docs_jp

MPC-MATLAB-Examples ドキュメント

著者: youkoutaku

リポジトリ: youkoutaku/MPC-MATLAB-Examples

日付: 2025年12月5日

システムモデル

対象システム：二重積分系

連続時間モデル：

$$\begin{aligned} \dot{x}_1 &= x_2 \quad \text{（位置の微分 = 速度）} \\ \dot{x}_2 &= u \quad \text{（速度の微分 = 加速度）} \end{aligned}$$

状態空間表現：

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

離散化（サンプリング時間 ts = 0.05〜0.1秒）：

```
sys_d = c2d(ss(A, B, C, D), ts);
Ad = sys_d.A;  % 離散化システム行列
Bd = sys_d.B;  % 離散化入力行列
```

ファイル構成

```
MPC-MATLAB-Examples/
├── README.md                          # README (日本語)
├── README_EN.md                       # README (English)
```

```
├── docs/
│   ├── docs.pdf                    # 詳細ドキュメント（English）
│   ├── docs_jp.pdf                 # 詳細ドキュメント（日本語）
│   ├── docs.md                     # 詳細ドキュメント（English）
│   └── docs_jp.md                  # 詳細ドキュメント（日本語）
├── LICENSE                         # MITライセンス
├── lib/                            # 補助関数・ライブラリ
│   ├── QP_Transform.m              # QP標準形変換関数
│   └── Augmented_System_Input.m    # 拡大系構築（レガシー）
├── test/                           # テストスクリプト
│   ├── Check_Constraints.m         # 制約検証ツール
│   └── Check_QP_Transform.m        # QP変換テスト
├── MPC_trajectory_main.m           # 軌道追従MPC（推奨）
├── MPC_con_main.m                  # 制約付きMPC
├── MPCvsLQR_QP_main.m              # MPC vs LQR比較
└── MPC_Toolbox.m                   # ツールボックス検証
```

1. メインシミュレーションファイル

MPC_trajectory_main.m 推奨

目的： 時変参照軌道追従のための基本的なMPC実装

理論：

- 予測区間

$$= 0$$

- コスト関数：

$$= {}_{1} \; {}_{=1} x(\;) \; x(\;)_2 \quad {}_{=1} u(\;)_2 \; x(\;) \; x(\;)_2$$

- 参照軌道：正弦波

$$x = \begin{bmatrix} \sin(0) \; 10 \\ 2\cos(0) \end{bmatrix}$$

- QP定式化：

$$\text{in} \; 0$$

使用方法：

```
run('MPC_trajectory_main.m')
```

MPC_con_main.m 制約付きMPC

目的： 入力・状態・変化率制約を考慮したMPC

理論：

- 制約条件：
  - 入力制約： u ， m/s²
  - 速度制約： $10, 10$ m/s
  - 位置制約： x $0, 20$ m
  - 変化率制約： u ， m/s³
- 制約付きQP：

$$\text{in} \quad \text{st} \quad A_{\text{ine}} \quad _{\text{ine}}$$

- 制約処理：累積和行列 $_{\text{s}}$ による入力制約、選択行列 C_x による状態制約

使用方法：

```
run('MPC_con_main.m')
```

検証ツール：

```
run('test/Check_Constraints.m')  % 制約充足性の詳細検証
```

MPCvsLQR_QP_main.m 性能比較

目的： MPCとLQRの制御性能比較

理論：

- MPC：予測制御による前向き最適化
- LQR：無限区間最適レギュレータ $u = (x\ \ x)$
- 同一重み行列で公平比較
- 入力制約： m/s² 飽和処理

使用方法：

```
run('MPCvsLQR_QP_main.m')
```

MPC_Toolbox.m ツールボックス検証

目的： MATLABツールボックスとの実装検証

理論：

- MATLAB MPC Toolbox の mpc() オブジェクト使用
- 重み設定、予測区間、制約を手動実装と同一に設定
- 手動QP実装との結果一致を確認

使用方法：

```
run('MPC_Toolbox.m')
% 注意：MATLAB MPC Toolboxが必要
```

---

2. 補助関数・ツール

QP_Transform.m QP変換

目的： MPCコスト関数をQP標準形に変換

理論：

入力：

$$A, B, , , ,$$

出力：

$$A, B, , , ,$$

予測モデル：

$$= A \ x() \ B$$

コスト関数変換：

$$\begin{aligned} &= (Ax \ B \ )(Ax \ B \ ) \\ &= 0 \quad \text{const} \end{aligned}$$

ここで：

- 状態予測行列:

$$A = \begin{matrix} A \\ A^2 \\ \\ A \end{matrix}$$

- B = 下三角構造の制御予測行列
- $= \text{dia}(, , ,)$
- $= \text{dia}(, , ,)$
- $= BA$
- $= BB$

使用方法：

```
% libフォルダにパスを追加
addpath('lib')
[Sx, Su, Qp, Rp, F, H] = QP_Transform(A, B, Q, R, Qf, Np);
```

Check_Constraints.m 制約検証ツール

場所： `test/Check_Constraints.m`
目的： 制約付きMPCの制約充足性を詳細検証

機能：

- 全ステップでの制約違反チェック（許容誤差 1e-6）
- 入力、速度、位置、変化率の制約充足統計
- 制約到達回数のカウント
- 3つの検証プロット生成

使用方法：

```
cd test
run('Check_Constraints.m')
% または
run('test/Check_Constraints.m')
```

出力：

- 制約違反の詳細レポート（ステップ番号、値、違反量）
- 制約統計サマリー
- 検証用3プロット（入力・速度・変化率 vs 制約）

Augmented_System_Input.m レガシー

場所： `lib/Augmented_System_Input.m` **目的：** 入力を含む拡大系の構築（目標値のダイナミクスまで考慮する）

注意： 現在の実装では使用していません。参考用に保存。

Check_QP_Transform.m 単体テスト

場所： `test/Check_QP_Transform.m` **目的：** QP_Transform関数の単体テスト

使用方法

基本的な実行手順

重要： 最初にlibフォルダをパスに追加してください：

```
addpath('lib')  % 補助関数をパスに追加
```

1. **軌道追従MPC：**

```
run('MPC_trajectory_main.m')
```

2. **制約付きMPC：**

```
run('MPC_con_main.m')
run('test/Check_Constraints.m')  % 制約検証
```

3. **性能比較：**

```
run('MPCvsLQR_QP_main.m')  % MPC vs LQR
run('MPC_Toolbox.m')         % 手動実装 vs ツールボックス
```

パラメータ調整

各ファイルで調整可能なパラメータ：

```
% 予測区間
Np = 30;  % 短い → 計算速度向上、長い → 制御性能向上

% 重み行列
Q_mpc = [1 0; 0 0.2];    % 状態追従重み（大 → 追従精度向上）
R_mpc = 10;              % 制御入力重み（大 → 滑らかな入力）

% サンプリング時間
ts = 0.05;  % 小さい → 精度向上、大きい → 計算負荷低減

% 制約（MPC_con_main.m のみ）
u_min = -5; u_max = 5;          % 入力制約
x2_min = -10; x2_max = 10;      % 速度制約
x1_min = 0; x1_max = 20;        % 位置制約
du_min = -3; du_max = 3;        % 変化率制約
```

理論背景

1. MPC の基本原理

予測モデル：

$$x(\ 1) = A\ x()\ B\ u()$$

予測区間での状態：

$$\begin{matrix} x(\ 1) \\ = x(\ 2) \end{matrix}$$

$$\begin{matrix} x(\ \ ) \\ = {}_{x}\ x()\ {}_{u}\ \ {}_{u}\ u(\ 1) \end{matrix}$$

コスト関数：

$$= {}_{1}\ {}_{=1} x(\ \ )\ x(\ \ )_2\ \ {}_{=1} u(\ \ )_2\ x(\ \ )\ x(\ \ )_2$$

最小化：

$$\text{in}\,()$$

最適化：

- 二次計画法（QP）による解法
- 制約なし：解析解 $= {}^{1}$
- 制約付き：`quadprog()` による数値解

---

2. 制約付きMPCの定式化

制約の種類：

1. **変化率制約：** $u_{\text{in}}\ \ u(\ \ )\ \ u_{\text{a}}$

$$A_u = \begin{bmatrix} \ \ \end{bmatrix},\quad {}_{u} = \begin{bmatrix} u_{\text{a}} \\ u_{\text{in}} \end{bmatrix}$$

2. **入力制約：** $u_{\text{in}} \quad u(\ 1) \quad u \quad u_{\text{a}}$

$$A_u = \begin{bmatrix} \text{s} \\ \text{s} \end{bmatrix}, \quad _u = \begin{bmatrix} u_{\text{a}} & u(\ 1) \\ & \end{bmatrix}$$

$$_{\text{s}} = \text{tril}() \quad \text{（累積和行列）}$$

3. **状態制約：** $x_{\text{in}} \quad x(\ ) \quad x_{\text{a}}$

$$A_x = \begin{bmatrix} C_{xu} \\ C_{xu} \end{bmatrix}$$

$C_x =$ 状態選択行列（位置・速度を個別に選択）

統合QP問題：

$$\begin{array}{ll} \text{in} & 0 \\ \text{st} & A_{\text{ine}} \quad _{\text{ine}} \end{array}$$

ここで：

$$A_{\text{ine}} = \begin{array}{l} A_u \\ A_u \\ A_x \end{array} \quad \text{（全制約を統合）}$$

---

3. MPCとLQRの比較

特性	MPC	LQR
最適化区間	有限区間（）	無限区間
制約対応	直接対応可能	困難（飽和処理）
計算負荷	高（オンラインQP）	低（事前計算ゲイン）
予測能力	将来の参照を考慮	現在の誤差のみ
最適性	局所最適	大域最適（線形系）

使い分け：

- 制約が重要 → MPC
- 高速制御が必要 → LQR
- 複雑な軌道追従 → MPC

重要な注意事項

1. 離散時間システムと連続時間システム

離散化の重要性：

連続時間システムを直接使用すると誤差が発生します。必ず `c2d()` で離散化してください。

正しい実装：

```
% システム定義（連続時間）
A_c = [0 1; 0 0];
B_c = [0; 1];

% 離散化
sys_d = c2d(ss(A_c, B_c, C, D), ts);
A = sys_d.A;  % これを使用
B = sys_d.B;  % これを使用

% シミュレーション
for k = 1:N
    u(k) = ...  % MPC or LQR
    x(k+1) = A * x(k) + B * u(k);
end
```

誤った実装（避けるべき）：

```
% 前進差分近似（精度が低い）
x(k+1) = x(k) + ts * (A * x(k) + B * u(k));
```

---

2. 最適ゲインの計算

推奨：離散系で直接計算

```
sys_d = c2d(sys_c, ts);
[Sx, Su, Qp, Rp, F, H] = QP_Transform(sys_d.A, sys_d.B, Q, R, Qf, Np);
```

```
[K, S, e] = dlqr(sys_d.A, sys_d.B, Q, R);  % LQR
```

避けるべき：連続系で計算後に変換

```
% 連続系LQR → 離散化は非推奨
[K_c, S_c, e_c] = lqr(A_c, B_c, Q, R);
% K_c を離散系に変換する処理が複雑で誤差の原因
```

---

3. 初期状態と制約の整合性

制約付きMPCでは、初期状態が制約を満たす必要があります：

```
% 制約
x2_min = -3; x2_max = 3;  % 速度制約

% 良い初期状態
x0 = [10; 2];  % 速度 2 m/s（制約内）

% 悪い初期状態
x0 = [10; 5];  % 速度 5 m/s（制約違反！） → QP不可解になる
```

---

4. 予測区間の選択

短い予測区間：

- ✓ 計算が速い
- ✓ リアルタイム制御に適する
- ✗ 制御性能がやや低下

長い予測区間：

- ✓ 制御性能が向上
- ✓ 制約を先読みして回避
- ✗ 計算時間が増加 ()

---

5. quadprog収束失敗への対処

`exitflag` 1 の場合の処理：

```
if exitflag ~= 1
    warning('QP failed at step %d', k);

    % 対処法1：重み行列調整
    初期誤差が大きい場合に，終端コストの重みと入力重みを上げる

    % 対処法2：制約を緩和
    % 状態制約の範囲を広げる

    % 対処法3：予測区間を増やす/減らす
    % Np を調整して再試行
end
```