CasADi を用いた非線形最適化の手順について

動的システム・ロボティクス研究室　B3　三輪 雄太

2022 年 1 月 24 日

1　前回までのまとめと今月の概要

前回までは体内グルコースモデルの近似, 簡略化を行っていた. 今月は, 非線形 MPC の実装にむけ, CasADi を用いた MPC の実装を行なった.

2　MPC とは

MPC(Model Predictive Control) とは, 「各時刻において時間有限の最適化制御問題を解き, その時刻での最適な制御入力を決定し, システムに加える」ということを時刻が進むたびに繰り返すことで実現されるフィードバック制御のことである.

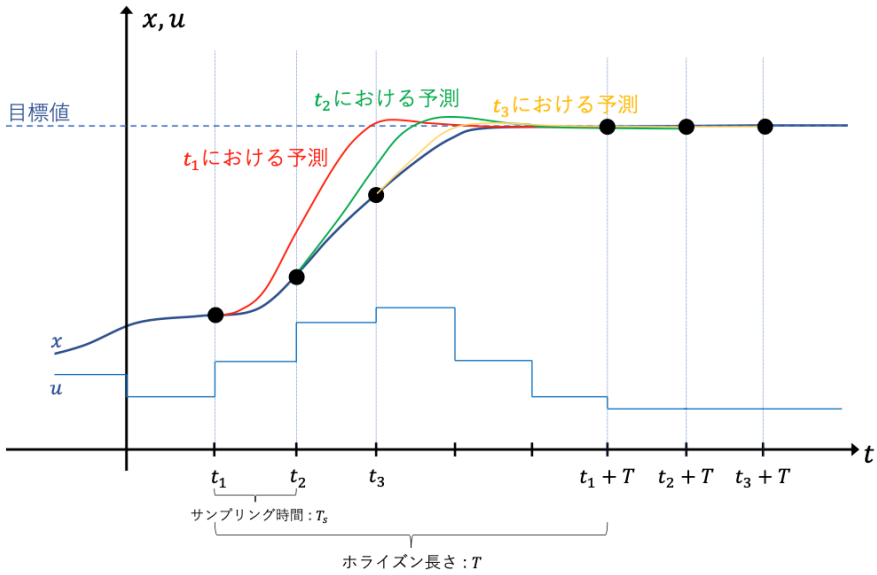


Fig.2.1: MPC のイメージ

制御対象として, 以下のような離散時間のシステムを考える.

$$x(t_{n+1}) = f(x(t_n), u(t_n), t_n), \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m, \quad t_{n+1} = t_n + T_s \tag{2.1}$$

ここで, t_n はサンプリング時間 T_s で離散化された時刻, $x(t_n)$ は状態を表すベクトル, $u(t_n)$ は制御入力を表すベクトルである.

まず, 時刻 t_n における最適制御問題について考える. 評価区間は $T = N \times T_s$ とする. 時刻 t_n における状態 $x(t_n)$ が分かっており, なんらかの制御入力の系列 $\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ を決めれば, 差分方程式 (2.1) より状態の系列 $\{\bar{x}(t_k)\}_{k=n}^{n+N}$, すなわち t_n から $t_n + T$ までの応答が求まる ($\bar{x}, \bar{u}$ の上線は, 予測値であり実際の状態と入力とは異なるという意味で用いている. ただし, $\bar{x}_n = x_n$). ここで, “期待する応答と予測された応答との差”を表す評価関数 J を次のようなスカラー値関数として定義すると,

$$J = L_f(\bar{x}(t_n + T)) + \sum_{k=n}^{n+N-1} L(\bar{x}(t_k), \bar{u}(t_k), t_k)) \tag{2.2}$$

これに $\{\bar{x}(t_k)\}_{k=n}^{n+N}, \{\bar{u}(t_n)\}_{k=n}^{n+N-1}$ を代入することで応答の良さを数値として求めることができる．よりよい応答を求めるには, J がより小さくなる制御入力を探せば良く, J を最小化する制御入力の系列 $\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ が分かれば, それが最適な制御入力となる．

つまり, 最適制御問題は次のような最適化問題へと落とし込むことができる．

$$\underset{\bar{x}(\cdot),\bar{u}(\cdot)}{\text{minimize}} \qquad L_f(x(t_n+T)) + \sum_{k=n}^{n+N-1} L(x(t_k), u(t_k), t_k)) \tag{2.3a}$$

$$\text{subject to} \qquad \begin{cases} g(x,u) = \begin{bmatrix} \bar{x}(t_{n+1}) - f(\bar{x}(t_n), \bar{u}(t_n), t_n) \\ \bar{x}(t_{n+2}) - f(\bar{x}(t_{n+1}), \bar{u}(t_{n+1}), t_{n+1}) \\ \vdots \\ \bar{x}(t_{n+N}) - f(\bar{x}(t_{n+N-1}), \bar{u}(t_{n+N-1}), t_{n+N-1}) \end{bmatrix} = 0 \\ \bar{x}_n = x_n \end{cases} \tag{2.3b}$$

先ほどの説明では, 状態の系列 $\{\bar{x}(t_k)\}_{k=n}^{n+N}$ が, 入力の系列 $\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ によって決まり, 未知変数は $\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ のみであるという捉え方をしていたが, (2.3) では $\{\bar{x}(t_k)\}_{k=n}^{n+N}$,$\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ がそれぞれ未知変数であり, 制約条件として各時刻における差分方程式 (2.1) が存在するという捉え方をしている．

純粋な最適制御は, こうして求まった $\{\bar{u}(t_k)\}_{k=n}^{n+N-1}$ を評価区間 t_n〜t_n+T にわたって制御対象に入力する．この場合はフィードフォワード制御となるため, 現実には必ず誤差が生じる．そこで, MPC では求まった制御入力の系列の一番最初の制御入力 $\bar{u}_n$ のみをシステムに加える．次ステップでも同じ最適制御問題を解き, 求まった制御入力の系列の一番最初の制御入力のみをシステムに加える．これを繰り返すことにより, ある時刻における入力は常にその時刻の状態を用いて計算されることになり, フィードバック制御が可能となる．

3 CasADi を用いた非線形 MPC の実装

3.1 CasADi とは

CasADi とは, 数値最適化用のソルバーを扱うためのインターフェースツールである．これを用いることで, 簡単に最適化計算を実装することができる．MATLAB と同様, シンボリックな値を用いることができるため, 直感的な定式化が可能である．今回は, 非線形最適化ソルバーの IPOPT を用いた非線形 MPC を mayataka 氏のサンプルコード [1] を元に MATLAB 言語で実装した．

3.2 CasADi による最適制御問題の定式化

ここに, CasADi による最適制御問題の定式化の中でおおよそやっていることを示す．離散化グリット数 N, サンプリング時間 Ts とし, 制御対象として, 以下のような連続時間のシステムが与えられているとする．

$$\dot{x}(t) = g(x(t), u(t)), \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m \tag{3.1}$$

①システムの離散化

$$x_{k+1} = f(x_k, u_k) = x_k + Ts \cdot g(x_k, u_k) \tag{3.2}$$

②最適化変数の定義

各時刻における状態変数, 入力変数を全てシンボリック変数として定義する．

$$w_x = [x_0, x_1, \ldots, x_N]^{\mathrm{T}} \in \mathbb{R}^{n\times(N+1)} \tag{3.3}$$

$$w_u = [u_0, u_1, \ldots, u_{N-1}]^{\mathrm{T}} \in \mathbb{R}^{m\times N} \tag{3.4}$$

③評価関数の定義

ここでは下記のように評価関数を定義している. Q, R はそれぞれ状態, 入力への重みである.

$$J(w_x, w_u) = \frac{1}{2} w_x^{\mathrm{T}} Q w_x + \frac{1}{2} w_u^{\mathrm{T}} Q w_u \tag{3.5}$$

$$Q = \text{block-diag}[\mathrm{Q}_0, \mathrm{Q}_1, \ldots, \mathrm{Q}_{\mathrm{N}}] \tag{3.6}$$

$$R = \text{block-diag}[\mathrm{R}_0, \mathrm{R}_1, \ldots, \mathrm{R}_{\mathrm{N}-1}] \tag{3.7}$$

④等式制約 (差分方程式による) の定義

$$g(w_x, w_u) = \begin{bmatrix} x_1 - f(x_0, u_0) \\ \vdots \\ x_N - f(x_{N-1}, u_{N-1}) \end{bmatrix} = 0 \tag{3.8}$$

以上に各変数の制約を加えて最適化問題を定式化し, ラグランジュ乗数ならびに各変数の初期推定解とともに IPOPT に渡して, 最適化問題を解いている.

3.3 コード

以下に実装したコードを示す. システムのダイナミクスを記述した部分は省略している.

Code 1: comfig.m

```
% シミュレーション
t_end     = 20;                 % シミュレーション時間
Ts_sim    = 0.001;              % サンプリング周期
nx        = 4;                  % 状態変数の次元
nu        = 1;                  % 制御入力の次元
x0        = [0;0;0;0];          % の初期条件x
u0        = [0];                % の初期条件u

% 最適化問題の設定
N           = 20;                                          % ホライゾン離散化グリッド数
Ts          = 0.05;                                        % サンプリング時間
x_ref       = [0.0; pi; 0.0; 0.0];                         % 目標値
x_lim       = [];                                          % 状態の制約
u_lim       = [15.0, -15.0];                               % 制御入力の制約
Q_stage     = Ts*diag([2.5, 10.0, 0.01, 0.01]);            % 状態への重みステージコスト()
R_stage     = Ts*diag(0.1);                                % 制御入力への重みステージコスト()
Q_tarminal = diag([2.5, 10.0, 0.01, 0.01]);                % 状態への重み終端コスト()


% Q,R,の行列を作成x_ref
Q = [];
R = [];
for i = 0:N-1
    Q = blkdiag(Q,Q_stage)
end
Q = blkdiag(Q,Q_tarminal)

for i = 0:N-1
    R = blkdiag(R,R_stage)
end

x_ref = repmat(x_ref , N+1, 1)
```

Code 2: main.m

```
import casadi.*

clear;
close all;
clc;

%% MPC Configuration
run('config.m');
mpc = MPC_Miwa(@mpc_dynamics , Ts, N, u_lim , x_lim , x_ref , Q, R, nx, nu);
mpc = mpc.NLP('forwardEular');    % forwardEular で離散化して非線形問題を定式化

%% Simulation
xs = [];
xt = [];
us = [];
x = x0;
```

```

for t = 0:Ts_sim:t_end
    disp(['t = ', num2str(t)])
    mpc = mpc.compute_u(x);
    u = mpc.u;

    [time, x_hist] = ode45(@(t,x)system_dynamics(t, x, u), [t, t+Ts_sim], x);
    x = x_hist(end,:)';
    xs = [xs; x_hist];
    xt = [xt; time];
    us = [us, u];

    % x1 = x + Ts_sim * cartpole.dynamics(x,u);
    % x  = x1;
    % xs = [xs; x1'];
    % xt = [xt; t];
end

%% Plot
xs1 = xs(:,1);
xs2 = xs(:,2);
xs3 = xs(:,3);
xs4 = xs(:,4);
tgrid = 0:Ts_sim:t_end;

figure
hold on
grid on
plot(xt, xs1)
plot(xt, xs2)
plot(xt, xs3)
plot(xt, xs4)
stairs(tgrid, us)
legend('y(x1)','th(x2)', 'dy(x3)', 'dth(x4)','u');
```

Code 3: Class MPC_Miwa.m

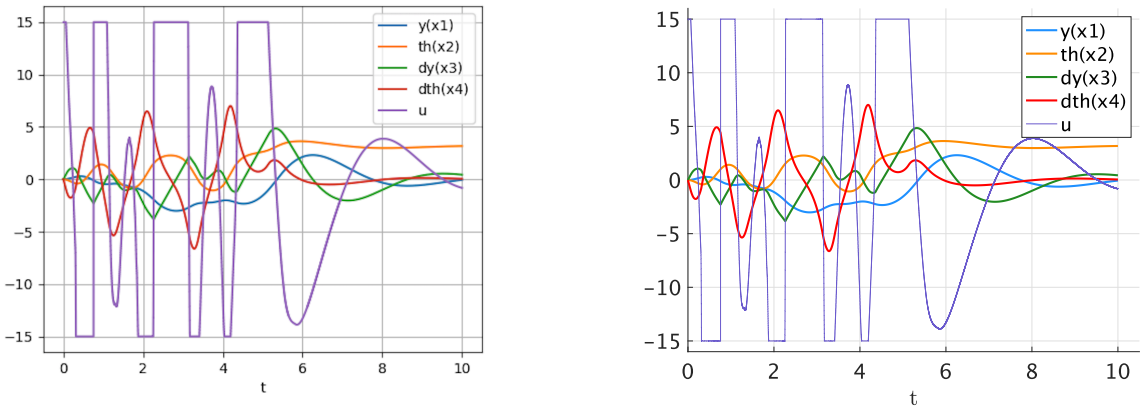
```
classdef MPC_Miwa
    properties
        J, w, g, x, lam_x, lam_g, lbx, ubx, lbg, ubg, nlp, solver, N, Ts,  ...
        x_dot, X, U, nx, nu, T, x0, u0, u_lim, x_lim, x_ref, Q, R, u, dynamics;
    end

    methods
        function self = MPC_Miwa(mpc_dynamics, Ts, N, u_lim, x_lim, x_ref, Q, R, nx, nu)
            self.dynamics = mpc_dynamics;
            self.nx = nx;
            self.nu = nu;
            self.N = N;
            self.Ts = Ts;
            self.T = Ts*N;
            self.u_lim = u_lim;
            self.x_lim = x_lim;
            self.x_ref = x_ref;
            self.Q = Q;
            self.R = R;
            self.u = [];
        end

        function self = NLP(self, method)
            % 非線形問題の定式化
            import casadi.*

            dynamics = self.dynamics;
            N = self.N;
            Ts = self.Ts;
            T = self.T;
            u_lim = self.u_lim;
            x_lim = self.x_lim;
            x_ref = self.x_ref;
            Q = self.Q;
            R = self.R;
            nx = self.nx;
            nu = self.nu;
            x_dot = self.x_dot;
            X = self.X;
            U = self.U;

            % 制約がない場合は x_lim,u_lim に Inf, -Inf を代入
            if isempty(x_lim)
                x_lim = [Inf(nx, 1), -Inf(nx, 1)];
            end

            if isempty(u_lim)
                u_lim = [Inf(nx, 1), -Inf(nx, 1)];
            end
```

```
50
51              w_x       = [];
52              w_u       = [];
53              w0_x      = [];
54              w0_u      = [];
55              lbw_x     = [];
56              lbw_u     = [];
57              ubw_x     = [];
58              ubw_u     = [];
59              lam_x0_x  = [];
60              lam_x0_u  = [];
61              g         = [];
62              ubg       = [];
63              lbg       = [];
64              lam_g0    = [];
65
66              Xk = SX.sym('X_0', nx);                 % 初期時刻の状態ベクトル x0
67              w_x       = [w_x; Xk];                  % x0 を最適化変数配列 (w_x)に追加
68              ubw_x     = [ubw_x; x_lim(:,1)];        % x0 の upper bound
69              lbw_x     = [lbw_x; x_lim(:,2)];        % x0 の lower bound
70              w0_x      = [w0_x; zeros(nx,1)];        % x0 の初期推定解
71              lam_x0_x = [lam_x0_x; zeros(nx,1)];     % ラグランジュ乗数の初期推定解
72
73              % 離散化ステージ 0~N-1 までの変数と制約を設定
74              for k = 0:N-1
75                  Uk = SX.sym(['U_', num2str(k)]);        % 時間ステージ k の制御入力 uk を表す変数
76                  w_u       = [w_u; Uk];                  % uk を最適化変数配列(w_u)に追加
77                  ubw_u     = [ubw_u; u_lim(1)];          % uk の upper-bound
78                  lbw_u     = [lbw_u; u_lim(2)];          % uk の lower-bound
79                  w0_u      = [w0_u; zeros(nu,1)];        % uk の初期推定解
80                  lam_x0_u = [lam_x0_u; zeros(nu,1)];     % ラグランジュ乗数の初期推定解
81
82                  Xk1 = SX.sym(['X_', num2str(k+1)], nx); % 時間ステージ k+1 の状態 xk+1 を表す変数
83                  w_x       = [w_x; Xk1];                 % xk+1 を最適化変数配列(w_x)に追加
84                  ubw_x     = [ubw_x; x_lim(:,1)];        % xk+1 の lower-bound
85                  lbw_x     = [lbw_x; x_lim(:,2)];        % xk+1 の upper-bound
86                  w0_x      = [w0_x;  zeros(nx,1)];       % xk+1 の初期推定解
87                  lam_x0_x = [lam_x0_x; zeros(nx,1)];     % ラグランジュ乗数の初期推定解
88
89                  % Forward Euler による離散化状態方程式
90                  if method == 'forwardEular'
91                      Xk_next = Xk + dynamics(Xk, Uk) .* Ts;
92                  end
93
94                  % 状態方程式(xk+1=xk+fk*dt) を等式制約として導入
95                  g       = [g; Xk_next-Xk1];
96                  ubg     = [ubg; zeros(nx,1)];       % 等式制約は lower-bound と upper-bound を同じ値にすることで設定
97                  lbg     = [lbg; zeros(nx,1)];       % 等式制約は lower-bound と upper-bound を同じ値にすることで設定
98                  lam_g0 = [lam_g0; zeros(nx,1)];     % ラグランジュ乗数の初期推定解
99                  Xk      = Xk1;
100             end
101
102             % 評価関数
103             J = 0.5 * (((w_x-x_ref)' * Q * (w_x-x_ref)) +  (w_u' * R * w_u));
104
105             self.J = J;
106             self.w = vertcat(w_x,w_u);
107             self.g = vertcat(g{:});
108             self.x = vertcat(w0_x,w0_u);
109             self.lam_x = vertcat(lam_x0_x, lam_x0_u);
110             self.lam_g = lam_g0;
111             self.ubx = vertcat(ubw_x,ubw_u);
112             self.lbx = vertcat(lbw_x,lbw_u);
113             self.ubg = ubg;
114             self.lbg = lbg;
115
116             % 非線形計画問題(NLP)
117             self.nlp = struct('f',self.J,'x',self.w,'g',self.g);
118             % Ipopt ソルバー，最小バリアパラメータを，最大反復回数を0.15， ウォームスタートをにON
119             self.solver = nlpsol('solver', 'ipopt', ...
120                         self.nlp, struct('calc_lam_p', true, 'calc_lam_x', true, 'print_time', false, ...
121                         'ipopt', struct('max_iter', 5, 'mu_min', 0.1, 'warm_start_init_point', 'yes', ...
122                         'print_level', 0, 'print_timing_statistics', 'no')));
123         end
124
125         function  self = compute_u(self, x0)
126             % 初期状態についての制約を設定
127             nx = size(x0, 1);
128             self.lbx(1:nx) = x0;
129             self.ubx(1:nx) = x0;
130
131             % primal variables (x) と dual variables ( ラグランジュ乗数 ) の初期推定解も与えつつ solve (warm start)
132             sol = self.solver('x0', self.x, 'lbx', self.lbx, 'ubx', self.ubx, 'lbg', self.lbg, ...
133                               'ubg', self.ubg, 'lam_x0', self.lam_x, 'lam_g0', self.lam_g);
134
135             % SX 型から double 型へ変更
136             self.x      = full(sol.x);
```

```
            self.lam_x = full(sol.lam_x);
            self.lam_g = full(sol.lam_g);
            self.u     = full(sol.x(self.nx*(self.N+1)+1 : self.nx*(self.N+1)+self.nu));
        end
    end
end
```

3.4 Example

サンプルコードの CartPole のシステムを, 今回実装した MPC においてもシミュレーションした. 方程式は以下の URL を参照 (https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-832-underactuated-robotics-spring-2009/readings/MIT6_832s09_read_ch03.pdf). 無事に実装できた.



(a) mayataka 氏の MPC による実行結果

(b) 今回作った MPC による実行結果

Fig.3.1: CartPole の MPC

4 まとめ

これまではグルコースインスリンモデルの簡略化を進めてきたが, 実際に非線形 MPC への適用を考える段階に入った. 実際に動いたことは良かったが, 分からないことがとても増えた. 特に最適化の理論など, きちんと勉強しつつ進めていきたい. また, 実際に糖尿病モデルにも適用し, グルコース動態の非線形 MPC についても随時進めていきたい.

参考文献

[1] mayataka. How to use casadi, 2021. https://gist.github.com/mayataka/ef178130d52b5b06d4dd8bb2c8384c54.