MPC for nonlinear systems using trajectory linearizations

Johan Löfberg

Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
WWW: http://www.control.isy.liu.se
Email: johanl@isy.liu.se

September 9, 1999

Report no.: LiTH-ISY-R-2173

Technical reports from the Automatic Control group in Linköping are available by anonymous ftp at the address ftp.control.isy.liu.se. This report is contained in the compressed postscript file 2173.ps.Z.
MPC for nonlinear systems using trajectory linearizations

Johan Löfberg

September 9, 1999

Abstract

This report presents a simple method to extend linear MPC to nonlinear continuous-time systems. The method is based on an approximation of the underlying optimal control problem. Experiments have been carried out and show that the control algorithm manages to control various nonlinear systems.

Keywords: MPC, Nonlinear control

1 Introduction

With Model Predictive Control, MPC, we mean the problem of solving an optimal control problem over a finite horizon at each sample, applying the first part of the calculated control sequence, and repeat this procedure at each sample instant. Another, more suitable name for this strategy is Receding Horizon Control.

The optimal control problem can in the most general case be formulated as finding a control sequence \( u(t) \) that minimizes a cost over a (in)finite horizon \( \tau \),

\[
\begin{align*}
\min_{u(t)} & \quad J(x(t), u(t), r(t)) \\
\text{subject to} & \quad u(t) \in \mathcal{U} \\
x(t) & \in \mathcal{X}
\end{align*}
\]

(1)

subject to constraints on both states and control,

\[
\begin{align*}
\min_{u(t)} & \quad J(x(t), u(t), r(t)) \\
\text{subject to} & \quad u(t) \in \mathcal{U} \\
x(t) & \in \mathcal{X}
\end{align*}
\]

(2)

(3)

given a desired trajectory \( r(t) \) for \( y(t) \) to follow, and a model of the system

\[
\begin{align*}
\dot{x}(t) & = f(x(t), u(t)) & x(t) & \in \mathbb{R}^n, u(t) & \in \mathbb{R}^m \\
y(t) & = g(x(t)) & y(t), r(t) & \in \mathbb{R}^p
\end{align*}
\]

(4)

The classical MPC setting is a linear (discrete time) system, linear constraints and a quadratic cost functional. This has been studied (and used in practice) a lot and has led to a fairly good understanding of the problem. For some practical aspects of linear MPC, see [4], while more theoretical issues are discussed in, e.g., [6].

The nonlinear case, as described above, has also been studied over the last years ([5], [2], [6], [3]), but as always with nonlinear problems, a general (practically applicable) solution is hard to find.
2 Prediction models

MPC can be divided into two main parts, the formulation of a model with some predictive power, and the solution of an optimal control problem, using the aforementioned model. In the following sections, two approaches to prediction models for nonlinear systems are described. By using these models, standard MPC algorithms apply.

In the linear discrete case, the plant model is in itself a linear prediction model, i.e., the future output is linearly dependent on the current state and the future control inputs. For a nonlinear system, this is not the case, but we can try to create an approximately linear discrete prediction model, and the natural approach is to use linearizations of the model.

2.1 MPC based on current state linearization

The first version of nonlinear MPC is based on a linearization of the model around the current state and the previously calculated control action for the following sample period, i.e., in $(\hat{x}(t | t), u(t | t - T))$. Close to the linearization point $(\hat{x}(t | t), u(t | t - T))$ we have the linear model

\[
\begin{align*}
\dot{x}(t) &= f(\hat{x}(t | t), u(t | t - T)) + A(x(t) - \hat{x}(t | t)) + B(u(t) - u(t | t - T)) \\
y(t) &= \hat{C}x(t)
\end{align*}
\]

We create a discrete time model, and from now on, we denote $x(t)$ as $x(k)$, $x(t + T)$ as $x(k + 1)$ etc. to emphasize that we are working with a discrete model. To simplify notation, we use the notation $(x_0, u_0) = (\hat{x}(t | t), u(t | t - T))$.

\[
\begin{align*}
x(k + 1) &= x_0 + F + A(x(k) - x_0) + B(u(k) - u_0) \\
y(k) &= Cx(k)
\end{align*}
\]

where we have the discrete state-space matrices

\[
\begin{align*}
F &= \int_0^T e^{Ax}f(x_0, u_0) \, ds \\
A &= e^{AT} \\
B &= \int_0^T e^{Ax}Bds \\
C &= \hat{C}
\end{align*}
\]

2.1.1 Predictor formulation

We use the linear model to approximately predict $\hat{x}(k + i)$

\[
\begin{align*}
\hat{x}(k + 1) &= x_0 + F + B(u(k) - u_0) \\
\hat{x}(k + 2) &= x_0 + F + A(\hat{x}(k + 1) - x_0) + B(u(k + 1) - u_0) \\
&= x_0 + F + AF + AB(u(k) - u_0) + B(u(k + 1) - u_0) \\
&\vdots \\
\hat{x}(k + N) &= x_0 + \sum_{i=0}^{N-1} A^iF + A^iB(u(k + N - i - 1) - u_0)
\end{align*}
\]
Since $\hat{y}(k + i) = C\hat{x}(k + i)$ we have the predictor

$$Y = H + \tilde{S}(\tilde{U} - U_0)$$  \hspace{1cm} (12)

where we have defined

$$Y = \begin{bmatrix} \hat{y}(k + N) \\ \hat{y}(k + N - 1) \\ \vdots \\ \hat{y}(k) \end{bmatrix}, \tilde{U} = \begin{bmatrix} u(k + N - 1) \\ u(k + N - 2) \\ \vdots \\ u(k) \end{bmatrix}, U_0 = \begin{bmatrix} u_0 \\ u_0 \\ \vdots \end{bmatrix}$$  \hspace{1cm} (13)

$$H = \begin{bmatrix} C\left(x_0 + \sum_{i=0}^{N-1} A^i F\right) \\ C\left(x_0 + \sum_{i=0}^{N-2} A^i F\right) \\ \vdots \\ C(x_0 + F) \end{bmatrix}$$  \hspace{1cm} (14)

$$\tilde{S} = \begin{bmatrix} CB & CAB & \cdots & CA^{N-1}B \\ 0 & CB & \cdots & CA^{N-2}B \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & CB \end{bmatrix}$$  \hspace{1cm} (15)

The control horizon $M$ can be chosen less than the prediction horizon $N$, and in that case we have to make an assumption about the remaining control sequence. A natural choice is

$$u(k + i) = u(k + M - 1) \quad i = M, M + 1, \ldots, N - 1$$  \hspace{1cm} (16)

In a stacked matrix form this can be written as

$$\tilde{U} = \Lambda U$$  \hspace{1cm} (17)

where we have defined

$$\Lambda = \begin{bmatrix} I_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ I_m & 0 & \cdots & 0 \\ 0 & I_m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_m \end{bmatrix}$$  \hspace{1cm} (18)

and the control sequence to calculate

$$U = \begin{bmatrix} u(k + M - 1) \\ u(k + M - 2) \\ \vdots \\ u(k) \end{bmatrix}$$  \hspace{1cm} (19)

All together we have the linear prediction

$$Y = H + \tilde{S}(\tilde{U} - U_0) = H + \tilde{S}\Lambda(U - U_0) = H + S(U - U_0)$$  \hspace{1cm} (20)
2.2 MPC based on trajectory linearization

The second algorithm is an extension of the previously described approach. In this algorithm, the control sequence\(^1\) calculated in the previous sample is applied to the system model, and a predicted future trajectory is obtained. In each sample point along this trajectory, a linear model is calculated, giving us an approximately linear discrete time-varying model over the future \(N\) samples.

2.2.1 Initial trajectory prediction

The exact way to calculate the future trajectory, \(\hat{x}(k+i)\), would be to perform an integration of the nonlinear model (4) with \(U_{k-1}\) as input. It would require a considerable amount of computation to perform this integration exactly, so we use an approximation instead. Since a discrete model of a linear system is an exact integration of a constant input on a linear continuous system, we can use our linearized models as approximate integrations of our piecewise constant control signal. In this way we create linear discrete time models over the whole future trajectory at the same time as we calculate the trajectory.

After linearization and discretization in the current state and previously calculated control input, \([\hat{x}(k \mid k) \ u(k \mid k-1)]\), we have

\[
\begin{align*}
\hat{x}(k+1 \mid U_{k-1}) &= \hat{x}(k \mid k) + F(k) + A(k)(x(k) - \hat{x}(k \mid k)) + \\
B(k)(u(k) - u(k \mid k-1))
\end{align*}
\]

\[
\hat{y}(k+1 \mid U_{k-1}) = C(k+1)\hat{x}(k+1 \mid U_{k-1})
\]

(21)

To get a more compact notation we define the predicted trajectory

\[
\tilde{x}(k+1) \triangleq \hat{x}(k+i \mid U_{k-1})
\]

(22)

We apply \(u(k \mid k-1)\) to the system and if we assume \(x(k) = \tilde{x}(k \mid k)\) we obtain

\[
\tilde{x}(k+1) = x(k) + F(k)
\]

(23)

Note that \(A(k), B(k)\) and \(C(k)\) are not used here, but they will be used in the next section.

The main idea is that we now re-linearize the system in this point, i.e. in \((\tilde{x}(k+1), u(k+1 \mid k-1))\), and in the same way as above

\[
\hat{x}(k+2) = \tilde{x}(k+1) + F(k+1) = x(k) + F(k) + F(k+1)
\]

(24)

This is repeated along the prediction horizon, giving us an approximation of the predicted trajectory.

\[
\tilde{x}(k+i) = x(k) + \sum_{j=0}^{i-1} F(k+j)
\]

(25)

\(^1\)The control sequence must be extended with one new control input
2.2.2 Predictor formulation

At every future sample we have a linear model defined by

\[ A(i), B(i), C(i), F(i) \quad i = k, k + 1, \ldots, k + N - 1 \tag{26} \]

We are now able to create a predictor just as in the previous section

\[
\hat{x}(k + 1 \mid U_k) = \hat{x}(k \mid k) + F(k) + A(k)(x(k) - \hat{x}(k \mid k)) \\
+ B(k)(u(k) - u(k \mid k - 1))
\]

Again, we define the short hand notation

\[
\hat{x}(k + i) \overset{\text{def}}{=} \hat{x}(k + i \mid U_k) \tag{27}
\]

and assume that \( x(k) = \hat{x}(k \mid k) \)

\[
\hat{x}(k + 1) = x(k) + F(k) + B(k)(u(k) - u(k \mid k - 1)) \\
= \hat{x}(k + 1) + B(k)(u(k) - u(k \mid k - 1)) \\
\hat{x}(k + 2) = \hat{x}(k + 1) + F(k + 1) + A(k + 1)(\hat{x}(k + 1) - \bar{x}(k + 1)) \\
+ B(k + 1)(u(k + 1) - u(k + 1 \mid k - 1)) \\
= \hat{x}(k + 1) + F(k + 1) + A(k + 1)(\bar{x}(k + 1) + \\
B(k)(u(k) - u(k \mid k - 1)) - \bar{x}(k + 1)) + \\
B(k + 1)(u(k + 1) - u(k + 1 \mid k - 1)) \\
= \hat{x}(k + 2) + A(k + 1)B(k)(u(k) - u(k \mid k - 1)) + \\
B(k + 1)(u(k + 1) - u(k + 1 \mid k - 1)) \tag{28}
\]

By introducing the auxiliary variable

\[
\Pi(i, n) = \begin{cases} A(k + i - 1)A(k + i - 2) \ldots A(k + n + 1) & (i > n + 1) \\ I & (i = n + 1) \\ 0 & (i < n + 1) \end{cases} \tag{29}
\]

the predictor can be written as

\[
\hat{x}(k + i) = \bar{x}(k + i) + \sum_{n=0}^{i-1} \Pi(i, n)B(k + n)(u(k + n) - u(k + n \mid k - 1)) \tag{30}
\]

In stacked form we have

\[
Y = H + \bar{S}(\bar{U} - U_0) \tag{31}
\]

where

\[
\bar{S} = \Gamma S \tag{32}
\]

\[
\Gamma = \begin{bmatrix} C(k + N) & 0 & \ldots & 0 \\ 0 & C(k + N - 1) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & C(k) \end{bmatrix} \tag{33}
\]
\[
\bar{S} = \begin{bmatrix}
B(k + N - 1) & \Pi(N, N - 2)B(k + N - 2) & \cdots & \Pi(N, 0)B(k) \\
0 & B(k + N - 2) & \cdots & \Pi(N - 1, 0)B(k) \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & B(k)
\end{bmatrix}
\]

(34)

\[
H = \begin{bmatrix}
C(k + N - 1)\tilde{x}(k + N - 1) \\
C(k + N - 2)\tilde{x}(k + N - 2) \\
\vdots \\
C(k + 1)\tilde{x}(k + 1)
\end{bmatrix}
\]

(35)

As before we make assumptions about the last part of the control sequence

\[
u(k + i) = u(k + M - 1) \quad (i = M, M + 1, \ldots, N - 1)
\]

(36)

and with the same method as above, we are able to define the predictor

\[
Y = H + \bar{S}(\bar{U} - U_{k-1}) = H + \bar{S}A(U - U_{k-1}) = H + S(U - U_{k-1})
\]

(37)

### 3 Filtered tracking error

MPC algorithms can sometimes be hard to tune. One solution to this is to include a filter on the tracking error, \( y(k) - r(k) \). In state space form this filter can be defined as

\[
\begin{align*}
\epsilon(k + 1) &= A_e\epsilon(k) + B_e(y(k) - r(k)) \\
\epsilon(k) &= C_e\epsilon(k) + D_e(y(k) - r(k))
\end{align*}
\]

(38)

We can stack these equations as

\[
E = H_e + S_e(Y - R)
\]

(39)

\[
H_e = \begin{bmatrix}
C_eA_e^N\epsilon(k)x_0 + C_eA_e^{N-1}B_e(y(k) - r(k)) \\
C_eA_e^{N-1}\epsilon(k)x_0 + C_eA_e^{N-2}B_e(y(k) - r(k)) \\
\vdots \\
C_eA_e\epsilon(k)x_0 + C_eB_e(y(k) - r(k))
\end{bmatrix}
\]

(40)

\[
S_e = \begin{bmatrix}
D_e & C_eB_e & C_eA_eB_e & \cdots & C_eA_e^{N-2}B_e \\
0 & D_e & C_eB_e & \cdots & C_eA_e^{N-3}B_e \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & D_e
\end{bmatrix}
\]

(41)

The main tuning of the closed loop dynamics can now be done by selecting the tracking error filters. For example, for a SISO plant, the choice \((A_e, B_e, C_e, D_e) = (0, 1, \alpha, 1 - \alpha)\) will lead to a first order filter were we control the time constants of the closed loop.
4 MPC formulation

The standard approach in MPC is to use a quadratic cost function

\[
J_N = e_{k+N}^T P e_{k+N} + \sum_{i=0}^{i=N-1} e_{k+i}^T Q_{e} e_{k+i} + \sum_{i=0}^{i=M-1} u_{k+i}^T Q_{u} u_{k+i} + \sum_{i=0}^{i=M-1} \delta u_{k+i}^T Q_{\delta u} \delta u_{k+i}
\]

\[
= ||e_{k+N}||^2 + ||E||^2 Q_{y} + ||U||^2 Q_{u} + ||\delta U||^2 Q_{\delta u}
\]  

(42)

Together with our linear prediction

\[
E = H_e + S_e (Y - R) = H_e + S_e (H + SU - R)
\]  

(43)

and a set of linear constraints on the control sequence

\[
A_d U \leq b_d
\]  

(44)

the problem setup is a standard QP problem, which can be solved efficiently with existing software. Constraints on the states are not studied in this report, but are treated in same way.

In the equations above, we have introduced control differences

\[
\delta u(k + i) = u(k + i) - u(k + i - 1) \quad i = 0, 1, \ldots, M - 1.
\]  

(45)

To define the vector \( \delta U \) we introduce the delta operator

\[
\Delta = \begin{bmatrix}
I_m & -I_m & 0 & \cdots & 0 \\
0 & I_m & -I_m & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & I_m & -I_m \\
0 & 0 & \cdots & 0 & I_m
\end{bmatrix}
\]  

(46)

and a vector with the previous control input on the last row

\[
U_{-1} = [0 \quad 0 \quad \cdots \quad u(k - 1)]^T
\]  

(47)

With this we can define the difference of the control sequence as

\[
\delta U = \Delta U - U_{-1}
\]  

(48)

4.1 Constraints

The main benefit with MPC is the possibility to include constraints on signals in the plant. A typical situation is hard bounds on the control signal

\[
u_{\text{min}} \leq u(i) \leq u_{\text{max}}
\]  

(49)

\[
\delta u_{\text{min}} \leq \delta u(i) \leq \delta u_{\text{max}}
\]  

(50)
If we define the matrices

\[
A_U = \begin{bmatrix} I_{Mn} \\ -I_{Mn} \\ \Delta \end{bmatrix}, \quad b_U = \begin{bmatrix} U_{\text{max}} \\ -U_{\text{min}} \\ \delta U_{\text{max}} + U_{-1} \\ \delta U_{\text{min}} + U_{-1} \end{bmatrix} \tag{51}
\]

and the \((Mm \times 1)\) sized vectors

\[
U_{\text{max}} = [u_{\text{max}}^T, u_{\text{max}}^T, \ldots, u_{\text{max}}^T]^T \\
U_{\text{min}} = [u_{\text{min}}^T, u_{\text{min}}^T, \ldots, u_{\text{min}}^T]^T \\
\delta U_{\text{max}} = [\delta u_{\text{max}}^T, \delta u_{\text{max}}^T, \ldots, \delta u_{\text{max}}^T]^T \\
\delta U_{\text{min}} = [\delta u_{\text{min}}^T, \delta u_{\text{min}}^T, \ldots, \delta u_{\text{min}}^T]^T \tag{52}
\]

we can write the constraints as

\[
A_U U \leq b_U \tag{53}
\]

4.2 Total QP formulation

Collecting the terms above give us the QP problem (for simplicity we choose \(P = Q_e\))

\[
U = \arg \min_U \quad ||H_e + S_e (H + SU - R)||_{Q_e} + ||U||_{Q_u} + ||\delta U||_{Q_{\delta u}} \\
A_U U \leq b_U \tag{54}
\]

4.3 Refinement of solution

For a linear system, the solution to the QP problem above will be the optimal solution. For a nonlinear system, where we use a linear approximation, the solution can be interpreted as a search direction, \(d_k = U_k - U_{k-1}\). We therefore iterate this procedure to refine the solution according to the following simple scheme:

1. Let \(U^* = U_{k-1}\).
2. Linearize along trajectory by approximately integrating the plant model with \(U^*\) as input (described in 2.2.1).
3. Create the linear QP and solve for \(U\).
4. If the solution has converged in some sense, for example \(||U - U^*|| < \epsilon\), or if we have reached the maximum number of iterations, terminate iteration.
5. Create a new control \(U^* = U^* + \alpha (U - U^*)\). The choice of \(\alpha\) is not easy, if we choose it too large, divergence may occur, and if it is too small, the convergence rate will be too slow. Simple approaches are for example constant \(\alpha\), or an \(\alpha\) obtained by a line search, i.e. we calculate the cost functional for \(\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]\) and choose the \(\alpha_i\) that minimizes the cost. A detailed discussion on this can be found in ([1]).

In most cases, 1-2 iterations are enough to approximately reach a local optimum.
4.4 Stability

Proving stability of the presented algorithm for general nonlinear systems is a formidable task. In literature, some result on similar approaches have been established ([3], [2]). In almost all proposed methods, the solution is to choose the endpoint-weight in a clever manner and/or append extra constraints on the end-state \( x(k+N) \).

4.4.1 Choice of terminal weight

For example, if we assume a linear system, and the cost functional

\[ J_N = x_{k+N}^T P x_{k+N} + \sum_{i=0}^{i=N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) \] (55)

then the optimal choice would (in some sense) be the solution to the Riccati equation arising in infinite horizon LQR. The proof in the unconstrained case is straightforward: since the infinite cost \( J_\infty \) is \( x_0^T P x_0 \), we would have

\[ J_N = x_{k+N}^T P x_{k+N} + \sum_{i=0}^{i=N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) \]

\[ = \sum_{i=N}^{i=\infty} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) + \sum_{i=0}^{i=N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) = J_\infty \] (56)

so the solution to our finite horizon problem would coincide with the solution to the infinite horizon solution, i.e., we would have recovered the LQR regulator. Since this \( P \) has given us the LQR solution, we also know that the system is stable (in the unconstrained case).

In the constrained case, we can add the extra constraint \( x(k+N) \in B \), where \( B \) is a positively invariant set with the infinite horizon control law and in which the unconstrained infinite horizon control law satisfies the control constraints.

There is, however, one major drawback with these methods: the reason we add an endpoint-weight is that we want to solve the optimization problem over a short horizon while maintaining stability, but with the new constraint, the problem might be infeasible, so we must have a longer horizon to make it feasible.

5 Examples

In all simulations, the endpoint-weight \( P = Q_e \) was used. The influence of noise and model errors has not been studied. The algorithm described in 2.1 is called algorithm 1, while the method from 2.2 will be referred to as algorithm 2.

\(^2\)Stability can be assured by using a long enough horizon.
5.1 State dependent gain

In the first example we study the nonlinear SISO system

\[ \ddot{x}(t) = \frac{u(t)}{1 + |x(t)|} \]  

(57)

with the control constraints

\[-1 \leq u(k) \leq 1 \]
\[-5 \leq \delta u(k) \leq 5 \]  

(58)

This is an example of a plant where algorithm 2 works much better than algorithm 1. The following parameters were used (for both algorithm 1 and 2).

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Q_e</th>
<th>Q_u</th>
<th>Q_{\delta u}</th>
<th>T</th>
<th>\alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0.00001</td>
<td>0.1 s</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Practically all tuning for this example can be done with the time-constant parameter \( \alpha \) (see section 3). To avoid ringing in the control signal, \( Q_{\delta u} \) is chosen non-zero.

A step response was simulated resulting in the behavior shown below.

![Step response comparison](image1.png)

Figure 1: Comparison of step response from algorithm 1 (thin) and algorithm 2 (thick).

![Control action](image2.png)

Figure 2: Control action with algorithm 1 (thin) and algorithm 2 (thick).

We see that algorithm 2 outperforms algorithm 1 here. The reason is easily understood; as \( x \) increases, the gain (from \( u \) to \( \ddot{x} \)) of the system decreases. Since algorithm 2 use future
linear models of the plant along the predicted trajectory, it is aware of the fact that the gain will decrease as we approach the setpoint, and is therefore somewhat restrictive on the use of control action. It is easy to realize that we need a prediction horizon of around 10 steps to achieve this.

Of course, it is possible to tune the parameters to generate a better response from algorithm 1. After some tuning, a new setup is

<table>
<thead>
<tr>
<th>$N$</th>
<th>$M$</th>
<th>$Q_e$</th>
<th>$Q_i$</th>
<th>$Q_{abs}$</th>
<th>$T$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0.01</td>
<td>0.00001</td>
<td>0.1</td>
<td>0.75</td>
</tr>
</tbody>
</table>

We see that the overshoot in the response for algorithm 1 is significantly improved, but the response is slower. Algorithm 2 has lost its previously good behavior with this tuning. We also see that the control capacity is not used in an efficient manner anymore with this tuning.

**Figure 3:** Step responses from algorithm 1 (thin) and algorithm 2 (thick) when tuning has been done to improve response from algorithm 2.

**Figure 4:** The control capacity is not used in a efficient manner with the new tuning.

### 5.2 Catalytic reaction in a CSTR

This example is often used in the chemical process literature as a benchmark problem. The task is to control concentration and level in a highly nonlinear process. The dynamic model
is given as

\[
\begin{align*}
\dot{x}_1 &= u_1 + u_2 - 0.2x_1^{1/2} \\
\dot{x}_2 &= (24.9 - x_2)x_1 + (0.1 - x_2)x_2 - \frac{1}{(1 + x_2)^2} \\
y_1 &= x_1 \\
y_2 &= x_2
\end{align*}
\]

(59) \hspace{1cm} (60) \hspace{1cm} (61) \hspace{1cm} (62)

and there are constraints on the control

\[
\begin{align*}
0 &\leq u_1 \leq 5 \\
0 &\leq u_2 \leq 5
\end{align*}
\]

(63) \hspace{1cm} (64)

A simulation of a step command from \([y_1 \ y_2] = [40 \ 0]\) to \([100 \ 2.787]\) is shown below. We

![Graph of the CSTR process with algorithms 1 and 2](image)

Figure 5: Step response on CSTR process with algorithm 1 (thin) and algorithm 2 (thick).

see that the proposed algorithm gives a minor improvement in the concentration regulation, with approximately 5 per cent less overshoot.

In the figure with the control actions we see that they differ somewhat, coming from the fact that algorithm 2 has a better knowledge of the future response from the process.

![Graph of the control actions](image)

Figure 6: Control on CSTR process with algorithm 1 (thin) and algorithm 2 (thick).

It would probably be possible to tune the parameters in both algorithms for better performance, but here we only want to show that most often algorithm 2 performs better already with poor tuning.
6 Conclusion

A method for MPC on nonlinear systems has been presented. The method is simple and all steps can be solved with standard algorithms and software. Simulation studies have been done and shows that control of nonlinear systems based on linearizations along future predicted trajectories is possible and gives good performance, and in some cases substantially better performance than schemes based on only one local linearization. The most interesting future extension is how to create an algorithm with guaranteed closed loop stability, and in particularly doing this by explicitly taking the nonlinearities into account.

References


