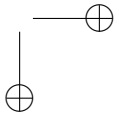
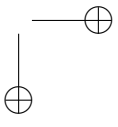
Model predictive control for renewable energy systems
Lecture Notes at University of Freiburg

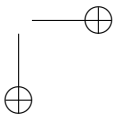
Lilli Frison, Jochem De Schutter and Moritz Diehl

Preliminary Version of
17 August 2023



Contents





Chapter 1

Introduction

1.1 A short primer on Model Predictive Control

If you split the term "Model Predictive Control" (MPC) into its meaningful parts, we obtain the following two distinctive ingredients.

1. Model-based: As the name implies, MPC is based on a model of the process. This model can be developed and represented in various forms suitable for control.

2. Predictive control: Using a simulation of its internal model, MPC computes a prediction of the future values of the process outputs and the states from the current time.

The third distinct ingredient of MPC is optimization that is used to calculate the actuated values which result in an optimal predicted state trajectory. The objective of the optimization problem can vary from tracking a setpoint or executing a certain task in minimal time, to economic objectives where some more general performance index is minimized. In MPC, the procedure of looking into the future using the process model followed by the formulation and solution of an optimization problem is repeated at every sampling time step, as new information on the state of the system becomes available. This creates a feedback action and is commonly known as a sliding horizon strategy or receding horizon control, see Figure 1.1.

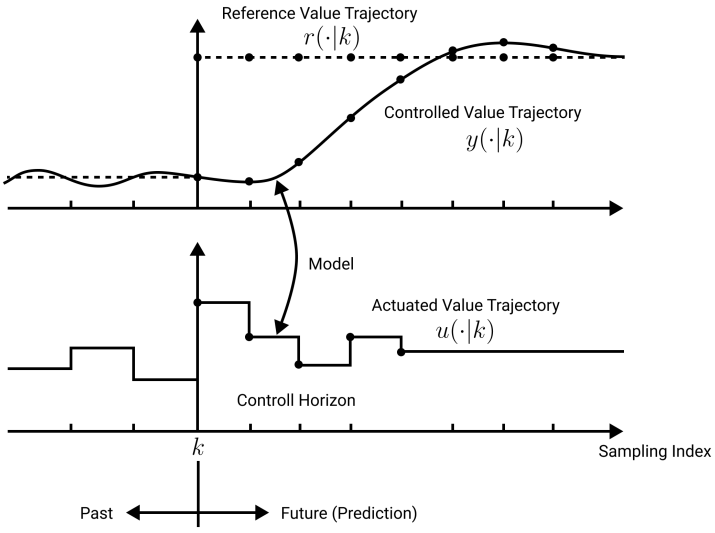


Figure 1.1: Working principle of MPC

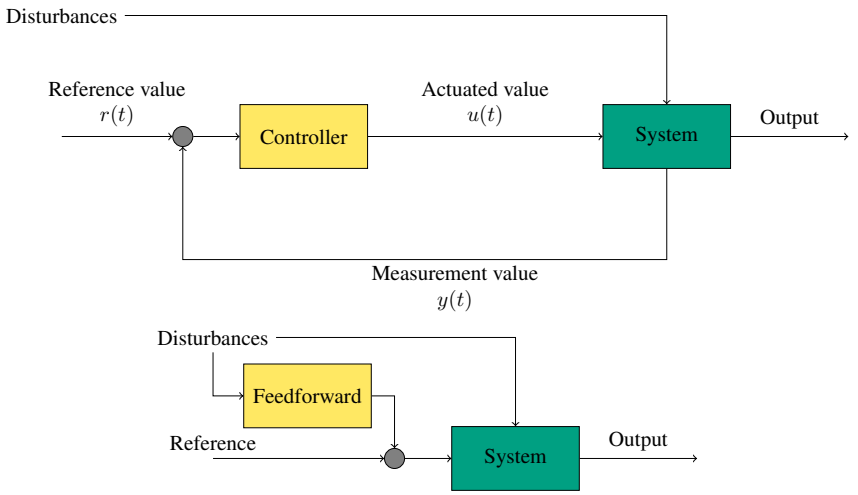


Figure 1.2: Feedback (top) and feedforward (bottom) control loop diagrams.

Feedback is a term that most people are familiar with. In the classical feedback control loop, the measured output of the process (called sensor values) is compared with a reference signal. The error between both is given to the controller which generates a manipulated variable (called actuator or actuated value) to control the process. An example is the PID controller that applies a correction based on proportional, integral, and derivative terms of the error value.
A common example is the thermostat in your home. It senses the air temperature in the room and increases the heating input if this temperature drops. When the air reaches the desired room temperature, the heating input goes back to zero. The room temperature sensor provides the feedback in this system and it allows a thermostat to effectively regulate the temperature of our homes. However, the result is a system with a very long control loop time, or in other words, it reacts with a lot of lag in case of disturbances.
A different approach is feedforward control, for which some kind of model describing the disturbance is needed. Figure 1.2 show the flow charts of a feedforward and a feedback controller. For example, if the thermostat detects a disturbance, such as a sensor showing a change in outside temperature or an open window, the model might extend the duration the heater is running or increase its temperature to compensate. This system is an example of a feedforward system and allows for very fast control loop times because adjustments are made to the inputs of the system before a disturbance can affect the output. A simple but commonly used example that reacts to changes in the outside temperature is a heating curve controller.
Compared to classical control structures, in the MPC controller, the feedforward and feedback action are combined in a single controller, as illustrated in Fig. 1.3. Typically, MPC is used as a state-feedback controller, which means that the control action is based on the recent states of the system. These states are necessary, as they are used for the prediction of the output values in the process model. Either the system states are all measurable or, more commonly, a state observer has to be applied to estimate the system states. MPC is particularly suitable for constrained systems and for control problems where the offline computation of the control function is very difficult or even impossible. One of the great advantages of MPC is its inherent ability to deal with the system constraints, which makes these methods very interesting for industry. Due to its great flexibility, MPC is probably the method with the most practical applications among modern control algorithms.
The following list summarizes the benefits associated with MPC:

1. System model for anticipatory control actions.
2. Integration of a disturbance model for disturbance rejection.
3. Ability to handle constraints and uncertainties.
4. Ability to handle time-varying dynamics.
5. Ability to cope with slow-moving processes with time delay.
6. Integration of energy conservation strategies in the controller formulation.

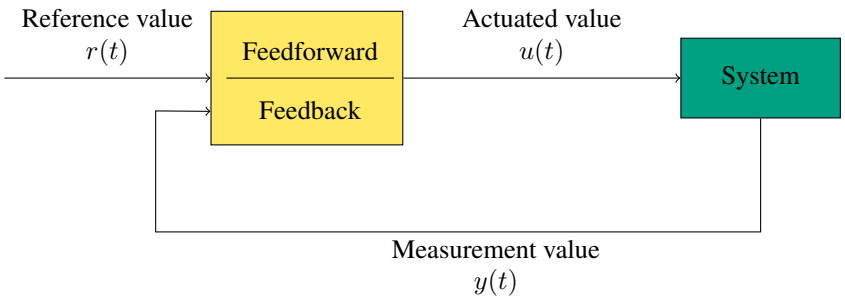


Figure 1.3: MPC can be interpreted as a combination of feedforward and feedback action in one single controller.

7. Use of a cost function for achievement of multiple objectives.
8. Use of advanced optimization algorithms for computation of control vectors.

In this course, our goal is to learn in detail about model predictive control concepts and, in particular, to apply it to the control of renewable energy systems.

1.2 Why Model Predictive Control of Renewable Energy Systems?

Along with the climate goals of the Paris Agreement, the national greenhouse gas strategies of industrialized countries involve the total restructuring of their energy systems. It is generally agreed that only the Net Zero Emissions by 2050 Scenario (NZE) can limit the global warming sufficiently. The NZE scenario sets out a pathway for the global energy sector to achieve net zero CO2 emissions by 2050. It doesn't rely on emissions reductions from outside the energy sector to achieve its goals[1]. For the worldwide electricity sector, the NZE scenario implies an enormous increase of the share of renewable forms of electricity generation, made possible by a massive expansion in solar PV and wind capacity.

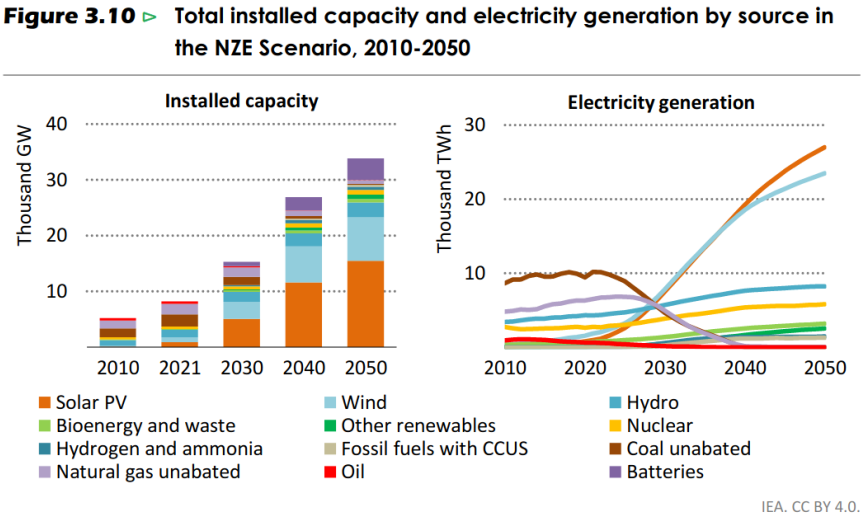


Figure 1.4: Global electricity sector outlook in the NZE scenario, figure from `https://www.iea.org/reports/net-zero-by-2050`.

In Germany, the climate protection plan (Klimaschutzplan 2050) was developed in 2016 by the German government to help reduce greenhouse gas emissions and combat climate change. It sets out a vision for a

[1] `https://www.iea.org/reports/net-zero-by-2050`

low-carbon, climate-resilient economy by 2050 with the main goal of reducing Germany's greenhouse gas emissions by 80-95% below 1990 levels by 2050. With the amendment to the climate protection plan in 2021, the German government has tightened climate protection targets and anchored the goal of greenhouse gas neutrality by 2045. Emissions are to be reduced by 65% by 2030 and 88% by 2040 (compared with 1990). To achieve this target, the widespread adoption of renewable energy sources such as wind, solar, and hydropower, is necessary.
To date, an ever larger share of our energy needs is increasingly being met by renewable energies: in 2022, around 46% of gross electricity consumption of which more than half can be attributed to wind energy, one fourth to solar energy and the rest to hydro power, renewable waste and biomass, see Figure 1.5.

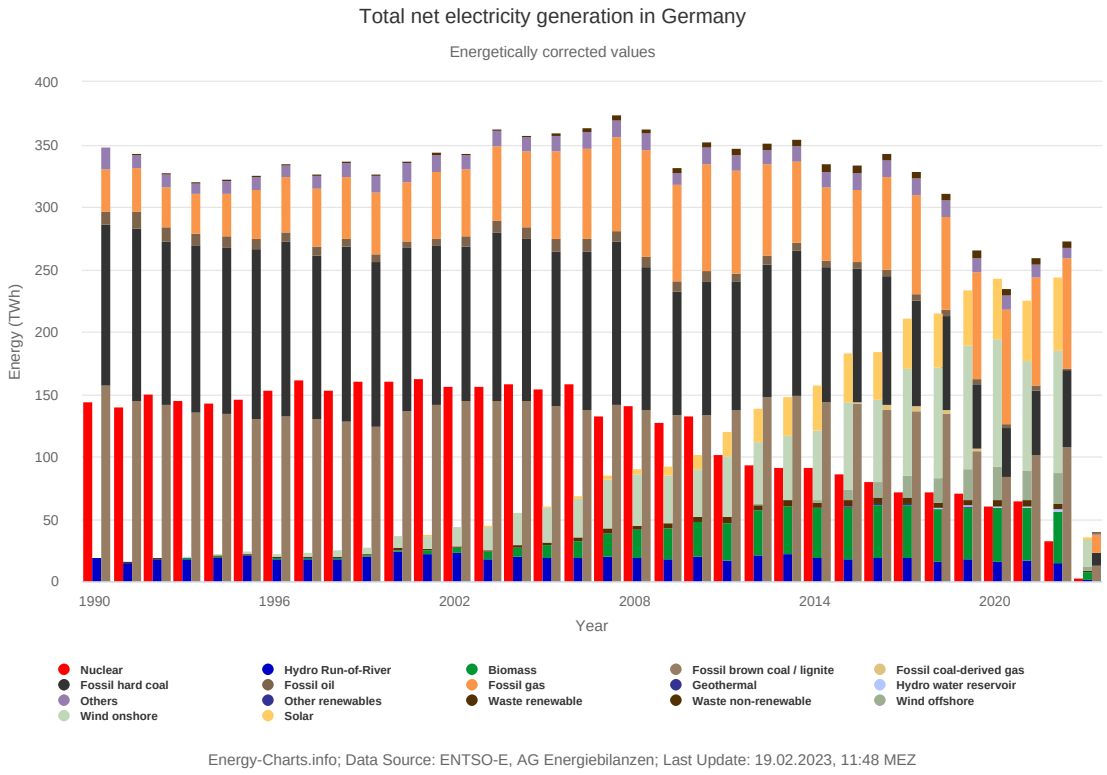


Figure 1.5: Annual total net power generation in Germany from 1990 to 2023 (partly) grouped by type of energy source (nuclear, renewable, fossil).

By 2030, this figure is expected to rise to at least 80%.
With the widespread adoption of a high share of renewable energy sources, today's and the future energy system is faced with new challenges:

- Integration of Renewable energy sources: Renewable energy sources are volatile, how to integrate them into out energy system? How can security of supply and grid stability be guaranteed?
- Energy storage: Another major challenge is the need for effective energy storage solutions to manage the intermittency of renewable energy sources. Batteries and other storage technologies will be crucial to balancing energy supply and demand.
- Energy Efficiency: Improving energy efficiency is a critical challenge, as reducing energy demand can help to alleviate the strain on the energy system and reduce greenhouse gas emissions. This requires the development and adoption of more efficient technologies, as well as changes in behavior and lifestyle.

Besides these technological challenges, the German energy transition faces other types of challenges, such as resistance from some communities to the installation of wind turbines, and the need for upgrading the country's energy infrastructure to accommodate the fluctuating nature of renewable energy sources.
In addition to increasing the energy efficiency of the energy system, smart operation of the power and heat generation and distribution systems and their components is very important. Classical controllers are often based on heuristic rules or traditional PID controllers, which neither control the system optimally nor consider all available data and predictions. At the same time, the objectives of the control problems and the constraints to be met are becoming more and more complex - think of sector coupling and the fluctuating production of renewable energies, heat recovery and waste heat recovery, very different user requirements in the building sector, prosumer-based, decentralized and low-temperature heating networks. Renewable-based energy system are highly diverse and complex multi-physics systems, often with complicated characteristics such as a challenging structural requirements and a high nonlinearity (wind energy), mixed-integer control variables (heating networks, operational constraints of heating plants) or combined fast and slow dynamics (heating network with seasonal storage, PV plants with converters).
Model predictive control is an obvious candidate technology to handle the increasing complexity of energy systems and technologies (such as e.g. heat pumps, PV/PVT, seasonal storages, airborne wind energy).

1.3 Recommended Literature

For additional readings, we recommend the following literature (free PDFs online).
Model Predictive Control:

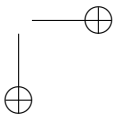
- Rawlings and Diehl – Model Predictive Control: Theory and Design [10]
- Grüne and Pannek – Nonlinear Model Predictive Control [5]
- On Linear and Hybrid Systems: Borelli, Bemporad, Morari – Predictive Control for Linear and Hybrid Systems [2]
- More a control perspective: Maciejowski – Predictive Control with Constraints [7].

Numerical optimization:

- Nocedal and Wright – Numerical Optimization [8].

Simulation methods:

- Chapra – Numerical Methods for Engineers [3].



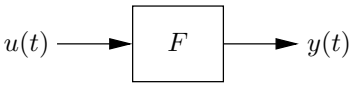
Chapter 2

Dynamic Systems Modelling

2.1 Fundamentals of Dynamic Systems Modelling

2.1.1 Introduction to Dynamic Systems

In this lecture, our major aim is to model *dynamic systems*, i.e. processes that are evolving with time. These systems can be characterized by *states* x and sometimes parameters p that allow us to predict the future behavior of the system. If the state and the parameters are not known, we first need to *estimate* them based on the available measurement information. We consider *controlled dynamic systems* in continuous or discrete time that depend on a parameter u that can change depending on the time and/or the state of the system. This parameter can be understood either as a control variable that can be actively influenced from the outside (e.g. acceleration in a vehicle) or as a disturbance that acts on the system but that cannot be influenced (e.g. road bumps in a car, outside temperature change). The ultimate purpose of modelling and system identification is to understand the system better and to be able to design and test good control strategies.
From a generic point of view, a (controlled) dynamic system responds to an input signal $u(t)$ with an output signal $y(t)$ as depicted in the following block diagram

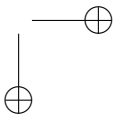


This behavior could be regarded as a 'mapping in time domain' of a function $u : t \mapsto u(t)$ to a function $y : t \mapsto y(t)$,

$$u \mapsto y = F\{u\} \tag{2.1}$$

Apart from controlled dynamic systems, there exist many systems that cannot be influenced at all, but that only evolve according to their intrinsic laws of motion. These *uncontrolled systems* have an empty control set, $\mathbb{U} = \emptyset$. If a dynamic system is both uncontrolled and time-invariant it is also called an *autonomous system*.
Any dynamic system evolves over time, but time can come in two variants: while the physical time is continuous and forms the natural setting for most technical and biological systems, other dynamic systems can best be modelled in discrete time, such as digitally controlled sampled-data systems, or games.
Continuous time systems can often be described by physical laws using ordinary differential equations (ODE), but sometimes they are described by other forms of more complex differential equations such as differential-algebraic equations (DAE), partial differential equations (PDE) or delay differential equations (DDE). As a part of the course, only ODE systems are covered.
Let us give some examples of systems that can be described by an ODE and write down for each some of the typically needed states.

- Race cars: states: position (x, y), orientation, velocity $(\dot{x}, \dot{y})$, angular velocity, control: gas pedal, wheel.

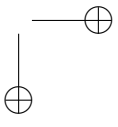


- Quadcopter: states: 3D position + orientation (6 states), 6 derivative, control: propeller rotation speeds.
- Satellite: states: 3D position + 3 velocities, control: force in each of 3 dimensions
- Airplanes in free flight:3D position + orientation, derivatives (velocities) of all these 6.
- Wind turbine: states: rotor position and speed, control: pitch of rotor blades, generator torque.
- Building model: states: temperature of the zone(s), temperature of the wall(layers), control: power of heating device.

Starting from the continuous-time system, all numerical simulation methods have to discretize the time interval in some form or other and thus effectively generate discrete time systems that can be implemented in digital devices like computers, microcontrollers and other forms of microprocessors. As we will see later, this can be done exactly for linear systems. For nonlinear systems, numerical integration methods which are subject to approximation errors have to be applied.

Modelling approaches There are many classifications of mathematical models. One is to divide mathematical models into two categories, based on
- physical theory (also called first principles models, white-box models)
- empirical descriptions (also called empirical models, black box models, data-driven models).

First principles models are based on physical and chemical laws: conservation laws like mass balance, momentum balance and energy balance, or chemical reaction kinetics. These models allow for a detailed simulation of the processes involved, such as thermodynamics and aerodynamics, and lead to a *continuous-time state-space model*.
In some cases, physics-based models are too complex to be used in a real-time optimization setting. In other cases, the underlying physical equations are not known precisely. In these cases, and if measurements are available, *data-based black-box models* can be used. A black-box model of a system is one that does not use any particular prior knowledge of the character or physics of the relationships involved. It is therefore more a question of a "curve-fitting" regression problem than "modeling". The basis for these models consists of input/output data derived from measurements. With these experimental data, a candidate model is chosen that is usually given in a parametrized form. Through the use of the measured data set, the parameters are estimated. Depending on the specific model type, black-box models can reproduce the system dynamics with a model of low complexity. The drawback is that a large amount of measurement data is needed for the calibration of the blackbox process model and that usually only a very limited extrapolation capability is given. In addition, no state estimation of physically meaningful quantities is possible.
Examples of such models used in process control are input-output models and transfer function models. As the name implies, with the input-output, the current output is dependent on the values of the past outputs and/or the past inputs of the process. The notion or concept of states of a process are not used. These model types can be derived from the state-space models or purely from measurement data. An example of an input-output model is the Auto Regressive Model with Exogenous Inputs (ARX model) where the output is a weighted sum of the past inputs and past outputs. Transfer function models describe the relationship between the inputs and outputs of a system in the frequency domain using a ratio of polynomials and are characterised by their poles and zeros. Most of these models are for linear single-input-single-output (SISO) systems. It is much harder to derive nonlinear parameterized function models. In this case, models from supervised machine learning such as artificial neural networks can be useful.
Another approach that is useful in modelling renewable energy systems is called *grey-box modelling*. In this approach, simplified, low-complexity physics-based models or analogies are used, sometimes in addition to sub-components that are modeled using black-box models.
In order to implement MPC, it is necessary to develop control-oriented models that accurately capture the relevant dynamics of the system. The process model used has a significant impact on the control performance, as the predictive capabilities of the model directly influence the quality of the control behavior. However, the models must also be simple enough to enable real-time optimization. Therefore, it is important to find a balance between model accuracy and simplicity, which is particularly crucial in renewable energy systems where complex multi-physics processes must be modeled.



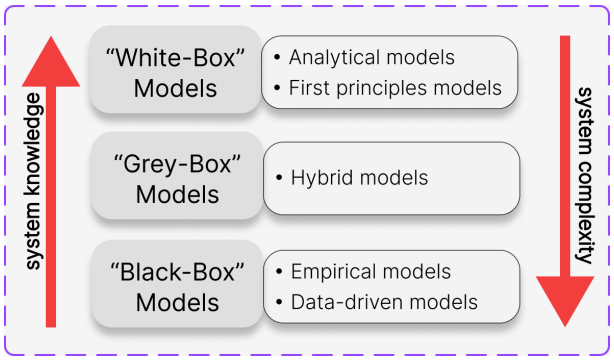
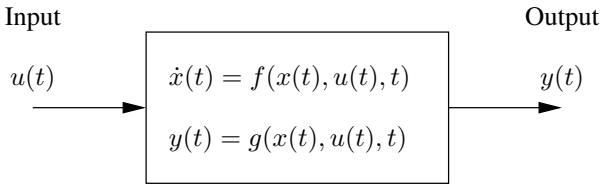


Figure 2.1: Modelling approaches

2.1.2 Dynamic System Modelling with ODE

Most commonly in MPC, state-space models derived from ODE are used. The use of the state-space representation offers several advantages compared to the other model representations. The states are used for the prediction of the output values in the process model. Multiple-input multiple-output (MIMO) systems can very naturally be handled and calculated. As the system states are explicitly given, the consideration of system states, e.g. for constraint handling is straightforward.
If the system dynamics is given by ordinary differential equations (ODE), the system can be represented as follows



- x is the n-dimensional internal state of the system. It can be regarded as 'memory' of the system.
- The dynamics are given by the equations of motion in form of an ODE

$$\dot{x}(t) = f(x(t), u(t), t) \tag{2.2}$$

  called *state equation* (or *system equation* or *dynamics*). It determines the time evolution of the state $x(t)$ by an ODE.

- The second equation

$$y(t) = g(x(t), u(t), t) \tag{2.3}$$

  is called *output equation* and maps the state (and input) to the output vector $y(t)$. Note that the output, state and input vectors can have a different dimensions. In general, the outputs y are the physically measurable quantities. In that context, it is important to remark that the states are not even unique, because different state-space realizations of the same input-output behavior exist.

Here, the different elements of the ODE are vector valued and $\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t}$ is the total derivative with regard to time.
The function f is a map from states, controls, and time to the rate of change of the states, i.e.

$$f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times [t_0, t_\mathrm{f}] \to \mathbb{R}^{n_x}.$$

In general, we have $t_0 = 0$. Sometimes, we consider parameters p (constant in time) within the dynamics so that $\dot{x}(t) = f(x(t), u(t), p(t), t)$. These parameters can either be constant in time or time-varying. Depending on the modelling decision, they can also beconsidered as decision variables that can be optimized.

However, it is of course possible to transform the parameters into control variables u, or if they are fixed, consider them as part of the model equation f.
We discuss one interesting system model in more detail.

Example (Climate modelling)**.** Typical weather models rely on a 3D-spatial discretization of the atmosphere and can only be simulated on supercomputers. Weather models are used to forecast day-to-day changes in weather, or more specifically, to predict what will happen at a specific place and point in time in the near future, typically no more seven days in advance. Model-based weather forecasts are generally less reliable beyond a week, because the atmosphere is an inherently chaotic system. Even minor variations in observed conditions, which are fed to the model regularly, can result in entirely distinct weather forecasts a week ahead, owing to the dynamic nature of the atmosphere.
Contrary to weather forecast models, the overall prediction accuracy of climate models seems much more reliable. This can be seen for example by comparing climate predictions made in the 1980s with today's measurements. Climate models are used to determine how the average conditions will change. Thus, they can be represented with an energy balance: how much power P_{in} does the earth receive from the sun in form of short wavelength solar irradiation, and how much power P_{out} does it emit back in form of long wavelength radiation? The latter is influenced by the greenhouse gas concentration, in particular by the concentration of CO_2 that we denote by $c_{\text{CO}_2}(t)$. One of the simplest possible models would have only two states, namely the average surface temperature $T(t)$ (in Kelvin) and the CO_2-concentration $c_{\text{CO}_2}(t)$ (in parts-per-million, or ppm). For the temperature evolution, one would need to know the heat capacity of the relevant part of the earth surface, e.g. of the upper 100 meters of water and land plus the atmosphere, that we describe by a total heat capacity C_{heat}. The resulting energy balance equation is as follows:

$$\dot{T} = \frac{P_{\text{in}} - P_{\text{out}}(c_{\text{CO}_2}, T)}{C_{\text{heat}}}.$$

Here, the total incoming solar power P_{in} would be a constant, while the outgoing power $P_{\text{out}}(c_{\text{CO}_2}, T)$ is a function of both c_{CO_2} and T and could e.g. be given by

$$P_{\text{out}}(c_{\text{CO}_2}, T) = (C_1 - C_2 c_{\text{CO}_2})T^4$$

with positive constants C_1 and C_2 (in suitable units) that describe how the CO_2-concentration influences the emissivity of the earth. The fourth power in the term T^4 comes from the Stefan-Boltzmann law. The evolution of c_{CO_2} would be in the simplest form be given by an accumulation of humanity's yearly CO_2 emissions $u(t)$ (in mol/year), that needs to be divided by the total amount of gas molecules in the atmosphere M (in mol) in order to make it a molar concentration (like ppm), such that we obtain

$$\dot{c}_{\text{CO}_2} = \frac{u}{M}.$$

Due to the slow time constants in the earth's climate system, today's climate predictions for the next 30 years are unfortunately not very strongly affected by humanity's upcoming emissions – or emission reductions – in the coming 30 years. Thus, a significant rise in the earth's average temperature in 2050 is nearly unavoidable. However, if humanity manages to reduce its emissions significantly today and in the next decade, we can positively influence the climate in 2100, with a high chance that the earth's temperature achieves its maximum between 2050 and 2100 and the climate of the year 2100 becomes similar to the climate in 2050.
More detailed, but still relatively simple climate models can e.g. be found on the following websites: `https://www.e-education.psu.edu/meteo469/node/137` and `https://en.wikipedia.org/wiki/Climate_model#cite_note-10`.

Properties of continuous-time systems with ODE

Let us sketch some relevant properties of continuous-time ODE systems. We are first interested in the question if the differential equation has a solution if the initial value $x(t_0)$ is fixed and also the controls $u(t)$ are fixed for all $t \in [t_0, t_{\text{f}}]$. In this context, the dependence of f on the fixed controls $u(t)$ is equivalent to a a further time-dependence of f, and we can redefine the ODE as $\dot{x} = \tilde{f}(x, t)$ with $\tilde{f}(x, t) := f(x, u(t), t)$.

Thus, let us first leave away the dependence of f on the controls, and just regard the time-dependent uncontrolled ODE:

$$\dot{x}(t) = f(x(t), t), \quad t \in [t_0, t_\mathrm{f}]. \tag{2.4}$$

An initial value problem (IVP) is given by (2.4) and the initial value constraint $x(t_0) = x_0$ with x_0 fixed. Existence of a solution to an IVP is guaranteed under continuity of f with respect to x and t according to a theorem from Peano. But existence alone is of limited interest as the solutions might be non-unique.

Example (Non-Unique ODE Solution)**.** The scalar ODE with $f(x) = \sqrt{|x(t)|}$ can stay for an undetermined duration in the point $x = 0$ before leaving it at an arbitrary time t_0. It then follows a trajectory $x(t) = (t - t_0)^2/4$ that can be easily shown to satisfy the ODE (2.4). We note that the ODE function f is continuous, and thus existence of the solution is guaranteed mathematically. However, at the origin, the derivative of f approaches infinity. It turns out that this is the reason which causes the non-uniqueness of the solution.

As we are only interested in systems with well-defined and deterministic solutions, we would like to formulate only ODE with unique solutions. Here helps the following theorem by Picard and Lindelöf.

Theorem 1 (Existence and Uniqueness of IVP)**.** *Regard the initial value problem* (2.4) *with $x(t_0) = x_0$, and assume that $f : \mathbb{R}^{n_x} \times [t_0, t_\mathrm{f}] \to \mathbb{R}^{n_x}$ is continuous with respect to x and t. Furthermore, assume that f is Lipschitz continuous with respect to x, i.e., that there exists a constant L such that for all $x, y \in \mathbb{R}^{n_x}$ and all $t \in [t_0, t_\mathrm{f}]$*

$$\|f(x, t) - f(y, t)\| \leq L\|x - y\|. \tag{2.5}$$

Then there exists a unique solution $x : [t_0, t_\mathrm{f}] \to \mathbb{R}^{n_x}$ of the IVP.

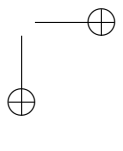
Here, $\|\| = \|\|_2$ refers the Euclidean norm, also known as the 2-norm, which is used a measure of the "length" or "magnitude" of a vector in $\mathbb{R}^n$. Lipschitz continuity is a strong form of continuity that quantifies the rate at which a function can change between two points. Lipschitz continuity of f with respect to x is not easy to check. It is much easier to verify if a function is differentiable. It is therefore a helpful fact that every function f that is differentiable with respect to x is also locally Lipschitz continuous, and one can prove the following corollary to the Theorem of Picard-Lindelöf.

Corollary 1 (Local Existence and Uniqueness)**.** *Regard the same initial value problem as in Theorem 1, but instead of global Lipschitz continuity, assume that f is continuously differentiable with respect to x for all $t \in [t_0, t_\mathrm{f}]$. Then there exists a possibly shortened, but non-empty interval $[t_0, t_\mathrm{f}']$ with $t_\mathrm{f}' \in (t_0, t_\mathrm{f}]$ on which the IVP has a unique solution.*

Note that for nonlinear continuous-time systems – in contrast to discrete time systems – it is very easily possible to obtain an "explosion", i.e., a solution that tends to infinity for finite times, even with innocently looking and smooth functions f.

Example (Explosion of an ODE)**.** Regard the scalar example $f(x) = x^2$ with $t_0 = 0$ and $x_0 = 1$, and let us regard the interval $[t_0, t_\mathrm{f}]$ with $t_\mathrm{f} = 10$. The IVP has the explicit solution $x(t) = 1/(1 - t)$, which is only defined on the half open interval $[0, 1)$, because it tends to infinity for $t \to 1$. Thus, we need to choose some $t_\mathrm{f}' < 1$ in order to have a unique and finite solution to the IVP on the shortened interval $[t_0, t_\mathrm{f}']$. The existence of this local solution is guaranteed by the above corollary. Note that the explosion in finite time is due to the fact that the function f is not globally Lipschitz continuous, so Theorem 1 is not applicable.

Note (Discontinuities with Respect to Time): It is important to note that the above theorem and corollary can be extended to the case that there are finitely many discontinuities of f with respect to t. In this case the ODE solution can only be defined on each of the continuous time intervals separately, while the derivative of x is not defined at the time points at which the discontinuities of f occur.But the transition from one interval to the next can be determined by continuity of the state trajectory, i.e. we require that the end state of one continuous initial value problem is the starting value of the next one. The fact that unique solutions still exist in the case of discontinuities is important because many state and parameter estimation problems are based on discontinuous control trajectories u(t). Fortunately, this does not cause difficulties for existence and uniqueness of the IVPs.



Discrete-time systems, Zero Order Hold and Solution Map

We call a system a *discrete-time system* whenever the time in which the system evolves only takes values on a predefined time grid, usually assumed to be integers. In this case, we typically use the index variable $k \in \mathbb{N}$, and write x_k or $x(k)$ for the state at time point k. The general time-variant discrete-time system is given by the following equation

$$x_{k+1} \quad = \quad f_k(x_k, u_k), \quad k = 0, 1, \ldots, N-1 \tag{2.6}$$

on a time horizon of length N, with N control input vectors $u_0, \ldots, u_{N-1} \in \mathbb{R}^{n_u}$ and $(N+1)$ state vectors $x_0, \ldots, x_N \in \mathbb{R}^{n_x}$.
If we know the initial state x_0 and the controls $u_0, \ldots, u_{N-1}$ we could recursively call the functions f_k in order to obtain all other states, $x_1, \ldots, x_N$. We call this a *forward simulation* of the system dynamics, see Figure 2.2.

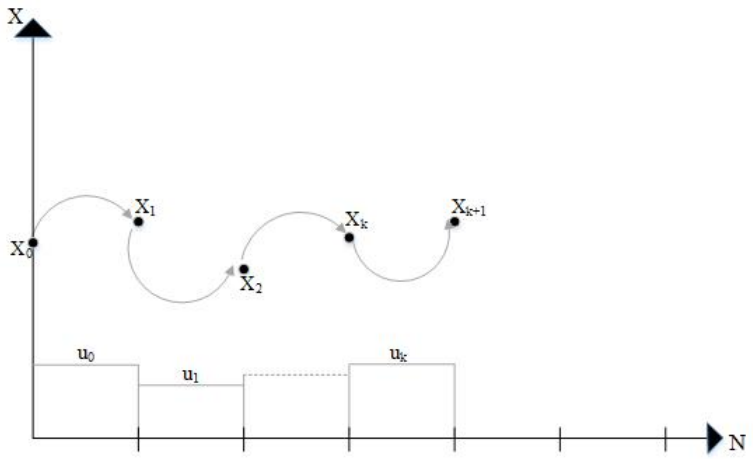


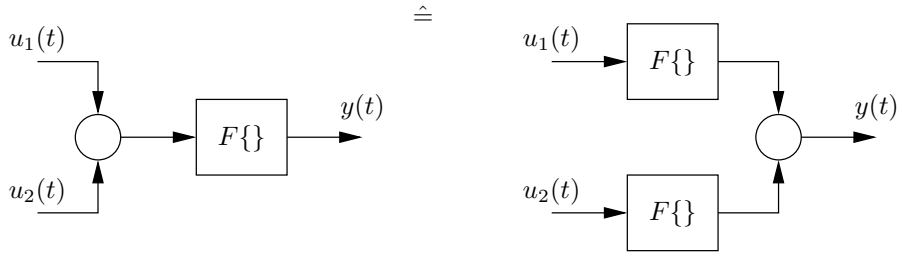
Figure 2.2: Illustration of the forward simulation of a discrete-time system .

In the age of digital control, the inputs u are often generated by a computer and implemented at the physical system as piecewise constant between two sampling instants. This approach is called *zero order hold*. The grid size is typically constant, say of fixed length $\Delta t > 0$, so that the sampling instants are given by $t_k = k \cdot \Delta t$. If our original model is a differentiable ODE model, but we have piecewise constant control inputs with fixed values $u(t) = u_k$ with $u_k \in \mathbb{R}^{n_u}$ on each interval $t \in [t_k, t_{k+1}]$, we might want to regard the transition from the state $x(t_k)$ to the state $x(t_{k+1})$ as a discrete time system. This is indeed possible, as the ODE solution exists and is unique on the interval $[t_k, t_{k+1}]$.

2.1.3 Linear Time-Invariant Systems

A special class of tremendous importance are linear time-invariant (LTI) systems. They are of principal interest in the field of automatic control. First, we give a very generic definition of LTI systems.
A dynamical system F is called *linear* if the following conditions are fulfilled:

1. Superposition principle
$$F\{u_1 + u_2\} = F\{u_1\} + F\{u_2\} \tag{2.7}$$
which can be illustrated as follows



2. Principle of amplification

$$F\{cu\} = cF\{u\} \tag{2.8}$$

depicted as follows

Both properties together imply that the overall response of an LTI system to a weighted sum of signals is the same as the weighted sum of the responses to each of the individual signals.

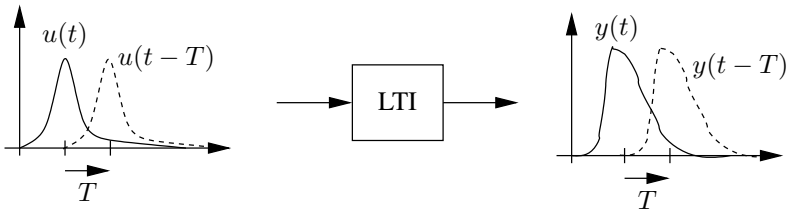
A dynamical system F is called *time-invariant*, if for any function $u(t)$

$$y \doteq F\{u\} \tag{2.9}$$

the equation

$$y_0 = F\{u_0\} \tag{2.10}$$

is valid for all T, where the function definitions $u_0 : t \mapsto u_0(t) \doteq u(t-T)$ and $y_0 : t \mapsto y_0(t) \doteq y(t-T)$ are introduced. This can be illustrated by

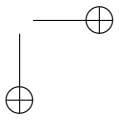


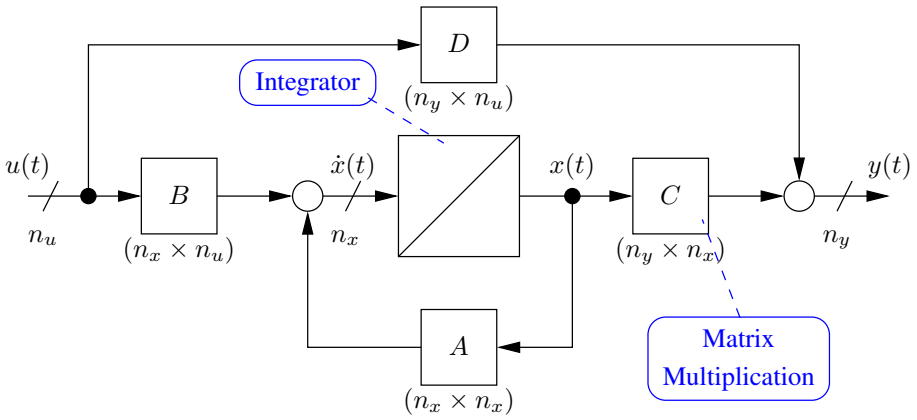
Time invariance means that whether we apply an input to the system now or T seconds from now, the output will be identical except for a time delay of T seconds.
Note: For time invariance, the initial (internal) states of the system have to be 0 (zero state).

If the dynamics are given by an ODE, the general LTI system in state-space is written as

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.11}$$
$$y(t) = Cx(t) + Du(t) \tag{2.12}$$

with fixed matrices $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$. This set of equations including dimensions of vectors and matrices can be drawn in the following block diagram





Note: Linearity can be easily shown for all dynamical systems of the following form:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t),\ t \in [t_0, t_\mathrm{f}],\ x(t_0) = x_0.$$

We can even allow the matrices to depend on time t, so we have a time-dependent system; the linearity is not destroyed by this.
An overview of linear and nonlinear and time-variant and time-invariant notation of the system dynamics is given in Table 2.1.

Table 2.1: Notation for linear and nonlinear and time-variant (time-dependent) and time-invariant systems

	time-invariant	time-dependent/time-variant
linear	$\dot{x} = Ax + Bu$	$\dot{x} = A(t)x + B(t)u$
nonlinear	$\dot{x} = f(x, u)$	$\dot{x} = f(t, x, u)$

The LTI system (2.11) and (2.12) is an example of a Multiple-Input Multiple-Output (MIMO) system. As a special case, we can consider a LTI system with only one input and one output, $n_u = 1$ and $n_y = 1$. This kind of system is called Single-Input Single-Output (SISO) and it is formulated as

$$\dot{x}(t) = Ax(t) + bu(t) \tag{2.13}$$
$$y(t) = c^\top x(t) + du(t) \tag{2.14}$$

where $A \in \mathbb{R}^{n_x \times n_x}$, $b \in \mathbb{R}^{n_x}$, $c \in \mathbb{R}^{n_x}$ and $d \in \mathbb{R}$.
Why is linearity so important? First of all, many physical systems can be modelled very accurately by linear differential equations. Examples are the main elements of electrical circuits - resistor, capacitor and inductor, thermal models or the heat conduction in solid bodies.
Even if we have a nonlinear system, if we care about what happens near an equilibrium point, it often suffices to approximate the nonlinear dynamics by their local linearization as we will see later in this chapter.
Secondly, linear systems have several mathematical properties that make their solution, analysis and control synthesis simple. Simulating the system and solving the resulting optimization problem within the MPC procedure is much harder for nonlinear systems. The superposition principle infers that the behavior of the resulting system subjected to a complex input can be described as a sum of responses to simpler inputs. In nonlinear systems, there is no such relation. For time-invariant systems this is the basis of the impulse response or the frequency response methods (see LTI system theory), which describe a general input function x(t) in terms of unit impulses or frequency components. Many important notions such as *controllability*, *stabilizability* and *observability*, and concepts such as the *impulse response* or *frequency response function* can be defined in terms of the matrices A, B, C and D alone.

Solution of the State Space ODE

In the following, the equation

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.15}$$

with $x(t_0) = x_0$ as initial condition will be solved.
The *homogeneous solution*

$$x(t) = e^{A(t-t_0)}x_0 \tag{2.16}$$

is the solution for

$$\dot{x}(t) = Ax(t) \tag{2.17}$$

which is the homogeneous part of (2.15). We used the *matrix exponential function*, which is defined by

$$e^{A(t-t_0)} \doteq \sum_{k=0}^{\infty} \frac{A(t-t_0)^{\nu}}{\nu!} \tag{2.18}$$

The time derivative reads

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} e^{A(t-t_0)} &= \frac{\mathrm{d}}{\mathrm{d}t} \sum_{k=0}^{\infty} \frac{A^k(t-t_0)^k}{k!} = \sum_{k=0}^{\infty} \frac{A^k k(t-t_0)^{k-1}}{k!} \\ &= A \sum_{k=1}^{\infty} \frac{A^{k-1}(t-t_0)^{k-1}}{(k-1)!} = Ae^{A(t-t_0)} \end{aligned} \tag{2.19}$$

Computing the time derivative of the solution (2.16) yields

$$\dot{x}(t) = A \underbrace{e^{A(t-t_0)}x_0}_{x(t)} = Ax(t) \tag{2.20}$$

and proves that the solution fulfills (2.17).

The *general solution* reads

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)Bu(\tau)\mathrm{d}\tau \tag{2.21}$$

with

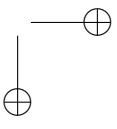
$$\Phi(t, t_0) \doteq e^{A(t-t_0)} \tag{2.22}$$

Note that the first term is the homogeneous solution due to the initial condition x_0 and the second term is a convolution integral of input $u(t)$. In order to show that (2.21) is a solution, we compute $\dot{x}(t)$ by deriving (2.21)

$$\begin{aligned} \dot{x}(t) &= A\Phi(t, t_0)x_0 + \underbrace{\Phi(t, t)}_{=I} Bu(t) + \int_{t_0}^{t} \underbrace{\frac{\mathrm{d}}{\mathrm{d}t}\Phi(t, \tau)}_{A\Phi(t,\tau)} Bu(\tau)\,\mathrm{d}\tau \\ &= \underbrace{A\left(\Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)Bu(\tau)\,\mathrm{d}\tau\right)}_{= x(t), \text{ compare (2.21)}} + Bu(t) = Ax(t) + Bu(t) \end{aligned} \tag{2.23}$$

□

The solution (2.21) can be computed if x_0 and $u(t) = u_{\text{const}}$ are fixed.
It is interesting to note that this map is well defined for all times $t \in \mathbb{R}$, as linear systems cannot explode. The function $f(x, u) = Ax + Bu$ is Lipschitz continuous with respect to x with Lipschitz constant $L = \|A\|$, so that the global solution to any initial value problem with a piecewise continuous control input can be guaranteed.



Example (Example system with one state)**.** We want to solve the following scalar ODE:

$$\dot{x}(t) = ax(t) + bu(t)$$

where $a \in \mathbb{R}$. We set the initial value as $x_0 = 0$ and use the constant control input $u(t) = 1$ for all time $t \in [0, \infty)$. The solution is given by

$$\begin{aligned} x(t) &= e^{at}x_0 + \int_0^t e^{a(t-\tau)}bu(\tau)\mathrm{d}\tau \\ &= 0 + \int_0^t e^{a(t-\tau)}b\mathrm{d}\tau \\ &= e^{at}\int_0^t e^{-a\tau}b\mathrm{d}\tau \\ &= e^{at}\left[\frac{b}{(-a)}e^{-a\tau}\right]_0^t \\ &= e^{at}\frac{b}{(-a)}(e^{-at} - 1) \\ &= \frac{b}{(-a)}(1 - e^{at}). \end{aligned}$$

We will encounter systems of this type frequently. Note that the term e^{at} approaches zero with increasing time t for $a < 0$, indicating that the system is *stable*, while it grows unbounded for $a > 0$, indicating that the system is *unstable*. The behavior of this system is illustrated in Figure 2.3.

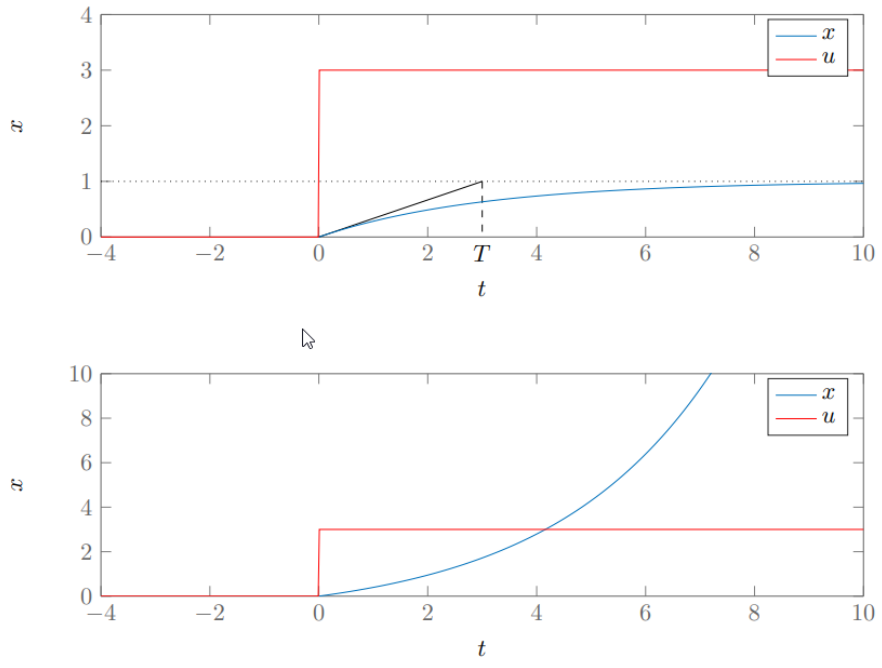


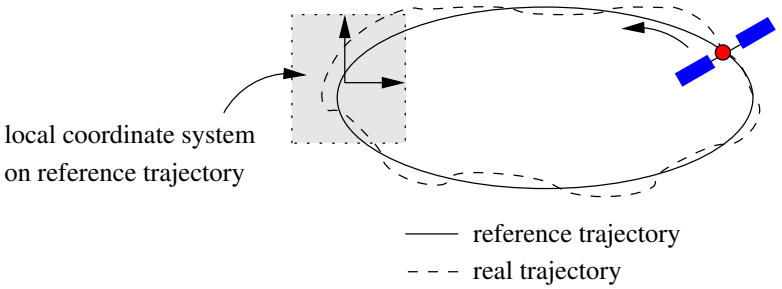
Figure 2.3: Simulation of the step response of the system $\dot{x} = ax + bu$ with initial condition $x(0) = 0$ and control $u(t) = 3$. Top: for $a = -\frac{1}{T}$ (and $b = \frac{1}{T^2}$ with $T = 3$), the system is stable. Bottom: for $a = \frac{1}{T}$ (and $b = \frac{1}{T^2}$), the system is unstable.

Linearizations along Trajectories

Many real world problems lead to nonlinear ODE. In this case, linearization is a powerful tool, which can be applied in many cases, because most nonlinear systems near an equilibrium position can be described very

well by linear systems. The idea is to consider the behavior of a system around a reference or steady-state point by linearization of the ODE.

Example. As example we consider trajectory control of a satellite on an orbit



In absence of disturbances and with zero steering input, the satellite would fly on the orbit, denoted as solid trajectory. By introduction of a local coordinate system, we only consider deviations from this reference trajectory. The state value $x = 0$ would then describe a satellite flying on the reference trajectory. As deviations are expected to be small compared to the overall trajectory, linearization of the spherical coordinate system is an adequate modelling approach.

Other examples can be found in engineering for mechanical systems, which can be regarded as nearly linear due to the small deviations, e.g., the oscillations and resonances occurring in every mechanical construction, be it in vibrating bridges or buildings, in vibrating airplanes, cars, or transformers, or in oscillating piezoelectric elements. In almost all cases, the displacement is so small that a linear system analysis can describe the system accurately.
A nonlinear system $\dot{x} = f(x, u)$ can be linearized using a first order Taylor expansion if one assumes that the system state $x(t)$ and the control $u(t)$ differ only little from a stationary state. Let us denote this rest position, which is called *steady state*, with x_{ss} and u_{ss}. x_{ss} and u_{ss} being a stationary state means that $\dot{x} = 0$ must hold, i.e. $0 = f(x_{\text{ss}}, u_{\text{ss}})$.
We define the differences as

$$\delta x(t) = x(t) - x_{\text{ss}} \quad \text{and} \quad \delta u(t) = u(t) - u_{\text{ss}}.$$

Because we consider the system in the neighborhood of the steady state, $\delta x(t)$ and $\delta u(t)$ can be assumed to be small. The linearization of $f(x, u)$ is given by the first-order Taylor expansion

$$f(x,u) \approx \underbrace{f(x_{\text{ss}}, u_{\text{ss}})}_{=0} + \frac{\partial f}{\partial x}(x_{\text{ss}}, u_{\text{ss}})\,\delta x + \frac{\partial f}{\partial u}(x_{\text{ss}}, u_{\text{ss}})\,\delta u.$$

Furthermore, since

$$\frac{\mathrm{d}(x_{\text{ss}} + \delta x(t))}{\mathrm{d}t} = \underbrace{\frac{\mathrm{d}x_{\text{ss}}}{\mathrm{d}t}}_{=0} + \frac{\mathrm{d}(\delta x(t))}{\mathrm{d}t} = \delta\dot{x}(t)$$

the linearization of $\dot{x}(t) = f(x(t), u(t))$ yields the linear time invariant differential equation

$$\delta\dot{x}(t) = \underbrace{\frac{\partial f}{\partial x}(x_{\text{ss}}, u_{\text{ss}})}_{\doteq A}\,\delta x(t) + \underbrace{\frac{\partial f}{\partial u}(x_{\text{ss}}, u_{\text{ss}})}_{\doteq B}\,\delta u(t). \tag{2.24}$$

This equation has exactly the form of the LTI ODE (2.11).

Example. The pendulum is a nonlinear system whose equilibrium is at the states $x_1 = x_2 = 0$ (angle, angular velocity) and control $u = 0$ (torque). Thee system equation are given by

$$f(x,u) = \begin{bmatrix} x_2 \\ -\frac{mgL}{I}\sin x_1 + \frac{1}{I}u \end{bmatrix}$$

By linearizing around the point $x_{\text{ss}} = [0, 0]^\top$ and $u_{\text{ss}} = 0$, we obtain the linear system describing the linear behavior in the neighborhood of this equilibrium point:

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{mgL}{I} & 0 \end{bmatrix} \quad \text{und} \quad B = \begin{bmatrix} 0 \\ \frac{1}{I} \end{bmatrix}.$$

To summarize, the following procedure can be applied for controller design around a stationary state

1. Set up general ODE.
2. Linearize system around equilibrium point.
3. Design controller.
4. Validate control design with general (nonlinear) ODE in numerical simulations.

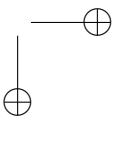
This procedure can be applied to any nonlinear system near an equilibrium position. In control engineering, we very often have to deal with small deviations, thus a linearized system approach is often sufficient. The linearization is essentially an approximation of the nonlinear dynamics around the desired operating point. This operating point does not need to be an equilibrium point. In fact, the goal of the control is usually to keep the system exactly or as close as possible in a stationary state, also called set-point or reference, so that the better the control performance, the better the assumption of linear system models is fulfilled. In this regard, we create systems with linear input/output response through the use of feedback.
Linear time-varying systems occur as a linearization of nonlinear systems along highly varying trajectories. Examples are the control of robot arm movements or in periodic processes such as the revolutions of a diesel engine. The following linearization procedure can be used to approximately take into account the transient nonlinear system behavior:

1. Obtain a new measurement or estimation for $x(k)$.
2. Build one linear model around the measured or estimated value $x(k)$.
3. Use this model for an LTI prediction over the entire prediction horizon.
4. Repeat this procedure in every sampling step

In each step, the operating point changes. Therefore, the equation has the form of a linear time-variant ODE with a non-zero affine term $f(x_{\text{op}}, u_{\text{op}})$:

$$\delta\dot{x}(t) = \underbrace{\frac{\partial f}{\partial x}(x_{\text{op}}(t), u_{\text{op}}(t))}_{\doteq A(t)} \delta x(t) + \underbrace{\frac{\partial f}{\partial u}(x_{\text{op}}(t), u_{\text{op}}(t))}_{\doteq B(t)} \delta u(t) + \underbrace{f(x_{\text{op}}(t), u_{\text{op}}(t))}_{\doteq c(t)}. \tag{2.25}$$

The main difference between LTI and LTV systems lies in the fact that in the LTV case, the matrices that capture the system dynamics change at each time step. Consequently, the matrices A and B require recalculation in every time step, whereas in the LTI scenario, they remain constant. It's worth noting that when computing the MPC control law, this difference leads to a nonlinear control law in the unconstrained LTV case, which is in contrast to the linear control law for LTI systems.



2.2 Modelling of renewable energy systems

Every model is always only a simplification of reality and there are always phenomena that are not covered by the model. Therefore, when building a model, it's essential to ensure that the model accurately represents the behavior or phenomenon that you want to analyze or predict. At this point, our examination of renewable energy systems such as solar and building energy systems is focused on a higher-level understanding, emphasizing the main modeling concepts without delving into excessive detail of the underlying physical properties.

2.2.1 Modelling the thermal behavior of buildings

We want to create a mathematical model of a building to simulate the thermal behavior of the building depending on the heat losses to the colder environment and heat gains from the heating operation. As a simplification, we consider only one room. The building's thermal behavior can be modelled using energy balance equations. The energy change in the building is determined by the sum of the incoming energy flows and, with negative signs, outgoing energy flows from the building. As shown in Figure 2.4, the incoming energy flow is given by the heat flow from the heating operation and the outgoing heat flow is due to the heat losses through the walls and windows, i.e., through the entire building envelope. Solar heat gains (due to sunlight) and internal heat gains (due to occupant behavior and electrical appliances) also heat up the building and are summarized by $\dot{Q}_{\mathrm{gains}}$. The model equation can be formulated as follows:

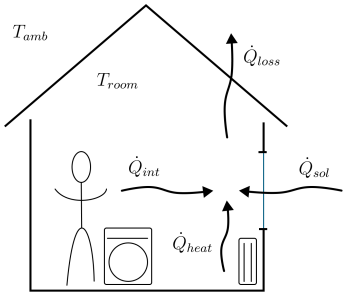


Figure 2.4: A simple building model with incoming and outgoing heat flows

$$\frac{\mathrm{d}T_{\mathrm{room}}}{\mathrm{d}t} \cdot C_{\mathrm{bldg}} = \dot{Q}_{\mathrm{heat}} - \dot{Q}_{\mathrm{loss}} + \dot{Q}_{\mathrm{gain}} \tag{2.26}$$

For now, we assume that there are no solar and internal heat gains. The energy change is the temporal change in room temperature T_{room} multiplied by the thermal capacity of the building C_{bldg}. The thermal capacity $C = C_{\mathrm{bldg}}$ (in $\mathrm{J/K} = \mathrm{Ws/K}$) indicates how much heat the building envelope can store per temperature change of 1K. The heat losses are proportional to the temperature difference between the indoor and outdoor temperatures T_{amb}:

$$\dot{Q}_{\mathrm{loss}} = U \cdot (T_{\mathrm{room}} - T_{\mathrm{amb}}) \tag{2.27}$$

U-values, also known as transmission heat transfer or transmission heat loss coefficient, and sometimes denotes by H, indicate the insulation effectiveness of the components that make up a building's structure. These values determine the degree to which heat can pass through the building's exterior and interior. The lower the U-value, the slower the transmission of heat, resulting in improved insulation performance. A building with lower U-values requires less energy to maintain comfortable indoor conditions. Given in $\mathrm{W/K}$, it determines the heat flow that passes through the building envelope at a temperature difference of 1 K. The U-value of the entire building envelope is determined by the specific U-values of all components multiplied by their respective surface areas, e.g., windows and walls (composed of various component layers). Specific U-values (based on the component surface area) are given in $\mathrm{W/(m^2K)}$ and usually range from 0.7 (triple thermal insulation glazing) to 5 (single glazing) for windows. The smaller the value, the

electrical circuit	thermal network
$C[F]$: el. capacitance	$\mathrm{C}[J/K]$: therm. capacitance
$\Delta U[V]$: voltage difference	$\Delta T[K]$: Temperature difference
$R[\Omega]$: el. resistance	$R[K/W]$: therm. Resistance
$I[A]$: el. current	$\dot{Q}[W = J/s]$: heat flux
el. current source	heat flow from outside

Table 2.2: Analogy of thermal moel to electrical RC-circuit

better the insulation. Note that the term 'U-value' may refer to both the specific U-value and the overall U-value, depending on the context. In this instance, we are considering the first variant of the U-value.
The U-value can also be replaced by the thermal resistance $R = 1/U$ (in K/W), which is simply the reciprocal of the U-value. The formulated model equation has the form:

$$\frac{\mathrm{d}T_{\mathrm{room}}}{\mathrm{d}t} = -\frac{U}{C} \cdot T_{\mathrm{room}} + \frac{U}{C} \cdot T_{\mathrm{amb}} + \frac{1}{C} \cdot \dot{Q}_{\mathrm{heat}} = -\frac{1}{RC} \cdot T_{\mathrm{room}} + \frac{1}{RC} \cdot T_{\mathrm{amb}} + \frac{1}{C} \cdot \dot{Q}_{\mathrm{heat}} \quad (2.28)$$

With this new form we can make an analogy to an electrical RC circuit consisting of a capacitor and an electrical resistor. The electrical capacitance of the capacitor is replaced by the thermal capacitance and the electrical resistance is replaced by the thermal resistance. The temperature differences correspond to the voltage differences and the thermal currents correspond to the electric currents.

The simple thermal model can be represented by a first-order RC circuit (abbreviated 1R1C) as shown in Figure 2.5.

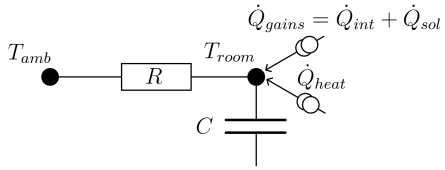


Figure 2.5: First-order thermal RC network (1R1C) of the building model

This type of modeling is common in building energy engineering and is especially useful when we want to refine the model so that, for example, the heat transfer through the different wall layers is also considered. Such a refined third-order RC model with additional nodes for the internal and external surface temperatures (T_{w} and $T_{\mathrm{w},a}$) of the building envelope is given in Figure 2.6.

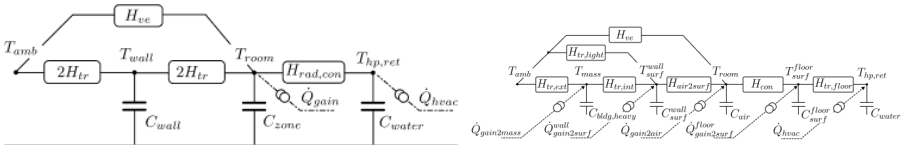


Figure 2.6: More complex building models as thermal RC networks (4R3C and 7R5C)

LTI system behavior We will now move on to analyze the dynamic behavior of the building model.
1.) First, we want to find out how the room temperature T_{room} changes with time, by solving the differential equation (2.28). We make the simplifying assumptions that the initial room temperature is equal to 20°C, the ambient temperature is constant 0°C, and the heating is off. The assumptions are thus

- $\dot{Q}_{\mathrm{heat}} = 0\mathrm{W}$: no heat flow from the heat generator
- $T_{\mathrm{room},0} = 20°\mathrm{C}$: initial room temperature
- $T_{\mathrm{amb}} = 0°\mathrm{C}$: constant ambient temperature

With the simplification, this now has the form

$$\frac{dT_{\text{room}}}{dt} = -\frac{1}{RC} \cdot T_{\text{room}} + \frac{1}{RC} \cdot T_{\text{amb}} \tag{2.29}$$

and corresponds to a linear time-invariant differential equation with state variables $x(t) = T_{\text{room}}(t)$ and external given influences $u(t) = T_{\text{amb}}(t)$.

Note: If the external influence variable $T_{\text{amb}}(t)$ is not defined as a kind of control variable $u(t)$, then this equation can also be conceived as a time-varying differential equation in which the model equation changes with time due to a varying $T_{\text{amb}}(t)$.
To be able to apply the solution formula for time-invariant, linear differential equations we stay with the first form. The unique solution for a LTI differential equation

$$\dot{x}(t) = ax(t) + bu(t) \tag{2.30}$$

with $t_0 = 0$ and initial value $x(0) = x_0$ is

$$\begin{aligned} x(t) &= e^{at}x_0 + \int_0^t \underbrace{e^{a(t-\tau)}}_{e^{at} - e^{a\tau}} \cdot b \underbrace{u(\tau)}_{u_{\text{const}}} \, \mathrm{d}\tau \\ &= e^{at}x_0 + b \cdot e^{at} \cdot u_{\text{const}} \underbrace{\int_0^t e^{-a\tau} \mathrm{d}\tau}_{-\frac{1}{a}e^{-a\tau}\Big|_0^t = -\frac{1}{a}e^{-at} + \frac{1}{a} \cdot 1} \\ &= e^{at}x_0 + \frac{b}{a} \cdot u_{\text{const}}(e^{at} - 1). \end{aligned} \tag{2.31}$$

Here we assumed that $u(t) = u_{\text{const}}$ is constant over $[0, t_{\text{f}}]$. Plugged into our original equation (with $a = -\frac{1}{RC}$ and $b = \frac{1}{RC}$) we get for room temperature the solution trajectory

$$\begin{aligned} T_{\text{room}} &= e^{-\frac{1}{RC}t} \cdot T_{\text{room},0} + \frac{1}{RC} \cdot e^{-\frac{1}{RC}t} \cdot T_{\text{amb,const}}[RC \cdot e^{\frac{1}{RC}t} - RC] \\ &= e^{-\frac{1}{RC}t} \cdot T_{\text{room},0} + T_{\text{amb,const}}[1 - e^{-\frac{1}{RC}t}]. \end{aligned} \tag{2.32}$$

With $T_{\text{U}_{\text{const}}} = 0°\text{C}$ and initial value $T_{\text{room},0} = 20°\text{C}$ we obtain

$$T_{\text{room}}(t) = e^{-\frac{1}{RC}t} \cdot 20°\text{C}. \tag{2.33}$$

After a time of $t = RC = T$, the room temperature has cooled from $20°\text{C}$ to about $7°\text{C}$:

$$T_{\text{room}}(T) = e^{-1} \cdot 20°\text{C} \approx \frac{1}{2.7} \cdot 20°\text{C} \approx 7°\text{C}.$$

On the other hand, if the ambient temperature is $T_U = 5°\text{C}$ instead of $0°\text{C}$, the room temperature will turn

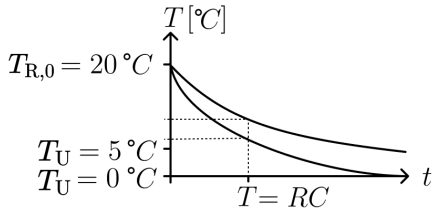


Figure 2.7: Time response of building cooling to heating failure for $T_{\text{amb}} = 0°\text{C}$ and $T_{\text{amb}} = 5°\text{C}$

out to be higher after a time of T:

$$T_{\text{room}}(T) = e^{-1} \cdot 20°\text{C} + 5°\text{C} \cdot (1 - e^{-1}) \approx 10°\text{C}.$$

The behavior is equal a 1st order delay element, also called PT1 element, or low-pass filter in electrical circuit terms. Due to the thermal inertia of the building, the room temperature reacts with a delayed temperature drop to a heating failure. The term $T = RC$ is called the time constant of the system. The time constant can be used to classify the thermal behavior of our simple building model, i.e., each building has a specific time constant. For example, an energy efficient new building may have a long time constant of $T = 3$ days, whereas an older existing building with thin walls and windows may only have a time constant of $T = 0.5$ days.
2.) The next question we are interested in is how much heat must be added to the building to maintain a desired set point temperature. For example, if we want to maintain a setpoint temperature of $T_{\text{setpoint}} = 20°\text{C}$, what is the heat input $\dot{Q}_{\text{heat}}$? To answer this question, we look for a steady state equilibrium characterized by $\frac{dT_{\text{room}}}{\text{d}t} = 0$. Plugged into the equation (2.26), we get

$$\begin{aligned} \frac{dT_{\text{room}}}{dt} \cdot C_{\text{bldg}} = 0 &= \dot{Q}_{\text{heat}} - \dot{Q}_{\text{loss}} \\ \Leftrightarrow \dot{Q}_{\text{heat}} = \dot{Q}_{\text{loss}} &= \frac{1}{R} \cdot (T_{\text{room}} - T_{\text{amb}}). \end{aligned} \tag{2.34}$$

This can be used to calculate how much heating energy must be supplied to the building to maintain a constant temperature, i.e., to balance the losses. Let us consider a simple example of a poorly insulated building with parameters $R = 2.5\frac{\text{K}}{\text{kW}}$ and $C = 5\frac{\text{kWh}}{\text{K}}$. The heating power required to maintain the room temperature at $20°\text{C}$ for a given ambient temperature of $0°\text{C}$ is equivalent to

$$\dot{Q}_{\text{heating,stat}} = \frac{1}{2.5\frac{\text{K}}{\text{kW}}}(20°\text{C} - 0°\text{C}) = 8\text{kW}.$$

3.) The same methods used in the previous section can be used to analyze the effect of a constant heating power on the room temperature. For the building just considered, a constant heating power of 8kW is needed to achieve (or nearly achieve) and maintain a costant room temperature of $20°\text{C}$ at an ambient temperature of $0°\text{C}$. The time constant for the building is $T = RC = 12.5\text{h}$. If the initial temperature from the building is now equal to $0°\text{C}$, it will have reached a room temperature of $2.5K/\text{kW} \cdot 8\text{kW} \cdot (1 - e^{-1}) \approx 14°\text{C}$ after $T = RC = 12.5\text{h}$ at a constant heating power of $\dot{Q}_{\text{heat,stat}} = 8\text{kW}$. The calculation is analogous to the previous calculation.

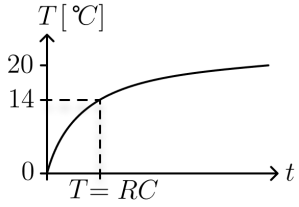


Figure 2.8: Step response for heating process example building

Heat distribution system We distinguish between air- and water-based heat distribution systems. Air-based heating systems such as air-to-air heat pumps (AA-HP), provide space heating by delivering heated air using air handling units (AHU) within specific rooms, or around a home through a series of ducts. Water-based heating systems auch as air-to-water heat pumps are paired with indoor central heating systems using water as a heat-transfer medium and are usually coupled with equipment such as radiators and underfloor heating. Water-based cooling may also be coupled with fan coils.
For water-based heating, we need to additionally model the hydronic system because the heat ist transfered from the water via the radiant surface to the room, resulting in a lower return temperature to the heater.
In a water-based system, we also consider the return temperature through the radiator back to the heat pump as a state. The resulting state vector is

$$x(t) = [T_{\text{room}}, T_{\text{hp,ret}}]$$

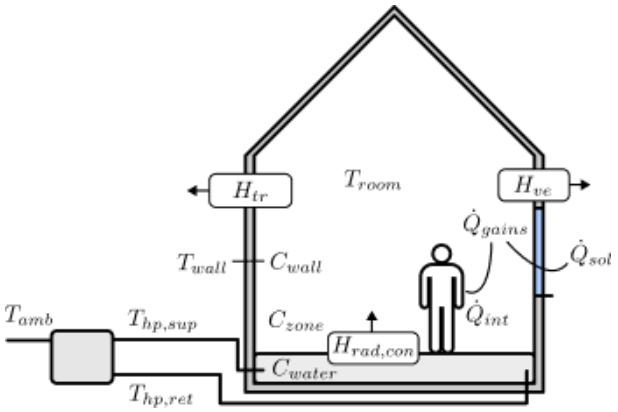


Figure 2.9: Building model illustration (states are zone temperature, wall temperature and heating water temperature)

The following building attributes have to be set, see Table 2.3 for a definition: $H_{\mathrm{ve,tr}}, H_{\mathrm{rad,con}}, C_{\mathrm{bldg}}, C_{\mathrm{water}}$ The control variable is usually the supply temperature of the heat pump $T_{\mathrm{hp,sup}}$, but can also be heat pump power or compressor frequency or something else. The resulting ODE system is as follows:

$$\dot{T}_{\mathrm{room}} = 1/C_{\mathrm{bldg}} \cdot (\dot{Q}_{\mathrm{gain}} + H_{\mathrm{rad,con}} \cdot (T_{\mathrm{hp,ret}} - T_{\mathrm{room}}) - H_{\mathrm{ve,tr}} \cdot (T_{\mathrm{room}} - T_{\mathrm{amb}}))$$
$$\dot{T}_{\mathrm{hp,ret}} = 1/C_{\mathrm{water}} \cdot (\dot{m}_{\mathrm{hp}} \cdot c_{\mathrm{p,water}} \cdot (T_{\mathrm{hp,sup}} - T_{\mathrm{hp,ret}}) - H_{\mathrm{rad,con}} \cdot (T_{\mathrm{hp,ret}} - T_{\mathrm{room}}))$$

Heat transfer coefficients for ventilation and transmission [W/K]	$H_{\mathrm{ve,tr}} = H_{\mathrm{ve}} + H_{\mathrm{tr}}$
Heat transfer coefficients for ventilation (indoors $\rightarrow$ ambient) [W/K]	H_{ve}
Heat transfer coefficients for transmission (indoors $\rightarrow$ ambient) [W/K]	H_{tr}
Heat transfer coefficients for radiation and convection (hvac $\rightarrow$ indoors) [W/K]	$H_{\mathrm{rad,con}}$
Heat capacity of entire building [J/K]	$C_{\mathrm{bldg}} = C_{\mathrm{wall}} + C_{\mathrm{zone}}$
Heat capacity of wall [J/K]	C_{wall}
Heat capacity of thermal zone [J/K]	$C_{\mathrm{zone}} = C_{\mathrm{air}} + C_{\mathrm{int}}$
Heat capacity of indoor air [J/K]	C_{air}
Heat capacity of the interior [J/K]	C_{int}
Heat capacity of the water in the hvac system [J/K]	C_{water}

Table 2.3: Parameters for the building model (hvac refers to heating, cooling, air-conditioning and ventilation)

Control task Different control tasks are:

- Reference tracking: Keep room temperature close to the set-point temperature despite disturbances such as heat loss to the ambient and heat gains due to people behavior and solar irradiations. Predictions about disturbances can used.
- Additionally minimize an "economic" cost function (e.g., energy cost)
- Demand responsive heating and cooling operation (e.g., load shifting, peak shaving (load shedding)), see Figure 2.10

2.2.2 Modelling of heat pumps

This section is based on the excellent dissertation of Wimmer about MPC for heat pumps (in German) [13]. A heat pump uses mechanical energy to transport heat energy from a lower to a higher temperature level. This process can be idealized by a Carnot process (see left figure in Figure 2.11). From a fluid - the so-called working fluid - the heat flow $\dot{Q}_1$ is absorbed at the temperature T_1. An isentropic temperature change

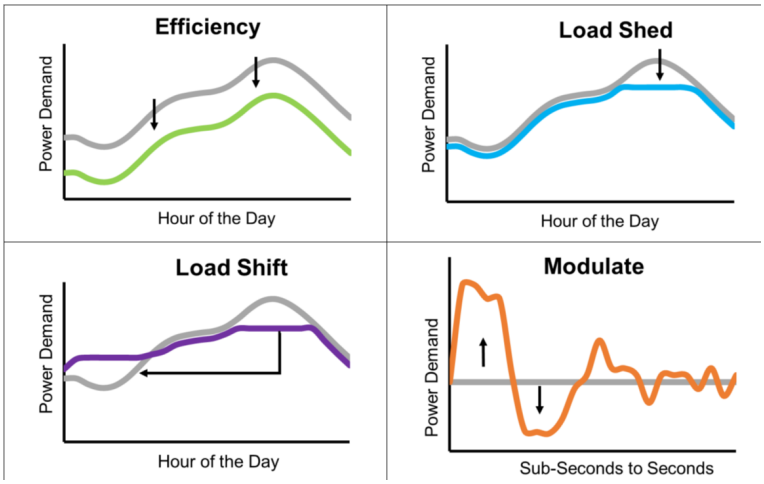


Figure 2.10: Building flexibility load curves. The grey curve represents an example baseline residential building load and the colored curves show the resulting building load. Source: "Grid-interactive Efficient Buildings: Overview of Research Challenges and Gaps," US Department of Energy, December 2019, `https://www1.eere.energy.gov/buildings/pdfs/75470.pdf`

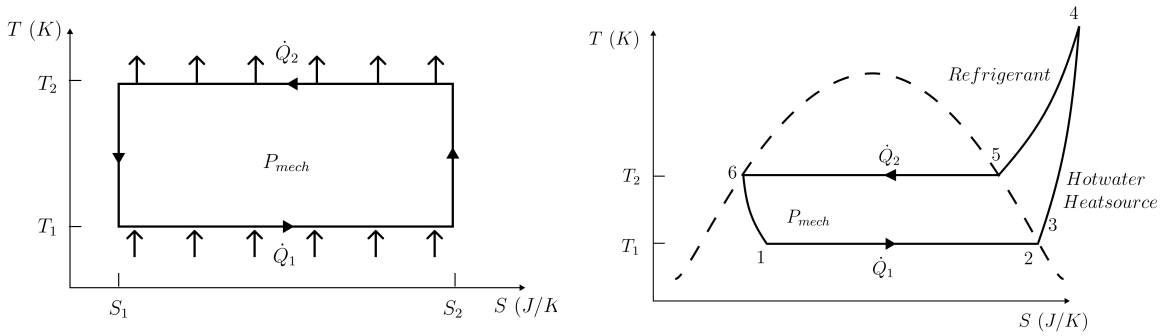


Figure 2.11: Ideal carnot process (left) and real thermodynamic cycle of heat pump (right) [13]

heats the fluid to the temperature T_2, at which the heat flow $\dot{Q}_2$ is released. Via a second isentropic change of state, cooling down to the original temperature T_1 takes place. The mechanical power for this process is given by the circular integral from the circular integral

$$P_{\text{mech}} = -\oint TdS. \tag{2.35}$$

A schematic representation of a heat pump is shown in Figure 2.12. The ideal carnot process and the real heat pump cycle process are shown in Figure 2.11.
In the evaporator, at evaporating temperature T_1, heat flow $\dot{Q}_1$ flows from the heat source to the working fluid (1 $\rightarrow$ 2). As a result, the working fluid is evaporated. To avoid damage to the very high compression compressor, the working fluid must be superheated so that it no longer contains liquid components (2 $\rightarrow$ 3). Compression in the compressor further heats the gaseous working fluid (3 $\rightarrow$ 4). Due to friction losses and engine waste heat, the process is no longer isentropic. In the condenser, the gas is cooled to condensation temperature T1 (4 $\rightarrow$ 5) and condenses releasing heat flow $\dot{Q}_2$ (5 $\rightarrow$ 6). In the expansion valve, the fluid is expanded, the fluid is brought back to the initial temperature T_1 (6 $\rightarrow$ 1). For a heat pump to work efficiently, it should use as little mechanical power as possible for the delivered thermal power $\dot{Q}_2$. The ratio of $\dot{Q}_2$ to P_{mech} is called COP (coefficient of performance). For the reversible Carnot process a

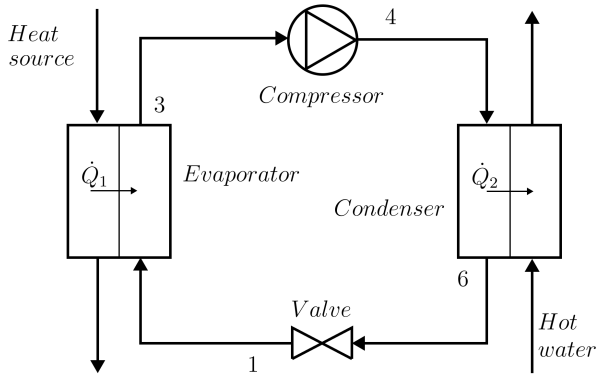


Figure 2.12: Schematic demonstration of a heat pump [13].

theoretical COP of

$$\mathrm{COP}_{\mathrm{th}} = \frac{\dot{Q}_2}{P_{\mathrm{mech}}} = \frac{T_2}{T_2 - T_1} \tag{2.36}$$

is possible. However, real heat pumps do not operate loss-free. The achievable COP of a heat pump is therefore smaller than the theoretical value by the exergetic efficiency η_{WP}: $\mathrm{COP} = \eta_{\mathrm{WP}}\mathrm{COP}_{\mathrm{th}}$. Modern air-to-water heat pumps achieve an efficiency of about 0.45. Often the dependence of the COP on the source and sink temperature (T_1, resp. T_2 in Equation (2.36)) is approximated by a second-ordner polynomial.

Example. The COP of a heat pump depends on the source and sink temperature (T_1, resp. T_2 in Equation (2.36)). Assuming a sink temperature of 308 K (35 °C) for T_2 and an outside air temperature of 273 K (0 °C) for T_1, the maximum theoretical COP would be $\mathrm{COP}_{\mathrm{th}} = 308/35 = 8.8$ and a realistic COP would be $\mathrm{COP} = 0.45 \cdot 8.8 \approx 4$. In a building with poor insulation, $T_2 = 328K$ (55 °C) might be needed, resulting in a lower $\mathrm{COP}_{\mathrm{th}} \approx 6$ and $\mathrm{COP} = 2.7$.

2.2.3 Solar thermal collector model

Solar thermal collectors (STC) absorb the radiation of sunlight, which leads to an increase in the internal energy of the heat transfer fluid. Most commonly, they are used in buildings to generate hot water for space heating or domestic hot water. In countries with a lot of sunlight, there exist also large solar power plants generating electricity from the produced heat.
See Figure 2.13 for an illustration of a flat-plate collector, the most simple STC type. For ontrol purposes, we will not consider a model of so much detail. The most important operating principle is that when sunlight strikes the absorber plate, it heats up and transfers this thermal energy to a fluid (usually water or a similar heat transfer fluid) that flows through the collector's tubing. The heated fluid can then be used directly for domestic hot water or space heating. For use within buildings, a hot water tank is often used to store the produced hot water for use as needed.

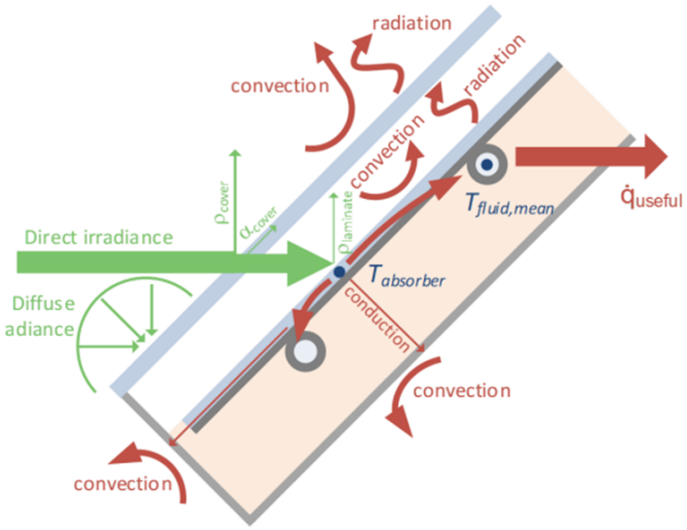


Figure 2.13: Typical flat plate collector

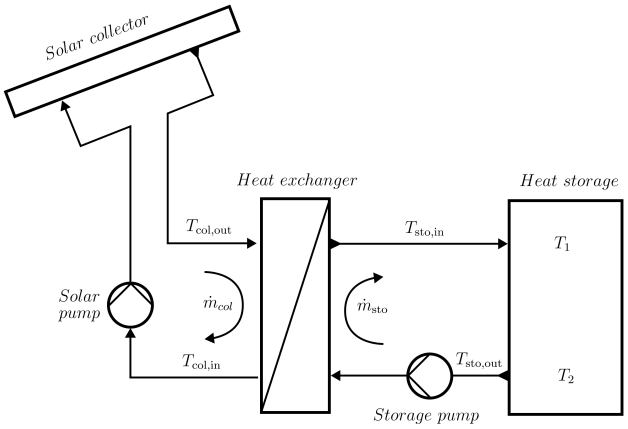


Figure 2.14: Hydraulic scheme of solar thermal collector consisting of collector-heat exchanger cycle (left) and heat exchanger-storage cycle (right). An internal rule-based controller regulates the pumps.

Figure 2.14 shows the hydraulic scheme of solar thermal collector coupled to a hot water tank via a heat exchanger. There are two circuits, which often contain different fluids: the collector circuit connected to the solar thermal collector (water with anti-freeze solution, e.g. water-glycol mix) and the solar circuit connected to the hot water tank (water).
The mass flow rates in both circuits are regulated by pumps controlled by low-level controllers. The pumps adjust the mass flows such that the temperature in the collector cycle does not become too high (in order to avoid evaporation) and the produced thermal heat is maximized.
The system model consists of two parts: the model of the STC and the model of the heat distribution system connecting the collector to the hot water storage tank.

Solar Collector Model

The steady-state behavior of a flat plate solar collector can be easily described using its efficiency, which can be quite well expressed as a 2nd-order polynomial, as follows

$$\eta = \alpha_0 - \alpha_1 \frac{T_{\mathrm{col,out}} - T_{\mathrm{amb}}}{G_{\mathrm{tot}}} - \alpha_2 \frac{(T_{\mathrm{col,out}} - T_{\mathrm{amb}})^2}{G_{\mathrm{tot}}} \tag{2.37}$$

Here $\alpha_0, \alpha_1, \alpha_2$ denote parameters which are dependent on the optical performances as well as the heat loss properties of the solar collector field. They can usually found on the manufacturer's data sheets of the STC. G_{tot} represents the total specific energy gain by radiation and can be calculated as a function of the diffusive and the beam solar radiation. T_{amb} denotes the ambient temperature. The used parameters of the flat plate collector are shown in Table 2.4.

α_0	α_1 [W/m^2/K]	α_2 [W/m^2/K^2]
0.81	3.869	0.014

Table 2.4: Parameters for an example flat plate collector. Determined from tests according to DIN EN 12975.

By knowing the total aperture area A_{c} of the solar collector field and using the solar collector efficiency, the total heat gain $\dot{Q}_{\mathrm{col}}$ of the heat transfer fluid along the solar collector field can be calculated according to the following equation.

$$\dot{Q}_{\mathrm{col}} = \eta A_{\mathrm{c}} G_{\mathrm{tot}} \tag{2.38}$$

In order to obtain the information about the transient behavior of the outlet temperature $T_{\mathrm{col,in}}$ of the solar collector panel which is the inlet temperature of the solar circuit of the heat exchanger, we apply the energy balance at the level of the absorber plate:

$$\frac{\mathrm{d}}{\mathrm{d}t} T_{\mathrm{col,out}} = \frac{1}{C_{\mathrm{STC}}} \left(\dot{Q}_{\mathrm{col}} - \dot{m}_{\mathrm{col}} c_{\mathrm{w,g}} \left(T_{\mathrm{col,out}} - T_{\mathrm{col,in}} \right) \right) \tag{2.39}$$

using C_{STC} as the lumped heat capacity of the solar collector panel. The specific heat capacities of the heating mediums can be found in Table 2.5. $\dot{Q}_{\mathrm{col}}$ is taken from Equation 2.38 whereas the collector efficiency is provided by Equation 2.37.
The heat flow produced by the solar collector and entering the storage tank is computed as follows:

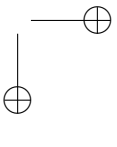
$$\dot{Q}_{\mathrm{sto}}(t) = \dot{m}_{\mathrm{sto}}(t) c_{\mathrm{w}} (T_{\mathrm{sto,in}}(t) - T_{\mathrm{sto,out}}(t)). \tag{2.40}$$

Heat Exchanger Model

The STC is usually connected to the storage tank via a heat exchanger (more precicely, a counter current flow plate heat exchanger). Instead of modelling the precice heat transfer behavior with a PDE, we use a lumped model with two discretization units resulting in a second order system for the heat exchanger dynamics. The heat exchanger model represents the dynamic behavior taking into account the nonlinearities arising from the heat transfer due to the mass flow rates as well as the temperature states. The following assumptions are made prior to the modeling of the system components:

- The medium considered for heating is incompressible.
- The medium density as well as its viscosity are considered constant.
- The thermal inertia of the medium in the pipes is neglected.
- There is no heat loss to the ambient.

The heat exchanger dynamics can be described with the following equations representing the energy flow inside the heat exchanger (energy balances):



$$\frac{\mathrm{d}}{\mathrm{d}t}T_{\mathrm{col,in}} = \frac{1}{m_{\mathrm{hx,col}}c_{\mathrm{w,g}}}\left(\dot{m}_{\mathrm{col}}c_{\mathrm{w,g}}\left(T_{\mathrm{col,out}} - T_{\mathrm{col,in}}\right) - U_{\mathrm{hx}}A_{\mathrm{hx}}\Delta T_{\mathrm{hx,mean}}\right)$$
$$\frac{\mathrm{d}}{\mathrm{d}t}T_{\mathrm{sto,in}} = \frac{1}{m_{\mathrm{hx,sto}}c_{\mathrm{w}}}\left(-\dot{m}_{\mathrm{sto}}c_{\mathrm{w}}\left(T_{\mathrm{sto,in}} - T_{\mathrm{sto,out}}\right) + U_{\mathrm{hx}}A_{\mathrm{hx}}\Delta T_{\mathrm{hx,mean}}\right)$$

ΔT_{mean} corresponds to the logarithmic mean

$$\Delta T_{\mathrm{hx,mean}} = \frac{\left(T_{\mathrm{col,out}} - T_{\mathrm{sto,in}}\right) - \left(T_{\mathrm{col,in}} - T_{\mathrm{sto,out}}\right)}{\ln\frac{T_{\mathrm{col,out}} - T_{\mathrm{sto,in}}}{T_{\mathrm{col,in}} - T_{\mathrm{sto,out}}}}$$

and is approximates by the algebraic mean temperature difference in the heat exchanger (to avoid the resulting nonlinearities)

$$\Delta T_{\mathrm{hx,mean}} = \frac{1}{2}\left(T_{\mathrm{col,out}} - T_{\mathrm{sto,in}}\right) + \frac{1}{2}\left(T_{\mathrm{col,in}} - T_{\mathrm{sto,out}}\right)$$

The last term in both equationd refers to the transfer of heat between the hot and the cold lumps.

Specific heat capacity water	$c_{\mathrm{w}} = 4.18\mathrm{kJ/kgK}$
Specific heat capacity water glycol (anti-freeze liquid)	$c_{\mathrm{w,g}} = 3.7\mathrm{kJ/kgK}$
Heat exchanger surface	$A_{\mathrm{hx}} = 4\mathrm{m}^2$
Specific heat transfer coefficient	$U_{\mathrm{hx}} = 3150.0\mathrm{W/m^2K}$
Mass heat exchanger material	$m_{\mathrm{hx,sto}} = 3.8\mathrm{kg}$
Mass heat exchanger material	$m_{\mathrm{hx,col}} = 3.8\mathrm{kg}$
Lumped heat capacity of the solar collectors including the contained medium	$C_{\mathrm{STC}} = 2.6\mathrm{MJ/K}$

Table 2.5: Parameters for the heat exchanger

Total Solar Thermal System Model

For the subsequent controller design the overall model is given using state-space formulation:

$$\dot{x} = f(x,u)$$
$$y = g(x)$$

with the state vector $x = [T_{\mathrm{col,in}}, T_{\mathrm{col,out}}, T_{\mathrm{sto,in}}]$ and the output vector $y = [T_{\mathrm{col,in}}, T_{\mathrm{sto,in}}]$ as well as the input $u = [\dot{m}_{\mathrm{col}}, \dot{m}_{\mathrm{sto}}]$ with the following model equations:

$$\begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}T_{\mathrm{col,in}} \\ \frac{\mathrm{d}}{\mathrm{d}t}T_{\mathrm{sto,in}} \\ \frac{\mathrm{d}}{\mathrm{d}t}T_{\mathrm{col,out}} \end{bmatrix} = \begin{bmatrix} \frac{1}{m_{\mathrm{hx,col}}c_{\mathrm{w,g}}}\left(\dot{m}_{\mathrm{col}}c_{\mathrm{w,g}}\left(T_{\mathrm{col,out}} - T_{\mathrm{col,in}}\right) - U_{\mathrm{hx}}A_{\mathrm{hx}}\Delta T_{\mathrm{hx,mean}}\right) \\ \frac{1}{m_{\mathrm{hx,sto}}c_{\mathrm{w}}}\left(-\dot{m}_{\mathrm{sto}}c_{\mathrm{w}}\left(T_{\mathrm{sto,in}} - T_{\mathrm{sto,out}}\right) + U_{\mathrm{hx}}A_{\mathrm{hx}}\Delta T_{\mathrm{hx,mean}}\right) \\ \frac{1}{C_{\mathrm{STC}}}\left(A_c\left(\alpha_0 G_{\mathrm{tot}} - \alpha_1\left(T_{\mathrm{col,out}} - T_{\mathrm{amb}}\right)\right.\right. \\ \left.\left.-\alpha_2\left(T_{\mathrm{col,out}} - T_{\mathrm{amb}}\right)^2\right) - \dot{m}_{\mathrm{col}}c_{\mathrm{w,g}}\left(T_{\mathrm{col,out}} - T_{\mathrm{col,in}}\right)\right) \end{bmatrix}.$$

$p(t) = [T_{\mathrm{amb}}, G_{\mathrm{tot}}, T_{\mathrm{sto,out}}]$ can be considered as external disturbances. Note that the system is nonlinear but can be expressed as a control-affine system:

$$\dot{x} = f_0(x) + f_1(x)u$$
$$y = g(x)$$

This makes a linearization approach easier (e.g., around a reference point of $T_{\mathrm{col,in}}$ and $T_{\mathrm{sto,in}}$).
If the STC is integrated into a broader building energy system or heating network, as depicted in Figure 2.15, it may be justifiable to omit the heat exchanger dynamics from the model. In this context, the complexities and interactions within the larger system may render the specific details of the heat exchanger less significant. Instead a rough estimation of the temperature offset can be used: $T_{\mathrm{sto,in}} = T_{\mathrm{col,out}} - \Delta T_{\mathrm{hx,offset}}$, $T_{\mathrm{col,in}} = T_{\mathrm{sto,out}} + \Delta T_{\mathrm{hx,offset}}$.

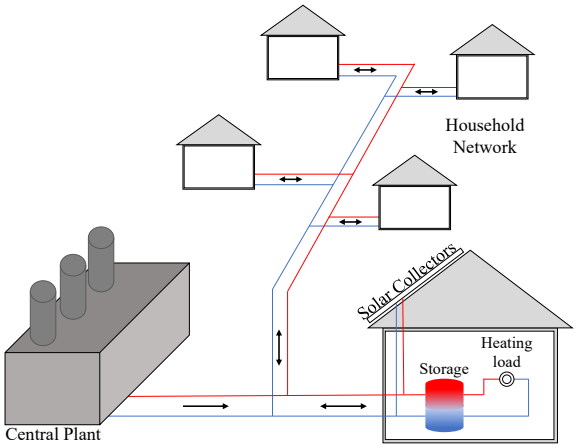


Figure 2.15: Prosumer-based heating network with building-level solar collectors connected to storage tanks

Control task

Different control tasks are:

- Control $\dot{m}_{\mathrm{sto}}$ and $\dot{m}_{\mathrm{col}}$ such that the heat exchanger outlet temperature $T_{\mathrm{col,in}}$ and/or $T_{\mathrm{sto,in}}$ follow a reference
- Control $\dot{m}_{\mathrm{sto}}$ and $\dot{m}_{\mathrm{col}}$ such that the produced thermal energy is maximized (i.e. thermal losses are minimized)
- Keep $T_{\mathrm{sto,in}}$, the temperature entering the STC system low (better efficiency due to smaller thermal losses)

2.2.4 Modelling of thermal energy networks

We now move from the building level to consider district heating networks, which supply a large number of consumers with heat. A district heating network consisting of several heat consumers (e.g. buildings or quarters consisting of several buildings) and one or more heat generators is given in Figure 2.16. Traditional district heating networks contain one central heat provider (and possibly several peak load boilers). In this case, each building independently controls its heat inflow and the central heat generator adjusts the heating operation to the total heat consumption in the heat network.

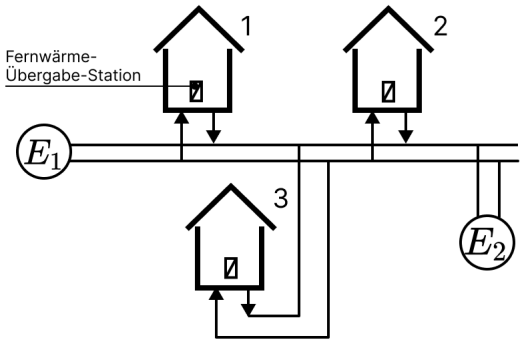


Figure 2.16: Heat grid with two heat producers

With multiple generators, the control is more complicated and must be done by a higher level production planning control algorithm taking into account various factors (energy prices, electricity prices, weather,...). District Heating Networks typically comprise a network of pipes responsible for distributing heat in the form of hot water or steam, from the source of generation to the end-users. The supply pipe delivers hot

water to the users, while the return pipe transfers the cold return water back. The hydraulic modelling of a heating network comprises pressure and mass flow rates in the pipes (or flow velocity), and the thermal model comprises supply and return temperatures and heat power at supply and load nodes.
The temperature dynamics in the district heating grid can be modeled using a physical pipe model based on advection transport (movement of fluid by velocity) with a one-dimensional partial differential equation (1D-PDE). A delay differential equation approach describing the time delay of heated fluid is also possible. However, the computational intensity of these models can be high and they are not suitable for control of the complete district heating system. Therefore, we apply a steady state modelling approach. Each component of the pipe (e.g. fluid, but also pipe, insulation, etc.) is assigned a mass whose temperature is equal to the average temperature of that component over the length of the pipe. As a simplification we consider only incoming and outgoing thermal power flows at the network nodes. Additionally, we neglect pressure loss due to pipe friction or other resistances like valves, bends and joints. Otherwise a nonlinear equation describing the pressure change would have to be considered.
By *continuity of flow* (analogues to network Kirchhoff's current law in electrical networks) we obtain the following rule: the power flow that enters into a node is equal to the power flow that leaves the node (which includes also the power consumption at the node), i.e.

$$\left(\sum P_{\text{in}}\right) - \left(\sum P_{\text{out}}\right) = 0. \tag{2.41}$$

Using a nodal-branch incidence matrix A_h where P_{wv} denotes the power flow of pipe $w - v$ between node w and v, $P_{s,i}$ is the power flow injected from the heat sources and $P_{l,j}$ is the power flow discharged to the heat load, we obtain the following system description.

$$\sum_{w} \mathrm{A}_h P_{wv} + \sum_{s,i \in w} P_{s,i} - \sum_{l,j \in w} P_{l,j} = 0 \tag{2.42}$$

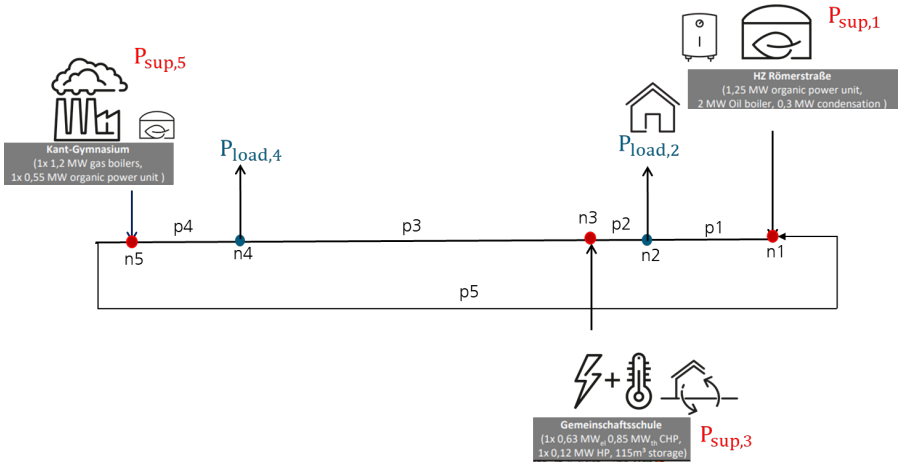


Figure 2.17: Simplified version of the district heating grid in Weil am Rhein with three heat producers and two cumulated heat consumers (think of small residential areas). The network contains five nodes and a loop.

Control tasks

Different control task arise in network production planning. For instance, if we consider a simplified version of the district heating grid in Weil am Rhein, see 2.17, the question is how to optimally operate heat producers (CHP vs. Biomass, auxiliary boilers) depending on environmental conditions (energy prices, electricity self-coverage) while ensuring that also buildings at the network edges get enough heat? If auxiliary boilers are needed, they may only be turned on once a day.

Multi-energy systems

An extension, which we do not consider further, is to move from the heating network level to consider so-called multi-energy systems, which supply a large number of consumers with electricity, heat and sometimes gas. Such a system involves the integration of an electrical system, a heat system, and sometime a gas network, and coupling units. The three networks are interconnected through the use of combined heat and power (CHP) units and electric boilers. These integrated networks are often found in the context of small regional energy clusters and self-contained island networks. With coordinated planning and scheduling, this multi-energy system can boost energy supply efficiency by leveraging the complementarity of different energy sources.

2.2.5 Wind energy systems modeling

In this course, we consider modern, variable-speed, horizontal-axis wind turbines. These wind turbines are large, heavy and flexible machines, with dynamics governed by intricate aero-elastic interactions between the turbine rotor blades and the wind stream. Therefore, accurate models of wind turbine dynamics usually require extensive aero-elastic simulations, which are computationally intensive. However, when we need to construct control-oriented models, we aim to represent only the most critical and fundamental dynamic behaviors to obtain a feasible model for computation, while disregarding unnecessary details.
Fig. 2.18 depicts a block diagram of a control-oriented wind turbine model that considers only the highest level control inputs and controlled outputs. The control inputs are the torque T_{G} applied to the generator and the collective pitch rate $\dot{\theta}$ of the rotor blades. While many different types of measurements are available, typically only two outputs are directly relevant for control. The first output is the electrical power output P_{el}. The second output to be influenced is the acceleration of the turbine tower top $\ddot{x}_{\mathrm{t}}$. The tower top motion is caused by the large aerodynamic thrust force exerted on the rotor, combined with the large lever arm of the tower. Minimizing the tower oscillation amplitude is an important control goal in order to reduce fatigue, and increase the life time of the turbine or reduce costs. Finally, the wind speed v_{w} is an uncontrolled input to the system, which is therefore modeled as a disturbance.

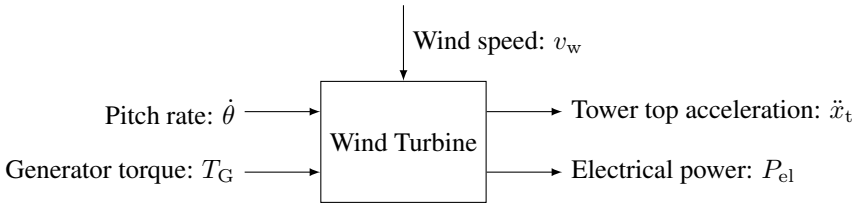


Figure 2.18: Block diagram of a control-oriented variable-speed wind turbine model with two inputs, two outputs and one disturbance (the wind speed).

The wind turbine model consists of three submodels, as depicted in the schematic diagram in Fig. 2.19:

1. **Aerodynamics**: The aerodynamics model is needed to compute the aerodynamic thrust force F_{A} and torque T_{A} generated on the rotor as a function of the pitch angle θ of the rotor blades, the rotor angular speed ω_{R} and the relative wind speed v_{rel}. This relative wind speed is the wind speed perceived by an observer fixed to the rotor. The rotor moves in the wind speed direction with the tower top speed $\dot{x}_{\mathrm{t}}$, so that $v_{\mathrm{rel}} = v_{\mathrm{w}} - \dot{x}_{\mathrm{t}}$. The tower top speed is an output of the rotor-tower model, and the rotor speed ω_{R} is an output of the drive train model.

2. **Drive train**: The drive train dynamics model the transfer of mechanical power from the rotor blades to the generator. This transfer does not occur instantaneously because of the elasticity of the rotor blades which cannot be neglected. Thus, the drive train receives as inputs the aerodynamic torque T_{A} on the rotor side and the generator torque T_{G} on the generator side. These inputs then determine the rotor speed ω_{R} and the electrical power output P_{el}.

3. **Rotor-tower dynamics:** The rotor-tower dynamics model the bending of the tower and the rotor blades in the wind direction as a function of the aerodynamic thrust force F_{A}. This bending results in a displacement x_{t} of the turbine tower top in the wind direction.

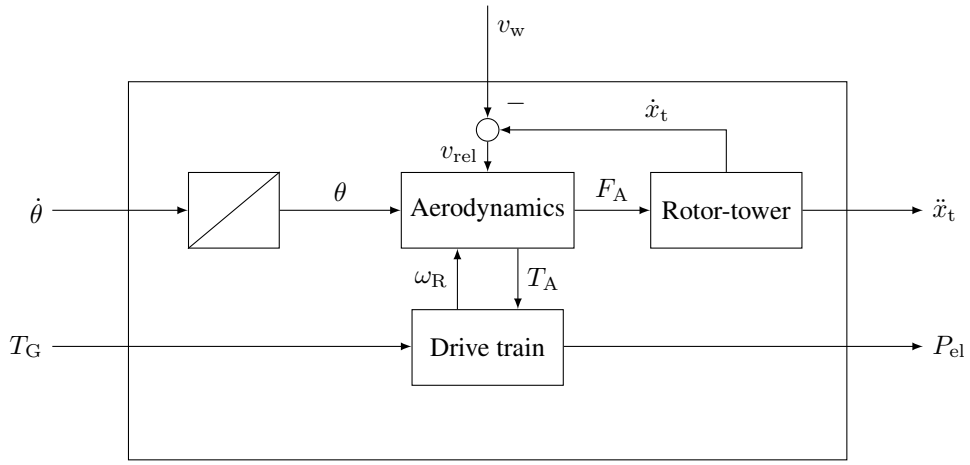


Figure 2.19: Block diagram of the wind turbine model with the three submodels and mutual dependencies.

In the following paragraphs, we will discuss each of these submodels in detail.

Aerodynamics The aerodynamic force F_A and torque T_A are implicitly determined by a force equilibrium equation. One the one hand, there are the aerodynamic forces generated by the rotor blades as a function of the pitch angle θ and the "tip-speed-ratio" (TSR) λ, which is defined as the ratio between the tip speed of the rotor blades and the relative wind speed v_rel:

$$\lambda = \frac{\omega_\mathrm{R} R}{v_\mathrm{rel}} \,, \tag{2.43}$$

with R the rotor radius. On the other hand, there is the change in wind speed, which is slowed down in front of the rotor as a result of the aerodynamic forces pointing in its direction. This wind speed reduction then again affects the aerodynamic forces that can be generated by the rotor blades.
This equilibrium can be modeled using momentum conservation equations, as done e.g. in Blade Element Momentum (BEM) theory, which typically results in a nonlinear and implicit mapping between $\theta, \omega_\mathrm{R}, v_\mathrm{rel}$ and F_A and T_A. This means that instead of an explicit expression for F_A and T_A, an implicit system of equations is formulated, that needs to be solved numerically for given values $\theta, \omega_\mathrm{R}, v_\mathrm{rel}$. This model is also static, i.e. it has no internal states, since it assumes that the change in aerodynamic equilibrium happens at a sufficiently fast time-scale.
The model can be summarized by

$$F_\mathrm{A} = \frac{1}{2}\rho\pi R^2 v_\mathrm{rel}^2 C_\mathrm{F}(\theta, \lambda) \tag{2.44}$$

$$P_\mathrm{A} = \frac{1}{2}\rho\pi R^2 v_\mathrm{rel}^3 C_\mathrm{P}(\theta, \lambda) \tag{2.45}$$

$$T_\mathrm{A} = P_\mathrm{A}\omega_\mathrm{R}^{-1} \,, \tag{2.46}$$

with ρ the air density and P_A the aerodynamic power extracted from the wind by the rotor. Computing the force and power coefficients $C_\mathrm{F}(\theta, \lambda)$ and $C_\mathrm{P}(\theta, \lambda)$ then requires the solution of a large set of implicit equations.
Applying this implicit model for embedded optimization is impractical and computationally too expensive. Therefore, the implicit model is solved offline for a grid of values of θ and λ within the relevant operational range. The force and power coefficients $C_\mathrm{F}(\theta, \lambda)$ and $C_\mathrm{P}(\theta, \lambda)$ are then provided as differentiable functions by means of polynomial approximations or by linear or splines-based interpolation of the computed data points.

[2]Wintermeyer-Kallen, T. et al., Weight-scheduling for linear time-variant model predictive wind turbine control toward field testing, Forschung im Ingenieurwesen, Vol. 85, pages 385-394 (2021)

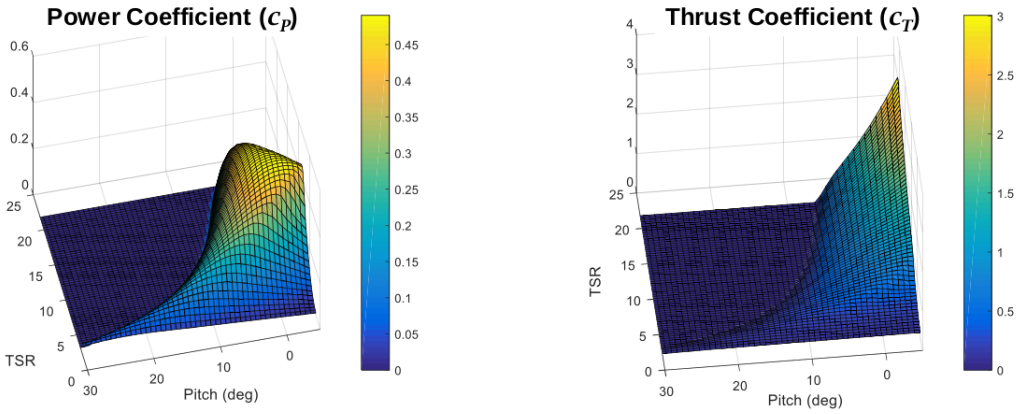


Figure 2.20: Example of a wind turbine power coefficient $C_{\mathrm{P}}(\theta, \lambda)$ (left) and thrust coefficient $C_{\mathrm{F}}(\theta, \lambda)$ (right), evaluated on a θ-λ-grid[2].

Fig. 2.20 shows an example of a power coefficient C_{P} and thrust coefficient C_{F} evaluated on a θ-λ-grid. The power coefficient reaches a maximum value at the location (θ^*, λ^*), with $C_{\mathrm{P,max}} = C_{\mathrm{P}}(\theta^*, \lambda^*)$ and $\theta^* = 0°$. The power coefficient can be interpreted as the fraction of the kinetic energy in the unperturbed wind speed v_{rel} that would flow through the entire rotor area, that is harvested by the rotor. It is important to note that there is a theoretical limit to the maximum power coefficient value, known as "Betz' limit". This limit states that a horizontal-axis rotor can harvest at most a factor $16/27 \approx 59\%$ of the kinetic energy in the wind.
This can be understood intuitively by considering the purely hypothetical and unphysical case where 100% of the kinetic energy is extracted from the wind by the rotor. In this case, the wind speed directly behind the wind turbine would be zero, and no wind would pass through the rotor, so that no power is extracted at all. Therefore, the maximum power coefficient is the result of a trade-off between extracting kinetic energy from the wind, without slowing down the wind too much. Precisely this trade-off is modeled with the BEM method used to produce the power coefficient data in Fig. 2.20.
To get a deeper understanding, it is instructive to examine a cross section of the power and force coefficient in Fig. 2.20 for $\theta^* = 0$. For $\lambda < \lambda^*$, it holds that $C_{\mathrm{F}}(\theta^*, \lambda) < C_{\mathrm{F}}(\theta^*, \lambda^*)$, meaning that the thrust force is lower than at the optimal operating point. Thus, for these values of λ, the rotor could rotate faster and apply more force to the wind to extract more energy and obtain a higher power coefficient. For $\lambda > \lambda^*$ on the other hand, it holds that $C_{\mathrm{F}}(\theta^*, \lambda) > C_{\mathrm{F}}(\theta^*, \lambda^*)$. Thus, by rotating faster than the optimal speed, a larger thrust force can be generated than in the optimal point. However, in this case, the corresponding decrease in wind speed at the rotor results in an overall decrease in power coefficient.
The main function of the variable pitch angle θ of the rotor blades is to give control authority over the wind turbine for wind speeds that do not allow for operation at the optimal TSR λ^*. To understand how and why this occurs, we have to first identify the different operating regions of a wind turbine as a function of the wind speed, as depicted in Fig. 2.21. We discern six different operating regions:

- **Region I**: (low wind/idle). For very low wind speeds, the turbine is idling with zero speed, which is enforced by a brake.

- **Region IIA**: (minimum rotor speed). Above the so-called "cut-in" wind speed, the turbine starts to operate. However, in order to avoid the low resonance frequencies of the tower construction, the rotor must rotate at a certain minimal speed $\omega_{\mathrm{R,min}}$. Therefore, directly above the cut-in wind speed, the turbine is forced to rotate at a sub-optimal TSR $\lambda = \frac{\omega_{\mathrm{R,min}} R}{v_{\mathrm{w}}} > \lambda^*$. Therefore, a fine pitch $\theta > \theta^*$ is applied to maximize $C_{\mathrm{P}}(\theta, \lambda)$ for the imposed TSR λ.

- **Region IIB**: (subrated regime). At some point the wind speed v_{w} reaches a value that yields the optimal TSR λ^* with the minimal rotor speed. From there on, the rotor speed ω_{R} increases linearly

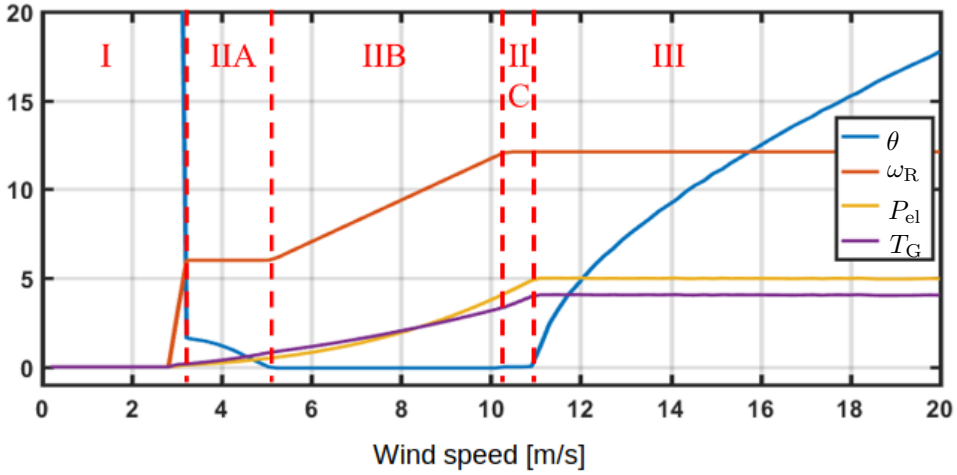


Figure 2.21: Operating regions of a variable-speed wind turbine as a function of wind speed.

with the wind speed to maintain λ^* and maximize C_{P} and thus maximize energy capture. The pitch angle is constant and set to $\theta^* = 0°$.

- **Region IIC**: (maximum rotor speed). When the maximum rotor speed $\omega_{\mathrm{R,max}}$ is reached, the TSR is again imposed by the wind speed, now to a value $\lambda = \frac{\omega_{\mathrm{R,max}} R}{v_{\mathrm{w}}} < \lambda^*$. However, in this region, the generator torque can still be increased and therefore again a fine pitch angle θ is applied to maximize C_{P} for the imposed TSR.
- **Region III**: (rated regime) In this region, the generator is operating at maximum speed and torque. However, the aerodynamic power captured by the turbine still grows $\sim v_{\mathrm{w}}^3$ and hence needs to be capped to satisfy these constraints. This is done by increasing the pitch angle of the rotor blades so as to reduce the "angle-of-attack" of the blades, which results in lower thrust and power coefficients.
- **Region IV**: (shutdown) At very high wind speeds the structural integrity of the wind turbine cannot be safeguarded anymore. Once the cut-off speed is reached, the turbine is shut down to zero rotor speed. The pitch angle of the blade is chosen so as to minimize the aerodynamic forces on the non-rotating rotor blades.

Drive train The drive train submodel takes into account the inertia and elasticity of the rotating blades around the axis of rotation, as well as the combined inertia of all drive train components, including the rotor hub, gear box and main generator shaft. This model accounts for the lowest eigenfrequency of drive train oscillations. It is modeled as linear, two-mass torsional spring-damper system as shown in Fig. 2.22.

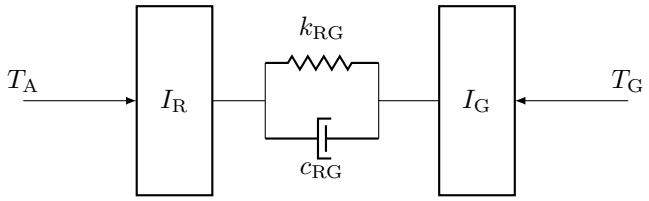


Figure 2.22: Two-mass torsional spring-damper system model of the wind turbine drive train dynamics.

The dynamic equations are given by

$$I_{\mathrm{R}}\dot{\omega}_{\mathrm{R}} = -c_{\mathrm{RG}}\Delta\dot{\varphi}_{\mathrm{RG}} - k_{\mathrm{RG}}\Delta\varphi_{\mathrm{RG}} + T_{\mathrm{A}} \tag{2.47}$$

$$I_{\mathrm{G}}\dot{\omega}_{\mathrm{G}} = c_{\mathrm{RG}}\Delta\dot{\varphi}_{\mathrm{RG}} + k_{\mathrm{RG}}\Delta\varphi_{\mathrm{RG}} + T_{\mathrm{G}} \tag{2.48}$$

$$\Delta\dot{\varphi}_{\mathrm{RG}} = \omega_{\mathrm{R}} - \omega_{\mathrm{G}} \ , \tag{2.49}$$

with I_{R} the moment of inertia of the rotor blades and I_{G} the moment of inertia of the generator, gear box and rotor hub. The angle $\Delta\varphi_{\mathrm{RG}}$ is the angular displacement of the rotor blades relative to the rotor hub. The generator angular speed is given by ω_{G}. Note that the quantities ω_{G}, T_{G}, and thus also I_{G}, are defined on the "rotor side" of the gear box, so that the gear ratio of the gear box is taken into account implicitly. The parameters k_{RG} and c_{RG} are the spring and damper coefficients respectively.
The drive train model thus has three states $x_{\mathrm{dt}} := (\omega_{\mathrm{R}}, \omega_{\mathrm{G}}, \Delta\varphi_{\mathrm{RG}})$ and two inputs $u_{\mathrm{dt}} := (T_{\mathrm{A}}, T_{\mathrm{G}})$ and is summarized by the LTI dynamics

$$\dot{x}_{\mathrm{dt}} = A_{\mathrm{dt}}x_{\mathrm{dt}} + B_{\mathrm{dt}}u_{\mathrm{dt}} \ . \tag{2.50}$$

It is left as an exercise for the reader to formulate the system matrix A_{dt} and control matrix B_{dt}.
The output of the drive train model is the electrical power $P_{\mathrm{el}} = T_{\mathrm{G}}\omega_{\mathrm{G}}$.

Rotor-tower The rotor-tower dynamics are modeled as an LTI system

$$\dot{x}_{\mathrm{rt}} = A_{\mathrm{rt}}x_{\mathrm{rt}} + B_{\mathrm{rt}}u_{\mathrm{rt}} \ , \tag{2.51}$$

with states $x_{\mathrm{rt}} := (x_{\mathrm{t}}, \dot{x}_{\mathrm{t}}, \phi_{\mathrm{b}}, \dot{\phi}_{\mathrm{b}})$ and input $u_{\mathrm{t}} := (F_{\mathrm{A}})$. The variable ϕ_{b} represents the collective flap-wise bending angle of the rotor blades. For a derivation of system matrices A_{rt} and B_{rt}, we refer to [6]. Fig. 2.23 gives a schematic overview of the model.

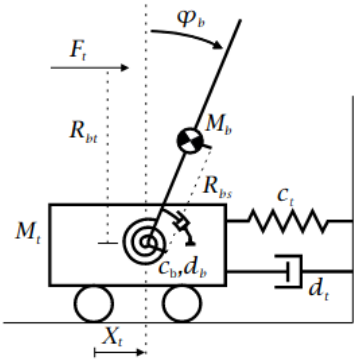


Figure 2.23: Rotor-tower system [6].

Summary The overall system model can be summarized as a nonlinear ODE and an output equation:

$$\dot{x}(t) = f(x(t), u(t), w(t)) \tag{2.52}$$

$$y(t) = g(x(t), u(t)) \tag{2.53}$$

with the nonlinearity arising in the static aerodynamic model, which depends on the state variables of the remaining sub-models. The state $x \in \mathbb{R}^8$, controls $u \in \mathbb{R}^2$, disturbance $w \in \mathbb{R}$ and outputs $y \in \mathbb{R}^2$ are defined as

$$x := (\theta, \omega_{\mathrm{R}}, \omega_{\mathrm{G}}, \Delta\varphi_{\mathrm{RG}}, x_{\mathrm{t}}, \dot{x}_{\mathrm{t}}, \phi_{\mathrm{b}}, \dot{\phi}_{\mathrm{b}}) \tag{2.54}$$

$$u := (\dot{\theta}, T_{\mathrm{G}}) \tag{2.55}$$

$$w := (v_{\mathrm{w}}) \tag{2.56}$$

$$y := (\ddot{x}_{\mathrm{t}}, P_{\mathrm{el}}) \ . \tag{2.57}$$

Chapter 3

Background on Optimization

The main principle behind MPC involves solving an optimization problem in real-time, tailored to reflect the control algorithm's objectives. This chapter covers various classifications of optimization problems and introduces the concept of convexity. Moreover, it examines the optimality conditions for nonlinear programs.

3.1 Definition of an Optimization Problem

Mathematical optimization refers to finding the best, or *optimal* solution among a set of possible decisions, where optimality is defined with the help of an *objective function*. Some solution candidates are *feasible*, others not, and it is assumed that *feasibility* of a solution candidate can be checked by evaluation of some *constraint functions* that need for example be equal to zero.
The most universal formulation of an optimization problem is that of a nonlinear optimization problem. They are also at the heart of nonlinear MPC. Traditionally, the field of nonlinear optimization is called *Nonlinear Programming* (similar to linear programming, mixed-integer programming, ...) because they were solved with the help of computer programs. The general form of a Nonlinear Programming Problem (NLP) is given by

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && \text{(3.1a)} \\ \text{subject to} \quad & g(x) = 0, && \text{(3.1b)} \\ & h(x) \leq 0, && \text{(3.1c)} \end{aligned}$$

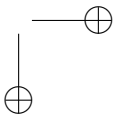
where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^{n_g}$, $h : \mathbb{R}^n \to \mathbb{R}^{n_h}$, are assumed to be continuously differentiable at least once, often twice and sometimes more. Differentiability of all problem functions allows us to use algorithms that are based on derivatives, in particular the so called "Newton-type optimization methods" which are the basis of many numerical optimization algorithms.

Example (A two dimensional example)**.**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & x_1^2 + x_2^2 && \text{(3.2)} \\ \text{subject to} \quad & x_2 - 1 - x_1^2 \geq 0, && \text{(3.3)} \\ & x_1 - 1 \geq 0. && \text{(3.4)} \end{aligned}$$

Definitions

Let us start by making a few definitions to clarify the language.



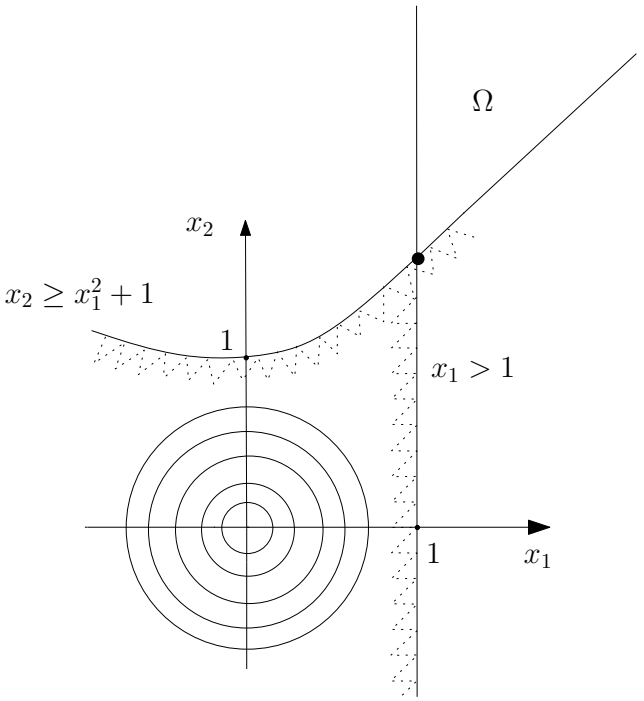


Figure 3.1: Visualization of Example 3.1, Ω is defined in Definition 1

Definition 1. *The "feasible set" Ω is defined to be the set $\Omega := \{x \in \mathbb{R}^n | g(x) = 0,\ h(x) \leq 0\}$.*

Definition 2. *The point $x^* \in \mathbb{R}^n$ is a "global minimizer" if and only if (iff) $x^* \in \Omega$ and $\forall x \in \Omega :\ f(x) \geq f(x^*)$.*

Definition 3. *The point $x^* \in \mathbb{R}^n$ is a "local minimizer" iff $x^* \in \Omega$ and there exists a neighborhood $\mathcal{N}$ of x^* (e.g. an open ball around x^*) so that $\forall x \in \Omega \cap \mathcal{N} :\ f(x) \geq f(x^*)$.*

Definition 4 (Active/Inactive Constraint)**.** *An inequality constraint $h_i(x) \leq 0$ is called "active" at $x^* \in \Omega$ iff $h_i(x^*) = 0$ and otherwise "inactive".*

Definition 5 (Active Set)**.** *The index set $\mathcal{A}(x^*) \subset \{1, \ldots, n_h\}$ of active constraints is called the "active set".*

3.2 Classes of Optimization Problems

In order to choose the right algorithm for a practical problem, we should know how to classify it and which mathematical structures can be exploited. Replacing an inadequate algorithm by a suitable one can make solution times many orders of magnitude shorter. Many problems have more structure, which we should recognize and exploit in order to solve problems faster. An important such structure is convexity which allows us to to find global minima by searching for local minima only.

3.2.1 Convex Optimization Problems

"The great watershed in optimization is not between linearity and nonlinearity, but convexity and nonconvexity"
R. Tyrrell Rockafellar

What is convexity, and why is it so important for optimizers?

Definition 6 (Convex Set)**.** *A set $\Omega \subset \mathbb{R}^n$ is convex if*

$$\forall x, y \in \Omega, t \in [0,1] : \; x + t(y - x) \in \Omega. \tag{3.5}$$

("all connecting lines lie inside set")

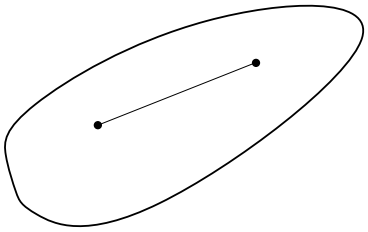


Figure 3.2: An example of a convex set

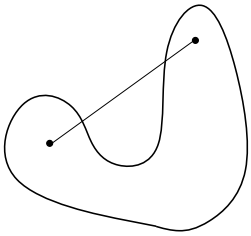


Figure 3.3: An example of a non convex set

Definition 7 (Convex Function)**.** *A function $f : \Omega \to \mathbb{R}$ is* convex*, if Ω is convex and if*

$$\forall x, y \in \Omega, t \in [0,1] : \; f(tx + (1-t)y) \leq tf(x) + (1-t)f(y). \tag{3.6}$$

("all secants are above graph").
A function $f : \Omega \to \mathbb{R}$ is strictly convex*, if Ω is convex and if*

$$\forall x, y \in \Omega, t \in [0,1] : \; f(tx + (1-t)y) < tf(x) + (1-t)f(y). \tag{3.7}$$

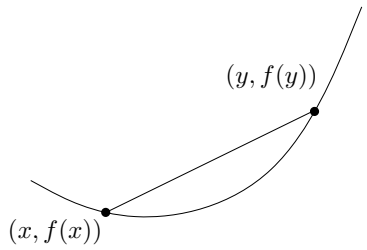


Figure 3.4: For a convex function, the line segment between any two points on the graph (secants) lies above the graph.

Definition 8 (Convex Optimization Problem)**.** *An optimization problem with convex feasible set Ω and convex objective function $f : \Omega \to \mathbb{R}$ is called a "convex optimization problem".*

Note that the feasible set Ω of an optimization problem is convex if the function g is affine and the functions h_i are convex.
For a convex optimization problem, every local minimum is also a global one. Note that this property does not imply uniqueness (nor existence). To guarantee a unique minimum, we must have a strictly convex objective function.

How to detect convexity of functions? Convexity is related to the positive curvature of a function, which can be determined by positive semi-definiteness of the Hessian (matrix of second-order partial derivatives).

Definition 9 (Generalized Inequality for Symmetric Matrices)**.** *We write for a symmetric matrix $B = B^T$, $B \in \mathbb{R}^{n \times n}$ that "$B \succcurlyeq 0$" if and only if B is positive semi-definite i.e., if $\forall z \in \mathbb{R}^n : \; z^T B z \geq 0$, or, equivalently, if all (real) eigenvalues of the symmetric matrix B are non-negative:*

$$B \succcurlyeq 0 \iff \min \operatorname{eig}(B) \geq 0.$$

Similarly, we write $B \succ 0$ iff B is positive definite, i.e. if $\forall z \in \mathbb{R}^n \setminus \{0\} : z^T B z > 0$, or, equivalently, if all eigenvalues of B are positive

$$B \succ 0 \iff \min \operatorname{eig}(B) > 0.$$

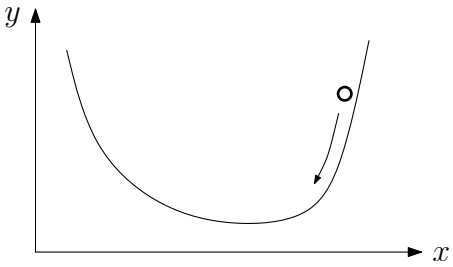


Figure 3.5: Every local minimum is also a global one for a convex function

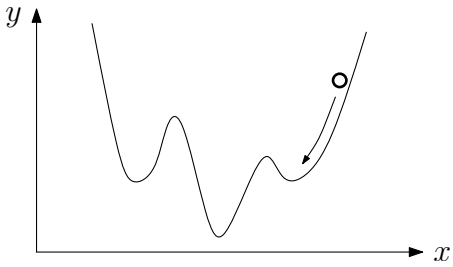


Figure 3.6: Not every local minimum is also a global one for this nonconvex function

Theorem 2 (Convexity for C^2 Functions)**.** *Assume that $f : \Omega \to \mathbb{R}$ is twice continuously differentiable and Ω convex and open. Then holds that f is convex if and only if for all $x \in \Omega$ the Hessian is positive semi-definite, i.e.*

$$\forall x \in \Omega : \quad \nabla^2 f(x) \succcurlyeq 0. \tag{3.8}$$

Similarly, f is strictly convex if and only if for all $x \in \Omega$ the Hessian is positive definit, i.e.

$$\forall x \in \Omega : \quad \nabla^2 f(x) \succ 0. \tag{3.9}$$

Example (Quadratic Function)**.** The function $f(x) = c^T x + \frac{1}{2} x^T B x$ is convex if and only if $B \succcurlyeq 0$, because $\forall x \in \mathbb{R}^n : \ \nabla^2 f(x) = B$.

3.2.2 Quadratic Programming (QP)

If in the general NLP formulation (3.1) the constraints g, h are affine, and the objective is a linear-quadratic function, we call the resulting problem a Quadratic Programming Problem or Quadratic Program (QP). Linear MPC relies heavily on solving QPs as the discretized form of the control problem takes the shape of a QP. A general QP can be formulated as follows.

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x + \frac{1}{2} x^T B x && \text{(3.10a)} \\ \text{subject to} \quad & Ax - b = 0, && \text{(3.10b)} \\ & Cx - d \leq 0. && \text{(3.10c)} \end{aligned}$$

Here, the problem data are $c \in \mathbb{R}^n, A \in \mathbb{R}^{n_g \times n}, b \in \mathbb{R}^{n_g}, C \in \mathbb{R}^{n_h \times n}, d \in \mathbb{R}^{n_h}$, as well as the "Hessian matrix" $B \in \mathbb{R}^{n \times n}$. Its name stems from the fact that $\nabla^2 f(x) = B$ for $f(x) = c^T x + \frac{1}{2} x^T B x$. If $B = 0$ the QP simplifies to a linear program (LP).
The eigenvalues of B decide on convexity or non-convexity of a QP, i.e. the possibility to solve it in polynomial time to global optimality, or not. If $B \succcurlyeq 0$ we speak of a convex QP, and if $B \succ 0$ we speak of a strictly convex QP. The latter class has the agreeable property that it always has unique minimizers.

Example (A non-convex QP)**.**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & \begin{bmatrix} 0 & 2 \end{bmatrix} x + \frac{1}{2} x^T \begin{bmatrix} 5 & 0 \\ 0 & -1 \end{bmatrix} x && \text{(3.11)} \\ \text{subject to} \quad & -1 \leq x_1 \leq 1, && \text{(3.12)} \\ & -1 \leq x_2 \leq 10. && \text{(3.13)} \end{aligned}$$

This problem has local minimizers at $x_a^* = (0, -1)^T$ and $x_b^* = (0, 10)^T$, but only x_b^* is a global minimizer.

Example (A strictly convex QP)**.**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & \begin{bmatrix} 0 & 2 \end{bmatrix} x + \frac{1}{2} x^T \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} x && (3.14) \\ \text{subject to} \quad & -1 \leq x_1 \leq 1, && (3.15) \\ & -1 \leq x_2 \leq 10. && (3.16) \end{aligned}$$

This problem has only one (strict) local minimizer at $x^* = (0, -1)^T$ that is also global minimizer.

3.2.3 Linear Programming (LP)

When the functions f, g, h are affine in the general formulation (3.1), the general NLP gets something easier to solve, namely a Linear Program (LP). Explicitly, an LP can be written as follows:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x && (3.17a) \\ \text{subject to} \quad & Ax - b = 0, && (3.17b) \\ & Cx - d \leq 0. && (3.17c) \end{aligned}$$

Here, the problem data is given by $c \in \mathbb{R}^n, A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q \times n}$, and $d \in \mathbb{R}^q$. Note that we could also have a constant contribution to the objective, i.e. have $f(x) = c^T x + c_0$, but that this would not change the minimizers x^*.
LPs can be solved very efficiently since the 1940's, when G. Dantzig invented the famous "simplex method", an "active set method", which is still widely used, but got an equally efficient competitor in the so called "interior point methods". LPs can nowadays be solved even if they have millions of variables and constraints. Every business student knows how to use them, and LPs arise in myriads of applications. LP algorithms are not treated in detail in this lecture, but please recognize them if you encounter them in practice and use the right software.

Example (LP resulting from oil shipment cost minimization)**.** We regard a typical logistics problem that an oil production and distribution company might encounter. We want to minimize the costs of transporting oil from the oil producing countries to the oil consuming countries, as visualized in Figure 3.7.

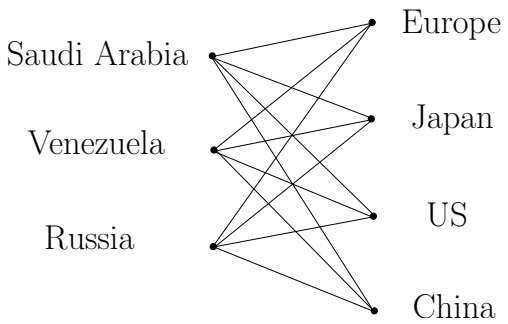


Figure 3.7: A traditional example of a LP problem: minimize the oil shipment costs while satisfying the demands on the right and not exceeding the production capabilities on the left.

More specifically, given a set of n oil production facilities with production capacities p_i with $i = 1, \ldots, n$, and given a set of m customer locations with oil demands d_j with $j = 1, \ldots, m$, and given shipment costs c_{ij} for all possible routes between each i and j, we want to decide how much oil should be transported along each route. These quantities, which we call x_{ij}, are our decision variables, in total nm real valued variables. The problem can be written as the following linear program.

$$
\begin{aligned}
\underset{x\in\mathbb{R}^{n\times m}}{\text{minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} \le p_i, \quad i = 1,\dots,n, \\
& \sum_{i=1}^{n} x_{ij} \ge d_j, \quad j = 1,\dots,m, \\
& x_{ij} \ge 0, \quad i = 1,\dots,n, \quad j = 1,\dots,m.
\end{aligned}
\tag{3.18}
$$

Software for solving LPs: Gurobi, CPLEX, MATLAB's linprog, open-source: lp_solve, Google OR-Tools, Pyomo, PuLP (Python), SciPy's optimize.linprog

3.2.4 Mixed-Integer Programming (MIP)

A Mixed-Integer Programming problem or Mixed-Integer Program (MIP) is a problem with both real and integer decision variables. A MIP can be formulated as follows:

$$
\begin{aligned}
\underset{\substack{x\in\mathbb{R}^n \\ z\in\mathbb{Z}^m}}{\text{minimize}} \quad & f(x,z) && && \text{(3.19a)} \\
\text{subject to} \quad & g(x,z) &=& \ 0, && \text{(3.19b)} \\
& h(x,z) &\le& \ 0. && \text{(3.19c)}
\end{aligned}
$$

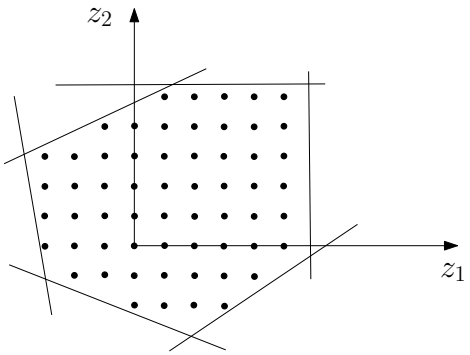


Figure 3.8: Visualization of the feasible set of an integer problem with linear constraints.

We can use 0–1 (binary) variables to

- to model decisions (yes/no)
- to force disjunctions (either-or)
- Enforce Logical Implications (If-Then)
- to model plant dynamics: on-off, minimum on-time/off-time, number of on-times, minimum power,...
- to model discontinuous dynamics (to some extend): phase change material (PCM) storage, changing flow direction

And, in the context of general modelling techniques:

- to model piecewise linear (continuous) functions
- to convexify/linearize nonlinear/nonconvex dynamics

Does a variable have the meaning of an indivisible physical size, it must be an integer: number of wind turbines to be placed, to take or not to take an (expensive) measurement, ...

Definition 10 (Mixed-Integer Linear Program (MILP))**.** *If f, g, h are affine in both x and z we speak of a Mixed-Integer Linear Program.*

In optimal control, mixed integer problems arise because some states or some controls can be integer valued, in which case the dynamic system is called a "hybrid system". Generally speaking, hybrid optimal control problems are more difficult to solve compared to problems with only continuous variables. One exception is the case when dynamic programming can be applied to a problem with purely discrete state and control spaces. Most of this lecture is concerned with continuous optimization.
These problems can be solved efficiently with commercial and open-source software.

Software for solving MILP: SCIP (open-source), BONMIN (Basic Open-source Nonlinear Mixed Integer Programming), CPLEX, Gurobi

Definition 11 (Mixed-Integer Nonlinear Program (MINLP))**.** *If f, g, h are twice differentiable in x and z we speak of a Mixed-Integer Nonlinear Program.*

Generally speaking, these problems are very hard to solve, due to the combinatorial nature of the variables z. However, if a *relaxed problem*, where the variables z are no longer restricted to the integers, but to the real numbers, is convex, often very efficient solution algorithms exist. More specifically, we would require that the following problem is convex:

$$\begin{aligned} \underset{\substack{x\in\mathbb{R}^n \\ z\in\mathbb{R}^m}}{\text{minimize}} \quad & f(x,z) && && \text{(3.20a)} \\ \text{subject to} \quad & g(x,z) && = \quad 0, && \text{(3.20b)} \\ & h(x,z) && \geq \quad 0. && \text{(3.20c)} \end{aligned}$$

The efficient solution algorithms are often based on the technique of "branch-and-bound", which uses partially relaxed problems where some of the z are fixed to specific integer values and some of them are relaxed. This technique then exploits the fact that the solution of the relaxed solutions can only be better than the best integer solution. This way, the search through the combinatorial tree can be made more efficient than pure enumeration.
Many of the aforementioned softwares for solving MILP can also solve convex MINLP.

3.3 Optimality Conditions

3.3.1 First Order Optimality Conditions

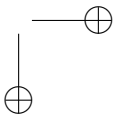
An important question in continuous optimization is if a feasible point $x^* \in \Omega$ satisfies necessary first order optimality conditions. If it does not satisfy these conditions, x^* cannot be a local minimizer. If it does satisfy these conditions, it is a hot candidate for a local minimizer. If the problem is convex, these conditions are even *sufficient* to guarantee that it is a global optimizer. Thus, most algorithms for nonlinear optimization search for such points. The first order condition can only be formulated if a technical "constraint qualification" is satisfied, which in its simplest and numerically most attractive variant coms in the following form.

Definition 12 (LICQ)**.** *The "linear independence constraint qualification" (LICQ) holds at $x^* \in \Omega$ iff all vectors $\nabla g_i(x^*)$ for $i \in \{1,\dots,n_g\}$ & $\nabla h_i(x^*)$ for $i \in \mathcal{A}(x^*)$ are linearly independent.*

To give further meaning to the LICQ condition, let us combine all active inequalities with all equalities in a map $\tilde{g}$ defined by stacking all functions on top of each other in a colum vector as follows:

$$\tilde{g}(x) = \begin{bmatrix} g(x) \\ h_i(x)(i \in \mathcal{A}(x^*)) \end{bmatrix}. \tag{3.21}$$

LICQ is then equivalent to full row rank of the Jacobian matrix $\frac{\partial \tilde{g}}{\partial x}(x^*)$.
This condition allows us to formulate the famous *Karush-Kuhn-Tucker (KKT)* optimality conditions.



Theorem 3 (KKT Conditions)**.** *If x^* is a local minimizer of the NLP* (3.1) *and LICQ holds at x^* then there exist so called multiplier vectors $\lambda^* \in \mathbb{R}^{n_g}$ and $\mu^* \in \mathbb{R}^{n_h}$ with*

$$\begin{aligned} \nabla f(x^*) + \nabla g(x^*)\lambda^* + \nabla h(x^*)\mu^* &= 0 && \text{(3.22a)} \\ g(x^*) &= 0 && \text{(3.22b)} \\ h(x^*) &\leq 0 && \text{(3.22c)} \\ \mu^* &\geq 0 && \text{(3.22d)} \\ \mu_i^* h_i(x^*) &= 0, \quad i = 1, \ldots, n_h. && \text{(3.22e)} \end{aligned}$$

We use the notation $\nabla g(x) = \frac{\partial g}{\partial x}(x)^T \in \mathbb{R}^{n \times n_g}$ where $\frac{\partial g}{\partial x}(x) \in \mathbb{R}^{n_g \times n}$ is the Jacobian matrix of g defined by

$$\frac{\partial g}{\partial x}(x) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{n_g}}{\partial x_1} & \cdots & \frac{\partial g_{n_g}}{\partial x_n} \end{bmatrix}.$$

Similarly, the Jacobian of h is defined.
Note: The KKT conditions are the first order necessary conditions for optimality (FONC) for constrained optimization, and are thus the equivalent to $\nabla f(x^*) = 0$ in unconstrained optimization. In the special case of convex problems, the KKT conditions are not only *necessary* for a *local* minimizer, but even *sufficient* for a *global* minimizer.

Theorem 4. *Regard a convex NLP and a point x^* at which LICQ holds. Then:*

$$x^* \text{ is a global minimizer} \iff \exists \lambda, \mu \text{ so that the KKT conditions hold.}$$

Definition 13 (Lagrangian Function)**.** *We define the so called "Lagrangian function" to be*

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x). \tag{3.23}$$

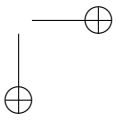
Here, we have used again the so called "Lagrange multipliers" or "dual variables" $\lambda \in \mathbb{R}^{n_g}$ and $\mu \in \mathbb{R}^{n_h}$. The Lagrangian function plays a crucial role in both convex and general nonlinear optimization, not only as a practical shorthand within the KKT conditions: using the definition of the Lagrangian, we have (3.22a) $\Leftrightarrow \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$.
The KKT conditions require the inequality multipliers μ to be positive, $\mu \geq 0$, while the sign of the equality multipliers λ is arbitrary. See Figures 3.9 and 3.10 for a graphical interpretation of the KKt conditions for equality and inequality constrained optimization problems. Figure 3.11 illustrates a failure of the LICQ constraint qulification condition defined in Definition 12.
The last three KKT conditions (3.22c)-(3.22e) are called the *complementarity* conditions. For each index i, they define an L-shaped set in the (h_i, μ_i) space. This set is not a smooth manifold but has a non-differentiability at the origin, i.e. if $h_i(x^*) = 0$ and also $\mu_i^* = 0$. This case is called a weakly active constraint. Often we want to exclude this case. On the other hand, an active constraint with $\mu_i^* > 0$ is called strictly active.

Definition 14. *Regard a KKT point (x^*, λ^*, μ^*). We say that strict complementarity holds at this KKT point iff all active constraints are strictly active.*

Strict complementarity is a favourable condition because it also makes many theorems easier to formulate and to prove, and is also required to prove convergence of some numerical methods.

3.3.2 Second Order Optimality Conditions

If LICQ holds and the KKT conditions are fulfilled at a point x^*, it can still be a point which is not a local minimizer. For checking if the candidate x^* is indeed a local minimizer, the SOSC can be examined which, together with LICQ and the KKT conditions, is a sufficient condition for optimality.



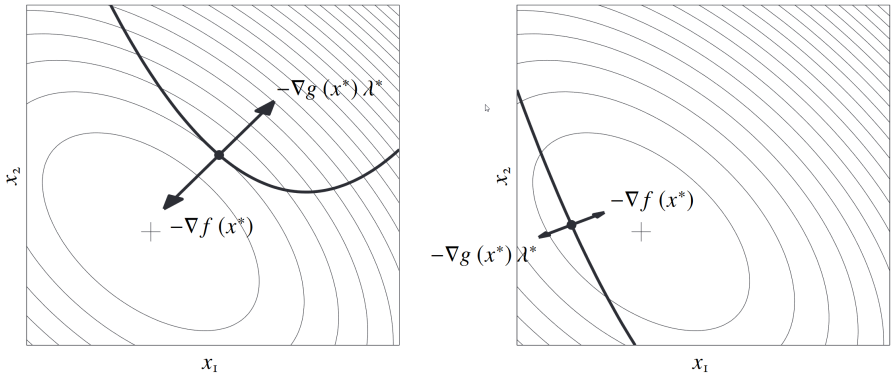


Figure 3.9: Illustration of the KKT conditions for an equality-constrained NLP. The contour lines (or level sets) of the objective function f and the constraint $g(x) = 0$ are shown. The negative gradient of the objective function $-\nabla f(x)$ points into the direction of the minimizer x^*. At x^* the sum of the gradients of the cost function and of the constraints (weighted with the Lagrange multiplier) equal to 0.

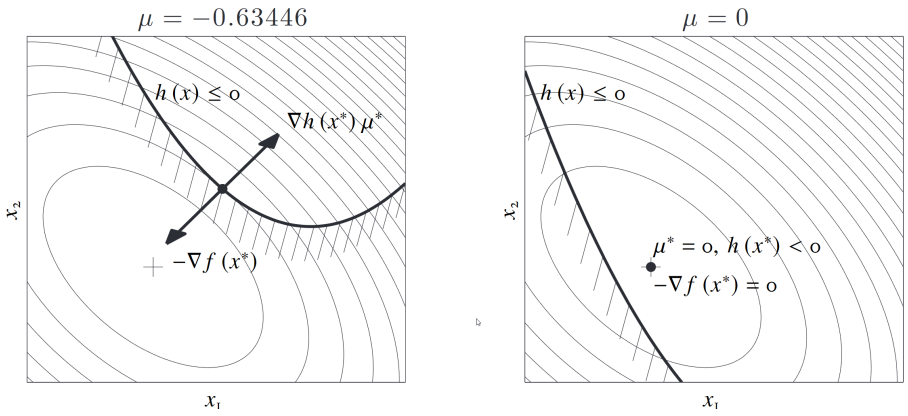


Figure 3.10: Illustration of the KKT conditions for an inequality-constrained NLP. The "slope" of the cost function $-\nabla f\,(x)$ pushes the solution towards its lowest point. The solution contained by the "barrier", i.e. the inequality constraints $h\,(x) \leq 0$ to remain within the feasible domain via the force $-\nabla h\,(x)\,\mu$, but is free to move along the barrier and towards the interior of the feasible domain. At the solution x^*, μ^*, the forces exerted by the barrier and the cost function even out. If the solution is in contact with the barrier, then the force is non-zero and pushes towards the interior of the feasible domain, i.e. $h(x^*) = 0$, $\mu > 0$ (left graph). Otherwise, the barrier exerts no force on the solution, i.e. $h(x^*) < 0$, $\mu = 0$ (right graph).

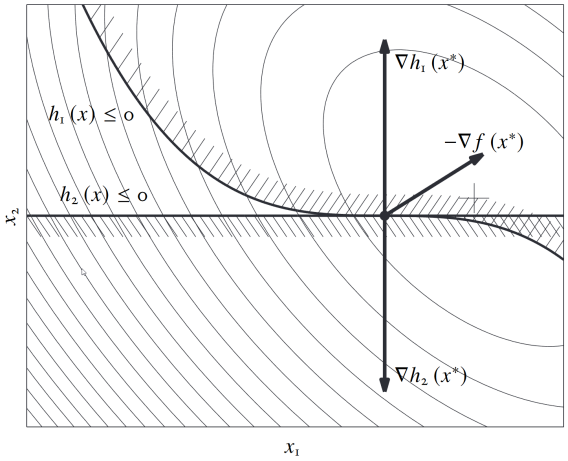


Figure 3.11: Failure of the LICQ condition. The optimal solution is not a KKT point. In this case, the forces exerted by the constraints $h_1(x)$ and $h_2(x)$ are collinear, and cannot balance the slope of the cost function $-\nabla f(x)$, even though the constraints prevent the solution from moving further toward the minimum of the cost function.

In case of strict complementarity at a KKT point (x^*, λ^*, μ^*), the optimization problem can locally be regarded to be a problem with equality constraints only, namely those within the function $\tilde{g}$ defined in Equation (3.21). Though more complex second order conditions can be formulated that are applicable even when strict complementarity does not hold, we restrict ourselves here to this special case. Second order conditions for optimality check whether the curvature of $\mathcal{L}$ is positive in all feasible directions.

Theorem 5 (Second Order Optimality Conditions)**.** *Let us regard a point x^* at which LICQ holds together with multipliers λ^*, μ^* so that the KKT conditions* (3.22a)-(3.22e) *are satisfied and let strict complementarity hold. The Second Order Sufficient Condition (SOSC) checks if the curvature is positive in all feasible directions s, i.e., if*

$$s^\top \nabla^2_{xx} \mathcal{L}\left(x^*, \lambda^*, \mu^*\right) s > 0 \tag{3.24}$$

holds for all feasible directions s such that

$$\nabla \tilde{g}_i^\top \left(x^*\right) \cdot s = 0, \forall i \in i = 1, \ldots, n_g \cup \mathcal{A}\left(x^*\right). \tag{3.25}$$

The feasible directions s are given by the dot product in (3.25), i.e., the constraint tangent space (assuming strict complementarity). The second order conditions can equally be defined if strict complementarity does not hold. If a convex optimization problem is given, the FONC and the fulfillment of LICQ are necessary and sufficient conditions, such that SOSC does not need to be checked.
The matrix $\nabla^2_x \mathcal{L}(x^*, \lambda^*, \mu^*)$ plays an important role in optimization algorithms and is called the *Hessian of the Lagrangian*.

Example (Quadratic Problems with Equality Constraints)**.** To illustrate the above optimality conditions, let us regard a QP with equality constraints only.

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad c^T x + \frac{1}{2} x^T B x \tag{3.26a}$$

$$\text{subject to} \quad Ax + b = 0. \tag{3.26b}$$

We assume that A has full row rank i.e., LICQ holds. The Lagrangian is $\mathcal{L}(x, \lambda) = c^T x + \frac{1}{2} x^T B x + \lambda^T (Ax + b)$ and the KKT conditions have the explicit form

$$c + Bx + A^T \lambda = 0 \tag{3.27a}$$

$$b + Ax = 0. \tag{3.27b}$$

This is a linear equation system in the variable (x, λ) and can be solved if the so called *KKT matrix*

$$\begin{bmatrix} B & A^T \\ A & 0 \end{bmatrix}$$

is invertible.

3.4 Optimization Algorithms

3.4.1 Newton-Type methods for Equality Constrained Optimization

Let us first regard an optimization problem with only equality constraints,

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && \text{(3.28a)} \\ \text{subject to} \quad & g(x) = 0 && \text{(3.28b)} \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^{n_g}$ are both two times continuously differentiable functions. The idea of the Newton-type optimization methods is to apply a variant of Newton's method to solve the nonlinear KKT conditions

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda) &= 0 && \text{(3.29a)} \\ g(x) &= 0 && \text{(3.29b)} \end{aligned}$$

In order to simplify notation, we define

$$w := \begin{bmatrix} x \\ \lambda \end{bmatrix} \text{ and } F(w) := \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ g(x) \end{bmatrix} \tag{3.30}$$

with $w \in \mathbb{R}^{n+n_g}$, $F : \mathbb{R}^{n+n_g} \to \mathbb{R}^{n+n_g}$, so that we can compactly formulate the above nonlinear root finding problem as

$$F(w) = 0. \tag{3.31}$$

Starting from an initial guess w_0, Newton's method generates a sequence of iterates $\{w_k\}_{k=0}^{\infty}$ by linearizing the nonlinear equation at the current iterate and setting it to zero

$$F(w_k) + \frac{\partial F}{\partial w}(w_k)(w_{k+1} - w_k) = 0 \tag{3.32}$$

and obtaining the next iterate as its solution, i.e.

$$w_{k+1} = w_k - \frac{\partial F}{\partial w}(w_k)^{-1} F(w_k) \tag{3.33}$$

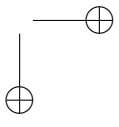
For equality constrained optimization, the linear system (3.32) has the specific form[1]

$$\begin{bmatrix} \nabla_x \mathcal{L}(x_k, \lambda_k) \\ g(x_k) \end{bmatrix} + \underbrace{\begin{bmatrix} \nabla_x^2 \mathcal{L}(x_k, \lambda_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix}}_{\text{KKT-matrix}} \begin{bmatrix} x_{k+1} - x_k \\ \lambda_{k+1} - \lambda_k \end{bmatrix} = 0 \tag{3.34}$$

Using the definition

$$\nabla_x \mathcal{L}(x_k, \lambda_k) = \nabla f(x_k) + \nabla g(x_k) \lambda_k \tag{3.35}$$

[1] In this script we use the convention $\nabla g(x) := \frac{\partial g}{\partial x}(x)^T$ that is consistent with the definition of the gradient $\nabla f(x)$ of a scalar function f being a column vector.



we see that the contributions depending on the old multiplier λ_k cancel each other, so that the above system is equivalent to

$$\begin{bmatrix} \nabla f(x_k) \\ g(x_k) \end{bmatrix} + \begin{bmatrix} \nabla_x^2 \mathcal{L}(x_k, \lambda_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} x - x_k \\ \lambda \end{bmatrix} = 0. \tag{3.36}$$

This formulation shows that the data of the linear system only depend on λ_k via the Hessian matrix. We need not use the exact Hessian matrix, but can approximate it with different methods. This leads to the more general class of Newton-type optimization methods. Using any such approximation $B_k \approx \nabla_x^2 \mathcal{L}(x_k, \lambda_k)$, we finally obtain the Newton-type iteration as

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ 0 \end{bmatrix} - \begin{bmatrix} B_k & \nabla g(x_k) \\ \nabla g^T(x_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla f(x_k) \\ g(x_k) \end{bmatrix} \tag{3.37}$$

If we use $B_k = \nabla_x^2 \mathcal{L}(x_k, \lambda_k)$, we recover the *exact Newton method*. So-called *Newton-type methods* choose different Hessian approximations B_k that may be easier to compute. The general *Newton-type method* is summarized in Algorithm 1.

Algorithm 1 Equality constrained full step Newton-type method

Choose: initial guesses x_0, λ_0, and a tolerance ϵ
Set: $k = 0$

while $\|\nabla \mathcal{L}(x_k, \lambda_k)\| \geq \epsilon$ or $\|g(x_k)\| \geq \epsilon$ **do**
  obtain a Hessian approximation B_k
  get x_{k+1}, λ_{k+1} from (3.37)
  $k = k + 1$
end while

In the exact Newton method, we set

$$B_k := \nabla_x^2 \mathcal{L}(x_k, \lambda_k)$$

But how can this matrix be computed? Many different ways for computing this second derivative exist. The most straightforward way is a finite difference approximation where we perturb the evaluation of $\nabla \mathcal{L}$ in the direction of all unit vectors $\{e_i\}_{i=1}^n$ by a small quantity $\delta > 0$. This yields each time one column of the Hessian matrix, as

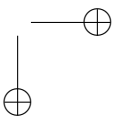
$$\nabla_x^2 \mathcal{L}(x_k, \lambda_k) e_i = \frac{\nabla_x \mathcal{L}(x_k + \delta e_i, \lambda_k) - \nabla_x \mathcal{L}(x_k, \lambda_k)}{\delta} + O(\delta) \tag{3.38}$$

Unfortunately, the evaluation of the numerator of this quotient suffers from numerical cancellation, so that δ cannot be chosen arbitrarily small, and the maximum attainable accuracy for the derivative is $\sqrt{\epsilon}$ if ϵ is the accuracy with which the gradient $\nabla_x \mathcal{L}$ can be obtained. Thus, we loose half the valid digits. If $\nabla_x \mathcal{L}$ was itself already approximated by finite differences, this means that we have lost three quarters of the originally valid digits. More accurate and also faster ways to obtain derivatives of arbitrary order will be presented in the chapter on algorithmic differentiation.

Local convergence rate: The exact Newton method has a *quadratic convergence rate*, i.e. $\|w_{k+1} - w^*\| \leq c\|w_k - w^*\|^2$. This means that the number of accurate digits doubles in each iteration. As a rule of thumb, once a Newton method is in its area of quadratic convergence, it needs at maximum 6 iterations to reach the highest possible precision.

3.4.2 Interior Point Methods for Inequality Constrained Optimization*

When a nonlinear optimization problem with inequality constraints shall be solved, two big families of methods exist, first, nonlinear interior point (IP), and second, sequential quadratic programming (SQP) methods. Both aim at solving the KKT conditions (3.22) which include the non-smooth complementarity



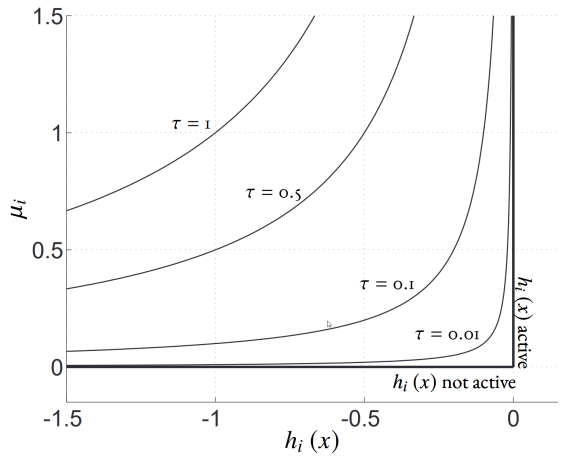


Figure 3.12: Relaxation of the complementarity slackness condition. We display here the manifold $\mu_i h_i(x) + \tau = 0$ for various values of τ. The original non-smooth manifold $\mu_i h_i(x) = 0$ arising in the KKT conditions is displayed as the thick lines.

conditions, but have different ways to deal with this non-smoothness. We will only cover the interior point method.

The basic idea of an interior point method is to replace the non-smooth L-shaped set resulting from the complementarity conditions with a smooth approximation, typically a hyberbola. Thus, a smoothing constant $\tau > 0$ is introduced and the KKT conditions are replaced by the smooth equation system

$$\begin{aligned} \nabla f(x^*) + \nabla g(x^*)\lambda^* + \nabla h(x^*)\mu^* &= 0 && \text{(3.39a)} \\ g(x^*) &= 0 && \text{(3.39b)} \\ \mu_i^* h_i(x^*) + \tau &= 0, \quad i = 1, \ldots, n_h. && \text{(3.39c)} \end{aligned}$$

Note that the last equation ensures that $-h_i(x^*)$ and μ_i^* are both strictly positive and on a hyperbola.[2] For τ very small, the L-shaped set is very closely approximated by the hyperbola, but the nonlinearity is increased. Within an interior point method, we usually start with a large value of τ and solve the resulting nonlinear equation system by a Newton method, and then iteratively decrease τ, always using the previously obtained solution as initialization for the next one.

One way to interpret the above smoothened KKT-conditions is to use the last condition to eliminate $\mu_i^* = -\frac{\tau}{h_i(x^*)}$ and to insert this expression into the first equation, and to note that $\nabla_x \left(\log(-h_i(x))\right) = \frac{1}{h_i(x)} \nabla h_i(x)$). Thus, the above smooth form of the KKT conditions is nothing else than the optimality conditions of a *barrier problem*

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) - \tau \sum_{i=1}^{n_h} \log\left(-h_i(x)\right) && \text{(3.40a)} \\ \text{subject to} \quad & g(x) = 0. && \text{(3.40b)} \end{aligned}$$

Note that the objective function of this problem tends to infinity when $h_i(x) \to 0$. Thus, even for very small $\tau > 0$, the barrier term in the objective function will prevent the inequalities to be violated. The *primal barrier method* just solves the above barrier problem with a Newton-type optimization method for equality constrained optimization for each value of τ. Though easy to implement and to interpret, it is not necessarily the best for numerical treatment, among other because its KKT matrices become very ill-conditioned for small τ. This is not the case for the *primal-dual IP method* that solves the full nonlinear equation system (3.39) including the dual variables μ.

[2] In the numerical solution algorithms for this system, we have to ensure that the iterates do not jump to a second hyperbola of infeasible shadow solutions, by shortening steps if necessary to keep the iterates in the correct quadrant.

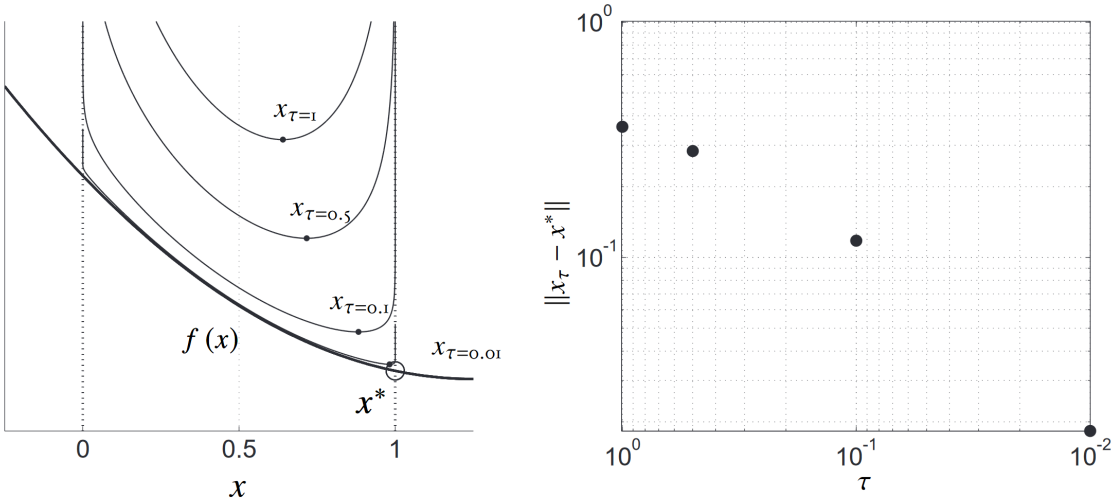


Figure 3.13: Illustration of the primal barrier method presented in (3.40). The left graph displays an illustrative cost function $f(x)$ (thick curve), and simple bounds $0 \leq x \leq 1$. The various objective functions with barrier $f(x) - \tau \sum_{i=1}^{n_h} \log(-h_i(x))$ are displayed for various values of τ, alongside their respective minima x_τ. The right graph displays the error between the actual solution to the problem x^*, and the solutions x_τ obtained from the barrier problem (3.40) for various values of τ.

For convex problems, very strong complexity results exist that are based on *self-concordance* of the barrier functions and give upper bounds on the total number of Newton iterations that are needed in order to obtain a numerical approximation of the global solution with a given precision. When an IP method is applied to a general NLP that might be non-convex, we can of course only expect to find a local solution, but convergence to KKT points can still be proven, and these *nonlinear IP methods* perform very well in practice.

Software: A very widespread and successful implementation of the nonlinear IP method is the open-source code IPOPT [12, 11]. Though IPOPT can be applied to convex problems and will yield the global solution, dedicated IP methods for different classes of convex optimization problems can exploit more problem structure and will solve these problems faster and more reliably. Most commercial LP and QP solution packages such as CPLEX or MOSEK make use of IP methods, as well as many open-source implementations such as the sparsity exploiting QP solver OOQP.

3.4.3 Generating derivatives*

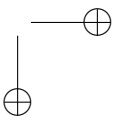
"Progress is measured by the degree of differentiation within a society."
Herbert Read

Derivatives of computer coded functions are needed everywhere in optimization. In order to just check optimality of a point, we need already to compute the gradient of the Lagrangian function. Within Newton-type optimization methods, we need the full Jacobian of the constraint functions. If we want to use an exact Hessian method, we even need second order derivatives of the Lagrangian.
There are many ways to compute derivatives: Doing it by hand is error prone and nearly impossible for longer evaluation codes. Computer algebra packages like Mathematica or Maple can help us, but require that the function is formulated in their specific language. More annoyingly, the resulting derivative code can become extremely long and slow to evaluate.
On the other hand, *finite differences* can always be applied, even if the functions are only available as black-box codes. They are easy to implement and relatively fast, but they necessarily lead to a loss of precision of half the valid digits, as they have to balance the numerical errors that originate from Taylor

series truncation and from finite precision arithmetic. Second derivatives obtained by recursive application of finite differences are even more inaccurate. The best perturbation sizes are difficult to find in practice. Note that the computational cost to compute the gradient $\nabla f(x)$ of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ is $(n+1)$ times the cost of one function evaluation.

A more efficient and more accurate way to evaluate the gradient of a scalar function is *algorithmic (or automatic) differentiation* (AD). It requires in principle nothing more than that the function is available in the form of source code in a standard programming language such as Python, C, C++ or FORTRAN.

Algorithmic differentiation uses the fact that each differentiable function is composed of several elementary operations, like multiplication, division, addition, subtraction, sine-functions, exp-functions, etc., whose derivatives we know. This is exploited by AD to calculate first and higher derivatives of functions efficiently and precisely. For example, we can calculate the gradient of f at only three times the cost of evaluating f itself, $\mathrm{cost}(\nabla f) \approx 3\,\mathrm{cost}(f)$. Many AD tools exist, e.g. CasADi [1].



Chapter 4

Linear Model Predictive Control

This chapter introduces the important field of linear model predictive control (LMPC), i.e., model predictive control (MPC) for linear systems. LMPC is widely used in industry because of the relatively low associated computational effort and since it typically results in convex optimization problems which can be reliably solved. In this chapter, we will discusses the general idea of MPC-based control and the discretization of continuous-time linear systems, before moving on to unconstrained and constrained LMPC formulations and solution strategies.

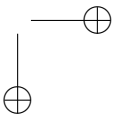
4.1 MPC control idea

Classical control loop Let us first consider the classical control structure for a discrete-time system given in Fig. 4.1. Classical control is typically based on two basic elements: (1) a measurement $\hat{y}_k$ of the controllable output y_k and (2) a fixed linear controller design C. The working principle of the control loop at time step k can then be summarized by the following steps:

1. Obtain output measurement $\hat{y}_k$.
2. Compute error w.r.t. reference signal: $e_k = r_k - \hat{y}_k$.
3. Compute control input u_k as a function of (past and present values of) e_k.
4. Apply u_k and repeat at time $k + 1$.

This control paradigm is very successful because of its simplicity (model-free, low computational complexity) and its effectiveness for disturbance rejection. However, the classical control loop concept is a powerful technique mainly for linear, single-input, single-output (SISO) systems without constraints. As soon as any of these aspects (nonlinearity, multiple inputs, constraints) needs to be considered, the control loop becomes quite complex and unmanageable very quickly.
For example, most actuator systems have physical limitations that are relevant during operation. These limitations need to be accounted for in the control loop, which is typically solved using ad-hoc solutions such as e.g. anti-windup. Also, in order to avoid these limitations, systems are often operated at a setpoint that is far away from constraints, even though this leads to suboptimal operation.

MPC control loop MPC addresses precisely this issue. By reformulating the control problem as an optimization problem, it is inherently capable of dealing with multiple-input, multiple-output (MIMO) systems, nonlinearity, and different sorts of constraints. Moreover, it allows one to directly optimize a certain performance index, as opposed to the indirect way of controller parameter-tuning. And finally, if future knowledge on the reference signal or on incoming disturbances is available, it can be taken directly into account to create anticipatory behavior.



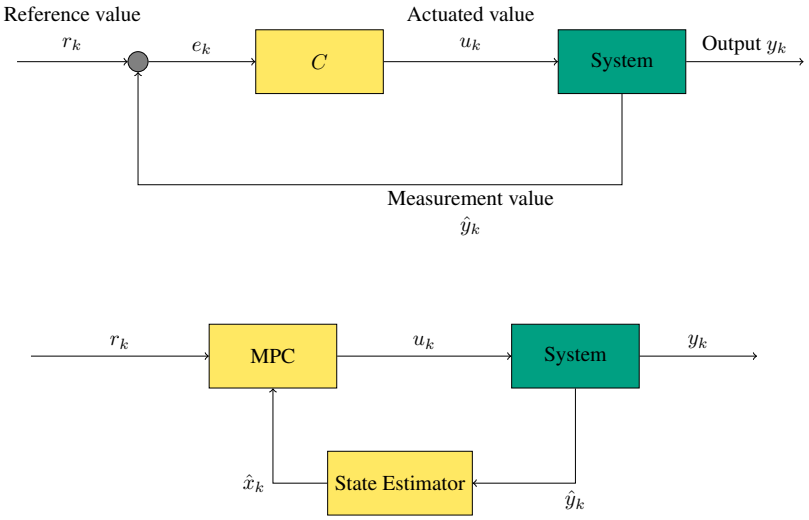


Figure 4.1: Classical (top) and MPC (bottom) control loop structures.

Fig. 4.1 shows the MPC control loop. Let us first consider what happens in the MPC controller block. In this block, the following optimal control problem is solved.

$$\begin{aligned} &\underset{\substack{\textit{control inputs}\\ \textit{state prediction}}}{\text{minimize}} && \textit{total cost over time horizon} \\ &\text{subject to} && \textit{start at current state}, \\ &&& \textit{state space model}, \\ &&& \textit{constraints} \end{aligned} \tag{4.1}$$

An optimal control problem is a "dynamic" optimization problem, i.e. an optimization problem that optimizes a state trajectory which is constrained by some system dynamics. In this case the optimization variables are the control and state trajectories over a certain *prediction* horizon. The state trajectory is constrained twofold: it must start at the current state of system, and it must satisfy the dynamics described by a given state space model. In this sense, the "real" degrees of freedom for the optimizer are only the control inputs. The objective of the optimization problem is a function of the state and control trajectory and is a quantification of the overall control objective, such as e.g. the deviation from a given setpoint r_k.
All physical system have constraints. Typical constraints are: physical constraints (e.g. actuator limits), safety constraints (e.g. temperature/pressure limits) and performance constraints (e.g. maximal overshoot). Such constraints can be elegantly integrated in the optimal control problem as optimization constraints.
This powerful formulation however comes with a complication: in order to compute a meaningful state prediction, the current system state needs to be known. In some cases, all of the system states can be directly measured. If this is not the case, an additional component needs to be added to the control loop: a *state estimator*. The state estimator also uses an internal state space model to compute an estimate of the state $\hat{x}_k$ based on the applied controls u_k and measurement values $\hat{y}_k$. This state estimate is then fed to the MPC block. In order to obtain good closed-loop performance, the estimator dynamics should be significantly faster than the controlled system dynamics. A highly popular estimator in many applications is the Kalman Filter, which we will not discuss further in this lecture. For now, we will assume that the state can be fully measured.
The control loop creates a feedback action via the so-called *receding horizon* strategy, illustrated by Fig. 4.2:

1. Estimate current state $\hat{x}_k$.
2. Solve optimal control problem (4.1).
3. Apply only the very first part of the computed optimal control trajectory.

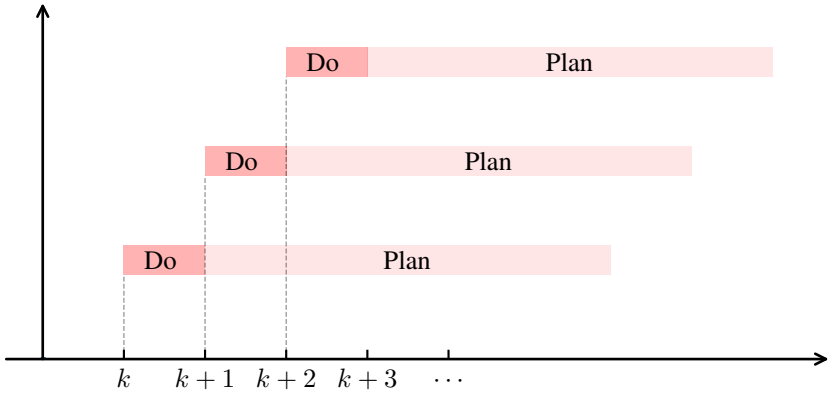


Figure 4.2: Receding horizon strategy introduces feedback in an MPC-based control loop.

4. Wait until new measurement becomes available at time $k+1$ and repeat.

Thus, to summarize, the MPC controller comes with many advantages, but, in comparison with the classical control loop, needs three additional elements: (1) an accurate state space model, (2) a state estimator and (3) an efficient optimization solver that solves the optimal control problem in real-time.

4.2 Discrete-time linear state space models

Linear MPC is a variant of MPC which only uses linear models in the optimal control problem. In this chapter, we will consider only linear time-invariant (LTI) systems. These systems can be described by the continuous-time state space model

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{4.2}$$
$$y(t) = C_c x(t) + D_c u(t) \tag{4.3}$$

with fixed matrices $A_\mathrm{c} \in \mathbb{R}^{n_x \times n_x}$ and $B_\mathrm{c} \in \mathbb{R}^{n_x \times n_u}$. The subscript $_c$ denotes "continuous-time".
However, in order to obtain a numerically tractable optimal control problem, it is required that the state trajectory is given in discrete form. Therefore, we first show how continuous-time LTI systems can be transformed into discrete-time LTI systems.

4.2.1 Discretization of LTI state-space models

A state space representation in discrete time is derived from a state space representation in continuous time by means of simulation (that is, integration over time). However, in case of LTI systems, it is even possible to derive an analytic expression for the state space system in discrete time, without having to resort to numerical integration.
First, we define a discrete time grid $t_0 < t_1 < \ldots < t_k < \ldots < t_N$ by choosing a horizon length N and constant sampling time T_s. The time grid points are then given by $t_k = kT_\mathrm{s}$ with an integer $k \in \mathbb{Z}$.
Second, the control input trajectory $u(t)$ must be re-written as function of a finite number of variables. The most common parameterization for MPC is "zero-order hold" (ZOH), where the control input is piecewise constant over the sampling intervals:

$$u(t) = \begin{cases} u_0 & \text{if } t \in [0, T_\mathrm{s}) \\ \vdots & \vdots \\ u_k & \text{if } t \in [kT_\mathrm{s}, (k+1)T_\mathrm{s}) \\ \vdots & \vdots \\ u_{N-1} & \text{if } t \in [(N-1)T_\mathrm{s}, NT_\mathrm{s}) \end{cases}, \tag{4.4}$$

so that the control trajectory is defined by the parameters $u_0, u_1, \ldots, u_{N-1} \in \mathbb{R}^{n_u}$.
Consider now the continuous-time LTI system given by (4.3). Thanks to the time-invariance property, the system representation is the same at all times. Therefore can we can consider the state evolution for arbitrary k at points (x_k, t_k) and (x_{k+1}, t_{k+1}) to obtain

$$\begin{aligned} x_{k+1} = x((k+1)T_\mathrm{s}) &= e^{A_c(t_{k+1}-t_k)}x_k + \int_{t_k}^{t_{k+1}} e^{A_c(t_{k+1}-\tau)}B_c u(\tau)\mathrm{d}\tau \\ &= e^{A_c T_\mathrm{s}}x_k + e^{A_c T_\mathrm{s}}\int_0^{T_\mathrm{s}} e^{-A_c t}\mathrm{d}t B_c u_k = Ax_k + Bu_k \ , \end{aligned} \tag{4.5}$$

where the fact that $u(t)$ is piecewise constant in between sampling instants has been exploited to move u_k outside the integral, and the change of variable $t = \tau - t_k$ is performed in the integrator. The discrete output equation (4.7) is simply obtained by evaluating equation (4.3) at the time $t = k\mathrm{T}_s$.
In summary, the state space representation of the discretized LTI system is given by

$$x_{k+1} = Ax_k + Bu_k \ , \tag{4.6}$$
$$y_k = Cx_k + Du_k \ , \tag{4.7}$$

with the constant matrices

$$A = e^{A_c \mathrm{T}_\mathrm{s}} \ , \tag{4.8}$$
$$B = e^{A_c T_\mathrm{s}}\int_0^{T_\mathrm{s}} e^{-A_c t}\mathrm{d}t B_c \ , \tag{4.9}$$
$$C = C_c \ , \tag{4.10}$$
$$D = D_c \ . \tag{4.11}$$

4.2.2 Solution of the state space ODE

The *homogeneous response* with zero input and initial state x_0 can be found by successive substitutions

$$\begin{aligned} x_1 &= Ax_0 \\ x_2 &= Ax_1 = A^2 x_0 \\ \cdots &= \cdots \\ x_k &= Ax_{k-1} = A^k x_0 \end{aligned}$$

Note that it is computed using only the matrix A.
The *forced response* with generic non-zero input is computed by induction. The expression for two consecutive substitutions

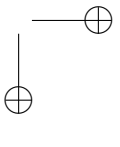
$$\begin{aligned} x_{k+2} &= Ax_{k+1} + Bu_{k+1} \\ &= A(Ax_k + Bu_k) + Bu_{k+1} \\ &= A^2 x_k + ABu_k + Bu_{k+1} \end{aligned}$$

can be generalized as

$$x_k = A^k x_0 + \sum_{m=0}^{k-1} A^{k-m-1}Bu_m \tag{4.12}$$

for $k \leq 0$.
The *system output response* is computed by substitution of (4.12) into the equation $y_k = Cx_k + Du_k$, obtaining

$$y_k = CA^k x_0 + \sum_{m=0}^{k-1} CA^{k-m-1}Bu_m + Du_k \tag{4.13}$$



4.2.3 Controllability and observability

Before we can proceed to a first MPC formulation, we need to introduce the important notions of *controllability* and *observability*. As we will see, we can only apply MPC succesfully to systems that are both controllable and observable.
A discrete system is controllable if in a finite number of time steps $n \geq n_x$, any initial state $x_0 \in \mathbb{R}^{n_x}$ can be driven to any given final state $x_n \in \mathbb{R}^{n_x}$ by an appropriate choice of control inputs $u_0, u_1, \ldots, u_{n-1}$. A convenient property of LTI systems is that its controllability can be checked by investigating the constant state space matrices A and B. For this, we construct the controllability matrix

$$S_{\mathrm{C}} = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \in \mathbb{R}^{n_x \times (n \cdot n_u)} \,. \tag{4.14}$$

Then, the LTI system is *controllable* if S_{C} is non-singular, i.e.:

$$\mathrm{rank}(S_{\mathrm{C}}) = n_x \Leftrightarrow \det(S_{\mathrm{C}} S_{\mathrm{C}}') \neq 0 \tag{4.15}$$

This can be proven by using the forced solution formula from (4.12):

$$\begin{aligned} x_n &= A^n x_0 + \sum_{m=0}^{n-1} A^{n-m-1} B u_m \\ x_n - A^n x_0 &= \sum_{m=0}^{n-1} A^{n-m-1} B u_m \\ &= S_{\mathrm{C}} \left[u_{n-1}^\top, u_{n-2}^\top, \ldots, u_0^\top \right]^\top \,. \end{aligned}$$

Thus, any state $x_n \in \mathbb{R}^{n_x}$ can be reached by an appropriate choice of controls, if the controllability matrix spans $\mathbb{R}^{n_x}$. Moreover, it can be shown that for any $n \geq n_x$ it holds

$$\mathrm{rank}(\begin{bmatrix} B & AB & \ldots & A^n B \end{bmatrix}) = \mathrm{rank}(\begin{bmatrix} B & AB & \ldots & A^{n_x} B \end{bmatrix})) \,, \tag{4.16}$$

which means that if any arbitrary state x_n can be reached in n steps, it can also be reached in n_x steps and vice versa. Thus, in order to check controllability, it suffices to check the rank of S_{C} for $n = n_x$.

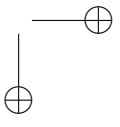
Example. (Double integrator) We consider the system $\ddot{y}(t) = u(t)$, whose state space representation is given by the matrices $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$, $D = 0$. We are only interested in the pair (A, B) and want to investigate if the system is controllable. Therefore we construct the matrix $S_{\mathrm{C}} = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, and we test if it has rank $n_x = 2$. This is the case, as $\det(S_{\mathrm{C}}) = -1 \neq 0$.

Example. Let us now consider the system with state space matrices $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. If we now construct $S_{\mathrm{C}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, we see that $\det(S_{\mathrm{C}}) = 0$, and thus that the system is uncontrollable. More precisely, by observing S_{C} we directly see that the second row only has zeros, and thus that the second state cannot be influenced by the control inputs.

A system is *observable* if from a finite number of measurements $y_0, y_1, \ldots, y_{n-1}$, with $n > n_x$, the original state x_0 can be reconstructed. Similar to controllability this can be controlled by checking that the observability matrix

$$S_{\mathrm{O}} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \in \mathbb{R}^{(n \cdot n_y \times n_x)} \tag{4.17}$$

has full rank. The proof can be constructed using expression (4.13), by writing all measurements as a function of x_0 and then resolving for x_0. Similar to controllability, if observability is shown for $n = n_x$, the system is also observable for $n > n_x$ and vice versa.



4.3 Unconstrained linear MPC

For a given discrete-time LTI state space model that is controllable and observable, we now consider the special case of unconstrained linear MPC. In particular we consider here the *regulation* problem, where the control objective is to drive the states $x_k \to 0$ for $k \to \infty$. Most commonly, this translates into a quadratic penalty on the state deviation from the origin, resulting in the following OCP with horizon N:

$$\begin{aligned} \underset{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad & \frac{1}{2}x_N^\top P_N x_N + \frac{1}{2}\sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\ \text{subject to} \quad & x_0 = \hat{x}_0, \\ & x_{k+1} = Ax_k + Bu_k, \quad k = 0,\dots,N-1, \end{aligned} \tag{4.18}$$

with the state weight matrix $Q \in \mathbb{R}^{n_x \times n_x}$, control weight matrix $R \in \mathbb{R}^{n_u \times n_u}$ and terminal weight matrix $P_N \in \mathbb{R}^{n_x \times n_x}$. Note that in this formulation the index "0" of the state estimate $\hat{x}_0$ refers to the internal time grid of the MPC problem and that it is updated at every time step of the control feedback loop.

Cost function The regulation cost function is chosen to be a quadratic penalty on the state deviation from the origin. Additionally, the cost function penalizes the control effort. As a matter of fact, this control regularization is necessary for the resulting quadratic program (QP) to be convex and have a well-defined minimum. To be more precise, the weight matrices have to be chosen so that $Q, P_N \succeq 0$ and $R \succ 0$.
Note that it is sometimes instructive to also penalize the *rate of change* of controls, in order to protect the actuators from fatigue. This can be done adding the controls u to the state vector and by introducing a pseudo-control $\nu \in \mathbb{R}^{n_u}$ and the state equation $\dot{u} = v$. Now, the variables ν can be penalized in the cost function.

Analytic solution Because the linear regulation problem results in an equality-constrained QP, it admits to an analytic solution. In order to compute this solution, we first eliminate the states from the optimal control problem using the closed-form solution (4.12). The expressions for all states in the prediction horizon can be written in matrix form:

$$\overbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}}^{X} = \overbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}^{E} \hat{x}_0 + \overbrace{\begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix}}^{\Theta} \overbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}^{U}, \tag{4.19}$$

or in more compact notation:

$$X = E\hat{x}_0 + \Theta U \,. \tag{4.20}$$

We can also rewrite the cost function of problem (4.18) as

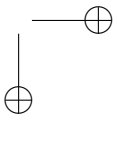
$$J = \frac{1}{2}(\hat{x}_0^\top Q \hat{x}_0 + X^\top \hat{Q} X + U^\top \hat{R} U) \tag{4.21}$$

with

$$\hat{Q} = \begin{bmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & P_N \end{bmatrix} \quad \text{and} \quad \hat{R} = \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{bmatrix}. \tag{4.22}$$

After substitution of X using (4.20), the cost function reads as

$$\begin{aligned} J &= \frac{1}{2}(\hat{x}_0^\top Q \hat{x}_0 + (E\hat{x}_0 + \Theta U)^\top \hat{Q}(E\hat{x}_0 + \Theta U) + U^\top \hat{R} U) && (4.23) \\ &= \frac{1}{2}\hat{x}_0^\top (Q + E^\top \hat{Q} E)\hat{x}_0 + \frac{1}{2}U^\top (\Theta^\top \hat{Q}\Theta + \hat{R})U + U^\top \Theta^\top \hat{Q} E \hat{x}_0 \,. && (4.24) \end{aligned}$$



The first term in this cost function is a constant and does not influence the optimal solution, hence it can be omitted. This results in the following unconstrained quadratic program

$$\underset{U}{\text{minimize}} \quad \frac{1}{2}U^\top(\Theta^\top\hat{Q}\Theta + \hat{R})U + U^\top\Theta\hat{Q}E\hat{x}_0 \,. \tag{4.25}$$

In order to find the optimal controls U^* we write down the KKT conditions for this QP, which comes down to setting the gradient of the cost function w.r.t. U to zero:

$$(\Theta^\top\hat{Q}\Theta + \hat{R})U + \Theta^\top\hat{Q}E\hat{x}_0 = 0 \,, \qquad \textit{(KKT conditions)} \tag{4.26}$$

resulting in the solution

$$U^* = -(\Theta^\top\hat{Q}\Theta + \hat{R})^{-1}\Theta^\top\hat{Q}E\hat{x}_0 \,. \tag{4.27}$$

Within the MPC feedback loop, we are only interested in the first control input u_0^*:

$$u_0^* = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} U^* = K\hat{x}_0 \,, \tag{4.28}$$

with the constant feedback matrix

$$K = -\begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} (\Theta^\top\hat{Q}\Theta + \hat{R})^{-1}\Theta^\top\hat{Q}E \,. \tag{4.29}$$

Interestingly, the optimal linear-quadratic regulator is a constant linear feedback controller. This is convenient since the optimal feedback matrix K can be computed offline and online computational effort is limited to a matrix-vector product.
Finally, we can check if the optimal solution U^* is a unique minimizer. This is the case if the second-order sufficient conditions are fulfilled, which is the case if the quadratic cost is strictly convex, i.e. when the Hessian of the cost function is strictly positive definite:

$$\nabla^2 J = \Theta^\top\hat{Q}\Theta + \hat{R} \succ 0 \,. \tag{4.30}$$

This condition is met when $Q, P_N \succeq 0$ and $R \succ 0$. Thus, by an appropriate choice of weights, we can guarantee convexity of the resulting optimization problem.

Finite horizon LQR The solution to the optimal control problem (4.18) is called the finite-horizon linear-quadratic regulator (LQR). Let us now investigate the behavior of this controller by considering the following example.

Example. Consider the slightly damped but stable, discrete-time system with state space model

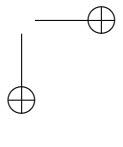
$$x_{k+1} = \begin{bmatrix} 1.988 & -0.998 \\ 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0.125 \\ 0 \end{bmatrix} u_k \,. \tag{4.31}$$

with $x \in \mathbb{R}^2$ and $u \in \mathbb{R}$. We now conduct a closed-loop experiment for two different finite-horizon LQR controllers, with horizon $N = 3$ and $N = 100$ respectively, but with identical weight matrices $Q = P_N = I$ and $R = 1$. The initial state is $x_0 = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top$.
Fig. 4.3 shows the evolution of the closed-loop system and the applied controls as well as the open-loop predictions for three consecutive time steps. While for the case of the very large horizon $N = 100$, the closed-loop response and open-loop predictions coincide, this is not the case for the case $N = 3$. This mismatch is a crucial property of finite-horizon MPC since it can cause the closed-loop to become unstable. Note that this mismatch occurs in the absence of model-plant mismatch, and is purely an effect of the finite horizon.
Fig. 4.4 shows the closed-loop response as well as the stage cost

$$J_k = x_k^\top Q x_k + u_k^\top R u_k \tag{4.32}$$

for both LQR controllers for a longer time sequence. The shorter horizon results in a larger overshoot and longer settling time. The longer horizon has a higher stage cost at the beginning of the trajectory, because it allows for a lower stage cost later. The shorter horizon LQR has a lower stage cost in the beginning, because it does not take into account that this leads to a higher cost later. Overall, the closed-loop cost of the shorter horizon LQR is 60% higher than that of the longer horizon LQR.

Thus, the finite horizon not only leads to a mismatch between prediction and closed-loop response, it also leads to suboptimal behavior compared to an LQR controller with an (in the limit) infinite horizon.



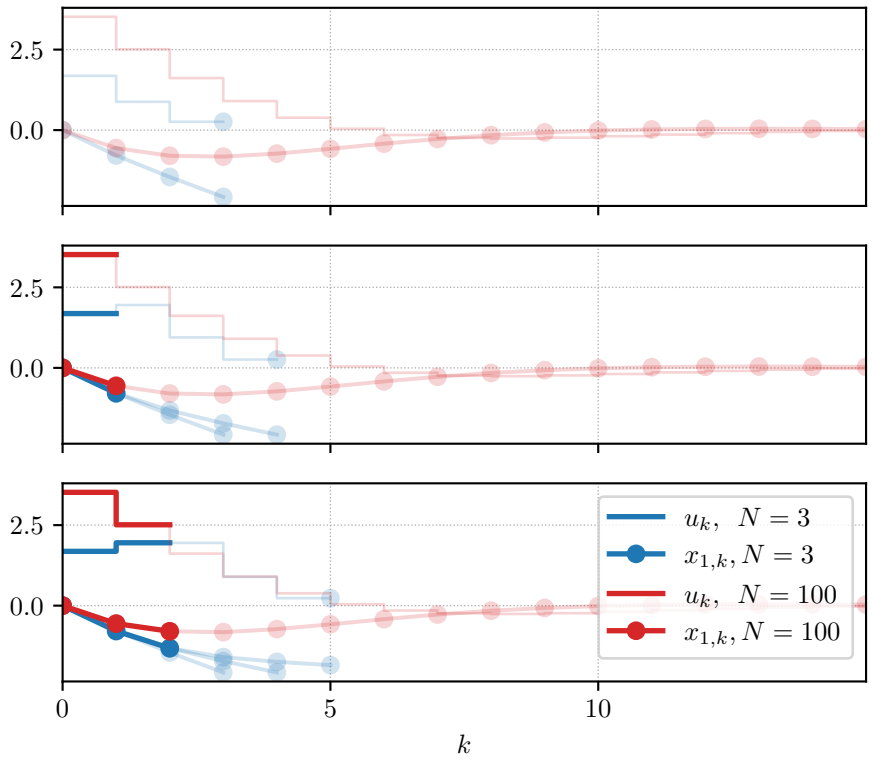


Figure 4.3: Closed-loop response and open-loop predictions of the finite-horizon LQR problem for different horizon lengths N.

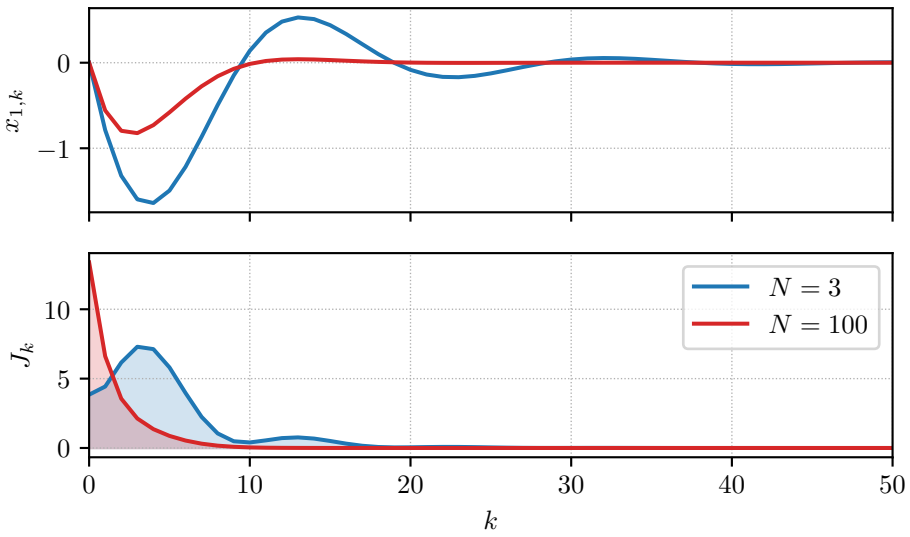


Figure 4.4: Closed-loop response (top) and closed-loop stage cost J_k (bottom) of the finite-horizon LQR problem for different horizon lengths N.

Infinite horizon LQR From the previous considerations, we learned that the best possible LQR controller is the one with an infinite horizon, i.e. the one that solves the following optimization problem:

$$\begin{aligned} \underset{\substack{x_0,x_1,\dots \\ u_0,u_1,\dots}}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{\infty} x_k^\top Q x_k + u_k^\top R u_k \\ \text{subject to} \quad & x_0 = \hat{x}_0, \\ & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, \infty \, . \end{aligned} \tag{4.33}$$

This optimization problem has infinitely many variables and constraints and does not allow for an explicit analytic solution.
However, it can be shown that the solution results in the linear feedback law

$$u_0^* = K_\infty \hat{x}_0 \, , \tag{4.34}$$

where the feedback matrix K_∞ can be computed numerically via

$$K_\infty = -(B^\top P_\infty B + R)^{-1} B^\top P_\infty A \, , \tag{4.35}$$

and where the infinite horizon cost matrix P_∞ is found by solving the discrete algebraic Ricatti equation (DARE)

$$P_\infty = A^\top P_\infty A - (A^\top P_\infty B)(R + B^\top P_\infty B)^{-1}(B^\top P_\infty A) + Q \, . \tag{4.36}$$

This cost matrix P_∞ determines the infinite-horizon cost-to-go J_∞. This is the cost of optimally driving the states from initial state $\hat{x}_0$ to zero in an infinite amount of steps, i.e.:

$$J_\infty(\hat{x}_0) = \hat{x}_0^\top P_\infty \hat{x}_0 \, . \tag{4.37}$$

Under some technical conditions which are typically fulfilled in practice for controllable systems, the DARE (4.36) has a unique positive definite solution P_∞ which can be found by performing the so-called *Ricatti recursion*.
The Ricatti recursion is performed by initializing P_∞ with the weight matrix Q and then updating P_∞ with the right hand side of the DARE (4.36) until the recursion converges to a steady-state value of P_∞. If the aforementioned technical conditions are fulfilled, the Ricatti recursion is guaranteed to converge.
Moreover, under the same conditions, the resulting closed-loop system

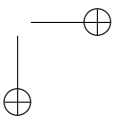
$$x_{k+1} = Ax_k + B(K_\infty x_k) = (A + BK_\infty)x_k \tag{4.38}$$

is guaranteed to be asymptotically stable.
Since the feedback matrix K_∞ can be computed efficiently offline, there are no benefits to applying finite-horizon LQR in real-world applications. The infinite-horizon LQR is always the better choice due to its optimal performance and stability guarantees. Hence, in practice, "the LQR ontroller" is a synonym for the infinite-horizon LQR version.
On a final note, the finite-horizon LQR controller with horizon length N can be made equivalent to the infinite horizon LQR controller by choosing as the terminal weight matrix $P_N = P_\infty$. In this case, the terminal cost takes into account the remaining cost (outside of the prediction horizon) to drive the final state x_N to zero in infinite time exactly.

4.4 Constrained linear MPC

Thus far, we have considered unconstrained linear MPC, with which we can tackle the regulating problem of designing a controller for a linear MIMO system, which:

1. Optimizes "performance"
2. Asymptotically stabilizes the system, i.e. $\lim_{k\to\infty} x_k = 0$.



Note that in order to guarantee stability, we can choose the prediction horizon to be infinite. However, in the presence of constraints, the control objective becomes more complex. Consider the following state and control constraints:

$$x \in \mathcal{X} \ , \ u \in \mathcal{U} \ , \tag{4.39}$$

where the sets $\mathcal{X}$ and $\mathcal{U}$ are polyhedral regions defined by the linear state and input constraints

$$\mathcal{X} = \{x \mid A_{\mathrm{x}} x \leq b_{\mathrm{x}}\} \ , \ \mathcal{U} = \{u \mid A_{\mathrm{u}} u \leq b_{\mathrm{u}}\} \ . \tag{4.40}$$

The control task is now augmented with the following objectives:

3. Satisfy the constraints $x_k \in \mathcal{X}$, $u_k \in \mathcal{U}$ for $k = 0, \ldots, \infty$.
4. Maximize the "feasible set", i.e. the set of initial conditions $\hat{x}_0$ for which the objectives 1-3 can satisfied.

While for the unconstrained case, the feasible set spans the entire state space, this is not the case in the constrained case. For example, the feasible set excludes the part of state space that violates the state constraints. Moreover, feasible initial states $\hat{x}_0 \in \mathcal{X}$ can nevertheless lead to an unavoidable constraint violation at a later time stage.
The best possible MPC controller that achieves objectives 1-4 is defined by the following infinite-horizon optimal control problem:

$$\begin{aligned}
\underset{\substack{x_0, x_1, \ldots \\ u_0, u_1, \ldots}}{\text{minimize}} \quad & \frac{1}{2} \sum_{k=0}^{\infty} x_k^\top Q x_k + u_k^\top R u_k \\
\text{subject to} \quad & x_0 = \hat{x}_0, \\
& x_{k+1} = A x_k + B u_k, \quad k = 0, \ldots, \infty, \\
& x_k \in \mathcal{X}, \qquad\qquad k = 0, \ldots, \infty, \\
& u_k \in \mathcal{U}, \qquad\qquad k = 0, \ldots, \infty \ .
\end{aligned} \tag{4.41}$$

However, this OCP has an infinite amount of variables and is numerically intractable. In contrast to unconstrained linear MPC, this OCP does not allow for an analytic solution (explicit or implicit). Therefore, the only feasible MPC formulation is one with a finite horizon:

$$\begin{aligned}
\underset{\substack{x_0, \ldots, x_N \\ u_0, \ldots, u_{N-1}}}{\text{minimize}} \quad & \frac{1}{2} x_N^\top P_N x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\
\text{subject to} \quad & x_0 = \hat{x}_0, \\
& x_{k+1} = A x_k + B u_k, \quad k = 0, \ldots, N-1, \\
& x_k \in \mathcal{X}, \qquad\qquad k = 0, \ldots, N-1, \\
& u_k \in \mathcal{U}, \qquad\qquad k = 0, \ldots, N-1, \\
& x_N \in \mathcal{X}_f,
\end{aligned} \tag{4.42}$$

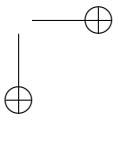
where, similar to the finite-horizon LQR, the terminal weight matrix P_N gives a quadratic approximation of the cost outside of the prediction horizon. The terminal region $\mathcal{X}_{\mathrm{f}}$, as we will see later in Section 4.5, allows us to account for the constraints outside of the prediction horizon.

Feasible set The feasible set $\mathcal{X}_{\mathrm{N}}$ is defined as the set of initial conditions x_0 for which finite-horizon problem (4.42) with horizon N is feasible. More formally, we define it as

$$\mathcal{X}_{\mathrm{N}} = \{x_0 \in \mathbb{R}^{n_x} \mid \exists (u_0, \ldots, u_{N-1}) \text{ such that } x_k \in \mathcal{X}, \ u_k \in \mathcal{U}, \ k = 0, \ldots, N-1, \tag{4.43}$$

$$x_N \in \mathcal{X}_{\mathrm{f}}, \text{ with } x_{k+1} = A x_k + B u_k\} \ . \tag{4.44}$$

Note that the feasible set is independent of the OCP cost function.
The feasible set depends on the constraint regions $\mathcal{X}, \mathcal{U}$ and $\mathcal{X}_{\mathrm{f}}$. If one of these regions shrinks, the feasible set also shrinks. The feasible set can be enlarged by increasing the horizon N. However, even for an infinite horizon it cannot become larger than the feasible state region $\mathcal{X}$, i.e. $\mathcal{X}_{\mathrm{N}} \subseteq \mathcal{X}, \forall N$.



Sparse solution method When considered from an optimization point of view, the constrained linear quadratic OCP (4.42) is a QP that can be summarized as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{2} w^\top \bar{H} w && \text{(4.45a)} \\ \text{subject to} \quad & \bar{A} w + \bar{c}\hat{x}_0 = 0, && \text{(4.45b)} \\ & \bar{D} w + \bar{e} \leq 0 && \text{(4.45c)} \end{aligned}$$

where it is important that the decision variables are ordered as $w = (x_0, u_0, x_1, u_1, \ldots, x_{N-1}, u_{N-1}, x_N)$. As soon as a new state estimate $\hat{x}_0$ becomes available, this QP needs to be solved, hence resulting in a large computational cost compared to the unconstrained case. The QP can then be solved online with any QP solver (as e.g. `quadprog` in Matlab). Moreover, the QP matrices that enter the KKT matrix, i.e. $\bar{H}, \bar{A}$ and $\bar{D}$, have a special, "banded" OCP structure with $N+1$ block matrices on the diagonal, which renders them sparse, e.g.:

$$\bar{H} = \begin{bmatrix} * & * & & & & & & \\ * & * & & & & & & \\ & & * & * & & & & \\ & & * & * & & & & \\ & & & & \ddots & & & \\ & & & & & & * & * \\ & & & & & & * & * \end{bmatrix} \tag{4.46}$$

As the horizon length N increases, they get increasingly sparse (that is, the percentage of elements that are zeros grows). This special structure can be efficiently exploited by specialized solvers, that work only with the dense sub-matrices.

Dense solution method Although the sparse QP (4.45) can be efficiently solved, it also has a large amount of of optimization variables, i.e. $n_w = N(n_x + n_u) + n_x$. For short horizons (where sparsity is relatively low), and a large amount of states, a *dense* QP formulation might be beneficial. In this formulation, the state variables are eliminated from the QP, similar as in the unconstrained case, using (4.20), resulting in the QP

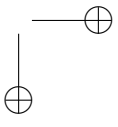
$$\begin{aligned} \underset{U}{\text{minimize}} \quad & \frac{1}{2} U^\top (\Theta^\top \hat{Q} \Theta + \hat{R}) U + U^\top \Theta \hat{Q} E \hat{x}_0 \, . \\ \text{subject to} \quad & \bar{D}_\mathrm{d} U + \bar{e}_\mathrm{d} \leq 0, \end{aligned} \tag{4.47}$$

where the matrices $\bar{D}_\mathrm{d}$, $\bar{e}_\mathrm{d}$ can be constructed by substituting the states X from the inequalities (4.45c) with (4.20).
The resulting dense QP has fewer optimization variables: $n_w = N n_u$, which results in smaller dimensions of the KKT matrix, which can result in faster computation times. However the matrices that make up the KKT matrix are now dense, e.g.:

$$(\Theta^\top \hat{Q} \Theta + \hat{R}) = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} , \tag{4.48}$$

which renders the KKT matrix more expensive to invert.
Moreover, the matrix Θ involves many matrix potentiations of the system matrix A, which can lead to an ill-conditioned KKT matrix for unstable systems.

Solution analysis The unconstrained MPC controller resulted in a linear control law. This was due to the fact that the OCP is comprised of equality constraints only. These equality constraints could be eliminated to obtain an unconstrained QP with corresponding analytic solution.
In constrained MPC, this method cannot be applied due to the inequality constraints, which cannot be eliminated since the active set typically varies depending on the initial state $\hat{x}_0$.



However, we can consider the case where within a region $\mathcal{X}_i$ around a given initial state $\hat{x}_0$, the active set $\mathcal{A}$ remains constant, i.e. all inequalities remain active or inactive. Within this region, the active inequalities can be treated as equality constraints. For the dense QP (4.47), this results in the locally equivalent QP

$$\begin{aligned} \underset{U}{\text{minimize}} \quad & \frac{1}{2}U^\top(\Theta^\top\hat{Q}\Theta + \hat{R})U + U^\top\Theta\hat{Q}E\hat{x}_0 \,. \\ \text{subject to} \quad & \bar{D}_{\mathrm{d},\mathcal{A}}U + \bar{e}_{\mathrm{d},\mathcal{A}} = 0, \end{aligned} \tag{4.49}$$

where the matrix $\bar{D}_{\mathrm{d},\mathcal{A}}$ and vector $\bar{e}_{\mathrm{d},\mathcal{A}}$ enforce only the active inequality constraints. The KKT conditions for this QP read as

$$\begin{bmatrix} (\Theta^\top\hat{Q}\Theta + \hat{R}) & \bar{D}_{\mathrm{d},\mathcal{A}}^\top \\ \bar{D}_{\mathrm{d},\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} U \\ \lambda \end{bmatrix} + \begin{bmatrix} 0 \\ \bar{e}_{\mathrm{d},\mathcal{A}} \end{bmatrix} + \begin{bmatrix} \Theta\hat{Q}E \\ 0 \end{bmatrix} \hat{x}_0 = 0 \,, \tag{4.50}$$

so that the optimal solution can be written as

$$u_0^* = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} U^* \\ \lambda^* \end{bmatrix} \tag{4.51}$$

$$= -\begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} (\Theta^\top\hat{Q}\Theta + \hat{R}) & \bar{D}_{\mathrm{d},\mathcal{A}}^\top \\ \bar{D}_{\mathrm{d},\mathcal{A}} & 0 \end{bmatrix}^{-1} \left(\begin{bmatrix} \Theta\hat{Q}E \\ 0 \end{bmatrix} \hat{x}_0 + \begin{bmatrix} 0 \\ \bar{e}_{\mathrm{d},\mathcal{A}} \end{bmatrix} \right) . \tag{4.52}$$

which is an affine control law that is optimal for all $\hat{x}_0 \in \mathcal{X}_i$. Outside of this region, the matrix $\bar{D}_{\mathrm{d},\mathcal{A}}$ changes, and thus so does the control law.
Thus, the solution of the constrained MPC problem is characterized by a set of non-overlapping regions $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n \subseteq \mathcal{X}_\mathrm{N}$ with corresponding affine feedback control laws:

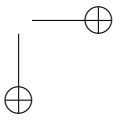
$$u_0^* = \begin{cases} K_1\hat{x}_0 + v_1, & \text{if } \hat{x}_0 \in \mathcal{X}_1, \\ \vdots & \vdots \\ K_n\hat{x}_0 + v_n, & \text{if } \hat{x}_0 \in \mathcal{X}_n, \end{cases} \tag{4.53}$$

which can also be described as a *piecewise affine* control law.
As a result, a constrained linear MPC controller is *nonlinear* controller.

Explicit MPC It is possible to compute the different regions $\mathcal{X}_i$ and corresponding feedback matrices K_i offline, using multi-parametric programming. This fact is exploited in *explicit MPC*. Here, the different regions and feedback matrices are pre-computed and stored in the form of a look-up table. In the online setting, when the state estimate comes in, the correct region needs to be identified, and the computational cost of the feedback law is then again that of a simple matrix-vector product, instead of that of solving a QP.
However tempting, this approach is only feasible for small systems and a low amount of possible active set changes. The reason is that for q optimization constraints, there are 2^q different possible active sets to be checked. Therefore, for larger systems, it is more efficient to repeatedly compute the numerical solution of either the sparse QP (4.45) or the dense QP (4.47) online.

4.5 Feasibility and stability

As discussed for the finite-horizon LQR controller, the short-sightedness of the controller causes a mismatch between the open-loop prediction and the closed-loop response of the controlled system. In the unconstrained MPC case, this can lead to a *loss of stability*. In the case of constrained MPC, this can additionally lead to infeasibility of the OCP: even if the initial state $\hat{x}_0$ is in the feasible set $\mathcal{X}_\mathrm{N}$ of the OCP, the closed-loop system might navigate to a state outside of the feasible set. In this case, the MPC problem is said to be not *recursively feasible*.
Of course, both recursive feasibility and closed-loop stability are necessary requirements for the practical application of MPC. In this section, we will investigate these properties and look into how we can enforce them by making the necessary modifications to the finite-horizon MPC problem



4.5.1 Recursive feasibility

In order to describe the property of recursive feasibility of MPC, the following definitions are useful. Since these concepts are applicable to nonlinear systems as well, we consider here the general dynamics $x_{k+1} = f(x_k, u_k)$.

Definition 15. *A set $\mathcal{O}$ is called "positive invariant" for the autonomous discrete system dynamics $x_{k+1} = f(x_k)$ if it holds that*

$$x_k \in \mathcal{O} \quad \Rightarrow \quad x_{k+1} \in \mathcal{O} \,, \quad \forall k \in \mathbb{N}_+ \,. \tag{4.54}$$

Thus, if $x_0 \in \mathcal{O}$, and the positive invariant set is within the constraints, i.e. $\mathcal{O} \in \mathcal{X}$, the system trajectory will never violate the constraints, and recursive feasibility holds.

Definition 16. *A set $\mathcal{O}_\infty \subset \mathcal{X}$ is "maximal positive invariant" with respect to $\mathcal{X}$ if it is positive invariant and if it contains all possible positive invariant sets.*

The maximal positive invariant set $\mathcal{O}_\infty$ is thus the largest possible set of initial states for which recursive feasibility of the autonomous system holds. Such a set is difficult to compute in general, but it can be done for linear systems with polyhedral state constraints.

Definition 17. *A set $\mathcal{C}$ is called "control invariant" for the controlled system $x_{k+1} = f(x_k, u_k)$, with constraint sets $\mathcal{X}$ and $\mathcal{U}$ if it holds that*

$$x_k \in \mathcal{C} \quad \Rightarrow \quad \exists u \in \mathcal{U} \quad \textit{s.t.} \quad x_{k+1} \in \mathcal{C} \,, \quad \forall k \in \mathbb{N}_+ \,. \tag{4.55}$$

Definition 18. *The set $\mathcal{C}_\infty$ is "maximal control invariant" for the controlled system $x_{k+1} = f(x_k, u_k)$ with constraint sets $\mathcal{X}$ and $\mathcal{U}$ if it is control invariant and if it contains all control invariant sets contained in $\mathcal{X}$.*

Thus, for all states in the maximal control invariant set $\mathcal{C}_\infty$, there exists a control law that ensures that the system constraints are never violated, i.e. the system can be made recursively feasible.
For constrained linear systems, it is often too complex to compute control invariant sets explicitly. For nonlinear systems, this is almost always the case. Rather, we employ constrained MPC which implicitly defines a control invariant set.
Note that the maximal control invariant set $\mathcal{C}_\infty$ represents the largest set of recursively feasible initial conditions that any controller can achieve! If the set $\mathcal{C}_\infty$ would be known, this ideal controller $\kappa(x)$ could be formulated as the solution of following optimization problem:

$$\begin{aligned} \kappa(x) = \; & \underset{u \,\in\, \mathcal{U}}{\arg\min} \quad J(x,u) \\ & \text{subject to} \quad f(x,u) \in \mathcal{C}_\infty \,, \end{aligned} \tag{4.56}$$

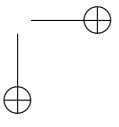
for any arbitrary cost function $J(x, u)$ (including $J(x, u) = 0$).
The maximal control invariant set is in general very difficult to compute, so that in practice, we cannot use (4.56) for our controller synthesis. However, we can also define $\mathcal{C}_\infty$ implicitly via the infinite-horizon constrained MPC problem

$$\begin{aligned} \underset{\substack{x_0, x_1, \ldots \\ u_0, u_1, \ldots}}{\text{minimize}} \quad & \sum_{k=0}^{\infty} J(x_k, u_k) \\ \text{subject to} \quad & x_0 = \hat{x}_0, \\ & x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, \infty, \\ & x_k \in \mathcal{X}, \qquad\quad k = 0, \ldots, \infty, \\ & u_k \in \mathcal{U}, \qquad\quad k = 0, \ldots, \infty \,. \end{aligned} \tag{4.57}$$

which is recursively feasible if and only if $\hat{x}_0 \in \mathcal{C}_\infty$, independent of the cost function J.
As we well know, this problem is computationally intractable. Therefore we approximate this problem with a finite-horizon MPC problem with horizon N, and we introduce the terminal constraint

$$x_N \in \mathcal{X}_\mathrm{f} \,, \tag{4.58}$$

with the terminal set $\mathcal{X}_\mathrm{f} \subset \mathcal{X}$. The terminal set $\mathcal{X}_\mathrm{f}$ and horizon N should be chosen so that:



- ... the feasible set of the MPC problem $\mathcal{X}_\mathrm{N}$ is control-invariant, i.e. every feasible initial condition is recursively feasible.
- ... the feasible set of the MPC problem $\mathcal{X}_\mathrm{N}$ is as large as possible (ideally $\mathcal{X}_\mathrm{N} = \mathcal{C}_\infty$).

Typically two variants are considered: a terminal point constraint at zero, and a terminal constraint in some convex set.

Terminal point constraint The terminal point constraint is formulated as

$$x_N = 0 \,. \tag{4.59}$$

Independent of the chosen horizon, this constraint enforces recursive feasibility for all feasible points. This can be proven as follows.
Consider any feasible initial condition $x_k \in \mathcal{X}_\mathrm{N}$ The corresponding feasible and optimal control and state trajectories are given by $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$, $\{x_0^*, x_1^*, \ldots, x_N^*\}$, with $x_N^* = 0$. We apply the first control and compute the next state $Ax_k + Bu_0^* = x_1^*$.
Recursive feasibility dictates that x_1^* must be feasible as well. This is the case, since we can construct the following feasible control trajectory: $\{u_1^*, u_2^*, \ldots, u_{N-1}^*, 0\}$. The first $N-1$ stages are feasible since they are identical to the last $N-1$ stages of the previous, feasible problem. The last stage is also feasible since it holds that $A \underbrace{x_N^*}_{=0} + B \cdot 0 = 0$, which satisfies the terminal point constraint. Hence, recursive feasibility holds for any $x_k \in \mathcal{X}_N$ and the feasible set is control invariant.
The terminal point constraint is a very simple way to impose recursive feasibility, but it often results in an overly restrictive feasible set, unless the horizon N is chosen very large, which is computationally expensive. Therefore a terminal set constraint is often preferable.

Terminal set constraint Since the terminal point constraint reduces the size of the feasible set, we can relax this constraint to a terminal set constraint, with convex set $\mathcal{X}_\mathrm{f}$:

$$x_N \in \mathcal{X}_\mathrm{f} \,. \tag{4.60}$$

How to choose this terminal set to ensure recursive feasibility? It must be control invariant!
That means that there must exist some local control law $\kappa_\mathrm{f}(x_k)$ such that

$$x_{k+1} = f(x_k, \kappa_\mathrm{f}(x_k)) \in \mathcal{X}_\mathrm{f}, \quad \text{for all } x_k \in \mathcal{X}_\mathrm{f} \,, \tag{4.61}$$

while satisfying the constraints

$$\mathcal{X}_\mathrm{f} \subseteq \mathcal{X}, \quad \kappa_\mathrm{f}(x_k) \in \mathcal{U}, \quad \text{for all } x_k \in \mathcal{X}_\mathrm{f} \,. \tag{4.62}$$

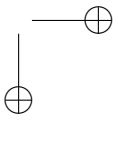
Recursive feasibility can now be shown in similar fashion to the terminal point constraint.
Consider a feasible initial state $x_k \in \mathcal{X}_\mathrm{N}$ and corresponding feasible and optimal control and state trajectories given by $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$, $\{x_0^*, x_1^*, \ldots, x_N^*\}$, with $x_N^* = 0$.
At the next state $x_{k+1} = x_1^*$, we can construct the following feasible control sequence:
$\{u_1^*, u_2^*, \ldots, u_{N-1}^*, \kappa_\mathrm{f}(x_N^*)\}$. This sequence is feasible because $x_N^* \in \mathcal{X}_\mathrm{f}$ which is positive invariant for the closed-loop system with control law κ_f. Thus: $\kappa_\mathrm{f}(x_N^*) \in \mathcal{U}$ and $Ax_N^* + B\kappa_\mathrm{f}(x_N^*) \in \mathcal{X}_\mathrm{f}$, so that recursive feasibility holds for all $x_k \in \mathcal{X}_\mathrm{N}$.

Example. As mentioned above, it is in general difficult to compute (maximal) control invariant sets. However, in the case of a linear system with quadratic cost, it can be done. Let us choose as a local control law the infinite horizon LQR controller K_∞, given by (4.35). The terminal set $\mathcal{X}_\mathrm{f}$ can now be chosen to be the maximum positive invariant set of the closed-loop system $x_{k+1} = (A + BK_\infty)x_k$, so that:

$$x_{k+1} = (A + BK_\infty)x_k \in \mathcal{X}_\mathrm{f}, \quad \text{for all } x_k \in \mathcal{X}_\mathrm{f} \,, \tag{4.63}$$

and so that all constraints are satisfied:

$$\mathcal{X}_\mathrm{f} \subseteq \mathcal{X}, \quad K_\infty x_k \in \mathcal{U}, \quad \text{for all } x_k \in \mathcal{X}_\mathrm{f} \,. \tag{4.64}$$



The "trick" of the terminal set is thus to construct a region inside of the constraints, for which it holds that an unconstrained controller can keep the system inside this region.
For increasing N, the (recursively) feasible set $\mathcal{X}_\mathrm{N}$ of the OCP problem will approach the maximum invariant set $\mathcal{C}_\infty$ of the constrained problem.
Recursive feasibility does not depend on the cost function, but it also does not guarantee stability of the system. To guarantee closed-loop stability, additional considerations need to be made.

4.5.2 Stability

Stability of nonlinear systems Let us first formally define stability for nonlinear, time-invariant, discrete systems of the form $x_{k+1} = f(x_k)$, around an equilibrium point $x_\mathrm{ss} = f(x_\mathrm{ss})$.

Definition 19. *The equilibrium point $x_\mathrm{ss} \in \Omega$ of the system $x_{k+1} = f(x_k)$ is asymptotically stable in the positive invariant set $\Omega \subseteq \mathbb{R}^{n_x}$ if it is Lyapunov stable and attractive, i.e.*

$$\lim_{k\to\infty} \|x_k - x_\mathrm{ss}\| = 0\,, \quad \forall x_0 \in \Omega\,. \tag{4.65}$$

If it additionally holds that $\Omega = \mathbb{R}^{n_x}$, then the equilibrium point is globally asymptotically stable.

In this context, the set Ω is also called the *region of attraction* of the equilibrium point.
How do we investigate stability of general nonlinear systems? To this aim, Lyapunov functions can be a useful tool.

Definition 20. *Consider the equilibrium point $x_\mathrm{ss} = 0$ of the system $x_{k+1} = f(x_k)$. Let $\Omega \subset \mathbb{R}^{n_x}$ be a closed and bounded positive invariant set for this system, containing the origin. A function $V : \mathbb{R}^{n_x} \to \mathbb{R}$, which*

- *is continuous at the origin.*
- *is finite for all $x \in \Omega$.*
- *satisfies*

$$V(0) = 0\,, \tag{4.66}$$
$$V(x) > 0, \qquad \forall x \in \Omega\backslash\{0\}\,, \tag{4.67}$$
$$V(f(x)) - V(x) \leq -\alpha(x), \quad \forall x \in \Omega\backslash\{0\}\,, \tag{4.68}$$

where $\alpha : \mathbb{R}^{n_x} \to \mathbb{R}$ is continuous positive definite,

is called a Lyapunov function.

If there exists a Lyapunov function $V(x)$ in Ω for the system dynamics f, then the equilibrium point x_ss is asymptotically stable in Ω.

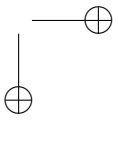
Example. Consider the autonomous nonlinear system $x_{k+1} = x_k^2$, with $x_{k+1}, x_k \in \mathbb{R}$. Consider now the trial function $V(x) = x^2$. We investigate the condition (4.68):

$$V(x_{k+1}) - V(x_k) = x_k^4 - x_k^2 = x_k^2(x_k^2 - 1). \tag{4.69}$$

It holds that $V(x_{k+1}) - V(x_k) < 0$ if and only if $(x_k^2 - 1) < 0$, i.e. if $|x_k| < 1$. Thus, the trial function is a Lyapunov function for this particular system with the region of attraction $\Omega = \{x \in \mathbb{R} \mid |x| < 1\}$. For states $x_k \notin \Omega$, the nonlinear system is unstable.

Example. Consider an unconstrained linear system with state $x \in \mathbb{R}^{n_x}$ that is being regulated with an infinite-horizon LQR controller K_∞ with the resulting closed-loop dynamics $x_{k+1} = (A + BK_\infty)x_k$. To show stability of the autonomous closed-loop system, we consider as a trial function the infinite-horizon cost-to-go:

$$V(x) = x^\top P_\infty x\,, \tag{4.70}$$



where $P_\infty \succ 0$ is the solution of the DARE (4.36). We recall that K_∞ is given as a function of P_∞ by (4.35). We now test the Lyapunov criterium (4.68):

$$V(x_{k+1}) - V(x_k) = x_k^\top (A + BK_\infty)^\top P_\infty (A + BK_\infty) x_k - x_k^\top P_\infty x_k \,. \tag{4.71}$$

This expression less then zero if and only if it holds that

$$(A + BK_\infty)^\top P_\infty (A + BK_\infty) - P_\infty \prec 0 \,. \tag{4.72}$$

We can reformulate the left hand side of this matrix inequality, first using expression (4.35), then using the DARE (4.36) to obtain:

$$A^\top P_\infty A + A^\top P_\infty B K_\infty - P_\infty - K_\infty^\top R K_\infty = -Q - K_\infty^\top R K_\infty \prec 0 \,. \tag{4.73}$$

The final inequality holds for any $Q \succeq 0$, $R \succ 0$. Thus the infinite-horizon LQR controller (if it exists) leads to a *globally* asymptotically stable closed-loop system, i.e, for all $x_k \in \mathbb{R}^{n_x}$.

Stability of linear MPC We now have all the necessary tools to investigate the closed-loop stability of a linear system controlled using linear MPC. Recall that this closed-loop system has piecewise linear (thus nonlinear) dynamics. Similar to recursive feasibility, we consider two cases of linear MPC: one with a terminal point constraint, and one with a terminal set and terminal cost. In both cases we will consider as a trial function the optimal MPC cost function:

$$V(x) = J^*(x) \,, \tag{4.74}$$

which is the optimal cost value obtained by solving (4.18) for $\hat{x}_0 = x$.
Note that for the case of a terminal point constraint $x_N = 0$, the terminal cost $x_N^\top P_N x_N$ is always zero, and it can be dropped.

Terminal point constraint We consider an initial state in the feasible set, i.e., $x_0 \in \mathcal{X}_\mathrm{N}$. The feasible set is recursively feasible, as shown in previous considerations. To investigate the Lyapunov criterium, we first evaluate the trial function at x_k:

$$V(x_k) = \sum_{i=0}^{N-1} x_i^{*\top} Q x_i^* + u_i^{*\top} R u_i^* \,. \tag{4.75}$$

with x_i^*, u_i^* the optimal MPC solution for initial state x_k. How to evaluate the trial function at x_{k+1}? We know that the MPC problem for initial state x_{k+1} admits as a feasible, but possibly suboptimal solution $(u_1^*, u_2^*, \ldots, u_{N-1}^*, 0)$, $(x_1^*, x_2^*, \ldots, x_N^*, 0)$. Thus, the optimal cost for this initial state must be less or equal than the cost for this feasible-but-suboptimal solution:

$$V(x_{k+1}) \leq \sum_{i=1}^{N-1} x_i^{*\top} Q x_i^* + u_i^{*\top} R u_i^* = V(x_k) - \underbrace{(x_0^{*\top} Q x_0^{*\top} + u_0^{*\top} R u_0^{*\top})}_{>0,\ \text{if } x_0^* \neq 0} \tag{4.76}$$

Thus we can conclude that for all $x_0^* \in \mathcal{X}_\mathrm{N} \backslash \{0\}$, it holds that

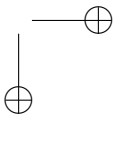
$$V(x_{k+1}) - V(x_k) < 0 \,, \tag{4.77}$$

which proves that the closed loop is asymptotically stable.

Terminal set constraint In the case of a terminal set constraint, we again assume that an unconstrained, invariance-inducing controller $\kappa_\mathrm{f}(x)$ is applied inside the terminal set $\mathcal{X}_\mathrm{f}$. In the case of linear MPC, this controller can be chosen as the infinite horizon LQR, i.e. $\kappa_\mathrm{f}(x) = K_\infty x$. We can now enforce stability by ensuring that the terminal cost $x^\top P_N x$ is a Lyapunov function in the terminal set (i.e., the control law κ_f is stabilizing inside the terminal set). Moreover, it should hold that:

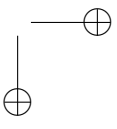
$$x_{k+1}^\top P_N x_{k+1} - x_k^\top P_N x_k \leq -x_k^\top Q x_k - \kappa_\mathrm{f}(x_k)^\top R \kappa_\mathrm{f}(x) \,. \tag{4.78}$$

This condition is satisfied for the case that $\kappa_\mathrm{f}(x) = K_\infty x$ and $P_N = P_\infty$, as can be seen from inequality (4.73).
We leave the proof as an exercise for the reader.



Summary We summarize the problem of feasibility and stability for MPC in the following way:

- An infinite-horizon MPC controller provides stability and recursive feasibility (invariance).
- The infinite horizon can be replaced by forcing the final predicted state into an invariant set for which an invariance-inducing and stabilizing controller exists, for which the infinite-horizon cost-to-go can be expressed in close form.
- The simplest terminal set is given by a point constraint, i.e. $\mathcal{X}_\mathrm{f} = \{0\}$. However, this may restrict the feasible set $\mathcal{X}_\mathrm{N}$ (and thus region of attraction) too much.
- The combination of a convex terminal set and terminal cost function can increase the feasible set. For linear systems and quadratic cost functions, such a set and cost function can be computed.
- The region of attraction can approach the maximum control invariant set by increasing the horizon length N.
- In practice, the terminal constraint is often omitted at the cost of a very large horizon length N. This then leads to a larger region of attraction, which is however difficult to characterize. In this case, stability cannot be guaranteed a priori, but is checked empirically via sampling.
- The concepts above can be directly applied to nonlinear MPC as well. However, in this case, computing a stabilizing terminal set and cost function is very difficult.



Chapter 5

Nonlinear Model Predictive Control

The nonlinear model predictive control (NMPC) approach allows more challenging control problems to be handled than those dealt with by the linear model predictive control (LMPC) method. NMPC employs nonlinear system models that accurately capture the dynamics of systems with inherent nonlinearities. This enables NMPC to provide superior performance and improved control outcomes compared to LMPC, especially when dealing with complex control problems.

5.1 Introduction to NMPC

The system model within LMPC is restricted to being a linear or a linearized model with a quadratic objective function such as for a reference tracking problem. However, the use of a linear model is often insufficient for representing the system behavior. In these cases, the NMPC approach allows for a better control performance than the LMPC. The term NMPC is used typically when the resulting optimization problem is neither a QP nor an LP. An NLP optimization problem results for MPC when a nonlinear system model, nonlinear constraints or a non-convex cost function is considered. Typically, nonlinear models are present as continuous-time models, e.g. because they are derived from physical modeling. The consideration of this continuous-time nonlinear models in an optimization-based control algorithm results in an OCP. An example of an OCP is depicted by the following optimization problem with cost function J_{ocp}.

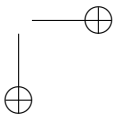
$$J_{\mathrm{ocp}}(x(t), u(t)) = \int_0^T L(x(t), u(t)) \ \mathrm{d}t \ \ + \ \ E\left(x(T)\right)$$

The cost function J_{ocp} in this example is subject to the system dynamics given as an ODE model, the fixed initial state of the system at $t = t_0$, the equality constraints on the final states T, and the path constraints which are present as inequality constraints.

$$\begin{array}{rcll}
\min_{x,u} & & J_{\mathrm{ocp}}(x(t), u(t)) & \\
\text{s.t.} & & & \\
\dot{x}(t) & = & f(x(t), u(t)), & t \in [0, T], \quad \text{(ODE model)} \\
x(0) & = & x_0, & \quad \text{(fixed initial value)} \\
h(x(t), u(t)) & \leq & 0, & t \in [0, T], \quad \text{(path constraints)} \\
r(x(T)) & = & 0 & \quad \text{(end point constraints)}
\end{array}$$

Equivalent formulations Other variants of the above presented OCP that result in equivalent reformulations or simple extensions could be considered as well. We remark that, for instance, other optimization variables could be present as well, such as a free parameter p that can be chosen but is constant over time, like e.g. the size of a component in the system. Such parameters could be added to the optimization formulation above by defining dummy states $p(t)$ that satisfy the dummy dynamic model equations

$$\dot{p} = 0.$$



This implies that they are constant over time. Note that the initial value of p_0 is not fixed by these constraints and thus we would have obtained our aim of having a time constant parameter vector that is free for optimization.
In optimization, we might have different requirements than just a fixed initial state. We might consider a free initial state that is also up to optimization. We might, for example, have both a fixed initial state and a fixed terminal state that we want to reach. Or we might just look for periodic sequences with $x(t_0) = x(T)$. All these desires on the initial and the terminal state can be expressed by a boundary constraint function

$$r(x(t_0), x(T)) = 0.$$

For the case of fixed initial value $x(t_0) = x_0$, this function would just be

$$r(x(t_0), x(T)) = x(t_0) - x_0$$

where x_0 is the fixed initial value and not an optimization variable. Another example would be to have both ends fixed, resulting in a function r as follows:

$$r(x(t_0), x(T)) = \begin{bmatrix} x(t_0) - x_0 \\ x(T) - x_T \end{bmatrix}.$$

Finally, periodic boundary conditions $x(t_0) = x(T)$, for instance to be found in period movement of a kite, can be imposed by setting

$$r(x(t_0), x(T)) = x(t_0) - x(T)$$

Of course, also constraints at other discrete time points t_k other that the initial and final time could be considered. Furthermore, the constraints could also be defined as inequality constraints instead of equality constraints, which is common practice for a terminal constraint on the final time $t = T$.
Other constraints that are usually present are *path constraint* inequalities of the form

$$h(x(t), u(t)) \leq 0, \quad 0 \leq t \leq T.$$

In the case of upper and lower bounds on the controls, $u_{\min} \leq u_k \leq u_{\max}$, the function h would just be

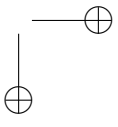
$$h(x, u) = \begin{bmatrix} u - u_{\max} \\ u_{\min} - u \end{bmatrix}.$$

Discretized OCP Due to the infinite number of optimization variables, OCPs usually cannot be solved in real time, especially when a nonlinear model is considered. Thus, the OCP cannot be directly used for practical NMPC applications. However, in order to obtain a tractable optimization problem, the OCP can be approximated by an NLP, that is, a finite-dimensional optimization problem. The transformation from the OCP to the NLP is conducted by a suitable discretization of the control and state trajectories on the time horizon. The process of the discretization is detailed in the following section.
Given a discretization method, a quite generic discrete time optimal control problem can be formulated as the following constrained NLP:

$$\begin{aligned}
\underset{\substack{x_0, u_0, x_1, \ldots, \\ u_{N-1}, x_N}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) && \text{(5.1a)} \\
\text{subject to} \quad & x_{k+1} - f(x_k, u_k) = 0, \quad k = 0, \ldots, N-1, && \text{(5.1b)} \\
& h(x_k, u_k) \leq 0, \quad k = 0, \ldots, N-1, && \text{(5.1c)} \\
& r(x_0, x_N) = 0. && \text{(5.1d)}
\end{aligned}$$

Note that an NLP also results if the system model, cost function, and the constraints are directly given in discrete time. Also for data-based modeling, usually a discrete-time system model results. The general NLP is usually of the following form:

$$\begin{aligned}
\min_{z \in \mathbb{R}^n} \quad & J(z) \\
\text{s.t.} \quad & g(z) = 0, \\
& h(z) \leq 0
\end{aligned}$$



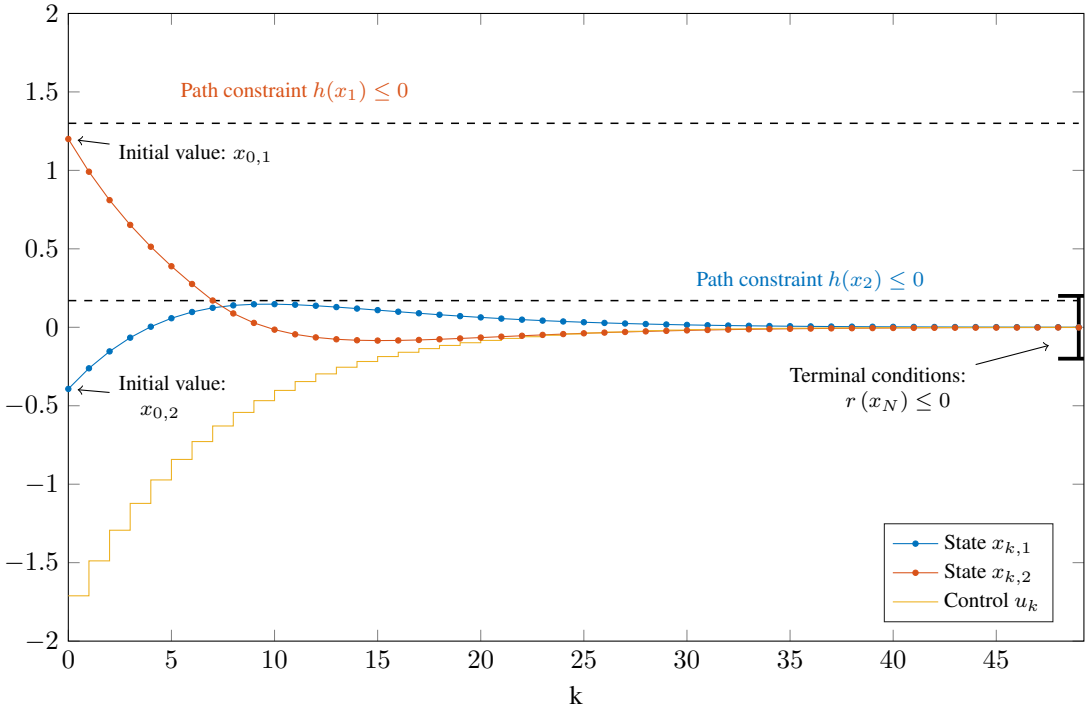


Figure 5.1: Variables of a discrete optimal control problem with $N = 49$

The optimization variables are denoted by $z \in \mathbb{R}^n$, while g defines the equality constraints and h the inequality constraints. Here, $z = (x_0, u_0, x_1, u_1, \ldots, x_{N-1}, u_{N-1}, x_N)$, g is given by the constraints (5.1b) and (5.1d) and h by (5.1c).

5.2 Numerical Integration Methods

Before we delve into NMPC, we should take a look at simulation methods, that is, methods for numerical integration in time. Within NMPC the system states need to be predicted, for which numerical simulation methods are used. The goal of numerical simulation is to compute the trajectory of $x(t)$ which, starting from the initial values, satisfies as best possible the system dynamics given by the ODE. This problem is also referred to as an initial value problem (IVP). The system dynamics and the initial value condition can be described by

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), t) \\ x(t_0) &= x_0 \end{aligned}$$

The trajectory $u(t)$ can be assumed to be part of the function f. Thus, the following differential equation with simplified notation is examined in the remainder of this section.

$$\begin{aligned} \dot{x}(t) &= f(x(t), t) \\ x(t_0) &= x_0 \end{aligned}$$

Different to linear systems, nonlinear differential equations allow an analytical solution only in a few special cases. Numerical integration methods are used to approximately solve a well-posed IVP that satisfies the conditions of Theorem 1.

All numerical integration methods start by discretizing the state trajectories over a discretization time grid over the integration interval $[t_0, t_f]$. For the sake of simplicity, let us assume a uniform time grid, i.e. having fixed interval sizes of $\Delta t = (t_f - t_0)/N$, where N is a positive integer. The discretization time grid is

then setup as $t_k := t_0 + k\Delta t$ with $k = 0, \ldots N$, and divides the time interval $[t_0, t_\mathrm{f}]$ into N subintervals $[t_k, t_{k+1}]$, each of length Δt. Then, the solution is approximated on the grid points t_k by discrete values s_k that shall satisfy

$$s_k \approx x(t_k),\ k = 0, \ldots, N$$

where $x(t)$ is the exact solution to the IVP.
Numerical integration methods differ in the ways they approximate the solution on the grid points and in between, but they all shall have the property that

$$s_k \to x(t_k) \text{ if } N \to \infty.$$

This property is labelled *convergence*. Methods differ in how fast the integrator converges as N increases. One says that a method is convergent with order p if

$$\max_{k=0,\ldots,N} \|s_k - x(t_k)\| = O(\Delta t^p).$$

In this case, the *local truncation error*

$$\tau_k = \|x(t_k) - s_k(x(t_{k-1}), \Delta t)\|,$$

that is, the error caused by one step s_k of the numerical method if started at $x(t_{k-1})$ must be of order $O(\Delta t^{p+1})$.
Numerical integration methods come in many different variants, and can be categorized according to two major branches, on the one hand the one-step vs. the multistep methods, on the other hand the explicit vs. the implicit methods.

One-Step Versus Multi-step Methods One-step methods only use the values s_k at the discrete time instance k to calculate s_{k+1}. Multi-step methods also use previous values $s_k, s_{k-1}, \ldots$ at the discrete time instance k for the calculation of s_{k+1}. One example for a one-step method is the Runge–Kutta 4 method. Examples for multi-step (not covered here) schemes are the Adams–Bashforth and the Adams–Moulton methods.

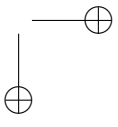
Implicit Versus Explicit Methods Explicit methods use for the calculation of s_{k+1} only derivatives at previous values of x up to time point t_k, e.g. $s_k, s_{k-1}, \ldots$. In implicit methods, s_{k+1} depends also on the derivative at time point t_{k+1}. Therefore, to determine s_{k+1}, an iterative solution method as the Newton method has to be used.

5.2.1 Explicit Methods

The simplest integrator is the explicit Euler method. It first sets $s_0 := x_0$ and then recursively computes, for $k = 0, \ldots, N-1$:

$$s_{k+1} := s_k + \Delta t\, f(s_k, t_k).$$

It is a first-order method, i.e. $p = 1$, and due to this low order it is very inefficient and should not be used in practice. Indeed, a few extra evaluations of f in each step can easily yield higher-order methods with higher accuracy.
Other methods exist that deliver the desired accuracy levels at much lower computational cost. One of the most widespread integrators is the *Runge-Kutta Method of Order Four*, abbreviated as *RK4*. One step of the RK4 method needs four evaluations of f and stores the results in four intermediate quantities $k_i \in \mathbb{R}^{n_x}$, $i = 1, \ldots, 4$.



Like the Euler integration method, the RK4 also generates a sequence of values s_k, $k = 0, \ldots, N$, with $s_0 = x_0$. At s_k, and using step size $h = \Delta t$, one step of the RK4 method proceeds as follows:

$$
\begin{aligned}
k_1 &= f(s_k, t_k) && \text{(5.2a)}\\
k_2 &= f(s_k + \frac{h}{2}\, k_1, t_k + \frac{h}{2}) && \text{(5.2b)}\\
k_3 &= f(s_k + \frac{h}{2}\, k_2, t_k + \frac{h}{2}) && \text{(5.2c)}\\
k_4 &= f(s_k + h\, k_3, t_k + h) && \text{(5.2d)}\\
s_{k+1} &= s_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) && \text{(5.2e)}
\end{aligned}
$$

Runge-Kutta (RK) methods methods use on each discretization interval $[t_k, t_{k+1}]$ not only one but m evaluations of f. They then hold intermediate state values $s_{k,i}$, $i = 1, \ldots, m$ within each interval $[t_k, t_{k+1}]$, which live on a grid of intermediate time points $t_{k,i} := t_k + c_i\, \Delta t$ with suitably chosen $c_i \in [0, 1]$. One RK step is then obtained via the following construction: There exist many other Runge–Kutta methods. They are given by

$$s_{k+1} = s_k + h \sum_{i=1}^{N} b_i k_i$$

where

$$
\begin{aligned}
k_1 &= f(s_k, t_k),\\
k_2 &= f(s_k + (a_{21}k_1)h, t_{k,2}),\\
k_3 &= f(s_k + (a_{31}k_1 + a_{32}k_2)h, t_{k,3}),\\
&\vdots\\
k_m &= f(s_k + (a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1})h, t_{k,m}).
\end{aligned}
$$

To specify a particular method, one needs to provide the integer m (the number of stages), and the coefficients a_{ij} (for $1 \le j < i \le m$), b_i (for $i = 1, 2, ..., m$) and c_i (for $i = 2, 3, ..., m$). The matrix $[a_{ij}]$ is called the Runge–Kutta matrix, while the b_i and c_i are known as the weights and nodes. Each RK method is characterized by its so-called Butcher tableau of dimension m:

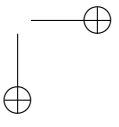
$$
\begin{array}{c|cccc}
0 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \ddots & \ddots & & \\
c_m & a_{m1} & \cdots & a_{m,m-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_m
\end{array}
$$

The RK4 Butcher tableau is

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

which yields a method of order $m = 4$, often simply referred to as the RK4 integration scheme. One step of RK4 is thus as expensive as four steps of the Euler method. But it can be shown that the local truncation error is of order h^5, therefore rendering RK4 a fourth-order method. In practice, this means that the RK4 method usually needs tremendously fewer function evaluations than the Euler method to obtain the same accuracy level.

Practical Runge-Kutta methods also have stepsize control, i.e. they adapt Δt depending on estimates of the local error, which are obtained by comparing two RK steps of different orders. Particularly efficient



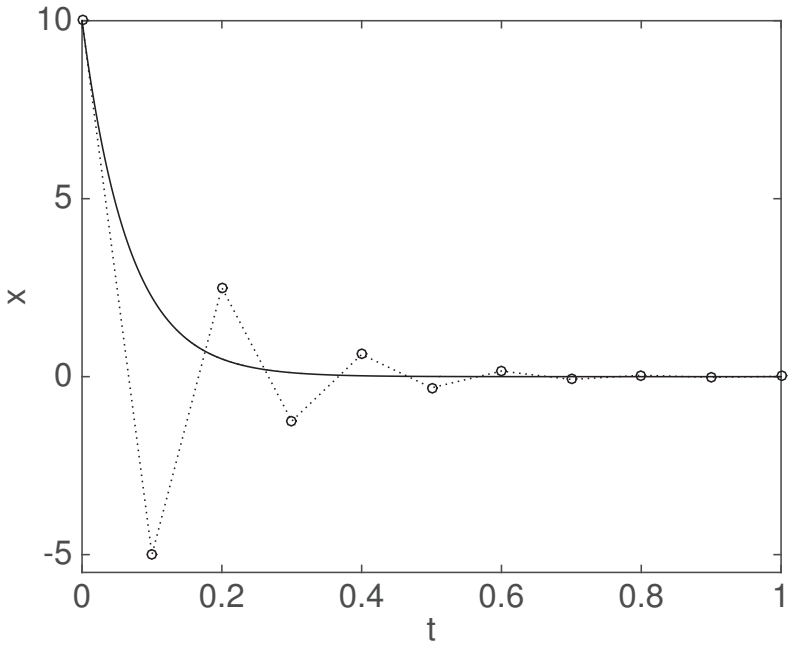


Figure 5.2: Numerical simulation of the first-order linear dynamics $\dot{x} = -15x$ using the explicit Euler method with $\Delta t = 0.1$, starting from the initial condition $x(0) = 10$. The exact solution is displayed as a plain curve, while the numerical solution is displayed using circles, connected by dotted lines. One can observe that due to the steep state derivative $\dot{x}$ in the early time of the integration, the explicit Euler scheme, which essentially computes the next state on the time grid via the tangent to the trajectory, significantly overshoots the exact solution of the ODE.

adaptive methods are the *Runge-Kutta-Fehlberg* methods, which reuse as many evaluations of f as possible between the two RK steps.
Because of its simplicity, the Euler method may appear appealing in practice, however it is strongly recommended to favor higher-order methods. To get an intuitive idea of why it is so, let us assume that we want to simulate an ODE on the interval $[0, 1]$ with an accuracy of $\epsilon = 10^{-3}$ and that a first-order method gives an accuracy $\epsilon = 10\Delta t$. Then a time step of $\Delta t = 10^{-4}$ is required, i.e. $N = 10000$ steps are necessary in order to achieve the desired accuracy. If a fourth-order method gives the accuracy $\epsilon = 10(\Delta t)^4$, a time step of $\Delta t = 0.1$ is needed, i.e. only $N = 10$ steps are required for the same accuracy. Given this enormous difference, the fourfold cost per RK step required to deploy the fourth-order method is more than outweighed by the low number of steps required, such that it is actually 250 times cheaper than the first-order Euler method. In practice, Runge-Kutta integrators with orders up to 8 are used, but the Runge-Kutta-Fehlberg method of fourth order (with fifth-order evaluation for error estimation and control) is the most popular one.

5.2.2 Stiff Systems and Implicit Integrators*

When an explicit integrator is applied to a very stable system, its steps can overshoot the actual trajectory of the ODE solution, resulting in an inaccurate numerical integration, or even outright instability. The simple prototypical first-order system is often used to discuss these issues:

$$\dot{x} = -\lambda x.$$

It takes the explicit, exact solution $x(t) = x(t_0)e^{-\lambda(t-t_0)}$. For a very large $\lambda \gg 1$ the ODE has a very fast stable mode decaying very quickly to zero. If we now use an explicit Euler method with stepsize Δt, then the trajectories of the discrete state s_k are defined by the discrete-time dynamic system:

$$s_{k+1} = s_k - \Delta t\, \lambda s_k = (1 - \Delta t\, \lambda)s_k, \quad s_0 = x(t_0),$$

which differs significantly from the exact trajectories $x(t)$, see Fig. 5.2 for an illustration. This discrete system actually becomes unstable if $\Delta t > \frac{2}{\lambda}$, which might be very small when λ is very large. Note that such a small stepsize is not necessary to obtain a high accuracy, but is only necessary to render the integrator stable.
It turns out that all explicit methods suffer from the fact that systems having very fast modes necessitate excessively short step sizes. This becomes particularly problematic if a system has both slow and fast decaying modes, i.e., if some of the eigenvalues of the Jacobian $\frac{\partial f}{\partial x}$ have a small magnitude while others are strongly negative, resulting in very quickly decaying dynamics. In such a case, one typically needs to perform fairly long simulations in order to capture the evolution of the slow dynamics, while very short steps are required in order to guarantee the stability and accuracy of the numerical integration due to the very fast modes. Such systems are called stiff systems.
Instead of using explicit integrators with very short stepsizes, stiff systems can be much better treated by implicit integrators. The simplest of them is the implicit Euler integrator, which in each integrator step solves the nonlinear equation in the variable s_{k+1}

$$s_{k+1} = s_k + \Delta t\, f(s_{k+1}, t_{k+1}).$$

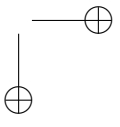
One ought to observe the subtle yet crucial difference between this equation and the one used for deploying an explicit Euler integrator. While explicit Euler requires implementing an explicit rule, the equation above provides s_{k+1} implicitly. If applied to the fast, stable test system from above, for which this equation can be solved explicitly because of the linear dynamics, the implicit Euler scheme yields the discrete dynamics

$$s_{k+1} = s_k - \Delta t\, \lambda s_{k+1} \quad \Leftrightarrow \quad s_{k+1} = s_k/(1 + \Delta t\, \lambda),$$

which are stable for any $\Delta t > 0$ and always converge to zero, like the true solution of the ODE. Hence the implicit Euler scheme is always stable for this example. This idea can be easily generalized to implicit Runge-Kutta methods where the nonlinear system needs typically to be solved by a Newton method.

5.3 Overview of Solution Methods for continuous-time OCP

Generally speaking, there are three basic families of approaches to address continuous-time optimal control problems, (a) state-space, (b) indirect, and (c) direct approaches, cf. the top row of Fig. 5.3. While we only consider the direct methods in this course, in particular direct multiple shooting, we give a brief overview about the three different approaches.
State-space approaches use the principle of optimality that states that each subarc of an optimal trajectory must be optimal. While this was the basis of dynamic programming in discrete time, in the continuous time case this leads to the so-called *Hamilton-Jacobi-Bellman (HJB) equation*, a partial differential equation (PDE) in the state space. Methods to numerically compute solution approximations exist, but the approach severely suffers from Bellman's "curse of dimensionality" and is restricted to small state dimensions.
Indirect Methods utilize the necessary conditions of optimality for infinite-dimensional problems, which are similar to the KKT conditions used in optimization problems with finite dimensions. While the KKT conditions result in a root-finding problem involving a system of equations, the necessary conditions of optimality for infinite-dimensional problems give rise to a boundary value problem (BVP) in ordinary differential equations (ODEs). A BVP can be seen as analogous to an initial value problem but with additional end-point constraints. This BVP must numerically be solved, and the approach is often sketched as "first optimize, then discretize", as the conditions of optimality are first written in continuous time for the given problem, and then discretized in one way or another in order for computing a numerical solution. The two major drawbacks are that the underlying differential equations are often difficult to solve due to strong nonlinearity and instability, and that changes in the control structure, i.e. the sequence of arcs where different constraints are active, are difficult to handle.
Direct methods transform the original infinite-dimensional optimal control problem into a finite-dimensional nonlinear programming problem (NLP) which is then solved by structure-exploiting numerical optimization methods. Roughly speaking, direct methods transform (typically via numerical methods) the continuous-time dynamic system into a discrete-time system and then proceed as described in the first two parts of this script. The approach is therefore often sketched as "first discretize, then optimize", as the problem is first



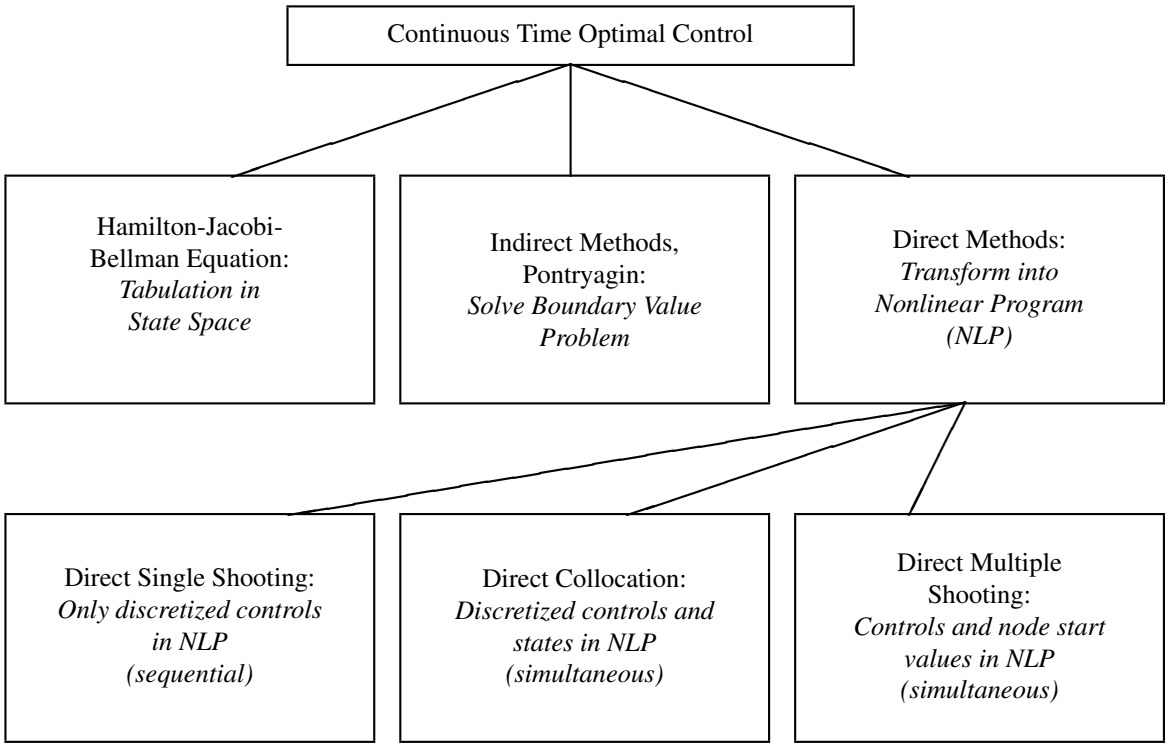
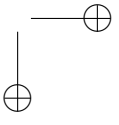


Figure 5.3: The optimal control family tree.

converted into a discrete one, on which optimization techniques are then deployed. One of the most important advantages of direct methods over indirect ones is that they can easily treat all sorts of constraints, such as e.g. the inequality path constraints in the formulation above. This ease of treatment stems from the fact that the activation and de-activation of the inequality constraints, i.e. structural changes in active constraints, occurring during the optimization procedure are treated by well-developed NLP methods that can efficiently deal with such active set changes. All direct methods are based on one form or another of finite-dimensional parameterization of the control trajectory, but differ significantly in the way the state trajectory is handled, cf. the bottom row of Fig. 5.3. For solution of constrained optimal control problems in real world applications, direct methods are nowadays by far the most widespread and successfully used techniques, and are therefore the focus of this script.

5.4 Discretization of the OCP via Direct Shooting Methods

Direct methods to continuous optimal control finitely parameterize the infinite dimensional decision variables, notably the controls $u(t)$, such that the original problem is approximated by a finite dimensional nonlinear program (NLP). This NLP can then be addressed by structure exploiting numerical NLP solution methods. For this reason, the approach is often characterized as "First discretize, then optimize."



The optimization problem formulation we address in this chapter typically read as (but are not limited to):

$$\begin{aligned}
\underset{x(.),u(.)}{\text{minimize}} \quad & \int_0^T L(x(t),u(t))\,dt \;+\; E\left(x(T)\right) \\
\text{subject to} \quad & x(0) - x_0 = 0, \;\; \text{(initial value)}, \\
& \dot{x}(t) - f\left(x(t),u(t)\right) = 0, \;\; \text{(system dynamics)}, \\
& h(x(t),u(t)) \leq 0, \;\; \text{(path constraints)}, \\
& r\left(x(T)\right) = 0 \;\; \text{(terminal constraints)}.
\end{aligned}$$

For many OCPs, the system state derivatives $\dot{x}(t)$ are provided via an implicit function, or even via a Differential-Algebraic Equation (DAE). The methods presented hereafter are applicable to all these cases with some minor modifications. The direct methods differ in how they transcribe this problem into a finite NLP. The optimal control problem above has a fixed initial value, which simplifies in particular the single shooting method, but all concepts can in a straightforward way be generalized to other OCP formulations with free initial values, or other variants.

All shooting methods use an embedded ODE solver in order to eliminate or dicretize the continuous time dynamic system. They do so by first parameterizing the control function $u(t)$, e.g. by polynomials, by piecewise constant functions, or, more generally, by piecewise polynomials. We denote the finite control parameters by the vector q, and the resulting control function by $u(t,q)$. The most widespread parameterization are piecewise constant controls, which we have encountered as zero-order hold in the LMPC chapter. We choose a fixed time grid $0 = t_0 < t_1 < \ldots < t_N = T$, and N parameters $q_i \in \mathbb{R}^{n_u}$, $i = 0, \ldots, N-1$, and then we set

$$u(t,q) = q_k \quad \text{for} \quad t \in [t_k, t_{k+1}).$$

Thus, the discretized control vector $q = (q_0, \ldots, q_{N-1})$ is of dimension Nn_u.

5.4.1 Direct Single Shooting

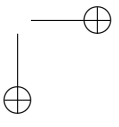
In single shooting, we regard the states $x(t)$ on $[0,T]$ as dependent variables that are obtained by a forward integration of the dynamic system using an embedded ODE solver, e.g., a Runge Kutta method, starting at x_0 and using the controls input $u(t,q)$. We denote the resulting trajectory as $x(t,q)$.

In order to discretize inequality path constraints, we choose a grid, typically the same as for the control discretization, at which we check the inequalities. Thus, in single shooting, we transcribe the optimal control problem into the following NLP, that is visualized in Figure 5.5.

$$\begin{aligned}
\underset{q \in \mathbb{R}^{Nn_u}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} L(x(t_k,q),q_k)\Delta t + E(x(t_N,q)) \\
\text{subject to} \quad & h(x(t_k,q),q_k) \leq 0, \quad k = 0,\ldots,N-1 \;\; \text{(path constraints)}, \\
& r\left(x(t_N,q)\right) = 0 \qquad\qquad\qquad\quad \text{(terminal constraints)}.
\end{aligned}$$

Here, $\Delta t = t_{k+1} - t_k$ the uniform grid step length. Note that we have used a discretized approach for the objective function using the *rectangular method* to approximate the integral. Often, a better solution is to use an integrator to compute the cost function (as we will do in the multiple shooting method).

Compared to the multiple shooting approach, which we will soon see, this is a *reduced problem* with much less variables. The ODE model is implicitly satisfied by definition of the forward simulation, and is not anymore a constraint of the optimization problem. The only remaining optimization variables are the discretized control vector q (and the initial value x_0 if it is not fixed). NLP problem can now be addressed again by Newton-type methods. Compared to the following multiple shooting approach, the exploitation of sparsity in the problem is less important. This is called the *sequential* approach, because the simulation problem and optimization problem are solved sequentially, one after the other. Note that the user can observe during all iterations of the optimization procedure what is the resulting state trajectory for the current iterate, as the model equations are satisfied by definition.



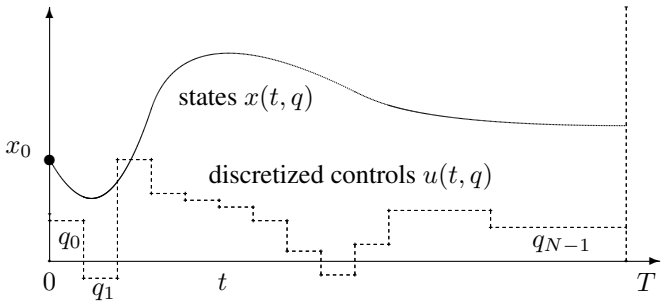


Figure 5.4: The NLP variables in the direct single shooting method.

Example. Let us illustrate the single shooting method using the following simple OCP:

$$\begin{array}{lll} \underset{x(.),u(.)}{\text{minimize}} & \displaystyle\int_0^5 x_1(t)^2 + 10x_2(t)^2 + u(t)^2\,dt & \\ \text{subject to} & \dot{x}_1(t) = x_1(t)\,x_2(t) + u(t), & x_1(0) = 0, \\ & \dot{x}_2(t) = x_1(t), & x_2(0) = 1, \\ & u(t) \geq -1, \qquad x_1(t) \geq -0.6, & t \in [0,\ T]. \end{array} \tag{5.3}$$

The resulting solution is illustrated in Figure 5.5, together with the sparsity patterns of the Jacobian of the inequality constraint function $x_1(t) \geq -0.6$, i.e.

$$\frac{\partial}{\partial q} h(x(t_i, q), u(t_i, q)),$$

and the one of the Hessian of the Lagrange function.

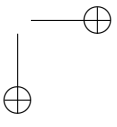
Nonlinearity propagation in direct single shooting Unfortunately, when deploying single shooting in the context of direct optimal control a difficulty can arise from the nonlinearity of the "simulation" function $x(t, q)$ with respect to the control inputs q for a large simulation time t.

Example. We illustrate this problem using the following example:

$$\dot{x}_1 = 10\,(x_2 - x_1) \tag{5.4a}$$
$$\dot{x}_2 = x_1\,(q - x_3) - x_2 \tag{5.4b}$$
$$\dot{x}_3 = x_1 x_2 - 3x_3 \tag{5.4c}$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ and $q \in \mathbb{R}$ is a constant control input. Note that the nonlinearities in this ODE result from apparently innocuous bilinear expressions. We are then interested in the relationship $q \to x(t, q)$ for different values of t. The initial conditions of the simulation were selected as $x(0) = (0, 0, 0)$ and $q \in [18,\ 38]$. The resulting relationship is displayed in Fig. 5.6. One can observe that while the relationship is not very nonlinear for small integration times t, it becomes extremely nonlinear for large times t, even though the ODE under consideration here appears simple and mildly nonlinear.

The function $x(t, q)$ resulting from the simulation of nonlinear dynamics can be extremely nonlinear. As a result, functions such as the constraints and cost function in the NLP resulting form the discretization of an optimal control problem via single-shooting can be themselves extremely nonlinear functions of the input sequence q. Because most NLP solvers proceed to find a candidate solution via taking successive linearization of the KKT conditions of the problem at hand, the presence of very nonlinear functions in the NLP problem typically invalidates these approximations outside of a very small neighborhood of the linearization point.



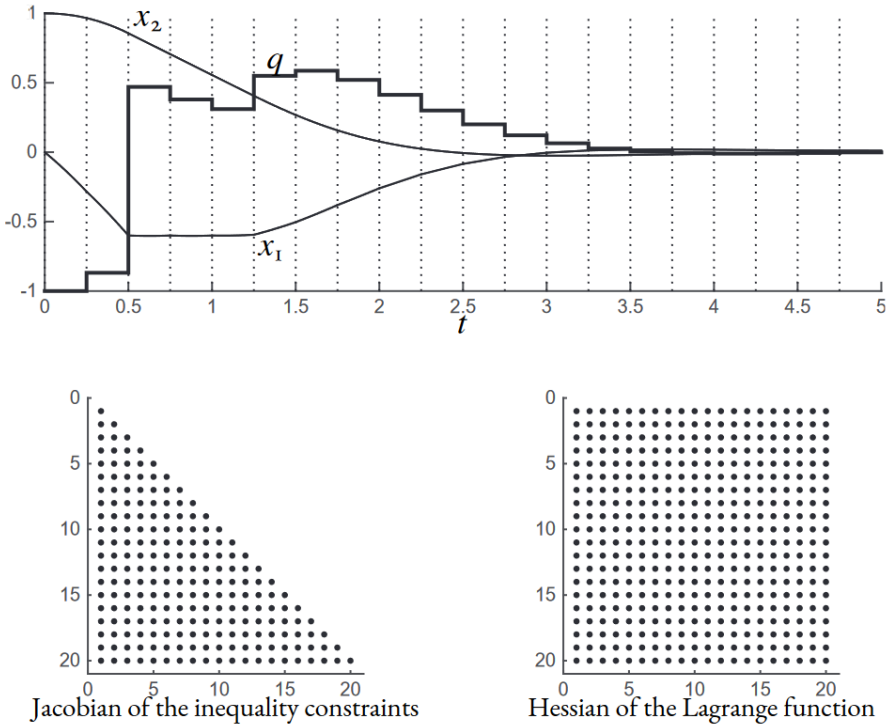


Figure 5.5: Solution to OCP (5.3) using a discretization based on single shooting, with $N = 20$ and using a 4-steps Runge-Kutta integrator of order 4. The upper graph reports the states and input trajectories. The lower graphs report the sparsity pattern of the Jacobian of the inequality constraints $h(x(t), u(t)) \leq 0$ in the resulting NLP and the sparsity pattern of the Hessian of the Lagrange function.

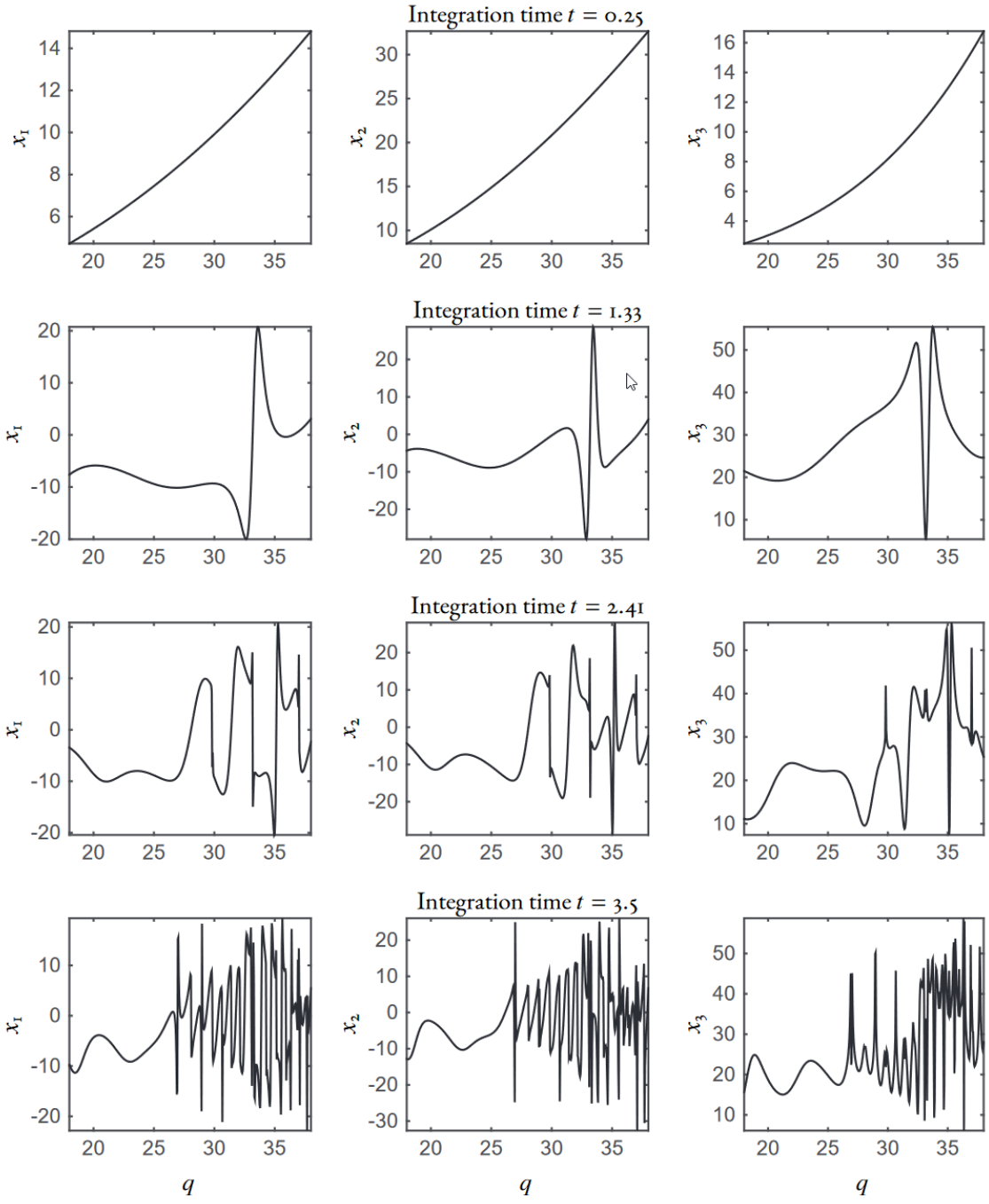


Figure 5.6: Illustration of the propagation of nonlinearities in the simple dynamic system (5.4). One can observe that for a short integration time $t = 0.25$ (first row), the relationship $q \to x(t, q)$ is close to linear. However, as the integration time increases to $t = 1.33, 2.41, 3.5$, the relationship $q \to x(t, q)$ becomes extremely nonlinear. While the effect of integration time is not necessarily as dramatic as for this specific example, large integration times yield strong nonlinear relationship $q \to x(t, q)$ for many nonlinear dynamics.

These observations entails that in practice, when using single-shooting, a very good initial guess for q is often required. For many problems, such an initial guess is very difficult to construct. As in the context of indirect methods, these observations motivate the use of alternative transcription techniques.

5.4.2 Direct Multiple Shooting

The idea behind the direct multiple-shooting approach, which is considered a *simultaneous approach*, stems from the observation that performing long integration of dynamics can be counterproductive for discretizing continuous optimal control problems into NLPs, and tackles the problem by limiting the integration over shorter time intervals.
In contrast to single shooting, the ODE is solved separately on each discretization interval $[t_k, t_{k+1}]$, starting with artificial initial values s_k:

$$\begin{aligned} \dot{x}(t, s_k, q_k) &= f\left(x(t, s_k, q_k), q_k\right), \quad t \in [t_k, t_{k+1}], \\ x(t_k, s_k, q_k) &= s_k. \end{aligned}$$

See Figure 5.7 for an illustration. Thus, we obtain trajectory pieces $x(t, s_k, q_k)$. Likewise, we numerically compute the integrals

$$l_k(s_k, q_k) := \int_{t_k}^{t_{k+1}} L\left(x(t, s_k, q_k), q_k\right) \mathrm{d}t.$$

The problem of piecing the trajectories together, i.e. ensuring the *continuity condition*

$$s_{k+1} = x(t_{k+1}, s_k, q_k)$$

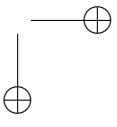
is left to the NLP solver. Finally, we choose a time grid on which the inequality path constraints are checked. It is common to choose the same time grid as for the discretization of the controls as piecewise constant, such that the constraints are checked based on the artificial initial values s_k.
The NLP arising from a discretization of an OCP based on multiple shooting typically reads as:

$$\begin{aligned} \underset{s,\,q}{\text{minimize}} \quad & \sum_{k=0}^{N-1} l_k(s_k, q_k) \;+\; E\left(s_N\right) & & \\ \text{subject to} \quad & x_0 - s_0 = 0, & & \text{(initial value)}, \\ & x(t_{k+1}, s_k, q_k) - s_{k+1} = 0, & k = 0, \ldots, N-1 & \text{(continuity)}, \\ & h(s_k, q_k) \leq 0, & k = 0, \ldots, N-1 & \text{(path constraints)}, \\ & r\left(s_N\right) = 0 & & \text{(terminal constraints)}. \end{aligned} \tag{5.5}$$

It is visualized in Figure 5.7.
Note that by defining $f(s_k, q_k) := x(t_{k+1}, s_k, q_k)$, the continuity conditions can be interpreted as discrete time dynamic system $s_{k+1} = f(s_k, q_k)$ and the above optimal control problem has exactly the same structure as the discrete time optimal control problem (5.2) discussed in the beginning of this chapter. The optimization variables are $z = (s_0, q_0, s_1, q_1, \ldots, s_{N-1}, q_{N-1}, s_N)$.
The nonlinear program (5.5) is large and structured and can thus in principle be solved by any NLP solver. Note that in this approach, all original variables, i.e., q_k and s_k, remain optimization variables of the NLP. Its name stems from the fact that the NLP solver has to simultaneously solve both, the simulation and the optimization problem. It is interesting to remark that the model equations over the whole time horizon will for most NLP solvers only be satisfied once the NLP iterations are converged.
The sparsity structure arising from a discretization based on multiple-shooting (see Figure 5.8 for an illustration) ought to be exploited in the NLP solver in order to be efficient.

Example. Let us tackle the OCP (5.3) of Example 5.4.1 via direct multiple-shooting. A 4-step RK4 integrator has been used here, deployed on $N = 20$ shooting intervals. Here the ordering of the equality constraints and variables is important in order to get structured sparsity patterns. In this example, the variables are ordered in time as:

$$s_{1,0},\ s_{2,0},\ q_0,\ s_{1,1},\ s_{2,1},\ q_1,\ \ldots,\ q_{N-1},\ s_{1,N},\ s_{2,N}$$



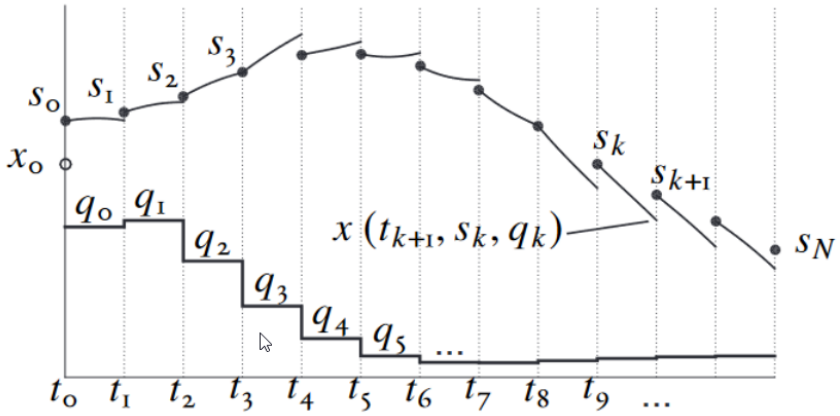


Figure 5.7: Illustration of the direct multiple shooting method. A piecewise-constant input profile parametrized by $q_{0,\ldots,N-1}$ is deployed on the time grid $t_{0,\ldots,N}$. The discrete states $s_{0,\ldots,N}$ act as "check-points" on the continuous state trajectories $x(t)$ at all discrete time points $t_{0,\ldots,N}$. Numerical integrators build the simulations $x\,(t, s_k, q_k)$ over each time interval $[t_k,\, t_{k+1}]$. The state trajectory held in the NLP solver becomes continuous only when the solution of the NLP is reached, where the continuity conditions $x\,(t_{k+1}, s_k, q_k) - s_{k+1}$ are enforced.

and the constraints are also ordered in time. The resulting solution is illustrated in Figure 5.8, together with the sparsity patterns of the Jacobian of the equality constraint function, and the one of the Hessian of the Lagrange function. One can observe the discrete state trajectories (black dots) at the discrete time instants $t_{0,\ldots,N}$ together with the simulations delivered by the integrators at the solution. One can also observe the very specific sparsity patterns of the Jacobian of the equality constraints and of the Hessian of the Lagrange function that arise from the direct multiple-shooting approach.

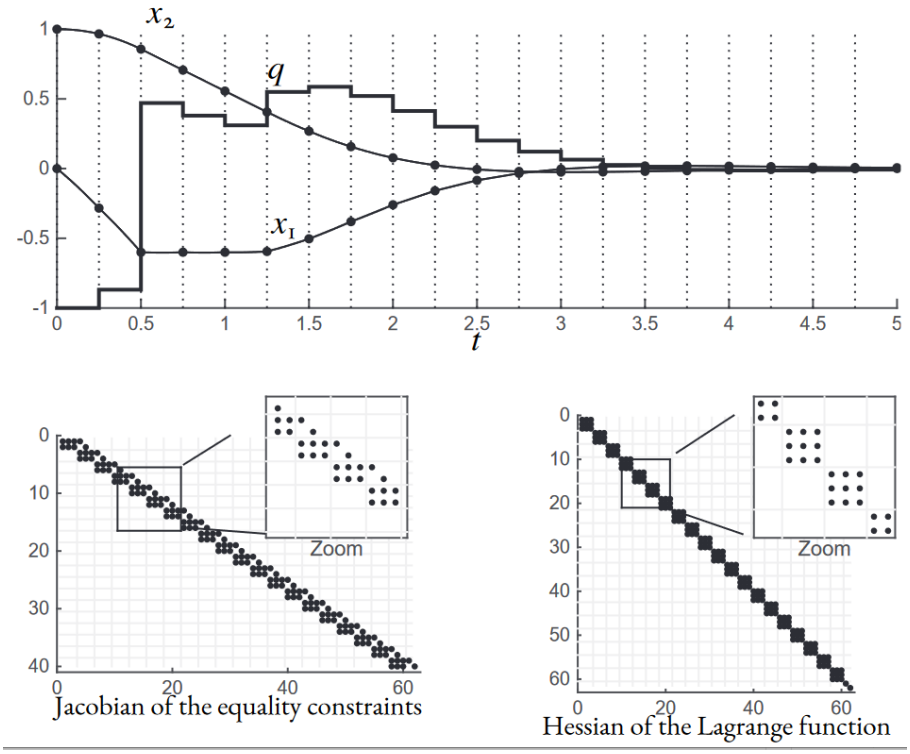


Figure 5.8: Solution to OCP (5.3) using a discretization based on multiple shooting, with $N = 20$ and using a 4-steps Runge-Kutta integrator of order 4. The upper graph reports the states and input trajectories at the solution, where the continuity condition holds. The lower graphs report the sparsity pattern of the Jacobian of the equality constraints (the continuity conditions) in the resulting NLP and the sparsity pattern of the Hessian of the Lagrange function. The Hessian of the Lagrange function arising from multiple-shooting is block-diagonal, due to the separability of the Lagrange function. The Jacobian of the equality constraints is diagonal in this example, and block-diagonal in general.

5.5 Practical aspects of MPC

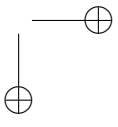
5.5.1 Soft Constraints

Constraints play an important role in MPC optimization problems, however, introducing constraints into the optimization problem can lead to infeasibility at specific time steps. In such scenarios, the feasible set for the optimization problem is empty and thus there is no suitable solution. Infeasibility can also arise due to disturbances, discrepancies between the model and the actual system, and inappropriate selection of MPC tuning parameters, such as setting the prediction horizon too short.
If constraints are only applied to the actuated values, and these constraints are set appropriately – i.e., the lower limit is less than the upper limit for box constraints – infeasibility does not become an issue. The feasible set is then directly determined by the constraint set of the actuated variables. Consequently, the actuated variable constraints can always be formulated exactly as *hard constraints*. For instance, a valve cannot exceed 100% open. However, when constraints are applied to system states or outputs, potential feasibility issues may arise. Feasibility issues arise with constraints on system states or outputs because these are influenced by the system's dynamics and the control inputs, making them interdependent and potentially conflicting. Constraints on just control inputs are more straightforward as they don't depend on the system's response, ensuring direct enforceability. Moreover, the feasibility of a single MPC step does not assure the feasibility of all subsequent steps, a characteristic known as recursive feasibility.
The prevalent strategy for managing infeasibility in practical scenarios involves using *soft constraints* rather than hard ones. The concept behind soft constraints is to permit minor violations of the original hard constraints, offset by imposing a significant cost in the cost function. An appropriate penalty function is added to the original cost function. With this enhanced cost function, the controller aims to minimize the violation of the original hard constraints. Ideally, when the original optimization problem is feasible, the optimizer of the modified optimization problem should yield the same solution. This is known as the 'exact penalty' property of the penalty function. Soft constraints also better represent real-world scenarios, as constraints on system states and outputs are generally not hard. The building room teperature, as an example of building heating control, should be kept above a certain comfort limit value. However, surpassing this limit value for a short time and with low excess is tolerable. The concept of soft constraints is detailed in the following. This section is based on [9, Chapter 6.1].

Original Optimization Problem We consider a generic OCP of the form:

$$
\begin{aligned}
&\min_{x,u} \quad J_{\text{orig}} \\
&\text{s.t. system dynamics,} \\
&\text{initial conditions,} \\
&u_{\min} \leq u(t) \leq u_{\max}, \\
&x_{\min} \leq x(t) \leq x_{\max}, \\
&\text{other constraints (hard constraints).}
\end{aligned}
$$

The hard constraints on the system states can be relaxed by the introduction of the slack variables s. Within the augmented optimization problem, the slack variables are considered as additional optimization variables which enter the cost function as well as the constraints. Of course, for a high number of constraints to be softened, many new optimization variables have to be introduced which increase the necessary computation time. In the following, various choices for the penalty function are discussed. Weighting matrices with positive values are used for appropriate penalization. In general, the location of a minimizer in an optimization problem can shift as a result of changes to the cost function, including changes brought about by the addition of a penalty term. The purpose of the penalty function is to "penalize" solutions that violate the constraints of the problem, thus driving the optimization algorithm towards solutions that meet the constraints.
The weighting factors in the penalty function determine the strength or severity of this penalty. If the weights are too low, the penalty for violating the constraints might not be severe enough to prevent the algorithm from finding a "cheaper" (in terms of the cost function) but less optimal solution that violates the constraints.



Quadratic penalty on the slack variables The original cost function is augmented by the penalty function J_{pen}, resulting in $J_{\text{aug}} = J_{\text{ocp}} + J_{\text{pen}}$. First, a quadratic penalty is applied. The resulting optimization problem is:

$$
\begin{aligned}
&\min_{x,u,s} \; J_{\text{OCP}} + c_q \int_0^{t_\text{f}} s^2(t)\,\mathrm{d}t \\
&\text{s.t. system dynamics,} \\
&\text{initial conditions,} \\
&u_{\min} \leq u(t) \leq u_{\max}, \\
&x_{\min} - s(t) \leq x(t), \\
&-x_{\max} - s(t) \leq -x(t), \\
&s(t) \geq 0, \\
&\text{other constraints (hard constraints).}
\end{aligned}
$$

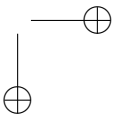
One advantage of the quadratic penalty function is that if the original optimization problem is strictly convex, the augmented optimization problem stays strictly convex and the Hessian remains positive definite. The quadratic penalty function does not introduce any exact penalties, which implies that the optimizer can be shifted even though there exists a solution without any violation of constraints.

Linear Penalty Function Instead of the 2-norm, the 1 -norm can be used to penalize s. The 1-norm penalizes the sum of the absolute values of the slack variables. For weighting of the penalties, $c_l \in \mathbb{R}^{n_x}$ (here for simplicity we assume $n_x = 1$) is used. As the slack variables are defined to be nonnegative, the following optimization problem can be used:

$$
\begin{aligned}
&\min_{x,u,s} \; J_{\text{OCP}} + c_l \int_0^{t_\text{f}} s^2(t)\text{t} \\
&\text{s.t.} \quad \text{system dynamics,} \\
&\text{initial conditions,} \\
&u_{\min} \leq u(t) \leq u_{\max}, \quad i = 0, \ldots, N_u - 1 \\
&x_{\text{min}} - s(t) \leq x(t), \quad , \\
&-x_{\max} - s(t) \leq -x(t), \\
&s(t) \geq 0, \\
&\text{other constraints (hard constraints).}
\end{aligned}
$$

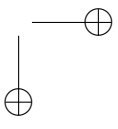
The main advantage of using a linear penalty term by the 1-norm is the fact that exact penalties can be obtained. The additional linear penalty function introduces a gradient of c_l. The weightings c_l of the slack variables just need to be chosen sufficiently high, such that the gradient of the penalty function is large enough. By making the weights and therefore the gradient of the penalty function at the original constrained minimizer sufficiently large, we ensure that any shift in the location of the minimizer due to the addition of the penalty term is minimized. The large gradient represents a strong pull or drive towards the original minimizer, which can help to maintain its location even when the penalty term is added to the cost function. When a feasible solution of the original problems exists, the location of the local minimizer thus is not changed by the augmented optimization problem.
In the case of adding the linear penalty function, it has to be considered that even if the original problem is strictly convex, the resulting optimization problem no longer is strictly convex. When QP solvers are applied that rely on strictly convex optimization problems, suitable regularization has to be added.

Quadratic Plus Linear Penalty Function The combination of a quadratic and a linear penalty term combines the advantages of both formulations. The combination sustains a strictly convex optimization problem. Additionally, for sufficiently high values of c_l, exact penalties are obtained. In practice, the combination of quadratic and linear penalization functions is used most widely. One example is shown in the following optimization problem where the maximum deviation, a scalar value, over the entire prediction horizon and all states is penalized.



$$
\begin{aligned}
&\min_{x,u,s} \quad J_{\mathrm{OCP}} + \int_0^{t_\mathrm{f}} c_q s^2(t) + c_l s(t) \mathrm{t} \\
&\text{s.t.} \\
&\text{system dynamics,} \\
&\text{initial conditions,} \\
&u_{\min} \leq u(t) \leq u_{\max}, \\
&x_{\min} - s \leq x(t), \\
&-x_{\max} - s \leq -x(t), \\
&s \geq 0 \\
&\text{other constraints (hard constraints).}
\end{aligned}
$$

These penalty functions can be used for both LMPC and NMPC to soften the constraints. In order to not shift the location of the local minimizer, the weighting factors have to be chosen such that the gradient of the penalty function at the original constrained minimizer is sufficiently large. However, the addition of the penalty function can lead to new minima in regions that previously were excluded by the constraints. In general, new local minimizers that exceed the original constraints can be found much easier for NMPC compared to LMPC. Hence, the NLP solver can find unwanted local minima outside of the operating region. This has to be considered when developing NMPC with soft constraints. This affects for instance the process model. It needs to be suitable for usage within NMPC even outside of the feasible region defined by the original hard constraints. This is especially true for data-driven models as their extrapolation capabilities are limited.



Chapter 6

Model predictive control of wind energy systems

6.1 Control task of wind turbines

6.1.1 Control objectives

The control objectives depend on the given wind speed (see Fig. 2.21):

- **Partial load:** Here the control goal is to maximize power extraction while respecting operational limits (Region IIB).
- **Full load:** Here the main control goal is to alleviate dynamic mechanical loads and maintain operational limits, including maximum rotational speed and electric power limits (Region III).
- **Transition:** A smooth transition is required between these two regions (Region IIC).

Industrial practice of wind turbine control The industrial practice in wind energy systems meets the control objectives by tackling each objective (in the partial- or the full-load region respectively) with a separate controller. The transition between controllers is handled via elaborate and carefully tuned heuristics. Both controllers typically have different parametrizations for varying wind speeds to account for the varying sensitivities.

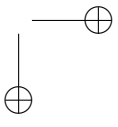
- **Baseline controller:** The baseline controller operates in the *partial load* region and mainly actuates the generator torque to follow the pre-defined optimal torque-speed curve. The pitch angle is typically actuated only little or not at all.
- **Load alleviating controller:** This controller operates in the *full load* region and mainly actuates the blade pitch angle to curtail the generated power and to counteract the tower oscillations. The generator torque can be used to dampen drive train oscillations.

In addition, many heuristic rules and filters are necessary in order to satisfy constraints and to account for cross-coupling of in- and outputs. These heuristic algorithms make up the largest part of the developed code! Controller and heuristics tuning needs to be done manually, and has to be more or less re-done before application on a new wind turbine variant, as it is not derived directly from model knowledge.

6.2 MPC design of wind turbines

6.2.1 LTV-MPC

Although the reduced-order wind turbine model is nonlinear in the aerodynamic submodel, a linear MPC scheme has thus far been preferred over nonlinear MPC approached. This is because nonlinear MPC is



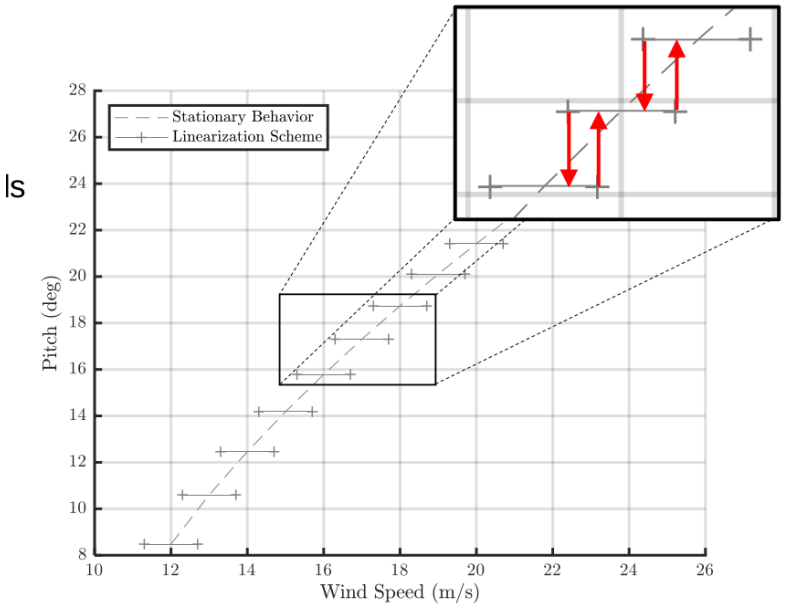


Figure 6.1: Linear models along the optimal WT operating curve and switching rule based on "hysteresis". Courtesy by T. Wintermeyer-Kallen.

associated with a larger computational cost and unpredictable solution times. Moreover, there are no hard convergence guarantees and convergence-enhancing strategies such as globalization come at an additional computational cost.

In order to control the system over the full nonlinear operating region, an obvious idea to use different linear MPC schemes corresponding to different linearizations of the system dynamics for different operating points that are well-known in advance. In order to avoid undesired oscillations between models, the switching rule can be based on a hysteresis principle, as illustrated by Fig. 6.1. The problem with this approach is that during large wind speed disturbances such as extreme gusts, the wind turbine is operating for a large time in an unsteady regime, where the linearizations aroudn steady operating points are a bad approximation, as illustrated in Fig. 6.2.

Therefore, in order to make the MPC scheme more robust, it is better to linearize the WT dynamics at every sampling instant at the *current (unsteady) operating point*. This approach is called a linear time-variant MPC scheme (LTV-MPC). While this scheme comes at an additional computational costs, and comes with the loss of a priori stability guarantees, it allows the stabilization of the plant also in extreme operating conditions.

The LTV-MPC scheme can then be summarized by the following steps:

1. Estimate $\hat{x}_0$ and the wind speed $\hat{v}_\mathrm{w}$.
2. Linearize at $\bar{x} = \hat{x}_0$, $\bar{u} = u_{k-1}$, $\bar{v}_\mathrm{w} = \hat{v}_\mathrm{w}$ to obtain the linear model

$$\dot{x}(t) = f(\bar{x}, \bar{u}, \bar{v}_\mathrm{w}) + \bar{A}(x(t) - \bar{x}) + \bar{B}(u(t) - \bar{u}) \tag{6.1}$$

   with

$$\bar{A} = \frac{\partial f}{\partial x}\Big|_{\bar{x},\bar{u},\bar{v}_\mathrm{w}} \quad \text{and} \quad \bar{B} = \frac{\partial f}{\partial u}\Big|_{\bar{x},\bar{u},\bar{v}_\mathrm{w}} \tag{6.2}$$

3. Discretize and use linear model over the entire prediction horizon.
4. Construct (dense) QP matrices.
5. Solve convex QP, apply u_0^* and repeat.

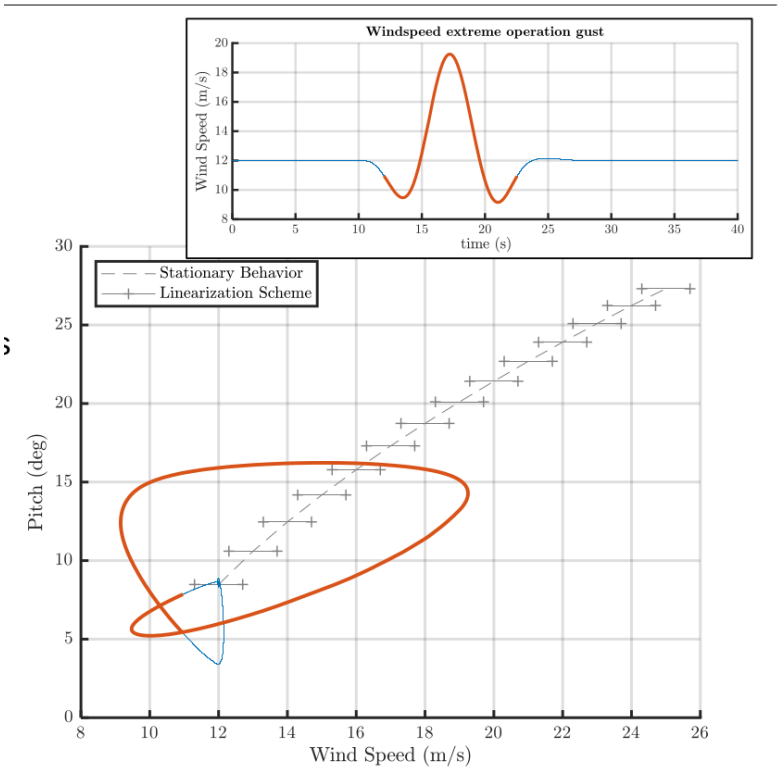


Figure 6.2: Unsteady WT behavior as a response to an extreme wind gust. Courtesy by T. Wintermeyer-Kallen.

6.2.2 MPC sampling time and horizon

The reduced-order wind turbine model consists of three different submodels with different time constants:

- Aerodynamics: static model
- Drive train dynamics: characteristic time constant $T \approx 0.3$ s.
- Rotor-tower dynamics: characteristic time constant $T \approx 4$ s.

The MPC sampling time should be chosen so that it can stabilize the fastest system dynamics. Hence the sampling time is chosen as $T_\mathrm{s} = 0.1$ s as determined by the drive train dynamics.
The horizon should be chosen so that the slowest relevant system dynamics are still considered within the prediction horizon. Thus, as determined by the rotor-tower dynamics, the horizon should be at least around 5 s, or equivalently, $N = 50$.

6.2.3 Move blocking

In order to capture the tower and rotor dynamics, the prediction horizon N has to be relatively long. This leads to a large QP with a computational load that is at the limits of real-time feasibility. In order to reduce the QP size, it is possible to introduce an additional horizon, called the "control horizon" $N_u < N$. After the control horizon, the control input is kept constant ("blocked"). Thus, in the dense formulation, the resulting QP has the size $N_u n_u$ instead of $N n_u$. The control horizon length is then chosen as low as possible without compromising closed-loop performance too much.
This approach can be generalized to the idea where the piecewise constant control actions are distributed on a non-uniform grid within the control horizon. This way, controls can be applied at a higher frequency in the beginning of the control horizon, and at a lower frequency at the end, as illustrated by Fig. 6.3

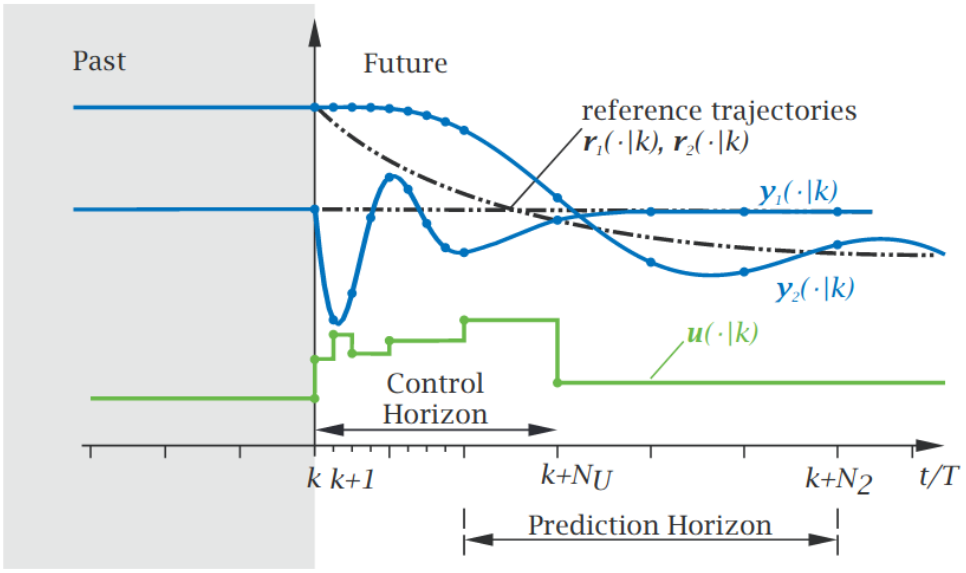


Figure 6.3: Illustrative example for different system dynamics considered within one MPC problem and the use a move-blocking strategy [6]

In the wind turbine case, for example, the control horizon can be chosen to be $N_\mathrm{u} = 10$. Three high-frequency control steps then suffice to control the drive train dynamics, followed by two lower-frequency control actions for the rotor-tower dynamics and one constant control action for the remainder of the total prediction horizon.

6.2.4 Weight scheduling

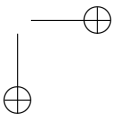
In LTV-MPC, the prediction model is linearized at the current unsteady operating point. However, the linearization is constant over the prediction horizon, even though the predicted system state might travel to a significantly different operating point, where the sensitivities of the outputs with respect to the inputs can be vastly different than assumed. This can lead to undesired closed-loop behavior.
As an alternative to full nonlinear MPC, this issue can be addressed via weight scheduling, where the weight matrices Q and R are dependent on the current operating point.
For example, in the partial load region, the system is operated so as to maximize power output for a given rotation speed, leading to $\frac{\partial c_\mathrm{P}}{\partial \theta} \approx \frac{\partial c_\mathrm{T}}{\partial \theta} \approx 0$. Away from this optimal point, the sensitivities change rapidly, but the LTV-MPC cannot take this into account.
Thus, in the linearized model, a much large change in pitch actuation is necessary to change the aerodynamic torque on the turbine, than is really needed. Since the pitch actuation is penalized in the objective, this leads to a too small pitch actuation and consequently to a too slow closed-loop response.
This problem can be countered by decreasing the weight entry of the pitch angle in the matrix R in the partial load region. In the full load region, the sensitivity w.r.t. to the pitch angle is less pronounced and its variation limited, and the weight can be chosen to be higher.
In order to have a continuous feedback law, the weight entries q_i and r_i of the matrices Q and R respectively should vary continuously, for example with the system operating point y_k. To achieve this for two different regions, a pair of limits (y_j^0, y_j^1) for a specific output y_j can defined, between which there is a transition in weight value from value $q_{i,0}$ to value $q_{i,1}$, e.g.:

$$q_i(y_k) = q_{i,0} + \left(\frac{y_{k,j} - y_j^0}{y_j^1 - y_j^0} \right)^2 (q_{i,1} - q_{i,0}) \,. \tag{6.3}$$

To change weights between full and partial load regions, the weights can also be varied with the estimated wind speed.

6.2.5 State-of-the-art experimental results

While simulative design of MPCs for wind turbines has been extensively studied, only a single study has been published on the real-world application of MPC to a full-scale wind turbine. In [4], the capability of an MPC controller is validated and tested on a 3 MW wind turbine. We refer the reader to the cited paper for more details.



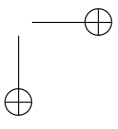
Chapter 7

Online state estimation

The upcoming topic we will discuss is state estimation. In order for optimization to be carried out with an MPC controller, all state variables of the model must be known. However, in many practical scenarios, we can only measure a small subset of the variables needed to accurately model the system. Additionally, the measurements we obtain are typically corrupted by sensor noise, the evolution of the system's state is affected by process noise, and modeling errors and system disturbances are present. Consequently, deriving a reliable state estimate that can be used effectively in the controller becomes a challenging task due to the noisy and incomplete nature of the output measurements. This challenge forms the essence of state estimation.
For instance, considering the building energy model, the state variables of the used model could be the heating system return temperature $T_{\mathrm{ret}}(t)$, the building floor temperature $T_{\mathrm{floor}}(t)$ and/or the building wall temperature $T_{\mathrm{wall}}(t)$ and the room temperature $T_{\mathrm{room}}(t)$. However, only $T_{\mathrm{room}}(t)$ is measured. To obtain the missing state variables, an observer can be used. In fact, since the building model is linear, a Kalman filter can be used. For this aim, the building model need to be fully observable (or satisfy another weaker condition called detectability) and controllable.
Aim of this chapter is to present methods that estimate the current state parameters from a series of measurements in the past. In fact, the same techniques can be applied if unknown system parameters have to be estimated. We start with the simplest scenario — a linear model influenced by normally distributed (called Gaussian) process and measurement noise. The optimal state estimator in this context is widely known as the Kalman filter (Kalman, 1960).
Exceeding the linear case and Gaussian noise, one powerful method for online state (and parameter) estimation uses the measurements on a moving time window in the past, and is called moving horizon estimation (MHE). This approach leads to optimization problems that have nearly the same structure as the optimal control problems treated earlier in this course. It is the second main topic of this chapter, and a technology often combined with nonlinear model predictive control (NMPC), with which its optimization problems share many characteristics. The typical combined observer and state feedback loop is depicted in Figure 7.1.
For the derivation of the Kalman filter and MHE , we refer to Chapters 1.4 and 4 of the MPC book by Rawlings, Mayne and Diehl [10] for further information.

7.1 Linear optimal state estimation (Kalman filter)

The Kalman filter is a recursive algorithm that is used to estimate the evolving state of a system in the presence of noise. It can be applied to any system that can be modeled by linear equations.
The basic idea is to maintain an ongoing estimate of the state of the system, and then to continuously correct that estimate based on new measurements. This is done in a two-step process, involving a prediction step and an update step.
We summarize the equations of the Kalman filter for a time-discrete linear system. The system model consisting of the state vector x_k and the output vector y_k representing the measurement, can be expressed in the following form:



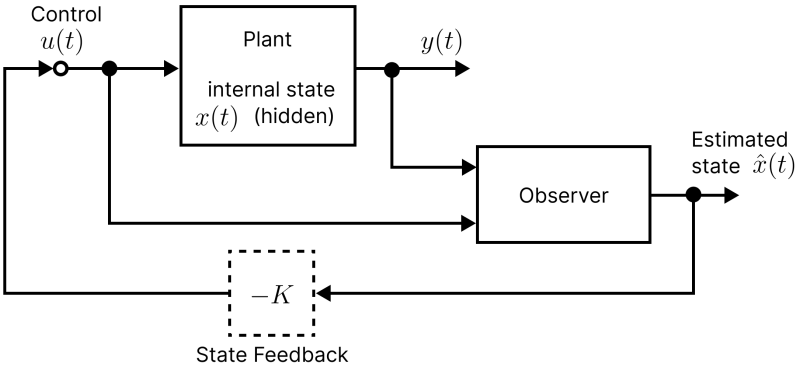


Figure 7.1: Combined observer and feedback control loop

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{7.1}$$
$$y_k = Cx_k + v_k \tag{7.2}$$

where

- x_k is the state vector at time k, u_k is the control vector at time k, y_k is the measurement vector at time k.
- A is the state transition matrix, B is the control-input matrix, C is the observation matrix.
- w_k is the process noise which is assumed to be Gaussian with zero mean and covariance Q ($w \sim \mathcal{N}(0, Q)$).
- v_k is the measurement noise which is assumed to be Gaussian with zero mean and covariance R ($v \sim \mathcal{N}(0, R)$).

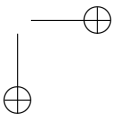
In fact, the auto-covariance matrices of the process and measurement error are given by

$$E\left\{w_k w_j^T\right\} = Q \cdot \delta_{kj} \qquad k, j \geq 0$$
$$E\left\{v_k v_j^T\right\} = R \cdot \delta_{kj} \qquad k, j \geq 0$$

Here, Q must be symmetric and positive semi-definite and R symmetric and positive-definite.
The first equation (7.1) signifies that the system state at time $k+1$ is a linear transformation of the state at time k, represented by matrix A, plus the effect of any control inputs (Bu_k), as well as the impact of any process noise (w_k). The second equation (7.2) represents the measurement at time $k+1$, which is a linear transformation of the state at time k, represented by matrix C, plus measurement noise (v_k). The input variables and output variables are superimposed with white noise.
For the Kalman Filter to perform optimally, we assume the system to be both observable and controllable (see Section 4.2.3 for definitions):

- Observability: For a Kalman Filter to work, it is crucial that it can determine the state of the system. If the system is not observable, then there might exist states that cannot be inferred from the measurements, which means the filter might not be able to accurately predict the future states or correct its estimates based on the measurements. If this condition is not fulfilled, the estimates it provides for the unobservable states may not converge to their true values, and thus would be inaccurate.

- Controllability: While controllability is not a strict requirement for a Kalman Filter to operate, it's important for the filter to be effective in control scenarios. If a system is not controllable, then no matter how good our estimate of the system state is, we may not be able to apply inputs to achieve a desired state.



These two conditions simplify the analysis. In fact, weaker conditions such as detectability can be imposed and controllability is not strictly necessary. Readers who are keen on delving deeper into this subject are encouraged to refer to the suggested literature
To sketch out the derivation of the Kalman filter, we will employ the following notation:

- $\hat{x}_{k|k-1}$ is the *priori predicted state vector* (predicted without knowing the current measurement y_k).
- $\hat{x}_{k|k}$ is the *posterior predicted state vector* (updated/filtered estimate).
- K is the *Kalman gain*, controlling the contribution of the measurement difference in the posterior prediction.
- $y_k - C\hat{x}_{k|k-1}$ is the difference between the actual and predicted measurements.
- $\Sigma_{k|k}$ is the state estimate error covariance matrix at time k after updating the Kalman Filter with all measurements through time k. That is, it is the error covariance for the updated/filtered state estimate.
- $\Sigma_{k|k-1}$ is the state estimate at time k after updating the Kalman Filter with all but the most recent measurement. That is, it is the error covariance for the predicted state estimate.

The Kalman filter calculates the estimate $\hat{x}_{k|k}$ of the real state vector x_k such that the estimate does not have a systematic error at each time point k and the estimation error covariance matrix $\Sigma_{k|k}$ is "minimal."

$$\Sigma_{k|k} = E\left\{\left[x_k - \hat{x}_{k|k}\right]\left[x_k - \hat{x}_{k|k}\right]^T\right\}$$

The solution to the optimization problem is given by the following algoritmic procedure. The Kalman Filter equations can be broken down into two main steps: the prediction step and the update step.

1. Initialization: Start with an initial guess of what the state might be along with an initial error covariance. This is the best guess of the state of the system and the error associated with it.

$$\begin{aligned} \hat{x}_0 &= x_{\text{initial}} && \text{// Initial state estimate} \\ \Sigma_0 &= \Sigma_{\text{initial}} && \text{// Initial error covariance} \end{aligned}$$

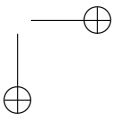
2. Prediction: Given a model of the system (how it changes over time without any measurements), predict the state and the error covariance at the next time step. This is often called the *prior estimate*. Since the prediction is without new data, the uncertainty (error covariance) increases.

$$\begin{aligned} \hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_{k-1} && \text{// Predict state (prior)} \\ \Sigma_{k|k-1} &= A\Sigma_{k-1|k-1}A^T + Q && \text{// Predict error covariance (prior)} \end{aligned}$$

3. Update: When a new measurement arrives, you correct the predicted estimate to form an "updated estimate" or "posterior estimate". Update the predicted state using the measurement residual and update the error covariance.

    - With the measurement data at time step k, the estimate is corrected using the Kalman gain, thus providing the optimal estimate of the state variables $\hat{x}_{k|k}$ needed for optimization. To this aim, calculate a weighted average of the predicted estimate and the new measurement. The weights are given by the Kalman gain, which determines how much trust to place in the new measurement versus the prediction from the model. If the model's predictions are very accurate, you'd put more weight on the predicted estimate; if the measurements are more accurate, you'd put more weight on the measured data.
    - Update the model's prediction: update the error covariance (how much uncertainty there is in the estimate) based on the difference between the prediction and the actual measurement (the "innovation" or "residual").

$$\begin{aligned} K_k &= \Sigma_{k|k-1}C^T(C\Sigma_{k|k-1}C^T + R)^{-1} && \text{// Compute Kalman Gain} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) && \text{// Update state estimate (posterior)} \\ \Sigma_{k|k} &= (I - K_kC)\Sigma_{k|k-1} && \text{// Update error covariance (posterior)} \end{aligned}$$



By rewriting the error covariance update equation and substituting in for the Kalman gain, we obtain the discrete-time Riccati equation (DARE), wich arises also in discrete-time optimal control problems and the LQR:

$$\Sigma_{k|k} = A\Sigma_{k-1|k-1}A^T - A\Sigma_{k-1|k-1}C^T(C\Sigma_{k-1|k-1}C^T + R)^{-1}C\Sigma_{k-1|k-1}A^T + Q$$

The estimation error covariance matrix $\Sigma_{k|k}$ is the only stabilizing solution to the DARE. Since the model used is observable and controllable, $\Sigma_{k|k} \equiv \Sigma$ can be set equal to $\Sigma_{k-1|k-1}$. This holds true because in an observable and controllable system, all state information can be extracted from the output measurements, and all states can be influenced by control inputs. Therefore, if the model accurately describes the dynamics of the system, the uncertainty about the system's state, as expressed by the error covariance matrix, would not increase over time.
The Kalman filter is intended to determine the value of the state variables as precisely as possible at each optimization time point. Therefore the sampling time is often set smaller than the sampling time for the control problem. In building heating control, for instance, in order to take into account the effects of switching the heat pump on and off, the observer sampling time is set to 60 seconds while the control sampling time may be set to 15 minutes.
The behavior of the filter can be adjusted with the intensity matrices Q and R.
The beauty of the Kalman filter is that it doesn't need to store and process all the data points from the history of the system. It maintains an ongoing, updated state estimate and uncertainty, and only needs the latest measurement to update this state. This property is what makes it "recursive" and extremely useful for real-time applications.

Extensions The Kalman Filter is used for linear systems with Gaussian noise. For non-linear systems, variants like Extended Kalman Filter (EKF) are used. The EKF handles nonlinearity by linearizing the process and measurement models at the current estimate of the state, and then it applies the standard Kalman Filter to this linearized model. Similar to the iterative linearization of a nonlinear process model in LTV-MPC, this approach has several limitations. The accuracy of the EKF can be poor if the system is highly non-linear, or if the initial state estimate is far from the true state.

7.2 Moving horizon estimation

When we work with nonlinear models or apply constraints on our estimates, we cannot compute the conditional density in a closed form, step-by-step manner, as was the case in Kalman filtering. To derive the otimal state estimate, we would need to simultaneously optimize all the states present in the trajectory x_T to derive the state estimates. However, as T enlarges, this optimization task proves to be computationally intractable. Moving horizon estimation (MHE) removes this issue by considering only the most recent N measurements and finds only the most recent N values of the state trajectory as depicted in Figure 7.2.
MHE is an optimization-based state-estimation technique where *the current state of the system is inferred based on a finite sequence of past measurements*. In many ways it can be seen as the counterpart to MPC – similar to the Kalman filter that can be seen as counterpart to LMPC. Traditional filtering methods such as the Kalman Filter are optimal under the assumption of linearity and Gaussian noise. MHE does not require these assumptions. Additionally, MHE is better suited to handle constraints on states and inputs than standard filtering techniques.

Formulation of MHE Problem The MHE algorithm can be formulated for nonlinear dynamic systems with state variables x, inputs u and measureable outputs y, which we consider of the following form for the continurous-time case:

$$\dot{x}(t) = f(x(t), u(t)) + w(t), \tag{7.3}$$
$$y(t) = h(x(t), u(t)) + v(t), \tag{7.4}$$

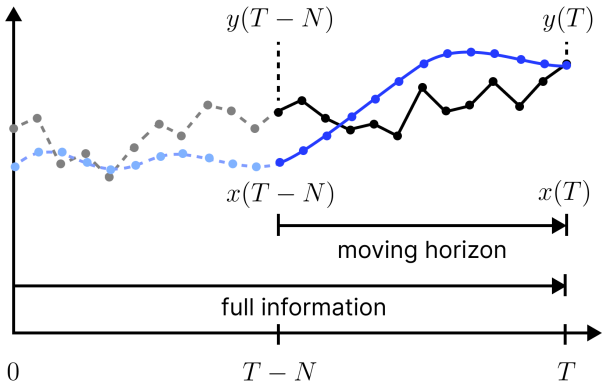


Figure 7.2: Moving horizon estimation (online) vs. full information estimation [10].

and for the discrete-time case by

$$x_{k+1} = f(x_k, u_k) + w_k, \tag{7.5}$$
$$y_k = h(x_k, u_k) + v_k. \tag{7.6}$$

Note that instead of additive noise, we could have also defined the noise in the process and measurement model. To simplify the notation, we consider the discrete-time case, e.g. after discretization with direct multiple shooting. The states that we're aiming to estimate are $x_N(T) = (x_{T-N}, \dots, x_T)$, given the measurement sequence $y_N(T) = (y_{T-N}, \dots, y_T)$. In this context, we disregard the initial phase when the estimation window is still gathering the first measurements and smaller than N and assume that the window is always full.
The principle of MHE is to solve a finite-horizon optimization problem at each time step. Given a window of past measurements and a system model, MHE minimizes the discrepancy between the predicted outputs (from the model) and the actual measurements, along with the model discrepancy over a certain finite horizon. The optimization problem is formulated to estimate the states and/or parameters of the system. The "moving horizon" aspect comes into play as new measurements are received. The horizon (the window of past measurements considered) shifts forward in time, discarding the oldest measurement and including the newest one, hence "moving horizon".
Following this concept, we formulate the MHE optimization problem as minimization problem with a least-squares cost function:

$$J_{\text{MHE}}\left(x_N(T), u_{N-1}(T)\right) = \frac{1}{2}\|x_{T-N} - \tilde{x}_{T-N}\|^2_{\Sigma_x} + \sum_{k=T-N}^{T-1} ||y_k - h(x_k, u_k)||^2_Q + ||u_k - \tilde{u}_k||^2_R$$

$$\min \quad J_{\text{MHE}}\left(x_N(T), u_{N-1}(T)\right) \tag{7.7}$$
$$\text{s.t.} \quad \left.\begin{aligned} x_{k+1} &= f(x_k, u_k), \\ g(x_k, u_k) &\leq 0 \end{aligned}\right\} \; k = T-N, \dots, T-1. \tag{7.8}$$

Here, we use the same notation as before, with the additions:

- N is the horizon length.
- $\tilde{u}_k$ are the fixed control inputs.
- f is the system dynamics function, h is the measurement output function, g are the constraints.
- The process and measurement noise w_k and v_k does not necessarily be Gaussian.

- $||.||_Q^2$ and $||.||_R^2$ represent weighted square norms (where, e.g., $\|x\|_Q^2 = x^T Q x$), which account for the uncertainty in measurements and model prediction, respectively.

The fundamental idea of MHE is that the current state of the system is inferred based on a finite sequence of N past measurements, while incorporating information from the dynamic system equation. This is formulated as an optimization problem, where the finite sequence of states and inputs are optimization variables. These sequences are determined, such that

- The initial state x_{T-N} of the sequence is coherent with the previous estimate $\tilde{x}_{T-N}$ (if applicable).
- The computed measurements match the true measurements very closely.
- The computed control match the given inputs very closely.
- The dynamic state equation is obeyed.

Similarly to MPC, the MHE optimization problem is solved repeatedly at each sampling instance. At each estimation step, the new initial state is the second element from the previous estimation and we take into consideration the newest measurement while dropping the oldest. This can be seen in the figure below, which depicts the successive horizon.
The first term in the least-squares cost function

$$\frac{1}{2}\|x_{T-N} - \tilde{x}_{T-N}\|_{\Sigma_x}^2$$

is called *arrival cost* and summarizes the prior knowledge on the states. It plays the same role as the cost-to-go function in MPC. MHE for a linear system with Gaussian noise is equivalent to the Kalman filtering if this prior weighing is set to the exact arrival cost. In practice, the arrival cost often discounted (i.e., set to 0). For a deeper understanding of the theory behind MHE, Chapters 1.4 and 4 of Rawlings et al. [10] are recommended.

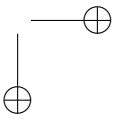
Advantages and Disadvantages MHE has several advantages:

- It is suitable for nonlinear and constrained systems.
- It provides a systematic way to handle model uncertainties.
- It can give better performance than standard filters for some systems, especially those with significant model errors or non-Gaussian noise.

However, MHE also has some disadvantages:

- It requires solving an optimization problem at each time step, which can be computationally expensive.
- It requires the selection of the horizon length and the weighting matrices, which can be challenging.
- It requires a good initial guess for the states, which might not always be available.

In conclusion, MHE is a powerful tool for state estimation in complex systems. It can provide superior performance compared to traditional filtering techniques under certain conditions, but it also comes with its own challenges and limitations.



Bibliography

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[2] Alberto Bemporad, Manfred Morari, and Fabrizio Borelli. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[3] Steven C Chapra. *Numerical methods for engineers*. Mcgraw-hill, 2010.

[4] S. Dickler, T. Wintermeyer-Kallen, and J. et al. Zierath. Full-scale field test of a model predictive control system for a 3MW wind turbine. *Forsch Ingenieurwes*, 85:313–323, 2021.

[5] Lars Grüne, Jürgen Pannek, Lars Grüne, and Jürgen Pannek. *Nonlinear model predictive control*. Springer, 2017.

[6] U. Jassmann, S. Dickler, J. Zierath, M. Hakenberg, and D. Abel. Model predictive wind turbine control with move-blocking strategy for load alleviation and power leveling. *Journal of Physics: Conference Series*, 735(052021), 2016.

[7] Jan M Maciejowski. *Predictive Control: with Constraints*. Pearson Education Limited, 1995.

[8] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[9] Thivaharan Albin Rajasingham. *Nonlinear model predictive control of combustion engines*. Springer, 2021.

[10] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

[11] A. Wächter and L. Biegler. IPOPT - an Interior Point OPTimizer. https://projects.coin-or.org/Ipopt, 2009.

[12] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[13] Roger Wilfried Wimmer. *Regelung einer Wärmepumpenanlage mit model predictive control*. ETH Zurich, 2004.