Optimale Steuerung und Regelung - Übung 2

Anton Bernecker - 62100410

May 13, 2025

Einleitung

In dieser Übung wird das Wagen-Pendel-System analysiert und mit Methoden der optimalen Steuerung geregelt. Ziel ist es, die optimale Trajektorie für das System unter gegebenen Anfangs- und Endbedingungen sowie Nebenbedingungen zu berechnen. Anschließend wird ein linearer Zustandsregler entlang der optimalen Trajektorie entworfen.

Aufgabe 1a: Diskretisierung und Gradientenberechnung

Das dynamische System ist gegeben durch die nichtlineare Differentialgleichung

$$\dot{x}(t) = f(x(t), u(t)) = \begin{pmatrix} \omega(t) \\ \frac{mga\sin\theta(t) - d\omega(t) + mau(t)\cos\theta(t)}{J+ma^2} \\ v(t) \\ u(t) \end{pmatrix},$$

mit dem Zustand $x = (\theta, \omega, s, v)^\top$ und der Steuergröße $u = \ddot{s}$. Die Anfangsbedingung lautet

$$x(0) = (\pi, 0, -0{,}5, 0)^\top.$$

Die Endbedingung wird durch $Mx(t_1) = 0$ mit gegebener Matrix M vorgegeben, sodass das Pendel in der instabilen Ruhelage liegt, die Winkelgeschwindigkeit sowie die Geschwindigkeit des Wagens ebenfalls auf null sind. Die Position des Wagens wird nicht explizit geregelt, jedoch werden durch die Gewichtungsmatrix **S** Abweichungen der Wagenposition bestraft.

Es wird ein Optimalsteuerungsproblem mit dem Kostenfunktional

$$J(u) = \frac{1}{2}x^\top(t_1)Sx(t_1) + \int_0^{t_1} \left(1 + \frac{1}{2}(x^\top Qx + Ru^2)\right) dt$$

formuliert. Die Matrizen Q, S und der Skalar R sind Gewichtungsfaktoren zur Bewertung der Zustände und der Stellgröße.

Die Lösung erfolgt mit der direkten Methode der Volldiskretisierung unter Verwendung der impliziten Trapezregel für die Dynamik:

$$x_{i+1} - x_i - \frac{\Delta t}{2} \left(f(x_i, u_i) + f(x_{i+1}, u_{i+1}) \right) = 0.$$

Die Gleichungsnebenbedingungen setzen sich aus den Startbedingungen, Endbedingungen sowie der diskretisierten Systemgleichung zusammen.

Zur Optimierung ist der Gradient des Funktionals erforderlich. Dieser hat die Form:

$$\frac{\partial J_d}{\partial y} = \left(\frac{\partial J_d}{\partial x_1}, \frac{\partial J_d}{\partial u_1}, \frac{\partial J_d}{\partial x_2}, \frac{\partial J_d}{\partial u_2}, \ldots, \frac{\partial J_d}{\partial x_N}, \frac{\partial J_d}{\partial u_N} \right).$$

Für die numerische Approximation des Gradienten wurde die Methode `approx_fprime` gewählt. Diese Funktion aus dem Modul `scipy.optimize` verwendet eine Vorwärtsdifferenz mit adaptiv gewähltem ε, um den Gradienten effizient zu approximieren:

$$\frac{\partial J_d}{\partial y_i} \approx \frac{J_d(y + \varepsilon e_i) - J_d(y)}{\varepsilon}.$$

Zur Lösung des Optimierungsproblems mit Gleichungsnebenbedingungen in der Form

$$\min_y \; J_d(y) \quad \text{u.B.v.} \quad g_d(y) = 0,$$

wird außerdem die Jacobi-Matrix $\frac{\partial g_d}{\partial y}$ benötigt. Diese enthält die partiellen Ableitungen aller Nebenbedingungen nach allen Freiheitsgraden und ist essenziell für gradientenbasierte Optimierungsalgorithmen wie `SLSQP`, da sie die Sensitivität der Nebenbedingungen bezüglich der Optimierungsvariablen beschreibt.

Dabei wird jede Komponente der Nebenbedingungen separat als skalare Funktion betrachtet, und ihr Gradient wird mithilfe der Methode `approx_fprime` approximiert. Diese Vorgehensweise ermöglicht eine effiziente numerische Bestimmung der vollständigen Jacobi-Matrix.

Aufgabe 1b: Minimale Anzahl an Diskretisierungspunkten

Damit das diskretisierte Optimierungsproblem stets eine Lösung besitzt, muss die Anzahl der Optimierungsvariablen größer oder gleich der Anzahl der Nebenbedingungen sein.

Für jede Stufe $i = 1, \ldots, N-1$ entstehen 4 Gleichungen aus der diskretisierten Systemdynamik. Zusätzlich ergeben sich 4 Anfangsbedingungen sowie 3 Endbedingungen. Insgesamt ergeben sich also

$$\underbrace{4(N-1)}_{\text{Systemgleichungen}} + \underbrace{4}_{\text{Anfangsbedingung}} + \underbrace{3}_{\text{Endbedingung}} = 4(N-1)+7$$

Nebenbedingungen.

Der Optimierungsvektor y besteht aus N Zustandsvektoren $x_i \in \mathbb{R}^4$ und N Steuergrößen $u_i \in \mathbb{R}$, also insgesamt **5N** Variablen.

Die Bedingung lautet also:

$$5N \geq 4(N-1)+7 \quad \Rightarrow \quad 5N \geq 4N - 4 + 7 = 4N + 3 \quad \Rightarrow \quad N \geq 3.$$

Es müssen somit mindestens N = 3 Diskretisierungspunkte gewählt werden, damit das Optimalsteuerungsproblem lösbar ist.

Aufgabe 1c: Lösung des Optimalsteuerungsproblems

Zur Lösung des Optimalsteuerungsproblems wurde die Methode der sequentiellen quadratischen Programmierung (SQP) verwendet. Diese Methode ist in `SciPy` über die Funktion `minimize` mit dem Argument `method='SLSQP'` implementiert.

Die Funktion `minimize` erwartet folgende wesentliche Argumente:

- **Zielfunktion:** Das diskrete Kostenfunktional $J_d(y)$, welches als Funktion `cost_function(y)` implementiert ist.
- **Startwert:** Ein initialer Vektor y_0, der hier so gewählt wurde, dass alle Zustände mit den gegebenen Anfangswerten besetzt sind.
- **Nebenbedingungen:** Die Gleichungsnebenbedingungen $g_d(y) = 0$ wurden als Python-Funktion `constraint_vector(y)` umgesetzt und der Optimierung über ein Dictionary der Form

  ```
  constraints = {'type': 'eq', 'fun': constraint_vector}
  ```

  übergeben.

- **Optionen:** Um die Konvergenz sicherzustellen, wurde die maximale Anzahl der Iterationen auf 500 erhöht:

```
options={'disp': True, 'maxiter': 500}
```

Die Methode `SLSQP` unterstützt Gleichungsnebenbedingungen und nutzt intern eine Quadratisierung der Zielfunktion sowie eine Linearisierung der Nebenbedingungen in jedem Iterationsschritt. Dadurch eignet sie sich besonders gut für nichtlineare Optimierungsprobleme mit glatten Strukturen wie im vorliegenden Fall.

Der resultierende Optimierungsvektor y^* enthält die optimalen Trajektorien für Zustände und Steuerung über die gesamte Zeitspanne.

Aufgabe 1d: Berücksichtigung von Eingangs- und Zustandsgrenzen

Im nächsten Schritt wurden zusätzlich Beschränkungen auf den Steuerverlauf $u(t)$ sowie auf die Wagenposition $s(t)$ berücksichtigt. Diese lauten:

$$-12 \leq u(t) \leq 12 \quad \text{und} \quad -0{,}8 \leq s(t) \leq 0{,}8 \quad \text{für alle } t \in [t_0, t_1].$$

Zur Umsetzung wurden für jede Diskretisierungsstufe individuelle Bounds definiert. Diese wurden in der Optimierung über das Argument `bounds` der Funktion `minimize` eingebunden. Dabei wurde folgender Aufbau verwendet:

```
bounds = [(None, None),   # theta
          (None, None),   # omega
          (-0.8, 0.8),    # s
          (None, None),   # v
          (-12, 12)] * N  # u
```

Als Startwert für die Optimierung wurde der zuvor berechnete optimale Vektor y^* aus Aufgabe 1c verwendet. Dieser Ansatz spart Rechenzeit, da der Startpunkt bereits nahe an der neuen, durch Bounds eingeschränkten Lösung liegt und somit die Anzahl der benötigten Iterationen reduziert wird.

Die Optimierung erfolgte erneut mit `method='SLSQP'` und denselben Nebenbedingungen wie zuvor. Die resultierende Lösung erfüllt nun zusätzlich die vorgegebenen Beschränkungen auf den Zustand $s(t)$ sowie die Stellgröße $u(t)$ über den gesamten Zeitraum hinweg.

Aufgabe 1e: Simulation des Systems ohne Regler

Da dieser Eingangsverlauf $u(t)$ nur an diskreten Zeitpunkten $t_0, t_1, \ldots, t_N$ gegeben ist, wurde zunächst eine kontinuierliche Näherung mittels linearer Interpolation erzeugt.

Hierzu wurde die Funktion `interp1d` aus dem Modul `scipy.interpolate` verwendet, um eine kontinuierliche Steuerfunktion zu definieren:

```
u_interp = interp1d(t, u_bounded, kind='linear')
```

Die Funktion `u_interp(t)` liefert nun für jeden beliebigen Zeitpunkt im Intervall $[t_0, t_1]$ einen interpolierten Eingangswert $u(t)$.

Diese interpolierte Funktion wurde anschließend in das dynamische System eingebettet und mithilfe von `solve_ivp` aus dem Modul `scipy.integrate` numerisch integriert.

Die Simulation kann der optimierten Trajektorie nicht exakt folgen. Insbesondere erreicht das Pendel am Endzeitpunkt nicht vollständig die instabile Ruhelage in der aufrechten Position.

Aufgabe 1f: Entwurf eines Reglers entlang der optimalen Trajektorie

Zur Regelung des Systems entlang der optimalen Trajektorie wurde ein zeitvarianter linearer Zustandsregler entworfen. Hierfür wird das System in jedem diskreten Zeitpunkt t_i entlang der optimalen Trajektorie $x^*(t), u^*(t)$ linearisiert. Es ergibt sich ein lineares, zeitabhängiges System der Form:

$$\dot{\tilde{x}}(t) = A(t)\tilde{x}(t) + B(t)\tilde{u}(t),$$

mit

$$\tilde{x}(t) = x(t) - x^*(t), \quad \tilde{u}(t) = u(t) - u^*(t).$$

Die Matrizen $A(t)$ und $B(t)$ wurden numerisch berechnet durch partielle Ableitung des nichtlinearen Systems am jeweiligen Punkt der optimalen Trajektorie. Diese Linearisierung erfolgt für jeden der N diskreten Zeitpunkte.

Um das linearisierte System optimal zu regeln, wurde ein quadratisches Kostenfunktional definiert, das die Abweichung von der Referenz minimiert:

$$J = \frac{1}{2}\tilde{x}^\top(t_1)S\tilde{x}(t_1) + \int_{t_0}^{t_1} \left(\tilde{x}^\top Q\tilde{x} + \tilde{u}^\top R\tilde{u}\right) dt.$$

Dieses Funktional bewertet sowohl den Zustand als auch die Steuerabweichung über die gesamte Zeit. Ziel ist es, diese Abweichungen möglichst gering zu halten.

Aus diesem Kostenfunktional lässt sich mittels Variation das bekannte Differentialgleichungssystem für die optimale Lösung herleiten: die matrixwertige Riccati-Differentialgleichung. Sie lautet:

$$\dot{P}(t) = -PA - A^\top P - Q + PBR^{-1}B^\top P,$$

mit der Endbedingung

$$P(t_1) = S.$$

Diese Gleichung wurde rückwärts in der Zeit integriert, ausgehend vom Endzeitpunkt t_1, mithilfe der Funktion `solve_ivp`. Die Lösung liefert für jeden Zeitpunkt eine Matrix $P(t)$, aus der dann die zeitabhängige Reglerverstärkung berechnet wurde:

$$K(t) = R^{-1}B(t)^\top P(t).$$

Das resultierende zeitabhängige Regelgesetz lautet gemäß Gleichung (9.10):

$$u(t) = u^*(t) - K(t)\left(x(t) - x^*(t)\right).$$

Dieses Regelgesetz wurde in das nichtlineare System eingebaut und simuliert. Die Simulation zeigt, dass das System mithilfe des Reglers stabil entlang der zuvor berechneten optimalen Trajektorie geführt wird. Besonders im Vergleich zur ungeregelten Simulation (Aufgabe 1e) zeigt sich, dass das Pendel unter Regelung zuverlässig die instabile Ruhelage erreicht.