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

2024-02-10

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", 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.

Gothenburg, 2024-02-10
Nikolce Murgovski

Contents

1 Introduction and course overview **6**
1.1 Background and motivation . . . 6
1.2 The receding horizon idea . . . 10

2 Review and preliminaries **14**
2.1 State space models . . . 14
2.2 Setpoint tracking and regulation . . . 18
2.3 Estimation, prediction and disturbance modeling . . . 20
2.4 Quadratic forms and stability . . . 24
2.5 LTI systems in Matlab . . . 25

3 Unconstrained receding horizon control **27**
3.1 The linear-quadratic (LQ) control problem . . . 27
3.2 Infinite horizon control . . . 32

4 Constrained receding horizon control **35**
4.1 Constrained receding horizon control . . . 35
4.2 Linear quadratic MPC . . . 38

5 MPC practice – setpoints, disturbances and observers **41**
5.1 Output predictions and the DMC scheme . . . 41
5.2 Steady state targets and zero offsets . . . 46

6 The Kalman filter and moving horizon estimation **51**
6.1 The Kalman filter . . . 51
6.2 Least-squares estimation . . . 53
6.3 Moving horizon estimation . . . 61

7 Optimization basics and convexity **63**
7.1 Introduction . . . 63
7.2 Conditions for optimality – the KKT conditions . . . 65
7.3 Convex optimization . . . 68
7.4 Duality . . . 73

8 Solving QP problems **78**
8.1 Newton's method . . . 78
8.2 Inequality constraints . . . 82

9 Feasibility **87**
9.1 Polyhedra and polytopes . . . 87
9.2 Reachable sets . . . 90
9.3 Recursive (persistent) feasibility . . . 91
9.4 Practical feasibility . . . 99

10 Explicit control laws for constrained linear systems **101**
10.1 Parametric programming . . . 101
10.2 Constrained LQ control . . . 102

11 Stability **108**
11.1 Stability and Lyapunov functions 108
11.2 Stability conditions for MPC 110
11.3 Stability of constrained linear quadratic MPC 114

12 Beyond linear MPC **116**
12.1 Nonlinear MPC 116
12.2 Robust MPC 122

13 MPC practice – implementation and tuning **125**
13.1 Design and tuning 126
13.2 Other variations of MPC 128
13.3 Implementation issues and computational efficiency 129

A Mathematical Background **134**
A.1 Operations with matrices 134
A.2 Properties of matrices 135
A.3 Matrix inversion 136
A.4 Matrix differentiation 138
A.5 Probability theory 140

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
$\mathbb{R}^n$	Euclidean 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
$\dot{x}, \dot{x}(t)$	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
$x(k+i\|k) \sim x(i\|k)$	Predicted state at instant $k+i$, given information at k
$\mathbf{x}(k) = x(0:N\|k)$	Vector of future state variables at instant k
u	Future control input
$\mathbf{u}$	Vector of future control inputs
Δu	Future control move
$\mathbf{\Delta u}$	Vector of future control moves
$\hat{y}$	Predicted output
$\mathbf{y}$	Vector of predicted outputs
y_f	Free response
$\mathbf{y_f}$	Vector of free response values
r	Future reference signal
$\mathbf{r}$	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
$u^*(0\!:\!N-1)$	Optimal sequence of future controls
$u(0\!:\!\infty)$	Infinite sequence of future controls
$u^*(0\!:\!\infty)$	Optimal infinite sequence of future controls
$x(0\!:\!N)$	Sequence of future states
$x^*(0\!:\!N)$	Optimal sequence of future states
	Feasibility related set symbols
$\mathrm{conv}(V)$	Convex hull of the vertices in matrix V
$\mathrm{proj}_x(\mathcal{S})$	Projection of $\mathcal{S}$ onto x
$\mathrm{reach}(\mathcal{S})$	The set reachable from $\mathcal{S}$
$\mathrm{pre}(\mathcal{S})$	The set of predecessor states of $\mathcal{S}$
$\mathcal{P} \oplus \mathcal{Q}$	Minkowski sum of polytopes $\mathcal{P}$ and $\mathcal{Q}$
$\mathcal{P} \ominus \mathcal{Q}$	Pontryagin difference between polytopes $\mathcal{P}$ and $\mathcal{Q}$
$f \circ \mathcal{P}$	Composition (mapping) of function f onto set $\mathcal{P}$
$\mathcal{C}, \mathcal{C}_\infty$	(Maximal) control invariant set
$\mathcal{K}_i(\mathcal{S}), \mathcal{K}_\infty(\mathcal{S})$	The i-step/maximal controllable set with target set $\mathcal{S}$
	State estimation symbols
$\hat{x}$	State estimate
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 optimization symbols
x^+	The value of x at the next iteration
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!

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

Goals of this section:

- To introduce the background of MPC and the ideas behind the course.
- To get motivated by being reminded about the limitations of linear design.
- To understand the receding horizon control (RHC) idea.
- To understand how the course is organized.
- To understand the learning objectives of the course.

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 petrochemical 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 optimize 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 optimization 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 summarized 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 multivariate 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 Section 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,

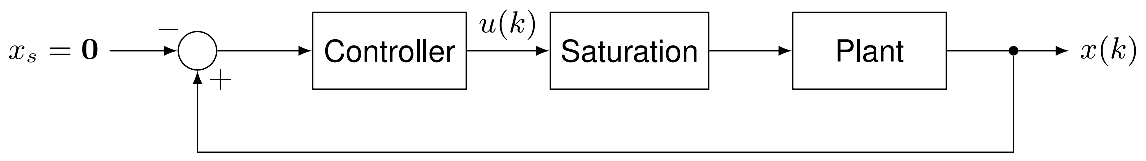


Figure 1: Feedback control loop with input saturation.

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

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

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

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

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

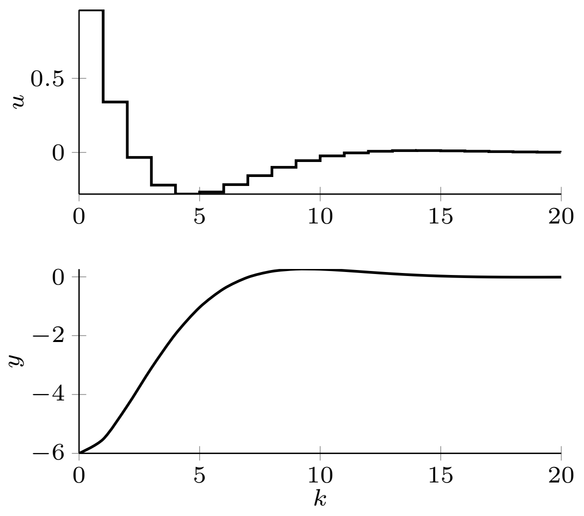
$$\mathrm{sat}(u) \triangleq \begin{cases} 1, & u > 1 \\ u, & |u| \le 1 \\ -1, & u < -1. \end{cases}$$

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

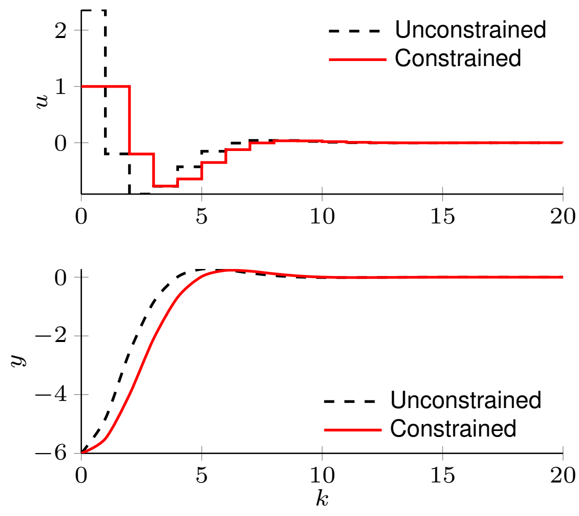
Cautious control. We may design a controller that minimizes a quadratic objective

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

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.

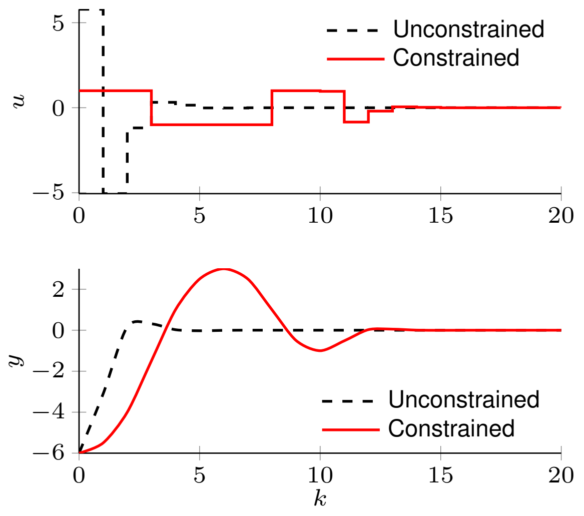


(a) Cautious design with $u(k) = -Kx(k)$ and $R = 20$.

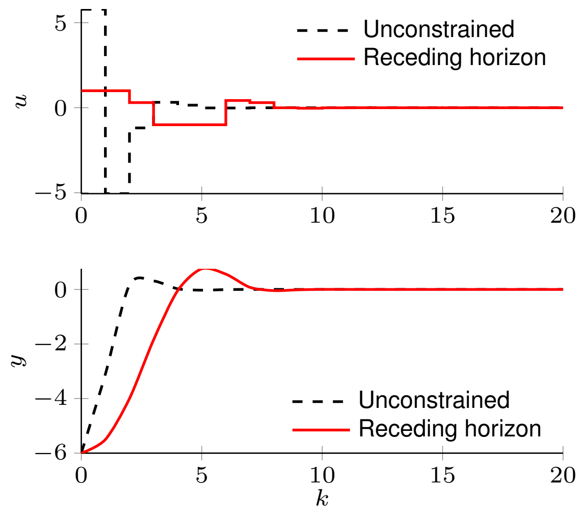


(b) Serendipitous design for constrained and unconstrained LQR with $u(k) = -Kx(k)$ and $u(k) = -\operatorname{sat}(Kx(k))$, respectively, and $R = 2$.

Figure 2: LQ control of the double integrator system.



(a) Serendipitous design for constrained and unconstrained LQR with $u(k) = -Kx(k)$ and $u(k) = -\operatorname{sat}(Kx(k))$, 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.

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.

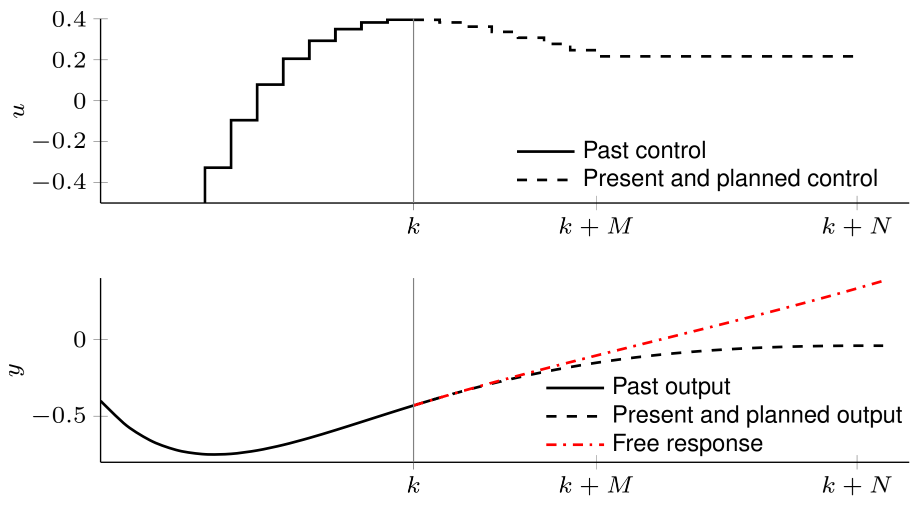


Figure 4: An illustration of a receding horizon control. The free response is the future response when the control signal stays at its current level.

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 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 forecast 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 summarize 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 summarize, 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.
- $r(k)$: a **reference signal** to be followed by the output.

For a **prediction horizon**: $N = 2$, 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 is

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

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,

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

Since there is only one variable to fulfill 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 $||\Theta\Delta u(k|k) - (\mathbf{r} - \mathbf{y_f})||^2$. Expressing this in Matlab notation gives

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

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:

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

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 optimization *objective* and that we have a relevant *optimization algorithm* to minimize 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 behavior.
- An **online optimization 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 previously in Section 1. A review is given of some concepts and results from previous courses that turn out to be useful. The aim of this 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].

Goals of this section:

- To master different discrete-time state space models used for MPC.
- To understand conditions for setpoint tracking and disturbance rejection.
- To refresh basic concepts used to solve systems of linear equations.
- To refresh and learn a new way to test for controllability and observability.
- To know how to use quadratic forms for stability investigations of linear systems.
- To refresh how to handle LTI systems in Matlab.

Learning outcomes:

- Understand and explain the basic principles of model predictive control, its pros and cons, and the challenges met in implementation and applications.
- Use software tools for analysis and synthesis of MPC controllers.

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 linearizing 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. The solution of (4) with initial condition $x(t)$ is

$$x(t+h) = e^{A_c h}x(t) + \int_0^h e^{A_c s}B_c u(s)\mathrm{d}s, \text{ where } e^A = \sum_{k=0}^{\infty}\frac{1}{k!}A^k. \tag{5}$$

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

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

By using this in (5) with $t = kh$, we get the discrete-time state equation

$$x(k+1) = \underbrace{e^{A_c h}}_{A}x(k) + \underbrace{\int_0^h e^{A_c s}\mathrm{d}sB_c}_{B}u(k) = Ax(k) + Bu(k),$$

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

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

Notice that the computation of the matrix B requires performing an integration. Below we propose another method without the need for integration.

Lemma 2.1. *Assuming that the control signal is constant over the sampling interval h, the matrices of the discrete system corresponding to the continuous linear system* (4) *can be computed as*

$$\begin{bmatrix} A & B \\ 0 & I \end{bmatrix} = e^{\begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}h}. \tag{6}$$

In Maltab, the solution can be obtained by calling the matrix exponential function `expm`.

Proof. Consider, without loss of generality, that the continuous system is currently at time 0. During one sampling interval h, a constant control action $\bar{u}$ is applied. Now rewrite the continuous system (4) as an autonomous system by creating an extended state vector

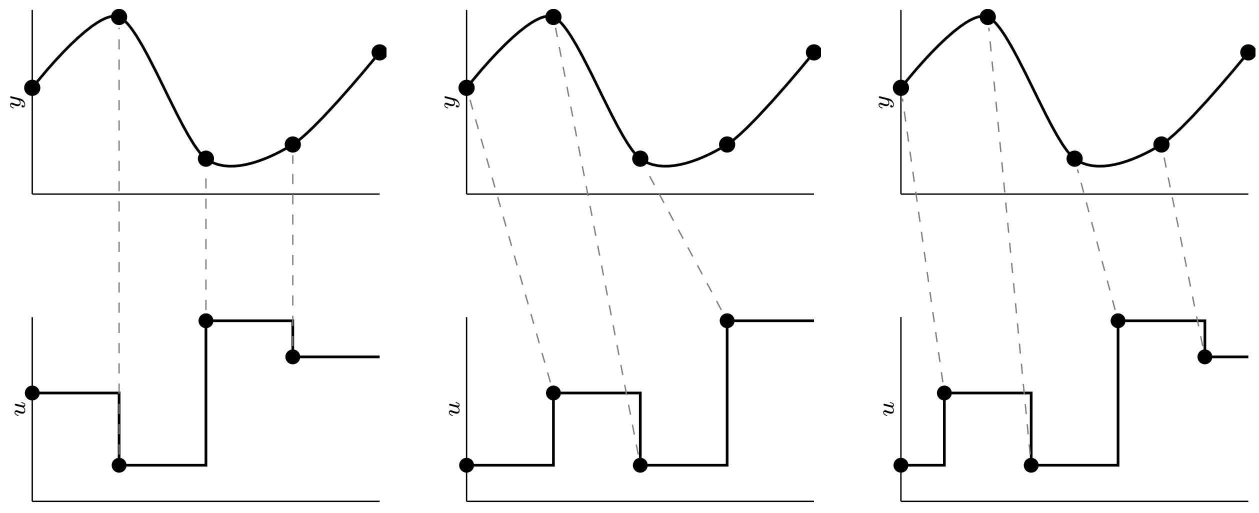
$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x(t) \\ \bar{u} \end{bmatrix} = \begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x(0) \\ \bar{u} \end{bmatrix}.$$

The solution of the autonomous system at time $t = h$ is

$$\begin{bmatrix} x(h) \\ \bar{u} \end{bmatrix} = e^{\begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}h}\begin{bmatrix} x(0) \\ \bar{u} \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}\begin{bmatrix} x(0) \\ \bar{u} \end{bmatrix}.$$

Comparing the matrices multiplying the vector $\begin{bmatrix} x(0) & \bar{u} \end{bmatrix}^\top$ proves Lemma 2.1. □

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.



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

Figure 5: Control action and computational delay.

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

$$\begin{aligned} x(k+1) &= e^{A_c(h-\tau)} x(kh+\tau) + \int_0^{h-\tau} e^{A_c s} B_c \, ds \cdot u(k) \\ &= e^{A_c(h-\tau)} \left(e^{A_c \tau} x(kh) + \int_0^{\tau} e^{A_c s} B_c \, ds \cdot u(k-1) \right) + \int_0^{h-\tau} e^{A_c s} B_c \, ds \cdot u(k) \\ &= e^{A_c h} x(kh) + \underbrace{e^{A_c(h-\tau)} \int_0^{\tau} e^{A_c s} B_c \, ds}_{B_1} \cdot u(k-1) + \underbrace{\int_0^{h-\tau} e^{A_c s} B_c \, ds}_{B_2} \cdot u(k) \\ &= A x(k) + B_1 u(k-1) + B_2 u(k). \end{aligned}$$

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

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

Similarly as with the system without delay, it is possible to compute the discrete system matrices without performing integration.

Lemma 2.2. *Assuming that the control signal is piece-wise constant, the matrices of the discrete system corresponding to the continuous linear system* (4) *with sampling interval h and delay τ, can be computed as*

$$A = A_{h\tau} A_\tau, \quad B_1 = A_{h\tau} B_\tau, \quad B_2 = B_{h\tau}$$

where

$$\begin{bmatrix} A_\tau & B_\tau \\ 0 & I \end{bmatrix} = e^{\begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}\tau}, \quad \begin{bmatrix} A_{h\tau} & B_{h\tau} \\ 0 & I \end{bmatrix} = e^{\begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}(h-\tau)}.$$

The proof is very similar to the one of Lemma 2.1, but with the state update performed in two stages, from 0 to τ and τ to h. A detailed proof is left as an exercise for the reader.

Other discretization strategies

Keeping a constant control action during one sampling interval is a common strategy in receding horizon control. This strategy is also known as *zero-order hold* (ZOH). It is possible to have other assumptions on the control input. One assumption is to assume that the control input is piece-wise linear,

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

which is also known as first-order hold (FOH). Notice that the control input to be determined at each sample is $v(kh)$, while $u(t)$ is dynamically dependent on the history of chosen values for $v(kh)$. Since $\dot{u}(t) = v(kh)$, it is possible to transform the system (4) by augmenting the state vector as

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} = \begin{bmatrix} A_c & B_c \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} v(t).$$

A discrete version of this system may now be obtained with the typical ZOH strategy, where $\begin{bmatrix} x(kh) & u(kh) \end{bmatrix}^\top$ is the state vector and $v(kh)$ is the piece-wise constant control input.

It is also possible to parameterize the control signal,

$$u(k+i|k) = \sum_{j=1}^{n_u} c_j u_j(i),$$

with *basis functions* $\{u_j(\cdot)\}$, e.g. polynomials. Once the coefficients $\{c_j\}$ have been determined (in the optimization step of the MPC), the control signal is defined over the whole prediction horizon. This is called *Predictive Functional Control.* For a detailed discussion on Predictive Functional Control and other discretization schemes, readers are referred to [1, pp. 501-511].

State space model with incremental control

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**,

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

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

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

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

$$x(k+1) = Ax(k) + Bu(k) \tag{7}$$
$$y(k) = Cx(k). \tag{8}$$

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

$$\begin{bmatrix} I-A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ y_{sp} \end{bmatrix}. \tag{9}$$

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 (9) 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 (9), 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:

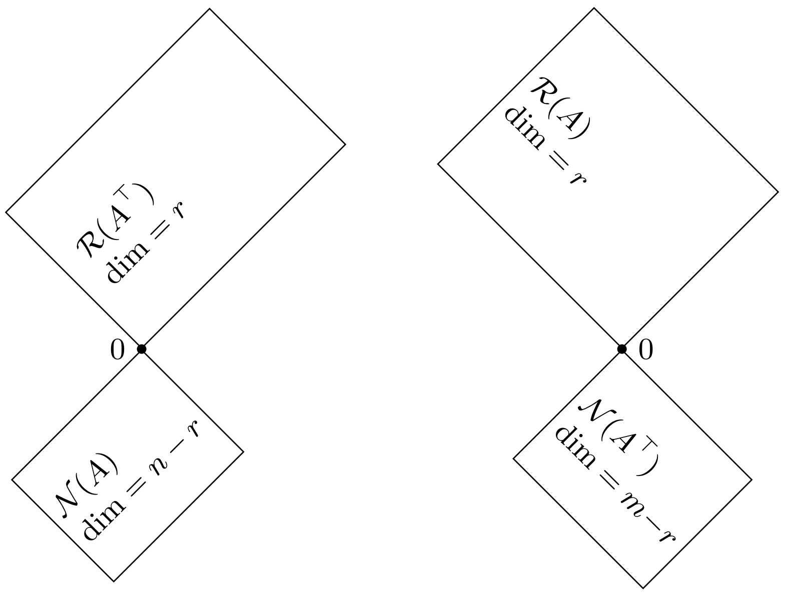


Figure 6: Illustration of range and nullspace of a matrix A and its transpose.

- 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.

Solution to over-determined linear system

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 *over-determined* system of linear equations

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

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

$$\min_{x\in\mathbb{R}^n} |Ax - b|_2^2$$

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 2.1. *$Ax^* = AA^\dagger b$ 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, $x^* = A\backslash b$.*

Solution to underdetermined linear system

In other situations we may encounter underdetermined problems that have many solutions. This happens when there are less equations than unknowns, i.e. $m < n$. Consider, e.g., the *underdetermined* 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 minimizes the least-norm

$$\min_{x \in \mathbb{R}^n,\, Ax=b} x^\top x. \tag{10}$$

Proposition 2.1. *The solution to* (10)*, where A is a full row rank, is*

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

Remark 2.2. *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 (10) can be found by minimizing 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 (9), 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

$$\begin{bmatrix} I - A & -B \\ C_z & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ z_{sp} \end{bmatrix} \tag{11}$$

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

2.3 Estimation, prediction and disturbance modeling

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

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

allows the construction of a *state observer/estimator*

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

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

$$\tilde{x}(k+1) = (A - LC)\tilde{x}(k).$$

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*

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

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

The measured outputs can be related to the current state $x(0)$ as

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{N-1} \end{bmatrix} x(0).$$

The states can be observed if the matrix to the right has n linearly independent columns. Moreover, according to the Cayley-Hamilton theorem [1], the matrix powers A^k for $k \geq n$, can be expressed as a linear combination of the powers from 0 to n. Hence, if we cannot observe x in n measurements, we cannot observe it any number of measurements.

Lemma 2.3 (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 2.3. *A weaker condition is* ***detectability****, which requires that any unobservable modes are strictly stable.*

Example 2.1. [Simple observable system] An example of a simple observable system is

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= x_2(k)/2 \\ y(k) &= x_1(k). \end{aligned}$$

Conclusions on observability can be made without the need to relate to some of the lemmas above. Namely, $x_1(k)$ can directly be observed from the current measurement $y(k)$, while $x_2(k)$ is observed through $x_1(k+1)$, i.e. after the measurement of $y(k+1)$. Hence, the system is observable. ■

Example 2.2. [Unobservable, but detectable system] Consider the system

$$\begin{aligned} x_1(k+1) &= x_1(k)/2 \\ x_2(k+1) &= 2x_2(k) \\ y(k) &= x_2(k). \end{aligned}$$

Here, $x_2(k)$ can directly be observed from the current measurement $y(k)$, but x_1 cannot be observed in any number of measurements. However, x_1 is stable, i.e. it approaches 0 asymptotically. Hence, the system is not observable, but it is detectable. ■

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

Definition 2.2 (Controllability)**.** *A linear, discrete time system*

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

is controllable if it is possible to steer the system from any state $x(0)$ to any state $x(k_f)$ 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 2.4. *A weaker condition is **stabilizability** which requires that any uncontrollable modes are strictly stable.*

More specifically, a linear discrete time system $x(k+1) = Ax(k) + Bu(k)$ is controllable if there exists a sequence of inputs $\{u(0), u(1), \dots, u(N-1)\}$ that can transfer the system to a certain (steady) state x_s in a finite time N. The steady state can then be written as

$$x_s = A^N x(0) + \begin{bmatrix} B & AB & \cdots & A^{N-1}B \end{bmatrix} \begin{bmatrix} u(N-1) \\ u(N-2) \\ \vdots \\ u_0 \end{bmatrix}.$$

The equality can be solved for the control inputs if the matrix including A and B has a full rank n. Moreover, according to the Cayley-Hamilton theorem [1], the matrix powers A^k for $k \geq n$, can be expressed as a linear combination of the powers from 0 to n. Hence, if we cannot reach x_s in n moves, we cannot reach it any number of moves.

Example 2.3. [Simple controllable system] An example of a simple controllable system is

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= x_2(k)/2 + u(k). \end{aligned}$$

It is clear that x_2 can directly be controlled from u, while x_1 is indirectly controlled through x_2, i.e. with a one sample delay from u. ■

Example 2.4. [Not controllable, but stabilizable] An example of not controllable, but stabilizable system is

$$\begin{aligned} x_1(k+1) &= x_2(k) + u(k) \\ x_2(k+1) &= x_2(k)/2. \end{aligned}$$

Here, the state x_2 cannot be controlled, but only x_1 can be controlled. However, x_2 is stable, i.e. it approaches 0 asymptotically. Hence, the system is not controllable, but stabilizable. ■

Example 2.5. [Parallel connection of two controllable 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

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u(k), \quad y = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}.$$

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.

■

Constant disturbance modeling

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 modeled by the state equation

$$d(k+1) = d(k).$$

Augmenting the state model (7) with this disturbance gives the model

$$\begin{aligned} \begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} \end{aligned} \tag{12}$$

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* (12) *is detectable if and only if the original system is detectable and the following condition holds:*

$$\operatorname{rank}\left(\begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} \right) = n + n_d \tag{13}$$

Remark 2.5. *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.

More specifically, consider a steady-state scenario described by

$$\begin{aligned} x_s &= Ax_s + Bu_s + B_d d \\ y_s &= Cx_s + C_d d = y_{sp}. \end{aligned}$$

In a compact form, this can be written as

$$\begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} \begin{bmatrix} x_s \\ d \end{bmatrix} = \begin{bmatrix} Bu_s \\ y_{sp} \end{bmatrix}.$$

This has a unique solution for x_s and d, in the least square sense, if the matrix to the left has a greater than or equal number of independent rows than columns. Number of rows is $n + p$ and number of columns is $n + n_d$. Hence, $n_d \leq p$.

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

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

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

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

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

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

Definition 2.3 (Stability). *The linear, discrete time system*

$$x(k+1) = Ax(k) \tag{14}$$

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

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

which can be interpreted as a generalization of the Euclidean norm $|x|$ (squared). Evaluated along solutions to (14), V changes according to

$$V(x(k+1)) - V(x(k)) = -x(k)^\top (S - A^\top SA)x(k) \equiv -x(k)^\top Qx(k).$$

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

Lemma 2.5 (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*

$$S - A^\top S A = Q. \tag{15}$$

Remark 2.6. *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 $x^\top(i)Qx(i) \to 0,\ i \to \infty$.

2.5 LTI systems in Matlab

It is useful to review and refresh some of the basics concerning how to represent, manipulate and analyze 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 Section 4 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 Section 4 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].

Goals of this section:

- To master the formulation of linear quadratic control (LQ).
- To understand how dynamic programming can be used to solve the LQ problem.
- To formulate and solve the finite-time LQ problem using the "batch approach".
- To formulate an unconstrained receding horizon control based on LQ.

Learning outcomes:

- Correctly state, in mathematical form, MPC formulations based on descriptions of control problems expressed in application terms.
- Describe and construct MPC controllers based on a linear model, quadratic costs and linear constraints.
- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples.

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:

$$S : x(k+i+1|k) = Ax(k+i|k) + Bu(k+i|k), \quad i = 0, \dots, N-1 \tag{16}$$

or, shortly, $x(i+1) = Ax(i) + Bu(i)$, where the dependence on k is not shown, for brevity.

[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.

Optimization 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 (16).

The objective, or criterion, or cost function:

$$\begin{aligned} V_N(x(0), u(0:N-1)) &= x^\top(N) P_f x(N) + \sum_{i=0}^{N-1} (x^\top(i) Q x(i) + u^\top(i) R u(i)) \\ &= l_f(x(N)) + \sum_{i=0}^{N-1} l(x(i), u(i)). \end{aligned} \tag{17}$$

Remark 3.1. *All $x(i)$ are functions of $x(0)$ and $u(0:N-1)$ via the model (16)!*

Remark 3.2. *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.

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 (16)". In fact, in the search for the optimizing 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 optimization variables, and the state model (16) will then act as an *equality constraint* in the slightly modified formulation:

$$\begin{aligned} \min_{u(0:N-1), x(1:N)} \quad & V_N(x(0), u(0:N-1), x(1:N)) \\ \text{subject to} \quad & x(i+1) = Ax(i) + Bu(i). \end{aligned} \tag{18}$$

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 optimization formulation

$$\min_{u(0:N-1)} \quad V_N(x(0), u(0:N-1)), \tag{19}$$

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 (16) gives

$$\underbrace{\begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N) \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{\Omega} x(0) + \underbrace{\begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix}}_{\Gamma} \underbrace{\begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(N-1) \end{bmatrix}}_{\mathbf{u}} \tag{20}$$

or, with a more compact notation,

$$\mathbf{x} = \Omega x(0) + \Gamma \mathbf{u}. \tag{21}$$

The LQ criterion (17) can now be written

$$\begin{aligned} V_N(x(0), \mathbf{u}) &= x^\top(0) Q x(0) + \mathbf{x}^\top \bar{Q} \mathbf{x} + \mathbf{u}^\top \bar{R} \mathbf{u} \\ &= x^\top(0) Q x(0) + (\Omega x(0) + \Gamma \mathbf{u})^\top \bar{Q} (\Omega x(0) + \Gamma \mathbf{u}) + \mathbf{u}^\top \bar{R} \mathbf{u} \\ &= \mathbf{u}^\top (\Gamma^\top \bar{Q} \Gamma + \bar{R}) \mathbf{u} + 2 x^\top(0) \Omega^\top \bar{Q} \Gamma \mathbf{u} + x^\top(0)(Q + \Omega^\top \bar{Q} \Omega) x(0) \end{aligned} \tag{22}$$

where $\bar{Q} = \mathrm{diag}(Q, \ldots, Q, P_f)$ and $\bar{R} = \mathrm{diag}(R, \ldots, R)$.

Since $V_N(x(0), \mathbf{u})$ is quadratic in $\mathbf{u}$, we can solve for the optimal control vector (either by differentiation or by completing the squares):

$$\mathbf{u}^*(k) = u(0 : N-1|k) = \underbrace{-(\Gamma^\top \bar{Q} \Gamma + \bar{R})^{-1} \Gamma^\top \bar{Q} \Omega}_{K_b} x(k)$$

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

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

where the full dependence on k can be seen and $x(k) = x(k|k)$. Note that $\mathbf{u}^*(k)$ is linear in $x(k)$ and V_N^* is quadratic in $x(k)$!

We finish this section by noting that the formulation (19) sometimes is referred to as the **condensed** version of the problem, since the number of optimization variables has been reduced relative to the **uncondensed** version (18). In both cases, the result of the optimization 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 optimization 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 $x(0\!:\!N) = \{x(0), x(1), \ldots, x(N)\}$, where we have used the short notation, e.g. $x(i)$ for $x(k+i|k)$. 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:

$$V_N = l(x(0), u(0)) + \ldots + \overbrace{l(x(N-2), u(N-2)) + \overbrace{l(x(N-1), u(N-1)) + l_f(x(N))}^{u(N-1)}}^{u(N-2)}. \tag{23}$$

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 (23) 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 (23) are quadratic in $u(N-1)$. We can rewrite this expression into one quadratic form by using the system model (16) 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

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

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

$$\begin{aligned}
V^*_{N-1\to N}(x) &= x^\top(N-1)\underbrace{(Q + A^\top P_f A - A^\top P_f B(R+B^\top P_f B)^{-1}B^\top P_f A)}_{P(N-1)}\, x(N-1) \\
&\equiv x^\top(N-1)P(N-1)x(N-1).
\end{aligned} \tag{24}$$

We have thus obtained the optimal last control input $u^*(N-1)$ as a linear feedback from the state $x(N-1)$ 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 (23) is given as an explicit function of $x(N-1)$ in (24), we now proceed to include one more term from (23), 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

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

and the optimal cost-to-go from state $x(N-2)$ at time $N-2$ to the final time N is

$$V^*_{N-2\to N}(x) = x^\top(N-2)P(N-2)x(N-2), \tag{25}$$

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. The sequence of optimal control laws (**control policy**) is computed as:

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

where the **Riccati equation** is

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

or equivalently

$$P(k-1) = Q + A^\top P(k)A + A^\top P(k)BK(k-1), \quad P(N) = P_f. \tag{27}$$

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 Section 1. We turn to this in Section 4.

Example 3.1. [Batch vs. DP solution, [1, pp. 195–199]] Consider the simple integrator system

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

We will study the finite-time optimal control problem with $x(0) = x(0|0) = 1$, $N = 3$ and $Q = R = P_f = 1$.

Using the batch approach, the optimal control sequence can be found as

$$u^*(0:2|0) = K_b(k)x(0), \quad K_b = -\begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = -\begin{bmatrix} 0.615 & 0.231 & 0.077 \end{bmatrix}^\top.$$

It follows,

$$\begin{aligned} u^*(0:2|0) &= -\begin{bmatrix} 0.615 & 0.231 & 0.077 \end{bmatrix}^\top \\ x^*(0:3|0) &= \begin{bmatrix} 1 & 0.385 & 0.154 & 0.077 \end{bmatrix}^\top. \end{aligned}$$

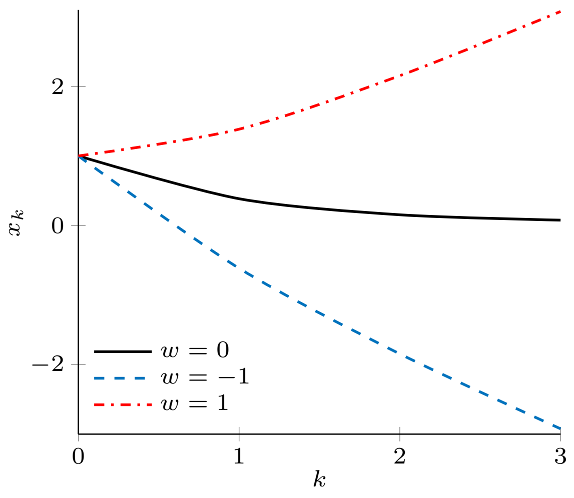
Using the DP approach, the solution can be obtained as $u^*(k) = K_d(k)x(k)$, where the optimal control gain can be found via backward recursion,

$$K_d = -\begin{bmatrix} 0.615 & 0.6 & 0.5 \end{bmatrix}^\top.$$

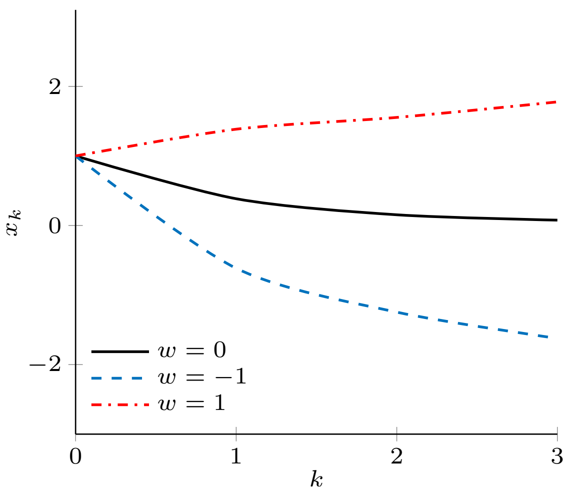
The optimal control sequence can now be obtained by simulating the system forward in time,

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

■



(a) Open-loop control.



(b) Closed-loop, feedback control.

Figure 7: Receding horizon control of an uncertain system with an additive process disturbance $w \in [-1, 1]$.

Clearly, and as expected, the example shows that 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 computes the entire optimal control sequence already at the first MPC update, at $k = 0$. The DP solution, however, computes only one control input at one MPC update, i.e. $u(k|k) = u(k)$ is computed when state $x(k) = x(k|k)$ is measured. Hence, to compute the entire control sequence, three MPC updates need to be performed. The solution provided by the DP approach is a closed-loop control policy, since the optimal control taken at a given time instant is an explicit function of the state measured at that instant. This will be discussed further in Section 10.

The way we computed the optimal control sequence with the batch approach is an open-loop control, since the controller does not require measurement of intermediate states, except the initial state at the given MPC update. However, the loop can be closed implicitly, by simply repeating the same calculation at every instant. This may sound like a silly procedure, since it will clearly give the same result, despite the increased amount of computation. However, it becomes important when the system is subject to disturbances.

Example 3.2. [Open loop vs. feedback solution under additive process disturbance] Consider the same system as in Example 3.1, but now subject to process disturbance $w \in [-1, 1]$,

$$x(k + 1) = x(k) + u(k) + w(k).$$

The solution from the batch approach is applied in open loop, using the state measurement at instant zero.

Results are shown 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 when disturbance is not zero. See Figure 7a for the open-loop control and Figure 7b for the closed-loop control. In practice, the batch solution is re-updated at every instant, in which case the loop is closed implicitly and the batch and DP solution would be identical as those in Figure 7b. Further information on this example can be found in [1, p. 195]. ■

3.2 Infinite 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 optimization problem over a finite horizon, but only using the first element of the optimal control sequence and letting the optimization horizon recede with time.

Using the result from the DP recursion above and initializing 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 $u^*(k) = K(k)x(k)$. Alternatively, the receding horizon controller can be obtained from the first element of the optimal batch solution, i.e.

$$u(k) = \begin{bmatrix} I_m & 0_m & \cdots & 0_m \end{bmatrix} \mathbf{u}^* = -\begin{bmatrix} I_m & 0_m & \cdots & 0_m \end{bmatrix} (\Gamma^\top \bar{Q} \Gamma + \bar{R})^{-1} \Gamma^\top \bar{Q} \Omega x(k). \tag{28}$$

Regardless of which version of the receding horizon controller we choose—the two solutions are of course identical (in the absence of disturbances)—we notice that the control law obtained is *linear*.

Example 3.3. [Optimality does not guarantee stability [1]] Consider the system

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

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 (26) 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 optimization horizon to *infinity* (which means that it is really not any longer a receding horizon controller).

The system

$$S: \quad x(k+1) = Ax(k) + Bu(k) \tag{29}$$

with optimization criterion

$$V_\infty(x(0), u(0\!:\!\infty)) = \sum_{i=0}^{\infty} (x^\top(i) Q x(i) + u^\top(i) R u(i)) \tag{30}$$

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

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

with a time invariant feedback gain K, given by

$$K = -(B^\top P B + R)^{-1} B^\top P A \tag{31}$$

$$P = Q + A^\top P A - A^\top P B (B^\top P B + R)^{-1} B^\top P A. \tag{32}$$

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

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

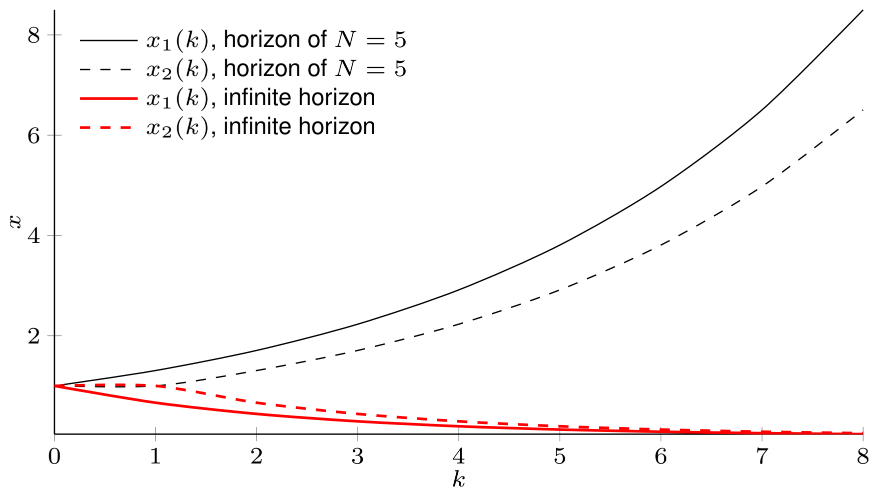


Figure 8: State evolution from the two optimal controllers, one with horizon of $N = 5$ and the other with an infinite horizon.

Example 3.4. [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.

In this particular example, the value function $x_0^\top P(0)x_0$ for predicted trajectories within a short horizon is initially lower than that of the infinite horizon, $x_0^\top P_\infty x_0$.

■

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 Section 3 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 optimization problem is found in Section 3.2.

Goals of this section:

- To understand the principles behind and to formulate a constrained receding horizon controller (RHC).
- To formulate an MPC based on linear models and quadratic criteria.

Learning outcomes:

- Describe and construct MPC controllers based on linear model, quadratic costs and linear constraints.
- Describe basic properties of MPC controllers and analyze algorithmic details on simple examples.

In Section 3, 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

$$x(k+1) = f(x(k), u(k))$$

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, the dependence on k is not shown, for brevity. 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

$$\begin{aligned} \min_{u(0:\infty)} \quad & V_\infty(x_0, u(0\!:\!\infty)) && (33)\\ \text{subject to} \quad & x(i+1) = f(x(i), u(i)), \quad x(0) = x_0 && (34)\\ & x(i) \in \mathbb{X}, \quad u(i) \in \mathbb{U}, \quad \text{for all } i \in (0, \infty). && (35) \end{aligned}$$

Here, we have added the requirement in (35) 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 Section 3, we used dynamic programming to solve the finite horizon optimization problem recursively, basically relying on the repeated use of the Bellman's equation. Recall that the dynamic programming recursion

$$\min_{u(0:N-1)} V_N(x(0), u(0:N-1)) = \min_{u(0)} \left\{ l(x(0), u(0)) + \min_{u(1:N-1)} V_{1\to N}(x(1), u(1:N-1) \right\}, \quad (36)$$

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

$$V_\infty^*(x(0)) = \min_u \left\{ l(x(0), u) + V_\infty^*(f(x(0), u)) \right\}. \quad (37)$$

However, adding constraints in dynamic programming may lead to intractable problems, and the numerical strategy often adopted requires quantizing the state-space, which introduces approximations in the problem and suboptimality in the obtained solution. We will not pursue this further, but we will resort to solving the optimization problem using a constrained batch approach within a receding horizon framework.

Finite horizon optimal control

As said earlier, a method for computing the solution of this equation can be defined, but obtaining the solution 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, 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 within a receding horizon. In the purely deterministic case without disturbances, 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

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

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

$$\begin{aligned} \min_{u(0:N-1)} \quad & V_N(x_0, u(0:N-1)) && (38)\\ \text{subject to} \quad & x(i+1) = f(x(i), u(i)), \quad x(0) = x_0 && (39)\\ & x(i) \in \mathbb{X}, \quad u(i) \in \mathbb{U}, \quad \text{for all } i = 0, \dots, N-1 && (40)\\ & x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. && (41) \end{aligned}$$

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:

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

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 optimization problem above is meaningful, i.e. possible to solve. This is referred to as *feasibility* of the optimization problem.

Feasibility

In the optimization 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 *implicit* set of allowed control sequences in general will depend on the initial state $x(0) = x_0$. Formally, we can express the implicit constraint on the control sequences as

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

where

$$\mathcal{U}_N(x_0) = \{u(0 : N-1) \mid x(i+1) = f(x(i), u(i)), x(0) = x_0, i = 0, \ldots, N-1, \\ u(i) \in \mathbb{U}, x(i) \in \mathbb{X}, x(N) \in \mathbb{X}_f\}$$

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 $i = 0, \ldots, N-1$. For some x_0, it may even be impossible to find *any* control sequence that fulfills the constraints. The optimization 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 9.

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 optimization 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 $x(k) = x(k|k)$ as the initial state vector, we will now change the notation appropriately.

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

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

for the optimal control sequence

$$u^*(0\!:\!N-1|k) = \{u^*(k|k), u^*(k+1|k), \dots, u^*(k+N-1|k)\}.$$

2. Apply the first control input
$$u(k) = u^*(k|k).$$
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

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

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,

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

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

$$u_{\min} \leq u(k) \leq u_{\max}, \quad \text{for all } k \geq 0 \tag{42}$$

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

$$x_{\min} \leq x(k) \leq x_{\max}, \quad \text{for all } k \geq 0. \tag{43}$$

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

$$\Delta u_{\min} \leq u(k) - u(k-1) \leq \Delta u_{\max}, \quad \text{for all } k \geq 0. \tag{44}$$

Such constraints can easily be put in a form that generalizes the inequalities (42) and (43) by introducing a new state vector

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

for which the augmented system model becomes

$$\xi(k+1) = \mathcal{A}\xi + \mathcal{B}\Delta u$$

with appropriately defined matrices (see Section 2). The constraint (44) can then be stated as

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

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

$$F\mathbf{u} + G\mathbf{x} \leq h \tag{45}$$

with $\mathbf{u}$, $\mathbf{x}$ defined in (21), 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 optimization problem

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

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

   for the optimal control sequence

$$u^*(0:N-1|k) = \{u^*(k|k), u^*(k+1|k), \dots, u^*(k+N-1|k)\}.$$

2. Apply the first control input
$$u(k) = u^*(k|k).$$

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

Linear quadratic MPC with vector notation: condensed form

In this formulation, the optimization variables are the sequence of control inputs $u(0:N-1)$ or, equivalently, the vector $\mathbf{u}$. 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 optimization problem can be given an equivalent formulation using the vector notation for the control and state sequences, i.e. $\mathbf{u}$ and $\mathbf{x}$ defined in (21). The quadratic criterion can then be expressed as in (22). Assuming all the constraint sets $\mathbb{X}, \mathbb{U}, \mathbb{X}_f$ are defined by affine inequalities, we can also use (45) to encode all the constraints. The equivalent formulation of our LQ MPC is thus: (Recall the batch solution for the definition of the matrices.)

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

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

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

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

Linear quadratic MPC: uncondensed (lifted) formulation

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 optimization. In doing so, we may consider all future states *and* controls as optimization 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 optimization problem

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

$$V_N(x, u(0:N-1), x(0:N)) = x^\top(N)P_f x(N) + \sum_{i=0}^{N-1} \big(x^\top(i)Qx(i) + u^\top(i)Ru(i)\big)$$

for the optimal control and state sequences

$$\begin{aligned} u^*(0:N-1|k) &= \{u^*(k|k), u^*(k+1|k), \dots, u^*(k+N-1|k)\} \\ x^*(0:N|k) &= \{x^*(k|k), x^*(k+1|k), \dots, x^*(k+N|k)\} \end{aligned}$$

and subject to

$$x(i+1) = Ax(i) + Bu(i), \quad x(0) = x \tag{46}$$
$$x(i) \in \mathbb{X} \quad u(i) \in \mathbb{U} \quad \text{for all } i = 0, \dots, N-1 \tag{47}$$
$$x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. \tag{48}$$

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

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

Remark 4.1. *The initial state $x(0)$ may or may not be included in the vector of optimization 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 optimization problem

$$\begin{aligned} \underset{\mathbf{u},\mathbf{x}}{\text{minimize}} \quad & V_N(x, \mathbf{u}, \mathbf{x}) = x^\top Qx + \mathbf{x}^\top \bar{Q}\mathbf{x} + \mathbf{u}^\top \bar{R}\mathbf{u} \\ \text{subject to} \quad & x(k+1) = Ax(k) + Bu(k) \\ & F\mathbf{u} + G\mathbf{x} \le h. \end{aligned}$$

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

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

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

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.

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 Section 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*.

Goals of this section:

- To understand how the state model is used for open loop prediction
- To formulate the DMC scheme with a constant output disturbance
- To interpret the DMC scheme in terms of state observers
- To formulate an MPC controller including setpoints and steady state targets
- To state conditions for offset-free control

Learning outcomes:

- Correctly state, in mathematical form, MPC formulations based on descriptions of control problems expressed in application terms
- Describe and construct MPC controllers based on a linear model, quadratic costs and linear constraints
- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples

5.1 Output predictions and the DMC scheme

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

$$x(k+1) = Ax(k) + Bu(k) \tag{49}$$

$$y(k) = Cx(k). \tag{50}$$

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 (49) 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 (20); 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

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

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 (20), repeated here for convenience.

The state model

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

gives the state predictions

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{x}(k+2|k) \\ \vdots \\ \hat{x}(k+N|k) \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}. \tag{51}$$

In effect, what equation (51) 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

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix} \hat{x}(k|k) + \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-1}B & CA^{N-2}B & \cdots & CB \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}. \tag{52}$$

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 (52), this gives

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

$$+ \underbrace{\begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB + CB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{N-1} CA^iB & \sum_{i=0}^{N-2} CA^iB & \cdots & CB \end{bmatrix}}_{\Theta} \underbrace{\begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+N-1|k) \end{bmatrix}}_{\mathbf{\Delta u}(k)} \tag{53}$$

or

$$\mathbf{y}(k) = \mathbf{y_f}(k) + \Theta \mathbf{\Delta u}(k). \tag{54}$$

Here we have used the notation $\mathbf{y_f}$ (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.

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 (54) is rewritten as

$$\mathbf{y}_{\mathcal{M}}(k) = \mathbf{y_f}(k) + \Theta \mathbf{\Delta u}(k), \tag{55}$$

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

[2]The notation $\hat{x}(k|k)$ may seem a bit odd here, since the estimate is not using any measurement 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.

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 $\mathbf{y}_{\mathcal{P}}$ is given by

$$\mathbf{y}_{\mathcal{P}}(k) = \mathbf{y}_{\mathbf{f}}(k) + \Theta\mathbf{\Delta u}(k) + d \cdot \mathbf{1}, \tag{56}$$

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.

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

where we have used the notation $\mathbf{r}$ 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 (55) result in

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

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

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. In this section, we will give an interpretation of the modification in terms of observer design, and this naturally opens up for generalizations.

To put things into focus and get an understand of the relationships between inputs, measurements and estimates, we will first consider a simple example of state estimation of a scalar system.

Example 5.2. [State estimation of a scalar system] Consider a scalar system:

$$\begin{aligned} x(k+1) &= ax(k) + bu(k) \\ y(k) &= cx(k). \end{aligned}$$

We want to estimate $\hat{x}(k+1|k)$, given noisy measurements $y(0\!:\!k)$. We will do this in two steps. Let us first estimate $\hat{x}(k|k)$ assuming steady-state. The best state estimate can be obtained as the expected value from all measurements, i.e.

$$\begin{aligned} \hat{x}(k|k) &= \frac{1}{(k+1)c}\sum_{i=0}^{k} y(i) = \frac{1}{(k+1)c}\left(y(k) + kc\frac{1}{kc}\sum_{i=0}^{k-1} y(i)\right) \\ &= \frac{1}{c}\underbrace{\frac{1}{k+1}}_{l}\left(y(k) + kc\hat{x}(k|k-1)\right) = l\frac{y(k)}{c} + (1-l)\hat{x}(k|k-1). \end{aligned}$$

The term $l = 1/(k+1) \in (0,1]$ weighs the importance between measurement and estimate uncertainty.

Alternatively, $\hat{x}(k|k)$ can be expressed using the difference between the measurement and model estimate

$$\hat{x}(k|k) = \hat{x}(k|k-1) + \frac{l}{c}\left(y(k) - c\,\hat{x}(k|k-1)\right)$$

which is most commonly found in literature. The estimate of $\hat{x}(k+1|k)$ can now be obtained by simulating the system forward, i.e.

$$\hat{x}(k+1|k) = a\hat{x}(k|k) + bu(k) = a\hat{x}(k|k-1) + bu(k) + a\frac{l}{c}\left(y(k) - c\,\hat{x}(k|k-1)\right).$$

■

By just following expectations, the scalar example showed a relationship between the estimate $\hat{x}(k+1|k)$, the control input $u(k)$, the measurements $y(k)$ and the previous estimate $\hat{x}(k|k-1)$. A similar relationship can be extended to general linear discrete time system, which we may now introduce in order to design a generalized state observer.

The observer consists of several elements:

System model:

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

Observer update:

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

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 a gain matrix L exists such that $(A - LC)$ is stable.

Example 5.3. [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

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

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 Section 6, 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 5.3 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 9.

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 fulfill the condition for *setpoint tracking* (repeated from equation (9))

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ y_{sp} \end{bmatrix}, \tag{57}$$

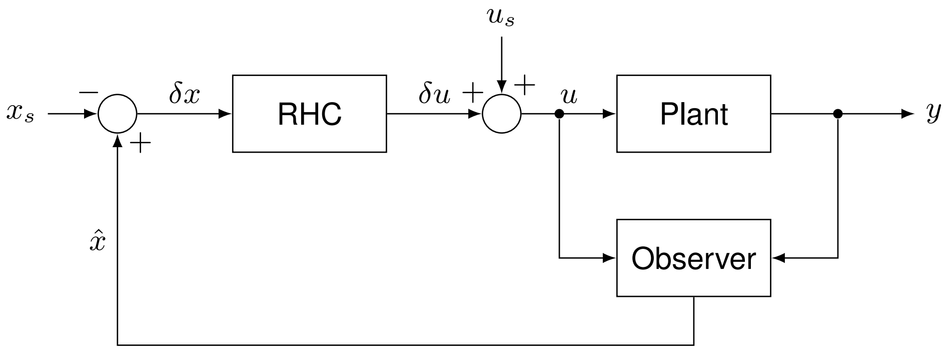


Figure 9: MPC overall structure with a receding horizon controller (RHC) working on deviation variables.

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 (57) 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 (12), again repeated here,

$$\begin{aligned}\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix}\end{aligned} \tag{58}$$

where the disturbance affects both state and output equations through the matrices B_d and C_d. The consequence of augmenting the state vector 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

$$\operatorname{rank} \begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} = n + n_d, \tag{59}$$

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 (57) for setpoint tracking will now have to be modified accordingly.

Equal number of inputs and outputs

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

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_d \hat{d} \\ y_{sp} - C_d \hat{d} \end{bmatrix}. \tag{60}$$

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

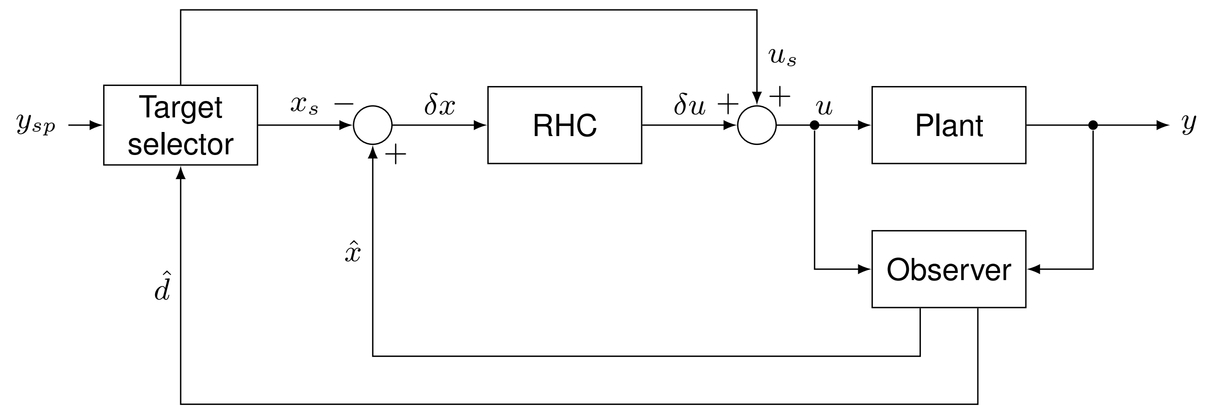


Figure 10: MPC block diagram with a receding horizon controller (RHC) working on deviation variables.

$\delta x(k) = \hat{x}(k) - x_s$ 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 MPC at each sampling instant:

1. Update the state estimate (including load disturbance).
2. Compute an updated steady state target, e.g. by solving equation (60).
3. Compute the next control action by solving a constrained optimization problem.

The structure of the complete MPC controller including these three blocks is depicted in Figure 10.

Less inputs than outputs

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 (60) or (57) 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 optimization problem, where the hard 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 ($p > m$), then:

1. Define desired setpoints for the outputs, y_{sp}.
2. Solve the following optimization 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

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

More inputs than controlled outputs

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 ($p_z < m$), then:

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 optimization problem to find feasible steady-state targets:

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

subject to

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

Two assumptions have been made in the previous formulation, namely:

1. The problem is feasible for the controlled variable setpoints z_{sp}, which is a natural assumption for existence of solution for x_s and u_s.

2. The matrix R_s is positive definite, which ensures that there is a unique solution.

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.

$$C_z = HC$$
$$C_y = C,$$

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 optimization problem needs to be modified in the following way to include the disturbance. Optimization 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

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

Off-set free control

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 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:*

$$\begin{aligned} \begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} &= \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix}. \end{aligned} \tag{61}$$

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

$$\operatorname{rank} \begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} = n + p. \tag{62}$$

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.

$$z_s = z_{sp}.$$

The previous proposition confirms that an integral action is achieved.

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 Section 4 of [1] and in Section 9 of [2]. It is only briefly discussed in Section 10.4 of [4].

Goals of this section:

- To refresh the theory on Kalman filter
- To formulate a state estimator based on least-squares (LS)
- To formulate a moving horizon estimator based on LS
- To formulate a moving horizon estimator with constraints

Learning outcomes:

- Understand and explain the basic principles of model predictive control, its pros and cons, and the challenges met in implementation and applications
- Correctly state, in mathematical form, MPC formulations based on descriptions of control problems expressed in application terms
- Describe and construct MPC controllers based on a linear model, quadratic costs and linear constraints

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.

Derivation using probability theory

The Kalman filter can be derived in various ways, but the most common approaches use probability theory, or least-squares. In this course, main focus will be placed on the least-squares approach that will be discussed later, in Section 6.2. Here, we provide a brief derivation using probability theory.

Let us consider a linear stochastic system

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + w(k), \quad x(0) \sim \mathcal{N}(x_0, P_0), \quad w \sim \mathcal{N}(0, Q) \\ y(k) &= Cx(k) + v(k), \quad v \sim \mathcal{N}(0, R) \end{aligned} \tag{63}$$

where w and v are process and measurement noise, respectively, assumed to be independent of each other. They are assumed to be normally distributed with zero mean and covariance Q and R, respectively. Let us also assume that the initial state is normally distributed with mean value x_0 and covariance P_0.

The state estimator consists of two steps:

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

The goal is to find the Kalman filter gain $L(k)$ that minimizes the posterior estimate error covariance $P(k|k)$.

The first step is to derive the posterior estimation error, given a prior estimate $\hat{x}(k|k-1)$. It can be defined as

$$\begin{aligned}
\tilde{x}(k|k) &= x(k) - \hat{x}(k|k) = x(k) - [\hat{x}(k|k-1) + \overbrace{L(k)(y(k) - C\hat{x}(k|k-1))}^{\text{Correction using the measurement } y(k)}] \\
&= x(k) - [\hat{x}(k|k-1) + L(k)(Cx(k) + v(k) - C\hat{x}(k|k-1))] \\
&= (I - L(k)C)(x(k) - \hat{x}(k|k-1)) - L(k)v(k).
\end{aligned}$$

The posterior estimate error covariance is

$$\begin{aligned}
P(k|k) &= \mathbb{E}[\tilde{x}(k|k)\tilde{x}(k|k)^\top] \\
&= (I - L(k)C)\underbrace{\mathbb{E}[(x(k) - \hat{x}(k|k-1))(x(k) - \hat{x}(k|k-1))^\top]}_{P(k|k-1)}(I - L(k)C)^\top \\
+ \quad & L(k)\underbrace{\mathbb{E}[v(k)v(k)^\top]}_{R}L(k)^\top
\end{aligned}$$

where $P(k|k-1)$ is the prior estimate error covariance. Notice that some terms disappeared when performing the expectation, since the error between the true value and the prior estimate is uncorrelated with the current measurement noise. The posterior error covariance can be expanded as

$$\begin{aligned}
P(k|k) &= (I - L(k)C)P(k|k-1)(I - L(k)C)^\top + L(k)RL(k)^\top \\
&= P(k|k-1) - P(k|k-1)C^\top L(k)^\top - L(k)CP(k|k-1) \\
&\quad + L(k)[CP(k|k-1)C^\top + R]L(k)^\top.
\end{aligned}$$

The Kalman gain $L(k)$ can be chosen to minimize the estimate uncertainty

$$\frac{\partial P(k|k)}{\partial L(k)} = -2P(k|k-1)C^\top + 2L(k)[CP(k|k-1)C^\top + R] = 0,$$

yielding

$$L^*(k) = P(k|k-1)C^\top[CP(k|k-1)C^\top + R]^{-1}. \tag{64}$$

Using the optimal Kalman gain, the posterior error covariance becomes

$$P^*(k|k) = P(k|k-1) - P(k|k-1)C^\top[CP(k|k-1)C^\top + R]^{-1}CP(k|k-1). \tag{65}$$

By simulating the system,

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

we obtain the prediction state estimate. The error estimate, which will serve as the prior error estimate at $k+1$, is

$$\begin{aligned}
\tilde{x}(k+1|k) &= x(k+1) - \hat{x}(k+1|k) = Ax(k) + Bu(k) + w(k) - [A\hat{x}(k|k) + Bu(k)] \\
&= A(x(k) - \hat{x}(k|k)) + w(k) = A\tilde{x}(k|k) + w(k).
\end{aligned}$$

The error covariance can be obtained as

$$\begin{aligned}
P(k+1|k) &= \mathbb{E}[\tilde{x}(k+1|k)\tilde{x}(k+1|k)^\top] \\
&= A\underbrace{\mathbb{E}[\tilde{x}(k|k)\tilde{x}(k|k)^\top]}_{P(k|k)}A^\top + \underbrace{\mathbb{E}[w(k)w(k)^\top]}_{Q} = AP(k|k)^*A^\top + Q.
\end{aligned}$$

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 prior 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*.

Stationary Kalman filter

Consider the linear system

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

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 Section 5. 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:

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

Riccati equation:

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

Kalman filter:

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

Riccati equation:

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

Let's reconsider the state estimation problem above with one distinction, namely that we are not any longer willing to adopt the stochastic modeling 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:

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

Optimization problem:

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

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

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

Objective function:

$$\begin{aligned}V_T(x(0:T)) = (x(0) - x_0)^\top P_0^{-1}(x(0) - x_0) + \sum_{i=0}^{T}(y(i) - Cx(i))^\top R^{-1}(y(i) - Cx(i)) \\ + \sum_{i=0}^{T-1}(x(i+1) - Ax(i) - Bu(i))^\top Q^{-1}(x(i+1) - Ax(i) - Bu(i)). \quad (66)\end{aligned}$$

The interpretation of this formulation is that we would like to explain the data we have (the prior 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 sub-problems, starting by rewriting the minimization problem as

$$\begin{aligned}&\min_{x(0:T)} V_T(x(0:T)) = \\ &\quad \min_{x(1:T)}\left\{\min_{x(0)}\left(\overbrace{(x(0)-x_0)^\top P_0^{-1}(x(0)-x_0)+(y(0)-Cx(0))^\top R^{-1}(y(0)-Cx(0))}^{V_0(x(0))}\right.\right.\\ &\qquad\left.\left.+\underbrace{(x(1)-Ax(0)-Bu(0))^\top Q^{-1}(x(1)-Ax(0)-Bu(0))}_{V_0(x(0),x(1))-V_0(x(0))}\right)+V(x(1:T))\right\}, \end{aligned} \tag{67}$$

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.*

Step 0: measurement update. In step 0, we will perform the inner minimization in (67) with respect to $x(0)$, and proceed in two 'sub-steps'. Consider the first two terms in (67), i.e. the terms corresponding to the prior guess x_0 and the measurement $y(0)$,

$$\begin{aligned}V_0(x(0)) &= (x(0)-x_0)^\top P_0^{-1}(x(0)-x_0)+(y(0)-Cx(0))^\top R^{-1}(y(0)-Cx(0))\\ &= x^\top(0)\underbrace{[P_0^{-1}+C^\top R^{-1}C]}_{P_m(0)}x(0)-2x^\top(0)[P_0^{-1}x_0+C^\top R^{-1}y(0)]+x_0^\top P_0^{-1}x_0+y^\top(0)R^{-1}y(0).\end{aligned} \tag{68}$$

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

$$P_m(0)=[P_0^{-1}+C^\top R^{-1}C]^{-1} \tag{69}$$

and using the matrix inversion lemma A.7, we get the relation

$$P_m(0)=P_0-P_0C^\top[CP_0C^\top+R]^{-1}CP_0. \tag{70}$$

Now, by completing the squares we can rewrite equation (68) 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

$$\begin{aligned}a &= P_m(0)[P_0^{-1}x_0+C^\top R^{-1}y(0)]\\ &= P_m(0)[(P_m^{-1}(0)-C^\top R^{-1}C)x_0+C^\top R^{-1}y(0)]\\ &= x_0+P_m(0)C^\top R^{-1}(y(0)-Cx_0)\\ &= x_0+(P_0-P_0C^\top[CP_0C^\top+R]^{-1}CP_0)C^\top R^{-1}(y(0)-Cx_0)\\ &= x_0+P_0C^\top(R^{-1}-[CP_0C^\top+R]^{-1}CP_0C^\top R^{-1})(y(0)-Cx_0)\\ &= x_0+P_0C^\top[CP_0C^\top+R]^{-1}([CP_0C^\top+R]R^{-1}-CP_0C^\top R^{-1})(y(0)-Cx_0)\\ &= x_0+\underbrace{P_0C^\top[CP_0C^\top+R]^{-1}}_{L(0)}(y(0)-Cx_0),\end{aligned}$$

and equation (70) has been used in the fourth line.

Defining

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

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.

Step 0: 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 (68); 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

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

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

$$x^*(0) = \hat{x}(0) + P_m(0) A^\top [\underbrace{A P_m(0) A^\top + Q}_{P_t(1)}]^{-1} (x(1) - A\hat{x}(0)) \tag{71}$$
$$V_1^- = \min_{x(0)} V(x(0), x(1)) = d = (x(1) - A\hat{x}(0))^\top P_t^{-1}(1) (x(1) - A\hat{x}(0))$$

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

Step 1: measurement update. In the next step we incorporate two more terms of the objective function. 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 (68). 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

$$V_1(x(1)) = (x(1) - \hat{x}(1))^\top P_m^{-1}(1)(x(1) - \hat{x}(1)). \tag{72}$$

Step 1: 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:

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

State estimator:

$$\begin{aligned}
P_t(k) &= AP_m(k-1)A^\top + Q, \quad P_t(0) = P_0 \\
L(k) &= P_t(k)C^\top[CP_t(k)C^\top + R]^{-1} \\
P_m(k) &= P_t(k) - P_t(k)C^\top[CP_t(k)C^\top + R]^{-1}CP_t(k) \\
\hat{x}(k) &= A\hat{x}(k-1) + Bu(k-1) + L(k)\left(y(k) - C(A\hat{x}(k-1) + Bu(k-1)\right); \; \hat{x}(-1) = x_0 \\
x^*(k) &= \hat{x}(k) + P_m(k)A^\top[AP_m(k)A^\top + Q]^{-1}\left(x^*(k+1) - A\hat{x}(k)\right); \; x^*(T) = \hat{x}(T).
\end{aligned} \tag{73}$$

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 (66) 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 $\{\hat{x}(k)\}_{k=0}^{T}$, see equation (73) above. The key relation is (71), which for a general time index k is

$$x^*(k) = \hat{x}(k) + P_m(k)A^\top[AP_m(k)A^\top + Q]^{-1}(x(k+1) - A\hat{x}(k)). \tag{74}$$

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

$$x^*(T) = \hat{x}(T), \tag{75}$$

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 (72).

The result (73) 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 $\hat{x}(0 : T|T)$. This is in contrast to the sequence of estimates $\{\hat{x}(k|k)\}$, where each $\hat{x}(k|k)$ 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 (73) means that earlier estimates $\hat{x}(k|k)$ need to be *smoothed*, using information *after* time k to provide the optimal estimate $x^*(k) = \hat{x}(k|T)$.

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 Section 5. 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 6.1 (Convergence of estimator cost)**.** *Given noise-free measurements $y(0 : T)$, the optimal estimator cost $V_T^*(y(0 : T))$ converges as $T \to \infty$.*

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

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

Proof of Lemma 6.1. Let the sequence of optimal state estimates at time T be

$$\{\hat{x}(0|T), \hat{x}(1|T), \ldots, \hat{x}(T|T)\} \tag{76}$$

and the corresponding optimal cost be $V_T^*(\hat{x}(0 : T|T))$. At time $T-1$ the optimal cost $V_{T-1}^*(\hat{x}(0 : T-1|T-1))$ may be obtained by a different sequence of optimal state estimates

$$\{\hat{x}(0|T-1), \hat{x}(1|T-1), \dots, \hat{x}(T-1|T-1)\}.$$

If the cost V_{T-1} is computed using the sequence (76), it would result in a possibly non-optimal cost

$$V_T^*(\hat{x}(0 : T|T)) - \left(\hat{v}(T|T)^\top R^{-1}\hat{v}(T|T) + \hat{w}(T-1|T)^\top Q^{-1}\hat{w}(T-1|T)\right)$$

where

$$\hat{w}(i|k) = \hat{x}(i+1|k) - (A\hat{x}(i|k) + Bu(i)) \tag{77}$$

$$\hat{v}(i|k) = y(i) - C\hat{x}(i|k). \tag{78}$$

We can thus conclude that

$$\begin{aligned} V_{T-1}^*(\hat{x}(0 : T-1|T-1)) \leq{} & V_T^*(\hat{x}(0 : T|T)) \\ & - \left(\hat{v}(T|T)^\top R^{-1}\hat{v}(T|T) + \hat{w}(T-1|T)^\top Q^{-1}\hat{w}(T-1|T)\right). \end{aligned}$$

Furthermore, since the quadratic terms in the cost function are nonnegative, the sequence of optimal costs is non-decreasing with increasing T. It can be established that the optimal cost V_{T-1}^* is bounded above by the optimal cost V_T^*, even when the optimal sequence of estimates (76) is used. Let the optimal sequence of estimates be

$$\{x(0), Ax(0) + Bu(0), \dots, A^T x(0) + \sum_{i=0}^{T-1} A^{T-i-1}Bu(i)\},$$

which achieves a cost $V_T^* = (x(0) - x_0)^\top P_0^{-1}(x(0) - x_0)$ that is independent of T (as no measurement noise is assumed). Since the optimal cost sequence is non-decreasing and bounded above, it follows that it converges as $T \to \infty$. □

Notice that the optimal cost converges regardless of the system observability. However, in order for the optimal estimate to converge to the true value of the state, additional restrictions are needed on (A, C) and Q, R, e.g. as those stated in Lemma 6.2.

Proof of Lemma 6.2. Suppose we are at time step $T + n$ and the measured data record

$$y(0 : T+n) = \{y(0), y(1), \dots, y(T), \dots, y(T+n)\}$$

is available to us and hence we have already solved the optimization problem

$$\min_{\hat{x}(0:T)} V_T(\hat{x}(0 : T)) \tag{79}$$

where the minimization is with respect to sequence of state estimates

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

which act as optimization variables in our estimation problem and the objective function V_T is given by

$$V_T(\hat{x}(0 : T)) = (\hat{x}(0) - x_0)^\top P_0^{-1}(\hat{x}(0) - x_0) + \sum_{i=0}^{T-1} \hat{w}(i)^\top Q^{-1}\hat{w}(i) + \sum_{i=0}^{T} \hat{v}(i)^\top R^{-1}\hat{v}(i) \tag{80}$$

where $\hat{w}(k)$ is the state prediction error, $\hat{v}(k)$ is the measurement prediction error, x_0 is the best initial guess about the true initial state $x(0)$ and P_0^{-1}, Q and R are weighting matrices, respectively, for the initial state guess x_0, process disturbance and measurement disturbance, respectively, and T is the length of data record (data recorded over last T time steps).

By solving problem (79) at time step $T+n$, we would compute the optimal state estimate sequence

$$\hat{x}(0:T+n) = \{\hat{x}(0|T+n), \hat{x}(1|T+n), \ldots, \hat{x}(T|T+n), \ldots, \hat{x}(T+n|T+n)\} \tag{81}$$

and the optimal cost V^*_{T+n} is also known. Before we proceed, to compress the notation somewhat, let us define

$$\begin{aligned}\hat{w}_T(i) &= \hat{x}(T+i+1|T+n) - A\hat{x}(T+i|T+n) - Bu(T+n)\\ \hat{v}_T(i) &= y(T+i) - C\hat{x}(T+i|T+n).\end{aligned}$$

Now using the optimal state estimates at time $T+n$ as decision variables for optimization problem at time T allows us to write the inequality

$$\begin{aligned}V^*_T(\hat{x}(0:T|T+n)) \le V^*_{T+n}(\hat{x}(0:T+n|T+n))\\ - \left(\sum_{i=-1}^{n-1} \hat{w}_T^\top(i) Q^{-1} \hat{w}_T(i) + \sum_{i=0}^{n} \hat{v}_T^\top(i) R^{-1} \hat{v}_T(i)\right)\end{aligned}$$

which compares the optimal cost V^*_{T+n} with V^*_T. Now according to the Lemma 6.1, the sequence of optimal costs converges with increasing T, and $Q^{-1},\ R^{-1} \succ 0$, and hence we can easily establish that

$$\hat{w}_T(i) \to 0 \ \text{ as } \ T \to \infty, \ \ \forall i = -1, 0, \ldots, n-1 \tag{82}$$

$$\hat{v}_T(i) = y(T+i) - C\hat{x}(T+i|T+n) \to 0 \ \text{ as } \ T \to \infty, \ \ \forall i = 0, 1, \ldots, n \tag{83}$$

assuming that $u \to 0$ as $T \to \infty$. From the system model (77), the relationship between the last n optimal state estimates — given in the optimal solution (81) of the optimization problem at time $T+n$, with data $y(0:T+n)$ — is given by

$$\underbrace{\begin{bmatrix}\hat{x}(T|T+n)\\ \hat{x}(T+1|T+n)\\ \vdots\\ \hat{x}(T+n|T+n)\end{bmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{bmatrix}I\\ A\\ \vdots\\ A^n\end{bmatrix}}_{\Psi} \hat{x}(T|T+n) + \underbrace{\begin{bmatrix}0 & 0 & \cdots & 0\\ I & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ A^{n-1} & A^{n-2} & \cdots & I\end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix}\hat{w}_T(0)\\ \hat{w}_T(1)\\ \vdots\\ \hat{w}_T(n-1)\end{bmatrix}}_{\hat{\mathbf{w}}} \tag{84}$$

We note that the measurements satisfy:

$$\underbrace{\begin{bmatrix}y(T)\\ y(T+1)\\ \vdots\\ y(T+n)\end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix}C\\ CA\\ \vdots\\ CA^n\end{bmatrix}}_{\mathcal{O}} x(T) \tag{85}$$

where $\mathcal{O}$ is the observability matrix. Now multiplying (84) by C and subtracting from (85) we get

$$\mathbf{y} - C\hat{\mathbf{x}} = \mathcal{O}x(T) - \underbrace{C\Psi}_{=\mathcal{O}} \hat{x}(T|T+n) - C\Phi\hat{\mathbf{w}} = \mathcal{O}\left(x(T) - \hat{x}(T|T+n)\right) - C\Phi\hat{\mathbf{w}} \tag{86}$$

Now applying the result (83) to the LHS of (86) it can be easily seen that LHS of (86) must go to zero as $T \to \infty$ i.e. $(\mathbf{y} - C\hat{\mathbf{x}}) \to 0$ as $T \to \infty$ which implies that RHS of (86) must also go to zero with increasing T. The second term on RHS will go to zero because according to (82) $\hat{\mathrm{w}} \to 0$ as $T \to \infty$. Since the second term goes to zero so the first term must also go to zero i.e. $\mathcal{O}\left(x(T) - \hat{x}(T|T+n)\right) \to 0$ as $T \to \infty$. Since it is assumed in the statement of the Lemma 6.2 that the system (A, C) is observable so it means the observability matrix $\mathcal{O}$ has full column rank (i.e. all columns are independent) which means the nullspace $\mathcal{N}(\mathcal{O}) = 0$ and hence

$$\begin{aligned} \mathcal{O}\left(x(T) - \hat{x}(T|T+n)\right) = 0 \ \text{ if and only if } \ (x(T) - \hat{x}(T|T+n)) = 0 \\ \Rightarrow \hat{x}(T|T+n) \to x(T) \ \text{ as } \ T \to \infty \end{aligned} \tag{87}$$

Thus we conclude that the smoothed estimate $\hat{x}(T|T+n)$ converges to the true state $x(T)$. Now from this we also want to establish that $\hat{x}(T|T) \to x(T)$ as $T \to \infty$. Because the $\hat{w}_T(i)$ terms go to zero with increasing T, the last line of (84) gives

$$\hat{x}(T+n|T+n) \to A^n \hat{x}(T|T+n) \ \text{ as } \ T \to \infty \tag{88}$$

and using (87) in (88), we can conclude that

$$\hat{x}(T+n|T+n) \to A^n x(T) \ \text{ as } \ T \to \infty. \tag{89}$$

From the system model with $u \to 0$ we can write $x(T+n) \to A^n x(T)$ and using in (89) we get

$$\hat{x}(T+n|T+n) \to x(T+n) \ \text{ as } \ T \to \infty \tag{90}$$

and therefore, after replacing $T+n$ by T in (90), we have

$$\hat{x}(T|T) \to x(T) \ \text{ as } \ T \to \infty$$

and the asymptotic convergence of the estimator is established. $\square$

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 fulfill such constraints seems appropriate and may contribute to make the control optimization 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:

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

Optimization 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

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

Objective function:

$$\begin{aligned}\hat{V}_T(x(k-T:k)) = &\sum_{i=k-T}^{k-1} (x(i+1) - Ax(i) - Bu(i))^\top Q^{-1} (x(i+1) - Ax(i) - Bu(i)) \\ &+ \sum_{i=k-T}^{k} (y(i) - Cx(i))^\top R^{-1} (y(i) - Cx(i)) .\end{aligned} \tag{91}$$

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:

$$\begin{aligned}\hat{V}_T(x(k-T:k)) = &(x(k-T) - x_{k-T})^\top P_0^{-1}(x(k-T) - x_{k-T}) \\ &+ \sum_{i=k-T}^{k-1} w(i)^\top Q^{-1} w(i) + \sum_{i=k-T}^{k} v(i)^\top R^{-1} v(i).\end{aligned}$$

Optimization problem:

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

with respect to

$$\{x(k-T:k), w(k-T:k-1), v(k-T:k)\}$$

and subject to

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

Remark 6.1. *This formulation of the moving horizon estimator includes the prior weighting penalty P_0. Alternative formulations are possible where $P_0 = 0$ and the only data used for the estimation is from the most recent T sampling instants. This is referred to in [1] as* zero prior weighting.

7 Optimization basics and convexity

The aim of this and Section 8 is to give a very brief introduction to constrained convex optimization, an important ingredient of MPC. This section takes the first steps in introducing convexity for sets and functions and formulating the class of convex optimization problems.

The textbooks [1] (Appendix A) and [2] (Sections 2.1-2.5) both provide an overview of optimization 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 optimization 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 optimization is a vast area, and it is not the purpose here to go very far.

Goals of this section:

- To formulate a general constrained optimization problem
- To formulate necessary conditions for optimality
- To master the basics of convex sets and convex functions
- To formulate a standard convex optimization problem
- To characterize a standard quadratic program (QP)

Learning outcomes:

- Understand and explain basic properties of the optimization problem as an ingredient of MPC, in particular concepts like linear, quadratic and convex optimization, optimality conditions, and feasibility

7.1 Introduction

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

$$\begin{aligned} \underset{\mathbf{u},\mathbf{x}}{\text{minimize}} \quad & V(x,\mathbf{u},\mathbf{x}) = x^\top Q x + \mathbf{x}^\top \bar{Q}\mathbf{x} + \mathbf{u}^\top \bar{R}\mathbf{u} \\ \text{subject to} \quad & x(k+1) = Ax(k) + Bu(k) \\ & F\mathbf{u} + G\mathbf{x} \le h \end{aligned} \tag{92}$$

and in the more general case the analogous formulation is

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

Both formulations are examples of *constrained optimization* or *constrained mathematical programming* problems. It is clear that a vital part of model predictive control is to understand the nature of these optimization 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 optimization in this and Section 8. 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 *optimization 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 optimization problem is formulated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned} \tag{93}$$

where

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

The *optimal solution* x^* has the smallest value of $f(\cdot)$ among all vectors x that belong to $\operatorname{dom} f$ (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,

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

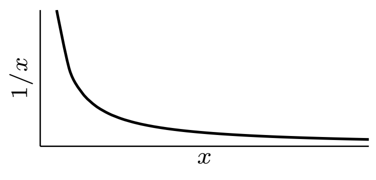
The *feasible set* $\mathcal{S}$ consists of all points satisfying the constraints, i.e.

$$\mathcal{S} = \{x \mid g_i(x) \leq 0, h_i(x) = 0, \forall i\}.$$

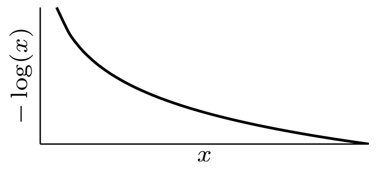
Notice that the optimization problem may have a unique optimal solution, several alternative optimal solutions, or no optimal solution at all (when the problem is infeasible or unbounded below).

Example 7.1. Some examples of the definitions above are:

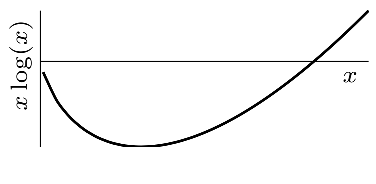
- $f(x) = 1/x$, $\operatorname{dom} f = \mathbb{R}_{++}$ (strictly positive reals), $p^* = 0$, no optimal solution.



- $f(x) = -\log x$, $\operatorname{dom} f = \mathbb{R}_{++}$, $p^* = -\infty$.



- $f(x) = x \log x$, $\operatorname{dom} f = \mathbb{R}_{++}$, $p^* = -1/e$, $x^* = 1/e$.



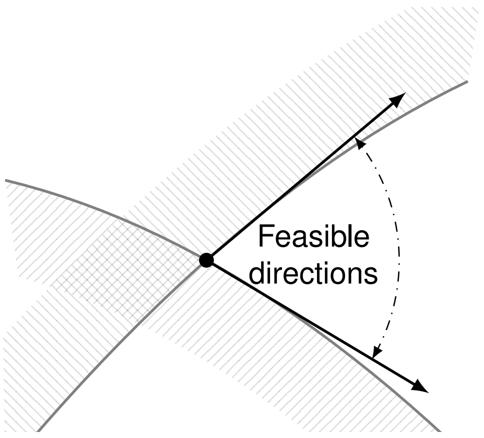
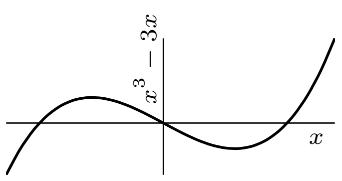


Figure 11: Feasible directions from the point of intersection of two nonlinear constraints.

- $f(x) = x^3 - 3x$, $p^* = -\infty$, local optimum at $x = 1$.



■

The definitions above concern what is often referred to as the *global* optimum. When discussing optimization 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 neighborhood of x^*, i.e. there is an $r > 0$ such that

$$x \text{ feasible and } \quad |x - x^*| \le r \quad \Rightarrow \quad f(x) \ge f(x^*). \tag{94}$$

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

7.2 Conditions for optimality – the KKT conditions

Many optimization algorithms use the properties of local optima to search for the optimal solution. The condition (94) 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 11 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,

$$f(x^* + \varepsilon d) - f(x^*) \ge 0, \quad \varepsilon \text{ small enough.} \tag{95}$$

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

$$f(x^* + \varepsilon d) - f(x^*) = \varepsilon \nabla f(x^*)^\top d + \frac{1}{2}\varepsilon^2 d^\top \nabla^2 f(x^*) d + \text{(higher order terms)}, \tag{96}$$

[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*.

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**

$$x^* \text{ is a stationary point} \quad \Rightarrow \quad \nabla f(x^*) = 0. \tag{97}$$

- **Second order sufficient conditions**

$$\nabla f(x^*) = 0 \text{ and } \nabla^2 f(x^*) \succ 0 \quad \Rightarrow \quad x^* \text{ is a strict local minimum.} \tag{98}$$

Equality constraints

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

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h(x) = 0 \end{array} \tag{99}$$

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^*:

$$h(x^* + \varepsilon d) \approx h(x^*) + \varepsilon \nabla h(x^*)^\top d = \varepsilon \nabla h(x^*)^\top d = 0, \tag{100}$$

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 (95), 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 optimization problem

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

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

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

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 optimization problem (93), 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 optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0. \end{aligned} \tag{101}$$

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

$$\nabla f(x^*) + {\mu^*}^\top \nabla g(x^*) + {\lambda^*}^\top \nabla h(x^*) = 0 \tag{102a}$$
$$\mu^* \geq 0 \tag{102b}$$
$$g(x^*) \leq 0, \quad h(x^*) = 0 \tag{102c}$$
$$\mu_i^* g_i(x^*) = 0, \quad i = 1, \ldots, m. \tag{102d}$$

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* (102b) specify that μ have to be nonnegative. The *primal variables* x should satisfy the *primal constraints* (102c). The first condition (102a) can conveniently be expressed as a condition on the **Lagrangian** $\mathcal{L}$,

$$\nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) = 0, \qquad \mathcal{L}(x, \mu, \lambda) = f(x) + \mu^\top g(x) + \lambda^\top h(x). \tag{103}$$

The last of the KKT conditions, $\mu_i^* g_i(x^*) = 0$, 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, i.e. it is *weakly active*).

The condition (102a) has an intuitive geometrical/physical interpretation, as illustrated in Figure 12. At a local optimum, you can think of ∇f as a force that is trying to gravitate the solution down towards lower function values. Equation (102a) expresses that this force is balanced by reactive 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 optimization 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.

- $\nabla\mathcal{L}(x^*, \mu^*) = \nabla f(x^*) + \mu^*\nabla g(x^*) = 0$.
- $-\nabla f(x^*)$ is analogous to *gravitational force* at x^*.
- $-\mu^*\nabla g(x^*)$ is analogous to *reactive force* from the constraint.

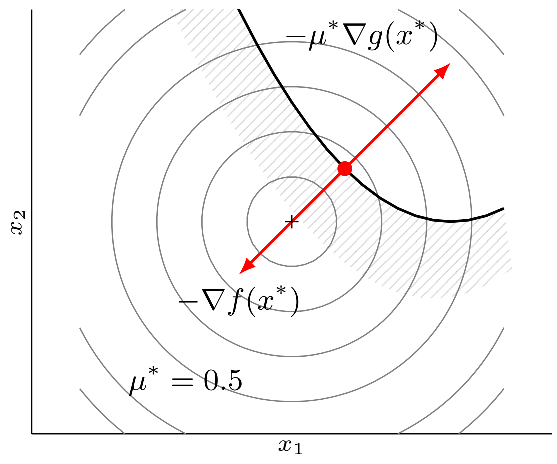


Figure 12: An illustration of KKT conditions for a system with two variables. The objective function f, depicted by contour lines, pulls the solution towards the middle, marked by a plus sign. The inequality constraint prevents reaching this solution. Instead, the optimum, i.e. the point of *minimum potential energy*, is achieved at the point marked by a filled circle.

Consider the optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0. \end{aligned} \tag{104}$$

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 optimization

General nonlinear programming (NLP) problems can be quite challenging to solve, and therefore there are several important sub-classes of optimization 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* optimization problems turns out to be of particular interest:

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

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 optimization 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 optimization problem, as will be seen later. However, in order to better understand the properties of convex optimization, we first need to learn some terminology of **convex sets** and **convex functions**.

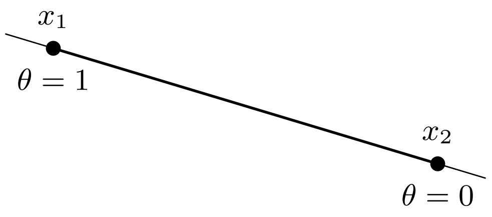


Figure 13: An illustration of an affine set in two dimensions.

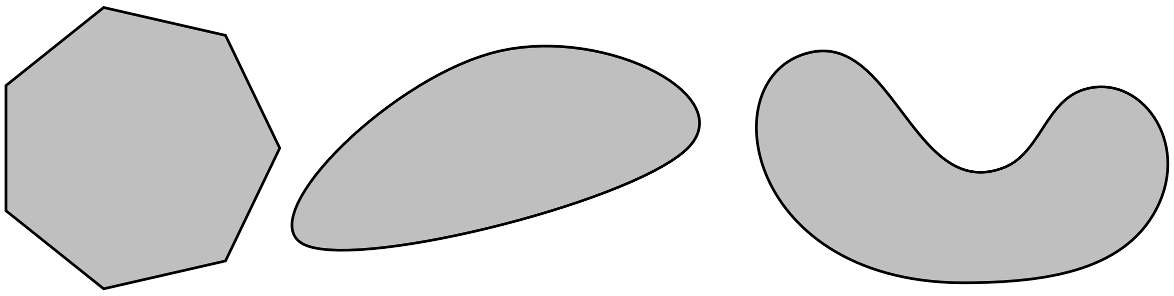


Figure 14: An example with two convex and one non-convex set.

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.

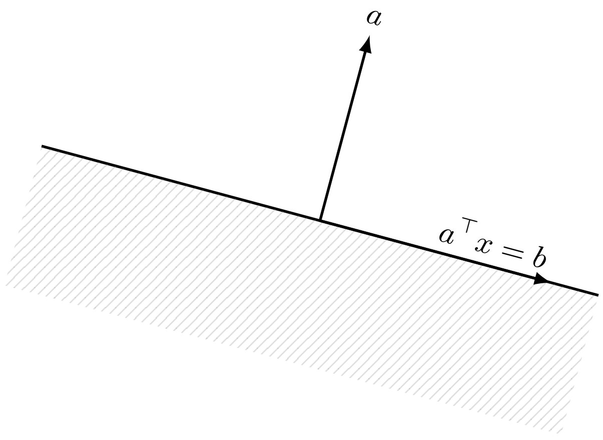


Figure 15: A hyperplane $a^\top x = b$ and a half-space depicted by the shaded region.

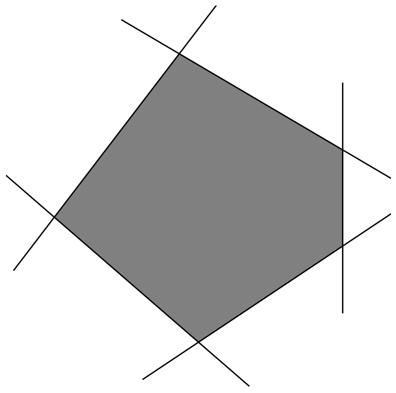


Figure 16: An example of a polyhedron, as the intersection of linear inequalities and equalities.

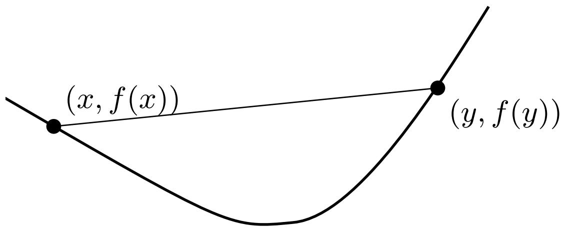


Figure 17: An example of a convex function.

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

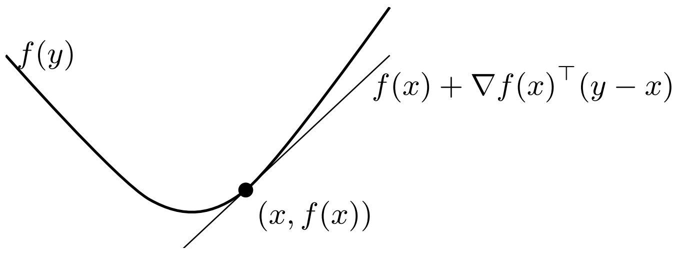


Figure 18: A convex function and its tangent.

- 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 optimization

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

Standard form convex optimization 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 7.1. *The* feasible set *of a convex optimization problem is convex.*

Example 7.2. [Convex or non-convex [5]?] Consider the following optimization problem:

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

We can see that the problem is not convex in the given form, since g_1 is not convex and h_1 is not affine. It is, however, possible to transform the given optimization 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 optimization problem, any globally optimal point is also a locally optimal point, i.e. it gives the lowest cost among feasible points in its neighborhood. A very important property of convex optimization 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 optimization 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 optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned} \tag{105}$$

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 optimization programs are

- **Linear programming (LP)**:

$$\begin{aligned} \text{minimize} \quad & c^\top x + d \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b; \end{aligned}$$

- **Quadratic programming (QP)**:

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

In both cases, the feasible set is a polyhedron.

The two types of optimization problems described above are probably the most well-known classes of convex optimization 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 (92)! 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

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

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

$$-\nabla f(x^*) = -(Qx^* + p) = \sum_{i \in \mathbb{A}} \mu_i G_i^\top, \qquad \text{for some } \{\mu_i\} \text{ with } \mu_i \geq 0, \tag{106}$$

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

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 optimization. This section gives some of the fundamentals.

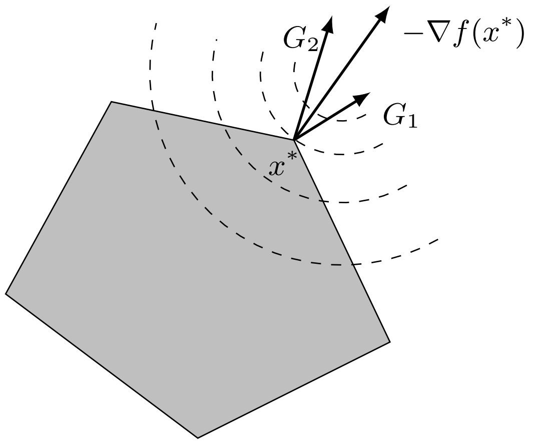


Figure 19: An illustration of a quadratic program with inequality constraints.

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 optimization 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).

Lagrange dual problem:

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

- 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

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

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:

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

■

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

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

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 $\tilde{x} \in \operatorname{int}\mathcal{D}$ (the interior of $\mathcal{D}$) such that (**Slater CQ**)
$$g_i(\tilde{x}) < 0, \quad i = 1, \dots, m; \quad A\tilde{x} = b.$$

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 $i = 1, \dots, m$

which, together with primal and dual constraints, we recognize 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 optimization problem

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

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 Section 7, we have investigated conditions for optimality of constrained optimization 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 optimization in [2] is again in Section 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].

Goals of this section:

- To formulate Newton's method to solve the KKT conditions in simple cases
- To understand the principles of active set and interior point methods for QP:s

Learning outcomes:

- Understand and explain basic properties of the optimization problem as an ingredient of MPC, in particular concepts like linear, quadratic and convex optimization, optimality conditions, and feasibility

8.1 Newton's method

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

$$\begin{aligned} \nabla_x \mathcal{L}(x,\lambda) &= 0 \\ h(x) &= 0 \end{aligned} \tag{107}$$

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, the Newton's method 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

$$r(x + \Delta x) \approx r(x) + r'(x)\Delta x = 0 \quad \Rightarrow \quad \Delta x = -(r'(x))^{-1} r(x). \tag{108}$$

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

$$\nabla r(x)^\top \Delta x = -r(x), \tag{109}$$

which is now a system of *linear* equations in the unknowns Δx. The new iterate can be obtained as

$$x^+ = x + \Delta x.$$

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. An illustration of the Newton method is provided in Figure 20.

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

1. $r = 1.7360$
2. $r = -2.7150$
3. $r = -0.4487$
4. $r = -0.0234$
5. $r = -0.0001$
6. $r = -8.43 \times 10^{-10}$

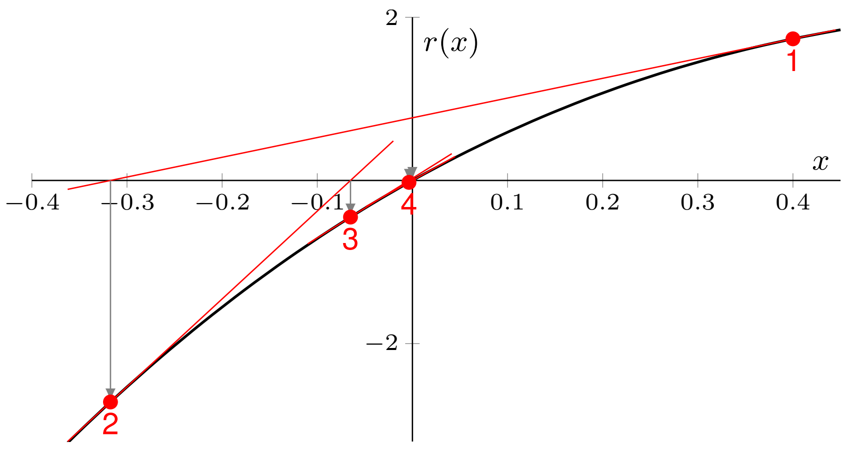


Figure 20: An illustration of the Newton method using a full step size. The method converges in about 4–6 steps, with a roughly quadratic convergence rate.

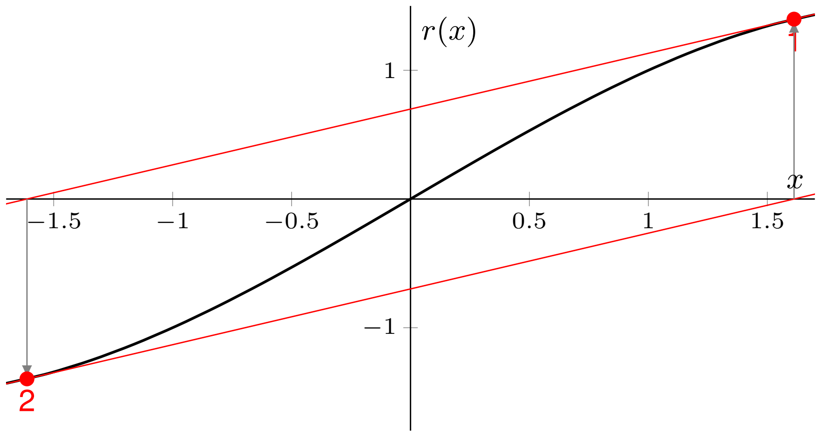


Figure 21: Failure of the Newton method to converge when using a full step size.

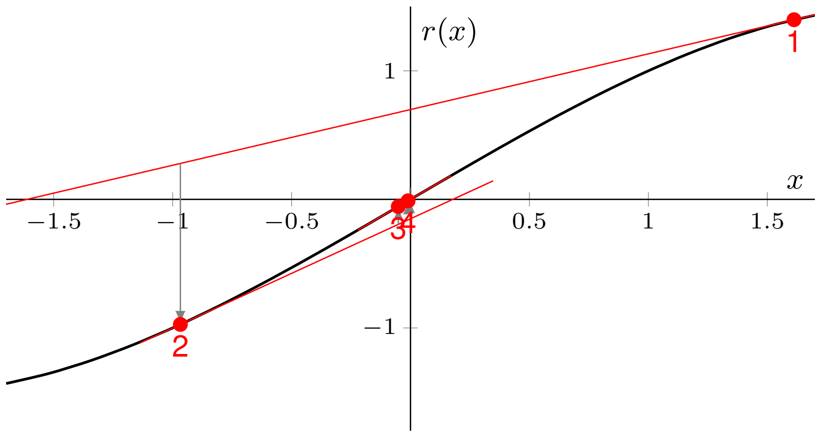


Figure 22: The Newton method using a reduced step size of 0.8. The method converges in about four steps.

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 (109). Instead, while still going in the *Newton direction* as prescribed by (109), the new iterate is obtained by using a *reduced step size* t, i.e.,

$$x^+ = x + t\Delta x, \quad t \in (0, 1]. \tag{110}$$

An example of failure when using a full step size is provided in Figure 21. Using a reduced step size of 0.8, successfully solves the problem, as illustrated in Figure 22. Yet, the Newton method may

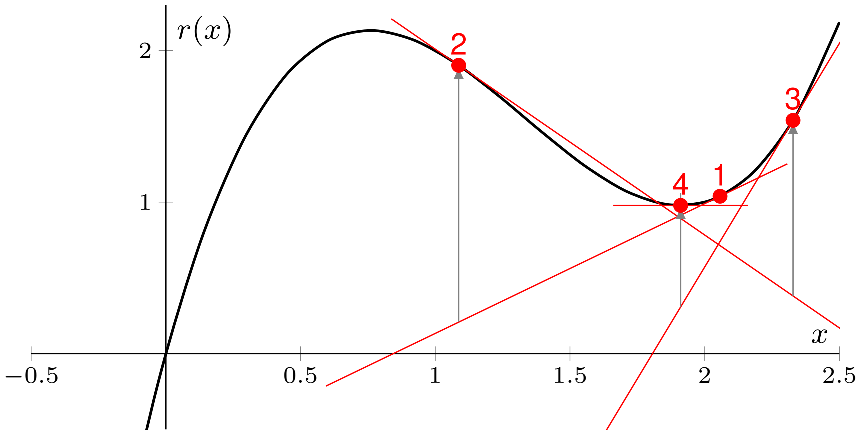


Figure 23: The Newton method converges to a stationary point where derivative is zero. The method loses a search direction and is not able to solve the problem.

still fail to find a solution. It can be seen in (109) that computing the search direction Δx requires inverting the Jacobian $\nabla r(x)$. At stationary points x, the Jacobian is zero and a search direction does not exist. Such failure of the Newton method to find the root of a nonlinear function is illustrated in Figure 23.

Newton's method for equality constrained problems

Let's now try to apply Newton's method to the KKT conditions (102d), observing that the unknowns are x, λ. We want to find the zero of the function

$$r(x,\lambda) = \begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ h(x) \end{bmatrix}.$$

The Newton step (109) becomes

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

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. An illustration of the Newton method to an equality constrained problem is provided in Figure 24 when the algorithm is initialized at different points.

Solutions to quadratic programs

Let us interpret this result for the QP case:

$$\text{minimize} \quad f(x) = \frac{1}{2}x^\top Q x + p^\top x, \quad Q \succ 0 \tag{111}$$

$$\text{subject to} \quad Ax = b, \quad A \in \mathbb{R}^{p\times n}. \tag{112}$$

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^\top \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x + \begin{bmatrix} 2 & 0 \end{bmatrix} x \\ \text{subject to} & x^\top x = 1 \end{array}$$

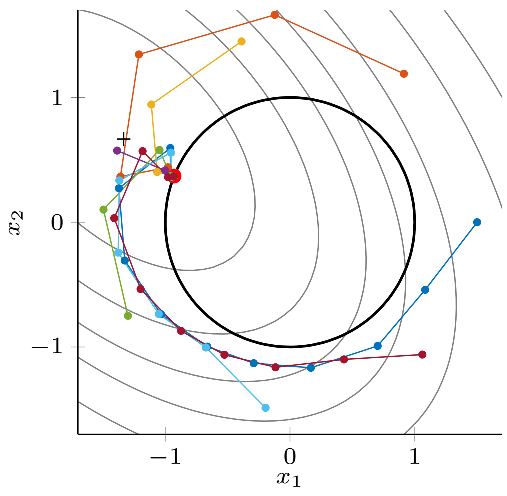


Figure 24: An illustration of the Newton method on an equality constrained problem. It can be seen that different initializations require different number of steps until convergence.

The Lagrangian is

$$\mathcal{L}(x, \lambda) = \frac{1}{2}x^\top Qx + p^\top x + \lambda^\top (Ax - b)$$

and the Newton's method gives

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \lambda^+ \end{bmatrix} = -\begin{bmatrix} Qx + p \\ Ax - b \end{bmatrix} \quad \Leftrightarrow$$
$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} -p \\ b \end{bmatrix}. \tag{113}$$

Notice that equation (113) 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 (113) is identical to the KKT conditions. An analytical solution can be derived as follows. Multiply the first block row in (113) by AQ^{-1} and subtract the result from the second block row. This gives the equivalent system of equations (replacing the $+$ sign with $*$ to indicate optimal solution)

$$\begin{bmatrix} Q & A^\top \\ 0 & -AQ^{-1}A^\top \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -p \\ AQ^{-1}p + b \end{bmatrix} \tag{114}$$

which immediately can be solved, first for λ^*, then for x^*:

$$\lambda^* = -(AQ^{-1}A^\top)^{-1}(AQ^{-1}p + b) \tag{115}$$
$$x^* = -Q^{-1}(p + A^\top \lambda^*) = -Q^{-1}\left(p - A^\top (AQ^{-1}A^\top)^{-1}(AQ^{-1}p + b)\right). \tag{116}$$

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

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

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 lead 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 optimization algorithms that are aimed for other convex optimization 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,

$$\begin{aligned}\text{minimize} \quad & \frac{1}{2}x^\top Qx + p^\top x, \quad Q \succeq 0 \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b.\end{aligned} \tag{117}$$

For easy reference, we repeat the KKT conditions (102a)-(102d) for this special case

$$Qx^* + p + G^\top \mu^* + A^\top \lambda^* = 0 \tag{118}$$

$$\mu^* \geq 0 \tag{119}$$

$$Gx^* - h \leq 0, \quad Ax^* - b = 0 \tag{120}$$

$$\mu_i^*(g_i^\top x^* - h_i) = 0, \quad i = 1, \dots, m. \tag{121}$$

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 (113) for the QP with only equality constraints[5]:

$$\begin{bmatrix} Q & A^\top & G_{\mathbb{A}}^\top \\ A & 0 & 0 \\ G_{\mathbb{A}} & 0 & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \\ \mu_{\mathbb{A}}^+ \end{bmatrix} = \begin{bmatrix} -p \\ b \\ h_{\mathbb{A}} \end{bmatrix}. \tag{122}$$

[5]Here we have returned to the notation x^+ etc. in order to stress the iterative nature of the algorithm.

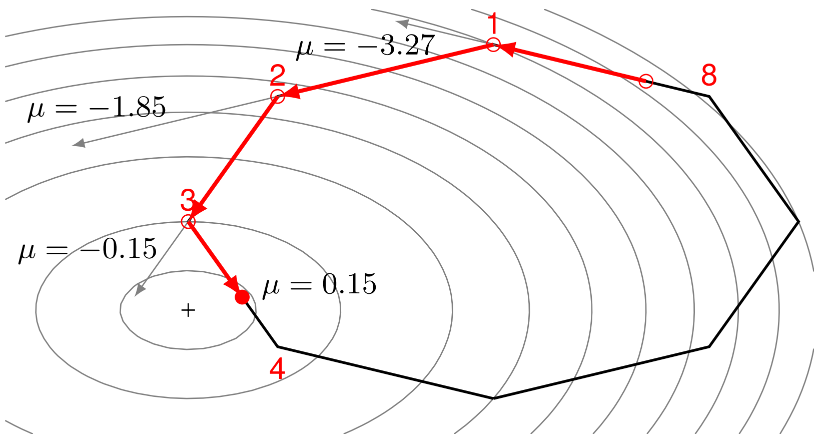


Figure 25: Illustration of active set method with two decision variables and initialized with one active constraint. The feasible region is within the octagon, the contour lines indicate the level sets of the cost function, the plus indicates the unconstrained optimum and the filled circle indicates the optimal solution. The thin gray arrows indicate the solutions of (123) for constraint assumed to be active. When this arrow crosses another constraint, that constraint becomes active. The procedure stops when all constraints are inactive, or when the Lagrange multipliers of the active constraints are nonnegative.

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 (122) for different candidate active sets $\mathbb{A}$. The idea is as follows: Assume that a feasible point is known with specific active constraints $\mathbb{A}$. With this $\mathbb{A}$, the system of equations

$$\begin{bmatrix} Q & A^\top & G_{\mathbb{A}}^\top \\ A & 0 & 0 \\ G_{\mathbb{A}} & 0 & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ \lambda^+ \\ \mu^+_{\mathbb{A}} \end{bmatrix} = \begin{bmatrix} -p \\ b \\ h_{\mathbb{A}} \end{bmatrix} \tag{123}$$

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, when the active set is not empty; they should all be nonnegative at the optimum. If this is not the case, the set of active constraints can be reduced by 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 step size 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.

An illustration of the active set method is provided in Figure 25.

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 optimization 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 optimization 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 (121).

The starting point is a reformulation, or rather approximation, of the original QP problem (117) into an equality constrained problem as follows. The QP problem

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{1}{2}x^\top Qx + p^\top x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

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 optimization 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**. An illustration is provided in Figure 26.

The sequence of optimization 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.

$$
\begin{aligned}
\text{minimize} \quad & f(x) = \frac{x^2}{2} - 2x \\
\text{subject to} \quad & -1 \leq x \leq 1
\end{aligned}
$$

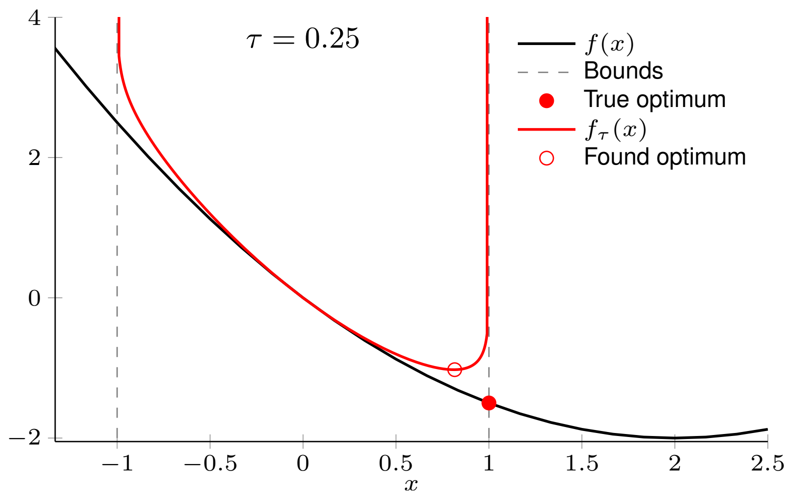


Figure 26: Illustration of the interior point barrier method.

Primal-dual interior point method

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 (118)-(121), 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

$$
\begin{aligned}
Qx + p + G^\top \mu + A^\top \lambda = 0 \\
Ax - b = 0 \\
Gx - h + s = 0 \\
\mu_i s_i = \tau \\
s > 0, \quad \mu > 0.
\end{aligned}
$$

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! An illustration of the newton method applied to an inequality constrained problem is illustrated in Figure 27 when the algorithm is initialized at different points.

$$\begin{aligned} &\text{minimize} && \frac{1}{2}x^\top \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x + \begin{bmatrix} 2 & 0 \end{bmatrix} x \\ &\text{subject to} && x^\top x \le 1 \end{aligned}$$

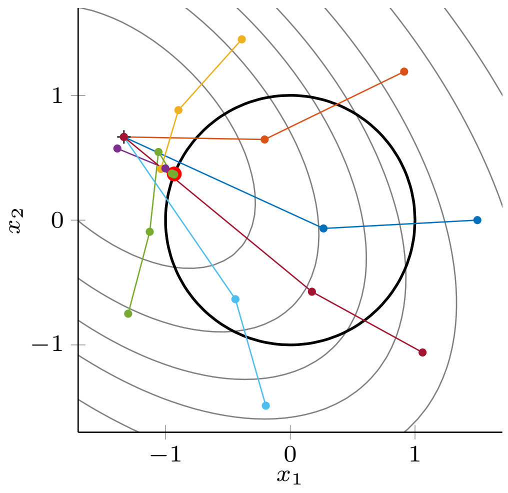


Figure 27: An illustration of the Newton method on an inequality constrained problem.

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 optimization 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 [3] provides an extensive background on feasibility, while the book [1] discusses in a more compact form feasibility and recursive feasibility in Section 2.

Goals of this section:

- To understand the issue of recursive feasibility and how to obtain it
- To understand the role of constraint management and back-up strategies

Learning outcomes:

- Understand and explain the basic principles of model predictive control, its pros and cons, and the challenges met in implementation and applications
- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples
- Understand and explain basic properties of the optimization problem as an ingredient of MPC, in particular concepts like linear, quadratic and convex optimization, optimality conditions, and feasibility

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 optimization 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 optimization, 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.

Another difference between MPC and other control schemes is the ability of MPC to directly impose constraints on states and control inputs. The receding horizon controller is defined *implicitly* as the solution of an optimization problem, which depends on the current state (estimate). However, it is in general not possible to a priori guarantee that the optimization problem posed is *feasible* at every sampling instant. More specifically, in the presence of state constraints, it may very well happen that the optimization 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 below in some detail. But first, let us define the geometry of the MPC feasible set and the concept of reachable sets.

9.1 Polyhedra and polytopes

Recall from the previous sections that the linear MPC can be formulated as a quadratic program (QP). QPs include inequality constraints of the type $Gx \leq h$. These constraints form a convex set that is called a polyhedron. Let us formally state the definition of a polyhedron and its cousin, a polytope:

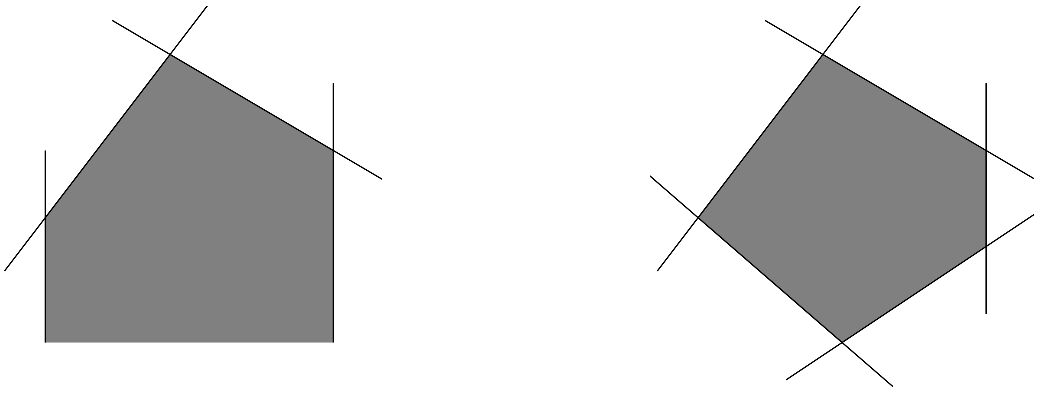

(a) Open polyhedron. (b) Closed polyhedron (polytope).

Figure 28: An illustration of an open polyhedron and a closed polyhedron, i.e. a polytope.

Definition 9.1 (Polyhedron and polytope)**.** *A* ***polyhedron*** *is the intersection of a finite number of closed half-spaces*

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Gx \leq h\}\,.$$

A ***polytope*** *is a bounded polyhedron.*

An illustration of a polyhedron and a polytope is provided in Figure 28.

A polyhedron is in a *minimal representation* if there are no redundant constraints. This means that a removal of any row in $Gx \leq h$ would change the set. Furthermore, a polytope can be described in two equivalent ways, which will be referred to as H- and V-representation.

An *H-representation* of a polyhedron $\mathcal{P} \in \mathbb{R}^n$ denotes an intersection of a finite set of closed half-spaces in $\mathbb{R}^n$,

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Gx \leq h\}\,.$$

This presentation clearly requires providing the matrix G and the vector h.

A *V-representation* of a polytope $\mathcal{P} \in \mathbb{R}^n$ is represented by the convex hull of vertices

$$\mathcal{P} = \mathrm{conv}(V)$$

where V is a set of vertices in the polytope.

The **convex hull** of a set of points $V = \{V_k\}_{k=1}^{N_V}$, with $V_k \in \mathbb{R}^n$, is a polytope defined as

$$\mathrm{conv}(V) = \left\{ x \in \mathbb{R}^n \mid x = \sum_{k=1}^{N_V} \alpha_k V_k, \sum_{k=1}^{N_V} \alpha_k = 1 \right\}.$$

An example of H- and V-representation of a polytope is provided in Figure 29.

Using the MPT toolbox in Matlab, H- and V-representation can be created as follows.

```
% H-polyhedron, G*x ≤ h:
P = Polyhedron(G, h);
% V-polytope, G*x ≤ h, with vertices in matrix V:
P = Polyhedron('V', V);
% Vertices from a polyhedron P can be extracted by
V = P.V;
```

Basic operations on polytopes/polyhedrons

When working with receding horizon control, we will need to compute feasibility of states and control inputs over all time instances along the receding horizon. This will require a basic set of operations to be performed over the polyhedrons describing the feasible sets at each instant.

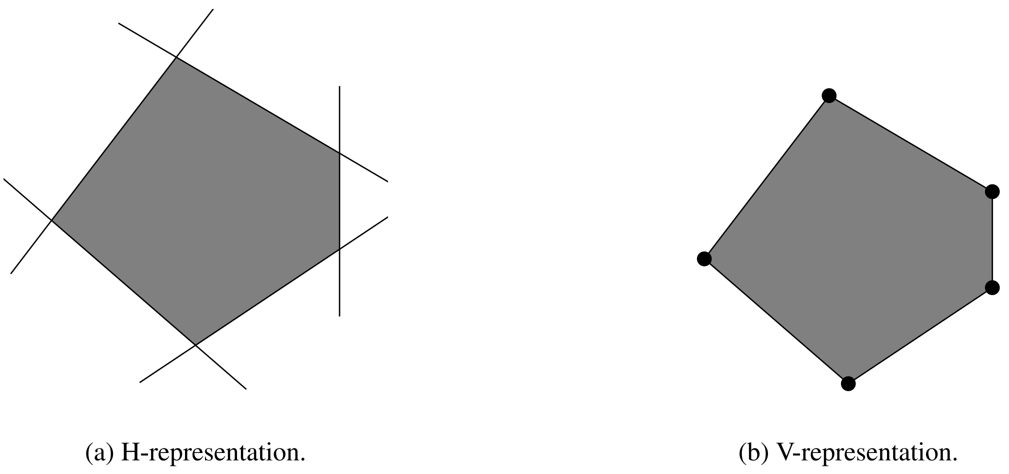

(a) H-representation. (b) V-representation.

Figure 29: Illustration of H- and V-representation of a polytope.

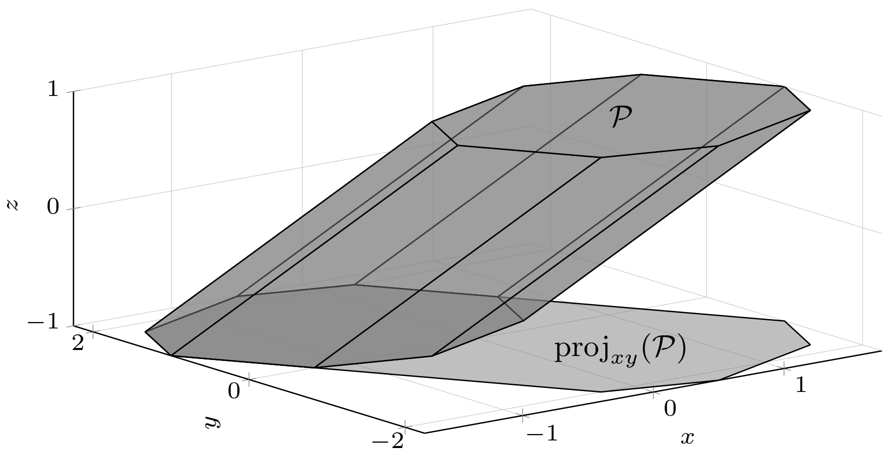


Figure 30: Illustration of a polytope projection onto the xy plane.

Projection: Given a polyhedron

$$\mathcal{P} = \{x \in \mathbb{R}^n, y \in \mathbb{R}^m \mid G_x x + G_y y \leq h\} \subset \mathbb{R}^{n+m}$$

the projection onto the x-space $\mathbb{R}^n$ is defined as

$$\mathrm{proj}_x(\mathcal{P}) = \{x \in \mathbb{R}^n \mid \exists y \in \mathbb{R}^m, G_x x + G_y y \leq h\}.$$

Minkowski sum: The Minkowski sum of two polytopes $\mathcal{P}$ and $\mathcal{Q}$ is a polytope

$$\mathcal{P} \oplus \mathcal{Q} = \{x + y \in \mathbb{R}^n \mid x \in \mathcal{P}, y \in \mathcal{Q}\}.$$

Pontryagin difference: The Pontryagin difference (also known as Minkowski difference) of two polytopes $\mathcal{P}$ and $\mathcal{Q}$ is a polytope

$$\mathcal{P} \ominus \mathcal{Q} = \{x \in \mathbb{R}^n \mid x + y \in \mathcal{P}, \forall y \in \mathcal{Q}\}.$$

Note: The above operations are valid only if the sets $\mathcal{P}$ and $\mathcal{Q}$ are nonempty. An illustration of a set projection is illustrated in Figure 30, while the Minkowski sum and Pontryagin difference are illustrated in Figure 31.

When dealing with dynamic systems, e.g. $x(k+1) = Ax(k)$, we may need to compute the feasible set of $x(k)$ or $x(k+1)$, given the feasible set of the other. This can be achieved via affine mapping.

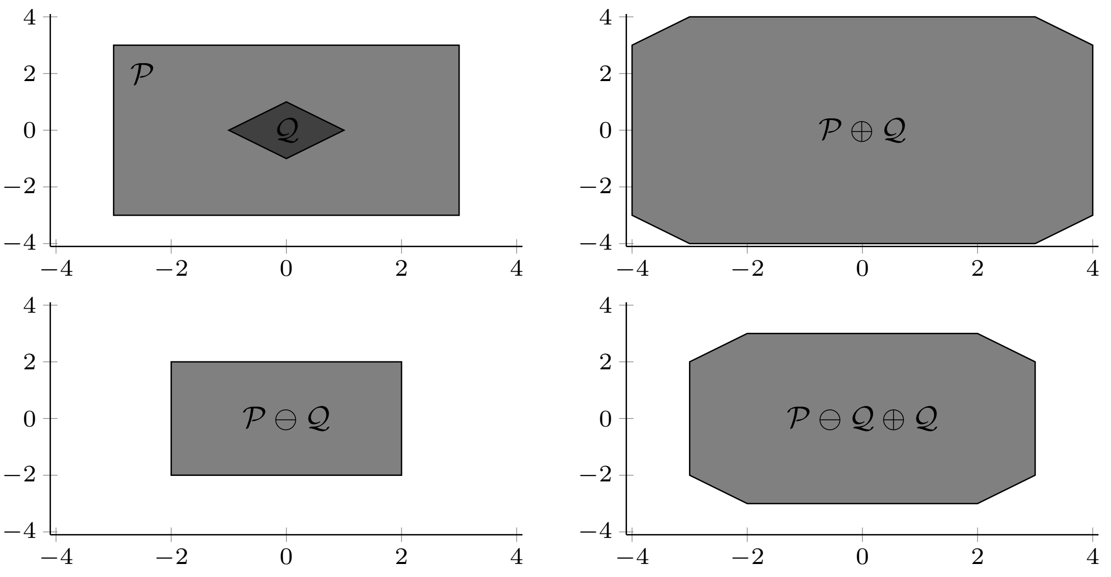


Figure 31: Illustration of basic set operations over polytopes.

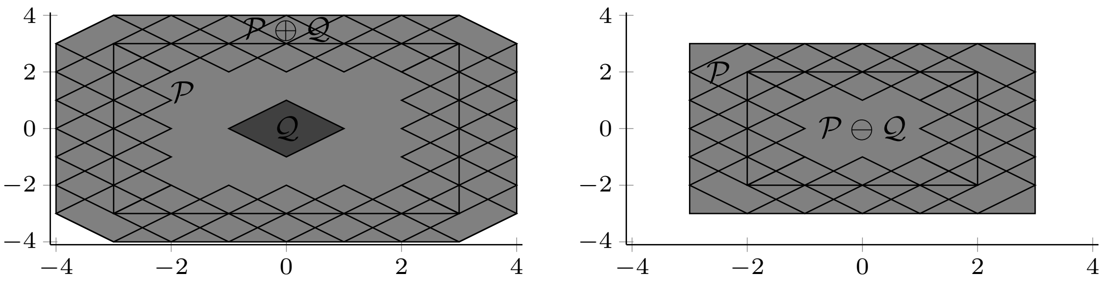


Figure 32: Illustration of Minkowski sum and Pontryagin difference. Since the sets are centered at the origin, the Minkowski sum $\mathcal{P} + \mathcal{Q}$ can be obtained graphically by moving the center of the set $\mathcal{Q}$ over the boundary of $\mathcal{P}$. The Pontryagin difference is obtained by moving $\mathcal{Q}$ just within the boundary of $\mathcal{P}$. It can clearly be seen that $\mathcal{P} \ominus \mathcal{Q} \oplus \mathcal{Q} \neq \mathcal{P}$.

Affine mapping: Consider a polyhedron $\mathcal{P} = \{x \in \mathbb{R}^n \mid Gx \leq h\}$, with an affine mapping $f(y)$,

$$f : y \in \mathbb{R}^m \to Ay + b, A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n.$$

The composition (mapping) of $\mathcal{P}$ onto f is the following polyhedron

$$\mathcal{P} \circ f \triangleq \{y \in \mathbb{R}^m \mid Gf(y) \leq h\} = \{y \in \mathbb{R}^m \mid G(Ay + b) \leq h\}.$$

Similarly, composition of f onto $\mathcal{P}$ is the polyhedron

$$f \circ \mathcal{P} \triangleq \{y \in \mathbb{R}^n \mid y = Ax + b, \forall x \in \mathbb{R}^n, Gx \leq h\}.$$

One way to compute the polyhedron $f \circ \mathcal{P}$ is to write $\mathcal{P}$ in V-representation, $\mathcal{P} = \operatorname{conv}(V)$, and map the vertices $V = \{V_1, \ldots, V_k\}$ through the transformation f. Because the transformation is affine, the set $f \circ \mathcal{P}$ is the convex hull of the transformed vertices, i.e.

$$f \circ \mathcal{P} = \operatorname{conv}(\{AV_1 + b, \ldots, AV_k + b\}). \tag{124}$$

9.2 Reachable sets

Let us return to the dynamic system that could either be autonomous, $x(k+1) = Ax(k)$, or a function of external inputs, $x(k+1) = Ax(k) + Bu(k)$. We are interested in finding the one-step reachable set for $x(k+1)$, given the feasible set for $x(k)$ and $u(k)$. Let $x \in \mathbb{X}$ and $u \in \mathbb{U}$. Then, the reachable set $x(k+1) \in \operatorname{reach}(\mathbb{X})$ can be defined as follows:

Definition 9.2 (Reachable set)**.**

$$\operatorname{reach}(\mathbb{X}) = \{x(k+1) \in \mathbb{R}^n \mid \exists x(k) \in \mathbb{X}, \exists u(k) \in \mathbb{U}, x(k+1) = Ax(k) + Bu(k)\}.$$

By applying the affine mapping operation, it follows that A is mapped onto $\mathbb{X}$, which is added to B mapped onto $\mathbb{U}$. Hence,

$$\operatorname{reach}(\mathbb{X}) = (A \circ \mathbb{X}) \oplus (B \circ \mathbb{U}). \tag{125}$$

Consequently, the reachable set can be computed by mapping A and B onto the vertices of $\mathbb{X}$ and $\mathbb{U}$, respectively, and then performing a Minkowski sum of the resulting polytopes.

Similarly, given the successor set $\mathcal{S}$, it is of interest computing the precursor set $\operatorname{pre}(\mathcal{S})$. For the system $x(k+1) = Ax(k) + Bu(k) \in \mathcal{S}$, where $\mathcal{S}$ is the successor set, the *precursor set* (or *backwards reachable set*) is defined as follows.

Definition 9.3 (Precursor set)**.**

$$\operatorname{pre}(S) = \{x(k) \in \mathbb{R}^n \mid \exists u(k) \in \mathbb{U}, x(k+1) = Ax(k) + Bu(k) \in \mathcal{S}\}.$$

Let the sets for $u(k)$, $x(k+1)$ be defined as

$$\mathbb{U} = \{u(k) \mid G_u u(k) \le h_u\}, \quad \mathcal{S} = \{x(k+1) \mid Gx(k+1) \le h\}.$$

Then, the precursor set is

$$\operatorname{pre}(\mathcal{S}) = \left\{ x(k) \in \mathbb{R}^n \mid \exists u(k) \in \mathbb{R}^m, \overbrace{\begin{bmatrix} GA & GB \\ 0 & G_u \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \le \begin{bmatrix} h \\ h_u \end{bmatrix}}^{\mathcal{T}} \right\} = \operatorname{proj}_x(\mathcal{T}). \tag{126}$$

The operation (126) for obtaining the precursor set can be written more compactly by using the definition for a Minkowski sum and the affine operations over polyhedra, as

$$\begin{aligned}
\operatorname{pre}(S) &= \{x(k) \mid u(k) \in \mathbb{U}, Ax(k) + Bu(k) \in \mathcal{S}\} \\
&= \{x(k) \mid x(k+1) = Ax(k) + Bu(k), u(k) \in \mathbb{U}, x(k+1) \in \mathcal{S}\} \\
&= \{x(k) \mid Ax(k) = x(k+1) - Bu(k), u(k) \in \mathbb{U}, x(k+1) \in \mathcal{S}\} \\
&= \{x(k) \mid Ax(k) \in \mathcal{P}, \mathcal{P} = \mathcal{S} \oplus (-B \circ \mathbb{U})\} \\
&= \{x(k) \mid x(k) \in \mathcal{P} \circ A, \mathcal{P} = \mathcal{S} \oplus (-B \circ \mathbb{U})\} \\
&= \{x(k) \mid x(k) \in (\mathcal{S} \oplus (-B \circ \mathbb{U})) \circ A\}.
\end{aligned}$$

9.3 Recursive (persistent) feasibility

Let us return to the receding horizon problem and assume that the current state $x(k) \in \mathcal{X}_N$ is feasible. This means that there is indeed a solution to the optimization problem for that particular x. The following question is then natural to ask: after having applied the control action computed, will the successor state $x(k+1)$ 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 analyze what happens in more detail, starting by defining formally recursive feasibility using the concept of *invariant sets*.

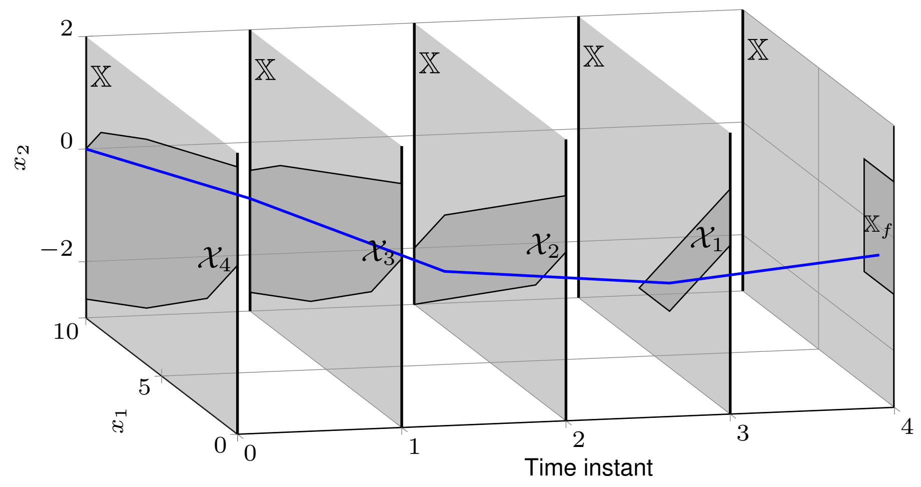


Figure 33: Illustration of N-step feasible sets and a state trajectory that steers the system to the target set $\mathbb{X}_f$.

Definition 9.4 (Invariant set)**.** *The set $\mathcal{S}$ is* positively invariant *for the autonomous system $x(k+1) = f_a(x(k))$ if*

$$x(0) \in \mathcal{S} \quad \Rightarrow \quad x(k) \in \mathcal{S}, \quad \forall k \in \mathbb{N}_+.$$

Definition 9.5 (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 $x(k+1) = f(x(k), \kappa_N(x(k)))$, 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 optimization problem to be solved at every time instant is feasible, provided the initial state is feasible.

To investigate the concept of feasible sets, we will consider the following example.

Example 9.1. [High inertia system] Consider the system

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) \end{aligned}$$

with constraints $u(k) \in [-1, 1]$, $x_1(k) \in [0, 10]$, $\forall k$ and x_2 unbounded. We will study the finite-time optimal control problem with

$$x(0) = \begin{bmatrix} 10 \\ 0 \end{bmatrix}, \quad Q = P_f = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1.$$

For a target set $\mathbb{X}_f = \{x \mid x_1 \in [0, 2], x_2 \in [-1, 1]\}$, the N-step feasible sets are illustrated in Figure 33. It can be observed that the feasible sets $\mathcal{X}_N$ are bounded, even though the set $\mathbb{X}$ is unbounded in x_2. This is a consequence from the need to steer the system to the target set in a finite number of steps. In fact, even when the target set is unbounded, e.g., $\mathbb{X}_f = \mathbb{X}$, the feasible sets $\mathcal{X}_N$ could still be bounded and may include only a subset of values of x_2. This is illustrated in Figure 34. The figure also illustrates optimal state trajectories for a controller with horizon $N = 2$, further detailed in Figure 35. It can be observed that the controller is feasible for the first three samples and MPC predicts that the first state will reach zero at the fourth sample. However, evaluating the controller at the third sample, we find that the problem is infeasible. It can be seen that even the

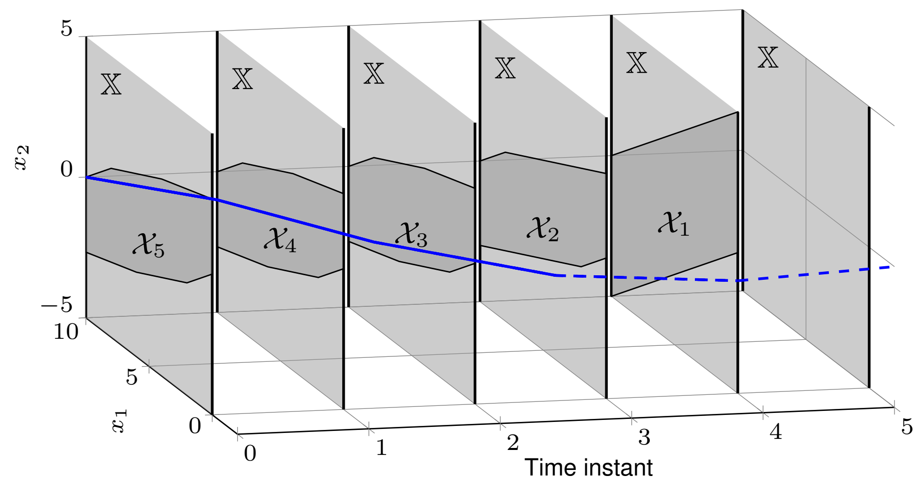


Figure 34: Illustration of N-step feasible sets when $\mathbb{X}_f = \mathbb{X}$. State trajectories are also illustrated for a controller with horizon $N = 2$. It can be observed that already at sample $k = 1$, the predicted trajectory is outside $\mathcal{X}_2$, but still within $\mathbb{X}$. At sample $k = 5$ the predicted optimal trajectory would be outside $\mathbb{X}$, even when maximal control action, $u = 1$, is taken.

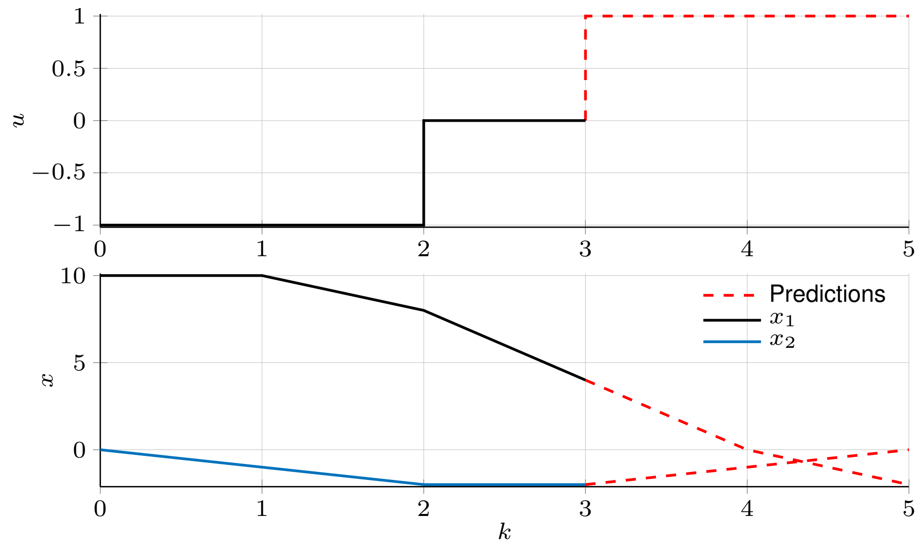


Figure 35: Evolution of states and control actions. The controller is feasible over the first three samples and MPC predicts that the first state will reach zero at the fourth sample. However, after evaluating the controller at the third sample, it is clear that even the highest control action, depicted by the dashed line, is not able to keep the x_1 above zero.

highest control action $u = 1$ is not able to keep the first state above zero, since the system is driven by the inertia of the second state (high negative value).

This is an example where for a given initial condition the controller is feasible over several updates, but it is not recursively feasible. Clearly, infeasibility could have been avoided if the second state was not allowed to drop too quickly, relative to the first state. This can be observed in sample 3 in Figure 34, where we see that the predicted trajectory is outside $\mathcal{X}_2$. This indicates that persistent feasibility is related to the target set, i.e., if the target set at step 3 was chosen to be smaller, e.g., within $\mathcal{X}_2$, then infeasibility might have been avoided. Alternatively, increasing the prediction horizon provides additional control flexibility that may avoid infeasibility. Indeed, Figure 36 shows that with a horizon of $N = 3$ the controller is indeed persistently feasible for the given initial condition.
■

There is one disadvantage with the way we have expressed recursive feasibility, namely the fact

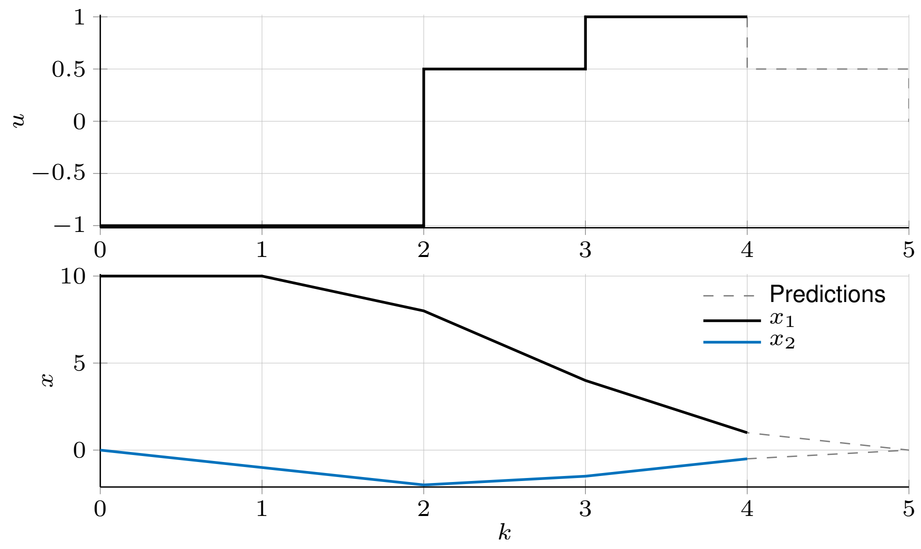


Figure 36: Evolution of states and control actions. For a prediction horizon of $N = 3$ the controller is recursively feasible for the given initial state.

that the condition depends on the receding horizon control law $\kappa_N(x(k))$, 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 $x(k|k) = x(k)$ is feasible, i.e. $x(k) \in \mathcal{X}_N$. Let

$$u(0\!:\!N-1|k) = \{u(k|k), u(k+1|k), \ldots, u(k+N-1|k)\} \in \mathcal{U}_N(k)$$

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

$$x(0\!:\!N|k) = \{x(k|k), x(k+1|k), \ldots, x(k+N|k)\} \in \mathbb{X}, \quad x(k+N|k) \in \mathbb{X}_f,$$

be the corresponding predicted state evolution. 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

$$x(k+1|k+1) = f(x(k|k), u(k|k)) = x(k+1|k).$$

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 $u = u(k+N-1|k+1)$ to be determined:

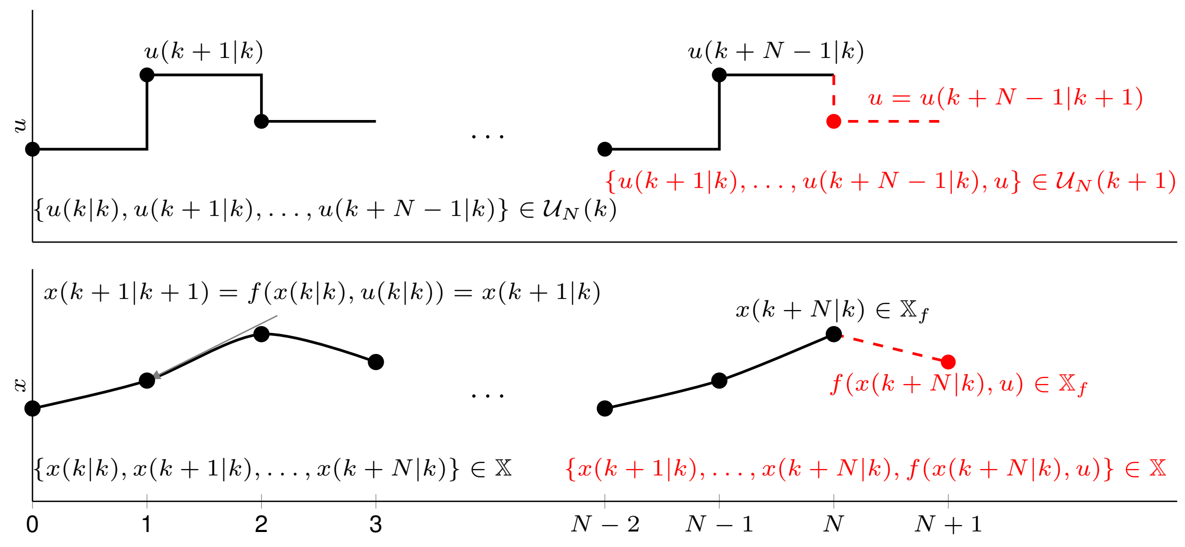
$$u(0\!:\!N-1|k+1) = \{u(k+1|k), \ldots, u(k+N-1|k), u\}.$$

Provided $u \in \mathbb{U}$, this control sequence satisfies the control constraints. The state sequence resulting from $u(0\!:\!N-1|k+1)$ is in an analogous way equal to the tail of $x(0\!:\!N|k)$, appended with a last element,

$$x(0\!:\!N|k+1) = \{x(k+1|k), \ldots, x(k+N|k), f(x(k+N|k), u)\}.$$

Figure 37 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 $x(0 : N|k+1)$, it is clear that it belongs to $\mathbb{X}$, except possibly the last element. However, we need a stronger condition to hold, namely that $f(x(k+N|k), u) \in \mathbb{X}_f$. At this point, we need the following definition:



$u(k+1|k)$
$u(k+N-1|k)$
$u = u(k+N-1|k+1)$
u
$\{u(k+1|k), \ldots, u(k+N-1|k), u\} \in \mathcal{U}_N(k+1)$
$\{u(k|k), u(k+1|k), \ldots, u(k+N-1|k)\} \in \mathcal{U}_N(k)$
$x(k+1|k+1) = f(x(k|k), u(k|k)) = x(k+1|k)$
$x(k+N|k) \in \mathbb{X}_f$
x
$f(x(k+N|k), u) \in \mathbb{X}_f$
$\{x(k|k), x(k+1|k), \ldots, x(k+N|k)\} \in \mathbb{X}$
$\{x(k+1|k), \ldots, x(k+N|k), f(x(k+N|k), u)\} \in \mathbb{X}$
0
1
2
3
$N-2$
$N-1$
N
$N+1$

Figure 37: 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.

Definition 9.6 (Control invariant set). *A set $\mathcal{C} \subseteq \mathbb{X}$ is a* control invariant set *of the system $x(k+1) = f(x(k), u(k))$ if*

$$x(k) \in \mathcal{C} \quad \Rightarrow \quad \exists u(k) \in \mathbb{U} \text{ such that } x(k+1) = f(x(k), u(k)) \in \mathcal{C}.$$

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 $x(k+N|k) \in \mathbb{X}_f$; if $\mathbb{X}_f$ is control invariant, then $u(k) \in \mathbb{U}$ can be chosen such that $f(x(k+N|k), u(k)) \in \mathbb{X}_f$. Hence, $x(k+1)$ is feasible. Since this argument can be applied repeatedly, the following result can been derived.

Theorem 9.1 (Sufficient condition for recursive feasibility). *The receding horizon controller based on the finite horizon optimal control problem* (38)-(41) *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 $\mathcal{C}_\infty$:

$$\Omega_{i+1} = \text{pre}(\Omega_i) \cap \Omega_i, \quad \Omega_0 = \mathbb{X}. \tag{127}$$

The recursion, 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 38 illustrates the different sets we have defined.

Example 9.2. [Maximum control invariant set] Let us revisit Example 9.1 for which we concluded that the controller is not recursively feasible. In order to guarantee recursive feasibility we replace the target set with the maximal control invariant set, i.e. $\mathbb{X}_f = \mathcal{C}_\infty$. The set is shown in Figure 39 and it can be seen that the determinedness index is 3. It can be seen that a bound is now imposed on x_2 that depends on the value of x_1. Simulating the system with this target set ensures that the states remain within the maximal control invariant set. ■

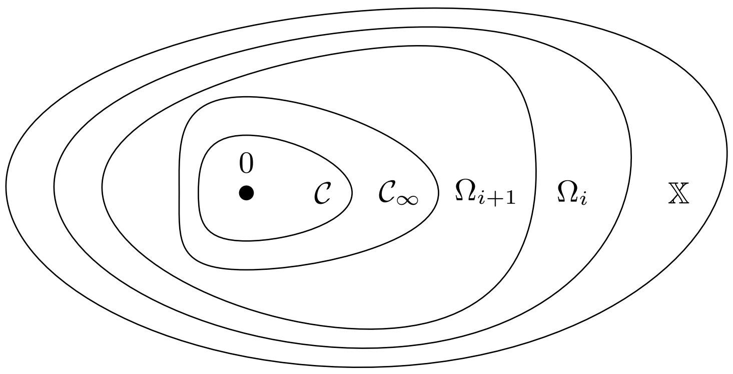


Figure 38: Construction of a maximal control invariant set $\mathcal{C}_\infty$ in $\mathbb{X}$. $\mathcal{C}$ is an arbitrary control invariant set.

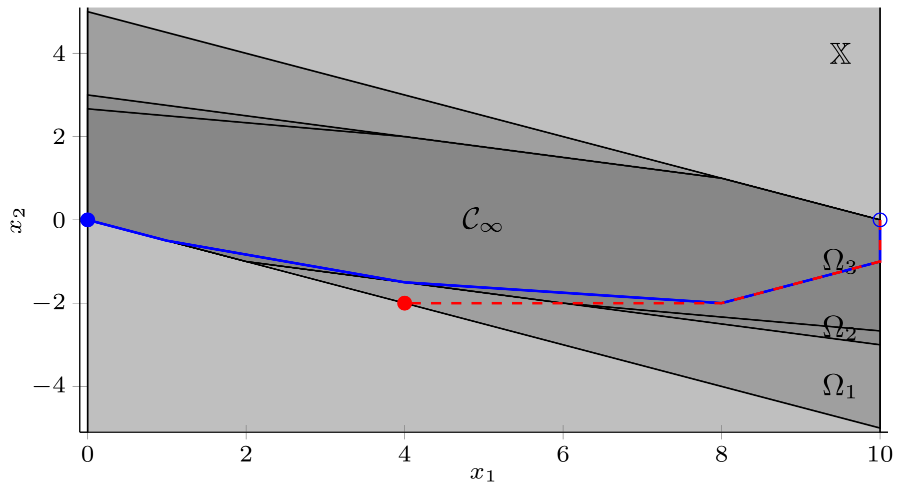


Figure 39: Iterations for computing the maximal control invariant set. The figure also illustrates two simulations of the system, one without a target set (red dashed line) and one where the target set is the maximal control invariant set (blue solid line).

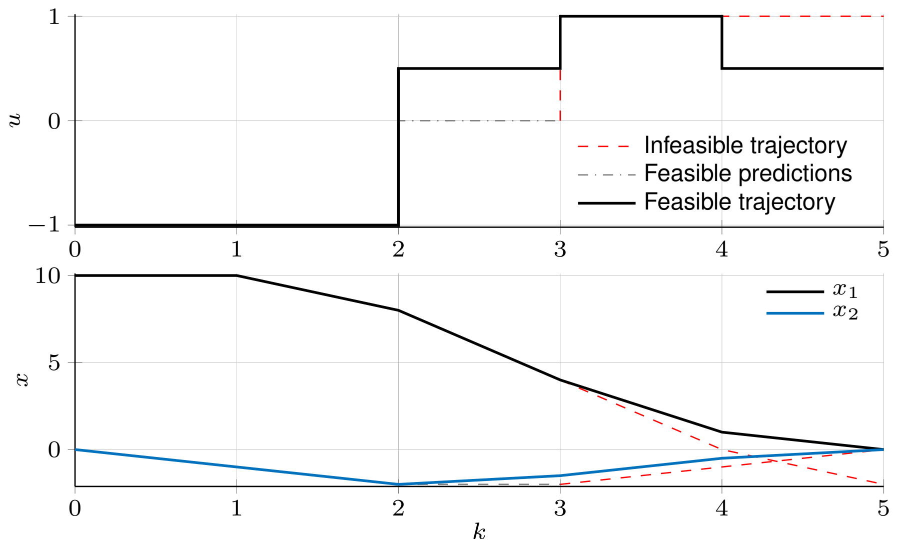


Figure 40: Comparison between the infeasible trajectories, depicted by dashed lines, and recursively feasible trajectories, solid lines, obtained when the target set is the maximal control invariant set.

How big is the feasible set?

It was mentioned when the feasible set $\mathcal{X}_N$ was introduced that it can sometimes be empty. From the assumption that $f(0, 0) = 0$ and that $\mathbb{U}$, $\mathbb{X}$ and $\mathbb{X}_f$ all contain the origin, it follows that $\mathcal{X}_N$ also

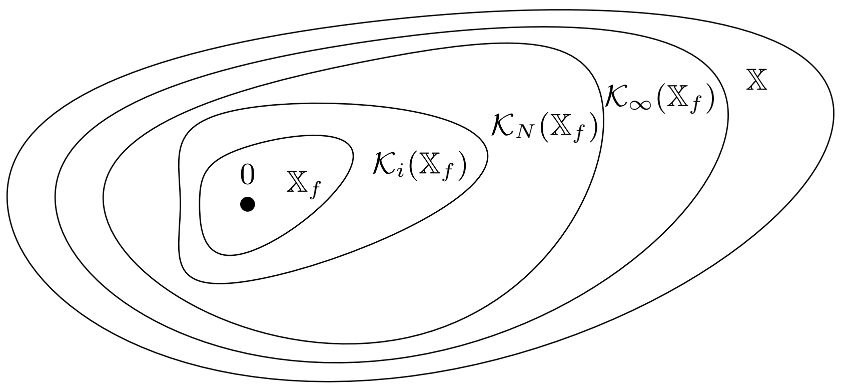

$\mathbb{X}$
$\mathcal{K}_\infty(\mathbb{X}_f)$
$\mathcal{K}_N(\mathbb{X}_f)$
$\mathcal{K}_i(\mathbb{X}_f)$
0
$\mathbb{X}_f$

Figure 41: Construction of the maximal controllable set $\mathcal{K}_\infty(\mathbb{X}_f)$.

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.

Definition 9.7 (N-step controllable set $\mathcal{K}_N(\mathcal{S})$)**.** *The* N-step controllable set $\mathcal{K}_N(\mathcal{S})$ *is defined as the set of initial states that can be driven to the target set $\mathbb{X}_f$ in N steps, while satisfying state and control constraints at all times. It can be computed recursively as*

$$\mathcal{K}_{i+1}(\mathbb{X}_f) = \mathrm{pre}(\mathcal{K}_i(\mathbb{X}_f)) \cap \mathbb{X}, \quad \mathcal{K}_0(\mathbb{X}_f) = \mathbb{X}_f.$$

From the definition, it follows that the feasible set $\mathcal{X}_N$ can alternatively be defined as

$$\mathcal{X}_N = \mathcal{K}_N(\mathbb{X}_f).$$

Definition 9.8 (Maximal controllable set)**.** *The maximal controllable set $\mathcal{K}_\infty(\mathbb{X}_f)$ is defined as*

$$\mathcal{K}_\infty(\mathbb{X}_f) = \bigcup_i \mathcal{K}_i(\mathbb{X}_f).$$

Figure 41 illustrates what has just been said. 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 $\bar{N}$.

A related concept to the controllable sets is the stabilizable sets.

Definition 9.9 (N-step stabilizable set)**.** *For a given target control invariant set $\mathcal{C}$, the N-step stabilizable set is the N-step controllable set $\mathcal{K}_N(\mathcal{C})$).*

Definition 9.10 (Maximal stabilizable set)**.** *For a given target control invariant set $\mathcal{C}$, the maximal stabilizable set is the maximal controllable set $\mathcal{K}_\infty(\mathcal{C})$.*

If the target set is chosen to be control invariant, then a control action exists that can keep the system within the target set. We will investigate later, in Section 11, the necessary conditions for stability later, but at least at this point we know how to keep the system persistently feasible.

Remark 9.1. *In general, the maximal stabilizable set $\mathcal{K}_\infty(\mathcal{C})$ is not equal to the maximal control invariant set $\mathcal{C}_\infty$, even for linear systems. For all control invariant sets $\mathcal{C}$ it holds $\mathcal{K}_\infty(\mathcal{C}) \subseteq \mathcal{C}_\infty$, as there could be initial states in $\mathcal{C}_\infty$ from which it is not possible to steer the system to $\mathcal{C}$.*

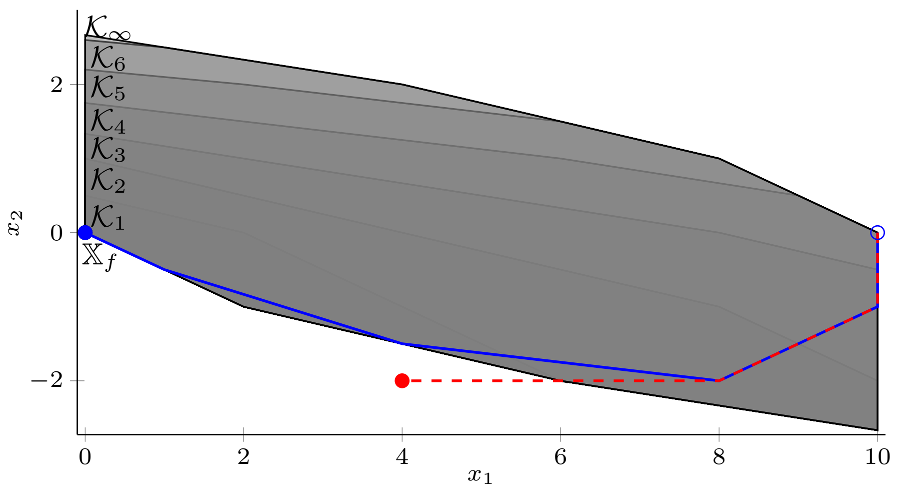


Figure 42: N-step controllable set for a target set $\mathbb{X}_f = \mathbf{0}$. The figure also illustrates two simulations of the system initialized at $\begin{bmatrix}10 & 0\end{bmatrix}^\top$ and without a target set (target states are free), for prediction horizon of $N = 2$ (red dashed line) and $N = 3$ (blue solid line).

Example 9.3. Consider the simple constrained one-dimensional system

$$x(k+1) = 2x(k) + u(k), \quad |x(k)| \leq 1, \quad |u(k)| \leq 1,$$

with the state feedback control law

$$u(k) = \begin{cases} 1 & x(k) \in [-1, -0.5] \\ -2x(k) & x(k) \in [-0.5, 0.5] \\ -1 & x(k) \in [0.5, 1] \end{cases}$$

has $\mathcal{K}_\infty(\mathcal{C}) = (-1, 1) \subseteq \mathcal{C}_\infty = [1, 1]$. Namely, the system remains persistently feasible for all initial states in $[-1, 1]$. Hence, the maximal control invariant set is $\mathcal{C}_\infty = [-1, 1]$. However, the equilibrium point 0 is asymptotically stable only for the open set $(-1, 1)$, as there is no feasible control law that can stabilize the system from $x(0) = 1$ or $x(0) = -1$ (which are also equilibrium points). Hence, for $\mathcal{C} = \{0\}$, it holds $\mathcal{K}_\infty(\mathcal{C}) = (-1, 1) \subseteq \mathcal{C}_\infty$. On the contrary, if the maximal stabilizable set is closed, it can be argued that the maximal stabilizable set is equal to the maximal control invariant set [3].

■

Example 9.4. [N-step controllable set] Let us compute the N-step controllable set for the system in Example 9.1. For a target set $\mathbb{X}_f = 0$, the N-step controllable set is illustrated in Figure 42. It can be seen that the maximal controllable set is reached at the 7th iterations, which means that the determinedness index is 7 for this example. The figure also illustrates two simulations of the system for $x(0) = \begin{bmatrix}10 & 0\end{bmatrix}^\top$ and without a target set (target state is free). When prediction horizon is $N = 2$, the system is not persistently feasible, as concluded previously in Example 9.1. When prediction horizon is $N = 3$ (or greater), the system is persistently feasible as it stays within the maximal control invariant set (here identical to the maximal controllable set). ■

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

$$\mathbb{X}_f^1 \subset \mathbb{X}_f^2 \quad \Rightarrow \quad \mathcal{K}_N(\mathbb{X}_f^1) \subset \mathcal{K}_N(\mathbb{X}_f^2).$$

Example 9.5. [N-step controllable set with a larger target set] For the system in Example 9.1, let us increase the target set to $\mathbb{X}_f = \{x \mid 0 \leq x \leq 1\}$. The iterations for computing the maximal controllable set are illustrated in Figure 43, where it can be seen that the determinedness index has decreased to 6. ■

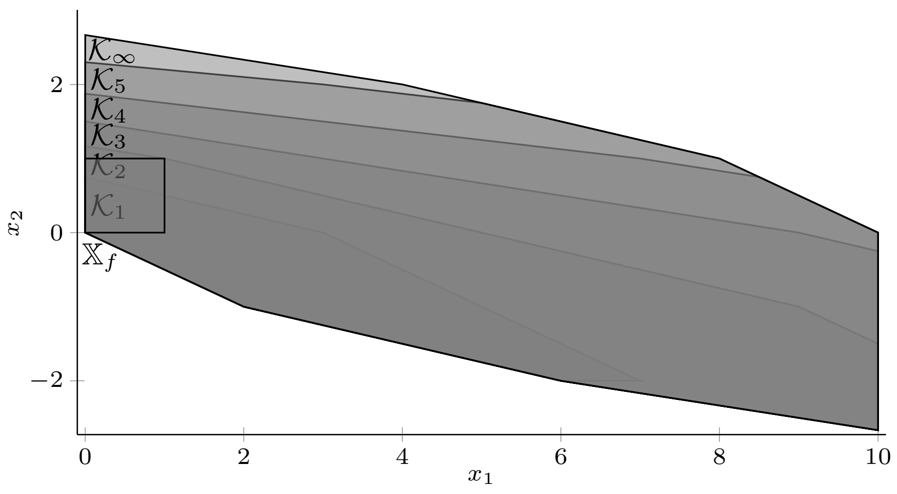


Figure 43: N-step controllable set for a larger target set $\mathbb{X}_f = \{x \mid 0 \leq x \leq 1\}$.

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 Section 11, there are other considerations that may point in a different direction.

9.4 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 optimization 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 optimization problem and deliver a control signal.

- Soft constraints can be introduced:

$$\min_{\mathbf{u}} \quad V_N(\mathbf{u}) + ro\|\varepsilon\| \tag{128}$$

$$\text{subject to} \quad F\mathbf{u} + G\mathbf{x} \leq e + \varepsilon \tag{129}$$

$$\varepsilon \geq 0. \tag{130}$$

- Reduce window in which constraints are enforced.
- Prioritize constraints $\rightarrow$ mixed-integer quadratic program.
- Don't forget time limitations – control output *must* be delivered!

Summary

Let's summarize the main points that have been raised concerning feasibility:

- It may be impossible, for some initial states, to solve the RHC optimization 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 Explicit control laws for constrained linear systems

The theme throughout this course has been to formulate and analyze 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 optimization problem repeatedly online. It turns out, however, that for simple process models, there is an alternative route, namely to solve a *parametric* optimization 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 Section 7 of [1], which contains additional details. Sections 6 and 7 of [2] contain related material.

Goals of this section:

- To understand how the concept of parametric programming can be applied to RHC
- To understand the principles behind so called explicit MPC

Learning outcomes:

- Describe and construct MPC controllers based on a linear model, quadratic costs and linear constraints
- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples

10.1 Parametric programming

The reason why we have to *repeatedly* solve an optimization problem in MPC algorithms is the fact that the current state x changes. This implies that the optimization problem and its solution change with x, which is referred to as a *parameter* of the problem. So called *parametric programming* deals with optimization problems taking the form

$$V^*(x) = \min_u \{V(x,u) \mid u \in \mathcal{U}(x)\}, \tag{131}$$

where x is the parameter of the problem. Whereas the solution to a conventional optimization 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,

$$\mathcal{U}(x) = \{u \mid (x,u) \in \mathcal{Z}\}. \tag{132}$$

The domain of the function $V^*(x)$ in (131) is the set $\mathcal{X}$, defined by

$$\mathcal{X} = \{x \mid \exists u \text{ such that } (x,u) \in \mathcal{Z}\} = \{x \mid \mathcal{U}(x) \neq \emptyset\}. \tag{133}$$

To fix ideas, let's have a look at a simple example of a parametric programming problem.

Example 10.1. [Parametric quadratic programming [1]] Consider the parametric quadratic program $\min_u\{V(x,u) \mid (x,u) \in \mathcal{Z}\}$ with

$$V(x,u) = \frac{1}{2}\left((x-u)^2 + u^2\right)$$
$$\mathcal{Z} = \{(x,u) \mid u \geq 1,\ u + x/2 \geq 2,\ u + x \geq 2\}.$$

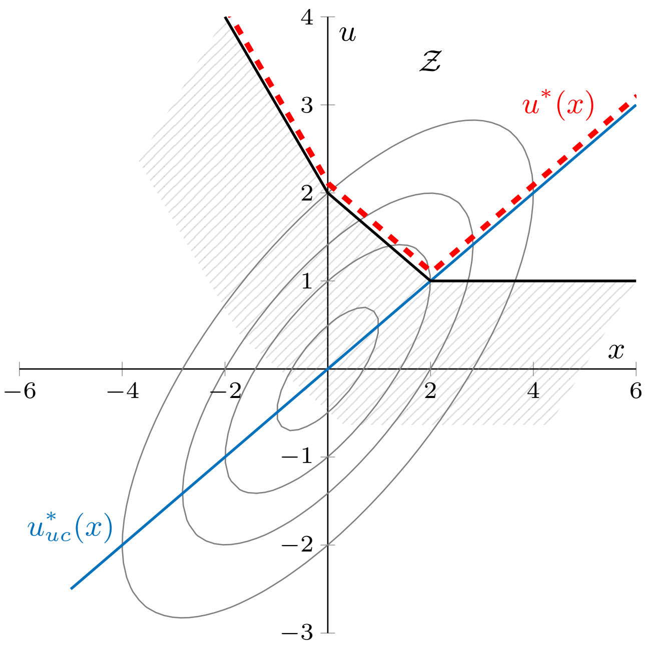


Figure 44: An illustration of the unconstrained, u^*_{uc}, and the constrained parametric solution, $u^*(x)$, to the quadratic program.

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 fulfill 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 44. The solution is *piecewise affine*, and the value function $V^*(x)$ is *piecewise quadratic*.

■

10.2 Constrained LQ control

As already indicated, the ideas presented above can be carried over to the LQ-type MPC focused during the course. Recall that the receding horizon controller is based on the finite-time optimization problem

$$V^*_N(x) = \min_{u(0:N-1)} \{V_N(x, u(0:N-1)) \mid u(0:N-1) \in \mathcal{U}_N(x)\}, \tag{134}$$

where the objective is given by

$$V_N(x, u(0:N-1)) = x^\top(N)P_f x(N) + \sum_{i=0}^{N-1} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right), \quad x(0) = x. \tag{135}$$

In this formulation, the optimization variables are the sequence of controls $u(0:N-1)$ or, equivalently, the vector $\mathbf{u} = \text{vec}(u(0), \ldots, u(N-1))$ as in (21). The states $x(i)$ in (135) can be thought of as short-hand notation for the expressions

$$x(i) = A^i x + \Gamma_i \mathbf{u}, \tag{136}$$

where Γ_i is the i^{th} row of the matrix Γ, see (20).

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 $\mathbf{u}$ 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 $(x, \mathbf{u})$ by using the state expressions (136). The conclusion is that the *implicit* constraint set $\mathcal{U}_N$ can be expressed as an *explicit*, polyhedral set

$$\mathbb{Z} = \{(x, \mathbf{u}) \mid F\mathbf{u} \leq Gx + h\} \tag{137}$$

for some matrices F, G and vector h.

Hence, the MPC optimization problem can now be stated as

$$\begin{aligned} \min_{\mathbf{u}} \quad & V_N(x, \mathbf{u}) = \frac{1}{2}(\mathbf{u}^\top \tilde{R}\mathbf{u} + x^\top \tilde{Q}x) + \mathbf{u}^\top Sx \\ \text{subject to} \quad & F\mathbf{u} \leq Gx + h \end{aligned} \tag{138}$$

for suitable definitions of the matrices $\tilde{R}, \tilde{Q}$, and S, cf. (22). 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 optimization problem (138) 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 optimization 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 optimization problem (138) can also be stated as

$$V^*(x) = \min_{\mathbf{u}}\{V(x, \mathbf{u}) \mid \mathbf{u} \in \mathcal{U}(x)\} \tag{139}$$

where

$$\mathcal{U}(x) = \{\mathbf{u} \mid (x, \mathbf{u}) \in \mathbb{Z}\} = \{\mathbf{u} \mid F\mathbf{u} \leq Gx + h\} \tag{140}$$
$$\mathbb{Z} = \{(x, \mathbf{u}) \mid F\mathbf{u} \leq Gx + h\} \tag{141}$$

and we have dropped the subscript N. The domain of V^* is

$$\mathcal{X} = \{x \mid \exists\mathbf{u} \text{ such that } (x, \mathbf{u}) \in \mathbb{Z}\} = \{x \mid \mathcal{U} \neq \emptyset\}.$$

Since $\tilde{R}$ has been assumed positive definite, the solution $\mathbf{u}^*(x)$ 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 $\mathbf{u}^*(x)$. 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 $w \in R_x^*$, $\mathbf{u}^*(w)$ is the solution of the optimization problem *with the same set of equality constraints*. On the set R_x^*, $\mathbf{u}^*(w)$ 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 $\mathbf{u}^*(x)$ satisfies the necessary and sufficient conditions for optimality, given in Example 7.3. The conditions of optimality are that $\mathbf{u}$ is feasible and that

$$-\nabla_{\mathbf{u}} V(x) = -(\tilde{R}\mathbf{u} + Sx) = \sum_{i \in \mathbb{A}} \mu_i F_i^\top, \qquad \text{for some } \{\mu_i\} \text{ with } \mu_i \geq 0 \tag{142}$$

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:

$$\begin{aligned} F\mathbf{u} &\leq Gx + h \\ -L_x^*(\tilde{R}\mathbf{u} + Sx) &\leq 0 \end{aligned} \tag{143}$$

where L_x^* is a matrix defined by the active constraints corresponding to $\mathbf{u}^*(x)$.

The solution $\mathbf{u}^*(x)$ 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 $F_x^*\mathbf{u} = G_x^* x + h_x^*$, we thus have

$$\mathbf{u}^*(x) = \arg\min_{\mathbf{u}} \{V(x, \mathbf{u}) \mid F_x^*\mathbf{u} = G_x^* x + h_x^*\}.$$

Step 2. At least for points w close to x, it sounds plausible that the optimal solution $\mathbf{u}^*(w)$ would have the same set of active constraints as x. If so, then $\mathbf{u}^*(w)$ 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,

$$\mathbf{u}_x^*(w) = \arg\min_{\mathbf{u}} \{V(w, \mathbf{u}) \mid F_x^*\mathbf{u} = G_x^* w + h_x^*\}. \tag{144}$$

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 (115). The result is that $\mathbf{u}_x^*(w)$ is affine in w and that $V_x^*(w)$ is quadratic in w:

$$\mathbf{u}_x^*(w) = K_x w + k_x \tag{145}$$

$$V_x^*(w) = \frac{1}{2} w^\top Q_x w + r_x^\top w + s_x. \tag{146}$$

The question that needs to be answered is for which set of w:s this solution is in fact optimal, i.e. $\mathbf{u}_x^*(w) = \mathbf{u}^*(w)$. But we have actually formulated necessary and sufficient optimality conditions corresponding to the active set for x in equation (143). Further conclusions can be made as follows:

By inserting the expression for $\mathbf{u}_x^*(w)$ in equation (143) and replacing x with w, we obtain a set of inequalities for w:

$$\begin{aligned} F(K_x w + k_x) &\leq Gw + h \\ -L_x^*(\tilde{R}(K_x w + k_x) + Sw) &\leq 0. \end{aligned} \tag{147}$$

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 $\mathbf{u}_x^*(w)$ satisfies the optimality conditions for the original problem. Hence,

$$\mathbf{u}^*(w) = \mathbf{u}_x^*(w) = K_x w + k_x, \quad \forall w \in R_x^*. \tag{148}$$

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.

- 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^*.
- Any arbitrary feasible $x \in \mathcal{X}$ belongs to a region R_x^*.
- 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 (138) is piecewise affine with a value function that is piecewise quadratic.
- Moreover, as shown in [1], the value function $V^*(x)$ and the minimizer $\mathbf{u}^*(x)$ 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 10.2. [LQ MPC for an integrator, cont'd] In Example **??**, we investigated an LQ-based MPC for a simple integrator system

$$x(k+1) = x(k) + u(k)$$

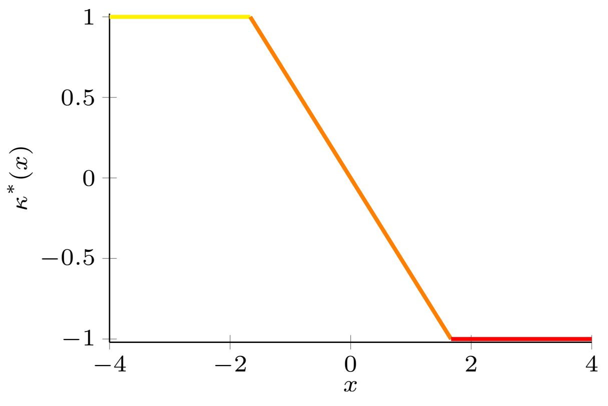
with a control constraint

$$u(k) \in \mathbb{U} = [-1, 1]$$

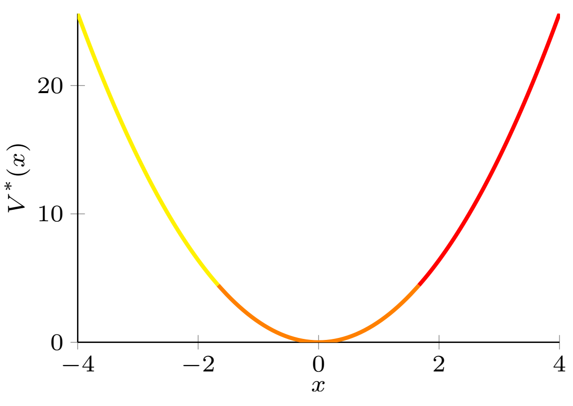
and with $Q = R = P_f = 1$, $N = 2$. For this example, we concluded that the receding horizon control law could be given in closed-form as

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

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. ■

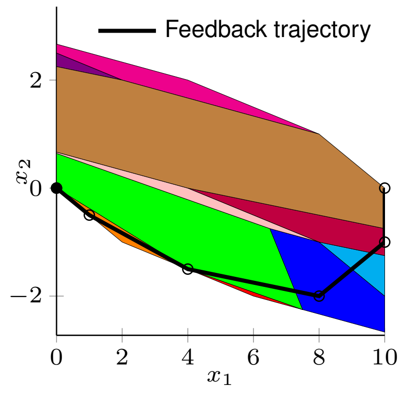


(a) Optimal feedback policy.

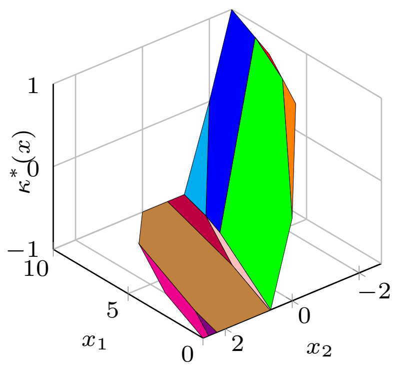


(b) Optimal value function.

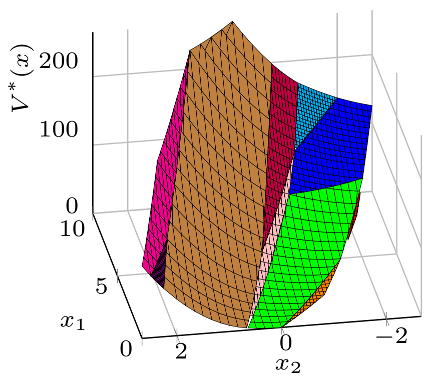
Figure 45: Optimal explicit control law and value function. The colors indicate different control partitions.



(a) Control partitions.



(b) Optimal feedback policy.



(c) Optimal value function.

Figure 46: Optimal explicit control law. The colors indicate different control partitions.

Example 10.3. [High inertia system, cont'd] In Example 9.1 we investigate a high inertia system. Let us revisit this system, where the target set is the maximal control invariant set $\mathcal{C}_\infty$. The optimal feedback policy and value function are depicted in Figure 46.

$$x(k+1) = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k), \quad y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k), \quad x(0) = \begin{bmatrix} 10 \\ 0 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},$$
$$R = 1, \quad P_f = P_\infty, \quad x_1(k) \in [0, 10], \quad u(k) \in [-1, 1], \forall k, \quad N = 2, \quad \mathbb{X}_f = \mathcal{C}_\infty.$$

■

Summary Let's summarize and interpret what we have concluded. When solving the optimization 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 optimization routine is implemented to deliver instead the *parameters* of the affine expression for the optimal $\mathbf{u}^*(x)$, 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 $\mathcal{X}$.

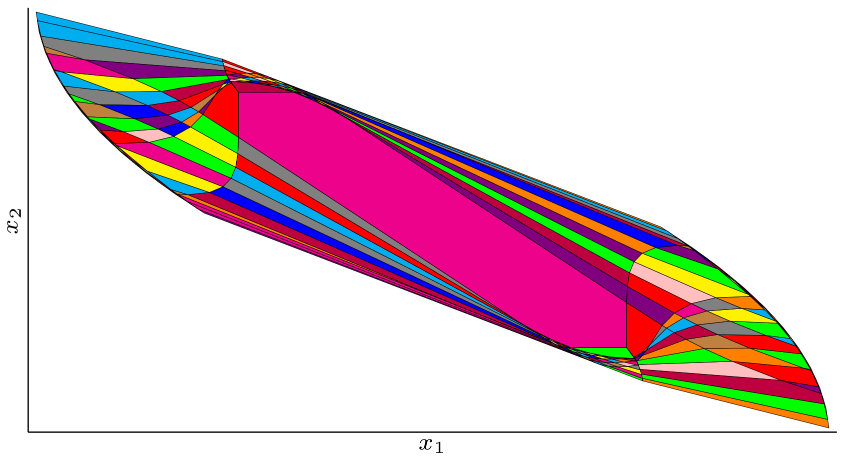


Figure 47: An illustration of a partitioned state-space for a system with 2 states and prediction horizon of 10 samples. There are 267 control partitions with a possibly different affine control law.

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, do the following:
   - 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.

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 47, which shows the regions for a simple second order system.

11 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 focused 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 page 170.

Goals of this section:

- To understand and be able to use Lyapunov functions to study simple stability problems
- To understand the fundamental ingredients in establishing stability for receding horizon controllers
- To formulate and verify the stability conditions of constrained linear quadratic MPC

Learning outcomes:

- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples

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. 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

$$x(k+1) = f(x(k), \kappa_N(x(k))),$$

where $\kappa_N(x(k)) = u^*(0|k)$ 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.

11.1 Stability and Lyapunov functions

Consider the discrete-time system

$$x(k+1) = f(x(k)),$$

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

$$x(k+1) = f(x(k)), \quad f(0) = 0. \tag{149}$$

Definition 11.1 (Local stability)**.** *The origin is* locally stable *for the system* (149) *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 11.2 (Global attraction)**.** *The origin is* globally attractive *for the system* (149) *if* $x(k) \to 0,\ k \to \infty$ *for any initial* $x(0)$.

Definition 11.3 (Global asymptotic stability)**.** *The origin is* globally asymptotically stable *for the system* (149) *if it is locally stable and globally attractive.*

Remark 11.1. *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 11.4 (Lyapunov function)**.** *A function* $V : \mathbb{R}^n \to \mathbb{R}_+$ *is said to be a Lyapunov function for the system* (149) *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 11.2. *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 11.1 (Lyapunov's theorem)**.** *Suppose* $V(\cdot)$ *is a Lyapunov function for the system*

$$x(k+1) = f(x(k)), \quad f(0) = 0.$$

Then the origin is globally asymptotically stable.

Example 11.1. [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 $x(k+1) = Ax(k)$ with the matrix A having its eigenvalues strictly inside the unit circle, Lyapunov's equation (15) implies that

$$\begin{aligned} V(x(k+1)) &= (Ax(k))^\top S(Ax(k)) = x^\top(k)A^\top SAx(k) = x^\top(k)Sx(k) - x^\top(k)Qx(k) \\ &= V(x(k)) - x^\top(k)Qx(k) \end{aligned}$$

so that the third property is also satisfied. Global asymptotic stability thus holds according to Proposition 11.1. ■

Example 11.2. [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. ■

11.2 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:

$$V_N(x_0, u(0{:}N-1)) = \sum_{i=0}^{N-1} l(x(i), u(i)) + V_f(x(N)). \tag{150}$$

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:

$$x(i+1) = f(x(i), u(i)), \quad x(0) = x_0 \tag{151}$$
$$x(i) \in \mathbb{X}, \quad u(i) \in \mathbb{U}, \quad \text{for all } i = 0, \dots, N-1 \tag{152}$$
$$x(N) \in \mathbb{X}_f \subseteq \mathbb{X}. \tag{153}$$

Feasible control sequences and initial states:

$$u(0{:}N-1) \in \mathcal{U}_N(x_0) \tag{154}$$
$$\mathcal{X}_N = \{x_0 \in \mathbb{X} \mid \mathcal{U}_N(x_0) \neq \emptyset\}. \tag{155}$$

Optimal control and state sequences:

$$\begin{aligned} u^*(0{:}N-1) &= \{u^*(k|k), u^*(k+1|k), \dots, u^*(k+N-1|k)\} \\ x^*(0{:}N) &= \{x^*(k|k), x^*(k+1|k), \dots, x^*(k+N|k)\}. \end{aligned} \tag{156}$$

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 11.2 (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 11.3. *Note that nothing is said about the properties of the* optimal value function $V_N^*(x)$ *or the control law $\kappa_N(x) = u^*(0|x(k))$. 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 optimal 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 11.3. [Equivalence between infinite and finite horizon LQ] The infinite horizon LQ problem is based on the criterion in equation (30),

$$V(x(0), u(0\!:\!\infty)) = \sum_{i=0}^{\infty} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right).$$

If the sum is split into two terms we get

$$\begin{aligned} V(x(0), u(0\!:\!\infty)) = &\sum_{i=0}^{N-1} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right) \\ &+ \sum_{i=N}^{\infty} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right). \end{aligned}$$

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 (32).

By minimizing the *finite time* horizon criterion

$$V(x(0), u(0\!:\!N-1)) = \sum_{i=0}^{N-1} \left(x^\top(i)Qx(i) + u^\top(i)Ru(i)\right) + \overbrace{x^\top(N)Px(N)}^{V_f(x(N))},$$

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 analyzing the stability properties of MPC. ■

Basic stability assumption

We proceed with the following basic assumption:

Assumption 11.1 (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 11.4. *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 9.6. The implication of this, together with the properties of $\mathbb{U}$, is that the minimum in the assumption above exists.*

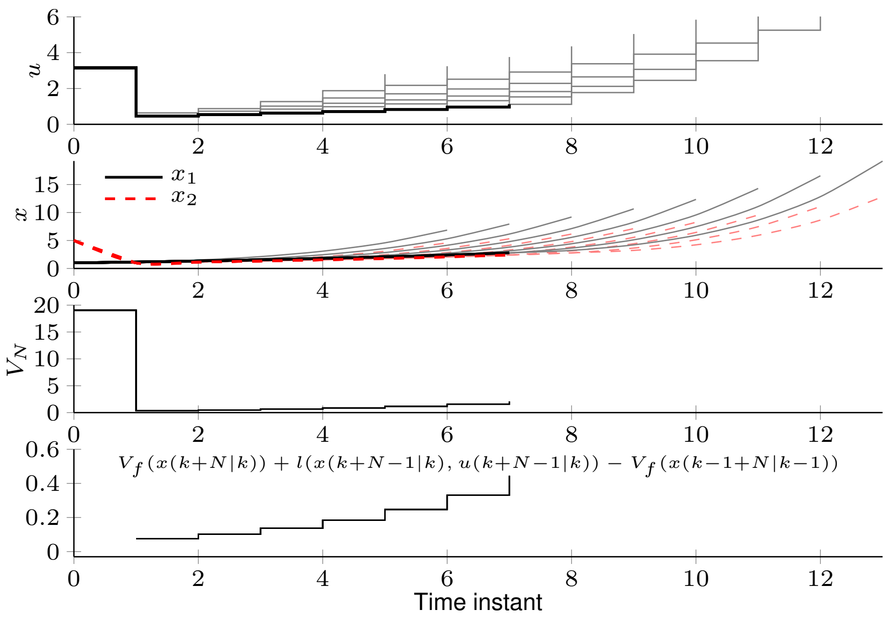


Figure 48: Optimal, bus unstable control. Evolution of the control and state trajectories and value function, and the failure to fulfill the Basic Stability Assumption 11.1.

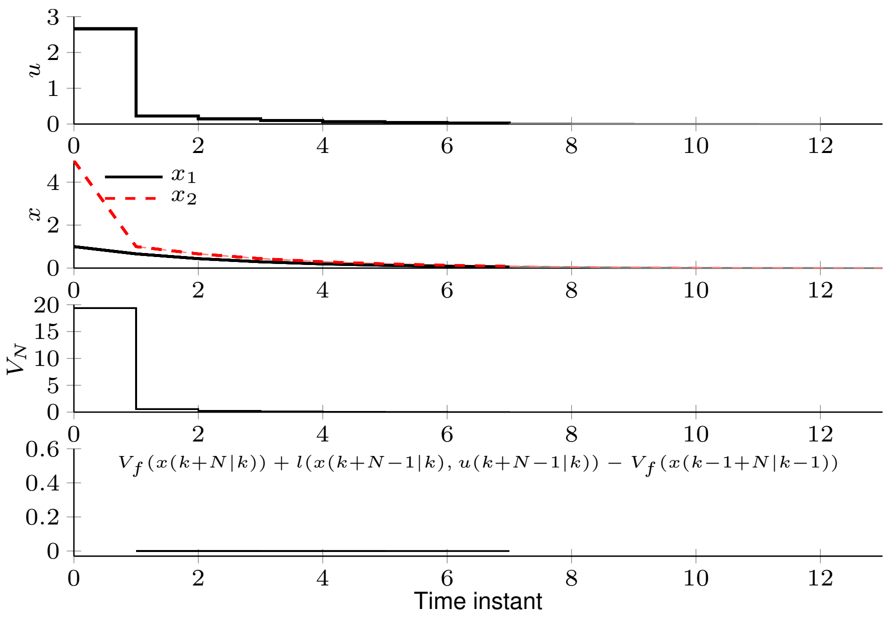


Figure 49: Optimal and stable control. Evolution of the control and state trajectories and value function, and fulfillment of the Basic Stability Assumption 11.1.

Example 11.4. [Example 3.3 revisited] In Example 3.3 it was shown that optimality does not guarantee stability. Let us revisit this example for horizon $N = 6$, initial state $x(0) = \begin{bmatrix} 1 & 5 \end{bmatrix}^\top$ and target cost $P_f = Q$.

The optimal state and control trajectories are shown in Figure 48. As concluded earlier, it can be seen that the system is unstable. Moreover, it is clear that the Basic Stability Assumption 11.1 is not fulfilled.

To improve stability, the target cost has been replaced with the cost of the infinite horizon problem, i.e., $P_f = P_\infty$, where P_∞ is the stationary Riccati matrix. The results are provided in Figure 49. Comparing the value function V_N with that of the unstable system, it can be seen that the unstable controller manages to reduce the initial cost of V_N more than that of the stable controller. However, despite the higher initial cost, the controller in Figure 49 keeps the system stable. The Basic Stability

Assumption 11.1 is here satisfied with equality. ■

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 11.1 (Value function decrease)**.** *Assume that Assumption 11.1 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 $x = x(k|k) = x(k)$ be any point in $\mathcal{X}_N$ with $V_N^*(x) = V_N(x, u^*(0 : N-1|k))$. The corresponding optimal control and state sequences are as in equation (156):

$$\begin{aligned} u^*(0{:}N-1|k) &= \{u^*(k|k), u^*(k+1|k), \dots, u^*(k+N-1|k)\} \\ x^*(0{:}N|k) &= \{x^*(k|k), x^*(k+1|k), \dots, x^*(k+N|k)\}, \end{aligned}$$

where $u^*(k|k) = \kappa_N(x)$ and the successor state is $x^*(k+1|k) = f(x, \kappa_N(x))$. In order to compare $V_N^*(x(k))$ with $V_N^*(x(k+1))$, we note that

$$V_N^*(x(k+1)) \leq V_N(x(k+1), \tilde{u}(0{:}N-1|k+1)) \tag{157}$$

for any feasible control sequence $\tilde{u}(0 : N-1|k+1)$ for the optimal control problem starting at the successor state $x(k+1)$. We will use a sub-optimal control sequence, constructed in the same way as we did when discussing recursive feasibility, see Figure 37. The construction is to simply copy the tail from the previous step and append a final control $u = u(k+N-1|k+1)$ that assures that $f(x^*(k+N|k), u) \in \mathbb{X}_f$, which is possible due to Assumption 11.1. The resulting feasible control sequence is

$$\tilde{u}(0{:}N-1|k+1) = \{u^*(k+1|k), \dots, u^*(k+N-1|k), u\}$$

and the state sequence resulting from $\tilde{u}(0{:}N-1|k+1)$ is

$$\tilde{x}(k+1) = \{x^*(k+1|k), \dots, x^*(k+N|k), f(x^*(k+N|k), u)\}.$$

Having assured the feasibility of the chosen control sequence, we can proceed to compare $V_N(x(k+1), \tilde{u}(0 : N-1|k+1))$ with $V_N^*(x(k))$, 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

$$\begin{aligned} V_N(x(k+1), \tilde{u}(0{:}N-1|k+1)) = {} & V_N^*(x(k)) - l(x(k), \kappa_N(x(k))) \\ & + \underbrace{l(x^*(k+N|k), u) + V_f(f(x^*(k+N|k), u)) - V_f(x^*(k+N|k))}_{\leq 0}. \end{aligned}$$

$$\Rightarrow V_N(x(k+1), \tilde{u}(0{:}N-1|k+1)) \leq V_N^*(x(k)) - l(x(k), \kappa_N(x(k))).$$

The inequality follows from the fact that u can be chosen according to Assumption 11.1. Combining this result with equation (157) completes the proof, i.e.

$$V_N^*(x(k+1)) \leq V_N^*(x(k)) - l(x(k), \kappa_N(x(k))).$$

□

Remark 11.5. *A special case of Lemma 11.1 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 11.1 is no longer needed. The final control 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 11.1 (MPC stability). *Assume that Assumption 11.1 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 $x(k+1) = f(x(k), \kappa_N(x(k)))$.

11.3 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

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

with (A, B) controllable and the stage cost is

$$l(x(k), u(k)) = x^\top(k)Qx(k) + u^\top(k)Ru(k)$$

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 11.1. 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.

$$V_f(x(k)) = V_\infty^{\mathrm{uc}}(x(k)) = x^\top(k)Px(k),$$

where P is the solution of the algebraic Riccati equation. The value function satisfies the equation

$$\begin{aligned} V_\infty^{\mathrm{uc}}(x(k)) &= \min_{u(k)}\{x^\top(k)Qx(k) + u^\top(k)Ru(k) + V_\infty^{\mathrm{uc}}(x(k+1))\} \\ &= x^\top(k)Qx(k) + (Kx(k))^\top R(Kx(k)) + \underbrace{V_\infty^{\mathrm{uc}}(Ax(k) + BKx(k))}_{V_f((A+BK)x(k))} \end{aligned}$$

which implies that

$$V_f((A+BK)x(k)) + x^\top(k)Qx(k) + (Kx(k))^\top R(Kx(k)) = V_f(x(k)).$$

This means that the chosen V_f satisfies the inequality of Assumption 11.1, *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:

1. $x(k) \in \mathbb{X}_f \Rightarrow Kx(k) \in \mathbb{U}$ and
2. $x(k) \in \mathbb{X}_f \Rightarrow (A+BK)^i x(k) \in \mathbb{X}_f$ for all $i \geq 0$.

The set $\mathbb{X}_f$ thus defined is control invariant. We summarize the conclusions in the following theorem:

Theorem 11.2 (Stability of constrained linear quadratic MPC)**.** *Consider the linear quadratic MPC with linear constraints applied to the controllable system $x(k+1) = Ax(k)+Bu(k)$ 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 $x(k+1) = Ax(k) + B\kappa_N(x(k))$.*

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 modeling 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.

12 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 focused 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. Section 3 in [1] treats robust MPC, as does Sections 8.1-8.5 of [4].

Goals of this section:

- To understand some of the ideas used to extend MPC to the nonlinear case
- To understand what is meant by robust MPC

Learning outcomes:

- Understand and explain the basic principles of model predictive control, its pros and cons, and the challenges met in implementation and applications

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.

12.1 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 PhD thesis [6].

MPC based on linearization

The first thing that comes to mind is whether linear MPC can be applied to a linearized version of a nonlinear control design problem stemming from a nonlinear time-invariant model

$$x(k+1) = f(x(k), u(k)) \tag{158}$$

with inequality constraints

$$g(x(k), u(k)) \leq 0. \tag{159}$$

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 optimization problem for the *deviation variables*. At every time instant k, solve a constrained optimization problem.

Optimization problem:

$$\begin{aligned}
&\underset{\boldsymbol{\delta}\mathbf{u}(k),\boldsymbol{\delta}\mathbf{x}(k)}{\text{minimize}} && \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \delta x(k+i|k) \\ \delta u(k+i|k) \end{bmatrix}^\top W \begin{bmatrix} \delta x(k+i|k) \\ \delta u(k+i|k) \end{bmatrix} && \text{(160a)} \\
&\text{subject to} && \delta x(k|k) = \hat{x}(k) - x^r(k|k) && \text{(160b)} \\
& && \delta x(k+i+1|k) = A\,\delta x(k+i|k) + B\,\delta u(k+i|k), \quad i = 0,\dots,N-1 && \text{(160c)} \\
& && F\,\delta u(k+i|k) + G\,\delta x(k+i|k) + h \le 0; \quad i = 0,\dots,N-1. && \text{(160d)}
\end{aligned}$$

Obtained solution:

$$(\boldsymbol{\delta}\mathbf{u}(k), \boldsymbol{\delta}\mathbf{x}(k)) = \text{QP}_{\text{MPC}}(\hat{x}(k), \mathbf{u}^r(k), \mathbf{x}^r(k)). \tag{160e}$$

Here, we have simplified the formulation slightly by assuming no terminal penalty and no terminal constraint, and the weighting matrix W is given by $W = \text{diag}(Q, R)$. Also notice the notation introduced, namely k indicating the sampling instant and the index i indicating the time step within the prediction interval. Thus, $\boldsymbol{\delta}\mathbf{x}(k) = (\delta x(k|k), \dots, \delta x(k+N-1|k))$ and $\boldsymbol{\delta}\mathbf{u}(k) = (\delta u(k|k), \dots, \delta u(k+N-1|k))$ comprise the optimization variables over the prediction horizon at time k. They are all *deviation variables* relative to the *reference trajectory*, obeying the equation

$$x^r(k+i+1|k) = A\,x^r(k+i|k) + B\,u^r(k+i|k). \tag{161}$$

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 (160) can without any complications in the solution be generalized to the linear *time-varying* case. At every time instant k, solve a constrained optimization problem.

Optimization problem:

$$\begin{aligned}
&\underset{\boldsymbol{\delta}\mathbf{u}(k),\boldsymbol{\delta}\mathbf{x}(k)}{\text{minimize}} && \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \delta x(k+i|k) \\ \delta u(k+i|k) \end{bmatrix}^\top W_{k,i} \begin{bmatrix} \delta x(k+i|k) \\ \delta u(k+i|k) \end{bmatrix} && \text{(162a)} \\
&\text{subject to} && \delta x(k|k) = \hat{x}(k) - x^r(k|k) && \text{(162b)} \\
& && \delta x(k+i+1|k) = A_{k,i}\,\delta x(k+i|k) + B_{k,i}\,\delta u(k+i|k) + r(k+i|k) && \text{(162c)} \\
& && F_{k,i}\,\delta u(k+i|k) + G_{k,i}\,\delta x(k+i|k) + h_{k,i} \le 0; \quad i = 0,\dots,N-1. && \text{(162d)}
\end{aligned}$$

Obtained solution:

$$(\boldsymbol{\delta}\mathbf{u}(k), \boldsymbol{\delta}\mathbf{x}(k)) = \text{QP}_{\text{MPC}}(\hat{x}(k), \mathbf{u}^r(k), \mathbf{x}^r(k)). \tag{162e}$$

Obtained control:

$$u(k) = u^r(k|k) + \delta u(k|k). \tag{162f}$$

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 $r(k+i|k)$ added to cover the case when the reference trajectory does not satisfy the model equation, i.e.

$$r(k+i|k) = A_{k,i}\,x^r(k+i|k) + B_{k,i}\,u^r(k+i|k) - x^r(k+i+1|k). \tag{163}$$

Now, returning to our nonlinear model, a natural idea is to apply the linear, time-varying MPC above to a linearized 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 linearize around a reference trajectory. Given nonlinear system dynamics, $x(k+i+1|k) = f(x(k+i|k), u(k+i|k))$, and nonlinear inequality constraints, $g(x(k+i|k), u(k+i|k)) \leq 0$, we may obtain:

$$A_{k,i} = \frac{\partial f(x,u)}{\partial x}|_{x^r(k+i|k),u^r(k+i|k)} \quad B_{k,i} = \frac{\partial f(x,u)}{\partial u}|_{x^r(k+i|k),u^r(k+i|k)} \tag{164a}$$

$$G_{k,i} = \frac{\partial g(x,u)}{\partial x}|_{x^r(k+i|k),u^r(k+i|k)} \quad F_{k,i} = \frac{\partial g(x,u)}{\partial u}|_{x^r(k+i|k),u^r(k+i|k)} \tag{164b}$$

$$r(k+i|k) = f(x^r(k+i|k), u^r(k+i|k)) - x^r(k+i+1|k) \tag{164c}$$

$$h_{k,i} = g(x^r(k+i|k), u^r(k+i|k)). \tag{164d}$$

With these definitions, the equality constraint (162c) and the inequality constraint (162d) approximate the nonlinear system dynamics and inequality constraints in the vicinity of the reference trajectory.

As usual, a limitation when using a linearized 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 optimization 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 (158) is a discretization 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 linearizations 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 a constrained optimization problem.

Optimization problem:

$$\underset{\mathbf{u}(k),\mathbf{x}(k)}{\text{minimize}} \quad \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} x(k+i|k) - x^r(k+i|k) \\ u(k+i|k) - u^r(k+i|k) \end{bmatrix}^\top W_{k,i} \begin{bmatrix} x(k+i|k) - x^r(k+i|k) \\ u(k+i|k) - u^r(k+i|k) \end{bmatrix} \tag{165a}$$

$$\text{subject to} \quad x(k|k) = \hat{x}(k) \tag{165b}$$

$$x(k+i+1|k) = f(x(k+i|k), u(k+i|k)); \quad i = 0, \ldots, N-1 \tag{165c}$$

$$g(x(k+i|k), u(k+i|k)) \leq 0; \quad i = 0, \ldots, N-1. \tag{165d}$$

Obtained solution:

$$(\mathbf{u}(k), \mathbf{x}(k)) = \text{NLP}(\hat{x}(k), \mathbf{u}^r(k), \mathbf{x}^r(k)). \tag{165e}$$

Obtained control:

$$u(k) = u(k|k). \tag{165f}$$

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 fulfill 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, where the Newton direction for an equality constrained problem can be found by solving a QP based on a linearization of the problem at the current iterate. Similarly, in the SQP method a sequence of QPs are solved iteratively, each derived from the current linearization. Applied to our NMPC problem, the SQP method would look as follows.

At every time instant k, the SQP optimization variables $(\mathbf{u}_k, \mathbf{x}_k)$ are initialized as

$$(\mathbf{u}(k), \mathbf{x}(k)) = (\mathbf{u}^{\text{guess}}(k), \mathbf{x}^{\text{guess}}(k)).$$

Then the following QP is solved repeatedly at the current SQP iterate $(\mathbf{u}(k), \mathbf{x}(k))$ for the Newton correction $(\Delta\mathbf{u}(k), \Delta\mathbf{x}(k))$, which gives the next iterate by taking a (reduced) Newton step

$$(\mathbf{u}(k), \mathbf{x}(k)) \leftarrow (\mathbf{u}(k), \mathbf{x}(k)) + t(\Delta\mathbf{u}(k), \Delta\mathbf{x}(k)).$$

Optimization problem:

$$\begin{aligned}
\underset{\Delta\mathbf{u}(k),\Delta\mathbf{x}(k)}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta x(k+i|k) \\ \Delta u(k+i|k) \end{bmatrix}^\top H_{k,i} \begin{bmatrix} \Delta x(k+i|k) \\ \Delta u(k+i|k) \end{bmatrix} + J_{k,i}^\top \begin{bmatrix} \Delta x(k+i|k) \\ \Delta u(k+i|k) \end{bmatrix} && \text{(166a)} \\
\text{subject to} \quad & \Delta x(k|k) = \hat{x}(k) - x(k|k) && \text{(166b)} \\
& \Delta x(k+i+1|k) = A_{k,i}\, \Delta x(k+i|k) + B_{k,i}\, \Delta u(k+i|k) + r(k+i|k) && \text{(166c)} \\
& F_{k,i}\, \Delta u(k+i|k) + G_{k,i}\, \Delta x(k+i|k) + h_{k,i} \le 0; \quad i = 0, \dots, N-1. && \text{(166d)}
\end{aligned}$$

Obtained SQP solution:

$$(\mathbf{u}(k), \mathbf{x}(k)) = \text{SQP}(\hat{x}(k), \mathbf{u}^{\text{guess}}(k), \mathbf{x}^{\text{guess}}(k), u^r(k), x^r(k)). \tag{166e}$$

Obtained control:

$$u(k) = u(k|k). \tag{166f}$$

In the SQP algorithm given, $H_{k,i}$ is an approximation of the Hessian of the Lagrangian of the original NLP, with the negative directions (eigenvalues) removed. The Hessian is often approximated for computational reasons, even when being positive definite; 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 (166c)-(166d) correspond to the expressions in (164) but now evaluated at the current iterate. Finally,

$$J_{k,i} = W_{k,i} \begin{bmatrix} x(k+i|k) - x^r(k+i|k) \\ u(k+i|k) - u^r(k+i|k) \end{bmatrix}.$$

There is a striking similarity between the linear MPC QP in (162) and the QP in the SQP for the NMPC. In fact, if we initialize the SQP algorithm with the reference trajectory, i.e.

$$(\mathbf{u}(k)^{\text{guess}}, \mathbf{x}^{\text{guess}}(k)) = (\mathbf{u}^{\mathbf{r}}(k), \mathbf{x}^{\mathbf{r}}(k))$$

then it can be seen that *the first iteration* of the SQP algorithm is identical to the linear MPC QP and $(\boldsymbol{\delta}\mathbf{u}(k), \boldsymbol{\delta}\mathbf{x}(k)) = (\Delta\mathbf{u}(k), \Delta\mathbf{x}(k))$. Hence, with a full Newton update, the solutions are identical (compare (162f) for the linear MPC with the Newton update and (166f) for the NMPC).

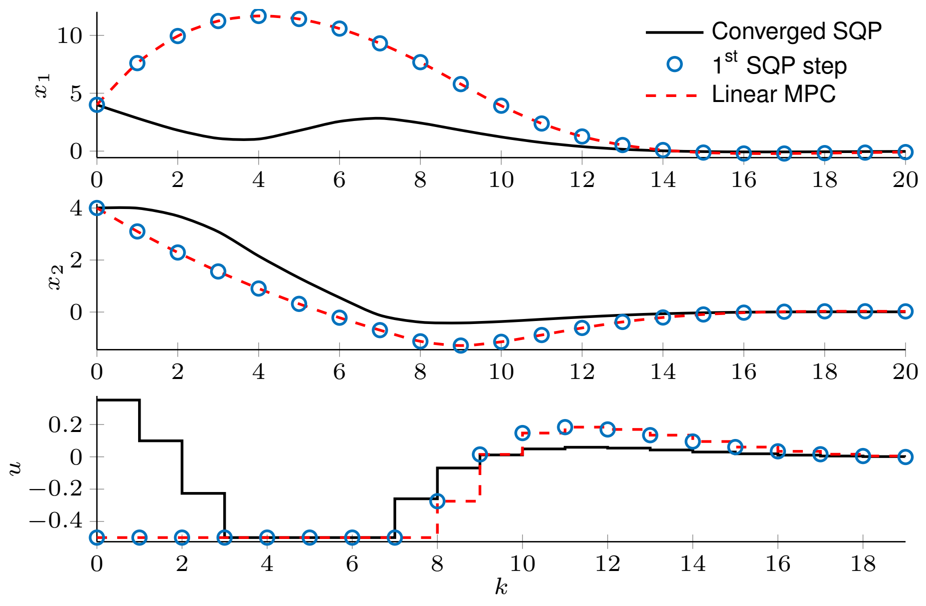


Figure 50: Comparison between receding horizon control solution from SQP after full convergence, SQP after one step and linear MPC.

Example 12.1. [Comparison between linear MPC and nonlinear MPC, [6]] Consider the following simple problem

$$\begin{aligned}
\underset{\mathbf{u}(k),\mathbf{x}(k)}{\text{minimize}} \quad & \sum_{i=0}^{N} \|x(k+i|k)\|^2 + 20\sum_{i=0}^{N-1} \|u(k+i|k)\|^2 && \text{(167a)}\\
\text{subject to} \quad & x(k|k) = \hat{x}(k) && \text{(167b)}\\
& x(k+i+1|k) = 0.9x(k+i|k) + \begin{bmatrix} \sin(\begin{bmatrix}0 & 1\end{bmatrix} x(k+i|k)) \\ u(k+i|k) + u(k+i|k)^3 \end{bmatrix} && \text{(167c)}\\
& |u(k+i|k)| \leq 0.5, \quad i = 0,\dots,N-1. && \text{(167d)}
\end{aligned}$$

In Figure 50, 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 analyzing 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 51 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 52 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 (164) have to be computed. This may lead to undesired long sampling periods of the

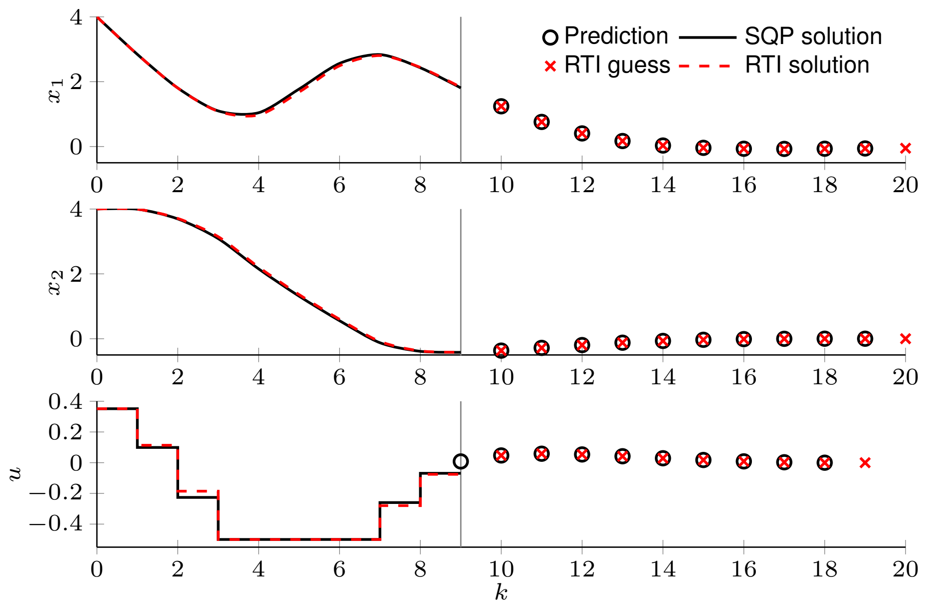


Figure 51: Real-time iteration scheme (RTI). The circles indicate predictions made by the MPC update. The crosses indicate the shifted predictions as an initial guess in RTI.

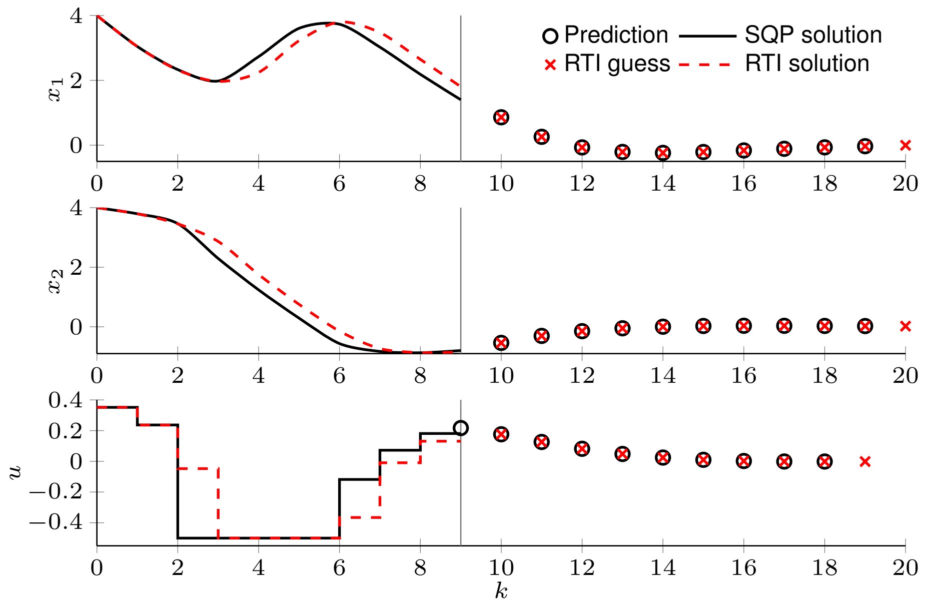


Figure 52: Real-time iteration scheme (RTI) in the presence of disturbances. The circles indicate predictions made by the MPC update. The crosses indicate the shifted predictions as an initial guess in RTI. The solid line shows the optimal solution for the disturbed state.

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:

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 favors an *approximate* solution that can be computed fast over an *exact* solution that requires more computations.

2. Computing the sensitivities in (164) 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.

Nonlinear continuous-time dynamics

So far, we have discussed nonlinear MPC on the basis of the nonlinear discrete-time system model (158). Normally, however, a nonlinear model that is based on first principles modeling 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 discretized dynamics and sensitivities as in (164). There are essentially two ways to approach this task.

First linearize, then discretize. 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, linearization is typically performed around a constant steady state or a reference trajectory, as seen earlier. In the nonlinear MPC case, linearization 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 discretize, then linearize. An alternative approach is to swap the order between linearization and discretization. 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.

12.2 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:

  $$x(k+1) = f(x(k), u(k)) + w(k)$$

  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

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

  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 12.2. [Feedback versus open-loop control, [1]] Consider again the simple integrator system

$$x(k+1) = x(k) + u(k) + w(k),$$

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 optimization problem

$$\min_{\mu(\cdot)} \max_{w \in [-1,1]} V_3(x, \mu, w),$$

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 optimization problem results either (by applying the batch solution) in the *open-loop control sequence*

$$u^*(0:2) = (-8/13 \cdot x(0), -3/13 \cdot x(0), -1/13 \cdot x(0))$$

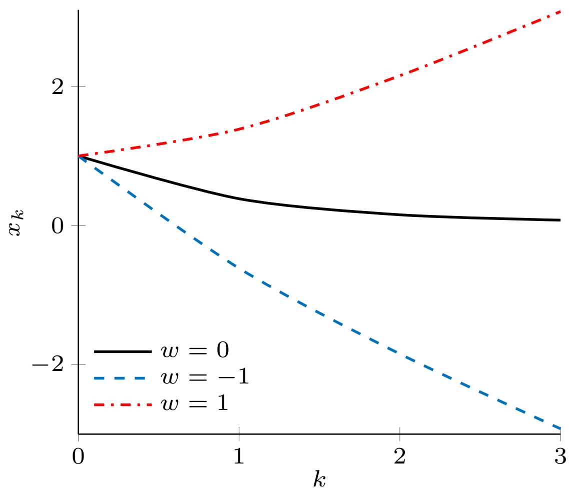
or (by applying DP) in the feedback policy

$$\mu(0:2) = (-8/13 \cdot x, -3/5 \cdot x, -1/2 \cdot x)\,.$$

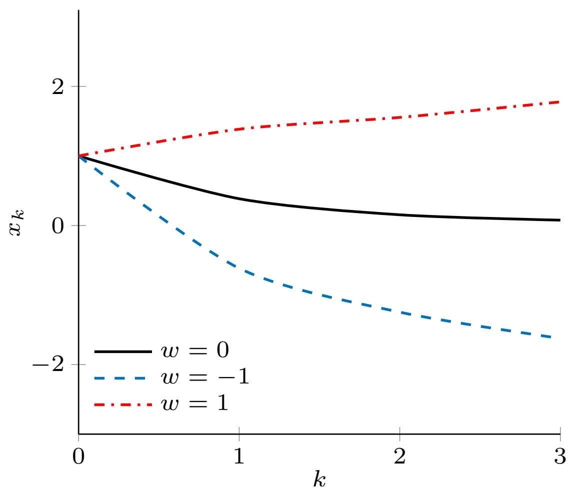
In the nominal case ($w = 0$), the two solutions are of course identical and can be shown to give the optimal state sequence

$$x^*(x) = (x, 5/13 \cdot x, 2/13 \cdot x, 1/13 \cdot x)\,,$$

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 53, 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. ■



(a) Open loop control.



(b) Closed-loop, feedback control.

Figure 53: Receding horizon control of an uncertain system with an additive process disturbance w.

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.

13 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. Section 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.

Goals of this section:

- To have an overview of important user choices for MPC
- To know some measures to improve computational efficiency

Learning outcomes:

- Understand and explain the basic principles of model predictive control, its pros and cons, and the challenges met in implementation and applications
- Describe basic properties of MPC controllers and analyze algorithmic details on very simple examples
- Understand and explain basic properties of the optimization problem as an ingredient of MPC, in particular concepts like linear, quadratic and convex optimization, optimality conditions, and feasibility

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 54. Each of the blocks shown in this diagram has some key characteristics that show similarities:

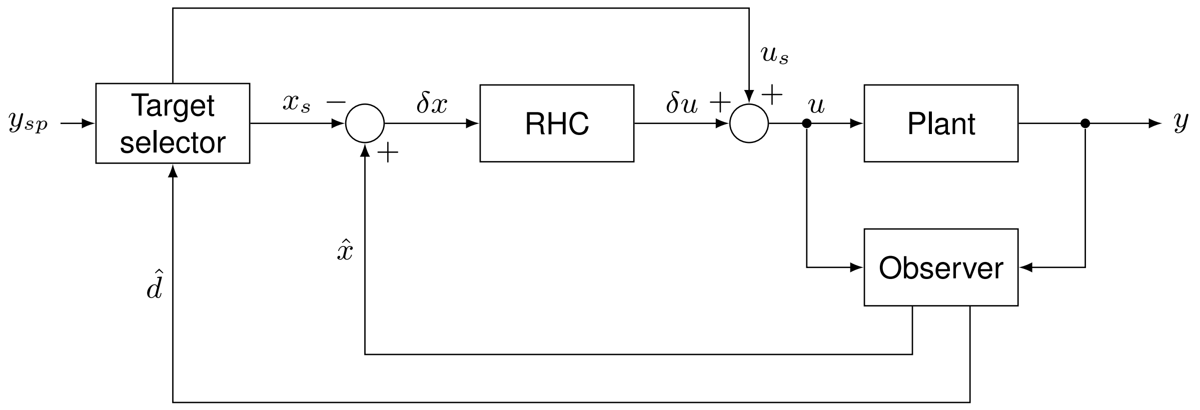


Figure 54: MPC block diagram with a receding horizon controller (RHC) working on deviation variables.

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 prior 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 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 optimization 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 behavior 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.

13.1 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
- Prediction and control horizons
- Constraints
- Disturbance models
- Observer dynamics
- Reference trajectories
- Steady state target calculations
- **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 horizon**s:
  - 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 fulfill!

- **Disturbance models and observers:**
  - Standard: step disturbances and off-set free control.
  - Disturbance feed-forward sometimes possible.
  - Stochastic disturbances: modeling $\rightarrow$ 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)!

13.2 Other variations of MPC

The state space model can easily be extended with **measurable disturbances**:

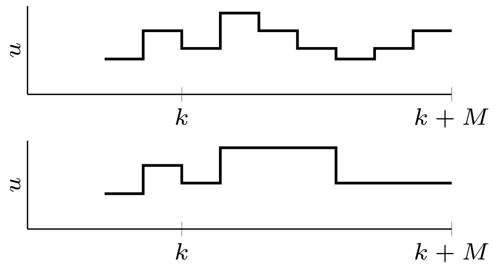
$$x(k+1) = Ax(k) + Bu(k) + B_d d_m(k).$$

- 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-parameterized:

- **Blocking**:

  Limit the number of optimization variables by keeping M low or by using *blocking*, i.e. keeping the control signal constant over several sampling intervals.



- **Basis functions**

$$u(k+i|k) = \sum_{j=1}^{n_u} c_j u_j(i),$$

  where $\{u_j(\cdot)\}$ are, e.g., polynomials. Once the coefficients $\{c_j\}$ have been determined (in the optimization step of the MPC), the control signal is defined over the whole prediction horizon. This is called *Predictive Functional Control (PFC).*

Stabilized 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.

As an input to the optimization in the MPC, the desired behavior 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).
- If the objective is based on economic or physical principles, it may be more convenient to define a generally non-quadratic cost function, e.g. one that minimizes energy, time or other costs.
- Combining the first 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.

13.3 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 optimization 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 optimization, 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 to 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 optimization solvers. As important as having algorithms that solve the optimization 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 optimization 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 optimization.
- Tailor-made code is typically much faster than Matlab.

Optimization tools

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 optimization 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 optimization, 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:

$$A_1 x_1 = b \quad A_2 x_2 = x_1 \quad \ldots \quad A_{k-1} x_{k-1} = x_{k-2} \quad A_k x = x_{k-1}. \tag{168}$$

The basic idea behind this is that the factorization gives factors $A_1, \ldots, A_k$ with a structure that permit fast solutions of the sub-problems. 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 sub-problems in (168) 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 (114). 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 13.1. [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 (113), 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 $p \times p$. ■

Ordering of variables

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 $z^\top = [\mathbf{x}^\top \quad \mathbf{u}^\top]$ gives rise to the equality constraint $F\mathbf{z} = h$ 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.

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 optimization*. Cambridge University Press 2004.

[6] Diehl, M. *Real-Time Optimization for Large Scale Nonlinear Processes.* PhD thesis, University of Heidelberg, 2001.

A Mathematical Background

In this appendix a brief background is provided on concepts used in the lecture notes. It is assumed that the reader is familiar with basic concepts in linear algebra.

A.1 Operations with matrices

Lemma A.1 (Completing the square)**.** *The expression $u^\top Hu + 2f^\top u$ where H is an invertible matrix and f and u are vectors of proper sizes, can be written as*

$$u^\top Hu + 2f^\top u = (u + H^{-1}f)^\top H(u + H^{-1}f) - f^\top H^{-1}f,$$

also known as ***completing the square****.*

Lemma A.2 (Solution to a system of equations by block-Gaussian elimination)**.** *The system of equations*

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}$$

has the solution

$$\begin{aligned} x &= A^{-1}p - A^{-1}B(D - CA^{-1}B)^{-1}(q - CA^{-1}p) \\ y &= (D - CA^{-1}B)^{-1}(q - CA^{-1}p) \end{aligned}$$

if A and $D - CA^{-1}B$ are invertible, or

$$\begin{aligned} x &= (A - BD^{-1}C)^{-1}(p - BD^{-1}q) \\ y &= D^{-1}q - D^{-1}C(A - BD^{-1}C)^{-1}(p - BD^{-1}q) \end{aligned}$$

if D and $A - BD^{-1}C$ are invertible.

Proof. Using block-Gaussian elimination, multiply the first row by $-CA^{-1}$ and add it to the second row, to obtain

$$\begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} p \\ q - CA^{-1}p. \end{bmatrix}$$

If $D - CA^{-1}B$ is invertible, then

$$\begin{aligned} y &= (D - CA^{-1}B)^{-1}(q - CA^{-1}p) \\ x &= A^{-1}(p - By). \end{aligned}$$

For the case with D and $A - BD^{-1}C$ invertible, the system can be first solved for x by multiplying the second row with $-BD^{-1}$ and then adding this to the first row. □

Definition A.1 (Kronecker product)**.** *If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the* ***Kronecker product*** *$A \oplus B$ is the $pm \times qn$ block matrix*

$$A \oplus B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Definition A.2 (Matrix exponential)**.** *The* ***matrix exponential*** *is defined as*

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k.$$

A.2 Properties of matrices

Definition A.3 (Determinant)**.** *The **determinant** is a scalar value that characterizes some properties of a square matrix A. The determinant of a 2×2 matrix is*

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc.$$

The following properties hold for determinants:

- *The determinant is nonzero if and only if the matrix is invertible,*
- $\det(A) = \det(A^\top)$*,*
- $\det(cA) = c^n \det(A)$*, for an $n \times n$ matrix A,*
- $\det(AB) = \det(A)\det(B)$*, for square matrices A and B,*
- $\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \det(A)\det(D - CA^{-1}B)$*, if A is invertible,*
- $\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \det(A - BD^{-1}C)\det(D)$*, if D is invertible,*
- $\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \det(AD - BC)$*, if A, B, C, D are square matrices of the same size,*
- $\det(X + AB) = \det(X)\det(I + BX^{-1}A)$*, if X is invertible. This is also known as the generalized Sylvester's determinant theorem.*

Definition A.4 (Eigenvalues and eigenvectors)**.** *An **eigenvector** v of a linear transformation Av, where A is a square matrix, is a nonzero vector that changes at most by a scalar factor, the **eigenvalue** λ, when that linear transformation is applied to it, i.e.,*

$$Av = \lambda v. \tag{169}$$

The eigenvalue identity (169) has a solution only if the **determinant** of $A - \lambda I$ is zero, i.e.

$$\det(A - \lambda I) = 0.$$

Definition A.5 (Trace)**.** *The trace of a square matrix A is the sum of its diagonal elements*

$$\operatorname{trace}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i.$$

It is also equal to the sum of its eigenvalues.

Lemma A.3 (Determinant and eigenvalues)**.** *The determinant can be computed as the product of the matrix eigenvalues*

$$\det(A) = |A| = \prod_{i=1}^{n} \lambda_i.$$

Lemma A.4 (Rank of a product)**.** *For any two matrices A and B, it holds*

$$\operatorname{rank}(AB) \leq \min(\operatorname{rank}(A), \operatorname{rank}(B)).$$

A.3 Matrix inversion

Let Q be an invertible matrix partitioned as

$$Q = \begin{bmatrix} A & B \\ C & D \end{bmatrix}.$$

Lemma A.5 (Inverse of a partitioned matrix 1)**.** *If A is invertible, then Q^{-1} can be written as*

$$Q^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BZCA^{-1} & -A^{-1}BZ \\ -ZCA^{-1} & Z \end{bmatrix} \tag{170}$$

where $Z = (D - CA^{-1}B)^{-1}$.

Proof. Let

$$Q^{-1} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}.$$

Then, it holds

$$QQ^{-1} = \begin{bmatrix} AW + BY & AX + BZ \\ CW + DY & CX + DZ \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{171}$$

From the $(1, 2)$ block matrix identity it follows

$$X = -A^{-1}BZ.$$

Replace this in the $(2, 2)$ identity to obtain

$$Z = (D - CA^{-1}B)^{-1}.$$

From the $(1, 1)$ block matrix identity it follows $W = A^{-1} - A^{-1}BY$. Replace this in the $(2, 1)$ identity to obtain

$$Y = -ZCA^{-1}.$$

Finally, the value for W can be obtained as

$$W = A^{-1} + A^{-1}BZCA^{-1}$$

which completes the proof.

□

Lemma A.6 (Inverse of a partitioned matrix 2)**.** *If D is invertible, then Q^{-1} can be written as*

$$Q^{-1} = \begin{bmatrix} W & -WBD^{-1} \\ -D^{-1}CW & D^{-1} + D^{-1}CWBD^{-1} \end{bmatrix} \tag{172}$$

where $W = (A - BD^{-1}C)^{-1}$.

Proof. From the $(2, 1)$ block matrix identity in (171) it follows $Y = -D^{-1}CW$. Replace this in the $(1, 1)$ identity to obtain

$$W = (A - BD^{-1}C)^{-1}.$$

From the $(2, 2)$ block matrix identity it follows $Z = D^{-1} - D^{-1}CX$. Replace this in the $(1, 2)$ identity to obtain

$$X = -WBD^{-1}.$$

□

Definition A.6 (Schur-Complement)**.** *The expression*

$$A - BD^{-1}C$$

is known as the ***Schur-Complement*** *of* D*.*

Corollary A.1. *The inverse of a* 2×2 *matrix* $Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ *is*

$$Q^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Proof. The proof follows directly from Lemma A.5 or Lemma A.6. □

Lemma A.7 (Matrix inversion lemma 1)**.** *The inverse of* $(A + \tilde{B}\tilde{D}C)$ *can be written as*

$$(A + \tilde{B}\tilde{D}C)^{-1} = A^{-1} - A^{-1}\tilde{B}(\tilde{D}^{-1} + CA^{-1}\tilde{B})^{-1}CA^{-1}, \tag{173}$$

also known as the ***Woodbury matrix identity****.*

Proof. Compare the $(1, 1)$ entries in (170) and (172) and substitute $\tilde{B} = -B$, $\tilde{D} = D^{-1}$. □

Lemma A.8 (Matrix inversion lemma 2)**.** *Another version of the* ***Matrix inversion lemma*** *is*

$$(A + \tilde{B}\tilde{D}C)^{-1}\tilde{B}\tilde{D} = A^{-1}\tilde{B}(\tilde{D}^{-1} + CA^{-1}\tilde{B})^{-1}.$$

Proof. Compare the $(1, 2)$ entries in (170) and (172) and substitute $\tilde{B} = -B$, $\tilde{D} = D^{-1}$. □

Lemma A.9 (Matrix inversion lemma: special case)**.** *The inverse of* $(I + P)$ *can be written as*

$$(I + P)^{-1} = I - (I + P)^{-1}P = I - P(I + P)^{-1}.$$

Proof. It holds

$$I = (I + P)^{-1}(I + P) = (I + P)^{-1} + (I + P)^{-1}P.$$

The lemma follows directly from this expression. □

Lemma A.10 (Sherman–Morrison formula)**.** *The inverse of* $(A + uv^\top)$ *can be written as*

$$(A + uv^\top)^{-1} = A^{-1} - \frac{A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u}$$

where u *and* v *are vectors of proper sizes.*

Proof. Substitute $\tilde{B} = u$, $\tilde{D} = 1$ and $C = v^\top$ in (173). □

A.4 Matrix differentiation

Definition A.7 (Jacobian)**.** *The derivative of a general differentiable vector function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ is*

$$\frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, \tag{174}$$

which is called the ***Jacobian*** *of f.*

Definition A.8 (Gradient)**.** *The* ***gradient*** *of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as the column vector*

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \left(\frac{\partial f(x)}{\partial x}\right)^\top .$$

Notice that the function derivative with respect to x according to (174) is actually a row vector. However, the vector notation (the gradient) is more commonly used in literature, and, here, the nabla operator ∇_x will specifically refer to the vector notation.

Definition A.9 (Hessian)**.** *The* ***Hessian*** *matrix of a scalar function f is defined as*

$$\nabla_x^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Lemma A.11. *The derivative of a linear function $f = Ax$, where f and x are vectors and A is a matrix, all of conforming sizes, is*

$$\frac{\partial f}{\partial x} = \frac{\partial (Ax)}{\partial x} = A.$$

Lemma A.12. *The derivative of $x^\top A$ is*

$$\frac{\partial (x^\top A)}{\partial x} = A^\top.$$

Proof. Transpose both sides in A.11 to obtain the identity. □

Lemma A.13. *The derivative of a quadratic function $x^\top Ax$ is*

$$\frac{\partial (x^\top Ax)}{\partial x} = x^\top A + x^\top A^\top = x^\top (A + A^\top).$$

If the matrix A is symmetric, this expression reduces to

$$\frac{\partial (x^\top Ax)}{\partial x} = 2x^\top A.$$

Lemma A.14 (Solution to a linear differential equation)**.** *The linear differential equation*

$$\dot{x}(t) = Ax(t) + b \tag{175}$$

where A, x and b are of conforming sizes, has the solution

$$x(t+h) = e^{Ah}x(t) + \int_0^h e^{A\tau} \mathrm{d}\tau \, b$$

at some time instant $t + h$.

Proof. Multiply both sides of (175) with e^{-At} and rearrange to obtain

$$e^{-At}\dot{x}(t) - e^{-At}Ax(t) = e^{-At}b.$$

Observe that

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(e^{-At}x(t)\right) = e^{-At}\dot{x}(t) - e^{-At}Ax(t).$$

Hence, integrating both sides from t to $t+h$ gives

$$e^{-A(t+h)}x(t+h) - e^{-At}x(t) = \int_t^{t+h} e^{-As}\mathrm{d}s\, b.$$

Multiply both sides with $e^{A(t+h)}$ and rearrange,

$$x(t+h) = e^{Ah}x(t) + \int_t^{t+h} e^{A(t+h-s)}\mathrm{d}s\, b.$$

Finally, change the integration variable to $\tau = t+h-s$ to complete the proof. $\square$

A.5 Probability theory

Definition A.10. *If $\mathbb{P}(B) > 0$ then the* ***conditional probability*** *that A occurs given that B occurs is defined to be*

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

The probability $\mathbb{P}(A \mid B)$ is read as probability of A given B. The probability $\mathbb{P}(A \cap B)$ is called **joint probability**.

It follows: $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$.

Definition A.11 (Law of total probability). *The law of total probability states that*

$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A \mid B_i)\,\mathbb{P}(B_i).$$

Definition A.12 (Bayes formula). *The Bayes formula can be stated as*

$$\mathbb{P}(B_j \mid A) = \frac{\mathbb{P}(A \mid B_j)\,\mathbb{P}(B_j)}{\sum_{i=1}^{n} \mathbb{P}(A \mid B_i)\,\mathbb{P}(B_i)}.$$

Definition A.13. *The* ***expectation*** *of a continuous random variable X with density function f is given by*

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x f(x)\mathrm{d}x.$$

Lemma A.15. *If X and Y are* ***independent*** *then $\mathbb{E}(XY) = \mathbb{E}(X)\,\mathbb{E}(Y)$.*

Proof.

$$\mathbb{E}(XY) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_{XY}(x,y)\mathrm{d}x\mathrm{d}y = \int_{-\infty}^{\infty} x f_X(x)\mathrm{d}x \int_{-\infty}^{\infty} y f_Y(y)\mathrm{d}y = \mathbb{E}(X)\,\mathbb{E}(Y).$$

□

Theorem A.1 (Cauchy-Schwarz inequality). *For random variables X and Y,*

$$\mathbb{E}(XY)^2 \leq \mathbb{E}(X^2)\,\mathbb{E}(Y^2)$$

with equality if and only $\mathbb{P}(aX = bY) = 1$ for some real a and b, at least one of which is non-zero.

Proof. Let $Z = aX - bY, b \neq 0$. Then,

$$0 \leq \mathbb{E}(Z^2) = \mathbb{E}(a^2X^2 - abXY + b^2Y^2) = a^2\,\mathbb{E}(X^2) - ab\,\mathbb{E}(XY) + b^2\,\mathbb{E}(Y^2).$$

Since it is nonnegative, it has at most one real root for a, hence its discriminant is less than or equal to zero, i.e.

$$\mathbb{E}(XY)^2 - \mathbb{E}(X^2)\,\mathbb{E}(Y^2) \leq 0.$$

The discriminant is zero if and only if the quadratic equation has a real root, i.e. $\mathbb{E}(Z^2) = 0$ for some a and b. The expected value can also be written as $\mathbb{E}(Z^2) = \sum_x x\,\mathbb{P}(X = x) = 0$. This implies that $\mathbb{P}(X = x) = 0$ for $x \neq 0$. Hence, $\mathbb{P}(X = 0) = 1$. □

Important properties of the expectation operator are:

- if $X > 0$, then $\mathbb{E}(X) \geq 0$,

- if $a, b \in \mathbb{R}$ then $\mathbb{E}(aX + bY) = a\,\mathbb{E}(X) + b\,\mathbb{E}(Y)$.

Definition A.14. *The* ***variance*** *and* ***standard deviation*** *of X are defined as*

$$\operatorname{var}(X) = \mathbb{E}(X - \mathbb{E}(X))^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2, \quad \sigma_X = \sqrt{\operatorname{var}(X)}.$$

Definition A.15. *The* ***covariance*** *of X and Y is defined as*

$$\operatorname{cov}(X, Y) = \mathbb{E}((X - \mathbb{E}(X))(Y - \mathbb{E}(Y))) = \mathbb{E}(XY) - \mathbb{E}(X)\,\mathbb{E}(Y).$$

The ***correlation (coefficient)*** *of X and Y is*

$$\rho(X, Y) = \frac{\operatorname{cov}(X, Y)}{\sqrt{\operatorname{var}(X)\operatorname{var}(Y)}}$$

as long as the variances are non-zero.

Theorem A.2. *For random variables X and Y,*

- $\operatorname{var}(aX) = a^2 \operatorname{var}(X)$ *for* $a \in \mathbb{R}$*,*
- $\operatorname{var}(X + Y) = \operatorname{var}(X) + \operatorname{var}(Y)$ *if X and Y are uncorrelated.*

Proof.

- $\operatorname{var}(aX) = \mathbb{E}((aX - \mathbb{E}(ax))^2) = a^2\,\mathbb{E}(X - \mathbb{E}(X)) = a^2 \operatorname{var}(X)$.
- $\operatorname{var}(X+Y) = \mathbb{E}((X+Y-\mathbb{E}(X+Y))^2) = \mathbb{E}((X-\mathbb{E}(X)^2+2(XY-\mathbb{E}(X)\,\mathbb{E}(Y))+(Y-\mathbb{E}(Y)^2)$ $= \operatorname{var}(X) + \operatorname{var}(Y)$.

$\square$