Implementing MPC using CVX

A case illustration

1 Purpose

The purpose of this document, is to illustrate how an MPC problem can be formulated in a straight forward, intuitive and flexible way in Matlab, using a plug-in called CVX [2].

CVX is a tool that allows the user to solve general convex problems, by formulating them in an intuitive and straight forward fashion. In order to use CVX, one must obey the rules governed by disciplined convex programming, refer to *e.g.* [4]. We do not attempt to give a thorough introduction to CVX here, for this the reader should refer to either [2] or [3]. Instead we focus on the sole application of implementing a model predictive controller, where the constrained optimization step, is performed by use of CVX.

The people targeted with this document are mainly students and perhaps researchers at the University of Aalborg, as it is the distinct impression of the author, that implementations of model predictive controllers are currently handled in a certain way, which is neither easy, intuitive or flexible. This document therefore serves to provide an alternative. Of course people outside the above mentioned targeted group are welcome to employ the tips provided by this document, however do notice and take into account any license restrictions, when you use CVX.

As indicated above, this document focusses solely on Matlab and CVX. Of course implementations can be made in a morass of different programming languages, but only Matlab is treated here.

2 Motivation

As explained above, there are several ways of implementing MPC, depending on the the way any control problem is regarded. A commonly employed way of formulating an MPC problem, is described in [5], where the problem of finding the optimal control input over a horizon, is formulated as a quadratic program (QP), by iterating from the current state the control system, to the end of the control horizon. The QP can then be solved using standard Matlab functions such as QUADPROG() or LINPROG() if the problem has been reduced to a linear program (LP).

However this way of arranging an MPC implementation is rather cumbersome, as it requires a lot of preliminary work, for instance in order to expand the control system, to cover the entire control horizon. We will illustrate this with an example later. Also, when using the standard Matlab functions described above, the problem is required to be formulated as an LP or QP. This is, even though often applicable, not as simple as having it formulated as a general convex problem.

Formulating a problem as an LP or QP is often possible, but it sometimes requires reformulating the actual optimization problem, by adding variables and constraints in order to get it into QP or LP form. This also requires a lot of cumbersome coding, which firstly is prone to errors, and secondly can take a lot of time, which would be better spend elsewhere. In the following we show how CVX can spare us from a lot of preliminary work.

It should be mentioned that CVX should not be used to reduce computation time. In fact it will typically run slower than both QUADPROG() and LINPROG(). CVX should instead be used for its flexible and intutive way of formulating problems, as well as the possibility of formulating generally convex problems, rather than only LP's and QP's.

3 Example application

In order to illustrate the use of CVX for an MPC implementation, the following describes an example application. We will first describe the problem, where after we will illustrate the simplified model we are employing. Afterwards we will arrange the optimization problem to be solved, and illustrate how this can be handled, both in what we call the traditional way using standard Matlab functions, and also when employing CVX.

3.1 Problem description

We observe a thermal system, consisting of a home with a built-in floor-heating system. The task is to maintain a temperature within the home, close to a provided set-point.

Heating the household is performed by delivering power to the heating system. The price of power varies throughout the day.

The home has a significant time-constant meaning that it takes a long time to cool down. The optimization task is then how to obtain an optimal trade-off between buying power when the price is right, without the inhabitants of the home experiences any significant discomfort, either by the home being too cold or too hot. In other words, the task is to find the optimal heating profile with respect to the power delivered to the heating system at any given time.

This problem is treated in [6], where from we have borrowed both the problem in its basic form, as well as the model we employ in the following, however with a somewhat different style of notation.

3.2 Modeling

Below we have outlined a somewhat simplified model of the thermal system consisting of a household and its appertaining heating system. We have assumed that the home is heated by electrically controlled floor-heating, and that the time constant of our thermal model is related to the energy stored as heat in the concrete floors of the household. In the following, we formulate the system as being discrete.

The floors have a thermal capacity related to both the material of which they are made, *e.g.* concrete, and the mass of the specific floor. We denote this thermal capacity by $c_M \in \mathbf{R}$. The heating of the room is further related to both the transmission of energy from the floors to the air in the room, as well as a loss in energy by transmission from the air in the room, through the walls and outside to the ambient air. This transportation of energy is governed by a heat transfer coefficient $u_A \in \mathbf{R}$. Finally the energy provided by the heating system to the floors are related to the coefficient of performance of the heating system, denoted $\gamma \in \mathbf{R}$.

With the parameters described above, we can arrange the following model

$$T(k+1) \quad = \quad aT(k) + bw(k) + eT_o(k) \tag{1}$$

with

$$a = 1 - h\frac{u_A}{c_M}, \quad b = h\frac{\gamma}{c_M}, \quad e = h\frac{u_A}{c_M} \tag{2}$$

where $T(k), T_o(k)$ and $w(k) \in \mathbf{R}$ are the room-, outside temperature and power supplied to the heating system at time k, respectively. The final parameter $h \in \mathbf{R}$, is a scaling factor related to the time interval in which quantities of power can be bought, typically corresponding to one hour.

For a more extensive treatment and arrangement of this model, consult [6].

3.3 Optimization formulation

The optimization problem of choosing when to supply the heating system with power, is both dependent of the price of electricity, as well as the price we put on discomfort for the inhabitants in the house, when the temperature deviates from the set-point $T_{\text{sp}} \in \mathbf{R}$. This is constrained by a maximum deviation from the setpoint, governed by a maximum and minumum allowed room temperature denoted by $T_{\max} \in \mathbf{R}$ and $T_{\min} \in \mathbf{R}$, respectively.

The problem is further constrained by the actuator limitations dictated by input saturations in the heating system, denoted by $w_{\max}$ and $w_{\min} \in \mathbf{R}$, respectively. We would typically have $w_{\min} = 0$.

The problem can now be formulated as:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} \left(w(k+i)p_w(k+i) + \left| T(k+i) - T_{\text{ref}}(k+i) \right| p_T(k+i) \right) \\ \text{subject to} & T(k+i) = aT(k+i-1) + bw(k+i-1) + eT_o(k+i-1) \\ & w(k+i) \leq w_{\max}, \\ & w(k+i) \geq w_{\min}, \\ & T(k+i) \leq T_{\max}, \\ & T(k+i) \leq T_{\min}, \end{array} \tag{3}$$

where $[w(k+1)\ \dots\ w(k+N)] \in \mathbf{R}^N$ is the variable, and $i \in \{1,\dots,N\}$, where $N \in \mathbf{R}$ is the control horizon.

Above, $[T_{\text{ref}}(k+1)\ \dots\ T_{\text{ref}}(k+N)] \in \mathbf{R}^N$ is the temperature reference, $[p_w(k+1)\ \dots\ p_w(k+N)] \in \mathbf{R}^N$ and $[p_T(k+1)\ \dots\ p_T(k+N)] \in \mathbf{R}^N$, are the costs of power consumption and temperature deviations respectively.

It should be noted that $p_w(k+i), i \in \{1,\dots,N\}$ should be regarded as estimates of the future electricity prices, as well as $T_o(k+i), i \in \{1,\dots,N\}$ should be regarded as estimates on future ambient temperatures, *i.e.* a whether forecast.

In the following sections, we illustrate two ways of accommodating this problem, both using the standard Matlab functions, and by using CVX.

3.4 Traditional approach

In this section, we solve the problem described in Eq. 3 by first reformulating it as an LP, and then using LINPROG() to solve the optimization.

This is a two step procedure. First we arrange matrices, governing the system dynamics throughout the control horizon. Afterwards we introduce an additional set of variables and constraints in order to formulate the cost function from Eq. 3, as an LP.

3.4.1 Expanding system dynamics

This step is described extensively in [5]. Expanding the system results in the following matrices and vectors:

$$\mathcal{T} = \begin{bmatrix} T(k+1) \\ \vdots \\ T(k+N) \end{bmatrix}, \quad \mathcal{T}_{\text{ref}} = \begin{bmatrix} T_{\text{ref}}(k+1) \\ \vdots \\ T_{\text{ref}}(k+N) \end{bmatrix}, \quad \mathcal{A} = \begin{bmatrix} a \\ a^2 \\ \vdots \\ a^N \end{bmatrix}$$

$$\mathcal{B} = \begin{bmatrix} b & 0 & 0 & \dots & 0 \\ ab & b & 0 & \dots & 0 \\ a^2b & ab & b & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ a^{N-1}b & a^{N-2}b & a^{N-3}b & \dots & b \end{bmatrix}, \quad \mathcal{E} = \begin{bmatrix} e & 0 & 0 & \dots & 0 \\ ae & e & 0 & \dots & 0 \\ a^2e & ae & e & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ a^{N-1}e & a^{N-2}e & a^{N-3}e & \dots & e \end{bmatrix}$$

We can then write an expanded system equation as

$$\mathcal{T} = \mathcal{A}T(k) + \mathcal{B}\mathcal{W} + \mathcal{E}\mathcal{T}_o \tag{4}$$

where we have also employed the notation

$$\mathcal{W} = \begin{bmatrix} w(k) \\ \vdots \\ w(k+N-1) \end{bmatrix}, \quad \mathcal{T}_o = \begin{bmatrix} T_o(k) \\ \vdots \\ T_o(k+N-1) \end{bmatrix}.$$

Above $T(k)$ dictates the initial, or current conditions of the system.

3.4.2 Introducing extra variables

The cost function in Eq. 3 involves an absolute value, and is thereby not linear in its current form. We work around this by introducing two additional sets of variables, and arranging constraints on these that makes it possible to arrange a new optimization problem, equivalent to Eq. 3, but is formulated as an LP.

By introducing these two sets of variables denoted $\mathcal{X}_1 \in \mathbf{R}^N$ and $\mathcal{X}_2 \in \mathbf{R}^N$, along with their corresponding constraints, together with the expansions derived above, our problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \mathcal{P}_w^T\mathcal{W} + \mathcal{P}_T^T\mathcal{X}_1 + \mathcal{P}_T^T\mathcal{X}_2 \\ \text{subject to} & \mathcal{W} \leq w_{\max}, \\ & \mathcal{W} \geq w_{\min}, \\ & \mathcal{T} \leq T_{\max}, \\ & \mathcal{T} \leq T_{\min}, \\ & -\mathcal{X}_1 \leq \mathcal{T} - \mathcal{T}_{\text{ref}} \\ & \mathcal{X}_2 \geq \mathcal{T} - \mathcal{T}_{\text{ref}}, \end{array} \tag{5}$$

with variables $\mathcal{W}, \mathcal{X}_1$ and $\mathcal{X}_2 \in \mathbf{R}^N$.

In the above problem, each inequality should be interpreted element-wise. Further, we have extended our calligraphic notation by introducing $\mathcal{P}_w = [p_w(k+1), \dots, p_w(k+N)]^T$ and similarly $\mathcal{P}_T = [p_T(k+1), \dots, p_T(k+N)]^T$.

Eq. 5 is now formulated as an LP, which can be solved using standard functions from Matlab. Below we have presented the actual implementation using Matlab code.

3.4.3 Matlab code

The Matlab code for the above approach is illustrated below. Even though it is in somewhat pseudoform, we have tried to keep close to true Matlab notation.

```
% Starting conditions and model parameters
Tk = thermalModel.currentTemperature;
Tref = thermalModel.referenceTemperature;
u_a = thermalModel.heatTransferCoefficient;
c_m = thermalModel.heatCapacity;
gamma = thermalModel.coefficientOfPerformance;
h = thermalModel.scaling;
N = optimizationParameters.Horizon;

% Constraints
Tmax = comfortModel.maxTemperature;
Tmin = comfortModel.minTemperature;
wmax = heaterModel.maxPower;
wmin = heaterModel.minPower;

% Expanding constraints
Tmax = Tmax*ones(N,1);
Tmin = Tmin*ones(N,1);
wmax = wmax*ones(N,1);
wmin = wmin*ones(N,1);

% Arrange model
a = 1-h*u_a/c_m;
b = h/c_m*gamma;
calA = a.^(1:N)';
calB = zeros(N,N);
calB(1,1) = b;
for k = 2:N
    calB(k,:) = a*calB(k-1,:);
    calB(k,k) = b;
end

e = h*u_a/c_m;
calE = zeros(N,N);
calE(1,1) = e;
for k = 2:N
    calE(k,:) = a*calE(k-1,:);
    calE(k,k) = e;
end

% Performance funktion P = sum_1^N (w(k)*p_w(k) + p_T*abs(T(k)-Tref))
% is rewritten into c'*x = p_w'*w+ p_T'*x1 + p_T'*x2
% x1 : -negative temperature deviations
% x2 : positive temperatur deviations

c = [h*p_w' p_T' p_T'];
G = [calB zeros(N,N) zeros(N,N);  %T<Tmax
```

```
    -calB zeros(N,N) zeros(N,N);  %-T<-Tmin
    -calB -eye(N,N) zeros(N,N);   %-x1<=-Tref+T
     calB zeros(N,N) -eye(N,N);  %-x2<=Tref-T
     eye(N,N) zeros(N,N) zeros(N,N)]; %w < wmax
f = [Tmax-calA*Tk-calE*To;
    -Tmin+calA*Tk+calE*To;
    -Tref+calA*Tk+calE*To;
     Tref-calA*Tk-calE*To;
     wmax*ones(N,1)-whotw/gamma];
 lb = zeros(3*N,1);
 X = linprog(c,G,f,[],[],lb);
 wopt = X(1:N,1);
 Topt = A*T0+B*wopt+E*Tomg;
 Popt = c'*X
```

Resulting from the above code `wopt` would be the optimal use of power, `Topt` would be the resulting temperature and `Popt` would be the final cost of our power consumption, with respect to the overall cost function.

3.5 Problem implementation using CVX

In Section 3.4, we made a somewhat rigorous illustration the steps connected to the MPC implementation. We did this with the intention of illuminating how much effort is actually needed, on preliminary tasks that is not related to the actual problem, and how cumbersome this preliminary work can be. In fact, from the entire code-snippet presented in Section 3.4.3, only one line is related to the actual optimization. Everything else is related to arranging the model, expanding model coefficients, and reformulating the cost function to an LP.

Below is presented the way the above would be implemented in CVX. As there is no preliminary work, extra formulations or necessary rewriting of any kind, we will simply show the Matlab code.

3.5.1 Matlab code

```
% Starting conditions and model parameters
Tk = thermalModel.currentTemperature;
Tref = thermalModel.referenceTemperature;
u_a = thermalModel.heatTransferCoefficient;
c_m = thermalModel.heatCapacity;
gamma = thermalModel.coefficientOfPerformance;
h = thermalModel.scaling;
N = optimizationParameters.Horizon;

% Constraints
Tmax = comfortModel.maxTemperature;
Tmin = comfortModel.minTemperature;
wmax = heaterModel.maxPower;
```

```
wmin = heaterModel.minPower;

% Arrange model
a = 1-h*u_a/c_m;
b = h/cm*gamma;
e = h*u_a/c_m;

% Bounding constraints
upp = wmax*ones(N,1) - (whotw/gamma);
low = zeros(N,1);

cvx_begin
    variables w(N) T(N)
    minimize(h*p_w'*w + p_T'*abs(T-Tref))
    subject to
        % System Dynamics:
        T(1:end) == a*[Tk; T(1:end-1)] + b*w(1:end) + e*To(1:end);

        % Comfort constraints:
        T <= Tmax;
        T >= Tmin;

        % Actuator limits:
        w <= upp;
        w >= low;
cvx_end
```

What the reader should take from this, is how close the actual coding of the problem is to the formulation in Section 3.3. Also, the implementation illustrated above is far more flexible than the approach described in Section 3.4, if it becomes necessary to change either the cost function, or the extend of the problem. We illustrate this in the following.

3.6 Extensions

As we have mentioned earlier, CVX can solve generally formulated convex problems, when obeying the rules of disciplined convex programming. This entails that the cost function can be any convex function, provided that CVX is able to certify it as such, consult the CVX manual for elaboration on this [3]. Below we have simple example of this.

3.6.1 Using different norms

Above, our cost function consisted of two terms. The first was the cumulated cost of buying power, and the other was a scaled 1-norm of the temperature deviation of the temperature from the set-point.

Different norms carries different characteristics when used in cost functions, see for instance [1]. Using the 1-norm would typically result in a sparse solution, where the temperature in most instances would correspond to the set-point, but in few instances, the deviation would be relatively high.

Using a 2-norm would instead minimize the average deviation, making the deviation at any point rather small, but only at rare instances would the temperature match the set-point.

Similarly, if an ∞-norm where used, the maximum deviation would be small, but all deviations would have roughly the same magnitude. Obviously, other concerns could be relevant, and correspondingly, other norms or cost functions could be relevant. However, the current extend of our discussion serves a sufficient point, and we will therefore not discuss this any further here.

Changing the cost function, or perhaps adding additional elements to it, in the approach presented in Section 3.4, would entail quite extensive work on reformulating the problem and system matrices involved. However the job is easily done in CVX. For instance, punishing the 2-norm of temperature deviation, instead of a 1-norm, could be achieved in the following fashion:

```
cvx_begin
    variables w(N) T(N)
    minimize(h*p_w'*w + norm(p_T.*(T-Tref),2))
    subject to
        % System Dynamics:
        T(1:end) == a*[Tk; T(1:end-1)] + b*w(1:end) + e*To(1:end);

        % Comfort constraints:
        T <= Tmax;
        T >= Tmin;

        % Actuator limits:
        w <= upp;
        w >= low;
cvx_end
```

or perhaps an ∞-norm:

```
cvx_begin
    variables w(N) T(N)
    minimize(h*p_w'*w + norm(p_T.*(T-Tref),inf))
    subject to
        % System Dynamics:
        T(1:end) == a*[Tk; T(1:end-1)] + b*w(1:end) + e*To(1:end);

        % Comfort constraints:
        T <= Tmax;
        T >= Tmin;

        % Actuator limits:
        w <= upp;
        w >= low;
cvx_end
```

In similar fashion, a large number of different cost functions could be arranged, and also a variety of constraints can be formulated in a general convex fashion, and interpreted by CVX [3].

3.6.2 Expanding system size

The example presented so far, have only been optimizing the power consumption for a single household, however, we can very easily extend it to involve a general number n. Here it should be mentioned, that if the problem size gets too big, CVX will run into numerical issues, and perhaps return an error message.

```
n = 2; % Number of Households

% Vectorized parameters
a = [a1 a2]; % If more, expand by additional a's
b = [b1 b2]; % If more, expand by additional b's
e = [e1 e2]; % If more, expand by additional e's

% Model matrices

A = diag(a); % a-coefficients for all households
B = diag(b); % b-coefficients for all households
E = diag(e); % e-coefficients for all households

% Constraint bounds
upp = [wmax1*ones(1,N); wmax2*ones(1,N);
low = zeros(n,N);

% Cost function matrices
TO = [To1'; To2'];
TREF = [Tref1'; Tref2'];
P_W = [p_w1'; p_w2'];
P_T = [p_T1'; p_T2'];
TK = [Tk1; Tk2];
TMAX = [Tmax1'; Tmax2];
TMIN = [Tmin1'; Tmin2];

cvx_begin
   variables w(n,N) T(n,N)
   minimize(sum(sum(h*P_W.*w + abs(T-TREF).*P_T)))
   subject to
       % System Dynamics:
       T(:,1:end) == A*[TK T(:,1:end-1)] + B*w(:,1:end) + E*TO(:,1:end);

```

```
        % Comfort constraints:
        T <= TMAX;
        T >= TMIN;

        % Actuator limits:
        w <= upp;
        w >= low;
cvx_end
```

The optimal power consumption would then be the matrix `w`, and the corresponding temperature would be `T`.

4 Closing remarks

In this document we have shown how CVX can be used for arranging simple implementations of MPC in Matlab.

We have shown how the cost function can be easily changed, without having to perform any work on rearranging system matrices. Finally we have briefly illustrated how problems can be expanded.

References

[1] S. P. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[2] M. Grant and S. P. Boyd. CVX: Matlab software for disciplined convex programming. `http://cvxr.com/cvx/`, 2011.

[3] M. Grant and S. P. Boyd. CVX Users guide for CVX version 1.21. `http://web.cvxr.com/cvx/cvx_usrguide.pdf`, 2011.

[4] M. Grant, S. P. Boyd, and Y. Ye. *Global Optimization: From Theory to Implementation.* Editors: L. Liberti and N. Maculan, Springer, 2006. Pages 155-210.

[5] J. Maciejowski. *Predictive control with constraints.* Prentice Hall, 2000.

[6] T. Pedersen, P. Andersen, K.M. Nielsen, H.L. Stærmose, and P.D. Pedersen. Using heat pump energy storages in the power grid. *IEEE Conference on Control Applications, Proceedings*, 2011.