TUBAF

Die Ressourcenuniversität.
Seit 1765.

Grundlagen der Optimierung

Wintersemester 2024/25

Dr. Max Winkler

Institut Numerik und Optimierung, TU Bergakademie Freiberg

14. Januar 2025

Inhaltsverzeichnis

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.

Aufgabenstellung

Gegenstand dieser Vorlesung sind Optimierungsaufgaben der folgenden Gestalt:

Definition 0.1 (Allgemeine Optimierungsaufgabe)

Ein allgemeines Optimierungsproblem, auch **Optimerungsaufgabe**, besitzt die Gestalt

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

Das offensichtliche Ziel ist es einen “Punkt” x zu finden, der

- die Funktion f minimiert, und dabei
- die gewisse Nebenbedingungen $g_i(x) \leq 0$ und $h_j(x) = 0$ erfüllt.

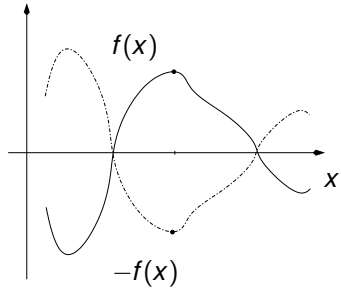
Bezeichnungen

Definition 0.2 (Zielfunktion, Nebenbedingungen, Variablen)

- Die Menge Ω heißt **Grundmenge**. Je nach Anwendung nutzt man
  - $\Omega = \mathbb{R}^n \ldots$ kontinuierliche Optimierung,
  - $\Omega = \mathbb{Z}^n \ldots$ diskrete Optimierung.
- Die Komponenten x_i, $i = 1, \ldots, n$ des Vektors x heißen **Optimierungsvariablen** oder kurz: **Variablen**.
- f heißt **Zielfunktion** oder **Kostenfunktion**, die wir minimieren möchten. Statt "Minimiere $f(x)$" schreibt man auch häufig "$f(x) \longrightarrow \min!$".
- Die Nebenbedingung $g_i(x) \leq 0$ heißt **Ungleichungsnebenbedingung** oder **Ungleichungsrestriktionen**.
- Die Nebenbedingung $h_j(x) = 0$ heißt **Gleichungsnebenbedingung** oder **Gleichungsrestriktionen**.

Maximierungsaufgaben

Es ist keine Einschränkung, dass wir uns mit Minimierungsaufgaben befassen. Das Maximum von $f(x)$ wird dort angenommen, wo $-f(x)$ ihr Minimum annimmt.

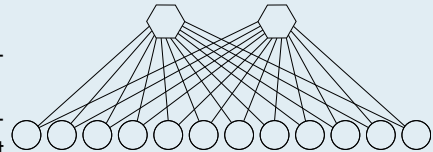


Ebenso können Ungleichungen der Form $g(x) \geq 0$ umgeschrieben werden zu $-g(x) \leq 0$.

Beispiel: Transportproblem

Beispiel 0.3

Eine Firma besitzt zwei Fabriken F_1, F_2 und zwölf Verkaufsstellen $V_1, \ldots, V_{12}$.
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}.
Wie muss die Produktion auf die Verkaufsstellen verteilt werden, um die Gesamtkosten des Transports zu minimieren und die Nachfrage zu decken?

Beispiel: Transportproblem

$$
\begin{aligned}
&\text{Minimiere} && f(x) := \sum_{i=1}^{2} \sum_{j=1}^{12} c_{ij}\, x_{ij} && \text{(Transportkosten)}, && x \in \mathbb{R}^{2\times 12} \\
&\text{sodass} && \sum_{j=1}^{12} x_{ij} \leq p_i, \quad i = 1, 2 && \text{(Produktionsrestriktionen)} && \\
&\text{und} && \sum_{i=1}^{2} x_{ij} \geq n_j, \quad j = 1, \ldots, 12 && \text{(Bedarfsdeckung)} && \\
&\text{sowie} && x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \ldots, 12. && \text{(Nicht-Negativität)} &&
\end{aligned}
\tag{1}
$$

Beispiel: Transportproblem

- 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 in der Regel als Vektor geschrieben werden. Dies zieht eine (einfache) Umformulierung der Zielfunktion und der Nebenbedingungen nach sich.
- Die Zielfunktion und alle Nebenbedingungen sind (affin-)lineare Funktionen der Entscheidungsvariablen x. Es handelt sich damit um eine **lineare Optimierungsaufgabe**, auch **lineares Programm** (**LP**) mit deren Lösung wir uns in Kapitel 2 beschäftigen werden.
- Die Ungleichungsnebenbedingungen sind ein wesentlicher Bestandteil der Aufgabenstellung (1). Lässt man etwa die Beschränkungen für die Bedarfsdeckung weg, so wird sich als Lösung $x_{ij} = 0$ für alle $i = 1, \ldots, 2$, $j = 1, \ldots, 12$ ergeben, also überhaupt kein Transport, der ja auch keine Kosten verursacht.
- Die Aufgabe (1) hat keine Gleichungsnebenbedingungen.

Ziele der Vorlesung

In dieser Lehrveranstaltung sollen Sie lernen,

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

Software einrichten

- Als Programmiersprache verwenden wir MATLAB. Die Installation auf dem eigenen Rechner ist auf `https://de.mathworks.com/academia/tah-portal/tu-bergakademie-freiberg-40758603.html` beschrieben. Lizenzen sind über die TU Bergakademie Freiberg erhältlich. In den PC-Pools ist Matlab bereits installiert.
- **Wichtig:** Während der Installation wird man gefragt, welche Toolboxen installiert werden sollen. Wir benötigen die **Optimization Toolbox**. Hier nicht benötigt, aber im Mathematik-Studium oft hilfreich, sind außerdem die **Symbolic Toolbox** und die **PDE Toolbox**.
- Öffne MATLAB über das Startmenü oder den Konsolen-Befehl `matlab` und tippe einfache Kommandos in die Matlab-Konsole ein, zum Beispiel:

```
x = linspace(0,2*pi)
y = sin(x)
plot(x,y)
```

- Erzeuge ein Skript mit namen `plot_sin.m`, kopiere die obigen Zeilen hinein und führe das Skript durch Eingabe von `plot_sin` aus.

Software einrichten

- Als Editor für Matlab-Code bietet sich auch JupyterLab an. Die Installation ist etwas aufwändiger:
  1. Installiere miniconda: `https://docs.anaconda.com/miniconda/miniconda-other-installer-links/`
  2. Öffne die Miniconda-Konsole und erstelle eine neue Umgebung mit einer lokalen Jupyter-Hub-Installation:

     ```
     conda create -n matlab
     conda activate matlab
     conda install jupyterlab
     pip install jupyter-matlab-proxy
     ```

  3. Starte JupyterLab mit dem Aufruf

     ```
     jupyter lab
     ```

     Im Browser öffnet sich nun ein Editor und wir können die Übungszettel mit der Endung `.ipynb` öffnen.

Inhaltsverzeichnis

Inhaltsverzeichnis

Problemstellung

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

Definition 1.1 (Unrestringiertes Optimierungsproblem)

Ein Optimierungsproblem der Form

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

heißt **unrestringiertes Optimierungsproblem**.

Grundvoraussetzung: $f(x)$ ist eine glatte Funktion, genauer, $f(x)$ ist zweimal stetig differenzierbar.

Lokale und globale Lösungen

Definition 1.2 (Lösungen von unrestringierten Optimierungsaufgaben)

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

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

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

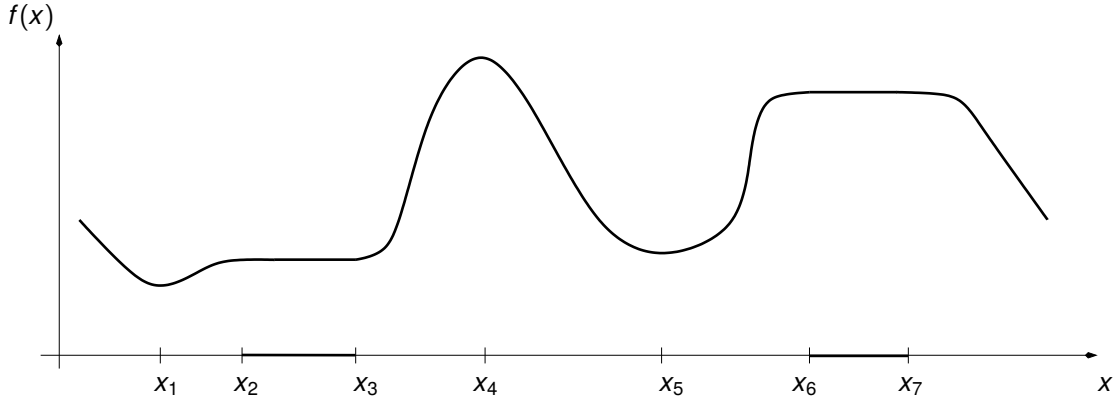
$$f(x^*) \leq f(x) \quad \text{für alle } x \in U(x^*).$$

3. Ein globales Minimum x^* heißt **strikt**, wenn gilt:

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

4. Analog definiert man ein **strikte lokales Minimum**.

Lokale und globale Lösungen



Übung 1.3

Bestimme alle (strikten) lokalen und globalen Optima?

Lösungen charakterisieren

Aus der Schule ist bekannt, dass für eine Funktion $f\colon \mathbb{R} \to \mathbb{R}$ gilt:

- Jedes Minimum x^* von f erfüllt die **notwendige Optimalitätsbedingung**

$$f'(x^*) = 0$$

- Erfüllt ein $x^* \in \mathbb{R}$ die notwendige Bedingung $f'(x^*) = 0$ sowie die **hinreichende Optimalitätsbedingung**

$$f''(x^*) > 0,$$

  dann ist x^* striktes lokales Minimum von f.
- Beachte: Es gibt auch Funktionen, deren Minima die hinreichende Bedingung verletzen. Finde hierfür Beispiele.

Gradient und Hessematrix

Definition 1.4 (Gradient und Hessematrix)

a) Eine Funktion $f\colon \mathbb{R}^n \to \mathbb{R}$ heißt **partiell differenzierbar**, wenn die Grenzwerte

$$\frac{\partial f}{\partial x_j}(x) := \lim_{h \to 0} \frac{f(x + h\, e_j) - f(x)}{h}$$

für alle $j = 1, \ldots, n$ existieren. $\frac{\partial f}{\partial x_j}$ heißt dann **partielle Ableitung** nach x_j. Hierbei ist $e_j := (0, \ldots, 0, \underbrace{1}_{j-\text{te Stelle}}, 0, \ldots, 0)^\top$

b) Der (Spalten)-Vektor

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

heißt **Gradient** von f im Punkt x.

Gradient und Hessematrix

Definition 1.4 (Gradient und Hessematrix, Fortsetzung)

c) Die Definition höherer Ableitungen erfolgt analog. Für zweite Ableitungen schreibt man

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \quad \text{bzw.} \quad \frac{\partial^2 f}{\partial x_i^2}, \text{falls } i = j.$$

d) Die Matrix

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

heißt **Hessematrix** von f in x. Sie ist symmetrisch, wenn f zwei mal stetig differenzierbar ist.

Notwendige Optimalitätsbedingung

Satz 1.5 (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 gilt

$$\nabla f(x^*) = 0.$$

Ein Vektor $x \in \mathbb{R}^n$ mit $\nabla f(x) = 0$ heißt ein **stationärer Punkt** von $f(x)$ und kann entweder Minimum, Maximum oder Sattelpunkt von f sein. Gewissheit darüber geben hinreichende Optimalitätsbedingungen.

Für Funktionen $f \in \mathbb{R}^2 \to \mathbb{R}$ kann man stationäre Punkte einfach am Funktionsgraphen erkennen. An stationären Punkten verläuft die Tangentialebene

$$T(x) := f(x^*) + \nabla f(x^*)^\top (x - x^*)$$

horizontal.

Optimalitätsbedingungen 2. Ordnung

Satz 1.6 (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.*

Mit einer kleinen Zusatzvoraussetzung bekommen wir schließlich die Rückrichtung:

Satz 1.7 (Hinreichende Optimalitätsbedingung 2. Ordnung)

Es sei $f(x)$ zweimal stetig diffbar in einer Umgebung $U(x^*)$ eines Punktes x^*, und es gelte:

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

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

Beispiel: Isolierung beim Hausbau

Übung 1.8

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. Die Heizkosten sind dann proportional zur Wärmeleitfähigkeit $(1/d_1 + 1/d_2)$. Zu minimieren ist die Summe aus Investitions- und Heizkosten (über 10 Jahre):

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

Vernünftige Werte für die Konstanten sind

$$c_0 = 20000\ \mathit{EUR} \cdot \mathit{cm}, \qquad c_1 = 300\ \mathit{EUR/cm}, \qquad c_2 = 400\ \mathit{EUR/cm}.$$

Bestimme die optimalen Werte für d_1 und d_2.

Inhaltsverzeichnis

Handhabung von `fminunc`

Wir wollen die das letzte Beispiel mit dem MATLAB-Optimierer `fminunc` lösen. Nach Eingabe von `help fminunc` in MATLAB bekommen wir eine Code-Dokumentation angezeigt:

```
fminunc - Find minimum of unconstrained multivariable
    function
    Nonlinear programming solver.

    Syntax
      x = fminunc(fun,x0)
      x = fminunc(fun,x0,options)
      x = fminunc(problem)
      [x,fval] = fminunc(___)
      [x,fval,exitflag,output] = fminunc(___)
      [x,fval,exitflag,output,grad,hessian] = fminunc(___)
```

Handhabung von fminunc

Bemerkung 1.9 (zu `fminunc`)

a) *`fminunc` muss mindestens mit einem Handle zur Zielfunktion `fun` und einem Startvektor passender Dimension `x0` aufgerufen werden. Die Zielfunktion kann beispielsweise wie folgt implementiert werden:*

```
fun = @(x)(x(1)^2 + x(2)^2)
```

oder alternativ

```
fun = function(x)
  return x(1)^2 + x(2)^2
end
```

und sollte in eine separate Datei namens `fun.m` abgespeichert werden.

Bemerkung 1.9 (zu `fminunc`)

b) *Optionale Parameter (Wahl des Solvers, Parameter für Abbruchkriterium, etc.) werden im `options`-Argument festgelegt, zum Beispiel:*

```
options = optimoptions('fminunc',
                       'Algorithm','trust-region',
                       'SpecifyObjectiveGradient',true);
```

c) *Die Anzahl der Rückgabewerte ist auch variabel. Man bekommt mit `x` die Minimalstelle, mit `fval` den Zielfunktionswert, etc..*

d) *Die restlichen Rückgabetypen und Optimierungsoptionen sind in der Code-Dokumentation erläutert.*

fminunc in der Anwendung

Übung 1.10 (Lösung des Hausbau-Problems)

Löse das Bespiel von Folie 23 mit `fminunc`*.*

a) *Zeige den Iterationsverlauf an und interpretiere die Ergebnisse.*

b) *Erweitere die Zielfunktion so, dass nicht nur der Funktionswert, sondern auch der Gradient zurückgegeben wird. Wie wirkt sich dies auf den Iterationsverlauf aus?*

c) *Probiere die Option* `'Display','iter-detailed'` *aus. Was können wir dort ablesen?*

```
_____________________________________________________________
   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)
```

Inhaltsverzeichnis

Allgemeine Funktionsweise

Mit der Standard-Option `'Algorithm', 'quasi-newton'` verwendet `fminunc` ein sogenanntes **Quasi-Newton-Verfahren** mit **Liniensuche**. Das Verfahren erzeugt eine Folge von Vektoren $\{x^{(k)}\} \subset \mathbb{R}^n$, $k = 0, 1, \ldots$, wobei $x^{(0)}$ der Startwert ist, den wir als Eingabeargument übergeben haben.

In jedem Schritt wird dann:

a) ein quadratisches Modell $q^{(k)}(\mathrm{d})$ der Zielfunktion aufgebaut,
b) das Minimum $d^{(k)}$ des Modells $q^{(k)}$ bestimmt,
c) eine geeignete Schrittlänge $\alpha^{(k)} > 0$ berechnet,
d) die nächste Iterierte berechnet über

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

Einordnung des Verfahrens

Bemerkung 1.11 (Liniensuchverfahren)

Die meisten Optimierungsverfahren erzeugen ihre Iterierten über die Vorschrift $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$ *mit* ***Suchrichtung*** $d^{(k)} \in \mathbb{R}^n$ *und* ***Schrittweite*** $\alpha^{(k)} \in \mathbb{R}_+$.

a) *Für* $d^{(k)} = -\nabla f(x^{(k)})$ *erhält man das* ***Gradientenverfahren****.*

Übung: Zeige, dass $-\nabla f(x)$ *immer die steilste Abstiegsrichtung der Funktion* f *im Punkt* x *ist, also die Aufgabe*

$$\text{Minimiere} \quad \nabla f(x)^\top d \quad \text{sodass} \quad \|d\| = 1$$

löst.

b) *Für* $d^{(k)} = -\nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)})$ *erhält man das* ***Newton-Verfahren****.*

Übung: Zeige, dass die Awendung des Newton-Verfahrens zur Nullstellensuche auf die Optimalitätsbedingung $\nabla f(x) = 0$ *genau dieses Verfahren mit* $\alpha^{(k)} = 1$ *liefert.*

Das Quasi-Newton-Verfahren

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

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

und 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.

Das Quasi-Newton-Verfahren

Die Berechnung der Hessematrix ist oft aufwändig. Daher nutzt das **Quasi-Newton**-Verfahren oft ein vereinfachtes Modell

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

mit einer Matrix $B^{(k)} \approx \nabla^2 f(x^{(k)})$ (dazu später mehr).

Übung 1.12

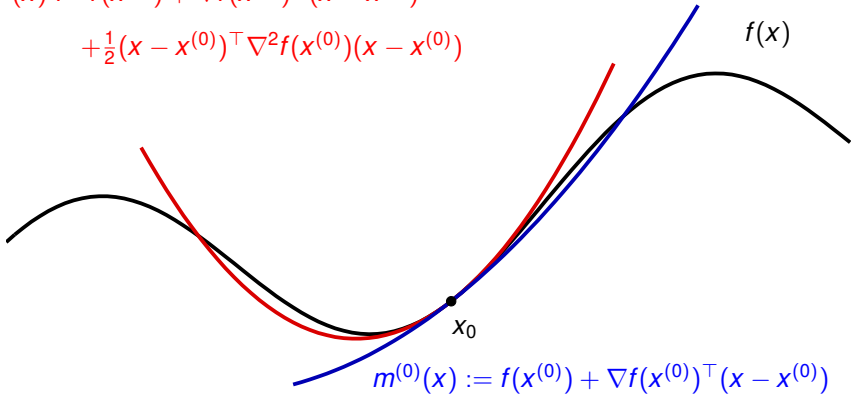
Vergewissere dich, dass das Modell (2) *die Eigenschaften*

$$m^{(k)}(x^{(k)}) = f(x^{(k)}) \quad \textit{und} \quad \nabla m(x^{(k)}) = \nabla f(x^{(k)})$$

erfüllt. Wie lautet $\nabla^2 m(x^{(k)})$*?*

Das Quasi-Newton-Verfahren

$$M^{(0)}(x) := f(x^{(0)}) + \nabla f(x^{(0)})^\top (x - x^{(0)}) + \tfrac{1}{2}(x - x^{(0)})^\top \nabla^2 f(x^{(0)})(x - x^{(0)})$$



$$m^{(0)}(x) := f(x^{(0)}) + \nabla f(x^{(0)})^\top (x - x^{(0)}) + \tfrac{1}{2}(x - x^{(0)})^\top B^{(0)} (x - x^{(0)})$$

Das Quasi-Newton-Verfahren

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 $q^{(k)}(d)$:

$$\begin{aligned} q^{(k)}(d) &:= m^{(k)}(x^{(k)} + d) \\ &= f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top B^{(k)}\, d. \end{aligned} \tag{3}$$

Im Teilschritt (2) soll das Minimum der quadratischen Modellfunktion $q^{(k)}(d)$ mit $d \in \mathbb{R}^n$ bestimmt werden.

Die Optimalitätsbedingung 1. Ordnung (Theorem 1.5) sagt aus, dass bei einem Minimum d^* von (3) auf jeden Fall

$$\nabla q^{(k)}(d^*) = \nabla f(x^{(k)}) + B^{(k)}\, d^* \stackrel{!}{=} 0 \qquad \Longrightarrow \qquad B^{(k)}\, d^* = -\nabla f(x^{(k)})$$

Das ist ein **lineares Gleichungssystem** zur Berechnung der Richtung $d^* = d^{(k)} \in \mathbb{R}^n$.

Das Quasi-Newton-Verfahren

Übung 1.13

Zeige, dass d^* stets eine Abstiegsrichtung von f im Punkt $x^{(k)}$ ist, d. h., dass außer in stationären Punkten immer $\nabla f(x^{(k)})^\top d^* < 0$ gilt. Welche Bedingung an $B^{(k)}$ ist dafür nötig?

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

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

Achtung: Man könnte auf die Idee kommen $\alpha = 1$ zu wählen. Dann wäre $x^{(k+1)} = x^{(k)} + d^{(k)}$ immer das Minimum des quadratischen Modells $m(x)$. Dieses naive Vorgehen scheitert allerdings schon an einfachen Beispielen.

Deshalb wird in Schritt (c) des Verfahrens noch eine sogenannte **Liniensuche** zur Bestimmung einer geeigneten Schrittweite $\alpha^{(k)}$ durchgeführt.

Liniensuche

Wir sehen uns die Liniensuchfunktion an:

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

und stellen uns die Frage, wie weit wir ausgehend von $x^{(k)}$ in Richtung $d^{(k)}$ gehen sollen.

Die Steigung dieser Funktion an einer Stelle α ist (nach Kettenregel)

$$\varphi'(\alpha) = \nabla f(x^{(k)} + \alpha\, d^{(k)})^\top d^{(k)} \quad \Rightarrow \quad \varphi'(0) = \nabla f(x^{(k)})^\top d^{(k)} < 0.$$

Es ist den Aufwand nicht wert, ein Minimum von $\varphi(\alpha)$ exakt zu bestimmen. Wir fordern:

Definition 1.14 (Armijo-Bedingung)

Eine Schrittweite $\alpha > 0$ erfüllt die sogenannte **Armijo-Bedingung** mit Parameter $0 < \sigma < 1$, falls

$$f(x^{(k)} + \alpha\, d^{(k)}) \leq f(x^{(k)}) + \sigma\, \alpha\, \nabla f(x^{(k)})^\top d^{(k)} \quad \text{bzw.} \quad \varphi(\alpha) \leq \varphi(0) + \sigma\, \alpha\, \varphi'(0). \qquad (4)$$

Liniensuche

Während die **Armijo-Bedingung** für Gradienten- und Newton-Verfahren ausreichend ist benötigen wir im Kontext von **Quasi-Newton-Verfahren** noch eine weitere Bedingung:

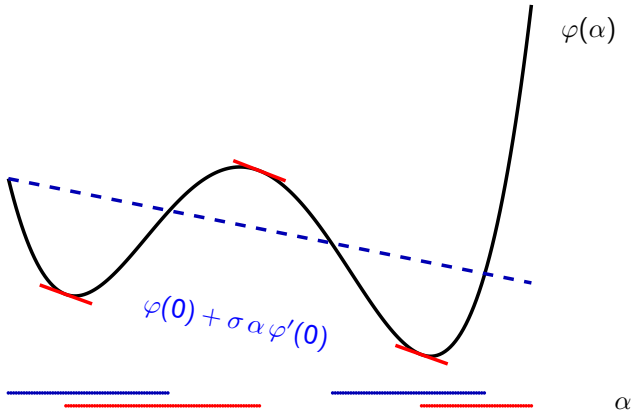
Definition 1.15 (Krümmungsbedingung)

Eine Schrittweite α erfüllt die sogenannte **Krümmungsbedingung** mit Parameter $0 < \sigma < \tau < 1$, falls

$$\nabla f(x^{(k)} + \alpha d^{(k)})^\top d^{(k)} \geq \tau \nabla f(x^{(k)})^\top d^{(k)} \quad \text{bzw.} \quad \varphi'(\alpha) \geq \tau \varphi'(0). \tag{5}$$

Ziel: Finde eine Schrittweite $\alpha^{(k)}$, welche sowohl **Armijo-** als auch **Krümmungsbedingung** erfüllen. Dabei sollen so wenige α's wie möglich ausprobiert werden.

Liniensuche



- Blauer Bereich: Armijo erfüllt,
- Roter Bereich: Krümmungsbedingung erfüllt.

Liniensuche

Übung 1.16

Interpretiere die Ausgabe von `fminunc`.

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)
```

Liniensuche

Bemerkung 1.17

a) *"Robuste" (auf den vielfachen praktischen Einsatz getrimmte) Optimierungsverfahren wie* `fminunc` *verwenden ausgefeilte Strategien um eine zulässige Schrittweite* $\alpha^{(k)}$ *zu finden.*

b) *Jedes "ausprobieren" einer Schrittweite* α *erfordert eine Funktionsauswertung, nämlich von* $f(x^{(k)} + \alpha\, d^{(k)})$.

c) *Ableitungen werden, falls nicht in der Zielfunktion definiert, automatisch über Differenzenquotienten der Form*

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \varepsilon\, e_j) - f(x)}{\varepsilon}$$

mit $\varepsilon \ll 1$ *approximiert. Beachte, dass dies n viele sind. Dies erklärt die 3-er Schritte in der Spalte* `Func-count` *auf der letzten Folie.*

Update von $B^{(k)}$

Noch zu klären bleibt wie die Approximation $B^{(k)}$ der Hessematrix $\nabla^2 f(x^{(k)})$ gesetzt wird.

Wenn etwa der Schritt $x^{(k)} \rightsquigarrow x^{(k+1)}$ 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. (2):

$$m^{(k+1)}(x) = f(x^{(k+1)}) + \nabla f(x^{(k+1)})^\top (x - x^{(k+1)}) + \frac{1}{2}(x - x^{(k+1)})^\top B^{(k+1)} (x - x^{(k+1)}) \quad (6)$$

Es gilt wieder (vgl Übung 1.12)

- $m^{(k+1)}(x^{(k+1)}) = f(x^{(k+1)})$
- $\nabla m^{(k+1)}(x^{(k+1)}) = \nabla f(x^{(k+1)})$

Update von $B^{(k)}$

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:

Definition 1.18 (Sekantenbedingung)

Man sagt $m^{(k+1)}$ erfülle die sogenannte **Sekantenbedingung**, falls

$$\begin{aligned} & \nabla m^{(k+1)}(x^{(k)}) = \nabla f(x^{(k)}) \\ \Leftrightarrow \quad & \nabla f(x^{(k+1)}) + B^{(k+1)}\,(x^{(k)} - x^{(k+1)}) = \nabla f(x^{(k)}) \\ \Leftrightarrow \quad & B^{(k+1)}\,(x^{(k+1)} - x^{(k)}) = \nabla f(x^{(k+1)}) - \nabla f(x^{(k)}). \end{aligned} \tag{7}$$

Dies liefert n zu erfüllende Bedingungen an $B^{(k+1)}$.

Update von $B^{(k)}$

Da auch die Sekantenbedingung (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):

$$B_{\text{BFGS}}^{(k+1)} = B_{\text{BFGS}}^{(k)} - \frac{B_{\text{BFGS}}^{(k)} s^{(k)} (s^{(k)})^\top B_{\text{BFGS}}^{(k)}}{(s^{(k)})^\top B_{\text{BFGS}}^{(k)} s^{(k)}} + \rho^{(k)} y^{(k)} (y^{(k)})^\top, \tag{8}$$

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.

Update von $B^{(k)}$

Lemma 1.19 (Positive Definitheit des BFGS-Updates)

Es sei $B_{BFGS}^{(k)}$ positiv definit. Wenn die Bedingung $(s^{(k)})^\top y^{(k)} > 0$ erfüllt ist, dann ist auch $B_{BFGS}^{(k+1)}$ wieder positiv definit.

Die offenbar hier geforderte Bedingung $(s^{(k)})^\top y^{(k)} > 0$ ist durch die Erfüllung der Krümmungsbedingung (5) bei der Liniensuche sichergestellt.

Bemerkung 1.20

*Für die Bestimmung der Suchrichtung $d^{(k)}$ muss in jeder Iteration das lineare Gleichungssystem $B^{(k)}d^{(k)} = -\nabla f(x^{(k)}) \iff d^{(k)} = -(B^{(k)})^{-1}\nabla f(x^{(k)})$ gelöst werden. Zur Effizienzsteigerung kann man auch mit der inversen Matrix $H^{(k)} := (B^{(k)})^{-1}$ arbeiten, welche sich mit der **inversen BFGS-Formel** berechnen lässt:*

$$H_{BFGS}^{(k+1)} = (I - \rho^{(k)}\, s^{(k)}\, (y^{(k)})^\top)\, H_{BFGS}^{(k)}\, (I - \rho^{(k)}\, y^{(k)}\, (s^{(k)})^\top) + \rho^{(k)}\, s^{(k)}\, (s^{(k)})^\top. \qquad (9)$$

Zusammenfassung

a) 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.

b) Die quadratischen Modelle benutzen anstelle der exakten Hessematrix der Zielfunktion bestimmte symmetrisch positiv definite Matrizen $B^{(k)}$. Diese ergeben sich jeweils durch ein leicht zu berechnendes Update (hier die BFGS-Formel) aus der vorangegangenen Matrix.

c) Das Verfahren wählt als anfängliche Hessematrix $B^{(0)}$ ein Vielfaches der Einheitsmatrix. Man kann diese jedoch auch benutzerdefiniert vorgeben.

d) Durch dieses Vorgehen
   - spart man sich erstens die Berechnung zweiter Ableitungen der Zielfunktion $f(x)$
   - und stellt sicher, dass die Matrizen $B^{(k)}$ positiv definit sind, wodurch die gemäß $B^{(k)}d^{(k)} = -\nabla f(x^{(k)})$ berechnete Suchrichtung $d^{(k)}$ eine Abstiegsrichtung für die Zielfunktion ist.

Inhaltsverzeichnis

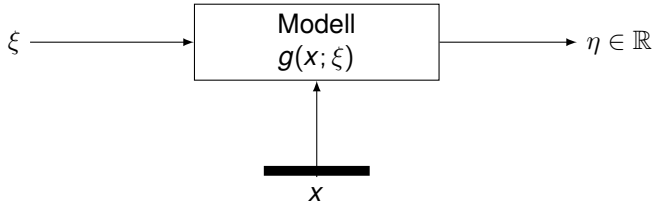
Inhaltsverzeichnis

Einführung

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:

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



Einführung

Aufgabe: Anhand von vorliegenden Messdaten $(\xi_i, \eta_i) \in \mathbb{R}^r \times \mathbb{R}$, $i = 1, \ldots, m$ soll der Parameter $x \in \mathbb{R}^n$ so gewählen werden, dass das Modell den Messdaten möglichst gut entspricht.

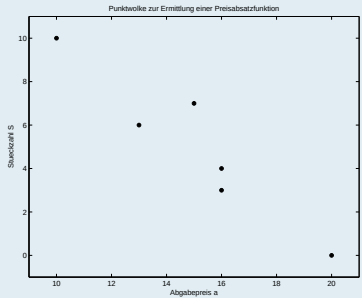
Das Auffinden des (in einem noch zu präzisierenden Sinne) besten Parametervektors $x \in \mathbb{R}^n$ wird als **Parameteridentifikation**, **Parameterschätzung**, **Modellkalibrierung** oder **Ausgleichsrechnung** bezeichnet.

Beispiel

Beispiel 2.1 (Kalibrierung einer Preis-Absatz-Funktion)

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”:

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



Beispiel

Beispiel 2.1 (Fortsetzung)

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 oder aus Vorwissen lässt sich die Vermutung gewinnen, dass ein linearer Zusammenhang zwischen S und a bestehen könnte:

$$S = g(x; a) := x_1 + x_2\, a. \tag{10}$$

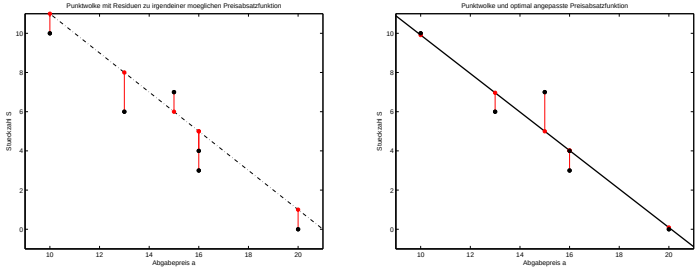
Dieser Zusammenhang ist unser Modell, das die noch unbekannten Parameter $x = (x_1, x_2) \in \mathbb{R}^2$ enthält.

Optimierungsaufgabe für Parameteridentifikation

Wir suchen nun die *optimalen* Parameter x, also diejenige Preis-Absatz-Funktion, bei der die Daten am besten "gefittet" werden. Als Maß für die Güte des Fits nutzen wir die **Residuen**

$$r_i(x) := \underbrace{g(x; a_i)}_{\text{Vorhersage des Modells}} - \underbrace{S_i}_{\text{Messwert}} \quad \text{für } i = 1, \ldots, m. \tag{11}$$

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).



Optimierungsaufgabe für Parameteridentifikation

Die 6 gegebenen Datenpaare liefern und also 6 Residuen $r_1, \ldots, r_6$. Diese sollen möglichst klein im Sinne der **kleinsten Fehlerquadratsumme** werden, das heißt, wie minimieren

$$
\begin{aligned}
f(x) &:= \sum_{i=1}^{m} |r_i(x)|^2 = \sum_{i=1}^{m} |g(x; a_i) - S_i|^2 = \sum_{i=1}^{m} \big(g(x; a_i) - S_i\big)^2 \\
&= \left\| \begin{pmatrix} g(x; a_1) - S_1 \\ \vdots \\ g(x; a_m) - S_m \end{pmatrix} \right\|_2^2 .
\end{aligned} \tag{12}
$$

Übung 2.2

Denke über andere mögliche Zielfunktionen nach.

Optimierungsaufgabe für Parameteridentifikation

Definition 2.3 (KQ-Aufgabe)

Eine Optimierungsaufgabe der Form

$$\text{Minimiere} \quad \sum_{i=1}^{m} \big(g(x;\xi_i) - \eta_i\big)^2, \quad x \in \mathbb{R}^n. \tag{13}$$

zu gegebenen Datenpaaren $(\xi_i, \eta_i) \in \mathbb{R}^r \times \mathbb{R}$, $i = 1, \ldots, m$, heißt eine **Kleinste-Quadrate-Aufgabe**.

In unserem ersten Beispiel sollte ein linearer Zusammenhang (10) zwischen $\xi = a$ und $\eta = S$ angenommen werden. Also lautet die Aufgabe genauer:

$$\text{Minimiere} \quad \sum_{i=1}^{m} \big(x_1 + x_2\, a_i - S_i\big)^2, \quad x \in \mathbb{R}^2. \tag{14}$$

Inhaltsverzeichnis

Vorüberlegung

(13) ist im Prinzip ein freies Optimierungsproblem, welche wir in Kapitel 1 ausführlich diskutiert haben. Wir nutzen hier allerdings nicht `fminunc`, da es bessere Verfahren gibt, die die spezielle Stuktur der Optimierungsaufgabe besser ausnutzen können.

Auch die Optimalitätsbedingungen aus den Sätzen 1.5 und 1.7 bleiben natürlich gültig. Betrachte dazu die äquivalente Darstellung von (14) mit Residuenvektor $r : \mathbb{R}^n \to \mathbb{R}^m$,

$$\text{Minimiere} \quad f(x) := ||r(x)||^2 = r(x)^\top r(x), \quad x \in \mathbb{R}^n. \tag{15}$$

Die notwendige Optimalitätsbedingung 1. Ordnung sieht wie folgt aus (Kettenregel):

$$\nabla f(x) = 2\,J(x)^\top r(x) = 0 \tag{16}$$

wobei $J(x) = r'(x)$ die Jacobi-Matrix von r ist.

Übung 2.4

Stelle die Optimalitätsbedingung für Beispiel 2.1 auf.

Aufruf von `lsqcurvefit`

Der Befehl `help lsqcurvefit` gibt uns folgende Ausgabe:

```
lsqcurvefit - Solve nonlinear curve-fitting (data-fitting)
   problems in least-squares sense
Nonlinear least-squares solver

Syntax
  x = lsqcurvefit(fun,x0,xdata,ydata)
  x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub)
  x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,A,b,Aeq,beq)
  x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,A,b,Aeq,beq,
     nonlcon)
  x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options)
  x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,A,b,Aeq,beq,
     nonlcon,options)
  x = lsqcurvefit(problem)
  [x,resnorm] = lsqcurvefit(___)
  [x,resnorm,residual,exitflag,output] = lsqcurvefit(___)
  [x,resnorm,residual,exitflag,output,lambda,jacobian] =
     lsqcurvefit(___)
```

Ein- und Ausgabeparameter von `lsqcurvefit`

Bemerkung 2.5 (zu `lsqcurvefit`)

a) *Die Pflicht-**Eingabeparameter** von `lsqcurvefit` sind*
   - `fun` *- Ein Handle auf die Funktion $g(x, a)$, in unserem Beispiel*

```
g = @(x,a) (x(1) + x(2)*a);
```

   - `x0` *- Eine Startschätzung für den Parameter $x \in \mathbb{R}$.*
   - `xdata` *- Das ξ-Argument aller Messwerte (als $m \times r$-Matrix).*
   - `ydata` *- Das η-Argument aller Messwerte (als $m \times 1$-Matrix).*

b) *Die **Ausgabeparameter** sind*
   - `x` *- Der Vektor der optimalen Parameter x^*.*
   - `resnorm` *- Das Residuum $\|r(x^*)\|$.*
   - `residual` *- Der Vektor der Residuen $r_i(x^*)$.*

Die restlichen Eingabeparameter dienen um zusätzliche Nebenbedingungen vorzugeben (hier nicht benötigt), außerdem lässt sich mit `options` wieder eine Struktur übermitteln, die diverse Einstellungen für den Optimierungsalgorithmus liefert.

Inhaltsverzeichnis

Das Levenberg-Marquardt-Verfahren

Das hier verwendete Verfahren ist ähnlich zum Quasi-Newton-Verfahren. Es verwendet ein quadratisches Modell zentriert in $x^{(k)}$:

$$\begin{aligned} q_{\mathrm{LM}}^{(k)}(d) &:= f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top B_{\mathrm{LM}}^{(k)} d \\ &\overset{(16)}{=} \frac{1}{2} r(x^{(k)})^\top r(x^{(k)}) + r(x^{(k)})^\top J(x^{(k)})\, d + \frac{1}{2} d^\top B_{\mathrm{LM}}^{(k)} d. \end{aligned} \tag{17}$$

Anstelle der exakten Hessematrix (Übung: $\nabla^2 f(x) = J(x)^\top J(x) + \sum_{i=1}^m r_i(x)\, \nabla^2 r_i(x)$) wird die Approximation

$$B_{\mathrm{LM}}^{(k)} := J(x^{(k)})^\top J(x^{(k)}) + \lambda^{(k)} I \tag{18}$$

verwendet.

Beachte: Der erste Teil ist auch Teil der exakten Hessematrix, der zweite Teil wird durch ein Vielfaches ($\lambda^{(k)} > 0$) der Einheitsmatrix ersetzt.

Das Levenberg-Marquardt-Verfahren

Wir nutzen also

$$B_{\mathrm{LM}}^{(k)} = J(x^{(k)})^\top J(x^{(k)}) + \lambda^{(k)} I$$
$$\approx \nabla^2 f(x^{(k)}) = J(x^{(k)})^\top J(x^{(k)}) + \sum_{i=1}^{m} r_i(x^{(k)}) \nabla^2 r_i(x^{(k)}). \tag{19}$$

Bemerkung 2.6 (Approximation der Hessematrix)

a) *Man nutzt den "guten" ersten Anteil $J(x)^\top J(x)$ der echten Hessematrix* (19)*. Dieser ist positiv semidefinit und lässt sich ohne zusätzlichen Aufwand bestimmen ($J(x)$ wird bereits die erste Ableitung $\nabla f(x)$ benötigt).*

b) *Den "schlechten" Anteil $\sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x)$ der Hessematrix* (19) *lässt man weg. Dieser wäre recht aufwändig zu berechnen und könnte die positive Definitheit zerstören. Weiterhin sollten in einer Lösung x^* die Residuen $r_i(x^*)$ ohnehin betragsmäßig klein sein.*

c) *Stattdessen addiert man den Ersatzterm $\lambda^{(k)} I$. Dieser sorgt durch $\lambda^{(k)} > 0$ für die positive Definitheit der Modell-Hessematrix $B_{LM}^{(k)}$.*

Das Levenberg-Marquardt-Verfahren

Wie schon beim Quasi-Newton-Verfahren ist man in der k-ten Iteration am Minimum des quadratischen Modells (17) interessiert. Wegen der positiven Definitheit von $B_{\mathrm{LM}}^{(k)}$ ist dieses Minimum eindeutig und lässt sich über das lineare Gleichungssystem

$$B_{\mathrm{LM}}^{(k)} d^{(k)} = -\nabla f(x^{(k)}) \tag{20}$$

und hier speziell

$$\left[J(x^{(k)})^\top J(x^{(k)}) + \lambda^{(k)} I\right] d^{(k)} = -J(x^{(k)})^\top r(x^{(k)}) \tag{21}$$

berechnen. Dies liefert die **Suchrichtung** $d^{(k)}$.

Wir könnten also jetzt wie beim Quasi-Newton-Verfahren 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 $\lambda^{(k)}$.

Das Levenberg-Marquardt-Verfahren

Wir betrachten dazu die Bestimmungsgleichung (21) für $d^{(k)}$:

a) Ist $\lambda^{(k)}$ sehr groß, so lautet (21) in etwa

$$\lambda^{(k)} d^{(k)} \approx -J(x^{(k)})^\top r(x^{(k)}) \quad \text{bzw.} \quad d^{(k)} \approx -\frac{1}{\lambda^{(k)}} \underbrace{J(x^{(k)})^\top r(x^{(k)})}_{\nabla f(x^{(k)})}. \tag{22}$$

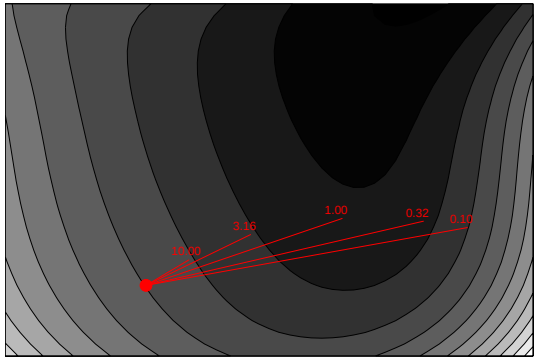
Das bedeutet, der Schritt $d^{(k)}$ entspricht einem kleinen Schritt in Richtung des negativen Gradienten, also einem kleinen Schritt in Richtung des steilsten Abstiegs.

b) Ist $\lambda^{(k)}$ dagegen sehr klein, so lautet (21) in etwa

$$J(x^{(k)})^\top J(x^{(k)})\, d^{(k)} = -J(x^{(k)})^\top r(x^{(k)}). \tag{23}$$

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 (19) verwendet wird.

Das Levenberg-Marquardt-Verfahren



Durch die Wahl von $\lambda^{(k)}$ kann stufenlos zwischen dem “vorsichtigen” Gradientenschritt (22) und dem “mutigeren” vereinfachten Newton-Schritt (23) hin- und hergeschaltet werden. Deshalb wird $\lambda^{(k)}$ auch als **Dämpfungsparameter** bezeichnet.

Das Levenberg-Marquardt-Verfahren

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:

$$\rho^{(k)} = \frac{\text{in der Zielfunktion } f(x) \text{ tatsächlich erreichter Abstieg}}{\text{durch die Modellfunktion } q_{\mathrm{LM}}^{(k)}(d) \text{ vorhergesagter Abstieg}} = \frac{f(x^{(k)}) - f(x^{(k)} + d^{(k)})}{q_{\mathrm{LM}}^{(k)}(0) - q_{\mathrm{LM}}^{(k)}(d^{(k)})}. \tag{24}$$

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

1. 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. $\lambda^{(k+1)} := \lambda^{(k)}/3$.
2. Ist $0.25 \leq \rho^{(k)} < 0.75$, dann ist der Schritt $x^{(k)} \rightsquigarrow x^{(k)} + d^{(k)}$ immerhin mäßig erfolgreich. Wir wählen $\lambda^{(k+1)} := \lambda^{(k)}$.
3. 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. $\lambda^{(k+1)} := 2\,\lambda^{(k)}$.

Das Levenberg-Marquardt-Verfahren

Wir fassen den Algorithmus zusammen:

1. Aufbau des quadratischen Modells $q_{\mathrm{LM}}^{(k)}(d)$ der Zielfunktion, siehe (17) und (18), dazu werden $r(x^{(k)})$ und $J(x^{(k)})$ ausgewertet.
2. Bestimme das eindeutige Minimum $d^{(k)} \in \mathbb{R}^n$ dieses Modells durch Lösen des linearen Gleichungssystems (20)
3. Berechne den Fortschrittsquotienten $\rho^{(k)}$ in (24)
4. Wähle den nächsten Dämpfungsparameter $\lambda^{(k+1)}$ in Abhängigkeit von $\rho^{(k)}$ wie auf Folie 67 beschrieben.
5. Setze die nächste Iterierte:

$$\begin{aligned} &\text{falls } \rho^{(k)} > 0\text{:} \quad x^{(k+1)} := x^{(k)} + d^{(k)} \quad && \text{(Schritt wird akzeptiert)} \\ &\text{falls } \rho^{(k)} \leq 0\text{:} \quad x^{(k+1)} := x^{(k)} && \text{(Schritt wird verworfen)} \end{aligned}$$

Ausgabe von `lsqcurvefit`

```
                                                                    
   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

                                          First-Order                    Norm of
 Iteration  Func-count     Residual       optimality       Lambda           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.
```

Abschließende Bemerkungen

Bemerkung 2.7

a) *Ähnlich wie bei* `fminunc` *können wir auch den Gradienten des Modells (bez. der Parameter x) bereitstellen. In diesem Fall muss die Funktion* `g` *den Gradienten als zweites Argument zurückgeben und es muss*
`options.SpecifyObjectiveGradient = true` *gesetzt sein.*

b) *Die Jacobi-Matrix* $J \in \mathbb{R}^{m \times n}$ *sollte eine "schlanke" Matrix sein, es sollte also* $m \gg n$ *gelten (mehr Datenpunkte als Parameter). In unserem Beispiel war* $m = 6 > 2 = n$.

c) *Zu komplexe Modelle (z.B. Polynome vom Grad 100) sind oft schlecht geeignet. Es kommt häufig zum "overfitting" und Ausreißer in den Daten verzerren das Modell stark.*

d) *In vielen Anwendungen kann man Beschränkungen der Form* $\ell \leq x \leq u$ *an die Parameter fordern. Dies ist auch in* `lsqcurvefit` *vorgesehen (weitere Eingabeparameter).*

Inhaltsverzeichnis

Inhaltsverzeichnis

Einführung

Lineare Optimierungsprobleme haben eine Vielzahl von Anwendungen insbesondere in den Wirtschaftswissenschaften. Derartige Probleme entstehen beispielsweise bei der Modellierung von

- Produktionsplanung
- Kostenrechnung
- Netzwerk- und Transportproblemen

Die Besonderheit ist dabei, dass sowohl Zielfunktion f als auch die Funktionen g_i und h_j aus den Nebenbedingungen **linear** sind. Insbesondere die Präsenz der Nebenbedingungen unterscheidet die die linearen Optimierungsaufgaben von denen, die wir im vorherigen Kapitel untersucht haben.

Beispiel: Mozartproblem

Beispiel 3.1 (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	

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

Beispiel: Mozartproblem

- Die **Optierungsvariablen** sind die Menge an Mozartkugeln x_1 und an Mozarttalern x_2.
- Die **Zielfunktion** lautet

$$f(x) = 9\,x_1 + 8\,x_2 \to \max!$$

- Als **Nebenbedingung** haben wir

$$\begin{aligned} x_1 + x_2 &\leq 6 && \text{(Marzipanbeschränkung)} \\ 2\,x_1 + x_2 &\leq 11 && \text{(Nougatbeschränkung)} \\ x_1 + 2\,x_2 &\leq 9 && \text{(Bitterschokoladenbeschränkung)} \end{aligned}$$

sowie die **Nichtnegativitätsbedingungen**

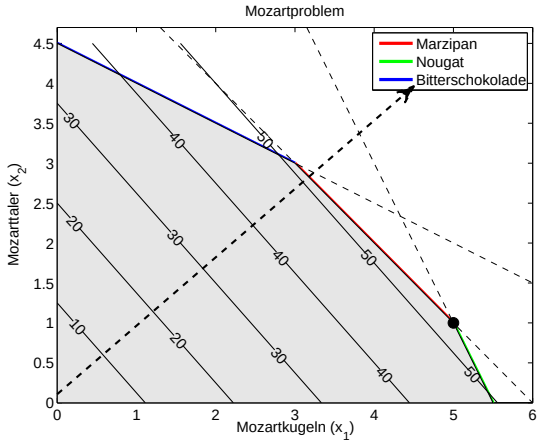
$$x_1, x_2 \geq 0.$$

- Kompakt:

$$\text{Maximiere} \quad \begin{pmatrix} 9 \\ 8 \end{pmatrix}^\top \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{sodass} \quad \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix}, \quad x_1, x_2 \geq 0.$$

Beispiel: Mozartproblem

Aufgrund der geringen Dimension ($n = 2$) lässt sich dieses Problem leicht grafisch lösen:



Allgemeine Lineare Optimierungsaufgaben

Definition 3.2 (Lineares Optimierungsproblem)

Eine Aufgabe der Form

$$\begin{aligned} \text{Minimiere} \quad & c^\top x, \quad x \in \mathbb{R}^n \\ \text{unter} \quad & A_{\text{ineq}}\, x \leq b_{\text{ineq}} \\ \text{und} \quad & A_{\text{eq}}\, x = b_{\text{eq}} \\ \text{sowie} \quad & \ell \leq x \leq u. \end{aligned} \tag{25}$$

heißt **lineares Optimierungsproblem**. Dabei sind A_{ineq}, A_{eq} bzw. c, b_{ineq}, b_{eq}, ℓ, u Matrizen bzw. Vektoren passender Dimension.

Der Vektor c wird auch als **Kostenvektor** bezeichnet. Die Beschränkungen $\ell \leq x \leq u$ werden auch **Box-Beschränkungen** genannt. Man kann diese zwar auch als Ungleichungsnebenbedingung auffassen, aber einige Optimierungsverfahren können diese Struktur sogar noch besser ausnutzen.

Allgemeine Lineare Optimierungsaufgaben

Definition 3.3 (Optima bei linearen Programmen)

a) Die Menge

$$X := \{x \in \mathbb{R}^n : A_{\text{ineq}}\, x \leq b_{\text{ineq}},\ A_{\text{eq}}\, x = b_{\text{eq}},\ \ell \leq x \leq u\} \tag{26}$$

heißt **zulässige Menge** von (25).

b) Ein Vektor $x^* \in \mathbb{R}^n$ heißt ein **(globales) Minimum** von (25), wenn gilt:

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

c) Ein globales Minimum x^* heißt **strikt**, wenn gilt:

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

Ein Minimum nennt man auch häufig **Optimum**, **Minimalstelle** oder **(Optimal)lösung** von (25).

Inhaltsverzeichnis

Verwendung von linprog

Für lineare Optimierungsaufgaben (oder: lineare Programme) stellt Matlab die Funktion `linprog` bereit. Aus der Dokumentation erfahren wir mit welchen Argumenten die Funktion aufzurufen ist:

```
linprog - Solve linear programming problems
    Linear programming solver

    Syntax
      x = linprog(f,A,b)
      x = linprog(f,A,b,Aeq,beq)
      x = linprog(f,A,b,Aeq,beq,lb,ub)
      x = linprog(f,A,b,Aeq,beq,lb,ub,options)
      x = linprog(problem)
      [x,fval] = linprog(___)
      [x,fval,exitflag,output] = linprog(___)
      [x,fval,exitflag,output,lambda] = linprog(___)
```

Verwendung von linprog

- (Optionale) **Eingabeparameter** sind
  - `f`: Der Kostenvektor c
  - `A` und `b`: Die Matrix A_{ineq} und der Vektor b_{ineq}
  - `Aeq` und `beq`: Die Matrix A_{eq} und der Vektor b_{eq}
  - `lb` und `ub`: Die Vektoren ℓ und u (Setze `+/-Inf` falls nicht vorhanden)
  - `options`: Zusätzliche Optionen für den Optimierer (siehe Dokumentation)
- Die **Ausgabeparameter** sind
  - `x`: Das Optimum x^*
  - `fval`: Der Zielfunktionswert am Optimum
  - ...

Verwendung von linprog

Angewendet auf unser Mozart-Problem sieht das so aus:

```
A = [1,1; 2,1; 1,2];
b = [6; 11; 9];
c = [-9; -8];
linprog(c, A, b)
```

und wir erhalten die Ausgabe

```
Optimal solution found.

ans =

    5.0000
    1.0000
```

Verwendung von linprog

Nicht vorhandene Parameter werden einfach durch `[]` ersetzt. Möchte man zum Beispiel den 8. Parameter `options` setzen, würde der Aufruf so aussehen:

```
options = optimoptions('linprog');
options
options.Display = 'iter'
linprog(c, A, b, [], [], [], [], options)
```

und Matlab gibt genauere Informationern zum Iterationsverlauf zurück.

Inhaltsverzeichnis

Normalform von LP's

`linprog` nutzt eine Variante des **Simplex-Verfahrens**, welches wir hier in der einfachsten Form genauer untersuchen möchten.

Dieses Verfahren ist für Probleme folgender Bauart gemacht:

Definition 3.4 (Lineare Optimierungsaufgabe in Normalform)

Ein lineares Optimierungsproblem in **Normalform** ist ein Problem der Gestalt

$$\begin{aligned} \text{Minimiere} \quad & c^\top x, \quad x \in \mathbb{R}^n \\ \text{unter} \quad & Ax = b \\ \text{sowie} \quad & x \geq 0, \end{aligned} \tag{27}$$

wobei $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$. Annahme: $m \leq n$.

Offensichtlich sind hier nur Gleichungsnebenbedingungen, sowie die Nichtnegativitätsbedingungen übrig geblieben. Ist dies eine Einschränkung der Problemklasse?

Umformen auf Normalform

Jedes allgemeine LP der Form (25) lässt sich auf Normalform bringen:

- Eine **Ungleichungsnebenbedingung** der Form $a^\top x \leq 0$ lässt sich durch Addition einer **Schlupfvariablen** $s \geq 0$ in eine Gleichungsnebenbedingung überführen:

$$a^\top x + s = 0.$$

  Beachte: s ist nun ebenfalls Optimierungsvariable! Wir haben damit auch die Dimension des Problems erhöht.

- Falls die **Nichtnegativitätsbedingung** $x_i \geq 0$ fehlt gibt es 2 Möglichkeiten:
  - Falls es eine untere Box-Beschränkung $\ell_i \leq x_i \iff x_i - \ell_i \geq 0$ gibt subtituiert man einfach $x_i - \ell_i =: \widetilde{x}_i$. Diese Variable erfüllt nun automatisch $\widetilde{x}_i \geq 0$.
  - Andernfalls zerlege x_i in $x_i = x_i^+ - x_i^-$ und fordere $x_i^+, x_i^- \geq 0$.

  Beachte: Diese Substitutionen müssen in Zielfunktion und allen anderen Nebenbedingungen ebenfalls durchgeführt werden.

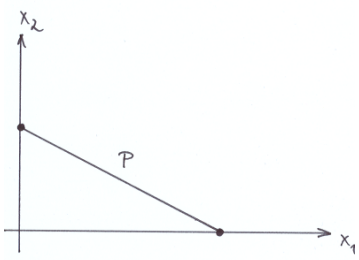
Übung 3.5

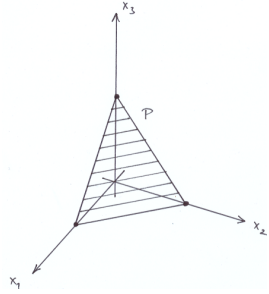
Bringe das Mozartproblem aus Beispiel 3.1 auf Normalform.

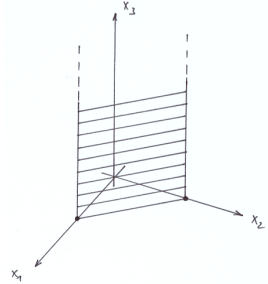
Das Simplex-Verfahren

Beobachtung: Die Optimallösungen einer linearen Optimierungsaufgabe sind in den Ecken der zulässigen Menge zu suchen. Dies gilt auch für Aufgaben in Normalform (27).

Hauptidee: Laufe von einer Ecke zu einer benachbarten Ecke, bis es keine bessere mehr gibt. Dazu muss das Verfahren in der Lage sein, Ecken zu berechnen. Da Ecken extremale 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:







Charakterisierung der Ecken

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

- Wähle eine Indexmenge $B \subset \{1, \ldots, n\}$, die sog. **Basis**, mit m Elementen ($|B| = m$).
- Alle anderen Indizes steckt man in die **Nichtbasis** $N = \{1, \ldots, n\} \setminus B$ ($|N| = n - m$).
- Die Menge B muss so gewählt sein, dass die zugehörigen Spalten der Matrix A linear unabhängig sind. Die Untermatrix, die nur aus den Spalten a_i mit $i \in B$ besteht bezeichnen wir mit A_B, genannt **Basismatrix**. Analog definieren wir A_N.

  Beispiel: Für $B = \{1, 3\}$ gilt

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \Rightarrow \quad A_B = \begin{pmatrix} 1 & 3 \\ 4 & 6 \end{pmatrix}, \quad A_N = \begin{pmatrix} 2 \\ 5 \end{pmatrix}.$$

Charakterisierung der Ecken

- Ebenso wie die Spalten der Matrix $A = [A_B, A_N]$ partitionieren wir Vektoren $x \in \mathbb{R}^n$ in $x = [x_B, x_N]$.
- Eine **Ecke** erhalten wir dann durch Setzen von

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

und x_B berechnen wir aus dem linearen Gleichungssystem

$$A_B x_B + A_N \underbrace{x_N}_{=0} = b, \quad \text{also} \quad A_B x_B = b. \tag{28}$$

- $x = [x_B, x_N]$ ist dann Ecke der zul. Menge, wenn $x_B \geq 0$ gilt.

Charakterisierung der Ecken

Übung 3.6

Berechne alle Ecken des Mozartproblems. Es gilt

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix} \quad \textit{und} \quad b = \begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix}$$

Theoretisch könnten wir alle zugehörigen Funktionswerte berechnen und damit schon das Minimum bestimmen. Für hochdimensionale Probleme ist das allerdings sehr teuer, denn man müsste

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

potentielle Ecken ausrechnen.

Der Simplex-Schritt - Reduzierte Kosten

Angenommen wir haben eine zulässige Ecke x zur Basis B und Nichtbasis N gegeben. Wir können wir ausgehend davon eine neue Ecke mit kleinerem Zielfunktionswert berechnen?

Sei x eine Ecke von X zur Basis B. Für einen beliebigen Punkt $z = [z_B, z_N] \in \mathbb{R}^n$ zeigen wir zunächst:

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

Den Vektor Δ_N bezeichnet man als die **reduzierten Kosten** zur Nichtbasis N.

Der Simplex-Schritt - Reduzierte Kosten

Für den Vergleich der Funktionswerte gilt also:

$$\underbrace{c^\top z}_{\text{Wert am Punkt } z} = \underbrace{c^\top x}_{\text{Wert am Punkt } x} + \sum_{i \in N} \Delta_i z_i. \tag{29}$$

Abbruchbedingung: Falls $\Delta_i > 0$ für alle $i \in N$ gilt, dann ist x bereits die Lösung, die wir suchen, denn zusammen mit $z \geq 0$ folgt aus (29): $c^\top x \leq c^\top z$ für alle $z \in X$.

Idee: Wähle einen Index $k \in N$ mit $\Delta_k < 0$, welcher in die Basis wechselt. Zusammen mit der Nichtnegativitätsbedingung $z_k \geq 0$ und $z_{N \setminus \{k\}} = 0$ ergibt sich dann:

$$c^\top z \leq c^\top x$$

Statt einem beliebigem Punkt z suchen wir nun eine benachbarte Ecke $\widehat{x}$ zur Basis $\widehat{B} = B \cup \{k\} \setminus \{r\}$.

Die Wahl des Index k, der von N in die neue Basis $\widehat{B}$ wechselt, wird als **pricing** bezeichnet. Den aus der Basis zu entfernenden Index r müssen wir noch ermitteln.

Der Simplex-Schritt - Richtung der Änderung

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

$$\widehat{x}_k(t) := t \geq 0, \quad \widehat{x}_i(t) := 0 \text{ für alle } i \in N \setminus \{k\}. \tag{30}$$

Dann ist der neue Funktionswert wegen (29) gleich

$$c^\top \widehat{x}(t) = c^\top x + \Delta_N \widehat{x}_N = c^\top x + t\,\Delta_k \leq c^\top x. \tag{31}$$

Die Gleichungsnebenbedingung $A\,\widehat{x}(t) = b$ liefert außerdem

$$\begin{aligned}
& A_B \widehat{x}_B(t) + A_N \widehat{x}_N(t) = b \\
\Leftrightarrow \quad & A_B \widehat{x}_B(t) + t\,a_k = b && (a_k \text{ ist die } k\text{-te Spalte von } A) \\
\Leftrightarrow \quad & \widehat{x}_B(t) = A_B^{-1}(b - t\,a_k) \\
\Leftrightarrow \quad & \widehat{x}_B(t) = x_B - t \underbrace{A_B^{-1} a_k}_{=:d_B}.
\end{aligned} \tag{32}$$

Der Vektor d_B beschreibt die **Richtung der Änderung** beim Übergang von x_B zu $\widehat{x}_B$. In Abhängigkeit von t wird $\widehat{x}_N(t)$ durch (30) und $\widehat{x}_B(t)$ durch (32) definiert.

Der Simplex-Schritt - Quotiententest

Wie groß soll nun $t \geq 0$ gewählt werden?

Die Gleichung (31) sagt voraus, dass die Reduktion der Zielfunktion beim Übergang von x zu $\widehat{x}$ proportional zu t ist. Also möchte man $t \geq 0$ möglichst groß wählen.

Andererseits ist $\widehat{x}_B(t) \geq 0$ zu beachten, also:

$$t \geq 0 \quad \text{und} \quad \widehat{x}_i(t) \overset{(32)}{=} x_i - t\, d_i \geq 0 \text{ für alle } i \in B. \tag{33}$$

Die größtmögliche Schrittweite ist also bestimmt durch

$$\widehat{t} := \min \left\{ \frac{x_i}{d_i} : i \in B,\ d_i > 0 \right\}. \tag{34}$$

Diese Berechnung nennt man den **Quotiententest** oder *ratio test*.

Derjenige Index $r \in B$, an dem das Minimum realisiert wird, wandert von der Basis B in die Nichtbasis $\widehat{N}$, denn für diesen Index ist (33) mit "$= 0$" erfüllt, also $\widehat{x}_r = 0$.

Der Simplex-Schritt - Quotiententest

Was passiert aber, wenn im Quotiententest alle $d_i \leq 0$ sind und deshalb in (34) kein Minimum gebildet werden kann?

Dann gibt es keine Einschränkung an die Schrittweite $t \geq 0$, und wir erhalten aus (31) 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. Eine solche Situation ist auf Folie 87 rechts illustriert.

Der Simplex-Schritt - Zusammenfassung

1. Berechne die reduzierten Kosten

$$\Delta_N := c_N - A_N^\top A_B^{-\top} c_B.$$

2. Wenn $\Delta_N \geq 0$ ist, dann ist x eine Optimallösung der Aufgabe (27). Andernfalls wähle einen Index $k \in N$ mit $\Delta_k < 0$ (*pricing*).
3. Berechne die Richtung der Änderung d_B aus dem linearen Gleichungssystem

$$A_B\, d_B = a_k.$$

4. Ist $d_B \leq 0$, dann hat das (LP) keine Optimallösung, da die Zielfunktion über der zulässigen Menge nach unten unbeschränkt ist. Andernfalls führe den Quotiententest

$$\widehat{t} := \min\left\{ \frac{x_i}{d_i} : i \in B,\ d_i > 0 \right\}$$

durch. Dabei sei $r \in B$ der Index bzw. einer der Indizes, an dem das Minimum auftritt.

Der Simplex-Schritt - Zusammenfassung

5. Aktualisiere die Basis und Nicht-Basis:

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

6. Berechne die Koordinaten der neuen Ecke über

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

Übung 3.7

Führe einen Simplex-Schritt für das Mozartproblem aus Beispiel 3.6 durch, beginnend mit der Basis $\{4, 5, 6\}$.

Startecke finden

Der Simplex-Algorithmus benötigt eine Startecke x und -basis B. Die Wahl von B ist hier nicht beliebig, da die Lösung von $A_B x_B = b$ nicht zwingend $x_B \geq 0$ liefert.

Im Fall, dass das ursprüngliche Problem nur aus Ungleichungsnebenbedingungen, also,

$$A\,x \leq b$$

bestand, und wenn $b \geq 0$ gilt, ist dies einfach. Die Umformung auf Normalform ergibt dann die Gleichungsnebenbedingung

$$\underbrace{[\,A \mid I\,]}_{=:\widetilde{A}} \underbrace{\begin{bmatrix} x \\ s \end{bmatrix}}_{=:\widetilde{x}} = b$$

mit Einheitsmatrix $I \in \mathbb{R}^{m\times m}$ und Schlupfvariablen $s \in \mathbb{R}^m$. Hier ist die offensichtliche Wahl für die Startbasis

$$B = \{n+1, \ldots, n+m\},$$

also die Indizes, die zu den Schlupfvariablen gehören. Die Ecke lautet dann

$$\widetilde{x}_N = x = 0, \quad \widetilde{x}_B = s = \widetilde{A}_B^{-1} b = b \geq 0.$$

Startecke finden

Im Allgemeinen Fall kann man eine Startbasis über ein **Phase-I-Problem** berechnen:

$$\begin{aligned} \text{Minimiere} \quad & e^\top s, \quad (x, s) \in \mathbb{R}^n \times \mathbb{R}^m \\ \text{unter} \quad & Ax + s = b \\ \text{sowie} \quad & x \geq 0, \quad s \geq 0. \end{aligned} \tag{35}$$

Eine Startbasis für dieses Problem ist bekannt, nämlich

$$B = \{n+1, \ldots, n+m\},$$

ähnlich zur vorherigen Folie. Der Kostenvektor ist hier der Vektor $e = (1, \ldots, 1)^\top \in \mathbb{R}^m$.

Mit dem Simplex-Verfahren selbst kann (35) nun gelöst werden und wir bezeichnen die Lösung mit (x^*, s^*).

Gilt $s^* = 0$, dann ist x^* zulässige Ecke des Ausgangsproblems. Mit dieser Ecke kann **Phase-II** (also der eigentliche Simplex-Algorithmus) gestartet werden.

Gilt $s^* \neq 0$, dann besitzt das Ausgangsproblem keine zulässigen Punkte und ist damit nicht lösbar.

Inhaltsverzeichnis

Einführung

Wir haben uns noch nicht überlegt, ob wir Optimallösungen über notwendige Optimalitätsbedingungen charakterisieren können. Natürlich ist $\nabla f(x) = 0$ hier keine notwendige Bedingung, denn wir haben ja auch noch Nebenbedingungen.

Der Gradient der Zielfunktion wäre der konstante Vektor $\nabla f(x) = c$ und dieser kann natürlich nicht 0 werden.

Wir können hier noch ausnutzen, dass lineare Optimierungsprobleme **konvex** sind, dadurch müssen wir nicht zwischen notwendigen und hinreichenden Optimalitätsbedingungen unterscheiden.

Wir werden in Kapitel 5 Optimalitätsbedingungen für allgemeine beschränkte Probleme genauer besprechen, daher geben wir hier nur die Folgerung aus dieser Theorie angewendet auf (25) an.

Optimalitätsbedingungen

Satz 3.8 (Notw. und hinr. Optimalitätsbedingung 1. Ordnung)

a) *Es sei $x^* \in \mathbb{R}^n$ ein (globales) Minimum von* (27)*. Dann kann man dazu passende Vektoren (genannt* ***Lagrange-Multiplikatoren****)*

$$\lambda \in \mathbb{R}^m, \quad \mu \in \mathbb{R}^n$$

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

$$\left.\begin{aligned} & A^\top \lambda - \mu + c = 0, \\ & \mu \geq 0, \quad x^* \geq 0, \quad \mu^\top x^* = 0, \\ & A x^* = b. \end{aligned}\right\} \tag{36}$$

b) *Sind umgekehrt $x^* \in \mathbb{R}^n$ und Vektoren $\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}^n$ bekannt, die das System* (36) *erfüllen, dann ist x^* ein (globales) Minimum von* (27)*.*

Optimalitätsbedingungen

Die zweite Bedingung in (36) wird als **Komplementaritätsbedingung** bezeichnet und besagt, dass der Multiplikator μ_i dann 0 wird, wenn die Nebenbedingung $x_i \geq 0$ nicht aktiv ist, also wenn $x_i > 0$.

Wir könnten auch ein ähnliches Optimalitätssystem direkt für die allgemeine Aufgabe (25) angeben, um die Theorie aber zu vereinfachen beschränken wir uns im Folgenden auf Probleme in **Normalform**.

Behauptung: Mit dem Originalproblem (27) löst man immer auch automatisch ein sogenanntes **duales Problem**, dessen Optimierungsvariablen die Lagrange-Multiplikatoren λ, μ sind.

Das duale Problem

Wir machen folgende Überlegung: Jeder zulässige Punkt $x \in \mathbb{R}^n$ erfüllt $Ax = b$ sowie $x \geq 0$ und nach (Skalar)-Multiplikation beider Seiten mit einem Vektor $y \in \mathbb{R}^m$ folgt außerdem

$$y^\top Ax = y^T b \quad \iff \quad x^\top A^\top y = b^\top y.$$

Wir fordern nun eine Nebenbedingung für y, nämlich $A^\top y \leq c$, wobei c der Kostenvektor aus dem Originalproblem ist. Dann folgt aus der oberen Gleichung

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

Das bedeutet, dass, falls die Vektoren $y \in \mathbb{R}^m$ die Ungleichung $A^\top y \leq c$ erfüllen, dann ist die Zahl $b^\top y$ immer eine untere Schranke für den Zielfunktionswert $c^\top x$, also insbesondere auch im Optimum. Diese Idee bringt uns auf das duale Problem auf der nächsten Folie.

Das duale Problem

Definition 3.9 (Duales Problem)

Das Optimierungsproblem

$$\left.\begin{aligned} \text{Maximiere} \quad & b^\top y, \quad y \in \mathbb{R}^m, \\ \text{unter} \quad & A^\top y \leq c \end{aligned}\right\} \tag{37}$$

heißt das zu (27) **duale Problem**.

Die Ungleichungsnebenbedingungen kann man auch wieder mit Schlupfvariablen in Gleichungsnebenbedingungen umformen:

$$\left.\begin{aligned} \text{Maximiere} \quad & b^\top y, \quad (y, z) \in \mathbb{R}^m \times \mathbb{R}^n \\ \text{unter} \quad & A^\top y + z = c \\ \text{und} \quad & z \geq 0 \end{aligned}\right\} \tag{38}$$

Das duale Problem

Wir halten eine wichtige Folgerung aus der Herleitung fest:

Satz 3.10 (Schwache Dualität in der linearen Optimierung)

Ist x primal zulässig für (27) *und y dual zulässig für* (37)*, dann gilt die Ungleichung $b^\top y \leq c^\top x$. Dies nennt man* ***schwache Dualität****.*

Was ist nun der Bezug zu den Lagrange-Multiplikatoren?

Wir rechnen den “Gap” aus:

$$c^\top x - b^\top y \overset{Ax=b}{=} x^\top c - x^\top A^\top y = x^\top (c - A^\top y) \overset{A^\top y + z = c}{=} x^\top z \overset{x,z \geq 0}{\geq} 0.$$

Der “Gap” hat also den Wert $x^\top z$. Ist dieser ‘Gap” gleich 0, dann muss x Lösung der primalen und z bzw. (y, z) Lösung der dualen Aufgabe sein. Die Bedingung $x^\top z = 0$ haben wir allerdings schonmal im Optimalitätssystem (36) gesehen!

Das duale Problem

Wir benennen die Variablen etwas um, damit wir den Bezug zu (36) besser erkennen:

$$z \leftrightarrow \mu, \qquad y \leftrightarrow -\lambda.$$

Zielfunktion und Nebenbedingung von (27) und (38) sehen dann wir folgt aus:

(primale Zielfunktion)	(duale Zielfunktion)
$c^\top x \to \min!$	$b^\top(-\lambda) \to \max!$
(primale Zulässigkeit)	(duale Zulässigkeit)
$Ax = b,\ x \geq 0$	$A^\top(-\lambda) + \mu = c,\ \mu \geq 0.$

Die Lagrange-Multiplikatoren sind (bis auf das Vorzeichen) also die Optimierungsvariablen der dualen Aufgabe. Ferner haben wir gesehen, dass Optimalität vorliegt, wenn

$$x^\top \mu = 0$$

gilt. Zusammen mit der primalen und dualen Zulässigkeit sind das gerade die Bedingungen aus Satz 3.8!

Das duale Problem

Bei genauerer Betrachtung sehen wir auch, dass der Simplex-Algorithmus die dualen Variablen schon mit verwendet. Die **reduzierten Kosten** waren definiert als

$$\Delta_N = c_N - A_N^\top \underbrace{A_B^{-\top} c_B}_{y} = c_N - A_N^\top y.$$

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

$$\begin{aligned} z_B &:= c_B - A_B^\top y = 0, \\ z_N &:= c_N - A_N^\top y \geq 0 \quad \Leftrightarrow \quad \Delta_N \geq 0. \end{aligned}$$

Das bedeutet auch, dass eine Optimallösung gefunden wurde ($\Delta_N \geq 0$ war das Kriterium dafür), wenn die implizit mitgeführten dualen Variablen (y, z) zulässig sind. Während der Simplex-Iteration ist dabei immer nur $z \geq 0$ verletzt.

Das duale Problem

Satz 3.11 (Starke Dualität der linearen Optimierung)

Das primale lineare Optimierungsproblem in Normalform (27) *hat genau dann eine Optimallösung, wenn das zugehörige duale Problem* (37) *bzw.* (38) *eine Optimallösung besitzt.*
Die Werte der Zielfunktion f^* und d^* sind dann gleich, und für jedes Paar aus primaler und dualer Optimallösung x und (y, z) ist die Komplementaritätsbedingung $x^\top z = 0$ erfüllt.

Man kann sogar noch die Fälle der Nichtlösbarkeit in Verbindung bringen:

	duales LP (38)		
primales LP (27)	lösbar $d^* \in \mathbb{R}$	unbeschränkt $d^* = \infty$	unzulässig $d^* = -\infty$
lösbar	(I)	—	—
unbeschränkt ($f^* = -\infty$)	—	—	(III)
unzulässig ($f^* = \infty$)	—	(III)	(II)

Schattenpreise

Wir diskutieren noch eine weitere Bedeutung der Multiplikatoren. Wir bezeichnen die primalen und dualen Lösungen mit x bzw. (y, z). Angenommen, beim Mozartproblem verändern sich die Lagerbestände, d. h., statt b haben wir ein $b + t\,\Delta b$, $t > 0$, $\Delta b \in \mathbb{R}^m$, auf der rechten Seite der Gleichung. Wir betrachten daher

$$\left.\begin{array}{rl} \text{Minimiere} & c^\top x \\ \text{sodass} & Ax = b + t\,\Delta b \\ \text{und} & x \geq 0 \end{array}\right\} \quad \left\{\begin{array}{rl} \text{Maximiere} & (b + t\,\Delta b)^\top y \\ \text{sodass} & A^\top y + z = c \\ \text{und} & z \geq 0. \end{array}\right. \tag{39}$$

Die duale Zulässigkeit ist weiterhin gegeben. Die primale Zulässigkeit könnte verletzt sein. Daher müssen wir x in Abhängigkeit von t entsprechend anpassen. Halte dazu (y, z) konstant und passe x_B entsprechend an:

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

Haben wir so schon eine Lösung des gestörten Problems gefunden?

Schattenpreise

Überprüfen wir ob x und (y, z) die Optimalitätsbedingungen (36) zur gestörten Aufgabe erfüllt:

primale Zulässigkeit:	$Ax = b + t\,\Delta b$	gilt nach Konstruktion,
duale Zulässigkeit:	$A^\top y + z = c$	gilt auch für gestörte Aufgabe,
primale Vorzeichenbed.:	$x(t) \geq 0$	noch zu prüfen,
duale Vorzeichenbed.:	$z \geq 0$	gilt auch für gestörte Aufgabe,
Komplementarität:	$x(t)^T z = 0$	leicht nachzurechnen.

Die letzte Bedingung folgt aus

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

Die Bedingung $x(t) \geq 0$ bleibt erfüllt, wenn die Störung hinreichend klein ist.

Schattenpreise

Der optimale Zielfunktionswert der gestörten Aufgabe ist

$$c^\top x(t) \overset{\text{starke Dualität}}{=} (b + t\,\Delta b)^\top y.$$

Der Unterschied zum ungestörten Problem ist also

$$c^\top x(t) - c^\top x = c^\top (x(t) - x) = t\,\Delta b^\top y = \underbrace{t\,\Delta b^\top}_{\text{Störung}} (\underbrace{-\lambda}_{\text{Faktor}}). \tag{40}$$

Damit erkennen wir eine weitere Bedeutung des Lagrange-Multiplikators λ. Er gibt den **Empfindlichkeitsfaktor** (die **Sensitivität**) an, mit der sich der optimale Zielfunktionswert ändert, wenn die rechte Seite von b zu $b + t\,\Delta b$ verändert wird.

Die gleiche Erkenntnis gilt auch für die Multiplikatoren zu Ungleichungsnebenbedingungen.

`linprog` erlaubt es auch die Multiplikatoren auszugeben, die dann eben nicht nur unnützer Ballast sind, sondern auch wertvolle Informationen tragen.

Schattenpreise - Beispiel

`linprog` gibt uns bei der Lösung des Mozartproblems

$$\text{Minimiere } \begin{pmatrix} -9 \\ -8 \end{pmatrix}^\top \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{sodass} \quad \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 11 \\ 9 \end{pmatrix} \quad \begin{pmatrix} \text{Marzipan} \\ \text{Nougat} \\ \text{Bitterschokolade} \end{pmatrix}$$

den Multiplikator

$$\lambda = \begin{pmatrix} 7 \\ 1 \\ 0 \end{pmatrix}$$

zurück.

Übung 3.12

Wie ändert sich der Gewinn, wenn wir mehr Marzipan, Nougat oder Bitterschokolade einkaufen? Setze $\Delta b = e_1, e_2, e_3$ in (40) ein und überlege was passiert.

Inhaltsverzeichnis

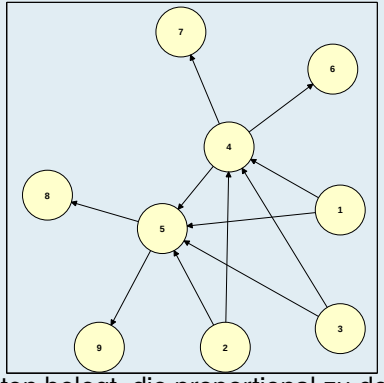
Einführung

Beispiel 3.13 (Transportproblem)

Ein Unternehmen verfügt ein **Transportnetzwerk** mit **Knoten**

- Produktionsstätten (1–3),
- Zwischenlagern (4–5)
- Verkaufsstätten (6–9).

Die **Kanten** zwischen den Knoten entsprechen den möglichen Transportwegen. Die Kapazitäten der Produktionsstätten sowie die Bedarfe der Verkaufsstätten sind bekannt.



Jeder Transportweg (Kante) ist dabei mit Transportkosten belegt, die proportional zu der Warenmenge sind, die über diesen Weg transportiert wird. Gesucht ist nun eine Verteilung der Transportströme über die Kanten, die die Gesamttransportkosten minimiert.

Graphen

Definition 3.14 (Graphen)

1. Ein **gerichteter Graph** (engl.: *directed graph*, kurz: **Digraph**) ist ein Paar $D = (V, E)$ bestehend aus einer Menge V von Knoten (engl.: *vertices*) und einer Menge $E \subset V \times V$ von gerichteten Kanten (Bögen, engl.: *edges*) zwischen Knoten.
2. 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.
3. 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.

Den Graphen von der letzten Folie können wir also beschreiben durch

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$E = \{(1,4), (1,5), (2,4), (2,5), (3,4), (3,5), (4,5), (4,6), (4,7), (5,8), (5,9)\}.$$

Graphen

Um mit Graphen rechnen zu können benötigen wir

Definition 3.15 (Adjazenz- und Inzidenzmatrix)

Es sei ein Graph $D = (V, E)$ mit Knotenmenge $V = \{v_1, \ldots, v_m\}$ und Kantenmenge $E = \{e_1, \ldots, e_n\}$ gegeben. Die Matrix der Dimension $m \times m$ und Einträgen

$$a_{ij}^{\text{ad}} = \begin{cases} 1, & \text{falls es die Kante von } v_i \text{ nach } v_j \text{ in der Kantenmenge } E \text{ gibt,} \\ 0, & \text{andernfalls.} \end{cases}$$

heißt **Adjazenzmatrix**.
Die Matrix der Dimension $m \times n$ und Einträgen

$$a_{ij}^{\text{in}} = \begin{cases} -1, & \text{falls die Kante } e_j \text{ im Knoten } v_i \text{ startet,} \\ 1, & \text{falls die Kante } e_j \text{ im Knoten } v_i \text{ endet,} \\ 0, & \text{sonst.} \end{cases}$$

heißt **(Knoten-Kanten-)Inzidenzmatrix**.

Übung 3.16

Stelle die Adjazenz- und Inzidenzmatrix für den Graphen aus der vorletzten Folie auf.

Rechnen mit Graphen

Welche Bedeutung haben Matrix-Vektor-Produkte mit den eben definierten Matrizen?

Sei $x \in \mathbb{R}^n$ die Belegung der Warenströme über die Kanten. Eine Komponente (hier die 5.) des Produkts $A^{\text{in}}x$, lautet

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{11} \end{pmatrix} = x_2 + x_4 + x_6 + x_7 - x_{10} - x_{11}$$

und das Ergebnis gibt die **Knotenbilanz** der Warenströme im Knoten 5 (Zwischenlager) an. Hierfür entscheidend sind Ein- bzw. Ausströmung über die Kanten mit den Nummern 2, 4, 6, 7, 10, 11.

Somit enthält der Vektor $A^{\text{in}}x$ die **Bilanzen** in allen Knoten.

Optimierungsprobleme auf Graphen

Wir können nun das Problem 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:

$$\sum_{j=1}^{n} c_j x_j = c^\top x.$$

Dabei sind c_j die Transportkosten pro Einheit über die Kante e_j. Weiterhin sind die Bilanzen b_i aller Knoten gegeben:

- **Bedarfsknoten:** $b_i > 0$
- **Angebotsknoten:** $b_i < 0$
- **Durchflussknoten:** $b_i = 0$

Die Erfüllung der Knotenbedingungen wird dann durch die Nebenbedingung

$$A^{\text{in}} x = b,$$

die sogenannte **Flusserhaltungsgleichung**, sichergestellt.

Optimierungsprobleme auf Graphen

Zusätzlich fordern wir natürlich, dass der Fluss über die Kanten nicht negativ sein darf (gerichteter Transport) und gewisse Kapazitätseinschränkungen erfüllen müssen:

$$0 \leq x_i \leq u_i$$

Kompakt kann man eine Optimierungsaufgabe des kostenminimalen Flusses also wie folgt schreiben:

$$\begin{aligned} \text{Minimiere} \quad & c^\top x, \quad x \in \mathbb{R}^n \\ \text{unter} \quad & A\,x = b \\ \text{sowie} \quad & 0 \leq x \leq u. \end{aligned} \tag{41}$$

Dabei gelten folgende Bezeichnungen:

$A =$ Inzidenzmatrix in $\mathbb{R}^{m \times n}$, $\quad c =$ Vektor der Kantenkosten in $\mathbb{R}^n$,

$b =$ Vektor der Knotenbilanzen in $\mathbb{R}^m$, $\quad u =$ Vektor der Kantenkapazitäten in $\mathbb{R}^n$.

Optimierungsprobleme auf Graphen

Übung 3.17

Berechne den optimalen Fluss über das Netzwerk aus Folie 115 für die Daten

$$b = \begin{pmatrix} -100 & -200 & -300 & 0 & 0 & 150 & 150 & 150 & 150 \end{pmatrix}^\top,$$
$$u = \vec{1},$$
$$c = \begin{pmatrix} 0.8 & 2.0 & 2.5 & 1.0 & 1.2 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{pmatrix}^\top.$$

*Es gilt offensichtlich $\sum_i b_i = 0$. Warum ist das nötig? Was passiert bei **Überangebot** oder **Mangel**?*

Inhaltsverzeichnis

Inhaltsverzeichnis

Einführung

In diesem Kapitel geht es nun um allgemeine **nichtlineare Optimierungsaufgaben** mit Beschränkungen:

$$\begin{aligned} &\text{Minimiere} && f(x), && x \in \mathbb{R}^n, \\ &\text{sodass} && g_i(x) \leq 0, && i = 1, \ldots, m, \\ &\text{und} && h_j(x) = 0, && j = 1, \ldots, p. \end{aligned} \tag{42}$$

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.

Alle Funktionen $f(x)$, $g_i(x)$ und $h_j(x)$ sollen als Generalvoraussetzung in diesem Kapitel mindestens einmal stetig partiell diffbar sein.

In der Aufgabe (42) sind zur Konkurrenz um den kleinsten Zielfunktionswert nur diejenigen $x \in \mathbb{R}^n$ zugelassen, die bezüglich *aller* Gleichungen und Ungleichungen **zulässig** sind:

$$\begin{aligned} X = \{x \in \mathbb{R}^n : g_i(x) \leq 0 \quad &\text{für alle } i = 1, \ldots, m, \\ h_j(x) = 0 \quad &\text{für alle } j = 1, \ldots, p\}. \end{aligned} \tag{43}$$

Lokale und globale Minima

Definition 4.1

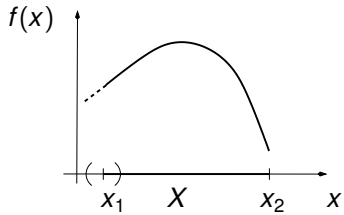
Betrachte (42) mit zulässiger Menge (43).

a) Ein Vektor $x^* \in X$ heißt **globales Minimum**, wenn $f(x^*) \leq f(x)$ für alle $x \in X$.

b) Ein globales Minimum heißt **strikt**, wenn $f(x^*) < f(x)$ für alle $x \in X$, $x \neq x^*$ gilt.

c) Ein Vektor $x^* \in X$ heißt **lokales Minimum**, wenn $f(x^*) \leq f(x)$ für alle $x \in X \cap U(x^*)$.

d) Ein lok. Minimum heißt **strikt**, wenn $f(x^*) < f(x)$ für alle $x \in X \cap U(x^*)$, $x \neq x^*$, gilt.



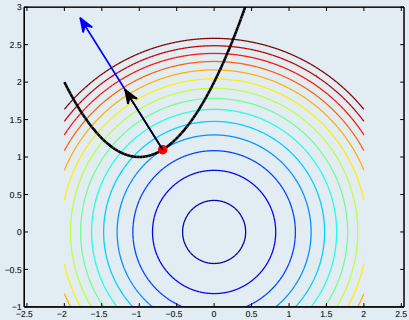
Ein erstes Beispiel

Beispiel 4.2 (Illustration notwendiger Optimalitätsbedingungen)

$$\begin{aligned} \text{Minimiere} \quad & f(x) := x_1^2 + x_2^2, \quad x = (x_1, x_2) \in \mathbb{R}^2 \\ \text{sodass} \quad & h(x) := x_2 - (x_1 + 1)^2 - 1 = 0. \end{aligned}$$

Die zulässige Menge besteht hier also aus dem Graphen einer Parabel:

$$X = \{x = (x_1, x_2) \in \mathbb{R}^2 : x_2 = (x_1 + 1)^2 + 1\}.$$



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.

Optimalitätsbedingungen

Die Beobachtungen aus dem letzten Beispiel motivieren folgende Idee, die uns auf eine notwendige Optimalitätsbedingung führt.

Die Vektoren $\nabla f(x^*)$ und $\nabla h(x^*)$ sind linear abhängig, d. h., es gibt eine Zahl $\mu \in \mathbb{R}$, sodass gilt:

$$\nabla f(x^*) + \lambda \nabla h(x^*) = 0. \tag{44}$$

Diese Beobachtung ist kein Zufall, sondern sie gilt (unter bestimmten Voraussetzungen) für jedes lokale Minimum nichtlinearer Optimierungsaufgaben (42). Die Zahl λ heißt der zum Minimum x^* gehörende **Lagrange-Multiplikator** für die Beschränkung $h(x^*) = 0$.

Die Bedingung (44) lässt sich bequem mit Hilfe der **Lagrange-Funktion** formulieren:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda h(x),$$

denn (44) ist nichts anderes als die Bedingung

$$\nabla_x \mathcal{L}(x^*, \lambda) = \nabla f(x^*) + \lambda \nabla h(x^*) = 0.$$

Optimalitätsbedingungen

Für Optimierungsaufgaben der Gestalt (42) mit mehreren Gleichungen und Ungleichungen gilt im Prinzip dieselbe Beobachtung:

Auch hier lässt sich in einem lokalen Minimum x^* der negative Gradient der Zielfunktion als Summe einer Linearkombination der Gradienten der Gleichungsnebenbedingungen und einer nichtnegativen Linearkombination der Gradienten aktiver Ungleichungsnebenbedingungen schreiben.

Dies kann man wieder bequem mit Hilfe der **Lagrange-Funktion** formulieren. Für die Aufgabe (42) lautet diese:

$$\begin{aligned} \mathcal{L}(x, \mu, \lambda) &:= f(x) + \sum_{i=1}^{m} \mu_i \, g_i(x) + \sum_{j=1}^{p} \lambda_j \, h_j(x) \\ &= f(x) \; + \mu^\top g(x) \qquad + \lambda^\top h(x). \end{aligned} \tag{45}$$

Die Vektoren μ und λ nennt man **Lagrange-Multiplikatoren**, und sie haben ebenso viele Komponenten, wie es Ungleichungs- bzw. Gleichungsnebenbedingungen in der Aufgabe gibt.

Optimalitätsbedingungen

Die Bedingung an die Linearkombination der Gradienten lässt sich nun schreiben als

$$\begin{aligned}\nabla_x \mathcal{L}(x, \mu, \lambda) &= \nabla f(x) + \sum_{i=1}^{m} \mu_i \nabla g_i(x) + \sum_{j=1}^{p} \lambda_j \nabla h_j(x) \\ &= \nabla f(x) + g'(x)^\top \mu \qquad + h'(x)^\top \lambda = 0.\end{aligned} \tag{46a}$$

Dabei sind $g'(x)$ und $h'(x)$ die Jacobimatrizen von $g(x)$ bzw. $h(x)$. Bei dieser Linearkombination spielen nur diejenigen Ungleichungen $g_i(x) \leq 0$ eine Rolle, die an der Stelle x **aktiv** sind (nur sie schränken lokal die Bewegungsfreiheit ein):

$$\mathcal{A}_g(x) = \{i = 1, \ldots, m : g_i(x) = 0\}.$$

Die Gradienten $\nabla g_i(x)$ der an der Stelle x **inaktiven** Ungleichungen

$$\mathcal{I}_g(x) = \{i = 1, \ldots, m : g_i(x) < 0\}$$

schaltet man in der Linearkombination (46a) dadurch aus, dass gefordert wird, dass die zugehörigen Lagrange-Multiplikatoren $\mu_i = 0$ sind für alle Indizes $i \in \mathcal{I}_g(x)$.

Optimalitätsbedingungen

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

$$g_i(x) \leq 0, \quad \mu_i \geq 0, \quad \mu_i \, g_i(x) = 0 \quad \text{für alle } i = 1, \ldots, m$$

oder äquivalent als

$$g(x) \leq 0, \quad \mu \geq 0, \quad \mu^\top g(x) = 0. \tag{46b}$$

Die Ungleichungsbedingungen erfüllen also zusammen mit dem zugehörigen Multiplikator wieder ein **Komplementaritätssystem**, wie wir es schon aus Kapitel 3 kennen, vgl. (36). Schließlich gilt natürlich auch

$$h_j(x) = 0 \quad \text{für alle } j = 1, \ldots, p$$

bzw. vektorwertig

$$h(x) = 0. \tag{46c}$$

Das System (46) nennt man die **Karush-Kuhn-Tucker-Bedingungen**, kurz: **KKT-Bedingungen** der Aufgabe (42). Sie stimmen mit den schon aus Kapitel 3 überein, falls (42) eine *lineare* Optimierungsaufgabe ist.

Optimalitätsbedingungen

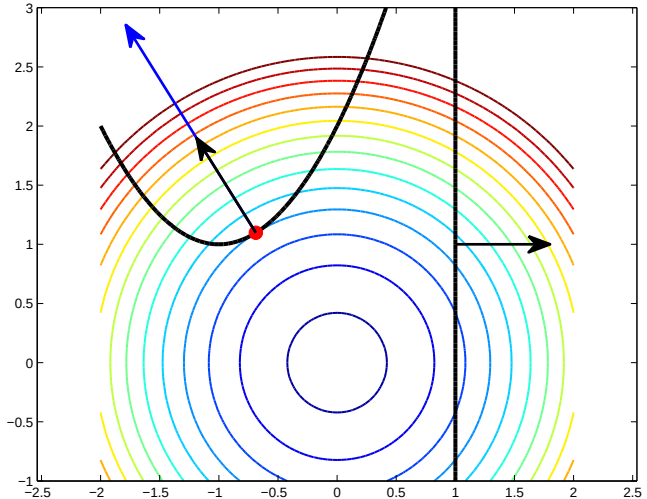


Illustration der KKT-Bedingungen für ein Problem mit einer zusätzlichen inaktiven Ungleichunsnebenbedingung.

Optimalitätsbedingung

Satz 4.3 (Notwendige Optimalitätsbedingung 1. Ordnung)

Es sei x^* ein lokales Minimum von (42)*. 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 (46) *aus Gleichungen und Ungleichungen (die sogenannten* ***Karush-Kuhn-Tucker-Bedingungen****, kurz:* ***KKT-Bedingungen****) erfüllt ist, hier nochmals zusammengefasst:*

$$\left.\begin{aligned} \nabla_x \mathcal{L}(x^*, \mu, \lambda) = \nabla f(x^*) + g'(x^*)^\top \mu + h'(x^*)^\top \lambda = 0, \\ \mu \geq 0, \quad g(x^*) \leq 0, \quad \mu^\top g(x^*) = 0, \\ h(x^*) = 0. \end{aligned}\right\} \tag{47}$$

Ein Tripel (x^*, μ, λ), das (47) erfüllt, heißt auch ein **KKT-Punkt** der Aufgabe (42).

Optimalitätsbedingungen

Bemerkung 4.4 (zu den Optimalitätsbedingungen aus Satz 4.3)

a) *Die KKT-Bedingungen* (47) *enthalten als Spezialfall diejenigen Bedingungen für lineare Optimierungsaufgaben* (36)*. Dass wir dort, also in Satz 3.8, keine Constraint Qualifications fordern mussten, liegt an der Linearität der Nebenbedingungen.*

b) *Im Unterschied zur linearen Optimierung sind die KKT-Bedingungen bei nichtlinearen Aufgaben jedoch lediglich notwendige (keine hinreichenden) Optimalitätsbedingungen. Die Umkehrung von Satz 4.3 gilt leider nicht, d.h., es kann Vektoren* (x^*, μ, λ) *geben, die die KKT-Bedingungen* (47) *erfüllen, und dennoch ist* x^* *kein lokales Minimum für* (42).
Ausnahme: *Falls die Optimierungsaufgabe* (42) *konvex ist, dann gilt auch die Umkehrung von Satz 4.3.*

Optimalitätsbedingungen

Übung 4.5

Stelle die Lagrange-Funktion und das KKT-System für folgende Aufgaben auf:

a) *Ein* ***lineares Optimierungsproblem*** *in Normalform*

$$\begin{aligned} &\textit{Minimiere} && c^\top x && \textit{über } x \in \mathbb{R}^n, \\ &\textit{sodass} && Ax = b, \\ &\textit{und} && x \geq 0. \end{aligned}$$

b) *Ein Optimierungsproblem mit* ***Box-Beschränkungen***

$$\begin{aligned} &\textit{Minimiere} && f(x) && \textit{über } x \in \mathbb{R}^n, \\ &\textit{sodass} && \ell \leq x \leq u. \end{aligned}$$

Constraint-Qualifications

Um nun Satz 4.3 zu verstehen, müssen wir noch den Begriff der **Constraint Qualification** (dt.: *Regularitätsbedingung für die Beschränkungen*) näher erläutern. Es gibt verschiedene Constraint Qualifications, die aber alle denselben Zweck erfüllen. Wir nennen hier nur eine solche Bedingung, die besonders einfach zu handhaben ist:

Definition 4.6 (Linear Independence Constraint Qualification (LICQ))

Es sei $\bar{x}$ zulässiger Punkt des Optimierungsproblems (42) und $\mathcal{A}_g(\bar{x})$ die Menge der dort aktiven Ungleichungen. Man sagt: $\bar{x}$ erfüllt die **(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.

Weitere (schwächere) Constraint-Qualifications wären die Guignard-CQ (GCQ), Abadie-CQ (ACQ), Mangasarian-Fromovitz-CQ (MFCQ) und es gilt

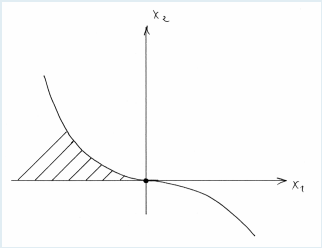
$$\text{(LICQ)} \quad \Rightarrow \quad \text{(MFCQ)} \quad \Rightarrow \quad \text{(ACQ)} \quad \Rightarrow \quad \text{(GCQ)}.$$

Wichtigkeit von Constraint Qualifications

Beispiel 4.7

Wir betrachten die Optimierungsaufgabe

$$\begin{aligned} \text{Minimiere} \quad & -x_1, \quad x = (x_1, x_2) \in \mathbb{R}^2 \\ \text{sodass} \quad & g_1(x) := x_2 + x_1^3 \leq 0 \\ \text{und} \quad & g_2(x) := -x_2 \leq 0. \end{aligned} \tag{48}$$



a) Wie lautet die Lösung dieser Aufgabe?
b) Überprüfe die LICQ.
c) Existieren Multiplikatoren, die das KKT-System erfüllen?

Inhaltsverzeichnis

Handhabung von `fmincon`

Mit `help fmincon` erhalten wir die Code-Dokumentation:

```
fmincon - Find minimum of constrained nonlinear multivariable
    function
    Nonlinear programming solver.

    Syntax
      x = fmincon(fun,x0,A,b)
      x = fmincon(fun,x0,A,b,Aeq,beq)
      x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
      x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
      x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
      x = fmincon(problem)
      [x,fval] = fmincon(___)
      [x,fval,exitflag,output] = fmincon(___)
      [x,fval,exitflag,output,lambda,grad,hessian] = fmincon(
          ___)
```

Handhabung von `fmincon`

Bemerkung 4.8 (zu `fmincon`)

a) *Die **Eingabeparameter** von `fmincon` sind:*

- `fun` *... ein Handle zur Zielfunktion, analog zu* `fminunc` *(siehe Bemerkung 1.9).*
- `x0` *... der Startvektor für das iterative Lösungsverfahren.*
- `A, b` *bzw.* `Aeq, beq` *... für lineare Ungleichungs- bzw. Gleichungsnebenbedingungen anzugeben, analog zu* `linprog`*, siehe Abschnitt 2.*
- `lb, ub` *... zur Implementierung von Box-Beschränkungen.*
- `nonlcon` *... nichtlineare Nebenbedingungen in der Form*

```
function [c,ceq] = mycon(x)
c = ...      % Compute nonlinear inequalities at x.
ceq = ...    % Compute nonlinear equalities at x.
```

Handhabung von `fmincon`

Bemerkung 4.8 (Fortsetzung)

b) *Die wichtigsten **Ausgabeparameter** sind:*
   - `x` *... die Lösung* x^*.
   - `fval` *... der zugehörige Funktionswert* $f(x^*)$.
   - `exitflag` *... eine Code-Nummer, die auf Erfolg oder einen Fehler hinweist.*
   - `output` *... eine Struktur, die u.A. eine Erfolgs- oder Fehlermeldung beinhaltet.*
   - `lambda` *... die Lagrange-Multiplikatoren für den optimalen KKT-Punkt.*

c) *Mit* `options = optimoptions('fmincon')` *kann man sich eine Struktur erzeugen, in der man einige Verfahrensparameter manuell setzen kann. Diese kann als letztes Eingabeargument an* `fmincon` *übergeben werden.*

d) *Kennt man die Gradienten von* f, g_i, h_j*, so kann man diese auch übergeben und der Algorithmus spart sich Rechenaufwand zur Bestimmung von Ableitungen. Setze hierfür* `options.SpecifyObjectiveGradient` *und* `options.SpecifyConstraintGradient` *auf* `true` *und erweitere die Funktionen für* f *und* $[g, h]$ *entsprechend.*

Beispiel

Übung 4.9

Löse das Problem aus Beispiel 4.7 mit `fmincon`*.*
Gibt den Iterationsverlauf durch setzen von `options.Display = 'iter'` *aus und interpretiere die Ausgabe.*
Implementiere auch die Gradienten von Zielfunktion und Nebenbedingungen.

Inhaltsverzeichnis

Problemstellung

Wir untersuchen hier einen allgemeinen Algorithmus zur Lösung des Problems

$$\left.\begin{aligned} \text{Minimiere} \quad & f(x), \qquad x \in \mathbb{R}^n, \\ \text{sodass} \quad & c_{\text{eq}}(x) = 0, \\ & c_{\text{inequ}}(x) \leq 0, \\ & A_{\text{eq}}\, x = b_{\text{eq}}, \\ & A_{\text{ineq}}\, x \leq b_{\text{ineq}}, \\ \text{sowie} \quad & \ell \leq x \leq u. \end{aligned}\right\} \tag{49}$$

Beachte, dass wir hier lineare Nebenbedingungen und Box-Beschränkungen separat aufgeführt haben, da wir diese algorithmisch teilweise besser behandeln können als allgemeine nichtlineare Nebenbedingungen.

Die zulässige Menge lautet hier

$$X = \{x \in \mathbb{R}^n \colon c_{\text{eq}}(x) = 0, c_{\text{ineq}}(x) \geq 0, \ell \leq x \leq u, \\ A_{\text{eq}}x = b_{\text{eq}}, A_{\text{ineq}}x \leq b_{\text{ineq}}\}.$$

Das SQP-Verfahren

Beachte, dass in der Routine `fmincon` verschiedene Algoritmen implementiert sind. Wir untersuchen hier die sogenannte **Sequential Quadratic Programming**-Methode, welche verwendet wird, wenn die Option `options.Algorithm = 'sqp'` gesetzt wurde.

Dieses Verfahren stellt eine Verallgemeinerung des Quasi-Newton-Verfahrens dar. Es wird wieder eine Folge von Vektoren $\{x^{(k)}\} \subset \mathbb{R}^n$ erzeugt, die gegen ein lokales Minimum konvergiert. Außerdem werden Folgen von Lagrange-Multiplikatoren $\{\lambda_{\text{ceq}}\}$, $\{\lambda_{\text{cineq}}\}$, $\{\lambda_{\text{Aeq}}\}$, $\{\lambda_{\text{Aineq}}\}$, $\{\lambda_\ell\}$, $\{\lambda_u\}$ erzeugt, die Zusammen mit $\{x^{(k)}\}$ im Grenzwert einen KKT-Punkt bilden.

Ein quadratisches Modellproblem

Ähnlich zum Quasi-Newton-Verfahren ersetzen wir die Zielfunktion durch ein **quadratisches Modell**, zentriert in $x^{(k)}$, und ersetzen wieder $x = x^{(k)} + d$ mit der gesuchten Richtung d. Die Nebenbedingungen werden hier außerdem **linearisiert**, d.h., durch lineare Funktionen ersetzt. Dies führt auf folgendes Ersatzproblem

$$
\left.
\begin{aligned}
\text{Minimiere} \quad & f(x^{(k)}) + \nabla f(x^{(k)})^\top d + \frac{1}{2} d^\top B^{(k)} d, \quad d \in \mathbb{R}^n, \\
\text{sodass} \quad & c_{\text{eq}}(x^{(k)}) + c'_{\text{eq}}(x^{(k)})\, d = 0, \\
& c_{\text{ineq}}(x^{(k)}) + c'_{\text{ineq}}(x^{(k)})\, d \geq 0, \\
& A_{\text{eq}}\, x^{(k)} + A_{\text{eq}}\, d = b_{\text{eq}}, \\
& A_{\text{ineq}}\, x^{(k)} + A_{\text{ineq}}\, d \leq b_{\text{ineq}}, \\
\text{sowie} \quad & \ell \leq x^{(k)} + d \leq u.
\end{aligned}
\right\} \tag{50}
$$

Ein quadratisches Modellproblem

(50) ist ein sogenanntes **quadratisches Programm**, also ein Optimierungsproblem mit quadratischer Zielfunktion und (affin) linearen Nebenbedingungen. Dies erklärt auch den Namen **sequential quadratic programming**, da in jeder Iteration ein quadratisches Programm gelöst werden muss.

Ähnlich zum Quasi-Newton-Verfahren wird wieder eine Approximation $B^{(k)}$ der Hessematrix

$$\begin{aligned}&\nabla^2 \mathcal{L}(x, \lambda_{\text{ceq}}, \lambda_{\text{cineq}}, \lambda_{\text{Aeq}}, \lambda_{\text{Aineq}}, \lambda_\ell, \lambda_u) \\ &\quad = \nabla^2 f(x) + \sum_{i=1}^{n_{\text{ceq}}} \lambda_{\text{ceq},i} \nabla^2 c_{\text{ceq}}(x) + \sum_{i=1}^{n_{\text{ineq}}} \lambda_{\text{ceineq},i} \nabla^2 c_{\text{ineq}}(x).\end{aligned}$$

der Lagrange-Funktion genutzt. Das Modell enthält somit uch Krümmungsinformationen zu den nichtlinearen Nebenbedingungen $c_{\text{eq}}(x)$ und $c_{\text{ineq}}(x)$.

Das Update der Matrix $B^{(k)}$ erfolgt ähnlich zur BFGS-Formel (8), wobei wieder auf positive Definitheit geachtet wird um die Lösung des Modellproblems eindeutig zu machen.

Funktionsweise des SQP-Verfahrens

Die Lösung des Modellproblems (50) ist etwas komplizierter und wird hier nicht weiter behandelt. Hierfür gibt es angepasste Verfahren wie die Aktive-Mengen-Strategie, ein Simplex-Verfahren speziell für QPs oder Innere-Punkte-Verfahren.

Hat man eine Lösung $d^{(k)}$ des quadratischen Hilfsproblem (50) berechnet, führt man anschließend wieder eine **Liniensuche** durch. Die Liniensuche achtet dabei nicht nur auf einen Abstieg im Funktionswert, sondern auch auf die Einhaltung der Nebenbedingungen. Daher wird eine Liniensuche bez. einer sogenannten **Nutzen-Funktion** (**merrit function**) durchgeführt.

Ist die Schrittweite bestimmt, so wird die nächste Iterierte als

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

gesetzt.

Ausgabe von `fmincon`

Übung 4.10

Interpretiere folgende Ausgabe von `fmincon`.

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)
```