SSY281
MODEL PREDICTIVE CONTROL

LECTURE NOTES

CHALMERS
UNIVERSITY OF TECHNOLOGY

Division of Systems and Control
Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden

2021-01-18

Preface

These lecture notes are aimed to serve as a complement to the textbooks listed for the course SSY281 Model Predictive Control in the master's program *Systems, control & mechatronics* at Chalmers. The primary textbook by Rawlings and Mayne [1], which is also publicly available in pdf form, has influenced these lecture notes considerably. Other useful texts are [2] and [3], both electronically available, and the previously used [4]. The lecture notes, which have been continuously developed since the course was started in 2012, were initially prepared to provide material not covered by the textbooks. Today, the notes can be seen as a condensed text for the material appearing during the lectures; each section in the notes indeed corresponds to one lecture. Since the notes are a bit condensed, the reader is advised to consult the textbooks for further details and, not the least, for more examples and case studies. References to the literature are also found in the books.

The lecture notes include "summary slides" (identified in these notes by a framed number), which to some extent are used in the lectures to complement a more conventional "talk and chalk" lecture style. The slides are uploaded to the course homepage.

The lecture notes were originally written by Bo Egardt. Useful comments and suggestions on the manuscript from Paolo Falcone and Sébastien Gros are gratefully acknowledged. Latest revision is provided by Nikolce Murgovski.

Notes on revisions

Gothenburg, 2021-01-18
Nikolce Murgovski

Contents

Table 1: Notation

	Basic math symbols
$\mathbb{N}, \mathbb{N}_+$	The set of (positive) integers
$\mathbb{R}, \mathbb{R}_+$	The set of (nonnegative) real numbers
$\mathbb{C}$	The set of complex numbers
$\mathbb{R}^n$	Euclidian space of dimension n
$\mathbb{R}^{m\times n}$	The set of m by n real matrices
$\dim$	The dimension of a vector
rank	The rank of a matrix
diag	Diagonal matrix
$\mathcal{K}_\infty$	The class of $\mathcal{K}_\infty$ functions
$\equiv$	Equality by definition
$\mathcal{D}$	Domain of a function
$\mathrm{dom}\, f$	Domain of the function f
int	Interior of a set
$\mathcal{S}$	General set
	Signals
x	State vector x
x^+	Next value of the state x
$\dot{x}, \dot{x}(t)$	Derivative of x
$x_{\min}, x_{\max}$	Lower and upper limit of x
u	Control input vector
$u_{\min}, u_{\max}$	Lower and upper limit of u
Δu	Control move (increment of u)
$\Delta u_{\min}, \Delta u_{\max}$	Lower and upper limit of Δu
y	Output vector
r	Reference signal
	RHC variables and parameters
$\boldsymbol{x}$	Vector of future state variables
u	Future control input
$\boldsymbol{u}$	Vector of future control inputs
Δu	Future control move
$\boldsymbol{\Delta u}$	Vector of future control moves
$\hat{y}$	Predicted output
$\boldsymbol{y}$	Vector of predicted outputs
y_f	Free response
$\boldsymbol{y_f}$	Vector of free response values
r	Future reference signal
$\boldsymbol{r}$	Vector of reference variables
N, M	Prediction and control horizons
$Q, \bar{Q}$	State penalty matrices in LQ
$R, \bar{R}$	Control penalty matrices in LQ
Ω, Γ	Matrices in LQ cost expression

	Optimal control symbols
V_N	Cost function over horizon N
V_N^*	Optimal cost function (or value function) over horizon N
$V_{k\to N}^*$	Optimal cost-to-go from time k to time N
V_∞	Cost function over infinite horizon
V_∞^*	Optimal cost function (value function) over infinite horizon
$\mathbb{X}$	State constraint set
$\mathbb{X}_f$	Terminal state constraint set
$\mathbb{U}$	Control constraint set
$\mathbb{Z}$	Constraint set in (x, u)-space
$\mathcal{U}_N$	Implied control constraint set with horizon N
$\mathcal{X}_N$	Feasible set of initial states with horizon N
$u(0:N-1)$	Sequence of future controls
$u(0:N-1;x)$	Ditto as function of initial state x
$u^*(0:N-1)$	Optimal sequence of future controls
$u^*(0:N-1;x)$	Ditto as function of initial state x
$u(0:\infty)$	Infinite sequence of future controls
$u^*(0:\infty)$	Optimal infinite sequence of future controls
$x(0:N)$	Sequence of future states
$x(0:N;x)$	Ditto as function of initial state x
$x^*(0:N)$	Optimal sequence of future states
$x^*(0:N;x)$	Ditto as function of initial states x
	Feasibility related set symbols
$\mathcal{C}, \mathcal{C}_\infty$	(Maximal) control invariant set
$\mathcal{K}_i(\mathcal{S}), \mathcal{K}_\infty(\mathcal{S})$	The i-step/maximal controllable set with target set $\mathcal{S}$
Pre $(\mathcal{S})$	The set of predecessor states of $\mathcal{S}$
	State estimation symbols
$\hat{x}$	State estime
L	Observer gain matrix
V_T	State estimation cost function
$\hat{V}_T$	Moving horizon state estimation cost function
$x(0:T)$	Sequence of state estimates
$\hat{x}(0:T)$	Sequence of optimal state estimates
$x(k-T:k)$	Sequence of moving horizon state estimates
$\mathbb{W}$	Process disturbance constraint set for MHE
$\mathbb{V}$	Output disturbance constraint set for MHE
$\sim \mathcal{N}$	Normally distributed
	General optimisation symbols
x^*	Optimal solution
p^*	Optimal value of primal problem
d^*	Optimal value of dual problem
$\mathcal{S}$	Feasible set
$\mathcal{L}$	Lagrangian
$\nabla_x \mathcal{L}, \nabla_x^2 \mathcal{L}$	Gradient and Hessian of the Lagrangian
μ, λ	Lagrange multipliers
$q(\mu, \lambda)$	Lagrange dual function
$\nabla f, \nabla g, \nabla h$	Gradients

$\nabla^2 f$	Hessian
$\mathbb{A}$	Set of active constraints
$\bar{\mathbb{A}}$	Set of inactive constraints
$\sim \mathcal{N}$	Normally distributed
$\Delta x, \Delta\lambda, \Delta\mu$	Algorithm updates
$\mathcal{X}$,$\mathcal{U}$, $\mathcal{Z}$	Parametric programming sets

1 Introduction and course overview

The objective of the first section is to give an introduction to Model Predictive Control (MPC). The aim is to provide a motivation for our study of MPC and to give a preview of the course contents. The chapter is introductory and lists questions that will be answered later in the course!

Chapter 1 in [2] is short, but gives a brief introduction to constrained control and estimation in a form which is relevant for our course. Chapter 1 in [4] gives a nice and broader introduction with reference to applications, a bit of history and some examples.

1.1 Background and motivation

Model predictive control goes back to the 1970s, when some of the basic ideas were introduced and applied to process control. The early development was carried out in petro-chemical industry as a response to some of the needs met in practice. A significant attribute of processes in this industry is that many of the process units operate (or rather, should operate) close to limits in order to optimise production throughput or efficiency, to achieve the best product quality, or to comply with safety limits. Hence, techniques to extend the classical linear control methods to this situation were looked for.

Linear control algorithms are not really aimed to operate close to limits or constraints. However, in practice there will always be constraints. The most obvious constraints are limits on the control inputs—any actuator has its minimum and maximum values—and often the rate of change (the *slew rate*) of the control signal is also limited. Another type of constraint is associated with process outputs or states—temperatures and pressures may not exceed certain limits, or a specific product property such as consistency or concentration has to be within limits. In fact, it is not uncommon that, while respecting such constraints, it is still desirable to operate close to some of them.

Because of the sheer scale of the petrochemical processes, even small improvements of control can pay off with remarkable economic results. This inspired some of the pioneers in the field to develop control algorithms that involved solving optimisation problems in real-time. One such algorithm is the famous *Dynamic Matrix Control*, DMC, which we will look into later in the course. The algorithms were developed without much supporting theory, but were still applied successfully in practice. One important reason for the success was the fact that the slow time scale of the processes allowed the computations to be carried out in real-time, despite the limited computational power available at the time.

MPC has evolved over the years, since its first appearance in the 70s. In addition to the ability to handle constraints, an important property of MPC is that it is applicable to multi-input, multi-output (MIMO) processes. This means that MPC is a viable option for extending the basic SISO functionality of PID regulators. In process industry, MPC is often located at the next higher level in the overall control hierarchy. As will be explained further during the course, MPC is also *model-based* in a very explicit way, a property that is commonly desired in current control design methodology.

The last 15-20 years have led to remarkable advances of MPC theory, techniques, and applications. Theoretical advances have led to improved understanding of MPC properties, in particular stability, and the techniques have been extended to nonlinear systems. The range of applications has developed immensely, far beyond the initial process control applications. An important enabling factor behind this development is that numerical computations, which are at the core of MPC, have become very efficient. This depends partly on the general development of computer technology, but another—perhaps even more important—factor is that the numerical algorithms have been developed to become extremely efficient. Again, we will come back to this at much more detail later on.

The origin of model predictive control (MPC) and some of its current driving forces can be summarised as follows:

- The origin:

  - Process control in petrochemical industry (1970s).
  - Dynamic Matrix Control (DMC).

- Early drivers:
  - Operation close to limits $\Rightarrow$ linear control shortcomings.
  - Large returns on small improvements in e.g. product quality or energy/material consumption.
  - Slow time scale allows time-consuming computations.

- Limitations of linear control design:
  - Saturation on control and control rates.
  - Process output limitations.
  - Buffer control (zone objectives).

- Important attributes of MPC:
  - MPC handles actuator limitations and process constraints.
  - MPC handles multivariable systems.
  - MPC is model based (step responses, state-space models etc.).

- Current driving factors:
  - Development of theory and new application areas.
  - Improvements in numerical computations.

Limitations of linear design

The step from classical linear control design to MPC is fairly large—this is one reason for offering this course—so a natural question to ask is if it would be possible to extend the linear design in some way in order to handle constraints. This is discussed in Chapter 1 of the textbook [2]. One possible way to go would be to apply a *cautious* design, i.e. to design the linear controller in such a way as to avoid the constraints, at least in normal operation. This can in principle be done to handle control constraints, even though performance would typically be compromised, but it would mean that operation close to process state constraints is not possible, a consequence not desired.

Another possibility is to extend the linear control design with heuristic add-ons that come into play close to constraints. Possible functions would be to change controller parameters close to limits, to device anti-windup schemes, or to switch in some fall-back strategies. All these and many other heuristic add-ons are indeed used in practice. MPC is distinguished from all these approaches in that constraints are taken into account in a systematic way as part of the control design.

Example 1.1 (MPC vs. LQ control [2])**.** Consider the linear system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned}$$

with

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix},$$

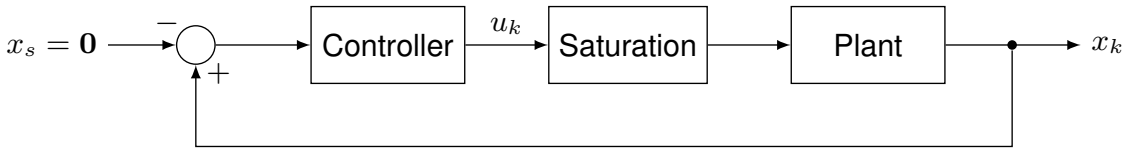


Figure 1: Feedback control loop with input saturation.

which is the zero-order hold discretisation with sampling period $h = 1\,\mathrm{s}$ of the double integrator

$$\frac{d^2y(t)}{dt^2} = u(t).$$

Initial state is set to $x_0 = \begin{bmatrix} -6 & 0 \end{bmatrix}^\top$ and the goal is to drive the state to $x_s = \mathbf{0}$ with a feedback control law as illustrated in Figure 1.

The control input is to be kept within bounds $u_k \in [-1, 1], \forall k$. For safety reasons, a saturation function is imposed

$$\mathrm{sat}(u) \triangleq \begin{cases} 1, & u > 1 \\ u, & |u| \leq 1 \\ -1, & u < -1. \end{cases}$$

We will consider three different control designs, which will be referred to as *cautious*, *serendipitous*, and *tactical*.

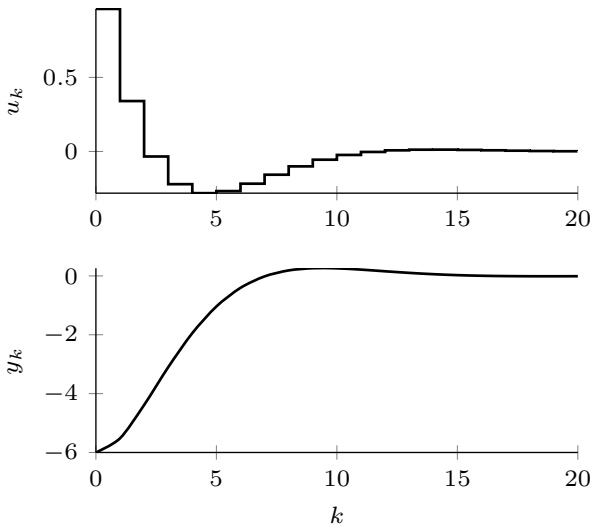
Cautious control. We may design a controller that minimizes a quadratic objective

$$V_N(\{x_k\}, \{u_k\}) = \frac{1}{2} x_N^\top P_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k)$$

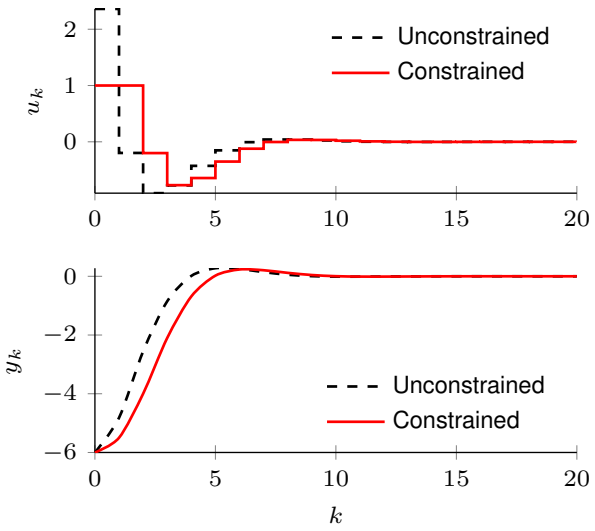
the details of which will be described later, in Section 3. Let us set, for simplicity, $P_f = 0$, $Q = C^\top C$ and $N \to \infty$. The value of R can be chosen cautiously, such that the control action stays within bounds. Indeed, this can be achieved with $R = 20$, for which the optimal feedback control low is $u_k = -Kx_k$, with $K = \begin{bmatrix} 0.1603 & 0.5662 \end{bmatrix}$. The optimal control actions and the system output are depicted in Figure 2a. We can see that control bounds are satisfied for this initial state, but the achieved output response is rather slow, as the *settling time* is in the order of 8 samples.

Serendipitous control. In order to decrease the settling time, we now try reducing the action weight to $R = 2$, while keeping the same Q matrix. Results are shown in Figure 2b where it can be seen that the settling time is indeed reduced to about 5 samples. However, we can also see that the saturation function is very important for this controller, since without it the constraints would be violated. We call this control law serendipitous since no special considerations of the presence of the constraints have been made while designing the controller.

We may now test our fortune by reducing the action weight even further, to $R = 0.1$. As can be seen in Figure 3a the unconstrained control indeed reduces the settling time to about 3 samples, but the control action seriously violates the constraints. After saturating the control action, we can see in Figure 3a that our luck has run out, since the system output experiences large overshoot and the settling time is actually increased to about 12 samples.

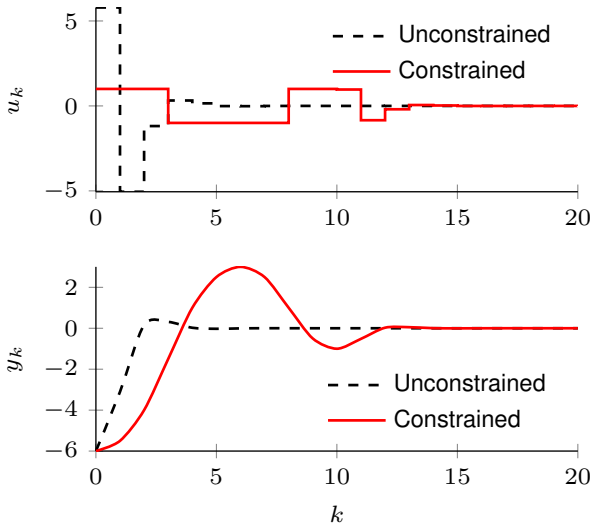


(a) Cautious design with $u_k = -Kx_k$ and $R = 20$.

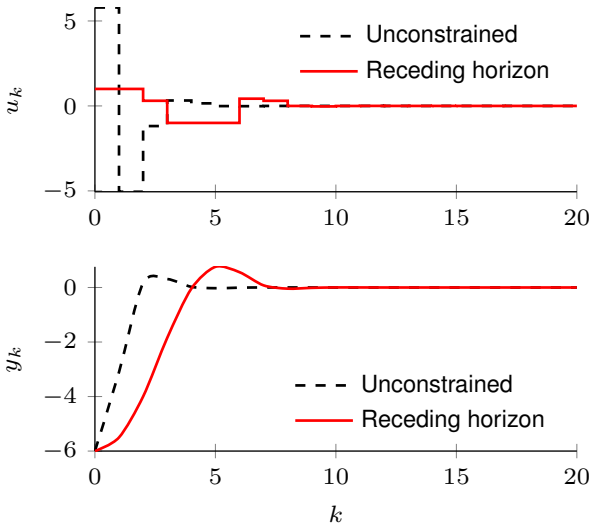


(b) Serendipitous design for constrained and unconstrained LQR with $u_k = -Kx_k$ and $u_k = -\text{sat}(Kx_k)$, respectively, and $R = 2$.

Figure 2: LQ control of the double integrator system.



(a) Serendipitous design for constrained and unconstrained LQR with $u_k = -Kx_k$ and $u_k = -\text{sat}(Kx_k)$, respectively, and $R = 0.1$.



(b) Unconstrained LQR with $u_k = -Kx_k$ and receding horizon control with $R = 0.1$.

Figure 3: Comparison between LQ and receding horizon control of the double integrator system.

Tactical control. Finally, we implement a receding horizon control by regarding the constraints within the design process. We use the same control weight, $R = 0.1$, and a prediction horizon of $N = 2$ samples. The details of the control design will be described later, in Section 4. Results are shown in Figure 3b, where it can be seen that low settling time is achieved, while keeping the control action within bounds. ■

1.2 The receding horizon idea

Model predictive control, or simply MPC, is the term that will most often be used during the course to designate the class of control algorithms investigated. Another popular and almost synonymous term is *receding horizon control*, which explicitly points to the actual core of MPC, the *receding*

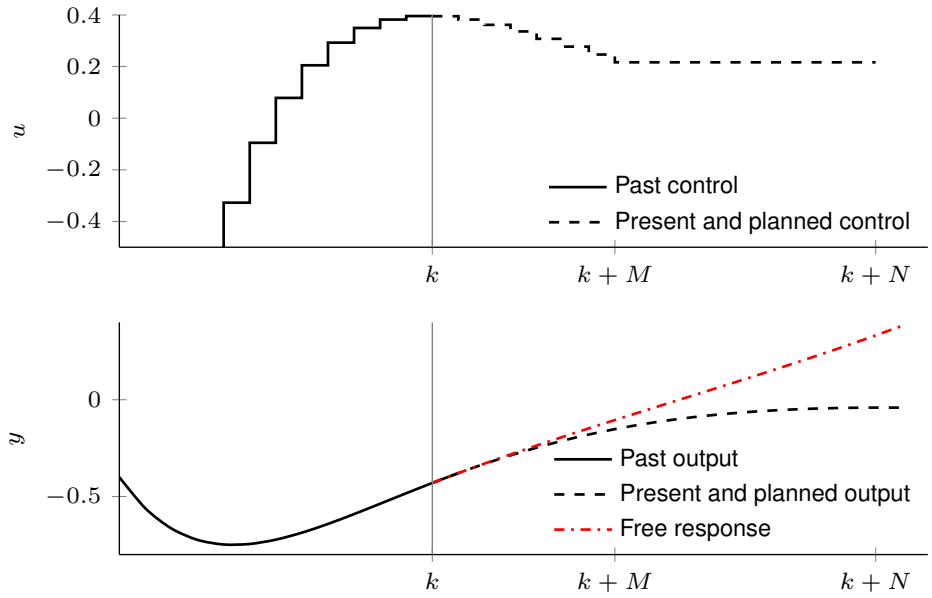


Figure 4: An illustration of a receding horizon control. The free response is the future response when the control signal stays at its current level.

horizon idea. The starting point is as follows: at the current sampling time, the controller takes into account what is known about the process *now* and what can be forecasted or predicted over some *future* sampling intervals, the *horizon*, including the chances of "running into" any constraints. A decision is taken concerning what control action to take, typically based on a significant amount of computations. At the next sampling interval, the procedure is repeated with a horizon that has now been advanced one sampling interval into the future—this is what motivates the term **receding horizon control**. We can summarise what has been said as follows:

1. At time instant k, *predict* the process response over a finite **prediction horizon** N; this response depends on the sequence of future control inputs over the **control horizon** M.

2. Pick the control sequence which gives the best performance in terms of a specified **objective**, **cost function** or **criterion**.

3. Apply the first element in the control sequence to the process, discard the rest of the sequence, and return to step 1.

Figure 4 illustrates the receding horizon idea, where the sampling instant is k. The picture also introduces the term **free response**, which is used to denote the future response in case the control signal stays at its current level. In the figure, it is assumed that the control horizon M is smaller than the prediction horizon N; this is not always the case. Notice that in this example it is assumed that the control stays constant beyond the control horizon; this is a common assumption, and we will return to this later.

Example 1.2 (Receding horizon control of an integrator system)**.** Consider the following system, a simple integrator with input u and output y,

$$y(k+1) = y(k) + u(k).$$

We will investigate the details of receding horizon control for this simple system *without constraints*. Viewing the system at time k, the sequence of past inputs $\{\dots, u(k-2), u(k-1)\}$ and outputs $\{\dots, y(k-1), y(k)\}$ are assumed to be known (note that $y(k)$ is known but not $u(k)$!). The model can then be used to *predict* the output of the system over the *prediction horizon* N, here chosen to be

equal to 2. Introduce the notation $\hat{y}(k+1|k)$ for the predicted output at time $k+1$, given information at time k, and similarly for $\hat{y}(k+2|k)$. The predicted outputs can be expressed in terms of future control inputs $u(k|k), u(k+1|k)$, or equivalently in future changes in control inputs $\Delta u(k|k), \Delta u(k+1|k)$. To summarise, we have the following:

- **System** at time k, with output dynamics: $y(k+1) = y(k) + u(k)$.
- **Past inputs**: $\{\dots, u(k-2), u(k-1)\}$.
- $u(k+1|k)$: **planned (future) control input** at time $k+1$, given information at time k.
- $\Delta u(k+1|k) = u(k+1|k) - u(k|k)$: **control increment**.
- $\hat{y}(k+1|k)$: **predicted output** at time $k+1$ given information at time k.
- **Prediction horizon**: $N = 2$.
- $r(k)$: a **reference signal** to be followed by the output.

The predicted outputs can be written as:

$$\begin{aligned}
\hat{y}(k+1|k) &= y(k) + u(k|k) = y(k) + u(k-1) + \Delta u(k|k) = y_f(k+1|k) + \Delta u(k|k) \\
\hat{y}(k+2|k) &= y(k) + u(k|k) + u(k+1|k) = y(k) + 2u(k|k) + \Delta u(k+1|k) \\
&= y(k) + 2u(k-1) + 2\Delta u(k|k) + \Delta u(k+1|k) \\
&= y_f(k+2|k) + 2\Delta u(k|k) + \Delta u(k+1|k)
\end{aligned}$$

where $y_f(\cdot|k)$ is the **free response**, i.e. the predicted output if the control input is frozen at time k to its last value $u(k-1)$. We will illustrate the receding horizon controller in a couple of different cases.

Case 1: control horizon of $M = 1$. Only one future control input is to be chosen, and we assume that the control stays constant after that, i.e. $\Delta u(k+1|k) = 0$. A natural criterion for having future outputs close to the reference signal

$$\begin{aligned}
V_2 &= (\hat{y}(k+1|k) - r(k+1))^2 + (\hat{y}(k+2|k) - r(k+2))^2 \\
&= (y_f(k+1|k) + \Delta u(k|k) - r(k+1))^2 + (y_f(k+2|k) + 2\Delta u(k|k) - r(k+2))^2.
\end{aligned}$$

We can solve for the optimal value by differentiating V_2, and set the derivative equal to zero:

$$\frac{\partial V_2}{\partial \Delta u(k|k)} = 2(y_f(k+1|k) + \Delta u(k|k) - r(k+1)) + 2(y_f(k+2|k) + 2\Delta u(k|k) - r(k+2)) \cdot 2 = 0.$$

This gives the optimal (incremental) control (remember that $u(k) = u(k-1) + \Delta u(k|k)$)

$$\Delta u(k|k) = \frac{1}{5}\left((r(k+1) - y_f(k+1|k)) + 2(r(k+2) - y_f(k+2|k))\right). \tag{1}$$

Notice that the control increment will be 0, i.e. the control signal stays where it is, if the free response coincides with the reference signal.

Alternatively, we could describe this procedure as trying to solve the system of linear equations

$$\begin{cases}
\hat{y}(k+1|k) = y_f(k+1|k) + \Delta u(k|k) = r(k+1) \\
\hat{y}(k+2|k) = y_f(k+2|k) + 2\Delta u(k|k) = r(k+2)
\end{cases}$$

or, using vector notation,

$$\begin{bmatrix} y_f(k+1|k) \\ y_f(k+2|k) \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Delta u(k|k) = \begin{bmatrix} r(k+1) \\ r(k+2) \end{bmatrix} \Leftrightarrow$$
$$\boldsymbol{y_f} + \Theta \Delta u(k|k) = \boldsymbol{r}.$$

Since there is only one variable to fulfil two conditions, a natural solution is to find the value of $\Delta u(k|k)$ that solves this system of linear equations in a least-squares sense, i.e. by minimizing $\|\Theta \Delta u(k|k) - (\boldsymbol{r} - \boldsymbol{y_f})\|^2$. Expressing this in Matlab notation gives

$$\Delta u(k|k) = \Theta \backslash (\boldsymbol{r} - \boldsymbol{y_f}) = (\Theta^\top \Theta)^{-1} \Theta^\top (\boldsymbol{r} - \boldsymbol{y_f}), \tag{2}$$

which is identical to the solution (1).

Case 2: control horizon of $M = 2$. There are two free variables. Hence, with the system of linear equations interpretation, the conditions to be fulfilled now become

$$\begin{bmatrix} y_f(k+1|k) \\ y_f(k+2|k) \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \end{bmatrix} = \begin{bmatrix} r(k+1) \\ r(k+2) \end{bmatrix}$$

which can now be solved uniquely for the optimal control sequence:

$$\begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}^{-1} (\boldsymbol{r} - \boldsymbol{y_f}). \tag{3}$$

Sticking to the receding horizon principle, only the first element in the optimal control sequence is used, namely $\Delta u(k|k)$, and the other element is discarded.

It should finally be stressed that this example is very simplified, and the main purpose is to clarify the character of the solution and the receding horizon principle. In more realistic examples, we would expect M and N to be chosen significantly larger, sometimes equal, sometimes with M smaller than N. In addition, constraints would be introduced in most cases. ■

Summary

The MPC recipe for the example:

1. At time k, predict the output N samples ahead:

$$\hat{y}(k+1|k), \ldots, \hat{y}(k+N|k).$$

2. The predictions depend on future control inputs

$$u(k|k), u(k+1|k), \ldots, u(k+M-1|k).$$

(Normally, $M \leq N$, and we assume that u is either 0 or unchanged after this.)

3. Minimize a criterion (now adopting the index notation as in Matlab)

$$V(k) = V(\hat{y}(k+1:k+N|k), u(k:k+M-1|k))$$

with respect to the control sequence $u(k:k+M-1|k)$.

4. Apply the first control signal in the sequence to the process:

$$u(k) = u(k|k).$$

5. Increment time $k := k + 1$ and go to 1.

It can be noticed that items 1 and 2 above require a process *model*, which can be used for *predictions*. Item 3 assumes that we formulate an optimisation *objective* and that we have a relevant *optimisation algorithm* to minimise this objective. Item 4 stresses the fundamental *receding horizon* principle. Based on the example, we can now summarize the main ingredients of model predictive control; these will be the focus for our study during the course with the objective to generalize the ideas described in the simple example:

MPC ingredients

- An **internal model** describing process and disturbances.
- An **estimator/predictor** to determine the evolution of the state.
- An **objective/criterion** to express the desired system behaviour.
- An **online optimisation algorithm** to determine future control actions.
- The **receding horizon** principle.
- Our focus: linear models, quadratic criteria with linear constraints.

2 Review and preliminaries

The second section takes a closer look at some of the basic ingredients of MPC as discussed in the first section. A review is given of some concepts and results from previous courses that turn out to be useful. The aim of the section is to give a clear picture of what is needed later on during the course; you are advised to go back and refresh this material if needed. The textbooks basically assume that this material is known.

Section 1.2 and Appendix A in [1] contain some bits and pieces of this material, as does Sections 5.2 and 5.4 of [2]. The discussions on how state variables can be chosen to include *control moves* Δu instead of absolute control u, and on how computational delays may be included in the model are found in Section 2.4-2.5 of [4].

2.1 State space models

The first important ingredient in model predictive control is an *internal model*. Linear state space models will be used throughout the course. The emphasis will be on discrete-time state space models, but often these models originate from continuous-time models of the process, sometimes obtained by linearising the original non-linear model.

Continuous State space model. We will study continues state space models of the form

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_y x(t) \\ z(t) &= C_z x(t) \end{aligned} \tag{4}$$

where

$x = (x_1, \ldots, x_n)$ is the state vector
$u = (u_1, \ldots, u_m)$ is the control input vector
$y = (y_1, \ldots, y_{p_y})$ is the vector of **measured outputs**
$z = (z_1, \ldots, z_{p_z})$ is the vector of **controlled outputs**.

Often, $z = y = (y_1, \ldots, y_p)$, and then we simply write $y(t) = Cx(t)$.

Discrete state space model. Assuming that the control signal is constant over the (constant) sampling interval, we may obtain an exact representation of the above system at the sampling instants. The solution of (4) with initial condition $x(t_0)$ is

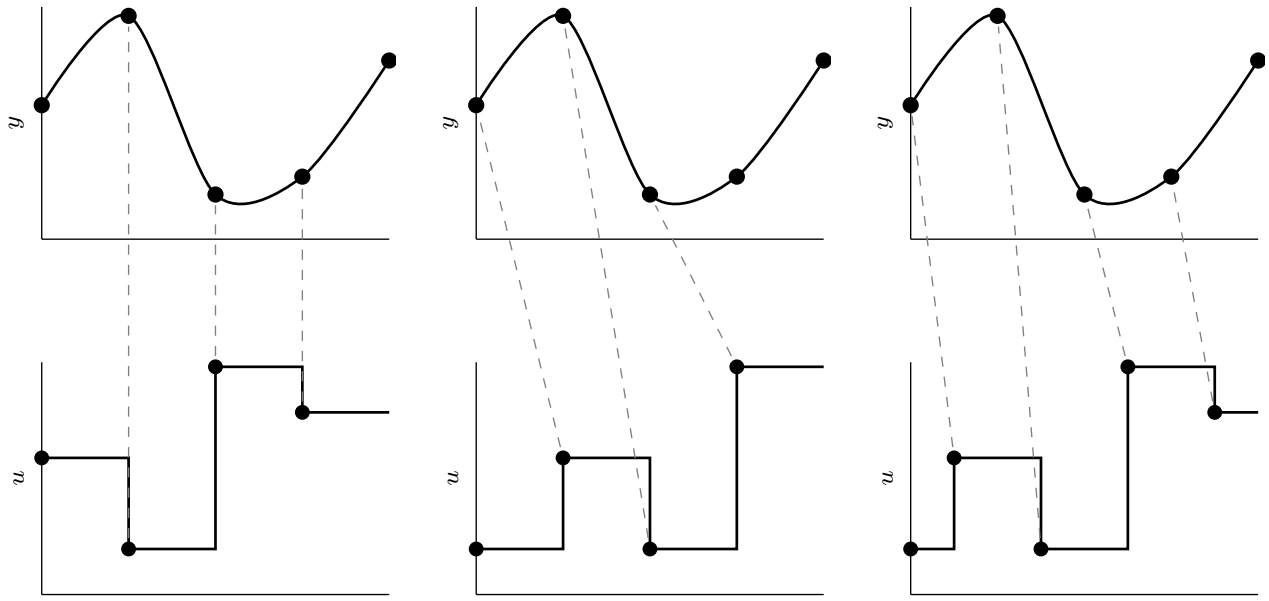
$$x(t) = e^{A_c(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A_c(t-s)}B_c u(s)ds. \tag{5}$$

Assume that the control signal u is piecewise constant (h is the sampling interval),

$$u(t) = u(kh), \quad kh \le t < (k+1)h.$$

By using this in (5) with $t = (k+1)h$ and $t_0 = kh$, we get the discrete time state equation

$$x(k+1) = e^{A_c h}x(k) + \left(\int_0^h e^{A_c s}B_c\, ds\right) u(k) = Ax(k) + Bu(k),$$



(a) Instantaneous computation. (b) Full sampling interval delay. (c) Fractional delay.

Figure 5: Control action and computational delay.

where, for simplicity of notation, h has been omitted from the time argument. The output equation of (4) is the same (but with t replaced by k). A compact version of the **discrete state-space model** in the case $z = y$ is

$$\begin{aligned} x^+ &= Ax + Bu \\ y &= Cx. \end{aligned}$$

In the discrete-time representation derived above, it is assumed that the control signal $u(k)$ is instantaneously available at the sampling instant, so that it can be applied to the process immediately. Sometimes, this may be unrealistic, in particular if the control algorithm involves significant computation, as in the MPC case. One way to solve this is to wait until the *next sampling instant* to deliver the control signal to the process, as illustrated in Figure 5b. This will, however, introduce an additional time delay of one sampling interval, which may be undesirable. A third alternative, shown in Figure 5c, is to allocate a fraction of the sampling interval to the computations, and then deliver the new control signal to the process *between two consecutive sampling instants*. The model obtained will differ compared to the other alternatives.

Computational delay. Allowing for a computational delay τ, the control signal is still piecewise constant but now given by

$$u(t) = \begin{cases} u(k-1), & kh \le t < kh + \tau \\ u(k), & kh + \tau \le t < (k+1)h. \end{cases}$$

The solution (5) is now obtained in two steps (corresponding to the two different input signal levels) as

$$\begin{aligned}
x(k+1) &= e^{A_c(h-\tau)}x(kh+\tau) + \int_0^{h-\tau} e^{A_c s}B_c\,ds\cdot u(k) \\
&= e^{A_c(h-\tau)}\left(e^{A_c\tau}x(kh) + \int_0^{\tau} e^{A_c s}B_c\,ds\cdot u(k-1)\right) + \int_0^{h-\tau} e^{A_c s}B_c\,ds\cdot u(k) \\
&= e^{A_c h}x(kh) + e^{A_c(h-\tau)}\int_0^{\tau} e^{A_c s}B_c\,ds\cdot u(k-1) + \int_0^{h-\tau} e^{A_c s}B_c\,ds\cdot u(k) \\
&= Ax(k) + B_1u(k-1) + B_2u(k).
\end{aligned}$$

This can be put in a standard form by introducing the augmented state vector

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}.$$

The new state space model becomes

$$\xi(k+1) = \begin{bmatrix} x(k+1) \\ u(k) \end{bmatrix} = \begin{bmatrix} A & B_1 \\ 0 & 0 \end{bmatrix}\xi(k) + \begin{bmatrix} B_2 \\ I \end{bmatrix}u(k).$$

An alternative model compared to the ones described above is obtained by focusing on *control signal changes* as opposed to absolute control values. It is not difficult to get the relevant state space model for **incremental control**,

$$\begin{aligned}
x^+ &= Ax + Bu \\
y &= Cx.
\end{aligned}$$

Introduce the incremental control (or *control move*)

$$\Delta u(k) = u(k) - u(k-1)$$

and the augmented state vector

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}.$$

This leads to the new model

$$\begin{aligned}
\xi^+ &= \mathcal{A}\xi + \mathcal{B}\Delta u \\
y &= \mathcal{C}\xi
\end{aligned}$$

with

$$\mathcal{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} B \\ I \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} C & 0 \end{bmatrix}.$$

2.2 Setpoint tracking and regulation

Consider the system

$$x^+ = Ax + Bu \tag{6}$$
$$y = Cx. \tag{7}$$

A steady state (x_s, u_s) of the system satisfies the equation

$$\begin{bmatrix} I - A & -B \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = 0.$$

The task to bring the system output y to a desired, constant setpoint y_{sp} is termed **setpoint tracking**. At steady state, this requires $Cx_s = y_{sp}$ and the condition for setpoint tracking becomes

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ y_{sp} \end{bmatrix}. \tag{8}$$

This is a system of $n + p$ equations with $n + m$ unknowns.

We will return shortly to the solution of the system of equations above. Assume for a moment, however, that the system of equations has a solution. It then turns out that it is relatively straightforward to translate the setpoint tracking problem into a **regulation problem for deviation variables**. This is why we, in the next couple of sections, will mainly focus on the regulation problem, for which the task is to steer the system state to the origin.

Assume that equation (8) holds. Then the deviation variables

$$\begin{aligned} \delta x(k) &= x(k) - x_s \\ \delta u(k) &= u(k) - u_s \end{aligned}$$

satisfy the following dynamics

$$\begin{aligned} \delta x(k+1) &= Ax(k) + Bu(k) - x_s = Ax(k) + Bu(k) - (Ax_s + Bu_s) = A\delta x(k) + B\delta u(k) \\ \delta y(k) &= y(k) - y_{sp} = Cx(k) - Cx_s = C\delta x(k). \end{aligned}$$

Thus, the deviation variables satisfy the same state equation as the original variables.

Going back to the solution of (8), we will first refresh some basic facts about solutions to systems of linear equations. For any $m \times n$ matrix $A \in \mathbb{R}^{m\times n}$ we have,

Rank of A :	$\operatorname{rank}(A) = \operatorname{rank}(A^\top) = r.$	
Range of A :	$\mathcal{R}(A) = \{Ax \mid x \in \mathbb{R}^n\}$	$\dim \mathcal{R}(A) = r.$
Nullspace of A :	$\mathcal{N}(A) = \{x \mid Ax = 0\}$	$\dim \mathcal{N}(A) = n - r.$
Orthogonal subspaces:	$\mathcal{R}(A^\top) \perp \mathcal{N}(A)$	$\mathcal{R}(A) \perp \mathcal{N}(A^\top).$

For the system of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{m\times n}$$

we have:

- There exists a solution x for every b if and only if $r = m$, i.e. $\mathcal{R}(A) = \mathbb{R}^m$.
- The solution is unique if and only if $r = n$, i.e. $\mathcal{N}(A) = \{0\}$.

The relations above can be illustrated as in Figure 6.

Sometimes we will be interested in solving a system of linear equations *approximately*. This happens, for example, when there are more equations than unknowns, i.e. $m > n$, in which case necessarily $r < m$. We have already encountered this situation in Example 1.2, where reference to Matlab's backslash operator was made. The mathematical meaning of this is summarized below.

Consider the *overdetermined* system of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{m\times n}$$

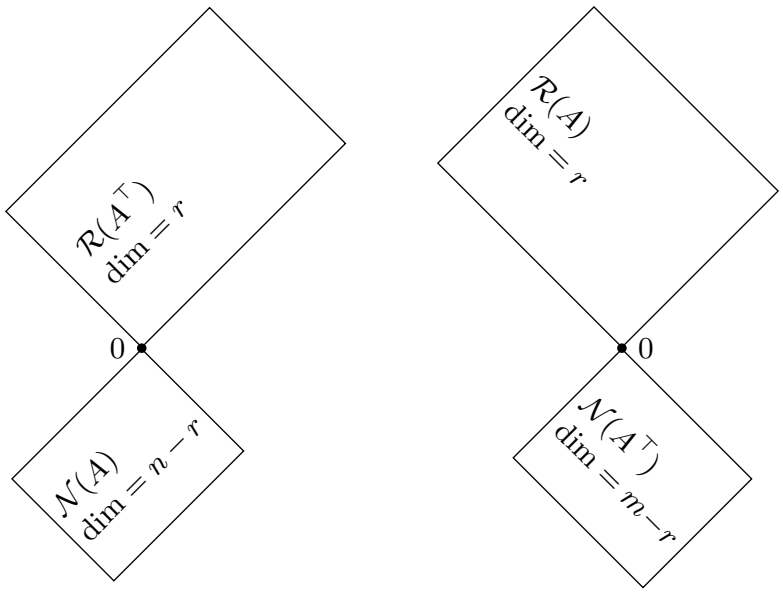


Figure 6: Illustration of range and nullspace of a matrix A and its transpose.

with $m > n$. Assume that A has maximal rank $r = n$. Then the solution to the minimization problem

$$\min_{x \in \mathbb{R}^n} |Ax - b|$$

is given by

$$x^* = A^\dagger b$$

where the *pseudo-inverse* $A^\dagger$ is defined by

$$A^\dagger = (A^\top A)^{-1} A^\top.$$

Remark. *$Ax^* = AA^\dagger b$ is the orthogonal projection of b onto $\mathcal{R}(A)$, i.e. x^* is mapped to the vector in $\mathcal{R}(A)$ that is closest to b. In Matlab notation, $x^* = A \backslash b$.*

In other situations we may encounter undetermined problems that have many solutions. This happens when there are less equations than unknowns, i.e. $m < n$. Consider, e.g., the *undetermined* system of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{m \times n}$$

with $m < n$. Assume that A has maximal rank $r = m$. Since the system has many solutions, a common choice is to pick a representative solution, e.g. the one which minimises the least-norm

$$\min_{x \in \mathbb{R}^n,\ Ax = b} x^\top x. \tag{9}$$

Proposition 2.1. *The solution to* (9)*, where A is a full row rank, is*

$$x^* = A^\top (AA^\top)^{-1} b.$$

Remark. *The matrix $A^\top (AA^\top)^{-1}$ is also known as right pseudo-inverse.*

Proof of Poposition 2.1. For the proof we will consider techniques that will be discussed in more details in Section 7.

The solution of problem (9) can be found by minimising the Lagrangian

$$\mathcal{L} = x^\top x + \lambda^\top (Ax - b)$$

where λ is a vector of Lagrange multipliers. Minimum can be found where the derivative is zero, i.e.

$$2x^{*\top} + \lambda^\top A = 0.$$

From here, $x^* = -A^\top \lambda/2$. To find λ, substitute x^* in the equality constraint

$$Ax^* = -\frac{1}{2} AA^\top \lambda = b.$$

This gives $\lambda = -2(AA^\top)^{-1}b$, where we have used the fact that $AA^\top$ is invertible, since A is a full row rank. Finally,

$$x^* = A^\top (AA^\top)^{-1} b.$$

□

Returning to equation (8), we may ask when there exists a solution. It is a system of linear equations with $n + p$ equations and $n + m$ unknowns. If we want the equation to have a solution for any y_{sp}, the matrix on the left hand side must have rank $n + p$. Hence, we must have $p \leq m$, i.e. we need at least as many inputs as outputs with setpoints. On the other hand, it is not uncommon that we have more measured output variables than manipulated inputs. Then we may choose a subset of the outputs, for which we would like to have setpoint tracking. We refer to these as *controlled outputs* z as in (4). The condition for setpoint tracking now becomes

$$\begin{bmatrix} I - A & -B \\ C_z & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ z_{sp} \end{bmatrix} \tag{10}$$

with $p_z \leq m$. We will return to a more thorough discussion of this in Section 5.

2.3 Estimation, prediction and disturbance modelling

From Section 1 we remind that another important MPC ingredient is an *estimator/predictor* to determine the state evolution. The fundamental result here (which should be known from previous courses) is that the state equation

$$\begin{aligned} x^+ &= Ax + Bu \\ y &= Cx \end{aligned}$$

allows the construction of a *state observer/estimator*

$$\hat{x}^+ = A\hat{x} + Bu + L(y - C\hat{x})$$

which provides estimates $\hat{x}$ of the, usually only partly measurable, state. Defining the state estimation error $\tilde{x} = \hat{x} - x$, the *error dynamics* can readily be found as

$$\tilde{x}^+ = (A - LC)\tilde{x}.$$

An important result is that the eigenvalues of the error dynamics matrix $(A - LC)$ can be freely assigned if the original system is **observable**; assigning purely stable eigenvalues guarantees that $\tilde{x} \to 0,\ t \to \infty$.

Definition 2.1 (Observability)**.** A linear, discrete time system

$$\begin{aligned} x^+ &= Ax \\ y &= Cx \end{aligned}$$

is observable if for some N, any $x(0)$ can be determined from $\{y(0), y(1), \ldots, y(N-1)\}$.

Lemma 2.2 (Observability)**.** *The system is observable if and only if any of the following, equivalent, conditions hold:*

- *The matrix* $\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$ *has full rank* n*.*
- *The matrix* $\begin{bmatrix} \lambda I - A \\ C \end{bmatrix}$ *has rank* n *for all* $\lambda \in \mathbb{C}$*.*

Remark. *A weaker condition is* ***detectability****, which requires that any unobservable modes are strictly stable.*

A similar result as the one above can be used to test for **controllability**.

Definition 2.3 (Controllability)**.** A linear, discrete time system

$$x^+ = Ax + Bu$$

is controllable if it is possible to steer the system from any state x_0 to any state x_f in finite time.

Lemma 2.4 (Controllability)**.** *The system is controllable if and only if any of the following, equivalent conditions hold:*

- *The matrix* $\begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$ *has full rank* n*.*
- *The matrix* $\begin{bmatrix} \lambda I - A & B \end{bmatrix}$ *has rank* n *for all* $\lambda \in \mathbb{C}$*.*

Remark. *A weaker condition is* ***stabilisability*** *which requires that any uncontrollable modes are strictly stable.*

Example 2.5 (Parallel connection of two controlable SISO systems)**.** Assume that we define a new system from the parallel connection of two separate, controllable sub-systems (A_1, B_1, C_1) and (A_2, B_2, C_2), respectively. The complete system is described by the state model

$$\begin{bmatrix} x_1^+ \\ x_2^+ \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u, \quad y = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

To investigate the controllability of the system, we apply the test given above:

$$\begin{bmatrix} \lambda I - A & B \end{bmatrix} = \begin{bmatrix} \lambda I - A_1 & 0 & B_1 \\ 0 & \lambda I - A_2 & B_2 \end{bmatrix} \sim \begin{bmatrix} \lambda I - A_1 & B_1 & 0 \\ 0 & B_2 & \lambda I - A_2 \end{bmatrix}.$$

Now assume that the rows of this matrix are linearly dependent; there are then three distinct cases:

1. The linear dependence is within the first block row—but this contradicts that system 1 is controllable.
2. The linear dependence is within the second block row—but this contradicts that system 2 is controllable.
3. The linear dependence involves rows from both blocks. This means that λ is an eigenvalue of both A_1 and A_2. This case can be ruled out, if we assume that the two systems do not have any pole in common.

■

Setpoint tracking (see Section 2.2), i.e. the ability to have no steady state errors (or *zero offset*) in the controlled (or output) variables, is a desirable property also when the system is subject to constant (but unknown) disturbances. This *disturbance rejection* property is often a basic requirement on the closed-loop system, and is indeed the reason why the integral action in a PID regulator is so common. A standard method to obtain similar properties of a model based control system (like MPC) is to include a model of the constant disturbance in the system model, estimate the disturbance based on this model, and then to counteract the effects of the disturbance in the control. We will return to this later in the course, but let us establish a few facts on the first two steps at once.

A constant disturbance d of dimension n_d can be modelled by the state equation

$$d^+ = d.$$

Augmenting the state model (6) with this disturbance gives the model

$$\begin{aligned} \begin{bmatrix} x \\ d \end{bmatrix}^+ &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} \end{aligned} \tag{11}$$

where the disturbance affects both state and output equations through the matrices B_d and C_d.

In order to be able to estimate the disturbance d in the model above, the state needs to be reconstructed from the measured output. A condition for this is given below.

Proposition 2.2 (Detectability of the augmented system)**.** *The augmented system* (11) *is detectable if and only if the original system is detectable and the following condition holds:*

$$\operatorname{rank}\left(\begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix}\right) = n + n_d \tag{12}$$

Remark. *In order for the rank condition to be satisfied, we must have*

$$n_d \leq p$$

i.e. we must have at least as many measured outputs as the dimension of the disturbance vector.

2.4 Quadratic forms and stability

Quadratic forms are used frequently in the sequel, both to formulate performance objectives and to investigate stability.

A *quadratic form* is given by the expression

$$x^\top S x = |x|_S^2$$

with $x \in \mathbb{R}^n$ and S a real, *symmetric* matrix. The matrix S is *positive definite* ($S \succ 0$) if and only if

$$x^\top S x > 0, \quad \forall \text{ nonzero } x \in \mathbb{R}^n$$

and the matrix S is *positive semidefinite* ($S \succeq 0$) if and only if

$$x^\top S x \geq 0, \quad \forall x \in \mathbb{R}^n.$$

Definition 2.6 (Stability)**.** The linear, discrete time system

$$x^+ = Ax \tag{13}$$

is asymptotically stable (solutions converge to the origin) if and only if **the magnitudes of the eigenvalues of A are strictly less than 1 (A is stable)**.

The evolution of the state trajectory can be studied via the quadratic form

$$V(x) = x^\top S x = |x|_S^2, \quad S \succeq 0$$

which can be interpreted as a generalization of the Euclidean norm $|x|$ (squared). Evaluated along solutions to (13), V changes according to

$$V(x^+) - V(x) = -x^\top (S - A^\top S A)x \equiv -x^\top Q x.$$

Hence, the matrix Q determines whether V will decay or not.

Lemma 2.7 (Stability)**.** *For a stable system, the following conditions are equivalent:*

(a) The matrix A is stable.

(b) For each $Q \succ 0$ there is a unique solution $S \succ 0$ of the discrete Lyapunov equation

$$S - A^\top S A = Q. \tag{14}$$

Remark. *Convergence to the origin under the given conditions follows from*

$$V(x(k)) - V(x(0)) = -\sum_{i=0}^{k-1} x^\top(i) Q x(i) \quad \Rightarrow \quad \sum_{i=0}^{k-1} x^\top(i) Q x(i) \leq V(x(0)).$$

Since the sum is upper bounded, we must have that $x^\top(i) Q x(i) \to 0,\ i \to \infty$.

2.5 LTI systems in Matlab

It is useful to review and refresh some of the basics concerning how to represent, manipulate and analyse LTI systems in Matlab—this will be needed later in the course, both for computer exercises and for the assignments. Here, some commands are used to investigate an example process from the book [4].

Example: Paper machine headbox – Ex 2.4 in [4].

```
% Create continuous time LTI object
Ac = [-1.93   0      0      0;
       0.394 -0.426  0      0;
       0      0     -0.63   0;
       0.82  -0.784  0.413 -0.426 ];
Bc = [ 1.274  1.274;
       0      0;
       1.34  -0.65;
       0      0 ];
Cc = [ 0      1      0      0;
       0      0      1      0;
       0      0      0      1 ];
Dc = zeros(3,2);
csys = ss(Ac,Bc,Cc,Dc);  %create state space model
```

```
% Assign variable names
set(csys,'InputName',{'Stock flowrate';'WW flowrate'},...
    'OutputName',{'Headbox level';'Feed tank conc';'Headbox conc'},...
    'StateName',{'Feed tank level';'Headbox level';...
    'Feed tank conc';'Headbox conc'},...
    'TimeUnit','minutes');
get(csys);


% Compute eigenvalues
eig(Ac);

% Create discrete time model
Ts=2;
dsys=c2d(csys,Ts);

% Compute eigenvalues
eig(dsys.a);
exp(eig(Ac)*Ts);

% Controllability
rank(ctrb(dsys));
rank(ctrb(dsys.a,dsys.b(:,1)));
rank(ctrb(dsys.a,dsys.b(:,2)));

% Observability
rank(obsv(dsys))
rank(obsv(dsys.a,dsys.c(1,:)));
rank(obsv(dsys.a,dsys.c(2,:)));
rank(obsv(dsys.a,dsys.c(3,:)));

% Step response
step(dsys);
```

3 Unconstrained receding horizon control

The objective of this and the following section is to provide some of the conceptual basis for model predictive control. The first step, taken in this section, is to review some basic results on linear-quadratic (LQ) control, and the machinery of dynamic programming (DP). There is a close link between DP and MPC, which will become clearer in the next section when we proceed with a more thorough introduction of the MPC ideas. The presentation here is predominantly influenced by [1].

The textbooks assume that the LQ case is well-known. Section 5.2 in [2] states the problem. The dynamic programming solution can be found in Section 1.3 of [1]. The elements for what we call the batch solution are given by Sections 2.6.1 (prediction) and 3.1 (unconstrained problems) in [4].

3.1 The linear-quadratic (LQ) control problem

We will start to review the LQ problem and a couple of alternative ways to solve it. The finite horizon linear-quadratic control problem for a discrete time, time invariant system concerns steering the states of the system to the origin, given an initial state[1]. The control law is derived by minimizing a criterion which is quadratic in the states and the controls. The LQ problem, which is based on the assumption that the states are available for (perfect) measurements, is summarized as follows:

System:

$$S: \quad x^+ = Ax + Bu. \tag{15}$$

Optimisation problem:

$$P: \quad \min_{u(0:N-1)} V_N(x(0), u(0:N-1))$$

where the minimization is with respect to the sequence of control inputs

$$u(0:N-1) = \{u(0), u(1), \ldots, u(N-1)\}$$

and subject to the system model (15).

The *objective* or *criterion* or *cost function* V_N is given by

$$\begin{aligned} V_N(x(0), u(0:N-1)) &= x^\top(N) P_f x(N) + \sum_{i=0}^{N-1} (x^\top(i) Q x(i) + u^\top(i) R u(i)) \\ &= l_f(x(N)) + \sum_{i=0}^{N-1} l(x(i), u(i)). \end{aligned} \tag{16}$$

Remark. *All $x(i)$ are functions of $x(0)$ and $u(0:N-1)$ via the model* (15)*!*

Remark. *The first term $x^\top(0)Qx(0)$ in the objective is really redundant but is kept for notational convenience.*

The objective V_N is quadratic in x and u with weights Q and R; in addition, the final state $x(N)$ is included in V_N with weight P_f. The matrices Q, R and P_f are all symmetric, Q and P_f positive semidefinite and R positive definite. We will refer to $l(x, u) = x^\top Q x + u^\top R u$ as the **stage cost** and $l_f(x)$ as the **final cost** or **terminal cost**.

The LQ problem can be solved in several different ways. We will first show the solution using the *batch approach* and then apply **dynamic programming** (DP) to derive the solution.

[1] We thus assume for now that the origin is the desired target, e.g. by formulating a model in terms of deviation variables; we will return to the general case later. Also note that the initial time is here chosen to be equal to 0 to keep notation simple; the reformulation for an arbitrary initial time is straightforward.

Batch solution of the LQ problem

In the formulation of the LQ problem above, it is not entirely clear what is meant by "subject to the system model (15)". In fact, in the search for the optimising control sequence $u(0:N-1)$, there are two slightly different approaches. In the first one, *both* the control inputs and the states are considered as optimisation variables, and the state model (15) will then act as an *equality constraint* in the slightly modified formulation:

$$\begin{aligned} \min_{u(0:N-1),x(1:N)} \quad & V_N(x(0), u(0:N-1), x(1:N)) \\ \text{subject to} \quad & x^+ = Ax + Bu. \end{aligned} \tag{17}$$

Notice, however, that in this purely *deterministic* setting (without any uncertainties), the state variables can be determined, once the initial state and the control inputs are defined. In the optimisation formulation

$$\min_{u(0:N-1)} \quad V_N(x(0), u(0:N-1)), \tag{18}$$

the states become instead *expressions* in the initial state and the controls. This is worked out in detail below in the **batch approach**.

Repeated use of the system equation (15) gives

$$\begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N) \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} x(0) + \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(N-1) \end{bmatrix} \tag{19}$$

or, with a more compact notation,

$$\boldsymbol{x} = \Omega x(0) + \Gamma \boldsymbol{u}. \tag{20}$$

The LQ criterion (16) can now be written

$$\begin{aligned} V_N(x(0), \boldsymbol{u}) &= x^\top(0) Q x(0) + \boldsymbol{x}^\top \bar{Q} \boldsymbol{x} + \boldsymbol{u}^\top \bar{R} \boldsymbol{u} \\ &= x^\top(0) Q x(0) + (\Omega x(0) + \Gamma \boldsymbol{u})^\top \bar{Q} (\Omega x(0) + \Gamma \boldsymbol{u}) + \boldsymbol{u}^\top \bar{R} \boldsymbol{u} \\ &= \boldsymbol{u}^\top (\Gamma^\top \bar{Q} \Gamma + \bar{R}) \boldsymbol{u} + 2 x^\top(0) \Omega^\top \bar{Q} \Gamma \boldsymbol{u} + x^\top(0) (Q + \Omega^\top \bar{Q} \Omega) x(0) \end{aligned} \tag{21}$$

where $\bar{Q} = \operatorname{diag}(Q, \ldots, Q, P_f)$ and $\bar{R} = \operatorname{diag}(R, \ldots, R)$. Since $V_N(x(0), \boldsymbol{u})$ is quadratic in $\boldsymbol{u}$, we can solve for the optimal control vector (either by differentiation or by completing the squares):

$$\boldsymbol{u}^* = -(\Gamma^\top \bar{Q} \Gamma + \bar{R})^{-1} \Gamma^\top \bar{Q} \Omega \, x(0)$$

and the optimal **cost-to-go**, or **value function**, is

$$V_N^*(x(0), \boldsymbol{u}) = x^\top(0) \left(Q + \Omega^\top \bar{Q} \Omega - \Omega^\top \bar{Q} \Gamma (\Gamma^\top \bar{Q} \Gamma + \bar{R})^{-1} \Gamma^\top \bar{Q} \Omega \right) x(0).$$

Note that $\boldsymbol{u}^*$ is linear in $x(0)$ and V_N^* is quadratic in $x(0)$!

We finish this section by noting that the formulation (18) sometimes is referred to as the **condensed** version of the problem, since the number of optimisation variables has been reduced relative to the **uncondensed** version (17). In both cases, the result of the optimisation is a sequence of optimal control inputs.

Dynamic programming solution

An alternative solution of the LQ problem is offered via **dynamic programming (DP)**. Dynamic programming is a general optimisation technique, which for us has the advantage to provide useful insights and connections with MPC. DP exploits the particular structure of the problem, which we will now reveal. First note that the variables involved in the problem statement are the control inputs $u(0\!:\!N-1) = \{u(0), u(1), \ldots, u(N-1)\}$ and the states $x(0\!:\!N) = \{x(0), x(1), \ldots, x(N)\}$. These variables are connected via the state equation, and we can illustrate the dependencies using arrows as follows,

$$\begin{array}{ccccccccccc} x(0) & \rightarrow & x(1) & \rightarrow & x(2) & \ldots & x(N-2) & \rightarrow & x(N-1) & \rightarrow & x(N) \\ u(0) & \nearrow & u(1) & \nearrow & u(2) & \ldots & u(N-2) & \nearrow & u(N-1) & \nearrow & \end{array}$$

Let us inspect the cost function V_N in the same way by spelling out the stage costs:

$$V_N = l(x(0), u(0)) + \ldots + \overbrace{l(x(N-2), u(N-2)) + \overbrace{l(x(N-1), u(N-1)) + l_f(x(N))}^{u(N-1)}}^{u(N-2)}. \qquad (22)$$

In the expression above, an important property of the problem has been indicated: each control input affects only a corresponding *tail* of the sum; the later the control input is, the shorter the tail. Starting at the very end, only the last two terms of (22) depend on the last control input $u(N-1)$; we can therefore rewrite the minimization problem as

$$\min_{u(0:N-1)} V_N(x(0), u(0\!:\!N-1)) = \\ \min_{u(0:N-2)} \left\{ V_{N-1}(x(0), u(0\!:\!N-2)) + \min_{u(N-1)} \left[l(x(N-1), u(N-1)) + l_f(x(N)) \right] \right\}.$$

This allows us to start unfolding the solution to the LQ problem by applying *backwards dynamic programming*. This equation actually encodes **Bellman's principle of optimality**, which states that for an optimal path, at any time it holds that *regardless of which decisions have been taken earlier, the remaining path must be optimal*. The equation is often called Bellman's equation.

The last two terms of (22) are quadratic in $u(N-1)$. We can rewrite this expression into one quadratic form by using the system model (15) and by completing the squares as follows:

$$\begin{aligned} & l(x(N-1), u(N-1)) + l_f(x(N)) \\ & \quad = x^\top(N-1)Qx(N-1) + u^\top(N-1)Ru(N-1) \\ & \qquad + (Ax(N-1) + Bu(N-1))^\top P_f (Ax(N-1) + Bu(N-1)) \\ & \quad = x^\top(Q + A^\top P_f A)x + u^\top(R + B^\top P_f B)u + 2x^\top A^\top P_f B u \\ & \quad = (u + (R + B^\top P_f B)^{-1} B^\top P_f A x)^\top (R + B^\top P_f B)(u + (R + B^\top P_f B)^{-1} B^\top P_f A x) \\ & \qquad + x^\top (Q + A^\top P_f A - A^\top P_f B (R + B^\top P_f B)^{-1} B^\top P_f A) x \end{aligned}$$

where we have dropped the time arguments for $x(N-1)$ and $u(N-1)$ in the second step, and the last step has been obtained by completing the squares. The latter makes it possible to simply read off the minimizing control

$$u^*(N-1) = u^*(N-1; x) = -(R + B^\top P_f B)^{-1} B^\top P_f A x \equiv K(N-1)x$$

and the resulting optimal cost-to-go from state x at time $N-1$ to the final time N is

$$V^*_{N-1\to N}(x) = x^\top (Q + A^\top P_f A - A^\top P_f B (R + B^\top P_f B)^{-1} B^\top P_f A) x \equiv x^\top P(N-1) x. \qquad (23)$$

We have thus obtained the optimal last control input $u^*(N-1)$ as a linear feedback from the state x and the optimal cost-to-go is a quadratic form in the state.

Using the result above, namely that the minimum value of the last two terms in (22) is given as an explicit function of $x(N-1)$ in (23), we now proceed to include one more term from (22), namely the one depending on $u(N-2)$:

$$\begin{aligned} &l(x(N-2),u(N-2)) + V^*_{N-1\to N}(x(N-1)) \\ &\quad = x^\top(N-2)Qx(N-2) + u^\top(N-2)Ru(N-2) \\ &\qquad + (Ax(N-2)+Bu(N-2))^\top P(N-1)(Ax(N-2)+Bu(N-2)). \end{aligned}$$

By carrying out calculations, completely analogous to the ones in step 1 (P_f is replaced by $P(N-1)$), we get the optimal control

$$u^*(N-2) = u^*(N-2;x) = -(R+B^\top P(N-1)B)^{-1}B^\top P(N-1)Ax = K(N-2)x,$$

where x is now short for $x(N-2)$, and the optimal cost-to-go from state x at time $N-2$ to the final time N is

$$V^*_{N-2\to N}(x) = x^\top P(N-2)x, \tag{24}$$

and $P(N-2)$ is given by the **Riccati equation**

$$P(N-2) = Q + A^\top P(N-1)A - A^\top P(N-1)B(R+B^\top P(N-1)B)^{-1}B^\top P(N-1)A.$$

The procedure is repeated until we arrive at the initial time (note that x is short for $x(k)$, the current state vector). The sequence of optimal control laws (**control policy**) is computed as:

$$\begin{aligned} u^*(k;x) &= K(k)x(k), \quad k=0,\dots,N-1 \\ K(k) &= -(R+B^\top P(k+1)B)^{-1}B^\top P(k+1)A \end{aligned}$$

where the **Riccati equation** is

$$P(k-1) = Q + A^\top P(k)A - A^\top P(k)B(R+B^\top P(k)B)^{-1}B^\top P(k)A, \quad P(N) = P_f \tag{25}$$

or equivalently

$$P(k-1) = Q + A^\top P(k)A + A^\top P(k)BK(k-1), \quad P(N) = P_f. \tag{26}$$

The **optimal cost-to-go** (from time k to time N) is

$$V^*_{k\to N}(x) = x^\top(k)P(k)x(k).$$

We note that the solution to the LQ problem is in the form of a time-varying feedback *control law* or, equivalently, a sequence of control laws, often referred to as a *control policy*. This makes it different to the batch approach solution, which resulted in a sequence of optimal *control inputs*. We could of course use the DP solution to pre-compute all the control signals based on the problem input data. This would, however, destroy the feedback nature of the solution, where the actual control actions taken will make use of the updated state information from the controlled system.

The LQ control problem formulated and solved is a *finite horizon* control problem. Regardless of which solution we choose, the solution is not directly applicable as a continuously operating controller, since time will eventually "run out". However, it may be used as the basis for *receding horizon* control in the manner outlined in the Section 1. We turn to this in the next section.

Example 3.1 (Batch vs. DP solution, [1])**.** Consider the simple integrator system

$$x^+ = x + u.$$

We will study the finite-time optimal control problem with $x_0 = 1$, $N = 3$ and $Q = R = P_f = 1$. Using the batch approach, the optimal control sequence can be found as

$$\begin{aligned}\boldsymbol{u}^* &= -\begin{bmatrix}4 & 2 & 1\\ 2 & 3 & 1\\ 1 & 1 & 2\end{bmatrix}^{-1}\begin{bmatrix}3\\ 2\\ 1\end{bmatrix} x_0 = -\begin{bmatrix}0.615\\ 0.231\\ 0.077\end{bmatrix} x_0\\ \boldsymbol{x}^* &= \begin{bmatrix}1 & 0.385 & 0.154 & 0.077\end{bmatrix}^\top.\end{aligned}$$

We may also solve the problem with DP. The solution can be obtained as $u^*(k) = K(k)x(k)$, where the optimal control gain can be found via backward recursion,

$$K(0:2) = (-0.615, -0.6, -0.5)\,.$$

The optimal control sequence can now be obtained by simulating the system forward in time,

$$\begin{aligned}\boldsymbol{u}^* &= \begin{bmatrix}-0.615 \cdot 1\\ -0.6 \cdot 0.385\\ -0.5 \cdot 0.154\end{bmatrix} = -\begin{bmatrix}0.615\\ 0.231\\ 0.077\end{bmatrix}\\ \boldsymbol{x}^* &= \begin{bmatrix}1 & 0.385 & 0.154 & 0.077\end{bmatrix}^\top.\end{aligned}$$

Clearly, and as expected, the solutions by the batch and DP approach are identical.

However, there is a conceptual difference between the two approaches. Notice that the batch approach returns an open loop control sequence that is completely determined once the initial state is measured or estimated. However, the DP solution returns an optimal feedback policy that is a function of the system state. The feedback policy, $u^*(k) = K(k)x(k) = \pi(x(k))$, is a parametric function where the state $x(k)$ is the parameter. This will be discussed further in Section 12. The difference between the two approaches can be seen explicitly if the system is subject to process disturbance $w \in [-1, 1]$,

$$x^+ = x + u + w.$$

This is illustrated in Figure 7, where the response to three different disturbance sequences are shown, namely $w(k) = 0$, $w(k) = -1$, and $w(k) = 1$, $\forall k$. Even for this short horizon, there is a clear difference between the open-loop solution, see Figure 7a, and the closed-loop, feedback policy, see Figure 7b.

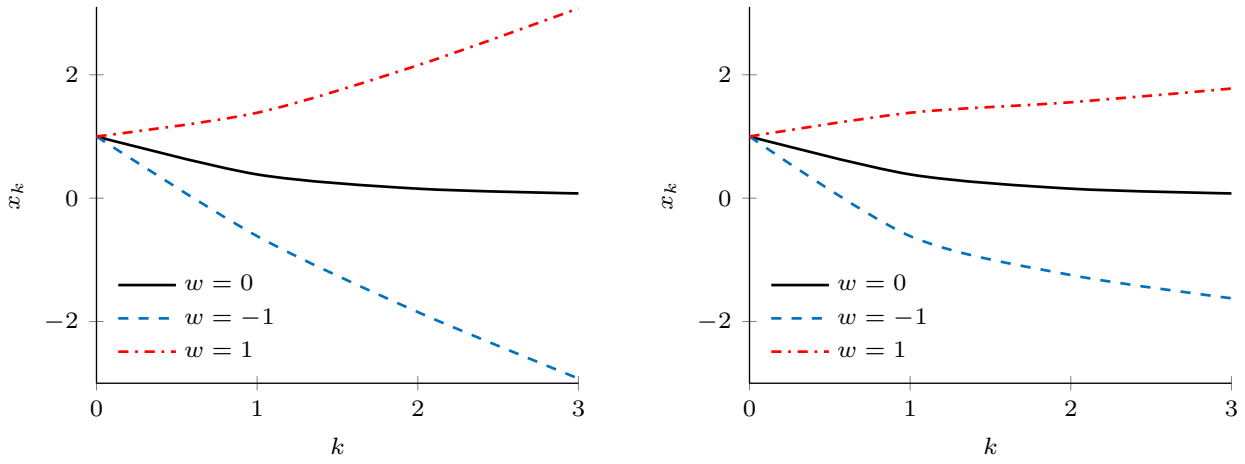
■

3.2 Unconstrained receding horizon control

Let us now return to the idea to use the finite time LQ solution as a basis for receding horizon control. We remind about the receding horizon idea, namely to repeatedly solve an optimisation problem over a finite horizon, but only using the first element of the optimal control sequence and letting the optimisation horizon recede with time.

Using the result from the DP recursion above and initialising the system at time index k (the solution will remain unchanged by shifting the time origin from 0 to k), the first control action in the optimal control sequence is $u^*(0; x(k)) = K(k)x(k)$. Alternatively, the receding horizon controller can be obtained from the first element of the optimal batch solution, i.e.

$$u(k) = \begin{bmatrix}I_m & 0_m & \cdots & 0_m\end{bmatrix}\boldsymbol{u}^* = -\begin{bmatrix}I_m & 0_m & \cdots & 0_m\end{bmatrix}(\Gamma^\top\bar{Q}\Gamma + \bar{R})^{-1}\Gamma^\top\bar{Q}\Omega x(k). \tag{27}$$



(a) Open loop control.

(b) Closed-loop, feedback control.

Figure 7: Receding horizon control of an uncertain system with an additive process disturbance w.

Regardless of which version of the receding horizon controller we choose—the two solutions are of course identical—we notice that the control law obtained is *linear*. However, since the control is based on optimality over a finite horizon, stability of the closed-loop system is not guaranteed.

Example 3.2 (Optimality does not guarantee stability [1])**.** Consider the system

$$x^+ = Ax + Bu = \begin{bmatrix} 4/3 & -2/3 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$
$$y = Cx = \begin{bmatrix} -2/3 & 1 \end{bmatrix} x$$

and design an LQ controller for $Q = P_f = C^\top C + \delta I$ with $\delta = 0.001$ and $R = 0.001$. The time horizon is $N = 5$. To compute the solution, the Riccati equation (25) is iterated 4 times with $P(5) = P_f$, and the controller gain is obtained as

$$K(0) = -(R + B^\top P(1)B)^{-1} B^\top P(1)A = \begin{bmatrix} -0.026 & 0.665 \end{bmatrix}.$$

The eigenvalues of the closed-loop matrix $A + BK(0)$ are $\{1.307, 0.001\}$, i.e. the closed-loop system is unstable. ■

In the unconstrained case, a stable closed-loop solution by the LQ-based receding horizon controller can be guaranteed by extending the optimisation horizon to *infinity* (which means that it is really not any longer a receding horizon controller).

The system

$$S: \quad x^+ = Ax + Bu \tag{28}$$

with optimisation criterion

$$V_\infty(x(0), u(0:\infty)) = \sum_{i=0}^{\infty} (x^\top(i)Qx(i) + u^\top(i)Ru(i)) \tag{29}$$

and with pair (A, B) controllable, and $Q, R \succ 0$, is an infinite horizon LQ control problem whose solution gives a stable closed-loop system

$$x^+ = Ax + BKx$$

with a time invariant feedback gain K, given by

$$K = -(B^\top PB + R)^{-1} B^\top PA \tag{30}$$
$$P = Q + A^\top PA - A^\top PB(B^\top PB + R)^{-1} B^\top PA. \tag{31}$$

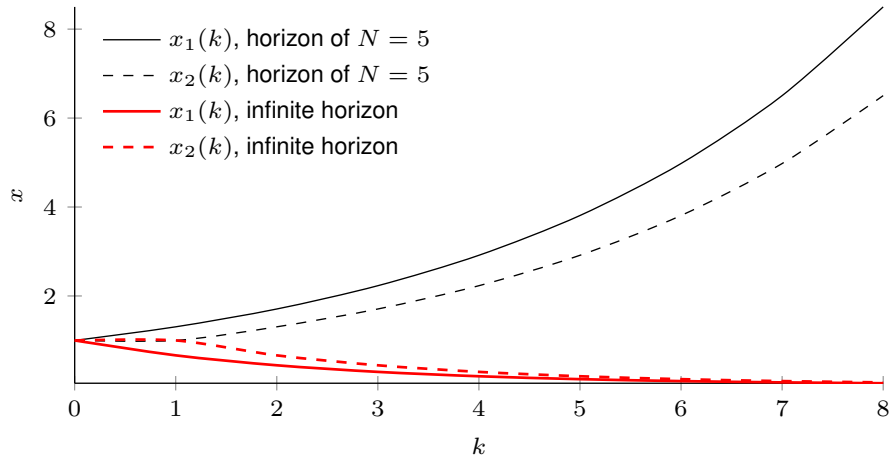


Figure 8: State evolution from the two optimal controllers, one with horizon of $N = 5$ and the other with an infinite horizon.

The latter equation is called the *algebraic Riccati equation*. The optimal cost is given by

$$V_\infty^*(x(0)) = x^\top(0)Px(0).$$

Example 3.3 (Optimality does not guarantee stability, cont'd)**.** Consider again the LQ design for the system in the previous example. If the time horizon is increased and chosen as $N = 7$, i.e. the Riccati equation is iterated two more times, then the eigenvalues of the closed-loop matrix become $\{0.989, 0.001\}$, i.e. the closed-loop system is now just stable. Continuing to iterate the Riccati equation, the solution converges to the solution of the algebraic Riccati equation, giving the infinite horizon closed-loop eigenvalues

$$\text{eig}(A + BK) = \{0.664, 0.001\}$$

i.e. the closed-loop system is stable as predicted.

The example indicates that it may be advantageous to select a large time horizon, even if it is not infinity. We will return to this observation later. Evolution of the state trajectories is provided in Figure 8. ■

4 Constrained receding horizon control

The objective of this section is to proceed to present a basic formulation of model predictive control. In order to do this, we extend the LQ formulation from the previous section to include *constraints*. The implication of doing this is that an explicit solution to the finite horizon optimal control problem is not any longer readily available. Instead, the receding horizon controller will be defined implicitly. Again, the presentation is much influenced by [1], Sections 2.1-2.3.

Our basic MPC with quadratic criterion and linear constraints is presented in Sections 5.1-5.4 in [2], but is first introduced in a more general setting in Sections 4.1-4.2. The corresponding presentation in [4] is in Sections 2.2-2.3, and the formulation as an optimisation problem is found in Section 3.2.

In the previous section, we considered the LQ problem without any constraints on states and/or control signals. We would now like to extend this to the case with constraints. In doing this, we will start to discuss the problem from the dynamic programming point of view. However, once realizing that the unconstrained case is really very special, we will soon adopt the receding horizon perspective. We will also start with a slightly more general formulation of the problem to prepare for later discussions.

4.1 Constrained receding horizon control

Consider the system described by the (possibly nonlinear) discrete time state equation

$$x^+ = f(x,u), \quad x(0) = x_0,$$

where $x_0 \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ and we assume that $f(0,0) = 0$. Reminding ourselves of the nice properties of the infinite horizon LQ problem, we would like to consider an infinite horizon control problem (again, we choose to start at time 0). With the cost function defined as

$$V_\infty(x_0, u(0:\infty)) = \sum_{i=0}^{\infty} l(x(i), u(i)),$$

with $l(\cdot,\cdot) \geq 0$ and $l(0,0) = 0$, the infinite horizon optimal control problem is

$$\begin{aligned} \min_{u(0:\infty)} \quad & V_\infty(x_0, u(0:\infty)) && (32)\\ \text{subject to} \quad & x^+ = f(x,u), \quad x(0) = x_0 && (33)\\ & x(k) \in \mathbb{X}, \quad u(k) \in \mathbb{U}, \quad \text{for all } k \in (0,\infty). && (34) \end{aligned}$$

Here, we have added the requirement in (34) that the state vector and the control vector should stay within the sets $\mathbb{X} \in \mathbb{R}^n$ and $\mathbb{U} \in \mathbb{R}^m$, respectively. As stated, this problem can *in principle* be approached using dynamic programming in the following way. In the previous section, we used dynamic programming to solve the finite horizon optimisation problem recursively, basically relying on the repeated use of the Bellman's equation

$$\min_{u(0:N-1)} V_N(x(0), u(0:N-1)) = \min_{u(0)} \left\{ l(x(0), u(0)) + \min_{u(1:N-1)} V_{1\to N}(x(1), u(1:N-1) \right\}, \quad (35)$$

which is then solved backwards in time. If we let $N \to \infty$ in the Bellman's equation above, and replace $x(0)$ by x_0, then the two optimal costs will be identical, and we get Bellman's equation for the infinite horizon problem,

$$V_\infty^*(x_0) = \min_u \left\{ l(x_0, u) + V_\infty^*(f(x_0, u)) \right\}. \quad (36)$$

As said earlier, this equation can *in principle* be solved, but this is an intractable task for most problems. Even if we approximate the infinite horizon with a finite one, we cannot expect to be able to derive an explicit control *law* that is valid for all x. Instead, what the MPC approach is based on a calculation of the specific control action to be taken, given the *particular value of the current state*, and then to repeat this calculation with a receding horizon. In the purely deterministic case, which we presently consider, the two approaches are equivalent, and thus MPC can be seen as a way to implement a finite horizon DP solution. With the cost function defined as

$$V_N(x_0, u(0\!:\!N-1)) = V_f(x(N)) + \sum_{i=0}^{N-1} l(x(i), u(i)),$$

with the final cost $V_f(\cdot) \geq 0$ and $V_f(0) = 0$, the finite horizon optimal control problem is

$$\begin{aligned}
\min_{u(0:N-1)} \quad & V_N(x_0, u(0\!:\!N-1)) && (37)\\
\text{subject to} \quad & x^+ = f(x,u), \quad x(0) = x_0 && (38)\\
& x(k) \in \mathbb{X}, \quad u(k) \in \mathbb{U}, \quad \text{for all } k \in (0, N-1) && (39)\\
& x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. && (40)
\end{aligned}$$

Note that we have added a *terminal constraint* $x(N) \in \mathbb{X}_f$! It is assumed that $\mathbb{U}$, $\mathbb{X}$, and $\mathbb{X}_f$ all contain the origin. The following notation for the optimal solution will be used:
Optimal cost-to-go:

$$V_N^*(x_0) = \min_{u(0:N-1)} \{V_N(x_0, u(0\!:\!N-1)) \mid u(0\!:\!N-1) \in \mathcal{U}_N(x_0)\}.$$

Optimal control and state sequences:

$$\begin{aligned}
u^*(0\!:\!N-1; x_0) &= \{u^*(0; x_0), u^*(1; x_0), \ldots, u^*(N-1; x_0)\}\\
x^*(0\!:\!N; x_0) &= \{x^*(0; x_0), x^*(1; x_0), \ldots, x^*(N; x_0)\}.
\end{aligned}$$

The finite horizon optimal control problem formulated above will be the starting point for our MPC controller. Before proceeding to that, we shall briefly discuss when the optimisation problem above is meaningful, i.e. possible to solve. This is referred to as *feasibility* of the optimisation problem.

Feasibility

In the optimisation problem formulated above, not all control sequences are allowed or *feasible*. Only those control sequences, which meet the control and state constraints and lead the state trajectory to the target set $\mathbb{X}_f$ in the last step of the horizon, are allowed. It is important to note that the set of allowed control sequences in general will depend on the initial state x_0. Formally, we can express this implicit constraint on the control sequences as

$$u(0\!:\!N-1) \in \mathcal{U}_N(x_0),$$

where $\mathcal{U}_N(x_0)$ is the set of all control sequences driving the initial state x_0 to the terminal set $\mathbb{X}_f$ while satisfying all control and state constraints at all times $k \in (0, N-1)$. For some x_0, it may even be impossible to find *any* control sequence that fulfils the constraints. The optimisation problem is therefore well-defined (meaningful) only for initial states belonging to a subset $\mathcal{X}_N$ of the state space, the *feasible set* of initial states (which in turn may be empty),

$$\mathcal{X}_N = \{x_0 \in \mathbb{X} \mid \mathcal{U}_N(x_0) \neq \emptyset\}.$$

We will return to a more thorough discussion of feasibility in Section 11.

A final remark is that we have consistently formulated the optimal control problem as starting with an initial state at time 0. Since both model and optimisation objective are time invariant, there would be no difference (except more cumbersome notation) considering an initial state at time $k > 0$. This is what will indeed be the case, when we move to the MPC formulation next.

Receding horizon control

The formulation of the finite time optimal control problem above will now be used to formulate our basic receding horizon controller. Remember that the idea is that, given the current value of the state vector, the finite time optimal control problem is solved for the optimal control sequence, but that only the first sample of this sequence is actually applied to the process — the procedure is repeated at the next sampling instant. To stress the fact that the optimal control problem is formulated and solved again and again, with the current state vector $x(k) = x$ as the initial state vector, we will now change the notation so that x_0 is replaced by x in the expressions. We will, however, use the time invariance property of the problem and thus keep the convenient zero initial time convention.

1. At sampling instant k, when we have access to the state $x(k) = x$ (assumed to be within the feasible set), solve the optimisation problem

$$V_N^*(x) = \min_{u(0:N-1)} \{V_N(x, u(0:N-1)) \mid u(0:N-1) \in \mathcal{U}_N(x)\}$$

   for the optimal control sequence

$$u^*(0:N-1;x) = \{u^*(0;x), u^*(1;x), \ldots, u^*(N-1;x)\}.$$

2. Apply the first control input

$$u(k) = u^*(0;x).$$

3. Let $k := k + 1$ and go to (1).

Note that the receding horizon controller described this way *implicitly* defines a feedback control law for all x belonging to the feasible set

$$u(x) = \kappa_N(x) \equiv u^*(0;x), \quad x \in \mathcal{X}_N.$$

It is important to note that the control law is *implicit*—we don't know the function κ_N, but have a procedure for obtaining the value of the function, $\kappa_N(x)$, for specific x, the same way a subroutine call works. The receding horizon controller can thus be seen as a way to solve the dynamic programming problem piece by piece, i.e. for one value of x at a time.

4.2 Linear quadratic MPC

Let's have a closer look at the case of particular interest in this course, namely linear quadratic MPC with constraints. This is a special case of the receding horizon controller described above. The system model is linear,

$$x^+ = Ax + Bu,$$

and the criterion is quadratic as described in section 3.1. Constraints on control signals due to actuator saturation take the form

$$u_{\min} \leq u(k) \leq u_{\max}, \quad \text{for all } k \geq 0 \tag{41}$$

and similar constraints on the states due to e.g. safety or quality concerns may be

$$x_{\min} \le x(k) \le x_{\max}, \quad \text{for all } k \ge 0. \tag{42}$$

Sometimes, it is relevant to be able to put constraints also on the rate of change of control inputs,

$$\Delta u_{\min} \le u(k) - u(k-1) \le \Delta u_{\max}, \quad \text{for all } k \ge 0. \tag{43}$$

Such constraints can easily be put in a form that generalizes the inequalities (41) and (42) by introducing a new state vector

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}$$

for which the augmented system model becomes

$$\xi^+ = \mathcal{A}\xi + \mathcal{B}\Delta u$$

with appropriately defined matrices (see Section 2). The constraint (43) can then be stated as

$$\begin{bmatrix} 0 & -I \\ 0 & I \end{bmatrix} \xi(k) + \begin{bmatrix} I \\ -I \end{bmatrix} u(k) \le \begin{bmatrix} \Delta u_{\max} \\ -\Delta u_{\min} \end{bmatrix}.$$

The conclusion is that all the linear constraints described can be written in the compact form

$$F\boldsymbol{u} + G\boldsymbol{x} \le h \tag{44}$$

with $\boldsymbol{u}$, $\boldsymbol{x}$ defined in (20), for some matrices F, G and some vector h, all of appropriate dimensions.

The linear quadratic MPC controller can now be summarized as follows:

1. At sampling instant k, when we have access to the state $x(k) = x$, solve the optimisation problem

$$V_N^*(x) = \min_{u(0:N-1)} \{V_N(x, u(0:N-1)) \mid u(0:N-1) \in \mathcal{U}_N(x)\},$$

$$V_N(x, u(0:N-1)) = x^\top(N) P_f x(N) + \sum_{i=0}^{N-1} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right); \; x(0) = x$$

   for the optimal control sequence

$$u^*(0:N-1; x) = \{u^*(0; x), u^*(1; x), \ldots, u^*(N-1; x)\}.$$

2. Apply the first control input

$$u(k) = u^*(0; x).$$

3. Let $k := k + 1$ and go to 1.

In this formulation, the optimisation variables are the sequence of control inputs $u(0:N-1)$ or, equivalently, the vector $\boldsymbol{u}$. The constraints are encoded somewhat implicitly via the constraint set $\mathcal{U}_N$, and similarly all the variables $\{x(i)\}$ are expressions in terms of $x(0) = x$ and $u(0:N-1)$ (via the model equations). The resulting *condensed form* of the optimisation problem can be given an equivalent formulation using the vector notation for the control and state sequences, i.e. $\boldsymbol{u}$ and $\boldsymbol{x}$ defined in (20). The quadratic criterion can then be expressed as in (21). Assuming all the constraint sets $\mathbb{X}, \mathbb{U}, \mathbb{X}_f$ are defined by affine inequalities, we can also use (44) to encode all the constraints. The equivalent formulation of our LQ MPC is thus:

1. At sampling instant k, when we have access to the state $x(k) = x$, solve the optimisation problem

$$\begin{aligned} \underset{\boldsymbol{u}}{\text{minimize}} \quad & V_N(x, \boldsymbol{u}) = x^\top Q x + (\Omega x(0) + \Gamma \boldsymbol{u})^\top \bar{Q}(\Omega x(0) + \Gamma \boldsymbol{u}) + \boldsymbol{u}^\top \bar{R} \boldsymbol{u} \\ \text{subject to} \quad & F\boldsymbol{u} + G(\Omega x(0) + \Gamma \boldsymbol{u}) \leq h. \end{aligned}$$

2. Apply the first control input $u(k) = \boldsymbol{u}(0)$.

3. Let $k := k + 1$ and go to 1.

Example 4.1 (LQ MPC for an integrator)**.** Consider a simple integrator system

$$x^+ = x + u$$

for which we want to design an LQ based MPC with $Q = R = P_f = 1$, $N = 2$ and a control constraint

$$u \in \mathbb{U} = [-1, 1].$$

Using the state equation and the notation $x(0) = x$ as above, we get the following expressions for the state variables involved in the optimisation:

$$x(0) = x \qquad x(1) = x + u(0) \qquad x(2) = x + u(0) + u(1).$$

The cost function to minimize with respect to $\boldsymbol{u} = [u(0)\ u(1)]^\top$, satisfying the control constraint, is

$$\begin{aligned} V_N(x, \boldsymbol{u}) &= x^2 + (x + u(0))^2 + (x + u(0) + u(1))^2 + u(0)^2 + u(1)^2 \\ &= \boldsymbol{u}^\top H \boldsymbol{u} + 2[2x \ \ x]\boldsymbol{u} + 3x^2 \equiv \boldsymbol{u}^\top H \boldsymbol{u} + 2c(x)^\top \boldsymbol{u} + d(x) \end{aligned}$$

with

$$H = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \qquad c(x) = \begin{bmatrix} 2x \\ x \end{bmatrix} \qquad d(x) = 3x^2.$$

Completing the squares, we can finally rewrite the cost function as

$$V_N(x, \boldsymbol{u}) = (\boldsymbol{u} - a(x))^\top H (\boldsymbol{u} - a(x)) + d(x) - a(x)^\top H a(x)$$

where

$$a(x) = -H^{-1}c(x) = Kx, \quad K = \begin{bmatrix} -3/5 \\ -1/5 \end{bmatrix}.$$

It is clear from the expression for $V_N(x, \boldsymbol{u})$ that $\boldsymbol{u} = a(x)$ is the global minimizer, and as long as this solution respects the constraints, it is also the solution of the constrained optimisation problem. This is indeed the case if $|x|$ is small. Consider first what happens for positive x and when x increases from 0: initially, the optimal $\boldsymbol{u}$ stays within the constraints; however, when $x = 5/3$, the first component of $\boldsymbol{u}$, $u(0)$, hits the constraint. Now, take a look at Figure 9, which depicts the feasible region for $\boldsymbol{u}$, the unconstrained minimizer $a(x)$, and some level curves for $V_N(x, \boldsymbol{u})$. From the figure, it can be seen that after having hit the constraint for $x = 5/3$, the solution to the constrained problem will slide along the left face of the constraint box, with the corresponding level curve being tangential to the vertical border of the box. It is also clear that increasing x further will eventually result in the optimal $\boldsymbol{u}$ being placed in the lower left corner of the box.

The conclusion from these arguments is that the receding horizon controller, which is determined from the first component of the minimizing control sequence, i.e. $u(0)$, is given by

$$u(x) = \kappa_N(x) = u^*(0; x) = \begin{cases} -3/5 \cdot x & 0 \leq x \leq 5/3 \\ -1 & x \geq 5/3. \end{cases}$$

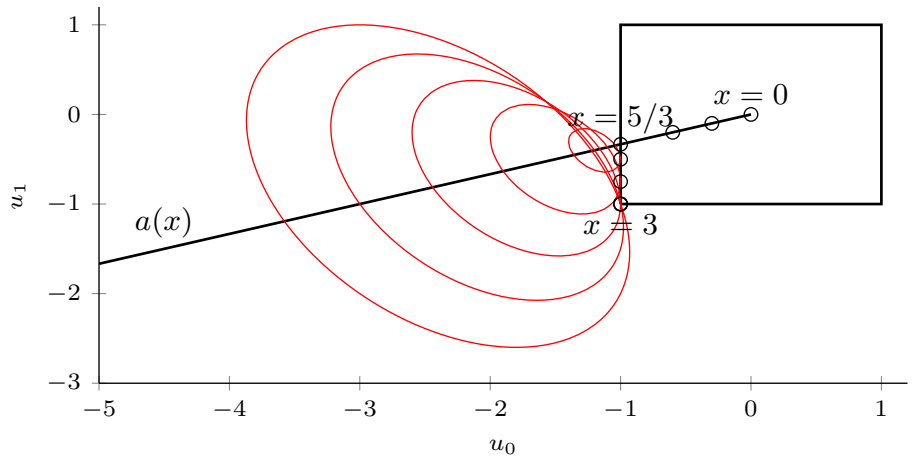


Figure 9: LQ MPC for the integrator system. Here, $a(x)$ is the unconstrained minimizer, and the red lines depict level curves for $V_N(x, \boldsymbol{u})$.

It is then easy to verify that the same arguments hold for negative x, and the resulting control law is indeed symmetric:

$$u(x) = \kappa_N(x) = u^*(0; x) = \begin{cases} 1 & x \leq -5/3 \\ -3/5 \cdot x & -5/3 \leq x \leq 5/3 \\ -1 & x \geq 5/3. \end{cases}$$

We note that the control law is piecewise linear and that it can readily be described as a linear feedback with saturation. Another way to arrive at this solution would of course be to apply the following *ad hoc* procedure: design the linear LQ controller for the unconstrained problem, and then append a saturation of the control. The fact that the receding horizon controller gives exactly the same result is a consequence of the particularly simple example—note, however, that this is *not* the case in general! ■

In the example above, we used the condensed form of the optimisation problem, i.e. the system equations are used to express the state variable over the prediction horizon in terms of the initial state x and the future control signals. This is essentially also what we did in equation (20) when deriving the batch solution for the unconstrained case. In doing so, we effectively eliminate all future states from the optimisation problem, which can be stated in terms of future controls only.

We have seen previously that there is an alternative way to formulate the algorithm. Instead of expressing future states as functions of the control inputs, the state equation can be included as an *equality constraint* in the optimisation. In doing so, we may consider all future states *and* controls as optimisation variables. By ordering all these variables chronologically when defining the variable vectors, it turns out that the associated matrices become *banded*, which is very desirable for computational reasons, and this is the reason why this alternative representation could be advantageous. The algorithm formulation for this case would read as follows:

1. At sampling instant k, when we have access to the state $x(k) = x$, solve the optimisation problem

$$V_N^*(x) = \min_{u(0:N-1), x(0:N)} \{V_N(x, u(0:N-1), x(0:N))\},$$

$$V_N(x, u(0:N-1), x(0:N)) = x^\top(N) P_f x(N) + \sum_{i=0}^{N-1} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right), \; x(0) = x$$

for the optimal control and state sequences

$$u^*(0\!:\!N-1;x) = \{u^*(0;x), u^*(1;x), \ldots, u^*(N-1;x)\}$$
$$x^*(0\!:\!N;x) = \{x^*(0;x), x^*(1;x), \ldots, x^*(N;x)\}$$

and subject to

$$x(k+1) = Ax(k) + Bu(k), \quad x(0) = x \tag{45}$$
$$x(k) \in \mathbb{X} \quad u(k) \in \mathbb{U} \quad \text{for all } k \in (0, N-1) \tag{46}$$
$$x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. \tag{47}$$

2. Apply the first control input $u(k) = u^*(0;x)$.

3. Let $k := k+1$ and go to 1.

Remark. *The initial state $x(0)$ may or may not be included in the vector of optimisation variables. In the latter case, the equality constraint $x(0) = x$ will not be needed.*

Finally, we give an equivalent formulation using the vector notation for the control and state sequences:

1. At sampling instant k, when we have access to the state $x(k) = x$, solve the optimisation problem

$$\begin{aligned} \underset{\boldsymbol{u},\boldsymbol{x}}{\text{minimize}} \quad & V_N(x, \boldsymbol{u}, \boldsymbol{x}) = x^\top Q x + \boldsymbol{x}^\top \bar{Q} \boldsymbol{x} + \boldsymbol{u}^\top \bar{R} \boldsymbol{u} \\ \text{subject to} \quad & x^+ = Ax + Bu \\ & F\boldsymbol{u} + G\boldsymbol{x} \leq h. \end{aligned}$$

2. Apply the first control input $u(k) = \boldsymbol{u}(1)$.

3. Let $k := k+1$ and go to 1.

Here, the equality constraint, the state equation, is expressed in terms of components of $\boldsymbol{x}$ and $\boldsymbol{u}$.

Let us finish by making a remark on horizons. In the formulation above, we have consistently used a control horizon M equal to the prediction horizon N. This assumption keeps notation as simple as possible, and it is also a realistic choice in many cases. On the other hand, under certain conditions, it may be preferable to select M smaller than N and to add some assumption on the control signals beyond the control horizon. It is not difficult to modify the formulations above to this situation, and this will be addressed in the assignments.

5 MPC practice – setpoints, disturbances and observers

So far, the presentation of the MPC ideas has been based on somewhat idealized assumptions: full state information, no disturbances, and regulation of the variables to the origin. The aim of this section is to take a step towards more practical situations, when some or all of these assumptions are not fulfilled. In the case with incomplete state information, it is standard practice to employ a state estimator (or observer), and then just replace the unknown state with the current estimate when computing the control action. It turns out that it is suitable to also address the case with disturbances at the same time, since one of the objectives of an observer may be to estimate the disturbance. The investigation of these questions is a continuation of the discussion of disturbance models from Section 2, and the associated question of offset free control.

The discussion on setpoint targets and disturbances is found in Section 1.5 of [1], and state estimation in Section 1.4. Observers are introduced in Section 5.5 of [2], but the main part of the material is presented in Chapter 9. In [4], mini-tutorial 2 on page 58 provides basic facts on state observers, and the Kalman filter is summarized in mini-tutorial 9 on page 226. Sections 2.6.2 and 2.6.3 discuss some special cases, in particular the popular so called *DMC scheme*.

5.1 Output predictions and the DMC scheme

Consider again our basic system model, i.e. the linear state equation

$$x(k+1) = Ax(k) + Bu(k) \tag{48}$$

$$y(k) = Cx(k). \tag{49}$$

When formulating the basic recipe for receding horizon control, we have already made use of this model to calculate our best "guess" or estimate of the state vector x, given the available information: we have used a *copy* of the system equations (48) to *predict* the state vector over the prediction horizon, given information about the current state vector $x(k)$. This was done already in Example 1.2 in Section 1, but it is most obvious when the batch approach is used, and the calculations are done in equation (19); in the DP approach, it is somewhat "hidden", since the state variables are involved in the calculations.

Let's repeat what was used for the batch approach in some more detail; let k denote current time and denote by $\hat{x}(k+i|k)$ the estimate of the state $x(k+i)$, *given information available at time* k. By iterating the system equations we get

$$\begin{aligned}
\hat{x}(k+1|k) &= A\hat{x}(k|k) + Bu(k) \\
\hat{x}(k+2|k) &= A\hat{x}(k+1|k) + Bu(k+1) = A^2\hat{x}(k|k) + ABu(k) + Bu(k+1) \\
&\vdots \\
\hat{x}(k+N|k) &= A^N\hat{x}(k|k) + A^{N-1}Bu(k) + A^{N-2}Bu(k+1) + \ldots + Bu(k+N-1).
\end{aligned}$$

In order to stress that all computations are carried out at time k and future controls are only *candidate* control actions in the MPC context, future control inputs are denoted $u(k+i|k)$ as in Section 1 and in [4]. Adopting this notation and collecting these results into vectors, we get a result similar to (19), repeated here for convenience.

The state model

$$x(k+1) = Ax(k) + Bu(k)$$

gives the state predictions

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{x}(k+2|k) \\ \vdots \\ \hat{x}(k+N|k) \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}. \tag{50}$$

In effect, what equation (50) means is that a copy of the system is 'simulated' with the assumed control sequence $u(k:k+N-1|k)$ and the best available estimate $\hat{x}(k|k)$ of the initial state vector $x(k)$. It is immediate to translate this to the corresponding equation for predicting the system output y.

The state model

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned}$$

gives the output predictions

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-1}B & CA^{N-2}B & \cdots & CB \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}. \quad (51)$$

It is possible to rephrase the above equations for predicting the system state or output, using instead *control moves*, i.e. changes in the control signal u. Defining $\Delta u(k) = u(k) - u(k-1)$, we have

$$\begin{aligned} u(k) &= \Delta u(k) + u(k-1) \\ u(k+1) &= \Delta u(k) + \Delta u(k+1) + u(k-1) \\ &\vdots \\ u(k+N-1) &= \Delta u(k) + \Delta u(k+1) + \ldots + \Delta u(k+N-1) + u(k-1) \end{aligned}$$

and the corresponding relations hold for the candidate control variables $u(k+i|k)$. Inserted into equation (51), this gives

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix} = \underbrace{\begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-1}B & CA^{N-2}B & \cdots & CB \end{bmatrix} \begin{bmatrix} I \\ \vdots \\ \vdots \\ I \end{bmatrix} u(k-1)}_{\text{free response}}$$
$$+ \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB + CB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{N-1} CA^iB & \sum_{i=0}^{N-2} CA^iB & \cdots & CB \end{bmatrix} \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+N-1|k) \end{bmatrix} \quad (52)$$

or

$$\boldsymbol{y}(k) = \boldsymbol{y_f}(k) + \Theta \boldsymbol{\Delta u}(k). \quad (53)$$

Here we have used the notation $\boldsymbol{y_f}$ (introduced in Section 1) for the *free response* of the system, i.e. the vector of future outputs resulting from freezing the control signal to its current value $u(k-1)$. The second term makes explicit how the output sequence is going to be affected by changing the control signal.

It should be noted that the expressions given above for predicted states and outputs have been given based on the assumption that the *prediction horizon* N and the *control horizon* M are equal, $N = M$. If $M < N$, then these expressions need to be modified accordingly, and you will have the chance to look into this in the assignments.

In the discussion above about predicting the state and/or the output, we just briefly noted that the expressions contain a term $\hat{x}(k|k)$, which is an estimate of the current state. The question is how this estimate is obtained? In previous sections, we have simply assumed that the state is available for perfect measurement, i.e. $\hat{x}(k|k) = x(k)$. We now turn to the case when this is no longer true.

Example 5.1 (The DMC scheme)**.** Assume that a plant is open loop stable, and that we choose the simplest observer available, namely a pure simulation of a model of the plant. The observer is thus described by the equation[2]

$$\hat{x}(k|k) = A\hat{x}(k-1|k-1) + Bu(k-1),$$

where we assume perfect knowledge of the system matrices A and B. Since A is stable,

$$\hat{x}(k|k) - x(k) \to 0,\ k \to \infty,$$

i.e. the state will asymptotically be estimated perfectly. To stress the fact that predicted outputs are based on a model of the plant, equation (53) is rewritten as

$$\boldsymbol{y}_{\mathcal{M}}(k) = \boldsymbol{y_f}(k) + \Theta\boldsymbol{\Delta u}(k), \tag{54}$$

where superscript $\mathcal{M}$ stands for 'model'. The fact that we have used a very simple observer has two immediate consequences.

1. Note that with the estimator used, the free response $\boldsymbol{y_f}$ depends only on previous control inputs. If there are no (active) constraints, the control action is computed from $\boldsymbol{y_f}$ and possibly a reference trajectory. Hence, the computed control signal does not depend on previous outputs, i.e. *there is no feedback*!

2. Because of the absence of feedback, we get problems with disturbances. Assume that there is a constant load disturbance d, so that the vector of plant outputs $\boldsymbol{y}_{\mathcal{P}}$ is given by

   $$\boldsymbol{y}_{\mathcal{P}}(k) = \boldsymbol{y_f}(k) + \Theta\boldsymbol{\Delta u}(k) + d \cdot \mathbf{1}, \tag{55}$$

   where $\mathbf{1}$ is a vector of 1's. In steady-state, the control signal will be determined so that the model output equals the reference, i.e.

   $$\boldsymbol{y}_{\mathcal{P}}(k) = \boldsymbol{y}_{\mathcal{M}}(k) + d \cdot \mathbf{1} = \boldsymbol{r}(k) + d \cdot \mathbf{1},$$

   where we have used the notation $\boldsymbol{r}$ for the vector of reference values introduced in Example 1.2. The conclusion is that there will be a steady-state error.

There is a simple remedy of the problem with steady-state error described above. It was originally devised for the DMC scheme, a very early MPC algorithm for process control. Start by changing the model to

$$y(k) = Cx(k) + \hat{d},$$

where $\hat{d}$ is the (unknown) constant disturbance. Now, estimate this disturbance simply by computing

$$\begin{aligned}\hat{d}(k|k) &= y(k) - C\hat{x}(k|k) = y(k) - C\left(A\hat{x}(k-1|k-1) + Bu(k-1)\right)\\ \hat{d}(k+i|k) &= \hat{d}(k|k), \quad i = 1, 2, \ldots\end{aligned}$$

This will instead of (54) result in

$$\boldsymbol{y}_{\mathcal{M}}(k) = \boldsymbol{y_f}(k) + \Theta\boldsymbol{\Delta u}(k) + \hat{d}(k|k) \cdot \mathbf{1}.$$

The consequence is that in steady state (and if $\hat{d} \to d$) we will now have $\boldsymbol{y}_{\mathcal{P}} = \boldsymbol{y}_{\mathcal{M}} = \boldsymbol{r}$. Note that the modification done has now introduced feedback into the controller! ■

[2]The notation $\hat{x}(k|k)$ may seem a bit odd here, since the estimate is not using any information at time k, but since this is a consequence of the deliberate choice of not using the measured output, we stick to the general notation anyway.

Observer interpretation of the DMC scheme

In the DMC scheme discussed above, the modification performed was ad hoc, i.e. without any real justification (except for the nice property that it gives no steady state error for a constant disturbance). The scheme is also constrained to the case with a stable plant. We will now give an interpretation of the modification in terms of observer design, and this naturally opens up for generalizations. Let's first, however, briefly summarize the fundamentals of how to design a state observer for a linear discrete time system.

Given a system model

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k), \end{aligned}$$

with observer update

$$\hat{x}(k+1|k) = A\hat{x}(k|k-1) + Bu(k) + L(y(k) - C\hat{x}(k|k-1))$$

and error dynamics

$$\begin{aligned} \tilde{x}(k) &= \hat{x}(k) - x(k) \\ \tilde{x}(k+1) &= (A - LC)\tilde{x}(k). \end{aligned}$$

If (A, C) is observable, then the eigenvalues of the error dynamics matrix $(A - LC)$ can be assigned arbitrarily.

Example 5.2 (DMC scheme continued)**.** The assumption that the plant is affected by a constant load disturbance d can be expressed as an extension of the original model,

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ d(k+1) &= d(k) \\ y(k) &= Cx(k) + d(k). \end{aligned}$$

By introducing the augmented state

$$\xi(k) = \begin{bmatrix} x(k) \\ d(k) \end{bmatrix}$$

we get the new model

$$\begin{aligned} \xi(k+1) &= \mathcal{A}\xi(k) + \mathcal{B}u(k) \\ y(k) &= \mathcal{C}\xi(k) \end{aligned}$$

with

$$\mathcal{A} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} C & I \end{bmatrix}.$$

By defining the observer gain as $L = \begin{bmatrix} L_x \\ L_d \end{bmatrix}$, a standard observer for the extended model is given by

$$\begin{aligned} \hat{\xi}(k+1) &= \mathcal{A}\hat{\xi}(k) + \mathcal{B}u(k) + L(y(k) - \mathcal{C}\hat{\xi}(k)) \\ &= \left(\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} - \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} C & I \end{bmatrix} \right) \hat{\xi}(k) + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} L_x \\ L_d \end{bmatrix} y(k). \end{aligned}$$

The observer dynamics is determined by the first big matrix in the last expression. With the simple choice $L_x = 0$ and $L_d = I$, we get the observer error dynamics given by

$$\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} - \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} C & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}.$$

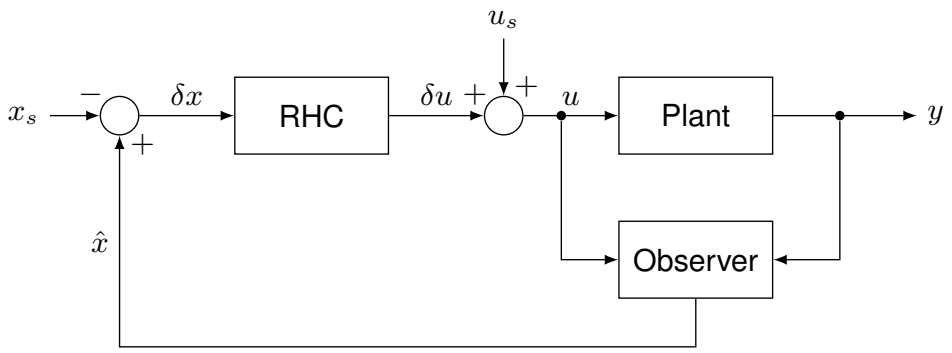


Figure 10: MPC overall structure with a receding horizon controller (RHC) working on deviation variables.

The eigenvalues of this matrix are given by the matrix A, which was assumed to be stable, and the rest of the eigenvalues are located in the origin, a choice that is usually referred to as *deadbeat* dynamics. ■

The conclusion of this example is that the DMC scheme uses a particularly simple observer design, which is based on a model of a constant load disturbance and the assumption of a stable plant. It is clear from the observer interpretation of the scheme, however, that there is plenty of freedom left in designing the observer dynamics, including the possibility to handle also the case with an unstable plant. We will return to this in the next section, but for the remaining part of this section, it is assumed that a state estimate $\hat{x}$ is available.

5.2 Steady state targets and zero offsets

The discussion about the DMC scheme in Examples 5.1 and 5.2 brought up the ability to augment the system model in order to model a constant but unknown disturbance. The model can then be used with an observer to estimate the disturbance, and this estimate can be included in the computation of the control action. We established already in Section 2 a basic fact, namely that we must have at least as many measured outputs as the dimension of the disturbance vector in order to be successful in estimating the disturbance, an intuitive and reasonable condition. We will now elaborate a bit on the problem to achieve *zero offset* relative to the set points (or steady-state targets), introduced in Section 2.

Before embarking on this task, it is relevant to make a comment on how the problem differs from what we are used to from classical control. For example, in a SISO PI control loop, the control error is computed as the difference between the setpoint y_{sp} and the actual process output y. With integral action, the controller "finds" the proper steady state (u_s, y_s) so that $u = u_s$ gives $y = y_s = y_{sp}$. This reasoning may be generalized to MIMO processes, provided there is a natural way to connect each output to a corresponding input, a technique often referred to as "pairing".

In contrast to what was just described, MPC is usually based on working with *deviation variables* $u - u_s$, $x - x_s$ and $y - y_s$ relative to a *steady state* (u_s, x_s, y_s), and this was also the assumption during the previous sections. The fact that the basic receding horizon controller works on deviation variables implies that the overall structure of our MPC can be depicted as in Figure 10.

It is clear from the block diagram above that, in order to fully describe the MPC algorithm, we need to specify the steady state variables. As discussed earlier, we want the steady state to fulfil the condition for *setpoint tracking* (repeated from equation (8))

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ y_{sp} \end{bmatrix}, \tag{56}$$

where y_{sp} is the setpoint. This system of linear equations is thus central in MPC algorithms, and it requires a thorough investigation. Let us first note that there are $n + p$ equations with $n + m$ unknowns, and depending on the relation between the number of inputs m and number of outputs p, different cases have to be distinguished.

Let's start with the case $p = m$, i.e. the system is square (the same number of inputs as outputs). Then there is a unique solution to equation (56) for any setpoint y_{sp}, provided the big matrix on the left hand side is non-singular. Now, if the plant model is augmented with a load disturbance model, as discussed for the DMC algorithm, the augmented model will be given by (11), again repeated here,

$$\begin{aligned} \begin{bmatrix} x \\ d \end{bmatrix}^+ &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix}. \end{aligned} \tag{57}$$

The consequence of doing this is that the state observer, in addition to producing $\hat{x}$, now also provides a disturbance estimate $\hat{d}$. Remember, however, the result in Proposition 2.2, stating that the detectability of this augmented system—an important property in order to make it possible to estimate the state—depends on the rank condition

$$\operatorname{rank} \begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} = n + n_d, \tag{58}$$

implying that there must be at least as many measured outputs as the dimension of the disturbance vector d. Given a disturbance estimate $\hat{d}$, the equation (56) for setpoint tracking will now have to be modified accordingly.

If the system is square ($p = m$):

1. Define desired setpoints for the outputs, y_{sp}.
2. Solve the following system of linear equations to find the steady-state targets

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_d \hat{d} \\ y_{sp} - C_d \hat{d} \end{bmatrix}. \tag{59}$$

Based on the steady-state target problem above, we can apply an MPC to deviation variables. Thus, the control problem is to determine the control deviation $\delta u(k)$ to bring the state deviation $\delta x(k) = \hat{x}(k) - x_s$ to the origin. Notice, however, that when a new disturbance estimate $\hat{d}$ is obtained, typically at each sampling instant, then we need to re-calculate the steady state target! Recognizing that the computation of the steady state target is a significant task of the MPC controller, we can identify three main computational tasks to be carried out by the MPC controller at each sampling instant:

1. Update the state estimate (including load disturbance).
2. Compute an updated steady state target, e.g. by solving equation (59).
3. Compute the next control action by solving a constrained optimisation problem.

The structure of the complete MPC controller including these three blocks is depicted in Figure 11.

It remains to discuss how to handle the situation when the system is not any longer square. First, note that we must have $p \leq m$ for equations (59) or (56) to always have a solution. If there are more outputs than inputs (which is often the case), then we cannot expect all setpoints to be reached in steady state, and we have to be satisfied coming as close as possible to the setpoint. The corresponding steady state can be determined by solving the following optimisation problem, where the hard

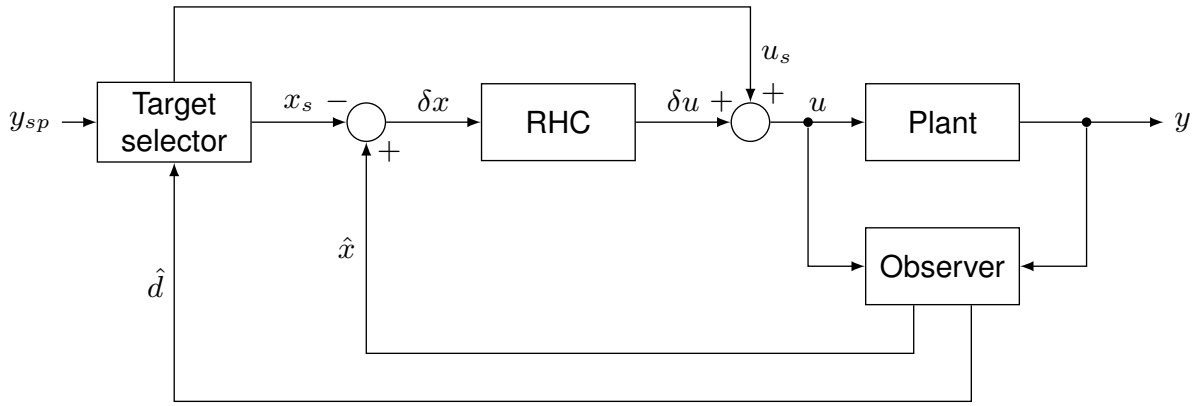


Figure 11: MPC block diagram with a receding horizon controller (RHC) working on deviation variables.

constraint on the steady state has been replaced by an objective to minimize. The equations are given for the case without disturbance, but it is straightforward to include it as above. We have added linear constraints on inputs and outputs, compatible with the corresponding constraints in the MPC control problem.

If there are more outputs than inputs ($p > m$):

1. Define desired setpoints for the outputs, y_{sp}.
2. Solve the following optimisation problem to find the best steady-state targets:

$$\min_{x_s,u_s} \left(|Cx_s - y_{sp}|_Q^2 \right), \quad Q \succeq 0$$

subject to

$$\begin{bmatrix} I - A & -B \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = 0$$
$$Eu_s \leq e$$
$$FCx_s \leq f.$$

If we have more outputs than inputs, another way to approach the steady state target problem is to focus on fewer *controlled outputs* $z = C_z x$, for which we require the setpoint z_{sp} to be reached in steady state. If $p_z < m$, then we can use the remaining flexibility to stay close to setpoints y_{sp} for non-controlled outputs $y = C_y x$ and/or desired values u_{sp} for the inputs. This leads to the following steady state target problem ($p < m$ is a special case).

If there are more inputs than controlled outputs ($p_z < m$):

1. Define setpoints for controlled outputs, z_{sp}, and desired values for the control input, u_{sp}, and non-controlled outputs y_{sp}.
2. Solve the following optimisation problem to find feasible steady-state targets:

$$\min_{x_s,u_s} \left(|u_s - u_{sp}|_{R_s}^2 + |C_y x_s - y_{sp}|_{Q_s}^2 \right), \quad R_s \succ 0$$

subject to

$$\begin{bmatrix} I - A & -B \\ C_z & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ z_{sp} \end{bmatrix}$$
$$Eu_s \leq e$$
$$FC_z x_s \leq f.$$

The interpretation of the above formulation is that we have put hard constraints on the controlled outputs in steady-state, and that we use the remaining flexibility of control to try to get as close as possible to defined setpoints for both controls and non-controlled outputs.

One common choice is to select the controlled outputs as a subset of the measured outputs, i.e.

$$\begin{aligned} C_z &= HC \\ C_y &= C, \end{aligned}$$

and the $p_z \times p$ selection matrix H with only 0's and 1's defines the selection mentioned. Assuming in this case that we get an estimate $\hat{d} = \hat{d}(k)$ of the load disturbance, the steady-state target optimisation problem needs to be modified in the following way to include the disturbance. Optimisation problem to find feasible steady-state target:

$$\min_{x_s, u_s} \left(|u_s - u_{sp}|^2_{R_s} + |Cx_s + C_d\hat{d} - y_{sp}|^2_{Q_s} \right), \quad R_s \succ 0$$

subject to

$$\begin{bmatrix} I - A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_d\hat{d} \\ z_{sp} - HC_d\hat{d} \end{bmatrix}$$
$$Eu_s \leq e$$
$$FHCx_s \leq f - FHC_d\hat{d}.$$

Based on the steady-state target problem above, we can apply an MPC to deviation variables, as discussed earlier. Because of the constraints, the resulting control system is nonlinear. However, if the closed-loop system operates in steady-state away from the constraints, then the following fundamental property holds for the control system obtained.

Proposition 5.1 (Off-set free control)**.** *Assume that the steady-state target problem is feasible and that an MPC with the following augmented model is used:*

$$\begin{aligned} \begin{bmatrix} x \\ d \end{bmatrix}^+ &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix}. \end{aligned} \tag{60}$$

Further assume that $C_z = HC$, $n_d = p$ and that B_d, C_d are chosen such that

$$\operatorname{rank} \begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} = n + p. \tag{61}$$

Assume that the closed-loop converges to a steady-state with constraints inactive. Then there is zero off-set in the controlled outputs, i.e.

$$z_s = z_{sp}.$$

6 The Kalman filter and moving horizon estimation

The aim of this section is to briefly discuss different options to approach the state estimation problem. The Kalman filter is a standard well-known solution. There are some alternatives that are related specifically to MPC. In particular, the introduction of constraints into the state estimation problem lead to algorithms that are very closely related to the algorithms presented earlier for receding horizon control.

The least-squares formulation of the estimation problem and *moving horizon estimation* is treated in Section 1.4 and Chapter 4 of [1] and in Chapter 9 of [2]. It is only briefly discussed in Section 10.4 of [4].

6.1 The Kalman filter

The Kalman filter offers a systematic way to design state observers, and its derivation is based on a formulation that involves describing the disturbances (process and measurement) as stochastic processes. In fact, assuming these processes are white noise Gaussian, the *Kalman filter* can be shown to provide the *optimal* observer in a mean square sense. Let's summarize the basic facts:

System model:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + w(k), \quad x(0) \sim \mathcal{N}(x_0, P_0), \quad w \sim \mathcal{N}(0, Q) \\ y(k) &= Cx(k) + v(k), \quad v \sim \mathcal{N}(0, R). \end{aligned}$$

State estimator:

$$\begin{aligned} &\text{Correction:} && \hat{x}(k|k) = \hat{x}(k|k-1) + L(k)[y(k) - C\hat{x}(k|k-1)], \quad \hat{x}(0|0) = x_0 \\ &\text{Prediction:} && \hat{x}(k+1|k) = A\hat{x}(k|k) + Bu(k). \end{aligned}$$

Kalman filter gain:

$$L(k) = P(k)C^\top [CP(k)C^\top + R]^{-1}.$$

State estimation error covariance update (note that $P(k) \equiv P(k|k-1)$):

$$\begin{aligned} &\text{Estimation error:} && P(k|k) = P(k) - P(k)C^\top [CP(k)C^\top + R]^{-1} CP(k), \; P(0|0) = P_0 \\ &\text{Prediction error:} && P(k+1) = AP(k|k)A^\top + Q. \end{aligned}$$

The Kalman filter has a time-varying gain, reflecting the fact that the a priori information about the state gradually looses its significance, since the process noise injects uncertainties and we have to rely more and more on the measurements obtained. Similar to the LQ case, the recursions of the covariance matrix will produce a stationary solution under certain conditions, and this leads to the *stationary Kalman filter*.

Consider the linear system

$$\begin{aligned} x^+ &= Ax + Bu + w, \quad w \sim \mathcal{N}(0, Q) \\ y &= Cx + v, \quad v \sim \mathcal{N}(0, R). \end{aligned}$$

If the pair (C, A) is observable and $Q, R \succ 0$, then the Kalman filter gain $L(k)$ and the prediction error covariance $P(k)$ converge to the solution of the (filtering) algebraic Riccati equation

$$\begin{aligned} L &= PC^\top [CPC^\top + R]^{-1} \\ P &= APA^\top - APC^\top [CPC^\top + R]^{-1} CPA^\top + Q. \end{aligned}$$

6.2 Least-squares estimation

There are strong connections between the linear-quadratic control problem and the state estimation problem formulated in the previous section. This is visible in the solutions, the LQ regulator and the Kalman filter, respectively. The two problems can in fact be shown to be *dual* to each other in a certain mathematical sense (but this goes beyond the scope of this course). Anyhow, we will now see that it is possible to reformulate the state estimation problem in a way that reveals the similarities to the LQ problem in a very explicit way.

LQ regulator:

$$\begin{aligned} u(k) &= K(k)x(k), \quad k = 0, \dots, N-1 \\ K(k) &= -(R + B^\top P(k+1)B)^{-1}B^\top P(k+1)A. \end{aligned}$$

Riccati equation:

$$P(k-1) = Q + A^\top P(k)A - A^\top P(k)B[R + B^\top P(k)B]^{-1}B^\top P(k)A, \quad P(N) = P_f.$$

Kalman filter:

$$\begin{aligned} \hat{x}(k+1|k) &= A\hat{x}(k|k-1) + Bu(k) + L(k)[y(k) - C\hat{x}(k|k-1)] \\ L(k) &= P(k)C^\top[CP(k)C^\top + R]^{-1}. \end{aligned}$$

Riccati equation:

$$P(k+1) = AP(k)A^\top + Q - AP(k)C^\top[CP(k)C^\top + R]^{-1}CP(k)A^\top, \quad P(0) = P_0.$$

Let's reconsider the state estimation problem above with one distinction, namely that we are not any longer willing to adopt the stochastic modelling framework for the disturbances. Instead, given a record of measurements $\{y(k)\}_{k=1}^N$, we would like to fit the corresponding sequence of state estimates to these measurements *in a least-squares sense*. The mathematical formulation is as follows.

System model:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k). \end{aligned}$$

Optimisation problem:

$$\min_{x(0:T)} V_T(x(0:T))$$

where the minimization is with respect to the sequence of state estimates

$$x(0:T) = \{x(0), x(1), \dots, x(T)\}.$$

The objective function V_T is given by

$$\begin{aligned} V_T(x(0:T)) = (x(0) - x_0)^\top P_0^{-1}(x(0) - x_0) &+ \sum_{i=0}^{T}(y(i) - Cx(i))^\top R^{-1}(y(i) - Cx(i)) \\ &+ \sum_{i=0}^{T-1}(x(i+1) - Ax(i) - Bu(i))^\top Q^{-1}(x(i+1) - Ax(i) - Bu(i)). \end{aligned} \quad (62)$$

The interpretation of this formulation is that we would like to explain the data we have (the a priori guess x_0 and the measurements $\{y(k)\}_{k=1}^N$) with our model as well as possible, the latter interpreted as follows: the state update equation and the output equation should hold for the state estimates $x(0:T)$ with as small error as possible. To reflect our confidence in the initial guess x_0, the state equation and the measurement equation, respectively, we use weighting matrices P_0, Q and R (it is no coincidence that we have used this notation, as will be explained shortly).

Dynamic programming solution

We will solve the estimation problem posed using dynamic programming. Contrary to the LQ case, we will however use *forward DP*, i.e. the recursions will run forward in time. This means that we unfold the solution by looking at subproblems, starting by rewriting the minimization problem as

$$\begin{aligned}\min_{x(0:T)} V_T(x(0:T)) = \\ \min_{x(1:T)} \Big\{ \min_{x(0)} \Big(&(x(0)-x_0)^\top P_0^{-1} (x(0)-x_0) + (y(0)-Cx(0))^\top R^{-1} (y(0)-Cx(0)) \\ &+ (x(1)-Ax(0)-Bu(0))^\top Q^{-1} (x(1)-Ax(0)-Bu(0)) \Big) + V(x(1:T)) \Big\}, \end{aligned} \quad (63)$$

where $V(x(1:T))$ is the objective function $V_T(x(0:T))$ without the terms depending on $x(0)$. For notational convenience, we will in the derivation assume there is no control signal, i.e. $u(\cdot) = 0$; it is straightforward to include u in the final expressions. Before going ahead with the DP calculations, we formulate a result that we will need shortly; the proof is left as an exercise.

Lemma 6.1 (Matrix inversion lemma)**.** *Assuming the invertability of the matrices involved, the following identity holds:*

$$(A+BCD)^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1}.$$

Step 0. In step 0, we will perform the inner minimization in (63) with respect to $x(0)$, and proceed in two 'sub-steps'.

Measurement update. Consider the first two terms in (63), i.e. the terms corresponding to the a priori guess x_0 and the measurement $y(0)$,

$$\begin{aligned} V_0(x(0)) &= (x(0)-x_0)^\top P_0^{-1} (x(0)-x_0) + (y(0)-Cx(0))^\top R^{-1} (y(0)-Cx(0)) \\ &= x^\top(0)[P_0^{-1} + C^\top R^{-1} C]x(0) - 2x^\top(0)[P_0^{-1}x_0 + C^\top R^{-1} y(0)] + x_0^\top P_0^{-1} x_0 + y^\top(0) R^{-1} y(0). \end{aligned} \quad (64)$$

Define (the subscript m stands for 'measurement update')

$$P_m(0) = [P_0^{-1} + C^\top R^{-1} C]^{-1} \quad (65)$$

and using the matrix inversion formula with $A = P_0^{-1}$, $B = C^\top$, $C = R^{-1}$, and $D = C$, we get the relation

$$P_m(0) = P_0 - P_0 C^\top [C P_0 C^\top + R]^{-1} C P_0. \quad (66)$$

Now, by completing the squares we can rewrite equation (64) as

$$\begin{aligned} V_0(x(0)) &= x^\top(0) P_m^{-1}(0) x(0) - 2x^\top(0)\left[P_0^{-1}x_0 + C^\top R^{-1} y(0)\right] + x_0^\top P_0^{-1} x_0 + y^\top(0) R^{-1} y(0) \\ &= (x(0)-a)^\top P_m^{-1}(0)(x(0)-a) - a^\top P_m^{-1}(0) a + x_0^\top P_0^{-1} x_0 + y^\top(0) R^{-1} y(0), \end{aligned}$$

where the vector a is given by

$$\begin{aligned} a &= P_m(0)[P_0^{-1}x_0 + C^\top R^{-1} y(0)] = P_m(0)[(P_m^{-1}(0) - C^\top R^{-1} C)x_0 + C^\top R^{-1} y(0)] \\ &= x_0 + P_m(0) C^\top R^{-1}(y(0) - Cx_0) = x_0 + P_0 C^\top [C P_0 C^\top + R]^{-1}(y(0) - Cx_0), \end{aligned}$$

and equation (66) has been used in the last step. Defining

$$\begin{aligned} L(0) &= P_0 C^\top [C P_0 C^\top + R]^{-1} \\ \hat{x}(0) &= a = x_0 + L(0)(y(0) - Cx_0), \end{aligned}$$

we can now write

$$V_0(x(0)) = (x(0) - \hat{x}(0))^\top P_m^{-1}(0)(x(0) - \hat{x}(0)) + c,$$

where the constant c can be shown to be given by

$$c = (y(0) - Cx_0)^\top [CP_0C^\top + R]^{-1} (y(0) - Cx_0),$$

i.e., it is independent of $x(0)$. We will therefore simply drop the term c from the objective in the sequel.

Time update. We now add the remaining term that depends on $x(0)$, denoting the result $V(x(0), x(1))$:

$$\begin{aligned} V(x(0), x(1)) &= V_0(x(0)) + (x(1) - Ax(0))^\top Q^{-1} (x(1) - Ax(0)) \\ &= (x(0) - \hat{x}(0))^\top P_m^{-1}(0) (x(0) - \hat{x}(0)) + (x(1) - Ax(0))^\top Q^{-1} (x(1) - Ax(0)). \end{aligned}$$

This equation is analogous to equation (64); the 'translation' is

$$x_0 \to \hat{x}(0), \quad P_0^{-1} \to P_m^{-1}(0), \quad y(0) \to x(1), \quad C \to A, \quad R^{-1} \to Q^{-1}.$$

We can therefore repeat the procedure involving completing of squares and leading to

$$\begin{aligned} V(x(0), x(1)) &= (x(0) - b)^\top [P_m^{-1}(0) + A^\top Q^{-1} A]^{-1} (x(0) - b) + d \\ b &= \hat{x}(0) + P_m(0)A^\top [AP_m(0)A^\top + Q]^{-1}(x(1) - A\hat{x}(0)), \end{aligned}$$

where d is a constant, independent of $x(0)$, and given by

$$d = (x(1) - A\hat{x}(0))^\top [AP_m(0)A^\top + Q]^{-1} (x(1) - A\hat{x}(0)).$$

It is now immediately seen that $x(0) = b$ minimizes $V(x(0), x(1))$. Denoting the optimal solution by $x^*(0)$ and the minimum by V_1^-, we thus have

$$\begin{aligned} x^*(0) &= \hat{x}(0) + P_m(0)A^\top [AP_m(0)A^\top + Q]^{-1} (x(1) - A\hat{x}(0)) \\ V_1^- &= \min_{x(0)} V(x(0), x(1)) = d = (x(1) - A\hat{x}(0))^\top P_t^{-1}(1) (x(1) - A\hat{x}(0)) \end{aligned} \tag{67}$$

where we introduced $P_t(1) = AP_m(0)A^\top + Q$ (t stands for 'time update').

Step 1. In the next step we incorporate two more terms of the objective function.

Measurement update. First include the new measurement $y(1)$ in the objective,

$$\begin{aligned} V_1(x(1)) &= V_1^-(x(1)) + (y(1) - Cx(1))^\top R^{-1} (y(1) - Cx(1)) \\ &= (x(1) - A\hat{x}(0))^\top P_t^{-1}(1) (x(1) - A\hat{x}(0)) + (y(1) - Cx(1))^\top R^{-1} (y(1) - Cx(1)). \end{aligned}$$

This expression looks exactly like the one we started with in step 0, equation (64). We can therefore only change the variable names and get analogous expressions for these, imitating what was done in step 0:

$$\begin{aligned} P_m(1) &= P_t(1) - P_t(1)C^\top [CP_t(1)C^\top + R]^{-1} CP_t(1) \\ L(1) &= P_t(1)C^\top [CP_t(1)C^\top + R]^{-1} \\ \hat{x}(1) &= A\hat{x}(0) + L(1)(y(1) - CA\hat{x}(0)). \end{aligned}$$

Similarly, the part of the objective function studied can be written

$$V_1(x(1)) = (x(1) - \hat{x}(1))^\top P_m^{-1}(1)(x(1) - \hat{x}(1)). \tag{68}$$

Time update. We proceed to include the final term depending on $x(1)$ and perform the minimization with respect to $x(1)$; the procedure follows what was done in step 0, giving as result the next optimal variable $x^*(1)$.

Steps 2–T. The procedure above can be repeated for every time instant up to $k = T$ (where only a measurement update is performed), and the complete DP solution can be summarized by the recursions below (where we have included the input term again).

System model:

$$x(k+1) = Ax(k) + Bu(k)), \quad y(k) = Cx(k).$$

Objective function:

$$V_T(x(0\!:\!T)) = (x(0) - x_0)^\top P_0^{-1}(x(0) - x_0) + \sum_{i=0}^{T}(y(i) - Cx(i))^\top R^{-1}(y(i) - Cx(i)) + \sum_{i=0}^{T-1}(x(i+1) - Ax(i) - Bu(i))^\top Q^{-1}(x(i+1) - Ax(i) - Bu(i)).$$

State estimator:

$$\begin{aligned} P_t(k) &= AP_m(k-1)A^\top + Q, \quad P_t(0) = P_0 \\ L(k) &= P_t(k)C^\top[CP_t(k)C^\top + R]^{-1} \\ P_m(k) &= P_t(k) - P_t(k)C^\top[CP_t(k)C^\top + R]^{-1}CP_t(k) \\ \hat{x}(k) &= A\hat{x}(k-1) + Bu(k-1) + L(k)\left(y(k) - C(A\hat{x}(k-1) + Bu(k-1))\right);\ \hat{x}(-1) = x_0 \\ x^*(k) &= \hat{x}(k) + P_m(k)A^\top[AP_m(k)A^\top + Q]^{-1}\left(x^*(k+1) - A\hat{x}(k)\right);\ x^*(T) = \hat{x}(T). \end{aligned} \tag{69}$$

Estimator properties

By now, it should be clear that there are very close connections between the least-squares estimator that we have derived and the Kalman filter. In fact, the recursions are identical with the following correspondences:

$$\begin{aligned} \hat{x}(k) &= \hat{x}(k|k) \\ P_t(k) &= P(k|k-1) \\ P_m(k) &= P(k|k). \end{aligned}$$

It can also be seen that in the recursion for $\hat{x}(k)$, we have combined the time and measurement updates, corresponding to what we called prediction and correction steps, respectively, in the Kalman filter. The weighting matrices Q and R in the LS objective were chosen ad hoc, and can now be seen to correspond to the noise covariance matrices in the Kalman filter with the same symbols. Having said all this, one may ask why we have bothered at all to re-derive an already known algorithm, albeit with a different motivation? We will return to this shortly.

A final comment about the result is motivated. In the DP derivation above, the goal was to minimize the least-squares objective (62) with respect to the sequence of state estimates $x(0\!:\!T)$. Looking at our solution again, we note that we did in fact not provide an explicit expression for the optimal solution, but it is instead defined implicitly via the sequence $\{\hat{x}(k)\}_{k=0}^{\top}$, see equation (69) above. The key relation is (67), which for a general time index k is

$$x^*(k) = \hat{x}(k) + P_m(k)A^\top[AP_m(k)A^\top + Q]^{-1}(x(k+1) - A\hat{x}(k)). \tag{70}$$

This is thus a recursion for the optimal solution, based on the estimates $\{\hat{x}(k)\}$ and with a final condition

$$x^*(T) = \hat{x}(T), \tag{71}$$

which stems from the fact that the last step in the DP recursion does not include a time update, which means that $\hat{x}(T)$ is the optimal choice of $x(T)$, as can be seen from the expression for V_T, corresponding to equation (68).

The result (69) is natural for the following reason. The optimal sequence of state estimates is found based on the complete set of measurements up to time T, and we may stress this fact by denoting the sequence of estimates by $x^*(0:T\,|\,T)$. This is in contrast to the sequence of estimates $\{\hat{x}(k)\}$, where each $\hat{x}(k)$ is the best estimate of $x(k)$, based on measurements up to and including time instant k. Only at the final time $k = T$ are the two estimates identical. The recursion (69) means that earlier estimates $\hat{x}(k)$ need to be *smoothed*, using information *after* time k to provide the optimal estimate $x^*(k)$.

The Kalman filter under certain conditions converges in the sense that the filter gain and the estimation error covariance converge to fixed values when time tends to infinity; see the previous section. This result holds under the standard assumptions used for the Kalman filter, i.e. with white noise process and measurement disturbances. Since the LS estimator is not based on any assumptions about disturbances, it is natural to ask for some other convergence property of the estimator. The simplest question to ask is whether the estimator produces reasonable results if there are no disturbances at all. The following result gives the answer.

Lemma 6.2 (Least squares estimator convergence)**.** *Assume that the LS estimator is applied to the system given by*

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k)) \\ y(k) &= Cx(k). \end{aligned}$$

If the pair (C, A) is observable and $Q, R \succ 0$, then the LS state estimate converges to the true system state

$$x^*(T|T) \to x(T), \quad T \to \infty.$$

6.3 Moving horizon estimation

We will now return to the question raised above, namely why we spent the effort to re-derive the algorithm for state estimation provided by the Kalman filter. The reason is a direct analogue to the optimal control problem treated in earlier sections. Remember that we started out to formulate a linear-quadratic optimal control problem for the infinite horizon case. Then, motivated by our desire to be able to treat cases beyond the *unconstrained LQ problem*, we moved to the *receding horizon* idea. In a similar way, we now argue that the Kalman filter (or the equivalent LS formulation) is limited to estimating the states of a linear system without constraints. Being able to generalize the concepts to nonlinear systems is thus a motivation for us (although not treated in this course), but the main reason is that we may want to include constraints on the estimated states. This desire reflects the fact that we often have additional physical insight—apart from the system model itself—that tells us that state variables are constrained in order to be physically meaningful. Making sure the state observer produces estimates that fulfil such constraints seems appropriate and may contribute to make the control optimisation better conditioned.

The Kalman filter (or the equivalent LS estimator) is based on an ever-increasing horizon, since all data from $k = 0$ is used. When we no longer can rely on the recursive solution, the natural way out is to focus on a finite record of measurements as the basis for the state estimation. If we use the T last measurements as the 'data window' (that will consequently slide as time passes by), we arrive at the following *moving horizon estimator* (MHE).

System model:

$$\begin{aligned} x(i+1) &= Ax(i) + Bu(i) \\ y(i) &= Cx(i). \end{aligned}$$

Optimisation problem:

$$\min_{x(k-T:k)} \hat{V}_T(x(k-T:k))$$

where the minimization is with respect to the sequence of state estimates

$$x(k-T:k) = \{x(k-T), x(k-T+1), \ldots, x(k)\}.$$

The objective function V_T is given by

$$\begin{aligned}\hat{V}_T(x(k-T:k)) = &\sum_{i=k-T}^{k-1} \left(x(i+1) - Ax(i) - Bu(i)\right)^\top Q^{-1} \left(x(i+1) - Ax(i) - Bu(i)\right) \\ &+ \sum_{i=k-T}^{k} \left(y(i) - Cx(i)\right)^\top R^{-1} \left(y(i) - Cx(i)\right). \end{aligned} \tag{72}$$

As outlined above, and in analogy with the LQ based receding horizon control formulation, it is now straightforward to add linear constraints to the moving horizon estimator. Changing the formulation slightly, we get the following scheme:

Objective function:

$$\hat{V}_T(x(k-T:k)) = \sum_{i=k-T}^{k-1} \omega(i)^\top Q^{-1} \omega(i) + \sum_{i=k-T}^{k} \nu(i)^\top R^{-1} \nu(i).$$

Optimisation problem:

$$\min \hat{V}_T(x(k-T:k))$$

with respect to

$$\{x(k-T:k),\, \omega(k-T:k-1),\, \nu(k-T:k)\}$$

and subject to

$$\begin{gathered} \omega(i) = x(i+1) - (Ax(i) + Bu(i)), \quad x(0) = x_0 \\ \nu(i) = y(i) - Cx(i) \\ x(i) \in \mathbb{X}, \quad \omega(i) \in \mathbb{W}, \quad \nu(i) \in \mathbb{V}, \quad \text{for all } i \in (k-T, k). \end{gathered}$$

Remark. *In this formulation of the moving horizon estimator, the only data used for the estimation is from the most recent T sampling instants. This is based on what is referred to in [1] as* zero prior weighting. *Alternative formulations with non-zero prior weighting are given in [1], but this goes beyond the scope of this course.*

7 Optimisation basics and convexity

The aim of the following two sections is to give a very brief introduction to constrained convex optimisation, an important ingredient of MPC. This section takes the first steps in introducing convexity for sets and functions and formulating the class of convex optimisation problems.

The textbooks [1] (Appendix A) and [2] (Sections 2.1-2.5) both provide an overview of optimisation techniques, although details differ from the introduction given here. In [4], mini-tutorial 3 on page 84 provides some basic facts, and Section 3.2 has some optimisation material embedded. If you want to read more on the topic, you are recommended to consult [5], from which a lot of material is available on the web. It is important to note, however, that optimisation is a vast area, and it is not the purpose here to go very far.

7.1 Introduction

In the previous sections, we have formulated a basic receding horizon controller, which in the LQ formulation involves an optimisation problem to be solved at each sampling instant,

$$\begin{aligned} \underset{\boldsymbol{u},\boldsymbol{x}}{\text{minimize}} \quad & V(x,\boldsymbol{u},\boldsymbol{x}) = x^\top Q x + \boldsymbol{x}^\top \bar{Q}\boldsymbol{x} + \boldsymbol{u}^\top \bar{R}\boldsymbol{u} \\ \text{subject to} \quad & x^+ = Ax + Bu \\ & F\boldsymbol{u} + G\boldsymbol{x} \leq h \end{aligned} \tag{73}$$

and in the more general case the analogous formulation is

$$\begin{aligned} \text{minimize} \quad & V_N(x, u(0\!:\!N-1)) = \sum_{i=0}^{N-1} l(x(i), u(i)) + V_f(x(N)) \\ \text{subject to} \quad & x^+ = f(x,u), \quad x(0) = x \\ & x(k) \in \mathbb{X}, \quad u(k) \in \mathbb{U} \quad \text{for all } k \in (0, N-1) \\ & x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. \end{aligned}$$

Both formulations are examples of *constrained optimisation* or *constrained mathematical programming* problems. It is clear that a vital part of model predictive control is to understand the nature of these optimisation problems, what type of difficulties they present, how algorithms can be devised, and which properties these algorithms have. Here, we will give a very brief introduction to this large area.

A note of caution regarding the notation used is in order. We will present some general concepts and results concerning constrained optimisation in this and the following section. This means that for most part, reference to the MPC applications will not be made. Therefore, we have chosen to adopt a general notation that is not linked to the MPC case. The *optimisation variables* or *decision variables* will thus be denoted x, which should not be confused with the state vector in the MPC context! With this in mind, we can proceed.

A basic optimisation problem is formulated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned} \tag{74}$$

where

$$\begin{aligned} x &= \{x_1, \ldots, x_n\} \text{ are the optimisation or decision variables} \\ f &: \mathbb{R}^n \to \mathbb{R} \text{ is the objective or cost function} \\ g_i &: \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m \text{ are inequality constraint functions} \\ h_i &: \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, p \text{ are equality constraint functions.} \end{aligned}$$

The *optimal solution* x^* has the smallest value of $f(\cdot)$ among all vectors x that belong to $\mathrm{dom}\, f$ (the *domain* of f, i.e. the subset of $\mathbb{R}^n$ where f is defined) and satisfy the constraints. The *optimal value* p^* is always defined,

$$\begin{aligned} p^* &= \inf\{f(x) \mid g_i(x) \leq 0,\ i = 1, \ldots, m; h_j(x) = 0,\ j = 1, \ldots, p\} \\ p^* &= \infty, \quad \text{if problem is infeasible} \\ p^* &= -\infty, \quad \text{if problem is unbounded below.} \end{aligned}$$

The *feasible set* $\mathcal{S}$ consists of all points satisfying the constraints, i.e. $\mathcal{S} = \{x \mid g_i(x) \leq 0, h_i(x) = 0, \forall i\}$. Notice that the optimisation problem may have a unique optimal solution, several alternative optimal solutions, or no optimal solution at all (when the problem is infeasible or unbounded below).

Example 7.1. Here are some examples of the definitions above:

- $f(x) = 1/x, \mathrm{dom}\, f = \mathbb{R}_{++}$(strictly positive reals) : $p^* = 0,$ no optimal solution
- $f(x) = -\log x, \mathrm{dom}\, f = \mathbb{R}_{++} : p^* = -\infty$
- $f(x) = x \log x, \mathrm{dom}\, f = \mathbb{R}_{++} : p^* = -1/e, x^* = 1/e$
- $f(x) = x^3 - 3x : p^* = -\infty,$ local optimum at $x = 1$.

■

The definitions above concern what is often referred to as the *global* optimum. When discussing optimisation algorithms and conditions for optimality, we also have use for the concept of *local* optimality. A *local optimum* x^* gives the lowest cost among feasible points in a neighbourhood of x^*, i.e. there is an $r > 0$ such that

$$x \text{ feasible and } \quad |x - x^*| \leq r \quad \Rightarrow \quad f(x) \geq f(x^*). \tag{75}$$

A global optimum is necessarily also a local optimum, but the converse is not generally true. Still, many optimisation algorithms are constructed based on properties of local optima, to which we will turn next.

7.2 Conditions for optimality – the KKT conditions

Many optimisation algorithms use the properties of local optima to search for the optimal solution. The condition (75) can alternatively be expressed using the concept of feasible directions. A *feasible direction* d at a feasible point x is any direction in which an infinitesimal (arbitrarily small) move can be made while staying inside the feasible set[3]. Figure 12 illustrates the concept for the case with two inequalities in $\mathbb{R}^2$ with the feasible directions from the point of intersection.

Using this "definition", a local optimum x^* is characterized by the property that, for any feasible direction d,

$$f(x^* + \varepsilon d) - f(x^*) \geq 0, \quad \varepsilon \text{ small enough.} \tag{76}$$

Assuming f is twice differentiable, we can use a Taylor expansion around x^* to express the objective in the vicinity of x^*,

$$f(x^* + \varepsilon d) - f(x^*) = \varepsilon \nabla f(x^*)^\top d + \frac{1}{2}\varepsilon^2 d^\top \nabla^2 f(x^*) d + \text{(higher order terms)}, \tag{77}$$

[3]The presentation is kept informal and intuitive here; refer to [1], Appendix C for a strict treatment in terms of so called *tangent cones*.

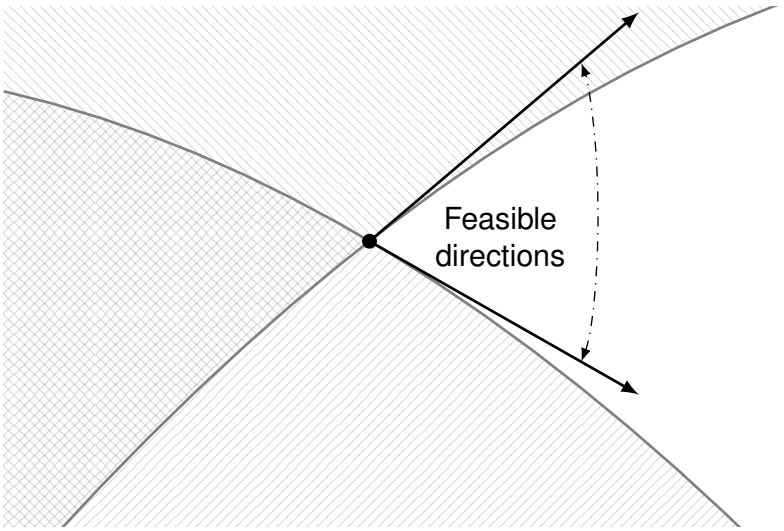


Figure 12: Feasible directions from the point of intersection of two nonlinear constraints.

where ∇f and $\nabla^2 f$ are the *gradient* column vector ($n \times 1$) and the *Hessian* matrix ($n \times n$) of the function f, respectively,

$$\nabla f = \left(\frac{\partial f}{\partial x}\right)^\top, \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2}.$$

In the unconstrained case (without equality or inequality constraints g and h), all directions d are feasible, and the two expressions above can be combined to arrive at the familiar conditions for (local) optimality.

- **First order necessary condition**

$$x^* \text{ is a stationary point} \quad \Rightarrow \quad \nabla f(x^*) = 0. \tag{78}$$

- **Second order sufficient conditions**

$$\nabla f(x^*) = 0 \text{ and } \nabla^2 f(x^*) \succ 0 \quad \Rightarrow \quad x^* \text{ is a strict local minimum.} \tag{79}$$

Equality constraints

Consider now the case with equality constraints, so that the optimisation problem becomes

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad h(x) = 0 \end{aligned} \tag{80}$$

where the vector-valued function $h(x)$ has p components $h_i(x)$. In this case, the feasible directions must be along the surface $h(x) = 0$, and we can investigate these by again using a Taylor expansion around the feasible point x^*:

$$h(x^* + \varepsilon d) \approx h(x^*) + \varepsilon \nabla h(x^*)^\top d = \varepsilon \nabla h(x^*)^\top d, \tag{81}$$

where we have used the convention

$$\nabla h = \begin{bmatrix} \nabla h_1 & \cdots & \nabla h_p \end{bmatrix}.$$

It follows that we must have $\nabla h(x^*)^\top d = 0$ for any feasible direction d. If we assume that x^* is *regular*, i.e. that $\nabla h(x^*)$ has full column rank, it can be shown that the converse also holds. Hence, if x^* is regular, then the set of feasible directions at x^* is given by the nullspace of $\nabla h(x^*)^\top$.

Going back to the local optimality condition (76), we can conclude that for any local optimum x^*, we must have $\nabla f(x^*)^\top d \geq 0$ for any d such that $\nabla h(x^*)^\top d = 0$, provided x^* is regular. From this, the following result can be shown.

Consider the optimisation problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h(x) = 0. \end{aligned}$$

Assume x^* is a local minimum and that x^* is regular. Then there is a unique vector λ^* such that

$$\begin{aligned} \nabla f(x^*) + \nabla h(x^*)\lambda^* &= 0 \\ h(x^*) &= 0. \end{aligned}$$

This is a system of non-linear equations having $n + p$ equations for the $n + p$ unknowns (x and λ). The vector λ contains the *Lagrange multipliers* λ_i, $i = 1, \ldots, p$.

Inequality constraints

Let us now return to the original constrained optimisation problem (74), containing also inequality constraints. For any feasible point x, it is important to distinguish between two cases: either $g_i(x) < 0$, in which case the constraint is *inactive*, or $g_i(x) = 0$, in which case the constraint is *active*. The set of indices for the active constraints is called the *active set* $\mathbb{A}$.

When investigating candidates for local optima, any active inequality constraints are similar to equality constraints, since the point lies on the boundary of the set, which is feasible with respect to that constraint. In line with this observation, the concept of a regular point is generalized as follows: a feasible point x is said to be *regular* if the active constraints are linearly independent, i.e.

$$\begin{bmatrix} \nabla g_{\mathbb{A}} & \nabla h \end{bmatrix} \text{ has full column rank.}$$

Here $\nabla g_{\mathbb{A}}$ is formed from the gradients of the active inequality constraints.

We are now ready to state the generalization of the first order necessary conditions. Consider the optimisation problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0. \end{aligned} \tag{82}$$

Assume x^* is a local minimum and that x^* is regular. Then there are unique vectors μ^* and λ^* such that

$$\nabla f(x^*) + \nabla g(x^*)\mu^* + \nabla h(x^*)\lambda^* = 0 \tag{83a}$$

$$\mu^* \geq 0 \tag{83b}$$

$$g(x^*) \leq 0, \quad h(x^*) = 0 \tag{83c}$$

$$\mu_i^* g_i(x^*) = 0, \quad i = 1, \ldots, m. \tag{83d}$$

These conditions are referred to as the *KKT (Karush-Kuhn-Tucker) conditions*.

In this case, the Lagrange multipliers (also called the *dual variables*) are μ for the inequality constraints and λ for the equality constraints. They differ in that the *dual constraints* (83b) specify that μ have to be nonnegative. The *primal variables* x should satisfy the *primal constraints* (83c). The first condition (83a) can conveniently be expressed as a condition on the *Lagrangian* $\mathcal{L}$,

$$\nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) = 0, \qquad \mathcal{L}(x, \mu, \lambda) = f(x) + \mu^\top g(x) + \lambda^\top h(x). \tag{84}$$

The last conditions (83d) of the KKT conditions are called the *complementary slackness* conditions. The implication of these is that if g_i is inactive at x^* then $\mu_i^* = 0$. Conversely, if g_i is active, then either $\mu_i > 0$ (the constraint is *strictly active*) or $\mu^* = 0$ (the constraint is not strictly active).

The condition (83a) has an intuitive geometrical/physical interpretation. At a local optimum, you can think of ∇f as a force that is trying to pull the solution down towards lower function values. Equation (83a) expresses that this force is balanced by forces from the active constraints. In doing this, equality constraints can exert forces in either direction, whereas inequality constraints can only pull into the feasible side of the constraint boundary, reflected by the condition $\mu > 0$.

The KKT conditions are very useful for searching candidates for local optima, and they are also the basis for many optimisation algorithms as we will see later on. However, it is important to remember that the KKT conditions only provide *necessary* conditions. As a complement to this, the following result states *sufficient* conditions for a local optimum.

Consider the optimisation problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & h(x) = 0. \end{array} \tag{85}$$

Assume x^* is regular and that x^*, μ^*, λ^* satisfy the KKT conditions with all active constraints being strictly active. Further assume that

$$d^\top \nabla_x^2 \mathcal{L}(x^*, \mu^*, \lambda^*) d > 0, \quad \text{for all } d \text{ such that } d^\top \begin{bmatrix} \nabla g_{\mathbb{A}} & \nabla h \end{bmatrix} = 0,$$

where $\nabla_x^2 \mathcal{L}$ is the Hessian of the Lagrangian. Then x^* is a local minimum.

7.3 Convex optimisation

General nonlinear programming (NLP) problems can be quite challenging to solve, and therefore there are several important subclasses of optimisation problems, whose structure can be exploited in different ways. One of the difficulties with general NLP is that there may be several local minima. From this perspective, the important class of *convex* optimisation problems turns out to be of particular interest:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

where the objective function f and constraint functions $\{g_i\}$ are convex, i.e.

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta) f(x_2), \quad 0 \leq \theta \leq 1$$

and the functions $\{h_i\}$ are *affine* (linear).

There are two main reasons for our interest in convex optimisation problems. The first one is that any local minimum is also a global minimum, and that it is possible to solve very large problems efficiently. The second one is that our constrained LQ based MPC leads to a special type of convex optimisation problem, as will be seen later. However, in order to better understand the properties of convex optimisation, we first need to learn some terminology of **convex sets** and **convex functions**.

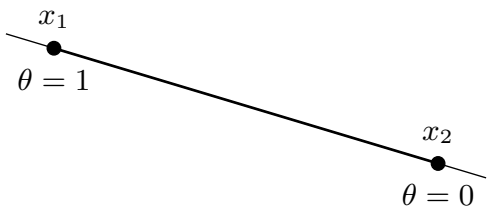


Figure 13: An illustration of an affine set in two dimensions.

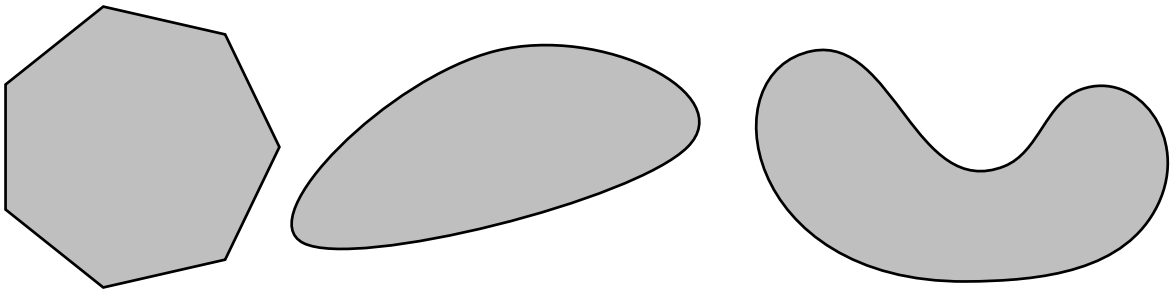


Figure 14: An example with two convex and one non-convex set.

Convex sets

- A **line** through x_1 and x_2 are all points x,

$$x = \theta x_1 + (1 - \theta)x_2, \quad \theta \in \mathbb{R}.$$

- An **affine set** contains the line through any two distinct points in the set, see Figure 13. An example is the solution set of linear equations $\{x \mid Ax = b\}$.

  All affine sets can be described as solutions to a system of linear equations.

- A **line segment** between x_1 and x_2 are all points x,

$$x = \theta x_1 + (1 - \theta)x_2, \quad 0 \leq \theta \leq 1.$$

- A **convex set** contains the line segments between every two points in the set (see Figure 14), i.e.

$$x_1, x_2 \in \mathcal{S} \;\Rightarrow\; \theta x_1 + (1 - \theta)x_2 \in \mathcal{S}, \quad 0 \leq \theta \leq 1.$$

- A **hyperplane** is a set of the form $\{x \mid a^\top x = b\}$.
- A **half-space** is a set of the form $\{x \mid a^\top x \leq b\}$.

  Hyperplanes are affine and convex; half-spaces are convex, see Figure 15.

- A **polyhedron** is the intersection of a finite number of half-spaces and hyperplanes or, equivalently, the solution set of a finite number of linear inequalities and equalities (see Figure 16), i.e.

$$\begin{aligned} Ax &\leq b \\ Cx &= d. \end{aligned}$$

  Polyhedra are convex sets.

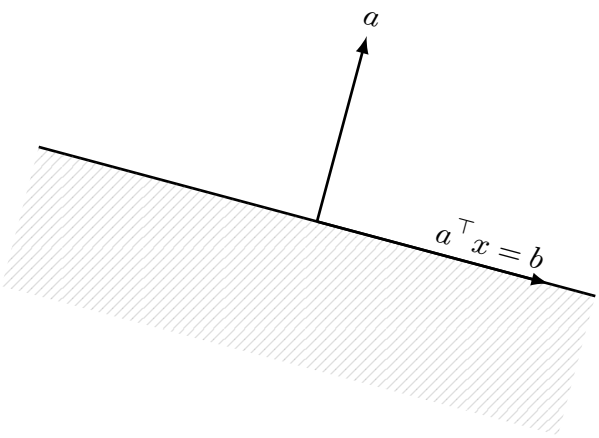


Figure 15: A hyperplane $a^\top x = b$ and a half-space depicted by the shaded region.

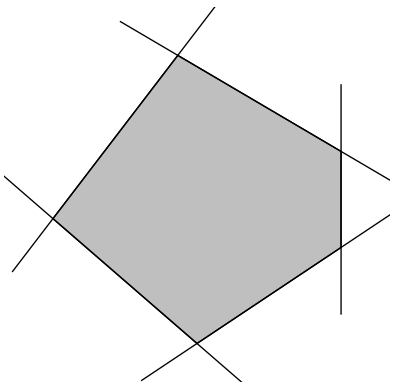


Figure 16: An example of a polyhedron, as the intersection of linear inequalities and equalities.

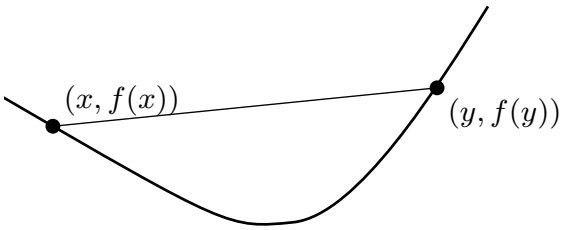


Figure 17: An example of a convex function.

Convex functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if $\operatorname{dom} f$ is convex and

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$$

for all $x, y \in \operatorname{dom} f$ and $0 \leq \theta \leq 1$. An illustration is provided in Figure 17.

Furthermore,

- f is *concave* if $-f$ is convex.
- f is *strictly convex* if
$$f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta)f(y)$$
for all $x, y \in \operatorname{dom} f$ and $0 < \theta < 1$.

Some examples of convex functions are

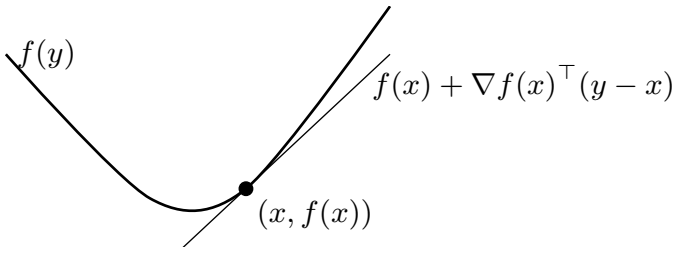


Figure 18: A convex function and its tangent.

- affine: $a^\top x + b$;
- exponential: e^{ax};
- powers: $x^\alpha, \quad x > 0$, for $\alpha \geq 1$ or $\alpha \leq 0$.

Examples of concave functions are

- affine: $a^\top x + b$;
- logarithm: $\log x, \ x > 0$;
- powers: $x^\alpha, \quad x > 0$, for $0 \leq \alpha \leq 1$.

First and second order conditions

- Differentiable f with convex domain is convex if and only if

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \text{for all } x, y \in \operatorname{dom} f.$$

  An illustration is provided in Figure 18.

- Twice differentiable f with convex domain is convex if and only if

$$\nabla^2 f(x) \geq 0 \quad \text{for all } x \in \operatorname{dom} f.$$

Operations that preserve convexity

- The **intersection** of convex sets is a convex set.
- If f is **affine** ($f(x) = Ax + b$), then the image of a convex set under f is convex, i.e.

$$\mathcal{S} \text{ convex } \Rightarrow \ f(\mathcal{S}) \text{ convex.}$$

- If f is affine, then the **inverse image** of f is convex, i.e.

$$\mathcal{S} \text{ convex } \Rightarrow \ f^{-1}(\mathcal{S}) = \{x \mid f(x) \in \mathcal{S}\} \text{ convex.}$$

  Examples include scaling, translation, projection.

- **Sub-level sets** $\mathcal{S}_\alpha$ of a convex function f are convex

$$\mathcal{S}_\alpha = \{x \in \operatorname{dom} f \mid f(x) \leq \alpha\}.$$

- **Nonnegative weighted sum** of convex functions is convex,

$$f_1, \ldots, f_N \text{ convex } \Rightarrow \ \sum_{i=1}^{N} \alpha_i f_i \text{ convex, for all } \alpha_i \geq 0.$$

- **The composition with an affine function** is convex,

$$f \text{ convex } \Rightarrow \ f(Ax + b) \text{ convex.}$$

Convex optimisation

Convex sets and convex functions, as discussed above, play an important role in the formulation and solution of *convex optimisation problems*. This class of optimisation problems is of great interest in general, and in particular for the design of model predictive controllers. The reason for this is that convex optimisation problems have 'nice' properties, which allow solutions to be computed efficiently, even for large size problems.

Standard form convex optimisation problem:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b, \quad \text{(affine equality constraints)} \end{array}$$

where f and $\{g_i\}$ are convex.

Remark. *The* feasible set *of a convex optimisation problem is convex.*

Example 7.2 ([5])**.** Consider the following optimisation problem:

$$\begin{array}{ll} \text{minimize} & f(x) = x_1^2 + x_2^2 \\ \text{subject to} & g_1(x) = x_1/(1 + x_2^2) \leq 0 \\ & h_1(x) = (x_1 + x_2)^2 = 0. \end{array}$$

It is not difficult to see that

- $f(x)$ is convex and the feasible set $\{(x_1, x_2) \mid x_1 = -x_2 \leq 0\}$ is convex, but ...
- the problem is not in the standard form, since g_1 is not convex and h_1 is not affine.

It is, however, in this case possible to transform the given optimisation problem into an equivalent, convex formulation:

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 \\ \text{subject to} & x_1 \leq 0 \\ & x_1 + x_2 = 0. \end{array}$$

■

For a general optimisation problem, any globally optimal point is also a locally optimal point, i.e. it gives the lowest cost among feasible points in its neighbourhood. A very important property of convex optimisation problems is that the converse is also true, as stated in the following proposition.

Proposition 7.1 (Local and global optima)**.** *Any local optimum of a convex optimisation problem is (globally) optimal.*

Proof. Suppose x is locally optimal and y is optimal with $f(y) < f(x)$. That x is locally optimal means there is an $r > 0$ such that

$$z \text{ feasible and } \quad |z - x| \leq r \quad \Rightarrow \quad f(z) \geq f(x).$$

Choose an r such that $r < |y - x|$. Consider now $z = \theta y + (1 - \theta)x$ with $\theta = r/(2|y - x|)$. Then

- $|y - x| > r$, implying $0 < \theta < 1/2$;

- z is a convex combination of two feasible points, hence also feasible;
- $|z - x| = r/2$ and
$$f(z) \leq \theta f(x) + (1-\theta) f(y) < f(x)$$
which contradicts the assumption that x is locally optimal.

□

The equivalence of local and global optima for convex problems opens up for strengthening the optimality conditions stated earlier: Consider the convex optimisation problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & h(x) = 0 \end{array} \tag{86}$$

where f and $\{g_i\}$ are convex and h is affine. Assume x^* is regular. Then x^* is globally optimal if and only if the KKT conditions are fulfilled for some $\mu^* \geq 0$, λ^*.

Two common examples of convex optimisation programs are

- **Linear programming (LP)**:
$$\begin{array}{ll} \text{minimize} & c^\top x + d \\ \text{subject to} & Gx \leq h \\ & Ax = b; \end{array}$$
- **Quadratic programming (QP)**:
$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^\top Q x + p^\top x, \quad Q \succeq 0 \\ \text{subject to} & Gx \leq h \\ & Ax = b. \end{array}$$

In both cases, the feasible set is a polyhedron.

The two types of optimisation problems described above are probably the most well-known classes of convex optimisation problems, and there are many efficient numerical algorithms around to solve them. This is good news for us, since the QP problem in particular is very appropriate for our needs; cf. problem (73)!

We end this section with an example with a special case of QP.

Example 7.3 (QP with inequality constraints only)**.** Consider the QP problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^\top Q x + p^\top x, \quad Q \succeq 0 \\ \text{subject to} & Gx \leq h. \end{array}$$

Assuming that G has full row rank, any point x is regular. Then global optimality is equivalent to the KKT conditions being fulfilled. Denoting the objective by $f(x)$, this can be stated in a simplified way as follows: the point x^* is optimal if and only if x^* is feasible (i.e. $Gx^* \leq h$) and

$$-\nabla f(x^*) = -(Qx^* + p) = \sum_{i \in \mathbb{A}} \mu_i G_i^\top, \quad \text{for some } \{\mu_i\} \text{ with } \mu_i \geq 0, \tag{87}$$

where G_i is the ith row of G and $\mathbb{A}$ is the active set. Figure 19 gives a geometric interpretation. ■

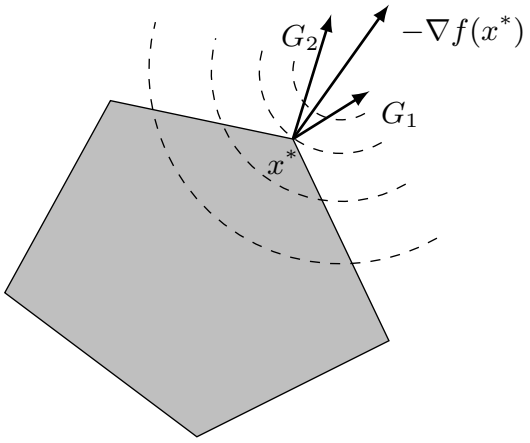


Figure 19: An illustration of a quadratic program with inequality constraints.

7.4 Duality*

We have already noted that the KKT conditions can conveniently be expressed in terms of the *Lagrangian* $\mathcal{L}$. In fact, the Lagrangian plays an important role in developing *duality theory*, which provides new perspectives on constrained optimisation. This section gives some of the fundamentals.

Lagrangian

Consider the standard form problem (not necessarily convex)

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

where $x \in \mathcal{D} \subseteq \mathbb{R}^n$.

The *Lagrangian* $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$, with $\operatorname{dom} \mathcal{L} = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$, is defined as

$$\mathcal{L}(x, \mu, \lambda) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{i=1}^{p} \lambda_i h_i(x) \equiv f(x) + \mu^\top g(x) + \lambda^\top h(x)$$

where

- μ_i is Lagrange multiplier associated with the constraint $g_i(x) \leq 0$;
- λ_i is Lagrange multiplier associated with the constraint $h_i(x) = 0$.

Note that for $\mu \geq 0$ and any feasible x, we have $\mathcal{L}(x, \mu, \lambda) \leq f(x)$.

For a convex optimisation problem, the Lagrangian is a linear combination of convex functions, hence convex. In many cases it is therefore possible to compute the unconstrained minimum of $\mathcal{L}$. This fact naturally leads to the next definition.

Lagrange dual function $q : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$,

$$q(\mu, \lambda) = \inf_{x \in \mathcal{D}} \mathcal{L}(x, \mu, \lambda) = \inf_{x \in \mathcal{D}} \left\{ f(x) + \mu^\top g(x) + \lambda^\top h(x) \right\}.$$

Properties:

- q is concave but may be $-\infty$ for some μ, λ;
- $q(\mu, \lambda) \leq p^*$ if $\mu \geq 0$ (p^* is the optimal value of the original problem).

The *Lagrange dual problem*

$$\begin{aligned} \text{maximize} \quad & q(\mu, \lambda) \\ \text{subject to} \quad & \mu \geq 0 \end{aligned}$$

- finds the best lower bound d^* on the primal optimal solution p^*,
- always is a convex, unconstrained problem,
- has *dual feasible* μ, λ if $\mu \geq 0$ and $(\mu, \lambda) \in \operatorname{dom} q$,
- always satisfies $d^* \leq p^*$ (*weak duality*).

Example 7.4 (The dual of an LP problem)**.** Consider the standard linear program

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b, \quad x \geq 0. \end{aligned}$$

The Lagrangian is given by

$$\mathcal{L}(x, \mu, \lambda) = c^\top x - \mu^\top x + \lambda^\top (Ax - b) = -b^\top \lambda + (c + A^\top \lambda - \mu)^\top x$$

The Lagrange dual function is then obtained by solving

$$q(\mu, \lambda) = \inf_x \mathcal{L}(x, \mu, \lambda) = \begin{cases} -b^\top \lambda, & A^\top \lambda - \mu + c = 0 \\ -\infty, & \text{otherwise.} \end{cases}$$

It holds:

- q is linear on the affine domain $\{(\mu, \lambda) \mid A^\top \lambda - \mu + c = 0\}$, i.e. concave
- lower bound: $p^* \geq -b^\top \lambda$ if $A^\top \lambda + c \geq 0$.

We can make the implicit constraint $(\mu, \lambda) \in \{(\mu, \lambda) \mid A^\top \lambda - \mu + c = 0\}$ explicit when formulating the dual problem:

$$\begin{aligned} \text{maximize} \quad & -b^\top \lambda \\ \text{subject to} \quad & A^\top \lambda + c \geq 0. \end{aligned}$$

■

Weak and strong duality

Weak duality: $d^* \leq p^*$

- always holds (even for non-convex problems);
- gives lower bound for the original (*primal*) problem.

Strong duality: $d^* = p^*$

- does not hold in general;
- often holds for convex problems;
- conditions that guarantee this are called *constraint qualifications*.

It is stated above that strong duality often holds for convex problems. In fact, we have already met conditions ensuring this, namely *regularity* conditions on candidates for local optima. Such conditions are called *constraint qualifications*.

Strong duality holds for a convex problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array}$$

if any of the following conditions are fulfilled:

1. The gradients of the equality constraints and the active inequality constraints are linearly independent (LICQ);
2. The problem is strictly feasible, i.e. there exists some $\tilde{x} \in \text{int}\mathcal{D}$ (the interior of $\mathcal{D}$) such that (Slater CQ)
$$g_i(\tilde{x}) < 0, \quad i = 1, \ldots, m; \quad A\tilde{x} = b.$$

Example 7.5 (Quadratic programming)**.** Consider the quadratic program (assuming P is positive definite, $P \succ 0$)

$$\begin{array}{ll} \text{minimize} & x^\top P x \\ \text{subject to} & Ax \leq b. \end{array}$$

The dual function is

$$q(\mu) = \inf_x (x^\top P x + \mu^\top (Ax - b)) = -\frac{1}{4}\mu^\top A P^{-1} A^\top \mu - b^\top \mu.$$

Hence, the dual problem is defined as

$$\begin{array}{ll} \text{maximize} & -\frac{1}{4}\mu^\top A P^{-1} A^\top \mu - b^\top \mu \\ \text{subject to} & \mu \geq 0. \end{array}$$

It follows from Slater's condition that $p^* = d^*$ if $A\tilde{x} < b$ for some $\tilde{x}$. In fact, for convex quadratic programs, we *always have strong duality*, i.e. $p^* = d^*$. ■

Now, assume that x^* is regular and hence strong duality holds with x^* primal optimal, (μ^*, λ^*) dual optimal. Then

$$\begin{aligned} f(x^*) &= \inf_x \left(f(x) + g(x)^\top \mu^* + h(x)^\top \lambda^* \right) \\ &\leq f(x^*) + g(x^*)^\top \mu^* + h(x^*)^\top \lambda^* \leq f(x^*) \end{aligned}$$

which means that the two inequalities must hold with equality. Therefore

- x^* minimizes $\mathcal{L}(x, \mu^*, \lambda^*)$
- $\mu_i^* g_i(x^*) = 0$ for $i = 1, \ldots, m$

which, together with primal and dual constraints, we recognise as the KKT conditions! The conclusion is that if strong duality holds, then any optimal x^*, μ^*, λ^* must satisfy the KKT conditions. This confirms the first order necessary conditions that we have arrived at earlier (assuming regularity).

Conversely, assume that x^*, μ^*, λ^* satisfy the KKT conditions. Then they are optimal, since

1. From complementarity conditions, it follows that $f(x^*) = \mathcal{L}(x^*, \mu^*, \lambda^*)$;
2. Since $\mathcal{L}$ is convex in x and its gradient is 0 from the KKT conditions, it follows that $q(\mu^*, \lambda^*) = \mathcal{L}(x^*, \mu^*, \lambda^*)$.

Hence, $f(x^*) = q(\mu^*, \lambda^*)$ and strong duality holds. Let's summarize. Consider the standard convex optimisation problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array}$$

where f and $\{g_i\}$ are convex.

Assume x^* is regular. Then the following statements are equivalent:

- x^* is a global optimum;
- there are μ^*, λ^* such that the KKT conditions hold.

8 Solving QP problems

In the previous section, we have investigated conditions for optimality of constrained optimisation problems, in particular convex ones. The objective of this section is to go one step further and have a look at how to construct algorithms to find the optimum. We will give the basic ideas of how Newton's method can be applied to general NLP problems, but we will focus in particular on the QP case, which we encounter in our MPC algorithms.

The material of the textbooks is relatively scarce. Most of the material on optimisation in [2] is again in Chapter 2; this section corresponds to Sections 2.5-2.6. Algorithms for QP are introduced in Section 8.3. In [4], Sections 3.3-3.4 contain some of the material of this section. For further reading, it is once again recommended to consult [5].

8.1 Newton's method

Let's start by recalling that for convex optimisation problems (assuming regularity), necessary and sufficient conditions for optimality are given by the KKT conditions. For the case with only equality constraints, the KKT conditions are particularly simple,

$$\begin{aligned} \nabla_x \mathcal{L}(x,\lambda) &= 0 \\ h(x) &= 0 \end{aligned} \tag{88}$$

where $\mathcal{L}(x,\lambda) = f(x) + \lambda^\top h(x)$. Finding the optimum is therefore equivalent to solving a system of nonlinear equations in the unknowns x, λ. There is a famous algorithm which does exactly this, namely *Newton's method*. In its simplest form, it is used to find the root of the scalar equation $r(x) = 0$. The idea is to approximate the function by a straight line at the current "guess" x, and to obtain the next guess[4] $x^+ = x + \Delta x$ as the root of the linear approximation

$$r(x + \Delta x) \approx r(x) + r'(x)\Delta x = 0 \quad \Rightarrow \quad \Delta x = -(r'(x))^{-1} r(x). \tag{89}$$

When x and $r(x)$ are vectors, the *Newton step* is a direct generalization of this, i.e.

$$\nabla r(x)^\top \Delta x = -r(x), \tag{90}$$

which is now a system of *linear* equations in the unknowns Δx. We have thus replaced the original problem to solve a system of nonlinear equations to a task to repeatedly solve a system of linear equations.

Since Newton's method is based on a linear approximation of $r(x)$, it turns out that it is usually wise in practice not to perform the *full Newton step* as given by (90). Instead, while still going in the *Newton direction* as prescribed by (90), the new iterate is obtained by using a *reduced step size*:

$$x^+ = x + t\Delta x, \quad t \in (0, 1]. \tag{91}$$

Newton's method for equality constrained problems

Let's now try to apply Newton's method to the KKT conditions (88), observing that the unknowns are x, λ. The Newton step (90) becomes

$$\begin{bmatrix} \nabla_x^2 \mathcal{L}(x,\lambda) & \nabla_{x,\lambda} \mathcal{L}(x,\lambda) \\ \nabla h(x)^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ h(x) \end{bmatrix},$$

[4] The notation x^+ is used here for the next iterate, not a time update.

which, by using $\mathcal{L}(x,\lambda) = f(x) + \lambda^\top h(x)$, can be simplified into

$$\begin{bmatrix} \nabla_x^2 \mathcal{L}(x,\lambda) & \nabla h(x) \\ \nabla h(x)^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + \nabla h(x)\lambda \\ h(x) \end{bmatrix},$$

and finally a simple re-organization gives

$$\underbrace{\begin{bmatrix} \nabla_x^2 \mathcal{L}(x,\lambda) & \nabla h(x) \\ \nabla h(x)^\top & 0 \end{bmatrix}}_{\text{The KKT matrix}} \begin{bmatrix} \Delta x \\ \lambda^+ \end{bmatrix} = - \begin{bmatrix} \nabla f(x) \\ h(x) \end{bmatrix},$$

where $\lambda^+ = \lambda + \Delta\lambda$ is the new iterate of the dual variable λ. Let us interpret this result for the QP case:

$$\text{minimize} \quad f(x) = \frac{1}{2} x^\top Q x + p^\top x, \quad Q \succ 0 \tag{92}$$

$$\text{subject to} \quad Ax = b, \quad A \in \mathbb{R}^{p \times n}. \tag{93}$$

The Lagrangian is

$$\mathcal{L}(x,\lambda) = \frac{1}{2} x^\top Q x + p^\top x + \lambda^\top (Ax - b).$$

and the Newton's method gives

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \lambda^+ \end{bmatrix} = - \begin{bmatrix} Qx + p \\ Ax - b \end{bmatrix} \quad \Leftrightarrow$$

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} -p \\ b \end{bmatrix}. \tag{94}$$

Notice that equation (94) gives the next iterates x^+, λ^+ as the solution of a system of linear equations, *independent of any previous iterate*! In the QP case, the Newton method thus gives the solution in one step. In fact, it is easily verified that equation (94) is identical to the KKT conditions (do this!). An analytical solution can be derived as follows. Multiply the first block row in (94) by AQ^{-1} and subtract the result from the second block row. This gives the equivalent system of equations (replacing the $+$-sign with 0 to indicate optimal solution)

$$\begin{bmatrix} Q & A^\top \\ 0 & -AQ^{-1}A^\top \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -p \\ AQ^{-1}p + b \end{bmatrix} \tag{95}$$

which immediately can be solved, first for λ^*, then for x^*:

$$\lambda^* = -(AQ^{-1}A^\top)^{-1}(AQ^{-1}p + b) \tag{96}$$

$$x^* = -Q^{-1}(p + A^\top \lambda^*) = -Q^{-1}\left(p - A^\top (AQ^{-1}A^\top)^{-1}(AQ^{-1}p + b)\right). \tag{97}$$

The procedure applied is a block Gaussian elimination and the matrix $-AQ^{-1}A^\top$ is called the *Schur complement* of Q in the KKT matrix.

We summarize our findings, namely that the solution to the QP problem without constraints or with only equality constraints, can be given as the solution to a system of linear equations.

- Unconstrained case

  $$\text{minimize} \quad f(x) = \frac{1}{2} x^\top Q x + p^\top x, \quad Q \succ 0$$

  has a solution

  $$Qx^* + p = 0.$$

- QP with equality constraint

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{1}{2}x^\top Qx + p^\top x, \quad Q \succ 0 \\ \text{subject to} \quad & Ax = b \end{aligned}$$

has a solution

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -p \\ b \end{bmatrix}.$$

8.2 Inequality constraints

We remind of the fact that the KKT conditions for the equality constrained QP problem leads to a set of *linear* equations to be solved. When we now turn to the QP problem including also inequality constraints, it will be seen that we instead get a system of *nonlinear* equations. Newton's method still provides the basic tool to solve this. In addition to providing the solution to QP problems, Newton's method is also the basis for many optimisation algorithms that are aimed for other convex optimisation problems. The gradient and the Hessian then give a local quadratic approximation of the objective function (and since they are local, have to be computed repeatedly). The Newton method in most cases has to be executed iteratively, since the optimal point will not be reached in one step. The method also has to be extended with some type of *line searching* technique, in which the *Newton search direction* is kept, but *backtracking* is used to take shorter steps. The reason is that the full *Newton step* may lead too far, where the local quadratic approximation is not any longer good enough.

With these general remarks, let's now turn to the QP problems with inequality constraints,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^\top Qx + p^\top x, \quad Q \succeq 0 \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b. \end{aligned} \tag{98}$$

For easy reference, we repeat the KKT conditions (83a)-(83d) for this special case

$$Qx^* + p + G^\top \mu^* + A^\top \lambda^* = 0 \tag{99}$$

$$\mu^* \geq 0 \tag{100}$$

$$Gx^* - h \leq 0, \quad Ax^* - b = 0 \tag{101}$$

$$\mu_i^*(g_i^\top x^* - h_i) = 0, \quad i = 1, \ldots, m. \tag{102}$$

Here $g_i^\top$ is the ith row of G and h_i is the ith element of h. The algorithms we will discuss are aimed at solving this system of equations. Note that due to the presence of inequality constraints, these equations are not any longer linear!

Active set methods

It was noted already when we discussed the general KKT conditions that the active inequality constraints play a similar role as the equality constraints. This observation is the idea behind so called *active set methods*. Begin by observing that the KKT conditions for the QP problem can be given an alternative form, which resembles the KKT conditions (94) for the QP with only equality constraints[5]:

$$\begin{bmatrix} Q & A^\top & G_{\mathbb{A}}^\top \\ A & 0 & 0 \\ G_{\mathbb{A}} & 0 & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \\ \mu_{\mathbb{A}}^+ \end{bmatrix} = \begin{bmatrix} -p \\ b \\ h_{\mathbb{A}} \end{bmatrix}. \tag{103}$$

[5]Here we have returned to the notation x^+ etc. in order to stress the iterative nature of the algorithm.

The crux of this solution is of course that we do not know the active set $\mathbb{A}$ beforehand. The equations should hold for the *correct* active set, meaning that $\mu_{\mathbb{A}}^{+} > 0$ for the (strictly) active constraints, and $G_{\bar{\mathbb{A}}}x < h_{\bar{\mathbb{A}}}$ for the inactive constraints ($\bar{\mathbb{A}}$ being the complement of $\mathbb{A}$). However, an algorithm can be constructed based on repeatedly solving (103) for different candidate active sets $\mathbb{A}$. The idea is as follows.

Assume that a feasible point is known with a specific $\mathbb{A}$ (and consequently $\bar{\mathbb{A}}$). With this $\mathbb{A}$, the system of equations (103) can be solved. Once this is done, there are two possible outcomes:

1. If the new point is feasible with respect to the (previously inactive) inequality constraints, we need to test if we are at the optimum. This is done by checking the Lagrange multipliers corresponding to the active set; they should all be nonnegative at the optimum. If this is not the case, the objective function can be further reduced by e.g. removing the constraint with the most negative multiplier from the active set, and the procedure is repeated.

2. If the new point is not feasible, then the stepsize is reduced so that the new point becomes (just) feasible. This happens at the intersection with one of the previously inactive constraints. This is now added to the active set and the procedure is repeated.

The active set method has the advantage that it produces iterates that are all feasible, which means that intermediate results are still usable in a case where there may not be time to wait for the optimal solution. A disadvantage is that with many inequality constraints, there are a large number of different combinations of constraints forming potential active sets, and many of these may have to be searched before the algorithm converges. In fact, this potential *combinatorial explosion* is the reason why there are no tight complexity bounds for active set methods; even though the algorithms may perform very well "on the average", specific problem instances may require significantly longer execution times, which may be a problem in a real-time application as MPC. Note also that the method requires an initial feasible point to start with; we will not pursue this further.

Interior point methods

There is another class of algorithms for solving convex optimisation problems called *interior point methods*. These have become quite popular, particularly for large problems, and also for MPC applications. Similar to the active set method, the interior point methods solve the original optimisation problem by applying Newton's method to a sequence of equality constrained problems, or to a sequence of (modified) sets of KKT conditions. In either case, the goal is to somehow avoid the very nonlinear complementarity conditions (102).

The starting point is a reformulation, or rather approximation, of the original QP problem (98) into an equality constrained problem as follows. The QP problem

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{1}{2}x^\top Q x + p^\top x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

can be approximated by the following problem

$$\begin{aligned} \text{minimize} \quad & f_\tau(x) = f(x) - \tau \sum_{i=1}^{m} \log(h_i - g_i^\top x) \quad (\tau > 0) \\ \text{subject to} \quad & Ax = b \end{aligned}$$

where $g_i^\top$ is the ith row of G, h_i is the ith element of h.

The *convex* function

$$\phi_\tau(x) = -\tau \sum_{i=1}^{m} \log(h_i - g_i^\top x)$$

is called the *logarithmic barrier* for the original QP problem.

The idea of this reformulation is that the function $\phi_\tau(x)$ for very small values of the parameter τ resembles an *indicator function*, which is zero for feasible x and infinity for infeasible x—it works as a 'barrier' towards entering into the infeasible set. The point in doing this is of course that the reformulated QP problem is now a problem with equality constraints only, a problem we know how to handle.

Let $\{x^*(\tau) \mid \tau > 0\}$, the *central path*, be the set of solutions to the modified optimisation problem stated above for different values of $\tau > 0$. It can then be shown that, under suitable assumptions, $x^*(\tau) \to x^*$ as $\tau \to 0$. Based on this characterization of the central path, it seems natural to choose a small enough value of τ and to solve the modified QP problem with equality constraints for this τ. It turns out, however, that a better algorithm is obtained by solving a sequence of problems with decreasing values of τ. The resulting algorithm is called the **barrier method**.

The sequence of optimisation problems within the barrier method are typically solved by applying Newton's method to the KKT conditions, as described earlier. Let's have a look at these conditions (note that we have got rid of the inequalities in the reformulated problem)

$$\begin{aligned} Qx + p + \tau \sum_{i=1}^{m} \frac{1}{h_i - g_i^\top x} g_i + A^\top \lambda = 0 \\ Ax - b = 0 \end{aligned}$$

which are valid only for *interior points*, i.e. those x satisfying $h_i - g_i^\top x > 0, \forall i$. It should come as no surprise that these equations pose numerical difficulties for Newton's method when the iterates get close to the boundary of the inequality constraints—the objective has a strong curvature here.

The barrier method has an intuitively appealing interpretation that will guide us further. By defining $\mu_i = \tau/(h_i - g_i^\top x)$, the KKT conditions for the barrier method can be rewritten as

$$\begin{aligned} Qx + p + \sum_{i=1}^{m} \mu_i g_i + A^\top \lambda = 0 \\ Ax - b = 0 \\ \mu_i (h_i - g_i^\top x) = \tau \end{aligned}$$

which, together with the conditions $h_i - g_i^\top x > 0$ and $\mu_i > 0$, can be seen as a version of the original KKT conditions (99)-(102), where the complementary slackness conditions have been *smoothed*. In principle, this set of nonlinear equations can be solved using Newton's method, complemented with backtracking to secure feasibility with respect to the two inequalities mentioned. However, a slight re-formulation using a non-negative *slack variable* s turns out to be advantageous and gives rise to the following **primal-dual interior point method**.

The QP problem

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{1}{2} x^\top Q x + p^\top x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

is characterized by the approximated (smoothed) KKT conditions

$$\begin{gathered} Qx + p + G^\top \mu + A^\top \lambda = 0 \\ Ax - b = 0 \\ Gx - h + s = 0 \\ \mu_i s_i = \tau \\ s > 0, \quad \mu > 0. \end{gathered}$$

Applying Newton's method on the equalities gives the Newton step

$$\begin{bmatrix} Q & A^\top & G^\top & 0 \\ A & 0 & 0 & 0 \\ G & 0 & 0 & I \\ 0 & 0 & \operatorname{diag}(s) & \operatorname{diag}(\mu) \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \\ \mu^+ \\ s^+ \end{bmatrix} = \begin{bmatrix} -p \\ b \\ h \\ \operatorname{diag}(s)\mu + \tau \end{bmatrix}.$$

Backtracking to secure $s > 0$ and $\mu > 0$ is simple!

9 Feasibility

Already in section 4, it was pointed out that the receding horizon control law is not defined for all states, since the associated optimisation problem may be infeasible. In this section, we will look into this aspect in some detail, and specifically discuss what is referred to as *recursive or persistent feasibility*.

The book [1] discusses feasibility and recursive feasibility in chapter 2.

From an implementation point of view, the big difference between MPC and other control schemes is the way the control action is computed: in most controllers, there is an explicit expression for the control action, but in MPC the controller is stated in terms of an optimisation problem to be solved on-line. There are several implications of this fact. Some of these are related to the implementation of the algorithms for the optimisation, since these are typically more computationally demanding than conventional controllers. It is important that the algorithm is capable of delivering the control action in a specified time, and if that is not possible, some predefined action has to be taken—these issues will be discussed below in some detail.

9.1 Feasibility

Let's stress again that the receding horizon controller is defined *implicitly* as the solution of an optimisation problem, which depends on the current state (estimate). However, it is in general not possible to a priori guarantee that the optimisation problem posed is *feasible* at every sampling instant. More specifically, in the presence of state constraints, it may very well happen that the optimisation problem is infeasible for some initial states, i.e. that no solution respecting the constraints exists. Expressed in a different way, the implicit control law $\kappa_N(x)$ is defined only for states belonging to the set of feasible states, $x \in \mathcal{X}_N$, see Section 4.1. This observation gives in turn rise to the concept of *recursive (or persistent) feasibility* to be discussed next.

Recursive (persistent) feasibility

Assume $x \in \mathcal{X}_N$ is a feasible state. This means that there is indeed a solution to the optimisation problem for that particular x. The following question is then natural to ask: after having applied the control action computed, will the successor state x^+ also belong to the feasible set $\mathcal{X}_N$? If so, the receding horizon control action would be defined also at the next sampling instant. This is clearly a desirable situation that is known as *recursive (or persistent) feasibility*. The somewhat disappointing answer is, however, that there is no general guarantee for this, even in the nominal case with a perfect model and no disturbances. We will analyse what happens in more detail, starting by defining formally recursive feasibility using the concept of *invariant sets*.

Definition 9.1 (Invariant set)**.** The set $\mathcal{S}$ is *positively invariant* for the autonomous system $x^+ = f_a(x)$ if

$$x(0) \in \mathcal{S} \quad \Rightarrow \quad x(k) \in \mathcal{S}, \quad \forall k \in \mathbb{N}_+.$$

Definition 9.2 (Recursive (persistent) feasibility)**.** The receding horizon controller is recursively (persistently) feasible if the feasible set $\mathcal{X}_N$ is positively invariant for the closed-loop system $x^+ = f(x, \kappa_N(x))$, i.e.

$$x(0) \in \mathcal{X}_N \quad \Rightarrow \quad x(k+1) = f(x(k), \kappa_N(x(k))) \in \mathcal{X}_N, \quad \forall k \in \mathbb{N}_+.$$

The implication of these definitions is that if the RHC is recursively feasible, then it is guaranteed that the optimisation problem to be solved at every time instant is feasible, provided the initial state is feasible.

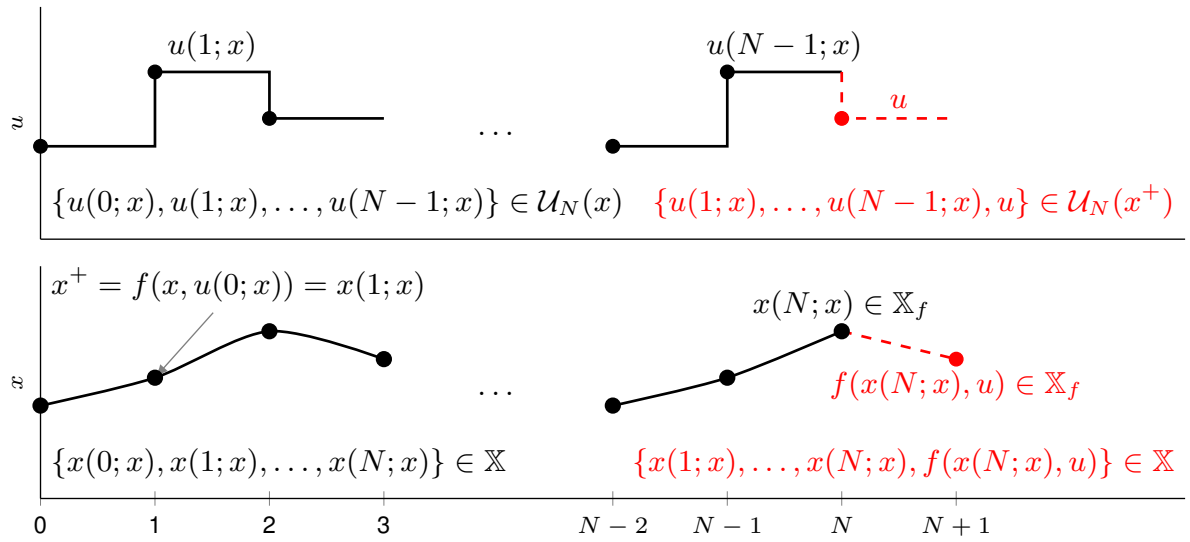


Figure 20: Illustration of recursive feasibility. After shifting the horizon and applying a control sequence constructed from the tail of the previous sequence and a new feasible control u, recursive feasibility requires that the new state sequence is also feasible.

There is one disadvantage with the way we have expressed recursive feasibility, namely the fact that the condition depends on the receding horizon control law $\kappa_N(x)$, which in turn depends on all design parameters such as cost function weights. Feasibility is, however, primarily concerned with constraints, and we therefore would like to have a condition reflecting this fact. In order to derive such a condition, we need to make a detailed analysis of successive candidate control sequences.

Assume the current state $x(k) = x$ is feasible, i.e. $x \in \mathcal{X}_N$. Let

$$u(0:N-1;x) = \{u(0;x), u(1;x), \ldots, u(N-1;x)\} \in \mathcal{U}_N(x)$$

be *any* control sequence (of the possibly many) driving the state to the terminal constraint set $\mathbb{X}_f$ in N steps, respecting control and state constraints, and

$$x(0:N;x) = \{x(0;x), x(1;x), \ldots, x(N;x)\} \in \mathbb{X}$$

be the corresponding predicted state evolution with $x(N;x) \in \mathbb{X}_f$. The question is now: can we give a condition that ensures that the next state, resulting from applying the first control action in the sequence to the plant, is feasible as well? The next state is

$$x^+ = f(x, u(0;x)) = x(1;x).$$

From this state, we can simply apply a control sequence, which is a copy of the tail of the control sequence we started with, appended with a final control u to be determined:

$$u(0:N-1;x^+) = \{u(1;x), \ldots, u(N-1;x), u\}.$$

Provided $u \in \mathbb{U}$, this control sequence satisfies the control constraints. The state sequence resulting from $u(0:N-1;x^+)$ is in an analogous way equal to the tail of $x(0:N;x)$, appended with a last element,

$$x(0:N;x^+) = \{x(1;x), \ldots, x(N;x), f(x(N;x), u)\}.$$

Figure 20 illustrates the situation described (the original sequences are with solid lines, and the appended elements with dashed lines).

Taking a closer look at the state sequence $x(0:N;x^+)$, it is clear that it belongs to $\mathbb{X}$, except possibly the last element. However, we need a stronger condition to hold, namely that $f(x(N;x), u) \in \mathbb{X}_f$. At this point, we need the following definition:

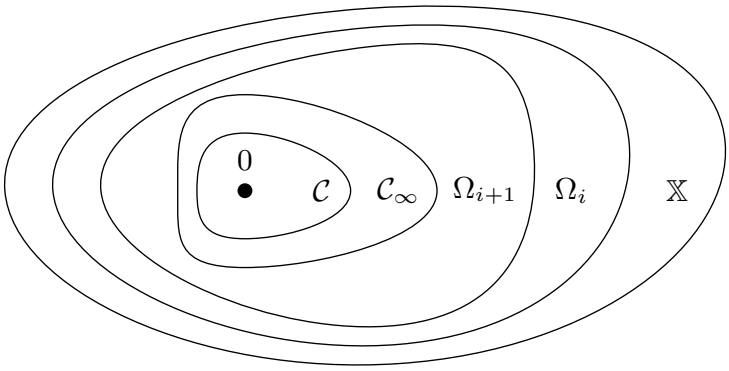


Figure 21: Construction of the maximal control invariant set $\mathcal{C}_\infty$ in $\mathbb{X}$. $\mathcal{C}$ is an arbitrary control invariant set.

Definition 9.3 (Control invariant set)**.** A set $\mathcal{C} \subseteq \mathbb{X}$ is a *control invariant set* of the system $x^+ = f(x,u)$ if

$$x \in \mathcal{C} \quad \Rightarrow \quad \exists u \in \mathbb{U} \text{ such that } x^+ = f(x,u) \in \mathcal{C}.$$

The *maximal control invariant set* contained in $\mathbb{X}$ is denoted $\mathcal{C}_\infty$ and contains all control invariant sets in $\mathbb{X}$.

This is exactly what we need: we know that $x(N;x) \in \mathbb{X}_f$; if $\mathbb{X}_f$ is control invariant, then $u \in \mathbb{U}$ can be chosen such that $f(x(N;x),u) \in \mathbb{X}_f$. Hence, x^+ is feasible. Since this argument can be applied repeatedly, the following result can been derived.

Theorem 9.4 (Sufficient condition for recursive feasibility)**.** *The receding horizon controller based on the finite horizon optimal control problem* (37)-(40) *is recursively feasible if the terminal constraint set $\mathbb{X}_f$ is control invariant.*

A natural question at this point is how to construct the terminal constraint set so that it becomes control invariant. A trivial choice is the singleton $\mathbb{X}_f = \{0\}$, which has been suggested in the literature. However, this is a quite strict constraint, which could be difficult to meet, and therefore less strict alternatives are of interest.

At the other extreme, $\mathbb{X}_f$ could be chosen as $\mathcal{C}_\infty$, the maximal control invariant set in $\mathbb{X}$. The following recursion can be used to find $\mathcal{C}_\infty$:

$$\begin{aligned} \Omega_0 &= \mathbb{X} \\ \Omega_{i+1} &= \text{Pre}\,(\Omega_i) \cap \Omega_i. \end{aligned} \tag{104}$$

Here, $\text{Pre}\,(\mathcal{S})$ is the set of states that can be driven into the target set $\mathcal{S}$ in one time step:

$$\text{Pre}\,(\mathcal{S}) = \{x \in \mathbb{X} \mid \exists u \in \mathbb{U} \text{ such that } x^+ = f(x,u) \in \mathcal{S}\}. \tag{105}$$

The recursion (104), which generates a sequence of decreasing sets, may or may not terminate; if it does, $\Omega_{i+1} = \Omega_i$ for some i, the *determinedness index* of $\mathcal{C}_\infty$, which is in this case *finitely determined*. Figure 21 illustrates the different sets we have defined.

How big is the feasible set?

It was mentioned when the feasible set $\mathcal{X}_N$ was introduced that it can sometimes be empty. In our case, it follows from the assumptions that $f(0,0) = 0$ and that $\mathbb{U}$, $\mathbb{X}$ and $\mathbb{X}_f$ all contain the origin that $\mathcal{X}_N$ always contains the origin, so it is actually non-empty. We would not be satisfied with a feasible set consisting of only the origin, however, so the question is if we can say anything more about the size of $\mathcal{X}_N$?

In order to get some insight, it is useful to introduce the following concept.

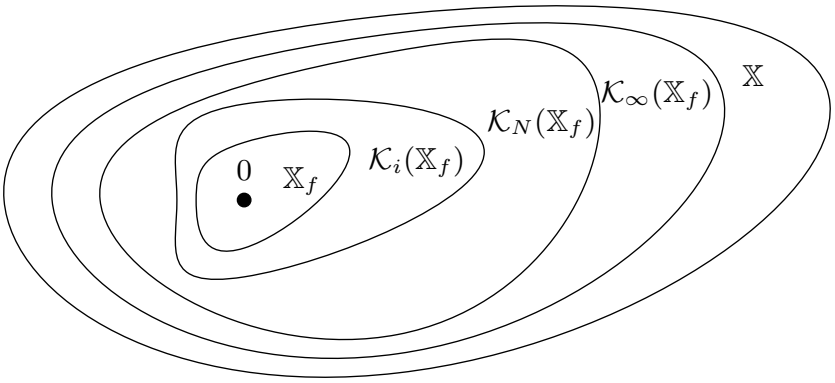


Figure 22: Construction of the maximal controllable set $\mathcal{K}_\infty(\mathbb{X}_f)$.

Definition 9.5 (i-step controllable set $\mathcal{K}_i(\mathcal{S})$)**.** The *$i$-step controllable set* $\mathcal{K}_i(\mathcal{S})$ is defined as the set of initial states that can be driven to the target set $\mathcal{S}$ in i steps, while satisfying state and control constraints at all times.

From the definition, it follows that the feasible set $\mathcal{X}_N$ can alternatively be defined as

$$\mathcal{X}_N = \mathcal{K}_N(\mathbb{X}_f).$$

If it is assumed that $\mathbb{X}_f$ is control invariant—which was shown above to guarantee persistent feasibility—then it is clear that the set sequence $\{\mathcal{K}_i(\mathbb{X}_f)\}$ is monotone in the sense

$$\mathbb{X}_f = \mathcal{K}_0(\mathbb{X}_f) \subseteq \ldots \subseteq \mathcal{K}_i(\mathbb{X}_f) \subseteq \mathcal{K}_{i+1}(\mathbb{X}_f) \subseteq \ldots \subseteq \mathcal{K}_N(\mathbb{X}_f) \subseteq \mathcal{K}_\infty(\mathbb{X}_f) \subseteq \mathbb{X}, \tag{106}$$

where $\mathcal{K}_\infty(\mathbb{X}_f)$ is the *maximal controllable set* with target set $\mathbb{X}_f$, defined by

$$\mathcal{K}_\infty(\mathbb{X}_f) = \bigcup_i \mathcal{K}_i(\mathbb{X}_f).$$

Figure 22 illustrates what has just been said. From (106) it follows that one way to increase the feasible set is to use a large prediction horizon N, which is intuitively reasonable. In some cases, there is a finite $\bar{N}$, the *determinedness index* for $\mathcal{K}_\infty(\mathbb{X}_f)$, for which $\mathcal{K}_{\bar{N}}(\mathbb{X}_f) = \mathcal{K}_\infty(\mathbb{X}_f)$. In these cases, it does not pay off (in terms of the size of the feasible set) to increase N beyond $\bar{N}$.

In addition to increasing the prediction horizon, the size of $\mathcal{X}_N = \mathcal{K}_N(\mathbb{X}_f)$ can be increased by choosing $\mathbb{X}_f$ large, possibly as large as possible, i.e. $\mathbb{X}_f = \mathcal{C}_\infty$. This follows from the property

$$\mathbb{X}_f^1 \subset \mathbb{X}_f^2 \quad \Rightarrow \quad \mathcal{K}_N(\mathbb{X}_f^1) \subset \mathcal{K}_N(\mathbb{X}_f^2).$$

Summing it all up, the conclusion is that in order to have a large feasible set, it is advantageous to use a large terminal constraint set $\mathbb{X}_f$ and a large prediction horizon N. However, as we will se later in Chapter 10, there are other considerations that may point in a different direction.

Practical feasibility

So far we have discussed feasibility in the *nominal case* with a perfect model and no disturbances. The concept of recursive feasibility points to the importance of choosing the terminal state constraint in a proper way. However, even if do this, we should keep in mind that the optimisation problem to be solved at each sampling instant depends on the current state estimate. In more realistic scenarios, i.e. in the presence of disturbances and model uncertainties, the state estimate may be affected in such a way that infeasibility appears. In such a case, it is essential that the MPC controller is equipped with some type of 'exception handling'. One possibility is to resort to some 'ad-hoc' solutions, e.g. use the control action from the previous sampling instant, or to switch in a fall-back regulator. A better alternative is to handle the constraints, which are the sources for possible difficulties, in a smarter way by *constraint management*.

Constraint management

There are many ways to handle the constraints to avoid running into infeasibility. The most obvious one is probably to avoid making the constraints *hard*; by *softening* the constraints, the MPC algorithm is given the opportunity to violate one or several constraints, *if necessary*, to solve the optimisation problem and deliver a control signal.

- Soft constraints can be introduced:

$$\begin{aligned} \min_{u} \quad & V_N(\boldsymbol{u}) + ro\|\varepsilon\| && (107)\\ \text{subject to} \quad & F\boldsymbol{u} + G\boldsymbol{x} \leq e + \varepsilon && (108)\\ & \varepsilon \geq 0. && (109) \end{aligned}$$

- Reduce window in which constraints are enforced.
- Prioritise constraints $\rightarrow$ mixed-integer quadratic program.
- Don't forget time limitations – control output *must* be delivered!

Let's summarize the main points that have been raised concerning feasibility:

- It may be impossible, for some initial states, to solve the RHC optimisation problem while respecting constraints on states and/or outputs. The problem is in these cases *infeasible*.
- In the nominal case, without disturbances and with a perfect model, *recursive feasibility* may be ensured by choosing the terminal constraint set $\mathbb{X}_f$ to be control invariant.
- In practice, infeasibility may still occur, possibly caused by too strict performance requirements, a large disturbance, model uncertainty, or by an unstable system.
- Ad hoc solutions are for example to keep the control as is (from the previous sampling instant), or to switch to a backup controller.
- A more systematic approach is to relax the constraints in some way; this is often referred to as *constraint management*.

10 Stability

The objective of this section is to investigate in some more detail the theoretical aspects of receding horizon control, and we will be particularly interested in the stability properties of the closed-loop system. This is a difficult problem, but we will at least be able to introduce a few key ideas that are frequently met in the scientific literature. In addition, we will formulate a couple of stability results that hold for the class of MPC algorithms focussed in the course.

The treatment here is influenced by Section 2.4 in [1]. Stability is also treated in [2] in Sections 4.4-4.5 (general case) and in Sections 5.6 (constrained LQ). Part of the material is covered by Sections 6.1-6.2 in [4], including mini-tutorial 6 on p170.

Let us recall from the first few sections of the course that the receding horizon control idea (and therefore MPC) emerged as a way to mimic the (in general unattainable) DP solution, but at the price of introducing approximations. Hence, we can no longer guarantee closed-loop stability—in fact, we have not even discussed the issue. But since feedback is involved, there is always the inherent risk of instability; indeed, we noted in Example 3.2 that the finite-time LQ controller can give rise to an unstable closed-loop system. The question arises: what can be said about stability for MPC controllers, and what type of insights can be gained from a theoretical analysis? We will provide some partial answers to these questions in this section.

The stability analysis will be carried out for the basic MPC formulation in Section 4, i.e. it is assumed that we have access to the system states and that there are no uncertainties. We will investigate the simple regulation case, where the intent is to steer the system state to the origin. Hence, the task is essentially to investigate the stability properties of the closed-loop system

$$x^+ = f(x, \kappa_N(x)),$$

where $\kappa_N(x) = u^*(0; x)$ is the implicit receding horizon control law. Before embarking on this task, we need to establish a few basic definitions and tools for the analysis.

10.1 Stability and Lyapunov functions

Consider the discrete-time system

$$x^+ = f(x),$$

with $f(0) = 0$, so that the origin is an equilibrium of the state equation. We will be interested in two kinds of stability properties for the equilibrium at the origin. The origin is *locally stable* if solutions starting close to the origin remain there and *(globally) asymptotically stable* if in addition the solutions asymptotically approach the origin (*globally* indicates this holds for any initial state). The formal definitions are as follows. Consider the system

$$x^+ = f(x), \quad f(0) = 0. \tag{110}$$

Definition 10.1 (Local stability)**.** The origin is *locally stable* for the system (110) if, for any $\epsilon > 0$, there exists a $\delta > 0$ such that $|x(0)| < \delta$ implies $|x(k)| < \epsilon$ for all $k \geq 0$.

Definition 10.2 (Global attraction)**.** The origin is *globally attractive* for the system (110) if $x(k) \to 0,\ k \to \infty$ for any initial $x(0)$.

Definition 10.3 (Global asymptotic stability)**.** The origin is *globally asymptotically stable* for the system (110) if it is locally stable and globally attractive.

Remark. *If the origin is attractive only for initial states within a certain set, the definitions can be modified and the properties then hold with a* region of attraction *that is not any longer the entire* $\mathbb{R}^n$.

The basic tool for establishing stability of nonlinear systems is Lyapunov stability theory. This is a topic pursued in detail in courses on nonlinear dynamical systems; here it will suffice to state a basic result, formulating sufficient conditions for global asymptotic stability using a *Lyapunov function*.

Definition 10.4 (Lyapunov function)**.** A function $V : \mathbb{R}^n \to \mathbb{R}_+$ is said to be a Lyapunov function for the system (110) if there are functions $\alpha_i \in \mathcal{K}_\infty, i = 1, 2$ and a positive definite function α_3 such that for any $x \in \mathbb{R}^n$,

$$\begin{aligned} V(x) &\geq \alpha_1(|x|) \\ V(x) &\leq \alpha_2(|x|) \\ V(f(x)) - V(x) &\leq -\alpha_3(|x|). \end{aligned}$$

Remark. *A function belongs to $\mathcal{K}_\infty$ if it is nonnegative, continuous, zero at zero, strictly increasing and unbounded. A positive definite function is continuous and positive everywhere except at the origin.*

What is interesting with the Lyapunov function is that it gives information about the evolution of the state trajectory (the solution of the system equations) without the need to solve for this explicitly. The function V can be thought of as a generalization of energy stored in the system, and stability is closely connected with the decay of this energy with time. Indeed, the third condition in the definition above tells that V decays along solutions of the system, and the fact that α_3 is a positive definite function indicates that the decay persists until the state x approaches the origin. The first and the second conditions can be seen as technical conditions to rule out 'strange' cases. The formal statement is given by the following proposition, given without proof.

Proposition 10.1 (Lyapunov's theorem)**.** *Suppose $V(\cdot)$ is a Lyapunov function for the system*

$$x^+ = f(x), \quad f(0) = 0.$$

Then the origin is globally asymptotically stable.

Example 10.5 (Lyapunov function for linear system)**.** A very common Lyapunov function is the quadratic form $V = x^\top Sx$ with S positive definite, which fulfills the first two conditions for being a Lyapunov function. In addition, for the linear system $x^+ = Ax$ with the matrix A having its eigenvalues strictly inside the unit circle, Lyapunov's equation (14) implies that

$$V(x^+) = (Ax)^\top S(Ax) = x^\top A^\top SAx = x^\top Sx - x^\top Qx = V(x) - x^\top Qx$$

so that the third property is also satisfied. Global asymptotic stability thus holds according to Proposition 10.1. ■

Example 10.6 (Stability for the infinite horizon LQ controller)**.** It was stated without proof in Section 3.1 that the infinite horizon LQ controller gives a stable closed loop under appropriate conditions. This can again be shown using a quadratic Lyapunov function $x^\top Px$, where P is the solution of the algebraic Riccati equation. Note that in this case, the Lyapunov function has the important interpretation as the optimal cost-to-go or value function, i.e. $V_\infty^*(x) = x^\top Px$. This observation will soon become very relevant for us. ■

10.2 Stability conditions for MPC

For easy reference, let's repeat the basic equations that define the receding horizon controller of interest; we will use the general notation of Section 4 and later specialize to the constrained LQ case that is our prime goal.

Cost function:

$$V_N(x_0, u(0\!:\!N-1)) = \sum_{i=0}^{N-1} l(x(i), u(i)) + V_f(x(N)). \tag{111}$$

Optimal cost-to-go:

$$V_N^*(x_0) = \min_{u(0:N-1)} \{V_N(x_0, u(0:N-1)) \mid u(0:N-1) \in \mathcal{U}_N(x_0)\}.$$

Constraints:

$$x^+ = f(x,u), \quad x(0) = x_0 \tag{112}$$
$$x(k) \in \mathbb{X}, \quad u(k) \in \mathbb{U}, \quad \text{for all } k \in (0,N) \tag{113}$$
$$x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. \tag{114}$$

Feasible control sequences and initial states:

$$u(0:N-1) \in \mathcal{U}_N(x_0) \tag{115}$$
$$\mathcal{X}_N = \{x_0 \in \mathbb{X} \mid \mathcal{U}_N(x_0) \neq \emptyset\}. \tag{116}$$

Optimal control and state sequences:

$$\begin{aligned} u^*(0:N-1;x_0) &= \{u^*(0;x_0), u^*(1;x_0), \ldots, u^*(N-1;x_0)\} \\ x^*(0:N;x_0) &= \{x^*(0;x_0), x^*(1;x_0), \ldots, x^*(N;x_0)\}. \end{aligned} \tag{117}$$

It will be assumed in the sequel that the functions $f(\cdot)$, $l(\cdot)$ and $V_f(\cdot)$ are continuous, $f(0,0) = 0$, $l(0,0) = 0$ and $V_f(0) = 0$. Further, the sets $\mathbb{X}$ and $\mathbb{X}_f$ are closed, and $\mathbb{U}$ is closed and bounded, and all sets contain the origin. These assumptions guarantee that the problem we study is meaningful and "nice".

Proposition 10.2 (Existence of solution)**.** *With the assumptions above, the following holds:*

(a) The function V_N is continuous on $\mathcal{X}_N \times \mathcal{U}_N$.

(b) For each $x \in \mathcal{X}_N$, the control constraint set $\mathcal{U}_N(x)$ is closed and bounded.

(c) For each $x \in \mathcal{X}_N$, a solution to the optimal control problem exists.

Remark. *Note that nothing is said about the properties of the* value function $V_N^*(x)$ *or the control law $\kappa_N(x) = u^*(0;x)$. In fact, these may both be discontinuous. However, if there are no state constraints (i.e. $\mathbb{X} = \mathbb{X}_f = \mathbb{R}^n$) or if the system is linear and the constraint sets are all polyhedral, then the value function is continuous.*

One of the approximations that was involved when moving from an intractable DP solution to MPC was the shift from an infinite time horizon to a finite one. The following example ties together these two cases for the unconstrained LQ problem, and at the same time gives a hint how to proceed.

Example 10.7 (Equivalence between infinite and finite horizon LQ)**.** The infinite horizon LQ problem is based on the criterion in equation (29),

$$V(x(0), u(0:\infty)) = \sum_{i=0}^{\infty} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right).$$

If the sum is split into two terms we get

$$V(x(0), u(0:\infty)) = \sum_{i=0}^{N-1} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right) + \sum_{i=N}^{\infty} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right).$$

Now, from the dynamic programming discussions we know that one way to minimize this criterion is to start by minimizing the tail, i.e. the second term with respect to the controls from time N to infinity. But this is again an infinite horizon LQ problem, this time starting at time N with initial state $x(N)$, and we know that the solution will give an optimal cost $x^\top(N)Px(N)$ where P is the solution to the algebraic Riccati equation, see (31). Hence, by minimizing the *finite time* horizon criterion

$$V(x(0), u(0:N-1)) = \sum_{i=0}^{N-1} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right) + x^\top(N)Px(N),$$

we will in fact generate exactly the same minimizing control sequence $u^*(0:N-1)$ as the infinite horizon formulation leads to. The implication is that the finite time solution inherits some of the nice properties of the infinite time solution, e.g. stability. The lesson to learn is that the terminal cost plays an important role when analysing the stability properties of MPC. ■

We proceed with the following basic assumption:

Assumption 10.3 (Basic stability assumption)**.** *The terminal set $\mathbb{X}_f$ is control invariant and the following inequality holds:*

$$\min_{u\in\mathbb{U}} \{V_f(f(x,u)) + l(x,u) \mid f(x,u) \in \mathbb{X}_f\} \leq V_f(x), \quad \forall x \in \mathbb{X}_f.$$

Remark. *Recall that control invariance of $\mathbb{X}_f$ means that there exists a $u \in \mathbb{U}$ such that $f(x,u) \in \mathbb{X}_f$, see Definition 9.3. The implication of this, together with the properties of $\mathbb{U}$, is that the minimum in the assumption above exists.*

We are now ready to establish our basic stability results for receding horizon control. The idea is again to have a look at the unconstrained LQ case. It was seen above that the value function for the infinite horizon LQ problem could be used as a Lyapunov function to prove stability of the closed-loop system. We will use a similar argument for the MPC case, but we will have to do with the finite horizon value function, which has not as ideal properties. The following lemma establishes the important decay property of the value function, which will later be used as a Lyapunov function.

Lemma 10.8 (Value function decrease)**.** *Assume that Assumption 10.3 holds. Then the optimal cost or value function fulfills the following inequality for all $x \in \mathcal{X}_N$:*

$$V_N^*(f(x, \kappa_N(x))) \leq V_N^*(x) - l(x, \kappa_N(x)).$$

Proof. Let x be any point in $\mathcal{X}_N$ with $V_N^*(x) = V_N(x, u^*(0:N-1;x))$. The corresponding optimal control and state sequences are as in equation (117):

$$u^*(0:N-1;x) = \{u^*(0;x), u^*(1;x), \ldots, u^*(N-1;x)\}$$
$$x^*(0:N;x) = \{x^*(0;x), x^*(1;x), \ldots, x^*(N;x)\},$$

where $u^*(0;x) = \kappa_N(x)$, $x^*(0;x) = x$ and the successor state is $x^+ = x^*(1;x) = f(x, \kappa_N(x))$. In order to compare $V_N^*(x)$ with $V_N^*(x^+)$, we note that

$$V_N^*(x^+) \leq V_N(x^+, \tilde{u}(0:N-1)) \tag{118}$$

for any feasible control sequence $\tilde{u}(0:N-1)$ for the optimal control problem starting at the successor state x^+. We will use a suboptimal control sequence, constructed in the same way as we did when discussing recursive feasibility, see Figure 20. The construction is to simply copy the tail from the previous step and append a final control u that assures that $f(x^*(N;x), u) \in \mathbb{X}_f$, which is possible due to Assumption 10.3. The resulting feasible control sequence is

$$\tilde{u}(0:N-1) = \{u^*(1;x), \ldots, u^*(N-1;x), u\}$$

and the state sequence resulting from $\tilde{u}(0:N-1)$ is

$$\tilde{x} = \{x^*(1;x), \ldots, x^*(N;x), f(x^*(N;x),u)\}.$$

Having assured the feasibility of the chosen control sequence, we can proceed to compare $V_N(x^+, \tilde{u}(0:N-1))$ with $V_N^*(x)$, by noting that most of the terms in the expressions for these are identical, because of the choice of $\tilde{u}(0:N-1)$. We have

$$\begin{aligned} V_N(x^+, \tilde{u}(0:N-1)) = V_N^*(x) - l(x,\kappa_N(x)) - V_f(x^*(N;x)) \\ + l(x^*(N;x),u) + V_f(f(x^*(N;x),u)) \leq V_N^*(x) - l(x,\kappa_N(x)) \end{aligned}$$

where the inequality follows from the fact that u can be chosen according to Assumption 10.3. Combining this result with equation (118) completes the proof. □

Remark. *A special case of Lemma 10.8 is presented in Theorem 6.1 in [4]. Here, the set $\mathbb{X}_f$ is the origin, i.e. a single point, and in this case Assumption 10.3 is no longer needed. The final control u in the proof is $u = 0$, which guarantees that the state remains in the origin, once there (since $f(0,0) = 0$), and the property $l(0,0) = 0$ is also used.*

Using the decay property of the value function, proved in the lemma, we can now employ Lyapunov's stability theorem to establish stability of the MPC scheme. The only thing that needs to be added are conditions that ensure upper and lower bounds on the Lyapunov function. The proof is omitted.

Theorem 10.9 (MPC stability)**.** *Assume that Assumption 10.3 holds and that the stage cost $l(\cdot)$ and the terminal cost $V_f(\cdot)$ satisfy*

$$\begin{aligned} l(x,u) &\geq \alpha_1(|x|) \quad \forall x \in \mathcal{X}_N, u \in \mathbb{U} \\ V_f(x) &\leq \alpha_2(|x|) \quad \forall x \in \mathbb{X}_f \end{aligned}$$

with $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$. Further, assume that $\mathbb{X}_f$ contains the origin in its interior. Then the origin is asymptotically stable with a region of attraction $\mathcal{X}_N$ for the system $x^+ = f(x, \kappa_N(x))$.

10.3 Stability of constrained linear quadratic MPC

We will now take a closer look at the case of particular interest to us, the MPC based on LQ control with linear constraints, described in Section 4. The system is now described by the state equation

$$x^+ = Ax + Bu$$

with (A, B) controllable and the stage cost is

$$l(x,u) = x^\top Q x + u^\top R u$$

with $Q, R \succ 0$. As before, the constraint sets $\mathbb{X}$ and $\mathbb{U}$ are polyhedral, i.e. described by linear inequalities. What remains is to determine V_f and $\mathbb{X}_f$ to fulfill the condition in Assumption 10.3. The idea is as follows: when the state gets close enough to the origin, then the constraints will no longer be active. Hence, the constrained LQ control becomes identical to the *unconstrained* LQ, and we can rely on the nice properties of the infinite horizon LQ controller.

Choose the terminal cost as the value function for the *unconstrained* LQ problem, i.e.

$$V_f(x) = V_\infty^{\text{uc}}(x) = x^\top P x,$$

where P is the solution of the algebraic Riccati equation. The value function satisfies the equation

$$V_\infty^{\text{uc}}(x) = \min_u \{x^\top Q x + u^\top R u + V_\infty^{\text{uc}}(x^+)\} = x^\top Q x + (Kx)^\top R(Kx) + V_\infty^{\text{uc}}(Ax + BKx)$$

which implies that

$$V_f((A + BK)x) + x^\top Q x + (Kx)^\top R(Kx) = V_f(x).$$

This means that the chosen V_f satisfies the inequality of Assumption 10.3, *if* we can ensure that constraints are not active in $\mathbb{X}_f$. This can indeed be guaranteed if we define $\mathbb{X}_f \subseteq \mathbb{X}$ to be the largest set fulfilling the following two conditions: (a) $x \in \mathbb{X}_f \Rightarrow Kx \in \mathbb{U}$ and (b) $x \in \mathbb{X}_f \Rightarrow (A+BK)^i x \in \mathbb{X}_f$ for all $i \geq 0$. The set $\mathbb{X}_f$ thus defined is control invariant. We summarize the conclusions in the following theorem:

Theorem 10.10 (Stability of constrained linear quadratic MPC)**.** *Consider the linear quadratic MPC with linear constraints applied to the controllable system $x^+ = Ax + Bu$ and with positive definite matrices Q and R. Further assume that the terminal cost V_f is chosen as the value function of the corresponding unconstrained, infinite horizon LQ controller, and that the terminal constraint set $\mathbb{X}_f$ is chosen as described above. Then the origin is asymptotically stable with a region of attraction $\mathcal{X}_N$ for the controlled system $x^+ = Ax + B\kappa_N(x)$.*

The stability analysis presented above is indicative of the type of analysis techniques used and results that can be obtained. However, it is clear that the assumptions we have made are quite restrictive, for example that the states are all available for measurement, and that there are no disturbances or modelling errors. Some of the assumptions can be removed at the expense of more complicated analysis. One example of this is that if the state is not completely measurable, then additional assumptions on detectability or observability are needed. For the case with model uncertainty and/or disturbances, the stability problem is still under investigation in the scientific literature.

11 MPC practice – implementation and tuning

This section discusses some of the issues that have to be dealt with when implementing and using MPC for practical applications. The literature is relatively scarce on these topics. It is likely that much practical experience has been built up around the commercial implementations, but this is not always accessible to a wider audience. We will touch on two aspects in particular: computational issues and tuning of MPC algorithms.

The book [1] contains little of this material. A similar comment applies to [2], but a few case studies are included to give some insights from applications. Chapter 7 in [4] discusses tuning aspects, whereas implementation details are scattered in the book, e.g. computational efficiency in Section 3.3 and constraint management in Section 10.2.

Having come this far in the development of model predictive controllers based on linear models, quadratic costs and affine constraints, it is time to discuss what remains to make the RHC design a viable and attractive control design alternative in applications. Let's start by reminding about the structure of the MPC controller developed so far, repeated here from Section 5 and illustrated in Figure 23.

Each of the blocks shown in this diagram has some key characteristics that show similarities:

The state observer provides state estimates in cases where the state is not fully accessible from measurements. As always when feedback is employed, it is important that a sufficient number of sensors/measurements is available, and we have seen that this is particularly important when it is desirable to estimate and reject *load disturbances*.

The state observer is often implemented as a *Kalman filter (KF)*, but we have seen that by instead implementing a *moving horizon estimator (MHE)*, there is a possibility to include constraints that represent a priori information about realistic ranges of state variables. The disadvantage is that computations become more demanding.

Regardless of the choice of state observer, there are a number of design parameters, which affect the observer dynamics and the trade-off between tracking ability and noise rejection. Examples of such parameters are process and measurement noise covariances (KF), and uncertainty weights and estimation horizon (MHE), respectively.

The target selector (TS) basically provides references for state and control variables, indicating to which values it is desirable to drive the closed-loop system. An important part of the target selector is to take into account estimated load disturbances, in effect adding integral action into the loop. Depending on the particular structure of the model, and the presence of constraints, the computations

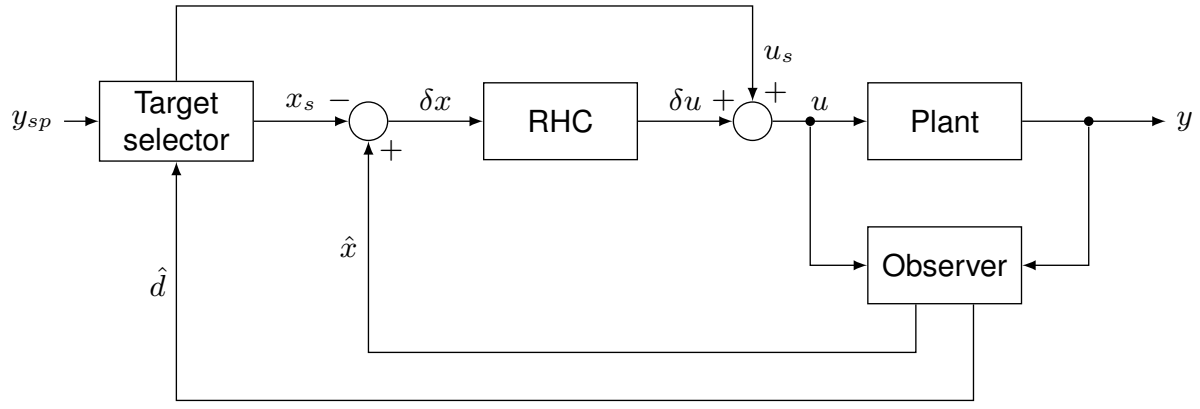


Figure 23: MPC block diagram with a receding horizon controller (RHC) working on deviation variables.

amount to solving systems of linear equations or to solve a quadratic program.

The core RHC block contains the repeated solution of a constrained finite-horizon optimisation problem that can be seen as an LQ problem, extended with affine constraints on controls and states. The basic design parameters remain the same as for LQ, e.g. weights on control and states in the cost function, and they share the same challenge to relate the tuning of the parameters to expected behaviour of the closed-loop system. In addition, the MPC brings in new design parameters, which affect the control performance *indirectly* such as horizons and constraints. The latter have implications not only for control performance, but also for feasibility and computations.

11.1 Design and tuning

When applying MPC, you are immediately faced with the task to determine or adjust a number of *design parameters*. Several of these are not new—they appear also in e.g. LQ or LQG design—but there are also some adjustable parameters that are specific for MPC. There are few strict rules for how to set these parameters; instead, you have to rely on a mix of rules-of-thumb, simulation investigations, and experience gained from practical applications. Here are a few of the choices that need to be made.

- Cost function weights:
  - which states and outputs are most important? which are the targets/setpoints?
  - penalty within system delay H_w is not meaningful
  - penalty on control or control moves?
  - only relations between Q and R important
- Prediction and control horizons:
  - important that predicted trajectories resemble what is obtained in closed-loop
  - general rule: choose M/N as large as possible (trade-off between stability and performance vs. computations)
  - rule-of-thumb: it is desirable that N covers the settling time and M covers at least the transient, but the routine choice $M = N$ is also common
- Some (very) special cases:
  - 'Mean-level control': $M = 1$, N large, $R = 0$ and r constant.
    One control action, steady-state important; slow response (roughly as open-loop).
  - 'Dead-beat control': $M = H_w = n$, $N \geq 2n$, $R = 0$, r constant.
    Enough control actions to settle the states before the output is observed.
  - 'Myopic control': $M = N = 1$, $R = 0$.
    Resembles inverting the plant dynamics (minimum phase!)
- Constraints:
  - constraints on u and/or Δu, compatible with physical constraints!
  - tighter constraints on u or Δu may be used to achieve smoother control (warning: unstable plant)
  - constraints on states and/or outputs is at the core of MPC, but too strict constraints may be impossible to fulfil!

- Disturbance models and observers:
  - standard: step disturbances and off-set free control
  - disturbance feedforward sometimes possible
  - stochastic disturbances: modelling $\rightarrow$ estimator/observer design
  - rule-of-thumb: observer dynamics should be faster than closed-loop settling time
- Reference trajectories and steady-state targets:
  - reference trajectories may be used to shape the response
  - piecewise constant setpoints common in e.g. process control
  - targets/setpoints are important, but some outputs may have no targets
  - more careful design to avoid hitting constraints ('reference governor')
  - look-ahead of references/set-points sometimes possible (e.g. batch processes)!

11.2 Implementation issues and computational efficiency

As pointed out in Section 9, there is a big difference between MPC and other control schemes, which has consequences for the implementation, namely that the MPC control "law" is defined *implicitly* in terms of an optimisation problem to be solved on-line. One implication of this is the issue of feasibility. There are other issues related to the implementation of the algorithms for the optimisation, since these are typically more computationally demanding than conventional controllers. It is important that the algorithm is capable of delivering the control action in a specified time, and if that is not possible, some predefined action has to be taken.

Whether the reason for a control signal being not ready for "delivery" in time is infeasibility or unexpectedly long computation time, a strategy needs to be chosen and implemented. We have already briefly touched on a couple of alternatives: one may be to switch in a backup controller, often a more basic control design using e.g. SISO PID control; another may be to temporarily freeze the control signal and wait for the next sampling instant. The latter is of course not a long-term solution, in particular if the open-loop system is unstable! In practice, the choice depends on the details of the application.

Computational efficiency was not a prime issue in the first applications of MPC in the 1970ies, simply because the processes involved were typically operating at a very slow time scale. However, as MPC has attracted increased interest in recent years, computational power and efficient numerical methods have become instrumental in approaching also very fast and time-critical applications. Model predictive control has of course benefited from general advances in e.g. numerical linear algebra and convex optimisation solvers. As important as having algorithms that solve the optimisation problems efficiently *on average* is to have a small spread in computation time, so that the algorithm finishes within the allocated time. There are some fine details that can contribute to meet the requirements in this regard:

- Problem size depends on n, M, and N, and number of constraints.
- Choice of optimisation algorithm matters (active set, interior point etc.)
- Matrix sparsity, representations and ordering of variables (condensed, partially condensed, non-condensed etc.).
- *Blocking*, i.e. requiring a piecewise constant u, gives fewer decision variables.

- Using the results from the previous sampling instant as initial guess may speed up the optimisation.
- Tailor-made code is typically much faster than Matlab.

In the quest for an efficient implementation of the MPC algorithm, there are several options to approach the problem, if you want to avoid coding the optimisation algorithm at the lowest level yourself. One is to go for a standard solver, available in source code form, and then to integrate this with your own MPC code. Another possibility is to use a code generation tool to have your solver being tailored to your specific problem. Several services that have emerged over the last few years have made the latter alternative a viable option; among these are the following (the list is continuously growing and it is by no means claimed to be complete):

- CasADi (https://web.casadi.org/)
- FORCES Pro (https://www.embotech.com/FORCES-Pro)
- ECOS (https://github.com/embotech/ecos)
- CVXGEN (http://cvxgen.com/docs/index.html)
- ACADO (http://acado.github.io/index.html)
- ACADOS (https://github.com/acados/acados)
- FiOrdOs (http://fiordos.ethz.ch/dokuwiki/)
- qpOASES https://github.com/coin-or/qpOASES
- OSQP https://osqp.org/
- HPIPM https://github.com/giaf/hpipm

Solving systems of linear equations

We have seen in the sections on optimisation, in particular the QP case, that solving the KKT equations is typically done by applying Newton's method, which in turn involves repeated solutions of systems of linear equations. This is important—the reason is that there are very efficient computational techniques for doing this, even for very large problems, and in particular when there is some structure in the matrices that can be exploited. Here, we will only scratch on the surface of this large area of computational linear algebra.

Consider the problem to solve the system of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}.$$

The cost (i.e. execution time) for solving this equation is generally of the order n^3 (measured in *flops* (floating point operations), which is in itself a very crude method for today's computers). As an example of what this can mean, let's assume it takes, say, 1 ms to solve a system with 100 variables, then it would take approximately 1 s to solve a problem with 1000 variables. If the matrix A is lower triangular, the situation changes drastically: by using *forward substitution* the variables can then be solved easily one by one, starting with x_1. The cost of doing this is only order n^2. The same holds for an upper triangular A, where the variables are solved in reverse order, so called *backward substitution*. Other special cases that are easy to solve involve diagonal or orthogonal ($A^{-1} = A^\top$) matrices.

The most common technique for solving $Ax = b$ is by first factorizing A as

$$A = A_1 A_2 \cdots A_k$$

and then to solve k simpler problems:

$$A_1 x_1 = b \quad A_2 x_2 = x_1 \quad \dots \quad A_{k-1} x_{k-1} = x_{k-2} \quad A_k x = x_{k-1}. \tag{119}$$

The basic idea behind this is that the factorization gives factors $A_1, \dots, A_k$ with a structure that permit fast solutions of the subproblems. This also means that the main computational burden is typically in performing the factorization, and it is in this step that exploitation of structure can be of great importance.

There are several ways to perform a factorization. An *LU factorization* factors any nonsingular matrix A as

$$A = LU$$

where L is lower triangular and U is upper triangular. A special case is for A being positive definite, in which case the factorization

$$A = LL^\top$$

is called a *Cholesky factorization*. In both these cases, the factorization can exploit structure by adding permutation matrices P_1 and P_2, e.g.

$$A = P_1 L U P_2,$$

which could lead to sparser factors and reduced computational cost. With these factorizations, the subproblems in (119) can be solved using forward and backward substitutions.

There are many other situations where the problem structure can be exploited. One case is when the structure allows a transformation into a *block triangular* form, in effect splitting a large size system of equations into two or more smaller ones. We have already encountered such an example when transforming the KKT conditions for the QP problem with equality constraints to the equivalent form in (95). In doing this, we transformed a system of linear equations of size $n + p$ into two subsystems (corresponding to an upper block triangular matrix) of size n and p, respectively. Another case where structure can be exploited is described in the next example.

Example 11.1 (Diagonal plus low rank)**.** Consider the system of equations

$$(D + BC)x = b$$

where D is diagonal, $B \in \mathbb{R}^{n \times p}$ and $C \in \mathbb{R}^{p \times n}$, and we assume p is small compared to n. Since there is no particular structure left if we form the matrix $A = D + BC$ and solve, the cost will in general be of order n^3. However, we can do better by rewriting the equation as

$$\begin{bmatrix} D & B \\ C & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

and then apply block elimination as we did for (94), giving

$$\begin{bmatrix} D & B \\ 0 & -I - CD^{-1}B \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ -CD^{-1}b \end{bmatrix}.$$

The transformed system of equations is now easy to solve, because D is diagonal and the lower right block of the matrix is only of dimension $p \times p$. ■

A particularly interesting case with structure, relevant for MPC implementations, is when the matrix is *banded*, i.e. the matrix elements are nonzero only when they are close to the diagonal. Such a structure can be obtained by ordering the variables in the appropriate way. As an illustration, consider how the model equations give rise to equality constraints with different structure, depending on the ordering of variables for a non-condensed formulation:

- Case 1: our standard ordering of decision variables $z^\top = [\boldsymbol{x}^\top \quad \boldsymbol{u}^\top]$ gives rise to the equality constraint $F\boldsymbol{z} = h$ with

$$F = \begin{bmatrix} I & & & & -B & & & \\ -A & I & & & & -B & & \\ & \ddots & \ddots & & & & \ddots & \\ & & -A & I & & & & -B \end{bmatrix}.$$

- Case 2: variables are ordered chronologically, i.e.

$$z^\top = \begin{bmatrix} x^\top(1) & u^\top(0) & \cdots & x^\top(N) & u^\top(N-1) \end{bmatrix}$$

so that the matrix F becomes

$$F = \begin{bmatrix} I & -B & & & & & \\ -A & 0 & I & -B & & & \\ & & -A & 0 & \ddots & & \\ & & & & \ddots & I & -B \\ & & & & & -A & 0 \end{bmatrix}$$

which gives the desired banded structure to be exploited for efficient factorization.

12 Explicit control laws for constrained linear systems

The theme throughout this course has been to formulate and analyse a model predictive controller based on the LQ formulation, extended with constraints. An integral part of the MPC algorithm is the task to solve an optimisation problem repeatedly online. It turns out, however, that for simple process models, there is an alternative route, namely to solve a *parametric* optimisation problem offline, resulting in explicit control laws that need significantly less online computations. The objective of this section is to give an introduction into these so-called *explicit MPC* algorithms.

This section is basically a condensed description of the contents of Chapter 7 of [1], which contains additional details. Chapters 6 and 7 of [2] contain related material.

12.1 Parametric programming

The reason why we have to *repeatedly* solve an optimisation problem in MPC algorithms is the fact that the current state x changes. This implies that the optimisation problem and its solution change with x, which is referred to as a *parameter* of the problem. So called *parametric programming* deals with these kind of optimisation problems, taking the form

$$V^*(x) = \min_u \{V(x,u) \mid u \in \mathcal{U}(x)\}, \tag{120}$$

where x is the parameter of the problem. Whereas the solution to a conventional optimisation problem is a *point* (or possibly a set), the solution to this parametric programming problem is actually a *function* $u^*(x)$ (which in the general case could be set valued).

The parametric constraint $u \in \mathcal{U}(x)$ can also be expressed as $(x,u) \in \mathcal{Z}$, where $\mathcal{Z}$ is a subset of (x,u)-space,

$$\mathcal{U}(x) = \{u \mid (x,u) \in \mathcal{Z}\}. \tag{121}$$

The domain of the function $V^*(x)$ in (120) is the set $\mathcal{X}$, defined by

$$\mathcal{X} = \{x \mid \exists u \text{ such that } (x,u) \in \mathcal{Z}\} = \{x \mid \mathcal{U}(x) \neq \emptyset\}. \tag{122}$$

To fix ideas, let's have a look at a simple example of a parametric programming problem.

Example 12.1 (Parametric quadratic programming [1])**.** Consider the parametric quadratic program $\min_u\{V(x,u) \mid (x,u) \in \mathcal{Z}\}$ with

$$V(x,u) = \frac{1}{2}\left((x-u)^2 + u^2\right)$$
$$\mathcal{Z} = \{(x,u) \mid u \geq 1,\ u + x/2 \geq 2,\ u + x \geq 2\}.$$

From $\nabla_u V(x,u) = -x + 2u$, it is clear that the unconstrained minimum is given by

$$u^*_{uc} = x/2,$$

but this solution does not fulfil the constraints for $x < 2$. Since $\nabla_u V(x,u) > 0$ for all $u > u^*_{uc}$, the constrained optimal solution $u^*(x)$ will lie on the boundary of $\mathcal{Z}$. The parametric optimal solution is illustrated in Figure 24. The solution is *piecewise affine*, and the value function $V^*(x)$ is *piecewise quadratic*.

■

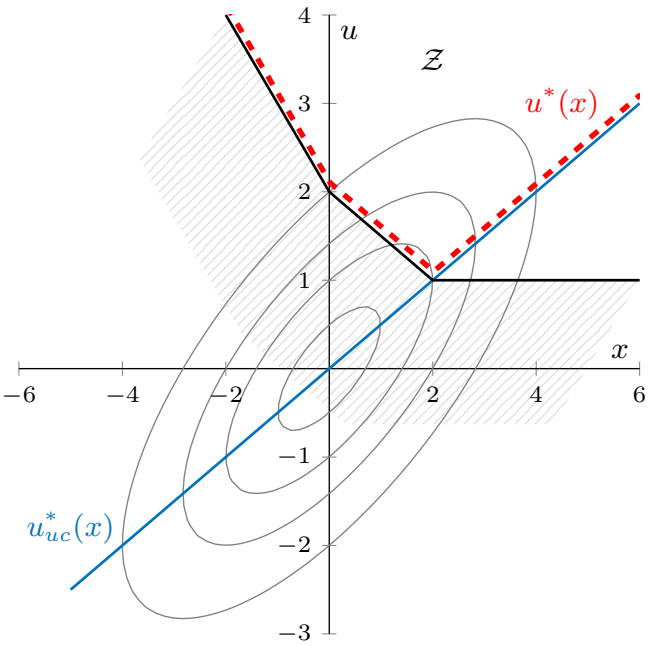


Figure 24: An illustration of the unconstrained, u^*_{uc}, and the constrained parametric solution, $u^*(x)$, to the quadratic program.

12.2 Constrained LQ control

As already indicated, the ideas presented above can be carried over to the LQ-type MPC focussed during the course. Recall that the receding horizon controller is based on the finite-time optimisation problem

$$V_N^*(x) = \min_{u(0:N-1)} \{V_N(x, u(0:N-1)) \mid u(0:N-1) \in \mathcal{U}_N(x)\}, \tag{123}$$

where the objective is given by

$$V_N(x, u(0:N-1)) = x^\top(N) P_f x(N) + \sum_{i=0}^{N-1} \left(x^\top(i) Q x(i) + u^\top(i) R u(i) \right), \quad x(0) = x. \tag{124}$$

In this formulation, the optimisation variables are the sequence of controls $u(0:N-1)$ or, equivalently, the vector $\boldsymbol{u} = \text{vec}\,(u(0), \ldots, u(N-1))$ as in (20). The states $x(i)$ in (124) can be thought of as short-hand notation for the expressions

$$x(i) = A^i x + \Gamma_i \boldsymbol{u}, \tag{125}$$

where Γ_i is the i^{th} row of the matrix Γ, see (19).

With the formulation above, all the constraints are encoded in the form of the set of allowed control sequences $u(0:N-1) \in \mathcal{U}_N(x)$, which depends on the initial state x. Taking a closer look at this set, we see that it is actually determined by the following constraints:

1. the control constraint set $\mathbb{U}$;
2. the state constraint set $\mathbb{X}$;
3. the terminal state constraint set $\mathbb{X}_f$.

Condition (1) implies that $\boldsymbol{u}$ is constrained by a polyhedral set. Conditions (2) and (3) constitute polyhedral constraints on $\{x(i)\}$, which can be translated to affine constraints expressed in terms of

$(x, \boldsymbol{u})$ by using the state expressions (125). The conclusion is that the *implicit* constraint set $\mathcal{U}_N$ can be expressed as an *explicit*, polyhedral set

$$\mathbb{Z} = \{(x, \boldsymbol{u}) \mid F\boldsymbol{u} \leq Gx + h\} \tag{126}$$

for some matrices F, G and vector h. Hence, the MPC optimisation problem can now be stated as

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & V_N(x, \boldsymbol{u}) = \frac{1}{2}(\boldsymbol{u}^\top \tilde{R}\boldsymbol{u} + x^\top \tilde{Q}x) + \boldsymbol{u}^\top Sx \\ \text{subject to} \quad & F\boldsymbol{u} \leq Gx + h \end{aligned} \tag{127}$$

for suitable definitions of the matrices $\tilde{R}, \tilde{Q}$, and S, cf. (21). In the sequel, it will be assumed that the matrix

$$\mathcal{Q} = \begin{bmatrix} \tilde{Q} & S^\top \\ S & \tilde{R} \end{bmatrix}$$

is positive definite, implying that both $\tilde{R}$ and $\tilde{Q}$ are positive definite.

We can now recognize that the optimisation problem (127) is parametric in x, since both the objective function and the constraints are given in terms of the parameter x, the current state. So far, we have not really exploited the structure of this parametric optimisation problem, except the fact that it is a QP that can be solved efficiently at every sampling instant for a new value of the state vector x. Note, however, that in the *unconstrained* case, we observed in Section 3.1 that the optimal control sequence is linear and the optimal cost quadratic in the initial state. We will shortly reveal that it is possible to generalize this (in a certain sense) to the constrained case. The key in doing this is to exploit the structure of the problem to relate solutions corresponding to *different* values of x.

Solution of the parametric program

Let's first note that the parametric optimisation problem (127) can also be stated as

$$V^*(x) = \min_{\boldsymbol{u}} \{V(x, \boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}(x)\} \tag{128}$$

where

$$\mathcal{U}(x) = \{\boldsymbol{u} \mid (x, \boldsymbol{u}) \in \mathbb{Z}\} = \{\boldsymbol{u} \mid F\boldsymbol{u} \leq Gx + h\} \tag{129}$$

$$\mathbb{Z} = \{(x, \boldsymbol{u}) \mid F\boldsymbol{u} \leq Gx + h\} \tag{130}$$

and we have dropped the subscript N. The domain of V^* is

$$\mathcal{X} = \{x \mid \exists \boldsymbol{u} \text{ such that } (x, \boldsymbol{u}) \in \mathbb{Z}\} = \{x \mid \mathcal{U} \neq \emptyset\}.$$

Since $\tilde{R}$ has been assumed positive definite, the solution $\boldsymbol{u}^*(x)$ is unique and its domain is $\mathcal{X}$.

We will now show the main ideas of how to exploit the structure of the problem, but for a stringent treatment please refer to [1], sections 7.3 and 7.4. The main steps to carry out are the following:

1. Pick an arbitrary state vector x, for which the optimal solution is $\boldsymbol{u}^*(x)$. Find a representation of this solution by using the fact that it is also the optimal solution of an equality constrained problem (with the subset of active inequality constraints being included as equality constraints and the remaining ones discarded).

2. Show that the same equality constrained problem, and its solution, is also valid for initial states close to x. In fact, there exists a polyhedron R_x^* in $\mathbb{R}^n$, containing x, such that, for each $w \in R_x^*$, $\boldsymbol{u}^*(w)$ is the solution of the optimisation problem *with the same set of equality constraints*. On the set R_x^*, $\boldsymbol{u}^*(w)$ is affine in w and $V^*(w)$ is quadratic in w.

3. Show that there are finitely many polyhedral regions $\{R_x^*\}$ covering the set of feasible states $\mathcal{X}$.

Step 1. For the arbitrary but, for the following discussion, fixed x, the unique optimal solution $\boldsymbol{u}^*(x)$ satisfies the necessary and sufficient conditions for optimality, given in Example 7.3. These conditions are that $\boldsymbol{u}$ is feasible and that

$$-\nabla_{\boldsymbol{u}} V(x) = -(\tilde{R}\boldsymbol{u} + Sx) = \sum_{i \in \mathbb{A}} \mu_i F_i^\top, \quad \text{for some } \{\mu_i\} \text{ with } \mu_i \geq 0 \tag{131}$$

where F_i is the i^{th} row of F and $\mathbb{A}$ is the active set of constraints. The latter condition can equivalently be expressed as a set of linear inequalities (see [1]), so that the optimality conditions can be written as a set of linear inequalities:

$$\begin{aligned} F\boldsymbol{u} &\leq Gx + h \\ -L_x^*(\tilde{R}\boldsymbol{u} + Sx) &\leq 0 \end{aligned} \tag{132}$$

where L_x^* is a matrix defined by the active constraints corresponding to $\boldsymbol{u}^*(x)$.

The solution $\boldsymbol{u}^*(x)$ can also be found as the solution to an equality constrained problem with only the active constraints. Denoting the subset of active constraints corresponding to x by $F_x^*\boldsymbol{u} = G_x^* x + h_x^*$, we thus have

$$\boldsymbol{u}^*(x) = \arg\min_{\boldsymbol{u}} \{V(x, \boldsymbol{u}) \mid F_x^*\boldsymbol{u} = G_x^* x + h_x^*\}.$$

Step 2. At least for points w close to x, it sounds plausible that the optimal solution $\boldsymbol{u}^*(w)$ would have the same set of active constraints as x. If so, then $\boldsymbol{u}^*(w)$ would actually be the solution of an equality constrained problem with *the same set of equality constraints*. We have not yet shown that this is the case, but let's define these solutions anyhow,

$$\boldsymbol{u}_x^*(w) = \arg\min_{\boldsymbol{u}} \{V(w, \boldsymbol{u}) \mid F_x^*\boldsymbol{u} = G_x^* w + h_x^*\}. \tag{133}$$

Here, the subscript x indicates that the solution is obtained by using the active constraints corresponding to the state x.

Now, the solution of a QP with linear constraints can be computed explicitly, see (96). The result is that $\boldsymbol{u}_x^*(w)$ is affine in w and that $V_x^*(w)$ is quadratic in w:

$$\boldsymbol{u}_x^*(w) = K_x w + k_x \tag{134}$$

$$V_x^*(w) = \frac{1}{2} w^\top Q_x w + r_x^\top + s_x. \tag{135}$$

The question that needs to be answered is for which set of w:s this solution is in fact optimal, i.e. $\boldsymbol{u}_x^*(w) = \boldsymbol{u}^*(w)$. But we have actually formulated necessary and sufficient optimality conditions corresponding to the active set for x in equation (132). By inserting the expression for $\boldsymbol{u}_x^*(w)$ in this equation and replacing x with w, we obtain a set of inequalities for w:

$$\begin{aligned} F(K_x w + k_x) &\leq Gw + h \\ -L_x^*(\tilde{R}(K_x w + k_x) + Sw) &\leq 0. \end{aligned} \tag{136}$$

This set of inequalities defines a polyhedron R_x^* for w. The conclusion is that for any $w \in R_x^*$, the previously computed solution $\boldsymbol{u}_x^*(w)$ satisfies the optimality conditions for the original problem. Hence,

$$\boldsymbol{u}^*(w) = \boldsymbol{u}_x^*(w) = K_x w + k_x, \quad \forall w \in R_x^*. \tag{137}$$

It follows that the value function is quadratic in R_x^*.

Step 3. We concluded in the previous step that the control law is an affine function of the state within a polyhedral set R_x^*, and that the optimal cost is a quadratic function of the state within the same set. Now, the set R_x^* is characterized by the set of active constraints, and there is a finite set of subsets of active constraints, telling us that there are finitely many polyhedral regions R_x^*. Also, any arbitrary feasible $x \in \mathcal{X}$ belongs to a region R_x^*. Hence, we can conclude that the feasible set $\mathcal{X}$ is partitioned by a collection of non-overlapping polyhedra $\cup\{R_x^*\} \mid x \in \mathcal{X}$. Corresponding to this partition, the optimal solution to the parametric program (127) is piecewise affine with a value function that is piecewise quadratic. Moreover, as shown in [1], the value function $V^*(x)$ and the minimizer $\boldsymbol{u}^*(x)$ are continuous in $x \in \mathcal{X}$.

One of the conclusions from the above is that the, so far implicit, control law is *piecewise affine*. We have actually noticed this in a previous example:

Example 12.2 (LQ MPC for an integrator, cont'd)**.** In Example 4.1, we investigated an LQ-based MPC for a simple integrator system

$$x^+ = x + u$$

with a control constraint

$$u \in \mathbb{U} = [-1, 1]$$

and with $Q = R = P_f = 1$, $N = 2$. For this example, we concluded that the receding horizon control law could be given in closed-form as

$$u(x) = \begin{cases} 1 & x \leq -5/3 \\ -3/5 \cdot x & -5/3 \leq x \leq 5/3 \\ -1 & x \geq 5/3. \end{cases}$$

We can thus verify that the control law is piecewise affine, as predicted, and that the partition of the state space is given by three intervals, two of which are semi-infinite. ■

Summary Let's summarize and interpret what we have concluded. When solving the optimisation problem for a specific x, we usually obtain the solution (the optimal control sequence) for this particular value of x *only*. However, what the analysis has revealed is the fact that, if the optimisation routine is implemented to deliver instead the *parameters* of the affine expression for the optimal $\boldsymbol{u}^*(x)$, then we actually have the optimal solution for *all* x belonging to R_x^*! In addition, the solution is given *explicitly* as an affine function of the state. This seems very promising, even though we can right away note that the number of sets R_x^* may be very large—indeed, each such set corresponds to one particular subset of active constraints!

Encouraged by the results quoted above, the following strategy could be envisioned for a potentially very efficient implementation of our MPC:

1. Find a partition $\{R_{x_i}\}$ of $\mathcal{X}$.

2. Calculate the optimal, affine control law on each of the regions R_{x_i} and store the result, i.e. controller parameters. This and the previous step can be done *off-line*.

3. During *on-line* operation, run the controller by

   - for the measured state x, identify the region to which x belongs;
   - look up the controller parameters for the found region;
   - compute the next control signal.

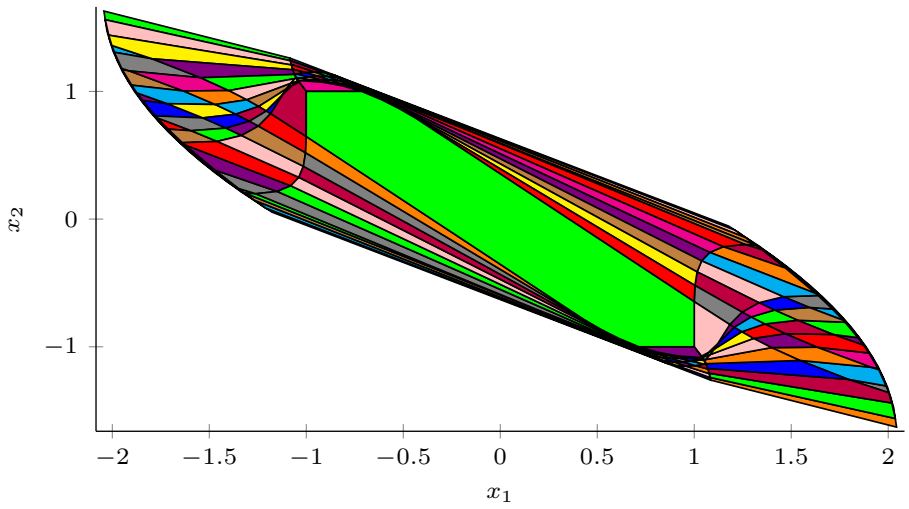


Figure 25: An illustration of a partitioned state-space with a possibly different, optimal, affine control law in each of the regions.

The resulting implementation of the receding horizon controller is often referred to as *explicit MPC*, although it is, strictly speaking, not an MPC in the usual sense any longer. Rather, it is a procedure to find explicit control laws for a class of finite time optimal control problems. Nevertheless, "explicit MPC" can be an attractive alternative to the classical MPC for small problems (i.e. fairly simple models and moderate prediction horizons). However, the computational and storage requirements are quickly becoming challenging as more realistic problems are considered. The main reason for this is that the number of regions easily grows to tens of thousands or more, and the advantages of the explicit approach become questionable. The situation may change in the future, since much research is conducted in the area.

It was mentioned above that the number of regions R_i may become very large. This is indicated in Figure 25, which shows the regions for a simple second order system.

13 Alternative formulations of MPC

This section will give an overview of alternative formulations of MPC controllers, as well as some extensions to the type of MPC algorithms studied in the course.

Part of the material is found in Sections 4.1 and 5.1-5.7 of [4].

13.1 Step/impulse response and transfer function models

The focus in this course is on model predictive control based on (linear) state space models. This reflects much of the recent and current development in the area, despite the fact that much of the early development favoured other types of models. There are a number of reasons to use state space models:

- Generality – linear and nonlinear.
- Often result from modelling from first principles.
- Linearisation and discretisation is straightforward.
- Computational delays can be incorporated.
- State observers can be designed.
- Numerical computations.

There are, however, alternatives to using state space models in MPC. One popular choice in the early days of MPC was step response models. They have the following characteristics:

+ Simple to understand.
+ Easy to get from step response experiments.
– Emphasis on low frequency behaviour.
– Constrained to stable plants.
– Inefficient model representation (e.g. first order lag with large T).

An LTI input-output model is given by

$$y(t) = \sum_{k=0}^{t} H(t-k)u(k), \tag{138}$$

where $\{H(0), H(1), \ldots\}$ is the (matrix valued) impulse response. The step response is obtained as

$$S(t) = \sum_{k=0}^{t} H(k).$$

If the step response (or the impulse response) is truncated at N with $S(N+1) \approx S(N)$, the remaining sequence is a model of the system:

- Dynamic matrix: $\{S(0), S(1), \ldots S(N)\}$ (used in DMC);
- FIR model: $\{H(0), H(1), \ldots H(N)\}$.

The two models above are very closely related. In fact, going from absolute control input u to control moves Δu illustrates this:

$$\begin{aligned} y(t) &= \sum_{k=0}^{t} H(t-k)u(k) = \sum_{k=0}^{t} \left(H(t-k) \cdot \sum_{i=0}^{k} \Delta u(i) \right) \quad \text{(set } u(0) = 0\text{)} \\ &= \sum_{k=0}^{t} H(t-k)\Delta u(0) + \sum_{k=1}^{t} H(t-k)\Delta u(1) + \sum_{k=2}^{t} H(t-k)\Delta u(2) + \dots \\ &= S(t)\Delta u(0) + S(t-1)\Delta u(1) + \dots + S(0)\Delta u(t) \\ &= \sum_{k=0}^{t} S(t-k)\Delta u(k), \end{aligned}$$

which should be compared with (138).

The state space model (with $D = 0$)

$$\begin{aligned} x^+ &= Ax + Bu \\ y &= Cx \end{aligned}$$

has an impulse response given by the *Markov parameters* $\{CA^iB\}$, giving the step response

$$S(k) = \sum_{i=0}^{k-1} CA^iB.$$

Using this in (52) reveals the close connections to the state space formulation:

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} S(1) \\ S(2) \\ \vdots \\ S(N) \end{bmatrix} u(k-1) + \begin{bmatrix} S(1) & 0 & \cdots & 0 \\ S(2) & S(1) & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ S(N) & S(N-1) & \cdots & S(1) \end{bmatrix} \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+N-1|k) \end{bmatrix}.$$

In a vector notation, this can also be written as

$$\boldsymbol{y}(k) = \boldsymbol{y_f}(k) + \Theta \boldsymbol{\Delta u}(k).$$

Remark. *It is shown in [4] how the free response can be computed from stored control moves over the settling time of the process.*

From the relation (assuming $D = 0$)

$$H(k) = CA^{k-1}B$$

the following *Hankel matrix* can be formed:

$$\begin{bmatrix} H(1) & H(2) & H(3) & \cdots \\ H(2) & H(3) & H(4) & \cdots \\ H(3) & H(4) & H(5) & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} CB & CAB & \cdots \\ CAB & CA^2B & \cdots \\ CA^2B & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} B & AB & A^2B & \cdots \end{bmatrix}.$$

By collecting step response data, the matrix above can be factorized for a suitable order (dimension of the Hankel matrix), and a state space representation (A, B, C) can be obtained from the factors.

Transfer function models

Transfer function models have also been used in the MPC literature, for example in connection with an MPC algorithm known as GPC (Generalized Predictive Control). Section 4.2 in [4] is devoted to this, but the material is not considered to be part of the course.

13.2 Other variations of MPC

The state space model can easily be extended with measurable disturbances:

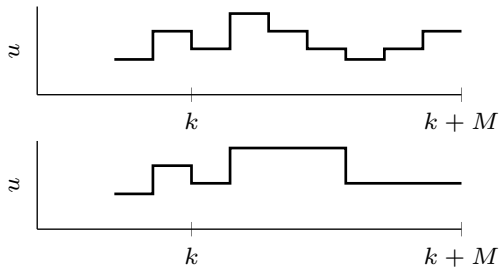
$$x(k+1) = Ax(k) + Bu(k) + B_d d_m(k).$$

- Include the disturbance in the state prediction using $B_d \hat{d}_m(k+i|k)$.
- The control signal computed will try to compensate for the anticipated effect of the disturbance ("look-ahead").

The control signal can be re-parametrised:

- Reduction of complexity:

  Limit the number of optimisation variables by keeping M low or by using *blocking*, i.e. keeping the control signal constant over several sampling intervals.



- The use of *stabilised predictions* can improve numerics:

$$u(k+i|k) = -K\hat{x}(k+i|k) + \tilde{u}(k+i|k),$$

  implying that the predictor expressions will contain $(A - BK)^i$ instead of A^i.

- Parameterize the control signal using *basis functions*:

$$u(k+i|k) = \sum_{j=1}^{n_u} c_j u_j(i),$$

  where $\{u_j(\cdot)\}$ are basis functions (e.g. polynomials). Once the coefficients $\{c_j\}$ have been determined (in the optimisation step of the MPC), the control signal is defined over the whole prediction horizon. This is called *Predictive Functional Control (PFC)*.

As an input to the optimisation in the MPC, the desired behaviour of the system within the prediction horizon needs to be expressed. There are many ways to do this — the following variants are discussed further in [4]:

- A *reference trajectory* $r(\cdot)$ for the output(s) is defined, starting at the current output and ending at the setpoint $s(\cdot)$. Typically, the objective contains a term $(y-r)^\top Q(y-r)$.
- If a setpoint is not really relevant, then it is possible to instead define a *zone objective*. In this case, an objective as above is not meaningful, and instead constraints are given for upper and lower bounds of the output(s).
- Combining the two ideas above, it is possible to define constraints that become stricter towards the end of the prediction horizon, thus forming a *funnel* in which the states or outputs should reside.

14 Beyond linear MPC

The objective of this section is to give an outlook on techniques for model predictive control, going beyond the class of problems treated so far in the course. We have focussed most of our attention on what is often called *linear MPC*, i.e. MPC algorithms based on linear state space models, quadratic objective and affine constraints. A lot of effort has been spent over the last couple of decades to extend the concepts to deal also with nonlinear models and to models including so called hybrid dynamics. Much research is still going on concerning robust formulations of MPC. We will take a brief look at these extensions.

Much of the conceptual discussion of MPC in [1] treats the nonlinear case. Chapter 3 in [1] treats robust MPC, as does Sections 8.1-8.5 of [4].

So far in the course, we have basically limited our attention to MPC based on linear models, quadratic costs and affine constraints. This class of model predictive algorithms is often referred to as *linear MPC*. There are strong reasons to focus on linear MPC when first exposed to the field, because of the strong relationships with unconstrained LQ control and the role of the well-known QP problem in its implementation. However, the MPC idea is not limited to the linear case, and parts of the investigation of the RHC concept have indeed been pursued during the course in a more general setting. In this section, we will take a brief look at some of the extensions of the MPC idea beyond the linear case.

14.1 Nonlinear MPC

Remarkable advancements have been made over the last 10-15 years to extend the MPC idea to the nonlinear case. We will sketch some of the ideas underlying this development, which has turned nonlinear MPC into a viable control design technique for many applications with high real-time demands. This section is very much inspired by the recent paper [6].

MPC based on linearisation

The first thing that comes to mind is whether linear MPC can be applied to a linearised version of a nonlinear control design problem stemming from a nonlinear time-invariant model

$$x^+ = f(x,u) \tag{139}$$

with inequality constraints

$$g(x,u) \leq 0. \tag{140}$$

We begin our discussion of this problem by once again reminding about the basic algorithm for linear MPC, based on repeatedly solving the following constrained optimisation problem for the *deviation variables*. At every time instant k, solve the constrained optimisation problem

$$\begin{aligned}
\underset{\delta u_k, \delta x_k}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \delta x_k(i) \\ \delta u_k(i) \end{bmatrix}^\top W \begin{bmatrix} \delta x_k(i) \\ \delta u_k(i) \end{bmatrix} && \text{(141a)} \\
\text{subject to} \quad & \delta x_k(0) = \hat{x}(k) - x_k^r(0) && \text{(141b)} \\
& \delta x_k(i+1) = A\,\delta x_k(i) + B\,\delta u_k(i); \;\; i = 0, \ldots, N-1 && \text{(141c)} \\
& F\,\delta u_k(i) + G\,\delta x_k(i) \leq h; \;\; i = 0, \ldots, N-1. && \text{(141d)}
\end{aligned}$$

The solution is written as

$$(\boldsymbol{\delta u}_k, \boldsymbol{\delta x}_k) = \text{QP}_{\text{MPC}}(\hat{x}(k), \boldsymbol{u}_k^r, \boldsymbol{x}_k^r). \tag{141e}$$

Here, we have simplified the formulation slightly by assuming no terminal penalty and no terminal constraint, and the weighting matrix W is given by $W = \mathrm{diag}(Q, R)$. Also notice the notation introduced, namely the subscript k indicating the sampling instant and the index i indicating the time step within the prediction interval. Thus, $\boldsymbol{\delta x}_k = (\delta x_k(0), \ldots, \delta x_k(N-1))$ and $\boldsymbol{\delta u}_k = (\delta u_k(0), \ldots, \delta u_k(N-1))$ comprise the optimisation variables over the prediction horizon at time k. They are all *deviation variables* relative to the *reference trajectory*, obeying the equation

$$x_k^r(i+1) = A\, x_k^r(i) + B\, u_k^r(i). \tag{142}$$

This in turn is a slight generalization of the case treated earlier, when the reference was given by constant steady state target variables x_s and u_s.

To proceed, note that the formulation of the LTI MPC in (141) can without any complications in the solution be generalized to the linear *time-varying* case. At every time instant k, solve the constrained optimisation problem

$$\begin{aligned}
\underset{\delta u_k, \delta x_k}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \delta x_k(i) \\ \delta u_k(i) \end{bmatrix}^\top W_{k,i} \begin{bmatrix} \delta x_k(i) \\ \delta u_k(i) \end{bmatrix} && \text{(143a)} \\
\text{subject to} \quad & \delta x_k(0) = \hat{x}(k) - x_k^r(0) && \text{(143b)} \\
& \delta x_k(i+1) = A_{k,i}\, \delta x_k(i) + B_{k,i}\, \delta u_k(i) + r_k(i); \;\; i = 0, \ldots, N-1 && \text{(143c)} \\
& F_{k,i}\, \delta u_k(i) + G_{k,i}\, \delta x_k(i) \leq h_{k,i}; \;\; i = 0, \ldots, N-1. && \text{(143d)}
\end{aligned}$$

The solution is written as

$$(\boldsymbol{\delta u}_k, \boldsymbol{\delta x}_k) = \mathrm{QP}_{\mathrm{MPC}}(\hat{x}(k), \boldsymbol{u}_k^r, \boldsymbol{x}_k^r). \tag{143e}$$

The control output is given by

$$u(k) = u_k^r(0) + \delta u_k(0). \tag{143f}$$

The resulting problem is still a QP, as indicated by the notation used, albeit with coefficients that may vary over the prediction interval and also with absolute time k. Note that the model equality constraint has a residual $r_k(i)$ added to cover the case when the reference trajectory does not satisfy the model equation, i.e.

$$r_k(i) = A_{k,i}\, x_k^r(i) + B_{k,i}\, u_k^r(i) - x_k^r(i+1). \tag{144}$$

Now, returning to our nonlinear model, a natural idea is to apply the linear, time-varying MPC above to a linearised version of the nonlinear model; this would be in analogy with standard practice in classical control design based on e.g. PID control. In the MPC case, we choose to linearise around a reference trajectory as follows:

$$\begin{aligned}
A_{k,i} &= \frac{\partial f(x,u)}{\partial x}\Big|_{x_k^r(i), u_k^r(i)} & B_{k,i} &= \frac{\partial f(x,u)}{\partial u}\Big|_{x_k^r(i), u_k^r(i)} && \text{(145a)} \\
G_{k,i} &= \frac{\partial g(x,u)}{\partial x}\Big|_{x_k^r(i), u_k^r(i)} & F_{k,i} &= \frac{\partial g(x,u)}{\partial u}\Big|_{x_k^r(i), u_k^r(i)} && \text{(145b)} \\
r_k(i) &= f(x_k^r(i), u_k^r(i)) - x_k^r(i+1) & h_{k,i} &= h(x_k^r(i), u_k^r(i)). && \text{(145c)}
\end{aligned}$$

With these definitions, the equality constraints (143b) and the inequality constraints (143c) approximate the nonlinear dynamics (139) and (140), respectively, in the vicinity of the reference trajectory.

As usual, a limitation when using a linearised system model is that the approximation of the original nonlinear problem will be less relevant if variables depart significantly from the reference trajectory. This said, the outlined procedure to apply MPC to nonlinear systems is in principle straightforward, and a clear advantage is that the optimisation problem to be solved at each sampling instant is still a QP. One needs to bear in mind, though, that calculation of the Jacobians may be associated with considerable computational effort, in particular when the model (139) is a discretisation of an underlying continuous-time model—we will return to this. If possible, it is tempting to perform as much as possible of these computations off-line in advance. This would require that the reference trajectories for different scenarios are also determined off-line.

Nonlinear MPC

In situations where it is not enough to capture the nonlinearities by linearisations around a reference trajectory, we may be motivated to consider *nonlinear MPC (NMPC)* in which the nonlinear dynamics and nonlinear constraints are taken care of explicitly. The formulation of the NMPC algorithm is as follows. At every time instant k, solve the constrained optimisation problem

$$\begin{aligned}
\underset{u_k, x_k}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} x_k(i) - x_k^r(i) \\ u_k(i) - u_k^r(i) \end{bmatrix}^\top W_{k,i} \begin{bmatrix} x_k(i) - x_k^r(i) \\ u_k(i) - u_k^r(i) \end{bmatrix} && \text{(146a)} \\
\text{subject to} \quad & x_k(0) = \hat{x}(k) && \text{(146b)} \\
& x_k(i+1) = f(x_k(i), u_k(i)); \quad i = 0, \ldots, N-1 && \text{(146c)} \\
& h(x_k(i), u_k(i)) \leq 0; \quad i = 0, \ldots, N-1. && \text{(146d)}
\end{aligned}$$

The solution is written as

$$(\boldsymbol{u}_k, \boldsymbol{x}_k) = \text{NLP}(\hat{x}(k), \boldsymbol{u}_k^r, \boldsymbol{x}_k^r). \tag{146e}$$

The control output is given by

$$u(k) = u_k(0). \tag{146f}$$

It can be seen in this formulation that we no longer rely on deviation variables, since linearity does not hold. However, the objective is still expressed in terms of deviations from a reference trajectory that may or may not fulfil the nonlinear state equations and constraints.

The finite horizon optimal control problem to be solved at each sampling instant is a nonlinear programming problem (NLP), hence the notation above. There are many ways to solve this problem, involving in one way or another (approximations of) Newton's method to search for solutions that satisfy necessary conditions for optimality. One common technique, the Sequential Quadratic Programming (SQP) method, can be motivated from a previous exercise, where it was seen how the Newton direction for an equality constrained problem can be found by solving a QP based on a linearisation of the problem at the current iterate. Similarly, in the SQP method a sequence of QPs are solved iteratively, each derived from the current linearisation. Applied to our NMPC problem, the SQP method would look as follows. At every time instant k, the SQP optimisation variables $(\boldsymbol{u}_k, \boldsymbol{x}_k)$ are initialized as

$$(\boldsymbol{u}_k, \boldsymbol{x}_k) = (\boldsymbol{u}_k^{\text{guess}}, \boldsymbol{x}_k^{\text{guess}}). \tag{147a}$$

Then the following QP is solved repeatedly at the current SQP iterate $(\boldsymbol{u}_k, \boldsymbol{x}_k)$ for the Newton correction $(\Delta\boldsymbol{u}_k, \Delta\boldsymbol{x}_k)$, which gives the next iterate by taking a (reduced) Newton step $(\boldsymbol{u}_k, \boldsymbol{x}_k) \leftarrow (\boldsymbol{u}_k, \boldsymbol{x}_k) + t(\Delta\boldsymbol{u}_k, \Delta\boldsymbol{x}_k)$:

$$\begin{aligned}
\underset{\Delta u_k, \Delta x_k}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta x_k(i) \\ \Delta u_k(i) \end{bmatrix}^\top H_{k,i} \begin{bmatrix} \Delta x_k(i) \\ \Delta u_k(i) \end{bmatrix} + J_{k,i}^\top \begin{bmatrix} \Delta x_k(i) \\ \Delta u_k(i) \end{bmatrix} && \text{(147b)} \\
\text{subject to} \quad & \Delta x_k(0) = \hat{x}(k) - x_k(0) && \text{(147c)} \\
& \Delta x_k(i+1) = A_{k,i}\, \Delta x_k(i) + B_{k,i}\, \Delta u_k(i) + r_k(i); && \text{(147d)} \\
& F_{k,i}\, \Delta u_k(i) + G_{k,i}\, \Delta x_k(i) + h_{k,i} \leq 0; \quad i = 0, \ldots, N-1. && \text{(147e)}
\end{aligned}$$

The solution of the SQP is written as

$$(\boldsymbol{u}_k, \boldsymbol{x}_k) = \text{SQP}(\hat{x}(k), \boldsymbol{u}_k^{\text{guess}}, \boldsymbol{x}_k^{\text{guess}}, u_k^r, x_k^r). \tag{147f}$$

The control output is given by

$$u(k) = u_k(0). \tag{147g}$$

In the SQP algorithm given, $H_{k,i}$ is an approximation of the Hessian of the Lagrangian of the original NLP; a common choice is the Gauss-Newton approximation, which in this case is simply given by $H_{k,i} = W_{k,i}$. Further, the coefficient matrices/vectors that appear in (147d)-(147e) correspond to the expressions in (145) but now evaluated at the current iterate. Finally,

$$J_{k,i} = W_{k,i} \begin{bmatrix} x_k(i) - x_k^r(i) \\ u_k(i) - u_k^r(i) \end{bmatrix}.$$

There is a striking similarity between the linear MPC QP in (143) and the QP in the SQP for the NMPC. In fact, if we initialize the SQP algorithm with the reference trajectory, i.e.

$$(\boldsymbol{u}_k^{\text{guess}}, \boldsymbol{x}_k^{\text{guess}}) = (\boldsymbol{u}_k^{\boldsymbol{r}}, \boldsymbol{x}_k^{\boldsymbol{r}})$$

then it can be seen that *the first iteration* of the SQP algorithm is identical to the linear MPC QP and $(\boldsymbol{\delta u}_k, \boldsymbol{\delta x}_k) = (\Delta \boldsymbol{u}_k, \Delta \boldsymbol{x}_k)$. Hence, with a full Newton update, the solutions are identical (compare (143f) for the linear MPC with the Newton update and (147g) for the NMPC).

Example 14.1 (Comparison between linear MPC and nonlinear MPC, [6])**.** Consider the following simple problem

$$\begin{aligned}
\underset{\boldsymbol{u}_k, \boldsymbol{x}_k}{\text{minimize}} \quad & \sum_{i=0}^{N} \|x_k(i)\|^2 + 20 \sum_{i=0}^{N-1} \|u_k(i)\|^2 && \text{(148a)} \\
\text{subject to} \quad & x_k(0) = \hat{x}(k) && \text{(148b)} \\
& x_k(i+1) = 0.9 x_k(i) + \begin{bmatrix} \sin(\begin{bmatrix} 0 & 1 \end{bmatrix} x_k(i)) \\ u_k(i) + u_k(i)^3 \end{bmatrix}, \quad i = 0, \ldots, N-1 && \text{(148c)} \\
& |u_k(i)| \leq 0.5, \quad i = 0, \ldots, N-1. && \text{(148d)}
\end{aligned}$$

In Figure 26, the solutions are compared for $N = 10$ using linear MPC and nonlinear MPC. For the NMPC, the results from using only one iteration of the SQP with full Newton step $t = 1$ and initialization at the reference is compared with running the SQP to full convergence. The equivalence between the linear MPC and the "one-step NMPC" is clearly seen.

The reference trajectory, used above to initialize the SQP algorithm, might not be a very good initial guess in many cases. A better choice is often the solution obtained by just picking the *tail* from the previous solution (or, with a different term, using the *shifted* solution), as was indeed done when analysing both recursive feasibility and stability. The last values of both control and state sequences need to be appended, and similar ideas as exploited earlier can be used. However, simpler choices are often used, e.g. copying the previous control action and making a one-step ahead prediction of the state based on this. Figure 27 illustrates that this strategy can be very effective, with only slight adjustments of the initial guess needed in the absence of disturbances or model errors, whereas Figure 28 illustrates that the initialization is not as good in the presence of disturbances. ■

Real-time aspects

In spite of the conceptual similarities between NMPC using SQP and linear MPC observed above, it is important to stress that the NMPC is potentially much more computational demanding, since a number of QPs have to be solved *iteratively* instead of just once, and in every iteration the Jacobians or sensitivities in (145) have to be computed. This may lead to undesired long sampling periods of the controller and/or computational delays that become a significant fraction of the sampling period. The Real-Time Iteration scheme (RTI) [6] (and there are other similar schemes) addresses this dilemma by adopting two mechanisms:

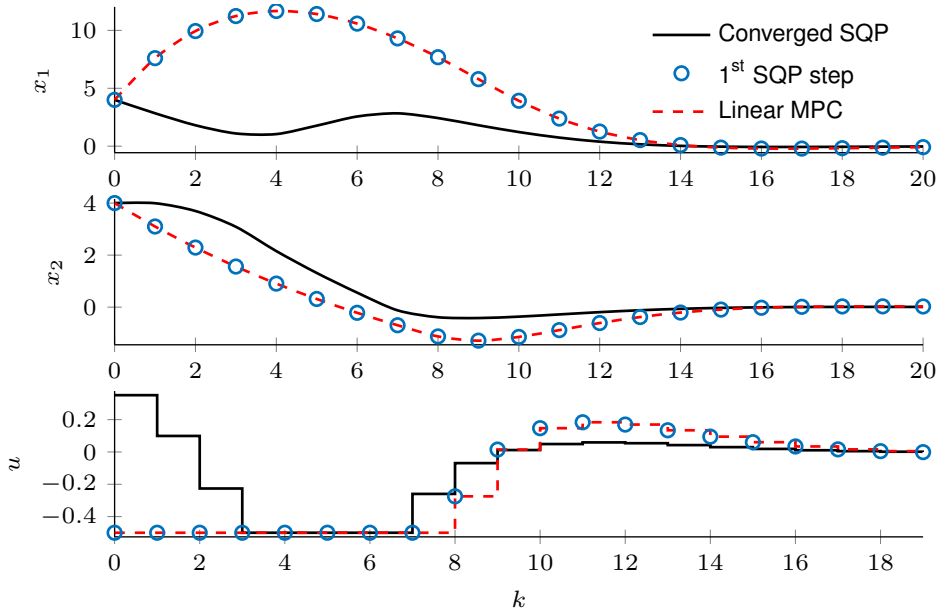


Figure 26: Comparison between NLP solution from SQP after full convergence, SQP after one step and linear MPC.

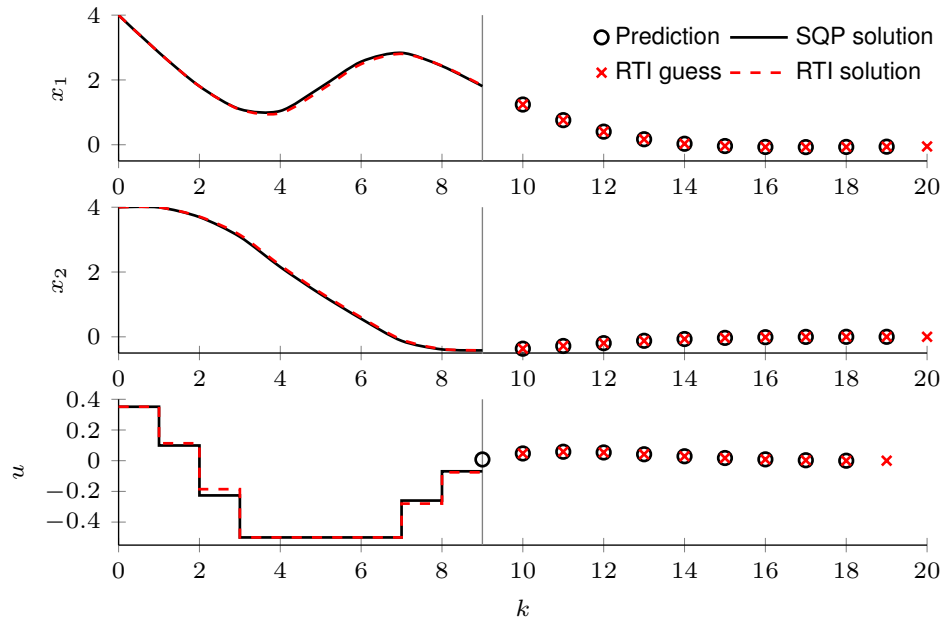


Figure 27: Real-time iteration scheme (RTI) in the absence of disturbances or model errors.

1. Computing only the first iteration of the SQP algorithm based on initialization using the previous, shifted solution, as demonstrated in the example above. This strategy favours an *approximate* solution that can be computed fast over an *exact* solution that requires more computations.

2. Computing the sensitivities in (145) in a *preparation step* during the *previous* sampling interval. This is possible because the initial guess is obtained from the previously computed solution.

The first mechanism implies that only one QP needs to be solved, similarly to the linear MPC case. The second mechanism implies that the computational delay from sampling the process to actuating the new control input involves only the QP algorithm and not the computation of the sensitivities. However, the total computation time, including both the preparation phase and solving the QP, is still what decides the achievable sampling rate of the MPC controller.

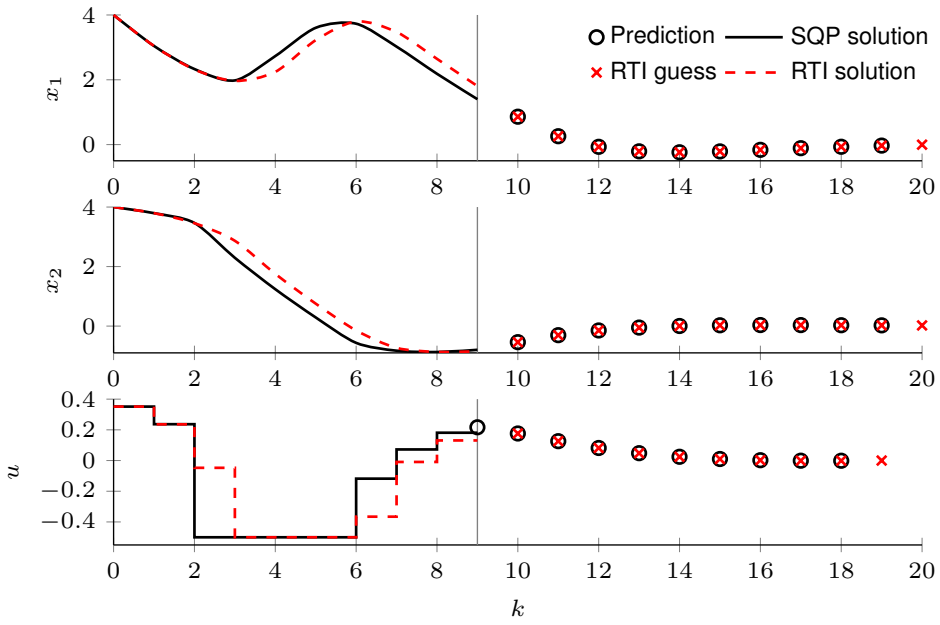


Figure 28: Real-time iteration scheme (RTI) in the presence of disturbances. The circles indicate predictions made during the previous MPC update, which are used as a basis to initialise RTI. The solid line shows the optimal solution for the disturbed state.

Nonlinear continuous-time dynamics

So far, we have discussed nonlinear MPC on the basis of the nonlinear discrete-time system model (139). Normally, however, a nonlinear model that is based on first principles modelling is formulated in continuous time, e.g. as a nonlinear state space model:

$$\dot{x}(t) = f(x(t), u(t)).$$

Even if we make the usual assumption that the control input is piece-wise constant, the application of nonlinear MPC to this system presents the principal difficulty to compute the discretised dynamics and sensitivities as in (145). There are essentially two ways to approach this task.

First linearise, then discretise. Assuming piecewise constant control signal, it is relatively straightforward *in the linear case* to go from a continuous-time model to a discrete-time model, either by means of the matrix exponential (the LTI case) or the transition matrix (the LTV case). In the linear MPC context, linearisation is typically performed around a constant steady state or a reference trajectory, as seen earlier. In the nonlinear MPC case, linearisation is performed around the previous, shifted solution, and will therefore be inexact. In addition, the matrix exponential will give a bad result in approximating the nonlinear dynamics if the sampling time is not negligible compared to the "time constant" of the nonlinear system.

First discretise, then linearise. An alternative approach is to swap the order between linearisation and discretisation. This basically means that the continuous-time dynamics is simulated between sampling instants, assuming piece-wise constant control input. For this, numerical integration schemes, like Euler's method or any type of Runga-Kutta method, can be used. This way, a much better approximation of the solution to the nonlinear state equation can be obtained. In addition, it turns out that sensitivities can also be efficiently computed by essentially differentiating the integration algorithm itself. The technique, which is referred to as *algorithmic differentiation (AD)*, has been introduced in this context only over the recent years, and there is still a lot of research going on in the area.

14.2 Robust MPC

When we introduced and motivated the receding horizon idea, the arguments went basically as follows:

1. Having observed the nice properties of infinite horizon LQ control, we pointed out that we would ideally like to solve an *infinite time constrained optimal control problem*. This is, however, typically not tractable.

2. With a reduced ambition, we formulated a *finite-time constrained optimal control problem*. If we could solve this using DP, we would get a result in the form of a feedback control law. Again, the solution is in general not tractable (but we have gained some additional insight in the previous subsection).

3. In the final step, we noted that we could instead solve the finite-time optimal control problem for the *optimal open-loop control sequence* and apply the first control signal according to the receding horizon principle. The price we have to pay is that the computations need to be repeated at each sampling instant for a new value of the state x.

In the sequence of simplifications of the problem above, it is important to notice that the 2nd and 3rd steps are completely equivalent, *if there are no uncertainties*! In the presence of uncertainties, however, there may be significant differences between open-loop and closed-loop solutions. In an MPC setting, there are several types of uncertainty that would be relevant to consider, for example:

- The state being not fully accessible. This requires an observer as discussed already.

- Disturbances entering the process, e.g. in the form of an additive process disturbance:

$$x^+ = f(x, u) + w$$

  Since the formulation of MPC often deals with hard constraints on the states, one natural choice of disturbance characterization is to assume w is uniformly bounded.

- The process model being uncertain. One common technique is based on introducing *parametric uncertainties*, which typically means that a linear model

$$x^+ = Ax + Bu$$

  is assumed, but that some parameters of A and B are only known to belong to certain intervals. The parameter vectors can then be described by polyhedral sets, which fits fairly well with the type of computations already considered in the course.

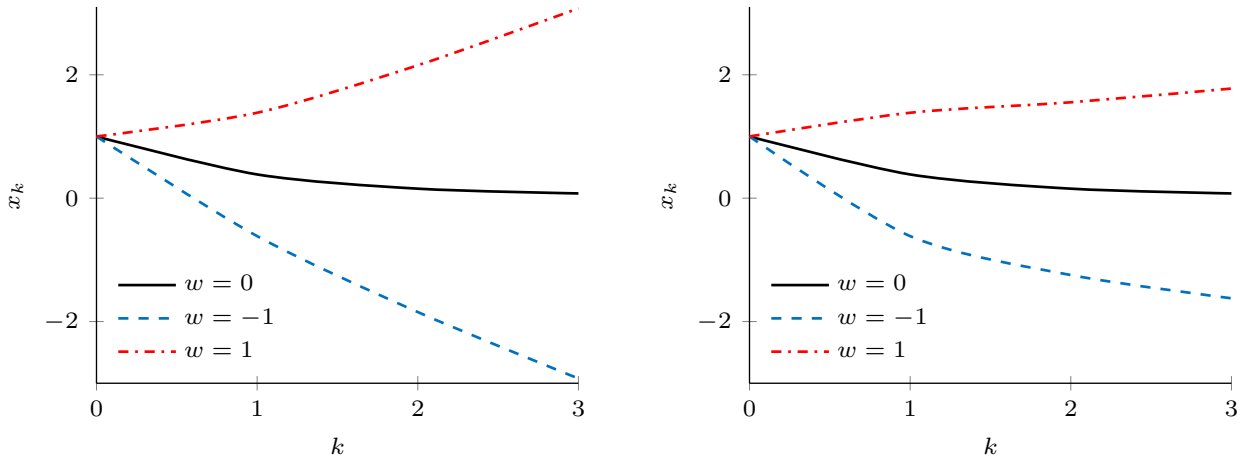
The following example illustrates the importance of distinguishing between open-loop and closed-loop control in the presence of uncertainties of the additive type mentioned above.

Example 14.2 (Feedback versus open-loop control, [1])**.** Consider again the simple integrator system

$$x^+ = x + u + w,$$

now with an additive disturbance w, assumed to be bounded $w \in [-1,\, 1]$. We will study the finite-time optimal control problem with $N = 3$ and $Q = R = P_f = 1$. With the disturbance w added to the model, there are several possible formulations of the cost function. One alternative is to consider the min-max optimisation problem

$$\min_{\mu(\cdot)} \max_{w\in[-1,1]} V_3(x, \mu, w),$$



(a) Open loop control.

(b) Closed-loop, feedback control.

Figure 29: Receding horizon control of an uncertain system with an additive process disturbance w.

where $\mu(\cdot)$ are feedback policies and V_3 is the usual quadratic 3-stage cost function with the initial state x. The intention with this formulation is to consider the disturbance sequence that gives the "worst-case" scenario. For simplicity, we will however stick to the nominal case with $w = 0$. Solving the standard unconstrained finite-time optimisation problem results either (by applying the batch solution) in the *open-loop control sequence*

$$u^*(0:2) = (-8/13 \cdot x(0), -3/13 \cdot x(0), -1/13 \cdot x(0))$$

or (by applying DP) in the feedback policy

$$\mu(0:2) = (-8/13 \cdot x, -3/5 \cdot x, -1/2 \cdot x)\,.$$

In the nominal case ($w = 0$), the two solutions are of course identical and can be shown to give the optimal state sequence

$$x^*(x) = (x, 5/13 \cdot x, 2/13 \cdot x, 1/13 \cdot x)\,,$$

where x is the initial state. When applied to the real system with nonzero w, however, there will be a difference in cost between the open-loop and closed-loop solutions. This is illustrated in Figure 29, where the response to three different disturbance sequences are shown, namely $w(\cdot) = 0$, $w(\cdot) = -1$, and $w(\cdot) = 1$. Even for this short horizon, there is a clear difference between the open-loop and closed-loop policies.

■

One conclusion of the example is that it is important that the optimal control problems formulated and solved on-line in MPC allow *feedback solutions*. This may be achieved by using a *sequence of control laws* as decision variables rather than a sequence of control actions. The MPC strategies formulated in this way are referred to as *feedback MPC*. One natural question is then if we are back to the search for a full DP solution? The answer is no, since the idea is to restrict attention to "likely" state trajectories over the prediction horizon, given the current x. In contrast, the full DP solution would be valid for all states. There is currently a lot of research in the area of robust control, but we will stop here since the available results go beyond this course.

References

[1] J.B. Rawlings, D.Q. Mayne, and M.M. Diehl. *Model Predictive Control: Theory, Computation, and Design, 2nd edition*. Nob Hill Publishing 2017.
Available online at `https://sites.engineering.ucsb.edu/~jbraw/mpc`

[2] G. Goodwin, M.M. Seron, and J.A. De Don·. *Constrained Control and Estimation*. Springer 2004. Available online via Chalmers Library.

[3] F Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Available online at http://www.mpc.berkeley.edu/mpc-course-material

[4] J. Maciejowski. *Predictive Control with Constraints*. Prentice Hall 2002.

[5] S. Boyd and L. Vandenberghe. *Convex optimisation*. Cambridge University Press 2004.

[6] Diehl, M. *Real-Time Optimization for Large Scale Nonlinear Processes.* PhD thesis, University of Heidelberg, 2001.