Engineering Optimization

Dieter Kraft[1]

[1]Munich University of Applied Siences, email: kraft@fhm.edu, 19. Dezember 2005

Inhaltsverzeichnis

Kapitel 1

Einführung

Eine große Anzahl technischer Probleme hängt von einem Satz von variabel anzunehmenden Parametern ab. Es ist eine Ingenieurskunst, diese Parameter im Verlauf der Entwurfsphase so einzustellen, dass ein gewähltes Kriterium, z.B. das Entwurfsgewicht, minimal wird. Dabei sind gewisse Beschränkungen, z.B. zulässige Spannungen, einzuhalten.

Mathematisch lässt sich dieses Problem wie folgt beschreiben

$$\text{minimiere } \{f(x) \mid x \in \mathcal{S} \subset \mathbb{R}^n\}, \tag{1.1}$$

wobei die zulässige Menge $\mathcal{S}$ durch Gleichungen und Ungleichungen definiert wird:

$$\mathcal{S} = \{x \mid c_i(x) = 0,\ i = 1, \ldots, m_{eq},\ c_i(x) \geq 0,\ i = m_{eq} + 1, \ldots, m\}. \tag{1.2}$$

In Gl. (1.1) repräsentiert die skalare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ die Kosten- oder Zielfunktion, und in Gl. (1.2) bildet die vektorwertige Funktion $c : \mathbb{R}^n \to \mathbb{R}^m$ die einzelnen Beschränkungen ab. Die variablen Parameter sind durch den Vektor x charakterisiert.

Es werden verschiedene Klassen von Optimierungsproblemen unterschieden. Wenn alle Funktionen linear (in den Variablen) sind, dann spricht man von linearer Optimierung oder auch linearer Programmierung, ist die Zielfunktion quadratisch und sind die Nebenbedingungen linear, dann handelt es sich um quadratische Optimierung, und wenn einige oder alle Funktionen nichtlinear sind, dann liegt nichtlineare Optimierung vor.

Außerdem unterscheidet man endlich-dimensionale oder Parameteroptimierung und unendlich-dimensionale Optimierung oder optimale Steuerung. Letztere liegt z.B. bei der Bahnplanung von Robotern vor, wenn die Elemente von x Funktionen der Zeit sind.

Kapitel 2

Lineare Optimierung

Das lineare Optimierungsproblem, oder lineares Programm, in der Standardform lautet

$$\begin{aligned} &\text{maximiere } \sum_{j=1}^{n} c_j x_j, \\ &\text{beschränkt durch } \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ &\qquad\qquad x_j \geq 0, \quad j = 1, \ldots, n. \end{aligned} \tag{2.1}$$

Die Konstanten a_{ij}, b_i und c_j sind die Daten des Problems, die Variablen x_j die gesuchten Optimalwerte.

Beispiel 1. (Resourcen-Allokation.)
Ein typisches Beispiel für die lineare Optimierung ist das Problem der Allokierung von Resourcen, wobei die Problemdaten folgende Bedeutung haben:

$c_j =$ Profit pro Einheit des Produkts j
$b_i =$ verfügbare Einheiten des Rohmaterials i
$a_{ij} =$ benötigte Einheiten des Rohmaterials i zur Erzeugung einer Einheit des Produkts j
$x_j =$ erzeugte Einheiten des Produkts j

Ein weitere lineare Optimierungsaufgabe ist das Problem der Mixtur von Diäten [20]. □[1]

Die Menge der Variablen, welche die Ungleichungen in (1) erfüllen, heißt zulässiges Gebiet. Nur wenn es nicht-leer ist, ist das Problem sinnvoll gestellt. Jede lineare Optimierungsaufgabe kann in die Form (2.1) überführt werden.

[1] □ kennzeichnet das Ende eines Beispiels oder einer Definition.

Kapitel 3

Simplexverfahren

Eine Lösungsmethode für das Problem (2.1) ist das Simplexverfahren, das in vielen Literaturquellen beschrieben ist, siehe z.B. [44]. Die Ungleichungen in Gl. (2.1) werden durch Definition nicht-negativer Schlupfvariablen w_i in Gleichungen überführt, und das Simplex-Tableau wird wie folgt geschrieben:

$$\zeta = \sum_{j=1}^{n} c_j x_j, \tag{3.1}$$

$$w_i = b_i - \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, m. \tag{3.2}$$

Die Variablen auf der rechten Seite der Gln. (2–3) werden als Nicht-Basisvariablen bezeichnet, die auf der linken Seite von Gl. (3) als Basisvariablen. Die Menge der Basisvariablen mit der Anzahl m heißt Basis, die der Menge der Nicht-Basisvariablen mit der Anzahl n heißt Nicht-Basis. Die Nicht-Basisvariablen haben immer den Wert Null, die Basisvariablen dürfen nicht negativ werden.

Das Simplexverfahren ist ein iteratives Verfahren, bei dem in jedem Schritt genau eine Variable aus der Basis gegen genau eine Variable aus der Nicht-Basis getauscht wird. Dieser Tausch legt nahe, dass wir die Menge der Variablen $[x_j,\ w_i]$ zu einem Variablenvektor x zusammenfassen, indem wir die w_i an die urprünglichen x_j „anhängen“ mit $x_{n+i} = w_i$. Der Vektor x enthält nun $n + m$ Elemente, und das Tableau wird zu

$$\zeta = \sum_{j=1}^{n} c_j x_j, \tag{3.3}$$

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, m. \tag{3.4}$$

Nun sei der Indexvektor der Basisvariablen mit $\mathcal{B}$, der der Nicht-Basisvariablen mit $\mathcal{N}$ bezeichnet. Zu Beginn des Verfahrens ist $\mathcal{B} = \{n+1,\ n+2, \ldots, n+m\}$ und $\mathcal{N} = \{1,\ 2, \ldots, n\}$, aber durch die folgenden Tauschoperationen ändern sich diese Vektoren. Damit ändern sich auch die Daten des Tableaus, und man kann mit dem Iterationsindex p schreiben:

$$\zeta = \zeta^p + \sum_{j=1}^{n} c_j^p x_j, \tag{3.5}$$

$$x_{n+i} = b_i^p - \sum_{j=1}^{n} a_{ij}^p x_j, \quad i = 1, \ldots, m. \tag{3.6}$$

Jede Iteration besteht aus drei Schritten: 1) der Auswahl der Variablen, die in die Basis eintritt, 2) der Auswahl der Variablen, die die Basis verlässt und 3) der durch den Tausch notwendigen algebraischen Operationen zum Update der Daten $a_{ij}^p,\ b_i^p,\ c_j^p$.

3.1 Basis-Eintritt

Eine Variable, die in die Basis eintritt, wird $x_j \geq 0$. Damit der Wert der Zielfunktion sich vergrößert, $\zeta^{p+1} > \zeta^p$, muss der Daten-Koeffizient der entsprechenden Variablen $c_j > 0$ sein. Die Austauschregel (pivot rule) lautet dementsprechend

$$\text{Regel I}: \qquad k = \arg\max_{j \in \mathcal{N}} \{c_j > 0\}. \tag{3.7}$$

Gibt es mehrere k-Werte, dann wähle den kleinsten. Sind alle $c_j \leq 0$, dann ist das Verfahren beendet.

3.2 Basis-Austritt

Die eintretende Variable soll so gross wie möglich gemacht werden, ohne dass die anderen Basisvariablen negativ werden. Eine dieser, nämlich diejenige, die die Basis verlässt, wird zu Null gemacht; und erfüllt damit die Qualifikation als Nicht-Basisvariable. Die Forderung ist also

$$b_i^p - a_{ik}^p x_k \geq 0, \quad i \in \mathcal{B},$$

oder als Austauschregel:

$$\text{Regel II}: \qquad l = \arg\max_{i \in \mathcal{B}} \left[\frac{a_{ik}^p}{b_i^p} \right], \tag{3.8}$$

wobei eine Division $\frac{0}{0}$ den Wert Null ergibt.

Gibt es wiederum mehrere solcher l-Indizes, dann wähle wieder den kleinsten. Die Auswahl des kleinsten Index in Abschnitt 3.1 und 3.2 wird als Regel von BLAND bezeichnet [44, Seite 36].

3.2.1 Unbeschränktes Problem

In Regel II kann das Problem auftreten, dass alle Quotienten $\frac{a_{ik}}{b_i}$ nicht-positiv sind. In diesem Fall ist das Problem unbeschränkt, da keine Basisvariable Null wird, wenn die eintretende Variable beliebig vergrößert wird. D.h. die Zielfunktion kann beliebig vergrößert werden.
Beispiel 2. (Unbeschränktes Problem.)
Betrachten Sie folgendes Tableau

$$\begin{aligned} \zeta &= 0 + 5x_1 + 4x_2 \\ x_3 &= 4 + 1x_1 - 0x_2 \\ x_4 &= 3 - 0x_1 - 2x_2 \\ x_5 &= 1 + 2x_1 + 2x_2 \end{aligned}$$

Die eintretende Variable ist x_1, die Quotienten zur Auswahl der austretenden Variablen sind $\{-\frac{1}{4}, \frac{0}{3}, -\frac{2}{1}\}$, keiner ist positiv, d.h. das Problem ist unbeschränkt. □

3.2.2 Degeneriertes Problem

In Regel II kann als weiteres Problem auftreten, dass für positives a_{ik} und $b_i = 0$ ein Quotient $\frac{a_{ik}}{b_i}$ positiv unendlich $+\infty$ wird. In diesem Fall wird das Problem degeneriert genannt. Die Variable x_i ist Kandidat zum Verlassen der Basis. Der Kandidat für die eintretende Variable x_k kann aber nicht positiv werden, da sonst x_i sofort negativ wird. Trotzdem kann man formal den Pivot-Schritt durchführen, allerdings bleibt der Wert der Kostenfunktion konstant.
Beispiel 3. (Degeneriertes Problem.)
Betrachten Sie folgendes Ausgangstableau

$$\begin{aligned}\zeta &= 0 + 5x_1 + 2x_2\\ x_3 &= 6 - 1x_1 - 1x_2\\ x_4 &= 0 - 1x_1 + 1x_2\end{aligned}$$

x_1 tritt in die Basis ein; der Test der Quotienten $\frac{a_{i1}}{b_i}$ zeigt, dass x_4 die Basis verlässt, da $1/0 > 1/6$ ist. Das neue Tableau ist

$$\begin{aligned}\zeta &= 0 - 5x_4 + 7x_2\\ x_3 &= 6 + 1x_4 - 2x_2\\ x_1 &= 0 - 1x_4 + 1x_2\end{aligned}$$

Beachte, dass sich (typisch für degenerierte Probleme) die Kostenfunktion ζ nicht verändert hat. Als nächste tritt x_2 in die Basis, x_3 verlässt sie ($2/6 > -1/0$) mit folgendem Tableau:

$$\begin{aligned}\zeta &= 21 - \frac{3}{2}x_4 - \frac{7}{2}x_3\\ x_2 &= 03 + \frac{1}{2}x_4 - \frac{1}{2}x_3\\ x_1 &= 03 - \frac{1}{2}x_4 - \frac{1}{3}x_3\end{aligned}$$

Dieses ist nicht mehr verbesserbar. □

Übung: Verfolgen Sie den weiteren Lösungsverlauf anhand des Beipiels (3.1) in [44, Seite 29].

3.3 Austausch-Algebra

Die Zeile l des Tableaus wird nun nach der eintretenden Variablen aufgelöst und deren rechte Seite in alle anderen Zeilen eingesetzt (Zeilenoperationen der linearen Algebra). Damit ist die Iteration p abgeschlossen.

3.4 Beispiel I

Es sei folgendes Optimierungsproblem betrachtet:[1]

$$\begin{aligned}\max\ & 3x_1 + 5x_2\\ \text{s.t.}\quad & x_1 \leq 4\\ & 2x_2 \leq 12\\ & 3x1 + 2x2 \leq 18\\ & x_1, x_2 \geq 0\end{aligned}$$

[1] "s.t. = subjet to" bedeutet „beschränkt durch"

Das zugehörige Ausgangs-Tableau ist

$$\begin{aligned}\zeta &= 00 + 3x_1 + 5x_2\\ x_3 &= 04 - 1x_1 - 0x_2\\ x_4 &= 12 - 0x_1 - 2x_2\\ x_5 &= 18 - 3x_1 - 2x_2\end{aligned}$$

Iteration 1:

Schritt 1: Anwendung von Regel I:[2]

$$k = \arg \max_{j \in \mathcal{N}=\{1,2\}} \{c_1^0, c_2^0\} = \arg\max\{3, 5\} = 2.$$

Schritt 2: Anwendung von Regel II:

$$l = \arg \max_{i \in \mathcal{B}=\{3,4,5\}} \left[\frac{a_{32}^0}{b_3^0}, \frac{a_{42}^0}{b_4^0}, \frac{a_{52}^0}{b_5^0}\right] = \left[\frac{0}{4}, \frac{2}{12}, \frac{2}{18}\right] = 4.$$

Schritt 3: Die Austausch-Algebra ergibt folgendes erneuertes Tableau:

$$\begin{aligned}\zeta &= 30 + 3x_1 - \frac{5}{2}x_4\\ x_3 &= 04 - 1x_1 - 0x_4\\ x_2 &= 06 - 0x_1 - \frac{1}{2}x_4\\ x_5 &= 06 - 3x_1 - 1x_4\end{aligned}$$

Iteration 2:

Schritt 1: Anwendung von Regel I:

$$k = \arg \max_{j \in \mathcal{N}=\{1,4\}} \{c_1^1, c_4^1\} = \arg\max\{3, -\frac{5}{2}\} = 1.$$

Schritt 2: Anwendung von Regel II:

$$l = \arg \max_{i \in \mathcal{B}=\{2,3,5\}} \left[\frac{a_{21}^1}{b_2^1}, \frac{a_{31}^1}{b_3^1}, \frac{a_{51}^1}{b_5^1}\right] = \left[\frac{1}{4}, \frac{0}{6}, \frac{3}{6}\right] = 5.$$

Schritt 3: Die Austausch-Algebra ergibt folgendes erneuertes Tableau:

$$\begin{aligned}\zeta &= 36 - 1x_5 - \frac{3}{2}x_4\\ x_3 &= 02 + \frac{1}{3}x_5 - \frac{1}{3}x_4\\ x_2 &= 06 - \frac{0}{3}x_5 - \frac{1}{2}x_4\\ x_1 &= 02 - \frac{1}{3}x_5 + \frac{1}{3}x_4\end{aligned}$$

Nach dieser zweiten Iteration sind beide Koeffizienten der Variablen in der Zielfunktion ζ kleiner als Null; damit ist die Optimallösung $\zeta = 36$ erzielt. Den Wert der Lösungvariablen liest man in auf der linken Seite von Zeile 4 und 5 des Tableaus ab: $x_1 = 2$ und $x_2 = 6$.

Übung: Es ist instruktiv, in Iteration 1 in Regel I nicht den größten Wert $c_2 = 5$ als Koeffizient der Pivot-Variablen x_2 zu wählen, sondern den (geringeren) Wert $c_1 = 3$ und damit x_1 als Pivot-Variable. Rechnen Sie nun das Beispiel durch. Welches Vorgehen ist effektiver?

[2] Der obere Index zeigt die Nummer der Iteration an; 0 ist das Ausgangsdatum.

3.5 Beispiel II

In Beispiel I liegen die Nicht-Basisvariablen $x_1^0 = 0$ und $x_2^0 = 0$ im zulässigen Bereich, deshalb sind alle Basisvariablen $x_3^0 = 4$, $x_4^0 = 12$ und $x_5^0 = 18$ grösser als Null. Ist dies nicht der Fall, dann muss erst ein zulässiger Ausgangspunkt gefunden werden. Dies geschieht in einer vorgeschalteten Phase-I des Simplex-Algorithmus. Hierbei wird eine Hilfsvariable x_0 eingeführt, deren (ursprünglich positiver) Wert zu Null minimiert wird,[3] und die dann wieder aus dem Problem eliminiert wird. Das aus Beispiel I hervorgegangene modifizierte Beispiel II erläutert die Vorgehensweise.[4]

Es sei folgendes Optimierungsproblem vorgelegt:

$$\begin{aligned} \max\ & 3x_1 + 5x_2 \\ \text{s.t.}\quad & 1x_1 \leq 4 \\ & 2x_2 \leq 12 \\ & 3x1 + 2x2 \leq 18 \\ & 1x_1 + 2x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Das zugehörige Ausgangs-Tableau ist

$$\begin{aligned} \zeta &= +00 + 3x_1 + 5x_2 \\ x_3 &= +04 - 1x_1 - 0x_2 \\ x_4 &= +12 - 0x_1 - 2x_2 \\ x_5 &= +18 - 3x_1 - 2x_2 \\ x_6 &= -02 + 1x_1 + 2x_2 \end{aligned}$$

Hier ist $x_6 < 0$ negativ, deshalb wird zunächst folgendes Pase-I-Problem betrachtet:

$$\begin{aligned} \max\ & -x_0 \\ \text{s.t.}\quad & 1x_1 - x_0 \leq 4 \\ & 2x_2 - x_0 \leq 12 \\ & 3x1 + 2x2 - x_0 \leq 18 \\ & 1x_1 + 2x_2 - x_0 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Das zugehörige Ausgangs-Tableau ist

$$\begin{aligned} \xi &= +00 - x_0 \\ x_3 &= +04 + x_0 - 1x_1 - 0x_2 \\ x_4 &= +12 + x_0 - 0x_1 - 2x_2 \\ x_5 &= +18 + x_0 - 3x_1 - 2x_2 \\ x_6 &= -02 + x_0 + 1x_1 + 2x_2 \end{aligned}$$

Hier kann nur die Variable x_0 in die Basis eintreten, und der Zweck dieser Variablen wird sofort klar, wenn man die negativen Basisvariablen (hier die Variable x_6) im Tableau betrachtet: x_0 wird gebraucht um die kleinste (negative) Basisvariable zu Null zu machen und damit

[3] In der Praxis wird ihr negativer Wert maximiert.

[4] Skizzieren Sie die Geometrie des Problems.

in die Nicht-Basis zu tauschen. Damit ergibt sich folgende modifizierte Pivot-Regel für die erste Iteration der Phase-I:

$$\text{Regel } I_{\text{mod}} : x_0 \text{ geht in die Basis} \tag{3.9}$$

und

$$\text{Regel } II_{\text{mod}} : x_l \text{ mit } l = \arg\min_{i \in \mathcal{B}^0} \{b_i^0\} \text{ verlässt die Basis} \tag{3.10}$$

Anmerkung 1: Ist der Wert der Kostenfunktion am Ende der Phase-I $\xi < 0$, dann ist das zulässige Gebiet des Originalproblems leer, das Problem ist unzulässig.

Iteration 1:[5]

Schritt 1: Anwendung von Regel I_{mod}: x_0 geht in die Basis.

Schritt 2: Anwendung von Regel II_{mod}: x_6 verlässt die Basis.

Schritt 3: Die Austausch-Algebra ergibt folgendes neues Tableau:

$$\begin{aligned}
\xi &= -02 - x_6 + 1x_1 + 2x_2 \\
x_3 &= +06 + x_6 - 2x_1 - 2x_2 \\
x_4 &= +14 + x_6 - 1x_1 - 4x_2 \\
x_5 &= +20 + x_6 - 4x_1 - 4x_2 \\
x_0 &= +02 + x_6 - 1x_1 - 2x_2
\end{aligned}$$

Iteration 2:

Schritt 1: Anwendung von Regel I: x_2 tritt in die Basis ein.

Schritt 2: Anwendung von Regel II:

$$l = \arg\max_{i \in \mathcal{B}} \left[\frac{2}{6}, \frac{4}{14}, \frac{4}{20}, \frac{2}{2}\right] = 0,$$

d.h. x_0 verlässt (wieder) die Basis.

Schritt 3: Die Austausch-Algebra ergibt folgendes neues Tableau:

$$\begin{aligned}
\xi &= 00 + 0x_6 + 0x_1 - 1x_0 \\
x_3 &= 04 + 0x_6 - 1x_1 + 1x_0 \\
x_4 &= 10 - 1x_6 + 1x_1 + 2x_0 \\
x_5 &= 16 - 1x_6 - 2x_1 + 2x_0 \\
x_2 &= 01 + \frac{1}{2}x_6 - \frac{1}{2}x_1 - \frac{1}{2}x_0
\end{aligned}$$

Kein Koeffizient in der Zielfunktion ist positiv, d.h. das Optimum der Phase-I ist gefunden mit $x_1 = 0$ und $x_2 = 1$, und $x_0 = 0$ kann wieder aus dem Problem eliminiert werden. Das Originalproblem ist zulässig, da $\xi = 0$, s.a. Anmerkung 1.

Übung: Skizzieren Sie die Geometrie dieses Problems und kennzeichnen Sie den zulässigen Startpunkt.

Nun kann die Phase-II, d.h. die eigentliche Lösungsberechnung, beginnen. Zunächst muss im Optimaltableau der Phase-I die modifizierte Zielfunktion ξ durch die originale Zielfunktion

[5] Merke: Nur in Iteration 1 werden die modifizierten Regeln angewendet!

$\zeta = 3x_1 + 5x_2$ durch Einsetzen von $x_2 = 1 - \frac{1}{2}x_1 + \frac{1}{2}x_6$ gemäß der letzten Zeile des letzten Tableaus berechnet werden:

$$\zeta = 3x_1 + 5\left(1 - \frac{1}{2}x_1 + \frac{1}{2}x_6\right) = 5 + \frac{1}{2}x_1 + \frac{5}{2}x_6$$

Das Starttableau für Phase-II ist demnach

$$\begin{aligned}
\zeta &= 05 + \frac{5}{2}x_6 + \frac{1}{2}x_1 \\
x_3 &= 04 + 0x_6 - 1x_1 \\
x_4 &= 10 - 1x_6 + 1x_1 \\
x_5 &= 16 - 1x_6 - 2x_1 \\
x_2 &= 01 + \frac{1}{2}x_6 - \frac{1}{2}x_1
\end{aligned}$$

Iteration 3:

Schritt 1: Anwendung von Regel I: x_6 tritt in die Basis ein.

Schritt 2: Anwendung von Regel II:

$$l = \arg\max_{i \in \mathcal{B}} \left[-\frac{0}{4}, \frac{1}{10}, \frac{1}{16}, -\frac{1/2}{1}\right] = 4,$$

d.h. x_4 verlässt die Basis.

Schritt 3: Die Austausch-Algebra ergibt folgendes neues Tableau:

$$\begin{aligned}
\zeta &= 30 - \frac{5}{2}x_4 + 3x_1 \\
x_3 &= 04 + 0x_4 - 1x_1 \\
x_6 &= 10 - 1x_4 + 1x_1 \\
x_5 &= 06 + 1x_4 - 3x_1 \\
x_2 &= 06 - \frac{1}{2}x_4 - 0x_1
\end{aligned}$$

Iteration 4:

Schritt 1: Anwendung von Regel I: x_1 tritt in die Basis ein.

Schritt 2: Anwendung von Regel II:

$$l = \arg\max_{i \in \mathcal{B}} \left[\frac{1}{4}, -\frac{1}{10}, \frac{3}{6}, \frac{0}{6}\right] = 5,$$

d.h. x_5 verlässt die Basis.

Schritt 3: Die Austausch-Algebra ergibt folgendes neues Tableau:

$$\begin{aligned}
\zeta &= 36 - \frac{3}{2}x_4 - 1x_5 \\
x_3 &= 02 - \frac{1}{3}x_4 + \frac{1}{3}x_5 \\
x_6 &= 12 - \frac{2}{3}x_4 - \frac{1}{3}x_5 \\
x_1 &= 02 + \frac{1}{3}x_4 - \frac{1}{3}x_5 \\
x_2 &= 06 - \frac{1}{2}x_4 - 0x_5
\end{aligned}$$

Beide Koeffizienten der Zielfunktion sind negativ, d.h. eine weitere Verbesserung dieser kann nicht erreicht werden. Das Optimum $\zeta = 36$ ist erzielt bei $x_1 = 2$ und $x_2 = 6$.

Kapitel 4

Dualität

Zu jedem linearen Programm (P)

$$\begin{aligned}(\mathbf{P}) \quad \text{maximiere } & \sum_{j=1}^{n} c_j x_j, \\ \text{beschränkt durch } & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n,\end{aligned}$$

existiert ein duales Programm (D)

$$\begin{aligned}(\mathbf{D}) \quad \text{minimiere } & \sum_{i=1}^{m} b_i y_i, \\ \text{beschränkt durch } & \sum_{i=1}^{m} y_i a_{ij} \geq c_j, \quad j = 1, \ldots, n, \\ & y_i \geq 0, \quad i = 1, \ldots, m.\end{aligned} \tag{4.1}$$

Beispiel 4. (Motivation.)
Aufgabe der Produktionsanlage. Wenn man eine Einheit des Produkts j weniger produziert, dann werden a_{ij} Einheiten der zur Erzeugung dieses Produkts j notwendigen Rohmaterialien frei. Verkauft man diese nicht benötigten Rohmaterialien zu einem Preis von y_i Euro pro Einheit Rohmaterial, dann erzielt man einen Gewinn von $\sum_{i=1}^{m} y_i a_{ij}$ Euro für das Produkt j. An diesem Verkauf ist man nur dann interessiert, wenn dieser Gewinn mindestens so groß ist wie der Verlust an entgangenem Profit durch die Minderproduktion des Produkts j:

$$\sum_{i=1}^{m} y_i a_{ij} \geq c_j.$$

Ein Käufer des gesamten Inventars möchte seinen Einsatz minimieren:

$$\min \sum_{i=1}^{m} b_i y_i.$$

Dieser Interessenausgleich motiviert das duale Problem. Eine weiteres Motivationbeispiel (Aufsuchen oberer Schranken) gibt [44, Abschnitt 5.1]. □

Das duale Programm zu (D) ist wiederum das primale Programm (P). Der Beweis folgt aus den Definitionen und ist einfach [44, Kapitel 5].

Ebenfalls einfach zu beweisen ist der folgende Satz [44, Satz 5.1]

Satz 5. *Sind* x *und* y *zulässige Variablen des primalen bzw. dualen Programms, dann gilt*

$$c^T x \leq b^T y. \tag{4.2}$$

Tiefer ist für lineare Probleme das zweite Resultat [44, Satz 5.2]

Satz 6. *Sind* x^* *und* y^* *die Optimallösungen des primalen bzw. dualen Programms, dann gilt*

$$c^T x^* = b^T y^*. \tag{4.3}$$

Modifikationen des o.a. Simplexalgorithmus basieren auf der Idee der Dualität. Wir wollen diese hier nicht weiter verfolgen. Wir benötigen das Prinzip zur Herleitung der wichtigen Methode der inneren Punkte [44, Kapitel 16 ff.].

Kapitel 5

Nichtlineare Optimierung

In diesem Kapitel wird die Optimierung von nichtlinearen Funktionen behandelt. Im ersten Abschnitt betrachten wir Funktionen von Variablen, deren zulässige Menge unbeschränkt ist; im zweiten Abschnitt ist die zulässige Menge beschränkt.

5.1 Unbeschränkte Optimierung

Sei $f: \mathbb{R}^n \to \mathbb{R}^1$ eine zweimal stetig differenzierbare Funktion, $f \in C^2(\mathbb{R}^n)$, dann ist der Gradient von $f(x)$ der (Spalten-)Vektor der partiellen Ableitungen,

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T,$$

und die (symmetrische) Hesse-Matrix $H(x) = \nabla^2 f(x)$,

$$H(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_n^n} \end{pmatrix}.$$

Ein Punkt x^* mit $\nabla f(x^*) = 0$ wird stationärer Punkt genannt.

Die Optimierungsaufgabe lautet

$$\min_{x \in \mathbb{R}^n} f(x). \tag{5.1}$$

Aus der Analysis sind folgende notwendige und hinreichende Bedingungen bekannt:[40]

Satz 7. (Notwendige Bedingung erster Ordnung.)
Sei x^* ein lokales Minimum von $f \in C^1$, dann ist

$$\nabla f(x^*) = 0. \tag{5.2}$$

Satz 8. (Notwendige Bedingung zweiter Ordnung.)
Sei x^* ein lokales Minimum von $f \in C^2$, dann ist

$$\begin{aligned} \nabla f(x^*) &= 0, \\ z^T H(x^*) z &\geq 0, \quad \forall z \neq 0. \end{aligned} \tag{5.3}$$

Die zweite Aussage des Satzes 8 heißt, dass $H(x^*)$ positiv semidefinit ist.

Satz 9. (Hinreichende Bedingung zweiter Ordnung.)
Sei $f \in C^2$, $\nabla f(x^*) = 0$ *und* $z^T H(x^*) z > 0, \forall z \neq 0$, *dann ist* x^* *lokales Minimum von* f.

Satz 7 wird nun verwendet, um ein numerisches Lösungsverfahren für Problem (5.1) zu entwickeln, das Newton-Verfahren.

5.1.1 Newton-Verfahren

Zunächst wird $f(x)$ in eine Taylor-Reihe um $\bar{x}$ bis zum zweiten Glied entwickelt

$$\begin{aligned} f(x) \approx h(x) := f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}) \\ + \frac{1}{2}(x - \bar{x})^T H(\bar{x})(x - \bar{x}). \end{aligned} \tag{5.4}$$

Ein Kandidat für $\min\{h(x)\}$ ist nach Satz 7 aus

$$\nabla h(x) = \nabla f(\bar{x}) + H(\bar{x})(x - \bar{x}) = 0 \tag{5.5}$$

zu gewinnen. Gl.(5.5) wird nach $(x - \bar{x})$ aufgelöst

$$x - \bar{x} = -H(\bar{x})^{-1} \nabla f(\bar{x}), \tag{5.6}$$

was als Newton-Richtung bezeichnet wird, und folgendes Verfahren motiviert:[1]

Newton-Verfahren:

Schritt 0 Gegeben x^0, setze $k \leftarrow 0$,

Schritt 1 Berechne $f(x^k)$, $\nabla f(x^k)$, $H(x^k)$,

Schritt 2 $d^k = -H(x^k)^{-1} \nabla f(x^k)$,
falls $d^k = 0$ stop.

Schritt 3 Wähle Schrittweite $\alpha^k = 1$,

Schritt 4 Setze $x^{k+1} \leftarrow x^k + \alpha^k d^k$,
Setze $k \leftarrow k + 1$,
Wiederhole ab Schritt 1.

Das Newton-Verfahren hat folgende

Eigenschaften:

- Kleiner Konvergenzbereich,
- deshalb Dämpfung $\alpha \in (0, 1]$,
- Aufwand pro Iteration $O(n^3)$,
- quadratische Konvergenz.

Erstere bedeutet, dass der Startwert x^0 für die Iterationenfolge relativ nahe am (zunächst unbekannten) Minimum liegen muss, letztere Eigenschaft heißt, dass für eine Zahlenfolge $\{s_i\}$

[1] In Iterationsverfahren wird für Vektoren und Matrizen hier und im folgenden für den Iterationszähler ein oberer Index k gewählt; dies bedeutet keine Potenzbildung.

mit dem Grenzwert $\lim_{i\to\infty} s_i = \bar{s}$ folgender Grenzwert des Quotienten zweier aufeinanderfolgender Iterationen erreicht wird

$$\lim_{i\to\infty} \frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|^2} = \delta < \infty, \tag{5.7}$$

d.h. die Anzahl der genauen Stellen verdoppelt sich von Iteration zu Iteration. Diese Eigenschaften sollen an folgendem Beispiel veranschaulicht werden.

Beispiel 10.

Minimiere $f(x) = 7x - \ln(x)$ mit $\nabla f(x) = f'(x) = 7 - \frac{1}{x}$ und $H(x) = \frac{1}{x^2}$ und $x^* = \frac{1}{7} \approx 0.1428571485714$. Die Suchrichtung ist

$$d = -H(x)^{-1}\nabla f(x) = -\frac{f'(x)}{f''(x)} = x - 7x^2$$

und der Suchschritt

$$x^{k+1} = x^k + \alpha^k(x^k - 7(x^k)^2) = 2x^k - 7(x^k)^2,$$

welche einfach in `Matlab`[37] zu programmieren sind. Das Ergebnis ist für unterschiedliche Startwerte x^0 in folgender Tabelle zusammengestellt. Deutlich sind die hohe Konvergenzgeschwindigkeit in der Nähe des Optimums für die Startwerte 0.1 und 0.01 zu erkennen. Der Konvergenzbereich ist klein, da bereits die Startwerte 0 und 1 nicht mehr zum Optimum konvergieren. Die deswegen notwendige Dämpfung des Newton-Schritts wird in Paragraph 5.1.3 besprochen. □

k	x^k	x^k	x^k	x^k
0	0.0	0.01000000000000	0.10000000000000	1.00000000000000
1	0.0	0.01930000000000	0.13000000000000	-5.00000000000000
2	0.0	0.03599257000000	0.14170000000000	-185.00000000000000
3	0.0	0.06291688433357	0.14284777000000	-239945.00000000000000
4	0.0	0.09812402832743	0.14285714224219	-4.030157010650000e+011
5	0.0	0.12884978210844	0.14285714285714	-1.136951587135200e+024
6	0.0	0.14148369977113	0.14285714285714	-9.048612380424754e+048
7	0.0	0.14284393843577	0.14285714285714	-5.731417020782329e+098
8	0.0	0.14285714163665	0.14285714285714	-2.299439874627938e+198
9	0.0	0.14285714285714	0.14285714285714	$-\infty$
10	0.0	0.14285714285714	0.14285714285714	$-\infty$

Tabelle 5.1: Iterationsverlauf des Newton-Verfahrens

Der Aufwand des Newton-Verfahrens ist mit $O(n^3)$ Gleitkomma-Operationen pro Iteration in der Inversion der Hesse-Matrix hoch. Deshalb werden Quasi-Newton-Verfahren vorgeschlagen, die $H(x)^{-1}$ in $O(n^2)$ Operationen approximieren, s.a. Abschnitt 5.1.2.

Beispiel 11.

Das zweite Beispiel ist eine Ingeneur-Anwendung aus [1]. Der Gleichgewichtszustand einer mechanischen Struktur ist durch die stationären Punkte der gesamten potentiellen Energie des Systems charakterisiert. Ist der stationäre Punkt ein Minimum, dann ist das System im stabilen Gleichgewicht.

Gegeben sei eine Stabwerkstruktur mit zwei Stäben entsprechend Abb. 5.1. Am Verbindungsknoten C greift eine Last W an, die ihn in die Position C' verschiebt. Aufgabe ist es, die Verschiebungskoordinaten x und y zu ermitteln. Zur Lösung wird die gesamte potentielle Energie des Tragwerks in Abhängigkeit von x und y formuliert und diese minimiert. Wenn

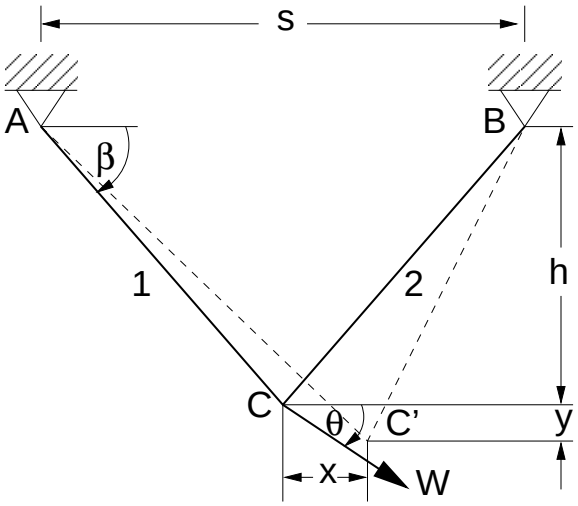


Abbildung 5.1: Stabwerkstruktur

die Verschiebungen ermittelt sind, dann lassen sich die Kräfte und Spannungen in den Stäben berechnen.

Bezeichnungen und Daten sind in folgender Tabelle zusmmengefasst.

E	Elastizitätsmodul (N/m^2)	207 GPa
s	Spannweite der Struktur (m)	1.5 m
h	Höhe der Struktur (m)	1.0 m
A_1	Querschnittsfläche des Stabes 1 (m^2)	1.0E-5 m^2
A_2	Querschnittsfläche des Stabes 2 (m^2)	1.0E-5 m^2
θ	Lastangriffswinkel °	30°
L	Stablänge (m)	$\sqrt{h^2 + s^2/4}$ m
W	Last (N)	10 kN
x	Horizontalverschiebung (m)	variabel
y	Vertikalverschiebung (m)	variabel

Die gesamte potentielle Energie unter der Annahme kleiner Verschiebungen ist

$$\begin{aligned} V(x,y) = &\frac{EA_1}{2L}(x\cos(\beta) + y\sin(\beta))^2 \\ &+ \frac{EA_2}{2L}(-x\cos(\beta) + y\sin(\beta))^2 \\ &- Wx\cos(\theta) - Wy\sin(\theta). \end{aligned}$$

Mit den o.a. Daten, den Beziehungen $\cos(\beta) = s/(2L)$ und $\sin(\beta) = h/L$ und der Annahme $A_1 = A_2 = A$ ist

$$V(x,y) = \frac{EA}{L}\left(\frac{s^2}{4L^2}x^2 + \frac{h^2}{L^2}y^2\right) - W\cos(\theta)x - W\sin(\theta)y,$$

eine quadratische Funktion in den Variablen x und y. Gemäß Satz 7 erhält man für quadratische Funktionen in mehreren Variablen ein lineares Gleichungssystem in ebendiesen:

$$2\frac{EA}{L}\begin{pmatrix} \frac{s^2}{4L^2} & 0 \\ 0 & \frac{h^2}{L^2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} W\cos(\theta) \\ W\sin(\theta) \end{pmatrix}.$$

Die Matrix in dieser Gleichung ist nicht-singulär, und man kann die Gleichung lösen und erhält mit den o.a. Zahlenwerten die Lösung $x = 7.263363893\text{mm}$ und $y = 2.358846618\text{mm}$.

Dieser Lösungsweg zeigt, dass unbeschränkte quadratische Optimierungsprobleme durch Auflösen eines linearen Gleichungssystems berechnet werden können. □

Übung: Geben Sie die Newton-Richtung für dieses Beispiel an und führen Sie das Newton-Verfahren durch.

Übung: Verifizieren Sie die Lösung des Beispiels in `Ampl`

```
#Minimierung der Verschiebungsenergie
var x;
var y;
param pi:=4*atan(1);
param h:=1;
param s:=3/2;
param A:=1e-5;
param E:=207e+9;
param W:=1e+4;
param Phi:=30*pi/180;
param L;
let L:=sqrt(h^2+s^2/4);

minimize V: E*A/L*(s^2/(4*L^2)*x^2+h^2/L^2*y^2)
          -W*(cos(Phi)*x+sin(Phi)*y);
option solver snopt;
solve;
display x, y, V;

dkraft@newton/home/dkraft/PROJECTS/ampl/SiOpt> ampl opt1.mod
SNOPT 7.2-1 : Optimal solution found.
2 iterations, objective -37.34840479
x = 0.00726336
y = 0.00235885
V = -37.3484
```

Übung: Verifizieren Sie die Lösung des Beispiels nun in `Maple`

```
> restart;
> with(linalg):
> M:=2*E*A/L*matrix([[s^2/4/L^2,0],[0,h^2/L^2]]);
```

$$M := 2\,EA \begin{bmatrix} 1/4\,\frac{s^2}{L^2} & 0 \\ 0 & \frac{h^2}{L^2} \end{bmatrix} L^{-1}$$

```
> b:=vector([W*cos(theta),W*sin(theta)]);
```

$$b := [W\cos(\theta), W\sin(\theta)]$$

```
> MI:=inverse(M);
```

$$MI := \begin{bmatrix} 2\,\frac{L^3}{EAs^2} & 0 \\ 0 & 1/2\,\frac{L^3}{EAh^2} \end{bmatrix}$$

```
> d:=multiply(MI,b);
```

$$d := [2\,\frac{L^3 W\cos(\theta)}{EAs^2}, 1/2\,\frac{L^3 W\sin(\theta)}{EAh^2}]$$

```
> e:=subs(L=sqrt(h^2+s^2/4),d[1],d[2]);
```

$$e := \left\{ 1/4\,\frac{\left(4h^2+s^2\right)^{3/2} W\cos(\theta)}{EAs^2}, 1/16\,\frac{\left(4h^2+s^2\right)^{3/2} W\sin(\theta)}{EAh^2} \right\}$$

```
> f:=subs(theta=Pi/6,h=1,s=3/2,A=1e-5,E=207e9,W=1e4,e[1],e[2]);
```

$$f := \left\{ 0.0008387010197\,\sqrt{25}\sqrt{4}\cos\left(1/6\,\pi\right), 0.0004717693236\,\sqrt{25}\sqrt{4}\sin\left(1/6\,\pi\right) \right\}$$

```
> evalf(f);
```

$$\{0.007263363893, 0.002358846618\}$$

5.1.2 Quasi-Newton-Verfahren

Der Aufwand des Newton-Verfahrens ist mit $O(n^3)$ Gleitkomma-Operationen pro Iteration in der Inversion der Hesse-Matrix hoch. Deshalb werden Quasi-Newton-Verfahren vorgeschlagen, die $H(x)^{-1}$ in $O(n^2)$ Operationen approximieren. Im Rahmen dieses Kurses wird nur der BFGS-Update[2] beschrieben, ausführlicher ist in Referenz [40] nachzulesen.

Die Suchrichtung des Newton-Verfahrens ist nach Gl.(5.6)

$$d^k = -H(x^k)^{-1}\nabla f(x^k).$$

In den Quasi-Newton-Verfahren wird die Hesse-Matrix $H(x^k)$ ausgehend von einer positiv definiten Schätzmatrix, z.B. der Einheitsmatrix, aus Informationen nullter und erster Ordnung als $B^k = B(x^k) \approx H(x^k)$ approximiert. Zunächst werden folgende Differenzvektoren gebildet:

$$s^k = x^{k+1} - x^k$$

und

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

und mit diesen

$$B^{k+1} = B^k - \frac{B^k s^k s^{kT} B^k}{s^{kT} B^k s^k} + \frac{y^k y^{kT}}{y^{kT} s^k} \tag{5.8}$$

durch Addition zweier Rank-Eins-Matrizen erneuert. Eine naive Berechnung von $d^k = d(x^k)$ mittels

$$B^k d^k = -\nabla f^k$$

würde wieder auf $O(n^3)$ Operationen führen, weswegen der Update der Cholesky-Faktorisierung

$$B^k = L^k D^k L^{kT}$$

gemäß Gl.(5.8) durchgeführt wird, und die Auflösung von

$$L^k D^k L^{kT} d^k = -\nabla f^k \tag{5.9}$$

erfordert nur $O(n^2)$ Gleitkomma-Operationen, s.a. [40, S. 201].

Das BFGS-Verfahren hat ähnliche Eigenschaften wie das Newton-Verfahren:

Eigenschaften:

- Kleiner Konvergenzbereich,
- deshalb Dämpfung $\alpha \in (0, 1]$,
- Aufwand pro Iteration $O(n^2)$,
- superlineare Konvergenz,

letztere bedeutet

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^*|}{|x_i - x^*|} = 0. \tag{5.10}$$

[2] Benannt nach den Autoren Broyden, Fletcher, Goldfarb und Shanno.

5.1.3 Eindimensionale Minimumsuche

Sowohl das Newton-Verfahren als auch die Quasi-Newton-Verfahren haben einen kleinen Konvergenzbereich, weswegen die Suchschritte d^k durch einen Faktor $\alpha^k \in (0, 1]$ gedämpft werden muss:

$$x^{k+1} = x^k + \alpha^k d^k. \tag{5.11}$$

Die Bestimmung von α^k geschieht über die angenäherte Optimierung einer Merit-Funktion, die von nur einer Variablen α abhängt. In der unbeschränkten Optimierung ist eine geeignete Merit-Funktion die zu minimierende Funktion selbst:

$$\Phi(\alpha) = f(x^k + \alpha d^k), \tag{5.12}$$

mit festem x^k und d^k, und das Problem ist

$$\alpha^k = \min_{\alpha>0} \Phi(\alpha). \tag{5.13}$$

Diese Minimierung muss nicht exakt durchgeführt werden; es muss nur die sogenannte Armijo-Bedingung erfüllt sein:

$$f(x^k + \alpha d^k) \leq f(x^k) + c\alpha \nabla f^{kT} d^k, \tag{5.14}$$

mit einer Konstanten $0 < c \ll 1$. Der Sachverhalt ist in Abb. 5.2 skizziert. Die Funktion Φ

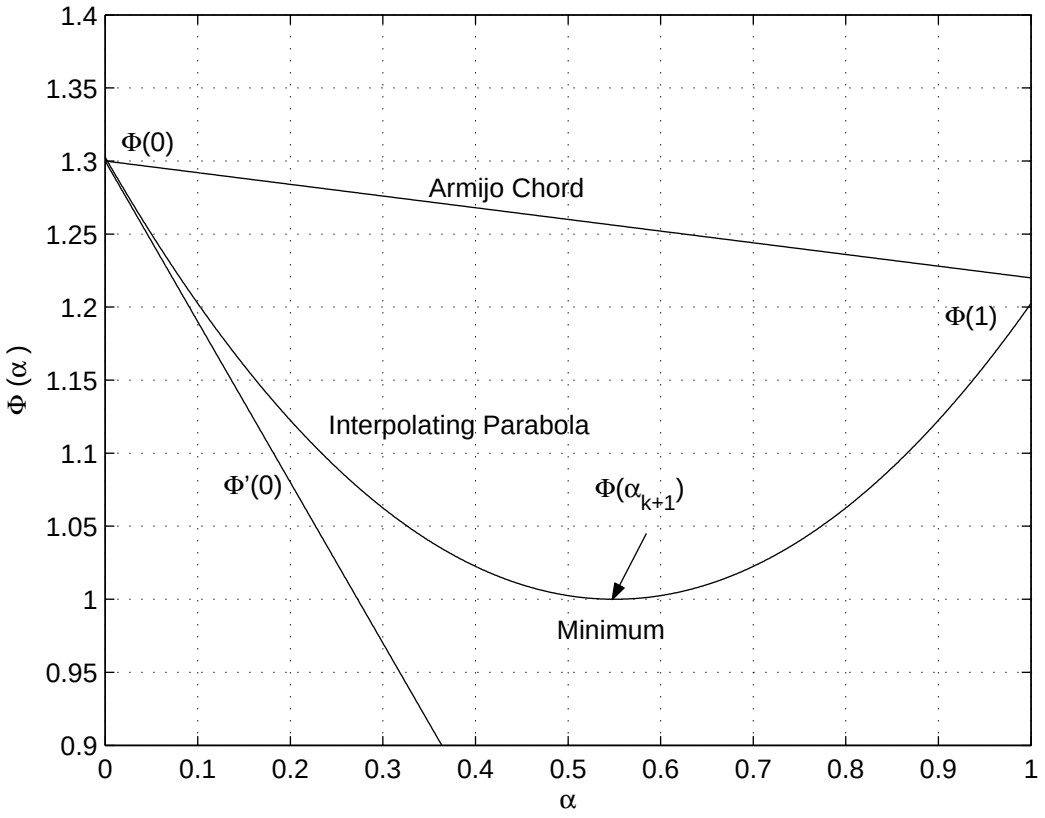


Abbildung 5.2: Eindimensionale Minimumsuche

wird mit Hilfe der drei Werte $\Phi(0)$, $\Phi'(0) = \nabla f^{kT} d^k$ und $\Phi(\alpha_0 = 1)$ als Parabel approximiert, deren Minimum,

$$\alpha_1 = -\frac{\Phi'(0)\alpha_0^2}{2(\Phi(\alpha_0) - \Phi(0) - \Phi'(0)\alpha_0)},$$

leicht bestimmt werden kann. Erfüllt α_1 die Armijo-Bedingung nicht, dann wird die Interpolation mit α_1 statt α_0 wiederholt.

Quasi-Newton-Verfahren mit Schrittweitensteuerung:

Schritt 0 Gegeben x^0, B^0, setze $k \leftarrow 0$,

Schritt 1 Berechne $f(x^k)$, $\nabla f(x^k)$,

Schritt 2 Berechne $B^k d^k = -\nabla f(x^k)$,
falls $d^k = 0$ stop.

Schritt 3 Wähle Schrittweite α^k
so dass $f(x^k + \alpha d^k) \leq f(x^k) + c\alpha \nabla f^{kT} d^k$,

Schritt Setze $x^{k+1} \leftarrow x^k + \alpha^k d^k$, berechne s^k und y^k,
Berechne B^{k+1} aus B^k mittels quasi-Newton Update (5.8),
Setze $k \leftarrow k + 1$,
Wiederhole ab Schritt 1.

5.2 Beschränkte Optimierung

Der Bereich der zulässigen Variablen, der bisher der gesamte $\mathbb{R}^n$ war wird nun durch nichtlineare Gleichungen und Ungleichungen beschränkt. Das Problem lautet jetzt

$$\min_{x \in \mathbb{R}^n} f(x) \tag{5.15a}$$

$$\text{s.t. } c_i(x) = 0, \quad i = 1, \ldots, m_{eq} \tag{5.15b}$$

$$c_i(x) \geq 0, \quad i = m_{eq} + 1, \ldots, m. \tag{5.15c}$$

Es werden zwei Indexmengen definiert: $\mathcal{E} = \{1, 2, \ldots, m_{eq}\}$ und $\mathcal{I} = \{m_{eq} + 1, \ldots, m\}$, dazu mit der Indexmenge der aktiven Ungleichungsbeschränkungen

$$\mathcal{J}(\bar{x}) = \{i \in \mathcal{I} | c_i(\bar{x}) = 0\} \tag{5.16}$$

die Indexmenge der aktiven Beschränkungen

$$\mathcal{A}(\bar{x}) = \mathcal{E} \cup \mathcal{J}(\bar{x}) \tag{5.17}$$

Die zulässige Menge Ω ist die Menge aller Punkte x, welche die Gleichungs- und Ungleichungsbeschränkungen in (5.15) erfüllen:

$$\Omega = \{x | c_i(x) = 0,\ i \in \mathcal{E};\ x | c_i(x) \geq 0,\ i \in \mathcal{I};\} \tag{5.18}$$

5.2.1 Charakterisierung der Lösung

Die notwendigen Bedingungen für das gleichungsbeschränkte Problem waren schon Lagrange[3] bekannt. Es dauerte fast 200 Jahre bis sie für das ungleichungsbeschränkte Problem formuliert wurden. Zunächst ist eine Bedingung an die Gradienten der Nebenbedingungen zu fordern:

Definition 12. (Constraint Qualification.)
An einem Punkt $\bar{x}$ ist die constraint qualification erfüllt, wenn die Gradienten der aktiven Nebenbedingungen $\{\nabla c_i(\bar{x}), i \in \mathcal{A}(\bar{x})\}$ linear unabhängig sind.

[3] Joseph Louis Lagrange (* 25. Januar 1736 in Turin; † 10. April 1813 in Paris) war ein italienischer Mathematiker und Astronom.

Merke: Aufgrund dieser Bedingung kann keiner der Gradienten der aktiven Nebenbedingungen Null sein.

Nun werden die Nebenbedingungen mittels Lagrange-Multiplikatoren λ an die Zielfunktion angekoppelt in der Lagrange-Funktion

$$\mathcal{L}(x,\lambda) = f(x) - \sum_{i\in\mathcal{E}\cup\mathcal{I}} \lambda_i c_i. \tag{5.19}$$

Satz 13. (Notwendige Bedingungen erster Ordnung von Karush, Kuhn und Tucker.)
Sei x^* eine lokale Lösung von (5.15) *und die constraint qualification 12 sei für x^* gewährleistet. Dann existiert ein Vektor λ^*, mit den Komponenten $\lambda_i^*, i \in \mathcal{E}\cup\mathcal{I}$, so dass die folgenden Bedingungen für (x^*,λ^*) gelten*

$$\nabla_x \mathcal{L}(x^*,\lambda^*) = 0, \tag{5.20a}$$
$$c_i(x^*) = 0, \quad \forall i \in \mathcal{E}, \tag{5.20b}$$
$$c_i(x^*) \geq 0, \quad \forall i \in \mathcal{I}, \tag{5.20c}$$
$$\lambda_i^* \geq 0, \quad \forall i \in \mathcal{I}, \tag{5.20d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \forall i \in \mathcal{E}\cup\mathcal{I}. \tag{5.20e}$$

Die Bedingung (5.20e) wird als Komplementaritätsbedingung bezeichnet. Sie bedingt, dass die Multiplikatoren der inaktiven Ungleichungsnebenbedingungen $\{i|c_i(x^*) > 0\}$ Null sind. Folgender Spezialfall wird gesondert bezeichnet.

Definition 14. (Strikte Komplementarität.)
Für ein lokales Minimum x^* von (5.15) und einen Vektor λ^*, der Satz 13 genügt, heißt das Paar (x^*,λ^*) strikt komplementär, wenn genau einer der beiden Werte λ_i^* und $c_i(x^*)$ Null ist für alle $i \in \mathcal{I}$.

Dies besagt, dass $\lambda_i^* > 0,\ \forall\ \mathcal{I} \cap \mathcal{A}(x^*)$, gilt, oder dass es keinen Index $i \in \mathcal{I}$ gibt, so dass $\lambda_i^* = c_i(x^*) = 0$ gilt.

Beispiel 15.
Das erste Beispiel hat 2 Variable, eine quadratische Zielfunktion und eine lineare Ungleichungsbeschränkung.

$$\begin{aligned} &\min\ (x_1-1)^2 + (x_2-1)^2 \\ &\text{s.t.}\ -x_1 - x_2 \geq 0 \end{aligned}$$

Die Lagrangefunktion lautet $\mathcal{L}(x_1,x_2,\lambda) = (x_1-1)^2 + (x_2-1)^2 + \lambda(x_1+x_2)$. Die Bedingung (5.20e) unterscheidet zwei Fälle:[4]

1) $\lambda^* = 0$
wofür die Bedingung (5.20) das (lineare) Gleichungssystem:

$$\begin{aligned} 2(x_1-1) &= 0, \\ 2(x_2-1) &= 0, \end{aligned}$$

mit der Lösung $(x_1 = 1,\ x_2 = 1)$ liefert, welche aber Bedingung (5.20c), $c(x^*) = -2 \not\geq 0$, verletzt.

2) $c(x^*) = 0$
welcher zusammen mit Bedingung (5.20) wieder ein lineares Gleichungssystem liefert:

$$\begin{aligned} 2(x_1-1) + \lambda &= 0, \\ 2(x_2-1) + \lambda &= 0, \\ -x_1 - x_2 &= 0, \end{aligned}$$

[4] Die Komplementaritätsbedingung wird sozusagen als Schaltbedingung verwendet.

mit der Lösung ($x_1 = 0,\ x_2 = 0,\ \lambda = 2$), welche (5.20d) erfüllt und somit die korrekte Lösung ist. □

Beispiel 16.

Das zweite Beispiel hat 2 Variable, eine lineare Zielfunktion und eine quadratische Gleichungsbeschränkung.

$$\begin{aligned} &\min\ x_1 + x_2 \\ &\text{s.t. } x_1^2 + x_2^2 - 2 = 0 \end{aligned}$$

Die Lagrangefunktion lautet $\mathcal{L}(x_1, x_2, \lambda) = x_1 + x_2 - \lambda(x_1^2 + x_2^2 - 2)$. Die Bedingungen (5.20a) und (5.20e) etablieren folgendes (nichtlineare) Gleichungssystem:

$$\begin{aligned} 1 - 2\lambda x_1 &= 0, \\ 1 - 2\lambda x_2 &= 0, \\ (x_1^2 + x_2^2 - 2) &= 0, \end{aligned}$$

mit den beiden Lösungen ($x_1 = 1,\ x_2 = 1,\ \lambda = \frac{1}{2}$) und ($x_1 = -1,\ x_2 = -1,\ \lambda = -\frac{1}{2}$). Welche der beiden das Minimum ist, kann systematisch erst durch Auswertung der Bedingungen zweiter Ordnung entschieden werden; hier „sieht“ man, dass das zweite Lösungstripel das gesuchte ist. (Merke: Die Multiplikatoren von Gleichungsnebenbedingungen haben keine Vorzeichenbeschränkung.) □

Übung: Lösen Sie Beispiel 2 aus Paragraph 5.1.1 unter der Beschränkung, dass $x \leq x_{max} = 3mm$.

Beispiel 17.

Hier folgt ein Maple-worksheet, in dem ein konvexes quadratisches Programm mit zwei Ungleichungsnebenbedingungen gelöst wird:

$$\begin{aligned} &\min\ \frac{1}{2} x^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} x + \begin{pmatrix} 3 & 1 \end{pmatrix} x \\ &\text{s.t. } \begin{pmatrix} -1 & -1 \\ 1 & 2 \end{pmatrix} x - \begin{pmatrix} 1 \\ 3 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Maple-Worksheet vom 02.11.2005

Loesung der Karush-Kuhn-Tucker-Bedingungen fuer ein konvexes quadratisches Programm

min (1/2)x'Qx+c'x

subject to Ax-b>=0

```
> restart;
> with(linalg):

Warning, the protected names norm and trace have been redefined and
unprotected
```

Hesse-Matrix der quadratischen Zielfunktion

```
> Q:=matrix([[2,1],[1,2]]);
```

$$Q := \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

```
> eigenvalues(Q);
```

$$3,\ 1$$

Eigenwerte sind positiv => Q ist positiv definit

Vektor des Linearterms der Zielfunktion

```
> c:=vector([1,2]);
```

$$c := [1, 2]$$

Matrix der linearen Nebenbedingungen

```
> A:=matrix([[-1,-1],[1,2]]);
```

$$A := \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix}$$

Vektor der linearen Nebenbedingungen

```
> b:=vector([1,3]);
```

$$b := [1, 3]$$

Vektor der Variablen

```
> x:=vector([x1,x2]);
```

$$x := [x1, x2]$$

Quadratische Kostenfunktion

```
> f:=1/2*multiply(x,Q,x)+multiply(c,x);
```

$$f := \frac{(2\,x1 + x2)\,x1}{2} + \frac{(x1 + 2\,x2)\,x2}{2} + x1 + 2\,x2$$

```
> f:=simplify(f);
```

$$f := x1^2 + x1\,x2 + x2^2 + x1 + 2\,x2$$

Gradient der Kostenfunktion

```
> g1:=diff(f,x1);
```

$$g1 := 2\,x1 + x2 + 1$$

```
> g2:=diff(f,x2);
```

$$g2 := x1 + 2\,x2 + 2$$

Unbeschraenktes Minimum

```
> s:=solve({g1,g2});
```

$$s := \{x1 = 0,\ x2 = -1\}$$

```
> lambda:=vector([l1,l2]);
```

$$\lambda := [l1, l2]$$

```
> cc:=matadd(multiply(A,x),-b);
```

$$cc := [-x1 - x2 - 1,\ x1 + 2\,x2 - 3]$$

Lineare Nebenbedingungen

```
> c1:=cc[1];
```

$$c1 := -x1 - x2 - 1$$

```
> c2:=cc[2];
```

$$c2 := x1 + 2\,x2 - 3$$

Lagrange-Funktion

```
> L:=f-multiply(lambda,cc);
```

$$L := x1^2 + x1\,x2 + x2^2 + x1 + 2\,x2 - l1\,(-x1 - x2 - 1) - l2\,(x1 + 2\,x2 - 3)$$

Gradient der Lagrange-Funktion

```
> dL1:=diff(L,x1);
```

$$dL1 := 2\,x1 + x2 + 1 + l1 - l2$$

```
> dL2:=diff(L,x2);
```

$$dL2 := x1 + 2\,x2 + 2 + l1 - 2\,l2$$

Fall 1) c1=0, c2=0

```
> s1:=solve({dL1,dL2,c1,c2});
```

$$s1 := \{x2 = 4,\ l2 = 10,\ l1 = 15,\ x1 = -5\}$$

```
> f1:=subs(s1,f);
```

$$f1 := 24$$

Dies ist bereits (wg. der Konvexitaet) die Loesung, da beide Lambda > Null

Fall 2) Lambda1=Lambda2=0 (das unbeschraenkte Minimum)

```
> dL1h:=subs(l1=0,l2=0,dL1);
```

$$dL1h := 2\,x1 + x2 + 1$$

```
> dL2h:=subs(l1=0,l2=0,dL2);
```

$$dL2h := x1 + 2\,x2 + 2$$

```
> s2:=solve({dL1h,dL2h});
```

$$s2 := \{x1 = 0,\ x2 = -1\}$$

```
> c12:=subs(s2,c1);
```

$$c12 := 0$$

```
> c22:=subs(s2,c2);
```

$$c22 := -5$$

```
> f2:=subs(s2,f);
```

$$f2 := -1$$

Unzulaessig, da c2 < 0

Fall 3) Lambda1=0, c2=0

```
> dL1hh:=subs(l1=0,dL1);
```

$$dL1hh := 2\,x1 + x2 + 1 - l2$$

```
> dL2hh:=subs(l1=0,dL2);
```

$$dL2hh := x1 + 2\,x2 + 2 - 2\,l2$$

```
> s3:=solve({dL1hh,dL2hh,c2});
```

$$s3 := \{x1 = 0,\ x2 = \frac{3}{2},\ l2 = \frac{5}{2}\}$$

```
> c13:=subs(s3,c1);
```

$$c13 := \frac{-5}{2}$$

```
> f3:=subs(s3,f);
```

$$f3 := \frac{21}{4}$$

Unzulaessig, da c1 < 0

Fall 4) Lambda2=0, c1=0

```
> dL1hg:=subs(l2=0,dL1);
```

$$dL1hg := 2\,x1 + x2 + 1 + l1$$

```
> dL2hg:=subs(l2=0,dL2);
```

$$dL2hg := x1 + 2\,x2 + 2 + l1$$

```
> s4:=solve({dL1hg,dL2hg,c1});
```

$$s4 := \{x1 = 0,\ x2 = -1,\ l1 = 0\}$$

```
> c14:=subs(s4,c2);
```

$$c14 := -5$$

```
> f4:=subs(s4,f);
```

$$f4 := -1$$

Unzulaessig, da $c1 < 0$

Für die notwendigen und hinreichenden Bedingungen zweiter Ordnung müssen Vektoren w definiert werden, die in der Tangentialebene der aktiven Nebenbedingungen an einem lokalen Minimum x^* liegen, und für die ein λ^* die KKT-Bedingungen[5] erfüllt.

Definition 18. (Tangentialvektoren.)
Vektoren w, die die Bedingungen

$$w \in \Psi(x^*, \lambda^*) \Leftrightarrow \begin{cases} \nabla c_i(x^*)^T w = 0, & \forall i \in \mathcal{E}, \\ \nabla c_i(x^*)^T w = 0, & \forall i \in \mathcal{J}, \quad \text{with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^T w \geq 0, & \forall i \in \mathcal{J}, \quad \text{with } \lambda_i^* = 0, \end{cases} \tag{5.21}$$

erfüllen, und damit in einer Tangentialebene der aktiven Nebenbedingungen liegen, werden kurz als Tangentialvektoren bezeichnet.

Mit den Vektoren der Definition 18 können die Charakterisierungsbedingungen zweiter Ordnung beschrieben werden.

Satz 19. (Notwendige Bedingungen zweiter Ordnung.)
Sei x^* eine lokale Lösung von (5.15) *mit Lagrange-Multiplikatoren λ^*, die die KKT-Bedingungen erfüllen, und es sei die Constraint Qualification nach Definition 12 erfüllt, dann gilt für alle $w \in \Psi(x^*, \lambda^*)$*

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0. \tag{5.22}$$

Der folgende Satz qualifiziert die Lösung des Problems (5.15).

Satz 20. (Hinreichende Bedingungen zweiter Ordnung.)
Sei $x^* \in \Omega(x^*)$ ein zulässiger Punkt im Problem (5.15) *mit Lagrange-Multiplikatoren λ^*, die die KKT-Bedingungen erfüllen, und es sei die Constraint Qualification nach Definition 12 erfüllt, dann ist unter der Voraussetzung*

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w > 0, \quad \forall w \in \Psi(x^*, \lambda^*),\ w \neq 0, \tag{5.23}$$

x^* lokale Lösung des Problems (5.15).

Beispiel 21. Vorgelegt sei folgendes Problem

$$\begin{aligned} \min\ & f := x_1^3 - 16x_1 - 3x_2^2 + 2x_2 \\ \text{s.t. } & c := 3 - x_1 - x_2 \geq 0 \end{aligned}$$

Die Lagrange-Funktion ist

$$L(x_1, x_2, \lambda) = x_1^3 - 16x_1 - 3x_2^2 + 2x_2 - \lambda(3 - x_1 - x_2),$$

[5] Die notwendigen Bedingungen erster Ordnung des Satzes 13 werden im folgenden als KKT-Bedingungen bezeichnet.

deren Gradient und Hesse-Matrix sind

$$\nabla_x L = \begin{pmatrix} 3x_1^2 + \lambda - 16 \\ -6x_2 + \lambda + 2 \end{pmatrix} \quad \text{und} \quad H = \begin{pmatrix} 6x_1 & 0 \\ 0 & -6 \end{pmatrix}.$$

Die KKT-Bedingungen (5.20) unterscheiden zwischen den beiden Fällen ($\lambda = 0, c > 0$) und ($\lambda > 0, c = 0$). Der erste Fall führt zu

$$x = \begin{pmatrix} \pm\frac{4}{3}\sqrt{3} \\ \frac{1}{3} \end{pmatrix},$$

der zweite auf

$$\begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 16 \end{pmatrix}.$$

Wir untersuchen die Bedingungen von Satz 20 Punkt für Punkt.
1) Für $x = (\frac{4}{3}\sqrt{3}, \frac{1}{3})^T$ ist H indefinit, und da $c(x) > 0$ ist, handelt es sich um einen Sattelpunkt der Kostenfunktion, also kein lokales Minimum.
2) Für $x = (-\frac{4}{3}\sqrt{3}, \frac{1}{3})^T$ ist H negativ definit, und da $c(x) > 0$ ist, handelt es sich um ein lokales Maximum der Kostenfunktion, also kein lokales Minimum.
3) Für $x = (2,1)^T$ ist $H = \begin{pmatrix} 12 & 0 \\ 0 & -6 \end{pmatrix}$ indefinit und $c(x) = 0$. Wir suchen Richtungen $w \neq 0$, die in der Tangentialebene von c liegen, also orthogonal zu $\nabla c = (-1,-1)^T$ stehen, d.h. die $w^T \nabla c = 0$ erfüllen. Offenbar ist $w = \kappa(1,-1)^T$, $\kappa \neq 0$, eine solche, und $w^T H w = 6\kappa^2 > 0$ erfüllt die hinreichende Bedingung (5.23), also ist x lokales Minimum.
4) Für $x = (0,3)^T$ ist H negativ semidefinit, erfüllt damit nicht die notwendigen Bedingungen zweiter Ordnung, also ist x kein lokales Minimum. □

5.2.2 Ingenieur-Anwendungen

Tank-Entwurf.

Ein zylindrischer Tank ist definiert durch seine Höhe H und seinen Radius R. Beide Variablen bestimmen das Volumen V und die Oberfläche A. Letztere bestimmt die Fabrikationskosten, ersteres ist Entwurfsvorgabe. Die Variablen H und R sind durch die Gegebenheiten des Aufstellungsortes beschränkt.

Damit ergibt sich folgendes Optimierungsproblem:

$$\begin{aligned} \min\ & 2\pi(R^2 + HR) \\ \text{s.t. } & \pi H R^2 - V = 0 \\ & H_{min} \leq H \leq H_{max} \\ & R_{min} \leq R \leq R_{max} \end{aligned}$$

Struktur-Entwurf.

Ein zylindrischer Hohlstab der Länge L ist einseitig fest eingespannt und steht unter einer axial wirkenden Schublast P. Entwurfsvariablen sind der Außenradius R und der Innenradius r. Entwurfsziel ist minimales Gewicht unter Einhaltung der Knickstabilität und einer vorgeschriebenen Spannung σ. Die Knicklast für die o.a. Konfiguration ist $\pi^2 EI/(4L^2)$ mit

dem Elastizitätsmodul E und dem dem Flächenträgheitsmoment $I = \frac{\pi}{4}(R^4 - r^4)$. Die Querschnittsfläche des Stabes ist $A = \pi(R^2 - r^2)$, seine Masse $m = \rho AL$.

Damit ergibt sich folgendes Optimierungsproblem:

$$\begin{aligned} \min\ & \pi\rho L(R^2 - r^2) \\ \text{s.t. } & \sigma - \frac{P}{\pi(R^2 - r^2)} \geq 0 \\ & \frac{\pi^3 E}{16L^2}(R^4 - r^2) - P \geq 0 \\ & R_{\min} \leq R \leq R_{\max} \\ & r_{\min} \leq r \leq r_{\max} \end{aligned}$$

Wenn noch eine Forderung an eine dünne Wandstärke vorgegeben wird, kann diese folgendermaßen zu den Beschränkungen hinzugefügt werden:

$$\frac{R + r}{R - r} - \kappa \geq 0,$$

mit einer gegebenen Konstanten $\kappa \geq 20$.

Übung: Formulieren Sie das Problem unter Verwendung der Entwurfsvariablen mittlerer Radius R und Wandstärke t.

5.2.3 Sequentielle quadratische Programmierung

Analog zum Newton-Verfahren zur Optimierung einer unbeschränkten Funktion unter Verwendung der notwendigen Bedingungen erster Ordnung des Satzes 7 soll jetzt ein numerisches Verfahren zur beschränkten Optimierung durch Lösung der KKT-Bedingungen des Satzes 13 entworfen werden. Es sind zwei Betrachtungsweisen möglich: a) Die direkte Lösung von (5.20) und b) die indirekte Lösung über ein quadratisches Programm.

Direkte Lösung der KKT-Gleichungen

Um die Beschreibung nicht zu kompliziert zu gestalten, wird zunächst das gleichungsbeschränkte Problem

$$\min\ f(x) \tag{5.24a}$$
$$\text{s.t. } c(x) = 0 \tag{5.24b}$$

betrachtet.[6] Die KKT-Bedingungen fordern, dass der Gradient der Lagrange-Funktion $\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x)$ bezüglich x Null wird. Unter Verwendung der Jacobi-Matrix $A(x)$ der Nebenbedingungen

$$A(x)^T = (\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_m(x)) \tag{5.25}$$

kann die KKT-Bedingung als ein System von $n + m$ Gleichungen in $n + m$ Unbekannten x und λ geschrieben werden:

$$F(x, \lambda) = \begin{pmatrix} \nabla f(x) - A(x)^T \lambda \\ c(x) \end{pmatrix} = 0 \tag{5.26}$$

[6] In diesem Abschnitt schreiben wir kompakt die Vektoren, z.B. $c : \mathbb{R}^n \to \mathbb{R}^m$ an Stelle ihrer Komponenten $c_i(x),\ i \in \mathcal{E} = \{1, \ldots, m\}$.

notiert werden. Wenn $A(x^*)$ vollen (Zeilen-)Rang hat, was wegen der Constraint Qualification der Definition 12 der Fall ist, dann erfüllt jede Lösung (x^*, λ^*) des Problems (5.24) die Gleichung $F(x^*, \lambda^*) = 0$.

Zur iterativen Lösung des (i.a. nichtlinearen) Gleichungssystems (5.26) wird das Newton-Verfahren verwendet [40, Kap. 11], dessen Suchrichtung $(d_x, d_\lambda)^T$ folgendermaßen bestimmt wird

$$\nabla F(x, \lambda) \begin{pmatrix} d_x \\ d_\lambda \end{pmatrix} = -F(x, \lambda), \tag{5.27}$$

mit der Jacobi-Matrix

$$\nabla F(x, \lambda) = \begin{pmatrix} W(x, \lambda) & -A(x)^T \\ A(x) & 0 \end{pmatrix} \tag{5.28}$$

und $W(x, \lambda) = \nabla^2_{xx} \mathcal{L}(x, \lambda)$ der Hesse-Matrix der Lagrange-Funktion. Diese ist positiv definit in der Tangentenebene der Nebenbedingungen $w^T W w > 0, \ \forall w \neq 0$, mit der Eigenschaft $A(x)w = 0$, s.a. Satz 20. Unter dieser Voraussetzung und der der Constraint Qualification ist $\nabla F(x, \lambda)$ nichtsingulär, und das Newton-Verfahren zur Lösung von (5.24) kann nach Auflösung von

$$\begin{pmatrix} W(x^k, \lambda^k) & -A(x^k)^T \\ A(x^k) & 0 \end{pmatrix} \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) + A(x^k)^T \lambda^k \\ -c(x^k) \end{pmatrix} \tag{5.29}$$

wie folgt iteriert werden:

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix} + \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix}. \tag{5.30}$$

Quadratisches Programm

Zu den Gleichungen (5.29) und (5.30) gelangt man ebenfalls über das folgende quadratische Optimierungsproblem

$$\min \frac{1}{2} d^T W(x^k, \lambda^k) d + \nabla f(x^k)^T d \tag{5.31a}$$

$$\text{s.t. } A(x^k) d + c(x^k) = 0. \tag{5.31b}$$

Die notwendigen Bedingungen zur Lösung dieses Problems sind mit den Lagrange-Multiplikatoren μ^k

$$\begin{aligned} W(x^k, \lambda^k) d^k + \nabla f(x^k) - A(x^k)^T \mu^k &= 0, \\ A(x^k) d^k + c(x^k) &= 0 \end{aligned}$$

oder kompakter

$$\begin{pmatrix} W(x^k, \lambda^k) & -A(x^k)^T \\ A(x^k) & 0 \end{pmatrix} \begin{pmatrix} d^k \\ \mu^k \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) \\ -c(x^k) \end{pmatrix}. \tag{5.33}$$

Subtrahiert man von der ersten Zeile der Gl. (5.29) $A(x^k)^T \lambda^k$, dann erhält man

$$\begin{pmatrix} W(x^k, \lambda^k) & -A(x^k)^T \\ A(x^k) & 0 \end{pmatrix} \begin{pmatrix} d_x^k \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) \\ -c(x^k) \end{pmatrix}, \tag{5.34}$$

und man erkennt die Äquivalenz des Newton-Verfahrens mit dem quadratischen Programm, wenn man

$$d_x^k = d^k \quad \text{und} \quad \lambda^{k+1} = \mu^k$$

setzt. Der indirekte Zugang über das quadratische Programm wird bevorzugt weil es ein Anzahl vorzüglicher Implementierungen, z.B. [22] oder [45], gibt und weil die Einführung von Ungleichungen in quadratische Programme einfacher ist.

Hier nun also das allgemeine Problem

$$\min f(x) \tag{5.35a}$$
$$\text{s.t. } c_i(x) = 0, \quad i \in \mathcal{E}, \tag{5.35b}$$
$$c_i(x) \geq 0, \quad i \in \mathcal{I}, \tag{5.35c}$$

für welches die Suchrichtung d^k und die Lagrange-Multiplikatoren λ^{k+1} der Lösung des quadratischen Programms

$$\min \frac{1}{2} d^T W(x^k, \lambda^k) d + \nabla f(x^k)^T d \tag{5.36a}$$
$$\text{s.t. } \nabla c_i(x^k) d + c_i(x^k) = 0, \quad i \in \mathcal{E}, \tag{5.36b}$$
$$\nabla c_i(x^k) d + c_i(x^k) \geq 0, \quad i \in \mathcal{I}, \tag{5.36c}$$

verwendet werden.

Folgender Modell-Algorithmus charakterisiert die

Sequentielle quadratische Programmierung:

Schritt 0 Gegeben x^0, λ^0, setze $k \leftarrow 0$,

Schritt 1 Berechne $f(x^k),\ c(x^k),\ \nabla f(x^k),\ A(x^k),\ W(x^k, \lambda^k)$,

Schritt 2 Löse (5.36) und erhalte d^k und μ^k,
falls $d^k = 0$ stop.

Schritt 3 Wähle Schrittweite $\alpha^k = 1$,

Schritt 4 Setze $x^{k+1} \leftarrow x^k + \alpha^k d^k$ und $\lambda^{k+1} = \mu^k$,
Setze $k \leftarrow k + 1$,
Wiederhole ab Schritt 1.

Quasi-Newton-Approximation

In Ingenieuranwendungen stehen häufig zweite Ableitungen der Funktionen f und c zur Bildung der Hesse-Matrix der Lagrange-Funktion W nicht zur Verfügung, weswegen — wie bereits im Falle der unbeschränkten Optimierung 5.1.2 — Quasi-Newton-Approximationen von W üblich und effizient sind. Wieder werden Vektoren von Differenzen zweier aufeinanderfolgender Iterationen gebildet:

$$s^k = x^{k+1} - x^k$$

und

$$y^k = \nabla \mathcal{L}^{k+1} - \nabla \mathcal{L}^k = \nabla \mathcal{L}(x^{k+1}, \lambda^{k+1}) - \nabla \mathcal{L}(x^k, \lambda^{k+1}).$$

Mit diesen Vektoren könnte man einen quasi-Newton Update als Approximation von $W = \nabla^2_{xx} \mathcal{L}$ wie in (5.8) durchführen. Es sind aber nicht immer alle Eigenwerte von $\nabla^2_{xx} \mathcal{L}$ positiv, und dann ist die Approximation durch eine positiv definite Matrix nicht angemessen. Ausserdem ist die Forderung an die Abstiegseigenschaft $s^{kT} y^k$ in diesem Fall nicht notwendigerweise erfüllt. Deshalb wird y^k nach einer Idee von Powell [41] folgendermaßen modifiziert, wenn $s^{kT} y^k < \theta^k s^{kT} B^k s^k$ ist:

$$\bar{y}^k = \theta^k y^k + (1 - \theta^k) B^k s^k$$

mit

$$\theta^k = \begin{cases} 1 & \text{wenn } s^{kT} y^k \geq \theta s^{kT} B^k s^k, \\ \dfrac{(1-\theta) s^{kT} B^k s^k}{s^{kT} B^k s^k - s^{kT} y^k} & \text{wenn } s^{kT} y^k < \theta s^{kT} B^k s^k, \end{cases}$$

wobei z.B. $\theta = 0.2$ ein Erfahrungswert ist. Nun kann der modifizierte BFGS-Update durchgeführt werden:

$$B^{k+1} = B^k - \frac{B^k s^k s^{kT} B^k}{s^{kT} B^k s^k} + \frac{\bar{y}^k \bar{y}^{kT}}{\bar{y}^{kT} s^k}. \tag{5.37}$$

Bestimmung der Schrittweite

Um die Konvergenz der sequentiellen quadratischen Programmierung auch von schlechten Startwerten zu garantieren, muss der Schritt 3 des Modell-Algorithmus modifiziert werden durch eine eindimensionale Minimumsuche ähnlich wie in Paragraph 5.1.3.[7]

Im Falle der unbeschränkten Optimierung bildet $\Phi(\alpha) = f(x^k + \alpha d^k)$ die „natürliche“ Meritfunktion, die bezüglich α approximativ minimiert wird. Bei der beschränkten Optimierung könnte eine solche Vorgehensweise die Nebenbedingungen zu stark verletzen, weshalb diese in die Meritfunktion integriert werden und so eine zu starke Verletzung bestrafen. Ein Vorschlag von Han [27] ist die sogenannte exakte Strafkosten-Funktion

$$\Phi(\alpha) = f(x^k + \alpha d^k) + \sum_{i \in \mathcal{E}} \mu_i^k |c_i(x^k + \alpha d^k)| - \sum_{i \in \mathcal{I}} \mu_i^k \min\{0, c_i(x^k + \alpha d^k)\}, \tag{5.38}$$

mit Strafkosten-Parametern $\mu_i \geq 0$, von der wie in Paragraph 5.1.3 das Minimum näherungsweise bestimmt wird. Die Strafkostenparameter können nach Powell [41] wie folgt bestimmt werden

$$\mu_i^k = \max\{|\lambda_i^k|, \frac{1}{2}(\mu_i^{k-1} + |\lambda_i^k|)\}. \tag{5.39}$$

Jetzt kann der modifizierte Modell-Algorithmus der sequentiellen quadratischen Programmierung notiert werden

Sequentielle quadratische Programmierung mit Schrittweitensteuerung und BFGS-Update:

Schritt 0 Gegeben x^0, λ^0, B^0, setze $k \leftarrow 0$,

Schritt 1 Berechne $f(x^0)$, $c(x^0)$, $\nabla f(x^0)$, $A(x^0)$,

Schritt 2 Löse (5.36) und erhalte d^k und μ^k,
falls $d^k = 0$ stop.

Schritt 3 Wähle Schrittweite α^k,
so dass $\Phi(\alpha) \leq \Phi(0) + c\alpha D\Phi(0)$,

Schritt 4 Setze $x^{k+1} \leftarrow x^k + \alpha^k d^k$ und $\lambda^{k+1} = \mu^k$,

Schritt 5 Berechne $f(x^{k+1})$, $c(x^{k+1})$, $\nabla f(x^{k+1})$, $A(x^{k+1})$, s^k, $\bar{y}^k$,
Berechne B^{k+1} aus B^k mittels quasi-Newton Update (5.37),
Setze $k \leftarrow k+1$ und wiederhole ab Schritt 2.

[7] Auf die alternativen Trust-Region-Verfahren soll hier aus Raum- und Zeitgründen nicht eingegangen werden, siehe z.B. [40, Abschitt 18.8].

Kapitel 6

Innere-Punkt-Verfahren

Die Entwicklung der Innere-Punkt-Verfahren[1] beginnt mit der Arbeit von Karmarkar [33] zu den linearen Versionen der Innere-Punkt-Verfahren.

6.1 Lineare Optimierung

6.1.1 Notwendige Bedingungen

Wir beginnen mit der Umformung des primalen Problems (P) und des dualen Problems (D) in Kapitel 4 in die gleichungsbeschränkte (Matrix/Vektor-)Form mit primalen Schlupfvariablen w und dualen Schlupfvariablen z

$$\begin{aligned} (\mathbf{P}') \quad & \max\ c^T x \\ & \text{s.t. } Ax + w = b, \\ & \qquad x, w \geq 0, \end{aligned} \tag{6.1}$$

und

$$\begin{aligned} (\mathbf{D}') \quad & \min\ b^T y \\ & \text{s.t. } A^T y - z = c, \\ & \qquad y, z \geq 0. \end{aligned} \tag{6.2}$$

Nun werden in $(\mathbf{P}')$ die Vorzeichenbeschränkungen an x und w als logarithmische Strafterme mit Strafparameter μ an die Kostenfunktion angekoppelt:

$$\begin{aligned} & \max\ c^T x + \mu \sum_{j=1}^{n} \ln x_j + \mu \sum_{i=1}^{m} \ln w_i \\ & \text{s.t. } Ax + w = b. \end{aligned} \tag{6.3}$$

Der Mechanismus dieser Ankopplung kann anhand von Fig. 6.1 erklärt werden. Wünschenswert wäre eine Addition zu $c^T x$ von 0, wenn $x \geq 0$, und $-\infty$, wenn $x < 0$, da in diesem Fall negative x-Werte unendlich hoch bestraft würden und so als Argumente für ein Maximum sicher ausfielen. Dieselben Gründe gelten für w. Diese zu addierende Funktion ist allerdings nicht glatt, aber die (beliebig häufig) differenzierbare logarithmische Straffunktion erfüllt einen ähnlichen Zweck für $\mu \downarrow 0$.

[1] Es wird auch von einer Innere-Punkt-Revolution gesprochen, s.a. New York Times vom 19. November 1984.

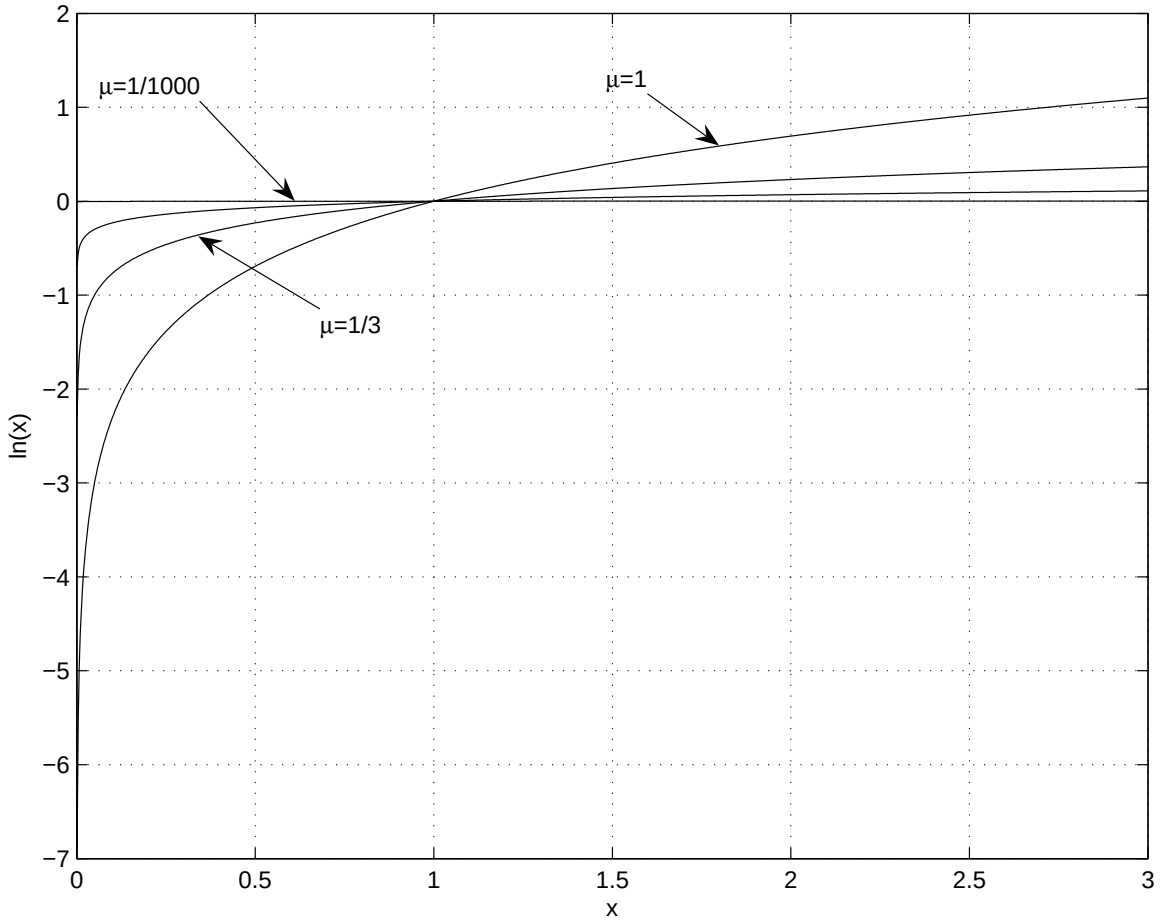


Abbildung 6.1: Barrieren-Funktion

Für dieses nun nurmehr gleichungsbeschränkte Problem wird die Lagrange-Funktion gebildet mit Lagrange-Multiplikatoren y

$$L(x, w, y) = c^T x + \mu \sum_{j=1}^{n} \ln x_j + \mu \sum_{i=1}^{m} \ln w_i + y^T(b - Ax - w). \tag{6.4}$$

Die notwendigen Bedingungen erster Ordnung für das Problem (6.3) lauten

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= c_j + \mu \frac{1}{x_j} - \sum_{i=1}^{m} y_i a_{ij} \quad = 0, \quad j = 1, \ldots, n, \\ \frac{\partial L}{\partial w_i} &= \mu \frac{1}{w_i} - y_i \qquad\qquad\quad = 0, \quad i = 1, \ldots, m, \\ \frac{\partial L}{\partial y_i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j - w_i \quad = 0, \quad i = 1, \ldots, m. \end{aligned} \tag{6.5}$$

Dieses Gleichungssystem kann mit den Matrizen $X = \mathrm{diag}(x_j)$, $W = \mathrm{diag}(w_i)$ und dem Vektor $e = (1,\ 1, \ldots, 1)^T$ kompakt als

$$\begin{aligned} A^T y - \mu X^{-1} e &= c, \\ y &= \mu W^{-1} e, \\ Ax + w &= b, \end{aligned}$$

geschrieben werden. Mit der Definition von $z := \mu X^{-1} e$ und nach Umordnung wird dieses als

$$Ax + w = b, \tag{6.7a}$$
$$A^T y - z = c, \tag{6.7b}$$
$$XZe = \mu e, \tag{6.7c}$$
$$YWe = \mu e, \tag{6.7d}$$

notiert. Die Gln. (6.7) bilden ein System von $2(m+n)$ Gleichungen in $2(m+n)$ Unbekannten $(x_\mu, y_\mu, w_\mu, z_\mu)$. Die Lösungsmenge

$$\mathcal{S} = \{(x_\mu, y_\mu, w_\mu, z_\mu),\ \mu > 0\} \tag{6.8}$$

wird als zentraler Pfad bezeichnet. Die beiden ersten Gln. in (6.7) sind die Gleichungsnebenbedingungen des primalen bzw. dualen Problems, und die beiden letzten (nichtlinearen) Gleichungen sind, für $\mu = 0$, die Komplementaritätsbedingungen.

Zwei Resultate über Existenz und Eindeutigkeit der Lösung von (6.7) werden ohne Beweis mitgeteilt, s.a. [32] und [44].

Satz 22. *Es existiert ein stationärer Wert von Gln.* (6.7) *dann und nur dann, wenn sowohl die zulässige Menge des primalen Problems* $\mathcal{P} = \{Ax \leq b,\ x \geq 0\}$ *als auch die zulässige Menge des dualen Problems* $\mathcal{D} = \{A^T y \geq c,\ y \geq 0\}$ *nicht leer sind.*

Satz 23. *Wenn ein stationärer Punkt der Gln.* (6.7) *existiert, dann ist er eindeutig und bildet das Maximum des linearen Programms.*

6.1.2 Newton-Verfahren

The KKT-System.

The system (6.7) is solved iteratively for a fixed value of μ by a damped NEWTON method to find the solution u^* of a function $F_\mu(u) = 0$

$$\nabla F_\mu(u_k)\Delta u_k = -F_\mu(u_k), \tag{6.9}$$

where $u = (x_\mu^T, y_\mu^T, w_\mu^T, z_\mu^T)^T$ is the vector of primal and dual variables and primal and dual slack variables, respectively, and the next step is generated by

$$u_{k+1} = u_k + \alpha_k \Delta u_k, \tag{6.10}$$

with an initial starting estimate $u_0 > 0$, Δu_k as the solution of Eq. (6.9), and $\alpha_k \in (0, 1]$ a damping factor. In (6.9) $F_\mu : \mathbb{R}^{2(m+n)} \mapsto \mathbb{R}^{2(m+n)}$ is the vector

$$F_\mu = \begin{pmatrix} Ax_k + w_k - b \\ A^T y_k - z_k - c \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix},$$

and $\nabla F_\mu : \mathbb{R}^{2(m+n)\times 2(m+n)} \mapsto \mathbb{R}^{2(m+n)}$ the JACOBIan matrix

$$\nabla F_\mu = \begin{pmatrix} A & & I & \\ & A^T & & -I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix}.$$

Summarized, we have to solve in each iteration an (in practice large but sparse) linear system

$$\begin{pmatrix} A & & I & \\ & A^T & & -I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \\ \Delta w_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} Ax_k + w_k - b \\ A^T y_k - z_k - c \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix}. \tag{6.11}$$

In system (6.11) the last two equations can be used to eliminate

$$\Delta w_k = Y_k^{-1}(\mu e - Y_k W_k e - W_k \Delta y_k) \tag{6.12}$$

and

$$\Delta z_k = X_k^{-1}(\mu e - X_k Z_k e - Z_k \Delta x_k), \tag{6.13}$$

thus reducing (6.11) to

$$\begin{pmatrix} X_k^{-1} Z_k & A^T \\ A & -Y_k^{-1} W_k \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} c - A^T y_k + \mu X_k^{-1} e \\ b - Ax_k - \mu Y_k^{-1} e \end{pmatrix}. \tag{6.14}$$

This is a symmetric system and the matrix is called the reduced KKT matrix. A numerical solution can be achieved e.g. by the HARWELL-Library subroutine MA27 or MA57 [15].

The Step Length.

It is required, see Eqs. (6.1) and (6.2), that all variables stay positive. Therefore, possibly the full stepsize from Eq. (6.11) cannot be applied but must be restricted to $u_k + \alpha_k \Delta u_k > 0$, or

$$\frac{1}{\alpha_k} > -\frac{\Delta u_k}{u_k}.$$

Also, we must not overstep the optimal NEWTON step $\alpha_k = 1$. Therefore the actual steplength implementation is

$$\alpha_k = \min\Big(1, \theta\Big(\max_i -\frac{\Delta u_{k_i}}{u_{k_i}}\Big)^{-1}\Big), \tag{6.15}$$

where $0.9 < \theta < 1$ is a safeguard to obtain strict positive variables.

Update of the Barrier Parameter.

There are several ways to treat the update of the barrier parameter μ. The so-called short-step method, see e.g. [32], starts with a large value of $\mu_0 \gg 1$ and reduce it by the following relationship

$$\mu_{l+1} = \mu_l\Big(1 - \frac{1}{6\sqrt{n}}\Big),$$

which for large n is near one.

We prefer VANDERBEIS [44] procedure

$$\mu_{l+1} = \delta \frac{x_{\mu_l}^T z_{\mu_l} + y_{\mu_l}^T w_{\mu_l}}{n+m}, \tag{6.16}$$

with δ a heuristic parameter to guarantee sufficient reduction of μ. In practice $\delta = 0.1$ has turned out to be a reliable value.

Convergence Criteria.

The following measures are defined: primal feasibility

$$\rho = b - Ax - w, \tag{6.17}$$

dual feasibility

$$\sigma = c - A^T y + z, \tag{6.18}$$

and complementarity

$$\gamma = x^T z + y^T w. \tag{6.19}$$

With these measures, the sequence (6.10) is stopped at a solution if the following convergence criteria are satisfied:

$$\|\rho\|_1 < \varepsilon \quad \wedge \quad \|\sigma\|_1 < \varepsilon \quad \wedge \quad \gamma < \varepsilon,$$

where ε is a small positive tolerance. Also, it is stopped if the following holds

$$\|x\|_\infty > M \quad \vee \quad \|y\|_\infty > M,$$

where M is a large positive tolerance. In the former case the primal problem is unbounded, in the latter case the dual problem is unbounded which means the primal is infeasible.

A Prototype Algorithm.

Here we present a prototype procedure of the algorithm presented in the previous section. It is implemented in `Matlab` [37].

```
1   function [u,k,status] = IPLP(A,b,c,u)
2   %Path-Following algorithm as described in Chapter 17 and 18 of rvdb's LP book
3   % max c^{T}x s.t. Ax<=b, x>=0
4   % [m,n]=size(A), n=length(c), m=length(b)
5   % u are the initial estimates of
6   % primal variables x, dim(x)=n,
7   % dual variables    y, dim(y)=m,
8   % primal slacks      w, dim(w)=m,
9   % dual slacks         z, dim(z)=n,
10  % in that order
11  % and xout are the respective output variables
12  % k is the number of iterations
13  % status=0: optimal solution found
14  % status=2: primal infeasibilty
15  % status=4: dual infeasibilty (unbounded problem)
16  % status=1: solution not found in 500 iterations
17  % version: implemented by the members of the class
18  % "Modeling, Simulation and Optimization"  summer 2004
19
20  % check errors
21  error(nargchk(3,4,nargin)); % either 3 or 4 input arguments
22  [m,n]=size(A);
23  nm=n+m;
24  nmm=nm+m;
25  nmmn=nmm+n;
```

```
26  p=nmmn;
27  if nargin == 4
28     l=length(u);
29     if l ~= p
30        error('length(u) must be 2*(m+n)');
31     end
32  end
33  if length(b) ~= m
34     error('length(b) must be m');
35  end
36  if length(c) ~= n
37     error('length(c) must be n');
38  end
39  if nargin == 3
40     u=ones(p,1); % default initial variables
41  end
42  u=u(:); % columnize!
43  em=ones(m,1);
44  en=ones(n,1);
45  DELTA=1/10;
46  R=9/10;
47  EPS=1e-8;
48  HUGE=1e10;
49  ITERMAX=500;
50  status=1;
51
52  for k=1:ITERMAX,  % iteration loop
53  x=u(1:n);
54  y=u(1+n:nm);
55  w=u(1+nm:nmm);
56  z=u(1+nmm:nmmn);
57  X=diag(x);
58  Xi=diag(1./x);
59  Y=diag(y);
60  Yi=diag(1./y);
61  W=diag(w);
62  Z=diag(z);
63  gamma=z'*x+y'*w; % duality gap
64  mu=DELTA*gamma/nm;
65  rho=b-A*x-mu*Yi*em; % primal infeasibilty
66  sigma=c-A'*y+mu*Xi*en; % dual infeasibilty
67  % check convergence (Paragraph 17.5.3)
68  if gamma<EPS && norm(b-A*x-w,1)<EPS && norm(c-A'*y+z,1)<EPS,
69     status=0;
70     break; % optimal
71  end % if
72  if norm(x,inf)>HUGE,
73     status=2;
74     break; %primal infeasibility
75  end % if
76  if norm(y,inf)>HUGE,
```

```
77      status=4;
78      break; %dual infeasibility
79    end % if
80    % reduced KKT-system (18.6-18.7)
81    KKT=[Xi*Z,A';A,-Yi*W]; % symmetric KKT-matrix
82    du=KKT\[sigma;rho]; % solve it
83    dx=du(1:n);
84    dy=du(1+n:nm);
85    % calculate DELTA slacks
86    dz=Xi*(mu*en-X*Z*en-Z*dx);
87    dw=Yi*(mu*em-Y*W*em-W*dy);
88    du=[du;dw;dz];
89    theta=max(-du./u); % ratio test (17.6)
90    theta=min(1,R/theta);
91    u=u+theta*du;
92    end % for
```

6.2 Quadratic Programming

6.2.1 Duality

A quadratic program usually is formulated as follows[2]

$$\begin{aligned} \text{minimize } & \frac{1}{2}x^T Q x + c^T x \\ \text{subject to } & Ax \geq b, \\ & x \geq 0, \end{aligned} \tag{6.20}$$

where it can be assumed that the matrix Q is symmetric.[3]

Associated to this program is a dual formulation[4]

$$\begin{aligned} \text{maximize } & b^T y - \frac{1}{2}x^T Q x \\ \text{subject to } & A^T y + z - Q x = c, \\ & y, z \geq 0. \end{aligned} \tag{6.21}$$

6.2.2 Necessary Conditions

As in linear programming, the general inequality constraints are changed to equality constraints by introducing slack variables w

$$\begin{aligned} \text{minimize } & \frac{1}{2}x^T Q x + c^T x \\ \text{subject to } & Ax - w = b, \\ & x, w \geq 0. \end{aligned}$$

[2] Attend that the cost function is quadratic in the variables and the constraints are linear. Also, the cost function is minimized and the constraints are non-negative.

[3] A non-symmetric $\bar{Q}$ can be made symmetric through $Q = \bar{Q}^T \bar{Q}/2$

[4] For a thorough discussion of duality in quadratic programming see [44, pp. 399–402]

Again, the bounds on the variables (x, w) are tied to the cost function via barrier functions[5]

$$\begin{aligned} &\text{minimize } \frac{1}{2}x^T Q x + c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{i=1}^{m} \ln w_i \\ &\text{subject to } Ax - w = b. \end{aligned} \tag{6.22}$$

After defining the LAGRANGE function

$$L(x, w, y) = \frac{1}{2}x^T Q x + c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{i=1}^{m} \ln w_i + y^T(b - Ax + w), \tag{6.23}$$

and setting its partial derivatives to zero we get a set of first-order necessary conditions for a critical point of problem (6.20)

$$\begin{aligned} Ax - w &= b, \\ A^T y + z - Qx &= c, \\ XZe &= \mu e, \\ YWe &= \mu e, \end{aligned} \tag{6.24}$$

where as in Eq. (3.10) we defined $z = \mu X^{-1} e$.

To guarantee a solution of problem (6.20) it is necessary that the matrix Q is positive semidefinite, i.e. the eigenvalues of Q must be non-negative. This means that the quadratic cost function of the primal problem is a convex function over the convex set $\mathcal{P} = \{Ax - w = b,\ x, w \geq 0\}$. If this is valid the following can be stated:

Satz 24. *If a convex quadratic program has a solution $(x^*, w^*) \in \mathcal{P}$ and its dual program has a solution $(x^*, y^*, z^*) \in \mathcal{D} = \{A^T y + z - Qx = c,\ x, y, z \geq 0\}$ such that*

$$0 = y^{*T} w^* + z^{*T} x^* \tag{6.25}$$

then the primal solution is optimal for the primal problem and the dual solution is optimal for the dual problem.

A proof of this theorem for which convexity is essential can be found in [44].

6.2.3 Newton's Method

The procedure to solve the necessary conditions (6.24) is analog to that in section 6.1.2 with the appropriate modifications.

The Search Direction.

Again, NEWTONS method is used to find the zero of (6.24)

$$\nabla F_\mu(u_k) \Delta u_k = -F_\mu(u_k),$$

in connection with the update of the variables

$$u_{k+1} = u_k + \alpha_k \Delta u_k.$$

[5] Note the minus signs because we are minimizing!

This time F_μ and ∇F_μ write as

$$F_\mu = \begin{pmatrix} Ax_k - w_k - b \\ A^T y_k + z_k - c - Qx_k \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix},$$

and

$$\nabla F_\mu = \begin{pmatrix} A & & -I & \\ -Q & A^T & & I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix}.$$

After elimination of Δw_k and Δz_k from the last two equations

$$\Delta w_k = Y_k^{-1}(\mu e - Y_k W_k e - W_k \Delta y_k), \tag{6.26}$$

and

$$\Delta z_k = X_k^{-1}(\mu e - X_k Z_k e - Z_k \Delta x_k), \tag{6.27}$$

we arrive at the following symmetric KKT system

$$\begin{pmatrix} -X_k^{-1}Z_k - Q & A^T \\ A & Y_k^{-1}W_k \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} c + Qx_k - A^T y_k - \mu X_k^{-1} e \\ b - Ax_k + \mu Y_k^{-1} e \end{pmatrix}. \tag{6.28}$$

The technical remainder of this section, namely steplength, barrier-update, and convergence criteria is the same as in section 6.1.2. Therefore it is omitted here, and we present, again implemented in `Matlab`

A Prototype Algorithm

```
1   function [u,k,status] = IPQP(A,b,c,Q,u)
2   %Path-Following QP-algorithm as described in Chapter 23 of rvdb's LP book
3   % min c'x+x'Qx/2 s.t. Ax>=b, x>=0
4   % [m,n]=size(A), n=length(c), m=length(b), [n,n]=size(Q)
5   % u are the initial estimates of
6   % primal variables x, dim(x)=n,
7   % dual variables    y, dim(y)=m,
8   % primal slacks      w, dim(w)=m,
9   % dual slacks         z, dim(z)=n,
10  % in that order
11  % and xout are the respective output variables
12  % k is the number of iterations
13  % status=0: optimal solution found
14  % status=2: primal infeasibilty
15  % status=4: dual infeasibilty (unbounded problem)
16  % status=1: solution not found in 500 iterations
17  % version: implemented by the members of the class
18  % "Modeling, Simulation and Optimization"  summer 2004
19
20  % check errors
21  error(nargchk(4,5,nargin)); % either 3 or 4 input arguments
22  [m,n]=size(A);
23  [q,q]=size(Q);
```

```
Q=(Q+Q')/2;
nm=n+m;
nmm=nm+m;
nmmn=nmm+n;
if nargin == 5
   l=length(u);
   if l ~= nmmn
      error('length(u) must be 2*(m+n)');
   end
end
if length(b) ~= m
   error('length(b) must be m');
end
if length(c) ~= n
   error('length(c) must be n');
end
if q ~=n
   error('size(Q) must be nxn');
end % if
if nargin == 4
   u=ones(nmmn,1); % default initial variables
end
u=u(:); % columnize!
em=ones(m,1);
en=ones(n,1);
DELTA=1/10;
R=9/10;
EPS=1e-8;
HUGE=1e10;
ITERMAX=500;
status=1;

for k=1:ITERMAX,  % iteration loop
x=u(1:n);
y=u(1+n:nm);
w=u(1+nm:nmm);
z=u(1+nmm:nmmn);
X=diag(x);
XI=diag(1./x);
Y=diag(y);
YI=diag(1./y);
W=diag(w);
Z=diag(z);
gamma=z'*x+y'*w; % duality gap
mu=DELTA*gamma/nm;
rho=b-A*x+w;
rho1=b-A*x+mu*YI*em; % primal infeasibilty
sigma=c-A'*y-z+Q*x;
sigma1=c-A'*y-mu*XI*en+Q*x; % dual infeasibilty
% check convergence (Paragraph 17.5.3)
if gamma<EPS && norm(rho,1)<EPS && norm(sigma,1)<EPS,
```

```
75      status=0;
76      break; % optimal
77    end % if
78    if norm(x,inf)>HUGE,
79      status=2;
80      break; %primal infeasibility
81    end % if
82    if norm(y,inf)>HUGE,
83      status=4;
84      break; %dual infeasibility
85    end % if
86    % reduced KKT-system (18.6-18.7)
87    KKT=[-(XI*Z+Q),A';A,YI*W]; % symmetric KKT-matrix
88    du=KKT\[sigma1;rho1]; % solve it
89    dx=du(1:n);
90    dy=du(1+n:nm);
91    % calculate DELTA slacks
92    dz=XI*(mu*en-Z*(X*en+dx));
93    dw=YI*(mu*em-W*(Y*em+dy));
94    du=[dx;dy;dw;dz];
95    theta=max(-du./u); % ratio test (17.6)
96    theta=min(1,R/theta);
97    u=u+theta*du;
98    end % for
```

Chapter 7

Finite Element Modeling

The finite element method is a numerical method to solve partial differential equations by which almost all engineering phenomena are modeled; think of the Navier-Stokes equation in fluid mechanics or the Maxwell equations in electro-magneto dynamics, to mention only two. This is by far not an exhausting analysis of the finite element method. We only give a short introduction into bar and beam elements as these constitute the models of mechanical structures which we want to design under some optimization aspects. A comprehensive treatment of the topic can be found in several references, e.g. Hughes [30].

7.1 One-Element Bar

Fig. 7.1 schematically shows a single-bar element with two joints and axial time-dependent displacement $u(x,t)$ and axial time-dependend load $f(x,t)$. We assume a homogeneous medium with density ρ and Young's or elasticity modulus E. In the model of the longitudinal vibration with coordinate x of a (slender)[1] bar with length L and cross section $A(x)$ the displacement $u(x,t)$ satisfies the following partial differential equation

$$\frac{\partial}{\partial x}\left(EA(x)\frac{\partial u(x,t)}{\partial x}\right) = \rho A(x)\frac{\partial^2 u(x,t)}{\partial t^2}, \tag{7.1}$$

as developed e.g. in Inman [31]. The static displacement satisfies for constant cross section

$$EA\frac{d^2u(x)}{dx^2} = 0,$$

which upon integration yields

$$u(x) = c_1 + c_2 x,$$

with "constants" of integration c_1 and c_2, which in the time-dependent case are functions of time:

$$u(x,t) = c_1(t) + c_2(t)x. \tag{7.2}$$

The joint displacements have to satisfy Eq. (7.2), therefore

$$c_1(t) = u(0,t) = u_1(t) \tag{7.3}$$

and

$$c_2(t) = \frac{u(L,t) - u(0,t)}{L} = \frac{u_2(t) - u_1(t)}{L}, \tag{7.4}$$

[1] We limit the discussion to one spatial (the axial) dimension.

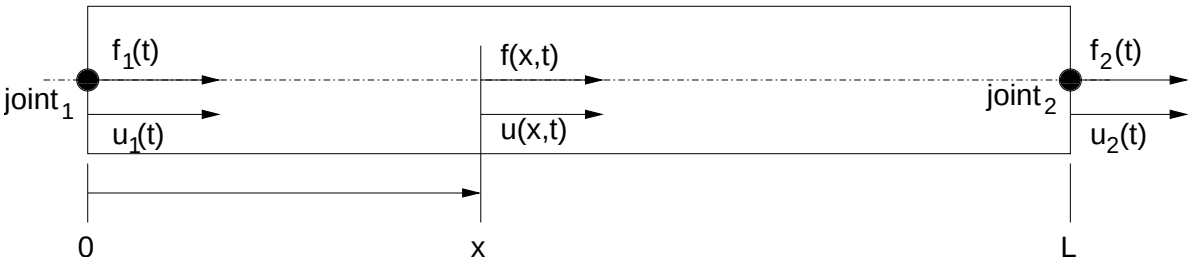


Figure 7.1: Single-Element Bar

which when inserted into Eq. (7.2) yield

$$u(x,t) = \left(1 - \frac{x}{L}\right) u_1(t) + \frac{x}{L} u_2(t) = w_1(x)u_1(t) + w_2(x)u_2(t), \tag{7.5}$$

where w_1 and w_2 are shape functions. Note, that thus $u(x,t)$ is factored into a product of a function which depends on x only and one that depends on t only.

The kinetic energy of the element, say k, can be expressed as

$$\begin{aligned} T^{(k)}(t) &= \frac{1}{2}\int_0^L \rho A \left(\frac{\partial u(x,t)}{\partial t}\right)^2 dx \\ &= \frac{1}{2}\int_0^L \rho A \left[\left(1 - \frac{x}{L}\right)\frac{du_1(t)}{dt} + \left(\frac{x}{L}\right)\frac{du_2(t)}{dt}\right]^2 dx \\ &= \frac{\rho A L}{6}(\dot{u}_1^2 + \dot{u}_1\dot{u}_2 + \dot{u}_2^2) \\ &= \frac{1}{2}\dot{u}(t)^T \mathbf{M_L} \dot{u}(t), \end{aligned} \tag{7.6}$$

where $\dot{u}(t) = [\dot{u}_1(t), \dot{u}_2(t)]^T$ is the displacement velocity vector and

$$\mathbf{M_L} = \frac{\rho A L}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

is the mass matrix expressed in the local (axial) coordinate system.

Similarly, the potential or strain energy of the same element can be expressed as

$$\begin{aligned} V^{(k)}(t) &= \frac{1}{2}\int_0^L EA \left(\frac{\partial u(x,t)}{\partial x}\right)^2 dx \\ &= \frac{1}{2}\int_0^L EA \left[-\left(\frac{1}{L}\right)u_1(t) + \left(\frac{1}{L}\right)u_2(t)\right]^2 dx \\ &= \frac{EA}{2L}(u_1^2 - 2u_1u_2 + u_2^2) \\ &= \frac{1}{2}u(t)^T \mathbf{K_L} u(t), \end{aligned} \tag{7.7}$$

where $u(t) = [u_1(t), u_2(t)]^T$ is the displacement vector and

$$\mathbf{K_L} = \frac{AE}{L}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

is the stiffness matrix in the axial coordinate system.

The virtual work due to the distributed external force $f(x,t)$ can be written as

$$\begin{aligned}\delta W^{(k)}(t) &= \int_0^L f(x,t)\delta u(x,t)dx \\ &= \int_0^L f(x,t)\left[\left(1-\frac{x}{L}\right)\delta u_1(t)+\left(\frac{x}{L}\right)\delta u_2(t)\right]dx \\ &= \int_0^L f(x,t)\left(1-\frac{x}{L}\right)dx\delta u_1(t)+\int_0^L f(x,t)\left(\frac{x}{L}\right)dx\delta u_2(t) \\ &= f(t)^T\delta u(t),\end{aligned} \tag{7.8}$$

with $\delta u(t) = [\delta u_1(t), \delta u_2(t)]^T$ and the joint or node forces in the axial coordinate system.

$$f_L(t) = \begin{pmatrix} \int_0^L f(x,t)\left(1-\frac{x}{L}\right)dx \\ \int_0^L f(x,t)\left(\frac{x}{L}\right)dx \end{pmatrix}.$$

The equations of motion for the single-element bar can be derived by the Lagrange-equation of the second kind

$$\frac{\partial}{\partial t}\left(\frac{\partial T(t)}{\partial \dot{u}_i(t)}\right) - \frac{\partial T(t)}{\partial u_i(t)} + \frac{\partial V(t)}{\partial u_i(t)} = f_i(t), \quad i = 1,2, \tag{7.9}$$

to

$$\mathbf{M_L}\ddot{u}(t) + \mathbf{K_L}u(t) = f_L(t). \tag{7.10}$$

7.2 Element Transformation

Structures composed of bar elements are classified as truss structures. As bars by definition can only carry axial loads, no transverse shearing forces and bending moments are experienced by any member. All members are connected to each other through pins that allow free rotation about the pin axis. The bar members of a truss generally have a different orientation against each other. Therefore the analysis of displacements and stresses requires the setup of a global coordinate system in which all interesting quantities of individual elements can be referenced, thus being able to assemble the structure as a whole and imposing boundary conditions on it.

Generally, every element k of the truss has two nodes i and j, respectively, with displacement components in a global coordinate system $X-Z$, namely $U_{(i)_X}$ and $U_{(i)_Z}$ and $U_{(j)_X}$ and $U_{(j)_Z}$. These components are written in a consecutive order for convenience, namely U_{2i-1}, U_{2i}, U_{2j-1}, and U_{2j}. For $i = 1$ and $j = 2$, these are $U_1, \ldots, U_4$, as is indicated in Fig.7.2. The two sets of joint displacements are related as follows

$$\begin{aligned} u_1(t) &= U_{2i-1}(t)\cos\theta + U_{2i}(t)\sin\theta \\ u_2(t) &= U_{2j-1}(t)\cos\theta + U_{2j}(t)\sin\theta \end{aligned}$$

which is written compactly as

$$u(t) = \Gamma U(t), \tag{7.11}$$

with the coordinate transformation matrix given by

$$\Gamma = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{pmatrix}.$$

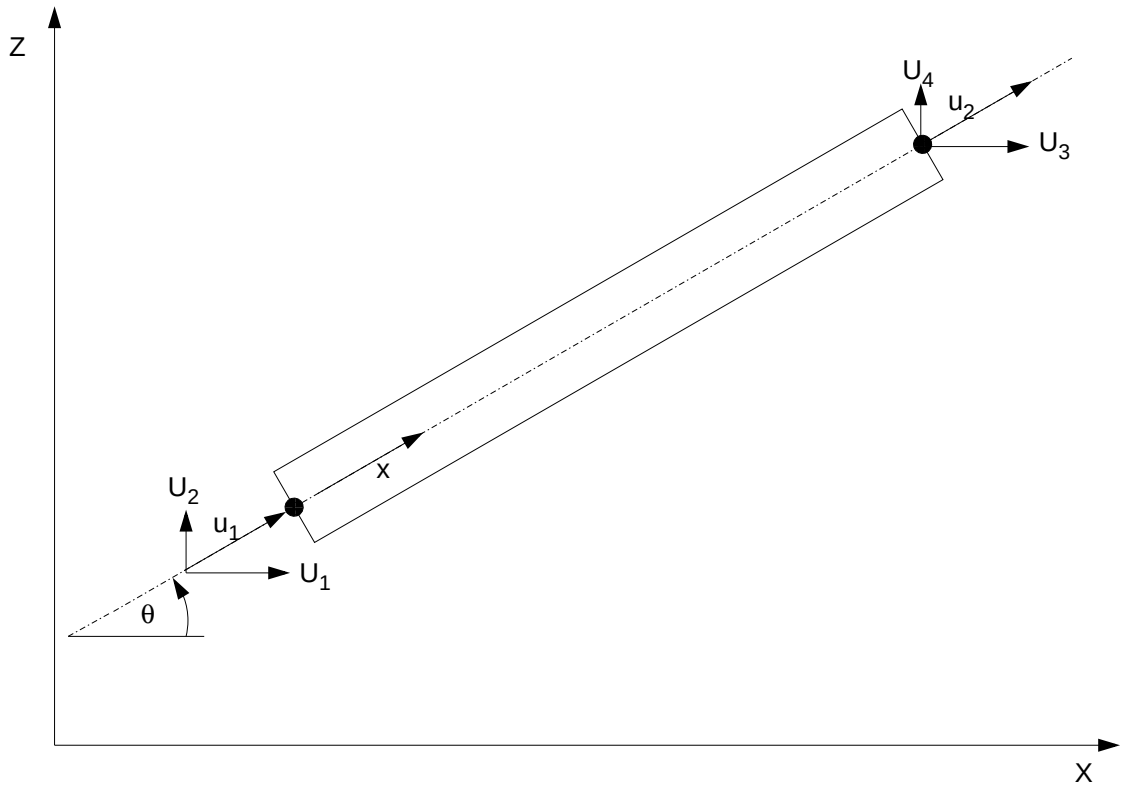


Figure 7.2: Element in Global Coordinates

As the matrix Γ is independent of time the displacement velocity can be expressed as

$$\dot{u}(t) = \Gamma \dot{U}(t). \tag{7.12}$$

Scalar entities like energy expressions are independent of coordinate systems and equal in any of them; therefore

$$T^{(k)}(t) = \frac{1}{2}\dot{U}(t)^T \mathbf{M}^{(k)} \dot{U}(t), \tag{7.13}$$

with the mass matrix written in the global coordinate system

$$\mathbf{M}^{(k)} = \Gamma^T \mathbf{M}_L \Gamma, \tag{7.14}$$

and

$$V^{(k)}(t) = \frac{1}{2}U(t)^T \mathbf{K}^{(k)} U(t), \tag{7.15}$$

with the stiffness matrix written in the global coordinate system

$$\mathbf{K}^{(k)} = \Gamma^T \mathbf{K}_L \Gamma. \tag{7.16}$$

Also, the external force can be transformed to the global coordinate system as follows

$$f^{(k)}(t) = \Gamma^T f_L(t). \tag{7.17}$$

Now, the model equation for a single bar element k in global coordinates is

$$\mathbf{M}^{(k)} \ddot{U}(t) + \mathbf{K}^{(k)} U(t) = f^{(k)}(t). \tag{7.18}$$

For the element in Fig.7.2 the mass matrix is calculated as

$$\mathbf{M}^{(1)} = \frac{\rho A L}{6} \begin{pmatrix} 2\cos^2\theta & 2\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ 2\cos\theta\sin\theta & 2\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \\ \cos^2\theta & \cos\theta\sin\theta & 2\cos^2\theta & 2\cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta & 2\cos\theta\sin\theta & 2\sin^2\theta \end{pmatrix}, \tag{7.19}$$

and the stiffness matrix is calculated as

$$\mathbf{K}^{(1)} = \frac{EA}{L}\begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & -\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ -\cos\theta\sin\theta & -\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \end{pmatrix}. \tag{7.20}$$

7.3 System Assembly

A complete truss structure generally consists of a number p of bars assembled together with pins as joining links. The joint displacements and velocities are concatenated in the following vectors $\bar{U}(t) = [U_1(t), \ldots, U_m(t)]^T$ and $\dot{\bar{U}}(t) = [\dot{U}_1(t), \ldots, \dot{U}_m(t)]^T$. The element displacements $U^{(k)}(t)$and velocities $\dot{U}^{(k)}(T)$ can be extracted from these vectors by premultiplying them with a matrix $\mathbf{I}^{(k)}$ which is essentially zero-matrix which has entries of ones at apropriate locations. For example, to excerpt $U^2 = [U_5, \ldots, U_8]^T$ from $U = [U_1, \ldots, U_{12}]^T$ the following multiplication yields the desired result

$$U^{(2)} = \begin{pmatrix} U_5 \\ U_6 \\ U_7 \\ U_8 \end{pmatrix} = \mathbf{I}^{(2)} \begin{pmatrix} U_1 \\ \vdots \\ \vdots \\ U_{12} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ \vdots \\ U_{12} \end{pmatrix}.$$

The kinetic energy of the assembly is collected from the sum of the kinetic energies of all elements

$$T(t) = \frac{1}{2}\dot{\bar{U}}(t)^T \mathbf{M} \dot{\bar{U}}(t) = \frac{1}{2}\sum_{k=1}^{p} \dot{\bar{U}}(t)^T \mathbf{I}^{(k)T} \mathbf{M}^{(k)} \mathbf{I}^{(k)} \dot{\bar{U}}(t), \tag{7.21}$$

with the mass matrix of the complete structure

$$\mathbf{M} = \sum_{k=1}^{p} \mathbf{I}^{(k)T} \mathbf{M}^{(k)} \mathbf{I}^{(k)}. \tag{7.22}$$

Likewise, we get the total strain energy as

$$V(t) = \frac{1}{2}\bar{U}(t)^T \mathbf{K} \bar{U}(t) = \frac{1}{2}\sum_{k=1}^{p} \bar{U}(t)^T \mathbf{I}^{(k)T} \mathbf{K}^{(k)} \mathbf{I}^{(k)} \bar{U}(t), \tag{7.23}$$

with the stiffness matrix of the complete structure

$$\mathbf{K} = \sum_{k=1}^{p} \mathbf{I}^{(k)T} \mathbf{K}^{(k)} \mathbf{I}^{(k)}. \tag{7.24}$$

Finally, the consideration of the virtual work gives the vector of joint forces of the complete structure

$$f(t) = \sum_{k=1}^{p} \mathbf{I}^{(k)T} f^{(k)}(t). \tag{7.25}$$

The calculation of the $f^k(t)$ from the local distributed forces $f_L(t)$ did not take into consideration any concentrated forces acting on any joint. These must be added to the respective row of the following model equation.

Now, that total kinetic and potential energy as well as the generalized force vector are given, the equation of motion of the structure can be developed from the principle of Lagrange Eq. (7.9)

$$\mathbf{M}\ddot{U}(t) + \mathbf{K}U(t) = f(t). \tag{7.26}$$

7.4 Boundary Conditions

A careful evaluation of the stiffness matrix $\mathbf{K}$ results that it is singular. The reason of this is that in the above derivation of Eq. (7.26) all the joints were assumed to be free and the overall structure was able to move as a rigid body under the influence of the joint forces. Usually the truss structure is supported at some joints thus not being able to perform rigid body motion. This is equivalent to imposing boundary conditions to Eq. (7.26). This can be achieved by deleting those rows and columns from the mass and stiffness matrix and those rows from the force vector which correspond to zero node displacement conditions, thereby reducing the differential equation from dimension m to say n.

7.5 Examples

We want to show the details of the described procedure by some examples.

7.5.1 Two-Bar Truss

In the first example the node displacement and the eigen-frequencies of a symmetric homogeneous two-bar truss with bars of length L and constant cross section A as shown in Fig. 7.3 are calculated. The material properties are density ρ and elastic modulus E. The three joints (in the case of a single bar element also the nodes) are indicated by circles, the two bar elements are indicated by squares. The structure is loaded by a point force P acting at node 2. We associate to node 1 the global displacement in X-direction as U_1 and in Z-direction

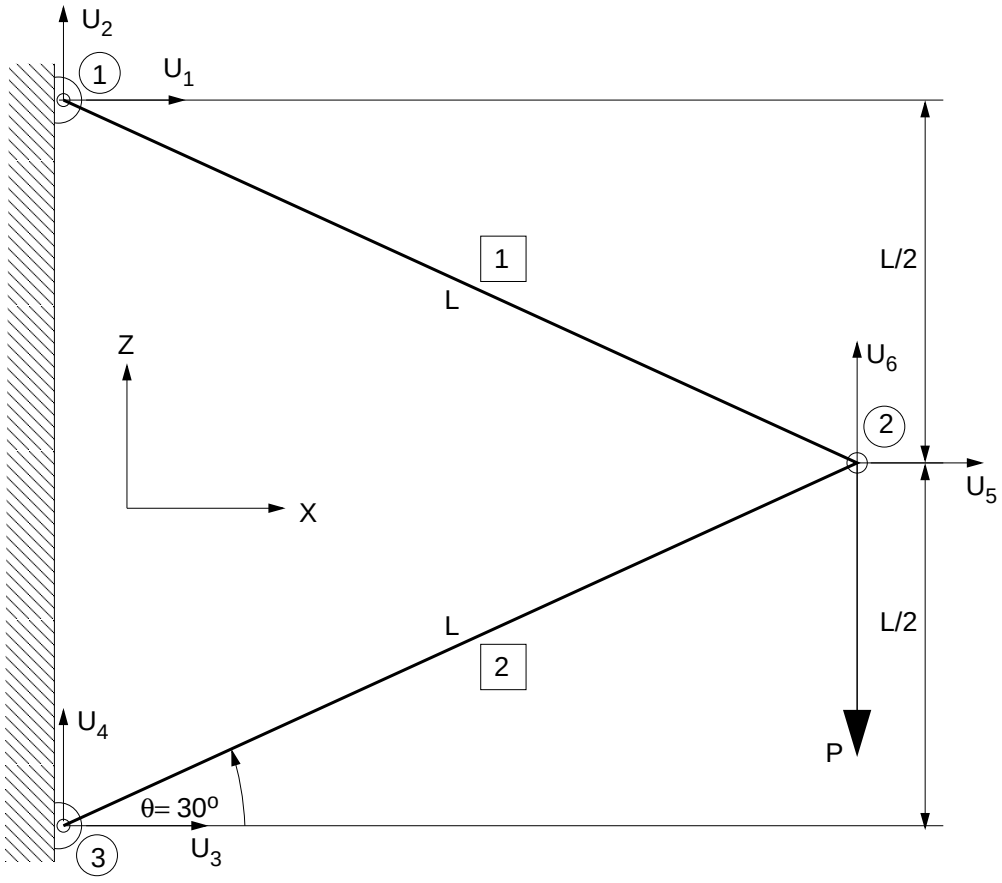


Figure 7.3: Two-Bar Truss

as U_2; similarly to node 3 the global displacements U_3 and U_4 and to node 2 the global displacements U_5 and U_6. For sake of brevity we omit the time-argument (t).

We get the local displacements for bar 1 as

$$u_1 = U_1 \cos\theta - U_2 \sin\theta$$
$$u_2 = U_5 \cos\theta - U_6 \sin\theta$$

or compactly with $u^{(1)} = [u_1, u_2]^T$ and $U^{(1)} = [U_1, U_2, U_5, U_6]^T$

$$u^{(1)} = \Gamma^{(1)} U^{(1)},$$

where

$$\Gamma^{(1)} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \end{pmatrix}.$$

With $\Gamma^{(1)}$ and

$$\mathbf{K_L} = \frac{AE}{L} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

the stiffness matrix for element 1 can be calculated

$$\mathbf{K}^{(1)} = \Gamma^{(1)T} \mathbf{K_L} \Gamma^{(1)} = \frac{EA}{L} \begin{pmatrix} \cos^2\theta & -\cos\theta\sin\theta & -\cos^2\theta & \cos\theta\sin\theta \\ -\cos\theta\sin\theta & \sin^2\theta & \cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & \cos\theta\sin\theta & \cos^2\theta & -\cos\theta\sin\theta \\ \cos\theta\sin\theta & -\sin^2\theta & -\cos\theta\sin\theta & \sin^2\theta \end{pmatrix},$$

and with

$$\mathbf{M_L} = \frac{\rho AL}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

we calculate the local mass matrix for element 1 as

$$\mathbf{M}^{(1)} = \Gamma^{(1)T} \mathbf{M_L} \Gamma^{(1)} = \frac{\rho AL}{6} \begin{pmatrix} 2\cos^2\theta & -2\cos\theta\sin\theta & \cos^2\theta & -\cos\theta\sin\theta \\ -2\cos\theta\sin\theta & 2\sin^2\theta & -\cos\theta\sin\theta & \sin^2\theta \\ \cos^2\theta & -\cos\theta\sin\theta & 2\cos^2\theta & -2\cos\theta\sin\theta \\ -\cos\theta\sin\theta & \sin^2\theta & -2\cos\theta\sin\theta & 2\sin^2\theta \end{pmatrix}.$$

The local displacements for bar 2 are

$$u_3 = U_3 \cos\theta + U_4 \sin\theta$$
$$u_4 = U_5 \cos\theta + U_6 \sin\theta$$

or compactly with $u^{(2)} = [u_3, u_4]^T$ and $U^{(2)} = [U_3, U_4, U_5, U_6]^T$

$$u^{(2)} = \Gamma^{(2)} U^{(2)},$$

where

$$\Gamma^{(2)} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{pmatrix}.$$

Now, mass and stiffness matrix are the same as in Eq. (7.19) and Eq. (7.20), respectively, which are repeated here for convenience:

$$\mathbf{M}^{(2)} = \frac{\rho AL}{6} \begin{pmatrix} 2\cos^2\theta & 2\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ 2\cos\theta\sin\theta & 2\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \\ \cos^2\theta & \cos\theta\sin\theta & 2\cos^2\theta & 2\cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta & 2\cos\theta\sin\theta & 2\sin^2\theta \end{pmatrix},$$

and

$$\mathbf{K}^{(2)} = \frac{EA}{L}\begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & -\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ -\cos\theta\sin\theta & -\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \end{pmatrix}.$$

We see from $U^{(1)} = [U_1, U_2, U_5, U_6]^T$ that

$$\mathbf{I}^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and from $U^{(2)} = [U_3, U_4, U_5, U_6]^T$ that

$$\mathbf{I}^{(2)} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Pre- and post-multiplying mass and stiffness matrices with $\mathbf{I}^{(i)T}$ and $\mathbf{I}^{(i)}$ for $i = 1, 2$, respectively gives

$$\mathbf{K}^{(1)} = \frac{EA}{L}\begin{pmatrix} \cos^2\theta & -\cos\theta\sin\theta & 0 & 0 & -\cos^2\theta & \cos\theta\sin\theta \\ -\cos\theta\sin\theta & \sin^2\theta & 0 & 0 & \cos\theta\sin\theta & -\sin^2\theta \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\cos^2\theta & \cos\theta\sin\theta & 0 & 0 & \cos^2\theta & -\cos\theta\sin\theta \\ \cos\theta\sin\theta & -\sin^2\theta & 0 & 0 & -\cos\theta\sin\theta & \sin^2\theta \end{pmatrix},$$

$$\mathbf{M}^{(1)} = \frac{\rho AL}{6}\begin{pmatrix} 2\cos^2\theta & -2\cos\theta\sin\theta & 0 & 0 & \cos^2\theta & -\cos\theta\sin\theta \\ -2\cos\theta\sin\theta & 2\sin^2\theta & 0 & 0 & -\cos\theta\sin\theta & \sin^2\theta \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \cos^2\theta & -\cos\theta\sin\theta & 0 & 0 & 2\cos^2\theta & -2\cos\theta\sin\theta \\ -\cos\theta\sin\theta & \sin^2\theta & 0 & 0 & -2\cos\theta\sin\theta & 2\sin^2\theta \end{pmatrix},$$

$$\mathbf{K}^{(2)} = \frac{EA}{L}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta \\ 0 & 0 & \cos\theta\sin\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ 0 & 0 & -\cos^2\theta & -\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ 0 & 0 & -\cos\theta\sin\theta & -\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \end{pmatrix},$$

$$\mathbf{M}^{(2)} = \frac{\rho AL}{6}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\cos^2\theta & 2\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ 0 & 0 & 2\cos\theta\sin\theta & 2\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \\ 0 & 0 & \cos^2\theta & \cos\theta\sin\theta & 2\cos^2\theta & 2\cos\theta\sin\theta \\ 0 & 0 & \cos\theta\sin\theta & \sin^2\theta & 2\cos\theta\sin\theta & 2\sin^2\theta \end{pmatrix}.$$

These component matrices can now be added to give

$$\mathbf{K} = \frac{AE}{L}\begin{pmatrix} \cos^2\theta & -\cos\theta\sin\theta & 0 & 0 & -\cos^2\theta & \cos\theta\sin\theta \\ -\cos\theta\sin\theta & \sin^2\theta & 0 & 0 & \cos\theta\sin\theta & -\sin^2\theta \\ 0 & 0 & \cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta \\ 0 & 0 & \cos\theta\sin\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta & 2\cos^2\theta & 0 \\ \cos\theta\sin\theta & -\sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta & 0 & 2\sin^2\theta \end{pmatrix}$$

and

$$\mathbf{M} = \frac{\rho AL}{6}\begin{pmatrix} 2\cos^2\theta & -2\cos\theta\sin\theta & 0 & 0 & \cos^2\theta & -\cos\theta\sin\theta \\ -2\cos\theta\sin\theta & 2\sin^2\theta & 0 & 0 & -\cos\theta\sin\theta & \sin^2\theta \\ 0 & 0 & 2\cos^2\theta & 2\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta \\ 0 & 0 & 2\cos\theta\sin\theta & 2\sin^2\theta & \cos\theta\sin\theta & \sin^2\theta \\ \cos^2\theta & -\cos\theta\sin\theta & \cos^2\theta & \cos\theta\sin\theta & 4\cos^2\theta & 0 \\ -\cos\theta\sin\theta & \sin^2\theta & \cos\theta\sin\theta & \sin^2\theta & 0 & 4\sin^2\theta \end{pmatrix}.$$

The external force vector is

$$f = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -P \end{pmatrix}^T.$$

Now, the differential model-equation (7.26) can be set up.

The boundary conditions arise from the fact that nodes 1 and 3 are pinned to the wall, therefore

$$u_1 = u_2 = u_3 = u_4 = 0.$$

With this information we extract from (7.26) the last to rows and columns of $\mathbf{M}$ and $\mathbf{K}$ and the last two rows of $\ddot{u}$, u and f by pre- and post-multiplying with $\mathbf{I}^{(BC)T}$ and $\mathbf{I}^{(BC)}$, respectively, where

$$\mathbf{I}^{(BC)} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Finally, the dynamic system ist described by

$$\frac{\rho AL}{6}\begin{pmatrix} 4\cos^2\theta & 0 \\ 0 & 4\sin^2\theta \end{pmatrix}\begin{pmatrix} \ddot{u}_5 \\ \ddot{u}_6 \end{pmatrix} + \frac{AE}{L}\begin{pmatrix} 2\cos^2\theta & 0 \\ 0 & 2\sin^2\theta \end{pmatrix}\begin{pmatrix} u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} 0 \\ -P \end{pmatrix},$$

which is shortened as

$$\mathbf{M}^{(BC)}\ddot{u}^{(BC)} + \mathbf{K}^{(BC)}u^{(BC)} = f^{(BC)}.$$

This is a decoupled linear differential equation of second order and constant coefficients, which is also well known from vibration engineering, see e.g. [31]. The eigenvalues are found by solving the generalized eigenvalue problem

$$\left|\mathbf{M}^{(BC)}\lambda - \mathbf{K}^{(BC)}\right| = 0,$$

as

$$\lambda_{1,2} = \frac{3E}{\rho L^2}.$$

The static deflection is calculated as

$$u^{(BC)} = \begin{pmatrix} u_5 \\ u_6 \end{pmatrix} = (\mathbf{K}^{(BC)})^{-1}\begin{pmatrix} 0 \\ -P \end{pmatrix} = \begin{pmatrix} 0 \\ -\dfrac{LP}{2AE\sin^2\theta} \end{pmatrix}.$$

The stress is calculated as

$$\sigma = E\epsilon = E\frac{\Delta L}{L},$$

or

$$\sigma_1 = E\frac{U_5\cos\theta - U_6\sin\theta}{L},$$
$$\sigma_2 = E\frac{U_5\cos\theta + U_6\sin\theta}{L}.$$

7.5.2 A Generalization

The foregoing type of symbolic computation is best accomplished by a mathematical spreadsheed like `Maple` [28] or `Mathematica` [48]. Here follows a generalization of the example as a `Maple` worksheet. Bar 1 has length L, which is also the distance of the fixed nodes. Bar 1 has a variable angle θ starting with $\theta = 0°$ counted clockwise from the horizontal axis. It is limited $0° \leq \theta \leq 45°$. Accordingly, the length L_2 of bar 2 varies with θ. Also the cross sections A_1 and A_2 of bar 1 and bar 2, respectively, are now free variables. Now, the load P has a horizontal component of κP, too. At the end of the worksheet the stresses in the two bars are exported to fortran-code, examplifying that any expression can be used for further calculations, e.g. optimization, in `Fortran` or (with minor modifications) in `Ampl`

FE Modell einer Zwei-Stab-Struktur (s.a. Manuskript Kap. 7)

```
> restart:
```

Fuer den TeX-output wird labeling=false gesetzt

```
> interface(labeling=false,warnlevel=0):
> with(LinearAlgebra):
> with(CodeGeneration):
```

Theta ist der Winkel zwischen oberem Stab und der Horizontalen

L ist die Laenge des oberen Stabes und der Abstand der (festen) Knoten 1 und 2

```
> #theta:=0:
> kappa:=1/2:
> L2:=sqrt(cos(theta)^2+(1-sin(theta))^2)*L:
> phi:=arctan(1-sin(theta),cos(theta)):
```

Struktur der Steifigkeitsmatrix

```
> k:=Matrix([[1,-1],[-1,1]]):
```

Struktur der Massenmatrix

```
> m:=Matrix([[2,1],[1,2]]):
```

Transformationmatrix von Stab 1 in globales Koordinatensystem

```
> G1:=Matrix([[cos(theta),-sin(theta),0,0],[0,0,cos(theta),-sin(theta)]
> ]):
```

Transformationmatrix von Stab 2 in globales Koordinatensystem

```
> G2:=Matrix([[cos(phi),sin(phi),0,0],[0,0,cos(phi),sin(phi)]]):
```

Transformation der S-Matrix und der M-Matrix von Stab 1 in globale Koordinaten

```
> K1:=MatrixMatrixMultiply(Transpose(G1),k):
> K1:=MatrixMatrixMultiply(K1,G1):
> M1:=MatrixMatrixMultiply(Transpose(G1),m):
> M1:=MatrixMatrixMultiply(M1,G1):
```

Transformation der S-Matrix und der M-Matrix von Stab 2 in globales Koordinatensystem

```
> K2:=MatrixMatrixMultiply(Transpose(G2),k):
> K2:=MatrixMatrixMultiply(K2,G2):
> M2:=MatrixMatrixMultiply(Transpose(G2),m):
> M2:=MatrixMatrixMultiply(M2,G2):
```

Transformationsmatrix zur Ergaenzung auf Gesamtkoordinaten [U1,....,U6]

Einfuegen von Nullzeilen und -spalten

```
> I1:=Matrix([[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1]])
> :
> K1:=MatrixMatrixMultiply(Transpose(I1),K1):
> K1:=MatrixMatrixMultiply(K1,I1):
> M1:=MatrixMatrixMultiply(Transpose(I1),M1):
> M1:=MatrixMatrixMultiply(M1,I1):
> I2:=Matrix([[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1]])
> :
> K2:=MatrixMatrixMultiply(Transpose(I2),K2):
> K2:=MatrixMatrixMultiply(K2,I2):
> M2:=MatrixMatrixMultiply(Transpose(I2),M2):
> M2:=MatrixMatrixMultiply(M2,I2):
```

Materialkonstanten Federkonstante und Masse

```
> cK1:=A1*E/L:
> cK2:=A2*E/L2:
> cM1:=rho*A1*L/6:
> cM2:=rho*A2*L2/6:
> K1:=cK1*K1:
> K2:=cK2*K2:
> M1:=cM1*M1:
> M2:=cM2*M2:
```

Addition der "globalisierten" Matrizen

```
> K:=MatrixAdd(K1,K2):
```

K ist singulaer

```
> Rank(K):
> M:=MatrixAdd(M1,M2):
```

Extrahieren der letzten beiden Zeilen und Spalten

Knoten 1 und 3 sind fest, d.h. U1=U2=U3=U4=0

```
> I3:=Matrix([[0,0],[0,0],[0,0],[0,0],[1,0],[0,1]]):
> KT:=MatrixMatrixMultiply(Transpose(I3),K):
> KT:=MatrixMatrixMultiply(KT,I3);
```

$$KT := \left[\frac{A1\,E\cos(\theta)^2}{L} + \frac{A2\,E\cos(\theta)^2}{(\cos(\theta)^2+(1-\sin(\theta))^2)^{(3/2)}\,L},\right.$$
$$\left.-\frac{A1\,E\cos(\theta)\sin(\theta)}{L} + \frac{A2\,E\cos(\theta)\,(1-\sin(\theta))}{(\cos(\theta)^2+(1-\sin(\theta))^2)^{(3/2)}\,L}\right]$$
$$\left[-\frac{A1\,E\cos(\theta)\sin(\theta)}{L} + \frac{A2\,E\cos(\theta)\,(1-\sin(\theta))}{(\cos(\theta)^2+(1-\sin(\theta))^2)^{(3/2)}\,L},\right.$$
$$\left.\frac{A1\,E\sin(\theta)^2}{L} + \frac{A2\,E\,(1-\sin(\theta))^2}{(\cos(\theta)^2+(1-\sin(\theta))^2)^{(3/2)}\,L}\right]$$

```
> MT:=MatrixMatrixMultiply(Transpose(I3),M):
> MT:=MatrixMatrixMultiply(MT,I3);
```

$$MT := \left[\frac{1}{3}\rho\,A1\,L\cos(\theta)^2 + \frac{1}{3}\frac{\rho\,A2\,L\cos(\theta)^2}{\sqrt{\cos(\theta)^2+(1-\sin(\theta))^2}},\right.$$
$$\left.-\frac{1}{3}\rho\,A1\,L\cos(\theta)\sin(\theta) + \frac{1}{3}\frac{\rho\,A2\,L\cos(\theta)\,(1-\sin(\theta))}{\sqrt{\cos(\theta)^2+(1-\sin(\theta))^2}}\right]$$
$$\left[-\frac{1}{3}\rho\,A1\,L\cos(\theta)\sin(\theta) + \frac{1}{3}\frac{\rho\,A2\,L\cos(\theta)\,(1-\sin(\theta))}{\sqrt{\cos(\theta)^2+(1-\sin(\theta))^2}},\right.$$
$$\left.\frac{1}{3}\rho\,A1\,L\sin(\theta)^2 + \frac{1}{3}\frac{\rho\,A2\,L\,(1-\sin(\theta))^2}{\sqrt{\cos(\theta)^2+(1-\sin(\theta))^2}}\right]$$

Generalisiertes Eigenwertproblem: |M*lambda-K|=0

```
> lambda:=Eigenvalues(KT,MT);
```

$$\lambda := \begin{bmatrix} \frac{3\,E}{\rho\,L^2} \\ -\frac{3}{2}\frac{E}{\rho\,L^2\,(-1+\sin(\theta))} \end{bmatrix}$$

```
> f:=Vector([kappa*P,-P]):
```

Berechnen der statischen Auslenkung des Knoten 2 (U5,U6)

```
> U56:=MatrixVectorMultiply(MatrixInverse(KT),f):U56:=simplify(%):
```

$$\left[\frac{1}{2}L\,P(2\,A1\,\sqrt{2-2\sin(\theta)} - 2\,A1\,\sin(\theta)\,\sqrt{2-2\sin(\theta)} - 2\,A1\,\sqrt{2-2\sin(\theta)}\cos(\theta)^2\right.$$
$$+ 2\,A1\,\sqrt{2-2\sin(\theta)}\cos(\theta)^2\sin(\theta) + 2\,A2 - 2\,A2\sin(\theta) - A2\cos(\theta)^2$$
$$- 4\cos(\theta)\,A1\,\sqrt{2-2\sin(\theta)}\sin(\theta) + 4\cos(\theta)\,A1\,\sin(\theta)^2\sqrt{2-2\sin(\theta)}$$
$$\left.+ 2\cos(\theta)\,A2 - 2\cos(\theta)\,A2\sin(\theta))\Big/(E\cos(\theta)^2\,A1\,A2)\right]$$
$$\left[-\frac{1}{2}L\,P(-2\,A1\sin(\theta)\,\sqrt{2-2\sin(\theta)} + 2\,A1\sin(\theta)^2\,\sqrt{2-2\sin(\theta)} + A2 - A2\sin(\theta)\right.$$
$$+ 4\cos(\theta)\,A1\,\sqrt{2-2\sin(\theta)} - 4\cos(\theta)\,A1\,\sqrt{2-2\sin(\theta)}\sin(\theta) + 2\cos(\theta)\,A2)/(E$$
$$\left.\cos(\theta)\,A1\,A2)\right]$$

Dehnung der Staebe

```
> u2:=(U56[1]*cos(theta)-U56[2]*sin(theta))/L:
> u4:=(U56[1]*cos(phi)+U56[2]*sin(phi))/L2:
```

Spannung in den Staeben

```
> sigma1:=E*u2:
> sigma1:=simplify(sigma1);
```

$$\sigma 1 := \frac{1}{2}\,\frac{P\,(1-\sin(\theta)+2\cos(\theta))}{\cos(\theta)\,A1}$$

```
> sigma2:=E*u4:
> sigma2:=simplify(sigma2);
```

$$\sigma 2 := -\frac{1}{2}\,\frac{\sqrt{2-2\sin(\theta)}\,(2\cos(\theta)-\sin(\theta))\,P}{\cos(\theta)\,A2}$$

```
> Fortran(sigma1,optimize);

      t1 = sin(theta)

      t2 = cos(theta)

      t10 = P * (0.1D1 - t1 + 0.2D1 * t2) / t2 / A1 / 0.2D1
> Fortran(sigma2,optimize);

      t1 = sin(theta)

      t4 = cos(theta)

      t14 = -sqrt(0.2D1) * sqrt(0.1D1 - t1) / t4 * (0.2D1 * t4 - t1) * P / A2 / 0.2D1
```

Exercise 25. Write an `Ampl` model file with A_1, A_2 and θ as variables to minimize the weight of the structure under the constraints

$$\sigma_i <= \sigma_{max}, \quad i = 1, 2.$$

□

7.5.3 Three-Bar Truss

This problem is taken from Arora [1, section 2.6.8, pp. 41–45]. Calculate, using the finite element method, the displacement of node 4 for the symmetric three-bar truss shown in Fig. 7.4. Again, `Maple` is used with the corresponding worksheet to follow.

```
> restart:
```

Three bar analysis (Arora Ch. 2.6.8)

```
> with(LinearAlgebra):
> interface(labeling=false):
```

Mass and stiffness elements

```
> T:=Matrix([[2,1],[1,2]]):
> V:=Matrix([[1,-1],[-1,1]]):
```

Geometry

```
> L2:=L*sqrt(2):
> phi:=Pi/4:
```

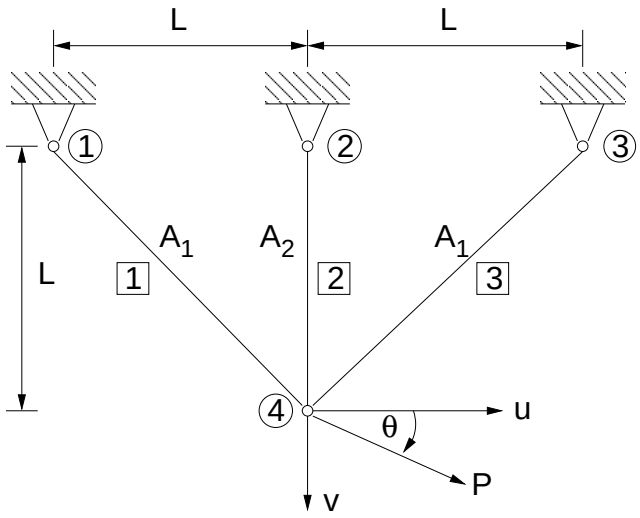


Figure 7.4: Three-Bar Truss

Description of bar elements 1..3:

element 1: u1,u2; element 2: u3,u4; element 3: u5,u6.

node 1: U1,U2; node 2: U3,U4; node 3: U5,U6; node 4: U7,U8.

```
> #c:=cos(phi);
> #s:=sin(phi);
> #u1:=U1*c-U2*s:
> #u2:=U7*c-U8*s:
> #u3:=-U4:
> #u4:=-U8:
> #u5:=-U5*c-U6*s:
> #u6:=-U7*c-U8*s:
```

Transformation matrices

```
> Gamma[1]:=Matrix([[c,-s,0,0,0,0,0,0],[0,0,0,0,0,0,c,-s]]):
> Gamma[2]:=Matrix([[0,0,0,-1,0,0,0,0],[0,0,0,0,0,0,0,-1]]):
> Gamma[3]:=Matrix([[0,0,0,0,-c,-s,0,0],[0,0,0,0,0,0,-c,-s]]):
```

Stiffness matrix K:

```
> K[1]:=MatrixMatrixMultiply(Transpose(Gamma[1]),V):
> K[1]:=MatrixMatrixMultiply(K[1],Gamma[1]):
> K1:=A1*E/L2:
> K[1]:=ScalarMultiply(K[1],K1):
> K[2]:=MatrixMatrixMultiply(Transpose(Gamma[2]),V):
> K[2]:=MatrixMatrixMultiply(K[2],Gamma[2]):
> K2:=A2*E/L:
> K[2]:=ScalarMultiply(K[2],K2):
> K[3]:=MatrixMatrixMultiply(Transpose(Gamma[3]),V):
> K[3]:=MatrixMatrixMultiply(K[3],Gamma[3]):
> K3:=A3*E/L2:
> K[3]:=ScalarMultiply(K[3],K3):
```

```
> S:=MatrixAdd(K[1],K[2]):
> S:=MatrixAdd(S,K[3]):
```

Incorporation of boundary conditions: U1..U6=0 (pinned joints)

```
> S:=SubMatrix(S,7..8,7..8):
```

Consistent mass matrix MA:

```
> M[1]:=MatrixMatrixMultiply(Transpose(Gamma[1]),T):
> M[1]:=MatrixMatrixMultiply(M[1],Gamma[1]):
> K1:=rho*A1*L2/6:
> M[1]:=ScalarMultiply(M[1],K1):
> M[2]:=MatrixMatrixMultiply(Transpose(Gamma[2]),T):
> M[2]:=MatrixMatrixMultiply(M[2],Gamma[2]):
> K2:=rho*A2*L/6:
> M[2]:=ScalarMultiply(M[2],K2):
> M[3]:=MatrixMatrixMultiply(Transpose(Gamma[3]),T):
> M[3]:=MatrixMatrixMultiply(M[3],Gamma[3]):
> K3:=rho*A3*L2/6:
> M[3]:=ScalarMultiply(M[3],K3):
> MA:=MatrixAdd(M[1],M[2]):
> MA:=MatrixAdd(MA,M[3]):
```

Boundary conditions

```
> MA:=SubMatrix(MA,7..8,7..8):
```

Eigenvalues from generalized eigenvalue problem

```
> Eigenvalues(S,MA):
> subs(A3=A1,c=cos(phi),s=sin(phi),%):
> lambda:=simplify(%);
```

$$\lambda := \begin{bmatrix} \frac{3\,E\,(\sqrt{2}\,A2 + A1)}{\rho\,L^2\,(\sqrt{2}\,A2 + 2\,A1)} \\ \frac{3\,E}{2\,\rho\,L^2} \end{bmatrix}$$

```
> P:=Vector([Pu,-Pv]):
> SI:=MatrixInverse(S):
```

Static equilibrium to calculate displacements w=(U7,U8)^T

```
> W:=MatrixVectorMultiply(SI,P):
> subs(A3=A1,c=cos(phi),s=sin(phi),%):w:=simplify(%):
> U7:=w[1];
```

$$U7 := \frac{Pu\,L\,\sqrt{2}}{E\,A1}$$

```
> U8:=w[2];
```

$$U8 := -\frac{2\,L\,Pv}{E\,(A1\,\sqrt{2} + 2\,A2)}$$

Calculate the stresses in the bars

```
> c:=cos(phi):s:=sin(phi):
```

```
> u1:=U1*c-U2*s:u2:=U7*c-U8*s;
```

$$u2 := \frac{Pu\,\mathrm{L}}{\mathrm{E}\,A1} + \frac{\mathrm{L}\,Pv\,\sqrt{2}}{\mathrm{E}\,(A1\,\sqrt{2}+2\,A2)}$$

```
> u3:=-U4:u4:=-U8;
```

$$u4 := \frac{2\,\mathrm{L}\,Pv}{\mathrm{E}\,(A1\,\sqrt{2}+2\,A2)}$$

```
> u5:=-U5*c-U6*s:u6:=-U7*c-U8*s;
```

$$u6 := -\frac{Pu\,\mathrm{L}}{\mathrm{E}\,A1} + \frac{\mathrm{L}\,Pv\,\sqrt{2}}{\mathrm{E}\,(A1\,\sqrt{2}+2\,A2)}$$

```
> sigma1:=E*u2/L2:sigma1:=simplify(%);
```

$$\sigma 1 := \frac{(2\,Pu\,A2 + Pu\,A1\,\sqrt{2} + Pv\,\sqrt{2}\,A1)\,\sqrt{2}}{2\,A1\,(A1\,\sqrt{2}+2\,A2)}$$

```
> sigma2:=E*u4/L;
```

$$\sigma 2 := \frac{2\,Pv}{A1\,\sqrt{2}+2\,A2}$$

```
> sigma3:=E*u6/L2:sigma3:=simplify(%);
```

$$\sigma 3 := -\frac{(2\,Pu\,A2 + Pu\,A1\,\sqrt{2} - Pv\,\sqrt{2}\,A1)\,\sqrt{2}}{2\,A1\,(A1\,\sqrt{2}+2\,A2)}$$

With this information we can pose an engineering optimization problem:

- minimize the weight of the structure
- subject to structural constraints:
  - geometric constraints,
  - deflection constraints,
  - stress constraints,
  - buckling constraints [21, Chap. 11],
  - natural frequency constraints,

or in mathematical setting

$$
\begin{aligned}
\min_{(A_1,A_2)} \quad & f = \rho L(2\sqrt{2}A_1 + A_2), \\
\text{s.t.} \quad & c_1: \ A_1 >= 0, \\
& c_2: \ A_2 >= 0, \\
& c_3: \ \frac{\sqrt{2}LP_u}{A_1 E} \le \Delta_u, \\
& c_4: \ \frac{2LP_v}{(\sqrt{2}A_1 + 2A_2)E} \le \Delta_v, \\
& c_5: \ \frac{\sqrt{2}[(\sqrt{2}A_1 + 2A_2)P_u + \sqrt{2}A_1 P_v]}{2A_1(\sqrt{2}A_1 + 2A_2)} \le \sigma_{max}, \\
& c_6: \ \frac{2P_v}{\sqrt{2}A_1 + 2A_2} \le \sigma_{max}, \\
& c_7: \ -\frac{\sqrt{2}[(\sqrt{2}A_1 + 2A_2)P_u + \sqrt{2}A_1 P_v]}{2A_1(\sqrt{2}A_1 + 2A_2)} \le \frac{\pi^2 E\beta A_1}{2L^2}, \\
& c_8: \ -\frac{2P_v}{\sqrt{2}A_1 + 2A_2} \le \frac{\pi^2 E\beta A_2}{L^2}, \\
& c_9: \ -\frac{\sqrt{2}[(\sqrt{2}A_1 + 2A_2)P_u - \sqrt{2}A_1 P_v]}{2A_1(\sqrt{2}A_1 + 2A_2)} \le \frac{\pi^2 E\beta A_1}{2L^2}, \\
& c_{10}: \ \frac{3E(A_1 + \sqrt{2}A_2)}{\rho L^2(2A_1 + \sqrt{2}A_2)} \ge 4\pi^2\omega^2.
\end{aligned}
$$

Exercise 26. Write a model file for `Ampl` for solving the above problem with the following data: $L = 1$ m, $P = 50$ kN, $\rho = 7850$ kg/m^3, $E = 210$ GPa, $\sigma_{max} = 150$ MPa, $\Delta_u = \Delta_v = 2e\text{-}2$ m, $\theta = 30°$, $\beta = 1/4$, $\omega = 1500$ rpm. □

7.5.4 A Bridge Structure

A bridge as shown in Fig. 7.5 is made up of 5 nodes and 7 bars is loaded with a load $P = 100$ kN at node 3. The span of each field is $L = 10$ m and the height is $H = 12$ m. Each bar has a cross section $A = 4 \times 10^{-4}$ m^2. The material data are $\rho = 7850$ kg/m^3 and $E = 210$ GPa.

Calculate the displacements of the nodes and the eigenvalues of the structure.

Here is the corresponding `Maple` worksheet.

Bridge Structure

```
> restart:
> interface(labeling=false):
> with(LinearAlgebra):
> d:=4/10000:
> phi:=arctan(H,L/2):
> c:=cos(phi):
> s:=sin(phi):
> L2:=sqrt(H^2+L^2/4):
> k1:=A1*E/L2:
> k2:=A2*E/L:
> k3:=A3*E/L2:
```

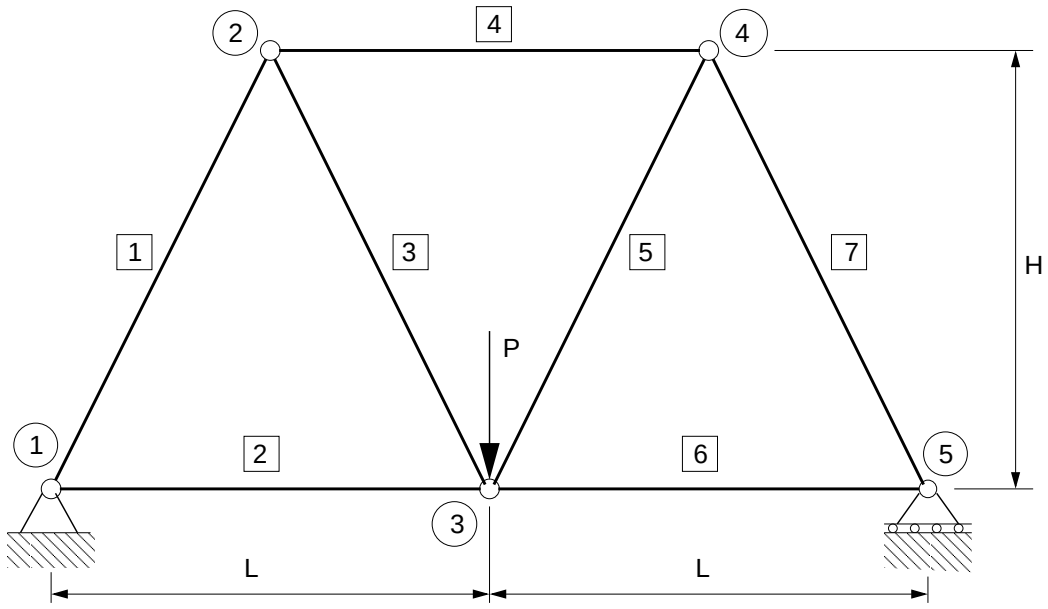


Figure 7.5: A Bridge Structure

```
> k4:=A4*E/L:
> k5:=A5*E/L2:
> k6:=A6*E/L:
> k7:=A7*E/L2:
> m1:=rho*A1*L2/6:
> m2:=rho*A2*L/6:
> m3:=rho*A3*L2/6:
> m4:=rho*A4*L/6:
> m5:=rho*A5*L2/6:
> m6:=rho*A6*L/6:
> m7:=rho*A7*L2/6:
> T:=Matrix([[2,1],[1,2]]):
> V:=Matrix([[1,-1],[-1,1]]):
> Gamma[1]:=Matrix([[c,s,0,0,0,0,0,0,0,0],[0,0,c,s,0,0,0,0,0,0]]):
> Gamma[2]:=Matrix([[1,0,0,0,0,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0]]):
> Gamma[3]:=Matrix([[0,0,c,-s,0,0,0,0,0,0],[0,0,0,0,c,-s,0,0,0,0]]):
> Gamma[4]:=Matrix([[0,0,1,0,0,0,0,0,0,0],[0,0,0,0,0,0,1,0,0,0]]):
> Gamma[5]:=Matrix([[0,0,0,0,c,s,0,0,0,0],[0,0,0,0,0,0,c,s,0,0]]):
> Gamma[6]:=Matrix([[0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,0,1,0]]):
> Gamma[7]:=Matrix([[0,0,0,0,0,0,c,-s,0,0],[0,0,0,0,0,0,0,0,c,-s]]):
> K[1]:=MatrixMatrixMultiply(Transpose(Gamma[1]),V):
> K[2]:=MatrixMatrixMultiply(Transpose(Gamma[2]),V):
> K[3]:=MatrixMatrixMultiply(Transpose(Gamma[3]),V):
> K[4]:=MatrixMatrixMultiply(Transpose(Gamma[4]),V):
> K[5]:=MatrixMatrixMultiply(Transpose(Gamma[5]),V):
> K[6]:=MatrixMatrixMultiply(Transpose(Gamma[6]),V):
> K[7]:=MatrixMatrixMultiply(Transpose(Gamma[7]),V):
```

```
> K[1]:=MatrixMatrixMultiply(K[1],Gamma[1]):
> K[2]:=MatrixMatrixMultiply(K[2],Gamma[2]):
> K[3]:=MatrixMatrixMultiply(K[3],Gamma[3]):
> K[4]:=MatrixMatrixMultiply(K[4],Gamma[4]):
> K[5]:=MatrixMatrixMultiply(K[5],Gamma[5]):
> K[6]:=MatrixMatrixMultiply(K[6],Gamma[6]):
> K[7]:=MatrixMatrixMultiply(K[7],Gamma[7]):
> K[1]:=k1*K[1]:
> K[2]:=k2*K[2]:
> K[3]:=k3*K[3]:
> K[4]:=k4*K[4]:
> K[5]:=k5*K[5]:
> K[6]:=k6*K[6]:
> K[7]:=k7*K[7]:
> k:=MatrixAdd(K[1],K[2]):
> k:=MatrixAdd(k,K[3]):
> k:=MatrixAdd(k,K[4]):
> k:=MatrixAdd(k,K[5]):
> k:=MatrixAdd(k,K[6]):
> k:=MatrixAdd(k,K[7]):
> k:=SubMatrix(k,3..9,3..9):
> M[1]:=MatrixMatrixMultiply(Transpose(Gamma[1]),T):
> M[2]:=MatrixMatrixMultiply(Transpose(Gamma[2]),T):
> M[3]:=MatrixMatrixMultiply(Transpose(Gamma[3]),T):
> M[4]:=MatrixMatrixMultiply(Transpose(Gamma[4]),T):
> M[5]:=MatrixMatrixMultiply(Transpose(Gamma[5]),T):
> M[6]:=MatrixMatrixMultiply(Transpose(Gamma[6]),T):
> M[7]:=MatrixMatrixMultiply(Transpose(Gamma[7]),T):
> M[1]:=MatrixMatrixMultiply(M[1],Gamma[1]):
> M[2]:=MatrixMatrixMultiply(M[2],Gamma[2]):
> M[3]:=MatrixMatrixMultiply(M[3],Gamma[3]):
> M[4]:=MatrixMatrixMultiply(M[4],Gamma[4]):
> M[5]:=MatrixMatrixMultiply(M[5],Gamma[5]):
> M[6]:=MatrixMatrixMultiply(M[6],Gamma[6]):
> M[7]:=MatrixMatrixMultiply(M[7],Gamma[7]):
> M[1]:=m1*M[1]:
> M[2]:=m2*M[2]:
> M[3]:=m3*M[3]:
> M[4]:=m4*M[4]:
> M[5]:=m5*M[5]:
> M[6]:=m6*M[6]:
> M[7]:=m7*M[7]:
> m:=MatrixAdd(M[1],M[2]):
> m:=MatrixAdd(m,M[3]):
```

```
> m:=MatrixAdd(m,M[4]):
> m:=MatrixAdd(m,M[5]):
> m:=MatrixAdd(m,M[6]):
> m:=MatrixAdd(m,M[7]):
> m:=SubMatrix(m,3..9,3..9):
> ks:=subs(A1=d,A2=d,A3=d,A4=d,A5=d,A6=d,A7=d,L=10,H=12,E=210000000000,k):
> ms:=subs(A1=d,A2=d,A3=d,A4=d,A5=d,A6=d,A7=d,L=10,H=12,rho=7850,m):
> lambda:=Eigenvalues(ks,ms):
> f:=Vector[column]([0,0,0,-100000,0,0,0]):
> U:=MatrixVectorMultiply(MatrixInverse(ks),f):
> evalf(U);
```

$$\begin{bmatrix} 0.004960317460 \\ -0.01114831349 \\ 0.002480158730 \\ -0.02126322751 \\ 0. \\ -0.01114831349 \\ 0.004960317460 \end{bmatrix}$$

```
> evalf(lambda);
```

$$\begin{bmatrix} 56525.94282 \\ 90837.13866 \\ 251667.7140 \\ 420600.4562 \\ 0.1073292770\,10^{7} \\ 0.2055333936\,10^{7} \\ 0.2536942181\,10^{7} \end{bmatrix}$$

Exercise 27. Now, assume that each cross section $A_1, \ldots, A_7$ is a variable. Minimize the weight of the bridge under the constraints $\sigma_1, \ldots, \sigma_7 \leq \sigma_{\max} = 100$ MPa. □

Chapter 8

Optimal Control

In engineering applications there often exists the need to solve the following optimal control problem (OCP):

> Find control functions $\mathbf{u}(t)$
> to minimize the cost function

$$J = \phi(\mathbf{y}(t_f), t_f), \tag{8.1}$$

subject to differential constraints

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t), t), \tag{8.2}$$

and algebraic constraints

$$\mathbf{g}(\mathbf{y}(t), \mathbf{u}(t), t) \geq 0, \tag{8.3}$$

and boundary conditions[1]

$$\begin{aligned} \mathbf{h_1}(\mathbf{y}(t_0), \mathbf{u}(t_0), t_0) &= 0, \\ \mathbf{h_2}(\mathbf{y}(t_f), \mathbf{u}(t_f), t_f) &= 0. \end{aligned} \tag{8.4}$$

The infinite-dimensional problem (OCP) can generally not be solved exactly on a computer [9]. Therefore (OCP) is approximated by a discrete version (DOCP) on a mesh $0, \ldots, N$. This can be accomplished in a number of different ways, e.g. finite difference methods, collocation methods, and finite element methods [2].

Define, for $k = 0, \ldots, N$, a mesh

$$\Delta := t_0 < t_1 < \cdots < t_{N-1} < t_N = t_f, \tag{8.5}$$

with uniform mesh size

$$h := \frac{t_f - t_0}{N}, \tag{8.6}$$

and grid points

$$t_{k+1} = t_k + h, \quad k = 0, \ldots, N-1, \tag{8.7}$$

and let

$$\mathbf{y}_k := \mathbf{y}(t_k), \quad \mathbf{u}_k := \mathbf{u}(t_k), \quad \mathbf{f}_k := \mathbf{f}(\mathbf{y}(t_k), \mathbf{u}(t_k), t_k).$$

We consider two methods to transform (OCP) to (DOCP)

$$\underset{\mathbf{y},\mathbf{u},t_f}{\text{minimize}}\, \Phi(\mathbf{y}_f, \mathbf{u}_f, t_f), \tag{8.8}$$

namely finite differences and collocation[2]

[1] For the sake of simplicty we discribe seperated boundary conditions. An example with multipoint boundary conditions is presented in section 9.2.4.

[2] Direct and indirect shooting methods are described in [9] and [42].

8.1 Finite Differences

One of the simplest and most easily understandable finite difference method is the trapezoidal rule, where in addition to the minimization of (8.8) the difference equations

$$\mathbf{h}_0 := \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h}{2}(\mathbf{f}_k + \mathbf{f}_{k+1}) = \mathbf{0}, \quad k = 0, \ldots, N-1, \tag{8.9}$$

and boundary conditions

$$\mathbf{h}_1(\mathbf{y}_0, \mathbf{u}_0, t_0) = \mathbf{0}, \tag{8.10a}$$

$$\mathbf{h}_2(\mathbf{y}_f, \mathbf{u}_f, t_f) = \mathbf{0}, \tag{8.10b}$$

and state and/or control constraints

$$\mathbf{g}(\mathbf{y}_k, \mathbf{u}_k, t_k) \geq \mathbf{0}, \quad k = 0, \ldots, N-1, \tag{8.11}$$

have to be satisfied.

Remarks:
1) Eq. (8.9) can also be interpreted as collocation at Lobatto points.
2) The Jacobian $\nabla \mathbf{h}_0$ of (8.9) is a sparse matrix.
3) Eqs. (8.8–8.11) is a nonlinear program, like (1.1).

8.2 Collocation

In this case, in addition to mesh (8.5), for a collocation method of order k, collocation points

$$0 \leq \rho_1 < \cdots < \rho_k \leq 1, \tag{8.12}$$

are defined on each subinterval $[t_i, t_{i+1}]$ together with subgrid points

$$t_{ij} = t_i + \rho_j h, \quad i = 0, \ldots, N, \quad j = 1, \ldots, k. \tag{8.13}$$

now, let $\mathbf{y}_\Delta(t)$ be an interpolating polynomial of order $k+1$ defined on $[t_i, t_{i+1}]$ by the interpolation conditions

$$\mathbf{y}_i := \mathbf{y}(t_i) = \mathbf{y}_\Delta(t_i), \tag{8.14}$$

and the collocation conditions

$$\dot{\mathbf{y}}_\Delta(t_{ij}) = \mathbf{f}(\mathbf{y}_\Delta(t_{ij}), \mathbf{u}_\Delta(t_{ij}), t_{ij}), \quad i = 0, \ldots, N, \quad j = 1, \ldots, k. \tag{8.15}$$

We express, on $[t_i, t_{i+1}]$, $\mathbf{y}_\Delta(t)$ as its Taylor expansion about t_i:

$$\mathbf{y}_\Delta(t) = \mathbf{y}_i + h \sum_{j=1}^{k} \frac{(t - t_i)^2}{j! h^j} w_{ij}, \tag{8.16}$$

where w_{ij} are free parameters to be adjusted. Eq. (8.16) has to be evaluated at points t_{ij} in the collocation conditions (8.15), therefore we have

$$\mathbf{y}_\Delta(t_{ij}) = \mathbf{y}_i + h \sum_{j=1}^{k} \frac{\rho^j}{j!} w_{ij},$$

and

$$\dot{\mathbf{y}}_\Delta(t_{ij}) = \sum_{j=1}^{k} \frac{\rho^{j-1}}{(j-1)!} w_{ij}.$$

We construct from the polynomial pieces a piecewise polynomial pp which is continuous on $[t_0, t_f]$, that means we require the following continuity conditions

$$y_\Delta(t_i) = y_\Delta(t_{i+1}), \quad i = 1, \ldots, N-1. \tag{8.17}$$

Now, the free variables in the collocation method are

$$\nu = (u_\Delta(t_{ij}), y_i, w_{ij})^T, \tag{8.18}$$

together with (in free end time problems) t_f and with $\dim(\nu) = nN + kmN + knN + (1)$. We summarize the collocation method:

> Minimize wrt. ν the cost function (8.1)
> subject to
> the collocation conditions (8.15),
> the continuity conditions (8.17),
> the boundary conditions (8.10a),
> and the state constraints (8.11).

Remarks:
1) Often, the collocation points ρ_i are chosen as the zeros of the Legendre polynomials of different order. Sometimes, this is called collocation at Gaussian points [12].
2) If $k = 1$, then $\rho_1 = \frac{1}{2}$, and the method is equivalent to the mid-point rule.

Chapter 9

Optimum Design

9.1 Optimization Codes

There are great many of proposed interior point algorithms (an excellent review is presented in [19]) but only a small number of state-of-the-art implementations into efficient software. The most well-known of these are IPOPT [46], KNITRO [10], and LOQO [43]. While the latter two programs are commercial codes, the first one is published as open source code under the Common Public License (CPL) [11]. IPOPT solves nonlinear optimization problems of the following form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x), \\ \text{s.t.} \quad & c(x) = 0, \\ & x_L \leq x \leq x_U, \end{aligned} \tag{9.1}$$

where inequality constraints, $c_i(x) \geq 0$, can be reformulated in the above format by introducing slack variables. For comparison, we include into our tests SNOPT [23], an algorithm of the active-set SQP type. We perform our tests in the AMPL environment [20], a modeling program for mathematical programming.

Before detailed results of some engineering applications will be presented the efficiency of the codes tested in the COPS benchmarking set [14] is evaluated in the following two tables.

In Table 9.1 the average time ranking for all three parameter sets of each example as taken from the respective Table Ex.2 is listed for each solver row-wise. The column-sum of these rankings is divided by the number of examples. The overall ranking shows the advantage of the interior point algorithms.

In Table 9.2 the test of this report on the COPS set are summarized. I only could include IPOPT 2.2.1c, LOQO 6.06 and SNOPT 6.2-1 in this comparison as I only have full unrestricted licenses for these solvers. Therefore, also in the analyze phase of the calculations MINOS was replaced by SNOPT. Shown are the the values of ampl's _total_solve_time in seconds for each example, parameter set and solver, respectively. The result indicated by **** means that the solver timed out (_total_solve_time>1800 seconds) or could not solve that problem. Detailled results can be obtained by the author. The ranking in the last line is performed as in Table 5.1. All computations were performed on an Intel Pentium 4 XEON 2GHz CPU with 1024GB RAM and a 512KB cache, running Linux Fedora Core 2. The default values of the solver parameters have been chosen. Again, the superiority of interior point solvers is obvious, as also has been pointed out by GOULD, recently [25].

No	Name	FILTER	KNITRO	LOQO	MINOS	SNOPT
1	polygon	5	1	4	3	2
2	elec	5	1	4	3	2
3	camshape	2	5	3	1	4
4	chain	3	1	2	3	3
5	pinene	5	1	2	3	4
6	marine	4	1	2	5	3
7	channel	5	1	2	3	4
8	robot	4	1	5	3	2
9	steering	3	1	5	4	2
10	rocket	4	3	1	5	2
11	glider	2	1	3	5	4
12	gasoil	5	2	1	3	4
13	methanol	5	2	1	3	4
14	catalyst	5	2	1	3	4
15	torsion	4	5	1	3	2
16	bearing	5	4	1	3	2
17	minsurf	2	4	1	3	5
18	triangle	2	1	4	5	3
19	tetra	3	1	2	5	4
20	lane_emden	5	1	2	3	4
21	dirichlet	3	2	1	4	5
22	henon	3	1	2	4	5
	$\sum/22$	3.82	1.91	2.27	3.50	3.36

Table 9.1: Code Ranking in the COPS Report

	Code	IPOPT			LOQO			SNOPT		
No	Name	par1	par2	par3	par1	par2	par3	par1	par2	par3
1	polygon	1.05	6.96	25.57	2.05	5.63	8.40	3.53	30.17	586.17
2	elec	1.11	20.89	114.13	1.00	13.35	200.28	0.79	4.82	50.57
3	camshape	1.19	1.49	2.02	77.26	102.99	3.32	3.78	6.25	8.76
4	chain	0.13	0.23	12.10	0.35	0.68	1.45	4.50	20.57	65.54
5	pinene	0.90	1.91	3.59	0.90	1.99	4.45	6.68	19.87	122.52
6	marine	0.50	1.05	2.24	0.71	1.64	4.49	3.89	10.19	33.48
7	channel	0.37	0.72	1.43	1.03	2.50	6.97	3.90	13.46	44.84
8	robot	0.75	1.81	4.50	****	****	****	3.25	15.42	137.77
9	steering	1.41	17.75	1.11	****	****	****	5.71	27.99	166.38
10	rocket	1.18	2.37	5.28	0.77	1.62	3.65	11.76	21.33	51.76
11	glider	4.77	25.25	102.48	****	5.49	****	8.18	****	****
12	gasoil	0.70	1.27	2.14	0.36	0.80	2.51	1.84	6.17	22.04
13	methanol	0.66	1.64	3.70	0.70	1.65	4.68	4.06	13.02	53.14
14	catalyst	0.36	0.70	1.46	0.43	1.03	2.97	10.93	36.50	88.52
15	torsion	0.87	1.45	1.88	0.74	1.11	1.52	7.64	23.73	52.94
16	bearing	0.69	1.03	1.58	0.59	0.91	1.23	6.87	19.58	42.96
17	minsurf	2.74	6.00	8.57	2.33	3.42	****	918.21	****	****
18	triangle	0.96	0.64	2.72	****	****	52.32	494.08	111.18	****
19	tetra	****	3.07	6.01	4.63	2.50	6.46	77.72	44.42	163.15
20	lane_emden	56.37	220.79	577.41	83.98	634.70	1572.53	735.66	****	****
21	dirichlet	95.15	241.69	1232.27	266.95	715.85	1350.49	****	****	****
22	henon	104.86	118.49	****	135.37	175.19	577.19	****	****	****
	Rank		1.36			1.91			2.73	

Table 9.2: My Cops Tests

9.2 Examples

Most of our test examples, all of which are dynamic optimization problems, are included in the COPS benchmarking set.[1]

9.2.1 Sounding Rocket

The problem of maximizing the altitude of a sounding rocket was first posed by R. H. Goddard in 1919 [7]. With state variables $(v(t),\ h(t),\ m(t))$, velocity, altitude, and mass, respectively, and control variable $T(t)$, the rocket thrust, the problem can be formulated as follows

$$\begin{aligned}
&\max_{T(t)} \quad h(t_f), \\
\text{s.t.} \quad &\dot{v}(t) = \frac{T(t) - D(v(t), h(t))}{m(t)} - g(h(t)), && v(t_0) = 0, \\
&\dot{h}(t) = v(t), && h(t_0) = h_0, \\
&\dot{m}(t) = -\frac{T(t)}{c}, && m(t_0) = m_0, \quad m(t_f) = c_m m(t_0), \\
&0 \le T(t) \le T_{max},
\end{aligned} \tag{9.2}$$

where h_0 is the radius of the earth, $0 < c_m < 1$ is a fraction, and c is the (constant) specific impulse of the rocket. Note, that in Eq. (9.2) the last equation represents a control constraint. The drag D depends on the velocity and altitude

$$D = c_D v^2 \exp\Big(-c_h \frac{h(t) - h_0}{h_0}\Big),$$

where c_D and c_h are constants, and g the gravitational force per unit mass varies with altitude

$$g = g_0 \Big(\frac{h_0}{h(t)}\Big)^2,$$

with g_0 the gravitational force per unit mass on the earth's surface. We apply the following data: $T_{max} = \frac{7}{2} m_0 g_0$, $c_D = \frac{1}{2} c_v m_0 / g_0$, $c = \frac{1}{2}\sqrt{g_0 h_0}$, $c_h = 500$, $c_m = \frac{3}{5}$, $c_v = 620$, and we normalize without loss of generality $g_0 = h_0 = m_0 = 1$.

To solve the optimal control problem (9.2) we apply the $O(h^2)$ trapezoidal discretization, i.e. for $\dot{z}(t) = f(z(t), u(t))$ the state $z(t)$ is approximated by the difference equation

$$z[i+1] = z[i] + \frac{1}{2} h (f(z[i], u[i]) + f(z[i+1], u[i+1])), \tag{9.3}$$

on a uniform mesh $i = 0 \ldots N$ with steplength $h = (t_f - t_0)/N$.

All three solvers solve the problem. The timing results are shown in line 10 of Table 9.2. Both the interior-point codes are more efficient than the active-set code by a factor of almost ten. The states and optimal control are presented in Fig. 9.1 and Fig. 9.2. The bang-singular-bang character of the thrust can be realized.

[1] The model and data files of those examples not included in the COPS benchmarking set can be downloaded from http://www-neos.mcs.anl.gov/neos/solvers/NCO:IPOPT/solver-sample.html

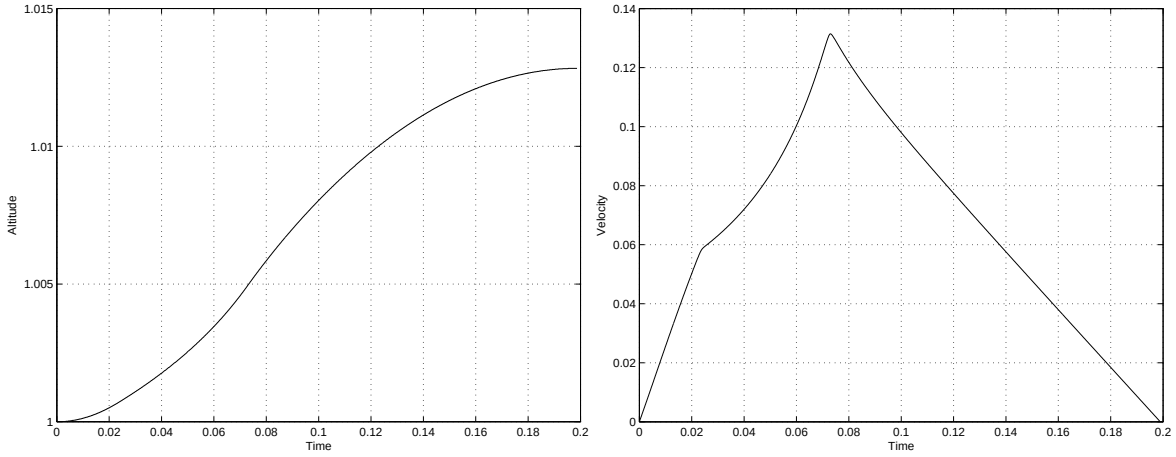


Figure 9.1: Altitude and Velocity of the Goddard Rocket Problem

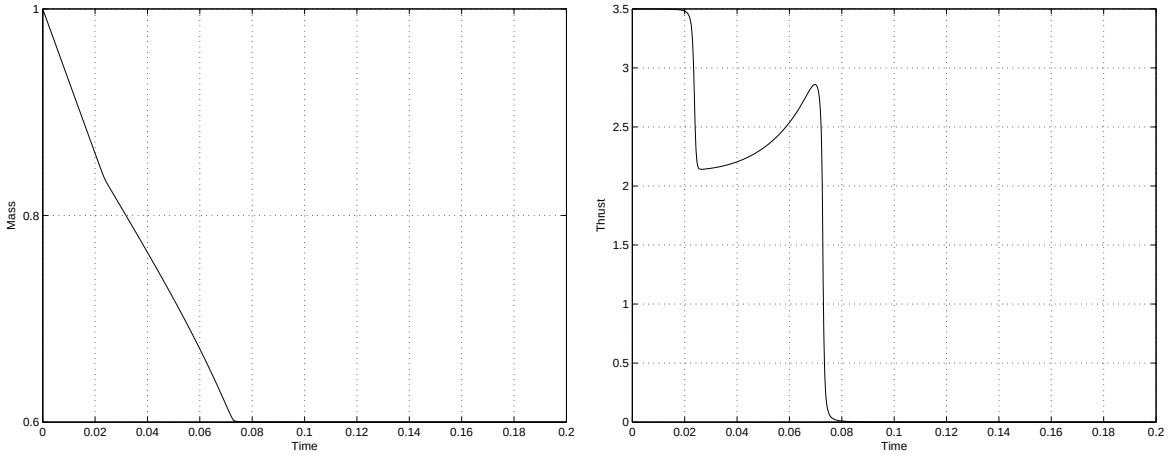


Figure 9.2: Mass and Thrust of the Goddard Rocket Problem

9.2.2 A Toy Car

A simplified description of the motion of a car in a horizontal plane is given by the following equations [3, pages 97–98]

$$\begin{aligned}
\dot{x} &= v\cos\theta, \quad & x_{min} \le x(t) \le x_{max}, \quad & t_0 < t < t_f, \quad & x(t_0) = x_0, \quad & x(t_f) = x_f, \\
\dot{y} &= v\sin\theta, \quad & y_{min} \le y(t) \le y_{max}, \quad & t_0 < t < t_f, \quad & y(t_0) = y_0, \quad & y(t_f) = y_f, \\
\dot{\theta} &= \frac{v}{L}\tan\psi, \quad & \theta_{min} \le \theta(t) \le \theta_{max}, \quad & t_0 < t < t_f, \quad & \theta(t_0) = \theta_0, \quad & \theta(t_f) = \theta_f, \\
\dot{v} &= a - \mu v, \quad & v_{min} \le v(t) \le v_{max}, \quad & t_0 < t < t_f, \quad & v(t_0) = v_0, \quad & v(t_f) = v_f,
\end{aligned} \tag{9.4}$$

where x and y are the coordinates of the car, θ is its heading angle measured positive counter-clockwise from the x-axis, v is the velocity of the car, $L = 11.5$ the distance between the fore and rear axle. The car is controlled by the its accelearation a, and the steering angle ψ which the front wheels make with the car's body. The friction of the car is described by the damping factor $\mu = 1/6$.

The problem is to satisfy Eqs. (9.4) and to

$$\begin{aligned}
&\min \quad (t_f - t_0), \\
\text{s.t.} \quad &a_{min} \le a(t) \le a_{max}, \\
&\psi_{min} \le \psi(t) \le \psi_{max}.
\end{aligned} \tag{9.5}$$

Again, we use trapezoidal discretization (9.3) to approximate the differential equations (9.6) on a grid with $n = 1000$ intervals with the following initial guess for $i = 0, \ldots, n$:

$$\begin{aligned}
t_{f,0} &= 1.0, \\
x_{i,0} &= x_f i/n, \\
y_{i,0} &= \begin{cases} 20i/n & \text{if } i < n/2 \\ 20(1 - i/n) & \text{if } i \ge n/2 \end{cases}, \\
\theta_{i,0} &= (1 - 2i/n)\frac{\pi}{2}, \\
v_{i,0} &= \begin{cases} 270i/n & \text{if } i < n/2 \\ 270(1 - i/n) & \text{if } i \ge n/2 \end{cases}, \\
a_{i,0} &= \begin{cases} a_{max} & \text{if } i < n/2 \\ a_{min} & \text{if } i \ge n/2 \end{cases}, \\
\psi_{i,0} &= \frac{1}{2}\psi_{min},
\end{aligned}$$

and the data summarized in Table 9.3.

	min	max	0	f
x	0	55	0	50
y	0	7	0	0
θ	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$
v	0	150	0	0
a	-300	300	—	—
ψ	$-\frac{\pi}{3}$	$\frac{\pi}{3}$	—	—

Table 9.3: Data for the Toy Car

The results are shown in Table 9.4. Both the interior point methods are faster than the active-set methods by a factor of more than ten. Graphs of the states and controls are given

Algorithm	Cost	Iterations	CPU
IPOPT	0.8766	56	7.11
LOQO	0.8766	78	10.51
MINOS	0.8767	6710	125.73
SNOPT	0.8799	21966	214.47

Table 9.4: Results for the Toy Car

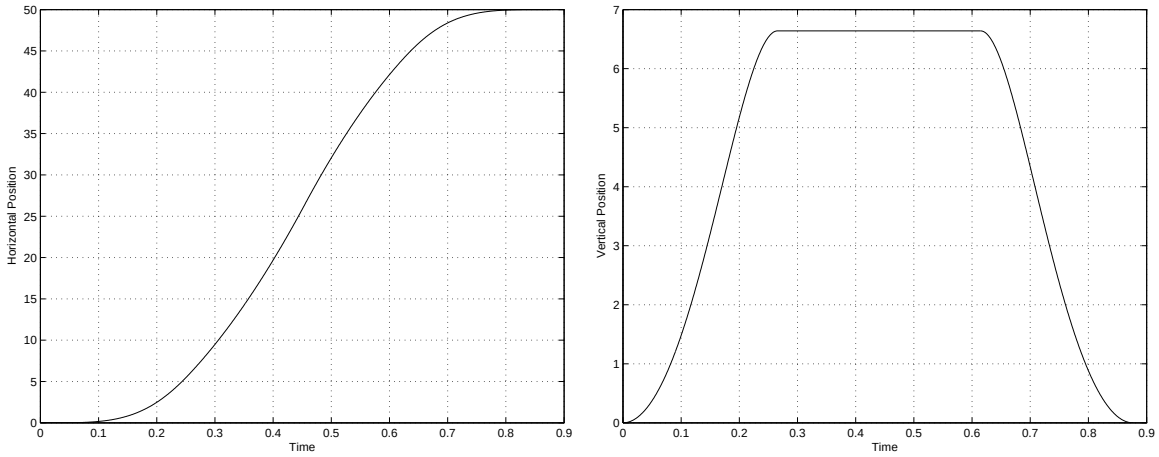


Figure 9.3: Horizontal and Vertical Position of the Toy Car

in Figs. 9.3 to 9.5. Both acceleration and steering angle are of bang-bang type. Note that none of the state constraints are active at the solution. They have been introduced to let LOQO, MINOS, and SNOPT find a solution, all of which outrun a time theshold of 1800 seconds without the state constraints.

9.2.3 Truck and Trailer

A single-axle trailer with length l is pivoted about the center point of the rear axle of a truck. Let ψ be the angle between the centerline of the truck and a reference axis x, α the angle between trailer and truck centerlines, and (x, y) the coordinates of the center point of the rear axle of the truck. The control variable is $u = (b/l)\tan(\delta)$, where δ is the steering angle of the truck front wheels and b the distance of the truck axles. The independent variable is

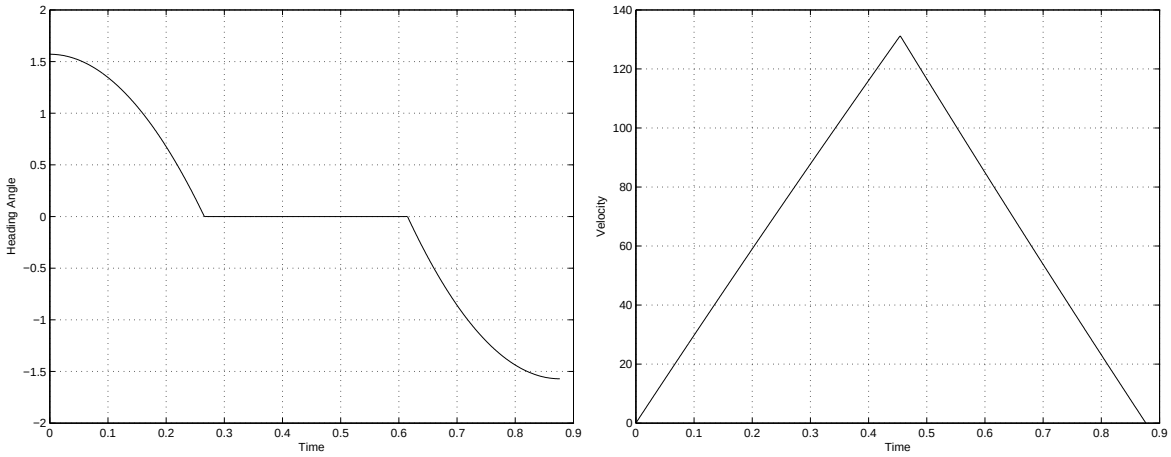


Figure 9.4: Heading Angle and Velocity of the Toy Car

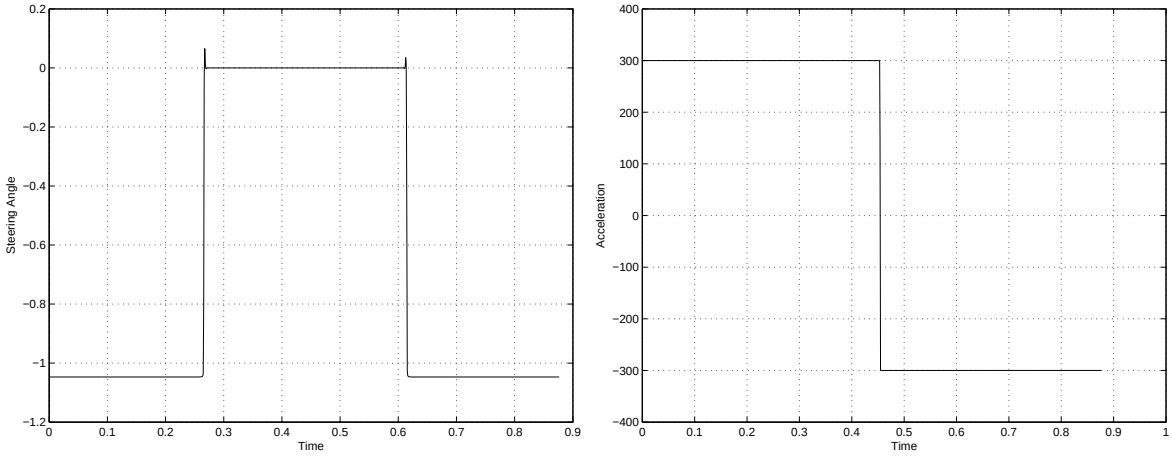


Figure 9.5: Steering Angle and Acceleration of the Toy Car

d the distance traveled by the center point of the truck rear axle, the variable (d, x, y) are normalized to the length l.

The equations of motion are described in [7, pages 397–400]:

$$\begin{aligned} \psi' &= u, & \psi(0) &= 0, & \psi(f) &= -3\pi/4, \\ \alpha' &= u - \sin(\alpha), & \alpha(0) &= 0, & \alpha(f) &= 0, \\ x' &= \cos(\psi), & x(0) &= 0, & x(f) &= 1.3, \\ y' &= \sin(\psi), & y(0) &= 0, & y(f) &= -0.3. \end{aligned} \tag{9.6}$$

The problem is to satisfy Eqs. (9.6) and to

$$\begin{aligned} &\max_{u(t)} \quad d_f, \\ &\text{s.t.} \quad u_{min} \leq u(t) \leq u_{max}. \end{aligned} \tag{9.7}$$

Again, we use trapezoidal discretization (9.3) to approximate the differential equations (9.6) on a grid with $n = 1000$ intervals with the following initial guess for $i = 0, \ldots, n$:

$$\begin{aligned} d_0 &= 1.0, \\ \psi_{i,0} &= 0.0, \\ \alpha_{i,0} &= 0.0, \\ x_{i,0} &= i/n, \\ y_{i,0} &= i/n, \\ u_{i,0} &= (1 - 2i/n)u_{max}. \end{aligned}$$

Only IPOPT is able to solve this problem within 196 iterations and _total_solve_time $= 17.22$ with final value of objective function $d_f = 5.899480$. Fig. 9.6 shows the control and the position of the truck.

Both LOQO and SNOPT stop "far from the solution" with iteration limit 10000 exceeded and with superbasics limit too small at 5050 iterations, respectively.

9.2.4 Ship Cruise

The famous Zermelo problem of a ship cruising with velocity v in a position dependent flow field [6, page 67 ff] has been modified by Erb [16]. The dynamic model of the ship with

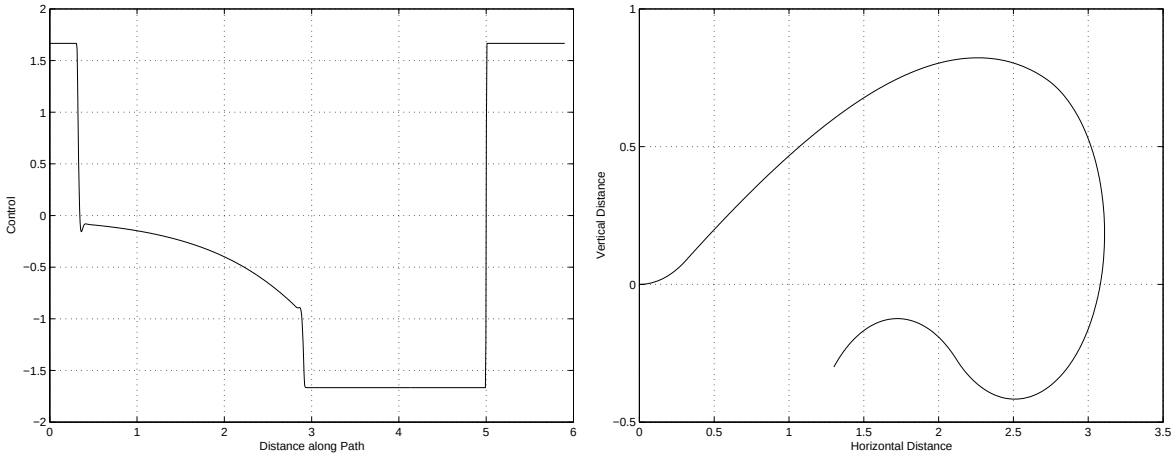


Figure 9.6: Control and Position of the Truck and Trailer Problem

coordinates x and y is given by

$$\begin{aligned} \dot{x} &= v\cos\theta + w, \\ \dot{y} &= v\sin\theta, \end{aligned} \tag{9.8}$$

where θ is the heading angle of the ship and $w = a\frac{y}{h}$ the velocity component of the current. We define a rudder angle ζ and assume that it controls the velocity by the following law

$$v = v_0(1 - \left(\frac{2\zeta}{\pi}\right)^2), \tag{9.9}$$

whereas it changes the heading angle by the following rule

$$\dot{\theta} = \sin(b\zeta). \tag{9.10}$$

Also we assume that there is a linear dependency between actuating the rudder and the rudder position

$$\dot{\zeta} = u. \tag{9.11}$$

Equations (9.8), (9.10), and (9.11) define the model of the vessel which starts at $x(t_0) = x_0,\ y(t_0) = y_0$ and shall arrive at $x(t_f) = x_f,\ y(t_f) = y_f$ in minimum time

$$\min(t_f - t_0). \tag{9.12}$$

Now, consider that the vessel should have a rendezvous with a support ship which also starts at time t_0 at position $x_{s0},\ y_{s0}$ and moves along a line

$$\begin{aligned} x_s &= x_{s0} - ct, \\ y_s &= y_{s0} - dt. \end{aligned} \tag{9.13}$$

The encounter of the two vehicles at time t_r defines a two-phase optimal control problem:[2]

- Phase 1: $t \in [t_0, t_r]$,
- Phase 2: $t \in [t_r, t_t]$.

Of course, the final time of phase 1 equals the initial time of phase 2, and the continuity of the trajectory is enforced by the equality constraints

$$\begin{aligned} x(tr-) &= x(tr+), \\ y(tr-) &= y(tr+), \\ \theta(tr-) &= \theta(tr-), \\ \zeta(tr-) &= \zeta(tr-). \end{aligned} \tag{9.14}$$

[2]Multiple phase problems can be solved in `ampl` using the named problems construct [20, section 14.5, Pages 311 ff.]. To show this technique we enclose the source code of `zermelo.mod` in Appendix 1.

Also two state constraints[3] are defined: In phase 1 a circular coral reef centered at x_c, y_c with radius r_c should not be entered

$$(x(t) - x_c)^2 + (y(t) - y_c)^2 \geq r_c^2, \quad \forall t \in]t_0, t_r[, \tag{9.15}$$

and in phase 2 a circular algae field centered at x_a, y_a with radius r_a should not be entered

$$(x(t) - x_a)^2 + (y(t) - y_a)^2 \geq r_a^2, \quad \forall t \in]t_r, t_f[. \tag{9.16}$$

The data of the problem are $a = 0.7$, $b = 2.0$, $c = 0.0825$, $d = 0.55$, $h = 1.0$, $x_0 = 3.66$, $y_0 = -1.86$, $x_f = -3.0$, $y_f = 1.8$, $x_a = -1.2$, $y_a = 0.9$, $r_a = 1.0$, $x_c = 4.0$, $y_c = 1.1$, $r_c = 1.28$, $x_{s0} = 2.0$, $y_{s0} = 3.0$. The timing results using a trapezoidal discretization (9.3) on a uniform mesh over $n = 100$ intervals are shown in Table 9.5, where also the needed iterations for solving phase 1 and 2 respectively are presented.

solver	ipopt	knitro	lancelot	loqo	minos	npsol	snopt
time	0.49	0.44		0.59	1.41	576.79	2.65
iter1	16	17		32	386	214	901
iter2	22	20		22	504	37	834

Table 9.5: Total Solving Time

The cruise is demonstrated graphically in the $x - y$—plane in Fig. 9.7, while the control function u for the two phase is given in Fig. 9.8.

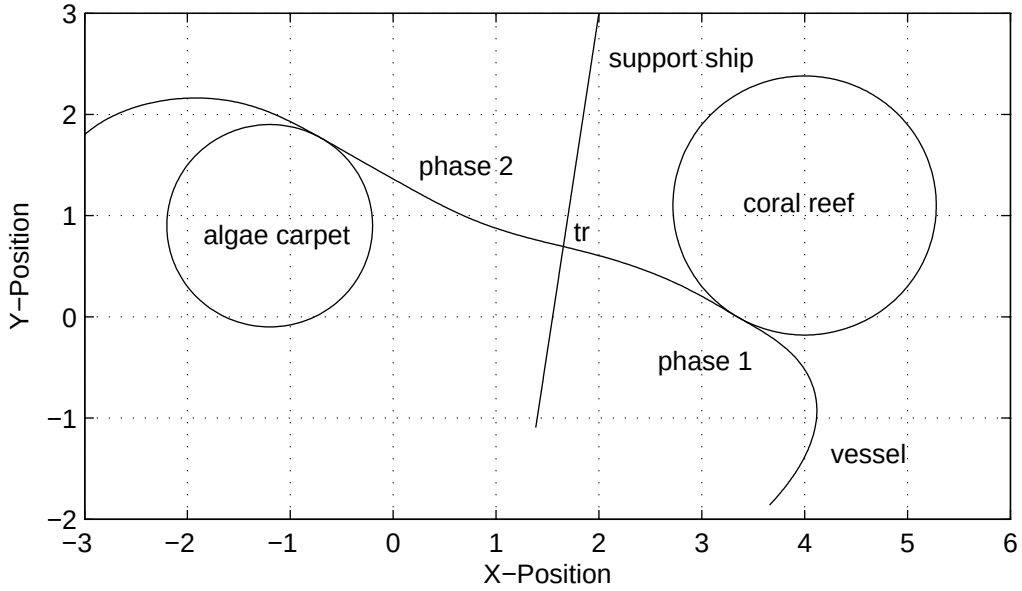


Figure 9.7: Two-phase Ship Cruise

[3]These physical constraints are augmented by box constraints on all states to alleviate the work of the minimizers; some of them cannot solve the problem because of making too large steps. These constraints can be taken from the source code which is given in Appendix 1.

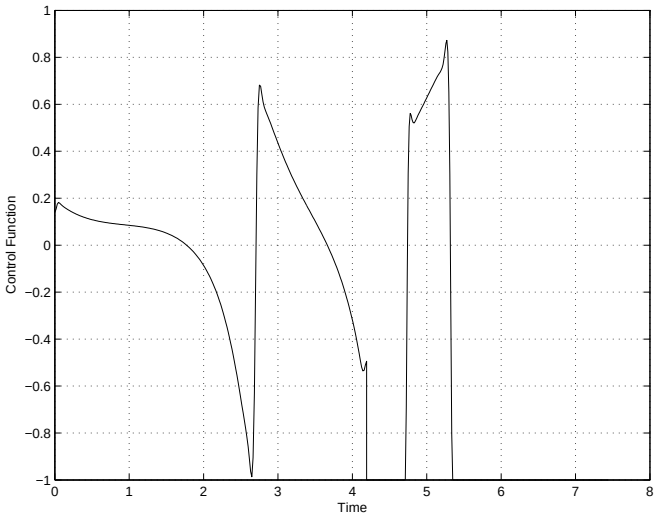


Figure 9.8: Ship Cruise: Control vs. Time

9.2.5 Robot Arm

A very much simplified model of a two-link robot arm with a spherical and a translational joint in spherical coordinates is [36]:

$$\begin{aligned} L\rho'' &= u_\rho, \quad \rho(t_0) = 4.5, \quad \rho(t_f) = 4.5, \quad \rho'(t_0) = \rho'(_{tf}) = 0, \\ J_\theta\theta'' &= u_\theta, \quad \theta(t_0) = 0, \quad \theta(t_f) = \frac{2}{3}\pi, \quad \theta'(t_0) = \theta'(_{tf}) = 0, \\ J_\phi\phi'' &= u_\phi, \quad \phi(t_0) = \frac{\pi}{4}, \quad \phi(t_f) = \frac{\pi}{4}, \quad \phi'(t_0) = \phi'(_{tf}) = 0, \end{aligned} \tag{9.17}$$

where L is the length of the base arm and

$$J_\theta = \frac{(L-\rho)^2 + \rho^3}{3}\sin(\phi)^2,$$

and

$$J_\phi = \frac{(L-\rho)^2 + \rho^3}{3},$$

are the mass moments of inertia about the θ- and ϕ-axes, respectively. The model is both state- and control-constrained:

$$\rho(t) \in [0, L], \quad |\theta(t)| \le \pi, \quad 0 \le \phi(t) \le \pi, \tag{9.18}$$

and

$$|u_\rho(t)| \le 1, \quad |u_\theta(t)| \le 1, \quad |u_\phi(t)| \le 1. \tag{9.19}$$

To enhance the operational effectiveness of the robot, a minimum-time solution of Eqs. (9.17), under the constraints (9.18) and (9.19) is searched. The second-order differential equations are transformed into a system of first-order differential equations representing positions and velocities, and these are again approximated by a trapezoidal discretization (9.3) on a uniform mesh over n intervals.

While `LOQO` is not able to solve this problem (iteration limit 10000 exceeded), `IPOPT` and `SNOPT` generate the solution $t_f = 9.140947$ with `_total_solve_time` shown in line 8 of Tab. 9.2. It ist interesting to note that the number of iterations (10654 for `SNOPT` and 26 for `IPOPT`) gives time/iteration ratio of 0.0129 and 0.1731 for the active-set and interior-point code, respectively. I.e. one iteration in the interior-point code takes more then ten times as much time then in the active-set code. This has been observed already in [25].

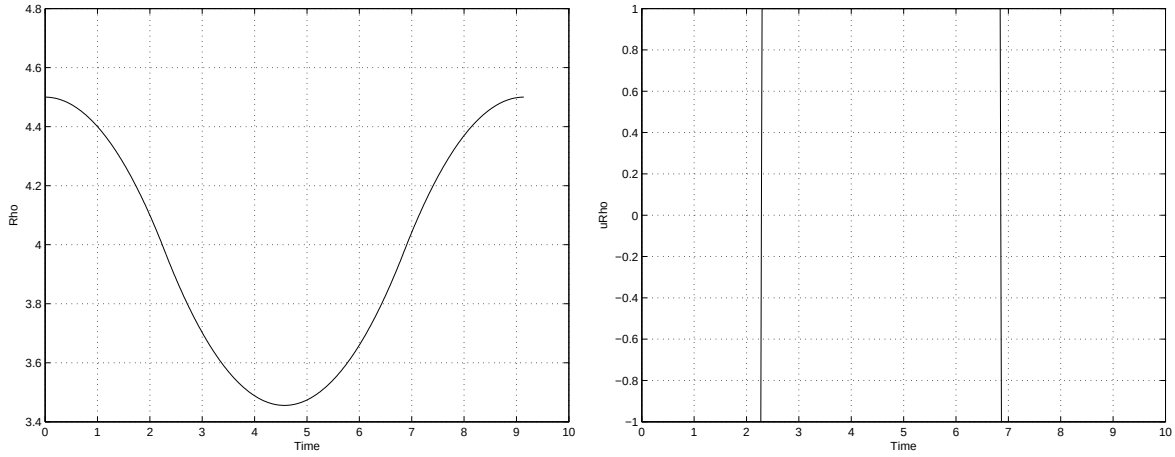


Figure 9.9: State ρ and Control u_ρ of the Robot Problem

In Fig. 9.9 the state ρ and the (bang-bang-)control u_ρ of the prismatic joint is shown, representatively.

9.2.6 Hang Glider

The problem of maximizing the range of a hang glider in the presence of a thermal upwind is described in [8]:

$$\begin{aligned}
\dot{x} &= v_x, & x(t_0) &= 0, & x(t_f) &\stackrel{!}{=} \max,\\
\dot{v}_x &= \frac{1}{m}(-L\sin(\eta) - D\cos(\eta)), & v_x(t_0) &= 13.23, & v_x(t_f) &= 13.23,\\
\dot{y} &= v_y, & y(t_0) &= 1000, & x(t_f) &= 900,\\
\dot{v}_y &= \frac{1}{m}(L\cos(\eta) - D\sin(\eta) - W), & v_y(t_0) &= -1.288, & v_x(t_f) &= 1.288,
\end{aligned} \tag{9.20}$$

with lift L, drag D, and weight W, respectively,

$$L = \frac{1}{2}c_L\rho S v_R^2, \quad D = \frac{1}{2}c_D\rho S v_R^2, \quad W = mg,$$

and flight path angle η and resultant velocity v_R

$$\eta = \arctan\left(\frac{v_y - u_A}{v_x}\right), \quad v_R = \sqrt{v_x^2 + (v_y - u_A)^2},$$

where $u_A(x)$ is the upwind profile

$$u_A(x) = u_{A_{max}} \exp\left(-\left(\frac{x}{R} - 2.5\right)^2\right)\left(1 - \left(\frac{x}{R} - 2.5\right)^2\right).$$

The glider is controlled by the constrained lift coefficient $c_L \le c_{L_{max}}$ which influences the drag by a quadratic polar

$$c_D = c_{D_0} + k c_L^2.$$

Data of the problem are collected as

$$\begin{aligned}
R &= 100[\mathrm{m}], & u_{A_{max}} &= 2.5[\mathrm{ms}^{-1}], & c_{L_{max}} &= 1.4,\\
c_{D_0} &= 0.034, & k &= 0.069662, & m &= 100[\mathrm{kg}],\\
S &= 14[\mathrm{m}^2], & g &= 9.81[\mathrm{ms}^{-2}], & \rho &= 1.13[\mathrm{kgm}^{-3}].
\end{aligned}$$

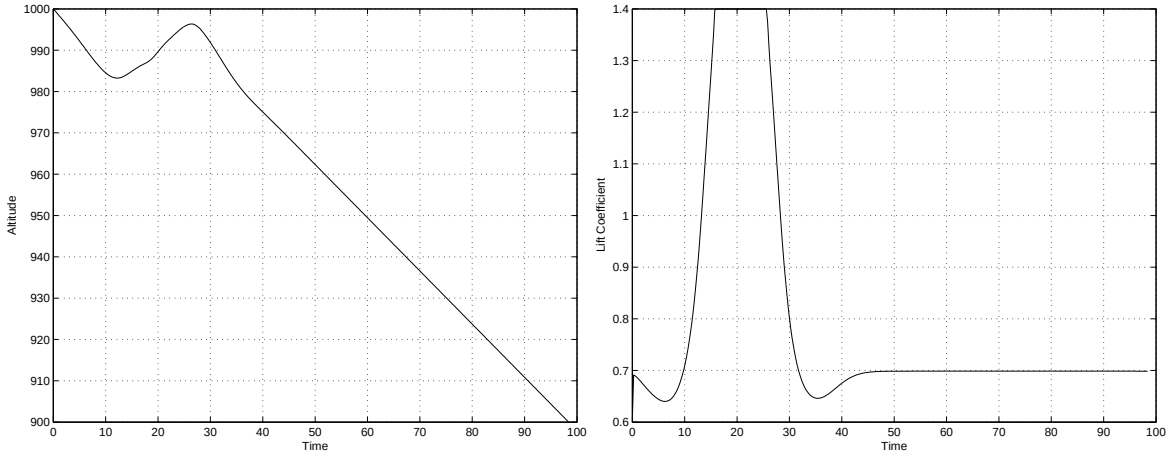


Figure 9.10: Altitude and Control c_L of the Hang Glider Problem

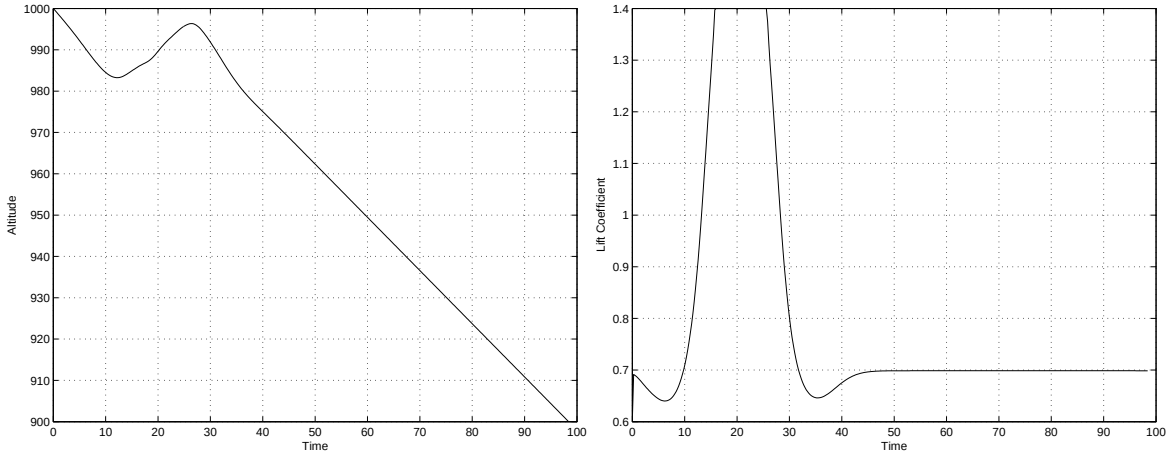


Figure 9.11: Horizontal and Vertical Velocity of the Hang Glider Problem

The results of Table 11.2 in the COPS set and those documented here are contradictory: SNOPT there solves with parameter set $n_h = 100$ and fails for both $n_h = 200$ and 400; here it fails for $n_h = 100$ and 400 while it solves for $n_h = 200$. LOQO there solves with parameter set $n_h = 200$ and fails for both $n_h = 100$ and 400; here it fails for $n_h = 200$ and 400 while it solves for $n_h = 100$. IPOPT solves all three cases as can be seen from line 11 in Table 9.2. Also, the free end-time is $t_f = 98.41$ instead of 105 as reported in [14]. Graphs of the states and the lift coefficient c_L are shown in Fig. 9.10 and Fig. 9.11.

9.2.7 Shuttle Reentry

The shuttle reentry problem is one of the classical problems in optimal control calculations [42, pages 191–197]; here we follow the model description given in [5, pages 133–138]:

$$
\begin{aligned}
&\max \theta(t_f),\\
&\text{subject to}\\
&\dot{h} = v\sin(\gamma), && h(t_0) = 260000, && h(t_f) = 80000,\\
&\dot{\theta} = \frac{v}{r}\cos(\gamma)\cos(\psi), && \theta(t_0) = 0, && \theta(t_f) = \text{free},\\
&\dot{v} = -\frac{D}{m} - g\sin(\gamma), && v(t_0) = 25600, && v(t_f) = 2500,\\
&\dot{\gamma} = \frac{L}{mv}\cos(\beta) + (\frac{v}{r} - \frac{g}{v})\cos(\gamma), && \gamma(t_0) = -1^\circ, && \gamma(t_f) = -5^\circ,\\
&\dot{\psi} = \frac{L}{mv\cos(\gamma)}\sin\beta + \frac{v}{r}\cos(\gamma)\sin(\psi)\tan(\theta), && \psi(t_0) = 0, && \psi(t_f) = \text{free},\\
&0 \le h,\\
&1 \le v,\\
&\theta_{min} \le \theta \le \theta_{max},\\
&\gamma_{min} \le \gamma \le \gamma_{max},\\
&\alpha_{min} \le \alpha \le \alpha_{max},\\
&\beta_{min} \le \beta \le \beta_{max},\\
&q = q_a q_r \le q_{max}.
\end{aligned}
\tag{9.21}
$$

The following data (all in English units, and α in degree) characterize the problem:

$$
\begin{aligned}
&a_0 = -0.20704, && a_1 = 0.029244, && b_0 = 0.07854,\\
&b_1 = -0.61592\times 10^{-2}, && b_2 = 0.621408\times 10^{-3}, && c_0 = 1.0672181,\\
&c_1 = -1.9213774\times 10^{-2}, && c_2 = 2.1286289\times 10^{-4}, && c_3 = -1.0117249\times 10^{-6},\\
&h_r = 23800, && \mu = 1.4076539\times 10^{16}, && \rho_0 = 2.378\times 10^{-3},\\
&m = 6309.44, && R_e = 2.09029\times 10^{7}, && S = 2690,\\
&D = \frac{1}{2}c_D S\rho v^2, && L = \frac{1}{2}c_L S\rho v^2, && g = \frac{\mu}{r^2}, \quad q_{max} = 70,\\
&r = R_e + h, && \rho = \rho_0\exp(-\frac{h}{h_r}), && c_L = a_0 + a_1\alpha,\\
&c_D = b_0 + b_1\alpha + b_2\alpha^2, && q_r = 17700\sqrt{\rho}(\frac{v}{10000})^{3.07}, && q_a = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3,\\
&\theta_{max} = -\theta_{min} = 89^\circ, && \gamma_{max} = -\gamma_{min} = 89^\circ, && \alpha_{max} = -\alpha_{min} = 90^\circ,\\
&\beta_{min} = -89^\circ, && \beta_{max} = 1^\circ.
\end{aligned}
$$

Again, we use trapezoidal discretization (9.3) to approximate the differential equations (9.6) with the following initial guess for $i = 0,\ldots,n$, where we vary the mesh size n to simulate

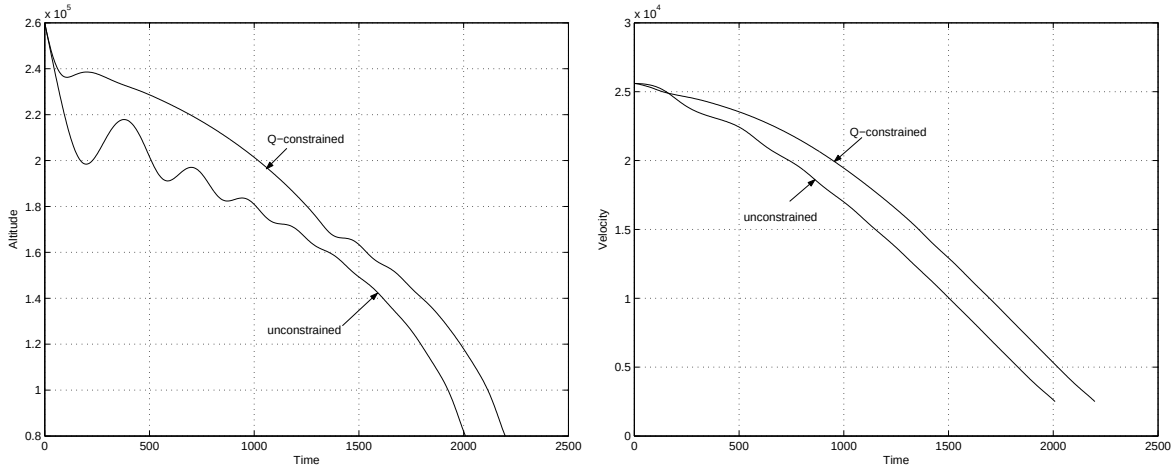


Figure 9.12: Altitude and Velocity of the Shuttle Problem

an ad-hoc mesh-refinement procedure.

$$
\begin{aligned}
t_{f_0} &= 2000.0,\\
h_{i,0} &= h(t_0) - i/n(h(t_0) - h(t_f)),\\
v_{i,0} &= v(t_0) - i/n(v(t_0) - v(t_f)),\\
\psi_{i,0} &= \psi(t_0) - i/n(\psi(t_0) - \psi(t_f)),\\
\theta_{i,0} &= \theta(t_0) + i/n(\theta(t_f) - \theta(t_0)),\\
\gamma_{i,0} &= \gamma(t_0) - i/n(\gamma(t_0) - \gamma(t_f))^4,\\
\alpha_{i,0} &= 17.4\pi/180,\\
\beta_{i,0} &= (i/n - 1)75\pi/180.
\end{aligned}
$$

Only IPOPT can solve this problem, both the competitors stop with "iteration limit exceeded". We summarize the results in Table 9.6. To give an idea of the "size" of the problem, the number of variables (vars), constraints (constr), lower and upper bounds (lbounds, ubounds) is appended.

n	t_f	$\theta(t_f)$	iter	time	vars	constr	lbounds	ubounds
250	2008.5932	34.14094	84	11.01	1750	1250	1499	1000
500	2008.5811	34.14112	88	23.60	3500	2500	2999	2000
750	2008.5749	34.14115	119	53.26	5250	3750	4499	3000
1000	2008.5696	34.14117	115	71.45	7000	5000	5999	4000
1500	2008.5769	34.14118	145	147.14	10500	7500	8999	6000
2000	2008.5499	34.14118	137	158.62	14000	10000	11999	8000

Table 9.6: Mesh-Refinement for the Shuttle Problem

9.2.8 Low-Thrust Orbit Transfer

The following problem is generally considered a hard one [5, page 155]. The motion of a space vehicle in Cartesian coordinates is governed by the following equation

$$\ddot{\mathbf{r}} + \mu\frac{\mathbf{r}}{r^3} = \mathbf{a}_d, \tag{9.22}$$

where $\mathbf{r}$ is the inertial position vector and r its magnitude, $\mathbf{a}_d$ is the is the disturbing acceleration, and μ the gravitational constant. Here, the minimum-fuel transfer of a spacecraft

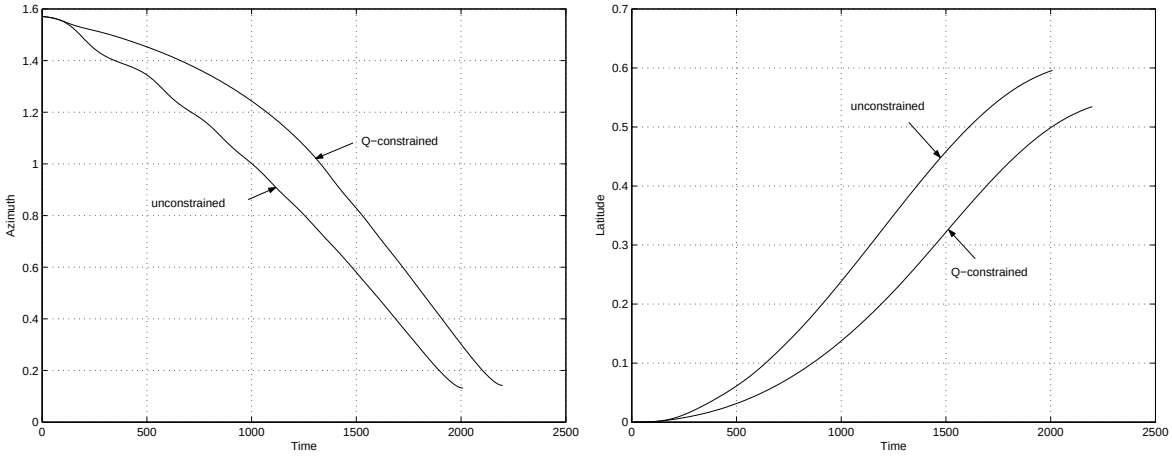


Figure 9.13: Azimuth and Latitude of the Shuttle Problem

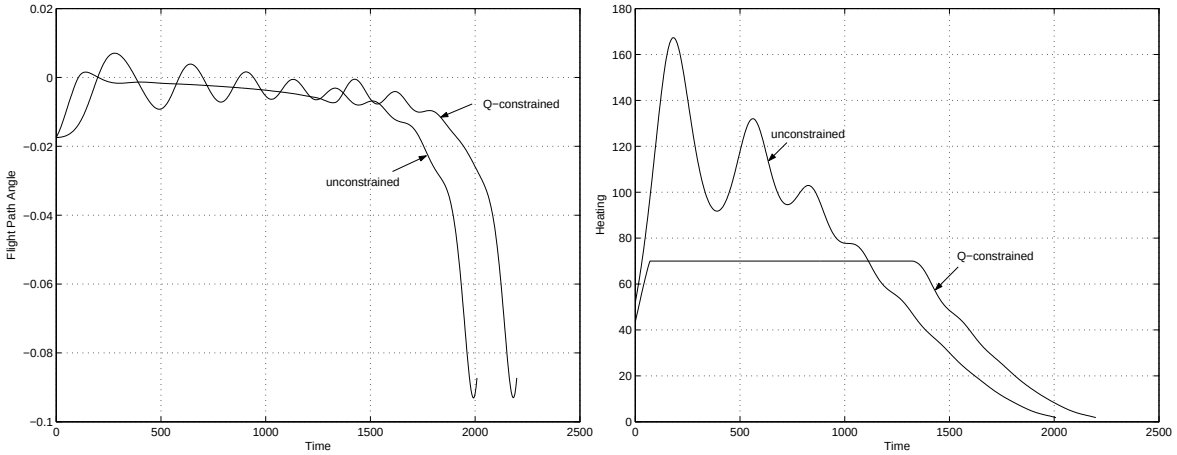


Figure 9.14: Flight Path and Heating of the Shuttle Problem

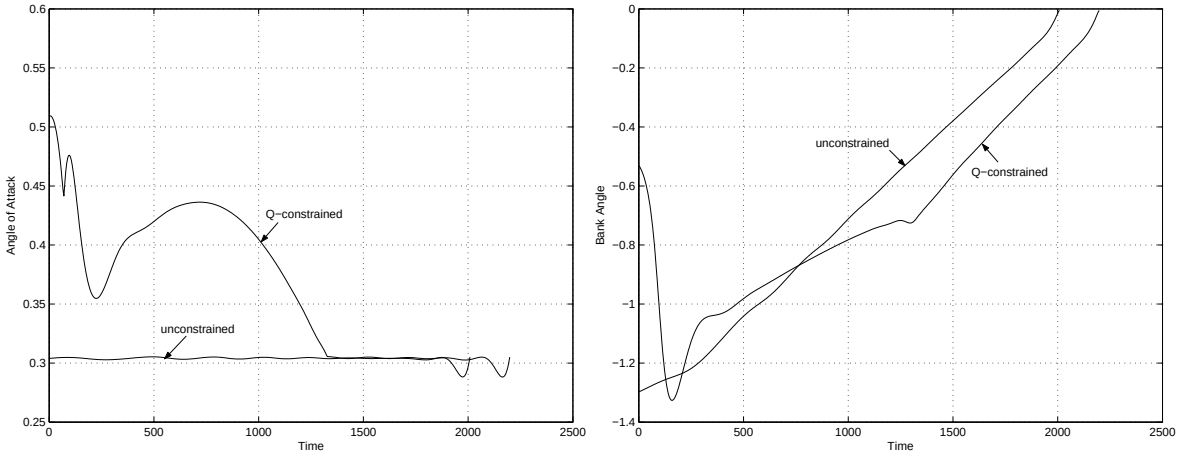


Figure 9.15: Angle of Attack and Bank Angle of the Shuttle Problem

from a low circular earth orbit to a mission orbit is described in modified equinoctial elements $\mathbf{y} = (p, f, g, h, k, L)'$ as follows [47]:

$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\Delta + \mathbf{b}, \tag{9.23}$$

where the dynamics matrix is

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{2p}{q}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin L & \frac{1}{q}\sqrt{\frac{p}{\mu}}((1+q)\cos L + f) & -\frac{g}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \\ -\sqrt{\frac{p}{\mu}}\cos L & \frac{1}{q}\sqrt{\frac{p}{\mu}}((1+q)\sin L + g) & \frac{f}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \\ 0 & 0 & \frac{s^2}{2q}\sqrt{\frac{p}{\mu}}\cos L \\ 0 & 0 & \frac{s^2}{2q}\sqrt{\frac{p}{\mu}}\sin L \\ 0 & 0 & \frac{1}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \end{pmatrix}$$

and the vector

$$\mathbf{b} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \sqrt{\mu p}\left(\frac{p}{q}\right)^2 \end{pmatrix}.$$

The equinoctial elements $\mathbf{y}$ are related to Cartesian coordinates $(\mathbf{r}, \boldsymbol{\nu})$ as follows

$$\mathbf{r} = \begin{pmatrix} \frac{r}{s^2}((1+\alpha^2)\cos L + 2hk\sin L) \\ \frac{r}{s^2}((1-\alpha^2)\sin L + 2hk\cos L) \\ \frac{2r}{s^2}(h\sin L - k\cos L) \end{pmatrix},$$

$$\boldsymbol{\nu} = \begin{pmatrix} -\frac{1}{s^2}\sqrt{\frac{p}{\mu}}((1+\alpha^2)\sin L - 2hk\cos L + (1+\alpha^2)g - 2fhk) \\ -\frac{1}{s^2}\sqrt{\frac{p}{\mu}}((-1+\alpha^2)\cos L + 2hk\sin L + (-1+\alpha^2)f - 2ghk) \\ \frac{2}{s^2}\sqrt{\frac{p}{\mu}}(h\cos L + k\sin L + fhgk) \end{pmatrix}.$$

The following variables are used

$$\begin{aligned} q &= 1 + f\cos L + g\sin L, \\ r &= \frac{p}{q}, \\ \alpha^2 &= h^2 - k^2, \\ s^2 &= 1 + h^2 + k^2. \end{aligned}$$

The disturbing acceleration vector $\mathbf{a}_d$ in Eq. (9.22) is replaced by Δ in Eq. (9.23) which is composed of two parts

$$\Delta = \Delta_g + \Delta_T,$$

the first representing the disturbance by the oblate earth and the second that of the thrust.

We obtain Δ_g as

$$\Delta_g = \mathbf{Q}'\boldsymbol{\delta}\mathbf{g} = \delta g_n \mathbf{i}_n - \delta g_r \mathbf{i}_r,$$

where

$$\delta g_n = -\frac{\mu\cos\phi}{r^2}\sum_{k=2}^{4}\left(\frac{R_e}{r}\right)^k P_k' J_k,$$

and

$$\delta g_r = -\frac{\mu}{r^2}\sum_{k=2}^{4}(1+k)\left(\frac{R_e}{r}\right)^k P_k J_k,$$

and ϕ is the geocentric latitude

$$\sin\phi = \frac{2}{s^2}(h\sin L - k\cos L),$$

$P_k(\sin\phi)$ is the k^{th}-order Legendre polynomial, P_k' its derivative and J_k are the zonal harmonic coefficients. The matrix $\mathbf{Q}$ is built up of the unit vectors

$$\mathbf{Q} = \begin{pmatrix} \mathbf{i}_r & \mathbf{i}_\theta & \mathbf{i}_h \end{pmatrix} = \begin{pmatrix} \dfrac{\mathbf{r}}{\|\mathbf{r}\|} & \dfrac{(\mathbf{r}\times\mathbf{v})\times\mathbf{r}}{\|\mathbf{r}\times\mathbf{v}\|\|\mathbf{r}\|} & \dfrac{\mathbf{r}\times\mathbf{v}}{\|\mathbf{r}\times\mathbf{v}\|} \end{pmatrix},$$

and the unit vector defining the local north direction is, with $\boldsymbol{e}_n = (0,0,1)'$,

$$\mathbf{i}_n = \frac{\boldsymbol{e}_n - (\boldsymbol{e}_n'\mathbf{i}_r)\mathbf{i}_r}{\|\boldsymbol{e}_n - (\boldsymbol{e}_n'\mathbf{i}_r)\mathbf{i}_r\|}.$$

The thrust component of Δ is

$$\Delta_T = \frac{(1+\tau/100)Tg_0}{w}\mathbf{u},$$

where $\mathbf{u} = (u_r, u_\theta, u_h)'$ is the vector of control variables, T is the magnitude of the (low) thrust, τ is a throttle factor, and w is the weight of the vehicle.

The development of the equinoctial elements in Eq. (9.23) is augmented by the weight equation

$$\dot{w} = -\frac{(1+\tau/100)T}{I_{sp}}, \tag{9.24}$$

in which I_{sp} is the specific impulse of the engine.

Now, we are in a position to complement the problem formulation:

$$\max_{\mathbf{u}} w(t_f) \tag{9.25}$$

subject to Eq. (9.23) and Eq. (9.24) and to the control constraints

$$\begin{aligned} \|\mathbf{u}\| &= 1, \\ \tau_L \le \tau &\le 0, \end{aligned} \tag{9.26}$$

with initial conditions

$$\begin{aligned} p(t_0) &= 21837080.052835, \\ f(t_0) &= 0.0, \\ g(t_0) &= 0.0, \\ h(t_0) &= -0.25396764647494, \\ k(t_0) &= 0.0, \\ L(t_0) &= \pi, \\ w(t_0) &= 1.0, \end{aligned}$$

and final conditions

$$\begin{aligned} p(t_f) &= 40007346.015232, \\ \sqrt{f^2+g^2} &= 0.73550320568829, \\ \sqrt{h^2+k^2} &= 0.61761258786099, \\ fh + gk &= 0.0, \\ gh - fk &\le 0.0. \end{aligned}$$

The following constants complete the problem

$$\begin{aligned} &\mu = 1.407645794\times10^{16}, \quad g_0 = 32.174, \quad I_{sp} = 450, \\ &T = 4.446618\times10^{-3}, \quad R_e = 20925662.73, \quad \tau_L = -50.0, \\ &J_2 = 1082.639\times10^{-6}, \quad J_3 = -2.565\times10^{-6}, \quad J_4 = -1.608\times10^{-6}. \end{aligned}$$

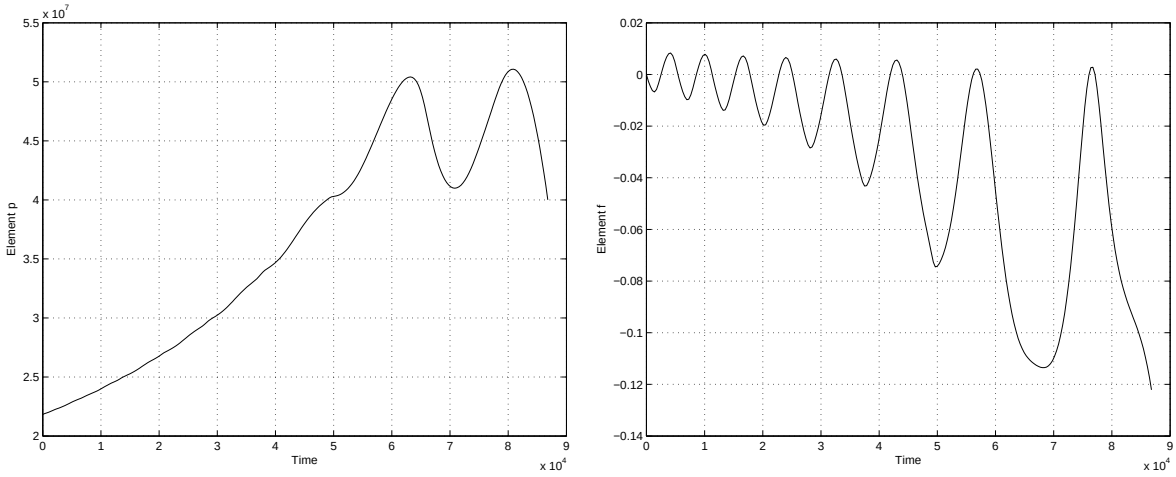


Figure 9.16: Equinoctial Elements of the Orbiter

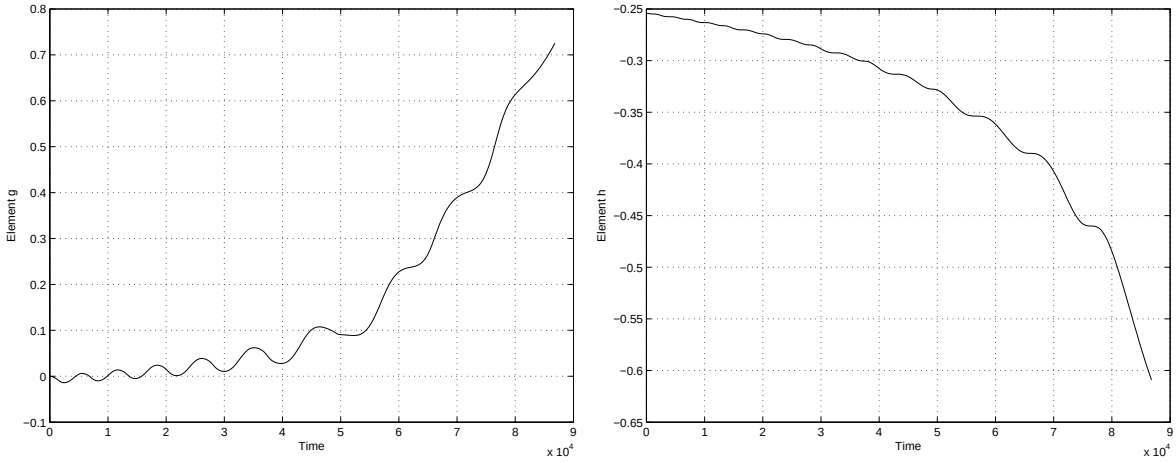


Figure 9.17: Equinoctial Elements of the Orbiter, contd.

To transcribe the optimal control problem into a finite-dimensional nonlinear programming problem we again use trapezoidal discretization (9.3) to approximate the differential equations. The initial guess can be taken from the Appendix in which we give the file orbit.mod for the convenience of the reader.

Only `IPOPT` is possible to solve this (hard) problem. We present The results in Table

n	t_f	$w(t_f)$	τ	niter	cputime
150	86629	0.2202604590366861	-8.91038	661	403
200	86796	0.2202252223455872	-9.08187	839	686
250	86802	0.2202070599215732	-9.08633	889	899
400	80943	0.2001646797364390	5.5219e-9	827	1376

Table 9.7: Results of the Orbiter Problem

State and control variables are shown in Figs. 9.16–9.19.

9.2.9 Cam Shape

Although the optimization of the shape of a cam is not an optimal control problem it plays an important role in the design and simulation of dynamic systems. Therefore it is included

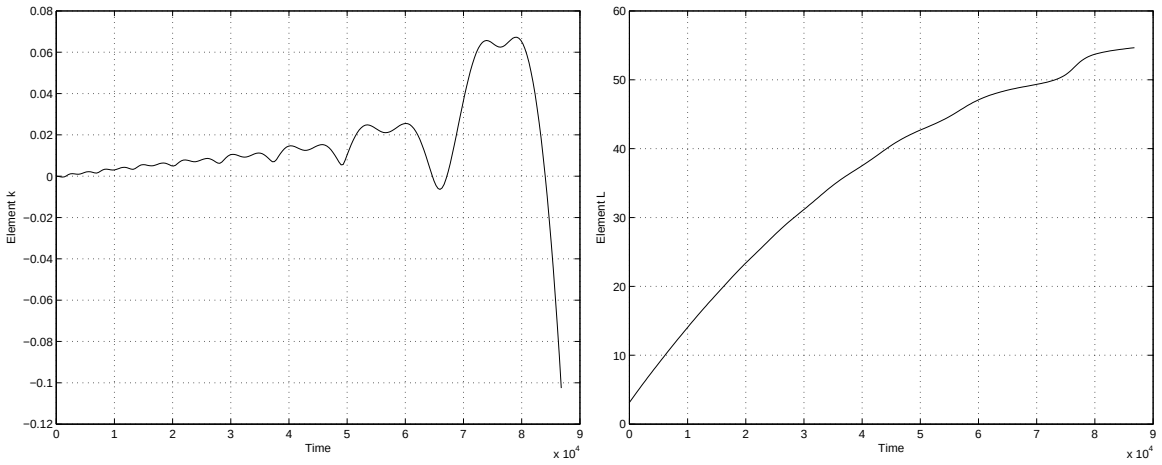


Figure 9.18: Equinoctial Elements of the Orbiter, contd.

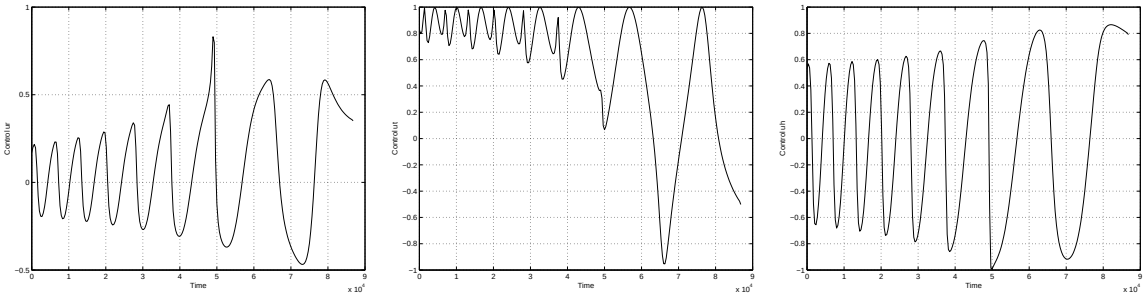


Figure 9.19: Control Variables of the Orbiter Problem

in this report.

The design goal is to maximize the area of the valve opening

$$f(r) = \frac{\pi r_v^2}{n} \sum_{i=1}^{n} r_i \tag{9.27}$$

for one rotation of a convex cam with constraints on the curvature and on the radii of the cam. The design variables, $r_{min} \le r_i \le r_{max}$, $i = 1, \ldots, n$, represent the radii of the cam at equally spaced angles distributed over an angle of $\frac{4}{5}\pi$. The remaining part ($\frac{6}{5}\pi$) of the circumference is circular with radius r_{min}. The convexity constraints can be formulated as

$$A(r_{i-1}, r_{i+1}) \le A(r_{i-1}, r_i) + A(r_i, r_{i+1}),$$

where $A(r_i, r_j)$ is the area of the triangle defined by the origin and the points r_i and r_j on the surface. This inequality is equivalent to

$$2r_{i-1}r_{i+1}\cos(\theta) \le r_i(r_{i-1} + r_{i+1}), \quad i = 0, \ldots, n+1, \tag{9.28}$$

where $r_{-1} = r_0 = r_{min}$, $r_{n+1} = r_{max}$, $r_{n+2} = r_n$, and $\theta = \frac{2}{5(n+1)}\pi$ is the angle between discretization points. The curvature constraint is expressed by

$$-\alpha \le \frac{r_{i+1} - r_i}{\theta} \le \alpha. \tag{9.29}$$

The following data are used: $r_{min} = 1$, $r_{max} = 2$, $r_v = 1$, and $\alpha = 1.5$.

All three solvers solve the problem as can be seen from line 3 of Table 9.2 with a strange behavior of LOQO for the lower number of parameters. Also, in this "easy" problem the

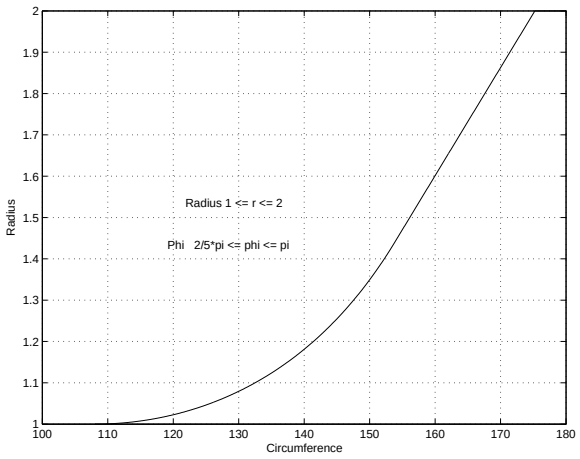


Figure 9.20: Radius of the Cam-Shape Problem

interior-point solver IPOPT shows an advantegeous behavior compared to the active-set solver SNOPT. The radius is constant $r = r_{min}$ along the angle $0 \leq \phi \leq 2\pi/5$ until it reaches the value $r = r_{max}$ at about 175° as is shown in Fig. 9.20 for the half-circumference of the cam. The modeling of this problem is certainly improvable as the shape of the cam is not continuously differentiable at $\phi \approx 180^\circ$.

Acknowledgements. I am grateful to Philip Gill, Michael Saunders and Bob Vanderbei for providing sources and/or permanent full licenses of their codes SNOPT, MINOS and LOQO, respectively. I thank Andreas Wächter for extensive discussion of his code IPOPT and I admire his decision to publish it as open source code. May this be an encouraging example for other code developers to follow. I appreciate the suggestions Klaus Well and Sven Erb made for some applications.

Anhang A

Konvexe Mengen und Funktionen

Das beschränkte Optimierungsproblem (5.15) führt über die Charakterisierung seiner Lösung (5.20) im allgemeinen nur auf eine lokale Lösung, von welcher das Problem mehrere mit unterschiedlichem Wert der Zielfunktion haben kann. In einem wichtigen Sonderfall, dem konvexen Programm, ist aber die lokale Lösung auch die globale.

Zunächst sei Konvexität gleichermaßen für Mengen und Funktionen definiert.
Definition 28. (Konvexe Mengen.)
Eine Menge $S \in \mathbb{R}^n$ ist konvex, wenn alle Punkte auf der Strecke zwischen je zwei Punkten x und y der Menge vollständig in der Menge liegen:

$$\alpha x + (1-\alpha)y \in S, \quad \forall \alpha \in [0,1]. \tag{A.1}$$

Konvexe Funktionen werden auf konvexen Mengen definiert.
Definition 29. (Konvexe Funktionen.)
Eine Funktion f ist konvex, wenn ihr Definitionsbereich[1] eine konvexe Menge ist, und für je zwei Punkte x und y des Definitionsbereichs der Graph von f unter der Strecke liegt, die die beiden Punkte $x, f(x)$ und $y, f(y)$ verbindet:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \quad \forall \alpha \in [0,1]. \tag{A.2}$$

Definition 30. (Konkave Funktionen.)
Eine Funktion f ist konkav, wenn ihr Negatives $-f$ konvex ist.

Die Definition (A.2) ist notwendig und hinreichend aber leider nicht konstruktiv, d.h. weil eine große Anzahl (∞) von Punkten überprüft werden muss. Folgender Satz gibt eine einfachere Aussage, ob eine Funktion konvex ist:
Satz 31. (Konvexität einer Funktion.)
Eine Funktion $f(x)$*, die auf einer konvexen Menge* S *definiert ist, ist konvex genau dann, wenn ihre Hesse-Matrix* $H(x) = \nabla_{xx} f(x)$ *positiv semidefinit für alle* $x \in S$ *ist. Ist sie positiv definit, nennt man die Funktion streng konvex.*

Für Funktionen einer rellen Variablen bedeutet dies, dass die zweite Ableitung der Funktion nicht-negativ sein muss.
Beispiel 32. Vorgelegt sei die Funktion $f(x) = x_1^2 + x_2^2 - 4$. Ihr Gradient und ihre Hesse-Matrix sind

$$\nabla f(x) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}, \quad H(x) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Beide Eigenwerte von H sind positiv, d.h. f ist streng konvex überall im $\mathbb{R}^2$.

[1] Dieser Definitionsbereich ist für das Problem (5.15) immer die zulässige Menge Ω nach (5.18).

Nun sei $f(x) = x_1^3 + x_2^2 - 4$, mit

$$\nabla f(x) = \begin{pmatrix} 3x_1^2 \\ 2x_2 \end{pmatrix}, \quad H(x) = \begin{pmatrix} 6x_1 & 0 \\ 0 & 2 \end{pmatrix}.$$

Jetzt ist f nur noch für $x_1 \geq 0$ konvex. Was ist sie für $x_1 \leq 0$? □

Die wichtigsten Ergebnisse für konvexe Programme sind in den folgenden Sätzen zusammengefasst.

Satz 33. (Konvexität des zulässigen Gebietes.)
Das zulässige Gebiet Ω (5.18) *ist konvex, wenn die Gleichungsnebenbedingungen,* c_i, $i \in \mathcal{E}$, (5.15b) *des Problems lineare Funktionen und die Ungleichungsnebenbedingungen,* $-c_i$, $i \in \mathcal{I}$, (5.15c) *konvexe Funktionen sind.*

Satz 34. (Globales Optimum.)
Wenn die Zielfunktion f *des Problems* (5.15) *konvex und das zulässige Gebiet* Ω *konvex ist, dann sind die KKT-Bedingungen* (5.20) *notwendig und hinreichend für ein globales Optimum.*

Beispiel 35. (Experiment zur Konvexität.)
Für Diagonalmatrizen sind die Diagonalelemente die Eigenwerte auf der Matrix. Verändern Sie diese im folgenden `Maple`-worksheet [28], um ein Gefühl für die Definitheit von Matrizen zu erhalten. Wählen Sie z.B. auch einen der Eigenwerte zu Null, um zu erkennen, was semidefinit bedeutet.

```
> restart;
```

```
> with(LinearAlgebra):
```

```
> Q:=Matrix([[2,0],[0,1]]);
```

$$Q := \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> v:=Vector([x,y]);
```

$$v := \begin{bmatrix} x \\ y \end{bmatrix}$$

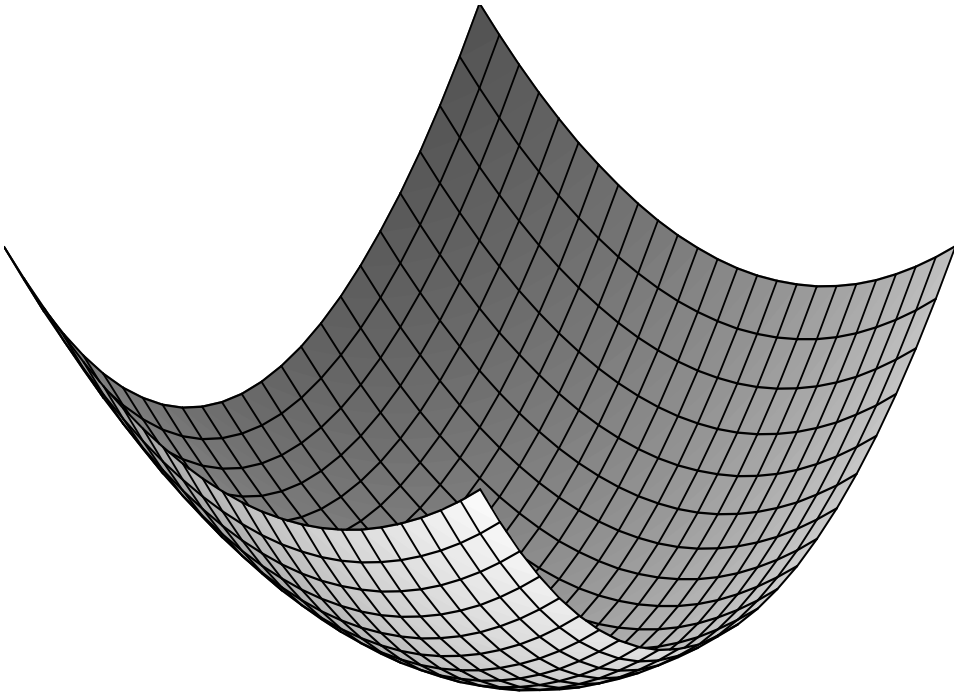
```
> c:=Vector([2,2]);
```

$$c := \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

```
> f:=evalm(Transpose(v)&*Q&*v+Transpose(c)&*v);
```

$$f := 2x^2 + y^2 + 2x + 2y$$

```
> plot3d(f,x=-2..2,y=-2..2);
```



□

Anhang B

Eigenwerte und Eigenvektoren

Definition 36. Für eine Matrix $A \in \mathbb{R}^{n\times n}$ und einen Vektor $x \in \mathbb{R}^n, x \neq 0$, gibt es einen skalaren Wert λ, so dass

$$Ax = \lambda x \tag{B.1}$$

gilt.

Der Skalar λ heisst Eigenwert zum Eigenvektor x. Da $x \neq 0$ vorausgesetzt wird, gilt nach (B.1)

$$|A - \lambda I| = 0. \tag{B.2}$$

Dies ist ein Polynom n-ten Grades in λ, dessen Lösung die n Eigenwerte der Matrix A sind.[1] Im Kontext diese Manuskripts ist die Matrix A häufig symmetrisch, und in diesem Fall gilt folgender

Satz 37. (Symmetrische, reelle Matrix.)
Eigenwerte und Eigenvektoren symmetrischer, reeller Matrizen sind reell.

Sind die Eigenwerte unterschiedlich, dann gilt zusätzlich

Satz 38. *Eigenvektoren zu unterschiedlichen Eigenwerten reeller, symmetrischer Matrizen sind orthogonal zueinander, d.h. ihr Skalarprodukt ist Null.*

Beispiel 39. Gegeben sei

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Aus $|A - \lambda I| = 0$ folgt $\lambda^2 - 4\lambda + 3 = 0$ mit den beiden (unterschiedlichen) Lösungen $\lambda_1 = 1$ und $\lambda_2 = 3$. Diese werden in Gl. (B.1) eingesetzt: 1)

$$Ax^1 = \lambda_1 x^1 \quad \text{oder} \quad \begin{pmatrix} 2-1 & 1 \\ 1 & 2-1 \end{pmatrix} \begin{pmatrix} x_1^1 \\ x_2^1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

was $x_1^1 = -x_2^1$ ergibt. Eine mögliche Lösung ist $x^1 = (1, -1)^T$.

Und 2)

$$Ax^2 = \lambda_2 x^2 \quad \text{oder} \quad \begin{pmatrix} 2-3 & 1 \\ 1 & 2-3 \end{pmatrix} \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

was $x_1^2 = x_2^2$ ergibt, oder z.B. $x^2 = (1, 1)^T$. Das Skalarprodukt $(x^1)^T x^2 = 1 \times 1 + 1 \times (-1)$ hat den Wert Null. □

[1] In der Praxis wird aus numerischen Gründen nicht Gl. (B.2) gelöst, sondern der QR-Algorithmus angewendet [13], [24].

Appendix C

AMPL-Syntax

Ampl [20] is a powerful meta language for formulating optimization problems. The models are described in files with the extension .mod, e.g. model.mod. The most important syntax rules for this course are described in the following. Each declaration ends with a semicolon, Comments (hash marks #) can be included within the lines of the file. Examples of the syntax are given in the following two extended model files. More detailled information can be found in the reference [20].

Set Declaration.

A set declaration is defined with the keyword set followed by a name and by a comma-separated list of members between braces { }.
set S = {1,2,3,4,5};
set S = {1..5};

Variable Declaration.

Variable declarations are defined with the keyword var followed by a name, which can be attributed with optional attributes, e.g. bounds ($<=, >=$) or initial values $:=$. The default value is Zero.
var x $>=$ 0, $:=$ 1;
In addition, indexed variables can be declared, where the index can be defined in a set, e.g.
var x {1..3}; set S = {1..100};
var x {S};

Parameter Declaration.

A single named numerical value is called a parameter in Ampl. Parameter declarations are defined with the keyword param followed by a name. Its value can be assigned immediately in the declaration or via a data section or data file.
param pi := 4*atan(1);

Objective Declaration.

The declaration of an objective is either `maximize` or `minimize` followed by the name of the objective which is separated from the expression which forms the objective by a colon.
`minimize f: x1^2+2*x2^3-sin(x3*pi/180);`

Constraint Declaration.

The declaration of a constraint can take three forms: either `subject to` (abbreviated as `s.t.`) followed by the name of the constraint or by its name alone. Again, the expression evaluating the constraint is separated by a colon from its name.
`subject to c1: x1^2 + 2*x2^2 >= 1;`
`s.t. c2: 0 <= 4*x1 - x2 <= 1;`
`c3: x1 + x2 = 0;`

Indexing.

Most entities in `ampl` can be defined in collections indexed over a set; individual items are identified by appending a bracketed subscript to the name of the entity.
`param N=100;`
`set S = {1..N};`
`param g{S};`
`var x{S};`
`minimize f: sum {j in S} g[j]*x[j];`

Print and Display.

The `display`, `print`, and `printf` commands print arbitrary expressions to standard output or redirect ($>$ or $\gg$) it to a file. The `printf` command prints its argument list in a prescribed format (pretty much like printf in the C language).
`set S = {0..N};`
`display x1, x2, f;`
`print {i in 0..N}: x[i];`
`printf {i in S}:  "%i %f10.5 \n", i, x[i] > file;`

Modeling Commands.

`let`
The `let` command changes the value of a possibly indexed set, parameter or variable to the value of the expression.
`let X := x1/t;`
`let L := l1–l2;`

`option`
`Ampl` maintains the values of a variety of `options` that influence the behavior of commands and solvers. In our context the most important `option` is the choice of a solver (the default solver — if the solver option is omitted — being `minos`[39]).

```
option solver snopt;
option snopt_options 'outlev=1';
```

`solve`
The `solve` command causes `ampl` to solve the problem of the model file with the solver chosen by the `option` solver.

Brachistochrone.

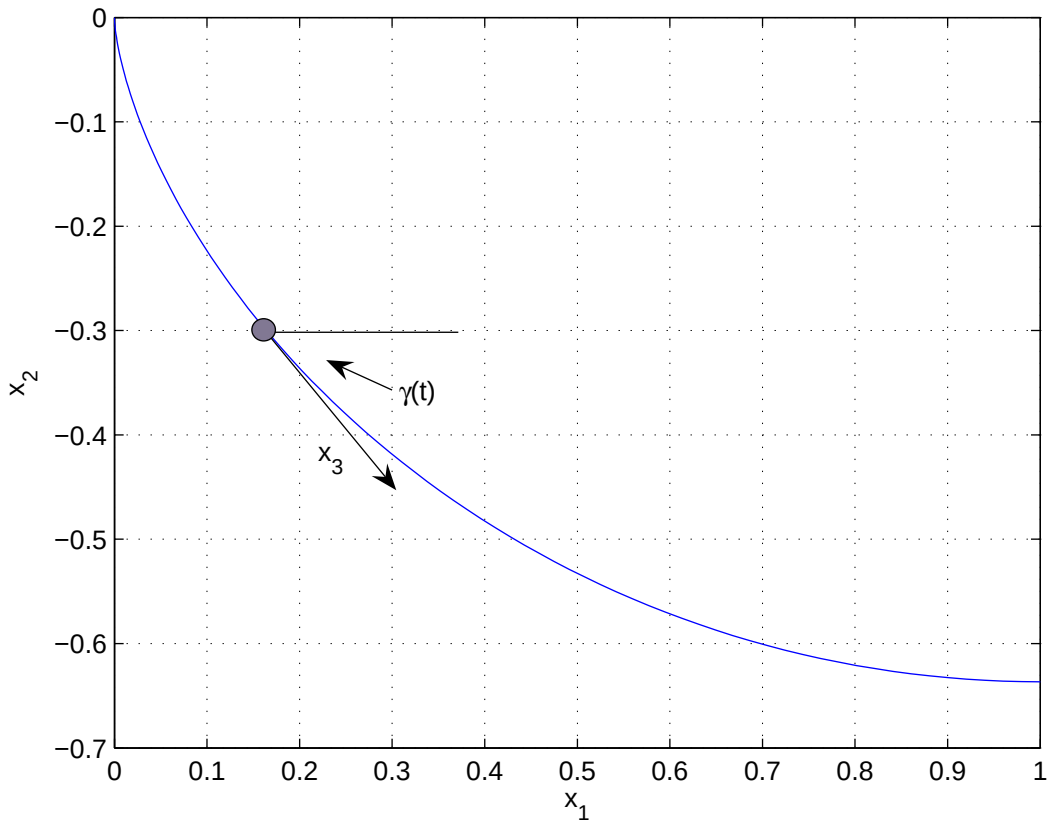
Als einfaches Beispiel f"ur die `ampl`-Syntax folgt der model file f"ur die Brachistochrone [4]: Finde die Kurve einer elastischen Schnur (repr"asentiert durch ihren Bahnwinkel γ gemessen von der Horizontalen), auf der eine Perle reibungsfrei im Schwerefeld der Erde von einer gegebenen Anfangsposition x_i zu einer gegebenen Endposition x_f in minimaler Zeit t_f gleitet.

$$\begin{aligned} &\min\ t_f \\ &\text{s.t. } \dot{x}_1(t) = x_3(t)\cos\gamma(t), \qquad x_1(t_i) = 0, \quad x_1(t_f) = 1, \\ &\qquad \dot{x}_2(t) = x_3(t)\sin\gamma(t), \qquad x_2(t_i) = 0, \quad x_2(t_f) = \text{frei}, \\ &\qquad \dot{x}_3(t) = g\sin\gamma(t), \qquad x_3(t_i) = 0, \quad x_3(t_f) = \text{frei}. \end{aligned}$$

```
# Brachistochrone Problem, trapezoidal rule

param n;                        # number of time intervals
param g;                        # normalized gravity acceleration
var T >= 0, := 1;               # final time and its estimate

set D = {1..3};                 # number of differential equations
set N = {0..n};                 # discrete times

param x0 {D} = 0;               # initial position
param xn {D} = 1;               # final   position

var x {D, N};                   # position and velocity
var u {N};                      # control

let g := 1;
let n := 100;
var h  = T/n;                   # step size (h is variable as T is)

minimize tf: T;

xi {j in D} : x[j,0] = x0[j];   # initial conditions
xf          : x[1,n] = xn[1];   # final conditions

# Difference equations
y {i in 0..n-1}: x[1,i+1] = x[1,i] + h/2*(x[3,i]*cos(u[i])+x[3,i+1]*cos(u[i+1]));
z {i in 0..n-1}: x[2,i+1] = x[2,i] + h/2*(x[3,i]*sin(u[i])+x[3,i+1]*sin(u[i+1]));
v {i in 0..n-1}: x[3,i+1] = x[3,i] + h/2*g*(sin(u[i])+sin(u[i+1]));

option snopt_options "outlev=1";
option solver snopt;
solve;
```

```
printf {i in N}: "%10.5f %10.5f %10.5f %10.5f \n", i/n*T, x[1,i], -x[2,i], u[i] > brach.out;
# brach.out can be loaded by the loaded into Matlab by load('brach.out')
# and the interesting entities can be plotted by e.g. plot(brach(:,2),brach(:,3))
```

as is shown in the following figure.



Appendix D

Model file zermelo.mod

```
# Modified Zermelo problem
# Two-phase-problem
# as of S.O. Erb: Mit dem Schiff von A nach B
# unpublished report
# Institute of Flight Mechanics and Flight Control
# University of Stuttgart
# implemented by
# Dieter Kraft
# Munich University of Applied Sciences
# September 2004

param pi := 4*atan(1);
param n1 > 0, integer;
param n2 > 0, integer;
param p{1..8};
param kappa;
param u1l;
param u1u;
param u2l;
param u2u;

var t1 >= 0.1, <= 4.5, :=4.50; #t*=4.20078 for n1=25
var t2 >= 0.1, <= 3.5, :=3.50; #t*=7.54770 for n2=25

var h1 = t1/n1/2;
var h2 = t2/n2/2;

var u1{0..n1};
var u2{0..n2};

var x1{1..4,0..n1};
var x2{1..4,0..n2};
var switch >= 0, integer;


option ipopt_options  'iprint=0';
option knitro_options 'outlev=1';
option lancelot_options 'outlev=1';
option loqo_options  'outlev=0 iterlim=10000';
option minos_options 'outlev=1';
option npsol_options 'outlev=1';
```

```
option snopt_options 'outlev=1';

let switch := 6;

if switch == 1 then option solver ipopt;
if switch == 2 then option solver knitro;
if switch == 3 then option solver lancelot;
if switch == 4 then option solver loqo;
if switch == 5 then option solver minos;
if switch == 6 then option solver npsol;
if switch == 7 then option solver snopt;

problem phase_1: t1, u1, x1;
############################


minimize tf1: t1 + kappa*sum{i in 0..n2-1} (u1[i+1]-u1[i])^2/h1*2;


# Differential constraints

de1{i in 0..n1-1}: x1[1,i+1] = x1[1,i]
+ h1*p[1]*((1-(2/pi*x1[4,i])^2)*cos(x1[3,i])-0.7*x1[2,i]
+ (1-(2/pi*x1[4,i+1])^2)*cos(x1[3,i+1])-0.7*x1[2,i+1]);

de2{i in 0..n1-1}: x1[2,i+1] = x1[2,i]
+ h1*p[1]*((1-(2/pi*x1[4,i])^2)*sin(x1[3,i])
+ (1-(2/pi*x1[4,i+1])^2)*sin(x1[3,i+1]));

de3{i in 0..n1-1}: x1[3,i+1] = x1[3,i]
+ h1*(sin(2*x1[4,i])+sin(2*x1[4,i+1]));

de4{i in 0..n1-1}: x1[4,i+1] = x1[4,i]
+ h1*(u1[i]+u1[i+1]);

# Algebraic constraints

ae1{i in 0..n1}: (x1[1,i]-4.0)^2 + (x1[2,i]-1.1)^2  >= p[6]^2;
#ae3{i in 0..n1}:    1 <= x1[1,i] <= 5;
#ae4{i in 0..n1}: p[3] <= x1[2,i] <= 1;

# Control constraint

ce1{i in 0..n1}: u1l <= u1[i] <= u1u;

# Initial conditions

ic1: x1[1,0] = p[2];
ic2: x1[2,0] = p[3];

# Final conditions
# Encounter with support ship

ic5: 2.0-0.15*p[1]*p[8]*t1 = x1[1,n1];
ic6: 3.0-p[1]*p[8]*t1      = x1[2,n1];
```

```
problem phase_2: t2, u2, x2;
############################


minimize tf2: t2 + kappa*sum{i in 0..n2-1} (u2[i+1]-u2[i])^2/h2*2;

# Differential constraints

de5{i in 0..n2-1}: x2[1,i+1] = x2[1,i]
+ h2*p[1]*((1-(2/pi*x2[4,i])^2)*cos(x2[3,i])-0.7*x2[2,i]
+ (1-(2/pi*x2[4,i+1])^2)*cos(x2[3,i+1])-0.7*x2[2,i+1]);

de6{i in 0..n2-1}: x2[2,i+1] = x2[2,i]
+ h2*p[1]*((1-(2/pi*x2[4,i])^2)*sin(x2[3,i])
+ (1-(2/pi*x2[4,i+1])^2)*sin(x2[3,i+1]));

de7{i in 0..n2-1}: x2[3,i+1] = x2[3,i]
+ h2*(sin(2*x2[4,i])+sin(2*x2[4,i+1]));

de8{i in 0..n2-1}: x2[4,i+1] = x2[4,i]
+ h2*(u2[i]+u2[i+1]);

# Algebraic state constraints

ae2{i in 0..n2}: (x2[1,i]+1.2)^2 + (x2[2,i]-p[7])^2 >= 1^2;
#ae5{i in 0..n2}: p[4] <= x2[1,i] <= 2;
#ae6{i in 0..n2}:    0 <= x2[2,i] <= 3;

# Control constraint

ce2{i in 0..n2}: u2l <= u2[i] <= u2u;

# Initial conditions
# Phase connecting conditions

ph1: x2[1,0] = x1[1,n1];
ph2: x2[2,0] = x1[2,n1];
ph3: x2[3,0] = x1[3,n1];
ph4: x2[4,0] = x1[4,n1];

# Final conditions

fc1: x2[1,n2] = p[4];
fc2: x2[2,n2] = p[5];

commands z1.dat;

solve phase_1;

display t1, x1[1,n1], x1[2,n1], x1[3,n1], x1[4,n1];

solve phase_2;

display t1, t2, t1+t2, x2[1,n2], x2[2,n2], x2[3,n1], x2[4,n1];
```

```
display _total_solve_time;

printf {i in 0..n1}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
i/n1*t1, {j in 1..4} x1[j,i], u1[i], 4.0+p[6]*cos(i/n1*2*pi), 1.1+p[6]*sin(i/n1*2*pi),
2-0.15*p[1]*p[8]*i/n1*t1, 3-p[1]*p[8]*i/n1*t1 > z;
printf {i in 0..n2}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
t1+i/n2*t2, {j in 1..4} x2[j,i], u2[i], -1.2+cos(i/n1*2*pi), p[7]+1.0*sin(i/n1*2*pi),
x2[1,0]-0.15*p[1]*p[8]*i/n2*t2, x2[2,0]-p[1]*p[8]*i/n2*t2 >> z;
```

Appendix E

Model file orbit.mod

```
# Low Thrust Orbit Maneuver
# J.T. Betts: Practical Methods for Optimal Control
# Using Nonlinear Programming,
# Example 5.3, p.147ff
# Discretization by trapezoidal formulation
# Implemented by Dieter Kraft,
# Munich University of Applied Sciences
# July, 2004

param pi := 4*atan(1);
param Isp := 450;
param mu := 1.407645794e16;
param G0 := 32.174;
param J2 := 1082.639e-6;
param J3 := -2.565e-6;
param J4 := -1.608e-6;
param T := 4.446618e-3;
param Re := 209.2566273e5;
param tauL := -50.0;
param p0 := 218.37080052835e5;
param f0 := 0.0;
param g0 := 0.0;
param h0 := -0.25396764647494;
param k0 := 0.0;
param L0 := pi;
param w0 := 1.0;
param wf := 0.2; # estimated
param pf := 400.07346015232e5;
param fgf := 0.73550320568829;
param hkf := 0.61761258786099;

param n; # discretization intervals

var tf >= 0;
var step = tf/n;

# state variables
# modified equinoctial elements
# http://www.cdeagle.com/pdf/mee.pdf

var p{0..n};
```

```
var f{0..n};
var g{0..n};
var h{0..n};
var k{0..n};
var L{0..n};
var w{0..n};

# control variables (thrust components)

var ur{0..n};
var ut{0..n};
var uh{0..n};
var tau; # throttle parameter

# dependent variables

var q{i in 0..n} = 1.0 + f[i]*cos(L[i]) + g[i]*sin(L[i]);
var r{i in 0..n} = p[i]/q[i];
var a2{i in 0..n} = h[i]^2 - k[i]^2;
var s2{i in 0..n} = 1.0 + h[i]^2 + k[i]^2;
var r1{i in 0..n} = r[i]/s2[i]*((1.0+a2[i])*cos(L[i])+2*h[i]*k[i]*sin(L[i]));
var r2{i in 0..n} = r[i]/s2[i]*((1.0-a2[i])*sin(L[i])+2*h[i]*k[i]*cos(L[i]));
var r3{i in 0..n} = 2*r[i]/s2[i]*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var v1{i in 0..n} = -1.0/s2[i]*sqrt(mu/p[i])
   *((1.0+a2[i])*sin(L[i])-2*h[i]*k[i]*cos(L[i])+(1.0+a2[i])*g[i]-2*f[i]*h[i]*k[i]);
var v2{i in 0..n} = -1.0/s2[i]*sqrt(mu/p[i])
   *((-1.0+a2[i])*cos(L[i])+2*h[i]*k[i]*sin(L[i])+(-1.0+a2[i])*f[i]+2*g[i]*h[i]*k[i]);
var v3{i in 0..n} = 2.0/s2[i]*sqrt(mu/p[i])
                  *(h[i]*cos(L[i])+k[i]*sin(L[i])+f[i]*h[i]+g[i]*k[i]);
var rnorm{i in 0..n} = sqrt(r1[i]^2+r2[i]^2+r3[i]^2);
var rcrossv1{i in 0..n} = r2[i]*v3[i]-v2[i]*r3[i];
var rcrossv2{i in 0..n} = r3[i]*v1[i]-v3[i]*r1[i];
var rcrossv3{i in 0..n} = r1[i]*v2[i]-v1[i]*r2[i];
var rcrossvnorm{i in 0..n} = sqrt(rcrossv1[i]^2+rcrossv2[i]^2+rcrossv3[i]^2);
var rcrossvcrossr1{i in 0..n} = rcrossv2[i]*r3[i]-rcrossv3[i]*r2[i];
var rcrossvcrossr2{i in 0..n} = rcrossv3[i]*r1[i]-rcrossv1[i]*r3[i];
var rcrossvcrossr3{i in 0..n} = rcrossv1[i]*r2[i]-rcrossv2[i]*r1[i];
var a12{i in 0..n} = 2*p[i]/q[i]*sqrt(p[i]/mu);
var a21{i in 0..n} = sqrt(p[i]/mu)*sin(L[i]);
var a22{i in 0..n} = 1/q[i]*sqrt(p[i]/mu)*((1+q[i])*cos(L[i])+f[i]);
var a23{i in 0..n} = -g[i]/q[i]*sqrt(p[i]/mu)*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var a31{i in 0..n} = -sqrt(p[i]/mu)*cos(L[i]);
var a32{i in 0..n} = 1/q[i]*sqrt(p[i]/mu)*((1+q[i])*sin(L[i])+g[i]);
var a33{i in 0..n} = f[i]/q[i]*sqrt(p[i]/mu)*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var a43{i in 0..n} = sqrt(p[i]/mu)*s2[i]/q[i]/2*cos(L[i]);
var a53{i in 0..n} = sqrt(p[i]/mu)*s2[i]/q[i]/2*sin(L[i]);
var a63{i in 0..n} = sqrt(p[i]/mu)/q[i]*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var sphi{i in 0..n} = 2*(h[i]*sin(L[i])-k[i]*cos(L[i]))/s2[i];
var P2{i in 0..n} = (3*sphi[i]^2-1)/2;
var P3{i in 0..n} = (5*sphi[i]^3-3*sphi[i])/2;
var P4{i in 0..n} = (35*sphi[i]^4-30*sphi[i]^2+3)/8;
var dP2{i in 0..n} = 3*sphi[i];
var dP3{i in 0..n} = (15*sphi[i]^2-3)/2;
var dP4{i in 0..n} = (70*sphi[i]^3-30*sphi[i])/4;
var dgn{i in 0..n} = -mu*sqrt(1-sphi[i]^2)/r[i]^2*
                    ((Re/r[i])^2*dP2[i]*J2+(Re/r[i])^3*dP3[i]*J3+(Re/r[i])^4*dP4[i]*J4);
```

```
var dgr{i in 0..n} = -mu/r[i]^2*
                    (3*(Re/r[i])^2*P2[i]*J2+4*(Re/r[i])^3*P3[i]*J3+5*(Re/r[i])^4*P4[i]*J4);
var ir1{i in 0..n} = r1[i]/rnorm[i];
var ir2{i in 0..n} = r2[i]/rnorm[i];
var ir3{i in 0..n} = r3[i]/rnorm[i];
var it1{i in 0..n} = rcrossvcrossr1[i]/(rcrossvnorm[i]*rnorm[i]);
var it2{i in 0..n} = rcrossvcrossr2[i]/(rcrossvnorm[i]*rnorm[i]);
var it3{i in 0..n} = rcrossvcrossr3[i]/(rcrossvnorm[i]*rnorm[i]);
var ih1{i in 0..n} = rcrossv1[i]/rcrossvnorm[i];
var ih2{i in 0..n} = rcrossv2[i]/rcrossvnorm[i];
var ih3{i in 0..n} = rcrossv3[i]/rcrossvnorm[i];
var hin1{i in 0..n} = -ir3[i]*ir1[i];
var hin2{i in 0..n} = -ir3[i]*ir2[i];
var hin3{i in 0..n} = 1-ir3[i]*ir3[i];
var innorm{i in 0..n} = sqrt(hin1[i]^2+hin2[i]^2+hin3[i]^2);
var in1{i in 0..n} = hin1[i]/innorm[i];
var in2{i in 0..n} = hin2[i]/innorm[i];
var in3{i in 0..n} = hin3[i]/innorm[i];
var dg1{i in 0..n} = dgn[i]*in1[i]-dgr[i]*ir1[i];
var dg2{i in 0..n} = dgn[i]*in2[i]-dgr[i]*ir2[i];
var dg3{i in 0..n} = dgn[i]*in3[i]-dgr[i]*ir3[i];
var g1{i in 0..n} = ir1[i]*dg1[i]+ir2[i]*dg2[i]+ir3[i]*dg3[i];
var g2{i in 0..n} = it1[i]*dg1[i]+it2[i]*dg2[i]+it3[i]*dg3[i];
var g3{i in 0..n} = ih1[i]*dg1[i]+ih2[i]*dg2[i]+ih3[i]*dg3[i];
var T1{i in 0..n} = G0*T*(1+tau/100)/w[i]*ur[i];
var T2{i in 0..n} = G0*T*(1+tau/100)/w[i]*ut[i];
var T3{i in 0..n} = G0*T*(1+tau/100)/w[i]*uh[i];
var Delta1{i in 0..n} = g1[i]+T1[i];
var Delta2{i in 0..n} = g2[i]+T2[i];
var Delta3{i in 0..n} = g3[i]+T3[i];

let n := 400;

# Cost function

maximize weight: 10000*w[n];

# differential equations (trapezoidal rule of discretization)

de1 {i in 0..n-1}: p[i+1] = p[i]+step*(a12[i]*Delta2[i]+a12[i+1]*Delta2[i+1])/2;
de2 {i in 0..n-1}: f[i+1] = f[i]+step*(a21[i]*Delta1[i]+a22[i]*Delta2[i]+a23[i]*Delta3[i]
                   +a21[i+1]*Delta1[i+1]+a22[i+1]*Delta2[i+1]+a23[i+1]*Delta3[i+1])/2;
de3 {i in 0..n-1}: g[i+1] = g[i]+step*(a31[i]*Delta1[i]+a32[i]*Delta2[i]+a33[i]*Delta3[i]
                   +a31[i+1]*Delta1[i+1]+a32[i+1]*Delta2[i+1]+a33[i+1]*Delta3[i+1])/2;
de4 {i in 0..n-1}: h[i+1] = h[i]+step*(a43[i]*Delta3[i]+a43[i+1]*Delta3[i+1])/2;
de5 {i in 0..n-1}: k[i+1] = k[i]+step*(a53[i]*Delta3[i]+a53[i+1]*Delta3[i+1])/2;
de6 {i in 0..n-1}: L[i+1] = L[i]+step*(a63[i]*Delta3[i]+a63[i+1]*Delta3[i+1]
                          + sqrt(mu*p[i])*(q[i]/p[i])^2
                          + sqrt(mu*p[i+1])*(q[i+1]/p[i+1])^2)/2;
de7 {i in 0..n-1}: w[i+1] = w[i]-step*T/Isp*(1+tau/100);

# control constraints

cc {i in 0..n}: sqrt(ur[i]^2+ut[i]^2+uh[i]^2) = 1.0;
tc: tauL <= tau <= 0.0;
```

```
u1 {i in 0..n}: -1 <= ur[i] <= 1;
u2 {i in 0..n}: -1 <= ut[i] <= 1;
u3 {i in 0..n}: -1 <= uh[i] <= 1;

# state constraints


# boundary values

ic1: p[0] = p0;
ic2: f[0] = f0;
ic3: g[0] = g0;
ic4: h[0] = h0;
ic5: k[0] = k0;
ic6: L[0] = L0;
ic7: w[0] = w0;
if1: p[n] = pf;
if2: sqrt(f[n]^2+g[n]^2) = fgf;
if3: sqrt(h[n]^2+k[n]^2) = hkf;
if4: f[n]*h[n]+g[n]*k[n] = 0.0;
if5: g[n]*h[n]-f[n]*k[n] <= 0.0;

# initial estimates

let {i in 0..n} p[i] := p0+(pf-p0)*i/n;
let {i in 0..n} f[i] := -0.12*i/n;
let {i in 0..n} g[i] := 0.7*(i/n)^4;
let {i in 0..n} h[i] := -0.6*(i/n)^3;
let {i in 0..n} k[i] := 0.02*i/n;
let {i in 0..n} L[i] := (16*pi)*i/n;
let {i in 0..n} w[i] := w0+(wf-w0)*i/n;
let {i in 0..n} ur[i] := 0.1;
let {i in 0..n} ut[i] := 0.7-i/n;
let {i in 0..n} uh[i] := -0.1;
let tau := -9.0;
let tf := 90000.0;

option solver ipopt; # (Andreas Waechter, CMU/IBM)
# www-124.ibm.com/developerworks/opensource/coin/Ipopt

solve;

display _total_solve_time, tf, tau,
if2, if2.slack, if3, if3.slack, if4, if4.slack, if5, if5.slack;

printf {i in 0..n}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
tf*i/n, p[i], f[i], g[i], h[i], k[i], L[i], w[i], ur[i], ut[i], uh[i]
> orbit.out;

# gnuplot; plot 'orbit.out' using 1:2 w lines
```

Literaturverzeichnis

[1] J. S. Arora: Introduction to Optimum Design. McGraw-Hill, Boston, 1989.

[2] U. M. Asher, R. M. M. Mattheij, R. D. Russel: Numerical Solution of Boundary Value Problems for Ordinary Differential Equations. SIAM, Philadelphia, 1995.

[3] U. M. Asher, L. R. Petzold: Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM, Philadelphia, 1998.

[4] Joh. Bernoulli: Wenn in einer verticalen Ebene zwei Punkte A und B gegeben sind, soll man dem beweglichen Punkte M eine Bahn AMB anweisen, auf welcher er von A ausgehend vermöge seiner eigenen Schwere in kürzester Zeit nach B gelangt. Acta Eridutorum, Juni 1696.

[5] J. T. Betts: Practical Methods for Optimal Control Using Nonliear Programming. SIAM, Philadelphia, 2001.

[6] A. E. Bryson, Y. C. Ho: Applied Optimal Control. Blaisdell, Waltham, MA, 1969.

[7] A. E. Bryson: Dynamic Optimization. Addison-Wesley, Menlo Park, 1999.

[8] R. Bulirsch, E. Nerz, H. J. Pesch, O. von Stryk: Combining direct and indirect methods in optimal control: Range maximization of a hang glider. In: R. Bulirsch, A. Miele, J. Stoer, K. H. Well (eds.): Optimal Control. Birkhäuser, Boston, 1993.

[9] R. Bulirsch, D. Kraft: Computational Optimal Control. Birkhäuser, Basel, 1994.

[10] R. Byrd, M. E. Hribar, J. Nocedal: An interior point method for large scale nonlinear programming. SIAM J. Optimization **9** (1999) 877–900.

[11] Common Public License. http://www.opensource.org/licenses/cpl.php

[12] C. De Boor, B. Swartz: Collocation at gauusian points. SIAM J. Numerical Analysis **11** (1973) 916–921.

[13] J. W. Demmel: Numerical Linear Algebra. SIAM, Philadelphia, 1996.

[14] E. D. Dolan, J. J. More, T. S. Munson: Benchmarking Optimization with COPS 3.0. Argonne National Laboratory, Technical Report ANL/MCS-TM273, 2004.
s.a. http://www-unix.mcs.anl.gov/ more/cops/

[15] I. S. Duff: MA57–A code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Softw. **30** (2004) 118–144.

[16] S. O. Erb: Mit dem Schiff von A nach B. Institut für Flugmechanik und Flugregelung. Universität Stuttgart. Unveröff. Bericht. 2003.

[17] R. Fletcher, S. Leyffer: Nonlinear programming without a penalty function. Math. Program. **91** (2002) 239–269.

[18] A. Forsgren, P. E. Gill: Primal-Dual Interior Methods for Nonconvex Nonlinear Programming. SIAM J. Optimization **8** (1998) 1132–1152.

[19] A. Forsgren, P. E. Gill, M. H. Wright: Interior methods for nonlinear optimization. SIAM Review **44** (2002) 525–597.

[20] R. Fourer, D. M. Gay, B. W. Kernighan: AMPL – A Modeling Language for Mathematical Programming. Thomson, Pacific Grove, ^{2}2003.
S.a. http://www.ampl.com

[21] J. M. Gere, S. P. Timoshenko: Mechanics of Materials. PWS Publishing Company, Boston, ^{4}1997.

[22] P. E. Gill, W. Murray, M. A. Saunders: User's Guide for qpopt: a Fortran Package for Quadratic Programming. Report NA 95-1, Department of Mathematics, University of California, San Diego, 1995.
S.a. ftp://www.cam.ucsd.edu/pub/peg/reports/qpopt.ps

[23] P. E. Gill, W. Murray, M. A. Saunders: SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM Review **47** (2005) 99–131..

[24] G. H. Golub, C. F. Van Loan: Matrix Computations. Johns Hopkins University Press, Baltimore, ^{3}1997.

[25] N. I. M. Gould: Some reflections on the current state of active-set and interior-point methods for constrained optimization. SIAG/OPT Views-and-News **14** (2003) 2–7.

[26] A. Griewank: Evaluating Derivatives: Principles and Techniques for Automatic Differentiation. SIAM, Philadelphia, 2000.

[27] S.-P. Han: A globally convergent method for nonlinear programming. J. Opt. Theory Applic. **22** (1977) 297–309.

[28] A. Heck: Introduction to Maple. Springer-Verlag, New York, ^{3}2003.

[29] D. J. Higham, N. J. Higham: Matlab Guide. SIAM, Philadelphia, 2000.

[30] T. J. R. Hughes: The Finite Element Method. Prentice-Hall, Englewood Cliffs, 1987.

[31] D. J. Inman: Engineering Vibration. Prentice-Hall, Englewood Cliffs, 1994.

[32] F. Jarre, J. Stoer: Optimierung. Springer-Verlag, Berlin, 2004.

[33] N. Karmarkar: A new polynomial-time algorithm for linear programming. Combinatorica **4** (1984) 373–395.

[34] D. Kraft: On converting optimal control problems into nonlinear programming problems. In: K. Schittkowski (ed.): Computational Mathematical Programming. Springer-Verlag, Berlin, 1985.

[35] D. Kraft: TOMP – Fortran modules for optimal control calculations ACM Trans. Math. Softw. **20** (1994) 262–281.

[36] M. Mössner-Beigel: Optimale Steuerung für Industrieroboter unter Berücksichtigung der getriebebedingten Elastizität. IWR, Universität Heidelberg, Diplomarbeit 1999.

[37] C. Moler: Numerical Computing with MATLAB. SIAM, Philadelphia, 2004.
S.a. http://www.mathworks.com/moler/

[38] B. A. Murtagh, M. A. Saunders: A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. Math. Prog. Study **16** (1982) 84–117.

[39] B. A. Murtagh, M. A. Saunders: MINOS 5.5 User's Guide. Technical Report SOL-83-20R. Stanford University, 1998.

[40] J. Nocedal, S. J. Wright: Numerical Optimization. Springer-Verlag, New York, 1999.

[41] M. J. D. Powell: A fast algorithm for nonlinearly constrained optimization calculations. In: G. A. Watson (ed.): Numerical Analysis. Lecture Notes in Mathematics, Vol. 630. Springer-Verlag, Berlin, 1978.

[42] J. Stoer, R. Bulirsch: Einführung in die numerische Mathematik. Bd. 2. Springer-Verlag, Berlin, 1978.

[43] R. J. Vanderbei, D. F. Shanno: An interior-point algorithm for nonconvex nonlinear programming. Statistics and Operations Research, Princeton University, SOR-97-21. Princeton, 1997.

[44] R. J. Vanderbei: Linear Programming – Foundations and Extensions. Kluwer Academic Publishers, Boston, ^{2}2001.
S.a. die online-Ausgabe unter http://www.princeton.edu/~rvdb

[45] R. J. Vanderbei: LOQO User's Manual. Operations Research and Financial Engineering Technical Report No. ORFE-99-xx. Princeton University, Princeton, 2000.

[46] A. Wächter, L. T. Biegler: On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. IBM T.J. Watson Research Center, Yorktown, 2004.
s.a. http://projects.coin-or.org/Ipopt

[47] M. J. H. Walker, B. Ireland, J. Owens: A set of modified equinoctial orbit elements. Celestial Mechanics **36** (1985) 809–819.

[48] S. Wolfram: The Mathematica Book. Cambridge University Press, Cambridge, ^{4}1999.