Optimale Steuerung und Regelung - Übung 1

Anton Bernecker - 62100410

March 18, 2025

1 Aufgabe a: Modellierung des Systems

Gegeben ist ein harmonischer Oszillator mit der Bewegungsgleichung:

$$m\ddot{y}(t) + \omega^2 y(t) = u(t), \quad y(0) = 1, \quad \dot{y}(0) = 0. \tag{1}$$

Das System wird mit einem PID-Regler der Form:

$$u(t) = K_P y(t) + K_D \dot{y}(t) + K_I \int_0^t y(\tau) d\tau \tag{2}$$

geregelt. Ziel ist es, das System auf die Position $y(t) \to 0$ zu bringen.

Zustandsraumdarstellung

Um das System als Zustandsraumdarstellung zu formulieren, definieren wir den Zustandsvektor, mit dem vierten Zustand der Kostenfunktion:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \\ \int_0^t y(\tau) d\tau \\ \frac{1}{2} \int_0^t \left(q_1 y^2 + q_2 \dot{y}^2 + q_3 x_3^2 + r_1 u^2 \right) d\tau \end{bmatrix}. \tag{3}$$

Die Dynamik des Systems lässt sich schreiben als:

$$\dot{x} = \begin{bmatrix} x_2 \\ \frac{1}{m}(u - \omega^2 x_1) \\ x_1 \\ \frac{1}{2}(q_1 x_1^2 + q_2 x_2^2 + q_3 x_3^2 + r_1 u^2) \end{bmatrix}. \tag{4}$$

2 Aufgabe b: Kostenfunktion und Gradientenberechnung

Das Kostenfunktional wird numerisch über den Zustandsvektor berechnet und ist definiert als:

$$F(K) = \frac{1}{2} \int_0^\infty \left(q_1 y^2 + q_2 \dot{y}^2 + q_3 \left(\int_0^t y(\tau) d\tau \right)^2 + r_1 u^2 \right) dt. \tag{5}$$

Der Gradient dieser Funktion wird numerisch mittels der Funktion `approx_fprime` aus SciPy berechnet.

3 Aufgabe c: Gradientenabstiegsverfahren

Das Gradientenabstiegsverfahren wird genutzt, um die optimalen Reglerparameter K_P, K_D, K_I zu bestimmen. Die Optimierung erfolgt iterativ durch schrittweise Anpassung der Parameter entlang des Gradienten der Kostenfunktion. Dabei wird folgender Algorithmus verwendet:

- Die Parameter K_P, K_D, K_I werden initialisiert.
- In einer Schleife wird der Gradient der Kostenfunktion numerisch berechnet.
- Die Parameter werden mit einer Schrittweite α aktualisiert, welche konstant gehalten wird.
- Ein Abbruchkriterium prüft, ob die Änderung aller Parameter $\mathbf{K}$ kleiner als ein vorgegebenes Toleranzniveau ϵ ist.
- Falls das Abbruchkriterium nicht erfüllt ist, wird die Iteration fortgesetzt.
- Um eine Endlosschleife zu verhindern, wird eine maximale Anzahl an Iterationen definiert. Falls diese erreicht ist, wird die Optimierung abgebrochen.

Der Update-Schritt für die Parameter erfolgt nach der Regel:

$$K^{(n+1)} = K^{(n)} - \alpha \nabla F(K^{(n)}) \tag{6}$$

Dabei ist $\nabla F(K)$ der numerisch berechnete Gradient der Kostenfunktion und α die Schrittweite. Dieser Schritt sorgt dafür, dass die Parameter sich in Richtung des steilsten Abstiegs bewegen.

Der Algorithmus stellt sicher, dass die Parameter iterativ in Richtung des Minimums der Kostenfunktion angepasst werden, während gleichzeitig eine Überprüfung auf Konvergenz erfolgt. Dies verhindert unendlich lange Berechnungen oder numerische Instabilität.

4 Aufgabe d: Gradientenabstieg mit adaptiver Schrittweite

Zur Optimierung der Reglerparameter K_P, K_D, K_I wird ein **Gradientenabstiegsverfahren mit Backtracking-Liniensuche** verwendet. Dabei wird die Schrittweite dynamisch angepasst, um eine effiziente Konvergenz sicherzustellen.

Die Backtracking-Liniensuche stellt sicher, dass die Kostenfunktion bei jedem Schritt ausreichend reduziert wird. Der Algorithmus funktioniert wie folgt:

- Die Parameter K_P, K_D, K_I werden initialisiert.
- Der Gradient der Kostenfunktion wird numerisch berechnet.
- Eine initial große Schrittweite α wird gesetzt.
- Falls die Kostenfunktion bei $K - \alpha \nabla F(K)$ nicht genügend abnimmt, wird α mit einem Faktor $\rho \in (0, 1)$ verkleinert, bis die Bedingung erfüllt ist:
$$F(K - \alpha \nabla F(K)) \leq F(K) - c\alpha \|\nabla F(K)\|^2. \tag{7}$$
- Sobald eine gültige Schrittweite gefunden ist, werden die Parameter aktualisiert:
$$K^{(n+1)} = K^{(n)} - \alpha \nabla F(K^{(n)}). \tag{8}$$
- Der Algorithmus iteriert, bis die Änderung der Parameter unterhalb eines Toleranzwertes ϵ liegt oder die maximale Anzahl an Iterationen erreicht ist.

Dieses Verfahren ermöglicht eine **robustere und schnellere Konvergenz** als eine feste Schrittweite, da sich die Schrittgröße flexibel an die Kostenlandschaft anpasst.

5 Aufgabe e: Analytische Lösung

In dieser Aufgabe wird die **Riccati-Gleichung** genutzt, um die optimalen Reglerparameter für ein lineares System mit quadratischem Kostenmaß zu bestimmen. Das Kostenfunktional ist gegeben durch:

$$J(x,u) = \frac{1}{2}\int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t)\right) dt \tag{9}$$

wobei:

- Q die symmetrische, positiv semidefinite Zustandsgewichts-Matrix ist,
- R die positiv definite Eingangsgewichts-Matrix ist,
- $x(t)$ der Zustandsvektor ist,
- $u(t)$ der Steuervektor ist.

Die optimale Zustandsrückführung wird durch die Lösung der **algebraischen Riccati-Gleichung (ARE)** bestimmt. Die Rückführmatrix K ist gegeben durch:

$$K = R^{-1}B^T S \tag{10}$$

wobei S die Lösung der Riccati-Gleichung ist.

Die Berechnung erfolgt mit der Funktion `lqr` aus dem `control`-Modul.

6 Aufgabe f: Vergleich mit SciPy-Optimierung

Zur Validierung der Ergebnisse wird das Verfahren mit der SciPy-Optimierungsfunktio `minimize` verglichen. Dabei werden zwei verschiedene Optimierungsmethoden genutzt:

- **BFGS (Broyden-Fletcher-Goldfarb-Shanno)**: Ein quasi-Newton-Verfahren, das eine Approximation der Hesse-Matrix verwendet, um eine schnelle Konvergenz zu erreichen. Da der Gradient nicht explizit angegeben wird, berechnet `minimize` diesen numerisch.
- **Nelder-Mead**: Ein direktes, gradientenfreies Optimierungsverfahren, das auf der Simplex-Methode basiert.