \quad \text{für alle } j = 1, \dots, p \quad (17.12c)$$

bzw. vektorwertig

$$h(x) = 0. \quad (17.12c')$$

Das System (17.12) nennt man die **Karush-Kuhn-Tucker-Bedingungen**, kurz: **KKT-Bedingungen** der Aufgabe (17.1). Sie stimmen mit den schon aus [Kapitel 3](#) bekannten KKT-Bedingungen (11.5) überein, falls (17.1) eine *lineare* Optimierungsaufgabe ist.

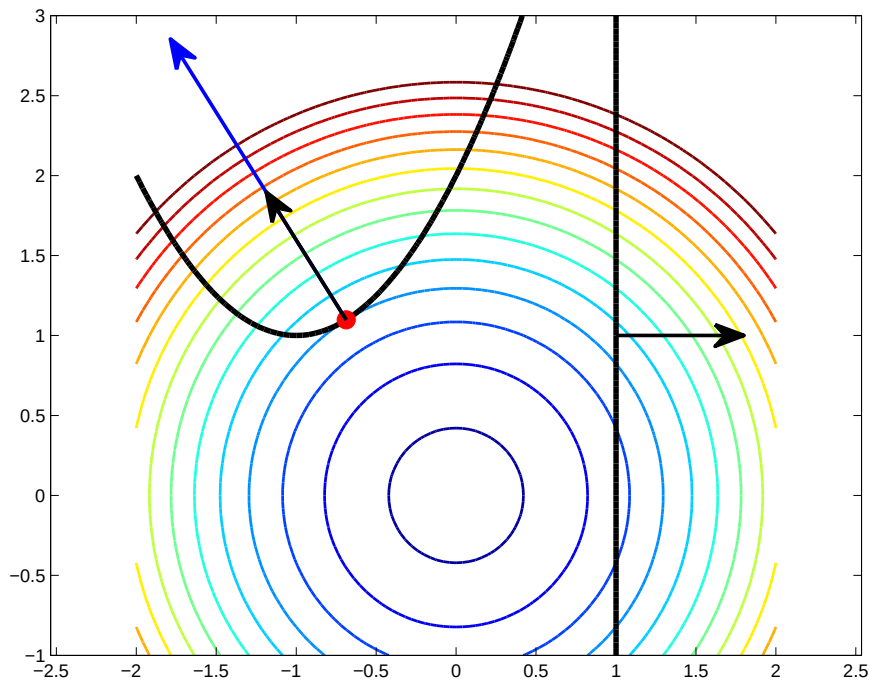


Abbildung 17.5.: Erweiterung von [Beispiel 17.3](#) um die zusätzliche Ungleichungsbeschränkung $g(x) := x_1 - 1 \leq 0$, die jedoch in der Lösung x^* inaktiv ist.

Wir können nun Optimalitätsbedingungen für die Aufgabe (17.1) formulieren.

Satz 17.4 (Notwendige Optimalitätsbedingung 1. Ordnung, vgl. [Satz 11.3](#)). *Es sei x^* ein lokales Minimum von (17.1). Außerdem nehmen wir an, dass an der Stelle x^* eine gewisse **Constraint Qualification** erfüllt ist. Unter dieser Voraussetzung kann man passende Vektoren (**Lagrange-Multiplikatoren**)*

$$\mu \in \mathbb{R}^m, \quad \lambda \in \mathbb{R}^p$$

finden, sodass das System (17.12) aus Gleichungen und Ungleichungen (die sogenannten **Karush-Kuhn-Tucker-Bedingungen**, kurz: **KKT-Bedingungen**) erfüllt ist, hier nochmals zusammengefasst:

$$\left. \begin{aligned} \nabla_x \mathcal{L}(x^*, \mu, \lambda) &= \nabla f(x^*) + g'(x^*)^\top \mu + h'(x^*)^\top \lambda = 0, \\ \mu &\geq 0, \quad g(x^*) \leq 0, \quad \mu^\top g(x^*) = 0, \\ h(x^*) &= 0. \end{aligned} \right\} \quad (17.13)$$

Ein Tripel (x^*, μ, λ) , das (17.13) erfüllt, heißt auch ein **KKT-Punkt** der Aufgabe (17.1).

Bemerkung 17.5 (zu den Optimalitätsbedingungen aus Satz 17.4).

- (i) Die KKT-Bedingungen (17.13) enthalten als Spezialfall diejenigen Bedingungen für lineare Optimierungsaufgaben (11.5). Dass wir dort, also in Satz 11.3, keine Constraint Qualifications fordern mussten, liegt an der Linearität der Nebenbedingungen.
- (ii) Im Unterschied zur linearen Optimierung sind die KKT-Bedingungen bei nichtlinearen Aufgaben jedoch lediglich notwendige (keine hinreichenden) Optimalitätsbedingungen. Die Umkehrung von Satz 17.4 gilt leider nicht, d. h., es kann Vektoren (x^*, μ, λ) geben, die die KKT-Bedingungen (17.13) erfüllen, und dennoch ist x^* kein lokales Minimum für (17.1), sondern z. B. ein lokales Maximum oder ein Sattelpunkt.

Ausnahme: Falls die Optimierungsaufgabe (17.1) konvex ist, dann gilt auch die Umkehrung von Satz 17.4. Dazu später mehr in Satz 17.10.

- (iii) Für nicht-konvexe Optimierungsaufgaben kann man⁸ mit Hilfe von notwendigen bzw. hinreichenden Optimalitätsbedingungen 2. Ordnung weiter unterscheiden, ob ein KKT-Punkt tatsächlich ein (lokales) Minimum ist oder auch nicht (sondern z. B. ein lokales Maximum oder Sattelpunkt). Solche Bedingungen sind aber nicht ganz einfach zu formulieren und auch numerisch nicht ganz einfach zu überprüfen. Wir verzichten daher hier darauf, Optimalitätsbedingungen 2. Ordnung für nichtlineare Optimierungsaufgaben anzugeben.⁹
- (iv) Die KKT-Bedingungen (17.13) können wir leicht auf nichtlineare Optimierungsaufgaben in der von fmincon behandelten Form (17.7) übertragen: Die Lagrange-Multiplikatoren sind dann Vektoren

$$\lambda_{ceq}, \quad \lambda_{cineq}, \quad \lambda_{Aeq}, \quad \lambda_{Aineq}, \quad \lambda_\ell, \quad \lambda_u$$

mit jeweils sovielen Einträgen, wie es Gleichungs- bzw. Ungleichungsbeschränkungen des betreffenden Typs in (17.7) gibt.

Die Lagrange-Funktion (17.11) hat dann die Gestalt

$$\begin{aligned} \mathcal{L}(x, \lambda_{ceq}, \lambda_{cineq}, \lambda_{Aeq}, \lambda_{Aineq}, \lambda_\ell, \lambda_u) \\ := f(x) + \lambda_{ceq}^\top c_{eq}(x) &+ \lambda_{cineq}^\top c_{ineq}(x) \\ &+ \lambda_{Aeq}^\top (A_{eq} x - b_{eq}) + \lambda_{Aineq}^\top (A_{ineq} x - b_{ineq}) \\ &+ \lambda_\ell^\top (\ell - x) &+ \lambda_u^\top (x - u). \end{aligned} \quad (17.14)$$

⁸wie bereits in Kapitel 1 bei Optimierungsaufgaben ohne Beschränkungen gesehen, vgl. Satz 2.4 und Satz 2.5

⁹Wer daran interessiert ist, findet Optimalitätsbedingungen 2. Ordnung z. B. in Nocedal, Wright, 2006, Chapter 12.5.

Die KKT-Bedingungen für (17.7) lauten:

$$\left. \begin{aligned}
 & \nabla f(x^*) + c'_{eq}(x^*)^\top \lambda_{ceq} + c'_{ineq}(x^*)^\top \lambda_{cineq} \\
 & \quad + A_{eq}^\top \lambda_{Aeq} \quad + A_{ineq}^\top \lambda_{Aineq} \\
 & \quad - \lambda_\ell \quad + \lambda_u = 0, \\
 & \lambda_{Aineq} \geq 0, \quad A_{ineq} x^* - b_{ineq} \leq 0, \quad \lambda_{Aineq}^\top (A_{ineq} x^* - b_{ineq}) = 0, \\
 & \lambda_{cineq} \geq 0, \quad c_{ineq}(x^*) \leq 0, \quad \lambda_{cineq}^\top c_{ineq}(x^*) = 0, \\
 & \lambda_\ell \geq 0, \quad \ell - x^* \leq 0, \quad \lambda_\ell^\top (\ell - x^*) = 0, \\
 & \lambda_u \geq 0, \quad x^* - u \leq 0, \quad \lambda_u^\top (x^* - u) = 0, \\
 & c_{eq}(x^*) = 0, \\
 & A_{eq} x^* - b_{eq} = 0.
 \end{aligned} \right\} \quad (17.15)$$

Beachte: Auch hier ist die linke Seite in der ersten Zeile in (17.15) der Gradient der Lagrange-Funktion bzgl. der Optimierungsvariablen x .¹⁰

Bevor wir den Begriff der **Constraint Qualification** (dt.: *Regularitätsbedingung für die Beschränkungen*) näher erläutern, wollen wir überprüfen, ob `fmincon` bei unserem Beispiel (17.3) tatsächlich einen KKT-Punkt gefunden hat. (Wie praktisch alle Verfahren der nichtlinearen Optimierung sucht auch `fmincon` nach einem KKT-Punkt, obwohl dieser nicht notwendig ein lokales Minimum ist.) Die Aufgabe besaß fünf Optimierungsvariablen $x = (a_1, a_2, c_1, c_2, m)$, eine nichtlineare Ungleichungsbeschränkung $I_0 - I_x(x) \leq 0$, vier lineare Ungleichungsbeschränkungen, zusammengefasst in den vier Zeilen von $A_{ineq} x \leq b_{ineq}$, sowie Schranken für die Variablen der Form $\ell \leq x \leq u$. Es traten keine Gleichungsbeschränkungen auf, weder lineare $A_{eq} x - b_{eq} = 0$ noch nichtlineare $c_{eq}(x) = 0$.

Neben x^* besteht ein KKT-Punkt für (17.6) also aus den Lagrange-Multiplikatoren

$$\lambda_{cineq} \in \mathbb{R}, \quad \lambda_{Aineq} \in \mathbb{R}^4, \quad \lambda_\ell \in \mathbb{R}^5, \quad \lambda_u \in \mathbb{R}^5,$$

wohingegen λ_{ceq} und λ_{Aeq} nicht auftreten. Alle Lagrange-Multiplikatoren liefert `fmincon` in der `MATLAB-struct` `lambda` zurück, die wir beim Aufruf von `fmincon` mit angefordert hatten:

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('fmincon','Algorithm','sqp','Display','iter-
    detailed');

% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda,grad] = fmincon(f,x0,Aineq,bineq
   ,[],[],l,u,gh,options);
```

¹⁰Laut Dokumentation von `fmincon` beinhaltet das *first-order optimality measure* die Norm von $\nabla_x \mathcal{L}(x, \lambda_{ceq}, \lambda_{cineq}, \lambda_{Aeq}, \lambda_{Aineq}, \lambda_\ell, \lambda_u)$ (siehe erste Zeile von (17.15)) sowie die Norm der vier Terme $\lambda_{Aineq}^\top (A_{ineq} x^* - b_{ineq})$, $\lambda_{cineq}^\top c_{ineq}(x^*)$, $\lambda_\ell^\top (\ell - x^*)$ und $\lambda_u^\top (x^* - u)$. Eine weitere Bedingung für den Stop von `fmincon` ist die Größe *feasibility*, die die Erfüllung der vier Ungleichungsnebenbedingungen und zwei Gleichungsnebenbedingungen der Aufgabe (17.7) misst.

Dabei gelten folgende Variablennamen:

$$\begin{aligned} \text{lambda.ineqnonlin} &\hat{=} \lambda_{\text{cineq}}, & \text{lambda.ineqlin} &\hat{=} \lambda_{\text{Aineq}}, & \text{lambda.lower} &\hat{=} \lambda_{\ell}, \\ \text{lambda.eqnonlin} &\hat{=} \lambda_{\text{ceq}}, & \text{lambda.eqlin} &\hat{=} \lambda_{\text{Aeq}}, & \text{lambda.upper} &\hat{=} \lambda_u. \end{aligned}$$

Zur Überprüfung, ob $(x^*, \lambda_{\text{cineq}}, \lambda_{\text{Aineq}}, \lambda_{\ell}, \lambda_u)$ die KKT-Bedingungen (17.15) erfüllen, müssen wir außerdem noch kennen:

- den Gradienten der Zielfunktion $\nabla f(x^*) \in \mathbb{R}^5$
- die Jacobimatrix der Ungleichungsbeschränkung $c'_{\text{ineq}}(x^*) \in \mathbb{R}^{1 \times 5}$.

Den Gradienten der Zielfunktion $\nabla f(x^*)$ im gefundenen Lösungspunkt x^* liefert uns `fmincon` zurück, und zwar in der Variablen `grad`. Aus unerfindlichen Gründen gibt es aber keine Möglichkeit, die Jacobimatrix der Beschränkungen $c'_{\text{ineq}}(x^*)$ oder auch $c'_{\text{eq}}(x^*)$ (bei Aufgaben mit nichtlinearen Gleichungsbeschränkungen) zurückgeliefert zu bekommen. Wir behelfen uns daher hier mit einer Approximation von $c'_{\text{ineq}}(x^*)$ mit Hilfe finiter Differenzen:¹¹ Wie in (4.1) müssen wir dafür berechnen:

$$c'_{\text{ineq}}(x^*) \approx \left(\frac{c_{\text{ineq}}(x^* + h e_1) - c_{\text{ineq}}(x^*)}{h} \quad \dots \quad \frac{c_{\text{ineq}}(x^* + h e_5) - c_{\text{ineq}}(x^*)}{h} \right). \quad (17.16)$$

Die (approximative) Berechnung dieser Jacobimatrix, die wir in MATLAB `Jacineq` nennen wollen, erledigen wir wie folgt:

Matlab/Querlenker_K
KT_pruefen.m

```
% Setzen der Schrittweite fuer die Differenzenapproximation
h = 1e-4;
% Vorbereiten der zu berechnenden Matrix Jacineq
Jacineq = zeros(1,5);
% Vorbereiten der Einheitsvektoren
E = eye(5);
% Auswertung der Beschraenkungen einmal an der Stelle xstar
[gstar,~] = gh(xstar);
% Berechnung von Jacineq Spalte fuer Spalte
for j = 1:5
    e = E(:,j);
    [g,~] = gh(xstar + h * e);
    Jacineq(:,j) = (g - gstar) / h;
end
```

Nun sind wir in der Lage, die KKT-Bedingungen (17.15) Schritt für Schritt zu überprüfen.¹² Zunächst die erste Zeile (Gradient der Lagrange-Funktion):

Matlab/Querlenker_K
KT_pruefen.m

```
gradL = grad + Jacineq' * lambda.ineqnonlin + Aineq' * lambda.ineqlin
        - lambda.lower + lambda.upper
```

¹¹Auch `fmincon` verwendet ja intern finite Differenzen, da wir die Ableitung der Beschränkungsfunktion $c_{\text{ineq}}(x)$ wie auch der Zielfunktion $f(x)$ nicht programmiert haben.

¹²Wegen der fehlenden Gleichungsbeschränkungen ignorieren wir in (17.15) alle Terme mit $c_{\text{eq}}(x^*)$, λ_{ceq} , A_{eq} , b_{eq} und λ_{Aeq} .

ergibt den Vektor

$$10^{-3} \begin{pmatrix} 0.0386 \\ 0.0441 \\ 0.0242 \\ -0.0074 \\ -0.3853 \end{pmatrix},$$

was ungefähr der Nullvektor ist. Die Euklidische Norm (Länge) dieses Vektors beträgt $3.9053 \cdot 10^{-4}$. Zum Überprüfen der zweiten Zeile lassen wir uns ausgeben:

```
[lambda.ineqlin, Aineq * xstar - bineq]
```

und erhalten als Antwort

$$\begin{pmatrix} 72.7983 & 0 \\ 0 & -133.2825 \\ 0 & -34.9237 \\ 0 & -176.7175 \end{pmatrix}.$$

Die erste der vier linearen Ungleichungsbedingungen¹³ (das ist $a_2 \geq a_1 + 20$) ist also im Punkt x^* aktiv, die restlichen drei sind inaktiv, und das Komplementaritätssystem ist erfüllt. Analog überprüfen wir die dritte Zeile von (17.15), die sich auf die nichtlineare Ungleichungsbedingung $c_{\text{ineq}}(x) \leq 0$ bezieht:

```
[lambda.ineqnonlin, gstar]
```

Dies liefert die Antwort

$$10^{-4} \cdot (0.8412 \quad -0.0256).$$

Hier ist also die Bedingung $\text{lambda.ineqnonlin} * \text{gstar} = \lambda_{\text{cineq}} c_{\text{ineq}}(x^*) = 0$ nur näherungsweise erfüllt.

Überprüfen wir nun noch die beiden Komplementaritätssysteme, die zu den unteren und oberen Schranken $\ell \leq x \leq u$ gehören:

```
[lambda.lower, l - xstar]
```

ergibt

$$\begin{pmatrix} 0 & -40.0000 \\ 38.9652 & 0 \\ 192.0090 & 0 \\ 0 & -33.2825 \\ 0 & -9.0000 \end{pmatrix}.$$

Zwei der fünf Geometriegrößen schlagen also jeweils an ihrer unteren Grenze an: $a_2^* = 240$ und $c_1^* = 30$. Für die oberen Schranken überprüfen wir:

```
[lambda.upper, xstar - u]
```

¹³Dies waren Einschränkungen an die Geometrie des Querlenkers.

ergibt

$$\begin{pmatrix} 0 & -\infty \\ 0 & -\infty \\ 0 & -\infty \\ 0 & -\infty \\ 238.0771 & 0 \end{pmatrix}.$$

Die fünfte Optimierungsvariable nimmt also in der Lösung ihre obere Grenze an: $m^* = 10$. Da keine Gleichungsbeschränkungen vorliegen, ist die Überprüfung von (17.15) hier beendet.

Bemerkung 17.6 (zum Beispiel 17.1).

- (i) Wir haben also festgestellt, dass `fmincon` für unser Beispiel 17.1 mit der Aufgabenformulierung (17.6) tatsächlich einen Punkt x^* zusammen mit zugehörigen Lagrange-Multiplikatoren $(\lambda_{\text{cineq}}, \lambda_{\text{Aineq}}, \lambda_\ell, \lambda_u)$ berechnet hat, die zusammen (wie immer bis auf gewisse Toleranzen) einen KKT-Punkt ergeben.

Damit ist aber noch nicht gezeigt, dass x^* ein lokales Minimum von (17.6) ist. Genauso gut könnten wir ein Maximum oder einen Sattelpunkt erhalten haben.¹⁴

- (ii) Die Lagrange-Multiplikatoren in einem lokalen Minimum x^* sind nicht nur schmückendes Beiwerk, sondern haben eine Bedeutung. Dies hatten wir bereits bei linearen Optimierungsaufgaben in § 13 festgestellt. Die Multiplikatoren geben (unter bestimmten Bedingungen) an, wie sich der Wert der Zielfunktion $f(x^*)$ ändert, wenn man die zum jeweiligen Multiplikator gehörende Beschränkung „verschiebt“, also beispielsweise statt $g_i(x) \leq 0$ dann $g_i(x) \leq \Delta$ fordert. Der Wert der Zielfunktion am neuen Minimum $x^*(\Delta)$ beträgt dann ungefähr

$$f(x^*) - \mu_i \Delta,$$

wobei μ_i der zur Ungleichung $g_i(x) \leq 0$ gehörende Multiplikator am alten Minimum $x^* = x^*(0)$ ist. Dieselbe Interpretation gilt auch bei Gleichungsbeschränkungen. Wie bereits in § 13 erwähnt, ist sie die Grund dafür, warum bei wirtschaftswissenschaftlichen Aufgabenstellungen die Lagrange-Multiplikatoren auch als **Schattenpreise** bezeichnet werden.

Ende der Woche 14

Um nun den Satz 17.4 zu verstehen, müssen wir noch den Begriff der **Constraint Qualification** (dt.: *Regularitätsbedingung für die Beschränkungen*) näher erläutern. Wir nehmen vorweg, dass es verschiedene Constraint Qualifications gibt, die aber alle denselben Zweck erfüllen. Wir nennen hier nur eine solche Bedingung, die besonders einfach zu handhaben ist:

Definition 17.7 (Constraint Qualification der linearen Unabhängigkeit (LICQ)). Es sei $\bar{x}$ zulässiger Punkt des Optimierungsproblems (17.1) und $\mathcal{A}_g(\bar{x})$ die Menge der dort aktiven Ungleichungen. Man sagt: $\bar{x}$ erfüllt die **Regularitätsbedingungen der linearen Unabhängigkeit** (engl.: **linear independence constraint qualification**, kurz: **LICQ**), wenn gilt: Die Gradienten der Nebenbedingungen $\{\nabla h_j(\bar{x})\}_{j=1}^p \cup \{\nabla g_i(\bar{x})\}_{i \in \mathcal{A}_g(\bar{x})}$ sind linear unabhängig.

¹⁴Verfahren der nichtlinearen Optimierung wie `fmincon` unternehmen jedoch einiges, um dies möglichst zu verhindern.

Beachte: Von den Ungleichungsbedingungen sind nur diejenigen relevant, die im Punkt $\bar{x}$ aktiv sind, für die also $g_i(\bar{x}) = 0$ gilt.

Im berechneten Lösungspunkt x^* von [Beispiel 17.1](#) ist LICQ erfüllt, nicht aber im folgenden [Beispiel 17.8](#). Um dies für [Beispiel 17.1](#) numerisch zu überprüfen, können wir wie folgt vorgehen: Zunächst prüfen wir, welche der Ungleichungen $A_{\text{ineq}} x^* - b_{\text{ineq}} \leq 0$ aktiv sind. Da es aufgrund numerischer Toleranzen unsauber ist, hierin nach Einträgen „= 0“ zu suchen, nutzen wir stattdessen die Komplementaritätsbedingung, vgl. (17.15). Demnach zeigen positive Komponenten im Multiplikator λ_{Aineq} (in MATLAB `lambda.ineqlin` genannt) aktive Ungleichungen an:¹⁵

```
% Bestimmen der aktiven Ungleichungen Aineq * xstar <= bineq
ix = find(lambda.ineqlin > 0);
M = Aineq(ix,:);
```

Die zweite Zeile sorgt dafür, dass die Gradienten, die zu den aktiven Ungleichungen gehören, als Zeilenvektoren in der Matrix M gespeichert werden. Genauso verfahren wir nun mit der einen nichtlinearen Ungleichungsbeschränkung und hängen ggf. deren Gradienten als weitere Zeile unten an die Matrix M an:

```
% Bestimmen der aktiven Ungleichungen cineq(xstar) <= 0
ix = find(lambda.ineqnonlin > 0);
M = [M; Jacineq(ix,:)];
```

Und schließlich fügen wir die Gradienten der aktiven unteren und oberen Box-Beschränkungen hinzu:

```
% Bestimmen der aktiven unteren Schranken u - xstar <= 0
ix = find(lambda.lower > 0);
M = [M; -E(ix,:)];

% Bestimmen der aktiven oberen Schranken xstar - l <= 0
ix = find(lambda.upper > 0);
M = [M; E(ix,:)];
```

Enthielte unsere Aufgabe (17.1) noch Gleichungsnebenbedingungen, so müssten wir auch deren Gradienten noch hinzufügen. Die erhaltene Matrix M hat fünf Zeilen (aktive Ungleichungen) und fünf Spalten (da $x \in \mathbb{R}^5$). Wir überprüfen ihren Rang:

```
% Bestimmen des Ranges von M
rank(M)
```

Der Rang ist gleich fünf, d. h., die fünf Zeilen von M (also die Gradienten der an der Stelle x^* aktiven Ungleichungsnebenbedingungen) sind linear unabhängig. Mit anderen Worten: Die LICQ ist hier tatsächlich erfüllt.

¹⁵Ganz korrekt ist unser Vorgehen dabei auch nicht, denn es könnte aktive Ungleichungsbeschränkungen geben, deren Multiplikator exakt null ist (sogenannte schwach aktive Beschränkungen). Diese würden wir dann übersehen. Allerdings ist eine solche Situation numerisch unwahrscheinlich.

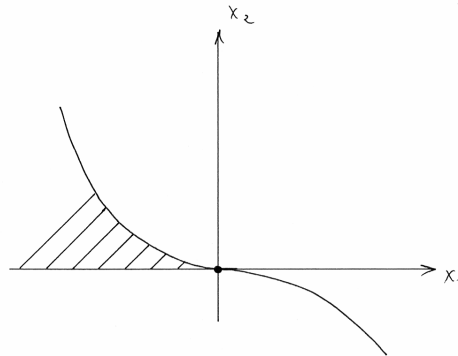
Wir wollen die Bedeutung von Constraint Qualifications noch einmal klar herausstellen: Constraint Qualifications sichern, dass ein lokales Minimum x^* einer nichtlinearen Optimierungsaufgabe (17.1) zu einem KKT-Punkt (x^*, μ, λ) erweitert werden kann. Mit anderen Worten: Constraint Qualifications sichern die Existenz Lagrangescher Multiplikatoren.¹⁶ Das ist wichtig, weil praktisch alle Verfahren zur Lösung nichtlinearer Optimierungsaufgaben nach KKT-Punkten suchen. Lokale Minima, für die keine Constraint Qualification erfüllt sind, können dabei übersehen werden.

Im folgenden Beispiel ist die LICQ verletzt (und auch alle anderen Constraint Qualifications, die hier nicht besprochen werden), und es existieren auch tatsächlich am globalen Minimum keine Lagrange-Multiplikatoren.

Beispiel 17.8 (Wichtigkeit von Constraint Qualifications). *Wir betrachten die Optimierungsaufgabe*

$$\begin{aligned} \text{Minimiere} \quad & -x_1, \quad x = (x_1, x_2) \in \mathbb{R}^2 \\ \text{sodass} \quad & g_1(x) := x_2 + x_1^3 \leq 0 \\ \text{und} \quad & g_2(x) := -x_2 \leq 0. \end{aligned} \tag{17.17}$$

Die zulässige Menge lässt sich wie folgt veranschaulichen:



Offenbar ist $x^* = (0, 0)^T$ eindeutiges (globales) Minimum von (17.17). Die Ungleichungen $g_1(x)$ und $g_2(x)$ sind beide aktiv im Punkt x^* , und es gilt

$$\begin{aligned} \nabla g_1(x) &= \begin{pmatrix} 3x_1^2 \\ 1 \end{pmatrix} & \Rightarrow & \nabla g_1(x^*) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \nabla g_2(x) &= \begin{pmatrix} 0 \\ -1 \end{pmatrix} & \Rightarrow & \nabla g_2(x^*) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}. \end{aligned}$$

Diese Vektoren sind jedoch linear abhängig, denn $\nabla g_1(x^*) + \nabla g_2(x^*) = 0$. Die LICQ ist also nicht erfüllt.

Wir überprüfen, ob zum Minimum x^* trotzdem Lagrange-Multiplikatoren $\mu = (\mu_1, \mu_2)$ existieren.

¹⁶Unter LICQ sind die zu einem Minimum x^* gehörenden Lagrange-Multiplikatoren sogar eindeutig.

Die Lagrange-Funktion und ihr Gradient bzgl. der Optimierungsvariablen lauten

$$\mathcal{L}(x, \mu) = -x_1 + \mu_1 (x_2 + x_1^3) + \mu_2 (-x_2)$$

$$\nabla_x \mathcal{L}(x, \mu) = \begin{pmatrix} -1 + 3x_1^2 \mu_1 \\ \mu_1 - \mu_2 \end{pmatrix}.$$

Aus den KKT-Bedingungen (17.13) erhalten wir Bedingungen an die Lagrange-Multiplikatoren, nämlich $\mu_1 \geq 0$, $\mu_2 \geq 0$ und

$$\Rightarrow \nabla_x \mathcal{L}(x^*, \mu) = \begin{pmatrix} -1 \\ \mu_1 - \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

was natürlich nicht erfüllbar ist. Folglich kann man das (einzige) Minimum x^* der Aufgabe (17.17) nicht finden, indem man nach KKT-Punkten sucht.

Zusatzfrage: Kann man die Zielfunktion evtl. durch eine andere ersetzen, sodass der gleiche Punkt immer noch eine Optimallösung ist und es dafür nun Lagrange-Multiplikatoren gibt?

Wir wollen nun überprüfen, wie sich fmincon bei Beispiel 17.8 verhält. Wir setzen die Problemformulierung:

```
% Setzen der Zielfunktion
f = @(x) -x(1);

% Setzen der beiden Ungleichungsbeschränkungen
% und der (leeren) Gleichungsbeschränkung
g = @(x) [x(2)+x(1)^3; -x(2)];
h = @(x) [];
gh = @(x) deal(g(x),h(x));

% Setzen der Startschätzung
x0 = [1;1];
```

und rufen fmincon auf:

```
% Setzen der Optionen fuer das Optimierungsverfahren
options = optimoptions('fmincon','Algorithm','sqp','Display','iter-
detailed');

% Aufruf des Optimierungsverfahrens
[xstar,fstar,exitflag,output,lambda,grad] = fmincon(f,x0
,[],[],[],[],[],[],gh,options);
```

Es ergibt sich die folgende Ausgabe:

Iter	Func-count	Fval	Feasibility	Step Length	Norm of step	First-order optimality
0	3	-1.000000e+00	2.000e+00	1.000e+00	0.000e+00	1.000e+00
1	6	-4.999970e-01	6.250e-01	1.000e+00	7.071e-01	6.250e-01

2	9	-3.333313e-01	3.704e-02	1.000e+00	5.270e-01	4.630e-01
3	12	-2.222209e-01	1.097e-02	1.000e+00	1.111e-01	2.390e-01
4	15	-1.481473e-01	3.251e-03	1.000e+00	7.407e-02	2.737e-01
5	18	-9.876485e-02	9.634e-04	1.000e+00	4.938e-02	2.866e-01
6	21	-6.584324e-02	2.855e-04	1.000e+00	3.292e-02	2.913e-01
7	24	-4.389550e-02	8.458e-05	1.000e+00	2.195e-02	2.931e-01
8	27	-2.926367e-02	2.506e-05	1.000e+00	1.463e-02	2.937e-01
9	30	-1.950912e-02	7.425e-06	1.000e+00	9.755e-03	2.940e-01
10	33	-1.300608e-02	2.200e-06	1.000e+00	6.503e-03	2.941e-01
11	36	-8.670728e-03	6.519e-07	1.000e+00	4.335e-03	2.941e-01
12	39	-5.780490e-03	1.931e-07	1.000e+00	2.890e-03	2.941e-01
13	42	-3.853665e-03	5.723e-08	1.000e+00	1.927e-03	2.941e-01
14	45	-2.569115e-03	1.696e-08	1.000e+00	1.285e-03	2.941e-01
15	48	-1.712748e-03	5.024e-09	1.000e+00	8.564e-04	2.941e-01
16	51	-1.141837e-03	1.489e-09	1.000e+00	5.709e-04	2.941e-01
17	54	-7.612297e-04	4.411e-10	1.000e+00	3.806e-04	2.941e-01
18	57	-5.074915e-04	1.307e-10	1.000e+00	2.537e-04	2.941e-01
19	60	-3.383326e-04	3.873e-11	1.000e+00	1.692e-04	2.941e-01
20	63	-2.255600e-04	1.148e-11	1.000e+00	1.128e-04	2.941e-01
21	66	-1.503783e-04	3.401e-12	1.000e+00	7.518e-05	2.941e-01
22	69	-1.002572e-04	1.008e-12	1.000e+00	5.012e-05	2.940e-01
23	72	-6.684309e-05	2.987e-13	1.000e+00	3.341e-05	2.940e-01
24	75	-4.456702e-05	8.852e-14	1.000e+00	2.228e-05	2.939e-01
25	78	-2.971632e-05	2.624e-14	1.000e+00	1.485e-05	2.939e-01
26	81	-1.981584e-05	7.781e-15	1.000e+00	9.900e-06	2.937e-01
27	84	-1.321553e-05	2.308e-15	1.000e+00	6.600e-06	2.935e-01
28	87	-8.815315e-06	6.850e-16	1.000e+00	4.400e-06	2.932e-01
29	90	-5.881839e-06	2.035e-16	1.000e+00	2.933e-06	2.928e-01
Iter	Func-count	Fval	Feasibility	Step Length	Norm of step	First-order optimality
30	92	-5.881839e-06	2.035e-16	4.900e-01	9.583e-07	3.538e-07

Optimization completed: The relative first-order optimality measure, 3.537638e-07, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 1.017444e-16, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric	Options
relative first-order optimality = 3.54e-07	OptimalityTolerance = 1e-06 (default)
relative max(constraint violation) = 1.02e-16	ConstraintTolerance = 1e-06 (default)

Überraschenderweise erhalten wir keine Fehlermeldung, sondern `fmincon` stoppt mit `exitflag == 1`, also First order optimality conditions satisfied. Ob damit ein KKT-Punkt gefunden wurde, werden wir nun überprüfen: `xstar` ist mit

$$x^* = 10^{-5} \begin{pmatrix} 0.5882 \\ 0.0000 \end{pmatrix}$$

tatsächlich nahe der erwarteten Lösung, dem Nullvektor. Der Punkt ist jedoch unzulässig, wenngleich die Verletzung der Beschränkung mit $g_1(x^*) = 2.0349 \cdot 10^{-16}$ sehr klein ist. Die berechneten Multiplikatoren `lambda.ineqnonlin` sind jedoch

$$\lambda_{\text{cineq}} = 10^9 \begin{pmatrix} 9.6107 \\ 9.6107 \end{pmatrix}$$

verdächtig groß.¹⁷ Ein Blick auf die KKT-Bedingungen der Aufgabe (17.17), insbesondere

$$\nabla_x \mathcal{L}(x, \mu) = \begin{pmatrix} -1 + 3x_1^2\mu_1 \\ \mu_1 - \mu_2 \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

¹⁷Ganz ähnlich sieht die Lösung und das Konvergenzverhalten aus, wenn wir `fmincon` zusätzlich die Ableitungen der Zielfunktion und der Beschränkungen zur Verfügung stellen.

zeigt, dass diese nur deshalb lösbar sind, weil in der Näherungslösung von `fmincon` die x_1 -Koordinate nicht exakt null ist. Im exakten Lösungspunkt $x^* = (0, 0)^T$ geht, wie wir schon wissen, die Lösbarkeit, also die Existenz Lagrangescher Multiplikatoren, verloren.

Wir halten fest: Im vorliegenden [Beispiel 17.8](#) findet `fmincon` eine gute Näherung der exakten Lösung, obwohl dieser kein KKT-Punkt ist. In anderen Fällen kann es vorkommen, dass eine solche Lösung nicht gefunden wird. Hinweise auf das Vorliegen einer solchen Situation sind die beobachtete langsame Konvergenz sowie die Größe der behaupteten Werte der Lagrange-Multiplikatoren.

Wir hatten in [Bemerkung 17.5](#) schon erwähnt, dass im Fall, dass (17.1) eine konvexe Optimierungsaufgabe darstellt, auch die Umkehrung von [Satz 17.4](#) gilt. Dies wollen wir nun noch präzisieren.

Definition 17.9 (Konvexe nichtlineare Optimierungsaufgaben). *Eine nichtlineare Optimierungsaufgabe (17.1) heißt **konvex**, wenn folgende Bedingungen alle erfüllt sind:*

- (i) *Die Zielfunktion $f(x)$ ist eine konvexe Funktion.*
- (ii) *Die Funktionen der Ungleichungsbedingungen $g_i(x)$ sind alles konvexe Funktionen, $i = 1, \dots, m$.*
- (iii) *Die Funktionen der Gleichungsbedingungen $h_j(x)$ sind alles (affin-)lineare Funktionen, $j = 1, \dots, p$, also gilt $h(x) = A_{eq} x - b_{eq}$.*

Satz 17.10 (Bedeutung konvexer Optimierungsaufgaben, vgl. [Satz 2.6](#)). *Die Aufgabe (17.1) sei konvex im Sinne der [Definition 17.9](#). Dann ist für jeden KKT-Punkt (x^*, μ, λ) die Komponente x^* bereits ein globales Minimum der Aufgabe (17.1). Insbesondere braucht man also nicht zwischen lokalen und globalen Minima zu unterscheiden.*

Wenn wir also wüssten, dass die Aufgabe (17.6) eine konvexe Optimierungsaufgabe ist, dann könnten wir sogar sicher sein, mit x^* ein Minimum berechnet zu haben (bis auf Toleranzen). Dazu müssten wir jedoch nachweisen, dass die Zielfunktion $f(x) = A(x)$ in (17.3) sowie die nichtlineare Nebenbedingung $c_{ineq}(x) = I_0 - I_x(x)$, siehe (17.5), jeweils konvexe Funktionen sind.¹⁸ Es lässt sich bei dieser noch recht einfachen Aufgabe aber feststellen, dass bereits die Zielfunktion $f(x)$ schon *nicht konvex* ist. Um dies zu überprüfen, bestimmen wir die erste und daraus die zweite Ableitung (Hessematrix) der Zielfunktion (17.3):

$$\nabla A(a_1, a_2, c_1, c_2, m) = \begin{pmatrix} (c_1 - c_2) \frac{m}{m+1} \\ c_2 \\ a_1 \frac{m}{m+1} \\ a_2 - a_1 \frac{m}{m+1} \\ \frac{a_1 (c_1 - c_2)}{(m+1)^2} \end{pmatrix} \quad (17.18)$$

¹⁸Gleichungsbeschränkungen lagen in (17.6) nicht vor, und die anderen Ungleichungen $A_{ineq} x \leq b_{ineq}$ und $\ell \leq x \leq u$ sind (affin-)linear und damit automatisch konvex.

und

$$\nabla^2 A(a_1, a_2, c_1, c_2, m) = \begin{pmatrix} 0 & 0 & \frac{m}{m+1} & -\frac{m}{m+1} & \frac{c_1-c_2}{(m+1)^2} \\ 0 & 0 & 0 & 1 & 0 \\ \frac{m}{m+1} & 0 & 0 & 0 & \frac{a_1}{(m+1)^2} \\ -\frac{m}{m+1} & 1 & 0 & 0 & -\frac{a_1}{(m+1)^2} \\ \frac{c_1-c_2}{(m+1)^2} & 0 & \frac{a_1}{(m+1)^2} & -\frac{a_1}{(m+1)^2} & -2\frac{a_1(c_1-c_2)}{(m+1)^3} \end{pmatrix}. \quad (17.19)$$

Beispielsweise hat die Hessematrix an der Stelle $x_0 = (180, 240, 30, 170, 1)^\top$, die bzgl. aller Nebenbedingungen zulässig ist, gerade den Wert

$$\nabla^2 A(x_0) = \begin{pmatrix} 0 & 0 & 0 & 0 & -35 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 45 \\ 0 & 1 & 0 & 0 & -45 \\ -35 & 0 & 45 & -45 & 6300 \end{pmatrix}.$$

Diese Matrix hat zwei negative und drei positive Eigenwerte. Damit ist die Zielfunktion $A(x)$ in der Nähe von x_0 und damit auch insgesamt nicht konvex.

Wir können also ohne weitere Prüfung (die wir hier nicht unternehmen) nicht ganz sicher sein, ob `fmincon` tatsächlich ein lokales Minimum unserer Aufgabe (17.6) gefunden hat. Dafür bräuhete man Optimalitätsbedingungen zweiter Ordnung. Eine Prüfung auf ein globales Minimum ist noch schwieriger.

§ 18 DAS VERFAHREN HINTER `fmincon`

Wir gehen nun noch kurz auf das Optimierungsverfahren hinter `fmincon` ein und beschreiben es für die allgemeine Aufgabe (17.7), die `fmincon` lösen kann. Mit der Option `'Algorithm', 'sqp'` haben wir eine sogenannte **Sequential Quadratic Programming**-Methode gewählt, eine Verallgemeinerung des Quasi-Newton-Verfahrens aus § 3. Ein SQP-Verfahren erzeugt eine Folge $\{x^{(k)}\}$ von Vektoren $x^{(k)} \in \mathbb{R}^n$, die i. d. R. gegen ein lokales Minimum konvergiert. Außerdem werden Folgen von Lagrange-Multiplikatoren $\{\lambda_{\text{ceq}}^{(k)}\}$, $\{\lambda_{\text{cineq}}^{(k)}\}$, $\{\lambda_{\text{Aeq}}^{(k)}\}$, $\{\lambda_{\text{Aineq}}^{(k)}\}$, $\{\lambda_\ell^{(k)}\}$ und $\{\lambda_u^{(k)}\}$ erzeugt, die i. d. R. zusammen mit $\{x^{(k)}\}$ im Grenzwert einen KKT-Punkt bilden, vgl. (17.15).

Ähnlich wie bei `fminunc` wird in jeder Iteration zunächst ein quadratisches Modell des Problems (17.7) aufgebaut. An der Iterierten mit dem Index $\cdot^{(k)}$ lautet dieses Modell

$$\begin{aligned} &\text{Minimiere} && f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top B^{(k)} d, \quad d \in \mathbb{R}^n, \\ &\text{sodass} && \begin{cases} c_{\text{eq}}(x^{(k)}) + c'_{\text{eq}}(x^{(k)})^\top d = 0, \\ c_{\text{ineq}}(x^{(k)}) + c'_{\text{ineq}}(x^{(k)})^\top d \leq 0, \\ A_{\text{eq}} x^{(k)} + A_{\text{eq}} d = b_{\text{eq}}, \\ A_{\text{ineq}} x^{(k)} + A_{\text{ineq}} d \leq b_{\text{ineq}}, \end{cases} \\ &\text{sowie} && \ell \leq x^{(k)} + d \leq u. \end{aligned} \quad (18.1)$$

Dabei haben wir wie in (3.2) die **Richtung** $d \in \mathbb{R}^n$, in die wir von der Stelle $x^{(k)}$ aus gehen wollen, als Unbekannte verwendet. In (18.1) erkennt man außerdem, dass sämtliche Nebenbedingungen durch ihre Linearisierung im Punkt $x^{(k)}$ ersetzt worden sind.

Die Aufgabe (18.1) ist ein sogenanntes **quadratisches Programm (QP)**, also eine Optimierungsaufgabe mit einem quadratischen Polynom als Zielfunktion und *linearen* Nebenbedingungen. Der Name **SQP-Verfahren (sequential quadratic programming)** kommt daher, dass in jeder Iteration eine QP-Aufgabe gelöst wird.

Die Matrix $B^{(k)}$ ist dabei ein Ersatz für die Hessematrix

$$\begin{aligned} \nabla_x^2 \mathcal{L}(x^{(k)}, \lambda_{\text{ceq}}^{(k)}, \lambda_{\text{cineq}}^{(k)}, \lambda_{\text{Aeq}}^{(k)}, \lambda_{\text{Aineq}}^{(k)}, \lambda_\ell^{(k)}, \lambda_u^{(k)}) \\ = \nabla^2 f(x^{(k)}) + \sum_{i=1}^{n_{\text{ceq}}} \lambda_{\text{ceq},i}^{(k)} \nabla^2 c_{\text{ceq}}(x^{(k)}) + \sum_{i=1}^{n_{\text{cineq}}} \lambda_{\text{cineq},i}^{(k)} \nabla^2 c_{\text{cineq}}(x^{(k)}) \end{aligned} \quad (18.2)$$

der Lagrange-Funktion (17.14). Neben der zweiten Ableitung (Krümmung) der Zielfunktion $f(x)$ enthält das Modell also auch Krümmungsinformation über die nichtlinearen Nebenbedingungen $c_{\text{eq}}(x)$ und $c_{\text{ineq}}(x)$.¹⁹

Die Matrix $B^{(k)}$ wird ähnlich wie in (3.8) durch ein BFGS-Quasi-Newton-Update aufdatiert, wobei auch Ableitungen der nichtlinearen Nebenbedingungen berücksichtigt werden. Auch hier wird wieder auf den Erhalt der positiven Definitheit von $B^{(k)}$ geachtet, obwohl diese Eigenschaft für die echte Hessematrix der Lagrange-Funktion im Lösungspunkt nicht gelten muss. Damit wird die Lösung von (18.1), die wir wieder mit $d^{(k)}$ bezeichnen wollen, eindeutig.²⁰

Ebenfalls ähnlich wie in § 3 wird noch eine Liniensuche durchgeführt, um eine geeignete Schrittweite $\alpha^{(k)}$ zu bestimmen. Die Anforderung an die Schrittweite ist jedoch nicht nur, einen hinreichenden Abstieg in der Zielfunktion zu garantieren, vgl. die Armijo-Bedingung (3.4). Vielmehr muss auch der Fortschritt bei der Einhaltung der Nebenbedingungen einbezogen werden. Daher wird die Liniensuche nicht bzgl. der Zielfunktion $f(x)$, sondern bzgl. einer sogenannten **Nutzen-Funktion** (engl.: **merit function**) durchgeführt, die beides berücksichtigt.

Ist die Schrittweite $\alpha^{(k)}$ bestimmt, so wird die nächste Iterierte als

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

gesetzt. Die Schrittweite $\alpha^{(k)} = 1$ ist dabei erstrebenswert für die schnelle Konvergenz des Verfahrens, ggf. müssen aber zwischendurch die Schrittweiten reduziert werden. Beachte, dass für die Schrittweiten $0 < \alpha^{(k)} \leq 1$ gilt und damit alle Iterierten stets $\ell \leq x^{(k+1)} \leq u$ erfüllen, falls dies auch für die Startschätzung $x^{(0)}$ gilt.²¹ Weiterhin werden die linearen

¹⁹Die *linearen* Nebenbedingungen $A_{\text{eq}} x \leq b_{\text{eq}}$ und $A_{\text{ineq}} x \leq b_{\text{ineq}}$ brauchen hier nicht berücksichtigt zu werden, da ihre zweiten Ableitungen null sind.

²⁰Es kann jedoch passieren, dass das QP-Problem (18.1) unzulässig ist, also keinen zulässigen Punkt und damit auch keine Lösung besitzt, weil sich Nebenbedingungen gegenseitig widersprechen. In diesem Fall muss das Verfahren besondere Maßnahmen treffen, die wir hier nicht besprechen.

²¹Die vom Benutzer gewählte Startschätzung wird von `fmincon` ggf. korrigiert, sodass $\ell \leq x^{(0)} \leq u$ immer gilt.

Nebenbedingungen $A_{\text{eq}} x \leq b_{\text{eq}}$ und $A_{\text{ineq}} x \leq b_{\text{ineq}}$ für alle nachfolgenden Iterationen exakt erfüllt, sobald nur einmal die Schrittweite $\alpha^{(k)} = 1$ gewählt wurde.

Wir können mit diesem Wissen über das SQP-Verfahren in `fmincon` nun nochmals die Ausgabe des Verfahrens für unser [Beispiel 17.1](#) interpretieren:

Iter	Func-count	Fval	Feasibility	Step Length	Norm of step	First-order optimality
0	6	2.520000e+04	1.500e+01	1.000e+00	0.000e+00	5.400e+03
1	12	2.085378e+04	0.000e+00	1.000e+00	6.986e+01	3.284e+03
2	18	1.448863e+04	0.000e+00	1.000e+00	4.255e+01	3.947e+02
3	24	1.391378e+04	0.000e+00	1.000e+00	1.362e+01	3.767e+02
4	30	1.356817e+04	0.000e+00	1.000e+00	1.099e+00	2.907e+02
5	36	1.333439e+04	0.000e+00	1.000e+00	1.288e+00	1.986e+01
6	42	1.333130e+04	0.000e+00	1.000e+00	7.724e-02	4.767e-02
7	48	1.333130e+04	0.000e+00	1.000e+00	3.047e-05	2.373e-05

Optimization completed: The relative first-order optimality measure, 8.514431e-08, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric	Options
relative first-order optimality = 8.51e-08	OptimalityTolerance = 1e-06 (default)
relative max(constraint violation) = 0.00e+00	ConstraintTolerance = 1e-06 (default)

- Die Spalte `Iter` bezeichnet unseren Iterationszähler $k = 0, 1, \dots$
- In der Spalte `Func-count` finden wir die Gesamtanzahl der Auswertungen unserer Zielfunktion $f(x)$ sowie der nichtlinearen Nebenbedingungen $c_{\text{eq}}(x)$ und $c_{\text{ineq}}(x)$. In unserem Beispiel treten Vielfache von sechs auf, da wir fünf Optimierungsvariablen haben und Finite-Differenzen-Approximationen der Ableitungen verwendet werden.
- Die Spalte `Fval` zeigt die Werte $f(x^{(k)})$ der Zielfunktion an der jeweils aktuellen Iterierten. Obwohl diese Werte in unserem Beispiel monoton fallend sind, ist die durchaus nicht immer der Fall, da zwischendurch Iterierte auftauchen können, die noch nicht alle Nebenbedingungen erfüllen und möglicherweise sogar einen kleineren Funktionswert besitzen als die Lösung. SQP-Verfahren sind also bzgl. der Zielfunktion *keine Abstiegsverfahren*.
- Die Spalte `Feasibility` zeigt an, wie stark die Iterierte $x^{(k)}$ die Nebenbedingungen noch verletzt.
- In der Spalte `Step Length` wird die mittels Liniensuche bestimmte Schrittweite $\alpha^{(k)}$ ausgegeben. Werte von $\alpha^{(k)} = 1$ sind wünschenswert und deuten auf einen guten Fortschritt des Verfahrens hin.
- Die Spalte `Norm of step` zeigt die Länge des Schrittes von $x^{(k)}$ zu $x^{(k+1)}$ an, also eine Norm von $\alpha^{(k)} d^{(k)}$.
- Schließlich finden sich in der letzten Spalte `First-order optimality` ein Maß für die Nicht-Erfüllung der KKT-Bedingungen (17.15).

Bemerkung 18.1 (Weitere Details zum SQP-Verfahren).

Wir haben bei der Beschreibung des SQP-Verfahrens in `fmincon` viele Details ausgelassen, z. B.

- welche Maßnahmen das Verfahren ergreift, wenn ein (18.1) unzulässig ist,
- wie das QP (18.1) überhaupt gelöst wird,
- wie die Nutzen-Funktion in der Liniensuche aussieht,
- wie das BFGS-Update durchgeführt wird, um die positive Definitheit von $B^{(k)}$ zu erhalten,
- wie die Lagrange-Multiplikatoren aufdatiert werden.

Einige Informationen dazu findet man auf der Hilfeseite zu `fmincon`, insbesondere unter dem Punkt *Constrained Nonlinear Optimization Algorithms*.

Ende der Woche 15

Kapitel A Mathematische Grundlagen

§ 19 MENGEN

Wir benötigen die Mengen

$$\mathbb{N} = \{1, 2, 3, \dots\} \quad (\text{natürliche Zahlen})$$

$$\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3, \dots\} \quad (\text{ganze Zahlen})$$

$$\mathbb{R} = \text{alle Dezimalzahlen} \quad (\text{reelle Zahlen}).$$

Teilmengen notieren wir mit „ $\subset$ “, etwa $\{1, 7, 129\} \subset \mathbb{N}$.

Oft haben wir es mit reellen **Intervallen** $I \subset \mathbb{R}$ zu tun:

$$[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\} \quad \text{abgeschlossenes Intervall}$$

$$(a, b) = \{x \in \mathbb{R} : a < x < b\} \quad \text{offenes Intervall.}$$

§ 20 VEKTOREN

Vektoren sind für uns praktisch, um insbesondere mehrere Optimierungsvariablen (Entscheidungsvariablen) zu einer Größe zusammenzufassen. Mit $\mathbb{R}^n$ bezeichnen wir die Menge aller Vektoren mit n Einträgen (reeller Zahlen). Ein solcher Vektor

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

besteht also aus n reellen Zahlen, seinen **Komponenten** x_i mit $1 \leq i \leq n$. Wir kennzeichnen Vektoren nicht durch Pfeile, schreiben also x und nicht $\vec{x}$. Auch den **Nullvektor** $0 \in \mathbb{R}^n$ kennzeichnen wir nicht besonders, die Anzahl seiner Einträge (die **Dimension**) geht aus dem Zusammenhang hervor.

Vektoren (derselben Dimension) können wir addieren und gleichzeitig noch mit Zahlen (Skalaren) multiplizieren („skalieren“). Sind z. B. x_1, x_2, x_3 Vektoren im $\mathbb{R}^n$ und $\alpha_1, \alpha_2, \alpha_3$ irgendwelche reellen Zahlen, so kann man

$$\sum_{i=1}^3 \alpha_i x_i = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$$

bilden. Dies nennt man eine **Linearkombination** der Vektoren x_1, x_2, x_3 .

Eine wichtige Frage zu einer Menge von Vektoren $x_1, x_2, \dots, x_m \in \mathbb{R}^n$ ist die nach ihrer linearen Unabhängigkeit. Dies kommt bei uns beispielsweise in § 12 (Beschreibung des Simplex-Verfahrens) und § 17 (bei LICQ) vor. Die Vektoren $x_1, x_2, \dots, x_m \in \mathbb{R}^n$ heißen **linear unabhängig**, wenn ihre Linearkombination zum Nullvektor

$$\sum_{i=1}^m \alpha_i x_i = 0 \in \mathbb{R}^n$$

nur möglich ist, indem alle Koeffizienten $\alpha_i = 0$ gewählt werden. In MATLAB kann man Vektoren auf lineare Unabhängigkeit testen, indem man sie nebeneinander in eine Matrix schreibt und deren Rang (siehe unten) bestimmt, beispielsweise:

```
X = [x1, x2, x3, x4];  
rank(X)
```

Stimmt der Rang überein mit der Anzahl der Vektoren (hier vier, allgemein: der Breite der Matrix `size(X,2)`), so sind die Vektoren linear unabhängig. Ist der Rang kleiner als die Anzahl der Vektoren, so sind diese linear abhängig.

Zwischen Vektoren $x, y \in \mathbb{R}^n$ ist das **Skalarprodukt** definiert:

$$x^\top y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \dots + x_n y_n.$$

In MATLAB kann man dies durch

```
x' * y
```

berechnen. Über das Skalarprodukt kommt man auch zur **Euklidischen Norm** (Länge) eines Vektors $x \in \mathbb{R}^n$:

$$\|x\| := (x^\top x)^{1/2} = \sqrt{x^\top x}$$

oder in MATLAB einfach:

```
norm(x)  
sqrt(x' * x)
```

Für Vektoren $x, y \in \mathbb{R}^n$ gilt die **Cauchy-Schwarzsche Ungleichung**:

$$x^\top y \leq \|x\| \|y\|.$$

Gleichheit ist möglich und tritt genau dann auf, wenn x ein nichtnegatives Vielfaches von y ist oder umgekehrt. Die Gültigkeit der Cauchy-Schwarzschen Ungleichung können Sie in MATLAB durch den Vergleich der Ausgabe von

```
[x' * y, norm(x) * norm(y)]
```

überprüfen.

Um mehrere Beschränkungen für Optimierungsvariablen einfach handhaben zu können, verwenden wir die Schreibweise (**vektorielle Ungleichung**)

$$x \leq y$$

auch für Vektoren $x, y \in \mathbb{R}^n$. Gemeint ist, dass die entsprechende Ungleichung für *alle* Komponenten der Vektoren erfüllt ist:

$$x_i \leq y_i \quad \text{für alle } i = 1, \dots, n.$$

In MATLAB können Sie folgende Befehle verwenden, um die Größenverhältnisse der Komponenten von Vektoren x und y einzeln bzw. gemeinsam zu überprüfen:

```
x <= y
all(x <= y)
```

§ 21 MATRIZEN

Eine (reelle) Matrix A mit m Zeilen und n Spalten ist ein rechteckiges Zahlenschema der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

mit Einträgen $a_{ij} \in \mathbb{R}$. Wir sprechen genauer von einer $m \times n$ -**Matrix** oder einer **Matrix vom Typ** (m, n) . Man schreibt kurz: $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, wobei $i = 1, \dots, m$ die Zeilen und $j = 1, \dots, n$ die Spalten nummeriert. Man nennt auch i den **Zeilenindex** und j den **Spaltenindex**. Matrizen bezeichnen wir immer mit Großbuchstaben. In MATLAB können Sie Matrizen folgendermaßen eingeben und ihre Größe bestimmen:

```
A = [1 2 3 4; 5 6 7 8]
A = [1 2 3 4; ...
5 6 7 8]
[m,n] = size(A)
```

Die **Transponierte** $B := A^T$ einer Matrix $A \in \mathbb{R}^{m \times n}$ erhält man durch Tauschen von Zeilen- und Spaltenindex:

$$B = (b_{ij}) \in \mathbb{R}^{n \times m} \quad \text{mit} \quad b_{ij} = a_{ji} \quad \text{für } i = 1, \dots, n \text{ und } j = 1, \dots, m.$$

In MATLAB:

```
B = A'
```

(Spalten-)Vektoren in $\mathbb{R}^n$ sind übrigens in MATLAB nichts anderes als „schlanke“ Matrizen in $\mathbb{R}^{n \times 1}$.

Der **Rang** einer Matrix $A \in \mathbb{R}^{m \times n}$ ist die maximale Anzahl linear unabhängiger Spaltenvektoren von A .¹ Dies stimmt immer überein mit der maximalen Anzahl linear unabhängiger Zeilenvektoren von A . Also ist $\text{rank}(A) \leq \min\{m, n\}$. In MATLAB kann man den Rang von A mittels

```
rank(A)
```

bestimmen.

Für Matrizen $A \in \mathbb{R}^{m \times n}$ und $B \in \mathbb{R}^{n \times p}$ kann man das **Matrix-Matrix-Produkt** $C = AB \in \mathbb{R}^{m \times p}$ bilden.² Der Eintrag c_{ij} ist gerade das Skalarprodukt der i -ten Zeile von A mit der j -ten Spalte von B . In MATLAB:

```
A = [1 2 3 4; 5 6 7 8]
B = [1 0 -3; -2 -1 0; 4 1 0; -5 1 2]
C = A * B
size(A)
size(B)
size(C)
```

Es gilt $(AB)^T = B^T A^T$, wie man leicht bestätigt:

```
(A * B)' - B' * A'
```

Die **Einheitsmatrix** in $\mathbb{R}^{n \times n}$ bezeichnen wir mit I :

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}$$

In MATLAB erhalten wir beispielsweise $I \in \mathbb{R}^{5 \times 5}$ durch

```
eye(5)
```

Eine quadratische Matrix $A \in \mathbb{R}^{n \times n}$ heißt **invertierbar** oder **regulär**, wenn es eine (eindeutig bestimmte) Matrix $B \in \mathbb{R}^{n \times n}$ (gleiche Größe) gibt, sodass gilt:

$$AB = BA = I.$$

Dies ist genau dann der Fall, wenn $\text{rank}(A) = n$ ist. In MATLAB erhalten Sie die inverse Matrix (zu einer hier zufällig erzeugten Matrix A) wie folgt:

```
A = 10*randn(5,5)
B = inv(A)
```

¹Man wählt also möglichst viele Spalten der Matrix A aus, sodass diese noch linear unabhängig sind.

²**Beachte:** Die Breite von A (hier: n) muss übereinstimmen mit der Höhe von B .

Für quadratische Matrizen $A \in \mathbb{R}^{n \times n}$ ist die **Determinante** wie folgt erklärt:

$$A \in \mathbb{R}^{1 \times 1} : \det(a_{11}) = a_{11}$$

$$A \in \mathbb{R}^{2 \times 2} : \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11} a_{22} - a_{21} a_{12}$$

$$A \in \mathbb{R}^{3 \times 3} : \det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{aligned} & a_{11} a_{22} a_{33} + a_{12} a_{23} a_{31} + a_{13} a_{21} a_{32} \\ & - a_{31} a_{22} a_{13} - a_{32} a_{23} a_{11} - a_{33} a_{21} a_{12}. \end{aligned}$$

(Für größere Matrizen wird diese explizite Definition unübersichtlich.) In MATLAB kann man die Determinante einer Matrix wie folgt berechnen:

```
det(A)
```

Anhand der Determinante kann man die Invertierbarkeit einer Matrix entscheiden:³

$$A \text{ ist invertierbar} \Leftrightarrow \det(A) \neq 0.$$

Wenn $A \in \mathbb{R}^{n \times n}$ eine quadratische Matrix ist, dann heißt die Zahl λ ein **Eigenwert** und $v \neq 0$ ein zugehöriger **Eigenvektor**, wenn gilt:

$$A v = \lambda v.$$

Im Allgemeinen können Eigenwerte und Eigenvektoren komplexwertig sein. Für reelle **symmetrische Matrizen** ($A = A^T$) sind sie jedoch immer reellwertig. In MATLAB erhält man die Eigenwerte und Eigenvektoren einer Matrix wie folgt:

```
[V,D] = eig(A)
```

Dabei enthält V spaltenweise die Eigenvektoren und D die Eigenwerte (als Diagonale). Es gilt $A = V D V^{-1}$. Nur die Eigenwerte (als Vektor) bekommt man über

```
d = eig(A)
```

Bei Diagonalmatrizen kann man sich die Berechnung der Eigenwerte sparen, denn dies sind einfach die Diagonaleinträge.

Bei Optimalitätsbedingungen 2. Ordnung sowie der Konvexität von Funktionen spielt die positive Definitheit von Matrizen eine wichtige Rolle. Dabei heißt eine quadratische Matrix $A \in \mathbb{R}^{n \times n}$ **positiv semidefinit**, wenn gilt:

$$x^T A x \geq 0 \quad \text{für alle } x \in \mathbb{R}^n.$$

A heißt sogar **positiv definit**, wenn gilt:

$$x^T A x > 0 \quad \text{für alle } x \in \mathbb{R}^n, x \neq 0.$$

³Numerisch ist dies allerdings kein gutes Vorgehen. Besser ist die Verwendung von $\text{cond}(A)$. Eine große Konditionszahl (oder sogar ∞) sagt aus, dass A „fast singulär“ ist.

Übrigens ist eine positiv definite Matrix immer invertierbar, und auch die inverse Matrix ist wieder positiv definit.

Für *symmetrische* Matrizen A kann man die Definitheit durch die Eigenwerte charakterisieren:

$$\begin{aligned} A = A^T \text{ ist positiv semidefinit} &\Leftrightarrow \text{alle Eigenwerte sind } \geq 0. \\ A = A^T \text{ ist positiv definit} &\Leftrightarrow \text{alle Eigenwerte sind } > 0. \end{aligned}$$

Die **Konditionszahl** einer quadratischen Matrix A kann definiert werden als

$$\text{cond}(A) = \frac{\sqrt{\lambda_{\max}(A^T A)}}{\sqrt{\lambda_{\min}(A^T A)}}.$$

Große Konditionszahlen (z. B. 10^6) wirken sich in vielen Belangen ungünstig aus, beispielsweise leidet darunter die Genauigkeit bei der Lösung linearer Gleichungssysteme mit A als Koeffizientenmatrix.

§ 22 FUNKTIONEN

FUNKTIONEN, DIE IN $\mathbb{R}$ ABBILDEN

Eine **Funktion** $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ist eine Vorschrift, die jedem Vektor $x \in \mathbb{R}^n$ (Eingabe) eine Zahl $y \in \mathbb{R}$ (Ausgabe) zuordnet. Wir betrachten in dieser Vorlesung nur hinreichend glatte (einmal oder zweimal stetig differenzierbare) Funktionen. Die **erste Ableitung** von $f(x)$ an einer Stelle $x \in \mathbb{R}^n$ ist zusammengesetzt aus den **partiellen Ableitungen**

$$\frac{\partial f(x)}{\partial x_j} = \lim_{h \rightarrow 0} \frac{f(x + h e_j) - f(x)}{h}, \quad j = 1, \dots, n.$$

Dabei ist $e_j = (0, \dots, 0, 1, 0, \dots, 0)^T$ einer der Einheitsvektoren des $\mathbb{R}^n$ mit der „1“ an Position j für $1 \leq j \leq n$.

Analytisch⁴ berechnet man die partiellen Ableitungen genauso wie die Ableitung einer Funktion $g : \mathbb{R} \rightarrow \mathbb{R}$, indem man sich einfach $f(x)$ als Funktion nur von x_j denkt und die übrigen Variablen festhält (konstant setzt). Beispielsweise ergeben sich die partiellen Ableitungen von

$$f(x_1, x_2) = x_2 \sqrt{x_1} + \ln x_1 + 2 \ln x_2$$

wie folgt:

$$\begin{aligned} \frac{\partial f(x_1, x_2)}{\partial x_1} &= \frac{x_2}{2\sqrt{x_1}} + \frac{1}{x_1} \\ \frac{\partial f(x_1, x_2)}{\partial x_2} &= \sqrt{x_1} + \frac{2}{x_2} \end{aligned}$$

⁴In MATLAB kann man nur mit Hilfe der Symbolic-Math-Toolbox symbolisch die Ableitungen einer Funktion ausrechnen. Dies wird im Rahmen dieser Vorlesung aber nicht verwendet. Weitere Möglichkeiten zur Ableitungsberechnung sind in § 4 genannt.

für alle Punkte $x = (x_1, x_2)$, an denen die Funktionen definiert sind, also $x_1 > 0$ und $x_2 > 0$.

Alle partiellen Ableitungen erster Ordnung zusammen sammelt man im **Gradienten**, üblicherweise als Spaltenvektor:⁵

$$\underbrace{\nabla f(x)}_{\substack{\text{sprich: Nabla} \\ \text{oder Gradient}}} = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n.$$

Mit seiner Hilfe kann man auch bequem das **Taylorpolynom** erster Ordnung schreiben, besser bekannt als **Tangente**. Das Taylorpolynom erster Ordnung ist allein durch die Daten

$$f(x^*) \in \mathbb{R} \quad \text{und} \quad \nabla f(x^*) \in \mathbb{R}^n$$

an der Entwicklungsstelle x^* festgelegt:

$$\begin{aligned} T_1(x; x^*) &:= f(x^*) && \text{Term nullter Ordnung (konstant)} \\ &+ \nabla f(x^*)^\top (x - x^*) && \text{Term erster Ordnung (linear).} \end{aligned}$$

Es ist das beste lineare Modell von $f(x)$ in der Nähe der Entwicklungsstelle x^* .

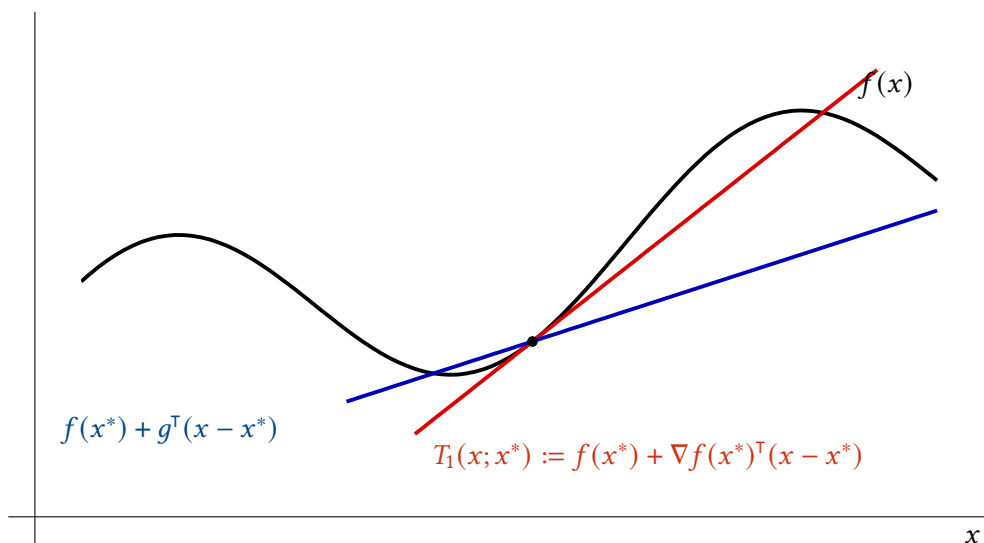


Abbildung 22.1.: Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ und ihr lineares Taylorpolynom $T_1(x; x^*)$ (in rot) mit Entwicklungspunkt x^* . Gezeigt wird außerdem ein anderes lineares Modell der Funktion (blau), dessen Ableitung g nicht mit $\nabla f(x^*)$ übereinstimmt. Man sieht, dass das Taylorpolynom $T_1(x; x^*)$ die Funktion $f(x)$ besser annähert als das vereinfachte Modell, das dafür ohne die Berechnung der Ableitung $\nabla f(x^*)$ auskommt.

⁵Für eindimensionale Funktionen ($n = 1$) besteht der Gradient $\nabla f(x)$ nur aus einer Zahl, der gewöhnlichen Ableitung.

Die partiellen Ableitungen zweiter Ordnung sind definiert als die Ableitungen der Ableitungen erster Ordnung:

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left(\frac{\partial f(x)}{\partial x_j} \right), \quad i, j = 1, \dots, n.$$

Alle partiellen Ableitungen zweiter Ordnung ordnet man in einer Matrix an, der sogenannten **Hessematrix**:⁶

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Im obigen Beispiel ist für $x_1 > 0$ und $x_2 > 0$

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} \frac{-x_2}{\sqrt{x_1}^3} - \frac{1}{x_1^2} & \frac{1}{2\sqrt{x_1}} \\ \frac{1}{2\sqrt{x_1}} & -\frac{1}{x_2^2} \end{pmatrix}.$$

Die hier erkennbare Symmetrie der Hessematrix ist typisch (Satz von Schwarz).

Mit Hilfe der Hessematrix an der Entwicklungsstelle x^* kann man das lineare Taylorpolynom zu einem quadratischen verbessern:

$$\begin{aligned} T_2(x; x^*) &:= f(x^*) && \text{Term nullter Ordnung (konstant)} \\ &+ \nabla f(x^*)^\top (x - x^*) && \text{Term erster Ordnung (linear)} \\ &+ \frac{1}{2} (x - x^*)^\top \nabla^2 f(x^*) (x - x^*) && \text{Term zweiter Ordnung (quadratisch).} \end{aligned}$$

Es ist das beste quadratische Modell der Funktion $f(x)$ in der Nähe des Entwicklungspunktes x^* .

Das Taylorpolynom erster Ordnung ist ein Beispiel für ein **lineares Polynom** in mehreren Variablen. Dieses hat die allgemeine Form

$$f(x) = g^\top x + c$$

mit einem Vektor $g \in \mathbb{R}^n$ und einer Konstanten $c \in \mathbb{R}$. Man spricht auch von einer (**affin-**) **linearen** Funktion. Der Vektor g ist der Gradient von $f(x)$ und gibt die Richtung des steilsten Anstiegs an.

Das Taylorpolynom zweiter Ordnung ist ein Beispiel für ein **quadratisches Polynom** in mehreren Variablen. Es hat die allgemeine Form

$$f(x) = \frac{1}{2} x^\top Q x + g^\top x + c.$$

⁶Für eindimensionale Funktionen ($n = 1$) besteht die Hessematrix $\nabla^2 f(x)$ nur aus einer Zahl, der gewöhnlichen zweiten Ableitung.

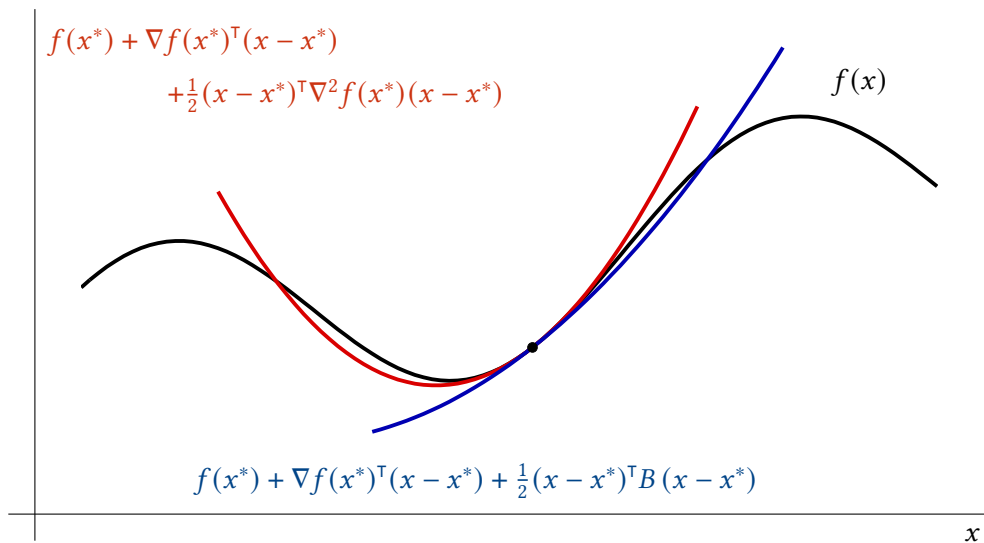


Abbildung 22.2.: Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ und ihr quadratisches Taylorpolynom $T_2(x; x^*)$ (in rot) mit Entwicklungspunkt x^* . Gezeigt wird außerdem ein vereinfachtes quadratisches Modell der Funktion (blau), dessen zweite Ableitung (Hessematrix) B nicht mit $\nabla^2 f(x^*)$ übereinstimmt. Man sieht, dass das Taylorpolynom $T_2(x; x^*)$ die Funktion $f(x)$ besser annähert als das vereinfachte Modell, das dafür ohne die Berechnung der zweiten Ableitung $\nabla^2 f(x^*)$ auskommt.

Die Matrix $Q \in \mathbb{R}^{n \times n}$ kann man dabei immer als symmetrisch annehmen. Wenn sie nicht symmetrisch wäre, könnte man sie einfach durch ihren **symmetrischen Anteil**

$$Q_{\text{sym}} := \frac{Q + Q^T}{2}$$

ersetzen, denn:

$$\begin{aligned} x^T Q x &= \frac{1}{2} x^T Q x + \frac{1}{2} x^T Q x \\ &= \frac{1}{2} x^T Q x + \frac{1}{2} (x^T Q x)^T \quad (\text{Transponieren einer Zahl ändert sie nicht}) \\ &= \frac{1}{2} x^T Q x + \frac{1}{2} (x^T Q^T x) \quad (\text{Transposition ausgeführt}) \\ &= \frac{1}{2} x^T (Q + Q^T) x \\ &= x^T Q_{\text{sym}} x. \end{aligned}$$

Quadratische Polynome haben in der Optimierung große Bedeutung, z. B. in den quadratischen Modellen, wie sie etwa bei Quasi-Newton-Verfahren verwendet werden, siehe § 3. Daher geben wir hier noch den Gradienten und die Hessematrix einer solchen Funktion an:⁷

$$\begin{aligned} f(x) &= \frac{1}{2} x^T Q x + g^T x + c \\ \nabla f(x) &= Q x + g \\ \nabla^2 f(x) &= Q. \end{aligned}$$

⁷Hier wird wie gesagt davon ausgegangen, dass Q schon symmetrisch ist. Ansonsten gilt nämlich $\nabla f(x) = (1/2)(Q + Q^T)x$ und $\nabla^2 f(x) = (1/2)(Q + Q^T) = Q_{\text{sym}}$.

Lineare Polynome sind hier natürlich mit enthalten, indem man $Q = 0$ (Nullmatrix) setzt.

§ 22.1 KONVEXITÄT

Im Verlauf der Vorlesung haben wir mehrfach den Begriff der **Konvexität** erwähnt. In der Optimierung spielen konvexe Optimierungsaufgaben eine besondere Rolle, die durch eine konvexe Zielfunktion und eine konvexe zulässige Menge gekennzeichnet sind. Auf die Bedeutung konvexer Optimierungsaufgaben wird in [Satz 2.6](#) und [Satz 17.10](#) eingegangen: Man braucht insbesondere nicht zwischen lokalen und globalen Minima zu unterscheiden.

Eine **Menge** $C \subset \mathbb{R}^n$ heißt **konvex**, wenn zu je zwei Punkten $x \in C$ und $y \in C$ immer auch die gesamte Verbindungsstrecke innerhalb der Menge C verläuft. Dies kann man wie folgt schreiben:

$$x, y \in C \quad \Rightarrow \quad \underbrace{\lambda x + (1 - \lambda) y}_{\text{Punkt auf der Verbindungsstrecke}} \in C \quad \text{für alle } \lambda \in [0, 1].$$

Eine **konvexe Funktion** ist immer auf einer konvexen Menge definiert ($f : C \rightarrow \mathbb{R}$) und erfüllt folgende Eigenschaft:

$$x, y \in C \quad \Rightarrow \quad f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y) \quad \text{für alle } \lambda \in [0, 1].$$

Diese Ungleichung bedeutet, dass der Graph der Funktion immer unterhalb jeder möglichen Sekante verläuft.

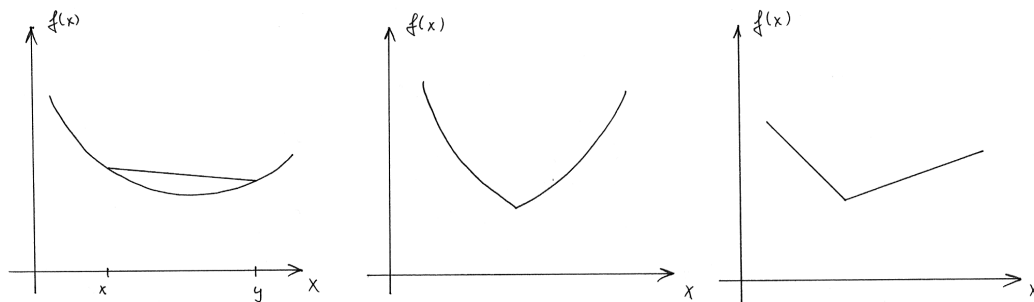


Abbildung 22.3.: Verschiedene konvexe Funktionen $f : \mathbb{R} \rightarrow \mathbb{R}$

Die Konvexität entspricht im Fall einer Funktion von einer Variablen (also $f : C \rightarrow \mathbb{R}$ mit $C \subset \mathbb{R}$) der Linkskrümmung der Funktion. Dies bedeutet, dass die zweite Ableitung ≥ 0 ist. Im mehrdimensionalen Fall ($f : C \rightarrow \mathbb{R}$ mit $C \subset \mathbb{R}^n$) gilt ein entsprechendes Resultat: Eine solche Funktion ist konvex, wenn ihre Hessematrix $\nabla^2 f(x)$ an jeder Stelle $x \in C$ positiv semidefinit ist.

Wir haben im Verlauf der Vorlesung mehrere Beispiele konvexer Optimierungsaufgaben gesehen:

- In [Kapitel 1](#) stellen [Beispiel 2.1](#) (Isolierung beim Hausbau) und die reduzierte Version von [Beispiel 6.1](#) (Getränkedose) konvexe unbeschränkte Optimierungsaufgaben (mit der zulässigen Menge $C = \mathbb{R}^n$) dar.
- Kleinste-Quadrate-Aufgaben ([Kapitel 2](#)) sind immer dann konvex, wenn die Modell- bzw. Residuenfunktion (affin-)linear von den gesuchten Parametern x abhängt. Dies war in [Beispiel 7.1](#) (Kalibrierung einer Preis-Absatz-Funktion) der Fall.
- Lineare Optimierungsaufgaben ([Kapitel 3](#)) sind immer konvex, denn die Zielfunktion ist sogar linear, und die zulässige Menge – beschrieben durch (affin-)lineare Gleichungen und Ungleichungen – ist ein konvexes Polyeder. **Achtung:** Sobald man Ganzzahligkeitsbedingungen (wie $x \in \mathbb{Z}^n$) an die Lösung hinzunimmt, verliert man die Konvexität der zulässigen Menge.⁸
- Die allgemeine Optimierungsaufgabe (17.1) ist konvex, falls die Zielfunktion $f(x)$ konvex ist, die Funktionen $g_i(x)$ in den Ungleichungsbeschränkungen ebenfalls alle konvex sind und die Funktionen $h_j(x)$ in den Gleichungsbeschränkungen (affin-)linear.⁹

Nichtlineare Optimierungsaufgaben ([Kapitel 4](#)) sind leider oft nicht konvex, oder ihre Konvexität ist schwer zu erkennen. [Beispiel 17.1](#) (Querlenker) ist nicht konvex, denn die Zielfunktion ist keine konvexe Funktion. Dies hatten wir auf [Seite 135](#) festgestellt. Auch [Beispiel 17.3](#) ist wegen der Gleichungsbeschränkung, die nicht (affin-)linear ist, nicht konvex.

§ 22.2 FUNKTIONEN, DIE IN $\mathbb{R}^m$ ABBILDEN

Eine **Funktion** $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ist eine Vorschrift, die einem Vektor $x \in \mathbb{R}^n$ (Eingabe) einen Vektor $y \in \mathbb{R}^m$ (Ausgabe) zuordnet. Man kann $f(x)$ schreiben als die Zusammensetzung der **Komponentenfunktionen**:

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{pmatrix}.$$

Jede Komponentenfunktion ist reellwertig: $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$, also eine Funktion, wie sie in [Anhang 22](#) behandelt wurde.

Die **erste Ableitung** von $f(x)$ an einer Stelle $x \in \mathbb{R}^n$ besteht aus den **partiellen Ableitungen** aller Komponentenfunktionen. Man ordnet diese in der **Jacobimatrix** wie folgt

⁸Bei den Beispielen in [§ 14](#) und [§ 16](#) brauchten wir die Ganzzahligkeit nicht extra zu fordern, da sich diese (bei ganzzahligen rechten Seiten b_{eq} und b_{ineq}) automatisch aus der totalen Unimodularität der Matrix $\begin{bmatrix} A_{\text{eq}} \\ A_{\text{ineq}} \end{bmatrix}$ ergab. Diese Beispiele stellen also konvexe Optimierungsaufgaben dar.

⁹Für die von `fmincon` behandelte Aufgabe (17.7) bedeutet dies, dass $f(x)$ konvex sein soll, $c_{\text{ineq}}(x)$ ebenfalls (komponentenweise) konvex und $c_{\text{eq}}(x)$ (affin-)linear.

an:

$$J(x) := \left[\frac{\partial}{\partial x_j} f_i(x) \right]_{ij} = \begin{bmatrix} | & & | \\ \frac{\partial}{\partial x_1} f(x) & \cdots & \frac{\partial}{\partial x_n} f(x) \\ | & & | \end{bmatrix} = \begin{bmatrix} -\nabla f_1(x)^\top - \\ \vdots \\ -\nabla f_m(x)^\top - \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

In jeder Zeile stehen also die partiellen Ableitungen (nach allen Variablen x_1 bis x_n) von einer der Komponentenfunktionen. In jeder Spalte stehen die partiellen Ableitungen aller Komponentenfunktionen nach einer der Variablen.

Die **Kettenregel** gilt auch für vektorwertige Funktionen mehrerer Veränderlicher: Sind $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ und $g: \mathbb{R}^p \rightarrow \mathbb{R}^n$ differenzierbar, dann auch die zusammengesetzte Funktion $(f \circ g)(x) = f(g(x))$. Deren Jacobimatrix ist

$$\underbrace{J_f(g(x))}_{\in \mathbb{R}^{m \times n}} \underbrace{J_g(x)}_{\in \mathbb{R}^{n \times p}} \in \mathbb{R}^{m \times p}.$$

Eine einfache Anwendung der Kettenregel ergibt sich bei der Skalierung von Variablen. Angenommen, man hat eine Funktion $f(z)$ vorliegen, sagen wir $f: \mathbb{R}^n \rightarrow \mathbb{R}$, beispielsweise die Zielfunktion einer Optimierungsaufgabe. Man entscheidet sich dann aber, eine Skalierung der Variablen $z \in \mathbb{R}^n$ vorzunehmen.¹⁰ Dazu führt man neue Variablen $x \in \mathbb{R}^n$ ein, die zu z in der Beziehung $z = Sx$ stehen. Dabei ist S eine (typischerweise diagonale) Skalierungsmatrix. Die zugehörige neue Funktion $\tilde{f}(x)$ in den neuen Variablen x ist dann

$$\tilde{f}(x) := f(Sx).$$

Deren Ableitungen kann man mit Hilfe der Kettenregel auf die Ableitungen von $f(z)$ zurückführen. Für die erste Ableitung ergibt sich

$$J_{\tilde{f}}(x) = J_f(Sx) S.$$

Durch Transposition kann man dies auch mit Hilfe der Gradienten schreiben:

$$\nabla \tilde{f}(x) = S^\top \nabla f(Sx).$$

Manchmal schreibt man zur Verdeutlichung auch noch bei der Differentiation die Namen der Variablen dazu, in denen die jeweilige Funktion ursprünglich definiert war, also hier:

$$\nabla_x \tilde{f}(x) = S^\top \nabla_z f(z) \Big|_{z=Sx}.$$

Für die zweite Ableitung (Hessematrix) ergibt sich entsprechend

$$\nabla_x^2 \tilde{f}(x) = S^\top \nabla_z^2 f(z) S \Big|_{z=Sx}.$$

¹⁰Ein Grund dafür kann sein, dass manche Optimierungsverfahren nicht skalierungsinvariant sind und dann am besten funktionieren, wenn alle Komponenten der erwarteten Lösung dieselbe Größenordnungen haben.

Literatur

- Alt, W. (2002). *Nichtlineare Optimierung*. Vieweg Studium: Aufbaukurs Mathematik. Eine Einführung in Theorie, Verfahren und Anwendungen. [An introduction to theory, procedures and applications]. Friedr. Vieweg & Sohn, Braunschweig. DOI: [10.1007/978-3-322-84904-5](https://doi.org/10.1007/978-3-322-84904-5).
- Boyd, S.; L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press, Cambridge. DOI: [10.1017/CB09780511804441](https://doi.org/10.1017/CB09780511804441).
- Bredies, K.; D. Lorenz (2011). *Mathematische Bildverarbeitung*. Vieweg & Teubner. DOI: [10.1007/978-3-8348-9814-2](https://doi.org/10.1007/978-3-8348-9814-2).
- Fletcher, R. (1987). *Practical Methods of Optimization*. 2. Aufl. A Wiley-Interscience Publication. Chichester: John Wiley & Sons Ltd.
- Gerdts, M.; F. Lempio (2011). *Mathematische Optimierungsverfahren des Operations Research*. de Gruyter. DOI: [10.1515/9783110249989](https://doi.org/10.1515/9783110249989).
- Nocedal, J.; S. Wright (2006). *Numerical Optimization*. 2. Aufl. New York: Springer. DOI: [10.1007/978-0-387-40065-5](https://doi.org/10.1007/978-0-387-40065-5).
- Vanderbei, R. J. (2008). *Linear Programming: Foundations and Extensions*. Operations Research, Management Science. New York, NY: Springer. DOI: [10.1007/978-0-387-74388-2](https://doi.org/10.1007/978-0-387-74388-2).
-