Simulation of mechatronic systems
(Simulation mechatronischer Systeme)

Lecture Notes

Jürgen Pannek

June 9, 2023
During summer term 2023 I give the lecture to the module *Simulation of mechatronic systems* (*Simulation mechatronischer Systeme*) at the Technical University of Braunschweig. To structure the lecture and support my students in their learning process, I prepared these lecture notes. As it is the first edition, the notes are still incomplete and are updated in due course of the lecture itself. Moreover, I will integrate remarks and corrections throughout the term.

The aim of the module is to classify simulation techniques from numerical mathematics and apply these to mechatronic case studies. After completing the module, the students shall be able to recall, categorize, apply, select and rate simulation methods to mechatronic use cases. Moreover, students shall be able to describe, explain, evaluate, analyze and assess simulation results. As such, students shall be capable to derive and apply automation procedures for deployment, simulation and testing of digital models.

To this end, we will tackle the subject areas

- dynamical systems,
- software development,
- simulation and testing,
- visualization, and
- process automation

within the lecture. To support students, we utilize university resources such as Git to assist students to create a professional software development tool chain and interact with cluster computing technology within the tutorial classes. The module itself is accredited with 5 credits.

An electronic version of this script can be found at

https://www.tu-braunschweig.de/itl/lehre/skripte
Contents

Contents iv

List of tables v

List of figures vii

List of definitions and theorems x

List of algorithms xi

1 Intention 1
  1.1 Aim of lecture ........................................ 1
  1.2 Software development .................................. 3
  1.3 Basic terms ........................................... 5
  1.4 Examples .............................................. 13
    1.4.1 Spring damper system ............................. 13
    1.4.2 Quarter car system ............................... 15
    1.4.3 Supply chain system .............................. 17

2 Dynamical systems 19
  2.1 Differential equations ................................. 20
  2.2 Explicit methods ..................................... 25
  2.3 Implicit methods .................................... 32
  2.4 Adaptive step size methods ........................... 36
  2.5 Adaptation for deployment ............................ 43

3 Simulation 45
  3.1 Parametrization ....................................... 45
  3.2 Processing of results ................................. 48
  3.3 Sensitivity ........................................... 57

Bibliography 61
List of Tables

2.1 Advantages and disadvantages of differential equations .................. 24
2.2 Advantages and disadvantages of one step methods ....................... 29
2.3 Butcher tableaux of explicit Runge-Kutta class methods ................ 30
2.4 Butcher tableaux of the classical Runge-Kutta method ................... 30
2.5 Order of consistence vs. number of function evaluations for one step methods . 31
2.6 Advantages and disadvantages of explicit Runge-Kutta class methods ........ 32
2.7 Butcher tableaux of implicit Runge-Kutta class methods ................. 33
2.8 Advantages and disadvantages of implicit Runge-Kutta class methods ........ 36
2.9 Butcher tableaux of the Runge-Kutta 4(3) method ......................... 37
2.10 Butcher tableaux of the Dormand-Prince 5(4) method ..................... 37
2.11 Advantages and disadvantages of adaptive step size methods ............ 42

3.1 Example of a data dump ................................................. 51
3.2 Sorted data dump from Table 3.1 with sort key t .......................... 52
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>V model for system development</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Agile development</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Term of a system</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Example of static and dynamic systems</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Model and remainder</td>
<td>9</td>
</tr>
<tr>
<td>1.6</td>
<td>Example of logistics system linking assembly and quality inspection</td>
<td>13</td>
</tr>
<tr>
<td>1.7</td>
<td>Mass spring system</td>
<td>14</td>
</tr>
<tr>
<td>1.8</td>
<td>Schematic drawing of a quarter car test bench</td>
<td>16</td>
</tr>
<tr>
<td>1.9</td>
<td>Sketch of a three stage supply network</td>
<td>18</td>
</tr>
<tr>
<td>2.1</td>
<td>Dimensions of model characteristics</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>Sketch of a dynamic flow and a trajectory</td>
<td>23</td>
</tr>
<tr>
<td>2.3</td>
<td>Sketch of UML diagram of class solution method</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Sketch of UML diagram of class explicit Runge-Kutta</td>
<td>32</td>
</tr>
<tr>
<td>2.5</td>
<td>Sketch of UML diagram of class implicit Runge-Kutta</td>
<td>35</td>
</tr>
<tr>
<td>2.6</td>
<td>Sketch of UML diagram of class one step method</td>
<td>41</td>
</tr>
<tr>
<td>2.7</td>
<td>Sketch of UML diagram of class multiscale method</td>
<td>44</td>
</tr>
<tr>
<td>3.1</td>
<td>Sketch of UML diagram of class simulator</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Sketch of UML diagram of class simulator with data processing</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Point cloud of three time series from Figure 3.4</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Sketch of three time series</td>
<td>54</td>
</tr>
<tr>
<td>3.5</td>
<td>Box plot of three time series from Figure 3.4</td>
<td>56</td>
</tr>
<tr>
<td>3.6</td>
<td>Sketch of UML diagram of class output</td>
<td>57</td>
</tr>
</tbody>
</table>
# List of Definitions and Theorems

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>System and process</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Time</td>
<td>6</td>
</tr>
<tr>
<td>1.6</td>
<td>State</td>
<td>7</td>
</tr>
<tr>
<td>1.8</td>
<td>Model</td>
<td>8</td>
</tr>
<tr>
<td>1.11</td>
<td>Simulator</td>
<td>10</td>
</tr>
<tr>
<td>1.14</td>
<td>Simulation</td>
<td>11</td>
</tr>
<tr>
<td>1.15</td>
<td>Cost function</td>
<td>12</td>
</tr>
<tr>
<td>1.17</td>
<td>Cost functional and assessment</td>
<td>12</td>
</tr>
<tr>
<td>2.1</td>
<td>Ordinary Differential Equation</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Initial Value Problem</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Lipschitz Condition</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Existence and Uniqueness</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>Simplified Existence and Uniqueness</td>
<td>23</td>
</tr>
<tr>
<td>2.11</td>
<td>Sampled time</td>
<td>25</td>
</tr>
<tr>
<td>2.12</td>
<td>One step method</td>
<td>25</td>
</tr>
<tr>
<td>2.13</td>
<td>Numerical solution method</td>
<td>26</td>
</tr>
<tr>
<td>2.16</td>
<td>Consistency</td>
<td>26</td>
</tr>
<tr>
<td>2.18</td>
<td>Consistency check</td>
<td>27</td>
</tr>
<tr>
<td>2.20</td>
<td>Stability</td>
<td>28</td>
</tr>
<tr>
<td>2.22</td>
<td>Convergence</td>
<td>28</td>
</tr>
<tr>
<td>2.23</td>
<td>Explicit Runge-Kutta class methods</td>
<td>29</td>
</tr>
<tr>
<td>2.26</td>
<td>Bounded order of consistency for explicit methods</td>
<td>31</td>
</tr>
<tr>
<td>2.29</td>
<td>Implicit Runge-Kutta class methods</td>
<td>33</td>
</tr>
<tr>
<td>2.30</td>
<td>Bounded order of consistency for implicit methods</td>
<td>33</td>
</tr>
<tr>
<td>2.31</td>
<td>Contraction of implicit Runge-Kutta methods</td>
<td>34</td>
</tr>
<tr>
<td>2.32</td>
<td>Embedded one step method</td>
<td>36</td>
</tr>
<tr>
<td>2.34</td>
<td>Approximated error for adaptive step size methods</td>
<td>39</td>
</tr>
<tr>
<td>2.35</td>
<td>Adaptive step size computation</td>
<td>40</td>
</tr>
<tr>
<td>2.37</td>
<td>Multiscale time grid</td>
<td>43</td>
</tr>
<tr>
<td>3.1</td>
<td>Parametrized control system</td>
<td>46</td>
</tr>
</tbody>
</table>
Definition 3.3 Computer program ............................................. 48
Definition 3.5 Terminating/non-terminating simulation ............. 48
Definition 3.6 Replication ....................................................... 49
Definition 3.8 Data dump ....................................................... 49
Definition 3.9 Widget ........................................................... 49
Definition 3.10 Data processing .............................................. 50
Definition 3.11 Data analytics ............................................... 50
Definition 3.12 Point cloud ................................................... 53
Definition 3.14 Time series / trajectories ................................ 54
Definition 3.17 Mean, sample variance and confidence interval .... 55
Definition 3.19 Statistical plots ............................................. 56
Definition 3.21 Sensitivity ..................................................... 57
Definition 3.24 Numerical sensitivity ..................................... 58
Definition 3.26 Scenario analysis ......................................... 59
1 Explicit Runge-Kutta methods ........................................... 31
2 Implicit Runge-Kutta methods ........................................... 34
3 Full step iteration ............................................................ 35
4 Adaptive step size method ............................................... 41
5 One step methods .......................................................... 42
6 Multiscale application of one step methods ......................... 43
7 Prototype of simulator ..................................................... 47
8 Prototype of simulation ................................................... 47
CHAPTER 1 INTENTION

Theory provides the maps that turn an uncoordinated set of experiments or computer simulations into a cumulative exploration.

David E. Goldberg

Within this introduction chapter we strive to lay the foundations for the overall lecture. As the above quote already indicates, simply doing a simulation without having a proper theory on how to analyze the results and design the experiments is just number crunching.

Within Section 1.1, we introduce the V model for development and discuss the purpose of simulation in different types of scenarios. The following Section 1.2 introduces the concept of software development, which we will cover in the tutorials in parallel to the lecture. Thereafter, in Section 1.3 we derive the representation of a system/process and of its model from basic norms. In order to be directly usable, we formally define a system/process and its model using a standard mathematical formulation from the literature. Thereafter, we characterize the term simulator, simulation and assessment, which will be used throughout the lecture. In the last Section 1.4, we introduce three different examples of models of systems, which we will use to illustrate the steps of simulation in the upcoming chapters.

1.1 Aim of lecture

As outlined in the foreword, the lecture provides insights into dynamical systems, software development, simulation and testing, visualization, and process automation. To this end, we discuss both the overall methodology but also specific methods within the steps of the overall methodol-
ogy. The lecture itself is driven by the idea of software development as it is currently found in basically any company, cf. Figure 1.1.

![V model for system development](image)

Note that the V model representation in Figure 1.1 already includes the cycles that are found in the so called agile approach. To learn these techniques, in the tutorial classes we utilize a state-of-the-art toolchain to implement, integrate, test and automate our example applications.

Taking one step back, we must first ask ourselves what the aim of simulation actually is. On a very fundamental level, there are three different types of scenarios for which simulation can be used in applications:

1. Understand a known scenario:
   In this setting, the scenario itself is both known and real data is available. Examples may range from driving/drifting behavior of cars, reaction of mechanical components to software behavior or accidents. Here, the aim is to gather insight into the scenario and understand connections, i.e. the question "why does something happen the way it happens".

2. Optimize a known scenario:
   Building up on the previous case, optimization aims to generate a wanted behavior. In this case, typically no real data is available as testing is rather expensive. Typical examples are timetables for truck fleets, performance tests for engines, material tests, reaction of software in application scenarios and so forth. Hence, simulation is used to improve insight into possible improvements which require real time tests later, i.e. the question "how can something be utilized to improve given criteria".

3. Predict unknown scenario:
   Using the extrapolation idea from the previous case, the idea of using simulation in an unknown scenario is to predict possible behavior and quantify uncertainty of the results. In
contrast to the previous case, no testing is possible. Examples are climate change, fine dust distribution prediction for cities, properties of new materials etc. Here, simulation is used to provide us with a range within which the truth most likely can be found, i.e. the question "is something likely to happen or to work out fine".

The lecture itself runs in parallel to the lecture *Modeling of mechatronic systems (Modellierung mechatronischer Systeme)*, within which models and modeling techniques are developed, which can be used for simulation. Within the tutorial, we deepen the understanding of simulation by using a full stack software development tool.

### 1.2 Software development

On the implementation side of simulation, modern instruments can be used to utilize an agile/DevOps approach for implementation. The idea of the latter is to work in sprints to first create a working prototype with basic functionality, which is then extended/improved in following sprints. The typical picture to be seen in the literature is given in Figure 1.2.

![Figure 1.2: Agile development](image)

**Remark 1.1**

We like to note that Figure 1.2 shows the extended version used in DevOps, which treats software
already in use. In the prelaunch phase or in pre-/development, the operate and monitor parts are
omitted.

As DevOps aims to significantly increase the applicability/usability of the software (and the satisfac-
tion of customers), DevOps is an endless loop and in many cases also drawn as a lying infinity
symbol.

Within the loop, the tasks correspond to the following:

- **Plan:** In this part, tasks are organized, set up and scheduled in a chosen management tool.
  The idea is to plan tasks using the user story process from the agile methodology. Writing
tickets in the form of a user story will allow Devs and Ops to understand what development
needs to be done and why. A perfect user story exhibits a what (who, where, trigger), a
why, and acceptance criteria.

- **Code:** Within this part, Devs are doing code development and code review. When the code
  is ready, they merge it. In DevOps practice, it is important to share a code tool between
Ops and developers teams.

- **Build:** In this step, the first step toward automation is done. The goal here is to build the
  source code into one desired format, compiling, testing, and deploying in a particular place
of the infrastructure. With this setup, continuous integration (CI) and delivery (CD) tools
can check and verify the source code from Source Code Management and build it.

- **Test:** Continuing the build task, continuous testing is applied to reduce risks of errors in
  the code. Automatic tests ensure that no bugs will be deployed to users. Therefore, testing
tools need to be implemented in the workflow to ensure the quality of the software.

- **Release:** After being tested, the software is made ready for deployment. Within this step,
  the process is handed over from Dev to Ops.

- **Deploy:** In this step, the Ops deploy the new software upon user systems. Similar to CI/CD,
a continuous deployment can be used for automation.

- **Operate/ configure infrastructure:** In this step, a scalable infrastructure is built and main-
tained by Ops. Additionally, security issues and log management are taken care of.

- **Monitor:** Last, monitoring concludes the circle and allows to identify issues and wishes,
  which are then fed to Devs in the planning step.

Within the tutorials of this lecture, we will consider a tool chain based on GitLab, which allows for
code maintenance, issue tracking, CI/CD, automatic testing and automatic deployment. Further
details will be given in the tutorial itself.
In the following, we will focus on simulation from the theoretical and methodological point of view. While complementing modeling, we still require the respective terms which we introduce next.

1.3 Basic terms

Within the lecture, we utilize definitions and common notation from the books of Lunze [5], Khalil [4] and Sontag [7]. We start with object we are actually interested in, that is the system or process at hand. Although used in various different scientific and non-scientific areas, the term is often not defined clearly but on a rather high level. In the literature [6], we see the following description for a system (translated from German):

A system is a set of interrelated elements that are viewed as a whole in a particular context and considered as distinct from their environment.

DIN IEC 60050-351 (2014)

Building on this description, a process is given as follows (translated from German):

A process is the entirety of relations and interacting elements in a system through which matter, energy or information is transformed, transported or stored.

DIN IEC 60050-351 (2014)

As we are interested in simulation, we have to be more specific and mathematically defined the following:

**Definition 1.2 (System and process).** Consider two sets \( \mathcal{U} \) and \( \mathcal{Y} \). Then a map \( \Sigma : \mathcal{U} \to \mathcal{Y} \) is called a system and the application of this map to an input \( u \in \mathcal{U} \) to obtain an output \( y = \Sigma(u) \in \mathcal{Y} \) is called a process.

In particular, the sets \( \mathcal{U} \) and \( \mathcal{Y} \) are called input and output sets. An element from the input set \( u \in \mathcal{U} \) is called an input, which act from the environment to the system and is not dependent on the system itself or its properties, cf. Figure 1.3.
We distinguish between inputs, which are used to specifically manipulate (or control) the system, and inputs, which are not manipulated on purpose. We call the first ones control or manipulation inputs, and we refer to the second ones as disturbance inputs. An element from the output set $y \in \mathcal{Y}$ is called an output. In contrast to an input, the output is generated by the system and influences the environment.

**Remark 1.3**

Note that in most cases not all measurable outputs are actually measured. Similarly, in many cases not all manipulable inputs are controlled.

To continue with a system, we need to introduce the concept of time:

**Definition 1.4** (Time).

A time set $\mathcal{T}$ is a subgroup of $(\mathbb{R}, +)$.

Time allows us to let a system evolve. To see this point, we consider two electrical systems illustrated in Figure 1.4 which represent an ideal resistance and an ideal capacitor.

The systems in Figure 1.4 possess the input variable $I(t)$, the output variable $U(t)$ and time $t$. For the resistance $R$ the output is uniquely defined by the input for every time instant $t$, i.e. we have

$$y(t) = U(t) = R \cdot I(t) = R \cdot u(t).$$  \hspace{1cm} (1.1)
If the outputs depend on the input at the same time instant, we call systems such as this one static. In contrast to this, the computation of the voltage $U(t)$ at the capacitor $C$ at time instant $t$ depends on the entire history $I(\tau)$ for $\tau \leq t$, i.e. we have

$$y(t) = U(t) = \frac{1}{C} \int_{-\infty}^{t} I(\tau) d\tau = \frac{1}{C} \int_{-\infty}^{t} u(\tau) d\tau. \quad (1.1)$$

If we additionally know the voltage $U(t_0)$ at a time instant $t_0 \leq t$, then only the history $t_0 \leq \tau \leq t$ of the current is required, i.e.

$$y(t) = U(t) = \frac{1}{C} \int_{-\infty}^{t} I(\tau) d\tau = \frac{1}{C} \int_{-\infty}^{t_0} I(\tau) d\tau + \frac{1}{C} \int_{t_0}^{t} I(\tau) d\tau = U(t_0) + \frac{1}{C} \int_{t_0}^{t} u(\tau) d\tau. \quad (1.2)$$

As we can see from (1.1), the initial value $U(t_0)$ contains all the information on the history $\tau \leq t_0$. For this reason, one typically refers to $U(t_0)$ as the internal state of the system capacitor at time instant $t_0$. If the output of the system depends not only on the input at the time instant but also on the history of the latter, we call these systems dynamic.

**Remark 1.5**

*Note that by this definition the set of dynamic systems covers the set of static systems.*

If for a system according to Figure 1.3 the outputs $y_1(t), \ldots, y_n(t)$ depend on the history of the inputs $u_1(\tau), \ldots, u_n(\tau)$ for $\tau \leq t$ only, then the system is called causal. As all technically feasible systems are causal, we will restrict ourselves to this case.

Our discussion so far allow us to give the general definition of states of a dynamical system:

**Definition 1.6 (State).**

Consider a system $f : U \rightarrow \mathcal{Y}$. If the output $y(t)$ uniquely depends on the history of inputs $u(\tau)$ for $t_0 \leq \tau \leq t$ and some $x(t_0)$, then the variable $x(t)$ is called state of the system.

**Link:** For further details on how to design inputs/controls such that system properties regarding outputs can be generated, we refer to the lectures *Control Engineering 1 & 2*.

In contrast to the input/output description of a system in Definition 1.2, states allow us to look inside a system. In engineering practice, the pure input/output description is also called black
box and the state description as white box. The states and their development over time allow us to deduce and interpret properties such as long term behavior (e.g. convergence or stability) and short term one (e.g. swing-overs or transients), which can be used to describe the system by these properties.

**Task 1.7**
*Which variable represents a state in case of induction?*

**Solution to Task 1.7**: Current through the inductor

In order to understand and utilize a system or process, we require a model of it. Technically, the mathematical description we gave in Definition 1.2 is already a model. Yet in engineering practice, the term model refers to a partial representation of a system/process, in particular representing those properties/parts which are of interest to the user of the model. Here, we define the following:

**Definition 1.8** (Model).
Consider a system $\Sigma$ with input and output sets $\mathcal{U}$ and $\mathcal{Y}$. Then a map $\Sigma_M : \mathcal{U} \rightarrow \mathcal{Y}$ is called a *model* of the system $\Sigma$ if $\Sigma_M \sim \Sigma$.

So if we use a model, there may be deviations between model prediction and reality, both on long and short time horizons depending on the choice of the model. As a model does not describe all of reality, the system/process is split into two parts,

- the model, which describes what we are interested in, and
- the remainder, which contains everything else.

Since we cannot tell anything about the remainder (as it is not modeled), interactions between model and remainder can only be interpreted as disturbances.

**Remark 1.9**
*Note that a disturbance for a system/system refers to unknown influences (e.g. environment) whereas a disturbance for a model indicates unmodeled influences (e.g. details of the system/process or environment).*
For further details on modeling, we refer to the lecture *Modeling of mechatronic systems* and for process modeling, identification and handling disturbances, we refer to the lecture *Systemics*.

![Diagram of model and remainder](image)

Figure 1.5: Model and remainder

For the models we consider in simulation, we assume the following to be satisfied:

**Assumption 1.10 (Principles of modeling)**

During the modeling process, six principles need to be met:

1. **Principle of Correctness**: A model needs to present the facts correctly regarding structure and dynamics (semantics). Specific notation rules have to be considered (syntax).

2. **Principle of Relevance**: All relevant items have to be modeled. Non-relevant items have to be left out, i.e. the value of the model doesn’t decline if these items are removed.

3. **Principle of Cost vs. Benefit**: The amount of effort to gather the data and produce the model must be balanced against the expected benefit.

4. **Principle of Clarity**: The model must be understandable and usable. The required knowledge for understanding the model should be as low as possible.

5. **Principle of Comparability**: A common approach to modeling ensures future comparability of different models that have been created independently from each other.

6. **Principle of Systematic Structure**: Models produced in different views should be capable of integration. Interfaces need to be designed to ensure interoperability.

Here, an interesting point arises: Since typically modeler and model user are different entities, which exhibit different perspectives on the process, a good model for the modeler may be very
different from a good model for the model user. For example, a detailed model may reflect reality very well, but it may be too complex to analyze it in simulations. Hence, modeling must be in line with the usage, and the quality of a model is determined by the degree it meets the requirements of the model user (“fitness for use”).

Within this lecture, our intended use is simulation where we are going to evaluate a model over time. Utilizing time allows us to consider not only instances of inputs $u$ but entire sequences $u(\cdot)$ where the input is no longer a single value but a map $u : \mathcal{T} \to \mathcal{U}$. To evaluate such sequences, we require the notion of a simulator to connect points over time:

**Definition 1.11 (Simulator).**
Consider a model $\Sigma_M$, a state $x_0$ at time $t_0$ and a terminal time $t_f$ to be given. Suppose that $t_0$ and $t_f$ are the lower and upper bounds of a set $\mathcal{T} \subset \mathcal{T}$ and an input sequence $u(t), t \in \mathcal{T}$ to be given. We call a method $\Phi : \mathcal{T} \times \mathcal{T} \times \mathcal{X} \times \mathcal{U} \times \mathcal{P} \to \mathcal{X}$ a simulator if

$$x(t) = \Phi(t, t_0, x_0, u(\cdot), p)$$

holds for any $t \in \mathcal{T}$.

More plastically speaking, a simulator evaluates the model — and thereby depends on it — for given initial value $(t_0, x_0)$, input sequence $u(\cdot)$ and parameters $p$ at any future time instant $t \in \mathcal{T}$ revealing the output of the model.

**Remark 1.12**
Here, we like to stress that a simulator is not unique, i.e. different methods can be applied. If the model is given by a set of differential equations, then both the analytic solution or numerical integrators or Koopman operators may be used as simulators.

**Task 1.13**
Consider the differential equation

$$\dot{x}(t) = x(t).$$

Define an analytical and a numerical simulator.
Solution to Task 1.13: The analytical solution of (1.4) is given by

\[ x(t) = C \cdot \exp(t) \]

which can be identified using initial values \( x(t_0) = x_0 \) to

\[ x(t) = x_0 \cdot \exp(t - t_0). \]

Hence, we obtain

\[ \Phi(t, t_0, x_0, u(\cdot)) := x_0 \cdot \exp(t - t_0). \]

A numerical solution of (1.4) can be derived using the Euler method

\[ x(t) = x(t_0) + \dot{x}(t_0) \cdot (t - t_0) \]

which again can be applied for given initial values \( x(t_0) = x_0 \). In this case, we obtain

\[ \Phi(t, t_0, x_0, u(\cdot)) := x_0 + \dot{x}(t_0) \cdot (t - t_0). \]

Both are simulators but will reveal different results. For this reason, it is important to select the right solution method for the task at hand.

In contrast to simulators, simulation is makes use of not only a specific initial value, time interval and input sequence but allows to consider entire sets of the latter.

**Definition 1.14** (Simulation).
Consider a model \( \Sigma_M \) and a simulator \( \Phi \) to be given. Suppose sets of initial values \( X_0 \subset X \), initial times \( T_0 \) and terminal times \( T_f \) as well as input sequences sets \( U^T := \{u(t) \mid t \in T\} \) and parameters \( P \) to be given. Then we call a map \( \Theta : T \times T \times X \times U^T \times P \rightarrow X^T \) given by

\[ \Theta(t, t_0, x_0, u(\cdot), p) = \left\{ \Phi(t, t_0, x_0, u(\cdot), p) \mid (t, t_0, x_0, u(\cdot), p) \in T_f \times T_0 \times X_0 \times U^T \times P \right\} \]

(1.5)

a simulation.

Based on the simulation results, an evaluation to assess the results is conducted. Basis of such an assessment are so called key performance indicators, which are defined on a high level via ISO 22400 [3]:
A key performance criterion is a function, which measures defined information retrieved from the system/process or model against a standard.

ISO 22400 (2014)

Similar to the high level definition of a system/process, in practice we require a more rigorous definition to actually calculate:

**Definition 1.15** (Cost function).
We call a key performance criterion given by a function $\ell : \mathcal{X} \to \mathbb{R}_0^+$ a cost function.

**Task 1.16**
*Design a cost function to compute the square deviation from a target $x^{\text{ref}}$.***

**Solution to Task 1.16:** Given the target value $x^{\text{ref}}$, the standard squared deviation is given by

$$\ell(x) := (x - x^{\text{ref}})^2.$$  

In that case the costs are minimal if the state of the model/system approaches the wanted target value $x^{\text{ref}}$.

The value of a key performance criterion reveals a snapshot only, i.e. the evaluation at one time instant $t \in \mathcal{T}$. To obtain the performance, we need to evaluate it over the operating period of the system. Since by doing so we define a function of a function, this is referred to as a functional.

**Definition 1.17** (Cost functional and assessment).
Consider a key performance criterion $\ell : \mathcal{X} \to \mathbb{R}_0^+$. Then we call

$$J(\Theta(\Sigma_M, \Phi)(t, t_0, x_0, u(\cdot), p)) := \int_{t_0}^{t} \ell(x(\tau))d\tau$$  \hspace{1cm} (1.6)

cost functional for the tuple $(t, t_0, x_0, u(\cdot), p)$. The combination of cost functionals for all tuples of the simulation $J(\Theta(\Sigma_M, \Phi))$ is called assessment.
Task 1.18

**Define the mean squared error of a simulation.**

**Solution to Task 1.18:** An assessment using the mean over all simulations in the set $S := T_f \times T_0 \times X_0 \times \mathcal{U}^T \times \mathcal{P}$ is given by

$$J(\Theta(\Sigma, \Phi)) := \sum_{(t, t_0, x_0, u(\cdot)) \in S} \frac{1}{\#S} J(\Theta(\Sigma, \Phi)(t, t_0, x_0, u(\cdot), p))$$

which is an all-out simulation of all possible combinations in the set $S$.

**Remark 1.19**

*Note that similar to the mean squared error, also other statistical parameters such as covariance, quantiles and statistical test can be performed.*

Having defined the basic terms we require for simulation, we next introduce examples which highlight the different application scenarios outlined in Section 1.1.

### 1.4 Examples

In the following, we will introduce three examples for our scenario cases of understanding, optimizing and predicting. The examples are given in order of complexity where in particular for the last example no analytical solution is available.

#### 1.4.1 Spring damper system

To illustrate the understanding scenario, we consider a logistics process, which links assembly and quality inspection via their dynamics as outlined in Figure 1.6.

[Diagram of logistics system]

Figure 1.6: Example of logistics system linking assembly and quality inspection

In the most simple formulation, the deviation from the standard production rate is given by the
following model consisting of two second order differential equations with constant coefficients

\[ \ddot{x}_1 + \alpha x_1 = \beta x_2 \]
\[ \ddot{x}_2 + \alpha x_2 = \beta x_1 \]

where \( \alpha, \beta > 0 \) represent the coupling constants of the processes. The model itself is equivalent to a mass spring system of two masses connected by springs given in Figure 1.7.

![Mass spring system](image)

Figure 1.7: Mass spring system

While being simple enough to allow for an analytical solution, the model still allows to integrate excitation such as, e.g., assembly irregularities or equipment failure in quality inspection, which results in coupled oscillations of both components.

To obtain an analytical solution of the system, we define the transfer functions

\[ X_1(s) := \mathcal{L}\{x_1(t)\}, \quad X_2(s) := \mathcal{L}\{x_2(t)\} \]

using the Laplace transform and obtain the following set of equations

\[ [s^2 \cdot X_1(s) - s \cdot 0 - v(0)] + \alpha \cdot X_1(s) = \beta \cdot X_2(s) \]
\[ [s^2 \cdot X_2(s) - s \cdot 0 + v(0)] + \alpha \cdot X_2(s) = \beta \cdot X_1(s) \]

Hence, we have

\[ (s^2 + \alpha) \cdot X_1(s) - \beta \cdot X_2(s) = v_0 \]
\[ -\beta \cdot X_1(s) + (s^2 + \alpha) \cdot X_2(s) = -v_0 \]

This a linear equation system with unknowns \( X_1(s) \) and \( X_2(s) \). Solving this equation system, we
obtain

\[ X_1(s) = v_0 \cdot \frac{1}{s^2 + \omega^2} \]
\[ X_2(s) = -v_0 \cdot \frac{1}{s^2 + \omega^2}, \]

where we used the abbreviation \( \omega^2 := \alpha + \beta \).

Both frequency domain functions are of type \( F(s) = \frac{1}{s^2 + \omega^2} \). Hence, we obtain the following time domain function:

\[ x_1(t) = \mathcal{L}^{-1} \{ X_1(s) \} = \mathcal{L}^{-1} \left\{ v_0 \cdot \frac{1}{s^2 + \omega^2} \right\} = v_0 \cdot \mathcal{L}^{-1} \left\{ \frac{1}{s^2 + \omega^2} \right\} \]
\[ = v_0 \cdot \frac{\sin(\omega t)}{\omega} = \left( \frac{v_0}{\omega} \right) \cdot \sin(\omega t) \]
\[ x_2(t) = \mathcal{L}^{-1} \{ X_2(s) \} = \mathcal{L}^{-1} \left\{ -v_0 \cdot \frac{1}{s^2 + \omega^2} \right\} \]
\[ = - \left( \frac{v_0}{\omega} \right) \cdot \sin(\omega t) = \left( \frac{v_0}{\omega} \right) \cdot \sin(\omega t + \pi) \]

Hence, both systems are oscillating with identical amplitude

\[ A = \frac{v_0}{\omega} = \frac{v_0}{\sqrt{\alpha + \beta}} \]

and frequency

\[ \omega = \sqrt{\alpha + \beta}, \]

yet with phase shift \( \pi \).

**Remark 1.20**

*Note that the poles of \( X_1(s) \) and \( X_2(s) \) are purely imaginary. Hence, we obtain a system which is rotating and not coming to a standstill. In reality, such a rotation would come to a stop by adapting to the disturbance, i.e. using friction as an input within the model.*

### 1.4.2 Quarter car system

In our second example, we aim for the optimizing scenario. To this end, we consider the more complex system of a quartercar test bench as depicted in Figure 1.8.

The system shall allow us to excite the test bench via the input \( u(\cdot) \), which resembles road conditions. As the input sequence can be structured arbitrarily, optimization for untrained input
is possible and in particular it is impossible to simulate all possible input sequences (as there exist infinitely many of them).

For the test bench, we assume that only vertical movements are considered. Moreover, we model the system such that the chassis is modeled as mass $m_1$ at position $y_1$, the suspension is modeled using a spring and a damper element $s_1$, $d_1$. Similarly, the wheel and the axis are modeled as mass $m_2$ at position $y_2$, the wheel is modeled using a spring and a damper element $s_2$. Last, road undulations are modeled via the road height function $u(t)$.

In order to apply d’Alemberts principle of equality of forces, we first model the individual forces

$$m_i \ddot{v}_i^m(t) = F_i^m(t), \quad d_i \dot{v}_i^d(t) = F_i^d(t), \quad s_i \dot{y}_i^s(t) = F_i^s(t)$$

for $i = 1, 2$ where we used

$$v_1^m(t) = \dot{y}_1(t), \quad v_1^d(t) = v_1(t) - v_2(t), \quad \dot{v}_1^s(t) = y_1(t) - y_2(t),$$

$$v_2^m(t) = \dot{y}_2(t), \quad v_2^d(t) = \dot{v}_2(t) - \dot{u}(t), \quad \dot{y}_2^s(t) = y_2(t) - u(t).$$

Now, we can combine the equations to describe the forces in all masses. To this end, the direction of the forces has to be treated carefully. In $m_1$, the force $F_1^m$ points into the upwards direction: Since $m_1$ is the upper end of the attached spring and damper, $F_1^d$ and $F_1^s$ also point upwards.
Hence, in $m_1$ we obtain

$$F_1^m + F_1^d + F_1^s = 0.$$ 

In $m_2$, forces $F_1^d, F_1^s$ point downwards, all other forces upwards and we obtain

$$F_2^m - F_1^d - F_1^s + F_2^d + F_2^s = 0.$$ 

Combined, we have

$$0 = F_1^m + F_1^d + F_1^s = m_1 \ddot{v}_1^t(t) + d_1 \dot{v}_1^d(t) + s_1 \dot{y}_1^s(t) = m_1 \dot{y}_1(t) + d_1 (\dot{y}_1(t) - \dot{y}_2(t)) + s_1 (y_1(t) - y_2(t))$$ 

and

$$0 = F_2^m - F_1^d - F_1^s + F_2^d + F_2^s = m_2 \ddot{v}_2^t(t) - d_1 \dot{v}_1^d(t) - s_1 \dot{y}_1^s(t) + d_2 \dot{v}_2^d(t) + s_2 \dot{y}_2^s(t) = m_2 \dot{y}_2(t) - d_1 (\dot{y}_1(t) - \dot{y}_2(t)) - s_1 (y_1(t) - y_2(t)) + d_2 (\ddot{y}_2(t) - \ddot{u}(t)) + s_2 (y_2(t) - u(t)).$$

These equations display two second order differential equations and can be reformulated as a system of four first order differential equations.

**Remark 1.21**

*Note that also a Lagrangian or Hamiltonian approach to derive the equations of motion can be used, yet the outcome will always be equivalent.*

The above model is still rather simple and can be adapted using the parameters of the dampers and springs. So depending on the requirement of simulation detail, the results may be sufficiently accurate. Still, the model is complicated enough such that for a simulation one would choose a numerical solver.

### 1.4.3 Supply chain system

Our last example aims at the prediction scenario for simulation. Here, we consider a supply chain as multi stage network consisting of supplier ($s$), manufacturer ($M$) and retailer ($R$), cf. Figure 1.9 for an illustration.
Within each of the blocks, we suppose the internal dynamics to be given by the model

\[
\begin{align*}
\dot{s}^p(t) &= f_s(a^p(t), \ell^p(t)) \quad \text{(Stock)} \\
\dot{o}^p_u(t) &= f_o(o^p(t), a^p(t)) \quad \text{(Unfulfilled order to stock)} \\
\dot{b}^p(t) &= f_b(d^p(t), \ell^p(t)) \quad \text{(Backlog from stock)}
\end{align*}
\]

where \( p \in S = \{S, M, R\} \) denotes the stages of the network. Moreover, \( a^p, \ell^p, o^p \) and \( d^p \) denote the arriving and leaving units as well as the order and demand rates. In order to be physically meaningful, we require that for all times \( t \geq 0 \) and stages \( p \in S \), the model is subject to the constraints

\[
\begin{align*}
0 &\leq o^p(t) \leq o^p_{\text{max}} \\
0 &\leq o^p_u(t) \leq o^p_{u,\text{max}} \\
0 &\leq s^p(t) \leq s^p_{\text{max}} \\
0 &\leq b^p(t) \leq b^p_{\text{max}}
\end{align*}
\]

as well as unknown customer orders \( o^C \) and fixed delivery delays \( \tau_{ij} \), where \( i, j \in S \) represent consecutive stages. The stages need to be linked since arrival/leaving as well as demand/order information is required to evaluate the dynamics. Here, we use \( a^i(t + \tau_{ij}) = \ell^i(t) \) and \( d^j(t + \tau_{ij}) = o^j(t) \) for consecutive nodes \( i, j \in S \) and \( a^{i}(\tau_{ii}) = o^{i}(t) \) for the supplier to define these connections. The state for each stage can be defined via \( x^p := (s^p, o^p_u, b^p)^\top \). As a result, we obtain a system of 9 differential equations coupled by 6 algebraic equations with and without delays.

For the above model no analytical solution is known. Additionally, the delays allow us to include stochastic variables. For such settings, it is not possible to conduct tests yet by applying numerical solvers simulation renders the model well suited for prediction.

In the following chapter, we will focus on description methods for such models and connect these to solution methods, which provide us with simulators.
In today's practice, dynamical systems are present in almost any scientific area. These systems arise from modeling of systems as outlined in the previous chapter, and may take several forms. Generally speaking, the (mathematical) description of models varies depending on the considered time, space and amplitude properties, cf. Figure 2.1 for a rough overview on these characteristics.

Within this lecture, we focus on models given by differential equations. These models arise naturally for mechatronic systems during modeling via, e.g. d’Alemberts principle or the Lagrangian or Hamiltonian approach. In the following Section 2.1, we recall important aspects from differential equations before stating numerical methods to solve the arising initial value problem in Sections 2.2–2.4. Last, we tackle the point of automated deployment of computations of solu-
tions in Section 2.5. For an in-depth understanding, we refer to the books of Deuflhard [2] and Stoer [1].

## 2.1 Differential equations

As indicated by Figure 2.1, dynamical systems differ regarding structural parameters such as time, space and amplitude:

- Regarding time, we start off with static models, which are characterized by the fact that inputs, outputs and measurements of the system are available. In contrast to that, continuous time models exhibit data streams being received continuously. Discrete time models differ from that by the availability of data, which is received at certain, not necessarily equidistant time instances. Last, event triggered models require issues to trigger receiving data.

- Regarding space, models may vary from a simple connection to complex systems.

- Regarding amplitude, models may differ regarding continuous spaces like e.g. mass and discrete spaces such as gear shifts.

Within this section, we focus on models $\Sigma_M$ given by differential equation (systems) of first order subject to continuous time, i.e. $T = \mathbb{R}$, high dimensional space and real amplitude, i.e. $\mathcal{X} = \mathbb{R}^{nx}$. Such an ordinary differential equation relates the derivative of a function $x : T \rightarrow \mathcal{X}$ with its one-dimensional argument and the function itself. More formally:

**Definition 2.1 (Ordinary Differential Equation).**
An ordinary differential equation in $\mathcal{X} = \mathbb{R}^{nx}$, $nx \in \mathbb{N}$, is given by the dynamic

$$\dot{x}(t) := \frac{d}{dt}x(t) = f(t, x(t))$$  \hspace{1cm} (2.1)

where $f : D \rightarrow \mathcal{X}$ is a continuous function with open subset of $D \subset T \times \mathcal{X} = \mathbb{R} \times \mathbb{R}^{nx}$.

**Task 2.2**
Models derived via the Lagrangian approach typically form ordinary differential equation systems of second order

$$\ddot{x}(t) = \ddot{f}(t, \dot{x}(t)).$$  \hspace{1cm} (2.2)

Reform the latter in the form (2.1).
Solution to Task 2.2: System (2.2) can be reformulated equivalently to obtain a system of first order differential equation

\[
\dot{x}(t) = \begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} x_2 \\ \tilde{f}(t, x_2(t)) \end{pmatrix} = f(t, x(t))
\]

where \( x_2 = \dot{x}. \)

The solution of (2.1) is a continuously differentiable function \( x : \mathcal{T} \to \mathcal{X}, \) which satisfies (2.1). In general, we will use the following denomination throughout the script:

- The independent variable \( t \) is referred to as time, although other interpretations are possible.
- For the time derivative \( \frac{d}{dt} x(t) \) we use the abbreviation \( \dot{x}(t). \)
- The function \( x(t) \) is called solution at time \( t \) and the entirety \( x(\cdot) \) is called trajectory.
- If the function \( f \) is independent of \( t, \) i.e. \( \dot{x}(t) = f(x(t)) \), then the differential equation is called autonomous.

An ordinary differential equation typically possesses infinitely many solutions, cf. Task 1.13. To obtain a unique solution, we have to introduce a constraint, the so called initial value constraint. Combined with the differential equation (2.1), this reveals the so called initial value problem:

**Definition 2.3** (Initial Value Problem).
Consider a function \( f : \mathcal{T} \times \mathcal{X} \to \mathcal{X} \) together with values \( t_0 \in \mathcal{T} \) and \( x_0 \in \mathcal{X} \) to be given. Then the initial value problem is to find the solution satisfying the differential equation

\[
\dot{x}(t) = f(t, x(t)) \quad (2.1)
\]

and the initial value condition

\[
x(t_0) = x_0. \quad (2.3)
\]

Here, the time \( t_0 \in \mathcal{T} \) is called initial time and the value \( x_0 \in \mathcal{X} \) is called initial value. Both the pair \((t_0, x_0)\) and equation (2.3) are called initial condition.

**Remark 2.4**
A continuously differentiable function \( x : \mathbb{D} \to \mathcal{X} \) solves the initial value problem (2.1), (2.3) for
some $t_0 \in \mathcal{T}$ and $x_0 \in \mathcal{X}$ if and only if for each $t \in \mathcal{T}$ the integral equation

$$x(t) = x_0 + \int_{t_0}^{t} f(\tau, x(\tau)) \, d\tau \quad (2.4)$$

holds. This follows directly by integration of (2.1) with respect to $t$ or via differentiation of (2.4) with respect to $t$ using the central theorem of differentiation and integration. Note that since continuity of $x(t)$ on the right hand side of (2.4) implies continuous differentiability of the right hand side, each continuous function $x(t)$ satisfying (2.4) is automatically continuously differentiable.

Under certain conditions, existence and uniqueness of a solution to the problem from Definition 2.3 can be shown. This is the so called Lipschitz condition

**Definition 2.5 (Lipschitz Condition).**
Consider a function $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$. Then $f$ is called Lipschitz in its second argument, if for each compact set $K \subset \mathcal{T} \times \mathcal{X}$ there exists a constant $L > 0$ and

$$\|f(t, x) - f(t, y)\| \leq L \|x - y\| \quad (2.5)$$

holds for all $t \in \mathcal{T}$ and all $x, y \in \mathcal{X}$ with $(t, x), (t, y) \in K$.

**Task 2.6**
Show that any function $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ is Lipschitz if it is continuously differentiable in its second argument.

**Solution to Task 2.6:** Since $f$ is continuously differentiable, we have that there exists $K \in \mathbb{R}_0^+$ such that

$$\lim_{x \to y} \frac{\|f(t, x) - f(t, y)\|}{\|x - y\|} \leq K$$

holds. Now we can set $L := K$ and multiply both sides by $\|x - y\|$ showing the assertion.

Using this property, we can show the following:
Theorem 2.7 (Existence and Uniqueness).
Consider a differential equation (2.1) with \( f : \mathcal{T} \times \mathcal{X} \to \mathcal{X} \). Moreover, \( f \) is considered to be continuous and Lipschitz continuous in the second argument. Then for each initial condition \((t_0, x_0) \in \mathcal{T} \times \mathcal{X}\), there exists a unique solution \( x(t; t_0, x_0) \) of the initial value problem (2.1), (2.3). This solution is defined for all \( t \) from an open maximal interval of existence \( I_{t_0, x_0} \) with \( t_0 \in I_{t_0, x_0} \).

Note that by Task 2.6, the following holds:

Corollary 2.8 (Simplified Existence and Uniqueness).
Consider a differential equation (2.1) with \( f : \mathcal{T} \times \mathcal{X} \to \mathcal{X} \). If \( f \) is continuously differentiable in its second argument, then the assertion of Theorem 2.7 holds.

Here, we like to note that the dynamic reveals a flow of the system at hand, whereas a trajectory is bound to a specific initial value and input sequence. The following Figure 2.2 illustrates the idea of flow and trajectory. In this case, the flow is colored to mark its intensity whereas the arrows point into its direction. The trajectory is evaluated for a specific initial value and „follows“ the flow accordingly.

Note that at the boundary of the interval of existence \( I_{t_0, x_0} \) the solution ceases to exist. If the interval is bounded, then there are two possible reasons for that: For one, the solution may diverge,
or secondly the solution converges to a boundary point of $\mathcal{T} \times \mathcal{X}$. In the remainder of this script, we will always assume that the assumptions of Theorem 2.7 are met without explicitly stating it.

Remark 2.9

1. A consequence of Theorem 2.7 is the so called cocycle property. This property states that for $(t_0, x_0) \in \mathcal{T} \times \mathcal{X}$ and two time instances $t_1, t \in \mathbb{R}$, we have

   $$x(t; t_0, x_0) = x(t; t_1, x_1)$$

   with $x_1 = x(t_1; t_0, x_0)$ given that all terms are defined according to Theorem 2.7.

2. Another consequence is that two solutions cannot intersect, as they would have to coincide for all times.

3. Some ordinary differential equations can be solved analytically via various methods. In general, this is not true and numerical methods must be used for this purpose. Yet, one typically not only applies numerical methods, but tries to show certain properties of the solution analytically.

Table 2.1: Advantages and disadvantages of differential equations

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Allows to model dynamics</td>
<td>✗ May require solvers</td>
</tr>
<tr>
<td>✓ Provides unique solutions</td>
<td>✗ Requires identification</td>
</tr>
</tbody>
</table>

Based on ordinary differential equations and the initial value problem, we next introduce numerical methods to compute a solution of the initial value problem. These methods can later be used as simulators in the sense of Definition 1.11 in simulations.

Remark 2.10

We like to note that apart from numerical methods also analytical methods can be used. Amongst these, the Fourier method (also called separation of variables) is the most prominent one. Still other methods like direct integration, substitution or variation of parameters exist. These methods are outside the scope of this lecture. While being exact, these methods exhibit the downside of not being generally applicable.

In the following, we will concentrate on a particular class of numerical solution methods, which is already indicated in the class diagram sketched in Figure 2.3.
We like to stress that any solution method is independent from the model. While not all solution methods can be applied to specific models, they are designed to compute a solution for particular classes of models. The separation of solution method and model will be relevant in the upcoming Chapter 3 where we will focus on simulation and handling results.

### 2.2 Explicit methods

The most prominent and also most simple class of numerical methods are one step methods. The key characteristic of these methods is that they completely rely on the solution computed in the previous step. Multi step methods, on the other hand, consider a series of solutions for previous steps. Both classes have in common that they operate using fixed step sizes, that is sampled time instances. Utilizing our Definition 1.4 of time, we obtain:

**Definition 2.11** (Sampled time).
Consider a time set $\mathcal{T} = \mathbb{R}$, a base time $t_0 \in \mathcal{T}$ and a number $T \in \mathbb{R}$. Then we call the set

$$\mathbb{T} := \{ t \in \mathcal{T} \mid t_i = t_0 + i \cdot T, \ i \in \mathbb{Z} \}$$

(2.7)

*sampled time set* or *time grid*. Moreover, we call $T$ *sampling period*.

As outlined, the idea of one step methods is to compute the solution of a differential equation based at a time instant using the solution at the previous one. In general, we define the following:

**Definition 2.12** (One step method).
Suppose a map $\Psi : \mathbb{R} \times \mathcal{X} \to \mathcal{X}$ and a parameter $h \in \mathbb{R}$ to be given. Then we call

$$x(t_{i+1}) = \Psi(h, x(t_i))$$

(2.8)
a one step method and h step size.

Technically, a one step method considers only one step from $t_i$ to $t_{i+1}$ and not the computation of a solution. Therefore, a numerical solution method is the generalization over time:

**Definition 2.13** (Numerical solution method).
Consider a map $\Psi : \mathbb{R} \times \mathcal{X} \rightarrow \mathcal{X}$, initial conditions $(t_0, x(t_0)) \in \mathbb{R} \times \mathcal{X}$ and a terminal time $\bar{T} \in \mathcal{T}$, $\bar{T} > t_0$ to be given. Then we call a map $\Psi : \mathcal{T} \times \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}$ defining

$$x(\bar{T}) = \Psi(\bar{T}, t_0, x(t_0))$$  \hspace{1cm} (2.9)

a numerical solution method.

**Remark 2.14**
Note that in many cases step size and sampling period are identical, i.e. $h = T$.

**Example 2.15**
Given a differential equation (2.1), the most simple one step methods are the so called Euler method

$$\Psi(h, x(t_i)) = x(t_i) + h \cdot f(t_i, x(t_i)),$$

and the Heun method

$$\Psi(h, x(t_i)) = x(t_i) + \frac{h}{2} \cdot (f(t_i, x(t_i)) + f(t_i + h, x(t_i) + h \cdot f(t_i, x(t_i)))) .$$

To apply any numerical approximation method for differential equations, we must make sure that the approximation error stays bounded with respect to the chosen sampling period $h$. To this end, we want the approximation to converge to the true solution if the sampling period is reduced, i.e. $h \rightarrow 0$. To ensure convergence, two properties are required:

**Definition 2.16** (Consistency).
Consider a one step method $\Psi : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}$. We call the method consistent if there exist $C, p \in \mathbb{R}_0^+$ such that

$$\| \Psi(h, x_0) - x(t_0 + h; t_0, x_0) \| \leq C \cdot h^{p+1}$$ \hspace{1cm} (2.10)
holds for all initial values $t_0 \in \mathcal{T}$, $x_0 \in \mathcal{X}$ and all at least $p$ times continuously differentiable models $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ and all $0 < h \leq h^*$. We call $p$ the order of consistency.

**Task 2.17**
Assess whether consistency is a local or global property with respect to a trajectory.

**Solution to Task 2.17**: By definition, consistency means that the one step method is bounded in each single step. For this reason, it represents a local error and therefore a local property.

In the autonomous case, that is the dynamic $f$ is independent from time, consistency can be checked using the following:

**Lemma 2.18** (Consistency check).
Consider a one step method $\Psi : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ and suppose the initial value problem to be autonomous. Then the method is consistent if

$$
\lim_{h \to 0} \left\| \frac{\Psi(h, x) - x}{h} - f(x) \right\| = 0.
$$

(2.11)

**Task 2.19**
Show that the Euler method is consistent.

**Solution to Task 2.19**: We directly obtain

$$
\left\| \frac{\Psi(h, x) - x}{h} - f(x) \right\| = \|f(x) - f(x)\| = 0.
$$

Having dealt with the single step error, we have to include that any one step method by definition will use the possibly error prone value to continue calculating. To cope with this source, we define the following:
**Definition 2.20** (Stability).
Suppose a one step method $\Psi : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ and a constant $M > 0$ to be given. Then we call $\Psi$ to be stable if
\[
\|\Psi(h, x_1) - \Psi(h, x_2)\| \leq (1 + h \cdot M) \cdot \|x_1 - x_2\| \tag{2.12}
\]
holds for all $x_1, x_2 \in \mathcal{X}$ and all $0 < h \leq h^\star$.

**Task 2.21**
*Show that the Euler method is stable.*

**Solution to Task 2.21**: Similar to consistency, we directly obtain
\[
\|\Psi(h, x_1) - \Psi(h, x_2)\| = \|x_1 + h \cdot f(\cdot, x_1) - x_2 - h \cdot f(\cdot, x_2)\|
\]
\[
= \|x_1 - x_2\| + h \cdot \|f(\cdot, x_1) - f(\cdot, x_2)\| \leq (1 + h \cdot L) \cdot \|x_1 - x_2\|
\]
where $L$ is the Lipschitz constant of $f$ and we get $M = L$.

Now, these two properties can be combined to obtain convergence of the method:

**Theorem 2.22** (Convergence).
*Suppose a one step method $\Psi : \mathbb{R} \times \mathcal{X} \to \mathcal{X}$ of order $p$, which is consistent and stable. Then for each $h \in \mathbb{R}$ there exists a constant $K(h) > 0$ such that
\[
\|x(t_i) - x(t_0 + i \cdot h; t_0, x_0)\| \leq K(h) \cdot h^p \tag{2.13}
\]
holds for all $i \in \mathbb{N}$ with $i \cdot h \leq h$ and all at least $p$ times continuously differentiable models $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$.*

Convergence can be seen as the global error of a one step method. What we can take from Theorem 2.22 together with Lemma 2.18 is that Lipschitz plus consistency (of order $p$) gives us convergence (of order $p - 1$).
The idea of the Heun method can be continued to derive methods of higher order. Similar as for the Heun method, these methods require additional evaluations of the dynamic $f$. These methods are combined in the class of so called Runge-Kutta methods:

**Definition 2.23** (Explicit Runge-Kutta class methods).

Suppose a map $\Psi : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}$ and a step size $h \in \mathbb{R}$ to be given. Let

$$k_1 = f(t_j + c_1 \cdot h, x(t_j)) \quad (2.14)$$
$$k_2 = f(t_j + c_2 \cdot h, x(t_j) + \alpha_{21} \cdot h \cdot k_1) \quad (2.15)$$
$$\vdots$$
$$k_m = f(t_j + c_m \cdot h, x(t_j) + \alpha_{m1} \cdot h \cdot k_1 + \ldots + \alpha_{mm-1} \cdot h \cdot k_{m-1}) \quad (2.17)$$

Then we call

$$\Psi(h, x(t_j)) = x(t_j) + h \cdot (\beta_1 \cdot k_1 + \ldots + \beta_m \cdot k_m) \quad (2.18)$$

an $m$-step explicit Runge-Kutta class method.

The Runge-Kutta class methods are defined uniquely by the coefficients $\alpha_{ij}$ and $\beta_i$.

**Remark 2.24**

*In order to reveal a convergent (consistent and stable) behavior, the coefficients $\alpha_{ij}$ and $\beta_i$ must satisfy certain conditions, which are outside the scope of this lecture.*

Note that the number $m$ refers to the required evaluations of the dynamics $f$ and is in general not identical with the order of the method. In short, the methods are given by so called Butcher tableaux.
Table 2.3: Butcher tableaux of explicit Runge-Kutta class methods

Task 2.25
Assemble the Butcher tableaux of the Euler and Heun method given in Example 2.15.

Solution to Task 2.25: The Euler and Heun method are given by

\[
\begin{array}{c|ccc}
0 & 0 & 1 \\
\hline
1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\
\end{array}
\]

Here, we like to point out that the classical Runge-Kutta method is a 4-step method is a special case of this class, cf. Table 2.4 for the respective Butcher tableaux.

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

Table 2.4: Butcher tableaux of the classical Runge-Kutta method

From an algorithmic point of view, explicit one step methods are very simple to implement:
Algorithm 1 Explicit Runge-Kutta methods

Input: Dynamic $f : T \times X \rightarrow X$
Input: Initial value $(t_0, x_0) \in T \times X$
Input: Step size $h \in \mathbb{R}_0^+$
Input: Terminal time $T$
Input: Butcher tableaux of explicit $m$ step Runge-Kutta class method $\Psi$

1: procedure CLASS EXPLICIT RUNGE-KUTTA($f, t_0, x_0, h, T, \Psi$)
2:      $N \leftarrow (T - t_0)/h$
3:    for $j = 0, \ldots, N - 1$ do
4:        for $i = 1, \ldots, m$ do
5:            $k_i = f(t_j + c_i \cdot h, x(t_j) + \alpha_{i1} \cdot h \cdot k_1 + \ldots + \alpha_{ii-1} \cdot h \cdot k_{i-1})$
6:        end for
7:        $x(t_{j+1}) \leftarrow x(t_j) + h \cdot (\beta_1 \cdot k_1 + \ldots + \beta_m \cdot k_m)$
8:        $t_{j+1} = t_j + h$
9:    end for
10: end procedure

Output: Endpoint of trajectory $x(T)$

While the method is simple in its implementation, the derivation of the coefficients is a very difficult task. We like to note that the order of convergence (and consistency) comes at a price, i.e. the higher the order the higher the required number of function evaluations, cf. Table 2.5.

| Order of consistence $p$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $\geq 9$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal steps $m$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>$\geq p + 3$</td>
</tr>
</tbody>
</table>

In general, the following holds:

**Theorem 2.26** (Bounded order of consistency for explicit methods).
Consider an explicit Runge-Kutta method with $m$ steps. Then the order of consistency $p$ of this method is bound by $p \leq m$.

**Remark 2.27**
At this point, we need to be careful: On the one hand, Theorem 2.26 assures that a higher order
of consistency and thereby accuracy of the solution can be achieved. The expense we have to pay comes in number of steps \( m \) causing the dynamic \( f \) to be evaluated respectively often. Up till now, there exists no answer to the question which solver should be chosen for a specific problem.

On the implementation side, we know that the explicit Runge-Kutta method is a derivative of a general one step method. As such, it should be a derivative of a respective object. In the following Figure 2.4, we sketch the connection of both. Here, we already separated the class model containing the dynamics of our system.

![Figure 2.4](image)

**Figure 2.4: Sketch of UML diagram of class explicit Runge-Kutta**

**Remark 2.28**

Note that Figure 2.4 is not complete and only a sketch of a possible class diagram.

Regarding explicit one step methods in general, we see the advantages and disadvantages outlined in Table 2.6.

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Allows for simple implementation</td>
<td>✓ Requires multiple evaluations</td>
</tr>
<tr>
<td>✓ Utilizes intermediate steps</td>
<td>✓ Propagates intermediate error</td>
</tr>
</tbody>
</table>

### 2.3 Implicit methods

In contrast to explicit methods, for implicit methods a full stack of coefficients is used. The idea of using additional coefficients is to raise the bound on consistency of the method, that is to allow for a more accurate approximation with less intermediate function evaluations.

More formally, we define the following:
Definition 2.29 (Implicit Runge-Kutta class methods).
Suppose a map $\Psi : T \times X \to X$ and a step size $h \in \mathbb{R}$ to be given. Let

$$k_i = f \left( t_i + c_i \cdot h, x(t_i) + \sum_{j=1}^{m} \alpha_{ij} k_j \right)$$

(2.19)

define intermediate steps for $i = 1, \ldots, m$. Then we call

$$\Psi(h, x(t_i)) = x(t_i) + h \cdot \sum_{i=1}^{m} \beta_i \cdot k_i$$

(2.20)

an $m$-step implicit Runge-Kutta class method.

The method is called implicit as the values of $k_i$, $i = 1, \ldots, m$ is not longer a definition but requires the solution of a $m \cdot n_x$ dimensional nonlinear equation system. Similar to the explicit case, the Butcher tableau can be defined as in Table 2.7.

<table>
<thead>
<tr>
<th>$c_i$</th>
<th>$\alpha_{11}$</th>
<th>$\alpha_{12}$</th>
<th>$\cdots$</th>
<th>$\alpha_{1m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_2$</td>
<td>$\alpha_{21}$</td>
<td>$\alpha_{22}$</td>
<td>$\cdots$</td>
<td>$\alpha_{2m}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$c_m$</td>
<td>$\alpha_{m1}$</td>
<td>$\alpha_{m2}$</td>
<td>$\cdots$</td>
<td>$\alpha_{mm}$</td>
</tr>
</tbody>
</table>

| $\beta_1$ | $\beta_2$ | $\cdots$ | $\beta_m$ |

Table 2.7: Butcher tableaux of implicit Runge-Kutta class methods

For these methods, we can show the following:

Theorem 2.30 (Bounded order of consistency for implicit methods).
Consider an implicit Runge-Kutta class method with $m$ steps. Then the order of consistency $p$ of this method is bound by $p \leq 2 \cdot m$.

Again similar to the explicit case, the implementation of implicit methods is a straight forward procedure as displayed in Algorithm 2.

While being a significant improvement regarding consistence/convergence, we can see from Algorithm 2 that implicit methods require the solution of the nonlinear equation system for the auxiliary variables $k_i$. The latter can be solved using the Banach fixpoint iteration. The latter,
Algorithm 2 Implicit Runge-Kutta methods

Input: Dynamic $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$
Input: Initial value $(t_0, x_0) \in \mathcal{T} \times \mathcal{X}$
Input: Step size $h \in \mathbb{R}_0^+$
Input: Terminal time $T$
Input: Butcher tableaux of implicit $m$ step Runge-Kutta class method $\Psi$

1: procedure CLASS IMPLICIT RUNGE-KUTTA $(f, t_0, x_0, h, T, \Psi)$
2: $N \leftarrow (T - t_0)/h$
3: for $j = 0, \ldots, N - 1$ do
4:   $k \leftarrow$ SOLUTION NONLINEAR EQUATION SYSTEM$(f, t_j, x(t_j), h, \Psi, \varepsilon)$
5:   $x(t_j) \leftarrow x(t_j) + h \cdot \sum_{i=1}^{m} \beta_i \cdot k_i$
6: $t_{j+1} = t_j + h$
7: end for
8: end procedure

Output: Endpoint of trajectory $x(T)$

However, requires that the map is a contraction. Here, we use the abbreviations

$$k := \begin{pmatrix} k_1 \\ \vdots \\ k_m \end{pmatrix} \quad \text{and} \quad F(k) := \begin{pmatrix} f \left( t_j + c_1 \cdot h, x(t_j) + \sum_{j=1}^{m} \alpha_{1j} k_j \right) \\ \vdots \\ f \left( t_j + c_m \cdot h, x(t_j) + \sum_{j=1}^{m} \alpha_{mj} k_j \right) \end{pmatrix}.$$ 

In order to be a contraction, we require

$$\|F(k^l) - F(k^i)\| \leq K\|k^l - k^i\|$$

to hold. Here, we have the following:

**Theorem 2.31** (Contraction of implicit Runge-Kutta methods).

Consider an implicit Runge-Kutta class method and suppose the dynamics $f$ to be Lipschitz with Lipschitz constant $L$. Then the constant $K$ in

$$\|F(k^l) - F(k^i)\| \leq K\|k^l - k^i\| \quad (2.21)$$

can be given by $K = h \cdot L$. If the step size is chosen such that $K = h \cdot L < 1$, then inequality (2.31) is a contraction.

Given that the assumptions made in Theorem 2.31 hold, the following Algorithm 3 can be applied to solve the nonlinear equation system (2.19)
2.3 Implicit methods

Algorithm 3  Full step iteration

Input: Dynamic \( f : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X} \)
Input: Initial value \((t_0, x_0) \in \mathcal{T} \times \mathcal{X}\)
Input: Step size \( h \in \mathbb{R}_0^+ \)
Input: Butcher tableaux of implicit \( m \) step Runge-Kutta class method \( \Psi \)
Input: Stopping threshold \( \varepsilon \in \mathbb{R}^+ \)

1: procedure SOLUTION NONLINEAR EQUATION SYSTEM \((f, t_0, x_0, h, \Psi, \varepsilon)\)
2: \( i = 1 \) and \( k^{i-1} \leftarrow 1, k^{i} \leftarrow 0 \)
3: while \( \| k^{i} - k^{i-1} \| \leq \varepsilon \) do
4: \( k^{i+1} = \begin{pmatrix} k_1^{i+1} \\ \vdots \\ k_m^{i+1} \end{pmatrix} \leftarrow \begin{pmatrix} f \left( t_0 + c_1 \cdot h, x_0 + \sum_{j=1}^{m} \alpha_{1j} k_j^{i+1} \right) \\ \vdots \\ f \left( t_0 + c_m \cdot h, x_0 + \sum_{j=1}^{m} \alpha_{mj} k_j^{i+1} \right) \end{pmatrix} = F(k^{i}) \)
5: \( i \leftarrow i + 1 \)
6: end while
7: end procedure

Output: Approximation \( k \) of solution of nonlinear equation system

Similar to the explicit case, on the implementation side we can inherit similar items from the class one step method. Internally, the implicit methods require the stopping threshold and a function to solve nonlinear equation systems as in Algorithm 3. Figure 2.5 provides a respective sketch of the class diagram.

![Figure 2.5: Sketch of UML diagram of class implicit Runge-Kutta](image)

Here, we like to note that the order of consistence of the implicit Runge-Kutta class methods depends on the order of approximation of the solution of the nonlinear equation system. In particular, we require the local error bound (2.16) to hold, which renders \( \varepsilon \) to be dependent on \( h \). Combined, we restate facts for implicit Runge-Kutta class methods outlined in Table 2.8.
Table 2.8: Advantages and disadvantages of implicit Runge-Kutta class methods

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Allows higher order of convergence</td>
<td>✗ Consistency depends on chosen error</td>
</tr>
<tr>
<td>✓ Reduces number of evaluations per order of consistency</td>
<td>✗ Requires solution of nonlinear equation system</td>
</tr>
<tr>
<td></td>
<td>✗ Limits step size</td>
</tr>
</tbody>
</table>

2.4 Adaptive step size methods

For one step methods, our basic assumption was that the step size $h$ is constant. As solutions to differential equations may exhibit regions with very fast and complex behavior as well as regions where this is not the case, constant step size is not efficient. In this case, efficient means that the number of required evaluations of the dynamic is as low as possible. For constant step size, always the worst case of fast/complex behavior must be assumed to guarantee a sufficiently good approximation.

The concept of adaptive step size methods is to eliminate or at least reduce this downside. To this end, first one iteration with step size $h$ is executed and the numerical error of the solution is estimated. Secondly, if the error is larger than a predefined threshold, then a smaller step size is calculated based on the estimated error and the iteration step is repeated. Lastly, if the error is smaller than the threshold, the step size is calculated based on the estimated error and applied for the next iteration step.

In order to compute an estimate of the error and a suitable new step size, the idea is to combine two one step methods of different order of consistency. Then we know that by construction the method of higher order is more reliant and can use both methods to correct one another. Since evaluating two different methods results in many evaluations of the dynamic, it is efficient to use methods for which function evaluations can be reuses – so called embedded methods.

**Definition 2.32 (Embedded one step method).**

Suppose two convergent one step methods $Ψ^1, Ψ^2 : T \times X \to X'$ of different order. Furthermore let the intermediate steps

$$k_i = f \left( t_i + c_i \cdot h, x(t_i) + \sum_{j=1}^{m} \alpha_{ij} k_j \right)$$

(2.22)
be identical for both methods for \( i = 1, \ldots, m \). Then the combination \( \Psi^1, \Psi^2 \) is called an embedded one step method.

**Remark 2.33**

By (2.22), the values \( \alpha_{ij} \) and \( c_i \) are identical for both methods. Since the methods are of different order, we obtain that \( \beta_j \) are different for these methods.

Two commonly found embedded methods are the Runge-Kutta 4(3) and the Dormand–Prince 5(4) method (in MATLAB called ode45) shown in Tables 2.9 and 2.10.

```
0  1  1/2  1/2
  1/2  0  1/2
  1  0  0  1
  1  1/6  2/6  1/6

Ψ^1  1/6  2/6  1/6  0
Ψ^2  1/6  2/6  0  1/6
```

Table 2.9: Butcher tableaux of the Runge-Kutta 4(3) method

```
0  1  5/6  1/6
  3/10  3/10  9/40
  3/10  3/10  9/40
  4/5  44/45  -56/15  32/9
  8/9  19372/6561  -25360/6561  64448/6561  -212/219

Ψ^1  5179/57600  0  7571/16695  393/640  -92097/339200  187/2100  1/40
Ψ^2  5179/57600  0  7571/16695  393/640  -92097/339200  187/2100  1/40
```

Table 2.10: Butcher tableaux of the Dormand-Prince 5(4) method
Before combining the latter to an algorithm, we first discuss and analyze the three described steps.

**Error estimation**

Since the step size can only be adapted for the current and for future iterations, it does not make sense to elaborate on errors, which already occurred in past iterations. Hence, we can focus on stability, i.e. one single step, and ignore convergence, i.e. the long run which is already given by construction of both one step methods.

As we use two different one step methods of different order, we want to estimate the error of the lower order method. To this end, we compare the true solution to the solution of the lower order method $\Psi^1$

$$
\varepsilon^1 := x(t_{j+1}) - \Psi^1(h, x(t_j))
$$

where $h$ is the current step size. Similarly, we obtain the error of the solution of the higher order method $\Psi^2$

$$
\varepsilon^2 := x(t_{j+1}) - \Psi^2(h, x(t_j))
$$

and the difference of both

$$
\varepsilon := \Psi^2(h, x(t_j)) - \Psi^1(h, x(t_j)).
$$

As $\varepsilon$ is the only expression we can actually compute, it should be the basis for computing a step size adaptation. Now, we can utilize that $\Psi^2$ is of higher order. Hence, we have that

$$
\lim_{h \to 0} \frac{\|\varepsilon^2\|}{\|\varepsilon^1\|} = 0.
$$

Defining $\theta := \|\varepsilon^2\| / \|\varepsilon^1\|$, we can reformulate the latter fraction to

$$
\frac{\|\varepsilon^2\|}{\|\varepsilon^1\|} = \frac{\|\varepsilon^1 - \varepsilon\|}{\|\varepsilon^1\|} = \theta \quad \iff \quad \|\varepsilon^1 - \varepsilon\| = \|\varepsilon^1\| \cdot \theta.
$$

Applying the triangle inequalities

$$
\|\varepsilon^1\| - \|\varepsilon\| \leq \|\varepsilon^1 - \varepsilon\| \leq \|\varepsilon^1\| + \|\varepsilon\|
$$
we obtain

\[
\frac{1}{1 + \theta} \|\varepsilon\| \leq \|\varepsilon^1\| \leq \frac{1}{1 - \theta} \|\varepsilon\|.
\]

Hence, for \( h \to 0 \) we have \( \theta \to 0 \) and the approximation

\[
\|\varepsilon\| \approx \|\varepsilon^1\|,
\]

holds. Therefore, we can use \( \|\varepsilon\| \) as approximation of the true error \( \|\varepsilon^1\| \) of the lower order method. This reveals

\begin{shaded}
\textbf{Theorem 2.34} (Approximated error for adaptive step size methods).
Consider two convergent one step methods \( \Psi^1, \Psi^2 : T \times \mathcal{X} \to \mathcal{X} \) of different order. If the step size \( h \) is sufficiently small, then the error of the lower order method can be approximated via the difference of the local errors of \( \Psi^1, \Psi^2 \)

\[
\varepsilon^1 \approx \varepsilon := \Psi^2(h, x(t_j)) - \Psi^1(h, x(t_j))
\]

for each iteration instant \( t_j = t_0 + j \cdot h \).
\end{shaded}

\subsection*{Computation of step size}

Based on the approximated error \( \varepsilon \), an adaptation of the step size shall be computed. To this end, a threshold \( \varepsilon \) for the error needs to be predefined for the adaptive step size method.

From consistency, we know that the lower order method satisfies

\[
\|\varepsilon^1\| \leq C \cdot h^{p+1}
\]

where \( p \) is the known order of the method and \( C \) is unknown. Hence, in the worst case, the latter holds with equality. Now we can use our approximation \( \varepsilon \approx \varepsilon^1 \) and the worst case to obtain

\[
\|\varepsilon\| \approx \|\varepsilon^1\| = C \cdot h^{p+1}
\]

and can identify

\[
C \approx \frac{\|\varepsilon\|}{h^{p+1}}.
\]
The new step size $h_{\text{new}}$ shall satisfy

$$C \cdot h_{\text{new}}^{p+1} \leq \varepsilon$$

where we can use the identified constant $C$ to obtain

$$C \cdot h_{\text{new}}^{p+1} \approx \frac{\|\varepsilon\|}{h^{p+1}} \cdot h_{\text{new}}^{p+1} \leq \varepsilon \iff h_{\text{new}} \approx \sqrt[p+1]{\frac{\varepsilon}{\|\varepsilon\|}} \cdot h.$$  

As this is still an approximation, in practice an additional safety constant $\eta = 0.9$ is used to derive the new step size. Combined, we obtain the following:

**Theorem 2.35** (Adaptive step size computation).

*Consider two convergent one step methods $\Psi^1, \Psi^2 : T \times X \to X$ of different order. Let $p$ be the lower order of both methods and suppose the step size $h > 0$ to be sufficiently small for Theorem 2.34 to hold. To satisfy a given error threshold $\varepsilon > 0$, the step size can be adapted using*

$$h_{\text{new}} = \eta \cdot \sqrt[p+1]{\frac{\varepsilon}{\|\varepsilon\|}} \cdot h.$$  

**(2.24)**

*with safety constant $\eta \in (0, 1)$.*

Note that the threshold is guaranteed for the lower order method. Still, the solution of the higher order method is already available and most likely better in comparison. For this reason, typically the solution of the higher order method is used to define the iteration step in practice.

**Remark 2.36**

*Regarding the next iteration step, an identical computation as in Theorem 2.35 can be used. In contrast to the current iteration step, however, not only a reduction of the step size but also an increase may occur.*

Integrating all of the above, we obtain Algorithm 4.
Algorithm 4 Adaptive step size method

**Input:**
- Dynamic $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$
- Initial value $(t_0, x_0) \in \mathcal{T} \times \mathcal{X}$
- Terminal time $T \in \mathcal{T}$
- Butcher tableaux of embedded method $\Psi^1, \Psi^2$
- Error threshold $\varepsilon \in \mathbb{R}^+$
- Safety factor $\eta \in (0, 1)$

1: **procedure** CLASS ADAPTIVE STEP SIZE($f, t_0, x_0, T, \Psi^1, \Psi^2, \varepsilon, \eta$)
2: 
3: $h \leftarrow T - t_0$
4: 
5: **while** $t_j \neq T$ **do**
6: 
7: $t_{j+1} \leftarrow t_j + h$
8: 
9: $\Psi^1(h, x(t_j)) \leftarrow$ CLASS EXPLICIT RUNGE-KUTTA($f, t_j, x(t_j), h, t_{j+1}, \Psi^1$)
10: 
11: $\Psi^2(h, x(t_j)) \leftarrow$ CLASS EXPLICIT RUNGE-KUTTA($f, t_j, x(t_j), h, t_{j+1}, \Psi^2$)
12: 
13: $\epsilon \leftarrow \Psi^2(h, x(t_j)) - \Psi^1(h, x(t_j))$
14: 
15: $h_{\text{new}} \leftarrow \eta \cdot p+1 \sqrt{\frac{\epsilon}{\|\epsilon\|}} \cdot h$
16: 
17: **if** $\epsilon > \varepsilon$ **then**
18: 
19: $h \leftarrow h_{\text{new}}$
20: 
21: **else**
22: 
23: $x(t_{j+1}) := \Psi^2(h, x(t_j))$
24: 
25: $h \leftarrow h_{\text{new}}$
26: 
27: $j \geq j + 1$
28: 
29: **end if**
30: 
31: **end while**
32: 
33: **end procedure**

**Output:** Endpoint of trajectory $x(T)$

Utilizing Algorithm 4, we can update our schematic sketch of the class one step method as outlined in Figure 2.6. We like to note that it is not necessary to re-implement the explicit Runge-Kutta in order to be used in the adaptive step size method.
Regarding adaptive step size methods, Table 2.11 summarizes advantages and disadvantages of such methods.

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Adapts step size to dynamics</td>
<td>✗ Consistency depends on lower order</td>
</tr>
<tr>
<td>✓ Reduces number of evaluations</td>
<td>✗ Requires two one step methods</td>
</tr>
<tr>
<td>✓ Reuses intermediate results</td>
<td>✗ Requires embedding of methods</td>
</tr>
</tbody>
</table>

We like to note that the classes discussed so far in Algorithms 1, 2 and 4 can be combined in the class of one step methods. This is exactly, what we already did using the class diagrams in Figure 2.6. Algorithm 5 formalizes the latter.

**Algorithm 5 One step methods**

**Input:** Dynamic \( f : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X} \)
**Input:** Initial value \((t_0, x_0) \in \mathcal{T} \times \mathcal{X}\)
**Input:** Step size \(h \in \mathbb{R}^+\)
**Input:** One step method \(\Psi\)
**Input:** Terminal time \(\bar{T} \in \mathcal{T}\)
**Input:** Error threshold \(\varepsilon \in \mathbb{R}^+\)
**Input:** Safety factor \(\eta \in (0, 1)\)

1: **procedure** CLASS ONE STEP METHOD\((f, t_0, x_0, h, \bar{T}, \Psi, \varepsilon, \eta)\)
2:  
3:  
4:  
5:  
6:  
7:  
8:  
9:  
10: **end procedure**
**Output:** Endpoint of trajectory \(x(\bar{T})\)

We can already observe from the algorithms above that the interface for the adaptive step size method slightly differs from the explicit and implicit methods. Within an implementation, the respective function of the derived objects must therefore be modified.
2.5 Adaptation for deployment

As we have seen in the previous sections, one step methods are a suitable approach to numerically solve differential equations. Yet right at the beginning, we distinguished between the sampling time $T$ of the system/process and the step size $h$ of the method. If there are no requirements posed to sampling time and step size, the standard choice is to set $h = T$ and define the sampling time. Such an approach, however, can only be applied to pure software cases.

In case there are restrictions for sampling time $T$, then a so called multiscale time grid can be used for deriving the solution of the differential equation (2.1).

**Definition 2.37** (Multiscale time grid).
Consider sampling time $T \in \mathbb{R}_{>0}$ to be given defining a time grid $\mathbb{T}$ according to (2.7). Then for each $i \in \mathbb{N}$ the step size $h := T/i$ defines a multiscale time grid $\mathbb{T}_h$ satisfying

$$T \subset \mathbb{T}_h. \quad (2.25)$$

In practice, this is particularly useful. Applying a multiscale time grid allows us to apply numerical solution methods to systems containing components operating at a much slower speed $T$ than the numerical methods. Upon implementation, we can utilize Algorithm 6 accordingly.

**Algorithm 6** Multiscale application of one step methods

| Input: Dynamic $f : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ |
| Input: Initial value $(t_0, x_0) \in \mathcal{T} \times \mathcal{X}$ |
| Input: Step size $h \in \mathbb{R}^+$ |
| Input: Sampling time $T \in \mathbb{R}^+$ |
| Input: One step method $\Psi$ |
| Input: Terminal time $\overline{T} \in \mathcal{T}$ |
| Input: Error threshold $\varepsilon \in \mathbb{R}^+$ |
| Input: Safety factor $\eta \in (0, 1)$ |

1: procedure **CLASS MULTISCALE**($f, t_0, x_0, h, T, \overline{T}, \Psi, \varepsilon, \eta$)  
2: $N_T \leftarrow (T - t_0) / T$  
3: for $i = 0, \ldots, N_T$ do  
4: $x(t_{i+1}) \leftarrow \text{CLASS ONE STEP METHOD}(f, t_i, x(t_i), h, t_i + T, \Psi, \varepsilon, \eta)$  
5: end for  
6: end procedure

**Output:** Endpoint of trajectory $x(\overline{T})$

Considering the class diagrams we discussed so far, we see that the class multiscale defines the values of step size $h$, sampling time $T$, terminal time $\overline{T}$ and the initial values $(t_0, x_0)$ to be used in the one step method. Hence, these values are shifted in the sketch given in Figure 2.7.
The multiscale application additionally allows us to use the cocycle property of differential equations, cf. Remark 2.9, and switch solutions methods at runtime.

**Remark 2.38**

*Here, we only consider the case where the sampling time $T$ is fixed. In practice, also the case of fixed sampling time $T$ and bounded step size $h$ exists. As there may not be a common multiscale structure, sampling and solution may be required to run asynchronously. Respective solutions are beyond the scope of this lecture.*
In the previous Chapter 2 we discussed methods to compute solutions of differential equations. Within the present chapter, we will utilize these methods to analyze a given model via its solutions. In that case, we talk about simulation. The difference between model and simulator is given by its usage: A model is a representation of system whereas a simulator uses a model to generate its behavior.

As we have seen in the previous chapter, we can apply one step methods to models, which satisfy the requirements of these methods. This provides us with two different possibilities to generate the behavior: For one, we can modify the parameters set by the solution method, and secondly we can consider parameters set within the model. In both cases, only those parameters may be changed, which do not compromise the properties of solution method and model, i.e. convergence (or order of convergence) for the method or order of the differential equation system for the model.

Within this chapter, we first address the simulation problem itself and then discuss how a parametrization impacts on the structure of that problem in Section 3.1. Thereafter, we focus on processing results from a statistical and visualization point of view in Section 3.2. This leads us to identifying relations between parameters, which we discuss using sensitivities in Section 3.3. Concluding the chapter, we shortly discuss automation of these results.

### 3.1 Parametrization

In Chapter 2, the methods we derived aimed to solve the initial value problem from Definition 2.3. As we outlined before, we want to generate the behavior of the model for a set of parameters within both the model itself as well as the solution method.

Starting with the model itself, we need to make explicit which parameters we aim to modify. To the end, we introduce the parametrized control system:
Definition 3.1 (Parametrized control system).

We call a map \( f : T \times X \times U \times P \rightarrow X \) given by
\[
\dot{x}(t) = f(t, x(t), u(t), p)
\] (3.1)
a parametrized control system. We also refer to (3.1) as parametrized model \( \Sigma_M \).

Within this definition, we distinguish between time dependent parameters, here \( u(\cdot) \) being a function of time, and time independent (constant) parameters indicated by \( p \). The first denotation \( u(\cdot) \) typically refers to inputs of the system/model, which may be set from the outside to obtain properties such as, e.g., stability of the system. Within this lecture, we only refer to it as time dependent parameters.

As we can directly observe, the parametrized control system allows us to modify the dynamics of the model via our simulator given in Definition 1.11.

Remark 3.2

Note that a model as given in Definition 1.8 is more accurately a map from input to output, whereas the parametrized model (3.1) maps to states. For completeness, one may add
\[
y(t) = h(t, x(t), u(t), p)
\]
to render \( \Sigma_M \) to be a map from input to output.

Recalling the definitions of a simulator and simulation, the simulator may additionally parametrized using the solution method as well as step size and sampling time. Based on that, we can now merge these with the solution methods and the parametrized model we defined, cf. Figure 3.1.

---

**Figure 3.1:** Sketch of UML diagram of class simulator

---
We like to stress that the connection drawn from class simulation to class simulator is an aggregation, i.e. an instance of class simulation may hold a variety of instances of class simulator. A practical implication of the latter is that a simulation may run several simulators at the same time, e.g. for different parameter combinations. Hence, an algorithmic implementation is rather straightforward as shown in Algorithms 7 and 8.

**Algorithm 7** Prototype of simulator

| Input: | Model $\Sigma_M$ |
| Input: | Parameters $(t_0, x(t_0), u(\cdot), p) \in \mathcal{T} \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P}$ |
| Input: | Step size $h \in \mathbb{R}^+$ |
| Input: | Sampling time $T \in \mathbb{R}^+$ |
| Input: | One step method $\Psi$ |
| Input: | Terminal time $\bar{T} \in \mathcal{T}$ |
| Input: | Error threshold $\varepsilon \in \mathbb{R}^+$ |
| Input: | Safety factor $\eta \in (0, 1)$ |

1: procedure CLASS SIMULATOR($\Sigma_M$, $t_0$, $x(t_0)$, $u(\cdot)$, $p$, $h$, $T$, $\Psi$, $\varepsilon$, $\eta$)  
2: $\Sigma_M \leftarrow$ CLASS MODEL($u(\cdot)$, $p$)  
3: $x(\bar{T}) \leftarrow$ CLASS SOLUTION METHOD($\Sigma_M$, $t_0$, $x(t_0)$, $h$, $T$, $\bar{T}$, $\Psi$, $\varepsilon$, $\eta$)  
4: end procedure  

Output: Endpoint of trajectory $x(\bar{T})$

**Algorithm 8** Prototype of simulation

| Input: | Model $\Sigma_M$ |
| Input: | Parameter sets $\mathcal{T} \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P}$ |
| Input: | Step size $h \in \mathbb{R}^+$ |
| Input: | Sampling time $T \in \mathbb{R}^+$ |
| Input: | One step method $\Psi$ |
| Input: | Terminal time $T \in \mathcal{T}$ |
| Input: | Error threshold $\varepsilon \in \mathbb{R}^+$ |
| Input: | Safety factor $\eta \in (0, 1)$ |

1: procedure CLASS SIMULATION($\Sigma_M$, $\mathcal{T}$, $\mathcal{X}$, $\mathcal{U}^T$, $\mathcal{P}$, $h$, $T$, $\bar{T}$, $\Psi$, $\varepsilon$, $\eta$)  
2: for all $(t_0, x(t_0), u(\cdot), p) \in \mathcal{T} \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P}$ do  
3: while $t_0 < \bar{T}$ do  
4: $x(t_0 + T) \leftarrow$ CLASS SIMULATOR($\Sigma_M$, $t_0$, $x(t_0)$, $h$, $T$, $\bar{T}$, $\Psi$, $\varepsilon$, $\eta$)  
5: $t_0 \leftarrow t_0 + T$  
6: end while  
7: end for  
8: end procedure  

Output: Trajectory $x(\cdot)$

The core component we use to implement a simulation is a computer program.
Definition 3.3 (Computer program).
A sequence or set of instructions for a computer to execute is called computer program. Moreover, computer programs or components of the latter are called user interface or HMI (human machine interface) if it allows data to be transferred to and from the computer program.

Having taken the first step to compute solutions for a given parameter set, our next aim is to analyze the results.

3.2 Processing of results

Based on a simulation, the respective results are used to understand the system behavior and generate predictions for it. To this end, the results are processed to

- design replications,
- estimate performance metrics (e.g., via statistics), and
- analyze system and experiment (e.g., via textual and graphical output).

Remark 3.4
Replication design is an aim to obtain the most (deterministic/statistical) information from simulation runs for least computational cost. To this end, one typically tries to minimize the number of replications/experiments or the length of the latter.

Performance metrics aim to compute point estimates or confidence intervals for system parameters of interest. Here, one focuses on sample size and independence of observations which may be critical in the analysis step.

System analysis and experimentation aim to understand system behavior, generate performance predictions for possibly different scenarios and highlight design trade-offs. Examples range from parametric analysis of inputs or scenarios of operation.

Starting point of processing are the results obtained as data from the simulation. Here, we like to note that these results differ significantly with respect to time, or more accurately with respect to the time considered by the simulation.

Definition 3.5 (Terminating/non-terminating simulation).
Consider a simulation \( \Theta : T \times T \times X \times U^T \times P \rightarrow X^T \). If the terminal time is finite, i.e. \( T < \infty \), then we refer to \( \Theta \) as terminating simulation. If \( T = \infty \), then \( \Theta \) is called non-terminating simulation.
3.2 Processing of Results

The difference between the latter two cases has a major impact on processing of results. In case of a terminating simulation, several simulation runs can be executed sequentially and the results can be combined in an aposteriori manner, i.e. after finishing the runs. For non-terminating simulations, it is not possible to wait for termination but results need to be processed while the simulation is running. Also subsequent runs are not possible but parallel runs are required. Here, we start with the outcome of a simulation:

**Definition 3.6 (Replication).**
Given a simulation $\Theta : T \times T \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P} \to \mathcal{X}^T$ with terminal time $\overline{T} < \infty$ and inputs $\{(t_0, x(t_0), u(\cdot), p)_{i \in I}\} \subset T \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P}$. Then the set of outputs $\mathcal{R} := \{(x_j(t_i))_{i, j \in I}\}$ is called replication.

**Remark 3.7**
Since a simulation may call several simulators with different parameters, we used the index $x_j$ to indicate the simulator result and $t_i$ to indicate the simulation time index.

Based on a replication, many different processing steps may be taken. The first and particularly important one is to make a replication accessible. One of the most basic procedures for accessibility is the so-called data dump, which is the essence of saving.

**Definition 3.8 (Data dump).**
Suppose a replication $\mathcal{R}$ to be given by a computer program. Then we call a transfer of the replication to another computer program a data dump.

While the latter definition is very general, typical implementations are export functions to store data to a storage device, e.g., as a csv or json file, or to transfer tables from one database to another one using, e.g., SQL. Note that the second computer program may also be the operating system. Hence, a data dump is/should be used upon termination of a (terminating) simulation. While it is also possible to execute a data dump while the simulation is running, it is quite uncommon as it results in unnecessary workload. During runtime of a simulation, other forms of output are used.

**Definition 3.9 (Widget).**
Given a replication $\mathcal{R}$ within a computer program. Then we call a transfer of the replication to a user interface a widget.
Hence, a widget refers to a textual or graphical interface. Note that also a console is a widget. Based on the latter two definitions, we can extend our sketch of a simulation UML diagram, cf. Figure 3.2.

Apart from accessibility, several other steps for data processing exist:

**Definition 3.10** (Data processing).
Consider a replication $\mathcal{R}$ to be given by a computer program. Then possible processing steps include

- Saving — making a replication accessible,
- Validation — ensuring correctness and relevance of data,
- Sorting — arranging data according to an order or in sets,
- Summation — reducing data to their statistical properties,
- Aggregation — combining specific data,
- Classification — separating data by properties, and
- Reporting — listing data or results of processing steps.

The data processing parts are quite fundamental but serve as foundation for so called data analytics used to gain insights from data to support business, social or technical processes.

**Definition 3.11** (Data analytics).
Given a replication $\mathcal{R}$ by a computer program, data analytics refers to steps including

- Inspecting — detecting corrupt / inaccurate / incomplete data,
- Cleaning — deleting, replacing, modifying or completing data,
- Transforming — converting data into formats / structures, and
- Modeling — defining and analyzing data requirements.

Within this lecture, we only deal with basic methods from data processing. In particular, we face two types of results emanating from terminating and non-terminating simulations: In case of a terminating simulation, the replication is a finite sequence of vectors \( \{x_j(t_i)\} \), \( i, j \in I \) whereas the sequence is infinite for non-terminating simulations.

There are several options to sort and report data. The most simple one is a table, cf. Table 3.1.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>−6</td>
<td>−2</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>−6</td>
<td>−3</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>−6</td>
<td>−3</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>−5</td>
<td>−2</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>−5</td>
<td>−2</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>−5</td>
<td>−2</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>−5</td>
<td>−3</td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>−5</td>
<td>−3</td>
<td>7</td>
</tr>
<tr>
<td>21</td>
<td>−5</td>
<td>−3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>−4</td>
<td>−2</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>−4</td>
<td>−2</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>−4</td>
<td>−3</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>−2</td>
<td>−2</td>
<td>13</td>
</tr>
<tr>
<td>23</td>
<td>−2</td>
<td>−3</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>−2</td>
<td>−3</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>−2</td>
<td>−4</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>−1</td>
<td>−1</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>−1</td>
<td>12</td>
</tr>
</tbody>
</table>
By including knowledge such as \( t \) denoting time, which can therefore be used as sort key, we can apply the data processing step *sorting* to Table 3.1 and obtain the sorted results given in Table 3.2.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>-1</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>-1</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>-1</td>
<td>-1</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>-2</td>
<td>-2</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>-4</td>
<td>-2</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>-4</td>
<td>-2</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>-5</td>
<td>-2</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>-5</td>
<td>-2</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>-5</td>
<td>-2</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>-6</td>
<td>-2</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>-6</td>
<td>-3</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>-6</td>
<td>-3</td>
<td>9</td>
</tr>
<tr>
<td>19</td>
<td>-5</td>
<td>-3</td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>-5</td>
<td>-3</td>
<td>7</td>
</tr>
<tr>
<td>21</td>
<td>-5</td>
<td>-3</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>-4</td>
<td>-3</td>
<td>6</td>
</tr>
</tbody>
</table>
3.2 Processing of Results

<table>
<thead>
<tr>
<th>Table 3.2 – continued from previous page</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
</tbody>
</table>

The data contained in that table can also be arranged as points in a so called point cloud.

**Definition 3.12** (Point cloud).

Given a replication $\mathcal{R}$, a *point cloud* is a widget aligning selected data as dots in a coordinate system.

**Task 3.13** (Point cloud)

*Use the state coordinates $x_1$, $x_2$, $x_3$ from the replication $\mathcal{R}$ given in Table 3.1 to plot a point cloud in Cartesian coordinates.*

**Solution to Task 3.13**: The respective point cloud is given in Figure 3.3.

![Figure 3.3: Point cloud of three time series from Figure 3.4](image)

While we can already obtain a reporting from both table and point cloud, it does not provide us with any additional information. One step towards aggregation is to include time in the visualization.
**Definition 3.14** (Time series / trajectories).
Given a replication \( \mathcal{R} \) including time, we call a widget aligning time \( t \) to respective data \( x(t) \) a *time series*. Linking points of a time series in timely order is called a *trajectory plot*.

**Task 3.15** (Time series)
*Use the state coordinates \( x_1, x_2, x_3 \) from the replication \( \mathcal{R} \) given in Table 3.1 to plot the time series.*

**Remark 3.16**
*A time series may be given by an entry of the state vector \( x = (x_1, \ldots, x_{n_x}) \), but may also correspond to the complete state as an outcome of a simulator \( x \).*

**Solution to Task 3.13**: The respective time series and trajectory plots are given in Figure 3.4.

![Figure 3.4: Sketch of three time series](image)

Within our example, we observe that the different time series seem to diverge to different levels within their subspace. To analyze the latter in particular with respect to convergence of a time
series or its variation, the statistical terms of mean value, sample variance and confidence interval can be applied. The respective information is contained in the aggregation and classification components of data processing.

**Definition 3.17** (Mean, sample variance and confidence interval).

Consider a replication $\mathcal{R} = \{(x_j)_{j \in I}\}$ to be given. Then we call

$$E(\mathcal{R}) := \frac{\sum_{j \in I} x_j}{\#I}$$

(3.2)

**mean or expected value** of $\mathcal{R}$. Moreover, we call

$$S(\mathcal{R}) := \frac{\sum_{j \in I} (x_j - E(\mathcal{R}))}{\#I - 1}$$

(3.3)

**sample variance** and

$$I(\mathcal{R}) := \left[ E(\mathcal{R}) \pm t(\#I - 1, 1 - \alpha/2) \frac{S(\mathcal{R})}{\sqrt{\#I}} \right]$$

(3.4)

**confidence interval** where $t(\#I - 1, 1 - \alpha/2)$ is the critical value of the t-distribution (or z-distribution) with confidence level $1 - \alpha/2$. We call

$$Q_1(\mathcal{R}) := E(\{x_j\} \mid x_j \text{ is in the lowest } 25\% \text{ of } \mathcal{R})$$

(3.5)

$$Q_3(\mathcal{R}) := E(\{x_j\} \mid x_j \text{ is in the highest } 25\% \text{ of } \mathcal{R})$$

(3.6)

**first and third quartile**. Last, we call any element of $\mathcal{R}$ satisfying

$$x_j \notin [Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$$

(3.7)

for $k > 0$ an **outlier**.

**Remark 3.18**

*There is no common definition of an outlier. Here, we use the so called Tukey’s fence.*

The latter components are combined in a so called boxplot.
**Definition 3.19** (Statistical plots).
Given a replication $\mathcal{R}$, a boxplot is a five number summary containing mean, minimum and maximum as lines and first and third quartile marking a box. Outlier may be added as points.

**Task 3.20** (Boxplot)
Compute boxplots for the states $x_1$, $x_2$, $x_3$ from the replication $\mathcal{R}$ given in Table 3.1.

**Solution to Task 3.20:** The respective boxplots are given in Figure 3.5.

![Figure 3.5: Box plot of three time series from Figure 3.4](image)

As a last idea of data processing, we include visualization using a representation of the system at hand, e.g. the models of the mass-spring system from Section 1.4, cf. Figure 1.7, or the quartercar, cf. Figure 1.8. For these cases, the results from the simulation $\mathcal{R}$ must be mapped to factors contained in the graphical models. The advantage of such a visualization is that the implications on reality, and in particular of dependencies of parameters and scenarios is intuitively accessible. Figure 3.6 provides an overview.

Unfortunately, the dependencies are in many cases hidden, but can be calculated using the approach of sensitivities, which we address next.
3.3 Sensitivity

In many applications, it is essential to find out how the choice of inputs affects the outputs. To assess this effect analytically, the idea of sensitivities can be applied at least to those inputs, which can be varied continuously.

**Definition 3.21** (Sensitivity). Consider a simulation $\Theta : \mathcal{T} \times \mathcal{T} \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P} \rightarrow \mathcal{X}$.

Furthermore suppose the inputs $(t_0, x(t_0), u(\cdot), p) \in \mathcal{T} \times \mathcal{X} \times \mathcal{U}^T \times \mathcal{P}$ to be fixed and the sets $\mathcal{T}, \mathcal{X}, \mathcal{U}^T$ and $\mathcal{P}$ to be continuous. Then we call

$$\frac{\partial x_j}{\partial t_0}(t_i), \quad \frac{\partial x_j}{\partial x(t_0)}(t_i), \quad \frac{\partial x_j}{\partial u(\cdot)}(t_i), \quad \text{and} \quad \frac{\partial x_j}{\partial p}(t_i)$$

(3.8)

sensitivity of the output $x_j(t_i)$ with respect to the inputs.

The idea of sensitivities is to assess the impact of an input change on the output change, i.e. whether the effect decreases the output, leaves it unchanged, increases it, structurally changes it or bifurcates it.

**Task 3.22**

Describe cases for input/output changing effects.

**Solution to Task 3.22**: A decreasing effect is given, e.g., for dampers. Unchanged output refers to independence of output from input. An increase occurs for a gain, e.g., via an electric amplifier. Structural changes may occur, e.g., if parameters affecting poles shift the latter from the negative to the positive complex halfplain. Last, bifurcation may occur if a
change of input results in a modification of equilibria and the solution converges to a different equilibrium.

**Remark 3.23**

In case \( T, X, U^T \) and \( \mathcal{P} \) are not continuous, there exists no derivative and hence this idea cannot be followed.

By definition, the sensitivity is locked to the chosen values of the inputs \((t_0, x(t_0), u(\cdot), p) \in T \times X \times U^T \times \mathcal{P}\), yet it may serve as a pointwise description, i.e. for fixed \( t_i \), or as a field, i.e. along the time series \((t_i)_{i \in \mathbb{T}}\). In practice, we will not compute the derivatives of the outputs but instead use an approximation. To this end, at least two simulation outputs for an input variation are required.

**Definition 3.24** (Numerical sensitivity).

Consider a simulation \( \Theta : \mathcal{T} \times \mathcal{T} \times X \times U^T \times \mathcal{P} \to X^T \) and suppose two inputs \((t_{0,1}, x_1(t_{0,1}), u_1(\cdot), p_1), (t_{0,2}, x_2(t_{0,2}), u_2(\cdot), p_2) \in \mathcal{T} \times X \times U^T \times \mathcal{P}\) to be given where \( \mathcal{T}, X, U^T \) and \( \mathcal{P} \) are continuous. Then we can use

\[
\frac{\partial x_j}{\partial t_0}(t_i) \approx \frac{x_{j,1}(t_i) - x_{j,2}(t_i)}{t_{0,1} - t_{0,2}}
\]

\[
\frac{\partial x_j}{\partial x(t_0)}(t_i) \approx \frac{x_{j,1} - x_{j,2}(t_i)}{x_1(t_{0,1}) - x_2(t_{0,2})}
\]

\[
\frac{\partial x_j}{\partial u(\cdot)}(t_i) \approx \frac{x_{j,1}(t_i) - x_{j,2}(t_i)}{u_1(\cdot) - u_2(\cdot)}
\]

\[
\frac{\partial x_j}{\partial p}(t_i) \approx \frac{x_{j,1}(t_i) - x_{j,2}(t_i)}{p_1 - p_2}
\]

as an approximation of the sensitivities provided the distances between the elements of respective inputs are small enough and the remaining inputs are equivalent.

**Remark 3.25**

Note that sensitivities have got nothing to do with statistics, i.e. the results is purely deterministic and not stochastic.

Based on the numerical approximation of sensitivities, we now discuss the possible outcomes of such an analysis more closely. In applications, the above mentioned cases are of very different
interest. Here, we highlight the implications but the background is beyond the scope of this lecture:

- Decreasing or unchanged output can be seen as robust behavior, yet to some extend allows us to ignore changes regarding these inputs.

- Increasing output indicates that the input change must be limited, yet as long as no structural or bifurcation changes occur may be tolerated.

- Structural changes reflect that properties of the behavior of the system are changed. For simulation, it is of particular interest to find those point of input changes, which form the boundary between two structures.

- Bifurcation changes indicate changes in properties of the system, not its behavior. Similar to structural changes, the switching points are of interest.

Utilizing the latter, we introduce the so called scenario analysis.

**Definition 3.26** (Scenario analysis).

A scenario analysis is given by a simulation $\Theta : \mathcal{T} \times \mathcal{T} \times \mathcal{X} \times U^T \times \mathcal{P} \rightarrow \mathcal{X}^T$ and three inputs reflecting

- the worst case,
- the best case, and
- the base case which reflects the most likely scenario.

Scenario analysis is focused on finding those switching values, which lead to a change from base to worst or to best case, and to identify the tolerable size of deviations.


During summer term 2023 I give the lecture to the module *Simulation of mechatronic systems* (*Simulation mechatronischer Systeme*) at the Technical University of Braunschweig. To structure the lecture and support my students in their learning process, I prepared these lecture notes.

The aim of the module is to classify simulation techniques from numerical mathematics and apply these to mechatronic case studies. After completing the module, the students shall be able to recall, categorize, apply, select and rate simulation methods to mechatronic use cases. Moreover, students shall be able to describe, explain, evaluate, analyze and assess simulation results. As such, students shall be capable to derive and apply automation procedures for deployment, simulation and testing of digital models.