Fraunhofer IITB Anwendungszentrum Systemtechnik

TECHNISCHE UNIVERSITÄT ILMENAU

Optimale Steuerung 2: Numerische Verfahren und Beispiele

Dr.-Ing. Eckhard Arnold

Fraunhofer-IITB, Anwendungszentrum Systemtechnik
Am Vogelherd 50, D-98693 Ilmenau
E-mail: Eckhard.Arnold@ast.iitb.fhg.de
Tel.: +49 (0) 3677/461-122
Fax: +49 (0) 3677/461-100

11. Januar 2005

Inhaltsverzeichnis

1 Übersicht

In diesem Dokument soll anhand von einfachen Beispielen der Einsatz von numerischen Methoden zur Lösung von Optimalsteuerungsaufgaben demonstriert werden. Der Schwerpunkt liegt dabei auf Verfahren die zur Lösung der Belegaufgaben zur Vorlesung „Optimale Steuerung 2“ Arnold (2004) verwendet werden können.

1. Direkte Lösungsverfahren:
   Approximation durch Nichtlineares Optimierungsproblem
   ohne explizite Verwendung der Optimalitätsbedingungen

   a) Mehrstufen-Steuerungsparametrisierung (Abschnitt 3)
   HQP (`http://sf.net/projects/hqp`)

   b) direkte Kollokation (Abschnitt 6)
   SNOPT (`http://scicomp.ucsd.edu/~peg`) zur Lösung des resultierenden großen nichtlinearen Optimierungsproblems

2. Indirekte Lösungsverfahren:
   Aufstellen der Optimalitätsbedingungen
   Formulierung als Zweipunkt- oder Mehrpunkt-Randwertaufgabe

   a) Schießverfahren (Abschnitt 4)
   numerische Integration des kanonischen Differentialgleichungssystems
   nichtlineares Gleichungssystem MATLAB: `ode45`, `fsolve`

   b) Kollokationsverfahren für Randwertaufgaben (Abschnitt 5)
   (großes) nichtlineares Gleichungssystem
   MATLAB: `bvp4c`

Beiliegend befinden sich die Quelltexte der Programme zur Lösung der Beispielaufgaben mit den angegebenen Verfahren.

Weitere Alternativen sind beispielsweise:

1. Lösung der Randwertaufgaben mit einem Tabellenkalkulationsprogramm (z. B. Microsoft Excel)
   `http://www.economics.ltsn.ac.uk/cheer/ch15_1/naevdal.htm`
   `http://www.nlh.no/ios/Publikasjoner/d2001/d2001-19.pdf`

2. direkte Kollokation nach Abschnitt 6 mit verfügbarer Software zur Lösung großer nichtlinearer Optimierungsprobleme, beispielsweise

   - IPOPT: `http://www.coin-or.org/Ipopt`

2 Beispielaufgaben

2.1 Doppelintegrator mit Steuerungsbeschränkung

Ein Doppelintegrator ist unter Berücksichtigung von Steuerungsbeschränkungen in einem vorgegebenen Zeithorizont (stell-)energieoptimal von einem festen Anfangs- in einen festen Endzustand zu überführen.

$$\dot{x}_1 = x_2 \tag{2.1a}$$
$$\dot{x}_2 = u \tag{2.1b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \mathbf{x}(1) = \mathbf{x}_f = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{2.1c}$$
$$J = \frac{1}{2} \int_0^1 u^2 dt \tag{2.1d}$$
$$|u| \leq u_{minmax} \tag{2.1e}$$

Mit der Hamiltonfunktion

$$H = \frac{1}{2} u^2 + p_1 x_2 + p_2 u \tag{2.2}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = 0 \tag{2.3a}$$
$$\dot{p}_2 = -p_1 \tag{2.3b}$$
$$u = \begin{cases} u_{minmax}, & \text{falls } -p_2 \geq u_{minmax} \\ -p_2, & \text{falls } -u_{minmax} < -p_2 < u_{minmax} \\ -u_{minmax}, & \text{falls } -p_2 \leq -u_{minmax} \end{cases} \tag{2.3c}$$

Eine Lösung der Aufgabe ist in Abschnitt 5 angegeben.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3	`Hqp_Beispiel/Prg_Di.[hC]`
Randwertaufgabe und Schießverfahren	4	`Matlab/shoot/doint_shoot.m`
Randwertaufgabe und Kollokation	5	`Matlab/bvp4c/doint_bvp.m`
Direkte Kollokation	6	`Matlab/dto/doint_dto.m`

2.2 Doppelintegrator mit Zustandsbeschränkung

Ein Doppelintegrator ist unter Berücksichtigung einer Zustandssbeschränkung in einem vorgegebenen Zeithorizont (stell-)energieoptimal von einem festen Anfangs- in einen festen Endzustand zu überführen.

$$\dot{x}_1 = x_2 \tag{2.4a}$$
$$\dot{x}_2 = u \tag{2.4b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \mathbf{x}(1) = \mathbf{x}_f = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{2.4c}$$
$$J = \frac{1}{2} \int_0^1 u^2 dt \tag{2.4d}$$
$$x_1 \leq x_{1,max} \tag{2.4e}$$

Eine Lösung der Aufgabe ist in Abschnitt 3.3.2 angegeben.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3	`Hqp_Beispiel/Prg_Di2.[hC]`
Direkte Kollokation	6	`Matlab/dto/doint2_dto.m`

2.3 Zeitoptimale Umsteuerung eines T_2-Glieds

Ein T_2-Glied ist unter Berücksichtigung von Steuerungsbeschränkungen zeitoptimal von einem festen Anfangs- in einen festen Endzustand zu überführen.

$$\dot{x}_1 = -0.5x_1 + x_2 \tag{2.5a}$$
$$\dot{x}_2 = -x_2 + u \tag{2.5b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \qquad \mathbf{x}(t_f) = \mathbf{x}_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{2.5c}$$
$$J = \int_0^{t_f} dt = t_f \tag{2.5d}$$
$$|u| \leq u_{minmax} \tag{2.5e}$$

Mit der Hamiltonfunktion

$$H = 1 + p_1(-0.5x_1 + x_2) + p_2(-x_2 + u) \tag{2.6}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = 0.5p_1 \tag{2.7a}$$
$$\dot{p}_2 = -p_1 + p_2 \tag{2.7b}$$
$$u = \begin{cases} u_{minmax}, & \text{falls } p_2 < 0 \\ -u_{minmax}, & \text{falls } p_2 > 0 \end{cases} \tag{2.7c}$$
$$H\big|_{t_f} = 0 \tag{2.7d}$$

Die optimale Steuerung weist das für zeitoptimale Umsteuerungsaufgaben linearer Systeme typische bang-bang-Verhalten auf. Nach dem Satz von Feldbaum ergibt sich für das vorliegende System 2. Ordnung mit reellen Eigenwerten ein Steuerungsverlauf mit maximal einem Umschaltzeitpunkt.

Lösungen der Aufgabe sind in den Abschnitten 3.3.1 und 4 angegeben.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3	`Hqp_Beispiel/Prg_T2Topt.[hC]`
Randwertaufgabe und Schießverfahren	4	`Matlab/shoot/t2topt_shoot.m`
Randwertaufgabe und Kollokation	5	`Matlab/bvp4c/t2topt_bvp.m`
Direkte Kollokation	6	`Matlab/dto/t2topt_dto.m`

2.4 Doppelintegrator mit singulärem Lösungsabschnitt

Die folgende Aufgabe ist Papageorgiou (1996) entnommen und wurde um eine Bewertung des Endzeitpunkts erweitert.

$$\dot{x}_1 = x_2 \tag{2.8a}$$
$$\dot{x}_2 = u \tag{2.8b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}, \qquad \mathbf{x}(t_f) = \mathbf{x}_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad t_f \text{ frei} \tag{2.8c}$$
$$J = \rho_{t_f} t_f + \frac{1}{2} \int_0^{t_f} \left(x_1^2 + x_2^2 \right) dt \tag{2.8d}$$
$$|u| \leq 1 \tag{2.8e}$$

Mit der Hamiltonfunktion

$$H = \frac{1}{2} x_1^2 + \frac{1}{2} x_2^2 + p_1 x_2 + p_2 u \tag{2.9}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = -x_1 \tag{2.10a}$$
$$\dot{p}_2 = -\left(x_2 + p_1 \right) \tag{2.10b}$$
$$u = \begin{cases} 1, & \text{falls } p_2 < 0 \\ -1, & \text{falls } p_2 > 0 \\ u_{sing}, & \text{falls } p_2 = 0 \end{cases} \tag{2.10c}$$
$$\rho_{t_f} + H\big|_{t_f} = 0 \tag{2.10d}$$

Unter Berücksichtigung der Randbedingungen (2.8c) ergibt sich aus (2.10d)

$$\rho_{t_f} + p_2(t_f) u(t_f) = 0 \tag{2.11}$$

Für eventuell auftretende singuläre Lösungsabschnitte $t = [t_{s1,i}, t_{s2,i}]$ muß gelten

$$p_2(t_{s1,i}) = 0 \tag{2.12a}$$
$$x_2(t_{s1,i}) + p_1(t_{s1,i}) = 0 \tag{2.12b}$$
$$u_{sing} = x_1 \qquad \text{für } t = [t_{s1,i}, t_{s2,i}] \tag{2.12c}$$

Eine Lösung der Aufgabe ist in Abschnitt 6 angegeben. In Abbildung 6.2 ist deutlich die Abfolge der Lösungsabschnitte (bang-bang-Verhalten bzw. singuläre Steuerung) zu erkennen.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3	`Hqp_Beispiel/Prg_Dising.[hC]`
Direkte Kollokation	6	`Matlab/dto/dising_dto.m`

Eine Lösung mit indirekten Verfahren ist schwierig, da offensichtlich sehr gute Startwerte für die Anfangs-Kozustände $\mathbf{p}(0)$ und die Dauer der Lösungsabschnitte benötigt werden.

3 Mehrstufen-Steuerungsparametrisierung mit HQP

3.1 Mehrstufen-Ansatz

HQP, siehe Franke (1998), ist eine Implementierung einer Mehrstufen-Steuerungsparametrisierung zur Lösung beschränkter Optimalsteuerungsprobleme

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad t \in [t_0, t_f],\ t \neq \tau_i \tag{3.1a}$$

$$\mathbf{g}(\mathbf{x}, \mathbf{u}, t) \leq \mathbf{0}, \quad t \in [t_0, t_f], \tag{3.1b}$$

$$\mathbf{h}(\mathbf{x}(t_f), t_f) = \mathbf{0} \tag{3.1c}$$

$$J = F(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} f_0(\mathbf{x}, \mathbf{u}, t)dt \longrightarrow \text{min!} \tag{3.1d}$$

Die Anfangszustände $\mathbf{x}(t_0)$, die Endzustände $\mathbf{x}(t_f)$ und der Zeithorizont $[t_0, t_f]$ können dabei fest, d. h. vorgegeben, oder frei sein.

Der Zeithorizont $[t_0, t_f]$ wird in K Zeitabschnitte (Zeitstufen) unterteilt

$$t_0 = t^0 < t^1 < \cdots < t^K = t_f \tag{3.2}$$

Das Optimalsteuerungsproblem wird nun durch ein nichtlineares Optimierungsproblem in den Steuergrößen $\mathbf{u}^k$ und den Zustandsgrößen $\mathbf{x}^k$ der Zeitstufen approximiert.

Die $\mathbf{u}^k$ sind Ansatzparameter eines parametrischen Ansatzes zur Beschreibung des Zeitverlaufs der kontinuierlichen Steuergrößen im Zeitabschnitt k, wobei im einfachsten Fall ein stufenförmiger Verlauf $\mathbf{u}(t) = \mathbf{u}^k$ angenommen wird. Die $\mathbf{x}^k$ setzen sich aus zeitdiskreten Zuständen $\mathbf{x}_d^k$ und den Anfangswerten der kontinuierlichen Zustandsgrößen zu Beginn des Zeitabschnitts zusammen. Die numerische Integration der Zustandsdifferentialgleichungen im Zeitabschnitte k liefert damit den Verlauf $\tilde{\mathbf{x}}^k(t)$.

$$\mathbf{u}(t) = \boldsymbol{\phi}^k(\mathbf{u}^k, t), \qquad t \in [t^k, t^{k+1}] \tag{3.3a}$$

$$\mathbf{x}^k = \begin{bmatrix} \mathbf{x}_d^k \\ \tilde{\mathbf{x}}^k(t^k) \end{bmatrix} \tag{3.3b}$$

$$\dot{\tilde{\mathbf{x}}}^k(t) = \mathbf{f}(\tilde{\mathbf{x}}^k(t), \boldsymbol{\phi}^k(\mathbf{u}^k, t), t), \quad t \in [t^k, t^{k+1}] \tag{3.3c}$$

Die Gleichungsbeschränkungen (Stufengleichungen) des nichtlinearen Optimierungsproblems

$$\mathbf{x}^{k+1} = \begin{bmatrix} \mathbf{f}_d^k(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})) \\ \tilde{\mathbf{x}}^k(t^{k+1}) \end{bmatrix}, \qquad k = 0, \ldots, K-1 \tag{3.4}$$

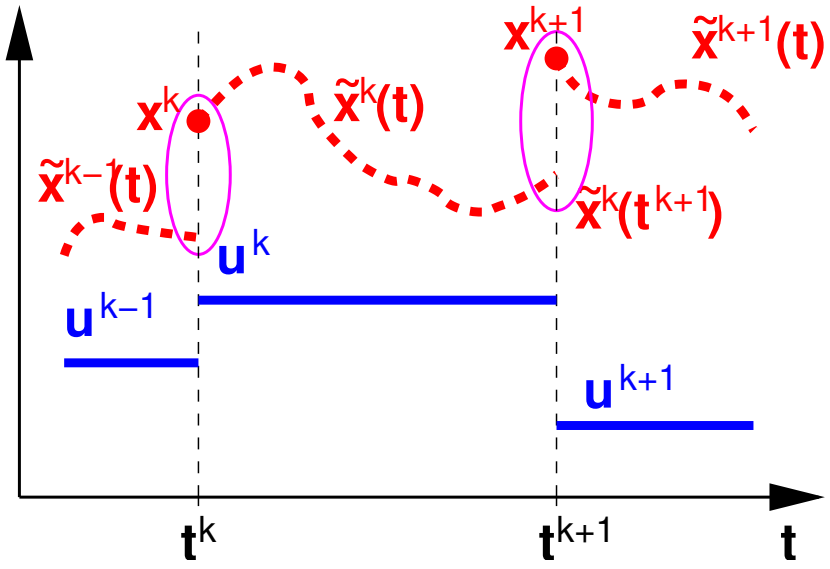

Abbildung 3.1: Mehrstufen-Steuerungsparametrisierung.

sichern dabei die Stetigkeit der Approximationen der kontinuierlichen Zustandsgrößen, siehe Abbildung 3.1.

Ungleichungsbeschränkungen

$$\mathbf{u}^k_{\min} \le \mathbf{u}^k \le \mathbf{u}^k_{\max}, \qquad k = 0, \ldots, K-1 \tag{3.5a}$$

$$\mathbf{x}^k_{\min} \le \mathbf{x}^k \le \mathbf{x}^k_{\max}, \qquad k = 0, \ldots, K \tag{3.5b}$$

$$\mathbf{c}^k_{\min} \le \mathbf{c}^k\left(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})\right) \le \mathbf{c}^k_{\max}, \qquad k = 0, \ldots, K-1 \tag{3.5c}$$

$$\mathbf{c}^K_{\min} \le \mathbf{c}^k\left(\mathbf{x}^K\right) \le \mathbf{c}^K_{\max} \tag{3.5d}$$

approximieren die Ungleichungsbeschränkungen (3.1b) des Optimalsteuerungsproblems. Durch geeignete Wahl von $\mathbf{x}^k_{\min}$, $\mathbf{x}^k_{\max}$, $\mathbf{c}^K_{\min}$ und $\mathbf{c}^K_{\max}$ können feste Anfangs- bzw. Endzustände sowie allgemeine Endbedingungen (3.1c) berücksichtigt werden.

Die Zielfunktion des nichtlinearen Optimierungsproblems wird nach

$$J = f_0^K(\mathbf{x}^K) + \sum_{k=0}^{K-1} f_0^k\left(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})\right) \tag{3.6}$$

gebildet. Ein Integralterm (Lagrange-Term) in (3.1d) ist daher gegebenfalls durch Einführung einer zusätzlichen Zustandsgröße

$$\dot{x}_{n+1} = f_0^K(\mathbf{x}, \mathbf{u}, t), \quad \text{mit } x_{n+1}(t_0) = 0, \tag{3.7a}$$

$$f_0^k = \tilde{x}^k_{n+1}(t^{k+1}) - x^k_{n+1}, \qquad k = 0, \ldots, K-1 \tag{3.7b}$$

einzubeziehen.

Eine freie Endzeit t_f des Optimalsteuerungsproblems kann durch Transformation auf einen festen Zeithorizont

$$t = t_0 + \tau(t_f - t_0), \quad \tau \in [0, 1] \tag{3.8a}$$

$$\frac{d\mathbf{x}}{d\tau} = \frac{d\mathbf{x}}{dt} \cdot \frac{dt}{d\tau} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t_0 + \tau t_f) \cdot (t_f - t_0) \tag{3.8b}$$

$$\int_{t_0}^{t_f} f_0(\mathbf{x}, \mathbf{u}, t)dt = \int_0^1 f_0(\mathbf{x}, \mathbf{u}, t_0 + \tau t_f) \cdot (t_f - t_0)d\tau \tag{3.8c}$$

und Betrachtung des Parameters t_f als (konstante) zeitdiskrete Zustandsgröße mit freiem Anfangswert in das Mehrstufenproblem einbezogen werden

$$x_d^{k+1} = x_d^k = t_f, \qquad k = 0, \ldots, K-1 \qquad \text{mit } x_d^0 = t_f \text{ frei} \tag{3.9}$$

3.2 Implementierung

HQP besteht aus mehreren Modulen, siehe Franke (1998), u. a.

- der SQP-Solver löst große, strukturierte, nichtlineare Optimierungsproblem mit einem SQP-(sequential quadratic programming)-Verfahren,
- der QP-Solver löst die linear-quadratischen Teilprobleme mit einem Interior-Point-Algorithmus,
- im Interface Omuses ist eine Mehrstufen-Steuerungsparametrisierung implementiert, die Optimalsteuerungsaufgaben durch nichtlineare Optimierungsproblem approximiert,
- der Matrix-Solver dient der Lösung der linearen Gleichungssysteme (unter Ausnutzung der Besetztheitsstruktur der Koeffizientenmatrizen) und basiert auf der Meschach-Library, siehe Stewart (1992),
- benötigte Ableitungen werden mittels automatischer Differentiation bestimmt, hierzu wird die Bibliothek ADOL-C eingesetzt, siehe Griewank u. a. (1996),
- verschiedene ODE/DAE-Solver dienen zur numerischen Integration der Zustandsdifferentialgleichungen, hier werden neben expliziten Runge-Kutta-Verfahren auch komplexe DAE-Solver wie DASSL, siehe Brenan u. a. (1989), verwendet.

Wesentliche Teile von HQP sind in C++ (und C) implementiert, einige externe Bibliotheken bzw. Solver (z. B. DASSL) in FORTRAN. HQP setzt die Skriptsprache Tcl zur Ablaufsteuerung ein, so daß eine einfache Bedienbarkeit von der Kommandozeile (Tcl-Shell) oder mittels Tcl-Skript gegeben ist.

3.3 Einbindung einer Optimalsteuerungsaufgabe in HQP

3.3.1 Zeitoptimale Umsteuerung eines T_2-Glieds

Zur Einbindung einer Optimalsteuerungsaufgabe in HQP müssen die Komponenten des Problems in einer (von der Klasse `Omu_Program` abgeleiteten) C ++ -Klasse bereitgestellt werden. Dies soll im folgenden am Beispiel der zeitoptimalen Umsteuerung des T_2-Glieds aus Abschnitt 2.3 demonstriert werden. Die Programmdateien finden sich im Verzeichnis `Hqp_Beispiel`.

Die Header-Datei `Prg_T2Topt.h` beinhaltet die Klassendeklaration. In der Methode

```
char *name() {return "T2Topt";}
```

wird der Name des Optimierungsproblems festgelegt, unter dem später die Aufgabe ausgewählt und initialisiert werden kann, siehe Listing 3.7 Zeile 67.

Die Klassendefinition erfolgt in `Prg_T2Topt.C`. Im (optionalen) Konstruktor werden Klassenparameter (Variable) initialisiert, beispielsweise in Zeile 18 des folgenden Listings 3.1 mit `set_K()` die Anzahl K der Zeitstufen. Soll von der Tcl-Ebene (Kommandozeile oder Skript) lesend oder schreibend auf Parameter zugegriffen werden, so müssen analog zu Zeile 21 und 22 entsprechende Interface-Elemente vorgesehen werden.

Listing 3.1: `Prg_T2Topt.C`, Konstruktor.

```
16 Prg_T2Topt::Prg_T2Topt()
   {
18     set_K(50);          // Anzahl Stufen (stages)
       _uminmax = 5.0;     // Steuerungsbeschraenkung
20     _tf = 2.0;          // Startnaeherung Endzeit
       _ifList.append(new If_Real("prg_uminmax", &_uminmax)); // Tcl-
       Interface
22     _ifList.append(new If_Real("prg_tf", &_tf));           // Tcl-
       Interface
   }
```

Ein gesonderter Destruktor ist für den dargestellten einfachen Anwendungsfall nicht notwendig. Gegebenenfalls könnte dort von der Klasse belegter dynamischer Speicher freigegeben werden.

In der Methode `setup_stages()` sind die Zeitstufen entsprechend (3.2) festzulegen. In den meisten Fällen kann das durch einen Aufruf von `stages_alloc()` (Zeile 28) unter Angabe der Anzahl der Zeitstufen K, eines Parameters, der die Anzahl interner Abtastzeitpunkte in den Zeitstufen festlegt (meist 1) und des (festen) Zeithorizonts – hier des entsprechend (3.8) auf [0.0, 1.0] normierten Zeithorizonts – erfolgen.

Die Methode `setup_stages()` wird vor Beginn des eigentlichen Optimierungslaufs einmalig aufgerufen.

Listing 3.2: `Prg_T2Topt.C`, Methode `setup_stages()`.

```
26 void Prg_T2Topt::setup_stages(IVECP ks, VECP ts)
   {
28     stages_alloc(ks, ts, K(), 1, 0.0, 1.0); // normierter Zeithorizont
                                               // [0.0, 1.0]
30 }
```

Die Methode `setup()` in Listing 3.3 wird vor Beginn des eigentlichen Optimierungslaufs einmalig für jede Zeitstufe k aufgerufen. Es sind die Anzahl der Zustandsgrößen $\mathbf{x}^k$ (Zeile 36), der Steuergrößen $\mathbf{u}^k$ (Zeile 52) und gegebenenfalls der allgemeinen Beschränkungen $\mathbf{c}^k$ in jeder Zeitstufe anzugeben. Dabei ist zu beachten, daß entsprechend den C-Konventionen die Indizierung der Vektoren mit 0 beginnt. Wenn zeitdiskrete Zustandsgrößen $\mathbf{x}_d^k$ vorgesehen sind, so stehen diese auf den ersten Indizes im Vektor $\mathbf{x}^k$ (also bei 0 beginnend).

Für die letzte Zeitstufe K ist kein Steuervektor $\mathbf{u}^K$ vorgesehen.

Weiterhin sind in `setup()` die Komponenten der Ungleichungsbeschränkungen (3.5) festzulegen, beispielsweise in den Zeilen 53 und 54. Bei Gleichheit von oberer und unterer Schranke wird die Komponente intern als Gleichungsbeschränkung behandelt. Dies wird zur Festlegung von festen Anfangs- (Zeilen 40 und 41) oder Endzuständen (Zeilen 48 und 49) genutzt.

Für jede Variable kann mit Hilfe der Komponenten `x.initial` bzw. `u.initial` eine numerische Initialisierung vorgenommen werden, Zeilen 43, 44, 45, 55. Dies dient der Festlegung von *Startnäherungen* der Zustands- und Steuergrößen und ist nicht mit den *Anfangswerten* der Zustandsdifferentialgleichung zu verwechseln.

Besonders wichtig ist die Vorgabe sinnvoller Startnäherungen für Größen, die die Längen von Zeitintervallen beschreiben, beispielsweise `x[0]` als t_f (Zeilen 20, 43). Solche Optimierungsvariable sollten zusätzlich auf einen sinnvollen Bereich eingeschränkt werden, beispielsweise $t_f \geq 0.1$ in Zeile 39. Die Einhaltung der Komponentenbeschränkungen (3.5a) und (3.5b) im Laufe des iterativen Lösungsprozesses ist gesichert, sofern die Startnäherungen im zulässigen Bereich liegen.

Listing 3.3: `Prg_T2Topt.C`, Methode `setup()`.

```
   void Prg_T2Topt::setup(int k,
34                        Omu_Vector &x, Omu_Vector &u, Omu_Vector &c)
   {
36     x.alloc(1+2);          // Anzahl Zustandsgroessen
                              // (1 zeitdiskret, 2 kontinuierlich)
38     if ( k == 0 ) {
         x.min[0] = 0.1; // freier Anfangswert zeitdiskrete Zustandsgroesse
           x0
40       x.min[1] = x.max[1] = -1.0; // fester Anfangszustand x1(0)
         x.min[2] = x.max[2] = 0.0;  // fester Anfangszustand x2(0)
42       // numerische Initialisierung (Startnaeherung)
         x.initial[0] = _tf;
44       x.initial[1] = -1.0;
         x.initial[2] = 0.0;
```

```
46      }
        else if ( k == K() ) {
48          x.min[1] = x.max[1] = 0.0; // fester Anfangszustand x1(tf)
            x.min[2] = x.max[2] = 0.0; // fester Anfangszustand x2(tf)
50      }
        if ( k < K() ) {
52          u.alloc(1);                // Anzahl Steuergroessen
            u.min[0] = -_uminmax;      // untere Schranke u
54          u.max[0] = _uminmax;       // obere Schranke u
            u.initial[0] = 0.0;        // numerische Initialisierung
56      }
}
```

Mit der (optionalen) Methode `init_simulation()` kann vor Beginn des eigentlichen Optimierungslaufs eine problemangepaßte numerische Initialisierung vorgenommen werden. Die Methode wird für jede Zeitstufe k aufgerufen, und zwischen den Aufrufen werden die kontinuierlichen Zustandsgleichungen integriert. Daher sind beim Aufruf für $k \geq 1$ die Zustandsgrößen `x` mit den Endwerten des vorangegangenen Zeitschritts belegt. Damit ist ein Simulationslauf mit stetigen Übergängen der kontinuierlichen Zustandsgrößen zwischen den Zeitschritten realisierbar.

Listing 3.4: `Prg_T2Topt.C`, Methode `init_simulation()`.

```
64  void Prg_T2Topt::init_simulation(int k,
                                     Omu_Vector &x, Omu_Vector &u)
66  {
        int i;
68      // Initialisierung in 1. Zeitstufe (k=0)
        // sonst Uebernahme Endwerte der vorherigen Stufe (default)
70      if ( k == 0 ) {
            for ( i = 0; i < (int) x->dim; i++ )
72              x[i] = x.initial[i];
        }
74      // Initialisierung Steuergroesse
        if ( k < K() )
76          u[0] = u.initial[0];
}
```

In der Methode `update()` werden für jede Zeitstufe k die Stufengleichungen (zeitdiskreter Anteil) $\mathbf{f}_d^k$ (3.4) (Parameter `f`), der Anteil an der Zielfunktion f_0^k (3.6) (Parameter `f0`) sowie die Ungleichungsbeschränkungen $\mathbf{c}^k$ (3.5c) bzw. (3.5d) (Parameter `c`) in Abhängigkeit von $\mathbf{x}^k$ und $\mathbf{u}^k$ (Parameter `x` und `u`) und $\tilde{\mathbf{x}}^k(t^{k+1})$ (Parameter `f` bei Aufruf) berechnet.

Der Index `kk` in den Zeilen 82 und 86 bezieht sich auf die oben erwähnten Abtastzeitpunkte, die hier mit den Zeitstufen k zusammenfallen.

Es ist zu beachten, daß entsprechend den C-Konventionen die Indizierung der Vektoren mit 0 beginnt. Beispielsweise bezeichnet `x[0]` die erste Zustandsvariable, hier den zeitdiskreten Zustand zur Modellierung der freien Endzeit nach Gleichung (3.9). Für die letzte Zeitstufe K (`KK`) ist kein Steuervektor `u` und keine Stufengleichung `f` vorgesehen.

Listing 3.5: `Prg_T2Topt.C`, Methode `update()`.

```
82 void Prg_T2Topt::update(int kk,
                          const adoublev &x, const adoublev &u,
84                        adoublev &f, adoublev &f0, adoublev &c)
{
86   if ( kk < KK() )
       f[0] = x[0];   // zeitdiskrete Zustandsgroesse x0(k+1) = x0(k)
88   else
       f0 = x[0];     // Zielfunktional: Endzeit
90 }
```

Sämtliche von `x` und `u` abhängige Zwischengrößen, die in die Berechnung von `f`, `f0` oder `c` eingehen müssen *aktive* Variable vom ADOL-C-Datentyp `adouble` (oder `adoublev`) sein, beispielsweise

```
adouble temp;
temp = 5.0*x[2]+u[0];
f0 = temp;
```

Nur so ist die Auswertung der Ausdrücke mittels automatischer Differentiation möglich. Sollen bedingte Anweisungen (`if-then-else`) o. ä. verwendet werden, sind die Hinweise in Griewank u. a. (1996) zu berücksichtigen.

Dies gilt sinngemäß auch für die Methode `continuous()`, die die kontinuierlichen Zustandsdifferentialgleichungen bereitstellt. Diese sind in *impliziter* Form

$$\mathbf{F} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) - \dot{\mathbf{x}} \tag{3.10}$$

anzugeben. Die implizite Darstellung ist vorgegeben, weil mit HQP auch Optimalsteuerungsprobleme mit Zustandsgleichungen in DAE-Form (differential-algebraische Gleichungen) gelöst werden können. Bei den hier betrachteten Aufgaben mit Zustandsgleichungen in ODE-Form dürfen die zeitlichen Ableitungen `xp[]` keinesfalls weggelassen werden, da das dadurch definierte DAE-System eine völlig anderes Systemverhalten aufweisen würde.

Die Unterscheidung zwischen zeitdiskreten und kontinuierlichen Zustandsgrößen wird anhand des Vektors `F` getroffen: ab dem ersten belegten Element (hier Index 1) werden die zugehörigen Zustandsgrößen als kontinuierlich betrachtet.

In den Zeilen 99 und 100 des Listings ist die Zeitnormierung entsprechend (3.8b) berücksichtigt, die Zustandsgröße `x[0]` ist zeitdiskret.

Listing 3.6: `Prg_T2Topt.C`, Methode `continuous()`.

```
93 void Prg_T2Topt::continuous(int kk, double t,
                              const adoublev &x, const adoublev &u,
95                            const adoublev &xp, adoublev &F)
{
97   // x0: zeitdiskrete Zustandsgroesse x0(k)=tf
     // Zustandsgleichungen mit Zeitnormierung in Nullform
99   F[1] = (-0.5*x[1]+x[2])*x[0] - xp[1];
```

```
    F[2] = (-x[2]+u[0])      *x[0] - xp[2];
}
```

Das Hauptprogramm und die Initialisierung der Tcl-Erweiterung findet sich in im C-Modul `Hqp_Beispiel.c`. Mit

```
Tcl_SetVar(interp, "tcl_rcFileName", "./Hqp_Beispiel.tcl",
TCL_GLOBAL_ONLY);
```

wird dort festgelegt, daß das Tcl-Skript `Hqp_Beispiel.tcl` (siehe unten) beim Programmstart ausgeführt wird.

Die C- und C++ -Module müssen übersetzt und mit den HQP- und Tcl-Bibliotheken gebunden werden. Diese Schritte sind für `make` in `Makefile` und für die Verwendung der Entwicklungsumgebung Dev-C++ in der Projektdatei `Hqp_Beispiel.dev` vorbereitet.

Mit dem Tcl-Skript `Hqp_Beispiel.tcl` erfolgt die Ablaufsteuerung. Durch Aufruf des Kommandos `prg_name` (Zeile 67) wird das Optimalsteuerungsproblem ausgewählt und die Klasse durch den Konstruktoraufruf initialisiert. `prg_setup` (Zeile 74) ruft die Methoden `setup_stages()` und `setup()`, `prg_simulate` (Zeile 76) die Methode `init_simulation()` mit den entsprechenden Parametern auf. Mit `sqp_init` (Zeile 78) wird der SQP-Solver initialisiert und mit dem Aufruf von `hqp_solve` (Zeile 80) der eigentliche Optimierungslauf gestartet.

Sollte das Optimierungsproblem keine zulässige Lösung besitzen oder der SQP-Algorithmus nicht konvergieren, bricht `hqp_solve` mit einer Fehlermeldung ab. Dieser Abbruch wird mit dem Tcl-Kommando `catch` abgefangen. Im Erfolgsfall wird die Lösung ausgewertet und mit dem Aufruf von `toASCII` (Zeile 88) in eine Textdatei geschrieben.

Die Tcl-Kommandos können auch interaktiv von der Kommandozeile aus eingegeben werden.

Listing 3.7: `Hqp_Beispiel.tcl`

```
65 foreach beispiel [list Di Di2 T2Topt Dising] {
       # Auswahl Optimalsteuerungsproblem
67     prg_name $beispiel
       # Initialisierung Problem: setup_stages(), setup()
74     prg_setup
       # Initialisierung Optimierungsvariable: simulate()
76     prg_simulate
       # Initialisierung Solver
78     sqp_init
       # Start Optimierungslauf
80     catch hqp_solve result
       puts [format "Status:          %s" $result]
82     if { $result == "optimal" } {
           puts [format "Zielfunktional: %g" [prg_f]]
84         puts [format "Rechenzeit:     %.1fs" [toc]]
           # Ausgabe [t x u] in ASCII-Datei
86         # Einlesen z.B. in Matlab:
           #     [t x1 x2 x3 u] = textread('Di_Ergebnis.txt','%f %f %f %f %f')
           ;
```

```
88          toASCII [prg_name]_Ergebnis.txt
            puts [format "Ergebnisse nach %s\n\n" [prg_name]_Ergebnis.txt]
90      }
}
```

Der Optimierungslauf für die zeitoptimale Umsteuerung des T_2-Glieds liefert die folgenden Ausgaben.

```
 it          obj    ||inf||   ||grdL|| [ qp res]      ||s||       s'Qs stepsize
  0            1     0.6065          1 [ 21 opt]          5      1e-05        1
  1          0.1     0.1066  5.224e-07 [ 28 opt]      5.343  8.525e-06      0.1
  2     0.165523    0.09595  0.0009487 [  8 opt]      4.809     0.1613        1
  3      0.77527    0.05515    0.00499 [ 31 opt]      1.848    0.01027        1
  4     0.821131   0.001442   0.005273 [ 30 opt]      4.832   0.002767        1
  5     0.815004  0.0005784   0.004824 [ 12 opt]      0.698  1.251e-05        1
  6     0.814645  4.932e-06    0.00111 [ 13 opt]  7.001e-06  2.051e-13        1
  7     0.814645  4.197e-14  8.616e-08
                                       143 qp-it
Status:          optimal
Zielfunktional: 0.814645
Rechenzeit:     0.7s
```

Dabei bezeichnet `it` den Iterationszähler, `obj` den Wert der Zielfunktion, `||inf||` die Norm der Verletzung von Gleichungs- und Ungleichungsbeschränkungen und `||grdL||` die Norm des Gradienten der dem nichtlinearen Optimierungsproblem zugeordneten Lagrangefunktion. In der Spalte `[ qp res]` sind die Anzahl der Interior-Point-Iterationsschritte zur Lösung des unterlagerten QP-Problems und eine Statusmeldung ausgegeben. `||s||` bezeichnet die Norm des Suchrichtungsvektors, `s'Qs` die aus Suchrichtung und Approximation der Hesse-Matrix gebildete quadratische Form und `stepsize` den Schrittweitenfaktor.

Hier ist nach sieben Iterationsschritten ein Abbruchkriterium erfüllt und der Iterationslauf wird mit dem Status `optimal` beendet. Typisch ist die geringe Änderung der Zielfunktion in den letzten (hier: drei) Iterationen, die Reduktion der Norm der Beschänkungsverletzung und die der Suchrichtung auf sehr kleine positive numerische Werte. Die Statusmeldung der QP-Iteration sollte immer `opt` sein, kann aber gelegentlich in der/den ersten SQP-Iteration(en) davon abweichen. Der Schrittweitenfaktor ist meist gleich 1, meist erfolgt nur in den ersten SQP-Iterationen eine Reduktion.

Die Ergebnisdatei kann beispielsweise in MATLAB mit

```
[t tf x1 x2 u] = textread('T2Topt_Ergebnis.txt','%f %f %f %f %f');
```

eingelesen werden. Ein Vergleich der grafischen Darstellung der Ergebnisse in Abbildung 3.2 mit Abbildung 4.1 zeigt – neben dem Fehlen der hier nicht berechneten Kozustandsgrößen und der Hamiltonfunktion – geringfügige Abweichungen im Verlauf der Steuergröße in der Umgebung des Sprungzeitpunkts.

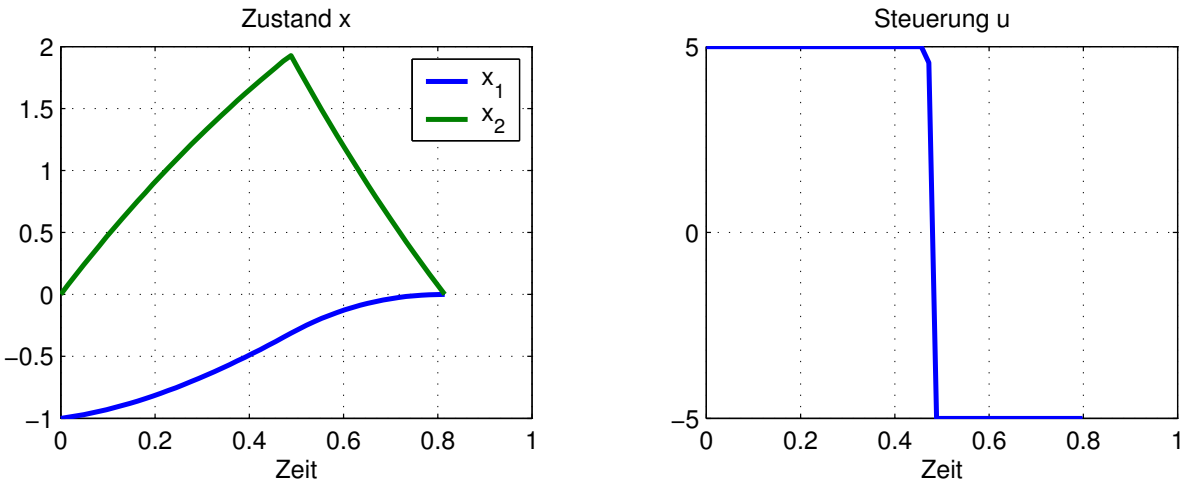

Abbildung 3.2: Zeitoptimale Umsteuerung T_2-Glied; Mehrstufen-Steuerungsparametrisierung mit HQP.

3.3.2 Doppelintegrator mit Zustandsbeschränkung

Zur Einbindung einer Optimalsteuerungsaufgabe in HQP müssen die Komponenten des Problems in einer (von der Klasse `Omu_Program` abgeleiteten) C++ -Klasse bereitgestellt werden. Im folgenden sollen für den Doppelintegrator mit Zustandsbeschränkung die wesentlichen Unterschiede zur Aufgabe des vorangehenden Abschnitts, d. h.

- die Zustandsbeschränkung (2.4e),
- die feste Endzeit t_f sowie
- der Integralterm im Zielfunktional (2.4d)

dargestellt werden. Die Programmdateien finden sich wieder im Verzeichnis `Hqp_Beispiel`.

Die Header-Datei `Prg_Di2.h` beinhaltet die Klassendeklaration. In der Methode

```
char *name() {return "Di2";}
```

wird der Name des Optimierungsproblems festgelegt.

Die Klassendefinition erfolgt in `Prg_Di2.C`. Im Konstruktor werden Klassenparameter (Variable) initialisiert.

Listing 3.8: `Prg_Di2.C`, Konstruktor.

```
16 Prg_Di2::Prg_Di2()
   {
18     set_K(50);          // Anzahl Stufen (stages)
       _x1max = 0.12;      // Zustandssbeschraenkung
20     _ifList.append(new If_Real("prg_x1max", &_x1max)); // Tcl-Interface
   }
```

In der Methode `setup()` in Listing 3.9 wird die Zustandsbeschränkung (2.4e) als obere Schranke für die Zustandsgrößen der Zeitstufen angegeben, Zeile 48.

Da die Endzeit t_f fest ist, wird die Zeittransformation (3.8) hier nicht benötigt und die zugehörige (zeitdiskrete) Zustandsvariable kann entfallen. `x[0]` bezeichnet somit die Zustandsvariable x_1.

Die zusätzliche Zustandsgröße `x[2]` wird gemäß (3.7) zur Berechnung des Integralterms im Zielfunktional eingeführt.

Listing 3.9: `Prg_Di2.C`, Methode `setup()`.

```
32 void Prg_Di2::setup(int k,
                       Omu_Vector &x, Omu_Vector &u, Omu_Vector &c)
{
34     x.alloc(2+1);          // Anzahl Zustandsgroessen (2 + Intergralterm)
       if ( k == 0 ) {
36         x.min[0] = x.max[0] = 0.0; // fester Anfangszustand x1(0)
           x.min[1] = x.max[1] = 1.0; // fester Anfangszustand x2(0)
38         x.min[2] = x.max[2] = 0.0; // fester Anfangswert zusaetzlicher
           Zustand
           // numerische Initialisierung (Startnaeherung)
40         x.initial[0] = 0.0;
           x.initial[1] = 0.0;
42         x.initial[2] = 0.0;
       }
44     else if ( k == K() ) {
           x.min[0] = x.max[0] = 0.0; // fester Endzustand x1(1)
46         x.min[1] = x.max[1] = -1.0;// fester Endzustand x2(1)
       } else
48         x.max[0] = _x1max;          // Zustandsbeschraenkung x(1)<=_x1max
       if ( k < K() ) {
50         u.alloc(1);                 // Anzahl Steuergroessen
           u.initial[0] = 2.0;         // numerische Initialisierung
52     }
}
```

In der Methode `update()` wird für jede Zeitstufe k der Anteil f_0^k (3.6) an der Zielfunktion (Parameter `f0`) in Abhängigkeit von $\mathbf{x}^k$ und $\mathbf{u}^k$ (Parameter `x` und `u`) und $\tilde{\mathbf{x}}^k(t^{k+1})$ (Parameter `f` bei Aufruf) berechnet, siehe Gleichung (3.7).

Listing 3.10: `Prg_Di2.C`, Methode `update()`.

```
78 void Prg_Di2::update(int kk,
                        const adoublev &x, const adoublev &u,
80                      adoublev &f, adouble &f0, adoublev &c)
{
82     if ( kk < KK() )
           f0 = f[2]-x[2]; // Zielfunktional: Anteil Zeitstufe
84 }
```

Die Methode `continuous()` stellt die kontinuierlichen Zustandsdifferentialgleichungen bereit. Die Gleichung für die zusätzliche Zustandsgröße ergibt sich aus dem Integranden des Zielfunktionals.

Listing 3.11: `Prg_Di2.C`, Methode `continuous()`.

```
void Prg_Di2::continuous(int kk, double t,
                         const adoublev &x, const adoublev &u,
                         const adoublev &xp, adoublev &F)
{
    // Zustandsgleichungen in Nullform
    F[0] = x[1]           - xp[0];
    F[1] = u[0]           - xp[1];
    F[2] = 0.5*u[0]*u[0] - xp[2];
}
```

Der Optimierungslauf für die Umsteuerung des Doppelintegrators mit Zustandsbeschränkung liefert die folgenden Ausgaben.

```
 it          obj    ||inf||   ||grdL|| [ qp res]      ||s||       s'Qs stepsize
  0            2          3       0.04 [ 18 opt]          8      19.41        1
  1      3.70657     0.5486  3.022e-07 [  8 opt]      9.707  1.328e-05        1
  2      3.70657  1.454e-14  1.142e-07
                                        26 qp-it
Status:          optimal
Zielfunktional: 3.70657
Rechenzeit:     0.3s
```

Eine grafische Darstellung der Ergebnisse findet sich in Abbildung 3.3.

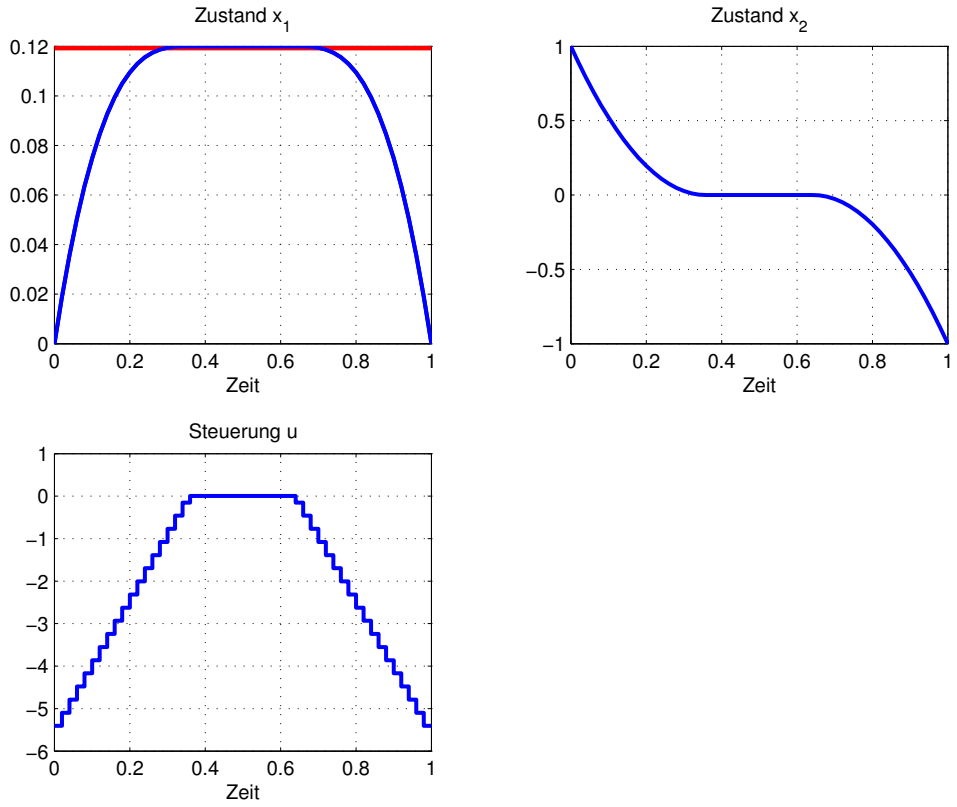

Abbildung 3.3: Doppelintegrator mit Zustandsbeschränkung; Mehrstufen-Steuerungsparametrisierung mit HQP.

4 Randwertaufgabe und Schießverfahren mit MATLAB

Die Auswertung der Optimalitätsbedingungen für ein Optimalsteuerungsproblem (3.1) führt im allgemeinen auf Mehrpunkt-Randwertaufgaben für das aus Zustands- und Kozustandsdifferentialgleichungen gebildete kanonische Differentialgleichungssystem. Dabei wird vorausgesetzt, daß die Schaltstruktur, d. h. die Abfolge der Zeitabschnitte mit jeweils unterschiedlichen Sätzen an Optimalitätsbedingungen (singuläre Abschnitte, aktive Zustandsbeschränkungen etc.) im Optimierungshorizont bekannt ist.

Für Optimalsteuerungsprobleme mit aktiven Zustandsbeschränkungen ergeben sich hierbei nicht zu unterschätzende Schwierigkeiten, so daß im Rahmen der Belegaufgaben „Optimale Steuerung 2“ der Einsatz von indirekten Verfahren nicht empfohlen wird.

Im Fall der zeitoptimalen Umsteuerung eines T_2-Glieds nach Abschnitt 2.3 ergibt sich die Zweipunkt-Randwertaufgabe

$$\dot{\mathbf{y}} = \underbrace{\begin{bmatrix} -0.5 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix}}_{\mathbf{A}_{kanon}} \mathbf{y} + \begin{bmatrix} 0 \\ u(p_2) \\ 0 \\ 0 \end{bmatrix}, \quad \text{mit } \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix}, \quad u(p_2) \text{ nach (2.7c)} \tag{4.1a}$$

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \mathbf{x}_0 \tag{4.1b}$$

$$\begin{bmatrix} y_1(t_f) \\ y_2(t_f) \end{bmatrix} = \mathbf{x}_f \tag{4.1c}$$

$$H\big|_{t_f} = 0 \tag{4.1d}$$

Bei einem Schießverfahren zur Lösung dieser Zweipunkt-Randwertaufgabe betrachtet man die Bedingungen am rechten Rand t_f des Optimierungshorizonts als Funktionen der nicht vorgegebenen Anfangswerte am linken Rand (hier der Anfangs-Kozustände $\mathbf{p}(0)$) und der weiteren freien Parameter (hier t_f). Das dadurch gebildete nichtlineare Gleichungssystem wird (iterativ) gelöst, wobei jede Auswertung der nichtlinearen Gleichungen die Lösung der zugehörigen Anfangswertaufgabe erfordert.

Aus Gründen der numerischen Genauigkeit ist es sinnvoll, nicht mit einem variablen Zeithorizont zu arbeiten, sondern die Aufgabe mit freiem Zeithorizont zunächst in eine Aufgabe mit festem Horizont zu transformieren.

An den Umschaltzeitpunkten der bang-bang-Steuerung (bei Nulldurchgang $p_2(t)$) ändert sich die rechte Seite der Differentialgleichung (4.1a) sprungartig. Die numerische Integration muß

aus Genauigkeitsgründen gestoppt und nach dem Umschaltvorgang neu gestartet werden. Auch hier ist eine Transformation auf Zeitintervalle fester Dauer und die Festlegung der Abfolge der Werte der Steuergrößen sinnvoll.

Diese beiden Transformationen lassen sich zu einer Transformation auf $\tau \in [0, 2]$ zusammenfassen (hier für genau einen Umschaltzeitpunt t_s)

$$t = \begin{cases} \tau t_s & \text{für } 0 \le \tau \le 1 \\ t_s + (\tau - 1)(t_f - t_s) & \text{für } 1 \le \tau \le 2 \end{cases} \tag{4.2}$$

Die Anfangswertaufgabe wird dann abschnittsweise gelöst (numerisch integriert), wobei nun angenommen wird, daß u im 1. Teilintervall auf der oberen und im 2. Teilintervall auf der unteren Schranke liegt.

$$\frac{d\mathbf{y}}{d\tau} = \mathbf{A}_{kanon}\mathbf{y} \cdot t_s + \begin{bmatrix} 0 \\ u_{minmax} \\ 0 \\ 0 \end{bmatrix}, \qquad 0 \le \tau \le 1, \quad \mathbf{y}(0) = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{p}(0) \end{bmatrix} \tag{4.3a}$$

$$\frac{d\mathbf{y}}{d\tau} = \mathbf{A}_{kanon}\mathbf{y} \cdot (t_f - t_s) + \begin{bmatrix} 0 \\ -u_{minmax} \\ 0 \\ 0 \end{bmatrix}, \quad 1 \le \tau \le 2 \tag{4.3b}$$

$\mathbf{y}(\tau)$ ist in $\tau = 1$ stetig.

Das mit dem Schießverfahren zu lösende nichtlineare Gleichungssystem setzt sich aus den verbleibenden Bedingungen am rechten Rand (4.1c) und (4.1d) sowie der Umschaltbedingung

$$y_3(\tau = 1) = p_2(t = t_s) = 0 \tag{4.4}$$

zusammen. Damit handelt es sich um eine echte Mehrpunkt-Randwertaufgabe.

Die m-Funktion `Matlab/shoot/t2topt_shoot.m`, siehe Listing 4.1, realisiert das Schießverfahren für die zeitoptimale Umsteuerung des T_2-Glieds. In Zeile 8 werden mit `odeset` sehr hohe Genauigkeitsforderungen für die numerische Integration eingestellt. Dies ist notwendig, da bei der Ableitungsberechnung mittels finiter Differenzen in `fsolve` numerische Fehler in der Lösung der Anfangswertaufgabe verstärkt werden. In den Zeilen 18–20 werden die Startnäherungen für die freien Variablen des Schießverfahrens festgelegt. Hier sind verschiedene Einstellungen zu testen, da das Verfahren sehr empfindlich hinsichtlich dieser Werte ist. Bei ungeeigneten Startwerten konvergiert der iterative Prozeß zumeist nicht gegen eine Nullstelle des nichtlinearen Gleichungssystems.

In Zeile 22 erfolgt dann der eigentliche Aufruf von `fsolve`, dem MATLAB-Standardsolver für nichtlineare Gleichungssysteme. Übergeben wird die Funktion, die die nichtlinearen Gleichungen auswertet als „function handle“ der Unterfunktion („subfunction“) `@shoot` und die Startnäherungen der Parameter. Bei erfolgreichem Abschluß wird der Lösungsvektor `w` zurückgegeben und anschließend ausgewertet.

Die Zeitverläufe der Lösungen des kanonischen Differentialgleichungssystems werden in der Matrix `tyu` abgespeichert. In Zeile 29 wird die Zeitachse entsprechend (4.2) entnormiert und in Zeile 30 die Hamiltonfunktion H berechnet.

Listing 4.1: `t2topt_shoot.m`

```
1  function t2topt_shoot(uminmax_)
   global x0 xf uminmax tyu odeoptions
8  odeoptions = odeset('AbsTol', 1e-10, 'RelTol', 1e-10, 'Refine', 10);
   % Steuerungsbeschraenkungen
15 uminmax = uminmax_;
   x0 = [-1; 0]; % Anfangswerte
17 xf = [0; 0];  % Endwerte
   ts = 0.5;     % Startnaeherung Schaltzeit
19 tf = 1;       % Startnaeherung Endzeit
   p0 = [-1; -0.1];% Startnaeherung Anfangskozustand
21 % Loesung nchtlineares Gleichungssystem
   w = fsolve(@shoot, [p0; ts; tf], optimset('Display','iter'));
23 % Auswertung Loesung
   ts = w(3);
25 tf = w(end);
   fprintf('\nOptimale Schaltzeit ts = %g\nOptimale Endzeit tf = %g\n', ...
27     ts , tf)
   t = tyu(:, 1);
29 t = [t(t<=1.0)*ts; ts+(t(t>1.0)-1.0)*(tf-ts)]; % Entnormierung
   H = 1+tyu(:, 4).*(-0.5*tyu(:, 2)+tyu(:, 3))+...
31     tyu(:, 5).*(-tyu(:, 3)+tyu(:, 6));
   clear global x0 xf uminmax tyu odeoptions rhs_par
```

Die rechten Seiten der Differentialgleichungen (4.3) sind in der Unterfunktion `rhs` implementiert, Listing 4.2, Zeile 51. Mit Hilfe der globalen Variable `rhs_par` (Zeile 46 bzw. 58) wird die Information zum Lösungsabschnitt, d. h. der Wert der Steuergröße `u` und des Zeitskalierungsfaktors `dt_dtnorm`, übermittelt.

Listing 4.2: `t2topt_shoot.m`, Unterfunktion `rhs`.

```
44 function yp = rhs(t, y)
   % rhs - kanonisches Dgl.-System
46 global rhs_par
   x = y(1:2);
48 p = y(3:4);
   u = rhs_par(1);
50 dt_dtnorm = rhs_par(2);
   yp = [-0.5*x(1)+x(2); -x(2)+u; 0.5*p(1); -p(1)+p(2)]*dt_dtnorm;
```

In der Unterfunktion `shoot`, die von `fsolve` aufgerufen wird, werden die Restfehler (Residuen) der nichtlinearen Gleichungen in Abhängigkeit von den von `fsolve` vorgegebenen Parameterwerten ausgewertet. Mit dem Parametervektor `w` werden Näherungswerte für die Anfangswerte der Kozustands-Differentialgleichung (Zeile 59), die Umschaltzeit (Zeile 60) und die Endzeit

(Zeile 61) übergeben. Mit der bedingten Anweisung in Zeilen 62–65 wird „unsinnigen“ Parameterwerten wie negativen Umschalt- oder Endzeiten, aber auch Werten von $q_2(0)$, die in Widerspruch zur Schaltbedingung (2.7c) stehen, sehr große Funktionswerte zugeordnet, die eine weitere Fortsetzung der Nullstellensuche in einer solchen Richtung verhindern.

In den Zeilen 67 und 68 werden die Werte für die Steuergröße und den Zeitskalierungsfaktor für den ersten Lösungsabschnitt $0 \leq \tau \leq 1$ gesetzt. Anschließend wird mir dem Aufruf von `ode45` die Anfangswertaufgabe (4.3a) gelöst. Dabei wird die Funktion zur Berechnung der zeitlichen Ableitungen als „function handle“ `@rhs`, der Zeithorizont für die normierte Zeit τ, die Anfangswerte und der in Zeile 8 festgelegte Optionenvektor übergeben.

In Zeile 70 wird der Lösungsverlauf für die Ergebnisausgabe abgespeichert. In Zeile 71 wird der Restfehler (Residuum) der Schaltbedingung (4.4) berechnet.

In den Zeilen 73 und 74 werden die Werte für die Steuergröße und den Zeitskalierungsfaktor für den zweiten Lösungsabschnitt $1 \leq \tau \leq 2$ gesetzt. Anschließend wird mir dem Aufruf von `ode45` die Anfangswertaufgabe (4.3b) gelöst, wobei als Anfangswerte $\mathbf{y}(\tau = 1)$ die Endwerte des ersten Lösungsabschnitts verwendet werden.

Abschließend werden die Hamiltonfunktion zum Endzeitpunkt $H|_{t_f}$ berechnet (Zeile 80) und die Restfehler der Endzustandsbedingung (4.1c) (Zeile 82) sowie der Transversalitätsbedingung (4.1d) (Zeile 83) im Rückgabeparameter `res` gespeichert.

Listing 4.3: `t2topt_shoot.m`, Unterfunktion `shoot`.

```
56 function res = shoot(w)
   % shoot - Integration kanonisches Dgl.-System und Auswertung OB
58 global x0 xf uminmax tyu odeoptions rhs_par
   p0 = w(1:2);
60 ts = w(3);
   tf = w(end);
62 if ts <= 0 | tf <= ts | p0(2) > 0
       res = inf*ones(size(w));
64     return;
   end
66 % 1. bang-bang-Abschnitt: 0<=t<=ts, 0<=tnorm<=1
   rhs_par(1) = uminmax;
68 rhs_par(2) = ts;        % t = tnorm*ts
   [t, y] = ode45(@rhs, [0 1], [x0; p0], odeoptions);
70 tyu = [t y uminmax*ones(size(t))];
   res(1) = y(end, 4);        % p2(ts) = 0
72 % 2. bang-bang-Abschnitt: ts<=t<=tf, 1<=tnorm<=2
   rhs_par(1) = -uminmax;
74 rhs_par(2) = tf-ts;    % t = ts+(tnorm-1)*(tf-ts)
   [t, y] = ode45(@rhs, [1 2], y(end, :)', odeoptions);
76 tyu = [tyu; t y -uminmax*ones(size(t))];
   x_tf = y(end, 1:2)';
78 p_tf = y(end, 3:4)';
   u_tf = tyu(end, 6);
80 H_tf = 1+p_tf(1)*(-0.5*x_tf(1)+x_tf(2))+p_tf(2)*(-x_tf(2)+u_tf);
   res = [res; ...
```

```
82      x_tf-xf; ...    % x(tf) = xf
        H_tf];          % H(tf) = 0
```

`fsolve` liefert die folgende Ausgabe.

```
                                         Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          5         32.7875                         101             1
     1         10        0.281224       0.599136          5.66             1
     2         15      0.00281587       0.111136         0.604           1.5
     3         20     1.10261e-009      0.00598119      0.000378           1.5
     4         25     2.30142e-025     3.82718e-006      5.47e-012          1.5
Optimization terminated: first-order optimality is less than options.TolFun.
```

Nach vier Iterationen ist die Abbruchbedingung erfüllt, der verbleibende Restfehler in den Rand- und Transversalitätsbedingungen (`f(x)` – Summe der Fehlerquadrate) ist sehr klein.

Abschließend sind die im Schießverfahren nicht explizit berücksichtigten Komponenten der Optimalitätsbedingungen (hier die Vorzeichenbedingung für den Kozustand $p_2(t)$ entsprechend (2.7c)) zu überprüfen. Gegebenenfalls ist die Schaltstruktur zu ändern und erneut die zugehörige Mehrpunkt-Randwertaufgabe aufzustellen und zu lösen.

Die Zeitverläufe in Abbildung 4.1 zeigen, daß die angenommene Schaltstruktur korrekt ist, zum Schaltzeitpunkt wechselt $p_2(t)$ von negativen zu positiven Zahlenwerten. Für die optimale Schaltzeit ergibt sich ein Wert von $t_s = 0.488063$, die Endzeit ist $t_f = 0.814619$.

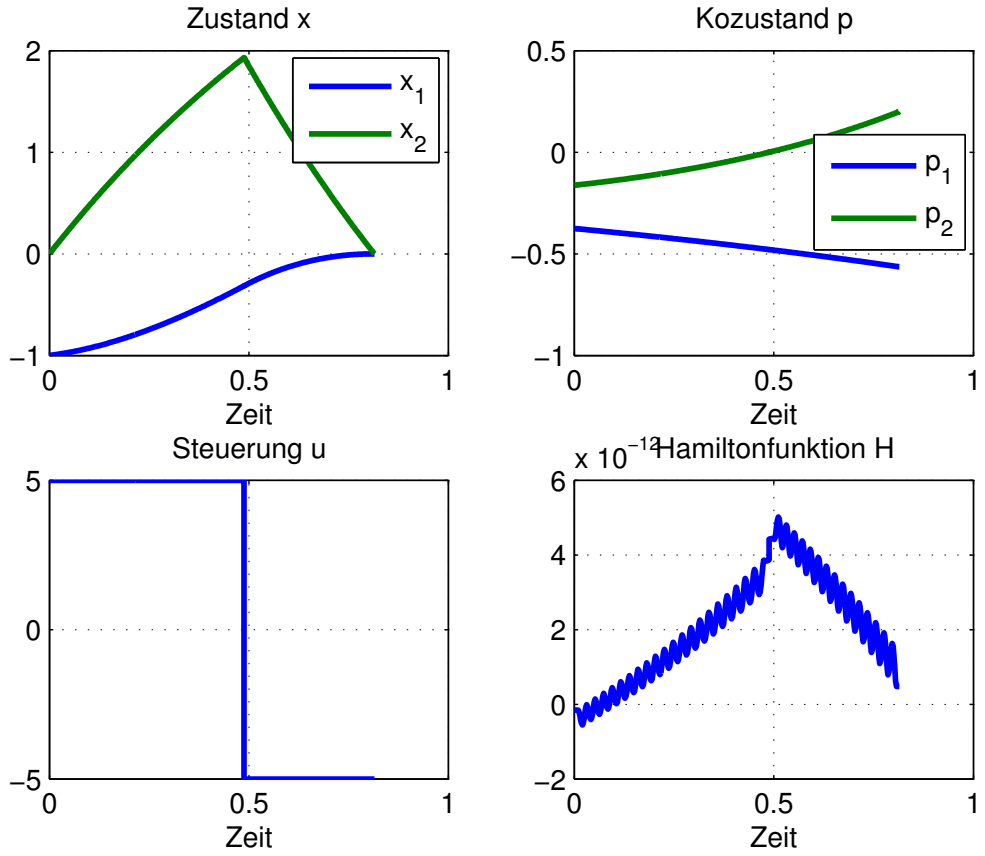

Abbildung 4.1: Zeitoptimale Umsteuerung T_2-Glied; Lösung der Randwertaufgabe mit Schießverfahren.

5 Randwertaufgabe und Kollokation mit MATLAB

Alternativ zu dem im Abschnitt 4 erläuterten Schießverfahren können Randwertaufgaben auch mit Kollokationsverfahren gelöst werden. Voraussetzung hierzu sind wiederum die Optimalitätsbedingungen und die Kenntnis der Schaltstruktur, so daß die zugehörige Mehrpunkt-Randwertaufgabe formuliert werden kann[1].

Im Fall des Doppelintegrators mit Steuerungsbeschränkung nach Abschnitt 2.1 ergibt sich die Zweipunkt-Randwertaufgabe

$$\dot{\mathbf{y}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\mathbf{A}_{kanon}} \mathbf{y} + \begin{bmatrix} 0 \\ u(p_2) \\ 0 \\ 0 \end{bmatrix}, \quad \text{mit } \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix}, \quad u(p_2) \text{ nach (2.3c)} \tag{5.1a}$$

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \mathbf{x}_0 \tag{5.1b}$$

$$\begin{bmatrix} y_1(1) \\ y_2(1) \end{bmatrix} = \mathbf{x}_f \tag{5.1c}$$

Beim Kollokationsverfahren werden die zu bestimmenden Zeitfunktionen (hier $\mathbf{y}(t)$) durch parametrische Ansätze beschrieben, beispielsweise durch Spline-Funktionen. Diese Ansatzfunktionen werden im allgemeinen die Differentialgleichungen nicht erfüllen, daher fordert man die Übereinstimmung der zeitlichen Ableitung der Ansatzfunktion mit der rechten Seite der Differentialgleichung an einer bestimmten Anzahl von sogenannten Kollokationspunkten. Dies führt auf ein nichtlineares Gleichungssystem zur Bestimmung der Parameter der Ansatzfunktionen, in das die Randbedingungen (hier (5.1b) und (5.1c)) einbezogen werden können. Die Anzahl der Kollokationspunkte wird dabei so gewählt, daß die Gesamtzahl der nichtlinearen Gleichungen mit der Anzahl der Ansatzparameter übereinstimmt.

In MATLAB steht mit der Funktion `bvp4c` ein derartiges Kollokationsverfahren zur Verfügung. Eine Anleitung und diverse Beispiele finden sich unter `http://www.mathworks.com/bvp_tutorial` und in Shampine u. a. (2003). Die folgenden Erläuterungen beziehen sich auf die MATLAB-Version 7.0 (R14), vorhergehende Versionen verwenden teilweise andere Funktionsbezeichnungen und unterscheiden sich in der Implementierung von Mehrpunkt-Randwertaufgaben.

In der m-Funktion `Matlab/bvp4c/doint_bvp.m`, siehe Listing 5.1, wird das Kollokationsverfahren zur Lösung der Zweipunkt-Randwertaufgabe für den Doppelintegrator mit Steuerungsbeschränkung angewandt. In Zeile 7 werden globale Variable zur Parameterübergabe an die

[1] Siehe Hinweis auf Seite 21, 2. Absatz.

Unterfunktionen vereinbart. In Zeile 27 wird mit dem Aufruf von `bvpinit` das Lösungsverfahren initialisiert. Der erste Aufrufparameter gibt ein Zeitgitter für die Splinefunktionen vor, das bis auf die Randpunkte später vom Kollokationsverfahren verfeinert und an die Zeitverläufe angepaßt wird. Der zweite Aufrufparameter gibt konstante Startnäherungen für die vier Komponenten des Vektors $\mathbf{y}(t)$ vor. Dies ist im Fall des Doppelintegrators ausreichend.

Das Konvergenzverhalten des Kollokationsverfahren hängt sehr stark von der vorgegebenen Startnäherung für die Funktionsverläufe ab. Die Bestimmung geeigneter Zeitverläufe der Kozustandsgrößen kann bei anspruchsvolleren Aufgaben problematisch sein.

In Zeile 29 erfolgt dann der eigentliche Aufruf des Kollokationsverfahrens. Es werden zwei Funktionen als „function handles“ `@ode` und `@bc` übergeben, die erste berechnet die zeitliche Ableitung der Funktionen aus der Differentialgleichung, die zweite den Restfehler der Randbedingungen. Weiterhin wird die Initiallösung und ein Satz von Einstellungen für Abbruchschranken und Zwischenausgaben an `bvp4c` übergeben.

Mit dem Aufruf von `deval` in Zeile 38 werden die Splineapproximationen an beliebig vorgebbaren Zeitpunkten ausgewertet. Aus dem Lösungsvektor werden dann die Variablen des Optimalsteuerungsproblems rückgerechnet (Zeilen 39–43). In Zeile 44 wird der optimale Wert des Zielfunktionals mittels Trapezregel berechnet.

Listing 5.1: `doint_bvp.m`

```
function doint_bvp(nr, uminmax_)
7  global x0 xf uminmax
   % Anfangs- und Endwerte
14      x0 = [0; 0];
        xf = [1; 0];
21 % Steuerungsbeschraenkungen
   uminmax = uminmax_;
26 % Initialisierung Loesung
   solinit = bvpinit(linspace(0, 1, 10), [0.5 1 0 0]);
28 % Loesung RW-Aufgabe
   sol = bvp4c(@ode, @bc, solinit, bvpset('stats', 'on', 'RelTol', 1e-6));
30 % Auswertung Loesung
   t = linspace(0, 1, 100)';
38      y = deval(sol, t);  % >= R14
   x = y(1:2, :)';
41 p = y(3:4, :)';
   u = max(-uminmax, min(-p(:, 2), uminmax));
43 H = 0.5*u.^2+p(:,1).*x(:,2)+p(:,2).*u;
   J = 0.5*trapz(t, u.^2);
45 fprintf('\nOptimaler Wert Zielfunktional: %g\n', J)
   clear global x0 xf uminmax
```

Die Funktion `ode` berechnet die Ableitungen der Zeitfunktionen anhand des kanonischen Differentialgleichungssystems (5.1a).

Listing 5.2: `doint_bvp.m`, Unterfunktion `ode`.

```
55 %————————    kanonisches Dgl.-System    ————————
```

```
function yd = ode(t, y)
global uminmax
% y = [x(1), x(2), p(1), p(2)]'
u = max(-uminmax, min(-y(4), uminmax));
yd = [y(2); u; 0; -y(3)];
```

Die Funktion `bc` wird von `bvp4c` mit den aktuellen Näherungen der Randwerte $\mathbf{y}(t_0)$ und $\mathbf{y}(t_f)$ aufgerufen. Im Rückgabeparameter werden die Restfehler der Randbedingungen erwartet (Zeile 65).

Listing 5.3: `doint_bvp.m`, Unterfunktion `res`.

```
%----------   Residuum Randbedingungen   ----------
function res = bc(y_t0, y_tf)
global x0 xf
res = [y_t0(1:2)-x0; y_tf(1:2)-xf];
```

Für den Doppelintegrator mit Steuerungsbeschränkung erhält man die folgenden Ausgaben, die optimalen Zeitverläufe sind in Abbildung 5.1 dargestellt.

```
>> doint_bvp(1, 4.5)
The solution was obtained on a mesh of 14 points.
The maximum residual is 3.121e-007.
There were 2533 calls to the ODE function.
There were 298 calls to the BC function.

Optimaler Wert Zielfunktional: 6.22856
```

Anspruchsvollere Aufgaben, wie z. B. Mehrpunkt-Randwertaufgaben mit zusätzlichen freien Parametern erfordern einige Erweiterungen der Vorgehensweise. Dies soll an der zeitoptimalen Umsteuerungsaufgabe für das T_2-Glied nach Abschnitt 2.3 demonstriert werden. Da die Randpunkte der Teilintervalle in `bvp4c` fest sind, muß die Mehrpunkt-Randwertaufgabe mit freier Umschalt- und Endzeit – wie Abschnitt 4 erläutert – in eine Aufgabe mit zwei Zeitintervallen fester Länge überführt werden.

In Zeile 27 des folgenden Listings 5.4 erfolgt wieder die Initialisierung durch den Aufruf von `bvpinit`. Die Trennstelle der beiden Teilintervalle der Mehrpunkt-Randwertaufgabe werden durch die doppelte Angabe des Zeitpunkts `1.0` im Zeitgitter für die Splinefunktionen gekennzeichnet. Weiterhin wird als dritter Aufrufparameter ein Vektor mit Startnäherungen der freien Parametern der Randwertaufgabe (hier: Umschaltzeit t_s und Endzeit t_f) übergeben.

Bei bestimmten Aufgaben kann zusätzlich eine zeitvariante Initialisierung der Zeitfunktionen über einen Funktionsaufruf der Form `y=init(t)` notwendig sein. Dann ist dieser Funktionsname als zweiter Aufrufparameter an `bvpinit` zu übergeben.

Zeile 34 und 35 zeigen, wie auf die Lösungen für die freien Parameter der Randwertaufgabe zugegriffen werden kann.

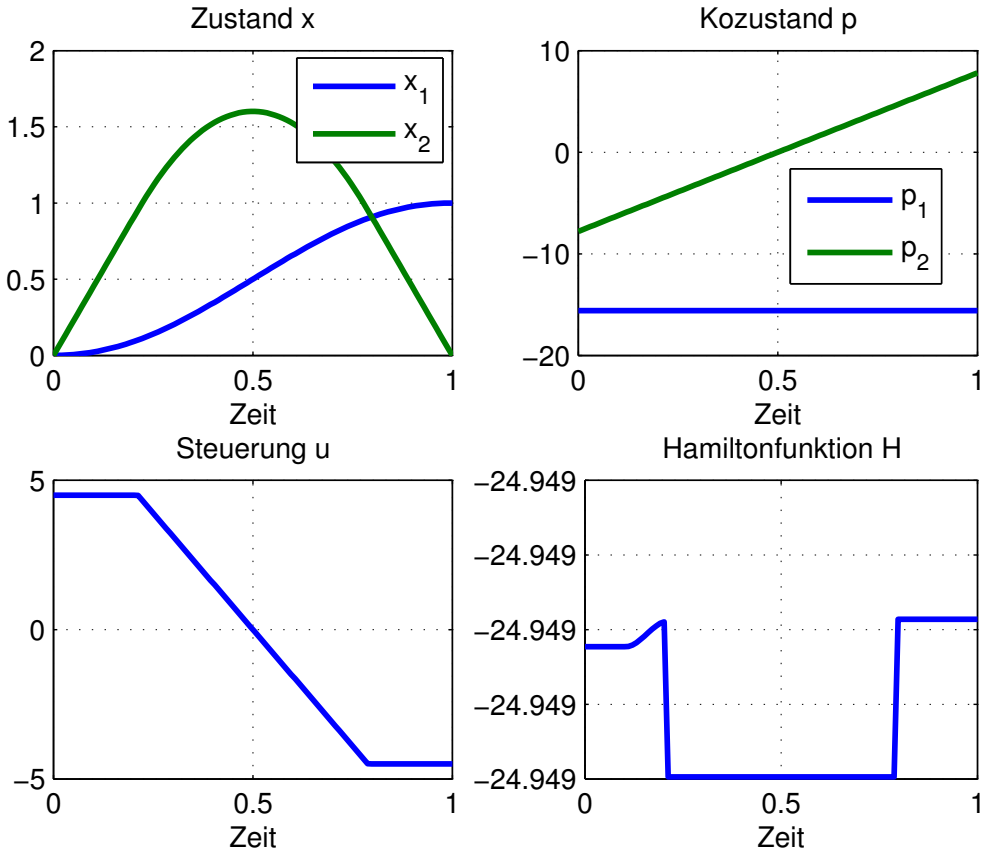

Abbildung 5.1: Doppelintegrator mit Steuerungsbeschränkung; Lösung der Randwertaufgabe mit BVP4C.

In den Zeile 57-71 findet sich die Funktion `ode`. Als zusätzliche Parameter werden nun das aktuelle Zeitintervall `int_nr` der Mehrpunkt-Randwertaufgabe und die Werte der freien Parameter übergeben.

In den Zeilen 76-89 findet sich die Funktion `bc`. Auch hier werden zusätzlich die Werte der freien Parameter der Aufgabe übergeben. Die Parameter `y_0` und `y_f` sind jetzt Matrizen, deren Spalten den Teilintervallen der Mehrpunkt-Randwertaufgabe zugeordnet sind. Die erste Spalte von `y_0` enthält die aktuellen Näherungswerte für $\mathbf{y}(t_0)$, d. h. die Werte am Beginn des ersten Teilintervalls. In der zweiten Spalte sind entsprechend die Werte für den Beginn des zweiten Teilintervalls $\mathbf{y}(t_s + 0)$ gespeichert. `y_f` enthält die Werte an den Endpunkten der Teilintervalle, also $\mathbf{y}(t_s - 0)$ und $\mathbf{y}(t_f)$.

Zu den Rand- und Transversalitätsbedingungen der Optimalsteuerungsaufgabe und der Schaltbedingung für den Kozustand $p_2(t_s)$ kommen die Bedingungen für einen stetigen Übergang der Zustands- und Kozustandsgrößen an der Trennstelle der Teilintervalle $\mathbf{y}(t_s - 0) = \mathbf{y}(t_s + 0)$ hinzu (Zeile 88).

Listing 5.4: `t2topt_bvp.m`

```
23 % Initialisierung Loesung
24 % Zeitnormierung:
25 %    1. Teilintervall (u = uminmax)   0 <= t <= ts --> 0 <= tnorm <= 1
26 %    2. Teilintervall (u = -uminmax) ts <= t <= tf --> 1 <= tnorm <= 2
27 solinit = bvpinit([linspace(0, 1.0, 10) linspace(1.0, 2.0, 10)], ...
28     [0 0 0 0], [ts; tf]);
29 % Loesung RW-Aufgabe
30 sol = bvp4c(@ode, @bc, solinit, bvpset('stats', 'on', 'RelTol', 1e-6));
31 % Auswertung Loesung
32 ts = sol.parameters(1);
33 tf = sol.parameters(2);
57 function yd = ode(t, y, int_nr, par)
58 % kanonisches Dgl.-System
59 global uminmax
60 x = y(1:2);
61 p = y(3:4);
62 ts = par(1);
63 tf = par(2);
64 if int_nr == 1            % 1. Teilintervall
65     u = uminmax;
66     dt_dtnorm = ts;       % t = tnorm*ts;
67 else                      % 2. Teilintervall
68     u = -uminmax;
69     dt_dtnorm = tf-ts;    % t = ts+(tnorm-1.0)*(tf-ts);
70 end
71 yd = [-0.5*x(1)+x(2); -x(2)+u; 0.5*p(1); -p(1)+p(2)]*dt_dtnorm;
76 function res = bc(y_0, y_f, par)
77 % Residuen Rand- und Transversalitaetsbedingungen
78 global x0 xf uminmax
79 x_0 = y_0(1:2, 1);                % x(0)
80 x_f = y_f(1:2, 2);                % x(tf)
81 p_f = y_f(3:4, 2);                % p(tf)
82 p_s = y_f(3:4, 1);                % p(ts)
83 u_f = -uminmax;                   % u(tf)
84 H_f = 1+p_f(1)*(-0.5*x_f(1)+x_f(2))+p_f(2)*(-x_f(2)+u_f); % H(tf)
85 res = [x_0-x0; ...                % x(0)  = x0
86     x_f-xf; ...                   % x(tf) = xf
87     H_f; ...                      % H(tf) = 0
88     y_f(:, 1)-y_0(:, 2); ...      % Stetigkeit x, p in ts
89     p_s(2)];                      % Schaltbedingung
```

6 Direkte Kollokation mit SNOPT und MATLAB

Bei direkten Kollokationsverfahren wird die Optimalsteuerungsaufgabe durch ein nichtlineares Optimierungsproblem approximiert. Die Auswertung von Optimalitätsbedingungen ist nicht erforderlich.

Wie im Zusammenhang mit den Kollokationsverfahren zur Lösung der Randwertaufgabe in Abschnitt 5 erläutert, besteht das Grundprinzip eines Kollokationsansatzes darin, die zu bestimmenden Zeitverläufe durch einen parametrischen Ansatz zu beschreiben und die Parameter durch Auswertung der Differentialgleichung an einer endlichen Anzahl von Kollokationspunkten zu bestimmen.

Beim direkten Kollokationsverfahren werden die Verläufe der Zustands- und Steuergrößen durch Ansatzfunktionen beschrieben und die Kollokationsbedingungen als Gleichungsbeschränkungen in ein nichtlineares Optimierungsproblem einbezogen, das das Optimalsteuerungsproblem approximiert.

Als günstig erweist sich die Anwendung von abschnittsweisen Polynomansätzen, deren Parameter die Werte der Steuergrößen $\mathbf{u}^k = \mathbf{u}(t^k)$ und der Zustandsgrößen $\mathbf{x}^k = \mathbf{x}(t^k)$ an den Gitterpunkten

$$t_0 = t^0 < t^1 < t^2 < \ldots < t^K = t_f \tag{6.1}$$

sind[1]. In Abhängigkeit von den gewählten Polynomansätzen und den Kollokationspunkten erhält man die Kollokationsbedingungen, siehe z. B. Betts (2001). Bei Approximation der Steuergrößen durch konstante Werte in den Teilabschnitten und der Zustandsgrößen durch lineare Interpolation

$$\mathbf{u}_{app}(t) = \mathbf{u}^k, \qquad t^k \le t < t^{k+1}, \quad k = 0, \ldots, K-1 \tag{6.2a}$$

$$\mathbf{x}_{app}(t) = \mathbf{x}^k + \frac{t - t^k}{t^{k+1} - t^k}(\mathbf{x}^{k+1} - \mathbf{x}^k), \quad t^k \le t \le t^{k+1}, \quad k = 0, \ldots, K-1 \tag{6.2b}$$

sowie Wahl der Intervallmittelpunkte $t^{k+1/2} = (t^k + t^{k+1})/2$ als Kollokationspunkte, siehe Abbildung 6.1, erhält man

$$\frac{1}{t^{k+1} - t^k}(\mathbf{x}^{k+1} - \mathbf{x}^k) = \mathbf{f}\left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2}\right), \quad k = 0, \ldots, K-1 \tag{6.3}$$

Dies entspricht der impliziten Mittelpunktsregel.

[1] Es besteht ein grundsätzlicher Unterschied zur Zeitstufenstruktur der Mehrstufen-Steuerungsparametrisierung nach (3.2): bei Kollokationsverfahren stimmen die Teilabschnitte $[t^k, t^{k+1}]$ mit den Diskretisierungsschritten der Differentialgleichung überein.

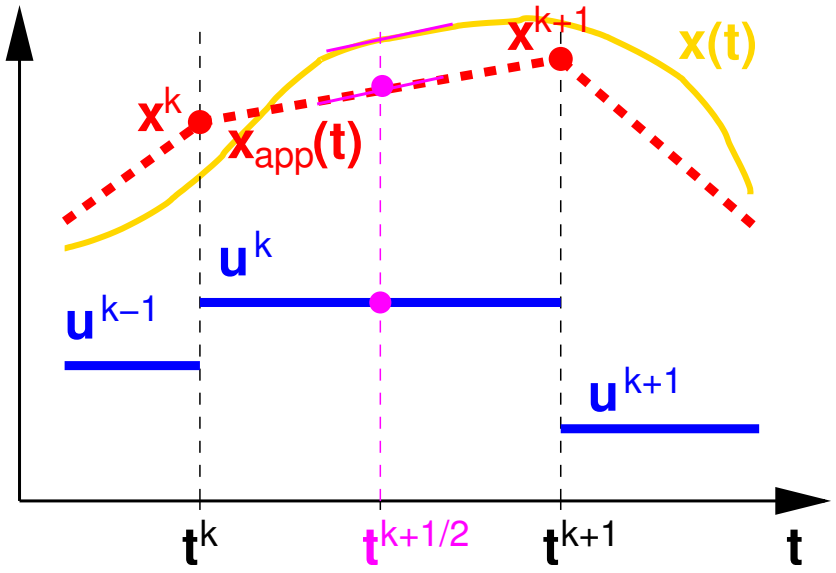

Abbildung 6.1: Kollokation.

Das Zielfunktional (3.1d) und die Ungleichungsbeschränkungen (3.1b) werden ebenfalls diskretisiert

$$J_{app} = F(\mathbf{x}^K, t^K) + \sum_{k=0}^{K-1} \left((t^{k+1} - t^k) f_0 \left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2} \right) \right) \tag{6.4a}$$

$$\mathbf{g} \left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2} \right) \leq \mathbf{0}, \quad k = 0, \ldots, K-1 \tag{6.4b}$$

und die Randbedingungen als zusätzliche Gleichungsbeschränkungen in das Optimierungsproblem aufgenommen.

Das resultierende nichtlineare Optimierungsproblem ist hochdimensional und „sparse“, d. h. sowohl die Jacobimatrizen der Gleichungs- und Ungleichungsbeschränkungen als auch die Hessematrix der zugehörigen Lagrangefunktion sind schwach besetzt. Ein geeigneter Solver für derartige Optimierungsprobleme ist SNOPT, siehe Gill u. a. (2002), der als eingeschränkte „Studentenlizenz“ unter `http://scicomp.ucsd.edu/~peg` kostenlos verfügbar ist.

SNOPT verfügt über ein MEX-Interface, kann also direkt aus MATLAB heraus aufgerufen werden. Das Optimierungsproblem ist in der Form

$$\min_{\mathbf{y} \in \mathbb{R}^{n_y}} \left\{ h_1(\mathbf{y}) \,\middle|\, \mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max}, \, \mathbf{h}_{\min} \leq \mathbf{h}(\mathbf{y}) \leq \mathbf{h}_{\max}, \, \mathbf{h} : \mathbb{R}^{n_y} \to \mathbb{R}^{n_h} \right\} \tag{6.5}$$

anzugeben. Lineare Gleichungs- und Ungleichungsbeschränkungen können auch separiert werden.

Das folgende Listing zeigt die Lösung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 mittels direkter Kollokation.

In den Zeilen 31–58 wird der Variablenvektor $\mathbf{y}$ des nichtlinearen Optimierungsproblems mit den Komponentenbeschränkungen initialisiert

$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ -u_{minmax} \\ -\infty \\ \vdots \\ -u_{minmax} \\ x_{f,1} \\ x_{f,2} \\ 1.0 \end{bmatrix} \leq \begin{bmatrix} x_1^0 \\ x_2^0 \\ u^0 \\ x_1^1 \\ \vdots \\ u^{K-1} \\ x_1^K \\ x_2^K \\ t_f \end{bmatrix} \leq \begin{bmatrix} x_{0,1} \\ x_{0,2} \\ u_{minmax} \\ \infty \\ \vdots \\ u_{minmax} \\ x_{f,1} \\ x_{f,2} \\ \infty \end{bmatrix} \tag{6.6}$$

Die Randbedingungen für die Zustandsgrößen werden als Gleichungsbeschränkungen für die entsprechenden Komponenten des Vektors $\mathbf{y}$ einbezogen. Die freie Endzeit t_f wird als zu optimierende Größe betrachtet, dabei schränkt die Vorgabe $t_f \geq 1.0$ die Suche auf einen sinnvollen Wertebereich ein.

Die Berechnung der Zielfunktion und der Beschränkungen

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leq \begin{bmatrix} J_{app} \\ x_1^1 - x1^0 - t_f/K \cdot (x_2^0 + x_2^1)/2 \\ \vdots \\ x_2^K - x_2^K - 1 - t_f/K \cdot u^{K-1} \end{bmatrix} \leq \begin{bmatrix} \infty \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.7}$$

erfolgt in der Funktion `dising_dto_dtc`, Listing 6.2. Für die Zielfunktion wird als untere Schranke 0 angenommen. Die Gleichheit von oberer und unterer Schranke für die Kollokationbedingungen erzwingt eine Behandlung als Gleichungsbeschränkungen.

In Zeile 63 erfolgt der Aufruf von `snopt`. Übergabeparameter sind die Anfangsnäherung für die Optimierungsvariablen `y0`, die Schranken für die Optimierungsvariablen `ymin` und `ymax` aus (6.6) sowie die Schranken für die Gleichungs- und Ungleichungssrestriktionen `Jhmin` und `Jhmax` aus (6.7). Weiterhin wird der Name der Funktion zur Berechnung von Zielfunktionswert und Beschränkungen (als Zeichenkette `'dising_dto_dtc'`) übergeben.

Listing 6.1: `dising_dto.m`

```
1  function dising_dto
   global rho_tf meth
16 K = 98;                          % Anzahl Intervalle
   uminmax = 1;   % Steuerungsbeschraenkung
18 rho_tf = 0.1;    % Bewertung Endzeit
   x0 = [-3; 0]; % Anfangswerte
20 xf = [0; 0];   % Endwerte
   tf = 5;         % Startnaeherung Endzeit
31     y0 = zeros((K+1)*2+K*1+1, 1); % Variablenvektor:
                                     % [x(0); u(0); ...; u(K-1); x(K); tf]
37 ymin = -inf*ones(size(y0));    % untere Schranke
   ymax = inf*ones(size(y0));     % obere Schranke
```

```
y0(end) = tf;                    % Endzeit: Startnaeherung
ymin(end) = 1;                   %          untere Schranke
ymin(1) = x0(1);                 % Anfangsbedingung x_1
ymax(1) = ymin(1);
ymin(2) = x0(2);                 % Anfangsbedingung x_2
ymax(2) = ymin(2);
    idx1 = length(y0)-2;
ymin(idx1) = xf(1);              % Endbedingung x_1
ymax(idx1) = ymin(idx1);
idx2 = idx1+1;
ymin(idx2) = xf(2);              % Endbedingung x_2
ymax(idx2) = ymin(idx2);
ymin(3:3:end-1) = -uminmax;      % untere Schranke Steuergroesse
ymax(3:3:end-1) = uminmax;       % obere Schranke Steuergroesse
Jhmin = zeros(1+K*2, 1);         % Zielfunktion und Beschraenkungen
Jhmax = Jhmin;
Jhmax(1) = inf;                  % Zielfunktion unbeschraenkt
% Aufruf SNOPT
[y, Jh, inform] = snopt(y0, ymin, ymax, Jhmin, Jhmax, 'dising_dto_dtc');
% Auswertung Loesung
fprintf('Optimale Endzeit: %g\nOptimaler Wert Zielfunktional: %g\n', ...
    y(end), Jh(1));
dising_plot(y)
clear global rho_tf meth
```

Listing 6.2: `dising_dto_dtc.m`

```
function [Jh, grad_Jh] = dising_dto_dtc(y)
global rho_tf meth
xk = [y(1:3:end-1) y(2:3:end-1)]; % Stuetzstellen Zustandsgroessen
uk = y(3:3:end-1);                % Stuetzstellen Steuergroessen
tf = y(end);                      % Endzeit
K = size(xk, 1)-1;                % Anzahl Intervalle
dt = 1.0/K;                       % Laenge normiertes Teilintervall
% Zustands- und Steuergroessen am Intervallmittelpunkt
xk12 = (xk(1:end-1, :)+xk(2:end, :))/2;
    uk12 = uk;                    % implizite Mittelpunktsregel
% Zielfunktional
J = rho_tf*tf + 0.5*sum(xk12(:, 1).^2+xk12(:, 2).^2)*dt*tf;
% Beschraenkungen: Approximation Zustands-Dgl.
% mit impliziter Mittelpunktsregel bzw. Trapezregel
h1 = xk(2:end, 1)-xk(1:end-1, 1)-dt*xk12(:, 2)*tf;
h2 = xk(2:end, 2)-xk(1:end-1, 2)-dt*uk12*tf;
h = [h1'; h2']; h = h(:);
% Rueckgabeparameter
Jh = [J; h];
grad_Jh = []; % keine Ableitungsberechnung
```

Innerhalb von 35 Iterationen findet SNOPT eine Lösung des nichtlinearen Optimierungsproblems, die zugehörigen Zeitverläufe zeigt Abbildung 6.2.

```
 Major Minors     Step    nCon Feasible  Optimal  MeritFunction     nS Penalty
     0    131                1  2.0E+00  5.6E-02  5.5739796E-01     78           r
     1     12  7.6E-02       2  1.1E+00  2.9E-02  2.3022656E+00     73 4.0E-01 n rl
     2     20  7.3E-02       3  7.6E-01  2.9E-02  4.2131863E+00     66 9.3E-01 s  l
     3    171  4.8E-01       4  1.3E-01  3.9E-02  9.8551984E+00     94 3.4E+00    l
...
    32      1  1.0E+00      33 (3.6E-09) 6.3E-06  8.0059746E+00     26 3.4E+00       c
    33      1  1.0E+00      34 (2.4E-09) 1.1E-05  8.0059744E+00     26 3.4E+00       c
    34      1  1.0E+00      35 (5.6E-11) 8.7E-06  8.0059742E+00     26 3.4E+00       c
    35      1  1.0E+00      36 (7.4E-10)(1.7E-06) 8.0059741E+00     26 3.4E+00       c

   EXIT -- optimal solution found
Optimale Endzeit: 3.88168
Optimaler Wert Zielfunktional: 8.00597
```

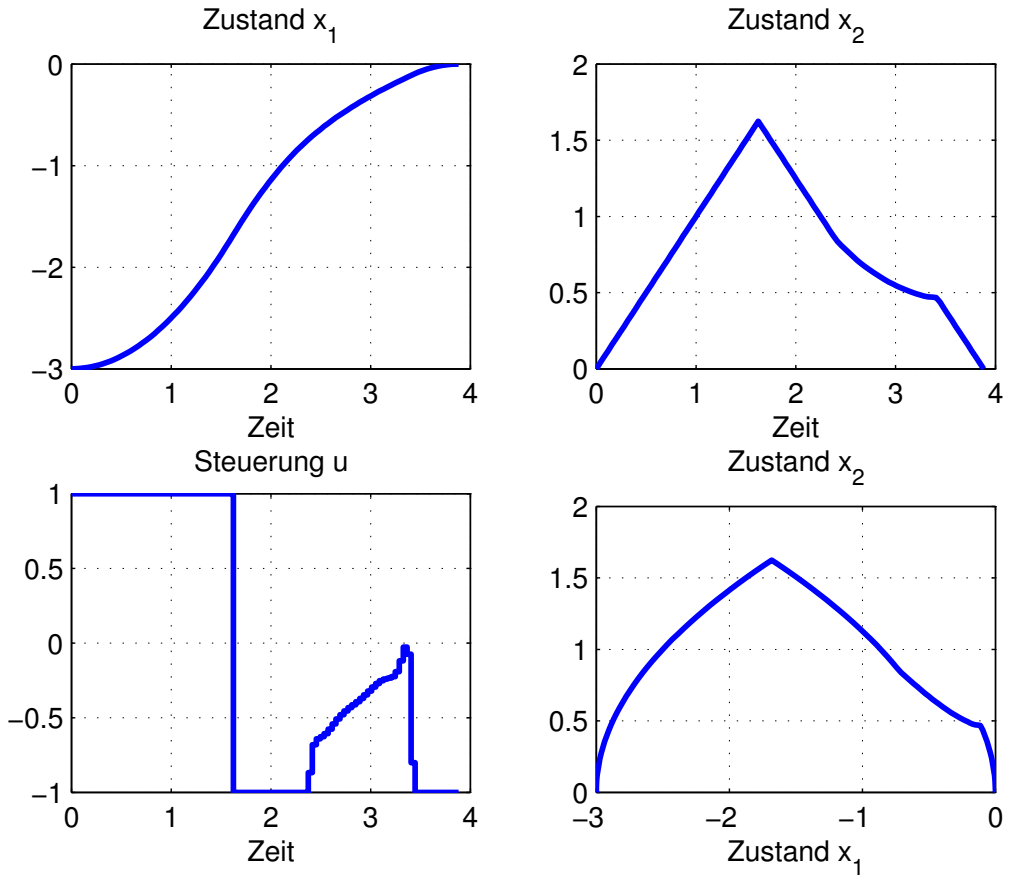

Abbildung 6.2: Doppelintegrator mit singulärem Lösungsabschnitt; direkte Kollokation.

Literaturverzeichnis

[Arnold 2004] Arnold, E.: *Optimale Steuerung 2*. Vorlesung TU Ilmenau, Foliensatz. 2004. – URL ftp://ftp.systemtechnik.tu-ilmenau.de/pub/lab/OptimaleSteuerung2.pdf 3

[Betts 2001] Betts, J. T.: *Practical methods for optimal control using nonlinear programming*. Philadelphia : SIAM, 2001 32

[Brenan u. a. 1989] Brenan, K. E. ; Campbell, S. L. ; Petzold, L. R.: *Numerical solution of initial-value problems in differential-algebraic equations*. North-Holland, 1989 10

[Franke 1998] Franke, R.: Omuses and HQP / Technische Universität Ilmenau. URL Hqp/omuses.pdf, 1998. – Forschungsbericht 8, 10

[Gill u. a. 2002] Gill, P. E. ; Murray, W. ; Saunders, M. A.: SNOPT: An SQP algorithm for large-scale constrained optimization. In: *SIAM J. Optim.* 12 (2002), Nr. 4, S. 979–1006. – URL ftp://www.scicomp.ucsd.edu/pub/peg/reports/snpaper.pdf 33

[Griewank u. a. 1996] Griewank, A. ; Juedes, D. ; Utke, J.: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. In: *ACM Trans. Math. Software* 22 (1996), June, Nr. 2, S. 131–167. – URL ftp://ftp.math.tu-dresden.de/pub/ADOLC/ADOLC_1.8/adolc_1.8.ps.gz 10, 14

[Papageorgiou 1996] Papageorgiou, M.: *Optimierung: statische, dynamische, stochastische Verfahren für die Anwendung*. München : Oldenbourg, 1996 7

[Shampine u. a. 2003] Shampine, L. F. ; Kierzenka, J. ; Reichelt, M. W.: Solving boundary value problems for ordinary differential equations in Matlab with `bvp4c` / The MathWorks. Natick, 2003. – Forschungsbericht. – URL http://www.mathworks.com/bvp_tutorial 27

[Stewart 1992] Stewart, D. E.: *Meschach: matrix computations in C*. Canberra, Australia: University of Canberra (Veranst.), 1992 10