CasADi vs. MPCTools

December 22, 2020

Below, we present an example file to show how much code is saved by using MPCTools. On the left side, we show the the script written using the pure CasADi module, while on the right, we show the script rewritten to use MPCTools.

```
% VDP Oscillator using native Casadi.
Delta = 0.5;
Nt = 20; % Controller horizon.



% Declare model variables
x1 = casadi.SX.sym('x1');
x2 = casadi.SX.sym('x2');
x = [x1; x2];
u = casadi.SX.sym('u');

% Model equations
xdot = [(1-x2^2)*x1 - x2 + u; x1];

% Objective term
L = x1^2 + x2^2 + u^2;

% Continuous time dynamics
f = casadi.Function('f', {x, u}, {xdot, L});

% Formulate discrete time dynamics Using
% fixed-step Runge-Kutta 4 integrator
M = 4; % RK4 steps per interval
DT = Delta/M;
f = casadi.Function('f', {x, u}, {xdot, L});
X0 = casadi.MX.sym('X0', 2);
U = casadi.MX.sym('U');
X = X0;
Q = 0;
for j = 1:M
   [k1, k1_q] = f(X, U);
   [k2, k2_q] = f(X + DT/2*k1, U);
   [k3, k3_q] = f(X + DT/2*k2, U);
   [k4, k4_q] = f(X + DT*k3, U);
   X = X + DT/6*(k1 + 2*k2 + 2*k3 + k4);
   Q = Q + DT/6*(k1_q + 2*k2_q + 2*k3_q + k4_q);
end
```

```
% VDP Oscillator using MPCTools.
mpc = import_mpctools();
Delta = 0.5;
Nt = 20; % Controller horizon.
Nx = 2;
Nu = 1;

% Define system model.
f = @(x, u) [(1 - x(2)^2)*x(1) ...
             - x(2) + u(1); x(1)];
l = @(x, u) x(1)^2 + x(2)^2 + u^2;

kwargs = struct('funcname', 'f', ...
                'rk4', true(), ...
                'Delta', Delta, ...
                'M', 4, ...
                'quad', l, ...
                'quadname', 'l');
[fcasadi, lcasadi] = ...
    mpc.getCasadiFunc(f, [Nx, Nu], ...
                      {'x', 'u'}, ...
                      '**', kwargs);
```

```
F = casadi.Function('F', {X0, U}, {X, Q}, ...
                    {'x0','p'}, {'xf', 'qf'});

% Choose parameters.
ulb = -1;
uub = 1;
x0 = [0; 1];
xf = [0; 0];

% Start with an empty NLP
w = {};
w0 = [];
lbw = [];
ubw = [];
J = 0;
g = {};
lbg = [];
ubg = [];

% "Lift" initial conditions
X0 = casadi.MX.sym('X0', 2);
w = {w{:}, X0};
lbw = [lbw; x0];
ubw = [ubw; x0];
w0 = [w0; x0];

% Formulate the NLP
Xk = X0;
for k = 0:(Nt - 1)
    % New NLP variable for the control
    Uk = casadi.MX.sym(['U_' num2str(k)]);
    w = {w{:}, Uk};
    lbw = [lbw; ulb];
    ubw = [ubw; uub];
    w0 = [w0;  0];

    % Integrate till the end of the interval
    Fk = F('x0', Xk, 'p', Uk);
    Xk_end = Fk.xf;
    J = J + Fk.qf;

    % New NLP variable for state at end
    % of interval
    Xk = casadi.MX.sym(['X_', ...
                        num2str(k+1)], 2);
    w = {w{:}, Xk};
    lbw = [lbw; -inf; -inf];
    ubw = [ubw; inf;  inf];
    w0 = [w0; 0; 0];

    % Add equality constraint
    g = {g{:}, Xk_end - Xk};
    lbg = [lbg; xf];
```

```
% Choose parameters.
ulb = -1;
uub = 1;
x0 = [0; 1];
xf = [0; 0];

% Formulate NLP.
N = struct('x', Nx, 'u', Nu, 't', Nt);
solver = mpc.nmpc('f', fcasadi, 'l', lcasadi, ...
                  'N', N, 'x0', x0, 'xf', xf, ...
                  'lb', struct('u', ulb), ...
                  'ub', struct('u', uub), ...
                  'verbosity', 5);
```

```
    ubg = [ubg; xf];
end

% Create an NLP solver
prob = struct('f', J, ...
              'x', vertcat(w{:}), ...
              'g', vertcat(g{:}));
solver = casadi.nlpsol('solver', ...
                       'ipopt', prob);

% Solve the NLP
sol = solver('x0', w0, 'lbx', lbw, ...
             'ubx', ubw, 'lbg', lbg, ...
             'ubg', ubg);

% Plot the solution
w_opt = full(sol.x);
x1 = w_opt(1:3:end);
x2 = w_opt(2:3:end);
u = w_opt(3:3:end);
t = (0:Nt)*Delta;

figure()
hold('on')

subplot(2, 2, 1);
plot(t, x1, '-ok');
ylabel('x_1', 'rotation', 0);

subplot(2, 2, 3);
plot(t, x2, '-ok');
ylabel('x_2', 'rotation', 0);

subplot(2, 2, 2);
stairs(t, [u; NaN], '-k');
ylabel('u', 'rotation', 0);
```

```
% Solve the OCP.
solver.solve();

% Plot the solution.
mpc.mpcplot(solver.var.x, solver.var.u, ...
            Delta*(0:Nt));
```

Even for this simple example, MPCTools can save a significant amount of coding, and it makes script files much shorter and more readable while still taking advantage of the computational power provided by CasADi.