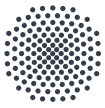
Institut für Systemdynamik
Univ.-Prof. Dr.-Ing. Dr. h. c. O. Sawodny



Universität Stuttgart

Numerische Lösung von Optimalsteuerungsproblemen

Dr.-Ing. Eckhard Arnold

Universität Stuttgart
Institut für Systemdynamik
Waldburgstr. 17/19
D-70563 Stuttgart

E-Mail: Eckhard.Arnold@isys.uni-stuttgart.de
Tel.: +49 (0)711/685-65928
Fax: +49 (0)711/685-66371

18. März 2020

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 Übungsaufgaben und der Mini-Projekte zur Vorlesung „Numerische Methoden der Optimierung und Optimalen Steuerung“ [5] verwendet werden können.

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

   a) Mehrstufen-Steuerungsparametrisierung
   HQP [10] (`https://omuses.github.io/hqp/doc/html/`, Abschnitt 3.1))
   ACADO [13] (`https://acado.github.io`, Abschnitt 3.2)
   MATLAB und CASADI [2] (`https://web.casadi.org`, Abschnitt 3.3)

   b) einfache Steuerungsparametrisierung
   MATLAB (Abschnitt 3.4)
   MATLAB und CASADI (Abschnitt 3.5)

   c) direkte Kollokation (Abschnitt 4)

   - Programmierung in MATLAB mit Solvern SNOPT (`http://scicomp.ucsd.edu/~peg/`, Abschnitt 4.1.1), IPOPT ([14], [20], Abschnitt 4.1.2), `fmincon` (Abschnitt 4.1.3) oder unter Nutzung von CASADI (Abschnitt 4.1.4)
   - Nutzung algebraischer Modellierungssprachen wie JUMP (`https://github.com/JuliaOpt/JuMP.jl`) oder AMPL (`https://www.ampl.com`) mit einem geeigneten Solver (z. B. IPOPT) zur Lösung des resultierenden großen nichtlinearen Optimierungsproblems (Abschnitte 4.2 und 4.3)
   - JMODELICA.ORG (`https://jmodelica.org`), ein Simulationswerkzeug, das die Formulierung und Lösung von Optimalsteuerungsaufgaben unterstützt (Abschnitt 4.4)

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

   a) Schießverfahren (Abschnitt 5)
   numerische Integration des kanonischen Differentialgleichungssystems
   nichtlineares Gleichungssystem MATLAB: `ode45`, `fsolve`

   b) Kollokationsverfahren für Randwertaufgaben (Abschnitt 6)
   (großes) nichtlineares Gleichungssystem
   MATLAB: `bvp4c`

c) indirekte Gradientenverfahren (Abschnitt 7)
Suchverfahren im Funktionenraum

Die Quelltexte der Programme zur Lösung der Beispielaufgaben mit den angegebenen Verfahren finden sich in der Datei **cocp_ex.zip**.

Weitere Alternativen sind beispielsweise:

1. Pseudo-Spektralmethoden, d. h. Kollokation mit globalen Ansatzfunktionen anstelle abschnittsweiser Poynomansätze (z. B. mit PSOPT https://sites.google.com/a/psopt.org/psopt/Home)
2. Lösung der Randwertaufgaben mit einem Tabellenkalkulationsprogramm (z. B. Microsoft Excel), siehe https://www.researchgate.net/publication/4932162_Numerical_Optimal_Control_in_Continuous_Time_Made_Easy.
3. Implementierung einer direkten Kollokation nach Abschnitt 4 in C oder C++ unter Nutzung verfügbarer Software zur Lösung großer nichtlinearer Optimierungsprobleme, beispielsweise IPOPT [14], [20].

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.

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

Mit der HAMILTON-Funktion

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

ergeben sich die Optimalitätsbedingungen

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

die zusammen mit den Zustandsgleichungen (2.1a), (2.1b) und den Randbedingungen (2.1c) im Optimum erfüllt sein müssen.

Eine Lösung der Aufgabe ist in Abschnitt 6 angegeben.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3.1	`Hqp_Beispiel/{Prg_Di.[hC],Hqp_Beispiel.tcl}`
	3.1.3	`Hqp_Beispiel/{DI.mo,FMI_Beispiel.tcl}`
	3.2	`cocp_ex/acado/doint_acado.m`
Einfache Steuerungsparametrisierung	3.4	`cocp_ex/cpara/doint_cpara.m`
	3.5	`cocp_ex/casadi/doint_single_shooting.m`
Direkte Kollokation	4.1.1	`cocp_ex/dto_snopt/doint_dto.m`
	4.1.2	`cocp_ex/dt_ipopt/doint_dto_ipopt.m`
	4.1.3	`cocp_ex/dt_fmincon/doint_dto_fmincon.m`
	4.2	`cocp_ex/JuMP/doint_jump.jl`
	4.2	`cocp_ex/JuMP/doint_cub_jump.jl`
	4.3	`cocp_ex/ampl/doint_ampl.m`
	4.4	`cocp_ex/JModelica.org/doint_opt.[mop,py]`
Randwertaufgabe und Schießverfahren	5	`cocp_ex/shoot/doint1_shoot.m`
Randwertaufgabe und Kollokation	6	`cocp_ex/bvp4c/doint1_bvp.m`
Indirektes Gradientenverfahren	7	`cocp_ex/cvi/doint1_cvi_fgrad.m`

Tabelle 2.1: Doppelintegrator mit Steuerungsbeschränkung

2.2 Doppelintegrator mit Zustandsbeschränkung

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

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

Die Zustandsbeschränkung ist 2. Ordnung, da ihre 2. Zeitableitung explizit von u abhängt:

$$\frac{d}{dt}(x_1 - x_{1,\max}) = x_2, \quad \frac{d^2}{dt^2}(x_1 - x_{1,\max}) = \dot{x}_2 = u$$

Mit der Hamilton-Funktion H, der erweiterten Hamilton-Funktion $\tilde{H}$ und dem Lagrange-Multiplikator $\mu(t)$

$$H = \frac{1}{2}u^2 + p_1 x_2 + p_2 u \tag{2.5}$$
$$\tilde{H} = H + \mu u \tag{2.6}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = 0 \tag{2.7a}$$
$$\dot{p}_2 = -p_1 \tag{2.7b}$$
$$u = -p_2 - \mu \tag{2.7c}$$
$$\mu(x_1 - x_{1,\max}) = 0 \tag{2.7d}$$

die zusammen mit den Zustandsgleichungen (2.4a), (2.4b) und den Randbedingungen (2.4c) im Optimum erfüllt sein müssen.

Wenn die Zustandsbeschränkung nicht aktiv ist ($x_1 < x_{1,\max}$), dann ist $\mu = 0$. In einem Teilintervall $[t_{s1}, t_{s2}]$ mit aktiver Zustandsbeschränkung ($x_1 = x_{1,\max}$) ist $u = 0$, und am Eintritts- t_{s1} bzw. Austrittspunkt t_{s2} muss gelten

$$x_1(t_{s1}) = x_{1,\max} \tag{2.7e}$$
$$x_2(t_{s1}) = 0 \tag{2.7f}$$
$$\mathbf{p}(t_{s1} - 0) = \mathbf{p}(t_{s1} + 0) + \eta_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \eta_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.7g}$$
$$H(t_{s1} - 0) = H(t_{s1} + 0) \tag{2.7h}$$
$$\mu(t_{s2}) = 0 \tag{2.7i}$$

$\eta_1 \geq 0$ und $\eta_2 \geq 0$ sind LAGRANGE-Multiplikatoren. Zudem ist

$$\mu(t) \geq 0, \quad \frac{d}{dt}\mu(t) \leq 0, \quad \frac{d^2}{dt^2}\mu(t) \geq 0, \quad t \in [t_{s1}, t_{s2}] \tag{2.7j}$$

Wenn das Teilintervall mit aktiver Zustandsbeschränkung zu einem Berührungspunkt $t_{s1} = t_{s2}$ entartet, dann ist $\mu = 0$ für $t \in [0, 1]$ und die Bedingungen (2.7i), (2.7j) entfallen.

Lösungen der Aufgabe mit direkten Verfahren sind in den Abschnitten 3.1.2.2, 3.4.1.1 und 3.5.1.1 angegeben. Die Anwendung indirekter Verfahren erfordert neben der Kenntnis der Schaltstruktur (Abfolge von aktiven und inaktiven Teilintervallen) sehr gute Startwerte für die numerische Lösung der Randwertaufgabe.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3.1	`Hqp_Beispiel/{Prg_Di2.[hC],Hqp_Beispiel.tcl}`
	3.1.3	`Hqp_Beispiel/{DI.mo,FMI_Beispiel.tcl}`
	3.2	`cocp_ex/acado/doint_acado.m`
Einfache Steuerungsparametrisierung	3.4.1.1	`cocp_ex/cpara/doint_cpara.m`
	3.5.1.1	`cocp_ex/casadi/doint_single_shooting.m`
Direkte Kollokation	4.1.1	`cocp_ex/dto_snopt/doint_dto.m`
	4.1.2	`cocp_ex/dt_ipopt/doint_dto_ipopt.m`
	4.1.3	`cocp_ex/dt_fmincon/doint_dto_fmincon.m`
	4.2	`cocp_ex/JuMP/doint_jump.jl`
	4.2	`cocp_ex/JuMP/doint_cub_jump.jl`
	4.3	`cocp_ex/ampl/doint_ampl.m`
	4.4	`cocp_ex/JModelica.org/doint_opt.[mop,py]`
Randwertaufgabe und Schießverfahren	5	`cocp_ex/shoot/doint2_shoot.m`
Randwertaufgabe und Kollokation	6	`cocp_ex/bvp4c/doint2_bvp.m`

Tabelle 2.2: Doppelintegrator mit Zustandsbeschränkung

2.3 Zeitoptimale Umsteuerung eines PT_2-Glieds

Ein PT_2-Glied ist unter Berücksichtigung von Steuerungsbeschränkungen zeitoptimal von einem festen Anfangs- in einen festen Endzustand zu überführen.

$$\dot{x}_1 = -0.5x_1 + x_2 \tag{2.8a}$$
$$\dot{x}_2 = -x_2 + u \tag{2.8b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \qquad \mathbf{x}(t_f) = \mathbf{x}_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{2.8c}$$
$$J = \int_0^{t_f} dt = t_f \tag{2.8d}$$
$$|u| \leq u_{\text{minmax}} \tag{2.8e}$$

Mit der Hamilton-Funktion

$$H = 1 + p_1(-0.5x_1 + x_2) + p_2(-x_2 + u) \tag{2.9}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = 0.5p_1 \tag{2.10a}$$
$$\dot{p}_2 = -p_1 + p_2 \tag{2.10b}$$
$$u = \begin{cases} u_{\text{minmax}}, & \text{falls } p_2 < 0 \\ -u_{\text{minmax}}, & \text{falls } p_2 > 0 \end{cases} \tag{2.10c}$$
$$H\big|_{t_f} = 0 \tag{2.10d}$$

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 3.1.2.1, 3.3.1.1 und 5.1.1.1 angegeben.

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3.1	`Hqp_Beispiel/{Prg_T2Topt.[hC],Hqp_Beispiel.tcl}`
	3.1.3	`Hqp_Beispiel/{T2Topt.mo,FMI_Beispiel.tcl}`
	3.2	`cocp_ex/acado/t2topt_acado.m`
	3.3	`cocp_ex/casadi/t2topt_multiple_shooting_tf.m`
Einfache Steuerungsparametrisierung	3.4	`cocp_ex/cpara/t2topt_cpara.m`
Direkte Kollokation	4.1.1	`cocp_ex/dto_snopt/t2topt_dto.m`
	4.1.2	`cocp_ex/dt_ipopt/t2topt_dto_ipopt.m`
	4.1.3	`cocp_ex/dt_fmincon/t2topt_dto_fmincon.m`
	4.2	`cocp_ex/JuMP/t2topt_jump.jl`
	4.2	`cocp_ex/JuMP/t2topt_cub_jump.jl`
	4.3	`cocp_ex/ampl/t2topt_ampl.m`
	4.4	`cocp_ex/JModelica.org/t2topt_opt.[mop,py]`
Randwertaufgabe und Schießverfahren	5.1.1.1	`cocp_ex/shoot/t2topt_shoot.m`
Randwertaufgabe und Kollokation	6	`cocp_ex/bvp4c/t2topt_bvp.m`
Indirektes Gradientenverfahren	7	`cocp_ex/cvi/t2topt_cvi_fgrad.m`

Tabelle 2.3: Zeitoptimale Umsteuerung eines PT_2-Glieds

2.4 Doppelintegrator mit singulärem Lösungsabschnitt

Die folgende Aufgabe ist [16] entnommen und wurde um eine Bewertung des Endzeitpunkts erweitert.

$$\dot{x}_1 = x_2 \tag{2.11a}$$
$$\dot{x}_2 = u \tag{2.11b}$$
$$\mathbf{x}(0) = \mathbf{x}_0 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}, \qquad \mathbf{x}(t_f) = \mathbf{x}_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad t_f \text{ frei} \tag{2.11c}$$
$$J = \rho_{t_f} t_f + \frac{1}{2} \int_0^{t_f} \left(x_1^2 + x_2^2 \right) dt \tag{2.11d}$$
$$|u| \leq 1 \tag{2.11e}$$

Mit der HAMILTON-Funktion

$$H = \frac{1}{2} x_1^2 + \frac{1}{2} x_2^2 + p_1 x_2 + p_2 u \tag{2.12}$$

ergeben sich die Optimalitätsbedingungen

$$\dot{p}_1 = -x_1 \tag{2.13a}$$
$$\dot{p}_2 = -\left(x_2 + p_1 \right) \tag{2.13b}$$
$$u = \begin{cases} 1, & \text{falls } p_2 < 0 \\ -1, & \text{falls } p_2 > 0 \\ u_{\text{sing}}, & \text{falls } p_2 = 0 \end{cases} \tag{2.13c}$$
$$\rho_{t_f} + H\big|_{t_f} = 0 \tag{2.13d}$$

Unter Berücksichtigung der Randbedingungen (2.11c) ergibt sich aus (2.13d)

$$\rho_{t_f} + p_2(t_f) u(t_f) = 0 \tag{2.14}$$

Verfahren	Abschnitt	Datei
Mehrstufen-Steuerungsparametrisierung	3.1	`Hqp_Beispiel/{Prg_Dising.[hC],Hqp_Beispiel.tcl}`
	3.1.3.1	`Hqp_Beispiel/{Dising.mo,FMI_Beispiel.tcl}`
	3.2	`cocp_ex/acado/dising_acado.m`
Einfache Steuerungsparametrisierung	3.4	`cocp_ex/cpara/dising_cpara.m`
Direkte Kollokation	4.1.1	`cocp_ex/dto_snopt/dising_dto.m`
	4.1.2	`cocp_ex/dt_ipopt/dising_dto_ipopt.m`
	4.1.3	`cocp_ex/dt_fmincon/dising_dto_fmincon.m`
	4.1.4.1	`cocp_ex/casadi/dising_collocation_imp_tf.m`
	4.2	`cocp_ex/JuMP/dising_jump.jl`
	4.2	`cocp_ex/JuMP/dising_cub_jump.jl`
	4.3	`cocp_ex/ampl/dising_ampl.m`
	4.4	`cocp_ex/JModelica.org/dising_opt.[mop,py]`
Randwertaufgabe und Schießverfahren	5	`cocp_ex/shoot/dising_shoot.m`
Randwertaufgabe und Kollokation	6	`cocp_ex/bvp4c/dising_bvp.m`
Indirektes Gradientenverfahren	7	`cocp_ex/cvi/dising_cvi_fgrad.m`

Tabelle 2.4: Doppelintegrator mit singulärem Lösungsabschnitt

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

$$p_2(t_{s1,i}) = 0 \tag{2.15a}$$

$$x_2(t_{s1,i}) + p_1(t_{s1,i}) = 0 \tag{2.15b}$$

$$u_{\text{sing}} = x_1 \qquad \text{für } t = [t_{s1,i}, t_{s2,i}] \tag{2.15c}$$

Lösungen der Aufgabe sind in den Abschnitten 3.1.3.1, 4.1.1, 4.1.2, 4.1.3, 4.3 und 4.4 angegeben. In Abbildung 4.2 ist deutlich die Abfolge der Lösungsabschnitte (bang-bang-Verhalten bzw. singuläre Steuerung) zu erkennen.

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

3 Steuerungsparametrisierung

Betrachtet werden numerische Verfahren zur Lösung beschränkter Optimalsteuerungsprobleme

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad t \in (t_0, t_f) && \text{(3.1a)}\\
\mathbf{x}(t_0) &= \mathbf{x}_0 && \text{(3.1b)}\\
\mathbf{g}(\mathbf{x}, \mathbf{u}, t) &\leq \mathbf{0}, \qquad t \in [t_0, t_f], && \text{(3.1c)}\\
\mathbf{h}(\mathbf{x}(t_f), t_f) &= \mathbf{0} && \text{(3.1d)}\\
J &= F(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} f_0(\mathbf{x}, \mathbf{u}, t)dt \longrightarrow \text{ min!} && \text{(3.1e)}
\end{aligned}$$

Der Anfangszustand $\mathbf{x}_0$ und der Zeithorizont $[t_0, t_f]$ können dabei fest, d. h. vorgegeben, oder frei sein.

In vielen Fällen ist es numerisch günstig, eine Aufgabe mit freiem Optimierungshorizont $t \in [t_0, t_f]$ in eine Aufgabe mit der skalierten Zeitvariablen τ und dem festem Horizont $\tau \in [0, 1]$ zu transformieren.

$$\begin{aligned}
t &= t_0 + \tau(t_f - t_0), \quad \tau \in [0,\, 1] && \text{(3.2a)}\\
\frac{d\mathbf{x}}{d\tau} &= \frac{d\mathbf{x}}{dt} \cdot \frac{dt}{d\tau} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t_0 + \tau(t_f - t_0)) \cdot (t_f - t_0) && \text{(3.2b)}\\
\int_{t_0}^{t_f} f_0(\mathbf{x}, \mathbf{u}, t)dt &= \int_0^1 f_0(\mathbf{x}, \mathbf{u}, t_0 + \tau(t_f - t_0)) \cdot (t_f - t_0)d\tau && \text{(3.2c)}
\end{aligned}$$

Die freie Endzeit t_f und ggf. die freie Anfangszeit t_0 sind dann Parameter des transformierten Problems, die zusammen mit der Lösung zu bestimmen sind.

Die Grundidee der Steuerungsparametrisierung besteht darin, das Optimalsteuerungsproblem (3.1) durch ein nichtlineares Optimierungsproblem zu approximieren, indem der Zeitverlauf der Steuergrößen durch einen parametrischen Ansatz

$$\mathbf{u}(t, \mathbf{w}) = \mathbf{\Phi}(\mathbf{w}, t) \tag{3.3}$$

beschrieben und die Zustandsgrößen durch numerische Lösung des Anfangswertproblems (3.1a), (3.1b) als abhängige Variable berechnet werden. Der Integralterm des Zielfunktionals (3.1e) wird ebenfalls durch numerische Integration ausgewertet. Die Trajektorienbeschränkungen (3.1c) werden durch eine endliche Anzahl von Ungleichungsbeschränkungen approximiert, beispielsweise durch Diskretisierung auf einem vorgegebenem Zeitgitter.

Die Ansatzparameter $\mathbf{w}$, eventuelle freie Komponenten des Anfangszustands $\mathbf{x}_0$ sowie die Endzeit t_f und die Anfangszeit t_0 – sofern diese frei sind – bilden den Variablenvektor $\mathbf{w}$ des nichtlinearen Optimierungsproblems

$$\min_{\mathbf{w}} \left\{ J(\mathbf{w}) \,\middle|\, \mathbf{g}(\mathbf{w}) \leq \mathbf{0},\, \mathbf{h}(\mathbf{w}) = \mathbf{0} \right\} \tag{3.4}$$

Dieses ist ist in der Regel unstrukturiert („dense“) und kann mit einem geeigneten Lösungsverfahren gelöst werden. Aufgrund der unterlagerten Lösung des Anfangswertproblems ist die Auswertung der Zielfunktion und der Beschränkungen aufwendig.

Wenn die Ableitungen der Zielfunktion und der Beschränkungen mittels finiter Differenzen approximiert werden, so ist zu beachten, dass der numerische Fehler bei der Lösung des Anfangswertproblems die Approximation der Ableitungen verfälschen kann. In diesem Fall sollte ein Optimalsteuerungsproblem mit freier Endzeit stets in ein solches mit fester Endzeit transformiert werden, und es sollten nach Möglichkeit numerische Integrationsverfahren mit fester Schrittweite eingesetzt werden.

Ein wesentlicher Nachteil einer solchen einfachen Steuerungsparametrisierung besteht darin, dass die Sensitivität der Zustandsgrößen und damit der Zielfunktion und der Beschränkungen des nichtlinearen Optimierungsproblems (3.4) zwischen den Ansatzparametern w_i stark variiert. Dadurch können insbesondere schwach gedämpfte oder instabile Lösungen der Zustandsdifferentialgleichung zu schlecht konditionierten nichtlinearen Optimierungsproblemen führen.

Dies wird bei einer Mehrstufen-Steuerungsparametrisierung durch eine Unterteilung des Zeithorizonts $[t_0, t_f]$ in K Zeitabschnitte (Zeitstufen)

$$t_0 = t^0 < t^1 < \cdots < t^K = t_f \tag{3.5}$$

vermieden. Der parametrische Ansatz für die Steuergrößen und die numerische Integration der Zustandsdifferentialgleichung(3.1a) erfolgt separat für die Zeitstufen $k = 0, \ldots, K-1$

$$\mathbf{u}(t, \mathbf{u}^k) = \boldsymbol{\phi}^k(\mathbf{u}^k, t), \qquad t \in [t^k, t^{k+1}) \tag{3.6a}$$

$$\dot{\tilde{\mathbf{x}}}^k(t) = \mathbf{f}(\tilde{\mathbf{x}}^k(t), \boldsymbol{\phi}^k(\mathbf{u}^k, t), t), \quad t \in (t^k, t^{k+1}), \quad \tilde{\mathbf{x}}^k(t^k) = \mathbf{x}^k \tag{3.6b}$$

Dabei sind $\mathbf{u}^k$ die Ansatzparameter in den Zeitstufen $k = 0, \ldots, K-1$, die zusammen mit den Anfangszuständen $\mathbf{x}^k$ der Zeitstufen den Variablenvektor des nichtlinearen Optimierungsproblems bilden. Die Stetigkeit der Zustandsgrößen an den Übergängen der Zeitstufen wird durch zusätzliche Gleichungsbeschränkungen

$$\tilde{\mathbf{x}}^k(t^{k+1}) = \mathbf{x}^{k+1}, \quad k = 0, \ldots, K-1 \tag{3.7}$$

gesichert, siehe Abbildung 3.1.

Die numerische Integration des Integralterms des Zielfunktionals (3.1e) erfolgt ebenfalls separat für jede Zeitstufe. Die Trajektorienbeschränkungen (3.1c) werden durch endlich viele Ungleichungsbeschränkungen, beispielsweise durch Auswertung in den Gitterpunkten t^k, $k = 0, \ldots, K-1$, approximiert.

Mit den Optimierungsvariablen $\mathbf{u}^k$, $k = 0, \ldots, K-1$ und $\mathbf{x}^k$, $k = 0, \ldots K$ ergibt sich ein großes und strukturiertes nichtlineares Optimierungsproblem

$$\mathbf{x}^{k+1} = \mathbf{f}^k(\mathbf{x}^k, \mathbf{u}^k), \quad k = 0, \ldots, K-1 \tag{3.8a}$$

$$\mathbf{x}^0 = \mathbf{x}_0 \tag{3.8b}$$

$$\mathbf{g}^k(\mathbf{x}^k, \mathbf{u}^k) \leq \mathbf{0}, \qquad k = 0, \ldots, K-1 \tag{3.8c}$$

$$\mathbf{h}(\mathbf{x}^K) = \mathbf{0} \tag{3.8d}$$

$$J = F(\mathbf{x}^K) + \sum_{k=0}^{K-1} f_0^k(\mathbf{x}^k, \mathbf{u}^k) \longrightarrow \min! \tag{3.8e}$$

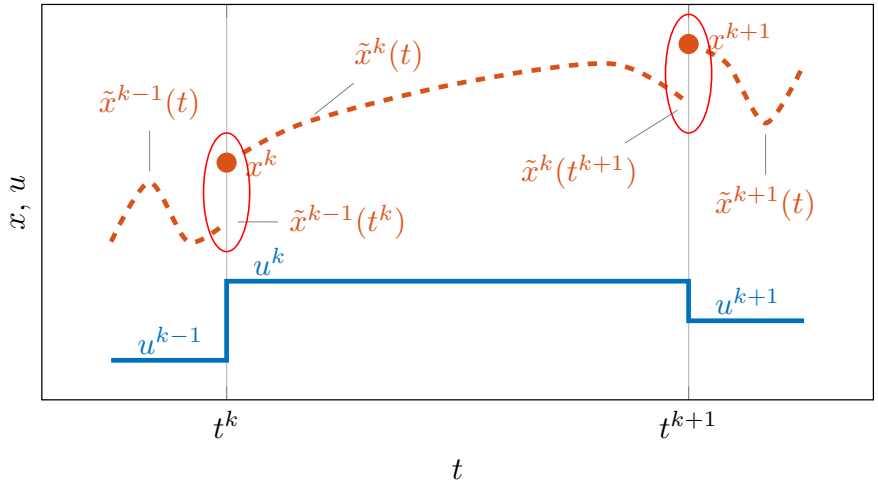


Abbildung 3.1: Mehrstufen-Steuerungsparametrisierung.

zur Approximation des ursprünglichen Optimalsteuerungsproblems (3.1).

3.1 Mehrstufen-Steuerungsparametrisierung mit Hqp

Hqp, siehe [10], ist eine Implementierung einer Mehrstufen-Steuerungsparametrisierung (3.8) zur Lösung beschränkter Optimalsteuerungsprobleme (3.1).

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 $\mathbf{u}(t, \mathbf{u}^k) = \mathbf{u}^k$, $t \in [t^k, t^{k+1})$ 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 $\tilde{\mathbf{x}}^k(t)$.

$$\mathbf{u}(t, \mathbf{u}^k) = \boldsymbol{\phi}^k(\mathbf{u}^k, t), \qquad t \in [t^k, t^{k+1}) \tag{3.9a}$$

$$\mathbf{x}^k = \begin{bmatrix} \mathbf{x}_d^k \\ \tilde{\mathbf{x}}^k(t^k) \end{bmatrix} \tag{3.9b}$$

$$\dot{\tilde{\mathbf{x}}}^k(t) = \mathbf{f}(\tilde{\mathbf{x}}^k(t), \boldsymbol{\phi}^k(\mathbf{u}^k, t), t), \quad t \in (t^k, t^{k+1}) \tag{3.9c}$$

Die Gleichungsbeschränkungen (Stufengleichungen) des nichtlinearen Optimierungsproblems

$$\mathbf{x}^{k+1} = \begin{bmatrix} \mathbf{f}_d^k(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})) \\ \tilde{\mathbf{x}}^k(t^{k+1}) \end{bmatrix}, \qquad k = 0, \ldots, K-1 \tag{3.10}$$

setzen sich aus den Zustandsgleichungen des zeitdiskreten Systems und den Stetigkeitsbedingungen der Approximationen der kontinuierlichen Zustandsgrößen (3.7) zusammen.

Ungleichungsbeschränkungen

$$\mathbf{u}_{\min}^k \le \mathbf{u}^k \le \mathbf{u}_{\max}^k, \qquad k = 0, \ldots, K-1 \tag{3.11a}$$

$$\mathbf{x}_{\min}^k \le \mathbf{x}^k \le \mathbf{x}_{\max}^k, \qquad k = 0, \ldots, K \tag{3.11b}$$

$$\mathbf{c}_{\min}^k \le \mathbf{c}^k\left(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})\right) \le \mathbf{c}_{\max}^k, \quad k = 0, \ldots, K-1 \tag{3.11c}$$

$$\mathbf{c}_{\min}^K \le \mathbf{c}^k\left(\mathbf{x}^K\right) \le \mathbf{c}_{\max}^K \tag{3.11d}$$

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

Die Zielfunktion des nichtlinearen Optimierungsproblems wird nach

$$J = f_0^K(\mathbf{x}^K) + \sum_{k=0}^{K-1} f_0^k\left(\mathbf{x}^k, \mathbf{u}^k, \tilde{\mathbf{x}}^k(t^{k+1})\right) \tag{3.12}$$

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

$$\dot{x}_{n+1} = f_0^k(\mathbf{x}, \mathbf{u}, t), \quad \text{mit } x_{n+1}(t_0) = 0, \tag{3.13a}$$

$$f_0^k = \tilde{x}_{n+1}^k(t^{k+1}) - x_{n+1}^k, \qquad k = 0, \ldots, K-1 \tag{3.13b}$$

einzubeziehen.

Eine freie Endzeit t_f des Optimalsteuerungsproblems kann durch Transformation auf einen festen Zeithorizont gemäß (3.2) und Betrachtung des Parameters t_f als (konstante) zeitdiskrete Zustandsgröße mit freiem Anfangswert in das Mehrstufenproblem einbezogen werden

$$x_d^{k+1} = x_d^k = t_f, \qquad k = 0, \ldots, K-1 \qquad \text{mit } x_d^0 = t_f \text{ frei} \tag{3.14}$$

3.1.1 Implementierung

HQP besteht aus mehreren Modulen, siehe [10], 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-Bibliothek (`https://homepage.divms.uiowa.edu/~dstewart/meschach/`), siehe [18],
- benötigte Ableitungen werden mittels automatischer Differentiation bestimmt, hierzu wird ADOL-C (`https://projects.coin-or.org/ADOL-C`) eingesetzt, siehe [12],
- verschiedene ODE/DAE-Solver dienen zur numerischen Integration der Zustandsdifferentialgleichungen, hier werden neben expliziten RUNGE-KUTTA-Verfahren auch komplexe DAE-Solver wie DASSL (`https://cse.cs.ucsb.edu/software`), siehe [7], 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.

3.1.2 Einbindung einer Optimalsteuerungsaufgabe in Hqp

3.1.2.1 Zeitoptimale Umsteuerung eines PT_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 PT_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

```
const 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 3.

Die Klassendefinition erfolgt in `Prg_T2Topt.C`. Im (optionalen) Konstruktor werden Klassenparameter (Variable) initialisiert, beispielsweise in Zeile 3 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 6 und 7 entsprechende Interface-Elemente vorgesehen werden.

Listing 3.1: `Prg_T2Topt.C`, Konstruktor.

```
1 Prg_T2Topt::Prg_T2Topt()
  {
3     set_K(50);          // Anzahl Stufen (stages)
      _uminmax = 5.0;     // Steuerungsbeschraenkung
5     _tf = 2.0;          // Startnaeherung Endzeit
      _ifList.append(new If_Real("prg_uminmax", &_uminmax)); // Tcl-Interface
7     _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 (3.5) festzulegen. In den meisten Fällen kann das durch einen Aufruf von `stages_alloc()` (Zeile 3) 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 (3.2) 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()`.

```
1 void Prg_T2Topt::setup_stages(IVECP ks, VECP ts)
  {
3     stages_alloc(ks, ts, K(), 1, 0.0, 1.0); // normierter Zeithorizont
                                              // [0.0, 1.0]
5 }
```

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 4), der Steuergrößen $\mathbf{u}^k$ (Zeile 20) 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 (3.11) festzulegen, beispielsweise in den Zeilen 21 und 22. Bei Gleichheit von oberer und unterer Schranke wird die Komponente intern als Gleichungsbeschränkung behandelt. Dies wird zur Festlegung von festen Anfangs- (Zeilen 8 und 9) oder Endzuständen (Zeilen 16 und 17) genutzt.

Für jede Variable kann mit Hilfe der Komponenten `x.initial` bzw. `u.initial` eine numerische Initialisierung vorgenommen werden, Zeilen 11-13, 23. 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 (Zeile 11). Solche Optimierungsvariable sollten zusätzlich auf einen sinnvollen Bereich eingeschränkt werden, beispielsweise $t_f \geq 0.1$ in Zeile 7. Die Einhaltung der Komponentenbeschränkungen (3.11a) und (3.11b) im Laufe des iterativen Lösungsprozesses ist gesichert, sofern die Startnäherungen im zulässigen Bereich liegen.

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

```
1  void Prg_T2Topt::setup(int k,
2                         Omu_Vector &x, Omu_Vector &u, Omu_Vector &c)
3  {
4      x.alloc(1+2);          // Anzahl Zustandsgroessen
5                             // (1 zeitdiskret, 2 kontinuierlich)
6      if ( k == 0 ) {
7          x.min[0] = 0.1; // freier Anfangswert zeitdiskrete Zustandsgroesse x0
8          x.min[1] = x.max[1] = -1.0; // fester Anfangszustand x1(0)
9          x.min[2] = x.max[2] = 0.0;  // fester Anfangszustand x2(0)
10         // numerische Initialisierung (Startnaeherung)
11         x.initial[0] = _tf;
12         x.initial[1] = -1.0;
13         x.initial[2] = 0.0;
14     }
15     else if ( k == K() ) {
16         x.min[1] = x.max[1] = 0.0; // fester Anfangszustand x1(tf)
17         x.min[2] = x.max[2] = 0.0; // fester Anfangszustand x2(tf)
18     }
19     if ( k < K() ) {
20         u.alloc(1);                // Anzahl Steuergroessen
21         u.min[0] = -_uminmax;      // untere Schranke u
22         u.max[0] = _uminmax;       // obere Schranke u
23         u.initial[0] = 0.0;        // numerische Initialisierung
24     }
25 }
```

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()`.

```
1  void Prg_T2Topt::init_simulation(int k,
2                                   Omu_Vector &x, Omu_Vector &u)
3  {
4      int i;
5      // Initialisierung in 1. Zeitstufe (k=0)
```

```
    // sonst Uebernahme Endwerte der vorherigen Stufe (default)
    if ( k == 0 ) {
        for ( i = 0; i < (int) x->dim; i++ )
            x[i] = x.initial[i];
    }
    // Initialisierung Steuergroesse
    if ( k < K() )
        u[0] = u.initial[0];
}
```

In der Methode `update()` werden für jede Zeitstufe k die Stufengleichungen (zeitdiskreter Anteil) $\mathbf{f}_d^k$ (3.10) (Parameter `f`), der Anteil an der Zielfunktion f_0^k (3.12) (Parameter `f0`) sowie die Ungleichungsbeschränkungen $\mathbf{c}^k$ (3.11c) bzw. (3.11d) (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 1 und 5 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 (3.14). Für die letzte Zeitstufe K (`KK`) ist kein Steuervektor `u` und keine Stufengleichung `f` vorgesehen.

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

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

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 [12] zu berücksichtigen.

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

$$\mathbf{F} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) - \dot{\mathbf{x}} \tag{3.15}$$

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 7 und 8 in Listing 3.6 ist die Zeitnormierung entsprechend (3.2b) berücksichtigt, die Zustandsgröße `x[0]` ist zeitdiskret.

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

```
1 void Prg_T2Topt::continuous(int kk, double t,
                              const adoublev &x, const adoublev &u,
3                             const adoublev &xp, adoublev &F)
{
5     // x0: zeitdiskrete Zustandsgroesse x0(k)=tf
      // Zustandsgleichungen mit Zeitnormierung in Nullform
7     F[1] = (-0.5*x[1]+x[2])*x[0] - xp[1];
      F[2] = (-x[2]+u[0])     *x[0] - xp[2];
9 }
```

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` vorbereitet.

Mit dem Tcl-Skript `Hqp_Beispiel.tcl` erfolgt die Ablaufsteuerung, siehe Listing 3.7. Durch Aufruf des Kommandos `prg_name` (Zeile 3) wird das Optimalsteuerungsproblem ausgewählt und die Klasse durch den Konstruktoraufruf initialisiert. `prg_setup` (Zeile 5) ruft die Methoden `setup_stages()` und `setup()`, `prg_simulate` (Zeile 7) die Methode `init_simulation()` mit den entsprechenden Parametern auf. Mit `sqp_init` (Zeile 9) wird der SQP-Solver initialisiert und mit dem Aufruf von `hqp_solve` (Zeile 11) 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 19) in eine Textdatei geschrieben.

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

Listing 3.7: `Hqp_Beispiel.tcl`

```
1 foreach beispiel [list Di Di2 T2Topt Dising] {
    # Auswahl Optimalsteuerungsproblem
3   prg_name $beispiel
    # Initialisierung Problem: setup_stages(), setup()
```

```
    prg_setup
    # Initialisierung Optimierungsvariable: simulate()
    prg_simulate
    # Initialisierung Solver
    sqp_init
    # Start Optimierungslauf
    catch hqp_solve result
    puts [format "Status:          %s" $result]
    if { $result == "optimal" } {
        puts [format "Zielfunktional: %g" [prg_f]]
        puts [format "Rechenzeit:     %.1fs" [toc]]
        # Ausgabe [t x u] in ASCII-Datei
        # Einlesen z.B. in Matlab:
        #    [t x1 x2 x3 u] = textread('Di_Ergebnis.txt','%f %f %f %f %f');
        toASCII [prg_name]_Ergebnis.txt
        puts [format "Ergebnisse nach %s\n\n" [prg_name]_Ergebnis.txt]
    }
}
```

Der Optimierungslauf für die zeitoptimale Umsteuerung des PT_2-Glieds liefert die folgenden Ausgaben.

```
 it          obj    ||inf||   ||grdL|| [ qp res]      ||s||       s'Qs stepsize
  0            1     0.6065          1 [ 20 opt]          5     1e-005        1
  1          0.1     0.1065 5.229e-007 [ 28 opt]      5.344 1.241e-005      0.1
  2     0.165523    0.09591  0.0009489 [  7 opt]      4.809     0.1633        1
  3      0.77527    0.05516   0.004427 [ 30 opt]      4.454    0.01059        1
  4     0.826284   0.004558   0.006084 [ 30 opt]      4.454   0.005148        1
  5     0.815175    0.00102   0.004907 [ 24 opt]      1.213 2.452e-005        1
  6     0.814645 1.269e-005    0.03182 [ 13 opt] 1.926e-005 2.402e-010        1
  7     0.814645 2.982e-013   0.005305 [ 12 opt] 1.448e-007 1.352e-010
                                       164 qp-it
Status:          optimal
Zielfunktional: 0.814645
Rechenzeit:     0.1s
```

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 Lagrange-Funktion. 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 Beschränkungsverletzung 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

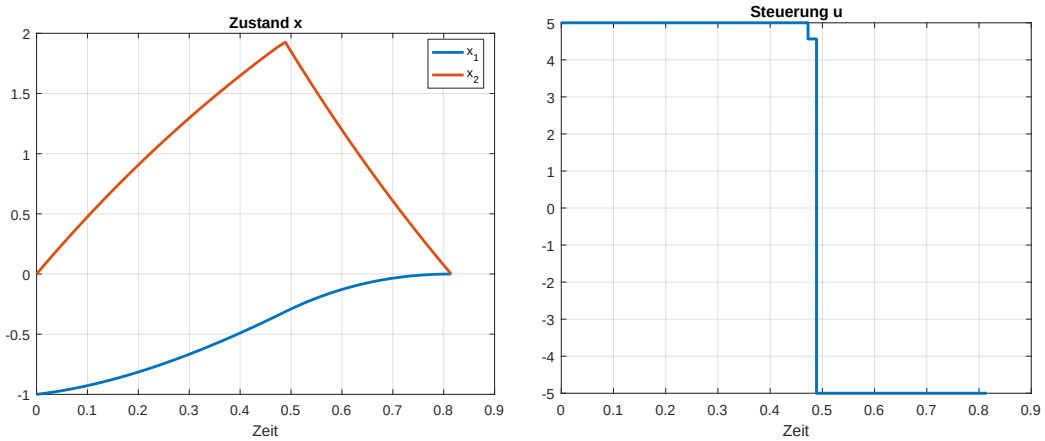


Abbildung 3.2: Zeitoptimale Umsteuerung PT_2-Glied; Mehrstufen-Steuerungsparametrisierung mit HQP.

Abbildung 5.1 zeigt – neben dem Fehlen der hier nicht berechneten Kozustandsgrößen und der HAMILTON-Funktion – geringfügige Abweichungen im Verlauf der Steuergröße in der Umgebung des Sprungzeitpunkts.

3.1.2.2 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

```
const 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.

```
Prg_Di2::Prg_Di2()
{
    set_K(50);          // Anzahl Stufen (stages)
    _x1max = 0.12;      // Zustandssbeschraenkung
    _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 18.

Da die Endzeit t_f fest ist, wird die Zeittransformation (3.2) 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 (3.13) zur Berechnung des Integralterms im Zielfunktional eingeführt.

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

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

In der Methode `update()` wird für jede Zeitstufe k der Anteil f_0^k (3.12) 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 (3.13).

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

```
void Prg_Di2::update(int kk,
                     const adoublev &x, const adoublev &u,
                     adoublev &f, adouble &f0, adoublev &c)
{
```

```
5     if ( kk < KK() )
          f0 = f[2]-x[2]; // Zielfunktional: Anteil Zeitstufe
7 }
```

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,
2                        const adoublev &x, const adoublev &u,
                         const adoublev &xp, adoublev &F)
4 {
    // Zustandsgleichungen in Nullform
6   F[0] = x[1]           - xp[0];
    F[1] = u[0]           - xp[1];
8   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-007 [  8 opt]      9.707 1.316e-005        1
  2      3.70657 1.471e-014 1.142e-007
                                        26 qp-it
Status:          optimal
Zielfunktional: 3.70657
Rechenzeit:      0.1s
```

Eine grafische Darstellung der Ergebnisse findet sich in Abbildung 3.3.

3.1.3 Einbindung einer Optimalsteuerungsaufgabe in Hqp unter Nutzung der FMI-Schnittstelle

Ein Functional Mock-Up Interface (FMI, siehe https://fmi-standard.org/) ist eine standardisierte Schnittstelle zum Modellaustausch zwischen und zur Co-Simulation mit unterschiedlichen Simulationswerkzeugen. Der Modellaustausch erfolgt in Form von Functional Mock-Up Units (FMU), dies sind Archivdateien, die sowohl die Modellbeschreibung im XML-Format als auch die Modellgleichungen als Binärdateien (DLL) und C- bzw. C++-Quelltext enthalten.

SIMULINK, DYMOLA, OPENMODELICA und viele weitere Simulationswerkzeuge unterstützen sowohl den Import als auch den Export von FMUs.

HQP besitzt eine FMI-Schnittstelle und kann damit ein Modell als FMU direkt einbinden. Die Modellgleichungen sind dabei

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad t \in (t_0, t_f), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{3.16a}$$

$$\mathbf{y} = \mathbf{e}(\mathbf{x}, \mathbf{u}, t) \tag{3.16b}$$

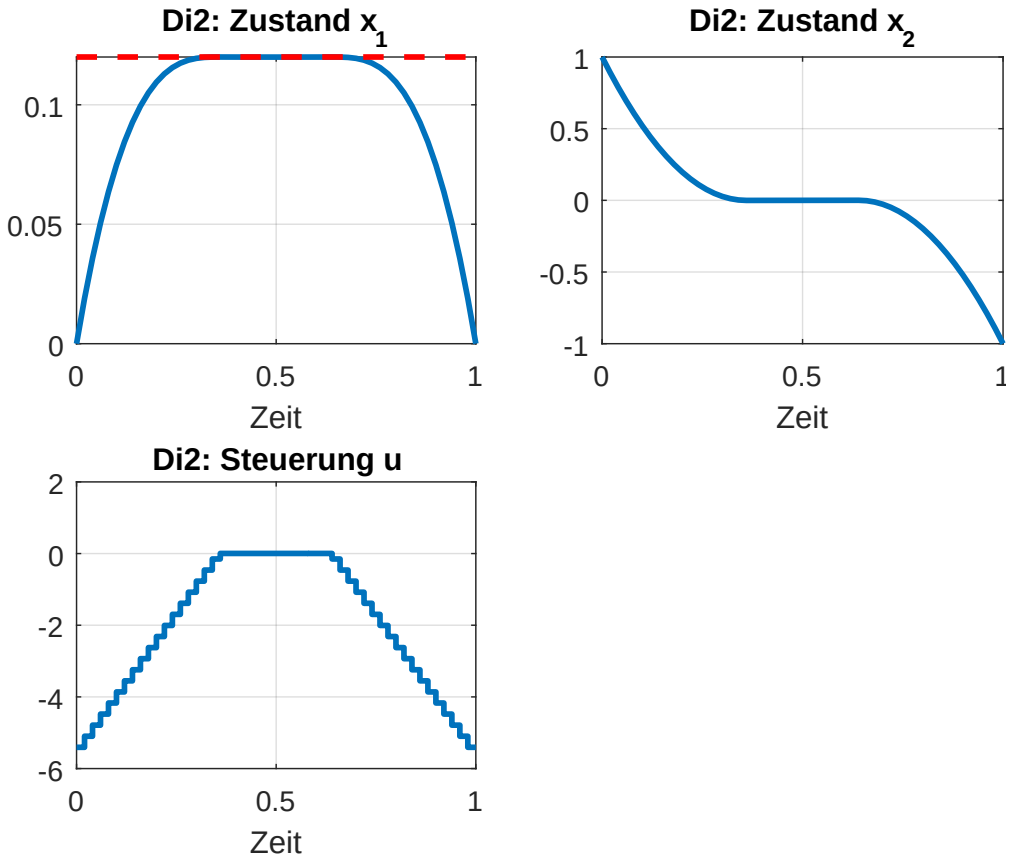


Abbildung 3.3: Doppelintegrator mit Zustandsbeschränkung; Mehrstufen-Steuerungsparametrisierung mit HQP.

Das vordefiniert Optimalsteuerungsproblem `DynamicOpt` implementiert Komponentenbeschränkungen für die Steuer- und Ausgangsgrößen in den Gitterpunkten sowie ein Zielfunktional, das sowohl die Ausgangs- als auch die Steuergrößen und ihre Änderungen linear und quadratisch bewertet.

$$\mathbf{u}_{\min}^k \leq \mathbf{u}^k \leq \mathbf{u}_{\max}^k \tag{3.16c}$$

$$\mathbf{y}_{\min}^k \leq \mathbf{y}^k \leq \mathbf{y}_{\max}^k \tag{3.16d}$$

$$\begin{aligned} J = \sum_{k=0}^{K} \Bigg(& \sum_{i=1}^{dim(\mathbf{y})} \left(w_{y,lin,i}(y_i^k - y_{\mathrm{ref},i}^k) + w_{y,quad,i}(y_i^k - y_{\mathrm{ref},i}^k)^2 \right) \\ & + \sum_{i=1}^{dim(\mathbf{u})} \left(w_{ud,lin,i}(u_i^{k+1} - u_i^k - u_{\mathrm{d,ref},i}^k) + w_{u,quad,i}(u_i^{k+1} - u_i^k - u_{\mathrm{d,ref},i}^k)^2 \right) \\ & + \sum_{i=1}^{dim(\mathbf{u})} \left(w_{u,lin,i}(u_i^k - u_{\mathrm{ref},i}^k) + w_{u,quad,i}(u_i^k - u_{\mathrm{ref},i}^k)^2 \right) \Bigg) \longrightarrow \min! \end{aligned} \tag{3.16e}$$

Wenn das zu lösende Optimalsteuerungsproblem zusätzliche Beschränkungen (beispielsweise allgemeine Trajektorienbeschränkungen (3.1c)) oder weitere (nicht-quadratische) Terme im Zielfunktional beinhaltet, dann ist das durch zusätzliche Ausgangsgrößen y_i mit entsprechenden Ausgangsgleichungen (3.16b) zu beschreiben.

Der wesentliche Vorteil bei der Nutzung der FMI-Schnittstelle besteht darin, dass keine C++-Programme zu erstellen sind. Das Modell des zu optimierenden Systems kann mit einem geeigneten Simulationsprogramm erstellt und getestet werden, und die Parameter des Optimalsteuerungsproblems werden direkt über die Tcl-Schnittstelle definiert.

3.1.3.1 Doppelintegrator mit singulärem Lösungsabschnitt

Listing 3.12 zeigt die Modellgleichungen des Doppelintegrators in MODELICA. Die Zustandsgrößen `y1` und `y2` sind zugleich Ausgangsgrößen. Die zweite Eingangsgröße `u2` geht nicht in die Modellgleichungen ein und wird später zur Zeitskalierung verwendet.

Listing 3.12: MODELICA-Modell `Dising.mo`.

```
model Dising "Doppelintegrator mit Zeitskalierung"
  parameter Real y1_start = -3.0 "Anfangswert Zustand 1";
  parameter Real y2_start = 0.0 "Anfangswert Zustand 2";
  input Real u1, u2;
  output Real y1, y2;
initial equation
  y1 = y1_start;
  y2 = y2_start;
equation
  der(y1) = y2;
  der(y2) = u1;
end Dising;
```

Das MODELICA-Modell `Dising.mo` wird nun als FMU mit dem Dateinamen `Dising.fmu` exportiert, hierzu wird der `Modelica`-Compiler `omc.exe` aus OPENMODELICA eingesetzt, siehe Listing 3.13.

Listing 3.13: `FMI_Beispiel.tcl`: Generierung FMU aus MODELICA-Modell mit `omc.exe`.

```
set omc {C:/OpenModelica1.13.264bit/bin/omc +simCodeTarget=Cpp}
    # call omc, check for errors and clean up
    if {[catch {eval [concat exec $omc "omc_tcl_commands.mos"]} log]} {
        cd $cwd
        error "Could not compile FMU: $log"
    }
```

Die Einbindung des Modells kann nun direkt in Tcl (ohne weitere C++-Programmierung) erfolgen, siehe Listing 3.14. Durch Aufruf des Kommandos `prg_name` in Zeile 4 wird das vordefinierte Optimalsteuerungsproblem `DynamicOpt` ausgewählt. `mdl_path` in Zeile 5 wählt die FMU-Datei aus. In den Zeilen 7–10 wird das Zeitgitter mit `prg_setup_stages` definiert. In den Zeilen 12–21 werden die Zeitstufen (Variable `ts`) und die Steuergrößen (Variable `us`) initialisiert. Der Anfangszustand wird mit `mdl_x0` in Zeile 23 und der Endzustand als Komponentenbeschränkung für den Endwert der Ausgangsgrößen mit `mdl_yf_min` und `mdl_yf_max` in den Zeilen 28 und 29 vorgegeben.

Die Wichtungen der Quadrate der Ausgangsgrößen im Integralterm des Zielfunktionals werden mit `mdl_y_weight2` in Zeile 39 vorgegeben.

Die Beschränkungen der Steuergrößen werden mit `mdl_u_min` und `mdl_u_max` in den Zeilen 25 und 26 definiert. Die erste Komponente der Steuergrößen ist die Steuergröße u des Optimalsteuerungsproblems (2.11), die zweite Komponente ist der freie Parameter Endzeit t_f, festgelegt mit `mdl_t_scale_idx` in Zeile 37. Beide Steuergrößen sind zwischen den Gitterpunkten konstant (Interpolationspolynom 0. Ordnung): `mdl_u_order` in Zeile 31, und beide Steuergrößen sind zu optimieren: `mdl_u_active` in Zeile 33. Mit der Vorgabe für `mdl_u_decimation` in Zeile 35 wird festgelegt, dass die erste Komponente der Steuergrößen in jedem Gitterpunkt geändert werden kann, während die zweite Komponente, der Parameter t_f, über den gesamten Optimierungshorizont konstant sein soll. Der MAYER-Term ρt_f in (2.11d) wird umgeformt in $\rho \int_0^{t_f} dt$ und mit $w_{ud,lin,2} = \rho$ sowie $u_{\mathrm{d,ref},2} = -1$ in in das Zielfunktional (3.16e) einbezogen (`mdl_der_u_ref` und `mdl_der_u_weight1` in Zeile 40 und 41). Die Änderung der zweiten Steuergröße ist null, da diese über den gesamten Optimierungshorizont konstant ist.

Wie in Abschnitt 3.1.2 beschrieben, initialisiert `prg_setup` (Zeile 44) das Optimierungsproblem und `prg_simulate` (Zeile 46) die Zustandsgrößen durch eine Simulation mit den Startwerten der Steuergrößen. Mit `sqp_init` (Zeile 48) wird der SQP-Solver initialisiert und mit dem Aufruf von `hqp_solve` (Zeile 50) der eigentliche Optimierungslauf gestartet.

Listing 3.14: `FMI_Beispiel.tcl`: Optimalsteuerungsproblem Doppelintegrator mit singulärem Lösungsabschnitt.

```
set fmuname Dising
package require Omuses
# Optimalsteuerungsproblem und Modell
prg_name DynamicOpt
mdl_path $fmuname.fmu
# Anzahl Gitterpunkte
prg_KK 100
prg_sps 1
prg_multistage 1
prg_setup_stages
# Initialwerte Steuergroessen und Zeitgitter
set KK [prg_KK]
set dt [expr 1.0/$KK]
set us {}
set ts {}
for {set kk 0} {$kk <= $KK} {incr kk} {
    lappend us {0.0 4.0}
    lappend ts [expr $dt*$kk]
}
mdl_us $us
prg_ts $ts
# Anfangszustand in Dising.mo festgelegt
mdl_x0          {-3.0 0.0}
# Trajektorienbeschraenkungen fuer alle Zeitpunkte
mdl_u_min       {-1.0 0.1}
mdl_u_max       {1.0 Inf}
# Beschraenkungen zum Endzeitpunkt
mdl_yf_min      {0.0 0.0}
mdl_yf_max      {0.0 0.0}
# Interpolationsordnung Steuergroessen
mdl_u_order     {0 0}
```

```
32 # aktive (zu optimierende) Steuergroessen
   mdl_u_active     {1 1}
34 # Zeitintervalle mit konstanten Steuergroessen
   mdl_u_decimation [list 1 $KK]
36 # Steuergroesse 1 zur Zeitskalierung
   mdl_t_scale_idx {1}
38 # Terme Zielfunktional
   mdl_y_weight2    {0.5 0.5}
40 mdl_der_u_ref    {0.0 -1.0}
   mdl_der_u_weight1 {0.0 0.1}
42 ## Loesung Optimierungsproblem
   # Initialisierung Problem
44 prg_setup
   # Initialisierung Optimierungsvariable
46 prg_simulate
   # Initialisierung Solver
48 sqp_init
   # Start Optimierungslauf
50 catch hqp_solve result
```

Das Tcl-Skript `FMI_Beispiel.tcl` wird beim Aufruf von `Hqp_Beispiel.exe` übergeben

```
Hqp_Beispiel FMI_Beispiel.tcl
```

Der Optimierungslauf für den Doppelintegrator mit singulärem Lösungsabschnitt liefert die folgenden Ausgaben.

```
 it          obj    ||inf||   ||grdL|| [ qp res]      ||s||       s'Qs stepsize
  0         18.4          3     0.4235 [ 24 opt]        3.9      3.588        1
  1     0.203894      1.462     0.4125 [ 20 opt]          5      12.47        1
  2      8.24266       19.5     0.2535 [ 25 opt]      4.934    0.01493        1
...
 21      8.00725  6.97e-013      1.651 [ 34 opt]  0.0009115 2.747e-005        1
 22      8.00722 9.842e-011       4.63 [  9 opt] 5.376e-005 1.786e-007        1
 23      8.00721 7.642e-013      11.57 [  5 opt] 1.887e-005 1.863e-008
                                      571 qp-it
Result   : optimal
Objective: 8.007213960619563
Obj-evals: 25
```

Eine grafische Darstellung der Ergebnisse findet sich in Abbildung 3.4.

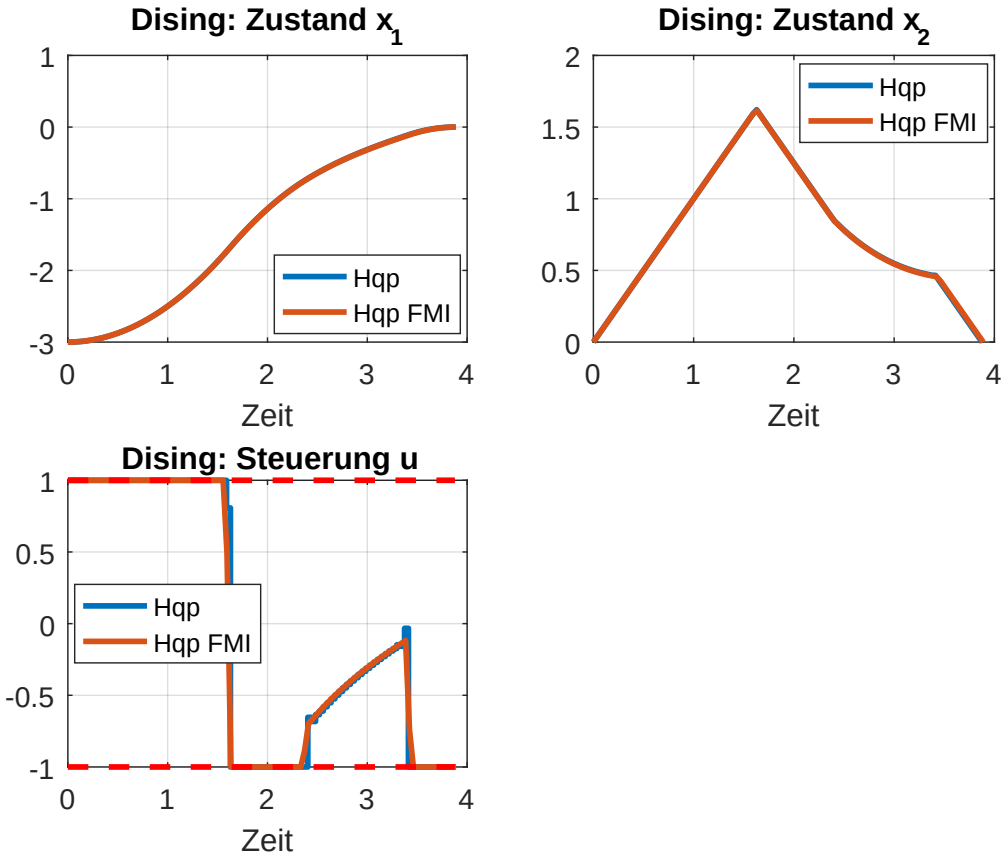


Abbildung 3.4: Doppelintegrator mit singulärem Lösungsabschnitt; Mehrstufen-Steuerungsparametrisierung mit HQP, Vergleich C++-Implementierung und Nutzung der FMI-Schnittstelle.

3.2 Mehrstufen-Steuerungsparametrisierung mit ACADO

ACADO (Toolkit for Automatic Control and Dynamic Optimization [13], https://acado.github.io) ist ebenfalls eine Implementierung einer Mehrstufen-Steuerungsparametrisierung, siehe Abschnitt 3. Das resultierende nichtlineare Optimierungsproblem wird mit einem SQP-Verfahren gelöst. Zur Lösung der lokalen quadratischen Optimierungsprobleme wird qpOASES eingesetzt, damit ist ACADO (derzeit) für sehr große Optimierungsprobleme mit einer großen Anzahl von Stufen K und/oder Eingangsgrößen $\mathbf{u}$ weniger gut geeignet.

Für die numerische Integration der Zustandsdifferentialgleichungen (auch differential-algebraische (DAE) und zeitdiskrete Modelle sind möglich) stehen mehrere Runge-Kutta- und BDF-Integratoren zur Verfügung. Zur Berechnung der benötigten Ableitungen kommt eine automatische Differentiation zum Einsatz. Zur Approximation der Hesse-Matrix der Lagrange-Funktion können quasi-Newton-Verfahren (BFGS), Gauss-Newton-Verfahren oder auch exakte 2. Ableitungen verwendet werden.

ACADO besitzt sowohl eine Matlab- als auch eine C++-Schnittstelle. Das Optimalsteuerungsproblem wird in einer einfachen Modellierungssprache formuliert, aus der dann mittels automatischer Codegenerierung der eigentliche C++-Programmcode erzeugt wird.

ACADO ist insbesondere geeignet zur Lösung von Optimierungsproblemen der nichtlinearen modell-prädiktiven Regelung (MPC). In diesem Modus wird C-Code generiert, und es kommt ein „real-time iteration scheme“ zum Einsatz.

Eine Dokumentation der Matlab-Schnittstelle von ACADO sowie zahlreiche Beispiele, die weitere Details wie die Initialisierung der Optimierungsvariablen, zeitabhängige Beschränkungen, die

Parameter	Wert (default)	Beschreibung
`MAX_NUM_ITERATIONS`	int (200)	maximale Anzahl SQP-Iterationen
`KKT_TOLERANCE`	double (10^{-6})	Abbruchschranke SQP-Algorithmus
`HESSIAN_APPROXIMATION`		Approximation Hesse-Matrix
	`CONSTANT_HESSIAN`	Hesse-Matrix konstant
	`FULL_BFGS_UPDATE`	BFGS-Update (dense)
	`BLOCK_BFGS_UPDATE` (*)	BFGS-Update (Blockstruktur)
	`GAUSS_NEWTON`	Gauss-Newton-Approximation
	`EXACT_HESSIAN`	exakte 2. Ableitungen
`DISCRETIZATION_TYPE`	`SINGLE_SHOOTING`	einfache Steuerungsparametrisierung
	`MULTIPLE_SHOOTING` (*)	Mehrstufen-Steuerungsparametrisierung
`INTEGRATOR_TYPE`		numerisches Integrationsverfahren
	`INT_RK12`	Euler-Verfahren (adaptiv)
	`INT_RK23`	Runge-Kutta Ordnung 2/3
	`INT_RK45`	Runge-Kutta Dormand-Prince 4/5
	`INT_RK78`	Runge-Kutta Dormand-Prince 7/8
	`INT_BDF`	Gear-Verfahren (BDF)
`INTEGRATOR_TOLERANCE`	double	relative Toleranz numerische Integration
`ABSOLUTE_TOLERANCE`	double	absolute Toleranz numerische Integration

Tabelle 3.1: ACADO: Parameter `algo.set( <Parameter> , <Wert> )`.

Definition der Zustandsdifferentialgleichung in einer separaten MATLAB-Funktion und Parameter für die `MEX`-Funktion beinhalten, sind über die o. g. ACADO-Website zu finden.

ACADO soll zukünftig um Kollokationsansätze erweitert werden. Die Einbindung zusätzlicher Solver für große und strukturierte Aufgaben, sowohl für die lokalen QP-Probleme als auch für das nichtlineare Optimierungsproblem, ist ebenso geplant wie eine SIMULINK-Schnittstelle.

3.2.1.1 Doppelintegrator mit singulärem Lösungsabschnitt

Das folgende Listing zeigt die Lösung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 mit ACADO unter Nutzung der MATLAB-Schnittstelle.

Listing 3.15: `dising_acado.m`

```
function dising_acado()
acado_path
% Parameter Problem
x0 = [-3; 0];    % Anfangswerte
xf = [0; 0];     % Endwerte
uminmax = 1;     % Steuerungsbeschraenkung
rho_tf = 0.1;    % Bewertung Endzeit
BEGIN_ACADO;
    acadoSet('problemname', 'dising');
    DifferentialState       x1 x2;
    Parameter               T;
    Control                 u;
    f = acado.DifferentialEquation(0.0, T);
    f.add( dot(x1) == x2 );
    f.add( dot(x2) == u );
    ocp = acado.OCP(0.0, T, 100);
    ocp.minimizeLagrangeTerm( 0.5*(x1^2+x2^2) );
    ocp.minimizeMayerTerm( rho_tf*T );
    ocp.subjectTo( f );
    ocp.subjectTo( 'AT_START', x1 == x0(1) );
    ocp.subjectTo( 'AT_START', x2 == x0(2) );
    ocp.subjectTo( 'AT_END', x1 == xf(1) );
    ocp.subjectTo( 'AT_END', x2 == xf(2) );
    if isfinite(uminmax)
        ocp.subjectTo( -uminmax <= u <= uminmax );
    end
    ocp.subjectTo( 0.1 <= T <= 10.0 );
    algo = acado.OptimizationAlgorithm(ocp);
    algo.set( 'INTEGRATOR_TYPE', 'INT_RK78' );
    %algo.set( 'INTEGRATOR_TOLERANCE', 1e-10 );
    %algo.set( 'ABSOLUTE_TOLERANCE', 1e-8 );
    algo.set( 'HESSIAN_APPROXIMATION', 'EXACT_HESSIAN' );
    %algo.set( 'KKT_TOLERANCE', 1e-12 );
    algo.set( 'MAX_NUM_ITERATIONS', 1000 );
END_ACADO;
out = dising_RUN();
tf = out.PARAMETERS(1, 2);
```

```
38 J = out.STATES(end, 4)+rho_tf*tf;
   fprintf('Optimaler Wert Zielfunktional: %g, optimale Endzeit: %g\n', J, tf);
```

In Zeile 2 wird der MATLAB-Suchpfad angepasst. Die Definition des Optimalsteuerungsproblems erfolgt zwischen den Schlüsselworten `BEGIN_ACADO` (Zeile 8) und `END_ACADO` (Zeile 35).

Zunächst wird mit `acadoSet()` (Zeile 9) ein Problemname festgelegt, aus dem dann die Dateinamen für den erzeugten C++-Code und die `MEX`-Datei gebildet werden.

Die Zustandsgrößen werden mit `DifferentialState`, Parameter mit `Parameter` und Steuergrößen mit `Control` vereinbart (Zeile 10–12). Eine freie Endzeit ist ein Parameter, für den zwingend der Variablenname `T` zu wählen ist.

In den Zeilen 13–15 wird die Zustandsdifferentialgleichung definiert und in Zeile 19 dem Optimalsteuerungsproblem als Beschränkung hinzugefügt. Dabei kennzeichnet `dot()` die Zeitableitung der Zustandsgröße.

In Zeile 16 wird das Optimalsteuerungsproblem unter Angabe des Zeithorizonts $[0, T]$ und (optional) der Anzahl der Zeitstufen K der Mehrstufen-Steuerungsparametrisierung initialisiert.

Die Komponenten des Zielfunktionals werden in Zeile 17 und 18 mit `minimizeLagrangeTerm()` und `minimizeMayerTerm()` angegeben.

Die Beschränkungen werden in den Zeilen 19–27 mit `subjectTo()` definiert, wobei `AT_START` Anfangsbedingungen und `AT_END` Endbedingungen bezeichnet.

In den Zeilen 28–34 werden schließlich Parameter für die numerische Lösung festgelegt, siehe auch Tabelle 3.1.

Mit dem Aufruf von `<problemname>_RUN()` in Zeile 36 wird die Codegenerierung, die Übersetzung und der eigentliche Optimierungslauf gestartet. Die Ergebnisse werden in der Datenstruktur `out` übergeben: `out.CONTROLS`, `out.STATES` und `out.PARAMETERS` sind Matrizen, deren Zeilen den Zeitstufen k zugeordnet sind und deren Spalten die Werte der (normierten) Zeit (Spalte 1) bzw. der Steuergrößen, Zustandsgrößen und Parameter beinhalten. Der integrierte LAGRANGE-Term des Zielfunktionals ist als zusätzliche Spalte in `out.STATES` enthalten.

Für $K = 100$ Zeitstufen findet ACADO mit 17 Iterationen eine Lösung des nichtlinearen Optimierungsproblems, die zugehörigen Zeitverläufe zeigt Abbildung 4.2.

```
sqp it | qp its |       kkt tol |       obj val |     merit val |      ls param |
     1 |      1 | 7.431344e+001 | 2.531984e-001 | 2.261541e+001 | 5.000000e-001 |
     2 |    161 | 5.419142e+001 | 5.516635e+000 | 5.791409e+001 | 5.000000e-001 |
     3 |    166 | 3.406628e+000 | 7.167420e+000 | 9.274432e+000 | 1.000000e+000 |
     4 |     85 | 8.896290e-001 | 8.017731e+000 | 8.064913e+000 | 1.000000e+000 |
     5 |     75 | 7.693392e-002 | 7.981862e+000 | 8.031140e+000 | 1.000000e+000 |
...
    15 |     76 | 2.181793e-006 | 8.006800e+000 | 8.006801e+000 | 1.000000e+000 |
    16 |     77 | 1.459703e-006 | 8.006799e+000 | 8.006800e+000 | 1.000000e+000 |
    17 |     77 | 9.711392e-007 | 8.006798e+000 | 8.006799e+000 | 1.000000e+000 |

Covergence achieved. Demanded KKT tolerance is 1.000000e-006.
```

```
Optimaler Wert Zielfunktional: 8.0068, optimale Endzeit: 3.87726
```

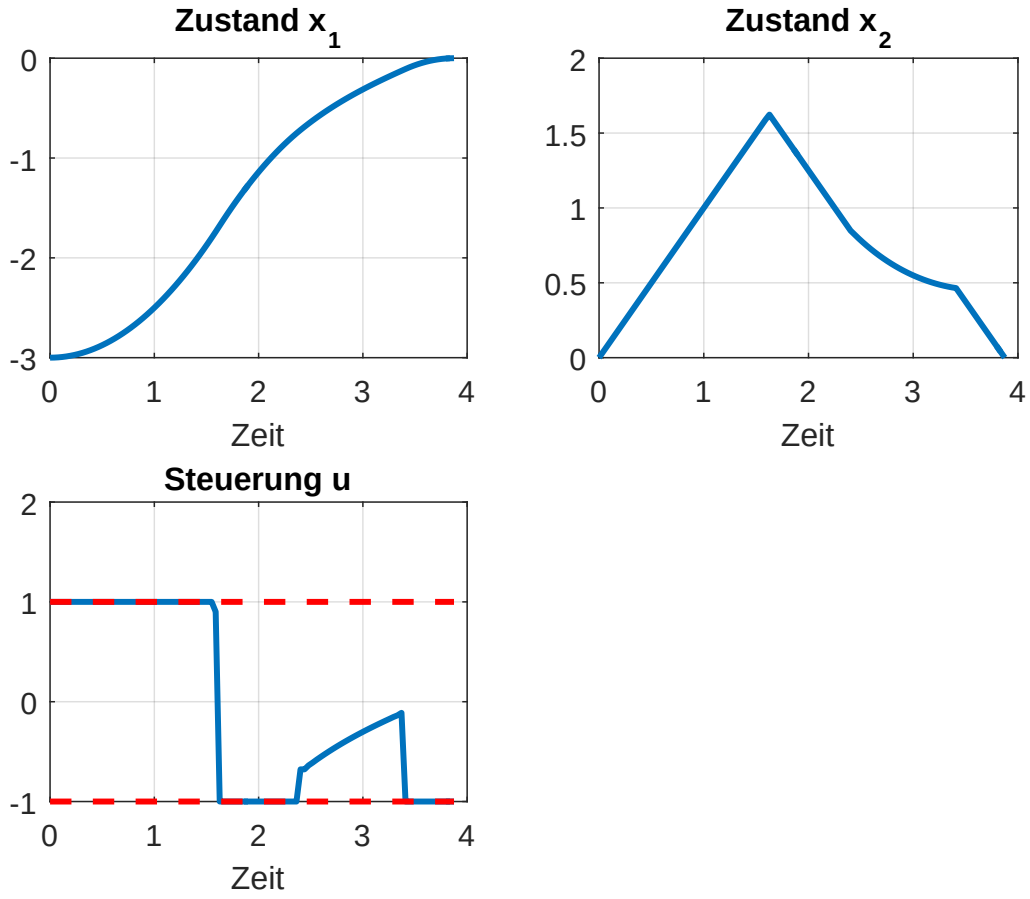


Abbildung 3.5: Doppelintegrator mit singulärem Lösungsabschnitt; Mehrstufen-Steuerungsparametrisierung mit ACADO ($K = 100$).

3.3 Mehrstufen-Steuerungsparametrisierung mit CasADi

CasADi [2] (https://web.casadi.org ist ein Software-Framework zur numerischen Optimierung. CasADi ist selbst kein Solver für Optimalsteuerungsprobleme, sondern stellt die wesentlichen Softwarebausteine für die Implementierung solcher Solver bereit. Dies beinhaltet automatische Differentiation, Anbindung verschiedener numerischer Integratoren zur Lösung von Anfangswertproblemen für gewöhnliche Differentialgleichungen und differential-algebraische Gleichungen sowie Solver für lineare, quadratische, nichtlineare und gemischt-ganzzahlige Optimierungsprobleme. CasADi besitzt Software-Schnittstellen u. a. zu C++, Python und Matlab.

Die Implementierung der Mehrstufen-Steuerungsparametrisierung in Matlab und CasADi basiert im wesentlichen auf

> `direct_multiple_shooting.m`
> An implementation of direct multiple shooting. Joel Andersson, 2016.

aus der Beispielsammlung `casadi-example_pack-v3.4.5`.

3.3.1.1 Zeitoptimale Umsteuerung eines PT_2-Glieds

Im Folgenden wird die zeitoptimale Umsteuerung des PT_2-Glieds nach Abschnitt 2.3 betrachtet. Der Zeithorizont wird normiert (siehe (3.2)), so dass die ursprünglich freie Endzeit als zu optimierender Parameter in das Problem eingeht. Das Zielfunktional ergibt sich dann zu

$$J = \int_0^1 t_f d\tau \,.$$

Die Steuergröße wird als abschnittsweise konstante Zeitfunktion auf einem äquidistanten Zeitgitter angesetzt.

Listing 3.16 zeigt die Implementierung in Matlab und CasADi. In den Zeilen 2 und 3 werden der Matlab-Suchpfad angepasst und das Package `casadi` wird importiert. In den Zeilen 5 bis 13 werden die Parameter des Optimalsteuerungsproblems definiert. `N` bezeichnet die Anzahl der Zeitstufen (K in (3.5)).

In den Zeilen 15–17 werden die symbolischen Variablen `x` und `u` für die Zustands- und Steuergrößen und `tf` für den Parameter t_f angelegt. `MX` steht für 'matrix expression', in CasADi eine Matrix (oder Vektor oder Skalar), deren Elemente symbolische Ausdrücke sind, die durch Folgen matrix- (oder vektor- oder skalar-)wertiger Rechenoperationen erzeugt werden. `xdot` (Zeile 19) und `L` (Zeile 21) sind die symbolischen Ausdrücke für die rechte Seite der Zustandsgleichung bzw. den Integranden des Zielfunktionals.

In den Zeilen 24–26 wird die Funktion `F` definiert, die die Zustandsdifferentialgleichung `xdot` und den Intergralterm des Zielfunktionals `L` über den Zeithorizont eines Abschnitts `T/N` mit dem Integrationsalgorithmus `CVODES` numerisch integriert.

Nun wird in den Zeilen 28–68 das nichtlineare Optimierungsproblem definiert. Der Vektor der Optimierungsvariablen `w` mit den Komponentenbeschränkungen `lbw`$\leq$`w`$\leq$`ubw` und den Initialwerten `w0` wird aus dem Parameter t_f (Zeile 28), den konstanten Approximationen der Steuergröße in den

Zeitstufen u^k (Zeile 46) und den Anfangszuständen der Zeitstufen $\mathbf{x}^k$ (Zeile 38 und 56) zusammengesetzt. Die nichtlinearen Beschränkungen `lbg`$\leq$`g(w)`$\leq$`ubg` sind hier die Stetigkeitsbedingungen der Zustandsgrößen (3.7) an den Übergängen der Zeitstufen (Zeilen 61–63) und der feste Endzustand (Zeilen 66–68).

Anschließend wird das Solver-Objekt `solver` mit dem zuvor definierten nichtlinearen Optimierungsproblem und dem Solver IPOPT in den Zeilen 70–71 definiert und in Zeile 73 aufgerufen und damit das nichtlineare Optimierungsproblem gelöst.

Aus dem Lösungsvektor `w_opt` werden die optimale Endzeit `tf_opt`, die Werte der Zustandsgrößen an den Gitterpunkten `x_opt` und die approximierten Steuergrößen `u_opt` extrahiert (Zeilen 76–80). Zuletzt wird das Zeitgitter entnormiert (Zeile 83).

Listing 3.16: `t2topt_multiple_shooting_tf.m`

```
function t2topt_multiple_shooting_tf(uminmax)
casadi_path()
import casadi.*
% Parameter Problem
x0 = [-1; 0];               % Anfangszustand
xf = [0; 0];                % Endzustand
nx = length(x0);
T = 1;                      % normierte Endzeit
tfmin = 0.1;                % Beschraenkung Endzeit
uinit = 0;                  % Initialwert Steuerung
xinit = [1; 1];             % Initialwert Zustand
tfinit = 1;                 % Initialwert Endzeit
N = 100;                    % Anzahl Intervalle
% Modellvariable fuer CasADi
x = MX.sym('x', nx);
u = MX.sym('u');
tf = MX.sym('tf');          % freie Endzeit als freier Parameter
% Zustands-DGL
xdot = [-0.5*x(1)+x(2); -x(2)+u]*tf;
% Lagrange-Term Zielfunktional
L = tf;
% Formulate discrete time dynamics
% CVODES from the SUNDIALS suite
ode = struct('x', x, 'p', [u; tf], 'ode', xdot, 'quad', L);
opts = struct('tf', T/N);
F = integrator('F', 'cvodes', ode, opts);
% Start with an empty NLP and optimization variable tf
w = {tf};
w0 = tfinit;
lbw = tfmin;
ubw = Inf;
J = 0;
g = {};
lbg = [];
ubg = [];
% "Lift" initial conditions
Xk = MX.sym('X0', nx);
w = [w(:)' {Xk}];
```

```
lbw = [lbw; x0];
ubw = [ubw; x0];
w0 = [w0; x0];
% Formulate the NLP
for k=0:N-1
    % New NLP variable for the control
    Uk = MX.sym(['U_' num2str(k)]);
    w = [w(:)' {Uk}];
    lbw = [lbw; -uminmax];
    ubw = [ubw;  uminmax];
    w0 = [w0; uinit];
    % Integrate till the end of the interval
    Fk = F('x0', Xk, 'p', [Uk; tf]);
    Xk_end = Fk.xf;
    J = J+Fk.qf;
    % New NLP variable for state at end of interval
    Xk = MX.sym(['X_' num2str(k+1)], nx);
    w = [w(:)' {Xk}];
    lbw = [lbw; -Inf*ones(nx, 1)];
    ubw = [ubw; Inf*ones(nx, 1)];
    w0 = [w0; xinit];
    % Add equality constraint
    g = [g(:)' {Xk_end-Xk}];
    lbg = [lbg; zeros(nx, 1)];
    ubg = [ubg; zeros(nx, 1)];
end
% Fixed final state components
g = [g(:)' {Xk_end(1:2)}];
lbg = [lbg; xf];
ubg = [ubg; xf];
% Create an NLP solver
prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
solver = nlpsol('solver', 'ipopt', prob);
% Solve the NLP
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw, 'lbg', lbg, 'ubg', ubg);
w_opt = full(sol.x);
% Separate the solution
tf_opt = w_opt(1);
for i = 1:nx
    x_opt(:, i) = w_opt(1+i:nx+1:end);
end
u_opt = w_opt(1+nx+1:nx+1:end);
% Zeit entnormieren
fprintf('Optimale Endzeit: %g\n', tf_opt)
t = (0:N)'/N*T*tf_opt;
```

Ipopt löst das Optimalsteuerungsproblem für die zeitoptimale Umsteuerung des PT_2-Glieds Kollokationsansatz in 15 Iterationen.

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 1.0000000e+000 2.00e+000 1.37e-018  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 2.3014615e+000 3.72e-002 1.19e+002  -1.0 1.99e+000    -  4.24e-001 1.00e+000h  1
```

```
   2 1.6166129e+000 8.13e-003 2.85e+001  -1.0 1.04e+000    -  8.08e-001 1.00e+000h  1
...
  13 8.1480323e-001 4.24e-008 4.31e-007  -5.7 1.43e-001    -  9.73e-001 1.00e+000f  1
  14 8.1461885e-001 1.97e-007 1.28e-007  -8.6 1.06e-001    -  9.91e-001 1.00e+000h  1
  15 8.1461867e-001 4.63e-010 2.92e-011  -8.6 1.63e-003    -  1.00e+000 1.00e+000h  1
```

3.4 Einfache Steuerungsparametrisierung mit Matlab

Die Lösung von Optimalsteuerungsaufgaben mittels Steuerungsparametrisierung lässt sich in MATLAB mit Interpolationsfunktionen zur Auswertung des parametrischen Ansatzes (3.3), ODE-Solvern zur numerischen Lösung des Anfangswertproblems (3.1a), (3.1b) und `fmincon` zur Lösung des beschränkten nichtlinearen Optimierungsproblems (3.4) realisieren.

Wenn in `fmincon` Ableitungen von Zielfunktion und Beschränkungen mit finiten Differenzen approximiert werden, dann sollten nicht die ODE-Solver von MATLAB (z. B. `ode45`) verwendet werden, da die Schrittweitensteuerung die Ableitungen verfälschen kann. Unter https://www.mathworks.com/matlabcentral/answers/uploaded_files/5693/ODE_Solvers.zip finden sich Implementierungen von ODE-Solvern mit fester Schrittweite.

3.4.1.1 Doppelintegrator mit Zustandsbeschränkung

Anhand der Optimalsteuerungsprobleme für den Doppelintegrator aus Abschnitt 2.1 und 2.2 soll die einfache Steuerungsparametrisierung erläutert werden. Listing 3.17 zeigt den MATLAB-Code.

Listing 3.17: `doint_cpara.m`

```
function doint_cpara(nr)
if nr == 1
    x0 = [0; 0];
    xf = [1; 0];
    uminmax = 4.5;
    x1max = Inf;
    x2max = 1.55;
else
    x0 = [0; 1];
    xf = [0; -1];
    uminmax = Inf;
    x1max = 0.12;
    x2max = Inf;
end
% Diskretisierung
Nu = 20;
N = 100;
% Parametervektor
w0 = zeros(Nu,1);
% Loesung NLP, Ableitungen mit finiten Differenzen
[w, J] = fmincon(@(x) obj(x, N, x0), w0, [], [], [], [], -uminmax*ones(Nu, 1), ...
    uminmax*ones(Nu, 1), @(x) con(x, N, x0, xf, x1max, x2max), ...
    optimoptions('fmincon', 'Display', 'iter', 'Algorithm', 'active-set'));
fprintf('Zielfunktional: %g\n', J)
% obj - Zielfunktional
function J = obj(w, N, x0)
xJ = solve_ode(w, N, x0);
J = xJ(end, 3);
% con - Beschrnkungen
function [c, ceq] = con(w, N, x0, xf, x1max, x2max)
```

```
    xJ = solve_ode(w, N, x0);
32  ceq = xJ(end, 1:2)'-xf;
    c = [];
34  if isfinite(x1max)
        c = [c; xJ(:, 1)-x1max];
36  end
    if isfinite(x2max)
38      c = [c; xJ(:, 2)-x2max];
    end
40  % solve_ode - Integration Zustands-DGL, Funktional
    function [xJ, ts] = solve_ode(w, N, x0)
42  persistent w_obj xJ_obj
    if isempty(w_obj) || any(w~=w_obj)
44      ts = linspace(0, 1, N)';
        xJ = ode5(@(t, x) rhs(t, x, w), ts, [x0; 0]);
46      w_obj = w;
        xJ_obj = xJ;
48  else
        xJ = xJ_obj;
50  end
    % rhs - Zustands-DGL, Zielfunktional
52  function xp = rhs(t, x, w)
    persistent w_spline pp_spline
54  if isempty(w_spline) || any(w ~= w_spline)
        pp_spline = spline(linspace(0, 1, length(w)), w);
56      w_spline = w;
    end
58  u = ppval(pp_spline, t);    % Steuerung
    xp = [x(2); u; 0.5*u*u];    % Zeitableitung Zustands-DGL, Funktional
```

In den Zeilen 2–14 werden die Parameter des Optimalsteuerungsproblems festgelegt. In Zeile 16 wird die Anzahl `Nu` der Parameter zur Approximation der Steuergröße festgelegt und in Zeile 17 die Anzahl `N` der Gitterpunkte für die Approximation der Zustandsbeschränkungen vorgegeben. Danach folgt die Initialisierung des Parametervektors `w` und der Aufruf des Solvers `fmincon` zur Lösung des nichtlinearen Optimierungsproblems. Dabei werden die Beschränkungen der Steuergröße (2.1e) als Beschränkungen für den Parametervektor

$$-u_{\text{minmax}}\mathbf{1} \leq \mathbf{w} \leq u_{\text{minmax}}\mathbf{1}$$

berücksichtigt. Die Ableitungen von Zielfunktion und Beschränkungen sollen durch finite Differenzen approximiert werden.

Die Berechnung der Zielfunktion erfolgt in der Funktion `obj()` ab Zeile 26. Dabei wird die um den Integralterm des Zielfunktionals erweiterte Zustandsgleichung aus (2.1) numerisch integriert (Aufruf `solve_ode()`). Der Endwert der 3. Zustandsgröße ist der Wert des Zielfunktionals `J`.

Die Auswertung der Beschränkungen erfolgt in der Funktion `con()` ab Zeile 30. Hier ist wieder die erweiterte Zustandsgleichung zu integrieren (Aufruf `solve_ode()`). Die Abweichungen der Endwerte der Zustandsgrößen von den Vorgaben `xf` (feste Endwerte $\mathbf{x}(1)$) werden als Gleichungsbeschränkungen `ceq` und die Verletzungen der Zustandsbeschränkungen $x_1(t) - x_{1,\max}$ bzw. $x_2(t) - x_{2,\max}$ in den Gitterpunkten als Ungleichungsbeschränkungen `c` betrachtet.

Die numerische Integration der Zustandsgleichungen erfolgt in der Funktion `solve_ode()` ab Zeile 41. Da dies sowohl zu Berechnung der Zielfunktion in `obj()` als auch zur Auswertung der Beschränkungen in `con()` notwendig ist, kann der Rechenaufwand reduziert werden, indem geprüft wird, ob sich die Werte der Optimierungsvariablen geändert haben, und gegebenenfalls auf die gespeicherten Ergebnisse des vorangegangenen Simulationslaufs zurück gegriffen wird (persistente Variable `w_obj`, `xJ_obj`). Die eigentliche numerische Integration erfolgt mit `ode5()`, einem RUNGE-KUTTA-Verfahren 5. Ordnung mit fester Schrittweite (Vorgabe Zeitschritte in der Variablen `ts`), siehe oben.

Die Zustandsgleichung ist in der Funktion `rhs()` implementiert. Hier erfolgt zunächst die Auswertung des parametrischen Ansatzes für $u(t)$ als Spline-Funktion, wobei durch persistente Variable und entsprechende Abfragen gesichert wird, dass die aufwendige Berechnung der Spline-Funktion mittels `spline()` nur dann erfolgt, wenn sich die Ansatzparameter geändert haben. Die Auswertung des Spline-Ansatzes erfolgt mit `ppval()` und in Zeile 59 sind die rechten Seiten der um den Integralterm des Zielfunktionals erweiterten Zustandsgleichungen implementiert.

`fmincon` löst das Optimalsteuerungsproblem für den zustandsbeschränkten Doppelintegrator nach Abschnitt 2.2 in 20 Iterationen mit 441 Zielfunktionsauswertungen, d. h. 441 Lösungen des Anfangswertproblems.

```
                                        Max      Line search  Directional  First-order
 Iter F-count            f(x)    constraint   steplength   derivative   optimality Procedure
    0      21               0             2                                        Infeasible start point
    1      42         3.89172     2.817e-07            1    -1.16e-09         28.7
    2      63         3.88456     1.977e-10            1      -0.0855       0.0414  Hessian modified
    3      84         3.80239     2.491e-09            1       -0.081       0.0318  Hessian modified
...
   18     399         3.70385     3.502e-10            1     -0.00167      0.00362  Hessian modified
   19     420         3.70384     1.128e-10            1    -0.000672      0.00338  Hessian modified
   20     441         3.70384     3.696e-12            1    -0.000203      0.00327  Hessian modified
```

Abbildung 3.6 zeigt die Lösung für $x_{1,\max} = 0.12$.

3.4.1.2 Zeitoptimale Umsteuerung eines PT_2-Glieds

Besonders einfach ist eine Steuerungsparametrisierung für die zeitoptimale Umsteuerung des PT_2-Glieds nach Abschnitt 2.3 zu realisieren, wenn man die Struktur der optimalen Lösung[1] ausnutzt: Die zeitoptimale Steuerungsaufgabe linearer Systeme führt auf bang-bang-Steuerungen, und nach dem Satz von FELDBAUM ergibt sich maximal $n = 1$ Umschaltpunkt. Aufgrund der vorgegebenen Anfangs- und Endzustände kann der Zeitverlauf der Steuerung mit 2 Parametern, der Umschaltzeit t_s und der Endzeit t_f, parametriert werden

$$u(t) = \begin{cases} u_{\text{minmax}}, & \text{falls } 0 \le t < t_s \\ -u_{\text{minmax}}, & \text{falls } t_s < t \le t_f \end{cases} \tag{3.17}$$

[1] Hierbei handelt es sich nicht um ein direktes Lösungsverfahren, da die Optimalitätsbedingungen ausgewertet werden.

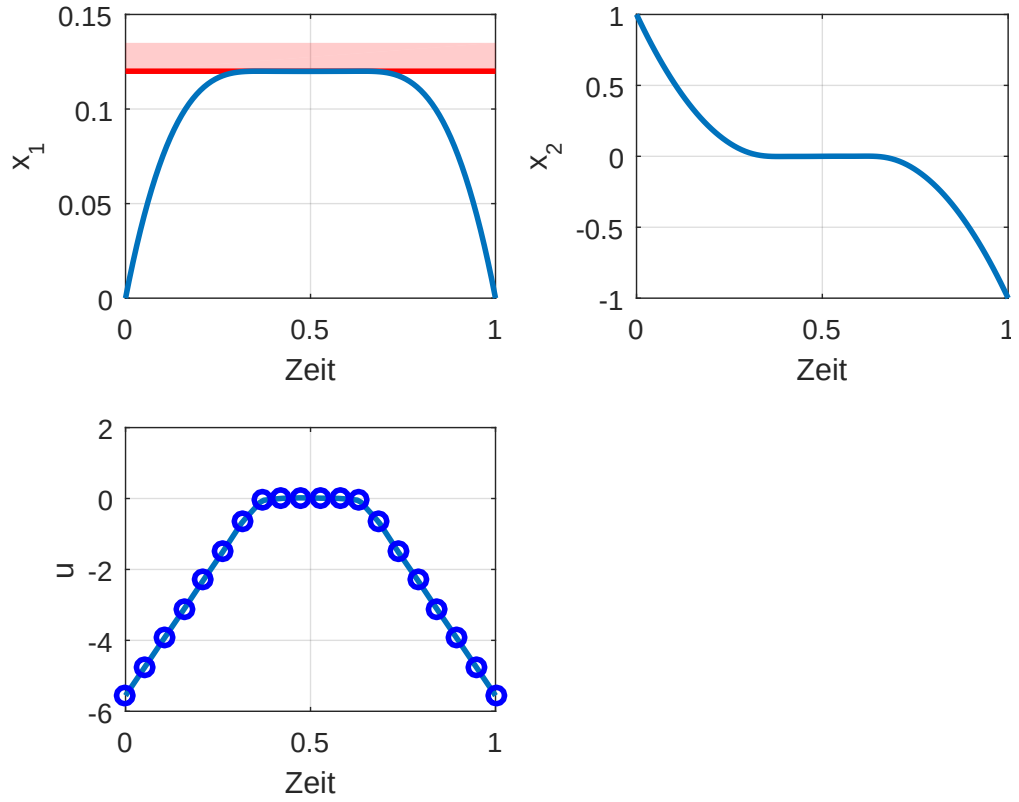


Abbildung 3.6: Doppelintegrator mit Zustandsbeschränkung: einfache Steuerungsparametrisierung.

Da das System linear ist, kann eine analytische Lösung der Zustandsdifferentialgleichung angegeben werden:

$$\begin{aligned} \mathbf{x}(t) &= e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau}\mathbf{b}u(\tau)d\tau \\ \mathbf{x}(t_f) &= e^{\mathbf{A}t_f}\mathbf{x}(0) + \mathbf{A}^{-1}\left(e^{\mathbf{A}t_f} - 2e^{\mathbf{A}(t_f-t_s)} + \mathbf{I}\right)\mathbf{b}u_{\text{minmax}} \end{aligned} \tag{3.18}$$

Die Bestimmung der Parameter t_s und t_f kann somit durch Lösung des nichtlinearen Gleichungssystems

$$\mathbf{x}(t_f) - \mathbf{x}_f = \mathbf{0} \tag{3.19}$$

erfolgen, Listing 3.18 zeigt den MATLAB-Code.

Listing 3.18: `t2topt_cpara.m`

```
A = [-0.5 1; 0 -1];
B = [0; 1];
% Residuum x(tf)-xf
res = @(x) expm(A*x(2))*x0+A\(expm(A*x(2))-2*expm(A*(x(2)-x(1)))+eye(2)) ...
    *B*uminmax-xf;
x = fsolve(res, [ts; tf], optimoptions('fsolve', 'Display', 'iter'));
ts = x(1);
tf = x(2);
```

3.5 Einfache Steuerungsparametrisierung mit CasADi

Die Implementierung der einfachen Steuerungsparametrisierung in MATLAB und CasADi basiert auf

> `direct_single_shooting.m`
> An implementation of direct single shooting. Joel Andersson, 2016.

aus der Beispielsammlung `casadi-example_pack-v3.4.5`.

3.5.1.1 Doppelintegrator mit Zustandsbeschränkung

Im Folgenden wird das Optimalsteuerungsproblem für den zustandsbeschränkten Doppelintegrator nach Abschnitt 2.2 betrachtet. Die Steuergröße wird als abschnittsweise konstante Zeitfunktion auf einem äquidistanten Zeitgitter $t^k = k\frac{t_f}{K}$, $k = 0, \dots K$ angesetzt.

Listing 3.19 zeigt die Implementierung in MATLAB und CasADi. In den Zeilen 2 und 3 werden der MATLAB-Suchpfad angepasst und das Package `casadi` wird importiert. In den Zeilen 5 bis 11 werden die Parameter des Optimalsteuerungsproblems definiert. `N` bezeichnet die Anzahl der Zeitstufen (K in (3.5)).

In den Zeilen 15 und 16 werden die symbolischen Variablen `x` und `u` für die Zustands- und Steuergrößen angelegt. `SX` steht für 'scalar expression', in CasADi eine Matrix (oder Vektor oder Skalar), deren Elemente symbolische Ausdrücke sind, die durch Folgen skalar-wertiger Rechenoperationen erzeugt werden. `xdot` (Zeile 18) und `L` (Zeile 20) sind die symbolischen Ausdrücke für die rechte Seite der Zustandsgleichung bzw. den Integranden des Zielfunktionals.

In den Zeilen 23–25 wird die Funktion `F` definiert, die die Zustandsdifferentialgleichung `xdot` und den Intergralterm des Zielfunktionals `L` über den Zeithorizont eines Abschnitts `T/N` mit dem Integrationsalgorithmus `CVODES` numerisch integriert.

Nun wird in den Zeilen 27–63 das nichtlineare Optimierungsproblem definiert. Der Vektor der Optimierungsvariablen `w` mit den Komponentenbeschränkungen `lbw`$\leq$`w`$\leq$`ubw` und den Initialwerten `w0` wird aus den konstanten Approximationen der Steuergröße in den Zeitstufen u^k (Zeile 40) zusammengesetzt. Die nichtlinearen Beschränkungen `lbg`$\leq$`g(w)`$\leq$`ubg` sind hier die Ungleichungsbeschränkung der Zustandsgröße (2.4e) in den Gitterpunkten des Zeitgitters (Zeilen 50–52) sowie der feste Endzustand (Zeilen 61–63).

Anschließend wird das Solver-Objekt `solver` mit dem zuvor definierten nichtlinearen Optimierungsproblem und dem Solver IPOPT in den Zeilen 65–66 definiert und in Zeile 68 aufgerufen und damit das nichtlineare Optimierungsproblem gelöst. Der Lösungsvektor `w_opt` entspricht den approximierten Steuergrößen `u_opt` (Zeile 71). Anschließend werden die optimalen Werte der Zustandsgrößen `x_opt` in den Gitterpunkten durch eine abschnittsweise Simulation mit den Eingangsgrößen `u_opt` berechnet (Zeilen 72–77).

Listing 3.19: `doint_single_shooting.m`

```
1 function doint_single_shooting(nr)
  casadi_path()
3 import casadi.*
  % Parameter Problem
```

```
5       x0 = [0; 1];
        xf = [0; -1];
7       uminmax = Inf;
        x1max = 0.12;
9       x2max = Inf;
    T = 1;                  % Endzeit
11  N = 50;                 % Anzahl Intervalle
    % Modellvariable fuer CasADi
13  x1 = SX.sym('x1');
    x2 = SX.sym('x2');
15  x = [x1; x2];
    u = SX.sym('u');
17  % Zustands-DGL
    xdot = [x2; u];
19  % Lagrange-Term Zielfunktional
    L = 0.5*u^2;
21  % Formulate discrete time dynamics
    % CVODES from the SUNDIALS suite
23  ode = struct('x', x, 'p', u, 'ode', xdot, 'quad', L);
    opts = struct('tf', T/N);
25  F = integrator('F', 'cvodes', ode, opts);
    % Start with an empty NLP
27  w = {};
    w0 = [];
29  lbw = [];
    ubw = [];
31  J = 0;
    g ={};
33  lbg = [];
    ubg = [];
35  % Formulate the NLP
    Xk = x0;
37  for k=0:N-1
        % New NLP variable for the control
39      Uk = MX.sym(['U_' num2str(k)]);
        w = [w(:)' {Uk}];
41      lbw = [lbw, -uminmax];
        ubw = [ubw,  uminmax];
43      w0 = [w0,  0];
        % Integrate till the end of the interval
45      Fk = F('x0', Xk, 'p', Uk);
        Xk = Fk.xf;
47      J = J+Fk.qf;
        % Add inequality constraint
49      if isfinite(x1max)
            g = [g(:)' {Xk(1)}];
51          lbg = [lbg; -Inf];
            ubg = [ubg;  x1max];
53      end
        if isfinite(x2max)
55          g = [g(:)' {Xk(2)}];
            lbg = [lbg; -Inf];
57          ubg = [ubg;  x2max];
```

```
    end
end
% Fixed final state
g = [g(:)' {Xk}];
lbg = [lbg; xf];
ubg = [ubg;  xf];
% Create an NLP solver
prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
solver = nlpsol('solver', 'ipopt', prob);
% Solve the NLP
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw, 'lbg', lbg, 'ubg', ubg);
w_opt = full(sol.x);
% Simulate the system
u_opt = w_opt;
x_opt = x0;
for k=0:N-1
    Fk = F('x0', x_opt(:, end), 'p', u_opt(k+1));
    x_opt = [x_opt, full(Fk.xf)];
end
x_opt = x_opt';
% Zeit
t = linspace(0, T, N+1)';
```

Ipopt löst das Optimalsteuerungsproblem für den zustandsbeschränkten Doppelintegrator nach Abschnitt 2.2 in 15 Iterationen mit 16 Zielfunktionsauswertungen, d. h. 16 Lösungen des Anfangswertproblems.

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 0.0000000e+000 2.00e+000 7.83e-002  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 8.4124614e+000 9.27e-007 1.93e+000  -1.0 1.07e+001    -  4.95e-001 1.00e+000f  1
   2 6.3571082e+000 8.46e-007 1.00e-006  -1.0 2.19e+000    -  1.00e+000 1.00e+000f  1
...
  13 3.7065678e+000 8.21e-008 3.20e-012  -8.6 8.25e-004    -  1.00e+000 1.00e+000h  1
  14 3.7065678e+000 5.80e-008 3.84e-013  -8.6 1.86e-004    -  1.00e+000 1.00e+000h  1
  15 3.7065678e+000 2.85e-009 1.05e-013  -8.6 7.12e-006    -  1.00e+000 1.00e+000h  1
```

Im Vergleich zur Lösung mit `fmincon` ohne `CasADi` wird nur ca. 10 % der Rechenzeit benötigt, dies ist im Wesentlichen auf die deutlich effizientere Ableitungsberechnung zurückzuführen.

4 Direkte Kollokation

Bei direkten Kollokationsverfahren wird die Optimalsteuerungsaufgabe (3.1) durch ein nichtlineares Optimierungsproblem approximiert. Die Auswertung von Optimalitätsbedingungen ist nicht erforderlich.

Wie im Zusammenhang mit den Kollokationsverfahren zur Lösung der Randwertaufgabe in Abschnitt 6 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 Zeitverläufe 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

$$t_0 = t^0 < t^1 < t^2 < \ldots < t^K = t_f \tag{4.1}$$

sind[1]. In Abhängigkeit von den gewählten Polynomansätzen und den Kollokationspunkten erhält man die Kollokationsbedingungen, siehe z. B. [19], [6].

Implizite Mittelpunktregel

Bei Approximation der Steuergrößen durch konstante Werte in den Teilabschnitten und der Zustandsgrößen durch lineare Interpolation („DP1“ in [19])

$$\mathbf{u}_{app}(t) = \mathbf{u}^k, \qquad t^k \le t < t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.2a}$$

$$\mathbf{x}_{app}(t) = \mathbf{x}^k + \frac{t - t^k}{t^{k+1} - t^k}(\mathbf{x}^{k+1} - \mathbf{x}^k), \quad t^k \le t \le t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.2b}$$

sowie Wahl der Intervallmittelpunkte $t^{k+1/2} = (t^k + t^{k+1})/2$ als Kollokationspunkte, siehe Abbildung 4.1, erhält man

$$\frac{1}{t^{k+1} - t^k}(\mathbf{x}^{k+1} - \mathbf{x}^k) = \mathbf{f}\left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2}\right), \quad k = 0, \ldots, K-1 \tag{4.3}$$

Dies entspricht der impliziten Mittelpunktregel zur numerischen Integration gewöhnlicher Differentialgleichungen.

[1] Es besteht ein grundsätzlicher Unterschied zur Zeitstufenstruktur der Mehrstufen-Steuerungsparametrisierung nach (3.5): bei Kollokationsverfahren stimmen die Teilabschnitte $[t^k, t^{k+1}]$ mit den Diskretisierungsschritten der Differentialgleichung überein.

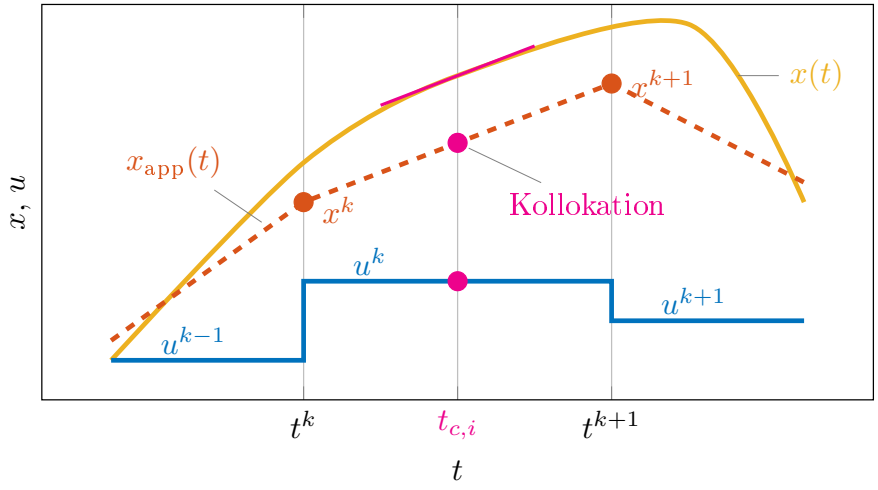


Abbildung 4.1: Kollokation.

Das Zielfunktional (3.1e) und die Ungleichungsbeschränkungen (3.1c) werden ebenfalls diskretisiert

$$J_{app} = F(\mathbf{x}^K, t^K) + \sum_{k=0}^{K-1} \left((t^{k+1} - t^k) f_0 \left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2} \right) \right) \tag{4.4a}$$

$$\mathbf{g}\left(\frac{1}{2}(\mathbf{x}^k + \mathbf{x}^{k+1}), \mathbf{u}^k, t^{k+1/2} \right) \leq \mathbf{0}, \quad k = 0, \ldots, K-1 \tag{4.4b}$$

und die Randbedingungen als zusätzliche Gleichungsbeschränkungen in das Optimierungsproblem aufgenommen.

Trapezregel

Bei linearer Interpolation der Steuergrößen und Approximation der Zustandsgrößen durch quadratische Polynome in den Teilabschnitten

$$\mathbf{u}_{app}(t) = \mathbf{u}^k + \frac{t - t^k}{t^{k+1} - t^k}(\mathbf{u}^{k+1} - \mathbf{u}^k), \quad t^k \leq t \leq t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.5a}$$

$$\mathbf{x}_{app}(t) = \mathbf{x}^k + \boldsymbol{\alpha}(t - t^k) + \boldsymbol{\beta}(t - t^k)^2 \quad t^k \leq t \leq t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.5b}$$

sowie Wahl der Gitterpunkte t^k als Kollokationspunkte erhält man nach Elimination von $\boldsymbol{\alpha}$ und $\boldsymbol{\beta}$

$$\frac{1}{t^{k+1} - t^k}(\mathbf{x}^{k+1} - \mathbf{x}^k) = \frac{1}{2}\left(\mathbf{f}\left(\mathbf{x}^k, \mathbf{u}^k, t^k \right) + \mathbf{f}\left(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}, t^{k+1} \right) \right), \quad k = 0, \ldots, K-1 \tag{4.6}$$

Dies entspricht der impliziten Trapezregel zur numerischen Integration gewöhnlicher Differentialgleichungen.

Das Zielfunktional (3.1e) und die Ungleichungsbeschränkungen (3.1c) werden ebenfalls diskretisiert

$$J_{app} = F(\mathbf{x}^K, t^K) + \sum_{k=0}^{K-1} \left((t^{k+1} - t^k) \frac{1}{2} \left(f_0\left(\mathbf{x}^k, \mathbf{u}^k, t^k\right) + f_0\left(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}, t^{k+1}\right) \right) \right) \tag{4.7a}$$

$$\mathbf{g}\left(\mathbf{x}^k, \mathbf{u}^k, t^k\right) \leq \mathbf{0}, \quad k = 0, \ldots, K \tag{4.7b}$$

und die Randbedingungen als zusätzliche Gleichungsbeschränkungen in das Optimierungsproblem aufgenommen.

Kubische Ansatzfunktion

Bei linearer Interpolation der Steuergrößen und Approximation der Zustandsgrößen durch kubische Polynome in den Teilabschnitten („DP2“ in [19])

$$\mathbf{u}_{app}(t) = \mathbf{u}^k + \frac{t - t^k}{t^{k+1} - t^k}(\mathbf{u}^{k+1} - \mathbf{u}^k), \qquad t^k \leq t \leq t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.8a}$$

$$\mathbf{x}_{app}(t) = \mathbf{x}^k + \boldsymbol{\alpha}(t - t^k) + \boldsymbol{\beta}(t - t^k)^2 + \boldsymbol{\gamma}(t - t^k)^3 \quad t^k \leq t \leq t^{k+1}, \quad k = 0, \ldots, K-1 \tag{4.8b}$$

sowie Wahl der Gitterpunkte t^k und der Intervallmittelpunkte $t^{k+1/2} = (t^k + t^{k+1})/2$ als Kollokationspunkte

$$\begin{aligned} \dot{\mathbf{x}}_{app}(t^k) &= \mathbf{f}\left(\mathbf{x}_{app}(t^k), \mathbf{u}_{app}(t^k), t^k\right) = \mathbf{f}^k, \quad k = 0, \ldots, K \\ \dot{\mathbf{x}}_{app}(t^{k+1/2}) &= \mathbf{f}\left(\mathbf{x}_{app}(t^{k+1/2}), \mathbf{u}_{app}(t^{k+1/2}), t^{k+1/2}\right), \quad k = 0, \ldots, K-1 \end{aligned}$$

erhält man nach Elimination von $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ und $\boldsymbol{\gamma}$

$$\mathbf{x}_{app}(t^{k+1/2}) = \frac{1}{2}\left(\mathbf{x}^k + \mathbf{x}^{k+1}\right) + \frac{t^{k+1} - t^k}{8}\left(\mathbf{f}^k - \mathbf{f}^{k+1}\right), \quad k = 0, \ldots, K-1 \tag{4.9}$$

die Kollokationsbedingung

$$\begin{aligned} \frac{3}{2(t^{k+1} - t^k)}\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right) - \frac{1}{4}\left(\mathbf{f}^k + \mathbf{f}^{k+1}\right) \\ = \mathbf{f}\left(\mathbf{x}_{app}(t^{k+1/2}), \frac{1}{2}\left(\mathbf{u}^k + \mathbf{u}^{k+1}\right), t^{k+1/2}\right), \quad k = 0, \ldots, K-1 \end{aligned} \tag{4.10}$$

Das Zielfunktional (3.1e) und die Ungleichungsbeschränkungen (3.1c) werden ebenfalls diskretisiert

$$\begin{aligned} J_{app} = F(\mathbf{x}^K, t^K) + \sum_{k=0}^{K-1} \frac{2(t^{k+1} - t^k)}{3} \Bigg(\frac{1}{4} f_0\left(\mathbf{x}^k, \mathbf{u}^k, t^k\right) + \\ + f_0\left(\mathbf{x}_{app}(t^{k+1/2}), \frac{1}{2}(\mathbf{u}^k + \mathbf{u}^{k+1}), t^{k+1/2}\right) + \frac{1}{4} f_0\left(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}, t^{k+1}\right) \Bigg) \end{aligned} \tag{4.11a}$$

$$\mathbf{g}\left(\mathbf{x}^k, \mathbf{u}^k, t^k\right) \leq \mathbf{0}, \quad k = 0, \ldots, K \tag{4.11b}$$

$$\mathbf{g}\left(\mathbf{x}_{app}(t^{k+1/2}), \frac{1}{2}(\mathbf{u}^k + \mathbf{u}^{k+1}), t^{k+1/2}\right) \leq \mathbf{0}, \quad k = 0, \ldots, K-1 \tag{4.11c}$$

und die Randbedingungen als zusätzliche Gleichungsbeschränkungen in das Optimierungsproblem aufgenommen.

Der Fehler[2] sowohl der impliziten Mittelpunktregel als auch der Trapezregel ist $\mathcal{O}\left(\max_k (t^{k+1} - t^k)^2\right)$, die der kubischen Ansatzfunktion hingegen $\mathcal{O}\left(\max_k (t^{k+1} - t^k)^4\right)$.

Das resultierende nichtlineare Optimierungsproblem ist hochdimensional und „sparse“, d. h. sowohl die JACOBI-Matrizen der Gleichungs- und Ungleichungsbeschränkungen als auch die HESSE-Matrix der zugehörigen LAGRANGE-Funktion sind schwach besetzt.

4.1 Direkte Kollokation mit Matlab

4.1.1 Direkte Kollokation mit SNOPT und Matlab

Ein besonders geeigneter Solver für derartige Optimierungsprobleme ist SNOPT, siehe [11], der als eingeschränkte „Studentenlizenz“ unter `http://scicomp.ucsd.edu/~peg/` kostenlos verfügbar ist.

SNOPT verfügt über ein `MEX`-Interface[3], kann also direkt aus MATLAB heraus aufgerufen werden. Das Optimierungsproblem ist in der Form

$$\min_{\mathbf{y}\in\mathbb{R}^{n_y}} \left\{ h_1(\mathbf{y}) \,|\, \mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max},\, \mathbf{h}_{\min} \leq \mathbf{h}(\mathbf{y}) \leq \mathbf{h}_{\max},\, \mathbf{h} : \mathbb{R}^{n_y} \to \mathbb{R}^{n_h} \right\} \tag{4.12}$$

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

4.1.1.1 Doppelintegrator mit singulärem Lösungsabschnitt

Das folgende Listing 4.1 zeigt die Lösung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 mittels direkter Kollokation.

In den Zeilen 9–28 wird der Variablenvektor $\mathbf{y}$ des nichtlinearen Optimierungsproblems mit den Komponentenbeschränkungen initialisiert

$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ -u_{minmax} \\ -\infty \\ \vdots \\ -u_{minmax} \\ x_{f,1} \\ x_{f,2} \\ 1.0 \end{bmatrix} \leq \begin{bmatrix} x_1^0 \\ x_2^0 \\ u^0 \\ x_1^1 \\ \vdots \\ u^{K-1} \\ x_1^K \\ x_2^K \\ t_f \end{bmatrix} \leq \begin{bmatrix} x_{0,1} \\ x_{0,2} \\ u_{minmax} \\ \infty \\ \vdots \\ u_{minmax} \\ x_{f,1} \\ x_{f,2} \\ \infty \end{bmatrix} \tag{4.13}$$

[2] Hier wird der Fehler in der Approximation der Differentialgleichung („Steigungsfehler“) betrachtet, in [19] hingegen der um eine Ordnung geringere Approximationsfehler.
[3] Hier wird SNOPT Version 7 verwendet.

Die Randbedingungen für die Zustandsgrößen werden als Gleichungsbeschränkungen für die entsprechenden Komponenten des Vektors **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

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leq \begin{bmatrix} J_{app} \\ x_1^1 - x1^0 - t_f/K \cdot (x_2^0 + x_2^1)/2 \\ \vdots \\ x_2^K - x_2^K - 1 - t_f/K \cdot u^{K-1} \end{bmatrix} \leq \begin{bmatrix} \infty \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.14}$$

erfolgt in der Funktion `dising_dto_dtc`, Listing 4.2. Für die Zielfunktion wird als untere Schranke 0 angenommen. Die Gleichheit von oberer und unterer Schranke für die Kollokationsbedingungen erzwingt eine Behandlung als Gleichungsbeschränkungen.

In Zeile 31 erfolgt der Aufruf von `snopt`. Übergabeparameter sind die Anfangsnäherung für die Optimierungsvariablen `y0`, die Schranken für die Optimierungsvariablen `ymin` und `ymax` aus (4.13) sowie die Schranken für die Gleichungs- und Ungleichungsrestriktionen `Jhmin` und `Jhmax` aus (4.14). Weiterhin wird der Name der Funktion zur Berechnung von Zielfunktionswert und Beschränkungen (als Zeichenkette `'dising_dto_dtc'`) übergeben.

Listing 4.1: `dising_dto.m`

```
function dising_dto
global rho_tf meth
K = 98;                              % Anzahl Intervalle
uminmax = 1;  % Steuerungsbeschraenkung
rho_tf = 0.1;    % Bewertung Endzeit
x0 = [-3; 0]; % Anfangswerte
xf = [0; 0];  % Endwerte
tf = 5;          % Startnaeherung Endzeit
    y0 = zeros((K+1)*2+K*1+1, 1); % Variablenvektor:
                                    % [x(0); u(0); ...; u(K-1); x(K); tf]
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, zeros(size(y0)), zeros(size(y0)),
Jhmin, Jhmax, zeros(size(Jhmin)), zeros(size(Jhmin)), @dising_dto_dtc);
% Auswertung Loesung
fprintf('Optimale Endzeit: %g\nOptimaler Wert Zielfunktional: %g\n', ...
    y(end), Jh(1));
```

Listing 4.2: `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 Mittelpunktregel
% Zielfunktional
J = rho_tf*tf + 0.5*sum(xk12(:, 1).^2+xk12(:, 2).^2)*dt*tf;
% Beschraenkungen: Approximation Zustands-Dgl.
% mit impliziter Mittelpunktregel 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 67 Iterationen findet SNOPT eine Lösung des nichtlinearen Optimierungsproblems, die zugehörigen Zeitverläufe zeigt Abbildung 4.2.

```
 Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
     0    131                1  5.0E-01  4.1E-02  5.5739796E-01     79          r
     1     12  3.3E-01       2  3.4E-01  5.6E-02  2.0050276E+02     88 5.9E+01   rl
     2     36  2.9E-01       4  2.1E-01  3.0E-02  2.2676069E+02     93 1.0E+02 sM l
     3     10  1.0E+00       5  7.9E-03  1.0E-01  1.3255742E+01     96 1.0E+01
...
    64      1  1.0E+00      74 (1.4E-09) 8.8E-06  8.0059727E+00     25 4.5E+00       c
    65      1  1.0E+00      75 (2.1E-09) 8.4E-06  8.0059727E+00     25 4.5E+00       c
    66      1  1.0E+00      76 (1.1E-09) 4.4E-06  8.0059727E+00     25 4.5E+00       c
    67      1  1.0E+00      77 (2.5E-09)(5.3E-07) 8.0059727E+00     25 4.5E+00   R   c
Optimale Endzeit: 3.88163
Optimaler Wert Zielfunktional: 8.00597
```

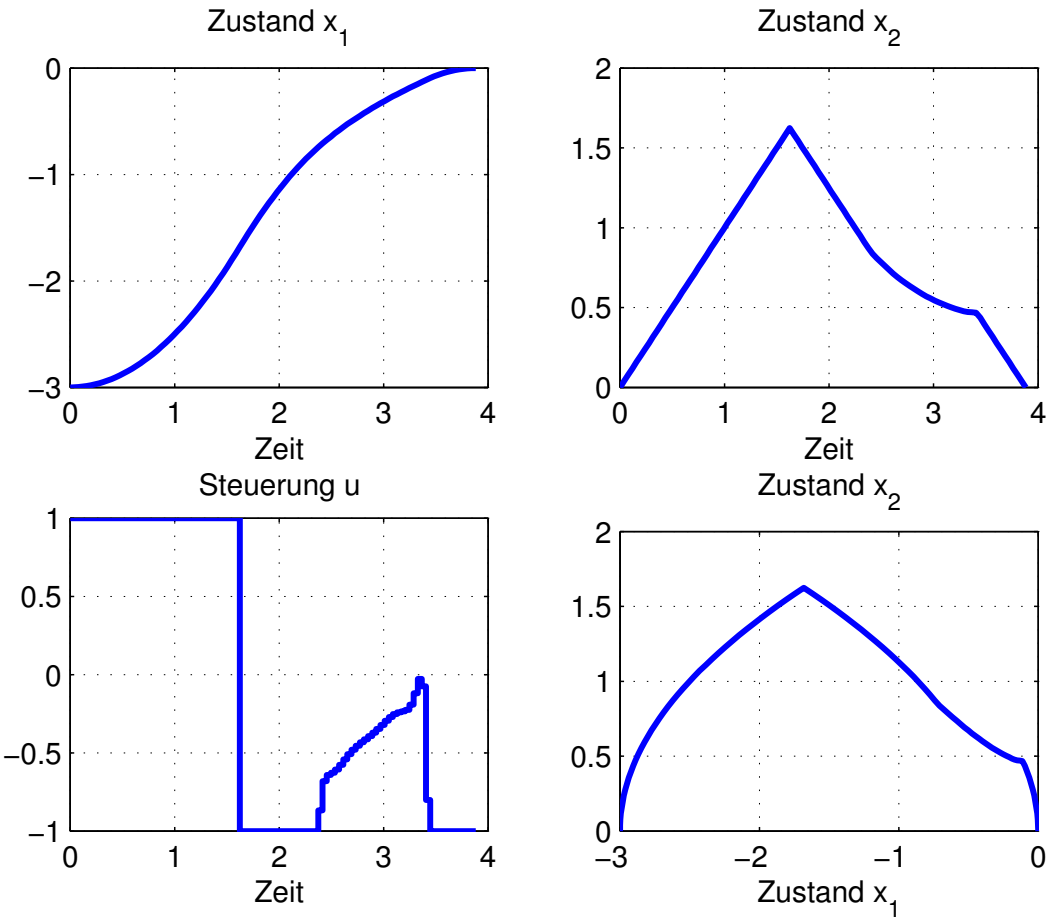


Abbildung 4.2: Doppelintegrator mit singulärem Lösungsabschnitt; direkte Kollokation.

4.1.2 Direkte Kollokation mit Ipopt und Matlab

IPOPT [14], [20] ist ein frei verfügbarer Solver für große nichtlineare Optimierungsprobleme und verfügt u. a. über C++-, AMPL- und MATLAB-Schnittstellen (MEX-Interface). Binärversionen können beispielsweise von `https://de.mathworks.com/matlabcentral/fileexchange/53040-ebertolazzi-mexipopt` heruntergeladen werden. Der Vorteil gegenüber dem in Abschnitt 4.1.1 beschriebenen Einsatz von SNOPT besteht darin, dass es keine lizenzbedingten Begrenzungen der Problemdimension gibt.

Das Optimierungsproblem ist in der Form

$$\min_{\mathbf{y}\in\mathbb{R}^{n_y}} \left\{ J(\mathbf{y}) \,\middle|\, \mathbf{y}_{\min} \le \mathbf{y} \le \mathbf{y}_{\max},\, \mathbf{h}_{\min} \le \mathbf{h}(\mathbf{y}) \le \mathbf{h}_{\max},\, \mathbf{h} : \mathbb{R}^{n_y} \to \mathbb{R}^{n_h} \right\} \tag{4.15}$$

anzugeben. Das MEX-Interface stellt eine MATLAB-Funktion `ipopt()` bereit, der beim Aufruf neben den Startwerten der Optimierungsvariablen Funktionen (function handles) zu übergeben sind, die die Zielfunktion J und die Beschränkungen $\mathbf{h}(\mathbf{y})$ sowie die zugehörigen 1. Ableitungen berechnen. IPOPT ermöglicht die Überprüfung der Ableitungen durch Vergleich mit einer finiten

Differenzenapproximation (Option `derivative_test='first-order'`), stellt jedoch keine solche Approximation für den Anwender bereit.

Die entsprechenden Funktionen für einen direkten Kollokationsansatz mittels impliziter Mittelpunkt- oder Trapezregel für ein Optimalsteuerungsproblem mit MAYER-Funktional, fester Anfangs- und Endzeit t_0 bzw. t_f und Komponentenbeschränkungen der Steuer- und Zustandsgrößen

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad t \in (t_0, t_f) \tag{4.16a}$$
$$J = F(\mathbf{x}(t_f)) \tag{4.16b}$$
$$\mathbf{x}_{\min}(t) \leq \mathbf{x}(t) \leq \mathbf{x}_{\max}(t), \quad t \in [t_0, t_f] \tag{4.16c}$$
$$\mathbf{u}_{\min}(t) \leq \mathbf{u}(t) \leq \mathbf{u}_{\max}(t), \quad t \in [t_0, t_f] \tag{4.16d}$$

sind in `dto_ipopt.m` vorbereitet. Anfangs- und Endbedingungen der Zustandsgrößen können mit geeigneten Werten für $\mathbf{x}_{\min}(t_0)$, $\mathbf{x}_{\max}(t_0)$ bzw. $\mathbf{x}_{\min}(t_f)$, $\mathbf{x}_{\max}(t_f)$ vorgegeben werden. Integralterme im Zielfunktional können ggf. durch eine zusätzliche Zustandsvariable in MAYER-Form überführt werden. Aufgaben mit freier Endzeit werden durch eine geeignete Transformation in Aufgaben mit fester Endzeit überführt.

4.1.2.1 Doppelintegrator mit singulärem Lösungsabschnitt

Für die Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 ergibt sich

$$\dot{x}_1 = x_2 x_4 \tag{4.17a}$$
$$\dot{x}_2 = u x_4 \tag{4.17b}$$
$$\dot{x}_3 = 0.5\left(x_1^2 + x_2^2\right) x_4 \qquad t \in (0,1) \tag{4.17c}$$
$$\dot{x}_4 = 0 \tag{4.17d}$$
$$J = \rho_{t_f} x_4(1) + x_3(1) \tag{4.17e}$$
$$\mathbf{x}_{\min}(0) = \begin{bmatrix} -3 \\ 0 \\ 0 \\ t_{f,\min} \end{bmatrix}, \qquad \mathbf{x}_{\max}(0) = \begin{bmatrix} -3 \\ 0 \\ 0 \\ +\infty \end{bmatrix} \tag{4.17f}$$
$$\mathbf{x}_{\min}(t) = \begin{bmatrix} -\infty \\ -\infty \\ -\infty \\ t_{f,\min} \end{bmatrix}, \qquad \mathbf{x}_{\max}(t) = \begin{bmatrix} +\infty \\ +\infty \\ +\infty \\ +\infty \end{bmatrix}, \qquad t \in (0,1) \tag{4.17g}$$
$$\mathbf{x}_{\min}(1) = \begin{bmatrix} 0 \\ 0 \\ -\infty \\ t_{f,\min} \end{bmatrix}, \qquad \mathbf{x}_{\max}(1) = \begin{bmatrix} 0 \\ 0 \\ +\infty \\ +\infty \end{bmatrix} \tag{4.17h}$$
$$-1 \leq u(t) \leq 1, \qquad t \in [0,1] \tag{4.17i}$$

Die Zeit ist normiert auf $t \in [0,1]$, und die (zeitlich konstante) Zustandsgröße x_4 mit freiem Anfangswert entspricht der freien Endzeit t_f der Ursprungsaufgabe. Die untere Schranke $t_{f,\min} > 0$ wird aus numerischen Gründen eingeführt.

In Listing 4.3 werden zunächst die Problemdaten aufbereitet (Zeilen 2–6), dann in Zeile 7 das Diskretisierungsgitter `ts` für den Kollokationsansatz definiert und weitere Optionen festgelegt. In Zeile 16/17 wird die Funktion `dto_ipopt()` aufgerufen, die die Aufbereitung des Kollokationsansatzes und den Aufruf von `ipopt()` übernimmt. An `dto_ipopt()` werden die Funktionen `xu_minmax()` (Festlegung Komponentenbeschränkungen, Anfangs- und Endbedingungen, Startwerte), `rhs()` (Auswertung Zustandsgleichung $\mathbf{f}$ (4.17a)–(4.17d) und JACOBI-Matrizen $\mathbf{f_x}$, $\mathbf{f_u}$) und `mayer()` (Auswertung MAYER-Term F (4.17e) und Ableitung $F_{\mathbf{x}(t_f)}$), sowie das Diskretisierungsgitter `ts`, der Schalter für den Kollokationsansatz `meth` und Optionen `options` für den Aufruf von `ipopt()` übergeben.

Listing 4.3: `dising_dto_ipopt.m`

```
1  function dising_dto_ipopt()
   % Parameter Problem
3  x0 = [-3; 0];     % Anfangswerte
   xf = [0; 0];      % Endwerte
5  uminmax = 1;      % Steuerungsbeschraenkung
   rho_tf = 0.1;     % Bewertung Endzeit
7  ts = linspace(0, 1, 200); % Diskretisierungsgitter
   meth = 'imp';             % Kollokationsansatz: 'imp', 'itrapz'
9  % Optionen IPOPT
   options.print_level = 5;  % Ausgabe Iterationen
11 options.tol = 1e-6;       % Genauigkeit
   options.max_iter = 200;   % max. Iterationszahl
13 options.mehrotra_algorithm = 'yes';
   %options.derivative_test = 'first-order'; % Test Ableitungen
15 % Aufruf Solver
   [xup, J] = dto_ipopt(@(t) xu_minmax(t, x0, xf, uminmax), @rhs, ...
17     @(xtf) mayer(xtf, rho_tf), ts, meth, options);
   %-------------------------------------------------------------------
19 % xuminmax - Komponentenbeschränkungen,
   %             Anfangs- und Endbedingungen, Startwerte
21 function [xmin, xmax, umin, umax, xinit, uinit] = xu_minmax(t, x0, xf, uminmax
   )
   if t == 0
23     xmin = [x0; 0; 0.1];
       xmax = [x0; 0; Inf];
25 elseif t == 1
       xmin = [xf; -Inf; 0.1];
27     xmax = [xf; Inf; Inf];
   else
29     xmin = [-Inf; -Inf; -Inf; 0.1];
       xmax = Inf*ones(4, 1);
31 end
   umin = -uminmax;
33 umax = uminmax;
   xinit = [0; 0; 0; 5];
35 uinit = 0;
   %-------------------------------------------------------------------
```

```
37 % rhs - Zustands-DGL und Jacobi-Matrizen
   function [f, fx, fu] = rhs(x, u, t)
39 f = [x(2); u; 0.5*(x(1)^2+x(2)^2); 0]*x(4);
   if nargout > 1
41     fx = [0 x(4) 0 x(2); 0 0 0 u; ...
           x(1)*x(4) x(2)*x(4) 0 0.5*(x(1)^2+x(2)^2); 0 0 0 0];
43     fu = [0; x(4); 0; 0];
   end
45 %-------------------------------------------------------------------------
   % mayer - Mayer-Funktional und Ableitung
47 function [F, Fx] = mayer(x, rho_tf)
   F = rho_tf*x(4)+x(3);
49 if nargout > 1
       Fx = [0; 0; 1; rho_tf];
51 end
```

In `dto_ipopt()` (siehe M-File `cocp_ex/dto_ipopt/dto_ipopt.m`) werden in der Funktion `objective()` der Zielfunktionswert, in `gradient()` die 1. Ableitung der Zielfunktion, in `constraints()` die aus dem Kollokationsansatz resultierenden Gleichungsbeschränkungen und in `jacobian()` die 1. Ableitungen der Gleichungsbeschränkungen berechnet.

Ergebnisparameter von `dto_ipopt()` sind die Werte der Zustands- und Steuergrößen sowie Approximationen der Kozustandsgrößen auf dem Diskretisierungsgitter `ts` in der Matrix `xup` sowie der Optimalwert des Zielfunktionals `J`.

Mit einem Diskretisierungsgitter mit 200 Gitterpunkten und damit $\dim(\mathbf{y}) = 999$ Optimierungsvariablen benötigt IPOPT 69 Iterationen zur Lösung des Problems.

```
This is Ipopt version 3.11.8, running with linear solver ma57.

Number of nonzeros in equality constraint Jacobian...:     7144
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:      994
                     variables with only lower bounds:      200
                variables with lower and upper bounds:      199
                     variables with only upper bounds:        0
Total number of equality constraints.................:      796
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 1.0947788e+000 2.28e-001 1.00e+001   0.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 1.2065838e+000 1.65e-001 7.75e+001   0.8 6.15e+000    -  9.16e-001 2.76e-001h  1
   2 6.5553841e+000 1.07e-001 1.55e+001  -1.1 6.25e+000    -  9.39e-001 1.00e+000h  1
...
```

```
  67 8.0065841e+000 3.00e-008 2.56e-005 -11.0 1.49e-002    -  1.00e+000 1.00e+000h  1
  68 8.0065841e+000 8.52e-009 1.02e-006 -11.0 6.96e-003    -  1.00e+000 1.00e+000h  1
  69 8.0065843e+000 2.31e-010 4.30e-007 -11.0 4.31e-003    -  1.00e+000 1.00e+000h  1

Number of Iterations....: 69

                                   (scaled)                 (unscaled)
Objective...............:  8.0065842511426890e+000   8.0065842511426890e+000
Dual infeasibility......:  4.2994487527567412e-007   4.2994487527567412e-007
Constraint violation....:  2.3125856785100041e-010   2.3125856785100041e-010
Complementarity.........:  7.5811879535335873e-011   7.5811879535335873e-011
Overall NLP error.......:  4.2994487527567412e-007   4.2994487527567412e-007


Number of objective function evaluations             = 70
Number of objective gradient evaluations             = 70
Number of equality constraint evaluations            = 71
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 71
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      0.273
Total CPU secs in NLP function evaluations           =      1.930

EXIT: Optimal Solution Found.
Optimalwert Zielfunktional: 8.00658, CPU-Zeit: 2.203s, Iterationen: 69
```

Die Ergebnisse stimmen mit den Ergebnissen aus Abschnitt 4.1.1 überein.

4.1.3 Direkte Kollokation mit `fmincon`

Das aus dem Kollokationsansatz resultierende nichtlineare Optimierungsproblem kann auch mit dem Solver `fmincon` aus der MATLAB OPTIMIZATION TOOLBOX gelöst werden. Der Vorteil gegenüber dem in Abschnitt 4.1.1 beschriebenen Einsatz von SNOPT besteht darin, dass es keine lizenzbedingten Begrenzungen der Problemdimension gibt. Andererseits ist SNOPT für die betrachteten Probleme deutlich effizienter und robuster als `fmincon`.

4.1.3.1 Doppelintegrator mit singulärem Lösungsabschnitt

Das Vorgehen soll wieder für Lösung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 gezeigt werden. Dabei entspricht die Implementierung weitgehend der mit SNOPT (siehe 4.1 und 4.2). Entsprechend den Anforderungen von `fmincon` ist die Berechnung von Zielfunktionswert und Beschränkungen auf die Funktionen `obj` (Listing 4.5) und `con` (Listing 4.6) aufgeteilt.

In Listing 4.4 werden zunächst die Problemdaten aufbereitet und in Zeile 41 wird der Solver `fmincon` aufgerufen.

Listing 4.4: `dising_dto_fmincon.m`

```
function dising_dto_fmincon
global rho_tf meth
K = 98;          % Anzahl Intervalle
uminmax = 1;     % Steuerungsbeschraenkung
rho_tf = 0.1;    % Bewertung Endzeit
x0 = [-3; 0];    % Anfangswerte
xf = [0; 0];     % Endwerte
tf = 5;          % Startnaeherung Endzeit
meth = 'imp';    % Diskretisierungsmethode: imp - implizite Mittelpunktsregel
                 %                          trapz - Trapezregel
if strcmp(meth, 'imp')
    y0 = zeros((K+1)*2+K*1+1, 1); % Variablenvektor:
                                  % [x(0); u(0); ...; u(K-1); x(K); tf]
else
    y0 = zeros((K+1)*3+1, 1);     % Variablenvektor:
                                  % [x(0); u(0); ...; x(K); u(K); tf]
end
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);
if strcmp(meth, 'imp')
    idx1 = length(y0)-2;
else
    idx1 = length(y0)-3;
end
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
% Aufruf fmincon
options = optimoptions('fmincon', 'Display', 'iter', ...
    'Algorithm', 'active-set', 'MaxFunEvals', 200000);
tic
[y, J] = fmincon(@obj, y0, [], [], [], [], ymin, ymax, @con, options);
% Auswertung Loesung
fprintf('Optimale Endzeit: %g\nOptimaler Wert Zielfunktional: %g\nCPU-Zeit:
%.3fs\n', ...
    y(end), J, toc);
```

Listing 4.5: Funktion `obj` zur Auswertung der Zielfunktion aus `dising_dto_fmincon.m`

```
function J = obj(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;
if strcmp(meth, 'imp')
    uk12 = uk;                    % implizite Mittelpunktregel
else
    uk12 = (uk(1:end-1, :)+uk(2:end, :))/2; % Trapezregel
end
% Zielfunktional
J = rho_tf*tf + 0.5*sum(xk12(:, 1).^2+xk12(:, 2).^2)*dt*tf;
```

Listing 4.6: Funktion `con` zur Auswertung der Beschränkungen aus `dising_dto_fmincon.m`

```
function [c, ceq] = con(y)
global 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;
if strcmp(meth, 'imp')
    uk12 = uk;                    % implizite Mittelpunktregel
else
    uk12 = (uk(1:end-1, :)+uk(2:end, :))/2; % Trapezregel
end
% Beschraenkungen: Approximation Zustands-Dgl.
% mit impliziter Mittelpunktregel 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'];
ceq = h(:);
c = [];
```

`fmincon` mit der Einstellung `algorithm=active-set` benötigt 90 Iterationen zur Lösung des nichtlinearen Optimierungsproblems. Mit `algorithm=interior-point` und `Hessian=lbfgs` werden 370 Iterationen benötigt und mit `algorithm=sqp` wird keine Lösung gefunden.

```
                                  Max     Line search  Directional  First-order
 Iter F-count        f(x)   constraint   steplength   derivative   optimality Procedure
    0     298    0.557398            3                                       Infeasible start po
    1     596     8.05694      0.00132            1      0.00565          237
    2     894     8.21457   1.528e-005            1        0.409         6.64
    3    1192     8.20654   1.389e-006            1      -0.0908         1.59
...
```

```
   87   26224      8.00614   7.845e-010          1    -0.000629       0.184  Hessian modified
   88   26522      8.00613   6.673e-009          1    -0.000305       0.182  Hessian modified
   89   26820      8.00613   5.149e-010          1    -0.000341       0.182  Hessian modified
   90   27118       8.0061    2.16e-007          1    -0.000341       0.182  Hessian modified

Local minimum possible. Constraints satisfied.

fmincon stopped because the predicted change in the objective function
is less than the default value of the function tolerance and constraints
were satisfied to within the default value of the constraint tolerance.
...
Optimale Endzeit: 3.89263
Optimaler Wert Zielfunktional: 8.0061
```

Die Ergebnisse stimmen weitgehend mit den Ergebnissen aus Abschnitt 4.1.1 überein. Die Genauigkeit ist aufgrund der Approximation der Ableitungen durch finite Differenzen limitiert.

Für andere Optimalsteuerungsaufgaben liefern die Einstellungen `algorithm=interior-point`, `Hessian=lbfgs` bessere Ergebnisse als `algorithm=active-set`.

Signifikant bessere Ergebnisse werden erhalten, wenn 1. und 2. Ableitungen analytisch berechnet werden (`algorithm=interior-point`, `Hessian=user-supplied`: 27 Iterationen).

4.1.4 Direkte Kollokation mit CasADi

Die Implementierung der direkten Kollokation in MATLAB und CASADI, siehe Listing 4.7, basiert auf

> `direct_collocation.m`
> An implementation of direct collocation. Joel Andersson, 2016.

aus der Beispielsammlung `casadi-example_pack-v3.4.5`.

CASADI ist frei verfügbar, daher gibt es keine lizenzbedingten Einschränkungen der Problemdimension.

4.1.4.1 Doppelintegrator mit singulärem Lösungsabschnitt

Im Folgenden wird das Optimalsteuerungsproblem für den Doppelintegrator mit singulärem Lösungsabschnitt nach Abschnitt 2.4 betrachtet. Es wird eine implizite Mittelpunktregel auf einem äquidistantem Zeitgitter (4.1) verwendet, d. h. zwischen den Gitterpunkten werden die Steuergröße u zeitlich konstant (4.2a) und die Zustandsgrößen zeitlich linear (4.2b) approximiert. Der Zeithorizont wird normiert (siehe (3.2)), so dass die ursprünglich freie Endzeit t_f als zu optimierender Parameter in das Optimierungsproblem eingeht.

Listing 4.7 zeigt die Implementierung in MATLAB und CASADI. In den Zeilen 2 und 3 werden der MATLAB-Suchpfad angepasst und das Package `casadi` wird importiert. In den Zeilen 5 bis 19 werden die Parameter des Optimalsteuerungsproblems definiert. `N` bezeichnet die Anzahl der Intervall des Zeitgitters (K in (4.1)).

In den Zeilen 21–25 werden die symbolischen Variablen `x` und `u` für die Approximation der Zustands- und Steuergrößen auf dem Zeitgitter (4.1) und `tf` für den Parameter t_f angelegt. `SX` steht für 'scalar expression', in CasADi eine Matrix (oder Vektor oder Skalar), deren Elemente symbolische Ausdrücke sind, die durch Folgen skalar-wertiger Rechenoperationen erzeugt werden.

`xdot` (Zeile 27) und `L` (Zeile 29) sind die symbolischen Ausdrücke für die rechte Seite der Zustandsgleichung bzw. den Integranden des Zielfunktionals, dabei wird die Normierung der Zeit berücksichtigt. In Zeile 31 wird die Funktion `f` definiert, die `xdot` und `L` in Abhängigkeit von `x`, `u` und `tf` berechnet.

Nun wird in den Zeilen 33–74 das nichtlineare Optimierungsproblem definiert. Der Vektor der Optimierungsvariablen `w` mit den Komponentenbeschränkungen `lbw`$\leq$`w`$\leq$`ubw` und den Initialwerten `w0` wird aus dem Parameter t_f (Zeile 33), den Approximationen der Steuergröße u^k (Zeile 50) und der Zustandsgrößen $\mathbf{x}^k$ (Zeile 43 und 57) auf dem Zeitgitter zusammengesetzt. Die Funktion `f` wird an den Intervallmittelpunkten ausgewertet, und die nichtlinearen Beschränkungen `lbg`$\leq$`g(w)`$\leq$`ubg` sind hier die Kollokationsbedingungen (4.3) (Zeilen 67–70). In der Zielfunktion `J(w)` werden die Anteile der Zeitintervalle gemäß (4.4a) aufsummiert (Zeile 72).

Anschließend wird das Solver-Objekt `solver` mit dem zuvor definierten nichtlinearen Optimierungsproblem und dem Solver Ipopt in den Zeilen 76–77 definiert und in Zeile 79 aufgerufen und damit das nichtlineare Optimierungsproblem gelöst. Aus dem Lösungsvektor `w_opt` werden die optimale Endzeit `tf_opt` sowie die Approximationen der Zustandsgrößen `x_opt` und der Steuergrößen `u_opt` auf dem Zeitgitter extrahiert (Zeilen 82–86). Zuletzt wird das Zeitgitter entnormiert (Zeile 88).

Listing 4.7: `dising_collocation_imp_tf.m`

```
1  function dising_collocation_imp_tf()
   casadi_path;
3  import casadi.*
   % Parameter Problem
5  x0 = [-3; 0];              % Anfangszustand
   xf = [0; 0];               % Endzustand
7  nx = length(x0);
   umin = -1;                 % Steuerungsbeschraenkung
9  umax = 1;
   xmin = [-Inf; -Inf];       % Zustandsbeschraenkung
11 xmax = [Inf; Inf];
   rho_tf = 0.1;              % Bewertung Endzeit
13 T = 1;                     % normierte Endzeit
   tfmin = 3;                 % Beschraenkung Endzeit
15 tfmax = 10;
   uinit = 0;                 % Initialwert Steuerung
17 xinit = [1; 1];            % Initialwert Zustand
   tfinit = 5;                % Initialwert Endzeit
19 N = 500;                   % Anzahl Intervalle
   % Modellvariable fuer CasADi
21 x1 = SX.sym('x1');
   x2 = SX.sym('x2');
23 x = [x1; x2];
   u = SX.sym('u');
25 tf = SX.sym('tf');         % freie Endzeit tf als Parameter
   % Zustands-DGL
```

```
xdot = [x2; u]*tf;
% Lagrange-Term Zielfunktional
L = (0.5*(x1^2+x2^2)+rho_tf)*tf;
% Zustands-DGL und Lagrange-Term Zielfunktional
f = Function('f', {x, u, tf}, {xdot, L});
% NLP: Variable tf
w = {tf};
w0 = tfinit;
lbw = tfmin;
ubw = tfmax;
J = 0;
g = {};
lbg = [];
ubg = [];
% NLP: Anfangszustand ("Lift" initial conditions)
Xk = SX.sym('X0', nx);
w = [w(:)' {Xk}];
lbw = [lbw; x0];
ubw = [ubw; x0];
w0 = [w0; xinit];
% NLP: Kollokationsintervalle
for k = 0:N-1
    % NLP: Variable Steuerung u(k)
    Uk = SX.sym(['U_' num2str(k)]);
    w = [w(:)' {Uk}];
    lbw = [lbw; umin];
    ubw = [ubw; umax];
    w0 = [w0;  uinit];
    % NLP: Variable x(k+1)
    Xk_end = SX.sym(['X_' num2str(k+1)], nx);
    w = [w(:)' {Xk_end}];
        if k < N-1
      lbw = [lbw; xmin];
      ubw = [ubw; xmax];
        else
      lbw = [lbw; xf];
      ubw = [ubw; xf];
        end
    w0 = [w0; xinit];
    % Kollokationsbedingungen implizite Mittelpunktregel
    [fj, qj] = f((Xk_end+Xk)/2, Uk, tf);
    g = [g(:)' {Xk_end-Xk-T/N*fj}];
    lbg = [lbg; zeros(nx, 1)];
    ubg = [ubg; zeros(nx, 1)];
    % Zielfunktional
    J = J + qj*T/N;
    Xk = Xk_end;
end
% NLP: Solver
prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
solver = nlpsol('solver', 'ipopt', prob);
% NLP: Aufruf Solver
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw, 'lbg', lbg, 'ubg', ubg);
```

```
w_opt = full(sol.x);
% Loesungskomponenten
tf_opt = w_opt(1);
for i = 1:nx
    x_opt(:, i) = w_opt(1+i:nx+1:end);
end
u_opt = w_opt(1+nx+1:nx+1:end);
% Zeit entnormieren
t = (0:N)'/N*T*tf_opt;
```

Ipopt löst das Optimalsteuerungsproblem für den Doppelintegrator mit singulärem Lösungsabschnitt mit dem Kollokationsansatz in 28 Iterationen.

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 5.4887500e+000 4.00e+000 1.04e-002  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 1.2764475e+001 5.95e-003 2.42e+000  -1.0 4.00e+000    -  4.06e-001 1.00e+000h  1
   2 8.9064952e+000 3.33e-003 2.90e+000  -1.0 3.01e+000    -  4.08e-001 1.00e+000f  1
...
  26 8.0067430e+000 1.18e-007 9.46e-007  -8.6 2.82e-001    -  7.84e-001 9.63e-001f  1
  27 8.0067427e+000 1.84e-008 1.19e-008  -8.6 1.36e-001    -  1.00e+000 1.00e+000f  1
  28 8.0067421e+000 6.32e-009 2.48e-009  -9.0 1.39e-001    -  1.00e+000 1.00e+000h  1
```

4.2 Direkte Kollokation mit JuMP

JuMP [8] (https://github.com/JuliaOpt/JuMP.jl) ist eine algebraische Modellierungssprache für lineare und nichtlineare Optimierungsprobleme, die von zahlreichen Solvern über entsprechende Schnittstellen unterstützt wird. JuMP ist in die Programmiersprache Julia (https://julialang.org) eingebettet. JuMP und Julia sind quelloffen, so dass bei Nutzung von frei verfügbaren Solvern wie Ipopt keine lizenzbedingten Einschränkungen hinsichtlich der Problemdimension bestehen. Die Dokumentation zu JuMP ist unter http://www.juliaopt.org/JuMP.jl/v0.21/ zu finden, weiterhin sei auf die JuMP-Kurzbeschreibung [3] verwiesen.

4.2.0.1 Doppelintegrator mit singulärem Lösungsabschnitt

Das folgende Listing 4.8 zeigt die Lösung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 mittels direkter Kollokation (kubische Ansatzfunktion).

Listing 4.8: `dising_cub_jump.jl`

```
# Modellparameter
x10 = -3.0          # Anfangszustand
x20 = 0.0
x1f = 0.0           # Endzustand
x2f = 0.0
uminmax = 1.0       # Steuerungsbeschränkung
rhotf = 0.1         # Bewertung Endzeit
```

```
8  N = 500              # Anzahl Diskretisierungsintervalle
   # Zusatzpakete
10 using JuMP, Ipopt, DelimitedFiles
   # Modell und Solver
12 mod = Model(optimizer_with_attributes(Ipopt.Optimizer, "max_iter" => 1000))
   # Optimierungsvariable
14 @variable(mod, tf >= 0.0, start = 5.0)          # Optimierungshorizont
   @variable(mod, -uminmax <= u[0:N] <= uminmax)   # Steuergröße
16 @variable(mod, x1[0:N])                         # Zustandsgrößen
   @variable(mod, x2[0:N])
18 # Randbedingungen
   @constraint(mod, x1[0] == x10)
20 @constraint(mod, x2[0] == x20)
   @constraint(mod, x1[N] == x1f)
22 @constraint(mod, x2[N] == x2f)
   # DGL: Kollokationsbedingungen kubische Ansatzfunktion
24 @NLconstraint(mod, x1_collo[k=0:N-1], (x1[k+1]-x1[k]) -0.25*tf/N/1.5*(x2[k]+x2[
   k+1]) == tf/N/1.5*(0.5*x2[k]+0.5*x2[k+1]+0.125*tf/N*(u[k]-u[k+1])))
   @NLconstraint(mod, x2_collo[k=0:N-1], (x2[k+1]-x2[k]) -0.25*tf/N/1.5*(u[k]+u[k
   +1]) == tf/N/1.5*(0.5*u[k]+0.5*u[k+1]))
26 # Zielfunktional
   @NLobjective(mod, Min, rhotf*tf+tf/N*2.0/3.0*0.5*sum(0.25*x1[k]^2+0.25*x2[k
   ]^2+(0.5*x1[k]+0.5*x1[k+1]+0.125*tf/N*(x2[k]-x2[k+1]))^2+(0.5*x2[k]+0.5*x2[k
   +1]+0.125*tf/N*(u[k]-u[k+1]))^2+0.25*x1[k+1]^2+0.25*x2[k+1]^2 for k=0:N-1))
28 # Lösung: Aufruf Solver
   optimize!(mod)
```

Die Modellparameter werden in den Zeilen 2–7 und die Anzahl der Diskretisierungsintervalle bzw. Gitterpunkte entsprechend 4.1 in Zeile 8 festgelegt. In Zeile 10 werden die benötigten Zusatzpakete geladen. In Zeile 12 wird das Modell initialisiert und dabei der Solver ausgewählt.

Die Optimierungsvariablen werden mit dem Makro `@variable()` definiert. Dies sind der Optimierungshorizont (Zeile 14), für den eine untere Schranke und eine numerische Initialisierung vorgegeben werden, sowie die Werte der Steuer- und Zustandsgrößen in den Gitterpunkten (Zeilen 15–17).

Die Anfangs- und Endbedingungen werden als lineare Gleichungsbeschränkungen mit dem Makro `@constraint()` in den Zeilen 17–21 vorgegeben. Die Kollokationsbedingungen für die Approximation der Zustandsdifferentialgleichungen mittels kubischer Ansatzfunktion finden sich in den Zeilen 24 und 25. Für diese nichtlineare Gleichungsbeschränkungen ist das Makro `@NLconstraint()` zu verwenden.

In der Zeile 27 ist schließlich mit dem Makro `@NLobjective()` die diskrete Approximation des Zielfunktionals angegeben, und in Zeile 29 wird der Solver aufgerufen.

`julia.exe` startet ein interaktives Kommandozeilenprogramm (REPL: Read-Evaluate-Print-Loop). Mit dem Befehl `include()` wird die Datei in REPL geladen und die Ausführung gestartet:

```
julia> include("dising_cub_jump.jl")
```

Alternativ kann man `julia.exe` beim Aufruf den Namen einer Julia-Programmdatei übergeben, die dann im nicht-interaktiven Modus ausgeführt wird:

```
D:\Users\Arnold>julia dising_cub_jump.jl
```

Die Ergebnisse stimmen mit den Ergebnissen aus Abschnitt 4.1.1 überein.

4.3 Direkte Kollokation mit AMPL

AMPL [9] ist eine algebraische Modellierungssprache für lineare und nichtlineare Optimierungsprobleme, die von zahlreichen Solvern über entsprechende Schnittstellen unterstützt wird. Unter https://www.ampl.com ist eine im Funktionsumfang eingeschränkte „Studentenlizenz“ zusammen mit mehreren Solvern kostenlos verfügbar. Neben der AMPL-Kurzbeschreibung [4] finden sich im Internet zahlreiche AMPL-Tutorien und Beispielsammlungen, siehe z. B. https://vanderbei.princeton.edu/ampl/nlmodels/.

Im Rahmen des „Network Enabled Optimization Service (NEOS)“ https://neos-server.org/neos/ können Optimierungsprobleme im AMPL-Format an einen Server geschickt werden, die Lösung erhält man per Web-Interface oder E-Mail.

4.3.0.1 Doppelintegrator mit singulärem Lösungsabschnitt

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

Listing 4.9: `dising.mod`

```
# Modellparameter
param x10 := -3;                          # Anfangszustand
param x20 := 0;
param x1f := 0;                           # Endzustand
param x2f := 0;
param uminmax > 0, default Infinity; # Steuerungsbeschraenkung
param rhotf := 0.1;                       # Bewertung Endzeit
param N > 0, integer;  # Anzahl Diskretisierungsintervalle
set Nt := {0..N};
set Nt1 := {0..N-1};
# Optimierungsvariable
var T >= 0 := 5;          # Optimierungshorizont
var u {Nt} <= uminmax, >= -uminmax;
var x1 {Nt};
var x2 {Nt};
# Anfangswerte
subject to x1_init: x1[0] = x10;
subject to x2_init: x2[0] = x20;
# Endwerte
subject to x1_final: x1[N] = x1f;
subject to x2_final: x2[N] = x2f;
# Kollokationsbedingungen Zustands-DGL (Trapezregel)
subject to x1_collo {i in Nt1}: x1[i+1]-x1[i] = T/N*(x2[i+1]+x2[i])/2;
subject to x2_collo {i in Nt1}: x2[i+1]-x2[i] = T/N*(u[i+1]+u[i])/2;
# Zielfunktional
```

```
minimize cost: rhotf*T + sum {i in Nt1} T/N*0.5*(x1[i]^2+x2[i]^2+x1[i+1]^2+x2[
i+1]^2)/2;
```

Die Modellparameter werden in den Zeilen 2–7 festgelegt. Die Anzahl der Diskretisierungsintervalle bzw. Gitterpunkte in Zeile 8 entsprechend 4.1 ist durch den eingeschränkten Funktionsumfang der „Studentenlizenz“ limitiert. In den Zeilen 9 und 10 werden Indexmengen definiert. Die Optimierungsvariablen sind der Optimierungshorizont (Zeile 12), für den eine untere Schranke und eine numerische Initialisierung vorgegeben werden, sowie die Werte der Steuer- und Zustandsgrößen in den Gitterpunkten (Zeilen 13–15). Die Anfangs- und Endbedingungen werden als Gleichungsbeschränkungen in den Zeilen 17–21 vorgegeben. Die Kollokationsbedingungen für die Approximation der Zustandsdifferentialgleichungen mittels Trapezregel finden sich in den Zeilen 23 und 24. In der Zeile 26 ist schließlich die diskrete Approximation des Zielfunktionals angegeben.

Die Parameterwerte werden in einer separaten Datendatei bereit gestellt, siehe Listing 4.10.

Listing 4.10: `disingdat`

```
1 param uminmax := 1;
  param N := 98;
```

Die Steuerung des Ablaufs der Optimierungsrechnung erfolgt durch die Datei `disingrun`.

Listing 4.11: `disingrun`

```
  # Auswahl Modell- und Datendatei
2 model dising.mod;
  data dising.dat;
4 # Auswahl Solver
  option solver snopt;
6 # Loesung
  solve;
8 # Ausgabe
  display T, cost;
10 printf "      t[i]         x1[i]        x2[i]       u[i]\n";
  printf {i in Nt}: "%10.5f %10.5f %10.5f %10.5f\n", i/N*T, x1[i], x2[i], u[i];
12 printf {i in Nt}: "%10.5f %10.5f %10.5f %10.5f\n", i/N*T, x1[i], x2[i], u[i] >
   dising_res.dat;
```

Hier werden zunächst die Modell- und Datendatei ausgewählt (Zeilen 2 und 3). Der Aufruf des zuvor ausgewählten Solvers erfolgt in Zeile 7, danach folgt die Ergebnisausgabe auf die Konsole und in die Textdatei `dising_res.dat`.

Der Aufruf in AMPL (hier mit dem Solver SNOPT) erfolgt von der Eingabeaufforderung mittels

```
C:\Users\Arnold\cocp_ex\ampl>ampl dising.run
SNOPT 7.5-1.2 : Optimal solution found.
447 iterations, objective 8.007272138
Nonlin evals: obj = 65, grad = 64, constrs = 65, Jac = 64.
T = 3.88384
cost = 8.00727
```

```
     t[i]       x1[i]      x2[i]     u[i]
   0.00000   -3.00000    0.00000    1.00000
   0.03963   -2.99921    0.03963    1.00000
   0.07926   -2.99686    0.07926    1.00000
...
   3.80458   -0.00314    0.07926   -1.00000
   3.84421   -0.00079    0.03963   -1.00000
   3.88384    0.00000    0.00000   -1.00000
```

Die Ergebnisse stimmen mit den Ergebnissen aus Abschnitt 4.1.1 überein.

Eine Anbindung an MATLAB ist einfach möglich, siehe Datei `cocp_ex/ampl/doint_ampl.m`. Dabei wird die Datendatei `dising.dat` von MATLAB geschrieben. Anschließend wird AMPL aufgerufen, und die Ergebnisse werden aus der Datei `dising_res.dat` wieder nach MATLAB zur grafischen Auswertung übernommen.

4.4 Direkte Kollokation mit Modelica und Optimica

Einige Simulationsumgebungen oder Simulationswerkzeuge unterstützen die Formulierung und Lösung von Optimalsteuerungsproblemen. JMODELICA.ORG [15] ist eine frei verfügbare (quelloffene) Simulationsumgebung, die die Modellbeschreibungssprache MODELICA (https://modelica.org) mit der Erweiterung OPTIMICA [1] zur Beschreibung von Optimalsteuerungsaufgaben nutzt. Die numerische Lösung erfolgt mit Kollokationsverfahren und dem Solver IPOPT.

MODELICA ist eine objektorientierte Beschreibungssprache für dynamische Modelle. Der objektorientierte oder physikalische Modellierungsansatz ist eine Erweiterung der Blockdiagramm-Modellierung, bei dem u. a. die Zuordnung von Eingängen und Ausgängen der Komponenten nicht während der Modellierung festgelegt werden muss, sondern während der Aufbereitung der Modellgleichungen (automatisch) erfolgt. Dies erleichtert die domänenübergreifende Systemsimulation und die Erstellung wiederverwendbarer Komponenten-Bibliotheken. MODELICA ist eine freie Modellierungssprache, die von verschiedenen Simulationsumgebungen genutzt wird.

OPTIMICA ist eine Spracherweiterung von MODELICA zur Formulierung von Optimalsteuerungsproblemen. Die neue Klasse `optimization` gestattet die Definition von Zielfunktional und Beschränkungen sowie die Festlegung des Optimierungshorizonts und von Verfahrensparametern.

JMODELICA.ORG stellt u. a. die entsprechenden Modellcompiler bereit, die die Umsetzung des in OPTIMICA formulierten Optimalsteuerungsproblems mittels Kollokationsansatz in ein nichtlineares Optimierungsproblem ermöglicht. Die Bereitstellung der Ableitungen erfolgt mittels automatischer Differentiation und CPPAD (https://coin-or.github.io/CppAD/doc/cppad.htm) oder CASADI (https://web.casadi.org), die numerische Lösung des nichtlinearen Optimierungsproblems mit IPOPT [14], [20].

4.4.0.1 Doppelintegrator mit singulärem Lösungsabschnitt

Listing 4.12 zeigt die Formulierung der Optimalsteuerungsaufgabe für den Doppelintegrator mit singulärem Lösungsabschnitt aus Abschnitt 2.4 in MODELICA/OPTIMICA. Die Zustandsgleichungen sind im Simulationsmodell `dising` (Zeile 3-13) implementiert. Die Klasse `dising_opt` (Zeile 15-26) erweitert das Modell `dising` um Zielfunktional und Beschränkungen. Dabei wird der Integralteil des Zielfunktionals mit dem Attribut `objectiveIntegrand`, der MAYER-Term mit dem Attribut `objective` und die freie Endzeit mit dem Attribut `finalTime` festgelegt. In Zeile 21 wird die Eingangsgröße `u`, die im Modell `dising` als Signaleingang (`Input`, Zeile 8) definiert ist, als freie Variable des Optimalsteuerungsproblems gekennzeichnet. Dabei werden Minimal- und Maximalwert sowie eine Startnäherung angegeben.

Listing 4.12: MODELICA-Modell `dising_opt.mop`

```
1  package dising_pack
   // Simulationsmodell
3  model dising
     // Zustandsgroessen
5    Real x1(start=-3,fixed=true);
     Real x2(start=0,fixed=true);
7    // Eingangsgroesse
     input Real u;
9  equation
     // Zustands-DGL
11   der(x1)=x2;
     der(x2)=u;
13 end dising;
   // Klasse Optimalsteuerungsproblem; Attribute: Zielfunktional, Endzeit
15 optimization dising_opt (objectiveIntegrand = 0.5*(x1^2+x2^2),
                            objective = rho*finalTime,
17                          finalTime(free=true,min=0.1,initialGuess=5))
     // Parameter
19   parameter Real rho=0.1;
     // Vererbung: dising_opt erbt/erweitert dising
21   extends dising(u(free=true,min=-1,max=1,initialGuess=0.1));
   constraint
23   // Endzustandsbeschraenkungen (festes Ende)
     x1(finalTime)=0;
25   x2(finalTime)=0;
   end dising_opt;
27 end dising_pack;
```

Die Übersetzung des Modells und des Optimalsteuerungsproblems, der Start der Optimierung sowie die grafische Ausgabe der Simulationsergebnisse wird durch das Python-Skript `dising_opt.py` (Listing 4.13) gesteuert. Nach dem Import einiger Bibliotheken und Pakete wird in Zeile 6 das Optimierungsproblem geladen. In Zeile 13 erfolgt der Aufruf des Solvers mit den in Zeile 8-11 festgelegten Optionen.

Listing 4.13: Python-Skript `dising_opt.py`

```
# JModelica.org Python Packages importieren
2 from pyjmi import transfer_optimization_problem
```

```
# Python-Bibliotheken importieren
import matplotlib.pyplot as plt
# Optimalsteuerungsproblem laden
dising=transfer_optimization_problem("dising_pack.dising_opt","dising_opt.mop")
# Optionen
opt_opts=dising.optimize_options()
opt_opts['discr'] = 'LGR' # Legendre-Gauss-Radau-Kollokation
opt_opts['n_e'] = 50      # Anzahl finite Elemente
opt_opts['n_cp'] = 5      # Kollokationspunkte je Element
# Optimalsteuerungsproblem loesen (IPOPT)
res=dising.optimize(options=opt_opts)
print('Endzeit: %.3f' % res['finalTime'][0])
```

Die Lösung wird mit `Ipopt` in 27 Iterationen gefunden, und die Ergebnisse stimmen mit den Ergebnissen aus Abschnitt 4.1.1 überein.

Für weitere Details wie die Generierung von angepassten Startlösungen, die grafische Ausgabe der Ergebnisse und die Einstellung von Verfahrensparametern und Optionen wird auf die umfangreiche Online-Hilfe https://jmodelica.org/downloads/UsersGuide-2.4.pdf (dort insbesondere Kapitel 6) verwiesen.

5 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 Übungsaufgaben und Mini-Projekte in diesem Fall der Einsatz von indirekten Verfahren nicht empfohlen wird.

5.1.1.1 Zeitoptimale Umsteuerung eines PT_2-Glieds

Im Fall der zeitoptimalen Umsteuerung eines PT_2-Glieds nach Abschnitt 2.3 ergibt sich die Zweipunkt-Randwertaufgabe

$$\dot{\mathbf{y}} = \underbrace{\begin{bmatrix} -0.5 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix}}_{\mathbf{A}_{\text{kanon}}} \mathbf{y} + \begin{bmatrix} 0 \\ u(p_2) \\ 0 \\ 0 \end{bmatrix}, \quad \text{mit } \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix}, \quad u(p_2) \text{ nach (2.10c)} \tag{5.1a}$$

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

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

$$H|_{t_f} = 0 \tag{5.1d}$$

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 (5.1a) 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)

$$t \;=\; \begin{cases} \tau t_s & \text{für } 0 \le \tau \le 1 \\ t_s + (\tau - 1)(t_f - t_s) & \text{für } 1 \le \tau \le 2 \end{cases} \tag{5.2}$$

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.

$$\frac{d\mathbf{y}}{d\tau} \;=\; \mathbf{A}_{\text{kanon}}\mathbf{y} \cdot t_s + \begin{bmatrix} 0 \\ u_{minmax} \\ 0 \\ 0 \end{bmatrix}, \qquad 0 \le \tau \le 1, \quad \mathbf{y}(0) \;=\; \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{p}(0) \end{bmatrix} \tag{5.3a}$$

$$\frac{d\mathbf{y}}{d\tau} \;=\; \mathbf{A}_{\text{kanon}}\mathbf{y} \cdot (t_f - t_s) + \begin{bmatrix} 0 \\ -u_{minmax} \\ 0 \\ 0 \end{bmatrix}, \quad 1 \le \tau \le 2 \tag{5.3b}$$

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

Das mit dem Schießverfahren zu lösende nichtlineare Gleichungssystem setzt sich aus den verbleibenden Bedingungen am rechten Rand (5.1c) und (5.1d) sowie der Umschaltbedingung

$$y_3(\tau = 1) \;=\; p_2(t = t_s) \;=\; 0 \tag{5.4}$$

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

Die M-Funktion `cocp_ex/shoot/t2topt_shoot.m`, siehe Listing 5.1, realisiert das Schießverfahren für die zeitoptimale Umsteuerung des PT_2-Glieds. In Zeile 3 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 8–10 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 12 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 19 wird die Zeitachse entsprechend (5.2) entnormiert und in Zeile 20 die HAMILTON-Funktion H berechnet.

Listing 5.1: `t2topt_shoot.m`

```
function t2topt_shoot(uminmax_)
global x0 xf uminmax tyu odeoptions
odeoptions = odeset('AbsTol', 1e-10', 'RelTol', 1e-10, 'Refine', 10);
% Steuerungsbeschraenkungen
uminmax = uminmax_;
x0 = [-1; 0]; % Anfangswerte
xf = [0; 0];  % Endwerte
ts = 0.5;     % Startnaeherung Schaltzeit
tf = 1;       % Startnaeherung Endzeit
p0 = [-1; -0.1];% Startnaeherung Anfangskozustand
% Loesung nchtlineares Gleichungssystem
w = fsolve(@shoot, [p0; ts; tf], optimoptions('fsolve', 'Display', 'iter'));
% Auswertung Loesung
ts = w(3);
tf = w(end);
fprintf('\nOptimale Schaltzeit ts = %g\nOptimale Endzeit tf = %g\n', ...
    ts , tf)
t = tyu(:, 1);
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))+...
    tyu(:, 5).*(-tyu(:, 3)+tyu(:, 6));
clear global x0 xf uminmax tyu odeoptions rhs_par
```

Die rechten Seiten der Differentialgleichungen (5.3) sind in der Unterfunktion `rhs` implementiert, Listing 5.2, Zeile 11. Mit Hilfe der globalen Variable `rhs_par` (Zeile 3 bzw. 2 in 5.1) wird die Information zum Lösungsabschnitt, d. h. der Wert der Steuergröße `u` und des Zeitskalierungsfaktors `dt_dtnorm`, übermittelt.

Listing 5.2: `t2topt_shoot.m`, Unterfunktion `rhs`.

```
function yp = rhs(t, y)
% rhs - kanonisches Dgl.-System
global rhs_par
x = y(1:2);
p = y(3:4);
u = rhs_par(1);
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` (Listing 5.3), 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 Kozustandsdifferentialgleichung (Zeile 4), die Umschaltzeit (Zeile 5) und die Endzeit (Zeile 6) übergeben. Mit der bedingten Anweisung in Zeilen 7–10 wird „unsinnigen“ Parameterwerten, wie negativen Umschalt- oder Endzeiten, aber auch Werten von $p_2(0)$, die in Widerspruch zur Schaltbedingung (2.10c) stehen, sehr große Funktionswerte zugeordnet, die eine weitere Fortsetzung der Nullstellensuche in einer solchen Richtung verhindern.

In den Zeilen 12 und 13 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 (5.3a) 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 festgelegte Optionenvektor übergeben.

In Zeile 15 wird der Lösungsverlauf für die Ergebnisausgabe abgespeichert. In Zeile 16 wird der Restfehler (Residuum) der Schaltbedingung (5.4) berechnet.

In den Zeilen 18 und 19 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 (5.3b) gelöst, wobei als Anfangswerte $\mathbf{y}(\tau = 1)$ die Endwerte des ersten Lösungsabschnitts verwendet werden.

Abschließend werden die HAMILTON-Funktion zum Endzeitpunkt $H|_{t_f}$ berechnet (Zeile 25) und die Restfehler der Endzustandsbedingung (5.1c) (Zeile 27) sowie der Transversalitätsbedingung (5.1d) (Zeile 28) im Rückgabeparameter `res` gespeichert.

Listing 5.3: `t2topt_shoot.m`, Unterfunktion `shoot`.

```
1  function res = shoot(w)
   % shoot - Integration kanonisches Dgl.-System und Auswertung OB
3  global x0 xf uminmax tyu odeoptions rhs_par
   p0 = w(1:2);
5  ts = w(3);
   tf = w(end);
7  if ts <= 0 || tf <= ts || p0(2) > 0
       res = inf*ones(size(w));
9      return;
   end
11 % 1. bang-bang-Abschnitt: 0<=t<=ts, 0<=tnorm<=1
   rhs_par(1) = uminmax;
13 rhs_par(2) = ts;        % t = tnorm*ts
   [t, y] = ode45(@rhs, [0 1], [x0; p0], odeoptions);
15 tyu = [t y uminmax*ones(size(t))];
   res(1) = y(end, 4);       % p2(ts) = 0
17 % 2. bang-bang-Abschnitt: ts<=t<=tf, 1<=tnorm<=2
   rhs_par(1) = -uminmax;
19 rhs_par(2) = tf-ts;    % t = ts+(tnorm-1)*(tf-ts)
   [t, y] = ode45(@rhs, [1 2], y(end, :)', odeoptions);
21 tyu = [tyu; t y -uminmax*ones(size(t))];
   x_tf = y(end, 1:2)';
23 p_tf = y(end, 3:4)';
   u_tf = tyu(end, 6);
25 H_tf = 1+p_tf(1)*(-0.5*x_tf(1)+x_tf(2))+p_tf(2)*(-x_tf(2)+u_tf);
   res = [res; ...
27     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 (2.10c)) zu überprüfen. Gegebenenfalls ist die Schaltstruktur zu ändern und erneut die zugehörige Mehrpunkt-Randwertaufgabe aufzustellen und zu lösen.

Die Zeitverläufe in Abbildung 5.1 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 $t_f = 0.814619$.

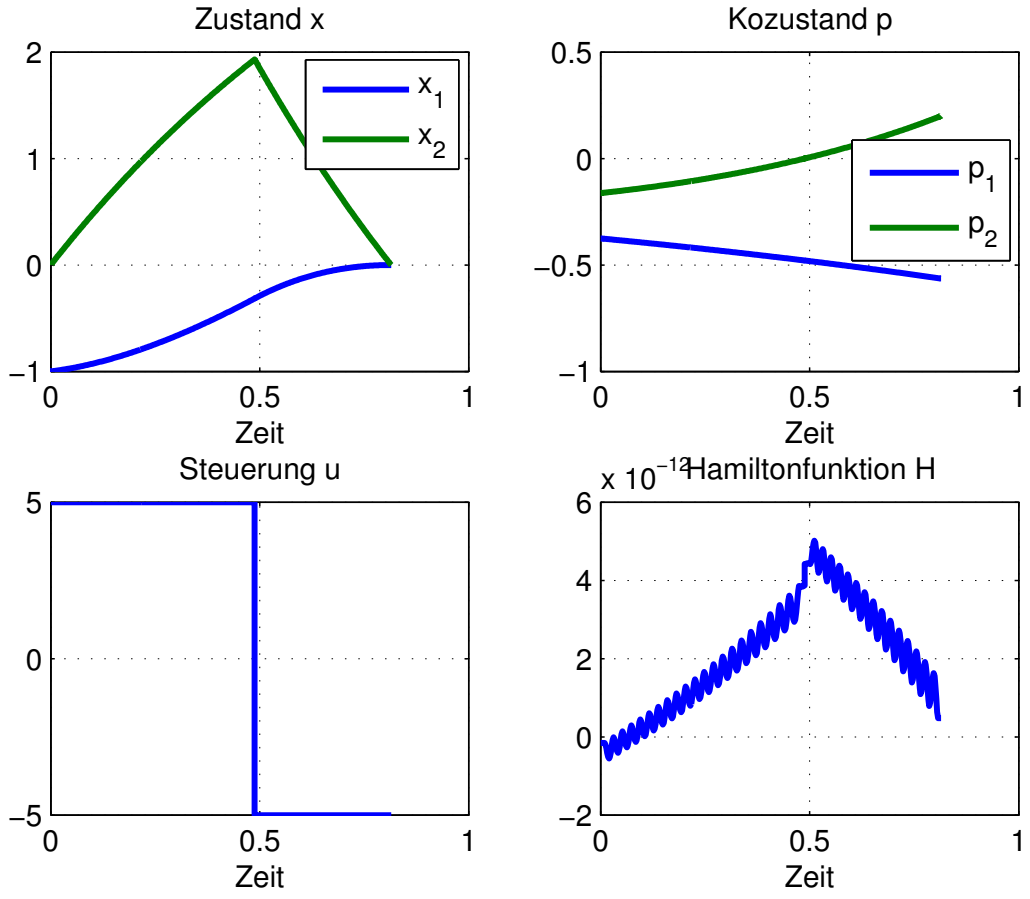


Abbildung 5.1: Zeitoptimale Umsteuerung PT_2-Glied; Lösung der Randwertaufgabe mit Schießverfahren.

6 Randwertaufgabe und Kollokation mit Matlab

Alternativ zu dem im Abschnitt 5 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].

Beim Kollokationsverfahren werden die zu bestimmenden Zeitfunktionen (Zustands- und Kozustandsgrößen) 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 so genannten Kollokationspunkten. Dies führt auf ein nichtlineares Gleichungssystem zur Bestimmung der Parameter der Ansatzfunktionen, in das die Rand- und Transversalitätsbedingungen 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 `https://de.mathworks.com/matlabcentral/fileexchange/3819-tutorial-on-solving-bvps-with-bvp4c` und in [17][2].

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.

[1] Siehe Hinweis auf Seite 67, 2. Absatz.

[2] Die folgenden Erläuterungen beziehen sich auf die MATLAB-Version 7.0 (R14) und höher, vorhergehende Versionen verwenden teilweise andere Funktionsbezeichnungen und unterscheiden sich in der Implementierung von Mehrpunkt-Randwertaufgaben.

6.1.1.1 Doppelintegrator mit Steuerungsbeschränkung

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

$$\dot{\mathbf{y}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\mathbf{A}_{\text{kanon}}} \mathbf{y} + \begin{bmatrix} 0 \\ u(p_2) \\ 0 \\ 0 \end{bmatrix}, \quad \text{mit } \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ p_1 \\ p_2 \end{bmatrix}, \quad u(p_2) \text{ nach (2.3c)} \tag{6.1a}$$

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \mathbf{x}_0 \tag{6.1b}$$

$$\begin{bmatrix} y_1(1) \\ y_2(1) \end{bmatrix} = \mathbf{x}_f \tag{6.1c}$$

In der `M`-Funktion `cocp_ex/bvp4c/doint1_bvp.m`, siehe Listing 6.1, wird das Kollokationsverfahren zur Lösung der Zweipunkt-Randwertaufgabe für den Doppelintegrator mit Steuerungsbeschränkung angewandt. Da die optimale Steuerung Knickstellen in den Übergängen zwischen aktiver und inaktiver Steuerungsbeschränkung aufweist und damit Ableitungen der Zustands- und Kozustandsgrößen dort unstetig sind, ist es sinnvoll, die Aufgabe numerisch als Mehrpunkt-Randwertaufgabe mit freien Umschaltzeitpunkten zwischen aktiver und inaktiver Steuerungsbeschränkung und den entsprechenden Schaltbedingungen (Eintritt bzw. Austritt von $u(t)$ in die oder aus der Steuerungsbeschränkung) zu behandeln.

Zunächst ist eine Annahme für die Schaltstruktur zu treffen: es wird davon ausgegangen, dass die Steuerung in einem Anfangszeitabschnitt auf der oberen Schranke liegt $u(t) = u_{\text{minmax}}$, $t \in [0, t_{s1}]$, dann ein unbeschränkter Zeitabschnitt $-u_{\text{minmax}} < u(t) < u_{\text{minmax}}$, $t \in (t_{s1}, t_{s2})$ folgt und schließlich die untere Schranke aktiv ist $u(t) = -u_{\text{minmax}}$, $t \in [t_{s2}, 1]$.

Die Umschaltzeiten t_{s1}, t_{s2} sind durch die Lösung der Randwertaufgabe zu bestimmen. Da die Randpunkte der Teilintervalle in `bvp4c` fest sind, muss die Mehrpunkt-Randwertaufgabe mit freien Umschaltzeiten – wie Abschnitt 5 erläutert – in eine Aufgabe mit mehreren Zeitintervallen fester Länge überführt werden. Hierzu wird eine normierte Zeit $\tau \in [0, 3]$ eingeführt, wobei $\tau = 0, 1, 2, 3$ den Zeitpunkten $t = 0, t_{s1}, t_{s2}, 1$ entspricht:

$$t = \begin{cases} \tau \cdot t_{s1} & \text{für } 0 \le \tau \le 1 \\ t_{s1} + (\tau - 1) \cdot (t_{s2} - t_{s1}) & \text{für } 1 < \tau \le 2 \\ t_{s2} + (\tau - 2) \cdot (1 - t_{s2}) & \text{für } 2 < \tau \le 3 \end{cases} \tag{6.2}$$

In Zeile 14 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. Die Trennstellen der Teilintervalle der Mehrpunkt-Randwertaufgabe werden durch die doppelte Angabe der Zeitpunkte `1.0` und `2.0` im Zeitgitter für die Splinefunktionen gekennzeichnet. Der zweite Aufrufparameter gibt konstante Startnäherungen für die vier Komponenten des Vektors $\mathbf{y}(t)$ vor. Dies ist im Fall des Doppelintegrators ausreichend. Weiterhin wird als dritter Aufrufparameter ein Vektor mit Startnäherungen der freien Parametern der Randwertaufgabe (hier: Umschaltzeiten t_{s1}, t_{s2}) übergeben.

In Zeile 17 erfolgt dann der eigentliche Aufruf des Kollokationsverfahrens. Es werden zwei Funktionen als „function handles“ (anonyme Funktionen, die `@ode` und `@bc` aufrufen) übergeben, die erste berechnet die zeitliche Ableitung der Funktionen aus der Differentialgleichung, die zweite den Restfehler der Randbedingungen. Durch die Nutzung anonymer Funktionen kann die Verwendung globaler Variabler zur Übergabe der Parameter `x0`, `xf` und `uminmax` vermieden werden. Weiterhin wird die Initiallösung und ein Satz von Einstellungen für Abbruchschranken und Zwischenausgaben an `bvp4c` übergeben.

Mit dem Aufruf von `deval` in Zeile 23 werden die Splineapproximationen an beliebig vorgebbaren Zeitpunkten ausgewertet. Aus dem Lösungsvektor werden dann die Variablen des Optimalsteuerungsproblems rückgerechnet (Zeilen 28–33). In Zeile 34 wird der optimale Wert des Zielfunktionals mittels Trapezregel berechnet.

Listing 6.1: `doint1_bvp.m`

```
function doint1_bvp(uminmax_)
% Anfangs- und Endwerte
x0 = [0; 0];
xf = [1; 0];
% Steuerungsbeschrnkungen
if nargin < 1
    uminmax_ = 4.5;
end
uminmax = uminmax_;
% Initialisierung Lsung
% Normierung Zeit: t=0,ts1,ts2,1 --> tau=0,1,2,3
ts1 = 0.1;
ts2 = 0.9;
solinit = bvpinit([linspace(0, 1, 5) linspace(1, 2, 5) linspace(2, 3, 5)], ...
    [0.5 1 -10 -1], [ts1; ts2]);
% Lsung RW-Aufgabe
sol = bvp4c(@(t, y, int_nr, par) ode(t, y, int_nr, par, uminmax), ...
    @(y0, yf, par) bc(y0, yf, par, x0, xf, uminmax), ...
    solinit, bvpset('stats', 'on', 'RelTol', 1e-6));
% Auswertung Lsung
tau = [linspace(0, 1-eps(1), 50) linspace(1+eps(1), 2-eps(2), 50) ...
    linspace(2+eps(2), 3, 50)]';
y = deval(sol, tau);  % >= R14
% Entnormierung Zeit: tau=0,1,2,3 --> t=0,ts1,ts2,1
ts1 = sol.parameters(1);
ts2 = sol.parameters(2);
end
t = (tau<=1).*tau*ts1 + (tau>1&tau<=2).*((tau-1)*(ts2-ts1)+ts1) + ...
    (tau>2).*((tau-2)*(1-ts2)+ts2);
x = y(1:2, :)';
p = y(3:4, :)';
u = max(-uminmax, min(-p(:, 2), uminmax));
H = 0.5*u.^2+p(:,1).*x(:,2)+p(:,2).*u;
J = 0.5*trapz(t, u.^2);
fprintf('Optimaler Wert Zielfunktional: %g\n', J)
```

Die Funktion `ode` berechnet die Ableitungen der Zeitfunktionen anhand des kanonischen Dif-

ferentialgleichungssystems (6.1a), wobei für jeden der Zeitabschnitte $t \in [0, t_{s1}]$, ($\tau \in [0,1]$), $t \in (t_{s1}, t_{s2}]$, ($\tau \in (1,2]$) und $t \in (t_{s2}, 1]$, ($\tau \in (2,3]$) (Übergabeparameter `int_nr`) die zugehörige Berechnungsvorschrift für $u(t)$ anzuwenden ist. Für die Zeitableitung ist entsprechend der Kettenregel

$$\frac{dy}{d\tau} = \frac{dy}{dt} \cdot \frac{dt}{d\tau}$$

und gemäß (6.2) mit der Länge des jeweiligen Zeitabschnitts zu multiplizieren (`dt_norm`).

Listing 6.2: `doint1_bvp.m`, Unterfunktion `ode`.

```
%------------     kanonisches Dgl.-System     ------------
function yd = ode(t, y, int_nr, par, uminmax)
% y = [x(1), x(2), p(1), p(2)]'
ts1 = par(1);
ts2 = par(2);
if int_nr == 1          % 1. Abschnitt: u=uminmax
    u = uminmax;
    dt_norm = ts1;
elseif int_nr == 2      % 2. Abschnitt: u frei, u=-p(2)
    u = -y(4);
    dt_norm = ts2-ts1;
elseif int_nr == 3      % 3. Abschnitt: u=-uminmax
    u = -uminmax;
    dt_norm = 1-ts2;
end
yd = [y(2); u; 0; -y(3)]*dt_norm;
```

Die Funktion `bc` wird von `bvp4c` mit den aktuellen Näherungen der Randwerte der Teilintervalle `y0` und `yf` aufgerufen. Im Rückgabeparameter werden die Restfehler der Randbedingungen erwartet (Zeilen 3–8 in Listing 6.3). Zu den Randbedingungen der ursprünglichen Aufgabe (6.1b), (6.1c) kommen noch die Stetigkeitsbedingungen für $\mathbf{x}(t)$ und $\mathbf{p}(t)$ in t_{s1} und t_{s2} sowie die Schaltbedingungen $u(t_{s1}) = u_{\text{minmax}}$ und $u(t_{s2}) = -u_{\text{minmax}}$ hinzu.

Listing 6.3: `doint1_bvp.m`, Unterfunktion `bc`.

```
%------------     Residuum Randbedingungen     ------------
function res = bc(y0, yf, par, x0, xf, uminmax)
res = [y0(1:2, 1)-x0; ...      % Anfangszustand x(0)=x0
    y0(:, 2)-yf(:, 1); ...     % Stetigkeit x(ts1), p(ts1)
    -y0(4, 2)-uminmax; ...     % Schaltbedingung -p2(ts1)=uminmax
    -yf(4, 2)+uminmax; ...     % Schaltbedingung -p2(ts2)=-uminmax
    y0(:, 3)-yf(:, 2); ...     % Stetigkeit x(ts2), p(ts2)
    yf(1:2, end)-xf];          % Endzustand x(1)=xf
```

Für den Doppelintegrator mit Steuerungsbeschränkung erhält man die folgenden Ausgaben, die optimalen Zeitverläufe sind in Abbildung 6.1 dargestellt.

```
>> doint1_bvp(4.5)
The solution was obtained on a mesh of 39 points.
The maximum residual is 5.702e-012.
There were 802 calls to the ODE function.
```

```
There were 142 calls to the BC function.

ts1=0.211325, ts2=0.788675
Optimaler Wert Zielfunktional: 6.22951
```

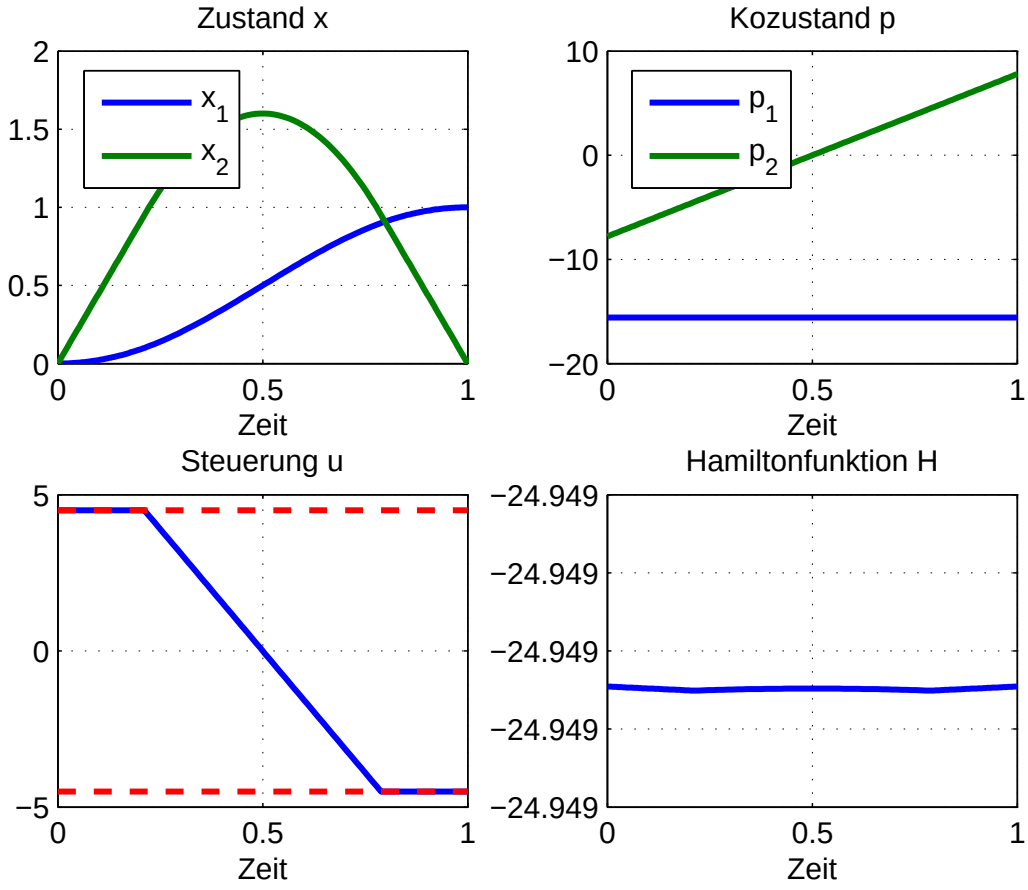


Abbildung 6.1: Doppelintegrator mit Steuerungsbeschränkung; Lösung der Randwertaufgabe mit BVP4C.

6.1.1.2 Zeitoptimale Umsteuerung eines PT_2-Glieds

Die zeitoptimale Umsteuerungsaufgabe für das PT_2-Glied nach Abschnitt 2.3 soll mit dem Kollokationsverfahren gelöst werden. Da die Randpunkte der Teilintervalle in `bvp4c` fest sind, muss die Mehrpunkt-Randwertaufgabe mit freier Umschalt- und Endzeit – wie Abschnitt 5 erläutert – in eine Aufgabe mit zwei Zeitintervallen fester Länge überführt werden.

In Zeile 5 des folgenden Listings 6.4 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.

Die Zeilen 10 und 11 zeigen, wie auf die Lösungen für die freien Parameter der Randwertaufgabe zugegriffen werden kann.

In den Zeilen 12–26 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 27–40 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 39).

Listing 6.4: `t2topt_bvp.m`

```
1  % Initialisierung Loesung
2  % Zeitnormierung:
3  %    1. Teilintervall (u = uminmax)   0 <= t <= ts --> 0 <= tnorm <= 1
4  %    2. Teilintervall (u = -uminmax) ts <= t <= tf --> 1 <= tnorm <= 2
5  solinit = bvpinit([linspace(0, 1.0, 10) linspace(1.0, 2.0, 10)], ...
6      [0 0 0 0], [ts; tf]);
7  % Loesung RW-Aufgabe
8  sol = bvp4c(@ode, @bc, solinit, bvpset('stats', 'on', 'RelTol', 1e-6));
9  % Auswertung Loesung
10 ts = sol.parameters(1);
11 tf = sol.parameters(2);
12 function yd = ode(t, y, int_nr, par)
13 % kanonisches Dgl.-System
14 global uminmax
15 x = y(1:2);
16 p = y(3:4);
17 ts = par(1);
18 tf = par(2);
19 if int_nr == 1            % 1. Teilintervall
20     u = uminmax;
21     dt_dtnorm = ts;       % t = tnorm*ts;
22 else                      % 2. Teilintervall
23     u = -uminmax;
24     dt_dtnorm = tf-ts;    % t = ts+(tnorm-1.0)*(tf-ts);
25 end
26 yd = [-0.5*x(1)+x(2); -x(2)+u; 0.5*p(1); -p(1)+p(2)]*dt_dtnorm;
```

```
27 function res = bc(y_0, y_f, par)
28 % Residuen Rand- und Transversalitaetsbedingungen
29 global x0 xf uminmax
30 x_0 = y_0(1:2, 1);                    % x(0)
31 x_f = y_f(1:2, 2);                    % x(tf)
32 p_f = y_f(3:4, 2);                    % p(tf)
33 p_s = y_f(3:4, 1);                    % p(ts)
34 u_f = -uminmax;                       % u(tf)
35 H_f = 1+p_f(1)*(-0.5*x_f(1)+x_f(2))+p_f(2)*(-x_f(2)+u_f); % H(tf)
36 res = [x_0-x0; ...                    % x(0)  = x0
37       x_f-xf; ...                    % x(tf) = xf
38       H_f; ...                       % H(tf) = 0
39       y_f(:, 1)-y_0(:, 2); ...       % Stetigkeit x, p in ts
40       p_s(2)];                       % Schaltbedingung
```

7 Indirektes Gradientenverfahren

Das indirekte Gradientenverfahren gehört zu den Suchverfahren im Funktionenraum, die bekannte Optimierungsverfahren aus dem Euklidischen Vektorraum $\mathbb{R}^n$ in allgemeine Funktionenräume übertragen. Es wird auch als „control vector iteration“ (CVI) bezeichnet.

Betrachtet wird ein Optimalsteuerungsproblem mit festem Horizont $[t_0, t_f]$, den Zustandsgrößen $\mathbf{x}(t) \in \mathbb{R}^n$, den Steuergrößen $\mathbf{u}(t) \in \mathbb{R}^m$, den Steuerparametern $\mathbf{w} \in \mathbb{R}^{n_w}$, festem Anfangszustand $\mathbf{x}_0$ und freiem Endzustand sowie einfachen Komponentenbeschränkungen für $\mathbf{u}(t)$ und $\mathbf{w}$.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t), \quad t \in (t_0, t_f), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{7.1a}$$

$$J = F(\mathbf{x}(t_f), \mathbf{w}) + \int_{t_0}^{t_f} f_0(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)\, dt \quad \longrightarrow \quad \min! \tag{7.1b}$$

$$\mathbf{u}_{\min}(t) \leq \mathbf{u}(t) \leq \mathbf{u}_{\max}(t), \quad t \in [t_0, t_f] \tag{7.1c}$$

$$\mathbf{w}_{\min} \leq \mathbf{w} \leq \mathbf{w}_{\max} \tag{7.1d}$$

Mit den Kozustandsgrößen $\mathbf{p}(t) \in \mathbb{R}^n$, der HAMILTON-Funktion $H = f_0 + \mathbf{p}^T f$ und unter Ausnutzung der Optimalitätsbedingungen kann bei gegebenem $\mathbf{u}(t)$ und $\mathbf{w}$ der reduzierte Gradient des Zielfunktionals $J_\mathbf{u}(t) \in \mathbb{R}^m$, $J_\mathbf{w} \in \mathbb{R}^{n_w}$ berechnet werden. Die Ungleichungsbeschränkungen (7.1c), (7.1d) werden dabei zunächst nicht berücksichtigt.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{7.2a}$$

$$\dot{\mathbf{p}} = -H_\mathbf{x} = -\left(f_{0,\mathbf{x}} + \mathbf{f}_\mathbf{x}^T \mathbf{p}\right), \qquad \mathbf{p}(t_f) = F_{\mathbf{x}(t_f)} \tag{7.2b}$$

$$J_\mathbf{u} = H_\mathbf{u} = f_{0,\mathbf{u}} + \mathbf{f}_\mathbf{u}^T \mathbf{p} \tag{7.2c}$$

$$J_\mathbf{w} = F_\mathbf{w} + \int_{t_0}^{t_f} H_\mathbf{w}\, dt \tag{7.2d}$$

Ein Optimalsteuerungsproblem mit freier Endzeit t_f kann mit 3.2 in ein Problem mit fester Endzeit transformiert werden, t_f wird dann zu einem Steuerparameter. Erweiterungen der Aufgabe (7.1a)-(7.1d), wie z. B. Endzustandsbeschränkungen $\mathbf{h}(\mathbf{x}(t_f)) = \mathbf{0}$ oder allgemeine Trajektorienbeschränkungen $\mathbf{g}(\mathbf{x}, \mathbf{u}, t) \leq \mathbf{0}$ können durch geeignete Straffunktionen berücksichtigt werden.

Die Zustandsdifferentialgleichung (7.2a) und die Kozustandsdifferentialgleichung (7.2b) werden mit einem geeigneten numerischen Integrationsverfahren diskretisiert. Dabei wird die Zustandsdifferentialgleichung (7.2a) mit den gegebenen Anfangswerten $\mathbf{x}(t_0)$ zeitlich vorwärts und die Kozustandsdifferentialgleichung (7.2b) mit den gegebenen Endwerten $\mathbf{p}(t_f)$ zeitlich rückwärts von t_f bis t_0 integriert. In den Beispielen wird hierzu ein HEUN-Verfahren mit fester Schrittweite ΔT genutzt.

Die Stützstellen der Steuergrößen $\mathbf{u}(t_k)$ auf dem sich dadurch ergebenden Diskretisierungsgitter $t_k = t_0 + k\Delta T$, $k = 0, \ldots, K$ mit $\Delta T = \frac{t_f - t_0}{K}$ und die Steuerparameter $\mathbf{w}$ werden im Vektor $\mathbf{y}$ zu-

sammengefasst:

$$\mathbf{y} = \begin{bmatrix} \mathbf{u}^0 \\ \vdots \\ \mathbf{u}^K \\ \mathbf{w} \end{bmatrix}, \qquad \mathbf{y}_{\min} = \begin{bmatrix} \mathbf{u}_{\min}(t^0) \\ \vdots \\ \mathbf{u}_{\min}(t^K) \\ \mathbf{w}_{\min} \end{bmatrix}, \qquad \mathbf{y}_{\max} = \begin{bmatrix} \mathbf{u}_{\max}(t^0) \\ \vdots \\ \mathbf{u}_{\max}(t^K) \\ \mathbf{w}_{\max} \end{bmatrix}, \qquad \nabla J = \begin{bmatrix} J_{\mathbf{u}^0} \\ \vdots \\ J_{\mathbf{u}^K} \\ J_{\mathbf{w}} \end{bmatrix}$$

Das Optimalsteuerungsproblem (7.1a)-(7.1d) wird so durch das nichtlineare Optimierungsproblem

$$\min \left\{ J(\mathbf{y}) \,\middle|\, \mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max} \right\} \tag{7.3}$$

approximiert. Zu dessen Lösung können ableitungsbehaftete Verfahren eingesetzt werden, bei denen der (negative) Gradient zur Bestimmung der Suchrichtung zur Verbesserung einer nicht-optimalen Trajektorie $\mathbf{u}(t)$ bzw. nicht-optimaler Steuerparameter $\mathbf{w}$ genutzt wird.

7.1.1.1 Projiziertes schnelles Gradientenverfahren nach Nesterov

In den Beispielen wird zur Lösung des nichtlinearen Optimierungsproblems (7.3) ein projiziertes schnelles Gradientenverfahren nach Nesterov genutzt. Dies ist ein Gradientenverfahren mit fester Schrittweite, bei dem durch Einbeziehung mehrerer Iterationspunkte (Hilfspunkte) die Konvergenz gegenüber einem einfachen Gradientenverfahren deutlich beschleunigt wird.

$\mathbf{y}^0$ {Startpunkt}
$\bar{\mathbf{y}}^0 \Leftarrow \mathbf{y}^0$ {Hilfspunkt}
$\alpha^0 \Leftarrow \frac{1}{2}(\sqrt{5}-1)$
$L > 0$ {Lipschitz-Konstante}
$k \Leftarrow 0$ {Iterationszähler}
repeat
  $\mathbf{y}^{k+1} \Leftarrow \Pi_{\mathbb{Y}}\left(\bar{\mathbf{y}}^k - \frac{1}{L}\nabla J(\bar{\mathbf{y}}^k)\right)$ {projizierter Gradientenschritt}
  $\alpha^{k+1} \Leftarrow \frac{\alpha^k}{2}\left(\sqrt{(\alpha^k)^2+4} - \alpha^k\right)$
  $\bar{\mathbf{y}}^{k+1} \Leftarrow \mathbf{y}^{k+1} + \frac{\alpha^k(1-\alpha^k)}{(\alpha^k)^2+\alpha^{k+1}}(\mathbf{y}^{k+1}-\mathbf{y}^k)$ {Hilfspunkt}
  $k \Leftarrow k+1$
until Abbruchbedingung erfüllt

Die Projektion $\Pi_{\mathbb{Y}}$ auf die zulässige Menge kann für die Komponentenbeschränkungen in (7.3) komponentenweise berechnet werden:

$$\Pi_{\mathbb{Y}}(y_i) = \begin{cases} y_{\min,i} & \text{falls } y_i < y_{\min,i} \\ y_{\max,i} & \text{falls } y_i > y_{\max,i} \\ y_i & \text{sonst} \end{cases}$$

Die Lipschitz-Konstante $L > 0$ wird numerisch-experimentell bestimmt: je kleiner L gewählt wird, um so schneller konvergiert das Verfahren, bei zu kleinem L verlangsamt sich die Konvergenz oder es divergiert sogar.

Als Abbruchbedingung kann die Änderung des Zielfunktionswerts oder die Norm des projizierten Gradienten herangezogen werden. Dabei ist zu beachten, dass das Verfahren kein Abstiegsverfahren ist und daher die Folge der Zielfunktionswerte $\{J(\mathbf{y}^k)\}$ u. U. nicht monoton fällt.

7.1.1.2 Zeitoptimale Umsteuerung eines PT_2-Glieds

In der M-Funktion `cocp_ex/cvi/t2topt_cvi_fgrad.m`, siehe Listing 7.1, wird das oben beschriebene Verfahren zur Lösung der zeitoptimalen Umsteuerungsaufgabe für das PT_2-Glied nach Abschnitt 2.3 verwendet.

Die Parameter des Problems werden in den Zeilen 2–12 definiert. Durch eine Zeitnormierung wird die freie Endzeit zum Steuerparameter $w = t_f$. Der Zeitverlauf der Steuergröße wird mit $u(t) = 0.1$ initialisiert, der Parameter w mit `tf_init`=1, siehe Zeile 17. Nach Vorgabe der Lipschitz-Konstanten L folgt in den Zeilen 21–40 das oben im Pseudocode dargestellte projizierte schnelle Gradientenverfahren.

Listing 7.1: `t2topt_cvi_fgrad.m`

```
1  function t2topt_cvi_fgrad( uminmax, tf_init )
   if ~nargin
3      uminmax = 5;      % Steuerungsbeschraenkung
       tf_init = 1.0;    % Startwert freie Endzeit (Parameter)
5  end
   % Parameter
7  x0 = [-1; 0];         % Anfangszustand
   xf = [0; 0];          % Endzustand
9  tf = 1;               % Endzeit (normiert)
   rho_xf = 20;          % Strafkoeffizient Endzustandsbeschraenkung
11 K = 100;              % Anzahl Diskretisierungsintervalle
   dT = tf/K;            % Zeitschritt (normiert)
13 % Startloesung [u(0), ..., u(K), tf]
   u_init = 0.1;         % Startwert Steuerung
15 Umin = [-uminmax*ones(K+1, 1); 0.1]; % untere Schranke
   Umax = [uminmax*ones(K+1, 1); Inf];  % obere Schranke
17 U = max(Umin, min([u_init*ones(K+1, 1); tf_init], Umax));
   % Lipschitz-Konstante
19 L = 50;
   % Iteration schnelles Gradientenverfahren
21 Y = U;
   alpha = 0.5*(sqrt(5)-1);
23 max_iter = 10000;                  % max. Iterationszahl
   J = zeros(max_iter+1, 2);          % Zielfunktional, Norm Gradient
25 for iter = 0:max_iter
       [Hu, X, P] = gradient(Y, x0, xf, rho_xf, tf, dT); % red. Gradient
27     Up = max(Umin, min(Y-1/L*Hu, Umax));              % Projektion
       alphap = alpha/2*(sqrt(alpha^2+4)-alpha);
29     beta = alpha*(1-alpha)/(alpha^2+alphap);
       Y = Up+beta*(Up-U);
31     alpha = alphap;
       U = Up;
33     J(iter+1, 1) = U(end)+rho_xf/2*norm(X(end, :)'-xf)^2; % Zielfunktional
       Hu_proj = Hu.*(U>=Umin+1e-6 & U<=Umax-1e-6);        % proj. red. Gradient
35     J(iter+1, 2) = sqrt(trapz(Hu_proj(1:K+1).^2)*dT)+abs(Hu_proj(end));
       if iter > 0 && abs(J(iter+1, 1)-J(iter, 1)) < 1e-8  && ...
37             J(iter+1, 2) < 1e-4                          % Abbruchtest
           break
39     end
```

```
end
```

In der Funktion `gradient`, siehe Listing 7.2, wird nach numerischer Integration der Zustandsdifferentialgleichung (Zeile 7–19) sowie der Kozustandsdifferentialgleichung (Zeile 20–34) der reduzierte Gradient (Zeile 35–37) und die HAMILTON-Funktion (Zeile 38–46) berechnet.

Listing 7.2: `t2topt_cvi_fgrad.m`: Unterfunktion `gradient`

```
1  % reduzierter Gradient, HEUN-Integration
   function [Hu, X, P, H] = gradient(U, x0, xf, rho_xf, tf, dT)
3  K = round(tf/dT);
   X = zeros(K+1, length(x0));
5  P = X;
   H = zeros(K+1, 1);
7  % Anfangszustand
   x = x0;
9  X(1, :) = x';
   % Zustands-DGL
11 for k = 0:K-1
       u = U(k+1);
13     s1 = [-0.5*x(1)+x(2); -x(2)+u]*U(end);
       x = x+dT*s1;
15     u = U(k+2);
       s2 = [-0.5*x(1)+x(2); -x(2)+u]*U(end);
17     x = X(k+1, :)'+dT/2*(s1+s2);
       X(k+2, :) = x';
19 end
   % Transversalitaetsbedingung
21 p = rho_xf*(X(end, :)'-xf);
   P(end, :) = p';
23 % Kozustands-DGL
   for k=K:-1:1
25     u = U(k+1);
       x = X(k+1, :)';
27     s1 = [0.5*p(1); -p(1)+p(2)]*U(end);
       p = p-dT*s1;
29     u = U(k);
       x = X(k, :)';
31     s2 = [0.5*p(1); -p(1)+p(2)]*U(end);
       p = P(k+1, :)'-dT/2*(s1+s2);
33     P(k, :) = p';
   end
35 % reduzierter Gradient
   Hu = [P(:, 2)*dT*U(end); ...
37     1.0+trapz((-0.5*X(:, 1)+X(:, 2)).*P(:, 1)+(-X(:, 2)+U(1:end-1)).*P(:, 2))*
       dT];
   % Hamiltonfunktion
39 if nargout > 3
       for k = 0:K
41         x = X(k+1, :)';
           p = P(k+1, :)';
43         u = U(k+1);
           H(k+1) = (p(1)*(-0.5*x(1)+x(2))+p(2)*(-x(2)+u))*U(end);
```

```
45      end
end
```

Mit einem Strafterm zur Berücksichtigung des festen Endzustands

$$\frac{\rho_f}{2}\|\mathbf{x}(t_f) - \mathbf{x}_f\|^2\,, \qquad \rho_f \;=\; 20$$

und einer Lipschitz-Konstante $L = 50$ wird nach 4093 Iterationen die in 7.1 dargestellte Lösung gefunden. Die optimale Endzeit ist $t_f = 0.797$, $x_1(t_f) = -0.0292$ und $x_2(t_f) = 0.00813$ weichen – bedingt durch die verwendete (äußere) Straffunktion – vom vorgegebenen Endzustand $\mathbf{x}_f$ ab.

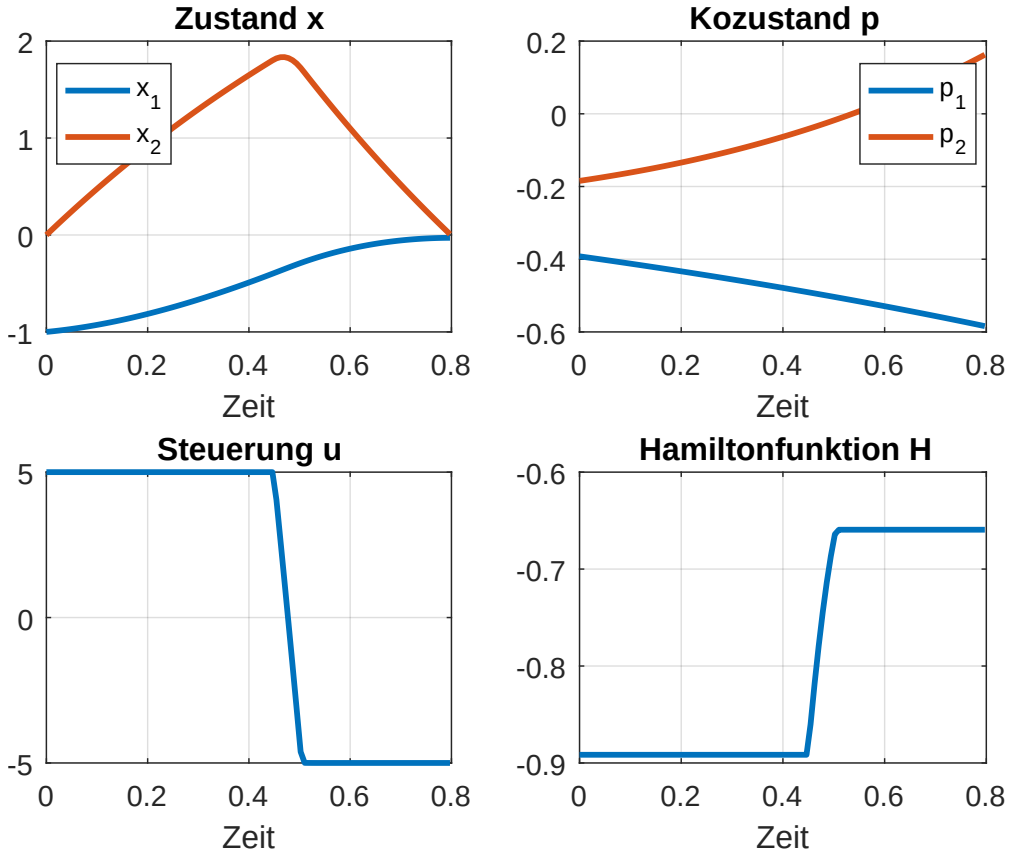


Abbildung 7.1: Zeitoptimale Umsteuerung PT_2-Glied; indirektes Gradientenverfahren ($K = 100$, $\rho_f = 20$).

Die anhand dieses Beispiels demonstrierten Eigenschaften des indirekten Gradientenverfahrens, wie die im Vergleich zu anderen Verfahren trotz hoher Iterationszahl ungenaue Approximation der Lösung, sind häufig zu beobachten.

Literatur

[1] J. Åkesson u. a. „Modeling and optimization with Optimica and JModelica.org – Languages and tools for solving large-scale dynamic optimization problems“. In: *Computers & Chemical Engineering* 34.11 (2010), S. 1737–1749. DOI: 10.1016/j.compchemeng.2009.11.011 (siehe S. 64).

[2] J. A. E. Andersson u. a. „CasADi – A software framework for nonlinear optimization and optimal control“. In: *Mathematical Programming Computation* 11.1 (2019), S. 1–36. DOI: 10.1007/s12532-018-0139-4 (siehe S. 3, 33).

[3] E. Arnold. *JuMP-Kurzbeschreibung*. 2020. URL: https://www.isys.uni-stuttgart.de/lehre/lehrveranstaltungen/nmopt/Aufgaben/JuMP_Kurzbeschreibung.pdf (siehe S. 60).

[4] E. Arnold. *AMPL-Kurzbeschreibung*. 2016. URL: https://www.isys.uni-stuttgart.de/lehre/lehrveranstaltungen/nmopt/Aufgaben/AMPL_Kurzbeschreibung.pdf (siehe S. 62).

[5] E. Arnold. *Numerische Methoden der Optimierung und Optimalen Steuerung*. Vorlesung Universität Stuttgart. 2020. URL: https://www.isys.uni-stuttgart.de/lehre/lehrveranstaltungen/nmopt (siehe S. 3).

[6] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. 2. Aufl. Philadelphia: SIAM, 2010 (siehe S. 44).

[7] K. E. Brenan, S. L. Campbell und L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. North-Holland, 1989 (siehe S. 15).

[8] I. Dunning, J. Huchette und M. Lubin. „JuMP: A modeling language for mathematical optimization“. In: *arXiv:1508.01982 [math.OC]* (2015). URL: https://arxiv.org/abs/1508.01982 (siehe S. 60).

[9] R. Fourer, D. M. Gay und B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury, 1993. URL: https://ampl.com/resources/the-ampl-book/ (siehe S. 62).

[10] R. Franke. *Omuses and HQP*. Techn. Ber. Technische Universität Ilmenau, 1998. URL: http://hqp.sourceforge.net/omuses.pdf (siehe S. 3, 14, 15).

[11] P. E. Gill, W. Murray und M. A. Saunders. „SNOPT: An SQP algorithm for large-scale constrained optimization“. In: *SIAM J. Optim.* 12.4 (2002), S. 979–1006. URL: http://www.ccom.ucsd.edu/~peg/papers/snpaper.pdf (siehe S. 47).

[12] A. Griewank, D. Juedes und J. Utke. „ADOL-C: A package for the automatic differentiation of algorithms written in C/C++“. In: *ACM Trans. Math. Software* 22.2 (Juni 1996), S. 131–167. DOI: 10.1145/229473.229474 (siehe S. 15, 18).

[13] B. Houska, H. J. Ferreau und M. Diehl. „ACADO toolkit – an open-source framework for automatic control and dynamic optimization“. In: *Optimal Control Appl. Methods* 32.3 (2011), S. 298–312. DOI: 10.1002/oca.939 (siehe S. 3, 29).

[14] *Ipopt: Interior-Point Optimizer for general large-scale nonlinear optimization*. 2013. URL: https://github.com/coin-or/Ipopt (siehe S. 3, 4, 50, 64).

[15] *JModelica.org: open source Modelica platform for modeling, simulation and optimization*. 2013. URL: https://jmodelica.org (siehe S. 64).

[16] M. Papageorgiou, M. Leibold und M. Buss. *Optimierung: statische, dynamische, stochastische Verfahren für die Anwendung*. Berlin: Springer, 2012. DOI: 10.1007/978-3-540-34013-3 (siehe S. 9).

[17] L. F. Shampine, J. Kierzenka und M. W. Reichelt. *Solving boundary value problems for ordinary differential equations in MATLAB with bvp4c*. Techn. Ber. Natick: The MathWorks, 2000. URL: https://de.mathworks.com/matlabcentral/fileexchange/3819-tutorial-on-solving-bvps-with-bvp4c (siehe S. 72).

[18] D. E. Stewart. *Meschach: matrix computations in C*. University of Canberra. Canberra, Australia, 1992 (siehe S. 15).

[19] O. von Stryk. „Numerische Lösung optimaler Steuerungsprobleme: Diskretisierung, Parameteroptimierung und Berechnung der adjungierten Variablen“. Fortschr.-Ber. VDI, Reihe 8, Nr. 441. Düsseldorf: VDI-Verlag 1995. Diss. Technische Universität München, 1994. URL: https://www.sim.informatik.tu-darmstadt.de/publ/download/1994-diss.html (siehe S. 44, 46, 47).

[20] A. Wächter und L. T. Biegler. „On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming“. In: *Mathematical Programming* 106.1 (2006), S. 25–57. DOI: 10.1007/s10107-004-0559-y (siehe S. 3, 4, 50, 64).