

Optimierung mit Matlab

1 Optimierungsaufgaben

Die allgemeine Aufgabenstellung der Optimierung besteht darin, zu einer gegebenen Menge $X \subseteq \mathbb{R}^n$, $n \in \mathbb{N} \setminus \{0\}$ und einer Funktion $f : X \rightarrow \mathbb{R}$ ein $x^* \in Z \subseteq X$ zu finden, so dass gilt: $f(x^*) \leq f(x)$ für alle $x \in Z$. Die Funktion f wird Zielfunktion, Z zulässiger Bereich und $f(x^*)$ wird Optimalwert bzw. Optimum genannt. Für diese Aufgabenstellung ist die Kurzschreibweise

$$\begin{array}{ll} \min & f(x) \\ \text{u.d.N} & x \in Z \end{array} \quad (1.1)$$

üblich, wobei u.d.N für unter den Nebenbedingungen bzw. unter der Nebenbedingung steht. Im Fall von $Z \subset X \subseteq \mathbb{R}^n$ spricht man von einem restringierten Optimierungsproblem, während man für $Z = \mathbb{R}^n$ von unrestringierter Optimierung spricht.

Der zulässige Bereich Z wird allgemein durch Nebenbedingungen (Restriktionen) definiert. Dies erfolgt durch Gleichungen und/oder Ungleichungen. Ist die Zielfunktion linear und werden die Nebenbedingungen durch ein System linearer Gleichungen und Ungleichungen definiert, so spricht man von *linearer Optimierung*. Ist dagegen entweder die Zielfunktion nichtlinear oder werden Nebenbedingungen durch nichtlineare Gleichungen und Ungleichungen definiert, dann spricht man von *nichtlinearer Optimierung*. Werden keine Gleichungen oder Ungleichungen angegeben, so kann für (allgemeinere) Teilmengen $Z \subset \mathbb{R}^n$ (z. B. $Z = \mathbb{R}_{\geq 0}^n$) auch kürzer

$$\min_{x \in Z} f(x) \quad (1.2)$$

geschrieben werden (wobei z.B. $Z = \mathbb{R}_{\geq 0}^n$ natürlich auch durch Ungleichungen dargestellt werden kann). Nachfolgend wird stets angenommen, dass $Z = X$ ist.

Es ist zu beachten, dass es Problemstellungen gibt, die ohne Restriktionen keine Lösung haben. Zum Beispiel hat das unrestringierte Problem (1.2) für die Zielfunktion $f(x) = x^3$ für den zulässigen Bereich $Z = \mathbb{R}$ keine Lösung, während das Problem (1.2) mit der zulässigen Menge $Z := \{x \in \mathbb{R} \mid x \geq 1\}$ die Lösung $x^* = 1$ hat.

Generell müssen in der Optimierung nur Minimierungsprobleme betrachtet werden, denn alle Maximierungsprobleme können wegen

$$\max_{x \in Z} f(x) = \min_{x \in Z} -f(x)$$

in ein Minimierungsproblem überführt werden. Zu den Ungleichungsrestriktionen ist anzumerken, dass die meisten Verfahren hier nur die „ $\leq$ “-Relation zulassen. Dies aber immer erreichbar, denn jede die „ $\geq$ “-Relation enthaltende Ungleichung lässt sich durch Multiplikation mit -1 in die gewünschte Form überführen.

Außerdem besteht zwischen unrestringierten Optimierungsproblemen und nichtlinearen Gleichungssystemen ein besonderer Zusammenhang: Dazu sei $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ mit $n, m \in \mathbb{N} \setminus \{0\}$ und $m \geq n$ eine nichtlineare Abbildung, wobei $F = (F_1, \dots, F_m)^T$ und $F_i : \mathbb{R}^n \rightarrow \mathbb{R}$ für $i = 1, \dots, m$. Das (im Fall $m > n$ überbestimmte) nichtlineare Gleichungssystem $F(x) = 0$ ist allgemein nicht lösbar. Ist aber x^* eine Lösung, so löst x^* auch das unrestringierte Optimierungsproblem

$$\min f(x) := \sum_{i=1}^m (F_i(x))^2 = \|F(x)\|_2^2, \quad (1.3)$$

denn für alle $x \in \mathbb{R}^n$ gilt $f(x) \geq 0 = f(x^*)$. Zum Beispiel in [2] wird das Problem (1.3) als *Ersatzproblem* für das nichtlineare Gleichungssystem $F(x) = 0$ bezeichnet, und zwar auch für den Fall, wenn $F(x) = 0$ nicht lösbar ist. Minimierungsprobleme der Form (1.3) bezeichnet man auch als *Ausgleichsprobleme*, da die Ausgleichung von Messdaten häufig auf Probleme des Typs (1.3) führt. Anstatt des Problems (1.3) werden nichtlineare Ausgleichsaufgaben in der l_2 -Norm oft auch durch das äquivalente Problem

$$\min f(x) := \frac{1}{2} \sum_{i=1}^m (F_i(x))^2 = \frac{1}{2} \|F(x)\|_2^2 \quad (1.4)$$

definiert.

2 Optimierungsverfahren - Überblick

2.1 Lineare Optimierung

Es gibt verschiedene Methoden zur Lösung eines linearen Optimierungsproblems. Am bekanntesten ist dabei das auf G. Dantzig zurückgehende Simplex-Verfahren, das entweder nach endlich vielen Schritten mit einer Lösung endet, oder die Unlösbarkeit bzw. Unbeschränktheit des Problems feststellt. Die Grundidee des Verfahrens besteht darin, ausgehend von einer Ecke des zulässigen Bereichs Z entlang der Kanten von Z zu einer optimalen Ecke zu laufen. Die Begriffe Ecke und Kanten rühren daher, dass der zulässige Bereich im Fall eines linearen Minimierungsproblems ein (konvexes) Polyeder ist. Ist der zulässige Bereich konvex, dann ist jede lokale Lösung auch gleichzeitig globale Lösung des Problems.

Der Simplex-Algorithmus gliedert sich in zwei Phasen. In Phase I wird lediglich eine Startecke aus Z berechnet, mit der dann in die Phase II übergegangen wird, in der dann iterativ versucht wird, aus einer zulässigen Lösung eine neue zulässige Lösung mit besserem Zielfunktionswert zu konstruieren. Zulässige Lösungen sind dabei immer Ecken des zulässigen Bereichs. Das skizzierte Vorgehen wird solange iteriert, bis keine Verbesserung mehr möglich ist, wobei jede Iteration der Lösung eines linearen Gleichungssystems entspricht. Für Details sei auf die weiterführende Literatur verwiesen.

2.2 Nichtlineare Optimierung

Hier gibt es praktisch (noch) fast keine Methoden, bei deren Anwendung man in den meisten Fällen eine Lösung des Problems erhält, die mit Sicherheit eine globale Lösung ist. Deshalb ist oft nur die Berechnung von lokalen Lösungen möglich. Welche Methode zur Berechnung einer lokalen Lösung verwendet wird, hängt von der Problemstellung bzw. den Eigenschaften der Zielfunktion und den Nebenbedingungen ab.

Ist zum Beispiel die Zielfunktion nur stetig, aber nicht differenzierbar, so ist die Verwendung von ableitungsfreien Verfahren erforderlich, wie zum Beispiel das Simplex-Verfahren von Nelder und Mead oder das Intervallhalbierungsverfahren. Ableitungsfreie Verfahren sind meist nur iterativ und weisen oft nur sehr langsame Konvergenz auf, sind aber relativ robust.

Verfahren, bei denen der Gradient der Zielfunktion benötigt wird, sind z.B. das Gradientenverfahren und Quasi-Newton-Verfahren. Diese Verfahren sind schneller als die ableitungsfreien Methoden, insbesondere, wenn der Gradient schnell berechnet werden kann. Ist die oft numerisch teure Berechnung der Hesse-Matrix gerechtfertigt, so kann diese Information zu einer weiteren Beschleunigung der Problemlösung führen. Ein Verfahren, das die Hesse-Matrix nutzt, ist zum Beispiel das Newton-Verfahren.

3 Lösung von Optimierungsproblemen mit Hilfe von Matlab

3.1 Minimierung ohne Ableitung und ohne Nebenbedingungen

Gegeben sei die quadratische Funktion $f(x) := (x - \begin{pmatrix} -2 \\ 1 \end{pmatrix})^T A (x - \begin{pmatrix} -2 \\ 1 \end{pmatrix})$ mit $A := \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$. Gesucht ist ein Minimum $x^* = (x_1^*, x_2^*)^T$ von f . Dazu ist zunächst die Funktion f zu implementieren:

```
function[f,df] = quadfkt(x)
    A = [1,-1; -1, 2];
    y = x - [-2;1];
    f = transpose(y)*A*y;
    df = 2*A*y;
```

Die Ableitung **df** wird bei der Minimierung mit **fminsearch** nicht notwendig benötigt, kann die Minimierung aber erheblich beschleunigen. Zur Berechnung des Minimums genügt nun der Aufruf:

```
x0 = [1;2];
[x,fx] = fminsearch(@quadfkt,x0);
```

x0 stellt dabei den Startwert des iterativen Minimierungsverfahrens dar, **x** ist das berechnete (lokale) Minimum x^* und **fx** der Funktionswert in **x**. Die Funktion **fminsearch** stellt eine Implementierung des ableitungsfreien Nelder-Mead-Verfahrens dar. Die Matlab-Dokumentation zur **fminsearch** erklärt eine Vielzahl von Optionen, die es ermöglichen, den Algorithmus zu beeinflussen. Dies erfolgt mit Hilfe der Funktion **optimset**. Zum Beispiel bewirkt der Aufruf

```
x0 = [1;2];
opt = optimset('Display','iter','TolX',1.0e-005,'MaxIter',100);
[x,fx] = fminsearch(@quadfkt,x0,opt);
```

dass detaillierte Informationen über den Verlauf der Iteration ausgegeben werden, das Abbruchkriterium **TolX** verändert wird und der Algorithmus auf maximal 100 Iterationen beschränkt wird.

Es besteht natürlich auch die Möglichkeit, parameterabhängige Funktionen zu minimieren. Soll zum Beispiel in der Beispielfunktion **quadfkt** die Matrix A nicht fest, sondern variabel sein, so ändert man diese wie folgt ab:

```
function[f,df] = quadfkt2(x,A)
    y = x - [-2;1];
    f = transpose(y)*A*y;
    df = 2*A*y;
```

Wichtig ist dabei immer, dass die zu minimierende Variable **x** am Anfang der Liste der Funktionsvariablen steht! Die Funktion **fminsearch** ist nun wie folgt aufzurufen:

```
[x,fx] = fminsearch(@(x) quadfkt2(x,A),x0,opt);
```

Matlab stellt keine **Maximierungsroutinen** zur Verfügung. Dies ist, wie bereits weiter oben erwähnt wurde, auch nicht notwendig, denn alle Maximierungsprobleme können wegen

$$\max_{x \in D} f(x) = \min_{x \in D} -f(x)$$

in ein Minimierungsproblem überführt werden. Für die Implementierung hat man also zwei Möglichkeiten: Entweder man definiert die entsprechende Matlab-Funktion für $-f$ oder mit der Definition von f wird wie folgt verfahren:

```
[x,fx] = fminsearch(@(x) -quadfkt2(x,A),x0,opt);
fx = -fx;
```

Die letzte Zeile sorgt dafür, dass **fx** dann wirklich das Maximum ist.

3.2 Minimierung mit Verwendung der Ableitung

Das zweite Argument `df` der Funktionen `quadfun` und `quadfun2` wurde bisher nicht benötigt. Die bis hierher vorgeschlagenen Minimierungsaufrufe hätten genauso funktioniert, wenn `[f,df]` in `quadfun.m` und `quadfun2.m` durch `f` ersetzt und die `df`-Zeile gelöscht wird. Die Verwendung des Gradienten `df` beschleunigt die Minimierung erheblich, wenn man zum Beispiel den Aufruf

```
x0 = [1;2];
opt = optimset('Display','iter','GradObj','on');
[x,fx] = fminunc(@quadfkt,x0,opt);
```

wählt. Die Funktion `fminunc` verwendet dann den Gradienten bei der Minimierung. Eine weitere Beschleunigung ist durch die Verwendung der Hessematrix H_f möglich. Weiteres erläutert die Matlab-Dokumentation zu `fminunc`.

3.3 Minimierung unter Nebenbedingungen

Die Funktion `fmincon` erlaubt die Minimierung der Zielfunktion mit Nebenbedingungen der folgenden Form:

$$c_{\text{ineq}}(x) \leq 0 \quad (3.1)$$

$$c_{\text{eq}}(x) = 0 \quad (3.2)$$

$$A_{\text{ineq}}x \leq b_{\text{ineq}}(x) \quad (3.3)$$

$$A_{\text{eq}}x = b_{\text{eq}}(x) \quad (3.4)$$

$$l \leq x \leq u \quad (3.5)$$

x , b_{ineq} , b_{eq} , l und u sind Vektoren, A_{ineq} und A_{eq} Matrizen, $c_{\text{ineq}}(x)$ und $c_{\text{eq}}(x)$ Funktionen, die Vektoren liefern. $c_{\text{ineq}}(x)$ und $c_{\text{eq}}(x)$ dürfen nichtlinear sein. Die Matlab-Dokumentation zu `fmincon` erklärt detailliert, wie die Nebenbedingungen angesteuert werden. Durch die Übergabe der Gradienten der Zielfunktion f und $c_{\text{ineq}}(x)$, $c_{\text{eq}}(x)$ kann die Konvergenz des Verfahrens erheblich beschleunigt werden. Die Auswahl des Minimierungsverfahrens hängt von den gegebenen Nebenbedingungen (3.1) - (3.5) ab und kann ferner mittels `optimset` gesteuert werden. Im folgenden seien einige Beispielaufrufe dokumentiert. Die Nebenbedingungen (3.1) - (3.5) können einzeln verwendet werden, aber auch miteinander kombiniert werden.

3.3.1 Lineare Gleichungsbedingungen

```
linA = [-1,-1];
linb = 1;
opt = optimset('Algorithm','interior-point');
[x,fx,exitflag,output] = fmincon(@quadfkt,x0,[],[],linA,linb,[],[],[],opt);
```

3.3.2 Einfache Ungleichungsnebenbedingungen (auch Box-Constraints genannt)

```
lb = [0;0];
ub = [0.5 ; 0.5];
[x,fx,exitflag,output] = fmincon(@quadfkt,x0,[],[],[],[],lb,ub,[],opt);
```

3.3.3 Nichtlineare Gleichungsnebenbedingungen

Zunächst wird in einer Funktion `coneq.m` die Nebenbedingung der Form (2) definiert (im nachfolgenden Beispiel ist dies $x_1^2 + x_2^2 = \frac{1}{4}$):

```
function[c,ceq] = coneq(x)
c=[];
ceq = x(1)^2 + x(2)^2 - 0.25;
```

Hier ist `ceq` skalarwertig. Werden mehrere Nebenbedingungen benötigt, so wird `ceq` als Spaltenvektor verwendet. Die Minimierung wird mit dem folgenden Aufruf gestartet:

```
x0 = [0,0];
opt = optimset('Display','iter');
[x,fx,exitflag,output] = fmincon(@quadfkt,x0,[],[],[],[],[],[],@cineq,opt);
```

3.3.4 Nichtlineare Ungleichungsnebenbedingungen

Definition der nichtlinearen Ungleichungsnebenbedingungen und Aufruf des Minimierers geschehen analog zu den nichtlinearen Gleichungsnebenbedingungen. Der Unterschied liegt in der Definition der Nebenbedingungsfunktion `conineq.m`, die jetzt die Ungleichungsnebenbedingung liefert (hier: $x_1^2 + x_2^2 \leq \frac{1}{4}$):

```
function[c,ceq] = conineq(x)
c = x(1)^2 + x(2)^2 - 0.25;
ceq=[];
```

Die Minimierung wird analog wie eben mit dem folgenden Aufruf gestartet:

```
x0 = [0,0];
opt = optimset('Display','iter');
[x,fx,exitflag,output] = fmincon(@quadfkt,x0,[],[],[],[],[],[],@conineq,opt);
```

3.4 Lösung von Ausgleichsproblemen

Zur Lösung von Problemen des Typs (1.3) bzw. (1.4) wird u.a. die Funktion `lsqnonlin` bereitgestellt. Ihre Verwendung wird im Beispiel 4.2 weiter unten demonstriert.

4 Beispiele

Beispiel 4.1 Eine Großmolkerei wird monatlich mit 24 Millionen Liter Milch beliefert, die zu Quark und Käse verarbeitet werden. Für die Herstellung von 1 kg Quark werden 4.62 Liter, für die von 1 kg Käse 11.23 Liter Milch benötigt. Ferner dürfen aus technischen Gründen die produzierten Massen an Quark und Käse zusammen 4000 Tonnen nicht übersteigen. Außerdem müssen aufgrund von Lieferverpflichtungen mindestens 1000 Tonnen Quark und 500 Tonnen Käse produziert werden. Pro Kilogramm Quark verdient die Molkerei nach Abzug aller Produktionskosten 11 Cent, bei einem Kilo Käse sind es 14 Cent. Wie viele Tonnen Käse und Quark müssen pro Monat produziert werden, damit der Gewinn unter den gegebenen Voraussetzungen maximiert wird?

Sei q die monatlich produzierte Menge an Quark in kg und k die monatlich produzierte Menge an Käse in kg. Dies führt auf das folgende *Maximierungsproblem*:

$$\begin{array}{llll} \max & 0.11 \cdot q & + & 0.14 \cdot k \\ \text{u.d.N.} & 4.62 \cdot q & + & 11.23 \cdot k \leq 24 \cdot 10^6 \\ & q & + & k \leq 4 \cdot 10^6 \\ & q & & \geq 10^6 \\ & & k & \geq 5 \cdot 10^5 \end{array}$$

Wegen den Bemerkungen in Abschnitt 1, müssen wir dieses Problem in ein Minimierungsproblem überführen, außerdem müssen die „ $\geq$ “-Relationen in „ $\leq$ “-Relationen umgewandelt werden. Es ist deshalb folgendes Minimierungsproblem zu lösen:

$$\begin{array}{rcll} \min & -0.11 \cdot q & - & 0.14 \cdot k \\ \text{u.d.N.} & 4.62 \cdot q & + & 11.23 \cdot k \leq 24 \cdot 10^6 \\ & q & + & k \leq 4 \cdot 10^6 \\ & -q & & \leq -10^6 \\ & & -k & \leq -5 \cdot 10^5 \end{array}$$

Zur numerischen Lösung des Problems mit Matlab, wird für die zu minimierende Zielfunktion $f(q, k) = -0.11 \cdot q - 0.14 \cdot k$ eine Funktion `gewinnfunktion.m` definiert, wobei wir zulassen, dass sich der Gewinn pro Kilogramm Quark bzw. Käse auch verändern kann:

```
function[wert] = gewinnfunktion(param,gewinn)
    wert = -gewinn(1)*param(1) - gewinn(2)*param(2);
```

Analog werden die Nebenbedingungen als Funktion definiert, wobei auch hier die Möglichkeit zugelassen wird, dass sich die Liefer-/Produktionsmengen ändern können:

```
function[c,ceq] = nebenbed(param,minKaese,minQuark,maxGesamt,milchMenge)
    % es gibt keine Gleichungsnebenbedingungen
    ceq = [];
    % die Ungleichungsnebenbedingungen
    c = [4.62*param(1) + 11.23*param(2) - milchMenge; ...
        param(1) + param(2) - maxGesamt; ...
        minQuark - param(1); ...
        minKaese - param(2)];
```

Das Maximierungsproblem wird nun durch folgende Eingaben gelöst:

```
gewinn = [0.11,0.14];
milchMenge = 24*10^6;
maxGesamt = 4*10^6;
minKaese = 5*10^5;
minQuark = 10^6;
startwert = [1,1];
[x,fx] = fmincon(@(param) gewinnfunktion(param,gewinn),startwert, ...
    [],[],[],[],[],@(param) nebenbed(param,minKaese,minQuark,maxGesamt,milchMenge), ...
    optimset('Display','off','Algorithm','active-set'));
fx = -fx;
```

Auf diese Weise hat man berechnet, dass das Milchwerk einen maximalen Gewinn von $fx = 465052.95$ Euro erzielt, wenn es $x(1) = 3164901.7$ kg Quark und $x(2) = 835098.34$ kg Käse produziert.

Beispiel 4.2 Die in der nachfolgenden Tabelle gegebenen Koordinaten (x_i, y_i) sollen durch eine Gerade $f(x) = ax + b$ approximiert werden, wobei die quadratische Abweichung der Punkte zur Geraden minimal sein soll:

i	1	2	3	4	5	6	7	8	9	10
x_i	1.9	2	4	4.5	5	6	7.8	8	9	10.3
y_i	0.087	1.5	2.03	4.1	6	6.9	7.4	8.03	8.2	10.03

Offensichtlich besteht die Lösung hier in der Anwendung der Methode der kleinsten Quadrate, d.h. wir müssen die Parameter a und b aus den gegebenen Koordinaten so bestimmen, dass der Fehler

$$F(a, b) := \sqrt{\sum_{k=1}^{10} (y_i - ax_i - b)^2} \quad (4.3)$$

minimal wird. Zur Lösung des Problems genügt es, die Fehlerfunktion $F(a, b)$ als Matlab-Funktion zu definieren; die Parameter a und b können dann mit der Funktion `fminsearch` berechnet werden. Wegen der Monotonie der Wurzelfunktion kann alternativ auch das Problem

$$(F(a, b))^2 := \sum_{k=1}^{10} (y_i - ax_i - b)^2 \quad (4.4)$$

minimiert werden, was also auf das Problem (1.3) führt. Zur Lösung müssen wir lediglich die Fehlerfunktion $(F(a, b))^2$ in Matlab definieren:

```
function[wert] = fehlerfunktion(param,x,y)
    wert = y - param(1)*x - param(2);
```

Die Parameter a und b , die über den Parameter `param` definiert werden, berechnen wir dann z.B. mit Hilfe der Funktion `lsqnonlin.m` wie folgt:

```
x = [1.9 , 2 , 4 , 4.5 , 5 , 6 , 7.8 , 8 , 9 , 10.3];
y = [0.087 , 1.5 , 2.03 , 4.1 , 6 , 6.9 , 7.4 , 8.03 , 8.2 , 10.03];
startwert = [0,0];
lb = [-inf,-inf];
ub = [inf,inf];
opt = optimset('Display','off','TolX',1e-010);
[param,fehler] = lsqnonlin(@fehlerfunktion,startwert,lb,ub,opt,x,y);
```

Als Ergebnis erhält man für die Beispieldaten:

```
param =

    1.1089    -1.0593

fehler =

    7.5943
```

Als beste Approximation an die Datenpunkte unter allen Geraden ergibt sich damit $f(x) = 1.1089x - 1.0593$, der Approximationsfehler beträgt $F(1.1089, -1.0593) = \sqrt{7.5943} = 2.7558$.

Literatur

- [1] Alt, W.: Nichtlineare Optimierung. Vieweg, Braunschweig/Wiesbaden, 2002
- [2] Geiger, C.; Kanzow, C.: Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben. Springer, Berlin/Heidelberg, 1999
- [3] Reemtsen, R.: Lineare Optimierung. Shaker Verlag, Aachen, 2001.
- [4] Werner, J.: Numerische Mathematik 2. Vieweg, Braunschweig / Wiesbaden, 1992