PythonとCasADiで学ぶモデル予測制御

7.4 NLPソルバーの比較

7.4 概要

片側を固定したゴム紐に対するMPCを題材に，2つのNLPソルバーを実装・比較する

7.4.1 比較するNLPソルバー

- **IPOPT**
  オープンソースのNLPソルバー．主双対内点法を用いてNLP問題を解く．
  どんな問題にも汎用的に使うことができる．
- **qrsqp**
  CasADiのチームによって開発．CasADi用のNLPソルバー．
  逐次二次計画法を用いてNLP問題を解く．
  適切な初期値が与えられている場合は，主双対内点法よりも高速な場合がある．

7.4.2 ゴム紐の制御

複数のNLPソルバーの比較を行う場合...
一般的にはベンチマークと呼ばれる共通の問題を解く時間を計ることで比較する.

> 5章の倒立振子や本節のゴム紐の制御もベンチマーク問題の一つである

7.4.2 ゴム紐の制御

ゴム紐問題

- ゴム紐は複数の質点がばねでつながった物理モデルとして表現
- ゴム紐の一方の端が原点に固定．もう一方の端は自由．

⇒ 制御の目的は，自由な方の端を制御してゴム紐を静止させること．

7.4.3 ゴム紐の状態方程式

ゴム紐の状態方程式の定式化を行う.

- 質点の質量： m
- ばね定数： k
- ばねの自然長： l
- ばねの個数： M
- 質点の個数： $M+1$
- 1番左側の質点（i=0）の位置は原点で固定
- 左から2番目以降の質点iの位置を$p_i = [x_i, y_i, z_i]^{\mathrm{T}} (i = 1, \cdots, M)$
- 重力加速度をgとし，重力を$g_z = [0, 0, -g]^{\mathrm{T}}$

7.4.3 ゴム紐の状態方程式

質点iから質点$i+1$へはたらく力を

$$F_{i,i+1} = k(\|\mathbf{p}_{i+1} - \mathbf{p}_i\| - l)\frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|} \quad (i = 1, \cdots, M-1) \quad (7.20)$$

とする.

力の大きさ

$$k(\|\mathbf{p}_{i+1} - \mathbf{p}_i\| - l)$$

力の向き（ベクトルの正規化）

$$\frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|}$$

7.4.3 ゴム紐の状態方程式

このとき，制御入力$\mathbf{u}$はゴム紐の最も右側の質点（$i = M$）を自由に動かすことであるとする．
運動方程式は次式で表される．

$$\begin{cases} \dot{p}_M = u \\ \ddot{p}_i = \frac{1}{m}(F_{i,i+1} - F_{i-1,i}) + g_z \quad (i = 1, \cdots, M-1) \end{cases} \tag{7.21}$$

ここで，状態変数を $\mathbf{x} = \left[p_1, \cdots, p_{M-1}, p_M, \dot{p}_1, \cdots, \dot{p}_{M-1}\right]^{\mathrm{T}}$ ，制御入力を $\mathbf{u} = \left[u\right]^{\mathrm{T}}$とする．

7.4.3 ゴム紐の状態方程式

その場合，式(7.21)を整理して，次式の状態方程式が求まる．

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) = \begin{bmatrix} \dot{p}_1 \\ \vdots \\ \dot{p}_{M-1} \\ u \\ \frac{1}{m}(F_{0,1} - F_{1,2}) + g_z \\ \vdots \\ \frac{1}{m}(F_{M-1,M} - F_{M-2,M-1}) + g_z \end{bmatrix} \tag{7.22}$$

7.4.3 ゴム紐の状態方程式

各種パラメータの値を以下のように設定.

- **状態方程式の各定数**
  $m = 0.1125, k = 2, l = 0.1375, g = 9.81$ (7.23)
- **状態変数の初期値** (x軸方向に等間隔で一直線に配置されているものとする)

$$\begin{cases} p_i = \left[\frac{8}{M+1-i}, 0, 0\right]^{\mathrm{T}} & (1 \leq i \leq M) \\ \dot{p}_i = [0, 8, 0]^{\mathrm{T}} & (i = 3) \\ \dot{p}_i = [0, 0, 0]^{\mathrm{T}} & (i \neq 3) \end{cases} \tag{7.24}$$

- **状態変数の目標値**

$$\begin{cases} p_M = [8, 0, 0]^{\mathrm{T}} \\ p_i = [0, 0, 0]^{\mathrm{T}} & (1 \leq i \leq M-1) \end{cases} \tag{7.25}$$

7.4.3 ゴム紐の状態方程式

- 制御入力の目標値

$$\mathbf{u}_{\mathrm{ref}} = [0, 0, 0]^{\mathrm{T}} \tag{7.26}$$

- 状態変数と制御入力の範囲

$$\begin{cases} \bar{\mathbf{x}}_{\mathrm{lb}} = -\infty, \quad \bar{\mathbf{x}}_{\mathrm{ub}} = \infty \\ \bar{\mathbf{u}}_{\mathrm{lb}} = [-5, -5, -5]^{\mathrm{T}}, \quad \bar{\mathbf{u}}_{\mathrm{ub}} = [5, 5, 5]^{\mathrm{T}} \end{cases} \tag{7.27}$$

- ホライゾン長，分割数，時間間隔

$$T = 2, \quad K = 20, \quad \Delta t = 0.1 \tag{7.29}$$

7.4.4 準備

制御を行わない場合の確認

図（a）は原点の右隣（$i = 1$）の質点p_1の位置
図（b）は制御入力を各時刻でプロットしたもの

7.4.5 MPCの計算時間の比較

〔1〕IPOPTによるMPC

7.4.5 MPCの計算時間の比較

〔2〕qrsqpの場合

7.4.5 MPCの計算時間の比較

IPOPTの1ステップあたりの平均計算時間： 約0.08s
qrsqpの1ステップあたりの平均計算時間： 約0.07s

⇒ 今回の場合ではqrsqpのほうがIPOPTより約10ms速い

各時刻でのソルバーの実行時間の比較

7.4.6 MPCの計算時間の比較（適切な初期値なし）

初期値をランダムで用意した場合のIPOPTとqrsqpで解いた場合の比較

〔1〕IPOPTの場合

特にはじめの数ステップの計算時間は，適切な初期値がある場合と比べて数倍かかっていることが確認できる．

7.4.6 MPCの計算時間の比較（適切な初期値なし）

〔2〕qrsqpの場合

最初のステップでの最適化計算で，ソルバーオブジェクトがエラーを出力．

qrsqpが最初のステップで最適化を終了できなかったのは，qrsqpの逐次二次計画法としての特性によるもの．
逐次二次計画法には，適切な初期値が与えられない場合には最適化を行うことが困難であるという特性がある．（付録 A.3.3）

7.4.7 まとめ

- 2つのNLPソルバーの比較を行った．
- IPOPTは初期値をランダムに与えても最適化を行うことができ，安定動作．
- qrsqpは初期値をランダムに与えた今回の場合では，エラーを生じてしまった．

初めはIPOPTで実装するのが吉