MPCTools
Nonlinear MPC using CasADi

Michael Risbeck

April 5th, 2017

Nonlinear MPC

By now, you're (hopefully) familiar with the standard nonlinear MPC setup

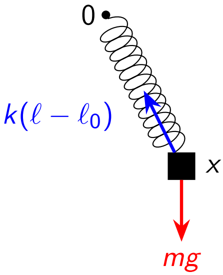
$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \quad & \sum_{k=0}^{N-1} \ell(x(k), u(k)) && \text{Stage Costs} \\
& \qquad\qquad + V_f(x(N)) && \text{Terminal Cost} \\
\text{s.t.} \quad & x(k+1) = f(x(k), u(k)) && \text{Model} \\
& (x(k), u(k)) \in \mathbb{Z} && \text{State/input Constraints} \\
& x(N) \in \mathbb{X}_f && \text{Terminal Set} \\
& x(0) = x && \text{Initial Condition}
\end{aligned}
$$

To solve example problems, we want fast algorithmic differentiation attached to powerful NLP solvers.

- CasADi provides both of these via Octave (and Matlab)
- Let's see an example

CasADi Basics

Consider the simple example of a hanging spring.



Total energy is minimized at equilibrium:

$$
\begin{aligned}
E &= k(\ell - \ell_0)^2 + mgh \\
&= k(\|x\| - \ell_0)^2 + mgx_2
\end{aligned}
$$

```
% Example use of CasADi
k = 10; L0 = 1; g = 9.8; m = 1;

x = casadi.SX.sym('x', 2);

E = k*(norm(x) - L0)^2 ...
    + m*g*x(2);

nlp = struct('x', x, 'f', E);
solver = ...
    casadi.nlpsol('springsolver', ...
                  'ipopt', nlp);

solution = solver('x0', [2; -1]);

% Equilibrium position:
disp(solution.x);
% >> DM([6.49685e-12, -1.49])
```

What's in CasADi?

CasADi = Computer algebra system + Algorithmic Differentiation

- Symbolic algebraic expression core
  - Can construct algebraic expressions and perform some simplification
  - Not a general-purpose computer algebra system
- State-of-the-art ODE and DAE integrators (e.g., CVODES, IDAS)
  - Can take derivatives of these objects!
- Links to state-of-the-art solvers (e.g., IPOPT, qpOASES)
  - Provides exact first and second derivatives
  - Initial support for discrete variables
- C code generation

See <`casadi.org`> for more information.

To paraphrase Spiderman:

> *With great power comes great possibility for people to write unreadable and unmaintainable code!*

From official CasADi Examples

```
for j=1:d+1
  % Construct Lagrange polynomials to get the
  % polynomial basis at the collocation point
  coeff = 1;
  for r=1:d+1
    if r ~= j
      coeff = conv(coeff, [1, -tau_root(r)]);
      coeff = coeff / (tau_root(j)-tau_root(r));
    end
  end
  % Evaluate the polynomial at the final time to get
  % the coefficients of the continuity equation
  D(j) = polyval(coeff, 1.0);

  % Evaluate the time derivative of the polynomial
  % at all collocation points to get the coefficients
  % of the continuity equation
  pder = polyder(coeff);
  for r=1:d+1
    C(j,r) = polyval(pder, tau_root(r));
  end

  % Evaluate the integral of the polynomial to get the
  % coefficients of the quadrature function
  pint = polyint(coeff);
  B(j) = polyval(pint, 1.0);
end
```

Things can escalate pretty quickly.

We don't want everyone writing this themselves!

What do we want?

- We want to solve nonlinear MPC problems.
- CasADi is more robust than our in-house software
- However, setting up an MPC problem in CasADi takes a lot of code
- Everyone copy/pasting their own code is bad
- A simpler interface means we can save a lot of time

Enter MPCTools

An Octave package (usually Matlab-compatible)

- Download from <https://bitbucket.org/rawlings-group/octave-mpctools>
- Put the `mpctools` folder somewhere and add it to Octave's path
- Running `mpc = import_mpctools()` gives access to functions via `mpc.*`
  - Can also call functions via `mpctools.*` without import

Comes with cheatsheet and full documentation (in the `doc` folder).

- Should get you started writing your own code.

Also includes a bunch of example files, e.g.,

- `cstr.m`: Example 1.11 using CasADi integrators and linearization
- `vdposcillator.m`: Example of linear vs. nonlinear MPC.
- `cstr_startup.m`: Nonlinear startup for Example 1.11 system.

System Model

Start by defining the system model as an Octave function.

```
function rhs = cstrode(x, u, p, pars)
    % Nonlinear ODE model for reactor.
    c = x(1); T = x(2); h = x(3) + eps();

    Tc = u(1); F = u(2);

    F0 = p(1);

    k = pars.k0*exp(-pars.E/T);
    rate = k*c;

    dcdt = F0*(pars.c0 - c)/(pars.A*h) - rate;
    dTdt = F0*(pars.T0 - T)/(pars.A*h) ...
           - pars.DeltaH/pars.rhoCp*rate ...
           + 2*pars.U/(pars.r*pars.rhoCp)*(Tc - T);
    dhdt = (F0 - F)/pars.A;

    rhs = [dcdt; dTdt; dhdt];
end%function
```

System Simulation

The nonlinear system can be simulated using CasADi `integrator` objects, created via a convenient wrapper.

```
% Turn into casadi function and simulator.
ode = @(x, u, p) cstrode(x, u, p, pars);
ode_casadi = mpc.getCasadiFunc(ode, [Nx, Nu, Np], ...
                               {'x', 'u', 'p'}, {'ode'});
cstrsim = mpc.getCasadiIntegrator(ode, Delta, [Nx, Nu, Np], ...
                                  {'x', 'u', 'p'}, {'cstr'});

% Simulate with nonlinear model.
x(:,t+1) = full(cstr(x(:,t), u(:,t), d(:,t)));
```

Note that `cstr` returns CasADi `DM` objects

- "Double Matrix", CasADi's internal (numeric) Matrix type
- Call to `full()` converts to native Octave matrix

Linear Unconstrained Control

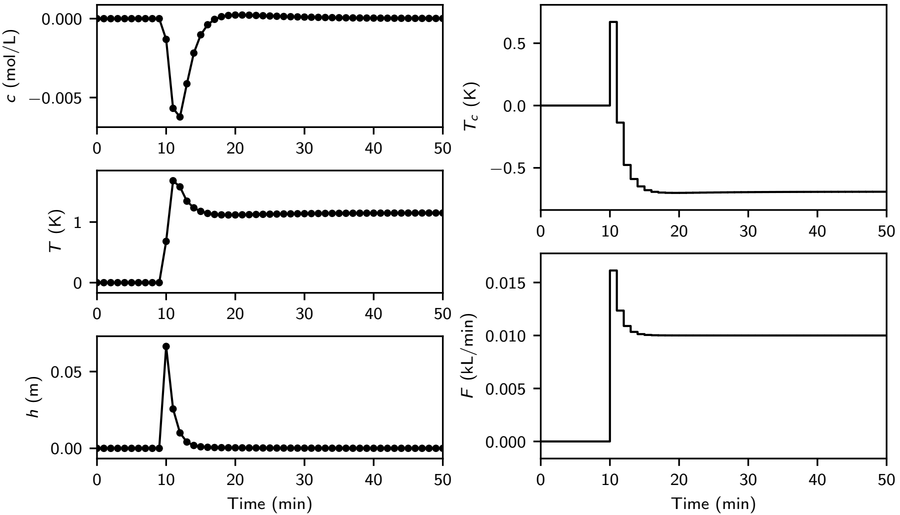
Set up linear controller and estimator.

```
% Get linearized model.
model = mpc.getLinearizedModel( ...
            ode_casadi, {xs, us, ps}, ...
            {'A', 'B', 'Bp'}, Delta);
A = model.A;
B = model.B;

% Find LQR.
[K, Pi] = dlqr(A, B, Q, R);
K = -K; % Note sign convention.

% Find Kalman Filter.
kf = mpc.KalmanFilter('A', A, ...
        'B', B, 'C', C, 'Bd', Bd, ...
        'Cd', Cd, 'Qw', Qw, ...
        'Rv', Rv, 'contvars', [1, 3]);
```

Simulate closed-loop.

```
for i = 1:(Nsim + 1)
    % Take measurement.
    y(:,i) = C*x(:,i) + v(:,i);

    % Advance state measurement.
    [xhat(:,i), dhat(:,i)] = ...
        kf.filter(y(:,i), xhatm(:,i), ...
                  dhatm(:,i));

    % Use steady-state target selector.
    [xtarg(:,i), utarg(:,i)] = ...
        kf.target(ysp(:,i));

    % Apply control law.
    u(:,i) = K*(xhat(:,i) - xtarg(:,i)) ...
             + utarg(:,i);

    % Evolve plant.
    x(:,i + 1) = full(cstrsim(x(:,i), ...
                   u(:,i), p(:,i)));

    % Advance state estimates
    [xhatm(:,i + 1), dhatm(:,i + 1)] = ...
        kf.predict(u(:,i), xhat(:,i), ...
                   dhat(:,i));
end
```

Results



c (mol/L)
0.000
−0.005
T (K)
0
1
h (m)
0.00
0.05
0 10 20 30 40 50
Time (min)
T_c (K)
0.5
0.0
−0.5
F (kL/min)
0.015
0.010
0.005
0.000
Time (min)

What can we do with MPCTools?

- Discrete-time linear MPC
- Discrete-time nonlinear MPC
  - Explicit models
  - Runge-Kutta discretization
  - Collocation
  - DAE systems
- Discrete-time nonlinear MHE
  - Explicit models
  - Runge-Kutta discretization
  - Collocation
- Steady-state target calculation
- Basic plotting

Example: Van der Pol Oscillator

```
% Define nonlinear and linearized models.
ode = @(x, u) [(1 - x(2).^2)*x(1) - x(2) + u(1); x(1)];
vdp = mpctools.getCasadiIntegrator(ode, Delta, [Nx, Nu], {'x', 'u'}, {'vdp'});
fnonlin = mpctools.getCasadiFunc(ode, [Nx, Nu], {'x', 'u'}, {'vdprk4'}, ...
                                 'rk4', true(), 'Delta', Delta);
linmodel = mpctools.getLinearizedModel(ode, {zeros(Nx, 1), zeros(Nu, 1)}, ...
                                       {'A', 'B'}, Delta);
Flin = mpctools.getCasadiFunc(@(x, u) linmodel.A*x + linmodel.B*u, [Nx, Nu], ...
                              {'x', 'u'}, {'vdplin'});

% Define objective functions.
stagecost = @(x, u) x'*x + u'*u;
l = mpctools.getCasadiFunc(stagecost, [Nx, Nu], {'x', 'u'}, {'l'});

termcost = @(x) 10*x'*x;
Vf = mpctools.getCasadiFunc(termcost, [Nx], ...
                            {'x'}, {'Vf'});

% Set bounds.
lb = struct('u', -0.75*ones(Nu, Nt));
ub = struct('u', ones(Nu, Nt));

% Build solvers.
N = struct('x', Nx, 'u', Nu, 't', Nt);
kwargs = struct('l', l, 'Vf', Vf, 'N', N, 'lb', lb, 'ub', ub);
solvers = struct();
solvers.LMPC = mpctools.nmpc('f', Flin,'**', kwargs);
solvers.NMPC = mpctools.nmpc('f', fnonlin, '**', kwargs);
```

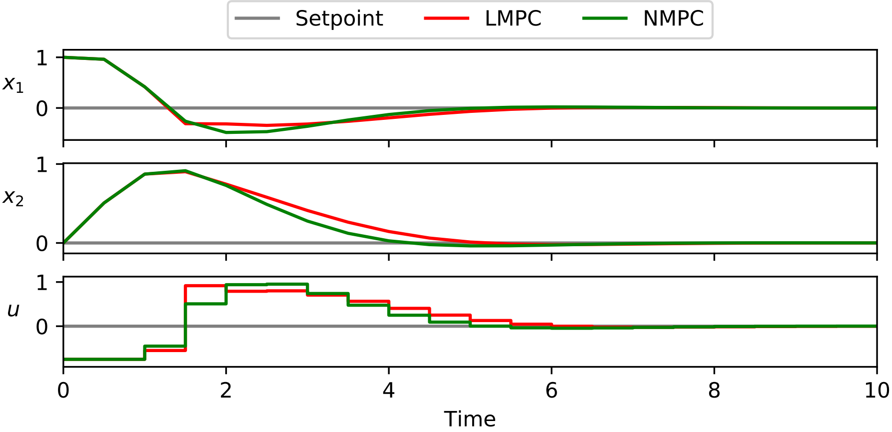
System Model:

$$\frac{dx}{dt} = \begin{pmatrix} (1 - x_2)^2 x_1 - x_2 + u \\ x_1 \end{pmatrix}$$

Simulation Results

For this problem, nonlinear MPC performs slightly better.

- The computation isn't much more time-consuming because of the power of CasADi.
- The problem isn't difficult to set up because of MPCTools.

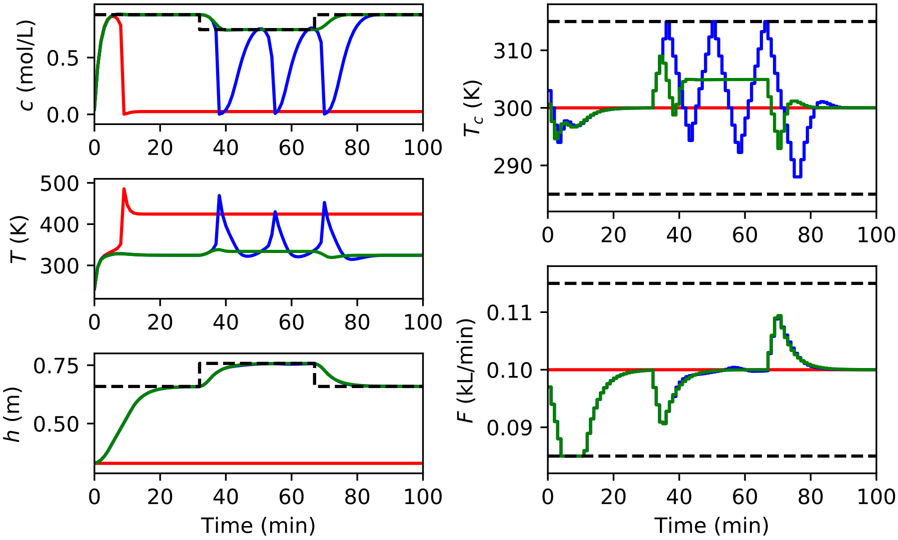


More Complicated Example

Using MPCTools, we can replace the LQR and KF from Example 1.11 with nonlinear MPC and MHE.

- `cstr_startup.m` shows basic structure and a setpoint change.
- `cstr_nmpc_nmhe.m` shows steady-state target finding and NMHE.
- See the cheatsheet for important functions and syntax.

Here, nonlinear MPC knows to be less aggressive.



What can't we do?

- True continuous-time formulation
  - Continuous-time models with explicit time dependence are not supported
  - Quadrature for continuous-time objective function is available via collocation or RK4
- Quality guess generation
  - Solve sequence of smaller problems
  - Use as initial guess for large problem
  - Must do by hand
- Stochastic MPC
- Robust MPC

That's all, folks!

- For questions, comments, etc., email <risbeck@wisc.edu>
- For bugs or feature requests, open an issue on Bitbucket
  - <https://bitbucket.org/rawlings-group/octave-mpctools>