UNIVERSITÄT TRIER

# Hybrid Modelling of Dynamical Systems in Mechanics

**DISSERTATION**

**zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)**

**vorgelegt am Fachbereich IV - Mathematik der Universität Trier von**

**Jan Sokolowski**

**Pellingen / Trier / Bissen im Juni 2022**

| | |
|---|---|
| **Gutachter:** | **Prof. Dr. Volker H. Schulz** |
| | **Prof. Dr. Nicole Marheineke** |

“*Newton was the greatest genius that existed, and the most fortunate, for we cannot find more than once a system of the world to establish.*”

Joseph-Louis Lagrange

“*Ce que nous connaissons est peu de chose, ce que nous ignorons est immense.*”

Pierre-Simon Laplace

“*Meine Ergebnisse habe ich schon, ich weiß nur noch nicht, auf welchem Wege ich zu ihnen gelangen werde.*”

Carl Friedrich Gauß

“*Just go on... And faith will soon return.*”

Jean-Baptiste le Rond d’Alembert

“*Es ist nicht das Wissen, sondern das Lernen, nicht das Besitzen, sondern das Erwerben, nicht das Dasein, sondern das Hinkommen, was den größten Genuß gewährt.*”

Carl Friedrich Gauß

UNIVERSITÄT TRIER

# *Abstract*

Fachbereich IV
Mathematik

**Hybrid Modelling of Dynamical Systems in Mechanics**

by Jan Sokolowski

Hybrid Modelling in general, describes the combination of at least two different methods to solve one specific task. As far as this work is concerned, Hybrid Models describe an approach to combine sophisticated, well-studied mathematical methods with Deep Neural Networks to solve parameter estimation tasks. To combine these two methods, the data structure of artificially generated acceleration data of an approximate vehicle model, the Quarter-Car-Model, is exploited. Acceleration of individual components within a coupled dynamical system, can be described as a second order ordinary differential equation, including velocity and displacement of coupled states, scaled by spring - and damping-coefficient of the system. An appropriate numerical integration scheme can then be used to simulate discrete acceleration profiles of the Quarter-Car-Model with a random variation of the parameters of the system. Given explicit knowledge about the data structure, one can then investigate under which conditions it is possible to estimate the parameters of the dynamical system for a set of randomly generated data samples. We test, if Neural Networks are capable to solve parameter estimation problems in general, or if they can be used to solve several sub-tasks, which support a state-of-the-art parameter estimation method. Hybrid Models are presented for parameter estimation under uncertainties, including for instance measurement noise or incompleteness of measurements, which combine knowledge about the data structure and several Neural Networks for robust parameter estimation within a dynamical system.

UNIVERSITÄT TRIER

# *Zusammenfassung*

Fachbereich IV
Mathematik

**Hybride Modellierung von Dynamischen Systemen in der Mechanik**

von Jan SOKOLOWSKI

Hybride Modellierung bezeichnet im Allgemeinen die Kombination von mindestens zwei unterschiedlichen Methoden zum Lösen einer speziellen Problemstellung. Soweit es diese Arbeit betrifft, beschreiben Hybride Modelle einen Ansatz, elegante und allseits bekannte mathematische Modelle mit tiefen Neuronalen Netzen zu kombinieren, um Parameterschätzprobleme zu lösen. Um diese beiden Methoden zu verbinden, wird die Datenstruktur von künstlich generierten Beschleunigungsdaten eines vereinfachten Fahrzeugmodells, dem Viertelfahrzeugmodell, ausgenutzt. Beschleunigung der unterschiedlichen Komponenten in einem gekoppelten dynamischen System kann als gewöhnliche Differentialgleichung zweiter Ordnung beschrieben werden und beinhaltet die Geschwindigkeit und die Verlagerung der gekoppelten Zustände, skaliert mit den Feder - und Dämpfungskoeffizienten des Systems. Eine angemessene numerische Integrationsmethode kann benutzt werden, um diskrete Beschleunigungsprofile des Viertelfahrzeuges mit einer zufälligen Variation der Systemparameter zu simulieren. Explizites Wissen über die Datenstruktur gibt Aufschluss darüber, unter welchen Umständen die Parameter des dynamischen Systems, für einen Satz von zufällig generierten Daten, bestimmt werden können. Wir untersuchen, ob Neuronale Netze die Fähigkeit besitzen, Parameterschätzprobleme im Allgemeinen zu lösen oder ob sie dazu benutzt werden können, Standardmethoden für die Parameterschätzung zu unterstützen. Es werden Hybride Modelle für Parameterschätzung unter Unsicherheiten, beispielsweise Messfehler oder Datenunvollständigkeit, vorgestellt, welche Vorwissen über die Datenstruktur und unterschiedliche Neuronale Netztypen für die robuste Parameterschätzung eines dynamischen Systems verbinden.

# *Acknowledgements*

First of all, I would like to thank my thesis advisor, Prof. Dr. Volker H. Schulz, who has always led the research of the thesis into the right direction, such that it has become possible to achieve the results, which are presented in this work. If investigations had shown to be misleading during the progress of this thesis, Volker Schulz was always capable of finding another good approach to bring the research back to course.
Moreover, I would also like to thank Prof. Dr. Nicole Marheineke for taking the time to deal with the thesis and for giving me valuable feedback to improve the content of my work.
Further, I would like to thank Dr. Udo Schröder, who has always been open to discuss the progress of the thesis, on a theoretical as well as on a practical basis, and to discuss possible further steps and investigations. I am thankful to had a lot of good discussions with him, for instance concerning the numerical part of the thesis. Regular discussions have shown to be essential for the thesis to converge to the end.
I would also like to thank Prof. Dr. Hans-Peter Beise, who has always shown to be the right contact to discuss abstract contexts, which have not been obvious for me. Regular discussions to talk about a specific theoretical problem, have been of high importance to clarify the theoretical concepts of the thesis.
Next, I would like to thank Dr. Thomas Stifter, who made it possible for me to have the time I needed to finish the thesis. It was further a benefit for me, to have the opportunity to discuss concepts of mechanics to a physicist.
I would like to thank Steve Dias Da Cruz, who has probably the broadest knowledge about training methods and architectures of Neural Networks, concerning the people I know. Although we have worked on complete different problems, it was always helpful for me to have short discussions with him.
Moreover, I would like to thank Manuel Klar, who helped me especially at the beginning of the thesis to ask the right questions concerning the problems, I wanted to solve. He has always found the time to discuss and talk about barriers that have been found, when defining the direction of the thesis. Thanks to Dominik Annen, to let me draw attention to several spelling and typing mistakes.
In addition, I would like to thank the company IEE S.A., especially the Basics and Mathematical Models department, to make this collaboration possible and to offer me the opportunity to write a PhD thesis in this specific field.
I would like to thank the Research Training Group on Algorithmic Optimization (ALOP) of the Trier University, where I have learned to present and discuss recent results of my research. Discussions with other PhD students throughout different fields of mathematical research has improved my way of handling the problems of my thesis.
Finally, I would also like to thank Marcel Dawen, who is a friend of mine for more than twenty years now and gave me mental support, especially throughout the last two years and the final days, that was required to keep on and finish this thesis.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ADAM** | **Ada**ptive **M**omentum |
| **ANN** | **A**rtificial **N**eural **N**etwork |
| **BPA** | **B**ack **P**ropagation **A**lgorithm |
| **CAE** | **C**onvolutional **A**uto - **E**ncoder |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **DNN** | **D**eep **N**eural **N**etwork |
| **DOF** | **D**egree **O**f **F**reedom |
| **ERM** | **E**mpirical **R**isk **M**inimization |
| **FC** | **F**ully **C**onnected |
| **FCM** | **F**ull **C**ar **M**odel |
| **GAN** | **G**enerative **A**dversarial **N**etwork |
| **HCM** | **H**alf **C**ar **M**odel |
| **IVP** | **I**nitial **V**alue **P**roblem |
| **LTI** | **L**inear **T**ime **I**nvariant |
| **MIMO** | **M**ultiple **I**nput **M**ultiple **O**utput |
| **MISO** | **M**ultiple **I**nput **S**ingle **O**utput |
| **MSD** | **M**ass **S**pring **D**amper |
| **MSE** | **M**ean - **S**quared - **E**rror |
| **ODE** | **O**rdinary **D**ifferential **E**quation |
| **PCA** | **P**rincipal **C**omponent **A**nalysis |
| **PDE** | **P**artial **D**ifferential **E**quation |
| **PSD** | **P**ower **S**pectral **D**ensity |
| **QCM** | **Q**uarter **C**ar **M**odel |
| **ReLU** | **Re**ctified **L**inear **U**nit |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **SGD** | **S**tochastic **G**radient **D**escent |
| **SIMO** | **S**ingle **I**nput **M**ultiple **O**utput |
| **SISO** | **S**ingle **I**nput **S**ingle **O**utput |
| **SLT** | **S**tatistical **L**earning **T**heory |
| **VAE** | **V**ariational **A**uto - **E**ncoder |
| **l.h.s.** | **l**eft **h**and **s**ide |
| **r.h.s.** | **r**ight **h**and **s**ide |
| **w.r.t.** | **w**ith **r**espect **t**o |
| **w.l.o.g.** | **w**ithout **l**oss **o**f **g**enerality |

# List of Symbols

**Sets**

| | |
|---|---|
| $\mathbb{N}$ | set of natural numbers |
| $\mathbb{R}$ | set of real numbers |
| $\mathbb{C}$ | set of complex numbers |

**Set Theory**

| | |
|---|---|
| $a \in A$ | element $a$ member of set $A$ |
| $\mathbb{1}_{\{a\in A\}}(a)$ | indicator function if $a$ is member of $A$ |
| $A \subset B$ | $A$ is proper subset of $B$ |
| $A \subseteq B$ | $A$ is subset of $B$ or equal to $B$ |
| $A \setminus B$ | elements in $A$ and not in $B$ |
| $\bigcap_{i=1}^{m} A_i$ | intersection of all sets $A_i$ |
| $\bigcup_{i=1}^{m} A_i$ | union of all sets $A_i$ |

**Relations**

| | |
|---|---|
| $a = b$ | $a$ equal to $b$ |
| $a \approx b$ | $a$ is approximately $b$ |
| $a \neq b$ | $a$ not equal to $b$ |
| $a > b$ | $a$ greater than $b$ |
| $a \gg b$ | $a$ much greater than $b$ |
| $a \geq b$ | $a$ greater or equal to $b$ |
| $a < b$ | $a$ less than $b$ |
| $a \ll b$ | $a$ much less than $b$ |
| $a \leq b$ | $a$ less or equal to $b$ |
| $a := G(x)$ | $a$ defined as expression $G(x)$ |

**Indexing**

| | |
|---|---|
| $a_k$ | $k$-th entry of vector $a$ |
| $a_{m;k}$ | $k$-th entry of vector $a_m$ |
| $a^l$ | $l$-th iteration or time-index |
| $a^{(l)}$ | value depending on layer $l$ |
| $A_{ij}$ | $i$-th row and $j$-th column entry of matrix $A$ |

**General Operators**

| | |
|---|---|
| $\lvert a \rvert$ | absolute value of $a$ |
| $\lVert a \rVert$ | Euclidean norm of $a$ |
| $\lfloor a \rfloor$ | largest integer below $a$ |

| | |
|---|---|
| $\overline{a}$ | complex conjugate of $a$ |
| $\mathfrak{Re}(a)$ | real part of $a$ |
| $\mathfrak{Im}(a)$ | imaginary part of $a$ |

**Differentiation and Integration**

| | |
|---|---|
| $\frac{\mathrm{d}^k f(t)}{\mathrm{d}t^k}$ | $k$-th total derivative of a function $f$ with respect to time variable $t$ |
| $\frac{\partial^k f(t,x)}{\partial x^k}$ | $k$-th partial derivative of a function $f$ with respect to variable $x$ |
| $\nabla_x F(x)$ | Gradient of a function $F$ with respect to variable $x$ |
| $\mathscr{C}^k(X,Y)$ | function space of $k$-times continuously differentiable functions from space $X$ to space $Y$ |
| $\mathscr{L}^1(X,Y)$ | function space of continuously integrable functions from space $X$ to space $Y$ |

**Probability Theory**

| | |
|---|---|
| $\mathbb{E}_{X\sim p_X}[X]$ | expected value of a random variable $X$ with distribution $p_X$ |
| $\mathrm{Var}[X]$ | variance of a random variable $X$ |
| $\mathscr{U}([a,b])$ | uniform distribution on interval $[a,b]$ |
| $\mathscr{N}(\mu,\sigma^2)$ | Gaussian distribution with mean $\mu$ and variance $\sigma^2$ |

*Dedicated to my parents, Monika and Joachim*
*and to my godchild, Quentin.*

# Chapter 1

# Introduction

## 1.1 Motivation

Individual passenger transport has become one of the most controversially discussed topics in recent years. Balancing professional, familial and private life can require high personal flexibility, if large distances separate working place, home and locations of leisure activities, since there is no area-wide and time-independent solution of public transport, which offers to manage the daily challenge of limited time. Some of the outcomes of this individual transport problem, are overfilled roads and traffic jams, above all in border regions and metropolitan areas. It is nearly impossible to achieve a significant reduction of the daily traffic, if one considers rural regions, with a non-existent rail-network, combined with a really sparse bus connection. Working from home, in the COVID-19 global pandemic, has shown one possibility to connect working place and private life, but is nevertheless only a compromise solution for the main problem.

To overcome the passenger transport problem in future and to connect rural areas with urban centres, autonomous solutions for public transport are currently tested to achieve a higher individual flexibility, without being restricted to fixed travel times. With no doubt, autonomous driving is not yet globally applicable to solve these problems within the following decades, due to high requirements in passenger safety and reliability of decision systems, which process and evaluate measurements of a large quantity of sensing devices.

Decision making for smart systems, requires interaction with sensing devices of a car interior and as far as autonomous driving is concerned, of course also interaction with exterior vehicles. It is therefore more than necessary, to have redundant devices, if measurement errors or inferences disturb the sensors. Thus, it is not surprising, that the amount of built-in sensing devices has grown significantly within the last years. If it is not pure autonomous driving by now, smart systems can already be used to enable driving assistance to support human interaction with the environment and to ensure passenger safety in case of a possible collision.

## 1.2 Automotive Sensors

Modern research about the developments of sensing devices has shown respective progress in vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) interaction with different types of sensors [49]. For instance, range detection technologies have been developed, using laser devices to produce point-clouds for obstacle detection, combined with computer vision software to calculate distances from vehicle to exterior infrastructure.

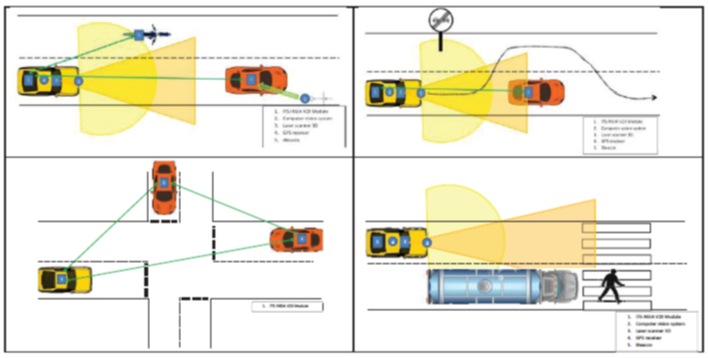


FIGURE 1.1: Possible situation of range detection as described by [49].

One prominent version of this laser technology are for instance LiDAR (Light Detection And Ranging) systems, where distance results from measuring time, light needs to be shot onto an object and to be reflected back to the antenna of the signal's source [90]. Besides other range finders, like Complementary Metal-Oxide Semi-conductor (CMOS) image sensors [48], modern RADAR technology, like Frequently Modulated Continuous Wave (FMCW) or Continuous Wave (CW), can also be used to measure distances, for instance between vehicles and pedestrians, making use of the Doppler-spectrum [8]. In summary, there is a large field of possible sensing devices to measure the distance of a vehicle to the environment, including pedestrians, moving vehicles or solid objects. Nevertheless, also vehicle-interior sensors can be used to support the decision making process of these range detection sensors.

For modern Advanced Driver Assistance Systems (ADAS), it is also relevant to detect unnatural behaviour of the human driver, since driver distraction is one of the main collision reasons. It is therefore not only sufficient to ensure the airbag function in case of a crash [95, 116], but to have a reliable early warning system to prevent possible collisions. Besides the areas of powertrain control systems to observe vehicle energy use or driving performance and chassis control systems for steering, suspension or vehicle stability, body control systems are relevant to ensure occupant safety, security and comfort for the passengers [26]. For these body control systems, piezoresistive (semi-conductor-based) sensors [25] can for instance be used to convert pressure into electrical signals, to ensure functionality of the airbag control.

Furthermore, Inertial Measurement Units (IMU) can explicitly be used to measure an object displacement, investigating a rate of change in the orientation of a body [46, 81]. These IMUs combine sensors like accelerometers (g-Sensors) or gyroscopes and can for instance be contained in so called Microelectromechanical Systems (MEMS), which give the opportunity to build complex sensing systems by small devices in millimeter-size [50, 71, 97]. Modern MEMS technology, uses WiFi-connections, which does not require permanent installation of a device at a local position within a vehicle.

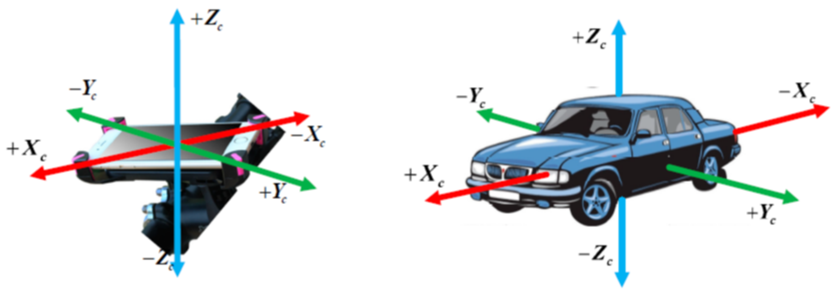


FIGURE 1.2: Orientation of two g-Sensors, one measuring the interior acceleration of a smartphone, the other measuring the acceleration of the chassis [115]. Both measurements can be compared, due to the identical orientation of the sensors.

Raw data of accelerometers, as component of IMUs or MEMS, can easily be acquired, since they are nowadays included in common devices like smartphones and can therefore be used to measure the relative displacement of a passenger with respect to a corresponding vehicle movement, by use of three-dimensional acceleration data, as it can be seen in Figure 1.2.

Overall, it is possible to acquire large amounts of data for the body control system, which can then be processed to give insights about the physical or mental condition of a vehicle's occupant. Due to the broad availability of g-Sensors, it is worth investigating what could be achieved via processing this simply acquirable data with Artifical Intelligence (AI).

## 1.3 Hybrid Neural Networks and Parameter Estimation

Neural Networks can be used to process big data, including time-series forecasting problems [23], and give insights about a priori unobservable inner data structures. Since Neural Networks have shown to be unstable for uncertainties and as a high redundancy is required for decision making processes, one should investigate robust architectures, when trying to include data-driven models, as it is the case for Hybrid Neural Networks.

Hybrid Neural Networks can efficiently be used to support control systems like energy and building management [55]. It can then be shown, that Neural Networks can serve as successful early warning systems, when forecasting anomalies in a given time-series of measurement data. Furthermore, partial knowledge about the underlying physical systems [21] can be exploited to develop robust Hybrid Models, which outperform pure data-driven models, for instance in parameter estimation problems or system identification.

There is no unique definition of the term "Hybrid Model", as they in general are capable to contain combinations of data-driven methods [113], or a mixture of data-driven and deterministic methods. We restrict to Hybrid Models, containing Neural Networks, that are capable to solve parameter estimation problems for dynamical systems [86], especially for approximate vehicle models.

Parameter estimation for vehicle models is a broad field, mainly focussing on motion control for automated driving [99, 110] or vehicle detection [66, 98]. The scope of this thesis is to focus on another field, namely parameter estimation for components of the car interior. Therefore, estimation methods with respect to robustness against measurement noise or data incompleteness are going to be investigated. By our knowledge, there are only a few examples of identification of an explicitly described vehicle model. Most investigations are restricted to sprung mass identification [4] or identification of the corresponding stiffness parameters of the chassis [16].

There is no approach, that makes use of a more complex passenger-based vehicle model for parameter estimation problems. Therefore, we are going to develop data-driven as well as common deterministic models for parameter estimation of such vehicle models, evaluating the performance in terms of robustness against additional measurement noise and incompleteness of the observed data. We want to answer the questions, if specific Neural Network architectures are capable to process discrete acceleration data of a dynamical system to give an appropriate estimate of the target parameters. In addition, it is shown that an appropriate estimate of the system parameters for heavily corrupted data or incomplete data is difficult and therefore one needs to make use of more sophisticated Hybrid Models.

## 1.4 Content

Chapter 2 contains an introduction into mathematical modelling for coupled dynamical systems. Therefore, Newton's laws of motion are used to define the kinetic and potential energy of a conservative system. The Euler-Lagrange equation can then be used to derive ordinary differential equations from the energy terms. Inclusion of the Rayleigh-Dissipation is then used to extend the method to non-conservative systems, which contain an exterior force that reacts to the components of the coupled systems. Considering rotational and translational energy terms, as for a pendulum model, approximate vehicle models like the Quarter-Car-Model (QCM), the Half-Car-Model (HCM) and the Full-Car-Model (FCM) can be derived from the Euler-Lagrange formalism. At a last step, a sinusoidal approach is introduced to generate random road-profiles, which are used as source term for the coupled dynamical systems.

Chapter 3 comprises a general introduction to the training of Neural Networks. At a first step, the general terminology, as it is shown in Statistical Learning Theory (SLT) is clarified. The training problem for Neural Networks can then be defined by finding an approximative solution to the Empirical Risk Minimization (ERM) problem. The second part of the section is more technical, as it gives an introduction to the layer arithmetic of specific Neural Network architectures. As far as our investigations are concerned, we need three different structures of Neural Networks, namely Convolutional Neural Networks (CNN) combined with Fully-Connected (FC) layers, Convolutional Auto-Encoders (CAE) and the so called U-Net, as a special type of a Convolutional Neural Network, containing skip-connections between the layers.

Chapter 4 deals with the solution of ordinary differential equations to generate artificial data for coupled dynamical systems. A short introduction into the solution theory of second order ordinary differential equations (ODE) will be given as a first step. Therefore, homogeneous and non-homogeneous equations are considered. Clarification about Hamiltonian systems, which can be connected to the Euler-Lagrange formalism of Chapter 2, then leads to a geometric integration scheme, which can efficiently be used to find appropriate solutions of the coupled system of second order ordinary differential equations. Afterwards, the system of second order ordinary differential equations is transferred into a system of first order equations, which can then easily be solved with different appropriate Euler integration schemes. A random variation of the parameter values and of the road-profiles can then be used to generate datasets of arbitrary size, which serve as simulated measurements for the interior of the Quarter-Car-Model.

Chapter 5 deals with system identification and parameter estimation of dynamical systems. The task is to estimate the parameters from several data samples, which have randomly been generated in Chapter 4. The first part will clarify the conditions, on which all parameters of the system can uniquely be determined. Then several experiments are described, which deal with parameter estimation of the Quarter-Car-Model. Therefore, it is shown, how Neural Networks and state-of-the-art methods can appropriately be combined to define Hybrid Models, which give sufficient estimates of the parameters. We therefore start with a simple, data-driven experiment, where it is tested, whether Neural Networks are capable to predict a two-dimensional parameter estimation problem appropriately. One can then compare the data-driven approach with an unlabelled objective function, using the same network architecture as for the first experiments. The two methods can then be compared in terms of robustness against Gaussian noise and generalization performance for the test samples. Further, one can test, if Neural Networks can also be used for denoising of corrupted sequential input data. In contrast to the two-dimensional parameter estimation problem, we then investigate parameter estimation for the entire Quarter-Car-Model. We will show, under which conditions it is possible to estimate all identifiable parameters of the system. Finally, we make use of several different network architectures to develop robust Hybrid Models for parameter estimation in an uncertain system.

Chapter 6 summarizes the main achievements of the work. It is further discussed what can be learned from the results shown in the previous section. Specific questions occur from the shown results, which could be investigated in future research.

Appendix A contains source code of the most important functions for parameter estimation and the training of Neural Networks, which has been used for the experiments of Chapter 5.

Appendix B shows the specific layer structure of the three different Neural Networks, which have been considered in this work.

# Chapter 2

# Mathematical Modelling

The following chapter deals with the problem of appropriately deriving a mathematical model of a coupled dynamical system from Newton's laws of motion. Therefore, we discuss in a first step, how one can derive equations of motion from the Euler-Lagrange formalism, as part of variational calculus.

Defining kinetic, potential and exterior energy terms, one can derive a mathematical description of any dynamical system, following the Euler-Lagrange approach. We explain the principle for conservative as well as for non-conservative systems. The derivation of a passenger-based Quarter-Car-Model, Half-Car-Model and Full-Car-Model, following the Euler-Lagrange formalism, is shown in detail in this chapter.

As a last step, we explain the idea of randomly generating road-profiles, following ISO 8606 road roughness classification, which we use to serve as exterior control of the derived approximate vehicle models.

## 2.1 Modelling with Physical Equations

We mainly focus on the concepts stated in [32, 54, 93], which serve as a general introduction into the field of mechanics, to derive mathematical models from principles of classical mechanics. Force equations within a mechanical system can appropriately be used to derive mathematical equations from energy terms of particles for a system of arbitrary dimension. These concepts can be applied to a system of coupled rigid masses, instead of free mass particles, connected by springs and dampers, serving as a simplification of a vehicle model for further investigations.

As far as mechanical systems are concerned, main principles can be derived from relations between force, work and power. The standardized units are given by $N = \frac{kg \cdot m}{s^2}$ (Newton) for force, $J = \frac{kg \cdot m^2}{s^2}$ (Joule) for work and $W = \frac{kg \cdot m^2}{s^3}$ (Watt) for power. Moreover, the concepts can mathematically be connected, as power describes the rate of change of work over time (derivative) and work describes the amount of power transferred from one point to another (integral). These basic dependencies will serve as fundamental properties of mathematical modelling in the following sections.

### 2.1.1 Motion of Particles and Newton's Laws

Referring to [54] in classical mechanics, the concepts of space and time are in general continuous problems. This can mathematically be experienced, when dealing with partial differential equations (PDE) which contain time and state derivatives. We can observe motion of individual or coupled objects with respect to time in classical mechanics. Theoretical concepts can then be derived from Newton's laws of motion, if we assume rigid bodies to be treated as point particles with centered mass. Therefore, we start with a system of arbitrary particles following [32].

We define a finite time horizon $\mathscr{T} := [t_0, t_n] \subset \mathbb{R}$ for which the $d$ - dimensional position vector or radius vector, with $d \in \mathbb{N}$, can be described as a twice continuously differentiable function $r_i \in \mathscr{C}^2(\mathscr{T}, \mathbb{R}^d)$, for a particle with index $i \in \{1, 2, ..., N\}$ and $N \in \mathbb{N}$. Further, the velocities of a particle at time $t \in \mathscr{T}$ are derivatives of the position with respect to time by

$$v_i(t) := \frac{\mathrm{d}r_i(t)}{\mathrm{d}t} = \dot{r}_i(t), \tag{2.1}$$

where we frequently use Newton's notation $\dot{r}_i(t)$ for derivatives, as far as dynamical systems are considered. Since all state vectors are twice differentiable functions with respect to time, we can also define acceleration of particles by

$$a_i(t) := \frac{\mathrm{d}v_i(t)}{\mathrm{d}t} = \frac{\mathrm{d}^2 r_i(t)}{\mathrm{d}t^2} = \ddot{r}_i(t). \tag{2.2}$$

As higher order derivatives are not considered in this context, we can now define Newton's laws of motion for a system of particles with constant, time-independent centered masses [32, 54].

**The first law** (*"principle of inertia"*) [32, 54] states that if there are no forces acting on a particle, it has constant velocity at that time. For each particle indexed by $i \in \{1, 2, ..., N\}$, the first law can mathematically be described by

$$F_i(t) = \sum_{j=1}^{N} F_{ij}(t) + F_i^e(t) = 0 \quad \Leftrightarrow \quad \frac{\mathrm{d}v_i(t)}{\mathrm{d}t} = 0, \tag{2.3}$$

where $F_{ij}(t)$ describes the internal force of the system, acting from particle with index $i$ in direction to particle $j$ for $i, j \in \{1, 2, ..., N\}$ and $N \in \mathbb{N}$. Furthermore, we denote by $F_i^e(t)$ a possible external force acting in direction to particle $i \in \{1, 2, ..., N\}$. Then $F_i(t)$ is the total resulting force for a particle with index $i \in \{1, 2, ..., N\}$ at time $t \in \mathscr{T}$.

**The second law** (*"principle of linear momentum"*) [32, 54] states that if the momentum of a particle with mass $m_i > 0$ for $i \in \{1,2,...,N\}$ is given by

$$p_i(t) := m_i v_i(t), \tag{2.4}$$

then the total force $F_i(t)$ equals the derivative of linear momentum with respect to time

$$F_i(t) = \frac{\mathrm{d}p_i(t)}{\mathrm{d}t} = m_i \frac{\mathrm{d}v_i(t)}{\mathrm{d}t} = m_i \dot{v}_i(t). \tag{2.5}$$

**The third law** (*"principle of actio and reactio"*) [32, 54] states that all particles of a system cause individual forces to the other particles, where $F_{ij}(t)$ describes the force acting from particle $i$ to particle $j$ and $F_{ji}(t)$ describes the force acting from particle $j$ to particle $i$. It then holds that

$$F_{ij}(t) = -F_{ji}(t) \tag{2.6}$$

for all $i, j \in \{1,2,...,N\}$ and thus we have $F_{ii}(t) = 0$.

We have shortly introduced a formal description of Newton's laws of motion with Eq. (2.3), Eq. (2.5) and Eq. (2.6) within an arbitrary particle system. The equations are necessary to derive the Euler-Lagrange formalism in a latter section, after having clarified the general concept of variational calculus.

### 2.1.2 Calculus of Variations

Since differential equations can be used to describe the behaviour of a system depending upon slight variations of state and time, we give a short example about the calculus of variations, which gives insights how to derive the general Euler-Lagrange equations of motion. Variational priniples based on classical mechanics can again be found in [32, 54], where a general theoretical introduction is given by [29, 67]. Principles of variational calculus with respect to physical or engineering problems are well described by [112].

**Definition 1** (Variation of State and Velocity [67])
*The position of a particle with dimension $d \in \mathbb{N}$ at time $t \in [t_0, t_n] \subset \mathbb{R}$ is given by $r \in \mathscr{C}^2([t_0,t_n], \mathbb{R}^d)$. The* ***variational state*** *vector $\rho \in \mathscr{C}^2([t_0,t_n] \times \mathbb{R}, \mathbb{R}^d)$ is given by*

$$\rho(t, \varepsilon) := r(t) + \varepsilon \eta(t) \tag{2.7}$$

*and the* ***variational velocity*** *analogously as*

$$\dot{\rho}(t, \varepsilon) := \dot{r}(t) + \varepsilon \dot{\eta}(t), \tag{2.8}$$

*where $\eta(t)$ is called the perturbation at $t \in [t_0, t_n]$ with $\eta \in \mathscr{C}^2([t_0,t_n], \mathbb{R}^d)$ and $0 < \varepsilon \ll 1$, such that the boundary conditions of the variational state $\rho(t_1, \varepsilon) = r(t_1)$ and $\rho(t_n, \varepsilon) = r(t_n)$*

*yield that*

$$\eta(t_1) = \eta(t_n) = 0. \tag{2.9}$$

**Lemma 1** (Fundamental Lemma of Variational Calculus [29])
*Let two continuous real-valued functions be given by* $g, h \in \mathscr{C}^0([t_0, t_n], \mathbb{R})$. *If*

$$\int_{t_0}^{t_n} g(t)h(t)\mathrm{d}t = 0 \tag{2.10}$$

*for all* $h$ *with* $h(t_0) = h(t_n) = 0$, *then it follows that* $g(t) = 0$ *for* $t \in [t_0, t_n]$.

*Proof.* Let us assume that $g(t) > 0$ for $t \in [a,b] \subset [t_0, t_n]$. We can construct a function $h$ such that $h(t) > 0$ for $t \in [t_0, t_n]$ and $h(t_0) = h(t_n) = 0$, for instance by $h(t) = (t - t_0)(t_n - t)$. As $g(t) = 0$ for $t \in [t_0, a] \cup [b, t_n]$, we have

$$\int_{t_0}^{t_n} g(t)h(t)\mathrm{d}t = \underbrace{\int_{t_0}^{a} g(t)h(t)\mathrm{d}t}_{=0} + \int_{a}^{b} g(t)h(t)\mathrm{d}t + \underbrace{\int_{b}^{t_n} g(t)h(t)\mathrm{d}t}_{=0} = \int_{a}^{b} g(t)h(t)\mathrm{d}t \neq 0, \tag{2.11}$$

since $g(t), h(t) > 0$ for $t \in [a,b]$ by construction. This contradicts the assumption, that the integral is supposed to be zero for $t \in [t_0, t_n]$. Therefore, $g(t) = 0$ for $t \in [a,b]$, which completes the proof. □

We consider a continuously differential functional $f \in \mathscr{C}^1\left(\left([t_0, t_n] \times \mathbb{R}^d \times \mathbb{R}^d\right), \mathbb{R}\right)$, defining a mapping $(t, \rho(t,\varepsilon), \dot{\rho}(t,\varepsilon)) \mapsto f(t, \rho(t,\varepsilon), \dot{\rho}(t,\varepsilon))$ for all $t \in [t_0, t_n]$. Then the **cost functional** can be defined via

$$I(\varepsilon) := \int_{t_0}^{t_n} f(t, \rho(t,\varepsilon), \dot{\rho}(t,\varepsilon))\mathrm{d}t, \tag{2.12}$$

where $\varepsilon \geq 0$ is a **variation rate**. The integral $I(\varepsilon)$ is called a cost functional, due to the fact that it describes the size of the area between a variational path and the optimal path defined by the state vectors $r(t)$ and the velocity $\dot{r}(t)$.

A cost functional $I(\varepsilon)$ is **stationary**, if $I(\varepsilon) = 0$ for $\varepsilon \geq 0$. Considering $\varepsilon = 0$, which implies that $\rho(t, 0) = r(t)$ for all $t \in [t_0, t_n]$, then $I(0) = 0$.

The **variation of the cost functional** is then given by

$$\delta I := \left.\frac{\mathrm{d}}{\mathrm{d}\varepsilon} I(\varepsilon)\right|_{\varepsilon=0} \tag{2.13}$$

and since it is said to be stationary at $\varepsilon = 0$, we need to analyse the equation $\delta I = 0$. As the Leibniz integral rule holds due to the continuity of the functional, we get

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon} I(\varepsilon) = \int_{t_0}^{t_n} \frac{\mathrm{d}}{\mathrm{d}\varepsilon} f(t, \rho(t,\varepsilon), \dot{\rho}(t,\varepsilon))\mathrm{d}t.$$

Computation of the general cost function's total derivative w.r.t. the variation rate $\varepsilon \geq 0$ yields

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\varepsilon} f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon)) &= \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} \frac{\partial \rho(t,\varepsilon)}{\partial \varepsilon} + \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon)} \frac{\partial \dot{\rho}(t,\varepsilon)}{\partial \varepsilon} \\ &= \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} \eta(t) + \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon)} \dot{\eta}(t).\end{aligned}$$

Using integration by parts, we get

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\varepsilon} I(\varepsilon) &= \int_{t_0}^{t_n} \left( \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} \eta(t) + \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon))} \dot{\eta}(t) \right) \mathrm{d}t \\ &= \int_{t_0}^{t_n} \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} \eta(t) \mathrm{d}t + \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon))} \eta(t) \Big|_{t_0}^{t_n} \\ &- \int_{t_0}^{t_n} \frac{\mathrm{d}}{\mathrm{d}t} \left( \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon))} \right) \eta(t) \mathrm{d}t. \end{aligned} \tag{2.14}$$

Due to $\eta(t_0) = \eta(t_n) = 0$, it follows from Eq. (2.14)

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon} I(\varepsilon) = \int_{t_0}^{t_n} \left( \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon))} \right) \eta(t) \mathrm{d}t = 0$$

for $\varepsilon = 0$. Thus, if we directly apply Lemma 1, setting $g(t) := \frac{\partial f(t,\rho(t,0),\dot{\rho}(t,0))}{\partial \rho(t,0)} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial f(t,\rho(t,0),\dot{\rho}(t,0))}{\partial \dot{\rho}(t,0)}$, if $\delta I = 0$, then

$$\left( \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \rho(t,\varepsilon)} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial f(t,\rho(t,\varepsilon),\dot{\rho}(t,\varepsilon))}{\partial \dot{\rho}(t,\varepsilon)} \right) \Big|_{\varepsilon=0} = 0, \tag{2.15}$$

which is known as the **Euler-Lagrange equation**. The herein shown formalism of variational calculus can now be applied to the dynamical system of particles to derive the equations of motion.

### 2.1.3 Energy Conservation and Euler-Lagrange Equation

After having described Newton's laws of motion for a system of particles with central mass and the concept of deriving the Euler-Lagrange equation from variational calculus, we now combine these approaches to get appropriate motion equations for conservative and non-conservative systems, again mainly based on [32, 54].

Let us therefore resume that for a particle with index $i \in \{1,2,...,N\}$ in equilibrium, Newton's second law states that

$$F_i(t) = \dot{p}_i(t) = m_i \ddot{r}_i(t),$$

which is equal to

$$(F_i(t) - m_i \ddot{r}_i(t)) = 0 \tag{2.16}$$

for all $i \in \{1,2,...,N\}$ and $t \in [t_0,t_n]$. The state vectors $r_i(t) \in \mathbb{R}^d$, with $i \in \{1,2,...,N\}$, $d \in \mathbb{N}$ and $t \in [t_0,t_n]$, have previously been defined as time-dependent, continuously differentiable functions. Following [54, 112], the state vectors are supposed to depend upon the time-variable, as well as on **generalized coordinates**

$$q(t) := (q_1(t), q_2(t), ..., q_d(t))^T \in \mathbb{R}^d, \tag{2.17}$$

where $q_j \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ with $d \in \mathbb{N}$.

One can then define the individual **variation of the generalized coordinates**, similar to the previously given principles in the last section, especially in Eq. (2.13), for $t \in [t_0,t_n]$ and $j \in \{1,2,...,N\}$, by

$$\delta q_j := \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\left(q_j(t) + \varepsilon\eta_j(t)\right)\big|_{\varepsilon=0},$$

where $\eta \in \mathscr{C}^2([t_0,t_n],\mathbb{R}^d)$ is a perturbation with variation rate $\varepsilon \geq 0$ in the sense of Lemma 1. Analogously, the **variation of the generalized velocities** is thus given by

$$\delta\dot{q}_j := \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\left(\dot{q}_j(t) + \varepsilon\dot{\eta}_j(t)\right)\big|_{\varepsilon=0} = \frac{\mathrm{d}}{\mathrm{d}t}\delta q_j.$$

For simplification, let us use $q_j := q_j(t)$ for all $t \in [t_0,t_n]$. Therefore, we use the short forms

$$r_i := r_i(t,q), \tag{2.18}$$

assuming that the state is differentiable for all $q_j$ with $j \in \{1,2,...,d\}$. Thus, it also holds that for the vector of velocities $\dot{r}_i$ and the vector of accelerations $\ddot{r}_i$ for $i \in \{1,2,...,N\}$, we make use of

$$\dot{r}_i := \frac{\mathrm{d}}{\mathrm{d}t}r_i(t,q), \qquad \ddot{r}_i := \frac{\mathrm{d}^2}{\mathrm{d}t^2}r_i(t,q). \tag{2.19}$$

The **variation of the state** [54] is then explicitly given by

$$\delta r_i := \sum_{j=1}^{d}\frac{\partial r_i}{\partial q_j}\delta q_j. \tag{2.20}$$

It holds true, that for variations $\delta r_i$ for $i \in \{1,2,...,N\}$ and Newton's second law, Eq. (2.16), we have

$$\sum_{i=1}^{N}\left(F_i(t) - m_i\ddot{r}_i\right)\delta r_i = 0$$

and we assume no forces of constraint [32] to occur in our context. The concept, mathematically described by Eq. (2.1.3) is also known as **D'Alembert's Principle**. As a next step, we consider the variation of **virtual work** by

$$\delta W := \sum_{i=1}^{N}F_i(t)\delta r_i, \tag{2.21}$$

such that from Eq. (2.1.3), it is obvious that the variation is also equal to

$$\delta W = \sum_{i=1}^{N} m_i \ddot{r}_i \delta r_i. \tag{2.22}$$

Thus, inserting Eq. (2.20) in Eq. (2.21) yields

$$\delta W = \sum_{i=1}^{N} F_i(t)\delta r_i = \sum_{i=1}^{N} F_i(t) \sum_{j=1}^{d} \frac{\partial r_i}{\partial q_j}\delta q_j = \sum_{j=1}^{d}\sum_{i=1}^{N} F_i(t)\frac{\partial r_i}{\partial q_j}\delta q_j = \sum_{j=1}^{d} Q_j \delta q_j, \tag{2.23}$$

where

$$Q_j := \sum_{i=1}^{N} F_i(t)\frac{\partial r_i}{\partial q_j} \tag{2.24}$$

is the **generalized force** for $j \in \{1,2,...,d\}$. Since the product rule for differentiation yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(m_i \dot{r}_i \frac{\partial r_i}{\partial q_j}\right) = m_i \ddot{r}_i \frac{\partial r_i}{\partial q_j} + m_i \dot{r}_i \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial r_i}{\partial q_j},$$

it follows that Eq. (2.22) equals

$$\begin{aligned}\delta W &= \sum_{i=1}^{N} m_i \ddot{r}_i \delta r_i \\ &= \sum_{i=1}^{N} m_i \ddot{r}_i \sum_{j=1}^{d} \frac{\partial r_i}{\partial q_j}\delta q_j \\ &= \sum_{j=1}^{d}\sum_{i=1}^{N}\left(\frac{\mathrm{d}}{\mathrm{d}t}\left(m_i \dot{r}_i \frac{\partial r_i}{\partial q_j}\right) - m_i \dot{r}_i \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial r_i}{\partial q_j}\right)\delta q_j. \end{aligned} \tag{2.25}$$

Following the dependencies as stated in Eq. (2.18), the total derivative of the state vector is given by

$$\dot{r}_i = \frac{\mathrm{d}}{\mathrm{d}t} r_i(t,q) = \sum_{j=1}^{d} \frac{\partial r_i}{\partial q_j}\dot{q}_j + \frac{\partial r_i}{\partial t}.$$

Thus $\frac{\partial \dot{r}_i}{\partial \dot{q}_j} = \frac{\partial r_i}{\partial q_j}$ for $i \in \{1,2,...,N\}$ and $j \in \{1,2,...,d\}$, such that Eq. (2.25) simplifies to

$$\delta W = \sum_{j=1}^{d}\sum_{i=1}^{N}\left(\frac{\mathrm{d}}{\mathrm{d}t}\left(m_i \dot{r}_i \frac{\partial \dot{r}_i}{\partial \dot{q}_j}\right) - m_i \dot{r}_i \frac{\partial \dot{r}_i}{\partial q_j}\right)\delta q_j. \tag{2.26}$$

As a next step, we consider the **kinetic energy** of a system to be generally defined as a function of time, state and velocity by

$$T : \mathscr{T} \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \tag{2.27}$$

$$(t,q,\dot{q}) \mapsto T(t,q,\dot{q}), \tag{2.28}$$

with a finite time horizon $\mathscr{T} = [t_0, t_n] \subset \mathbb{R}$. Making use of Eq. (2.19), it can be shown that

$$T(t,q,\dot{q}) = \sum_{i=1}^{N} \frac{1}{2} m_i \frac{\mathrm{d}}{\mathrm{d}t} r_i(t,q)^2 = \sum_{i=1}^{N} \frac{1}{2} m_i \dot{r}_i^2 \tag{2.29}$$

such that the **variation of the kinetic energy** is given by

$$\delta T := \sum_{i=1}^{N} m_i \dot{r}_i \delta \dot{r}_i. \tag{2.30}$$

Regarding the derivatives of Eq. (2.26), it then holds that

$$\sum_{i=1}^{N} \frac{\mathrm{d}}{\mathrm{d}t} m_i \dot{r}_i \frac{\partial \dot{r}_i}{\partial \dot{q}_j} = \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial}{\partial \dot{q}_j} \sum_{i=1}^{N} \frac{1}{2} m_i \dot{r}_i^2 = \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} \tag{2.31}$$

and analogously

$$\sum_{i=1}^{N} m_i \dot{r}_i \frac{\partial \dot{r}_i}{\partial q_j} = \frac{\partial}{\partial q_j} \sum_{i=1}^{N} \frac{1}{2} m_i \dot{r}_i^2 = \frac{\partial T(t,q,\dot{q})}{\partial q_j}, \tag{2.32}$$

which yields that the variation of virtual work can be expressed in terms of derivatives of the kinetic energy. In order to apply Lemma 1, we use the following integral description, including the variation of the kinetic energy such that we derive the following expression with the help of integration by parts and the boundary conditions $\delta r_i(t_0) = \delta r_i(t_n) = 0$ of the variations $\delta r_i$:

$$\begin{aligned}
\int_{t_0}^{t_n} \delta T \,\mathrm{d}t &= \int_{t_0}^{t_n} \sum_{i=1}^{N} m_i \dot{r}_i \delta \dot{r}_i \mathrm{d}t \\
&= \sum_{i=1}^{N} \left( m_i \dot{r}_i \delta r_i \big|_{t_0}^{t_n} - \int_{t_0}^{t_n} m_i \ddot{r}_i \delta r_i \mathrm{d}t \right) \\
&= - \int_{t_0}^{t_n} \sum_{i=1}^{N} m_i \ddot{r}_i \delta r_i \mathrm{d}t \\
&= - \int_{t_0}^{t_n} \delta W \mathrm{d}t.
\end{aligned}$$

We can therefore derive the following expression in the sense of variational calculus by

$$\int_{t_0}^{t_n} (\delta T + \delta W) \,\mathrm{d}t = 0, \tag{2.33}$$

which is known as **energy conservation** for a physical system. A dynamical system, satisfying Eq. (2.33) is therefore called **conservative system**.

The equations of motion, summarizing the results as stated by the generalized force in Eq. (2.24), the simplified expression in Eq. (2.26) and the dependencies of the variation of virtual work in terms of differentials of the kinetic energy in Eq. (2.31) and Eq. (2.32) are given through

$$
\begin{aligned}
\delta I := \int_{t_0}^{t_n} (\delta T + \delta W)\,\mathrm{d}t &= \sum_{j=1}^{d} \int_{t_0}^{t_n} \left( \left( \frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} \right) \delta q_j + Q_j \delta q_j \right) \mathrm{d}t \\
&= \sum_{j=1}^{d} \int_{t_0}^{t_n} \left( \frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} + Q_j \right) \delta q_j \mathrm{d}t \qquad (2.34) \\
&\overset{!}{=} 0. \qquad (2.35)
\end{aligned}
$$

Since the generalized coordinates are independent [32, 54] on each other and the conditions of Lemma 1 hold, it follows that

$$\frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} + Q_j = 0 \tag{2.36}$$

for all $j = \{1,2,...,d\}$. The generalized forces $Q_j$ with $j \in \{1,2,...,d\}$ in Eq. (2.36) need now to be specified for a conservative system. Following the assumption, that a **potential energy function**

$$
\begin{aligned}
V : \mathscr{T} \times \mathbb{R}^d \times \mathbb{R}^d &\to \mathbb{R} \qquad (2.37) \\
(t,q,\dot{q}) &\mapsto V(t,q,\dot{q}) \qquad (2.38)
\end{aligned}
$$

for a finite time horizon $\mathscr{T} = [t_0, t_n] \subset \mathbb{R}$ is differentiable w.r.t. the generalized coordinates of the vector $q \in \mathbb{R}^d$, we define the generalized forces as

$$Q_j := -\frac{\partial V(t,q,\dot{q})}{\partial q_j}$$

for all $j \in \{1,2,...,d\}$ such that $\frac{\partial V(t,q,\dot{q})}{\partial \dot{q}_j} = 0$. Thus the potential energy for a conservative system does not depend upon the generalized velocities $\dot{q}_j$. Defining the **Lagrange function** as the difference between kinetic and potential energy of a conservative system, thus

$$
\begin{aligned}
L : \mathscr{T} \times \mathbb{R}^d \times \mathbb{R}^d &\to \mathbb{R} \qquad (2.39) \\
(t,q,\dot{q}) &\mapsto T(t,q,\dot{q}) - V(t,q,\dot{q}), \qquad (2.40)
\end{aligned}
$$

it can then be shown that Eq. (2.36) equals

$$
\begin{aligned}
& \frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} + Q_j = 0 \\
\Leftrightarrow \quad & \frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\partial V(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} = 0 \qquad (2.41) \\
\Leftrightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} = 0. \qquad (2.42)
\end{aligned}
$$

The above stated Eq. (2.41) and Eq. (2.42) are the **Lagrangian equations of motion for a conservative dynamical system**. Since we focus on coupled dynamical systems in the following chapters including an external driving force, we specify the generalized force in a different way with the help of the Rayleigh-Dissipation function in the next section, such that we can derive a similar expression as Eq. (2.42) for **non-conservative systems**.

### 2.1.4 Rayleigh Dissipation and Potential Energy

We have already derived the general equation of motion for a conservative particle system in Eq. (2.41), resulting from the Euler-Lagrange formalism of variational calculus. Since we want to investigate the equations of motions of an approximate vehicle model, we restrict to coupled Mass-Spring-Damper (MSD) systems. The main target of this section is therefore to derive a similar expression as Eq. (2.41) for a coupled dynamical MSD system.

To distinguish between conservative systems in the sense of closed particle systems and non-conservative systems, which we assume to be dynamical systems, driven by external forces, we need to again specify the generalized forces $Q_j$ for $j \in \{1,2,...,d\}$. We will later on see, that the analogy to $d \in \mathbb{N}$ being the number of generalized coordinates of a system with free particles, is that $d \in \mathbb{N}$ equals the degrees of freedom (DOF) of a dynamical system with coupled components. The main approaches to specify such generalized forces are again based on [32, 54].

We assume that for a non-conservative system, the generalized forces for each particle can be derived from the generalized coordinates $q_j$ of a potential energy function $V$, as well as from the generalized velocities $\dot{q}_j$ of a corresponding additional potential function $U$, known as the **Rayleigh-Dissipation-Function** [32], defined as

$$\begin{aligned} U : \mathscr{T} \times \mathbb{R}^d &\to \mathbb{R} && (2.43)\\ (t,\dot{q}) &\mapsto U(t,\dot{q}). && (2.44) \end{aligned}$$

In summary, this then yields that for each particle, indexed by $j \in \{1,2,...,d\}$, the **generalized force for a non-conservative system** can be expressed in terms of

$$Q_j^{nc} := -\frac{\partial V(t,q,\dot{q})}{\partial q_j} - \frac{\partial U(t,\dot{q})}{\partial \dot{q}_j}. \tag{2.45}$$

It is obvious, that inserting Eq. (2.45) for the generalized force into Eq. (2.36) directly gives an expression for non-conservative systems, which is essential to derive systems of ordinary differential equations for approximative vehicle models in the next section. Therefore it holds for all $j \in \{1,2,...,d\}$ that

$$\begin{aligned} & \frac{\partial T(t,q,\dot{q})}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} + Q_j^{nc} = 0 \\ \Leftrightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial T(t,q,\dot{q})}{\partial q_j} - Q_j^{nc} = 0 \\ \Leftrightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T(t,q,\dot{q})}{\partial \dot{q}_j} + \frac{\partial V(t,q,\dot{q})}{\partial q_j} + \frac{\partial U(t,\dot{q})}{\partial \dot{q}_j} = 0 \qquad (2.46) \end{aligned}$$

$$\Leftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} + \frac{\partial U(t,\dot{q})}{\partial \dot{q}_j} = 0. \tag{2.47}$$

Since it always holds in our case that $\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V(t,q,\dot{q})}{\partial \dot{q}_j} = 0$, it is most sufficient and preferable for sake of simplicity, to use Eq. (2.46) instead of the more general Eq. (2.47).

We have clarified all required terms of the general equations of motion for non-conservative systems, stated by Eq. (2.47). Besides, the derivation of the Euler-Lagrange equation for non-conservative systems, including Rayleigh-Dissipation, can be done via definition of the generalized force $Q_j$ for all coordinates $q_j$ with index $j \in \{1,2,...,d\}$. We can therefore move from the theoretical concept of general particle systems to more specific coupled dynamical systems in the next section.

## 2.2 Coupled Dynamical Systems

Given the theoretical concepts derived in Section 2.1, the Euler-Lagrange equations for non-conservative systems can be used to derive mathematical expressions for coupled Mass-Spring-Damper (MSD) systems, serving as approximative vehicle models. We therefore derive now systems of second order differential equations, by definition of the energy terms and applying Eq. (2.46) to get a variant of the prominent Quarter-Car-Model (QCM), the Half-Car-Model (HCM) and the Full-Car-Model (FCM). In general, the mentioned models, distinguish in the number of the modelled wheel-suspensions, which results in a variation of the degrees of freedom (DOFs) of the model. It is possible to derive detailed and realistic vehicle models [107], when there is a very large number of DOFs for the individual components of the model. Nevertheless, it is quite difficult to analyse the dependencies between complex models and data simulated by these models, since a larger number of DOFs corresponds to a significantly larger number of system parameters.

In case of a general three-dimensional Cartesian system, the motion of a rigid body can be described through six DOFs, three for motion along the axes and three for rotation around the axes of a Cartesian coordinate system. In engineering, the three motions along the axes, the rotational degrees of freedom, are also called **"rolling"**, **"pitching"** and **"yawing"**, where the translational degrees of freedom are called **"heaving"**, **"swaying"** and **"surging"**.

For a coupled MSD system, the motion of each component can be described by a maximum of six DOFs, nevertheless, the complexity of the system depends upon the total number of DOFs of the whole vehicle model. It is then also possible to connect a dynamical model of a vehicle to another passenger model, which can again be described as a complex MSD system [1]. Since we are interested in deriving generalized vehicle models, we start with a three components Quarter-Car-Model [59], where each component has only one DOF in vertical direction ("heaving"). Similar approaches for a simplified passenger-based vehicle model can be extended to the Half-Car-Model [60] as well as to the Full-Car-Model [37]. For the latter two models, it is necessary to also include rotational DOFs, such as "rolling" and "pitching".

### 2.2.1 Rotational and Translational Energy

Since we want to investigate approximative vehicle models, where the complexity of the models depends on the number of translational and rotational DOFs per rigid mass component, we introduce a simple 1DOF Pendulum model and a simple 1DOF MSD model. Combining these concepts and the Lagrange-Equation methodology of Section 2.1, leads to a sophisticated method to derive approximate vehicle models from physical equations.

<u>**Mathematical Pendulum**</u>

We herein restrict to a simple mathematical pendulum, as it is shown in Figure 2.1 or described in [18]. Theoretical investigations have also been done to derive systems of coupled pendulum models, as it is stated in [7] containing "Mathematical and Physical Pendulum" or about coupled pendulum models with variable mass in [61]. As far as the rotational motion of approximate vehicle models is concerned, it is sufficient to use a simple pendulum for clarification as described below.

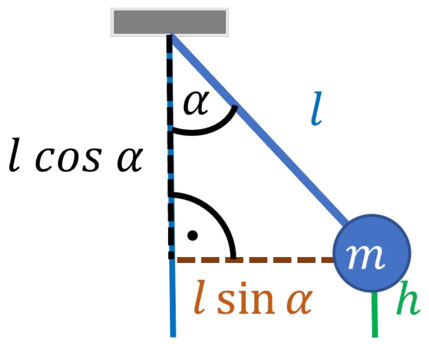


FIGURE 2.1: Simple Pendulum Model with mass $m > 0$, pendulum length $l > 0$ and angle $\alpha \in [-\pi, \pi]$.

The mathematical pendulum, as it is shown in Figure 2.1, depends upon one generalized coordinate. Since we always consider time-dependent variables in the context of dynamical systems, the angular displacement $\alpha(t) \in [-\pi, \pi]$ for $t \in [t_0, t_n]$ is in fact a twice continuously differential function $\alpha \in \mathscr{C}^2([t_0, t_1], [-\pi, \pi])$. For sake of simplicity, we use $\alpha := \alpha(t)$. We have previously shown the Euler-Lagrange formalism for dynamical systems, therefore we can now directly derive the equation of motion with respect to the rotational angular displacement, via defining the kinetic energy $T$ and the potential energy $V$ for the pendulum model. The herein considered model is conservative and has therefore no external forces acting on it. Similar to the kinetic energy being defined for point masses of linear motion through the product of mass and relative velocity, the kinetic energy for rotational motions depends upon the mass moment of inertia and the angular velocity of the model.

For the simple model, with pendulum length $l > 0$ and mass $m > 0$ the **mass moment of inertia** for the rotation angle $\alpha$ is given by

$$I_\alpha := ml^2. \tag{2.48}$$

The kinetic energy $T(t, \alpha, \dot{\alpha})$ for the simple model is then

$$T(t, \alpha, \dot{\alpha}) = \frac{1}{2} I_\alpha \dot{\alpha}^2$$

and the potential energy, due to the influence of gravitational force with constant $g \approx 9.81 \frac{m}{s^2}$ is defined as

$$V(t, \alpha, \dot{\alpha}) = mgh$$

for the relative height $h \geq 0$ to the equilibrium point, when $\alpha = 0$. Since the generalized coordinates of a system need to be derivable from the potential energy function $V$, there is the need of a dependence of the potential energy and the rotation angle $\alpha$, which cannot be verified from $V(t, \alpha, \dot{\alpha}) = mgh$. Due to $l^2 = l^2(\cos^2\alpha + \sin^2\alpha) = (l \sin\alpha)^2 + (l \cos\alpha)^2$ it follows, and can also be verified by having a look at Figure 2.1, that with $l = l(\cos\alpha + (1 - \cos\alpha))$, we get $h = l(1 - \cos\alpha)$.

The potential energy is therefore given by

$$V(t, \alpha, \dot{\alpha}) = mgl(1 - \cos\alpha)$$

and since there is no exterior force, the Lagrange function yields

$$L(t, \alpha, \dot{\alpha}) = T(t, \alpha, \dot{\alpha}) - V(t, \alpha, \dot{\alpha}) = \frac{1}{2}\left(I_\alpha \dot{\alpha}^2\right) - mgl(1 - \cos\alpha) = \frac{1}{2}\left(I_\alpha \dot{\alpha}^2\right) + mgl\cos\alpha - mgl. \tag{2.49}$$

The Euler-Lagrange equation for conservative systems with the generalized coordinate $q_1 = \alpha$, then directly gives

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(t, \alpha, \dot{\alpha})}{\partial \dot{\alpha}} - \frac{\partial L(t, \alpha, \dot{\alpha})}{\partial \alpha} = 0 \\ \Leftrightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}t} I_\alpha \dot{\alpha} + mgl\sin\alpha = 0 \\ \Leftrightarrow \quad & I_\alpha \ddot{\alpha} + mgl\sin\alpha = 0. \end{aligned} \tag{2.50}$$

The equation of motion for the simple pendulum model is given by Eq. (2.50) and is often simplified to

$$I_\alpha \ddot{\alpha} + mgl\alpha = 0 \tag{2.51}$$

which uses the power series expansion of the cosine for small oscillations by

$$\cos\alpha = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!} \approx 1 - \frac{\alpha^2}{2}.$$

The potential energy $V(t,\alpha,\dot{\alpha})$ then simplifies to $V(t,\alpha,\dot{\alpha}) = \frac{1}{2}mgl\alpha^2$ and then leads to Eq. (2.51).

The pendulum model is not discussed in detail for the further progress of the section, nevertheless, when defining the kinetic energy term for the approximative vehicle models, the terms of mass moment of inertia and the rotational displacements need to be clarified beforehand.

**Mass-Spring-Damper**

Mass-Spring-Damper (MSD) systems are broadly used to model the motion profile for several experiments, for instance a model of a bouncing ball [75]. Nevertheless, we mainly focus on MSD systems to model several components of approximate vehicle models in the course of the section.

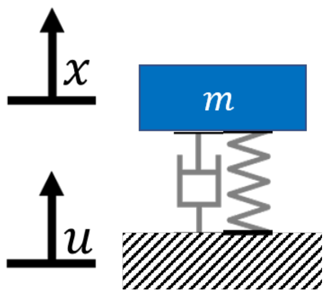


FIGURE 2.2: Simple driven Mass-Spring-Damper system with external source displacement $u$. A displacement of the external source causes a deformation of the spring and damper, coupled to a rigid body with mass $m > 0$. The relative displacement of the rigid body is denoted by $x$.

The model can be visualized by having a look at Figure 2.2. Similar to the pendulum model, we have one generalized coordinate, described by the time-dependent state variable $x$. In contrast to the conservative pendulum, an additional control variable is shown, denoted by $u$. The variable $u$ leads to a deformation of the spring and the damper, that are connected to the rigid body with mass $m > 0$. The vertical displacement of this rigid body can then be recognized by the trajectory of the state variable $x$.

We now focus on a non-conservative dynamical system, where the mentioned variables are continuously differentiable functions with $x \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ and $u \in \mathscr{C}^1([t_0,t_n],\mathbb{R})$. Since the system is driven by the control variable $u$, we assume that the system is in equilibrium at the initial point of the defined time frame, meaning we have $u_0 = u(t_0) = 0$ as well as $x_0 = x(t_0) = 0$.

We have already mentioned that the system is non-conservative. Therefore, to derive the equation of motion with the help of the Euler-Lagrange formalism, we need to define, as usual, the kinetic and potential energy and the Rayleigh-Dissipation function. At a given time $t \in [t_0, t_n]$, a displacement of a driving term, denoted by $u(t)$, results into a deformation of the spring and damper, coupled to a body with mass $m$. Is is herein assumed that corresponding spring forces are linear in relative displacement and damping forces behave linear to relative velocity.

As usual, the kinetic energy $T(t,x,\dot{x})$ of an object with mass $m$ and relative state $x := x(t)$ for $t \in [t_0, t_n]$ can be expressed as

$$T(t,x,\dot{x}) = \frac{1}{2} m \dot{x}^2.$$

As the potential energy is given by the effective spring force with constant $K > 0$, which behaves proportional to the current relative spring length $l > 0$, here obviously $l = |x - u|$, with $u := u(t)$ for $t \in [t_0, t_n]$ we get

$$V(t,x,\dot{x}) = \frac{1}{2} K (x-u)^2$$

and in analogy, the Rayleigh-Dissipation is then given for a damper with constant $C > 0$ proportional to the relative effective velocity $|\dot{x} - \dot{u}|$ by

$$U(t,\dot{x}) = \frac{1}{2} C (\dot{x} - \dot{u})^2.$$

Since we only have one generalized coordinate, namely $q_1 = x$, it is obvious with

$$Q_1^{nc} = -\frac{\partial U(t,\dot{x})}{\partial \dot{x}} - \frac{\partial V(t,x,\dot{x})}{\partial x}$$

that the Euler-Lagrange equation then yields

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t,x,\dot{x})}{\partial \dot{x}} + \frac{\partial V(t,x,\dot{x})}{\partial x} + \frac{\partial U(t,\dot{x})}{\partial \dot{x}} = 0. \tag{2.52}$$

Explicit derivation of Eq. (2.52) then directly gives the equations of motion for the MSD system by

$$m\ddot{x} + C(\dot{x} - \dot{u}) + K(x-u) = 0.$$

We have finally derived all necessary components with the simple pendulum model and the herein described MSD system to now define more complex Euler-Lagrange equations to derive mathematical systems of approximate vehicle models.

### 2.2.2 Quarter-Car-Model

The Quarter-Car-Model (QCM) is one of the most prominent concepts, when dealing with approximations of vehicles in research. General models can for instance be investigated in [27, 108] without considering a coupled passenger component. Passenger models can for instance be modelled with the use of complex, hierarchically coupled MSD systems as it is shown in [74], but we restrict for sake of simplicity to a passenger model, consisting of a single rigid mass [59]. As far as understanding the principle to come from Quarter-Car-Model to Half-Car-Model and Full-Car-Model, there is no benefit at this point to consider complex passenger models.

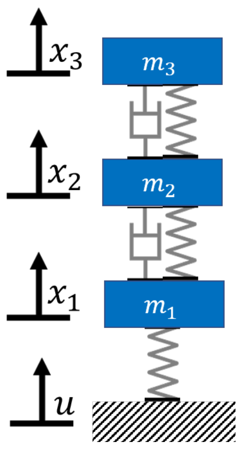


FIGURE 2.3: Passenger-based Quarter-Car-Model. Similar to the Mass-Spring-Damper model, a source term is denoted by $u$, which causes a deformation of a spring, coupled to the wheel-suspensions with mass $m_1 > 0$ and relative displacement $x_1$. The wheel-suspensions are coupled to the chassis, via an additional spring and damper, with mass $m_2 > 0$ and state $x_2$. Finally, the chassis is again coupled with another spring-damper-pair to the seat with state $x_3$, where $m_3 > 0$ describes the mass of the seat and an occupant located on it.

Based upon the visualized structure of the Quarter-Car-Model in Figure 2.3, similar to the general MSD system of the previous section, a driving term $u(t)$, for instance a displacement of the road-surface at time $t \in [t_0, t_n]$ affects a system. In this case, the coupled system, which contains three rigid bodies with non-zero masses $m_1$, $m_2$ and $m_3$ is systematically for the dynamical behaviour of a quarter part of a vehicle model. Here, $m_1$ describes the mass of a wheel-suspension, $m_2$ is the mass of the car body (chassis) and $m_3$ is the combined mass of a seat and a passenger located on the seat. We assume, that the displacement of the wheels, affected by the road displacement can be described via a spring connection between road-profile and wheel-suspension with spring constant $K_1 > 0$. Similar to a real car, there are of course several springs and dampers, connecting the suspensions to the car body. Therefore the connections of wheel-suspensions and chassis with masses $m_1$ and $m_2$ is done with the help of a spring and a damper with non-zero constants $K_2$ and $C_2$. Since there are also springs and dampers, connecting the seat to the chassis, another pair of springs and dampers with non-zero constants $K_3$ and $C_3$ can be used to model this connection. It is then obvious, due to this hierarchically coupled structure, that a displacement of the road $u(t)$ at $t \in [t_0, t_n]$, deforms the spring and results in a displacement of the wheel-suspension state $x_1(t)$. This then causes a deformation of the connected springs and dampers and results in a displacement of the car-body, denoted by $x_2(t)$, and finally a displacement of the seat, denoted by $x_3(t)$.

Since in most physical systems, we assume to be able to at least measure acceleration at each time $t \in [t_0, t_n]$, the displacements of the above defined components should satisfy to be defined as twice continuously differentiable functions $x_i \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ for all $i = 1, 2, 3$. Since the connection of road-surface to suspension is modelled without damper, there is no need for the control function to be differentiable. It is therefore sufficient to have a continuous $u \in \mathscr{C}^0([t_0, t_n], \mathbb{R})$. It further holds for the initial values of the states, similar to the simple MSD system, that $x_i(t_0) = 0$, $\dot{x}_i(t_0) = 0$ as well as $\ddot{x}_i(t_0) = 0$ for $i = 1, 2, 3$, meaning that the system is in equilibrium at $t = t_0$. The relative displacement for the individual states is therefore given with respect to the equilibrium for the entire time frame.

We can now already define the kinetic and the potential energy, as well as the Rayleigh-Dissipation, since we again have a non-conservative dynamical system. The kinetic energy is given by

$$T(t, x, \dot{x}) = \frac{1}{2} \sum_{i=1}^{3} m_i \dot{x}_i^2,$$

the potential energy, resulting from the relative spring length is defined as

$$V(t, x, \dot{x}) = \frac{1}{2} \left( K_1 (x_1 - u)^2 + K_2 (x_2 - x_1)^2 + K_3 (x_3 - x_2)^2 \right)$$

and finally, the Rayleigh-Dissipation function is expressed in terms of damping-constants and relative velocities as

$$U(t, \dot{x}) = \frac{1}{2} \left( C_2 (\dot{x}_2 - \dot{x}_1)^2 + C_3 (\dot{x}_3 - \dot{x}_2)^2 \right),$$

since the model only considers two dampers in the coupled structure. We recognize that $T(t, x, \dot{x})$ depends upon the derivative of $d = 3$ generalized coordinates, $x_i$ for $i = 1, 2, 3$, therefore we consider three Euler-Lagrange equations

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial T(t, x, \dot{x})}{\partial \dot{x}_i} + \frac{\partial U(t, \dot{x})}{\partial \dot{x}_i} + \frac{\partial V(t, x, \dot{x})}{\partial x_i} = 0 \tag{2.53}$$

for $i = 1, 2, 3$. Since we try to formalize the visualization of the QCM, we derive the second order ODEs in hierarchical order, starting from the top component (occupant and seat). In addition, we can assume $m_i > 0$ for all components of the vehicle, because there are no wheels, car bodies or seats without a mass. Then, finally Eq. (2.53) gives

$$\begin{aligned}
& m_3\ddot{x}_3 + C_3(\dot{x}_3 - \dot{x}_2) + K_3(x_3 - x_2) = 0 \\
\Leftrightarrow\quad & m_3\ddot{x}_3 = -C_3\dot{x}_3 + C_3\dot{x}_2 - K_3x_3 + K_3x_2 \\
\Leftrightarrow\quad & \ddot{x}_3 = -\frac{C_3}{m_3}\dot{x}_3 + \frac{C_3}{m_3}\dot{x}_2 - \frac{K_3}{m_3}x_3 + \frac{K_3}{m_3}x_2, \qquad (2.54)
\end{aligned}$$

$$\begin{aligned}
& m_2\ddot{x}_2 + C_3(\dot{x}_3 - \dot{x}_2)\cdot(-1) + C_2(\dot{x}_2 - \dot{x}_1) + K_3(x_3 - x_2)\cdot(-1) + K_2(x_2 - x_1) = 0 \\
\Leftrightarrow\quad & m_2\ddot{x}_2 = C_3\dot{x}_3 - (C_2 + C_3)\dot{x}_2 + C_2\dot{x}_1 + K_3x_3 - (K_2 + K_3)x_2 + K_2x_1 \\
\Leftrightarrow\quad & \ddot{x}_2 = \frac{C_3}{m_2}\dot{x}_3 - \frac{C_2 + C_3}{m_2}\dot{x}_2 + \frac{C_2}{m_2}\dot{x}_1 + \frac{K_3}{m_2}x_3 - \frac{K_2 + K_3}{m_2}x_2 + \frac{K_2}{m_2}x_1, \qquad (2.55)
\end{aligned}$$

$$\begin{aligned}
& m_1\ddot{x}_1 + C_2(\dot{x}_2 - \dot{x}_1)\cdot(-1) + K_2(x_2 - x_1)\cdot(-1) + K_1(x_1 - u) = 0 \\
\Leftrightarrow\quad & m_1\ddot{x}_1 = C_2\dot{x}_2 - C_2\dot{x}_1 + K_2x_2 - (K_2 + K_1)x_1 + K_1u \\
\Leftrightarrow\quad & \ddot{x}_1 = \frac{C_2}{m_1}\dot{x}_2 - \frac{C_2}{m_1}\dot{x}_1 + \frac{K_2}{m_1}x_2 - \frac{K_2 + K_1}{m_1}x_1 + \frac{K_1}{m_1}u. \qquad (2.56)
\end{aligned}$$

Summarizing, Eq. (2.54) - (2.56) show, that the acceleration of the three components in a QCM can be expressed in terms of displacement and velocity, combined with the quotient of the corresponding spring - and damping - coefficients and the masses.

It should be clear, that the above system of second order ODE can formally also be described as a high-dimensional system of first order ODE, expressed by

$$\begin{pmatrix} \ddot{x}_3 \\ \ddot{x}_2 \\ \ddot{x}_1 \\ \dot{x}_3 \\ \dot{x}_2 \\ \dot{x}_1 \end{pmatrix} = \begin{bmatrix} -\frac{C_3}{m_3} & \frac{C_3}{m_3} & 0 & -\frac{K_3}{m_3} & \frac{K_3}{m_3} & 0 \\ \frac{C_3}{m_2} & -\frac{C_2+C_3}{m_2} & \frac{C_2}{m_2} & \frac{K_3}{m_2} & -\frac{K_2+K_3}{m_2} & \frac{K_2}{m_2} \\ 0 & \frac{C_2}{m_1} & -\frac{C_2}{m_1} & 0 & \frac{K_2}{m_1} & -\frac{K_2+K_1}{m_1} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} \dot{x}_3 \\ \dot{x}_2 \\ \dot{x}_1 \\ x_3 \\ x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{K_1}{m_1}u \\ 0 \\ 0 \\ 0 \end{pmatrix}. \qquad (2.57)$$

The system described by Eq. (2.57) is a complete formal description for the motions of equation in the QCM and could also be denoted by

$$\dot{\omega} = A\omega + b, \qquad (2.58)$$

where the parameter matrix $A$ is then also called the **system matrix** of the QCM. A more detailed mathematical description of such linear first order systems is given in Chapter 4. The same concepts are now applied to derive similar systems of first order ODE for more complex vehicle models.

### 2.2.3 Half-Car-Model

We want to give a short overview, how a more detailed vehicle model, in comparison to the simple QCM, can be derived. Therefore, we herein present the Half-Car-Model (HCM), including two components of wheel-suspensions, leading to additional translational DOFs for the car-body [31]. Similar to the QCM, we present a passenger-based HCM, comparable to [1], but again restricting the seat and passenger mass to be described by a single rigid body. The QCM can be interpreted as a 3DOF vehicle model, since we restricted it to three translational generalized coordinates. The now presented HCM is a 5DOF vehicle model, consisting of four translational generalized coordinates and one additional rotational coordinate. We further assume that the seat is located at the back of the car body. Of course, it is also possible to derive a similar model with the seat located at the front or at the center of the chassis.

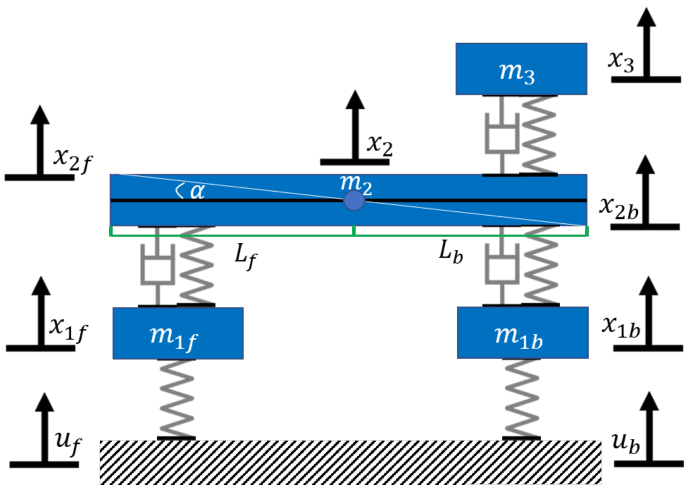


FIGURE 2.4: Passenger-based Half-Car-Model. Similar structure to the Quarter-Car-Model. Displacements of the source terms are denoted with $u_f$ for the front of the car and $u_b$ for the back. Displacements of the wheel-suspensions are denoted by $x_{1f}$ and $x_{2f}$ with masses $m_{1f}$, $m_{2f} > 0$, displacements caused by rotational and translational states for the chassis by $x_{2f}$ and $x_{2b}$ for front and back, and $x_2$ for the center of the chassis with mass $m_2 > 0$. The displacement of the seat is denoted by $x_3$, where $m_3 > 0$ is the mass of the seat and a possible occupant on it.

As it can be seen in Figure 2.4, we now discuss a double wheel-suspension model, containing two different control functions, $u_f$ and $u_b$, acting in parallel on the vehicle model. Since the vehicle is supposed to drive along one specific road, it could be sufficient to have $u_b(t) = u_f(t + \Delta t)$, where $\Delta t$ describes the time, the vehicle needs to pass a distance of $L_f + L_b$. Therefore, $L_f$ describes the distance from front-wheel to the mass center of the car-body and analogously $L_b$ describes the distance from back-wheel to the mass center of the chassis. Thus, we have in principle two parallel QCMs, coupled by the car-body. The difference of both displacements then results in a rotational displacement $\alpha$ of the car-body. We have already seen with the help of the simple pendulum, how such an angular displacement can be modelled. In this case, both, $L_b$ and $L_f$, can be interpreted as the pendulum length, acting in different direction having the same rotation angle. Since we only assume small rotation angles for the car-body, we use $L_f \cos\alpha \approx L_f$ and $L_b \cos\alpha \approx L_b$, which describes the rotational behaviour of the partial car-bodies. It then follows for the vertical displacement of the car-bodies, that it can be described by $L_f \sin\alpha$ and $L_b \sin\alpha$. Since we have assumed to have only small rotation angles, we use another approximation $\sin\alpha \approx \alpha$.

The displacements at the front and the back, resulting from translational and rotational motion can be described by

$$x_{2f} = x_2 + L_f\alpha \tag{2.59}$$

$$x_{2b} = x_2 - L_b\alpha, \tag{2.60}$$

where $x_2$ describes the reference state at the mass center of the car-body. The displacement $x_3$ is then directly dependent on the displacement of the back part of the car-body, denoted by $x_{2b}$. Holding in mind that for angular displacement, the mass moment of inertia $I_\alpha$ needs to be included in the computation of the kinetic energy, we can already define the required energy terms to apply the Euler-Lagrange formalism, using $x \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}^5\right)$. We get

$$\begin{aligned}
T(t,x,\dot{x}) =& \frac{1}{2}(m_{1f}\dot{x}_{1f}^2 + m_{1b}\dot{x}_{1b}^2 + m_2\dot{x}_2^2 + m_3\dot{x}_3^2 + I_\alpha\dot{\alpha}^2),\\
V(t,x,\dot{x}) =& \frac{1}{2}(K_{1f}(x_{1f}-u_{1f})^2 + K_{1b}(x_{1b}-u_{1b})^2 + K_{2f}(x_{2f}-x_{1f})^2 + K_{2b}(x_{2b}-x_{1b})^2\\
&+ K_3(x_3-x_{2b})^2)\\
=& \frac{1}{2}(K_{1f}(x_{1f}-u_{1f})^2 + K_{1b}(x_{1b}-u_{1b})^2 + K_{2f}(x_2+L_f\alpha-x_{1f})^2\\
&+ K_{2b}(x_2-L_b\alpha-x_{1b})^2 + K_3(x_3-x_2+L_b\alpha)^2),\\
U(t,\dot{x}) =& \frac{1}{2}(C_{2f}(\dot{x}_{2f}-\dot{x}_{1f})^2 + C_{2b}(\dot{x}_{2b}-\dot{x}_{1b})^2 + C_3(\dot{x}_3-\dot{x}_{2b})^2)\\
=& \frac{1}{2}(C_{2f}(\dot{x}_2+L_f\dot{\alpha}-\dot{x}_{1f})^2 + C_{2b}(\dot{x}_2-L_b\dot{\alpha}-\dot{x}_{1b})^2 + C_3(\dot{x}_3-\dot{x}_2+L_b\dot{\alpha})^2),
\end{aligned}$$

where $x(t) := (x_3, x_2, x_{1f}, x_{1b}, \alpha)^T$. Evaluating the individual Euler-Lagrange equations with respect to the generalized states as denoted by $x$, we get the system of ordinary differential equations by

$$m_3\ddot{x}_3 = -C_3\dot{x}_3 + C_3\dot{x}_2 - L_bC_3\dot{\alpha} - K_3x_3 + K_3x_2 - L_bK_3\alpha,$$

$$\begin{aligned}
m_2\ddot{x}_2 =& C_3\dot{x}_3 - (C_3+C_{2f}+C_{2b})\dot{x}_2 + C_{2f}\dot{x}_{1f} + C_{2b}\dot{x}_{1b} + (L_bC_3 + L_bC_{2b} - L_fC_{2f})\dot{\alpha}\\
&+ K_3x_3 - (K_3+K_{2b}+K_{2f})x_2 + K_{2b}x_{1b} + K_{2f}x_{1f} + (L_bK_3 + L_bK_{2b} - L_fK_{2f})\alpha,
\end{aligned}$$

$$m_{1f}\ddot{x}_{1f} = C_{2f}\dot{x}_2 - C_{2f}\dot{x}_{1f} + L_fC_{2f}\dot{\alpha} + K_{2f}x_2 - (K_{2f}+K_{1f})x_{1f} + L_fK_{2f}\alpha + K_{1f}u_f,$$

$$m_{1b}\ddot{x}_{1b} = C_{2b}\dot{x}_2 - C_{2b}\dot{x}_{1b} - L_bC_{2b}\dot{\alpha} + K_{2b}x_2 - (K_{2b}+K_{1b})x_{1b} - L_bK_{2b}\alpha + K_{1b}u_b,$$

$$\begin{aligned} I_\alpha\ddot{\alpha} = &- C_3L_b\dot{x}_3 + (L_bC_3 + L_bC_{2b} - L_fC_{2f})\dot{x}_2 - L_bC_{2b}\dot{x}_{1b} + L_fC_{2f}\dot{x}_{1f} \\ &+ (L_b^2C_3 - L_b^2C_{2b} - L_f^2C_{2f})\dot{\alpha} - L_bK_3x_3 + (L_bK_3 + L_bK_{2b} - L_fK_{2f})x_2 \\ &- L_bK_{2b}x_{1b} + L_fK_{2f}x_{1f} + (L_b^2K_3 - L_b^2C_{2b} - L_f^2C_{2f})\alpha. \end{aligned}$$

Consequently, a first order ODE system $\dot{\omega} = A\omega + b$ can be derived, similar to Eq. (2.58) with the help of the block-matrix structure

$$A = \begin{bmatrix} D & S \\ I & 0 \end{bmatrix}, \tag{2.61}$$

where $D \in \mathbb{R}^{5\times5}$ is the **damping-coefficient-matrix** with

$$D = \begin{bmatrix} -\frac{C_3}{m_3} & \frac{C_3}{m_3} & 0 & 0 & -\frac{L_bC_3}{m_3} \\ \frac{C_3}{m_2} & -\frac{C_3+C_{2f}+C_{2b}}{m_2} & \frac{C_{2f}}{m_2} & \frac{C_{2b}}{m_2} & \frac{L_b(C_3+C_{2b})-L_fC_{2f}}{m_2} \\ 0 & \frac{C_{2f}}{m_{1f}} & -\frac{C_{2f}}{m_{1f}} & 0 & \frac{L_fC_{2f}}{m_{1f}} \\ 0 & \frac{C_{2b}}{m_{1b}} & 0 & -\frac{C_{2b}}{m_{1b}} & -\frac{L_bC_{2b}}{m_{1b}} \\ -\frac{L_bC_3}{I_\alpha} & \frac{L_b(C_3+C_{2b})-L_fC_{2f}}{I_\alpha} & \frac{L_fC_{2f}}{I_\alpha} & -\frac{L_bC_{2b}}{I_\alpha} & \frac{L_b^2(C_3-C_{2b})-L_f^2C_{2f}}{I_\alpha} \end{bmatrix} \tag{2.62}$$

and $S \in \mathbb{R}^{5\times5}$ is the **spring-coefficient-matrix** with

$$S = \begin{bmatrix} -\frac{K_3}{m_3} & \frac{K_3}{m_3} & 0 & 0 & -\frac{L_bK_3}{m_3} \\ \frac{K_3}{m_2} & -\frac{K_3+K_{2f}+K_{2b}}{m_2} & \frac{K_{2f}}{m_2} & \frac{K_{2b}}{m_2} & \frac{L_b(K_3+K_{2b})-L_fK_{2f}}{m_2} \\ 0 & \frac{K_{2f}}{m_{1f}} & -\frac{K_{2f}}{m_{1f}} & 0 & \frac{L_fK_{2f}}{m_{1f}} \\ 0 & \frac{K_{2b}}{m_{1b}} & 0 & -\frac{K_{2b}}{m_{1b}} & -\frac{L_bK_{2b}}{m_{1b}} \\ -\frac{L_bK_3}{I_\alpha} & \frac{L_b(K_3+K_{2b})-L_fK_{2f}}{I_\alpha} & \frac{L_fK_{2f}}{I_\alpha} & -\frac{L_bK_{2b}}{I_\alpha} & \frac{L_b^2(K_3-K_{2b})-L_f^2K_{2f}}{I_\alpha} \end{bmatrix}. \tag{2.63}$$

The identity $I \in \mathbb{R}^{5\times5}$ and the null matrix $O \in \mathbb{R}^{5\times5}$ have the same size as the damping-coefficient matrix and the spring-coefficient matrix. Furthermore, we have

$$\omega := \begin{pmatrix} \dot{x}_3 \\ \dot{x}_2 \\ \dot{x}_{1f} \\ \dot{x}_{1b} \\ \dot{\alpha} \\ x_3 \\ x_2 \\ x_{1f} \\ x_{1b} \\ \alpha \end{pmatrix} \quad \text{and} \quad b := \begin{pmatrix} 0 \\ 0 \\ \frac{K_{1f}}{m_{1f}}u_f \\ \frac{K_{1b}}{m_{1b}}u_b \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

where $\omega \in \mathbb{R}^{10}$ contains the velocity and state variables of the system, while the source vector $b \in \mathbb{R}^{10}$ directly shows, which states are affected by the control variables. Summarizing, one can see that we have developed a straight forward modelling technique to derive first order ODE systems from the energy terms $T(t,x,\dot{x})$, $V(t,x,\dot{x})$ and $U(t,\dot{x})$. Therefore, any coupled dynamical system could in principle be derived from the Euler-Lagrange equation, having pre-defined energy terms. Anyhow, we can see that if $F \in \mathbb{N}$ describes the DOFs of the system, then for each system matrix $A$ it holds that $A \in \mathbb{R}^{2F \times 2F}$. Consequently, describing more complex approximate vehicle models, makes it much more difficult to explicitly describe the mathematical system.

### 2.2.4 Full-Car-Model

Following the principles developed in the previous sections, we can derive a mathematical system for a Full-Car-Model (FCM) with 8DOF from the well-known Euler-Lagrange formalism. In this case, we consider a model with four wheel-suspensions, containing four different displacements of the road for each point of time. As far as the HCM has been concerned, we have used a rotational generalized coordinate $\alpha$, resulting from the longitudinal difference of the road displacements. Since we also consider lateral road displacements, we get an additional rotational coordinate, denoted by $\beta$. We herein follow the 7DOF-FCM as it can be found in [47, 72], where we again extend the model to a coupled passenger MSD component.

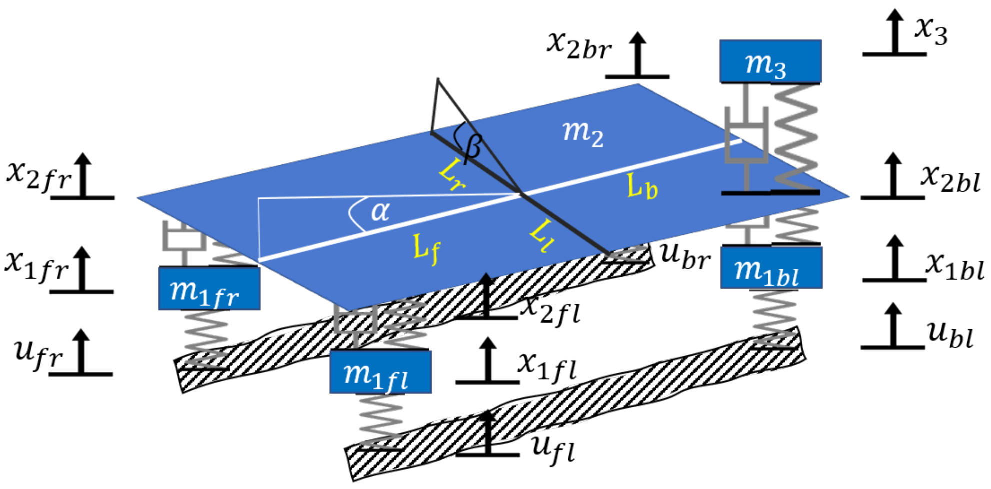


FIGURE 2.5: Passenger-based Full-Car-Model. Description of the Full-Car-Model is done in analogy to the Half-Car-Model, including four source terms, caused by displacements of the road, denoted by $u_{fr}$ for the front-right, $u_{fl}$ for the front-left, $u_{br}$ for the back-right and $u_{bl}$ for the back-left. Description of the displacements with corresponding masses then follow with consistent notation , comparable to the Half-Car-Model.

We shortly describe the principle of the FCM with the help of the above Figure 2.5. As already mentioned in the introduction of this section, we regard longitudinal displacements of the road as well as lateral displacements indexed by **"fr"** for the displacement that acts on the front right wheel, **"fl"** for the front left, **"br"** for the back right and finally **"bl"** for the back left. The FCM could therefore be considered as a coupled HCM, where the length $L_f$ describes the distance from front wheel to center of mass of the chassis, $L_b$ the distance from center to back, $L_r$ from center to right suspensions and $L_l$ from center to left suspensions. Then the total length of the chassis equals $L_f + L_b$, where the total width is given by the term $L_r + L_l$.

It therefore follows, that the displacement resulting from rotation with respect to $\alpha$ for the left of the chassis is given by $L_f\alpha$, for the right of the wheel by $-L_b\alpha$. Analogously, we have for the pure rotational displacement with respect to $\beta$, for the right of the chassis the vertical displacement $L_r\beta$ and for the left of the chassis as a consequence $-L_l\beta$.

The displacements for the edges of the chassis, denoted by $x_{2fr}$,$x_{2fl}$, $x_{2br}$ and $x_{2bl}$ can then be defined, similar to the HCM for one rotational DOF in Eq. (2.59) and Eq. (2.60) relative to the center state of the car-body $x_2$ via

$$x_{2fr} = x_2 + L_f\alpha + L_r\beta, \tag{2.64}$$

$$x_{2fl} = x_2 + L_f\alpha - L_l\beta, \tag{2.65}$$

$$x_{2br} = x_2 - L_b\alpha + L_r\beta, \tag{2.66}$$

$$x_{2bl} = x_2 - L_b\alpha - L_l\beta. \tag{2.67}$$

Furthermore, we assume that the passenger's seat is located at the left back of the chassis. Of course, the location of the seat is variable and can also vary for another car model. In addition, one could also model a multi-seat vehicle model with the herein described assumptions. For obvious reasons, we will not derive the explicit system matrix of the FCM at this point. We end this modelling section, with describing further steps, if one is interested in the system matrix of the FCM. As the potential energy $V$ depends on the spring length and the Rayleigh-Dissipation $U$ depends on the difference of velocity, one needs to take the l.h.s. of Eq. (2.64) - (2.67) to define the energy terms. Then, the r.h.s. needs to be inserted in the energy terms, analogously to the definition of the energy terms for the HCM, to guarantee a dependence on the generalized coordinates $x_2$, $\alpha$ and $\beta$.

As a final step, we define the point-wise evaluations of the generalized coordinate function $x \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}^8\right)$ with the help of the following expression

$$x(t) := \left(x_3, x_2, x_{1fr}, x_{1fl}, x_{1br}, x_{1bl}, \alpha, \beta\right)^T, \tag{2.68}$$

for $t \in [t_0,t_n]$. Then we can derive the equations of motion with the help of Eq. (2.46) from the energy terms $T(t,x,\dot{x})$, $V(t,x,\dot{x})$ and $U(t,\dot{x})$ and transfer them into a system of first order

ODE to get the explicit description of the system matrix $A \in \mathbb{R}^{16\times 16}$ for the FCM.

Deriving mathematical systems of approximate vehicle models, as coupled MSD systems, requires basic knowledge about rotational displacements and energy terms from the simple pendulum model and knowledge about vertical displacements in a MSD model. Then it is possible to derive a mathematical model for any coupled system, using the Euler-Lagrange formalism, based on variational calculus for motion in a system of free particles.

## 2.3 Road Modelling

We have shown how arbitrary dynamical systems can be derived from the Euler-Lagrange formalism in the previous section, especially for approximate vehicle models. Since for those models, we have always assumed, that a dynamical behaviour of the components results from a driving term, namely the road-profile, we need now to clarify, how such a profile can be modelled. Then, given any parameter configuration and realizations of the road-profile, the ODE system could even explicitly be solved.

We give a detailed description, how a road-profile according to the ISO 8606 road roughness classification can be generated. Therefore, we will mainly focus on [106] and develop a methodology based on Power Spectral Density (PSD) for roughness classification and a sinusoidal approximation of the road displacement.

### 2.3.1 Power Spectral Density and Fourier Transform

For the course of the following section, we assume the reader to be familiar with the basic concepts of continuous random variables and stochastic processes. For a more detailed introduction, we refer to [22]. For a more sophisticated description of probability theory, based on the concepts of measurable spaces, one could for instance refer to [11]. Furthermore, details concerning Fourier analysis that are used, can be found in [34] or [19]. We start by giving some basic probabilistic definitions to clarify the general setting.

**Definition 2** (Random Variable [11])
*Let $(\Psi, \mathscr{A}, \mathbb{P})$ be a probability space and $(S, \mathscr{S})$ be a measurable space. An $\mathscr{A} - \mathscr{S}-$measurable mapping*

$$X : \Psi \to S$$

*is called a* ***random variable****.*

**Definition 3** (Stochastic Process [11])
*Let $(\Psi, \mathscr{A}, \mathbb{P})$ be a probability space and $(S, \mathscr{S})$ be a measurable space. A collection of random variables $X = \{X(t)\}_{t\in\mathscr{T}}$, where $\mathscr{T} \subset \mathbb{R}$ is a finite time horizon, with values in $S$ is called a* ***stochastic process****.*

**Definition 4** (Stationary Process [11])
*A stochastic process $\{X(t)\}_{t\in\mathscr{T}}$ is **weakly stationary**, for all $t\in\mathscr{T}$ if*

- *The expected value $\mathbb{E}[X(t)]$ is constant for all $t\in\mathscr{T}$.*
- *The variance $\operatorname{Var}[X(t)] = \mathbb{E}\left[(X(t)-\mathbb{E}[X(t)])^2\right]$ is finite for all $t\in\mathscr{T}$.*

*Then the **autocorrelation function** is independent on time and only depends on the time delay, such that for $t,\ s\in\mathscr{T}$ it holds that*

$$R(\tau) := \mathbb{E}\left[X(t)\overline{X(t+\tau)}\right] = \mathbb{E}\left[X(s)\overline{X(s+\tau)}\right], \tag{2.69}$$

*with $\tau\in\mathbb{R}$, such that $t+\tau,\ s+\tau\in\mathscr{T}$, where $\overline{X(t)}$ is the complex conjugate of $X(t)$.*

If $X(t)$ is real-valued for all $t\in\mathscr{T}$, then $\overline{X(t)} = X(t)$. Furthermore, let us assume that $s=t-\tau$, then with Eq. (2.69), we get for a real-valued stationary process that

$$R(\tau) = \mathbb{E}[X(s)X(s+\tau)] = \mathbb{E}[X(t-\tau)X(t)] = \mathbb{E}[X(t)X(t-\tau)] = \mathbb{E}[X(t)X(t+\tau)]. \tag{2.70}$$

We restrict ourselves to real-valued signals, here expressed by realizations of a stochastic process. The Fourier transform [34] can in general be defined for continuously integrable functions, which we use for the following section.

**Definition 5** (Fourier Transform [34])
*Let $f\in\mathscr{L}^1(\mathbb{R},\mathbb{R})$ be a continuously integrable function. Then the **Fourier Transform** $\hat{f}:\mathbb{R}\to\mathbb{R}$ is defined by*

$$\hat{f}(s) = \int_{-\infty}^{\infty} e^{-isx} f(x)\mathrm{d}x \tag{2.71}$$

*for $s\in\mathbb{R}$ and $i=\sqrt{-1}$ the imaginary variable.*

**Theorem 1** (Inversion of Fourier Transform [34])
*Let $f\in\mathscr{L}^1(\mathbb{R},\mathbb{R})$ be a continuously integrable function. Let $\hat{f}:\mathbb{R}\to\mathbb{R}$ be the Fourier transform of $f$. Then it holds that*

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{isx}\hat{f}(s)\mathrm{d}s \tag{2.72}$$

*for $x\in\mathbb{R}$.*

Since we want to define roughness degrees for the road-profiles, we define the Power Spectral Density (PSD) with the help of the following theorem.

**Theorem 2** (Wiener-Khintchine-Theorem [70])
*Let the autocorrelation function for a time delay $\tau\in\mathbb{R}$ of a stationary process $\{X(t)\}_{t\in\mathbb{R}}$ be defined by $R(\tau)$. Let furthermore, the spacial frequency $\Omega$ be defined by $\Omega=\frac{2\pi}{\lambda}$, where $\lambda>0$ is the wave length of a signal. The **Power Spectral Density** is then defined by*

$$S(\Omega) := \lim_{T\to\infty}\mathbb{E}\left[\frac{1}{2T}\left|\mathfrak{F}\{X_T\}(\Omega)\right|^2\right] \tag{2.73}$$

*for $T \geq 0$ and with the Fourier transform of the truncated process being defined as*

$$\mathfrak{F}\{X_T\}(\Omega) := \int_{-T}^{T} X(\tau)e^{-i\Omega\tau}\mathrm{d}\tau.$$

*It then holds that*

$$S(\Omega) = \int_{-\infty}^{\infty} R(\tau)e^{-i\Omega\tau}\mathrm{d}\tau \tag{2.74}$$

*and further*

$$R(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\Omega)e^{i\Omega\tau}\mathrm{d}\Omega. \tag{2.75}$$

*Proof.* We mainly follow [70] to prove the theorem. Since the absolute square of a complex number is given by $|z|^2 = z\bar{z}$, it can be verified with the help of Fubini's theorem that

$$\begin{aligned}
|\mathfrak{F}\{X_T\}(\Omega)|^2 &= \mathfrak{F}\{X_T\}(\Omega)\overline{\mathfrak{F}\{X_T\}(\Omega)} \\
&= \mathfrak{F}\{X_T\}(\Omega)\int_{-T}^{T} X(s)e^{i\Omega s}\mathrm{d}s \\
&= \int_{-T}^{T} \mathfrak{F}\{X_T\}(\Omega)X(s)e^{i\Omega s}\mathrm{d}s \\
&= \int_{-T}^{T}\int_{-T}^{T} X(t)e^{-i\Omega t}\mathrm{d}tX(s)e^{i\Omega s}\mathrm{d}s \\
&= \int_{-T}^{T} X(t)\int_{-T}^{T} X(s)e^{-i\Omega(t-s)}\mathrm{d}s\mathrm{d}t.
\end{aligned}$$

We now use change of variables with $s = t - \tau$. It then holds, since we want to compute the integral for $s \in [-T, T]$, that $\tau \in [T+t, -T+t]$ and we set $\mathrm{d}s = -\mathrm{d}\tau$. Then the above integral equals again with Fubini's theorem

$$\begin{aligned}
\int_{-T}^{T} X(t)\cdot\left(-\int_{T+t}^{-T+t} X(t-\tau)e^{-i\Omega\tau}\mathrm{d}\tau\right)\mathrm{d}t &= \int_{-T}^{T} X(t)\int_{-T+t}^{T+t} X(t-\tau)e^{-i\Omega\tau}\mathrm{d}\tau\mathrm{d}t \\
&= \int_{-T}^{T}\int_{-T+t}^{T+t} X(t)X(t-\tau)e^{-i\Omega\tau}\mathrm{d}t\mathrm{d}\tau.
\end{aligned}$$

Computation of the expected value for both sides of the equation and applying Eq. (2.70) for real-valued functions then yields

$$\begin{aligned}
\mathbb{E}\left[|\mathfrak{F}\{X_T\}(\Omega)|^2\right] &= \int_{-T}^{T}\int_{-T+t}^{T+t} \mathbb{E}\left[X(t)X(t-\tau)\right]e^{-i\Omega\tau}\mathrm{d}t\mathrm{d}\tau \\
&= \int_{-T}^{T}\int_{-T+t}^{T+t} 1\cdot R(\tau)e^{-i\Omega\tau}\mathrm{d}t\mathrm{d}\tau \\
&= 2T\int_{-T}^{T} R(\tau)e^{-i\Omega\tau}\mathrm{d}\tau.
\end{aligned}$$

Dividing both sides by $2T$ and computing the limit, then finally gives

$$S(\Omega) = \int_{-\infty}^{\infty} R(\tau)e^{-i\Omega\tau}\mathrm{d}\tau,$$

meaning, that the power spectral density $S(\Omega)$ equals the Fourier transform of the autocorrelation function $R(\tau)$. Then it directly follows with Theorem 1 that

$$R(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\Omega)e^{i\Omega\tau}\mathrm{d}\tau,$$

which completes the proof of the theorem. $\square$

Since we consider the stochastic process $\{X(t)\}_{t\in\mathbb{R}}$ to serve as a random, real-valued signal, the autocorrelation function is also real and even, such that $R(\tau) = R(-\tau)$. The PSD then simplifies to

$$S(\Omega) = 2\int_{0}^{\infty} R(\tau)\mathfrak{Re}(e^{-i\Omega\tau})\mathrm{d}\tau = 2\int_{0}^{\infty} R(\tau)\cos(\Omega\tau)\mathrm{d}\tau \tag{2.76}$$

and the autocorrelation yields

$$R(\tau) = \frac{1}{\pi}\int_{0}^{\infty} S(\Omega)\mathfrak{Re}(e^{i\Omega\tau})\mathrm{d}\Omega = \frac{1}{\pi}\int_{0}^{\infty} S(\Omega)\cos(\Omega\tau)\mathrm{d}\Omega. \tag{2.77}$$

Defining the one-sided PSD by $\Phi(\Omega) := 2S(\Omega)$, for $\Omega \geq 0$, then gives

$$R(\tau) = \frac{1}{2\pi}\int_{0}^{\infty} \Phi(\Omega)\cos(\Omega\tau)\mathrm{d}\Omega. \tag{2.78}$$

The herein shown and proved relation between PSD and autocorrelation is essential for the following section, where we finally model a one-dimensional longitudinal road-profile.

### 2.3.2 Random Time Series and Sinusoidal Approach

We introduce a method to compute random road-profiles from realizations of uniformly distributed random variables, following a sinusoidal approximation approach [106]. General definitions from probability theory are again based on [11, 22].

**Definition 6** (Uniform Distribution [11])
*Let $\phi$ be a continuous random variable with $\phi : \Psi \to S$ and values $s = \phi(x)$ in $[a,b] \subset \mathbb{R}$. If the probability distribution is given by $p(s) = \frac{1}{b-a}\mathbb{1}_{[a,b]}(s)$, then $\phi$ is called a* ***uniformly*** *distributed random variable. We denote $\phi \sim \mathscr{U}([a,b])$ as a short notation.*

The following theorem needs to be proved to uniquely define the road-profiles.

**Theorem 3** (Sinusoidal Road-Profiles [106])
*Let $\phi_i \sim \mathscr{U}([0,2\pi))$ be identical and independently distributed (i.i.d.) random variables for $i \in \{1,2,...,N\}$ with $N \in \mathbb{N}$, such that a* ***track of a road-profile*** *can be defined by*

$$u(s) := \sum_{i=1}^{N} A_i \sin(\Omega_i s - \phi_i), \tag{2.79}$$

*where $A_i \geq 0$ is the amplitude, $\Omega_i$ a specific frequency and $s \in [0, L_u]$ the current position of the track for a road with length $L_u \geq 0$. It then holds that*

$$\mathbb{E}[u(s)] = 0 \quad \text{and} \quad \operatorname{Var}[u(s)] = \frac{1}{2}\sum_{i=1}^{N} A_i^2.$$

*Proof.* A very short description of the proof is shown in [106]. For a further understanding, more intermediate steps are done at this point by ourself. Since $\sin(x-y) = \sin(x)\cos(y) - \cos(x)\sin(y)$ for $x, y \in \mathbb{R}$, it holds that

$$\begin{aligned}
\mu := \mathbb{E}[u(s)] &= \sum_{i=1}^{N} A_i \mathbb{E}[\sin(\Omega_i s - \phi_i)] \\
&= \sum_{i=1}^{N} A_i \left(\sin(\Omega_i s)\mathbb{E}[\cos(\phi_i)] - \cos(\Omega_i s)\mathbb{E}[\sin(\phi_i)]\right) \\
&= 0,
\end{aligned}$$

which results from the uniform distribution of the random variables $\phi_i$ with

$$\mathbb{E}[\cos(\phi_i)] = \int_{\mathbb{R}} \cos(x)\frac{1}{2\pi}\mathbb{1}_{[0,2\pi)}(x)\mathrm{d}x = \frac{1}{2\pi}\int_0^{2\pi}\cos(x)\mathrm{d}x = 0$$

and

$$\mathbb{E}[\sin(\phi_i)] = \int_{\mathbb{R}} \sin(x)\frac{1}{2\pi}\mathbb{1}_{[0,2\pi)}(x)\mathrm{d}x = \frac{1}{2\pi}\int_0^{2\pi}\sin(x)\mathrm{d}x = 0,$$

where $\mathbb{1}_{[0,2\pi)}(x)$ is the indicator function for the interval $[0, 2\pi)$. Further the variance can be computed, using the property of independence for the expected values. Then

$$\begin{aligned}
\sigma^2 := \operatorname{Var}[u(s)] &= \mathbb{E}\left[(u(s)-\mu)^2\right] = \mathbb{E}\left[(u(s))^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{N} A_i \sin(\Omega_i s - \phi_i)\right)\left(\sum_{j=1}^{N} A_j \sin(\Omega_j s - \phi_j)\right)\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{N} A_i^2 \sin^2(\Omega_i s - \phi_i)\right) + \left(\sum_{\substack{i,j=0 \\ i\neq j}}^{N} A_i A_j \sin(\Omega_i s - \phi_i)\sin(\Omega_j s - \phi_j)\right)\right] \\
&= \sum_{i=1}^{N} A_i^2 \mathbb{E}\left[\sin^2(\Omega_i s - \phi_i)\right] + \sum_{\substack{i,j=0 \\ i\neq j}}^{N} A_i A_j \mathbb{E}[\sin(\Omega_i s - \phi_i)\sin(\Omega_j s - \phi_j)] \\
&= \sum_{i=1}^{N} A_i^2 \mathbb{E}\left[\sin^2(\Omega_i s - \phi_i)\right] + \sum_{\substack{i,j=0 \\ i\neq j}}^{N} A_i A_j \underbrace{\mathbb{E}[\sin(\Omega_i s - \phi_i)]}_{=0}\underbrace{\mathbb{E}[\sin(\Omega_j s - \phi_j)]}_{=0} \\
&= \sum_{i=1}^{N} A_i^2 \mathbb{E}\left[\sin^2(\Omega_i s - \phi_i)\right].
\end{aligned}$$

The expected value for the squared sine equals

$$\mathbb{E}\left[\sin^2(\Omega_i s - \phi_i)\right] = \int_{\mathbb{R}} \sin^2(\Omega_i s - x)\frac{1}{2\pi}\mathbb{1}_{[0,2\pi)}(x)\mathrm{d}x$$

$$
\begin{aligned}
&= \frac{1}{2\pi}\int_0^{2\pi} \sin^2(\Omega_i s - x)\mathrm{d}x \\
&= \frac{1}{2\pi}\int_{\Omega_i s}^{\Omega_i s+2\pi} \sin^2(y)\mathrm{d}y \\
&= \frac{1}{2\pi}\left(\frac{1}{2}\left(y-\cos(y)\sin(y)\right)|_{\Omega_i s}^{\Omega_i s+2\pi}\right) \\
&= \frac{1}{2}
\end{aligned}
$$

for all $i \in \{1,2,...,N\}$. Therefore,

$$\sigma^2 = \frac{1}{2}\sum_{i=1}^{N} A_i^2,$$

which completes the proof. □

It is easy to verify with Eq. (2.70), that for $\tau = 0$, the autocorrelation $R(0)$ is equal to the variance $\sigma^2$. Then by Eq. (2.78),

$$\sigma^2 = R(0) = \frac{1}{2\pi}\int_0^{\infty}\Phi(\Omega)\mathrm{d}\Omega \approx \frac{1}{2\pi}\sum_{i=0}^{N}\Phi(\Omega_i)\Delta\Omega \tag{2.80}$$

for $N$ sufficiently large, where $\Omega_i$ are the wave numbers to lie in an equidistant grid with $\Omega_{i+1} = \Omega_i + \Delta\Omega$ and $\Delta\Omega = \frac{\Omega_N - \Omega_1}{N-1}$. As it is described in [106], the ISO 8606 uses $\Omega_0 = 1\frac{1}{m}$, $\Omega_1 = 0.02\pi\frac{1}{m}$ and $\Omega_N = 6\pi\frac{1}{m}$.

It then follows with Theorem 3 that

$$\sigma^2 = \frac{1}{2}\sum_{i=1}^{N} A_i^2 \approx \frac{1}{2}\sum_{i=1}^{N}\Phi(\Omega_i)\frac{\Delta\Omega}{\pi},$$

which implies that for all $i \in \{1,2,...,N\}$

$$A_i \approx \sqrt{\Phi(\Omega_i)\frac{\Delta\Omega}{\pi}}. \tag{2.81}$$

Following again ISO 8606 the PSD can be computed as

$$\Phi(\Omega_i) = \Phi(\Omega_0)\left(\frac{\Omega_1^{-2}}{\Omega_0}\right) \quad \text{for} \quad \Omega_1 < \Omega_i \le \Omega_N, \tag{2.82}$$

where the degree of roughness then depends upon the value of $\Phi(\Omega_0)$. In detail, for road-profile classification [106], we have a road of class **A**, if $\Phi(\Omega_0) = 4^0 \cdot 10^{-6} m^3$, a road of class **B** for $\Phi(\Omega_0) = 4^1 \cdot 10^{-6} m^3$ and a road of class **C**, if $\Phi(\Omega_0) = 4^2 \cdot 10^{-6} m^3$. where the value $\Phi(\Omega_0)$ characterizes the degree of roughness, which directly affects the scaling of the amplitudes $A_i$.

The below Figure 2.6 shows an example for a sinusoidal approximation of a longitudinal road-profile, using the same realization of the random variables $\phi_i$ for different $\Phi(\Omega_0)$. The result is then a change in the amplitude of the road-profile, which characterizes the degree of

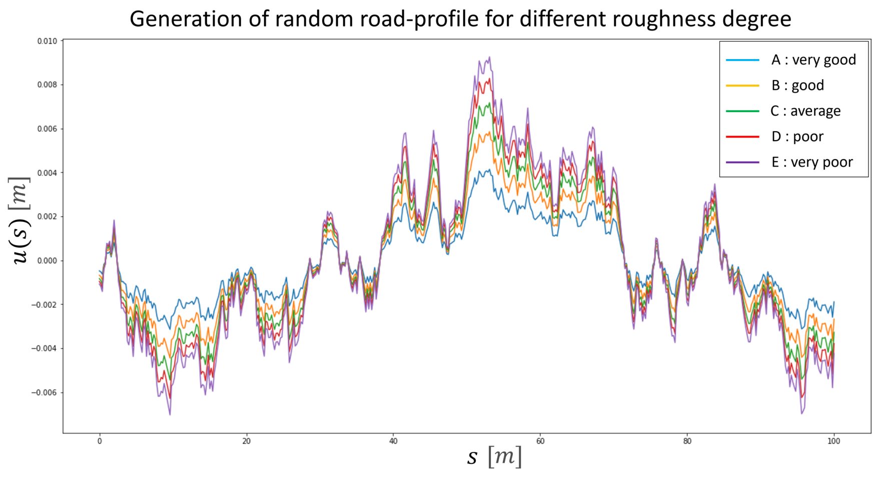


FIGURE 2.6: Road-Profile Modelling for one realization of the random-variable $\phi_i$ for $i \in \{1,2,...,1000\}$. The current position of the road is given by $s$ on the abscissa, where the corresponding road-displacement $u(s)$ is shown on the ordinate. One can recognize that the realizations differ for a higher roughness degree on the scaling of the road-profile for class A.

roughness from good profiles to bad profiles.

We have shown and derived a detailed concept to construct mathematical descriptions of approximate vehicle models, especially the Quarter-Car-Model, driven by an external displacement of a randomly generated road-profile in this section. As the models can be described as a system of first order ordinary differential equations, it is therefore possible to explicitly solve the equations, resulting in an explicit description of the displacements for the individual components of the model.

# Chapter 3

# Neural Networks

Artificial Neural Networks (ANN), especially Deep Neural Networks (DNN), have become one of the main attractors in modern research. Some of the most successful achievements in Deep Learning (DL) are probably in autonomous game solving by Reinforcement Learning (RL) [88, 103] and of course image processing, dimension reduction and object classification via Convolutional Neural Networks (CNN).

For the latter, Neural Network algorithms have been developed that even outperform human beings as far as classification skills are concerned. Moreover, compressing networks like Auto-Encoders [111], have intensively been studied to find low-dimensional representations of high-dimensional input samples in competition to Principle Component Analysis (PCA). This pure reduction, using the bottleneck structure of an Auto-Encoder, also named latent space, has been extended to generative networks like the prominent Variational Auto-Encoder (VAE) [57] or Generative Adversarial Networks (GAN) [20]. Since it can in general be expensive to acquire a sufficiently large amount of data to efficiently train a Neural Network to serve as a function estimator for a specific mapping from input space to target space, generative models are used to artificially extend given small-sized datasets.

In case of sequential data, creating synthetic dataset of a dynamical system could of course also be done by "white box models" (numerical solvers) instead of the herein described generative "black box models" (ANN), as we will investigate in Chapter 4.

As Deep Learning can be categorized in the broad field of Statistical Learning Theory (SLT), for data-driven multidimensional function approximations, we start this section with a short introduction to function estimation and risk minimization in Section 3.1. Connecting risk minimization to Empirical Risk Minimization (ERM), can then be applied to the common Deep Learning terminology of training, testing and generalization. We will investigate the general training of Neural Networks as gradient-based optimization, using the Backpropagation-algorithm (BP) and Stochastic Gradient Descent (SGD).

Section 3.2 then gives more specific insights in explicit Neural Network architectures, especially Fully-Connected (FC) networks, Convolutional Autoencoders (CAE) and U-Net, which play essential roles for parameter estimation problems for uncertain systems in Chapter 5.

## 3.1 Statistical Learning Theory

We want to give a short introduction to Statistical Learning Theory (SLT), as it is given in detail by [109]. The general setting and motivation is then sufficient to describe the learning problem for Neural Networks as they can be defined to result from minimizing a risk functional connected to a learning machine. A modern mathematical description of the Deep Learning problem and function approximation is also stated in [10] in the context of SLT. We therefore focus and combine the main principles given in [10, 109] to clarify the setting of the Neural Network training problem.

### 3.1.1 Learning Problem and Risk

The learning problem in general is set in continuous spaces, as data samples can in theory be assumed to be drawn from a probability distribution [109]. We therefore assume to have data samples $x \in \mathcal{X}$ and corresponding target values $y \in \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are general spaces. Then a pair $(x,y) \in \mathcal{X} \times \mathcal{Y}$ can be assumed to follow a joint probability density function $p(x,y)$. If we assume that for instance $x$ is vector-valued with dimension $d \in \mathbb{N}$, then $\mathcal{X} = \mathbb{R}^d$. The same holds for $y$.

Given finite subsets of the joint space $\mathcal{X} \times \mathcal{Y}$, denoted by $\mathcal{S}$ and $\mathcal{T}$ with $\mathcal{S} \cap \mathcal{T} = \emptyset$, the task of the learning problem is then defined to find a mapping $f : \mathcal{X} \to \mathcal{Y}$ such that $f(x) = y$ for all $(x,y) \in \mathcal{S} \cup \mathcal{T}$. The finite subsets are specified in a latter part of this section. As a first step, we give a detailed definition about the function space.

**Definition 7** (Set of measurable functions [44])
*Assume $(\mathcal{X}, \mathscr{A})$ and $(\mathcal{Y}, \mathscr{B})$ to be measurable spaces, where $\mathscr{A}$ is a $\sigma$-algebra on $\mathcal{X}$ and $\mathscr{B}$ a $\sigma$-algebra on $\mathcal{Y}$. A function*

$$f : \mathcal{X} \to \mathcal{Y}$$

*is called $(\mathscr{A}, \mathscr{B})$-measurable if*

$$f^{-1}(B) = \{x \in \mathcal{X} \mid f(x) \in B\} \in \mathscr{A} \tag{3.1}$$

*for all $B \in \mathscr{B}$. The set of $(\mathscr{A}, \mathscr{B})$-measurable functions is defined by the* ***function space***

$$\mathscr{M}(\mathcal{X}, \mathcal{Y}) := \{f : \mathcal{X} \to \mathcal{Y} \mid f \text{ is } (\mathscr{A}, \mathscr{B}) - \text{measurable}\}. \tag{3.2}$$

The set $\mathscr{M}(\mathcal{X}, \mathcal{Y})$ in general describes all possible measurable functions, mapping from $\mathcal{X}$ to $\mathcal{Y}$. As far as statistical learning for Neural Networks is concerned, we restrict to a specific type of parametrized functions, depending upon a finite set of parameters. Let therefore the parameters be denoted by $\theta$ of a parameter space $\Theta$, which characterizes the $(\mathscr{A},\mathscr{B})$-measurable functions $f_\theta$. If the function is characterized by $n \in \mathbb{N}$ different parameters, given as entries in $\theta$, then $\Theta = \mathbb{R}^n$. Applied to Eq. (3.2), the **parametrized function space** can therefore be defined by

$$\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y}) := \{f_\theta : \mathcal{X} \to \mathcal{Y} \ \mid \ f_\theta \text{ is } (\mathscr{A},\mathscr{B}) - \text{measurable}, \theta \in \Theta\}. \tag{3.3}$$

The above definition of parametrized function spaces states that $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y}) \subset \mathscr{M}(\mathcal{X},\mathcal{Y})$, which means that due to the constraint of parametrizable functions $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, there exist other measurable functions $f \in \mathscr{M}(\mathcal{X},\mathcal{Y}) \setminus \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$. If we restrict to parametrizable functions $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, it is obviously to have a bijection $\theta \leftrightarrow f_\theta$. One can therefore restrict to the parameter space $\Theta$ instead of considering the function space $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, which is in addition significantly simpler. Nevertheless, it is possible to have $\theta_\alpha$, $\theta_\beta \in \Theta$ with $f_{\theta_\alpha}(x) = y$ for all $\mathcal{S} \cup \mathcal{T}$ and $f_{\theta_\beta}(x) = y$ for all $\mathcal{S} \cup \mathcal{T}$, which does in general not imply $\theta_\alpha = \theta_\beta$.

Since not necessarily $f_\theta(x) = y$ for all $(x,y) \in \mathcal{S} \cup \mathcal{T}$ for any $\theta \in \Theta$, one needs to make use of a **loss-function**, which evaluates the choice of $\theta \in \Theta$ for the learning problem.

**Definition 8** (Loss-Function [10])
*Let $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$ be a set of measurable functions $f_\theta : \mathcal{X} \to \mathcal{Y}$ and $\theta \in \Theta$. Then*

$$\begin{aligned} \mathscr{L} : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R} \\ (\tilde{y}, y) &\mapsto \mathscr{L}(\tilde{y}, y) \end{aligned} \tag{3.4}$$

*is called a **loss-function** for a Neural Network $f_\theta$, if $\tilde{y} = f_\theta(x)$.*

**Example 1** (Regression)
*Let $f_\theta : \mathbb{R}^n \to \mathbb{R}$, $n \in \mathbb{N}$ and $(x,y) \in \mathbb{R}^n \times \mathbb{R}$. Then for a **regression** learning task [109], one can use the loss-function*

$$\mathscr{L}(f_\theta(x), y) = (f_\theta(x) - y)^2.$$

**Example 2** (Binary Classification)
*Let $f_\theta : \mathbb{R}^n \to \{0,1\}$, $n \in \mathbb{N}$ and $(x,y) \in \mathbb{R}^n \times \{0,1\}$. Then for a **binary classification** task [109], one can use the loss-function*

$$\mathscr{L}(f_\theta(x), y) = \begin{cases} 0, & \text{if} \quad f_\theta(x) = y \\ 1, & \text{else.} \end{cases}$$

**Example 3** (Denoising)
*Let $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$, $n \in \mathbb{N}$, where for $\tilde{x} \in \mathcal{X} = \mathbb{R}^n$ and $x \in \mathcal{Y} = \mathbb{R}^n$ it holds that $\tilde{x} = x + \xi$. Here, by $\xi \in \mathbb{R}^n$ we denote an additive error term, hence $\tilde{x}$ is a corrupted version of a clean sample $x$. Then for a **denoising** task, one can use the loss-function*

$$\mathscr{L}(f_\theta(\tilde{x}), x) = \frac{1}{n} \sum_{i=1}^{n} (f_\theta(\tilde{x})_i - x_i)^2.$$

Data samples $(x,y)$ have been assumed in the introduction of this section to be randomly drawn from $\mathcal{X} \times \mathcal{Y}$ following the joint density function $p(x,y)$. As a next step, we want to define optimality for a loss-function, since we want to find criteria to solve the training problem for Neural Networks.

**Definition 9** (Expected risk [109])
*Assume that there is a set of parametrized measurable functions given by $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$ and a loss-function $\mathscr{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Then the **expected risk** on $\mathcal{X} \times \mathcal{Y}$ is defined as*

$$\mathscr{R}(f_\theta) := \mathbb{E}\left[\mathscr{L}\left(f_\theta(x),y\right)\right] = \int_{\mathcal{X}\times\mathcal{Y}} \mathscr{L}\left(f_\theta(x),y\right) \mathrm{d}p(x,y). \tag{3.5}$$

**Definition 10** (Bayes Error and Bayes Optimality [109])
*Assume that a measurable parametrized function $f_{\theta^*} \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, has the smallest value for the expected risk $\mathscr{R}^* = \mathscr{R}(f_{\theta^*})$ for a given loss function $\mathscr{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, namely*

$$\mathscr{R}(f_{\theta^*}) := \inf_{\theta\in\Theta} \mathscr{R}(f_\theta). \tag{3.6}$$

*Then $\mathscr{R}^*$ is called **Bayes Error** and $f_{\theta^*}$ is a **Bayes-optimal** function.*

Since in general, the density function $p(x,y)$ for $(x,y) \in \mathcal{X} \times \mathcal{Y}$ is unknown, it is not possible to analytically find a Bayes-optimal function in sense of Definition 10. We therefore need to restrict to finite subsets of the continuous data spaces, to find a sufficiently good estimate $\hat{\theta} \approx \theta^*$.

### 3.1.2 Empirical Risk Minimization

As we cannot in general compute the expected risk due to the unknown data distribution, we make use of Empirical Risk Minimization (ERM), as described by [10, 109] to approximate the true value of the expected risk. We therefore focus on a finite training dataset with arbitrary size, which can also be used to find an approximation to the Bayes-optimal function.

Let us therefore assume that $(x_m,y_m) \in \mathcal{X} \times \mathcal{Y}$, for $m \in \{1,2,...,N_S\}$ with $N_S \in \mathbb{N}$, describe a finite set of realizations of the continuous joint space $\mathcal{X} \times \mathcal{Y}$. Then, one can define a set, which contains all pairs $(x_m,y_m)$, for $m \in \{1,2,...,N_S\}$ by

$$\mathcal{S} = \{(x_m,y_m)\}_{m=1}^{N_S} \subset \mathcal{X} \times \mathcal{Y}. \tag{3.7}$$

Such a finite set $\mathcal{S}$ can for instance be used to describe a **training dataset**, where the elements $(x_m,y_m) \in \mathcal{S}$ for all $m \in \{1,2,...,N_S\}$, satisfy several data-specific conditions. A precise example of such a dataset is given in Chapter 4. Similar to Definition 9, which describes the expected risk for the continuous data space, we can then use a finite dataset $\mathcal{S}$ to find an analogous discrete expression.

**Definition 11** (Empirical risk [109])
*The **empirical risk** for an arbitrary large dataset $\mathcal{S} = \{(x_m,y_m)\}_{m=1}^{N_S} \subset \mathcal{X} \times \mathcal{Y}$ with $N_S \in \mathbb{N}$,*

*a parametrized measurable function $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$ and a loss-function $\mathscr{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is defined by*

$$\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta) := \frac{1}{N_S} \sum_{m=1}^{N_S} \mathscr{L}\left(f_\theta(x_m), y_m\right). \tag{3.8}$$

*If $\mathcal{S}$ is a training dataset, then $\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta)$ can also be referred to the **training error**.*

Comparable to the learning problem for the continuous risk functional, we can now search for a candidate $\hat{\theta} \in \Theta$ that minimizes the empirical risk. The aim of minimizing the empirical risk for a sufficiently large training dataset is then that $\hat{\theta} \approx \theta^*$. This is motivated by the weak law of large numbers (WLLN) [22], where we assume to have convergence in probability of the empirical risk to the expected risk, given by

$$\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta) = \frac{1}{N_S} \sum_{m=1}^{N_S} \mathscr{L}\left(f_\theta(x_m), y_m\right) \xrightarrow[N_S \to \infty]{\mathbb{P}} \mathbb{E}\left[\mathscr{L}\left(f_\theta(x), y\right)\right] = \mathscr{R}(f_\theta). \tag{3.9}$$

As a consequence this means, that if $\hat{\theta}$ minimizes the empirical risk for $N_S \to \infty$, then $\hat{\theta}$ is also a good candidate to find an approximation of the Bayes-optimal function by $f_{\hat{\theta}} \approx f_{\theta^*}$.

Since the parameter $\theta \in \Theta$ is usually a vector of thousands or millions of elements, it is again non-trivial to find an appropriate value $\hat{\theta}$, therefore, one tries to find a sequence, $\{\theta^l\}_{l=1}^N \subset \Theta$ with $N \in \mathbb{N}$, for instance iteratively, to get

$$\hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^N}) \approx \hat{\mathscr{R}}_{\mathcal{S}}(f_{\hat{\theta}}) \tag{3.10}$$

for $N \to \infty$. As the training dataset is a finite subset of the entire data space $\mathcal{X} \times \mathcal{Y}$, it can happen, that there exist elements $(x,y) \neq \mathcal{S}$ with a large value of a given loss-function. Therefore, we want to define a disjoint set of finite samples, to have another measure of quality for measurable functions $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$.

Let us therefore have $\mathcal{S} = \{(x_m, y_m)\}_{m=1}^{N_S} \subset \mathcal{X} \times \mathcal{Y}$ with $N_S \in \mathbb{N}$ and assume that $\mathcal{S}$ is a finite training set. Then

$$\mathcal{T} = \{(x_m, y_m)\}_{m=N_S+1}^{N_T} \subset \mathcal{X} \times \mathcal{Y} \tag{3.11}$$

is another set of realizations, with $N_T \in \mathbb{N}$ elements, which is not used to find a sequence $\{\theta^l\}_{l=1}^N \subset \Theta$ that minimizes the empirical risk, therefore $\mathcal{S} \cap \mathcal{T} = \emptyset$. The disjoint set $\mathcal{T}$ is then called a **test dataset**.

Similar to the empirical risk for the training data can be defined as the training error, we can also apply the definition of the empirical risk to the test dataset $\mathcal{T}$, giving

$$\hat{\mathscr{R}}_{\mathcal{T}}(f_\theta) := \frac{1}{N_T} \sum_{m=N_S+1}^{N_T} \mathscr{L}\left(f_\theta(x_m), y_m\right), \tag{3.12}$$

which is known as the **test error**. One can then compare the terms $\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta)$ and $\hat{\mathscr{R}}_{\mathcal{T}}(f_\theta)$, which should be comparable since they are averaged functions of the loss-functions.

### 3.1.3 Training of Neural Networks

We have shown, that in theory, given a sufficiently large training set $\mathcal{S}$ of data samples, the empirical risk converges to the expected risk for the samples on a data space, which is the general idea of the generalization property for large training sets. The basic setting for the learning problem should therefore be sufficiently clear at this point. We have restricted the setting to general parametrized measurable functions in the previous section and now deal with more specific functions, namely (Deep) Neural Networks. As far as the following gradient-based optimization techniques, including the Back-Propagation algorithm [43] are concerned, it is not necessary to specify the explicit network structure at a first step.

As an initial step, we assume that a generalized DNN $f_\theta : \mathcal{X} \to \mathcal{Y}$ can be expressed by combinations of parametrized sub-functions $f_{\theta_j}^{(j)}$ for $j \in \{1,2,...,M\}$ with $M \in \mathbb{N}$, such that the full network can be expressed as

$$f_\theta = f_{\theta_M}^{(M)} \circ f_{\theta_{M-1}}^{(M-1)} \circ \cdots \circ f_{\theta_2}^{(2)} \circ f_{\theta_1}^{(1)} \tag{3.13}$$

where $\theta = \{\theta_1, ..., \theta_M\}$ is called the set of network parameters and $\theta_j$ are the layer-specific parameters of a layer, indexed by $j \in \{1,2,...,M\}$. The $j$-th layer can therefore formally be expressed by the sub-function $f_{\theta_j}^{(j)}$.

Each layer $f_{\theta_j}^{(j)}$ then consists of $K_j \in \mathbb{N}$ individual layer parameters, which can explicitly be defined by

$$\theta_j = \{\theta_{j;1}, \theta_{j;2}, ..., \theta_{j;K_j}\}$$

for $j \in \{1,2,...,M\}$.

For the set of functions $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, we assume $M$ to be fixed for one specific training process and the layers to only differ in the realization of the corresponding layer parameters. The number of layers $M$ is then also called **depth of the Neural Network**.

As can be found in [10], the training of a Neural Network can be formally defined by the now following definition. The training algorithm will later on be specified.

**Definition 12** (Training for Neural Networks [10])
*Given a training dataset $\mathcal{S} = \{(x_m, y_m)\}_{m=1}^{N_S}$ with size $N_S \in \mathbb{N}$, a set of Neural Networks $\mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$ and a local minimal point $\hat{\theta} \in \Theta$, which satisfies*

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \ \hat{\mathscr{R}}_{\mathcal{S}}(f_\theta),$$

*a* ***training algorithm*** *is given by iteratively applying the mapping*

$$\mathfrak{A} : \mathcal{S} \times \Theta \to \Theta \tag{3.14}$$

$$(S, \theta) \mapsto \mathfrak{A}(S, \theta), \tag{3.15}$$

*with $S \subset \mathcal{S}$, such that for a finite series, indexed by $l \in \{0,1,...,N\}$, we have $\theta^{l+1} = \mathfrak{A}(S^l, \theta^l)$. The current value of the network parameters are denoted by $\theta^l$, the subset of training samples, used for update $l+1$ is denoted by $S^l \subset \mathcal{S}$. The aim of the training algorithm is to terminate after $N \in \mathbb{N}$ iterations, such that $\hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^N}) \approx \hat{\mathscr{R}}_{\mathcal{S}}(f_{\hat{\theta}})$.*

We now want to apply gradient-based optimization techniques to find an explicit method for definition of a training algorithm. As the empirical risk (training error) needs to be minimized and as it is a sum of individual loss-functions, it is necessary to assume $\mathscr{L} \in \mathscr{C}^1(\mathcal{Y} \times \mathcal{Y}, \mathbb{R})$ and one can also conclude that it is then necessary to have $f_\theta \in \mathscr{C}^1(\Theta \times \mathcal{X}, \mathcal{Y})$.

Assume that $\hat{\theta} \in \Theta$ is a local minimal point for the empirical risk $\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta)$, then

$$\nabla_\theta \hat{\mathscr{R}}_{\mathcal{S}}(f_{\hat{\theta}}) = \frac{1}{N_S} \sum_{m=1}^{N_S} \nabla_\theta \mathscr{L}(f_{\hat{\theta}}(x_m), y_m) = 0. \tag{3.16}$$

If we have an appropriate candidate $\theta^N \in \Theta$ with $\hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^N}) \approx \hat{\mathscr{R}}_{\mathcal{S}}(f_{\hat{\theta}})$, then it should also hold that $\nabla_\theta \hat{\mathscr{R}}_{\mathcal{S}}(f_\theta)\big|_{\theta=\theta^N} \approx 0$.

Computation of $\nabla_\theta \mathscr{L}(f_\theta(x), y)$ for $(x,y) \in \mathcal{X} \times \mathcal{Y}$ is a non-trivial problem due to the depth of a Neural Network and the non-linearity of the layers. For the sake of simplicity we use the short notation $\nabla_\theta \mathscr{L} = \nabla_\theta \mathscr{L}(f_\theta(x), y)$. It is then obvious that

$$\nabla_\theta \mathscr{L} = (\nabla_{\theta_1} \mathscr{L}, ..., \nabla_{\theta_M} \mathscr{L})^T$$

and further

$$\nabla_{\theta_j} \mathscr{L} = \left( \frac{\partial \mathscr{L}}{\partial \theta_{j1}}, ..., \frac{\partial \mathscr{L}}{\partial \theta_{jK_j}} \right)^T$$

with $\frac{\partial \mathscr{L}}{\partial \theta_{ji}} = \frac{\partial \mathscr{L}}{\partial f_\theta} \frac{\partial f_\theta}{\partial \theta_{ji}}$ for $i \in \{1,2,...,K_j\}$ and $j \in \{1,2,...,M\}$. Since $f_\theta$ is a function composition, it obviously holds that

$$\frac{\partial f_\theta}{\partial \theta_{ji}} = \left( \prod_{i=M}^{j+1} \frac{\partial f_{\theta_i}}{\partial f_{\theta_{i-1}}} \right) \cdot \frac{\partial f_{\theta_j}}{\partial \theta_{ji}},$$

which will be explained in more detail for the now following Backpropagation algorithm.

We refer to [43] to a general explanation of the Backpropagation calculus and apply it to our notation of DNN. We mention at this point, that there are more explanations for specific layer structures and loss-functions as can be seen in [64, 91].

## Backpropagation and Gradient-Descent

We now show an efficient way to compute $\nabla_\theta \mathscr{L}$ in terms of $\frac{\partial \mathscr{L}}{\partial \theta_{ji}}$ for $i \in \{1,2,...,K_j\}$ and $j \in \{1,2,...,M\}$, where $M \in \mathbb{N}$ is the depth of the network, by applying the chain rule and investigating shared terms for the derivatives. Let us therefore, for simplicity, restrict to a general **training sample** $(x,y) \in \mathcal{S}$. Furthermore, assume that we can define the input-output relation for layer $j$ by

$$a^{(j)} := f^{(j)}_{\theta_j}\left(a^{(j-1)}\right) \tag{3.17}$$

for $j \in \{1,2,...,M\}$ with $a^{(0)} := x$ and $a^{(M)} := f_\theta(x)$. It is then obvious that for a layer $j \in \{1,2,...,M\}$, we have

$$\frac{\partial a^{(j)}}{\partial \theta_{ji}} = \frac{\partial a^{(j)}}{\partial a^{(j-1)}} \frac{\partial a^{(j-1)}}{\partial \theta_{ji}}.$$

For the output layer $j = M$ and the parameters of the $M$-th layer, it then holds that

$$\frac{\partial \mathcal{L}}{\partial \theta_{Mi}} = \frac{\partial \mathcal{L}}{\partial a^{(M)}} \frac{\partial a^{(M)}}{\partial \theta_{Mi}} = \delta^M \frac{\partial a^{(M)}}{\partial \theta_{Mi}}, \tag{3.18}$$

where $\delta^M := \frac{\partial \mathcal{L}}{\partial a^{(M)}}$. For the parameters of an arbitrary hidden layer $j \in \{1,2,...,M-1\}$, it can be shown that

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta_{ji}} &= \frac{\partial \mathcal{L}}{\partial a^{(M)}} \frac{\partial a^{(M)}}{\partial a^{(M-1)}} \cdots \frac{\partial a^{(j+1)}}{\partial a^{(j)}} \frac{\partial a^{(j)}}{\partial \theta_{ji}} \\ &= \delta^j \frac{\partial a^{(j)}}{\partial \theta_{ji}} \\ &= \delta^{j+1} \frac{\partial a^{(j+1)}}{\partial a^{(j)}} \frac{\partial a^{(j)}}{\partial \theta_{ji}}, \end{aligned}$$

where we can define the **differential operator** $\delta^j$ for $j \in \{1,2,...,M-1\}$ by

$$\delta^j := \frac{\partial \mathcal{L}}{\partial a^{(M)}} \prod_{l=M}^{j+1} \frac{\partial a^{(l)}}{\partial a^{(l-1)}} = \delta^M \prod_{l=M}^{j+1} \frac{\partial a^{(l)}}{\partial a^{(l-1)}}. \tag{3.19}$$

Obviously, it therefore holds that

$$\delta^j = \delta^{j+1} \frac{\partial a^{(j+1)}}{\partial a^{(j)}} \tag{3.20}$$

for $j \in \{1,2,...,M-1\}$. Then, Eq. (3.20) mathematically explains, what is meant by **"Backpropagation"**: Starting at the output layer $M$, one can use the equation to describe how the loss **"propagates"** through the hidden layers until it reaches the input layer.

One can find another description of the Backpropagation algorithm [64], including the terms **"weights"**, **"biases"**, **"activation functions"** and **"neurons"**. The description then restricts to Fully-Connected Neural Networks, which will be precisely explained in the next section. It can then also occur that the terms $\frac{\partial a^{(l)}}{\partial a^{(l-1)}}$ again need to be computed by applying the chain rule, since layers are in general combinations of simple linear operations evaluated by non-linear functions. Nevertheless, this general description of Backpropagation also holds and is sufficient to explain the computational effort to get the gradient of the loss-function for the training of Neural Networks.

Since DNN can consist of thousands or millions of parameters, and due to the non-linear structure of the Neural Network, it is almost impossible to find the optimal set of parameters of this high-dimensional non-convex optimization problem. Therefore Gradient-Descent methods are commonly used in training DNN, as they are capable to iteratively deliver good approximations to a local minimal point.

**Definition 13** (Gradient Descent for Neural Networks [33])
*Given an initial parameter set $\theta^0 \in \Theta$ for fixed Neural Network structures $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$, a loss-function $\mathscr{L} \in \mathscr{C}^1((\mathcal{Y}\times\mathcal{Y}),\mathbb{R})$ and a training dataset $\mathcal{S} \subset \mathcal{X}\times\mathcal{Y}$ with size $N_S \in \mathbb{N}$, the update-rule*

$$\theta^{k+1} = \theta^k - \eta_k \nabla_\theta \hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^k}) \tag{3.21}$$

*is called **Gradient-Descent optimization** scheme for a training step $k \in \mathbb{N}$, where $\eta_k > 0$ is the step-width or **learning-rate** of the training algorithm.*

It is obvious, due to $\hat{\mathscr{R}}_{\mathcal{S}}(f_\theta) = \frac{1}{N_S}\sum_{m=1}^{N_S} \mathscr{L}(f_\theta(x_m), y_m)$ that for a large training set, one step of Gradient-Descent optimization can require high computational effort. We therefore introduce now Stochastic Gradient-Descent, which is the standard optimization technique to efficiently train Deep Neural Networks for large training datasets.

## Stochastic Gradient Descent

We introduce Stochastic Gradient-Descent (SGD) for fast training of DNN as it can be found in [12, 33]. Therefore, some terms have to be explained beforehand.

**Definition 14** (Training Batch)
*Given a training set $\mathcal{S} \subset \mathcal{X}\times\mathcal{Y}$ with size $N_S \in \mathbb{N}$ such that we find a description*

$$\mathcal{S} = \bigcup_{j=1}^{B} \{(x_m, y_m)\}_{m\in I_j} = \bigcup_{j=1}^{B} S_j \tag{3.22}$$

*with $\bigcup_{j=1}^{B} I_j = \{1,2,...,N_S\}$ and $\bigcap_{j=1}^{B} I_j = \emptyset$ for $B \in \mathbb{N}$. Then $S_j = \{(x_m,y_m)\}_{m\in I_j}$ is called **training batch** and $|S_j|$ is the **batch-size** of $S_j$ for $j = \{1,2,...,B\}$.*

If it holds that the remainder of dividing $N_S$ by $B$ equals zero, denoted with $N_S(\mod B) = 0$, then one can choose batches of size $|I_j| = \frac{N_S}{B}$ for $j \in \{1,2,...,B\}$. Otherwise, the set can not be separated into batches of equal size and we need to choose $|I_j| = \lfloor \frac{N_S}{B-1} \rfloor$ for $j \in \{1,2,...,B-1\}$ and $|I_B| = N_S - (B-1)\lfloor \frac{N_S}{B-1} \rfloor$, such that the conditions of the disjoint representation of the training data are fulfilled.

**Example 4**
*Assume we have $N_S = 100$ training samples and we want to have $B = 7$ batches. Then $\lfloor \frac{N_S}{B-1} \rfloor = 16$ for batches $j \in \{1,2,...,6\}$ and $N_S - (B-1)\lfloor \frac{N_S}{B-1} \rfloor = 100 - 6\cdot 16 = 4$ elements for the last batch.*

**Definition 15** (Stochastic Gradient-Descent for Neural Networks [33])
*Given a batched training set $\mathcal{S} = \bigcup_{j=1}^{B} S_j$ and a loss-function $\mathscr{L} \in \mathscr{C}^1((\mathcal{Y}\times\mathcal{Y}),\mathbb{R})$, the*

***Stochastic Gradient-Descent optimization*** *scheme for training a DNN with initial value* $\theta^0 \in \Theta$ *is given by*

$$\theta^{k+1} = \theta^k - \eta_k \nabla_\theta \hat{\mathscr{R}}_{S_j}(f_{\theta^k}) = \theta^k - \frac{\eta_k}{|I_j|} \sum_{m \in I_j} \nabla_\theta \mathscr{L}(f_{\theta^k}(x_m), y_m) \tag{3.23}$$

*for a batch* $S_j$ *with* $j \in \{1,2,...,B\}$.

If we have two different disjoint representations, denoted by $\alpha$ and $\beta$, with $\mathcal{S} = \bigcup_{j=1}^{B} S_j^\alpha = \bigcup_{j=1}^{B} S_j^\beta$, where the resulting index set $I_j^\alpha$ and $I_j^\beta$ are not necessarily equal for $j \in \{1,2,...,B\}$, then one can use the batches $S_j^\alpha$ for $B$ optimization steps of SGD. Exactly these $B$ optimization steps of the same disjoint batch-representation of the training set are called a **training epoch**. If we use the batches $S_j^\beta$ for the next $B$ optimization steps, then this is another epoch of training.

**Definition 16** (Empirical Generalization Error [33])
*Assume that for the training of a Neural Network,* $\theta^N \in \Theta$ *is the value of the network parameters after* $N \in \mathbb{N}$ *optimization steps. Then for a training set* $\mathcal{S}$ *with size* $N_S \in \mathbb{N}$ *and a test set* $\mathcal{T}$ *with size* $N_T \in \mathbb{N}$*, the absolute value*

$$\varepsilon^N = \left| \hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^N}) - \hat{\mathscr{R}}_{\mathcal{T}}(f_{\theta^N}) \right| \tag{3.24}$$

*is called* ***generalization error*** *after* $N \in \mathbb{N}$ *optimization steps.*

As long as $0 \leq \varepsilon^N \leq \varepsilon_{max}$, where $\varepsilon_{max}$ is an acceptable error bound, we say that a Neural Network $f_{\theta^N}$ **generalizes** well on the test data. Otherwise the model **overfits**, for instance if $\varepsilon^N > \varepsilon_{max}$ and $\left|\hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^N})\right| < \left|\hat{\mathscr{R}}_{\mathcal{T}}(f_{\theta^N})\right|$.

Since one optimization step for SGD strongly depends upon the randomly chosen batch, it is possible that the empirical risk $\hat{\mathscr{R}}_{S_j}(f_{\theta^k})$ of a batch $S_j \in \mathcal{S}$ strongly differs from $\hat{\mathscr{R}}_{\mathcal{S}}(f_{\theta^k})$ for a step $k \in \mathbb{N}$. It can therefore also happen, since a batch $S_{j+1}$ is used to compute $\theta^{k+1}$, that $\hat{\mathscr{R}}_{S_j}(f_{\theta^k}) < \hat{\mathscr{R}}_{S_{j+1}}(f_{\theta^{k+1}})$, although a steady decrease of the empirical risk is desired. Therefore one can use momentum Gradient-Descent methods like ADAM [56] or similar, to average the gradient for several batches. Furthermore to overcome the problem of overfitting, several regularization methods for Neural Network training have been developed, like Batch-Normalization [92], Drop-Out [100], Data-Augmentation [80] or $L^2$-regularization of the network parameters [62], which need to be mentioned at this point, but is not explained in detail here. A more detailed overview about specific Neural Network architectures, with different types of layers, is shown in the now following section.

## 3.2 Neural Network Architectures

We have restricted Neural Networks to general combinations of non-linear layer functions in the previous section to clarify the main principles of Neural Network training. Of course, there is a fast growing amount of explicit Neural Network architectures in the modern Deep Learning community, trying to reach new benchmarks for individual tasks. One of the most prominent examples for image classification are for instance AlexNet [3, 58] or LeNet [65] for CNN and Long-Short-Term-Memory (LSTM) [45] for Recurrent Neural Networks (RNN) to solve time-series forecasting problems or language processing.

We clarify the layer structures of arbitrary Convolutional Neural Networks and Convolutional Auto-Encoders. As a specialized type, one can then also easily explain the U-Net architecture [89].

### 3.2.1 Layer Arithmetic

A Neural Network has previously been defined as a measurable function $f_\theta \in \mathscr{M}_\Theta(\mathcal{X},\mathcal{Y})$ for training and test samples $(x,y) \in \mathcal{S} \cup \mathcal{T}$. Furthermore, the input layer has been denoted by $a^{(0)} = x$, the output layer by $a^{(M)} = f_\theta(x)$ and an arbitrary hidden layer by $a^{(l)} = f^{(l)}_{\theta_l}(a^{(l-1)})$ for $l \in \{1,2,...,M-1\}$, where $M \in \mathbb{N}$ is the depth of the network . It is therefore obvious, that the structure of the layers strongly depends upon the dimension of $\mathcal{X}$ and $\mathcal{Y}$, if no pre-processing is done to change the dimension of the data.

**Fully-Connected Layers** [33]
Probably the most simple type of all layer structures is the Fully-Connected (FC) layer. Therefore, we assume that for an arbitrary DNN $f_\theta : \mathcal{X} \to \mathcal{Y}$ with depth $M \in \mathbb{N}$, we have one specific layer $l \in \{1,2,...,M-1\}$ as stated by Eq. (3.17),

$$a^{(l)} = f^{(l)}_{\theta_l}\left(a^{(l-1)}\right)$$

with $a^{(l-1)} \in \mathbb{R}^{d_{l-1}}$ and $a^{(l)} \in \mathbb{R}^{d_l}$ vector-valued or vectorized layer input and output with input dimension $d_{l-1}$ and $d_l \in \mathbb{N}$ for layer $(l-1)$ and layer $l$. Resuming that we have described the layers as non-linear functions, a Fully-Connected layer can explicitly be defined as

$$a^{(l)} = \sigma^{(l)}\left(W^{(l)}a^{(l-1)} + b^{(l)}\right), \tag{3.25}$$

where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is a **weighting matrix**, transforming the input from dimension $d_{l-1}$ to dimension $d_l$, the term $b^{(l)} \in \mathbb{R}^{d_l}$ is a **bias vector** for multi-dimensional affine transformation and $\sigma^{(l)} : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$ is a point-wise applied non-linear **activation function** for the elements of an input array.

We herein give four examples of activation functions, which are mainly used for the architecture of Neural Networks. Of course, one can define individual activation functions for different optimization problems.

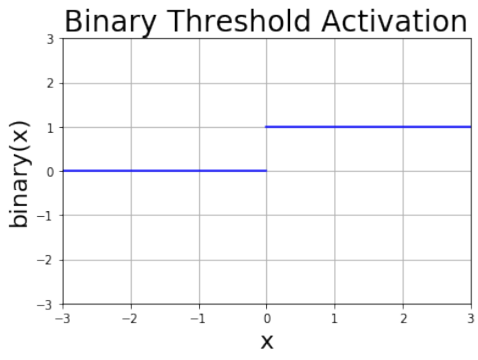
**Example 5**

*The first example is the **Binary Threshold function**.*

*As it can be seen from the figure and the function description, there are only two possible*

$$
\begin{aligned}
\sigma : \mathbb{R} &\to \{0,1\} \\
x &\mapsto \mathbb{1}_{\{x>0\}}(x)
\end{aligned}
$$

(a) Function description



(b) Plot

FIGURE 3.1: Binary Threshold Function

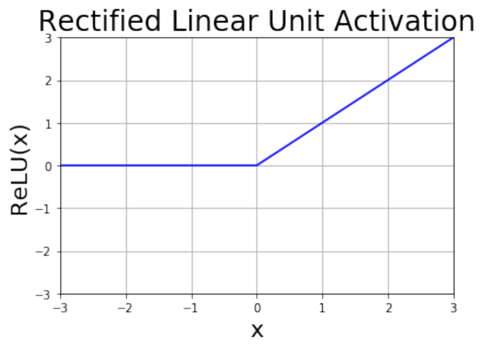
*values for the output of the Binary Threshold. Therefore it can for instance be used as an activation function for binary classification, where negative values result from a false classification and are not relevant for following layers, since the product of weighting matrix and the null-vector is also equal to zero. The activation of the next layer then only depends upon the bias vector.*

*Similar to the Binary Threshold, the **Rectified Linear Unit function** (ReLU) is used to activate positive values of the function input.*

*Since it can make a difference, if the input value is close to zero or not, the ReLU function*

$$
\begin{aligned}
\sigma : \mathbb{R} &\to [0,\infty) \\
x &\mapsto \max\{x,0\}
\end{aligned}
$$

(a) Function description



(b) Plot

FIGURE 3.2: ReLU Function

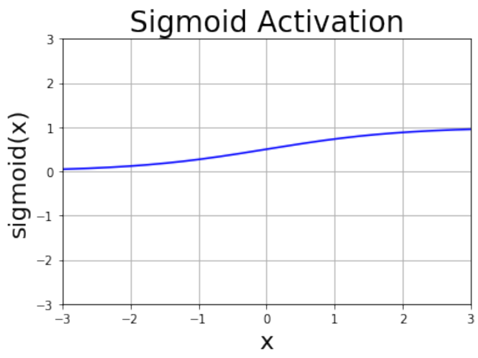
*is linear for positive values and zero for all negative values. Neural Networks with ReLU activations can therefore be used for (piecewise) linear function approximations.*

*Besides the introduced "one-sided" activation functions, there are also several symmetric activation functions, like the **Sigmoid activation function**, which also maps negative input values to a specific interval.*

*For $x = 0$ the sigmoid function has value $\sigma(0) = \frac{e^0}{1+e^0} = \frac{1}{2}$. For large negative values, it*

$$\begin{aligned}\sigma : \mathbb{R} &\to (0,1) \\ x &\mapsto \frac{e^x}{1+e^x}\end{aligned}$$

(a) Function description



(b) Plot

FIGURE 3.3: Sigmoid Function

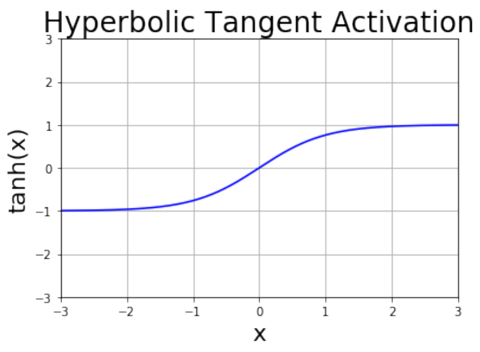
*converges to* 0*, for large positive values, it converges to* 1*. The activation function can therefore for instance be used, if we want to find discrete probability estimations, as it is the case for multi-classification problems. Then the output of a vector-valued input, gives insights about the discrete probability distribution for several classes.*

*The last activation function, we present, is the* ***Hyperbolic Tangent****. Similar to the sigmoid function, negative and positive input values are equally weighted, due to the symmetric shape of the function.*

*For* $x = 0$*, we get* $\tanh(x) = 0$*. For large negative values, the function converges to* $-1$*,*

$$\begin{aligned}\sigma : \mathbb{R} &\to (-1,1) \\ x &\mapsto \tanh(x)\end{aligned}$$

(a) Function description



(b) Plot

FIGURE 3.4: Hyperbolic Tangent Function

*for large positive values, it converges to* 1*. Therefore, the Hyperbolic Tangent function can be used, if we want to process sequential data, since in case for dynamical systems, negative displacements and positive displacements give both useful information and should therefore both be considered for further processing by the Neural Network.*

If we want to apply the BP algorithm to a Fully-Connected layer structure, we need to compute the partial derivatives $\frac{\partial a^{(l)}}{\partial a^{(l-1)}} = \frac{\partial \sigma^{(l)}\left(W^{(l-1)}a^{(l-1)}+b^{(l)}\right)}{\partial a^{(l-1)}}$, which is critical for the Binary Threshold function and the ReLU function for an argument close to zero. Contrarily, the derivatives are well defined for the Sigmoid activation function and the Hyperbolic Tangent.

**Definition 17** (Deep Fully-connected Neural Network)
*Assume $f_\theta \in \mathscr{M}_\Theta(\mathcal{X}, \mathcal{Y})$ to be a DNN with depth $M \in \mathbb{N}$. If*

$$a^{(l)} = \sigma^{(l)}\left(W^{(l)} a^{(l-1)} + b^{(l)}\right)$$

*for layers $l \in \{1, 2, .., M\}$ with input $a^{(0)} = x$, output $a^{(M)} = f_\theta(x)$, weighting matrices $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$, bias vectors $b^{(l)} \in \mathbb{R}^{d_l}$, point-wise activation functions $\sigma^{(l)} : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$, layer input dimension $d_l \in \mathbb{N}$ for $l \in \{0, 1, ..., M\}$ and $x \in \mathbb{R}^{d_0}$, $y \in \mathbb{R}^{d_M}$, then $f_\theta$ is a **Deep Fully-connected Neural Network**.*

Since $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{d_l}$, the generalized layer parameters can be expressed by the set

$$\theta^{(l)} = \left\{W_{ij}^{(l)} | i \in \{1, 2, ..., d_l\}, j \in \{1, 2, ..., d_{l-1}\}\right\} \cup \left\{b_i^{(l)} | i \in \{1, 2, .., d_l\}\right\}. \tag{3.26}$$

for $l \in \{1, 2, ..., M\}$. In summary, the generalized layer parameters contain all entries of the weighting-matrices and all elements of the bias-vector.

**Flattening and Reshaping** [33]
Fully-Connected Neural Networks are restricted to vector-valued input data. Nevertheless, a multi-dimensional array $x \in \mathbb{R}^{d_1 \times d_2 \times ... \times d_n}$ with $d_j \in \mathbb{N}$ for $j \in \{1, 2, ..., n\}$ can be processed via FC layers, using a **flattening** operation [33] $\nu : \mathbb{R}^{d_1 \times d_2 \times ... \times d_n} \to \mathbb{R}^{d_1 \cdot d_2 \cdot ... \cdot d_n}$. In addition, a **reshaping** operation $\rho : \mathbb{R}^{d_1 \cdot d_2 \cdot ... \cdot d_n} \to \mathbb{R}^{d_1 \times d_2 \times ... \times d_n}$ serves as inverse to the flattening operation, such that $\rho(\nu(x)) = x$.

Reshaping and flattening are necessary operations, if we want to define Neural Networks with a bottleneck structure, for instance in case of dimension reduction for arbitrary images. We will later on explain the importance of such operations, when talking about Auto-Encoders.

**Convolutional Layers** [33]
Although Fully-Connected Neural Networks could in principle be used to process any multi-dimensional input sample $x \in \mathcal{X}$, it can often be inefficient to use FC-layers, due to a resulting high number of network parameters causing overfitting. For Fully-Connected layers, each element of the input vector is separately weighted, ignoring possible dependencies within the elements of the inputs.

Fully-Connected structures often make sense, if the elements (features) of the data samples are independent on each other. In most cases, data is not structured by well-defined attributes. Neural Networks can be used to process data structures like images or sequential data. For images, there can be local dependencies, like the contours of an object or the color, while for sequential data, it is possible to process the dynamical structure for a specific time horizon.

In both cases, convolutional layers can be used, processing multi-dimensional arrays without having the need to use a flattening operation. The convolutional layers make use of efficient parameter sharing by filters (kernels) of weights. Detailed descriptions of convolutional layers can for instance be found in [33, 36, 114]. We give a short introduction, how convolutional operations can be defined, describing 1D, 2D and 3D convolution.

**1D Convolution** [114]

We herein focus on [114] to describe a one-dimensional convolution for vector-valued as well as for matrix-valued inputs. Furthermore, detailed examples for convolutional layers are given by [24]. Assume we have $a^{(l-1)} \in \mathbb{R}^{h_l \times w_l}$ as input to a layer $l \in \{1,2,...,M\}$ and a weighting kernel $\kappa \in \mathbb{R}^{h_l \times w_l'}$ with $M$, $h_l$, $w_l$, $w_l' \in \mathbb{N}$ . Then

$$a^{(l)} = \sigma^{(l)} \left( a^{(l-1)} * \kappa + b^{(l)} \right) \tag{3.27}$$

with

$$c_{j'}^{(l)} = \left( a^{(l-1)} * \kappa \right)_j^{(1D)} := \sum_{m=1}^{h_l} \sum_{n=1}^{w_l'} a_{m,j-n+1}^{(l-1)} \kappa_{mn}. \tag{3.28}$$

The dimension for $a^{(l-1)} * \kappa$, $b^{(l)}$ and $\sigma^{(l)}$ have not been specified yet, since they are equal and strongly depend upon the type of convolution, such that $a^{(l)} \in \mathbb{R}^{w_{l+1}}$ with

$$w_{l+1} = \left\lfloor \frac{w_l - w_{l'} + 2p}{s} \right\rfloor + 1. \tag{3.29}$$

It then holds that for the elements of the convolution $c_{j'}^{(l)}$, we have $j' \in \{1,2,...,w_{l+1}\}$ and $j \in \{0,1,...,w_l + w_l' - 1\}$, if we choose a stride of size 1. **Zero padding**, connected to the number $p \in \mathbb{N}$, is required, if we want to use $a_{i,j-n+1}^{(l-1)}$ with $j-n+1 \leq 0$ or $j-n+1 > w_l$. We then use $a_{i,j-n+1}^{(l-1)} = 0$ to apply the convolution anyway. We assume to then have zero-valued elements equally distributed at the left and right border of the array. The value $2p$ therefore describes the total number of artificially created zero elements for the one-dimensional convolution. Furthermore a different **striding scheme** with $s \in \mathbb{N}$ can be chosen to only compute a convolution for $j = s\imath + 1$ with $\imath \in \mathbb{N}$. Then, the index set for the input elements is reduced, such that $j \in \{0, s, 2s, ...w_l + w_l' - 1\}$. This can for instance be used to control the reduction of dimension for the output. Since further the height of kernel and input array are equal for the one-dimensional convolution, the output is vector-valued and therefore it holds that $b^{(l)}, c^{(l)} \in \mathbb{R}^{w_{l+1}}$ and $\sigma^{(l)} : \mathbb{R}^{w_{l+1}} \to \mathbb{R}^{w_{l+1}}$.

**2D Convolution and higher order** [114]

Comparable to the one-dimensional convolution, higher order convolutional layers can be defined for an input element $a^{(l-1)} \in \mathbb{R}^{h_l \times w_l}$ and a filter $\kappa \in \mathbb{R}^{h_{l'} \times w_{l'}}$, simply via

$$c_{i',j'}^{(l)} = \left( a^{(l-1)} * \kappa \right)_{ij}^{(2D)} := \sum_{m=1}^{h_{l'}} \sum_{n=1}^{w_{l'}} a_{i-m+1,j-n+1}^{(l-1)} \kappa_{mn} \tag{3.30}$$

with $i' \in \left\{1,2,...,\left\lfloor \frac{h_l - h_{l'} + 2p_q}{s_q} \right\rfloor + 1\right\}$, $j' \in \left\{1,2,...,\left\lfloor \frac{w_l - w_{l'} + 2p_r}{s_r} \right\rfloor + 1\right\}$ and $i \in \{0, s_q, 2s_q, ...h_l + h_l' - 1\}$, $j \in \{0, s_r, 2s_r, ...w_l + w_l' - 1\}$.

The above description of the two-dimensional convolution can simply be verified by applying a one-dimensional convolution along the rows and also along the columns of the layer input. Therefore, padding and stride scheme have to be defined for the row-convolution with $p_r \in \mathbb{N}$ and $s_r \in \mathbb{N}$ and the column-convolution with $p_q \in \mathbb{N}$ and $s_q \in \mathbb{N}$. The output $c^{(l)}$ is then a **feature map**. Since one single kernel may not be sufficient to extract all dependencies of the input, it is required to use $d_l \in \mathbb{N}$ kernels for convolutional layer $l$, producing multiple feature maps, such that $c^{(l)} \in \mathbb{R}^{h_{l+1} \times w_{l+1} \times d_l}$. In case that another convolutional layer follows layer $l$, it is possible to proceed in two ways: Assume there are another $d_{l+1}$ kernels for layer $l+1$, then a two-dimensional convolution could be applied to each input map, such that the output of the convolutional layer is given by $a^{(l+1)} \in \mathbb{R}^{h_{l+2} \times w_{l+2} \times d_l \cdot d_{l+1}}$. Another possibility, which is also used to control the dimensionality, is to compute the sum of the convolution for the feature maps, such that $a^{(l+1)} \in \mathbb{R}^{h_{l+2} \times w_{l+2} \times d_{l+1}}$.

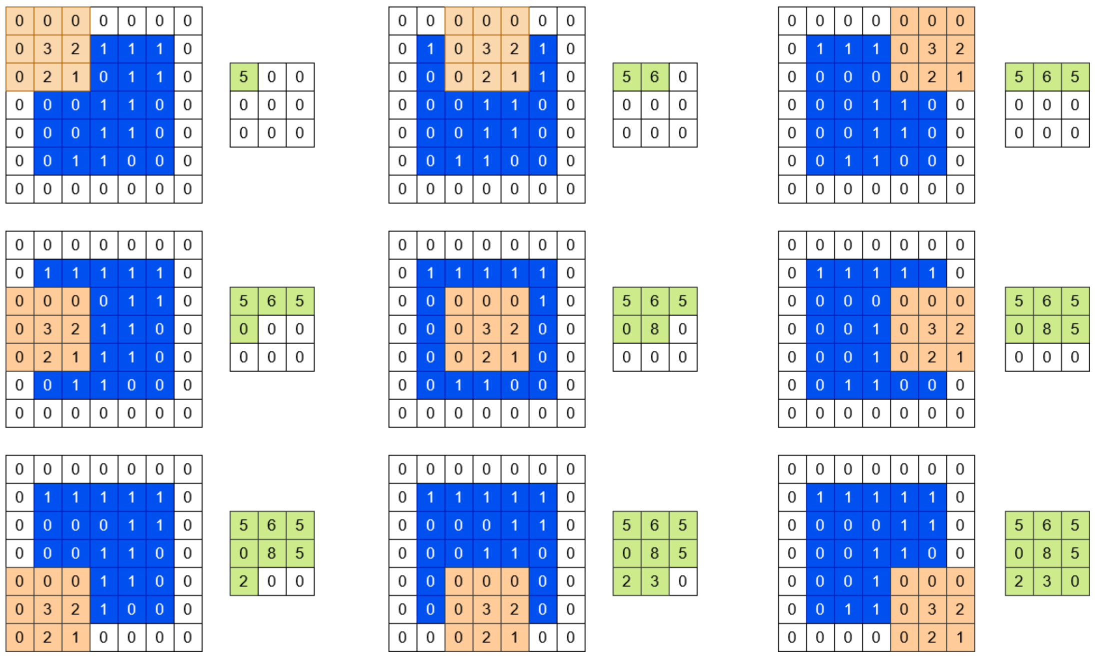


FIGURE 3.5: 2D Convolution of a $5 \times 5$ input map, using a $3 \times 3$ weighting kernel. The kernel is passed along the rows and the columns to compute the sum of the Hadamard product for each intersection. The computed values are stored in a $3 \times 3$ output map, which requires $3 \cdot 3 = 9$ computational steps.

We can see one example of a discrete convolution in Figure 3.5. Given a $3 \times 3$ weighting-kernel and a $5 \times 5$ input map, the convolution computes a feature map with $p_r = 1$, $p_q = 1$, $s_r = 2$ and $s_q = 2$, which can easily be verified using Eq. (3.29) for vertical (column) and horizontal (row) convolution.

We mainly focus on one-dimensional and two-dimensional convolutional layers. Nevertheless the above formula can be applied to 3D convolution or even higher dimensions. If we have input values $a^{(l)} \in \mathbb{R}^{h_l \times w_l \times d_l}$ and kernels $\kappa \in \mathbb{R}^{h_{l'} \times w_{l'} \times d_{l'}}$ such that we get

$$c^{(l)}_{i'j'k'} = \left(a^{(l-1)} * \kappa\right)^{(3D)}_{ijk} := \sum_{m=1}^{h_{l'}} \sum_{n=1}^{w_{l'}} \sum_{o=1}^{d_{l'}} a^{(l-1)}_{i-m+1,j-n+1,k-o+1} \kappa_{mno} \tag{3.31}$$

with $i' \in \left\{1,2,..., \left\lfloor \frac{h_l - h_{l'} + 2p_q}{s_q} \right\rfloor + 1\right\}$, $j' \in \left\{1,2,..., \left\lfloor \frac{w_l - w_{l'} + 2p_r}{s_r} \right\rfloor + 1\right\}$, $k' \in \left\{1,2,..., \left\lfloor \frac{d_l - d_{l'} + 2p_t}{s_t} \right\rfloor + 1\right\}$ and $i \in \{0, s_q, 2s_q, ...h_l + h'_l - 1\}$, $j \in \{0, s_r, 2s_r, ...w_l + w'_l - 1\}$, $k \in \{0, s_t, 2s_t, ...d_l + d'_l - 1\}$, similar to the two-dimensional scheme.

**Transposed Convolutional Layers** [114]

In general, convolutional layers are used to reduce the dimension of the input. Nevertheless, it can also be used to maximize the dimension via so called **transposed convolution**. For the sake of simplicity, we restrict to row-convolution, and as it has previously been shown, the dependence of the dimension of layer $l$ and layer $l+1$ is given by $w_{l+1} = \left\lfloor \frac{w_l + w_{l'} + 2p}{s} \right\rfloor + 1$, where $w_l$, $w_{l'}$, $p$, $s \in \mathbb{N}$ are pre-defined sizes. For a transposed convolution, the dimension formula also holds, but here, the target dimension $w_{l+1}$ and the input dimension $w_l$ are fixed. Therefore, one needs to find a kernel-size $w_{l'}$, a padding-scheme $p$ and a stride-size $s$ such that

$$\left\lfloor \frac{w_l - w_{l'} + 2p}{s} \right\rfloor + 1 \overset{!}{=} w_{l+1},$$

where $w_{l'}$, $p$ and $s$ are variable and thus there is no unique method for the construction of transposed convolutional operators.

**Example 6**

*Assume we have an output width of size $w_{l+1} = 4$, input width of size $w_l = 2$. Then with $w_{l'} = 3$, $p = 2$ and $s = 1$, it holds that $\left\lfloor \frac{2-3+2\cdot 2}{1} \right\rfloor + 1 = 4$. In a similar way, we can also find sizes to apply a transposed convolution to order 2 or higher, which is straight forward.*

**Pooling and Upsampling** [33]

Instead of using convolutional or transposed convolutional layers to adjust dimension, we can also use **pooling-layers** for dimension reduction or **upsampling-layers** for dimension extension. Assume therefore that we have a pooling layer $l \in \{1,2,...,M\}$, where $M \in \mathbb{N}$ is the depth of the network with input $a^{(l-1)} \in \mathbb{R}^{h_l \times w_l}$. We then want to map a sub-matrix of $0 < m \leq h_l$ rows and $0 < n \leq w_l$ columns to a real number.

**Example 7**

*Assume we have a matrix $a \in \mathbb{R}^{m \times n}$, we then define three types of pooling operations $p : \mathbb{R}^{m \times n} \to \mathbb{R}$ for the matrix a, which is a sub-matrix of the layer's input $a^{(l-1)} \in \mathbb{R}^{h_l \times w_l}$:*

- *Maximum-Pooling:* $p(a) = \max_{\substack{i \in \{1,2,...,m\} \\ j \in \{1,2,...,n\}}} a_{ij}$,
- *Average Pooling:* $p(a) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}$,

- *$L^p$-Pooling:* $p(a) = \left|\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^p\right|^{\frac{1}{p}}$ for $p \in \{1,2,...,\infty\}$.

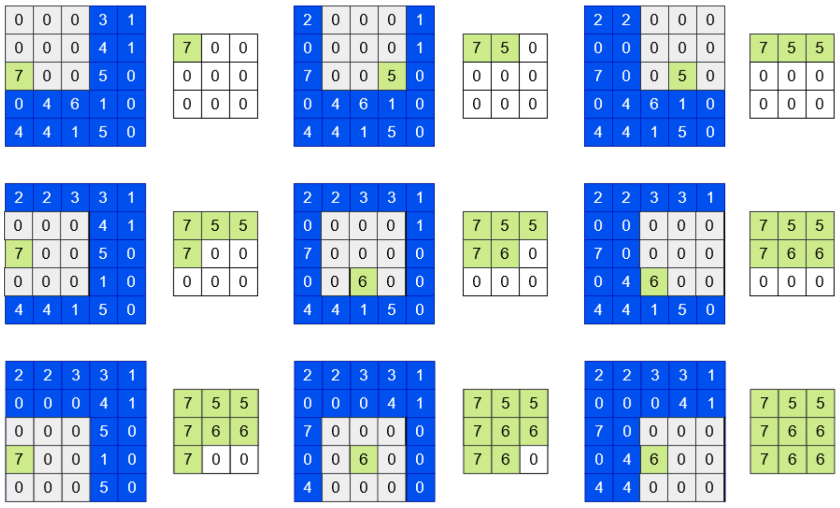


FIGURE 3.6: 2D Pooling of a $5 \times 5$ input map, using a $3 \times 3$ pooling kernel. The kernel is passed along the rows and columns of the input map. The shown operation is a Maximum-Pooling layer, where the maximum of the intersection is mapped to one element of the $3 \times 3$ output map. The pooling operation requires a total number of $3 \cdot 3 = 9$ computational steps.

We can see one example for Maximum-Pooling of a $5 \times 5$ input map with a $3 \times 3$ pooling kernel and strides $s = 1$ in Figure 3.6. The output is shown to be a $3 \times 3$ pooled feature map, where again the dimension formula of Eq. (3.29) can be used to verify the dimension reduction, since there are no padding elements for pooling here.

As an opposite operation to pooling-layers, one can also define upsampling-layers to artificially enlarge an input value $a^{(l-1)} \in \mathbb{R}^{h_l \times w_l}$. Assume we have an upsampling kernel of $m$ rows and $n$ columns.

**Example 8**
*Assume we have a scalar $a \in \mathbb{R}$ and an upsampling operator $u : \mathbb{R} \to \mathbb{R}^{m \times n}$. Then similar to a pooling layer, we give some examples about possible upsampling operators. Further let $i' \in \{1,2,...,m\}$ and $j' \in \{1,2,...,n\}$ be a fixed target index, for instance $(i',j') = (1,1)$. Two types of upsampling or* ***unpooling*** *are then given by*

- *Maximum-Unpooling:* $u(a)_{ij} = \begin{cases} a, & \text{if } \; i = i', j = j' \\ 0, & \text{else.} \end{cases}$,
- *Average-Unpooling:* $u(a)_{ij} = \frac{a}{mn}$ *for all* $i \in \{1,2,...,m\}$ *and* $j \in \{1,2,...,n\}$.

**Residual Blocks and Skip-Connections** [89]

One special structure of layer connections are so called **Residual blocks** as it is explained by [42] and essential to understand the concept of an U-Net architecture. Therefore we assume to have a DNN with depth $M \in \mathbb{N}$ such that $a^{(l)} \in \mathbb{R}^{h_l \times w_l \times d_l}$ and $a^{(l+\lambda)} \in \mathbb{R}^{h_l \times w_l \times d_l}$ and

$l+\lambda \leq M$. Again, there are different possibilities to define the residual blocks of a Neural Networks.

If we refer to [42], then for the above outputs of layers $l$ and $l+\lambda$, a residual block or skip-connection can be described as

$$r^{(l+\lambda)} = a^{(l)} + a^{(l+\lambda)} \in \mathbb{R}^{h_l \times w_l \times d_l}. \tag{3.32}$$

Anyway, we have a different definition for residual blocks in [10, 89] for U-Net, namely

$$r^{(l+\lambda)} = a^{(l)} \oplus a^{(l+\lambda)} := \left(a^{(l)}, a^{(l+\lambda)}\right) \in \mathbb{R}^{h_l \times w_l \times (2d_l)}, \tag{3.33}$$

which is in precise a concatenation for values of different dimension. Then of course, one can combine the concatenated value with a pooling or transposed convolution operator, to reduce the up-sized dimension of the residual block.

**Batch-Normalization** [33]
It has already been mentioned before that there are a lot of regularization methods to prevent DNN from overfitting. We herein shortly introduce Batch-Normalization [33, 92], which is one of the most helpful and simplest regularization methods and can easily be used as an additional layer of the Neural Network. Since usually, a whole batch can be passed simultaneously through the network, **Batch-Normalization** can be defined for an arbitrary batch $S_j = \{(x_m, y_m)\}_{m \in I_j} \subset \mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$ with $j \in \{1, 2, ..., B\}$.

Assume w.l.o.g. that $x_m \in \mathbb{R}^d$ with $d \in \mathbb{N}$ for all $m \in I_j$ with $j \in \{1, 2, ..., B\}$. The **point-wise batch-mean** $\mu(S_j) \in \mathbb{R}^d$ can then be defined as

$$\mu(S_j) := \frac{1}{|I_j|} \sum_{m \in I_j} x_m \tag{3.34}$$

and analogously, the **point-wise batch-variance** $\sigma^2(S_j)$ by

$$\sigma^2(S_j) = \frac{1}{|I_j|} \sum_{m \in I_j} (x_m - \mu(S_j))^2. \tag{3.35}$$

Then, for all $x_m$, one uses the **normalization**

$$\hat{x}_{m;k} = \frac{x_{m;k} - \mu(S_j)_k}{\sqrt{\sigma^2(S_j)_k + \varepsilon_k}} \tag{3.36}$$

for all elements $k \in \{1, 2, ..., d\}$ of a sample $x_m \in \mathbb{R}^d$, such that $x_{m;k} := (x_m)_k \in \mathbb{R}$ and an arbitrary large stabilization term $\varepsilon_k > 0$ is used to prevent division by zero. Then as a final step one uses the **parameter shift**

$$\bar{x}_{m;k} = \gamma_k \cdot \hat{x}_{m;k} + \beta_k, \tag{3.37}$$

where $\gamma, \beta \in \mathbb{R}^d$ are additional trainable parameters, such that

$$\theta^{(1)} = \left\{ W_{ij}^{(1)} | i \in \{1,2,...,d_1\}, j \in \{1,2,...,d\} \right\} \cup \left\{ b_i^{(1)} | i \in \{1,2,..,d_1\} \right\} \cup \{\gamma, \beta\}, \quad (3.38)$$

if we use Batch-Normalization before processing it by a first Fully-Connected layer. The input of the next layer is then given by

$$a^{(1)} = \sigma^{(1)} \left( W^{(1)} \bar{x}_m + b^{(l)} \right) \quad (3.39)$$

for elements $x_m \in S_j$.

It is obvious, that Batch-Normalization in Eq. (3.37) can preserve identity mapping with $\gamma_k = \sqrt{\sigma^2 (S_j)_k + \varepsilon_k}$ and $\beta_k = \mu (S_j)_k$ for all $k \in \{1,2,...,d\}$. Therefore, it is also possible to apply Eq. (3.36) to any layer with general input $a^{(l)}$ with $l \in \{1,2,...,M\}$.

We now shortly present some specific Neural Network architectures in the next sections, which are later on used to compute valuable results for the parameter estimation problem in Chapter 5.

### 3.2.2 Convolutional Neural Networks

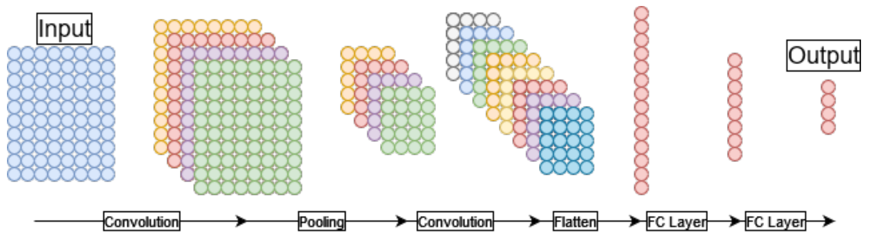


FIGURE 3.7: CNN architecture, combining convolutional layers with a Fully-Connected network. An input map of size $10 \times 8$ is processed via several convolutional, pooling and fully-connected layers to an output dimension of 4.

We consider a specific Convolutional Neural Network (CNN) architecture in Figure 3.7, similar to [10]. As one can see, the herein described architecture shows a Neural Network, consisting of several convolutional layers and finally connected to a Fully-Connected subnetwork. In this specific example, we have a matrix valued input map, which is processed via 4 weighting kernels to a feature map with depth 4. Then a pooling layer follows, halving the dimension along the rows and the columns. Afterwards another convolutional layer is chosen, with weighting-kernels, such that the depth of the output map is equal to 8. Afterwards, the output is flattened, connecting it to three Fully-Connected layers of different size. The output layer has dimension 4. Therefore, the herein shown CNN maps matrices of a specific size to a low-dimensional vector-valued output and could therefore for instance be used for classification, if the number of different classes for the input samples would be equal to 4, the dimension of the output layer.

### 3.2.3 Convolutional Auto-Encoders

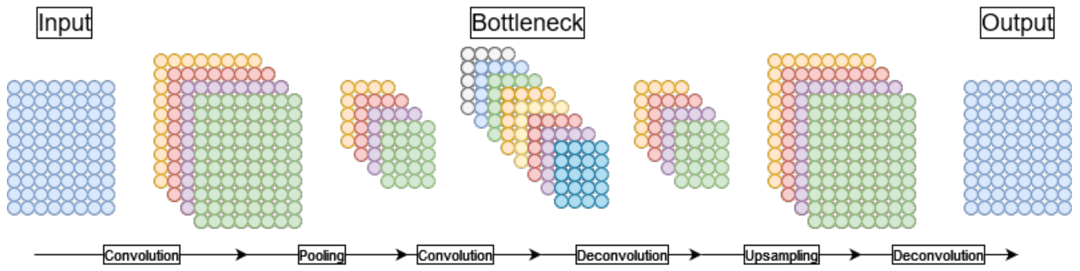


FIGURE 3.8: Convolutional Autoencoder architecture, which maps an input map of size $10 \times 8$ to an output map of equal size, using convolutional and transposed convolutional layers. The transposed operation is also called "Deconvolution" in some cases. For this specific architecture, the input map is processed, using a convolutional layer with 4 weighting kernels, followed by a pooling layers. Another convolutional layer with 2 weighting kernels for each feature map, then defines the bottleneck of the network architecture, given by 8 feature maps of size $5 \times 4$. This structure is then mapped back to the original space, using transposed layer operations.

Where CNN architectures as shown above can be used for feature extraction, another type of Neural Network architectures are (Convolutional) Auto-Encoders [63], as it is visualized in Figure 3.8. As far as Auto-Encoders are concerned, the main task is to reduce a high-dimensional input image to a latent space representation and re-transform it to the original space. In this context, one uses the terms **"Encoder"**, **"Bottleneck"** and **"Decoder"**. It can be simply explained that the Encoder maps the input-space to a lower dimensional latent space, which is also referred to as the "Bottleneck". Then, this latent space representation is projected back to the dimension of the input-data, using a "Decoder", which could in principle be described as the transposed operation of the Encoder. Therefore let us assume to have a Neural Network with input space $\mathcal{X} = \mathbb{R}^{h_0 \times w_0 \times d_0}$. Then the Auto-Encoder is a Neural Network, defined as a function

$$f_\theta : \mathbb{R}^{h_0 \times w_0 \times d_0} \to \mathbb{R}^{h_0 \times w_0 \times d_0}$$
$$x \mapsto f_\theta(x),$$

with layers

$$f^{(1)}_{\theta_1} : \mathbb{R}^{h_0 \times w_0 \times d_0} \to \mathbb{R}^{h_1 \times w_1 \times d_1},$$
$$f^{(2)}_{\theta_2} : \mathbb{R}^{h_1 \times w_1 \times d_1} \to \mathbb{R}^{h_2 \times w_2 \times d_2},$$
$$f^{(3)}_{\theta_3} : \mathbb{R}^{h_2 \times w_2 \times d_2} \to \mathbb{R}^{h_1 \times w_1 \times d_1},$$
$$f^{(4)}_{\theta_4} : \mathbb{R}^{h_1 \times w_1 \times d_1} \to \mathbb{R}^{h_0 \times w_0 \times d_0}.$$

It is then obvious, that the Encoder can be defined as $f^{(2)}_{\theta_2} \circ f^{(1)}_{\theta_1}$ and the decoder as $f^{(4)}_{\theta_4} \circ f^{(3)}_{\theta_3}$. It should then also be clear, that one can moreover define Fully-Connected Auto-Encoders, using vector-valued input and output, as well as vector-valued layers. Auto-Encoder structures can for instance be used for denoising tasks, or to find the lowest dimension of the latent space, such that the reconstructed image is close to the original input image.

### 3.2.4 U-Net

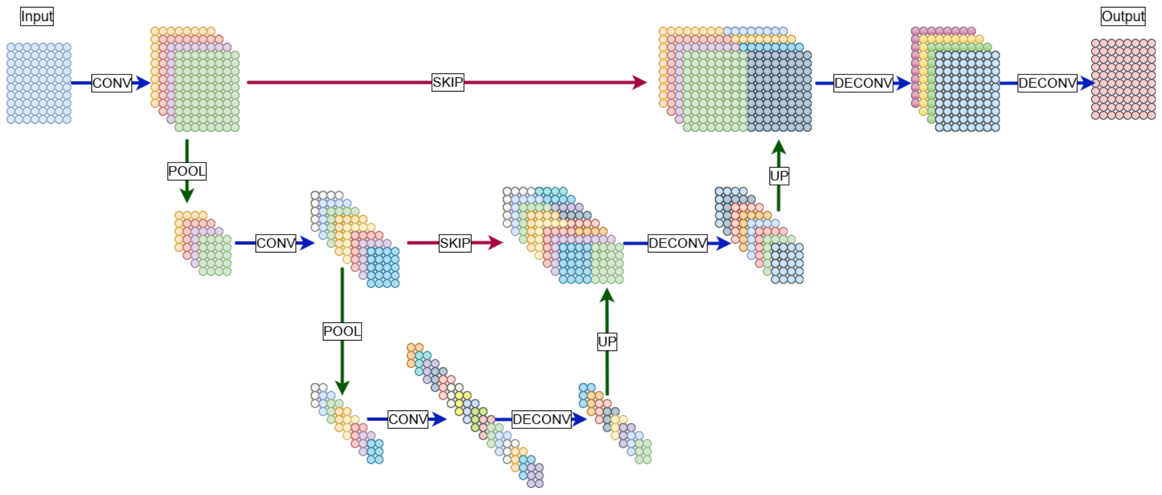


FIGURE 3.9: U-Net architecture, mapping a $10 \times 8$ input map to an output map of equal size. The structure can be described via two directions, where similar to a Convolutional Auto-Encoder, the general structure can be described by an Encoder, mapping the input to the bottleneck via convolutional and pooling layers. Contrarily, the Decoder uses transposed convolution and upsampling layers. The second direction is a horizontal alignment of the layers, combining elements of the Encoder and the Decoder, which have equal size via skip-connections.

The last structure we herein want to present is the U-Net by [89] and schematically shown in Figure 3.9. As one can imagine, the name "U-Net" is not an abbreviation for a Neural Network specific, technical term, but refers to the structure of the shown figure. In general, the U-Net can be described as a Convolutional Neural Network, serving as an Encoder structure, combined with an appropriate transposed network as Decoder structure. Furthermore, layers with equal dimension are coupled via skip-connections.

More in detail, we can recognize convolutional layers and pooling layers on the down-sizing part of the U-Net for an input map with size $10 \times 8$, similar to the CAE structure. For the bottleneck of the network, we have a total number of 16 feature maps with size $3 \times 2$. Then the first residual block adds the $3 \times 2$ feature maps to the reshaping layer via concatenation and processes the residual block via convolution. Further up-sizing operations can either be done via upsampling or transposed convolution. As for the down-sizing part, only pooling layers have been used for dimension reduction, it is close to use up-sampling layers as there are additional convolutional layers to process the residual blocks. The combination of up-sampling, concatenation and processing of the residual blocks is then done, until the output reaches the dimension of the original input space. The U-Net architecture has achieved excellent results in biomedical image processing and segmentation.

We have introduced the motivation of applying the training of Neural Networks in the sense of SLT and in precise introduced several possible layer structures, which will be used for the parameter estimation problem including Neural Networks. The herein described network structures give sufficient insights, how to combine several layers to define complex Neural Network architectures. It could also be mentioned here, that regularization methods, like the presented Batch-Normalization, can easily be combined as a pre-processing layer for the presented architectures. Technically, Neural Networks are combinations of more or less simple, non-linear sub-functions, namely the layer functions.

# Chapter 4

# Ordinary Differential Equations

A mathematical description of approximate vehicle models, for instance the Quarter-Car-Model, has already been introduced in Section 2. It has also been shown that the acceleration of the components, namely wheel-suspensions, chassis and seat, including the occupant, can be expressed as a system of second order ordinary differential equations, depending on displacement and velocity and the corresponding parameters of mass, spring-constant and damping-constant. Analogously, this system can be redefined as a system of first order ordinary differential equations by doubling the dimension of the system and by defining a system matrix, which contains smaller square block-matrices. A general description of this linear system equation has been shown in the same section.

In case an exterior force acting on the dynamical system is considered, which results from scaling the road-profile displacements with the mass-spring-coefficient of the wheel-suspensions, it is possible to explicitly solve the system of differential equations by computing the homogeneous and non-homogeneous solutions. An explicit solution of the displacements and velocities makes it then possible to even compute the acceleration of the vehicle's components, which can then serve as simulated data of a g-sensor for further theoretical investigations.

The differential equations of the Quarter-Car-Model can therefore either be explicitly solved by computing the solution of each second order ordinary differential equation individually or by solving the linear first order system in parallel. We show, that even the computation of the homogeneous explicit solution can be time-consuming. Thus it is necessary to define efficient discrete methods to get sufficiently good approximations of the exact solution via numerical schemes.

We start this chapter with a short introduction to general second order ordinary differential equations for homogeneous as well as for non-homogeneous equations. The next section then is an analogous description of finding the solution of general first order systems, resulting from a reduction of the second order equations. It should then be clarified, which method is preferable in terms of computational efficiency and approximation quality. Therefore, we compare different numerical methods and introduce structure-preserving algorithms as for instance the Symplectic Euler scheme, which is one efficient method to find stable approximations of the coupled states within an appropriate computational time. The numerical methods can then be used, as mentioned at the beginning of this section, to generate synthetic data of arbitrary size, where a variation of the road-profile and the system parameters, following a pre-defined distribution, can be used to develop appropriate training and test datasets for Neural Network investigations. The generation of such data is the aim of Chapter 4.

## 4.1 Second Order Ordinary Differential Equations

Acceleration of a particle in a mechanical system can formally be described using a second order ordinary differential equation, following Newton's laws of motion. Especially for the coupled dynamical system of the Quarter-Car-Model, a system of second order ODE can be used to formally express the interconnected relation of the acceleration profiles in terms of states and velocities following the characteristics of the equation. We are therefore going to give insights about how the general solution of a single second order ODE can be computed. Further, we give a short introduction about the theory of solving non-homogeneous differential equations and show the complexity of even simple non-homogeneous equations. Therefore, we evaluate the approximation quality of straight forward numerical approximation schemes.

### 4.1.1 Homogeneous Equations

We clarify the most relevant theoretical aspects in solving second order ordinary differential equations. For further reading, we refer for instance to [13, 104]. The now introduced theorems and definitions are mainly based upon [76], but can also be found in the previously mentioned sources. We introduce the section with the following general definition.

**Definition 18** (Second Order Ordinary Differential Equation [76])
*Let $t \in [t_0, t_n] \subset \mathbb{R}$ be a time variable in a finite time horizon and let further $y \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ be a twice continuously differentiable function. A **second order linear differential equation** for $y$ is then defined by*

$$\ddot{y}(t) + a_1(t)\dot{y}(t) + a_0(t)y(t) = b(t), \tag{4.1}$$

*where $\ddot{y}(t) = \frac{\mathrm{d}^2 y(t)}{\mathrm{d}t^2}$ and $\dot{y}(t) = \frac{\mathrm{d}y(t)}{\mathrm{d}t}$ are the first and second order total derivatives with respect to time and $a_1 : [t_0, t_n] \to \mathbb{R}$, $a_2 : [t_0, t_n] \to \mathbb{R}$, $b : [t_0, t_n] \to \mathbb{R}$ are arbitrary continuous functions.*

*Then, the differential equation is **homogeneous**, if and only if $b(t) = 0$ for all $t \in [t_0, t_n]$. Further, if $a_1$ and $a_2$ are constant on the finite time horizon, then the equation is said to be of **constant coefficients**, otherwise it has **variable, time-dependent coefficients**.*
*For simplicity reasons, we may use the notation*

$$\ddot{y} + a_1(t)\dot{y} + a_0(t)y = b(t) \tag{4.2}$$

*instead of Eq. (4.1). Nevertheless, it should be clear that all components depend upon the time variable $t \in [t_0, t_n]$.*

Two models of dynamical systems have already been shown in Chapter 2. We can now show and define them as second order differential equations.

**Example 9** (Mathematical Pendulum)
*The mathematical pendulum, as given by Eq. (2.51) can be described by the simplified equation*

$$I_\alpha \ddot{\alpha} + mgl\alpha = 0,$$

*where $I_\alpha = ml^2$ is the mass moment of inertia with the mass $m > 0$, the length of the pendulum $l > 0$, the gravitational constant $g \approx 9.81 \frac{m}{s^2}$ and the function of angular displacement $\alpha \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ as a twice continuously differentiable function and therefore $\ddot{\alpha}(t) = \frac{\mathrm{d}^2\alpha(t)}{\mathrm{d}t^2}$ is the angular acceleration.*

*Since $I_\alpha > 0$, it holds that the pendulum equation is equal to*

$$\begin{aligned} & \ddot{\alpha} + \frac{mgl\alpha}{I_\alpha} = 0 \\ \Leftrightarrow \quad & \ddot{\alpha} + \frac{mgl\alpha}{ml^2} = 0 \\ \Leftrightarrow \quad & \ddot{\alpha} + \frac{g}{l}\alpha = 0. \end{aligned}$$

*The mathematical pendulum can therefore be defined as a second order linear differential equation with $a_1(t) = 0$, $a_2(t) = \frac{g}{l}$ and $b(t) = 0$ for all $t \in [t_0, t_n]$ and is therefore a **homogeneous equation with constant coefficients**.*

**Example 10** (Mass-Spring-Damper)
*A simplified Mass-Spring-Damper model is given via the equation*

$$m\ddot{x} + d\dot{x} + kx = f(t)$$

*for $t \in [t_0, t_n]$ with the mass-parameter $m > 0$, the damping constant $d \geq 0$, the spring-constant $k \geq 0$ and the non-linear function $f \in \mathscr{C}^0([t_0, t_n], \mathbb{R})$. Further, the displacement is given by the twice continuously differentiable function $x \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$. Similar to the mathematical pendulum, following the assumption of a non-zero mass parameter $m > 0$, the above equation is equal to*

$$\ddot{x} + \frac{d}{m}\dot{x} + \frac{k}{m}x = \frac{1}{m}f(t).$$

*Therefore, following the general definition of a second order ODE, we have $a_1(t) = \frac{d}{m}$, $a_0(t) = \frac{k}{m}$ and $b(t) = \frac{1}{m}f(t)$ for $t \in [t_0, t_n]$. Thus it is a **non-homogeneous differential equation with constant coefficients**.*

The now introduced concepts for uniqueness and existence of the general solution of a second order ordinary differential equation should be in principle well-known and can be found for instance in [13, 76]. We therefore shortly clarify the theoretical framework to show that the dynamical systems do have unique solutions. Most of the theorems are shown without proof, but can be found in [76], as for instance the following.

**Theorem 4** (Unique Solution of Initial Value Problems [76])
*Assume $a_1$, $a_0$, $b \in \mathscr{C}^0([t_0,t_n],\mathbb{R})$ and $y \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$. Let further $y_0 := y(t_0)$ and $v_0 := \dot{y}(t_0)$ be arbitrary constants. The **initial value problem** (IVP) for a second order ordinary differential equation, given by*

$$\ddot{y} + a_1(t)\dot{y} + a_0(t)y = b(t) \tag{4.3}$$

$$y(t_0) = y^0, \quad \dot{y}(t_0) = v^0 \tag{4.4}$$

*has a **unique solution** on $[t_0,t_n]$.*

*Proof.* A detailed description to prove the uniqueness of the IVP, can also be found in [104], applying the theorem of Picard-Lindelöf to the second order problem. □

In order to find the general solution of a homogeneous second order ordinary differential equation, one is probably already familiar with the concepts of linear operators. Nevertheless, we quickly introduce them for consistence of formalism. Again, the following set of theorems and definitions are mainly based on [13, 76].

**Theorem 5** (Linearity of the Homogeneous Equation [76])
*Let us define an operator of the second order equation by*

$$L(y) = \ddot{y} + a_1(t)\dot{y} + a_0(t)y \tag{4.5}$$

*with $a_1$, $a_0 \in \mathscr{C}^0([t_0,t_n],\mathbb{R})$ and $y \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$. Then $L(y)$ is a **linear operator**.*

*Proof.* Since for constants $c_1$, $c_2 \in \mathbb{R}$ and a pair of functions $y_1$, $y_2 \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$, it holds for a linear operator that $L(c_1y_1 + c_2y_2) = c_1L(y_1) + c_2L(y_2)$, we make use of Eq. (4.5) to proof the condition. It is obvious that with $y = c_1y_1 + c_2y_2$, we get

$$\begin{aligned} L(c_1y_1 + c_2y_2) &= (c_1\ddot{y}_1 + c_2\ddot{y}_2) + a_1(t)(c_1\dot{y}_1 + c_2\dot{y}_2) + a_0(t)(c_1y_1 + c_2y_2) \\ &= c_1(\ddot{y}_1 + a_1(t)\dot{y}_1 + a_0(t)y_1) + c_2(\ddot{y}_2 + a_1(t)\dot{y}_2 + a_0(t)y_2) \\ &= c_1L(y_1) + c_2L(y_2), \end{aligned}$$

for $t \in [t_0,t_n]$, which completes the proof. □

**Theorem 6** (General Homogeneous Solution [76])
*Assume $y_1$, $y_2 \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ to be linearly independent, satisfying $L(y_i) = \ddot{y}_i + a_1(t)\dot{y}_i + a_0(t)y_i = 0$ for $i \in \{1,2\}$ with $a_1$, $a_0 \in \mathscr{C}^0([t_0,t_n],\mathbb{R})$. Then the **general solution** $y \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$, which solves $L(y) = 0$ is given by*

$$y(t) = c_1y_1(t) + c_2y_2(t) \tag{4.6}$$

*for $t \in [t_0,t_n]$ and $c_1$, $c_2 \in \mathbb{R}$ are arbitrary constants.*

**Definition 19** (Characteristic Polynomial [76])
*Given the homogeneous second order ordinary differential equation for a function $y \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ with constant coefficients $a_1 := a_1(t) \in \mathbb{R}$, $a_0 := a_0(t) \in \mathbb{R}$ for all $t \in [t_0, t_n]$ by*

$$\ddot{y} + a_1 \dot{y} + a_0 y = 0,$$

*the **characteristic polynomial** is defined by*

$$\rho(r) := r^2 + a_1 r + a_0 \tag{4.7}$$

*with $r \in \mathbb{C}$.*

**Theorem 7** (General Homogeneous Solution with Constant Coefficients [76])
*Let $p(r) = r^2 + a_1 r + a_0$ be the characteristic polynomial with $r \in \mathbb{C}$ for the second order homogeneous equation $\ddot{y} + a_1 \dot{y} + a_0 y = 0$ with $a_1$, $a_0 \in \mathbb{R}$ and $y \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$.*
*If $r_1$, $r_2 \in \mathbb{C}$ are the roots of $p(r)$ with $r_1 \neq r_2$ and $c_1$, $c_2 \in \mathbb{R}$ are arbitrary constants, then the **general homogeneous solution** is given by*

$$y_h(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t} \tag{4.8}$$

*for $t \in [t_0, t_n] \subset \mathbb{R}$. If in contrast $r_1 = r_2$, then the homogeneous solution is given by*

$$y_h(t) = c_1 e^{r_1 t} + c_2 t e^{r_1 t} \tag{4.9}$$

*for $t \in [t_0, t_n] \subset \mathbb{R}$.*

Due to Theorem 4, given initial conditions $y(t_0) = y_0$ and $\dot{y}(t_0) = v_0$, a unique solution exists. Therefore, the coefficients $c_1$ and $c_2$ can uniquely be determined to satisfy the IVP. We again now show an application to the above concepts, analysing the homogeneous solutions for the mathematical pendulum and the mass-spring damper model.

**Example 11**
*We again regard the mathematical pendulum as earlier in this section described by the simplified homogeneous equation*

$$\ddot{\alpha} + \frac{g}{l}\alpha = 0,$$

*where $\alpha \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ is the angular displacement function, $g \approx 9.81 \frac{m}{s^2}$ is the gravitational constant and $l > 0$ is the pendulum length. The characteristic polynomial is then given by*

$$\rho(r) = r^2 + \frac{g}{l}.$$

*Hence it follows that the general solution is given by*

$$\alpha(t) = c_1 e^{i\sqrt{\frac{g}{l}}t} + c_2 e^{-i\sqrt{\frac{g}{l}}t} \tag{4.10}$$

*with $i = \sqrt{-1} \in \mathbb{C}$ and it can be derived that the velocity can be expressed as*

$$\dot{\alpha}(t) = c_1 i\sqrt{\frac{g}{l}}e^{i\sqrt{\frac{g}{l}}t} - ic_2\sqrt{\frac{g}{l}}e^{-i\sqrt{\frac{g}{l}}t} \tag{4.11}$$

*as well as the acceleration by*

$$\ddot{\alpha}(t) = -c_1\frac{g}{l}e^{i\sqrt{\frac{g}{l}}t} - c_2\frac{g}{l}e^{-i\sqrt{\frac{g}{l}}t} = -\frac{g}{l}\alpha(t). \tag{4.12}$$

*Consequently, Eq.* (4.10) *satisfies the second order equation.*
*Furthermore, if any unique initial conditions $\alpha(t_0) = y_0$ and $\alpha(t_0) = v_0$ are given and if $t_0 = 0$, then*

$$\begin{aligned} &\alpha(t_0) = c_1e^0 + c_2e^0 = y_0 \\ \Leftrightarrow\quad &c_1 = y_0 - c_2 \end{aligned}$$

*and*

$$\begin{aligned} &\dot{\alpha}(t_0) = c_1 i\sqrt{\frac{g}{l}}e^0 - c_2 i\sqrt{\frac{g}{l}}e^0 = v_0 \\ \Leftrightarrow\quad &c_2 = \frac{1}{2}\left(y_0 - \frac{v_0\sqrt{l}}{i\sqrt{g}}\right), \end{aligned}$$

*thus $c_1 = \frac{1}{2}\left(y_0 + \frac{v_0\sqrt{l}}{i\sqrt{g}}\right)$. The unique solution of the IVP, satisfying the second order homogeneous equation as well as the initial condition is therefore given by*

$$\begin{aligned} \alpha(t) &= \frac{1}{2}\left(y_0 + \frac{v_0\sqrt{l}}{i\sqrt{g}}\right)e^{i\sqrt{\frac{g}{l}}t} + \frac{1}{2}\left(y_0 - \frac{v_0\sqrt{l}}{i\sqrt{g}}\right)e^{-i\sqrt{\frac{g}{l}}t} && (4.13) \\ &= y_0\cos\left(\sqrt{\frac{g}{l}}t\right) + v_0\sqrt{\frac{l}{g}}\sin\left(\sqrt{\frac{g}{l}}t\right). && (4.14) \end{aligned}$$

**Example 12**
*We now consider a homogeneous mass-spring-damper system, following the second order equation*

$$\ddot{x} + \frac{d}{m}\dot{x} + \frac{k}{m}x = 0 \tag{4.15}$$

*for $x \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ the displacement function, $d \geq 0$ the damping-constant, $k \geq 0$ the spring-constant and $m > 0$ the mass of the model. In a similar way, compared to the pendulum model, one can find the roots of the characteristic polynomial*

$$\rho(r) = r^2 + \frac{d}{m}r + \frac{k}{m},$$

*which gives the general homogeneous solution*

$$x(t) = c_1 e^{\frac{1}{2m}\left(-d+\sqrt{d^2-4km}\right)t} + c_2 e^{\frac{1}{2m}\left(-d-\sqrt{d^2-4km}\right)t}. \tag{4.16}$$

*For the sake of simplicity, we make use of the general form*

$$x(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t} \tag{4.17}$$

*to show, if the second order equation is satisfied. The first and second order derivatives of the general solution are thus given by*

$$\dot{x}(t) = c_1 r_1 e^{r_1 t} + c_2 r_2 e^{r_2 t} \tag{4.18}$$

*and*

$$\ddot{x}(t) = c_1 r_1^2 e^{r_1 t} + c_2 r_2^2 e^{r_2 t}. \tag{4.19}$$

*Therefore, it follows that*

$$\begin{aligned}
& \ddot{x}(t) + \frac{d}{m}\dot{x}(t) + \frac{k}{m}x(t) \\
= \quad & c_1 r_1^2 e^{r_1 t} + c_2 r_2^2 e^{r_2 t} + \frac{d}{m}\left(c_1 r_1 e^{r_1 t} + c_2 r_2 e^{r_2 t}\right) + \frac{k}{m}\left(c_1 e^{r_1 t} + c_2 e^{r_2 t}\right) \\
= \quad & c_1 e^{r_1 t}\left(r_1^2 + \frac{d}{m}r_1 + \frac{k}{m}\right) + c_2 e^{r_2 t}\left(r_2^2 + \frac{d}{m}r_2 + \frac{k}{m}\right) \\
= \quad & c_1 e^{r_1 t}\rho(r_1) + c_2 e^{r_2 t}\rho(r_2) = 0.
\end{aligned}$$

*We now assume, that initial conditions are given by $x(t_0) = x(0) = y_0$ and $\dot{x}(t_0) = \dot{x}(0) = v_0$. It is then clear to see that*

$$\begin{aligned}
& x(0) = c_1 + c_2 = y_0 \\
\Leftrightarrow \quad & c_1 = y_0 - c_2
\end{aligned}$$

*and*

$$\begin{aligned}
& \dot{x}(0) = c_1 r_1 + c_2 r_2 = v_0 \\
\Leftrightarrow \quad & c_2 = \frac{v_0 - r_1 y_0}{r_2 - r_1},
\end{aligned}$$

*which finally gives*

$$c_1 = -\frac{v_0 - r_2 y_0}{r_2 - r_1}.$$

*Thus the exact homogeneous solution is given by*

$$x(t) = \left(-\frac{v_0 - r_2 y_0}{r_2 - r_1}\right) e^{r_1 t} + \left(\frac{v_0 - r_1 y_0}{r_2 - r_1}\right) e^{r_2 t}, \tag{4.20}$$

*holding in mind that $r_1 = \frac{d}{2m}\left(-d + \sqrt{d^2 - 4km}\right)$ and $r_2 = \frac{d}{2m}\left(-d - \sqrt{d^2 - 4km}\right)$. One can recognize that even the homogeneous solution of a simple second order ordinary differential equation requires exact computation of a relatively complex term.*

We have shown a compact introduction in solving general homogeneous second order ordinary differential equations. Since the Quarter-Car-Model is a system of coupled non-homogeneous second order equations, the general solution requires the computation of a particular solution in addition to the general homogeneous solution. The complexity in solving this task becomes clear, if we consider the following section.

### 4.1.2 Non-homogeneous Equations

Similar to the previous section, we now introduce the most relevant theoretical concepts in solving non-homogeneous second order equations. We therefore show, how to compute the particular solution of a second order equation, given the general homogeneous solution with the help of the Wronskian [13]. The definitions and theorems presented, are again strongly based on [76], but can also be found in [13, 104]. Detailed proofs can also be found in the references, but are mostly skipped at this point.

**Theorem 8** (General Solution of Non-Homogeneous Second Order Equations [76])
*Given a linear operator $L(y) = \ddot{y} + a_1\dot{y} + a_0y$, of the homogeneous second order ordinary differential equation for a finite time horizon $[t_0,t_n] \subset \mathbb{R}$ with constant coefficients $a_1$, $a_0 \in \mathbb{R}$, and a two-times continuously differentiable function $y \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}\right)$, a non-homogeneous equation can be defined by*

$$L(y) = b(t) \tag{4.21}$$

*for a continuous function $b \in \mathscr{C}^0\left([t_0,t_n],\mathbb{R}\right)$. Then every **solution of the non-homogeneous ordinary differential equation** follows*

$$\begin{aligned} y(t) &= c_1y_1(t) + c_2y_2(t) + y_p(t) \\ &= y_h(t) + y_p(t), \end{aligned} \tag{4.22}$$

*for $t \in [t_0,t_n]$ with constants $c_1$, $c_2 \in \mathbb{R}$, which is the combination of the homogeneous solutions $y_1$, $y_2 \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}\right)$ with $L(y_1) = 0$ and $L(y_2) = 0$ and where $y_p \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}\right)$ is one particular solution, satisfying $L(y_p) = b(t)$ for all $t \in [t_0,t_n]$.*

A particular solution for simple equations can in some cases be found by guessing, but is non-trivial for more complex functions $b$. Nevertheless, there is a method to explicitly find a particular solution, using the Wronskian of the linearly independent homogeneous solution of the equation.

**Definition 20** (Wronskian [76])
*Given two differentiable functions $y_1$ and $y_2$ on $[t_0,t_n]$, the **Wronskian** is defined by*

$$W(t) = y_1(t)\dot{y}_2(t) - \dot{y}_1(t)y_2(t) \tag{4.23}$$

*for $t \in [t_0,t_n]$.*

**Theorem 9** (Variation of Parameters [76])
*For the non-homogeneous second order ordinary differential equation*

$$L(y) = \ddot{y} + a_1\dot{y} + a_0 y = b(t),$$

*let $a_1,\ a_0 \in \mathbb{R}$ be constants, $b \in \mathscr{C}^0([t_0,t_n],\mathbb{R})$ and $y \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$. Then a particular solution to $L(y) = b(t)$ is given by the **variation of parameters** formula*

$$y_p(t) = u_1(t)y_1(t) + u_2(t)y_2(t) \tag{4.24}$$

*for $t \in [t_0,t_n]$ with $y_1$ and $y_2$ general homogeneous solutions, satisfying $L(y_1) = 0$ and $L(y_2) = 0$ and*

$$u_1(t) = \int -\frac{y_2(t)b(t)}{W(t)}\mathrm{d}t, \quad u_2(t) = \int \frac{y_1(t)b(t)}{W(t)}\mathrm{d}t, \tag{4.25}$$

*where $W(t) = y_1(t)\dot{y}_2(t) - \dot{y}_1(t)y_2(t)$ is the Wronskian, such that $W(t) \neq 0$ for $t \in [t_0,t_n]$, since the general homogeneous solutions are linearly independent.*

**Example 13**
*We show that solving a "simple" non-homogeneous second order differential equation with the help of Theorem 9 is straight forward, but it is not obvious at a first glance, that the computed particular solution indeed solves the second order equation. Let us therefore assume to have a single Mass-Spring-Damper system, driven by a general sine-wave function, with arbitrary amplitude, frequency and phase. The equation is thus given by*

$$\ddot{x} + \frac{d}{m}\dot{x} + \frac{k}{m}x = a\sin(\omega t - \varphi), \tag{4.26}$$

*with $x \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ describing the state function, $d \geq 0$ is the damping constant, $k \geq 0$ the spring constant, $m > 0$ the mass of the object, $a \in \mathbb{R}$ is the amplitude of the sine-wave, $\omega = 2\pi f \in \mathbb{R}$ is the angular frequency, where $f$ is the ordinary frequency of the wave and $\varphi \in [0,2\pi)$ is the phase shift. All of the herein introduced parameters are assumed to be constant for $t \in [t_0,t_n]$.*

*It should be clear that the general homogeneous solutions are given by the twice continuously differentiable functions $x_1,\ x_2 \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$ with $x_1(t) = e^{r_1 t}$ and $x_2(t) = e^{r_2 t}$ for $t \in [t_0,t_n]$, where $r_1,\ r_2 \in \mathbb{C}$ are the roots of the characteristic polynomial $\rho(r) = r^2 + \frac{d}{m}r + \frac{k}{m}$. Furthermore, it follows that the Wronskian can explicitly be written as*

$$\begin{aligned} W(t) &= x_1(t)\dot{x}_2(t) - \dot{x}_1(t)x_2(t) \\ &= e^{r_1 t} r_2 e^{r_2 t} - r_1 e^{r_1 t} e^{r_2 t} \\ &= (r_2 - r_1)e^{(r_1+r_2)t}. \end{aligned}$$

*It is obvious that*

$$u_1(t) = \int -\frac{e^{r_2 t} a\sin(\omega t - \varphi)}{(r_2 - r_1)e^{(r_1+r_2)t}}\mathrm{d}t = -\frac{a}{r_2 - r_1}\int e^{-r_1 t}\sin(\omega t - \varphi)\mathrm{d}t \tag{4.27}$$

*and it therefore also holds that*

$$u_2(t) = \int \frac{e^{r_1 t} a \sin(\omega t - \varphi)}{(r_2 - r_1) e^{(r_1+r_2)t}} \mathrm{d}t = \frac{a}{r_2 - r_1} \int e^{-r_2 t} \sin(\omega t - \varphi) \mathrm{d}t. \tag{4.28}$$

*Let us therefore use a general variable $r \in \{r_1, r_2\}$ to compute the indefinite integral for both terms. We can verify that*

$$\begin{aligned}\int e^{-rt} \sin(\omega t - \varphi) \mathrm{d}t &= -e^{-rt} \cos(\omega t - \varphi) \frac{1}{\omega} - \int r e^{-rt} \cos(\omega t - \varphi) \frac{1}{\omega} \mathrm{d}t \\ &= -\frac{1}{\omega} e^{-rt} \cos(\omega t - \varphi) - \frac{r}{\omega} \int e^{-rt} \cos(\omega t - \varphi) \mathrm{d}t \\ &= -\frac{1}{\omega} e^{-rt} \cos(\omega t - \varphi) - \frac{r}{\omega^2} e^{-rt} \sin(\omega t - \varphi) - \frac{r^2}{\omega^2} \int e^{-rt} \sin(\omega t - \varphi) \mathrm{d}t.\end{aligned}$$

*Hence, it follows with*

$$1 + \frac{r^2}{\omega^2} = \frac{\omega^2 + r^2}{\omega^2}$$

*that we finally get*

$$\int e^{-rt} \sin(\omega t - \varphi) \mathrm{d}t = -\frac{\omega^2}{\omega^2 + r^2} \left( \frac{1}{\omega} e^{-rt} \cos(\omega t - \varphi) + \frac{r}{\omega^2} e^{-rt} \sin(\omega t - \varphi) \right).$$

*Then the particular solution of the non-homogeneous equation $x_p \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ is given, using*

$$u_1(t) = \frac{a}{r_2 - r_1} \frac{\omega^2}{\omega^2 + r_1^2} \left( \frac{1}{\omega} e^{-r_1 t} \cos(\omega t - \varphi) + \frac{r_1}{\omega^2} e^{-r_1 t} \sin(\omega t - \varphi) \right)$$

*and*

$$u_2(t) = -\frac{a}{r_2 - r_1} \frac{\omega^2}{\omega^2 + r_2^2} \left( \frac{1}{\omega} e^{-r_2 t} \cos(\omega t - \varphi) + \frac{r_2}{\omega^2} e^{-r_2 t} \sin(\omega t - \varphi) \right),$$

*by*

$$\begin{aligned}x_p(t) = \frac{a}{r_2 - r_1} \Bigg[ &\frac{\omega}{\omega^2 + r_1^2} \left( \cos(\omega t - \varphi) + \frac{r_1}{\omega} \sin(\omega t - \varphi) \right) \\ &- \frac{\omega}{\omega^2 + r_2^2} \left( \cos(\omega t - \varphi) + \frac{r_2}{\omega} \sin(\omega t - \varphi) \right) \Bigg].\end{aligned}$$

*As a next step, we need to verify, if the computed particular solution indeed solves the differential equation*

$$\ddot{x}_p + \frac{d}{m} \dot{x}_p + \frac{k}{m} x_p = a \sin(\omega t - \varphi) \tag{4.29}$$

*for $t \in [t_0, t_n]$. Therefore, one needs to compute the first and second order derivatives. Thus we get by differentiating that*

$$\begin{aligned}\dot{x}_p(t) = \frac{a}{r_2 - r_1} \Bigg[ &\frac{\omega}{\omega^2 + r_1^2} \left( -\sin(\omega t - \varphi) \omega + r_1 \cos(\omega t - \varphi) \right) \\ &- \frac{\omega}{\omega^2 + r_2^2} \left( -\sin(\omega t - \varphi) \omega + r_2 \cos(\omega t - \varphi) \right) \Bigg]\end{aligned}$$

*and therefore the second order derivative yields*

$$\ddot{x}_p(t) = \frac{a}{r_2 - r_1}\left[\frac{\omega}{\omega^2 + r_1^2}\left(-\cos(\omega t - \varphi)\omega^2 - r_1\omega\sin(\omega t - \varphi)\right)\right.$$
$$\left. - \frac{\omega}{\omega^2 + r_2^2}\left(-\cos(\omega t - \varphi)\omega^2 - r_2\omega\sin(\omega t - \varphi)\right)\right].$$

*Sorting by terms of* $a\sin(\omega t - \varphi)$ *and* $a\cos(\omega t - \varphi)$*, one can recognize that the l.h.s. of Eq.* (4.29) *equals*

$$a\sin(\omega t - \varphi)\underbrace{\left[\frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(-r_1\omega - \frac{d}{m}\omega + \frac{k}{m}\frac{r_1}{\omega}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(-r_2\omega - \frac{d}{m}\omega + \frac{k}{m}\frac{r_2}{\omega}\right)\right)\right]}_{=:\alpha}$$
$$+a\cos(\omega t - \varphi)\underbrace{\left[\frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(-\omega^2 + \frac{d}{m}r_1 + \frac{k}{m}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(-\omega^2 + \frac{d}{m}r_2 + \frac{k}{m}\right)\right)\right]}_{=:\beta}.$$

*Consequently, if* $\alpha = 1$ *and* $\beta = 0$*, then the second order equation holds for the particular solution. In order to show that* $\beta = 0$*, we need to remind that due to* $\rho(r) = 0$ *for* $r \in \{r_1, r_2\}$*, it holds that* $r\frac{d}{m} + \frac{k}{m} = -r^2$*. We get*

$$\begin{aligned}\beta &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(-\omega^2 + \frac{d}{m}r_1 + \frac{k}{m}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(-\omega^2 + \frac{d}{m}r_2 + \frac{k}{m}\right)\right)\\ &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}(-\omega^2 - r_1^2) - \frac{\omega}{\omega^2 + r_2^2}(-\omega^2 - r_2^2)\right)\\ &= \frac{1}{r_2 - r_1}(-\omega + \omega) = 0.\end{aligned}$$

*We need to use the relation* $-\frac{d}{m} = r + \frac{1}{r}\frac{k}{m}$ *for the roots of the characteristic polynomial to prove* $\alpha = 1$*. Computing the product of the roots* $r_1$ *and* $r_2$ *moreover yields that*

$$r_1 r_2 = \left(-\frac{d}{2m} + \sqrt{\left(\frac{d}{2m}\right)^2 - \frac{k}{m}}\right)\left(-\frac{d}{2m} - \sqrt{\left(\frac{d}{2m}\right)^2 - \frac{k}{m}}\right) = \frac{k}{m}.$$

*Then it is obvious that*

$$\begin{aligned}\alpha &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(-r_1\omega - \frac{d}{m}\omega + \frac{k}{m}\frac{r_1}{\omega}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(-r_2\omega - \frac{d}{m}\omega + \frac{k}{m}\frac{r_2}{\omega}\right)\right)\\ &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(-r_1\omega + \left(r_1 + \frac{1}{r_1}\frac{k}{m}\right)\omega + \frac{k}{m}\frac{r_1}{\omega}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(-r_2\omega + \left(r_2 + \frac{1}{r_2}\frac{k}{m}\right)\omega + \frac{k}{m}\frac{r_2}{\omega}\right)\right)\\ &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(\frac{k}{m}\frac{\omega}{r_1} + \frac{k}{m}\frac{r_1}{\omega}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(\frac{k}{m}\frac{\omega}{r_2} + \frac{k}{m}\frac{r_2}{\omega}\right)\right)\\ &= \frac{1}{r_2 - r_1}\left(\frac{\omega}{\omega^2 + r_1^2}\left(\frac{k}{m}\frac{r_1^2 + \omega^2}{\omega r_1}\right) - \frac{\omega}{\omega^2 + r_2^2}\left(\frac{k}{m}\frac{r_2^2 + \omega^2}{\omega r_2}\right)\right)\\ &= \frac{1}{r_2 - r_1}\left(\omega\frac{k}{m}\frac{1}{\omega r_1} - \omega\frac{k}{m}\frac{1}{\omega r_2}\right)\end{aligned}$$

$$= \frac{1}{r_2 - r_1}\left(r_1 r_2 \frac{r_2 - r_1}{r_1 r_2}\right) = 1.$$

*We have therefore shown that the l.h.s of Eq. (4.29) is equal to* $a\sin(\omega t - \varphi)$. *Hence,* $x_p$ *is a particular solution to the non-homogeneous second order equation.*

The above example shows that even the computation of a simple non-homogeneous equation, like the Mass-Spring-Damper system, can require high effort and cannot easily be verified. Since the sequential data in this section is discrete, we therefore want to focus now on efficient numerical methods, either explicit, implicit or semi-implicit, to find an appropriate approximation to the true continuous solution at each discrete point.

### 4.1.3 Hamiltonian Systems and Geometric Integration

We have in general introduced the main concepts to non-homogeneous second order ordinary differential equations in the previous section. Since we are considering dynamical systems, as introduced in Chapter 2, the states and the derivatives of the system follow physical laws. Therefore, we shortly introduce the connection of Lagrangian and Hamiltonian equations of motion [39, 40, 73, 102]. The general idea is based on Hamiltonian conservative systems. We make use of such conservative systems, though we are in general considering non-conservative systems, which also contain the Rayleigh-Dissipation for the damping and the exterior driving term of the system, simulated by the road-profile. Nevertheless, the Hamiltonian approach can be used to later on derive efficient numerical approximation schemes in the next section.

To introduce the Hamiltonian of a conservative system, we shortly recapture simplified definitions of the energy terms, as they have been defined in Chapter 2. Therefore, we make again use of a system of $d \in \mathbb{N}$ particles, where we have the generalized coordinates $q_j := q_j(t)$ with $j \in \{1, 2, ..., d\}$ and the derivatives $\dot{q}_j = \frac{\mathrm{d}q_j}{\mathrm{d}t}$ for $t \in [t_0, t_n]$.

The **kinetic energy** then depends on time $t \in [t_0, t_n]$ and velocity $\dot{q} = (\dot{q}_1, \dot{q}_2, ..., \dot{q}_d)^T \in \mathbb{R}^d$, such that it can be expressed as a differentiable function $T(t, \dot{q})$ with $T \in \mathscr{C}^1\left(\left([t_0, t_n] \times \mathbb{R}^d\right), \mathbb{R}\right)$.

Let us further define a **potential energy** term with $V(t, q)$, where $q = (q_1, q_2, ..., q_d) \in \mathbb{R}^d$ is the vector of generalized coordinates, such that $V \in \mathscr{C}^1\left(\left([t_0, t_n] \times \mathbb{R}^d\right), \mathbb{R}\right)$. The potential energy in the conservative case only depends upon the generalized coordinates and time, and is therefore independent on the generalized velocities.

The **Lagrangian** $L \in \mathscr{C}^1\left(\left([t_0, t_n] \times \mathbb{R}^d \times \mathbb{R}^d\right), \mathbb{R}\right)$ is then defined as the difference between kinetic and potential energy and can be expressed as

$$L(t, q, \dot{q}) := T(t, \dot{q}) - V(t, q).$$

The Euler-Lagrange equations of motion are then given for the conservative system by

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} = 0$$

for $j \in \{1,2,...,d\}$.

**Definition 21** (Conjugate Momenta and Hamiltonian [40])
*Let $L \in \mathscr{C}^1\left(\left([t_0,t_n]\times\mathbb{R}^d\times\mathbb{R}^d\right),\mathbb{R}\right)$ be the Lagrangian with $L(t,q,\dot{q}) = T(t,\dot{q}) - V(t,q)$ on $t \in [t_0,t_n]$, with $q \in \mathbb{R}^d$ the vector of generalized coordinates and $\dot{q} \in \mathbb{R}^d$ the corresponding vector of generalized velocities, where $d \in \mathbb{N}$ and $T(t,\dot{q}) \in \mathscr{C}^1\left(\left([t_0,t_n]\times\mathbb{R}^d\right),\mathbb{R}\right)$ is the kinetic energy as well as $V(t,q) \in \mathscr{C}^1\left(\left([t_0,t_n]\times\mathbb{R}^d\right),\mathbb{R}\right)$ is the potential energy for a point $t \in [t_0,t_n]$. The* ***conjugate momenta*** *are then defined by*

$$p_j := \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} \tag{4.30}$$

*for $j \in \{1,2,...,d\}$, where it is again point-wise defined by $p_j := p_j(t)$ for all $t \in [t_0,t_n]$. The vector of all conjugate momenta can therefore be expressed as $p = (p_1,p_2,...,p_d)^T \in \mathbb{R}^d$. Then, the* ***Hamiltonian*** *$H \in \mathscr{C}^2\left(\left([t_0,t_n]\times\mathbb{R}^d\times\mathbb{R}^d\right),\mathbb{R}\right)$ for conjugate momenta and generalized states $(p,q)$ is defined as*

$$H(t,p,q) := p^T\dot{q} - L(t,q,\dot{q}), \tag{4.31}$$

*where $\dot{q} = \dot{q}(t,p,q)$ is assumed to explicitly depend upon time, conjugate momenta and generalized coordinates.*

It is straight forward to verify that the Hamiltonian describes the **total energy** of the system. If there are mass parameters $m_j \geq 0$ for each particle $j \in \{1,2,...,d\}$, with $d \in \mathbb{N}$, then the kinetic energy can explicitly be written in the form

$$T(t,\dot{q}) = \frac{1}{2}\sum_{j=1}^{d} m_j\dot{q}_j^2.$$

Since

$$p_j = \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} = \frac{\partial(T(t,\dot{q})-V(t,q))}{\partial \dot{q}_j} = \frac{\partial T(t,\dot{q})}{\partial \dot{q}_j} = m_j\dot{q}_j,$$

we can verify that the Hamiltonian indeed is equal to

$$\begin{aligned}
H(t,p,q) &= p^T\dot{q} - L(t,q,\dot{q}) \\
&= \sum_{j=1}^{d} p_j\dot{q}_j - L(t,q,\dot{q}) \\
&= 2\cdot\frac{1}{2}\sum_{j=1}^{d} m_j\dot{q}_j^2 - T(t,\dot{q}) + V(t,q) \\
&= 2T(t,\dot{q}) - T(t,\dot{q}) + V(t,q)
\end{aligned}$$

$$= T(t,\dot{q}) + V(t,q),$$

which is the **total energy** of the conservative system. Since we have now clarified the most important terms for the Hamiltonian, we can now use and proof the following theorem [39] to connect the Lagrangian to the Hamiltonian formalism.

**Theorem 10** (Hamiltonian Equations of Motion [40])
*Let the Lagrangian be given by* $L \in \mathscr{C}^1\left(\left([t_0,t_n] \times \mathbb{R}^d \times \mathbb{R}^d\right), \mathbb{R}\right)$ *with* $L(t,q,\dot{q}) = T(t,\dot{q}) - V(t,q)$ *for* $t \in [t_0,t_n]$*,* $q \in \mathbb{R}^d$ *the vector of generalized states and* $\dot{q} \in \mathbb{R}^d$ *the vector of generalized velocities, such that* $\dot{q}_j = \frac{\mathrm{d}q_j}{\mathrm{d}t}$ *for* $j \in \{1,2,...,d\}$*. Further let the conjugate momenta* $p \in \mathbb{R}^d$ *be point-wise defined by* $p_j = \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j}$ *with* $j \in \{1,2,...,d\}$*. The Hamiltonian* $H \in \mathscr{C}^2\left(\left([t_0,t_n] \times \mathbb{R}^d \times \mathbb{R}^d\right), \mathbb{R}\right)$ *is thus given by* $H(t,p,q) = p^T\dot{q} - L(t,q,\dot{q})$ *with* $\dot{q} = \dot{q}(t,p,q)$*.*

*Then the Euler-Lagrange equations*

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} = 0$$

*for* $j \in \{1,2,...,d\}$ *are equivalent to the* ***Hamiltonian equations of motion****, given by*

$$\dot{p}_j = -\frac{\partial H(t,p,q)}{\partial q_j}, \quad \dot{q}_j = \frac{\partial H(t,p,q)}{\partial p_j} \tag{4.32}$$

*for all* $j \in \{1,2,...,d\}$*.*

*Proof.* Since the Hamiltonian is equal to

$$H(t,p,q) = \sum_{j=1}^{d} p_j\dot{q}_j - L(t,q,\dot{q}),$$

it holds that

$$\frac{\partial H(t,p,q)}{\partial p_j} = \dot{q}_j + p_j\frac{\partial \dot{q}_j}{\partial p_j} - \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j}\frac{\partial \dot{q}_j}{\partial p_j} = \dot{q}_j, \tag{4.33}$$

by definition of the conjugate momenta. Further, we have

$$\frac{\partial H(t,p,q)}{\partial q_j} = p_j\frac{\partial \dot{q}_j}{\partial q_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} - \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j}\frac{\partial \dot{q}_j}{\partial q_j} = -\frac{\partial L(t,q,\dot{q})}{\partial q_j}. \tag{4.34}$$

Then, since

$$\dot{p}_j = -\frac{\partial H(t,p,q)}{\partial q_j} = \frac{\partial L(t,q,\dot{q})}{\partial q_j} \tag{4.35}$$

and due to

$$p_j = \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j}, \tag{4.36}$$

it finally holds that

$$\dot{p}_j = -\frac{\partial H(t,p,q)}{\partial q_j}$$

$$\begin{aligned}
&\Leftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t} p_j = \frac{\partial L(t,q,\dot{q})}{\partial q_j} \\
&\Leftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} - \frac{\partial L(t,q,\dot{q})}{\partial q_j} = 0
\end{aligned}$$

for all $j \in \{1,2,...,d\}$, which completes the proof. □

For the sake of simplicity, let us assume that the equations of motion can be defined in a more compact sense [14], using the gradient of the Hamiltonian with respect to the vectors of conjugate momenta and generalized coordinates to get

$$\dot{p} := -\nabla_q H(t,p,q) = \left(-\frac{\partial H(t,p,q)}{\partial q_1}, -\frac{\partial H(t,p,q)}{\partial q_2}, ..., -\frac{\partial H(t,p,q)}{\partial q_d}\right)^T \in \mathbb{R}^d, \tag{4.37}$$

$$\dot{q} := \nabla_p H(t,p,q) = \left(\frac{\partial H(t,p,q)}{\partial p_1}, \frac{\partial H(t,p,q)}{\partial p_2}, ..., \frac{\partial H(t,p,q)}{\partial p_d}\right)^T \in \mathbb{R}^d. \tag{4.38}$$

The above gradient-based notation of the Hamiltonian equations of motion, is further used in the following section, to derive efficient numerical integration schemes for dynamical systems, in comparison to state-of-the-art methods. Therefore, one uses the equations of motion and directly applies them to the examples given in the next section. It is then obvious, that the conjugate momenta simplify to more specified terms of the dynamical system.

### 4.1.4 Numerical Solution

We have described the basic concepts to explicitly solve non-homogeneous second order ordinary differential equations in the previous section. In addition, we have investigated that explicitly solving Mass-Spring-Damper systems as non-homogeneous equations is in principle possible, nevertheless it requires a relatively high computational effort to find the general solution. The aim of this section is therefore to introduce numerical methods to approximate the solution of non-homogeneous second order ordinary differential equations.

Since there exist efficient and well-known methods to solve first order equations, we make use of the following theorem, introduced and proved for instance in [76].

**Theorem 11** (Reduction of Order [76])
*Assume a non-homogeneous second order ordinary differential equation is given by*

$$L(y) = \ddot{y} + a_1\dot{y} + a_0 y = b(t)$$

*with constants $a_1$, $a_0 \in \mathbb{R}$, a non-linear continuous function $b \in \mathscr{C}^0([t_0,t_n],\mathbb{R})$ and a twice continuously differentiable state function $y \in \mathscr{C}^2([t_0,t_n],\mathbb{R})$. Then $y$ is a solution to the non-homogeneous equation $L(y) = b(t)$ for all $t \in [t_0,t_n]$, if and only if, $x_1 = y$ and $x_2 = \dot{y}$ are* *solutions to the* ***system of first order ordinary differential equations***

$$\dot{x}_1 = x_2 \tag{4.39}$$

$$\dot{x}_2 = -a_0 x_1 - a_1 x_2 + b(t), \tag{4.40}$$

*for all* $t \in [t_0, t_n]$ *with constants* $a_1$, $a_0 \in \mathbb{R}$ *and* $b \in \mathscr{C}^0\left([t_0, t_n], \mathbb{R}\right)$.

The system of first order ODE can be expressed w.l.o.g. as

$$\dot{x}_1 = f(t, x_1, x_2) \tag{4.41}$$

$$\dot{x}_2 = g(t, x_1, x_2) \tag{4.42}$$

with $f(t, x_1, x_2) = x_2$ and $g(t, x_1, x_2) = -a_0 x_1 - a_1 x_2 + b(t)$. We can therefore now use an appropriate time discretization of the continuous finite time horizon to then apply numerical solution methods for the first order system

**Definition 22** (Time-Discretization)
*Given a continuous time horizon* $\mathscr{T} = [t_0, t_n]$, *a* ***discretization*** *of* $\mathscr{T}$ *is given by the set*

$$\mathscr{D}(\mathscr{T}) = \left\{t^0, t^1, ..., t^N\right\}, \tag{4.43}$$

*containing* $N+1$ *discrete time points* $t^i \in \mathscr{T}$ *for* $i \in \{0, 1, ..., N\}$ *with* $N \in \mathbb{N}$. *If* $t^{i+1} - t^i = h$ *is constant for all* $i \in \{0, 1, ..., N-1\}$, *then* $\mathscr{D}_h(\mathscr{T}) := \mathscr{D}(\mathscr{T})$ *is called an* ***equidistant discretization*** *of the time horizon* $\mathscr{T}$.

We in general regard numerical integrators for reduced order systems, based on the concepts introduced in [14, 17]. Therefore, we assume that one is familiar with the concepts of one-step methods, to solve a system of ordinary differential equations, as they are given by Eq. (4.41) and Eq. (4.42).

## Euler Schemes

We herein introduce numerical integrators [14, 17] for the following general initial value problem of a first order ordinary differential system

$$\dot{x}(t) = \Phi(t, x(t)), \quad x(t_0) = x^0 \tag{4.44}$$

where the time-variable is given by $t \in [t_0, t_n]$, the state function by $x \in \mathscr{C}^1\left([t_0, t_n], \mathbb{R}^d\right)$ and $\Phi \in \mathscr{C}^0\left(\left([t_0, t_n], \mathbb{R}^d\right), \mathbb{R}^d\right)$ is a Lipschitz-continuous function with dimension $d \in \mathbb{N}$. The initial condition is given by the constant $x^0 \in \mathbb{R}^d$.

Given an equidistant time-horizon $\mathscr{D}_h(\mathscr{T}) = \{t^0, t^1, ..., t^N\}$ with $N \in \mathbb{N}$ and $h = t^{i+1} - t^i > 0$ for all $i \in \{0, 1, ..., N-1\}$, such that $t^0 = t_0$, a **forward Euler scheme** [17] is defined by the equation

$$u^{k+1} = u^k + h\Phi(t^k, u^k), \tag{4.45}$$

where a **backward Euler scheme** [17] is in contrast defined by the equation

$$u^{k+1} = u^k + h\Phi(t^{k+1}, u^{k+1}). \tag{4.46}$$

We use the terms $u^k$ with $u^0 = x^0$ and $k \in \{1,2,...,N\}$ to ensure to differ the approximated values $u^k$ and the true values $x^k = x(t^k)$ for all $t^k \in \mathscr{D}_h(\mathscr{T})$. One therefore is able to compare pairs $(u^k, x^k) \in \mathbb{R}^{d\times 2}$ for all $k \in \{1,2,...,N\}$, such that the **approximation error**[14] can be defined by

$$e_h^k = \|x^{k+1} - u^k - h\Phi(t^k, u^k)\| \tag{4.47}$$

for the forward Euler scheme and in an analogous way also for the backward Euler scheme.

## Semi-Implicit Euler Scheme for Conservative Systems

We have already introduced the general Hamiltonian equations of motion Eq. (4.32) in the previous section, as it can be found in [39, 40, 73, 102]. Furthermore one can also use these equations to derive **sympletic** or **semi-implicit** numerical integration schemes for dynamical systems [14, 39, 40].

Since the Hamiltonian has been shown to equal the total energy of a dynamical system, we now assume that for a conservative system of dimension $d \in \mathbb{N}$, the kinetic energy is defined by

$$T(t,\dot{q}) = \frac{1}{2}\sum_{i=1}^{d} m_i \dot{q_i}^2, \tag{4.48}$$

where $t \in [t_0, t_n]$, $m_i > 0$ is the mass of a particle indexed by $i \in \{1,2,...,d\}$ and $\dot{q} \in \mathbb{R}^d$ is the vector of generalized velocities. Further let the potential energy be defined by

$$V(t,q) = \frac{1}{2}\sum_{i=1}^{d} k_i q_i^2, \tag{4.49}$$

where $t \in [t_0, t_n]$, $k_i \geq 0$ is a characteristic parameter, for instance a spring-constant and $q \in \mathbb{R}^d$ is the vector of generalized coordinates of the system. Let us further hold in mind that the Lagrangian of the system is defined by the difference of kinetic and potential energy, $L(t,q,\dot{q}) = T(t,\dot{q}) - V(t,q)$, where the Hamiltonian of the system is the total energy, $H(t,p,q) = T(t,\dot{q}) + V(t,q)$.

Since Eq. (4.37) and Eq. (4.38) hold for the conjugate momenta and the generalized velocities of a system with $d \in \mathbb{N}$, namely

$$\dot{p} = -\nabla_q H(t,p,q) \in \mathbb{R}^d,$$
$$\dot{q} = \nabla_p H(t,p,q) \in \mathbb{R}^d,$$

a **Symplectic Euler scheme** [14, 39, 40] can be defined by

$$p^{k+1} = p^k - h\nabla_q H(t^k, p^{k+1}, q^k) \tag{4.50}$$
$$q^{k+1} = q^k + h\nabla_p H(t^k, p^{k+1}, q^k) \tag{4.51}$$

or

$$p^{k+1} = p^k - h\nabla_q H(t^k, p^k, q^{k+1}) \tag{4.52}$$

$$q^{k+1} = q^k + h\nabla_p H(t^k, p^k, q^{k+1}), \tag{4.53}$$

for an equidistant discrete time-horizon $\mathscr{D}_h(\mathscr{T}) = \{t^0, t^1, ..., t^N\}$, such that $h = t^{k+1} - t^k$ constant for all $k \in \{0, 1, ..., N-1\}$, $N \in \mathbb{N}$ and initial values $p^0 \in \mathbb{R}^d$ and $q^0 \in \mathbb{R}^d$. The Symplectic Euler method is therefore also called **Semi-Implicit Euler**, since it is an explicit method for one of the two arguments and an implicit method for the other one.

Following Eq. (4.48) and Eq. (4.49), one can recognize that the components of $\nabla_q H(t, p, q)$, namely the derivatives of the conjugate momenta, can be computed by

$$\dot{p}_j = -\frac{\partial H(t,p,q)}{\partial q_j} = \frac{\partial L(t,q,\dot{q})}{\partial q_j} = -\frac{\partial V(t,q)}{\partial q_j} = -k_j q_j \tag{4.54}$$

for all $j \in \{1, 2, ..., d\}$. Furthermore, it holds that the conjugate momenta $p_j$ for each component, equal the impulse $m_j \dot{q}_j$ for a particle with centred mass $m_j > 0$ and generalized velocity $\dot{q}_j$ with $j \in \{1, 2, ..., d\}$. This can be verified using the equation and definition of conjugate momenta,

$$p_j = \frac{\partial L(t,q,\dot{q})}{\partial \dot{q}_j} = \frac{\partial T(t,\dot{q})}{\partial \dot{q}_j} = m_j \dot{q}_j. \tag{4.55}$$

Since the Hamiltonian is derivable with respect to the conjugate momenta, one can further notice that the components of $\nabla_p H(p, q)$ are given as

$$\dot{q}_j = \frac{\partial H(t,p,q)}{\partial p_j} = \frac{\partial}{\partial p_j}\frac{1}{2}\sum_{i=1}^{d} m_i \dot{q}_i^{\,2} = \frac{\partial}{\partial p_j}\frac{1}{2}\sum_{i=1}^{d}\frac{(m_i \dot{q}_i)^2}{m_i} = \frac{\partial}{\partial p_j}\frac{1}{2}\sum_{i=1}^{d}\frac{p_i^2}{m_i} = \frac{p_j}{m_j}$$

for all $j \in \{1, 2, ..., d\}$. It is therefore obvious that the Symplectic Euler scheme can also be defined per components, such that the iteration rule for the conjugate momenta gives

$$p_j^{k+1} = p_j^k - h\frac{\partial H(t^k, p^{k+1}, q^k)}{\partial q_j} = p_j^k - hk_j q_j^k \tag{4.56}$$

$$\Leftrightarrow \quad m_j \dot{q}_j^{k+1} = m_j \dot{q}_j^k - hk_j q_j^k \tag{4.57}$$

$$\Leftrightarrow \quad \dot{q}_j^{k+1} = \dot{q}_j^k - h\frac{k_j}{m_j} q_j^k, \tag{4.58}$$

which means, that the update for the conjugate momenta is equal to the update of the generalized velocities, since the parameters are assumed to be constant over time. Further, the update rule for the components of the generalized velocity vector are given by

$$q_j^{k+1} = q_j^k + h\frac{\partial H(t^k, p^{k+1}, q^k)}{\partial p_j} \tag{4.59}$$

$$\Leftrightarrow \quad q_j^{k+1} = q_j^k + h\frac{p_j^{k+1}}{m_j} \tag{4.60}$$

$$\Leftrightarrow \quad q_j^{k+1} = q_j^k + h\frac{m_j \cdot \dot{q}_j^{k+1}}{m_j} \tag{4.61}$$

$$\Leftrightarrow \quad q_j^{k+1} = q_j^k + h\dot{q}_j{}^{k+1} \tag{4.62}$$

$$\Leftrightarrow \quad q_j^{k+1} = q_j^k + h\left(\dot{q}_j^k - h\frac{k_j}{m_j}q_j^k\right). \tag{4.63}$$

We have derived a semi-implicit or symplectic Euler integration scheme as shown by [14, 40, 39]. In a final step, we assume that the scheme can also be applied to non-conservative systems. Summarizing, the symplectic schemes are used to update the generalized velocities and then use the updated velocities to update the generalized states, as can be seen in Eq. (4.58) and Eq. (4.62).

## Semi-Implicit Scheme for Non-Conservative Systems

It is obvious that Eq. (4.58) is a forward integration scheme for a canonical second order ordinary differential equation of the form

$$\ddot{q}_j + \frac{k_j}{m_j}q_j = 0, \tag{4.64}$$

for each component $j \in \{1,2,...,d\}$, with $d \in \mathbb{N}$ the size of the system. It is therefore a forward scheme for a homogeneous second order ordinary differential equation with constant coefficients, using the reduction of order method.

Assume that a non-homogeneous second order ordinary differential equation for a **non-conservative system** could be defined by

$$\ddot{q}_j + \frac{d_j}{m_j}\dot{q}_j + \frac{k_j}{m_j}q_j = \frac{1}{m_j}b_j(t^k), \tag{4.65}$$

for $t \in [t_0, t_n]$, an additional set of parameters $d_j \geq 0$ for all $j \in \{1,2,...,d\}$ and a continuous source function $b \in \mathscr{C}^0\left([t_0,t_n], \mathbb{R}^d\right)$.

Then, in analogy to the conservative system, a **Semi-Explicit integration** scheme for $d \in \mathbb{N}$ individual components can be defined by

$$\dot{q}_j^{k+1} = \dot{q}_j^k + h\left(-\frac{d_j}{m_j}\dot{q}_j^k - \frac{k_j}{m_j}q_j^k + \frac{1}{m_j}b_j(t^k)\right) \tag{4.66}$$

$$q_j^{k+1} = q_j^k + h\dot{q}_j^{k+1}. \tag{4.67}$$

Since Eq. (4.66) and Eq. (4.67) are no implicit schemes at all, in the sense of Eq. (4.46), we now distinguish between the following general integration schemes for a reduced order system.

Assume that for a second order ordinary differential equation in the sense of Theorem 11, a reduced order system is given by

$$\dot{x}_1 = f(t,x_1,x_2) \tag{4.68}$$

$$\dot{x}_2 = g(t,x_1,x_2) \tag{4.69}$$

with $f(t,x_1,x_2) = x_2$ and $g(t,x_1,x_2) = -a_0x_1 - a_1x_2 + b(t)$ for $x_1,\, x_2 \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}\right)$. Further, let again $a_1,\, a_0 \in \mathbb{R}$ be constants and $b \in \mathscr{C}^0\left([t_0,t_n],\mathbb{R}\right)$ a continuous source term on $[t_0,t_n] \subset \mathbb{R}$.

If two initial values $x_1(t^0) = x_1^0 \in \mathbb{R}$ and $x_2(t^0) = x_2^0 \in \mathbb{R}$ are given, then for an equidistant $\mathscr{D}_h\left(\mathscr{T}\right) = \{t^0,t^1,...,t^N\}$, $N \in \mathbb{N}$ and $k \in \{0,1,...,N-1\}$, the **forward Euler scheme** is given by

$$x_1^{k+1} = x_1^k + hf(t^k,x_1^k,x_2^k) \tag{4.70}$$

$$x_2^{k+1} = x_2^k + hg(t^k,x_1^k,x_2^k) \tag{4.71}$$

for the reduced system.

Further let us consider the **Semi-Explicit or Semi-Forward Euler scheme**

$$x_1^{k+1} = x_1^k + hf(t^k,x_1^k,x_2^k) \tag{4.72}$$

$$x_2^{k+1} = x_2^k + hg(t^k,x_1^{k+1},x_2^k), \tag{4.73}$$

as it is formally, implicit for one argument of the second equation. Nevertheless, the second update rule can be applied straight forward, since the implicit argument results from the forward Euler scheme of the first equation.

Since the **Symplectic or Semi-Implicit Euler scheme** is always implicit for one argument and explicit for the other, we define it for a reduced order system by

$$x_1^{k+1} = x_1^k + hf(t^k,x_1^{k+1},x_2^k) \tag{4.74}$$

$$x_2^{k+1} = x_2^k + hg(t^k,x_1^{k+1},x_2^k) \tag{4.75}$$

which requires solving the first equation with respect to $x_1^{k+1}$. In addition, one could also consider other numerical integration schemes, like the **implicit Midpoint method**, applied to the first component, to get

$$x_1^{k+1} = x_1^k + hf\left(t^k + \frac{h}{2}, \frac{1}{2}\left(x_1^k + x_1^{k+1}\right), x_2^k\right) \tag{4.76}$$

$$x_2^{k+1} = x_2^k + hg(t^k,x_1^{k+1},x_2^k) \tag{4.77}$$

or a **Runge-Kutta ("RK4") scheme**, resulting in

$$\begin{aligned}
k_1 &= f\left(t^k, x_1^k, x_2^k\right) \\
k_2 &= f\left(t^k + \frac{h}{2}, x_1^k + \frac{h}{2}k_1, x_2^k\right) \\
k_3 &= f\left(t^k + \frac{h}{2}, x_1^k + \frac{h}{2}k_2, x_2^k\right) \\
k_4 &= f\left(t^k + h, x_1^k + hk_3, x_2^k\right) \\
x_1^{k+1} &= x_1^k + h\frac{(k_1 + 2(k_2 + k_3) + k_4)}{6} && (4.78) \\
x_2^{k+1} &= x_2^k + hg(t^k, x_1^{k+1}, x_2^k). && (4.79)
\end{aligned}$$

We now give an example for a non-homogeneous second order equation, where we compare the true solution at discrete time-steps with the above defined Euler schemes for a reduced order system.

**Example 14**

*We herein refer to the already introduced Example 13 and compare the numerical solutions for*

$$\ddot{x} + \frac{d}{m}\dot{x} + \frac{k}{m}x = a\sin(\omega t - \varphi),$$

*with the specified parameters $d = 615.0$, $k = 98935.0$, $m = 100.0$, $a = 2.0$, $\omega = 6\pi$ and $\varphi = \frac{\pi}{2}$. Furthermore, since we have a non-homogeneous equation, we choose the initial conditions $x(t^0) = x^0 = 0$ and $\dot{x}(t^0) = v^0 = 0$ for a time horizon $\mathscr{T} = [0,1]$, therefore $t_0 = t^0 = 0$ and $t_n = 1$.*

*Let further for the equidistant discrete time-horizon $\mathscr{D}_h(\mathscr{T})$, the approximate solution for the true value $x^k = x(t^k)$ be given by the expression $u^k$ for $k \in \{1,2,...,N\}$, with a given step-size $h > 0$, such that we can define the* ***total approximation error***

$$\zeta_1(x) := \frac{1}{h}\sum_{k=1}^{N}|x^k - u^k| \tag{4.80}$$

*and the* ***Euclidean distance***

$$\zeta_2(x) := \frac{1}{h}\sqrt{\sum_{k=1}^{N}|x^k - u^k|^2}, \tag{4.81}$$

*which are two appropriate functions to measure the approximation error of the numerical integration schemes. We therefore compare the Forward, Symplectic, Semi-Forward Euler, implicit Midpoint and Runge-Kutta method for step-sizes $h = 10^{-2}$, $h = 10^{-3}$ and $h = 10^{-5}$ of the reduced order system. Therefore, we have $N = 10^3$ discrete steps for the first, $N = 10^5$ steps for the second and $N = 10^7$ steps for the third step-size. It is well known, that numerical approximations converge to the true value of the solution for $h \to 0$ [76]. The approximation errors, as shown in Table 4.1, give an overview about the quality of the introduced numerical*

| $\mathbf{h = 10^{-2}}$ | $t$ | $\zeta_1(x)$ | $\zeta_2(x)$ |
|---|---|---|---|
| Forward Euler | 0.00031 | 49.26396 | 6.60912 |
| Semi-Explicit Euler | 0.00030 | 3.99688 | 0.47119 |
| Symplectic Euler | 0.00032 | 3.69807 | 0.41979 |
| Midpoint | 0.00062 | 3.80150 | 0.43549 |
| Runge-Kutta | 0.00122 | 3.79504 | 0.43442 |
| $\mathbf{h = 10^{-3}}$ | $t$ | $\zeta_1(x)$ | $\zeta_2(x)$ |
| Forward Euler | 0.00274 | 10.57208 | 1.30188 |
| Semi-Explicit Euler | 0.00272 | 3.95689 | 0.45932 |
| Symplectic Euler | 0.00271 | 3.63999 | 0.41217 |
| Midpoint | 0.00600 | 3.74183 | 0.42371 |
| Runge-Kutta | 0.01112 | 3.74115 | 0.42361 |
| $\mathbf{h = 10^{-5}}$ | $t$ | $\zeta_1(x)$ | $\zeta_2(x)$ |
| Forward Euler | 0.26928 | 9.75392 | 1.21097 |
| Semi-Explicit Euler | 0.26767 | 3.97149 | 0.46094 |
| Symplectic Euler | 0.26936 | 3.65582 | 0.41466 |
| Midpoint | 0.55707 | 3.75953 | 0.42559 |
| Runge-Kutta | 1.10489 | 3.75952 | 0.42559 |

TABLE 4.1: Computation of the approximative solution for the introduced integration schemes and varying step-sizes. On the left, the computational time (in seconds) is given to compute the solution. On the two columns on the right, the error constants of Euler integration schemes for total approximation error ($\zeta_1(x)$) and Euclidean distance ($\zeta_2(x)$) are given. To have comparable measures, we evaluated the error terms for all $t \in \mathscr{D}_{0.01}([0,1])$.

*schemes for step-sizes* $h \in \{10^{-2}, 10^{-3}, 10^{-5}\}$. *It needs to be mentioned, that* $h = 0.01$ *is not an appropriate step-size for the Forward Euler scheme to compare it to the remaining methods, since it is then not numerically stable. Nevertheless, it can be mentioned, that the Forward Euler method converges to the exact solution at each* $t \in \mathscr{D}_{0.01}([0,1])$ *for smaller step-sizes. If we compare the remaining methods, it is obvious that the error terms are approximately within the same range for all methods, even for smaller step-sizes. Regarding the computational time, we can recognize almost equal times for the Euler methods, where the Midpoint rule and the Runge-Kutta method require a larger amount of time, due to the more sophisticated integration schemes.*

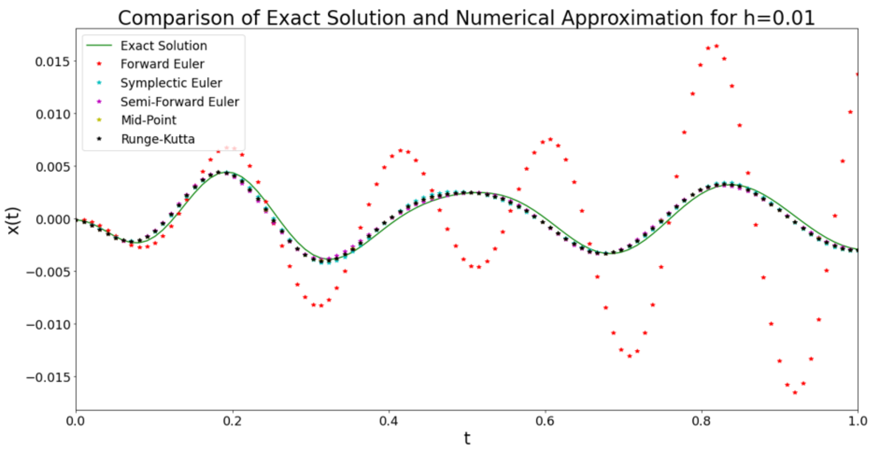


FIGURE 4.1: Exact solution (green) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for the interval $[0,1]$ and step-size $h = 10^{-2}$.

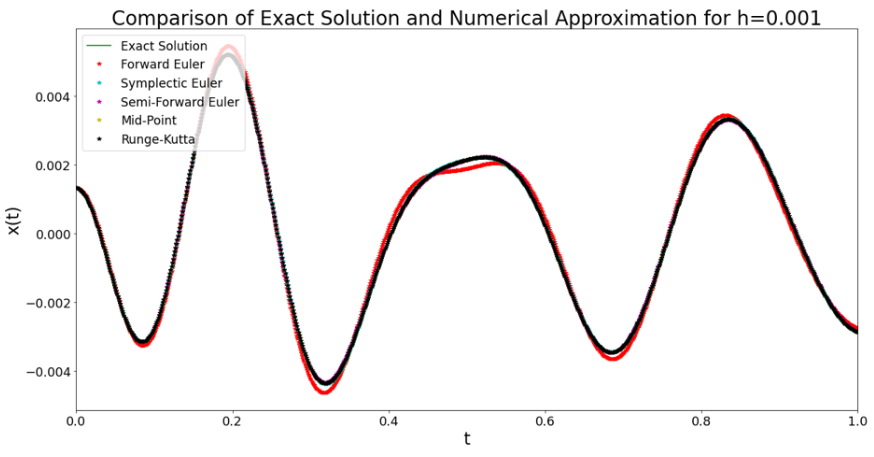


FIGURE 4.2: Exact solution (green) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for the interval $[0,1]$ and step-size $h = 10^{-3}$.

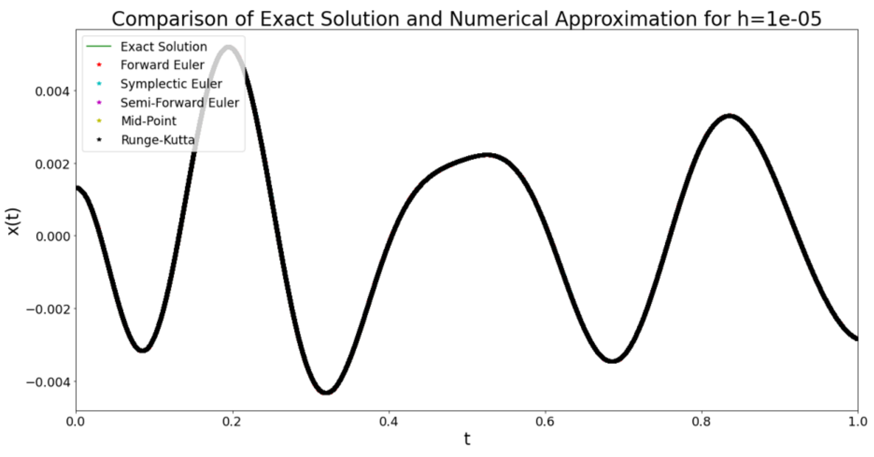


FIGURE 4.3: Exact solution (green) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for the interval $[0,1]$ and step-size $h = 10^{-5}$.

*Visualizations of the numerical results for $h \in \{10^{-2}, 10^{-3}, 10^{-5}\}$ are shown in Figure 4.1 - 4.3. As it has been mentioned for the error terms, one can recognize that the Forward Euler is not stable for smaller step-sizes, but converges to the exact solution for $h \to 0$. All introduced integration schemes converge to the exact solution for small step-sizes, where the approximations of the Semi-Forward, Symplectic, Midpoint and Runge-Kutta scheme are comparably exact for all shown step-sizes. In terms of efficiency, one should consider to use an Euler integration method, as it requires significantly less computational time and is within the same error range as the methods of higher order.*

## 4.2 Linear Systems

We have seen in the previous section that a second order non-homogeneous ordinary differential equation can be transformed into a system of two first order ordinary differential equations. Analogously, a system of three second order ordinary differential equations, like the Quarter-Car-Model, can therefore also be transformed into a system of six first order differential equations, which makes the numerical computation of the approximate solution significantly simpler. The aim of this section is therefore in a final step to compute the numerical solution of the Quarter-Car-Model.

We again give some insights about solving general homogeneous and non-homogeneous systems of arbitrary size explicitly and finally apply the numerical solution methods of the previous section to such systems. Finally, the proposed structure of the system matrix for the Quarter-Car-Model, offers the opportunity to apply the Semi-Implicit Euler scheme to approximate the solution of the system and to compare it to other state-of-the-art approximation methods.

### 4.2.1 Non-Homogeneous Equations

Again, we clarify some general concepts and definitions of linear differential systems, mainly based on [13, 76, 104]. Therefore, we start with general descriptions of the systems, similar to the general second order ordinary differential equations in the previous section. Nevertheless, since the concepts of homogeneous and non-homogeneous equations should be clear by now, we suppose that it is sufficient to shortly discuss them, without splitting it into separate sections again.

The theorems and definitions now presented are broadly based on [76], where one can also find a detailed description including proofs of the theorems.

**Definition 23** (First Order Differential System [76])
*A **system of first order ordinary differential equations**, given by*

$$\dot{x}(t) = A(t)x(t) + b(t) \tag{4.82}$$

*for $t \in [t_0, t_n]$ with $A(t) \in \mathbb{R}^{d\times d}$ a continuous coefficient matrix with $d \in \mathbb{N}$, the unknown state vector $x \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}^d\right)$ and a continuous source vector $b(t) \in \mathbb{R}^d$, is **homogeneous** if $b(t) = 0$ for all $t \in [t_0,t_n]$ and of **constant coefficients**, if $A := A(t)$ is constant for $t \in [t_0,t_n]$.*

The above definition is the general description of a non-homogeneous first order differential system. We now introduce, analogously to the second order equations, a theorem [76] to state that there indeed exists a unique solution to the differential system, if there are initial conditions given.

**Theorem 12** (Existence and Uniqueness of Solutions to First Order Systems [76])
*Assume that $A \in \mathbb{R}^{d\times d}$ is constant and $b \in \mathscr{C}^0\left([t_0,t_n],\mathbb{R}\right)$ is a continuous function. Then for a constant $x_0 \in \mathbb{R}^d$, there **exists a unique function** $x \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}^d\right)$ with $[t_0,t_n] \subset \mathbb{R}$, which solves the **initial value problem***

$$\dot{x}(t) = Ax(t) + b(t) \tag{4.83}$$

$$x(t_0) = x_0. \tag{4.84}$$

**Theorem 13** (Unique Solution of Non-Homogeneous Differential Systems)
*The initial value problem of the first order homogeneous differential system for $t \in [t_0,t_n]$, given by*

$$\begin{aligned}\dot{x}(t) &= Ax(t) + b(t)\\ x(t_0) &= x_0\end{aligned}$$

*with a continuously differentiable function $x \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}^d\right)$, a constant matrix $A \in \mathbb{R}^{d\times d}$, a continuous source vector $b \in \mathscr{C}^0\left([t_0,t_n],\mathbb{R}^d\right)$ and dimension $d \in \mathbb{N}$ has a **unique solution***

$$x(t) = e^{A(t-t_0)}x_0 + e^{At}\int_{t_0}^{t} e^{-A(\tau-t_0)}b(\tau)\,\mathrm{d}\tau \tag{4.85}$$

*for $t \in [t_0, t_n]$.*

**Example 15**

*Similar to the non-homogeneous second order differential equation, we here give an example how the exact solution of a non-homogeneous system can be computed with the help of Theorem 13. Since we have discussed the steps in detail for the second order ordinary differential equation cases, we give the main idea how to solve non-homogeneous systems. Let us therefore assume that a non-homogeneous equation is given by*

$$\dot{x}(t) = Ax(t) + a\sin(\omega t - \varphi) \tag{4.86}$$

*for $t \in [t_0, t_n]$, with $x \in \mathscr{C}^1\left([t_0, t_n], \mathbb{R}^d\right)$ and a constant matrix $A \in \mathbb{R}^{d\times d}$. For the non-linear driving term, we assume that*

$$a\sin(\omega t - \varphi) = \left(a_1\sin(\omega t - \varphi), a_2\sin(\omega t - \varphi), ..., a_d\sin(\omega t - \varphi)\right)^T \tag{4.87}$$

*differs for each dimension in the amplitudes $a_j \in \mathbb{R}$, $j \in \{1, 2, ..., d\}$, where the remaining parameters, namely frequency $\omega \in \mathbb{R}$ and phase $\varphi \in [0, 2\pi)$, are equal for all states, such that*

$$\frac{\mathrm{d}}{\mathrm{d}t} a\sin(\omega t - \varphi) = \omega a\cos(\omega t - \varphi).$$

*Further, let $t_0 = 0$ and $x_0 = 0$. The general unique solution of the differential system therefore simplifies to*

$$x(t) = e^{A(t-t_0)}x_0 + e^{At}\int_{t_0}^{t} e^{-A(\tau - t_0)} a\sin(\omega\tau - \varphi)\mathrm{d}\tau = e^{At}\int_{t_0}^{t} e^{-A\tau} a\sin(\omega\tau - \varphi)\mathrm{d}\tau. \tag{4.88}$$

*One can then verify, using two-times integration by parts such that computation of the indefinite integral gives*

$$\int e^{-At} a\sin(\omega t - \varphi)\mathrm{d}t = \left(I + \frac{1}{\omega^2}A^2\right)^{-1}\left(-\frac{1}{\omega}e^{-At}a\cos(\omega t - \varphi) - \frac{1}{\omega^2}Ae^{-At}\sin(\omega t - \varphi)\right),$$

*where $I \in \mathbb{R}^{d\times d}$ is the identity matrix. Thus the solution is given by*

$$x(t) = e^{At}\left(I + \frac{1}{\omega^2}A^2\right)^{-1}\left(-\frac{1}{\omega}e^{-At}a\cos(\omega t - \varphi) - \frac{1}{\omega^2}Ae^{-At}a\sin(\omega t - \varphi) + c_0\right), \tag{4.89}$$

*with*

$$c_0 = \frac{1}{\omega}a\cos(\varphi) - \frac{1}{\omega^2}Aa\sin(\varphi) \tag{4.90}$$

*which ensures that $x(t_0) = x_0 = 0$. Finally, one can recognize that*

$$\begin{aligned}\dot{x}(t) = Ax(t) + e^{At}\left(I + \frac{1}{\omega^2}A^2\right)^{-1}&\left(\frac{1}{\omega}Ae^{-At}a\cos(\omega t - \varphi) + e^{-At}a\sin(\omega t - \varphi)\right.\\ &\left. + \frac{1}{\omega^2}A^2e^{-At}\sin(\omega t - \varphi) - \frac{1}{\omega}Ae^{-At}a\cos(\omega t - \varphi)\right)\end{aligned}$$

*and it therefore follows that*

$$\begin{aligned}
\dot{x}(t) &= Ax(t) + e^{At}\left(I + \frac{1}{\omega^2}A^2\right)^{-1}\left(e^{-At}b(t) + \frac{1}{\omega^2}A^2e^{-At}b(t)\right) \\
&= Ax(t) + e^{At}\left(I + \frac{1}{\omega^2}A^2\right)^{-1}\left(I + \frac{1}{\omega^2}A^2\right)e^{-At}b(t) \\
&= Ax(t) + e^{At}e^{-At}b(t) \\
&= Ax(t) + b(t).
\end{aligned}$$

So far, we have seen how to compute the exact solution of a non-homogeneous second order ordinary differential equation, as well as the solution of a non-homogeneous first order differential system, where we on the one hand used Theorem 9 for the first, and Theorem 13 for the latter. We have also already seen that we can define efficient numerical integration schemes for a reduced order system, which converge to the true solution for small step-sizes. Therefore, we are now going to apply the same for the first order system.

### 4.2.2 Numerical Solution

We now focus on a specific linear system, which describes a coupled dynamical model as it is given by the already mentioned and broadly introduced Quarter-Car-Model of Chapter 2. In general, the system can be described by the common equation

$$\dot{x}(t) = Ax(t) + b(t), \tag{4.91}$$

where we now have a more specified structure for the coupled system, such that Eq. (4.91) can equally be described as the system

$$\begin{pmatrix} \dot{v}(t) \\ \dot{q}(t) \end{pmatrix} = \begin{bmatrix} D & S \\ I & O \end{bmatrix} \begin{pmatrix} v(t) \\ q(t) \end{pmatrix} + \begin{pmatrix} b_v(t) \\ b_q(t) \end{pmatrix}, \tag{4.92}$$

where $D$, $S \in \mathbb{R}^{d\times d}$ are the matrices of relevant damping and spring-coefficients, $I \in \mathbb{R}^{d\times d}$ is the identity matrix and $O \in \mathbb{R}^{d\times d}$ is the null matrix. Further, let $v \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}^d\right)$ and $q \in \mathscr{C}^2\left([t_0,t_n],\mathbb{R}^d\right)$ and $b_x(t) = 0$ for all $t \in [t_0,t_n]$, such that it can be decoupled into a system of first order differential equations with

$$\begin{aligned}
\dot{v}(t) &= Dv(t) + Sq(t) + b_v(t) \\
\dot{q}(t) &= Iv(t)
\end{aligned}$$

for a continuous function $b_v \in \mathscr{C}^0\left([t_0,t_n],\mathbb{R}^d\right)$. It is then obvious that one can define the function $f(t,q(t),v(t)) := Dv(t) + Sq(t) + b_v(t)$ as well as $g(t,q(t),v(t)) := v(t)$ for all $t \in [t_0,t_n]$ such that we can directly apply the numerical integration schemes, as they are stated by Eq. (4.70), Eq. (4.71) for the Forward Euler, by Eq. (4.72), Eq. (4.73) for the Semi-Explicit Euler and by Eq. (4.74), Eq. (4.75) for the Symplectic Euler scheme.

Since we consider the Quarter-Car-Model, which has a hierarchical structure, depending upon the source term, let us assume that $v(t) := (v_3(t), v_2(t), v_1(t))^T \in \mathbb{R}^d$ and $q(t) := (q_3(t), q_2(t), q_1(t))^T \in \mathbb{R}^d$ for all $t \in [t_0, t_n]$. Then, index 3 describes the motion and velocity for the occupant and the seat, index 2 for the chassis and index 1 for the wheel-suspensions.

Given initial conditions $v^0 = v(t^0) \in \mathbb{R}^d$ and $q^0 = q(t^0) \in \mathbb{R}^d$ and a time discretization $\mathscr{D}_h(\mathscr{T}) = \{t^0, t^1, ..., t^N\}$, a **Forward Euler scheme** for the Quarter-Car-Model is then defined for an equidistant step-width $h = t^{k+1} - t^k$, $k \in \{0, 1, ..., N-1\}$ by

$$\begin{aligned} v^{k+1} &= v^k + h\left(Dv^k + Sq^k + b_v(t^k)\right) \\ q^{k+1} &= q^k + hv^k. \end{aligned}$$

A **Semi-Explicit Euler scheme** can analogously be defined by the equations

$$v^{k+1} = v^k + h\left(Dv^k + Sq^k + b_v(t^k)\right) \tag{4.93}$$

$$q^{k+1} = q^k + hv^{k+1} \tag{4.94}$$

and a **Symplectic Euler scheme** by

$$\begin{aligned} & v^{k+1} = v^k + h\left(Dv^{k+1} + Sq^k + b_v(t^k)\right) \\ \Leftrightarrow \quad & v^{k+1} = (I - hD)^{-1}\left(v^k + h\left(Sq^k + b_v(t^k)\right)\right) \\ & q^{k+1} = q^k + hv^{k+1}. \end{aligned}$$

In comparison to the previous example, we can also derive an adapted **Midpoint scheme**, resulting in

$$\begin{aligned} v^{k+1} &= v^k + h\left(D\frac{1}{2}\left(v^{k+1} + v^k\right) + Sq^k + b_v\left(t^k + \frac{h}{2}\right)\right) \\ q^{k+1} &= q^k + hv^{k+1} \end{aligned}$$

and a **Runge-Kutta ("RK4") method**

$$\begin{aligned} k_1 &= \left(Dv^k + Sq^k + b_v\left(t^k\right)\right) \\ k_2 &= \left(D\left(v^k + \frac{h}{2}k_1\right) + Sq^k + b_v\left(t^k + \frac{h}{2}\right)\right) \\ k_3 &= \left(D\left(v^k + \frac{h}{2}k_2\right) + Sq^k + b_v\left(t^k + \frac{h}{2}\right)\right) \\ k_4 &= \left(D\left(v^k + hk_3\right) + Sq^k + b_v\left(t^k + h\right)\right) \\ v^{k+1} &= v^k + \frac{h}{6}\left(k_1 + 2(k_2 + k_3) + k_4\right) \\ q^{k+1} &= q^k + hv^{k+1}, \end{aligned}$$

where we update the first component with the explicit methods of higher order and use the updated velocity $v^{k+1}$ to compute the new value of the state $q^{k+1}$.

We recognize, that both, the Forward Euler as well as the Semi-Explicit Euler and the Runge-Kutta scheme can be applied straight forward, where the Symplectic Euler scheme and the Midpoint scheme, due to the implicit update rule, require computation of a constant inverse for each step.

**Example 16**
*We regard a first order differential system, which can be separated into a velocity differential system and a displacement differential system via a first order reduction by*

$$\begin{pmatrix} \dot{v}(t) \\ \dot{q}(t) \end{pmatrix} = \begin{bmatrix} D & S \\ I & O \end{bmatrix} \begin{pmatrix} v(t) \\ q(t) \end{pmatrix} + \begin{pmatrix} b_v(t) \\ b_q(t) \end{pmatrix}, \tag{4.95}$$

*where we have $v \in \mathscr{C}^1\left([0,0.5],\mathbb{R}^3\right)$ and $q \in \mathscr{C}^2\left([0,0.5],\mathbb{R}^3\right)$ with initial values $v^0 = q^0 = (0,0,0)^T$. Further, the damping and spring coefficient matrices have been generated from the reference model, given*

$$D = \begin{bmatrix} -6.15 & 6.15 & 0 \\ 0.28 & -2.48 & 2.19 \\ 0 & 32.67 & -32.67 \end{bmatrix}, \quad S = \begin{bmatrix} -989.35 & 989.35 & 0 \\ 45.80 & -114.86 & 69.060 \\ 0 & 1028.77 & -1304.63 \end{bmatrix}, \tag{4.96}$$

*resulting from parameters [60] with $C_3 = 615\frac{\mathrm{Ns}}{\mathrm{m}}$, $C_2 = 4761\frac{\mathrm{Ns}}{\mathrm{m}}$, $C_1 = 0$, $K_3 = 98935\frac{\mathrm{N}}{\mathrm{m}}$, $K_2 = 149171\frac{\mathrm{N}}{\mathrm{m}}$, $K_1 = 40000\frac{\mathrm{N}}{\mathrm{m}}$, $m_3 = 80\mathrm{kg}$, $m_2 = 2160\mathrm{kg}$ and $m_1 = 145\mathrm{kg}$ for the Quarter-Car-Model. It should be obvious that we have $I \in \mathbb{R}^{3\times 3}$ for the identity block and $O \in \mathbb{R}^{3\times 3}$ for the null block. Finally, we use $b_x(t) = (0,0,0)^T$ for all $t \in [0,0.5]$ and $b_v(t) = (0,0,a\sin(\omega t - \varphi))^T$ with $a = 2.0$, $\omega = 6\pi$ and $\varphi = \frac{\pi}{2}$.*

*Similar to the numerical solution of the reduced system in the previous section, we can now compare the numerical solution of the coupled system. Since we regard a first order transformation of the Quarter-Car-Model, we have in total three approximate solutions of the velocities of the system, as well as three approximate solutions for the displacements. We regard the approximation error for $t \in \mathscr{D}_{0.01}([0,1])$ in Table 4.2 for all components and different step-sizes. Applying the numerical schemes to step-sizes smaller than 0.01, requires interpolation of the road-profile, since all road-profiles are generated using $h = 0.01$. We can therefore recognize that a reduction of the step-size does not necessarily lead to a reduction of the approximation error for the Runge-Kutta scheme, which considers intermediate steps $\frac{h}{2}$. Semi-Explicit and Symplectic Euler method show similar error terms for $h \in \left\{10^{-3}, 10^{-4}\right\}$. The implicit Midpoint rule is on average the most accurate method for smaller step-sizes but has increasing error terms from $h = 10^{-3}$ to $h = 10^{-4}$. As seen in the previous section, the Forward Euler method is not stable for $h = 10^{-2}$, but delivers accurate results for appropriate step-sizes.*

*In addition to Table 4.2, we can also analyse the Euclidean distance of the numerical*

| **Method** | **Error** | $\mathbf{h = 10^{-2}}$ | $\mathbf{h = 10^{-3}}$ | $\mathbf{h = 10^{-4}}$ |
|---|---|---|---|---|
| Forward | | 43.03835 | 9.31297 | 8.49140 |
| Semi-Explicit | | 2.03304 | 1.52442 | 1.41172 |
| Symplectic | $\zeta_1(v_3)$ | 1.11990 | 1.46738 | 1.65275 |
| Midpoint | | 0.49203 | 0.05602 | 0.35486 |
| Runge-Kutta | | 2.99215 | 3.91158 | 4.09688 |
| Forward | | 3.09337 | 1.72503 | 1.64962 |
| Semi-Explicit | | 0.95686 | 0.74430 | 0.71432 |
| Symplectic | $\zeta_1(v_2)$ | 0.57501 | 0.70889 | 0.73998 |
| Midpoint | | 0.24170 | 0.02696 | 0.04241 |
| Runge-Kutta | | 0.53120 | 0.64547 | 0.66906 |
| Forward | | 34.72506 | 22.73031 | 21.77863 |
| Semi-Explicit | | 14.78967 | 11.55345 | 11.19035 |
| Symplectic | $\zeta_1(v_1)$ | 9.21607 | 11.19727 | 11.55005 |
| Midpoint | | 3.73631 | 0.353710 | 0.27885 |
| Runge-Kutta | | 7.72363 | 9.18361 | 9.54374 |
| Forward | | 1.27215 | 0.27731 | 0.25536 |
| Semi-Explicit | | 0.07970 | 0.06761 | 0.06260 |
| Symplectic | $\zeta_1(q_3)$ | 0.03109 | 0.03717 | 0.05002 |
| Midpoint | | 0.04481 | 0.03810 | 0.04176 |
| Runge-Kutta | | 0.10371 | 0.13047 | 0.13870 |
| Forward | | 0.09163 | 0.06008 | 0.05325 |
| Semi-Explicit | | 0.03721 | 0.03251 | 0.03808 |
| Symplectic | $\zeta_1(q_2)$ | 0.02756 | 0.02878 | 0.03570 |
| Midpoint | | 0.02788 | 0.02740 | 0.03393 |
| Runge-Kutta | | 0.03629 | 0.03864 | 0.04848 |
| Forward | | 0.88201 | 0.54094 | 0.51510 |
| Semi-Explicit | | 0.57264 | 0.52426 | 0.52051 |
| Symplectic | $\zeta_1(q_1)$ | 0.14363 | 0.16574 | 0.17354 |
| Midpoint | | 0.34463 | 0.29878 | 0.29581 |
| Runge-Kutta | | 0.36435 | 0.40312 | 0.41164 |

TABLE 4.2: Comparison of the constant of the approximation error for different step-sizes $h$ and the introduced Euler schemes as given by Eq. (4.80). To have comparable measures, we evaluated the error terms for all $t \in \mathscr{D}_{0.01}([0,1])$. The dimension of the solution is equal to the dimension of the system, therefore we have in total six solutions to compare, describing the velocities $(v_3, v_2, v_1)$ and the displacements $(q_3, q_2, q_1)$.

| **Method** | **Error** | $\mathbf{h = 10^{-2}}$ | $\mathbf{h = 10^{-3}}$ | $\mathbf{h = 10^{-4}}$ |
|---|---|---|---|---|
| Forward | | 5.85837 | 1.14705 | 1.05320 |
| Semi-Explicit | | 0.24262 | 0.18341 | 0.17108 |
| Symplectic | $\zeta_2(v_3)$ | 0.13611 | 0.17745 | 0.20302 |
| Midpoint | | 0.05853 | 0.00710 | 0.05296 |
| Runge-Kutta | | 0.37002 | 0.48972 | 0.51262 |
| Forward | | 0.38865 | 0.20880 | 0.20022 |
| Semi-Explicit | | 0.12487 | 0.09578 | 0.09280 |
| Symplectic | $\zeta_2(v_2)$ | 0.07336 | 0.09081 | 0.09407 |
| Midpoint | | 0.03102 | 0.00358 | 0.00627 |
| Runge-Kutta | | 0.06456 | 0.07710 | 0.07995 |
| Forward | | 4.34927 | 2.93091 | 2.81282 |
| Semi-Explicit | | 1.96151 | 1.51540 | 1.46704 |
| Symplectic | $\zeta_2(v_1)$ | 1.19827 | 1.46536 | 1.51229 |
| Midpoint | | 0.49035 | 0.04721 | 0.03469 |
| Runge-Kutta | | 0.99868 | 1.16918 | 1.21271 |
| Forward | | 0.17030 | 0.03477 | 0.03187 |
| Semi-Explicit | | 0.00986 | 0.00828 | 0.00819 |
| Symplectic | $\zeta_2(q_3)$ | 0.00388 | 0.00458 | 0.00605 |
| Midpoint | | 0.00538 | 0.00460 | 0.00524 |
| Runge-Kutta | | 0.01283 | 0.01631 | 0.01728 |
| Forward | | 0.01109 | 0.00734 | 0.00631 |
| Semi-Explicit | | 0.00455 | 0.00397 | 0.00469 |
| Symplectic | $\zeta_2(q_2)$ | 0.00321 | 0.00341 | 0.00462 |
| Midpoint | | 0.00327 | 0.00316 | 0.00413 |
| Runge-Kutta | | 0.00443 | 0.00473 | 0.00624 |
| Forward | | 0.11171 | 0.07066 | 0.06735 |
| Semi-Explicit | | 0.07488 | 0.06762 | 0.06723 |
| Symplectic | $\zeta_2(q_1)$ | 0.01831 | 0.02095 | 0.02186 |
| Midpoint | | 0.04551 | 0.03954 | 0.03938 |
| Runge-Kutta | | 0.04808 | 0.05281 | 0.05400 |

TABLE 4.3: Comparison of the Euclidean distance for different step-sizes $h$ and the introduced Euler schemes as given by Eq. (4.81). To have comparable measures, we evaluated the error terms for all $t \in \mathscr{D}_{0.01}([0,1])$. The dimension of the solution is equal to the dimension of the system, therefore we have in total six solutions to compare, describing the velocities $(v_3, v_2, v_1)$ and the displacements $(q_3, q_2, q_1)$.

| **Method** | **Time** | $\mathbf{h = 10^{-2}}$ | $\mathbf{h = 10^{-3}}$ | $\mathbf{h = 10^{-4}}$ |
|---|---|---|---|---|
| Forward | | 0.00119 | 0.00892 | 0.08807 |
| Semi-Explicit | | 0.00119 | 0.00899 | 0.09278 |
| Symplectic | $t$ | 0.00150 | 0.01395 | 0.13597 |
| Midpoint | | 0.00220 | 0.02510 | 0.25327 |
| Runge-Kutta | | 0.00297 | 0.03008 | 0.30434 |

TABLE 4.4: The computational time (in seconds) to approximate the solution of the Quarter-Car-Model with one of the introduced numerical integration schemes is compared for different step-sizes $h$.

*approximations in Table 4.3. The results are consistent with those shown in the previous section. Therefore, we can conclude that all introduced methods can be appropriately used to generate numerical approximations of the reduced Quarter-Car-Model. Nevertheless, we need to consider efficient methods, such that one can handle generations of the road-profile, restricted to* $h = 0.01$.

*We compare the computational effort to numerically generate the approximate solutions of the system with the introduced schemes by regarding Table 4.4. Obviously, the Midpoint and the Runge-Kutta scheme require a larger amount of time, due to the more complex integration scheme per step. The computational time for Forward and Semi-Explicit Euler are approximately the same throughout all variations of h, where the Symplectic Euler requires additional computational time, due to the computation of the matrix inverse at the starting point of the integration.*

*Examples of the numerical solutions of the component's velocities for all introduced*

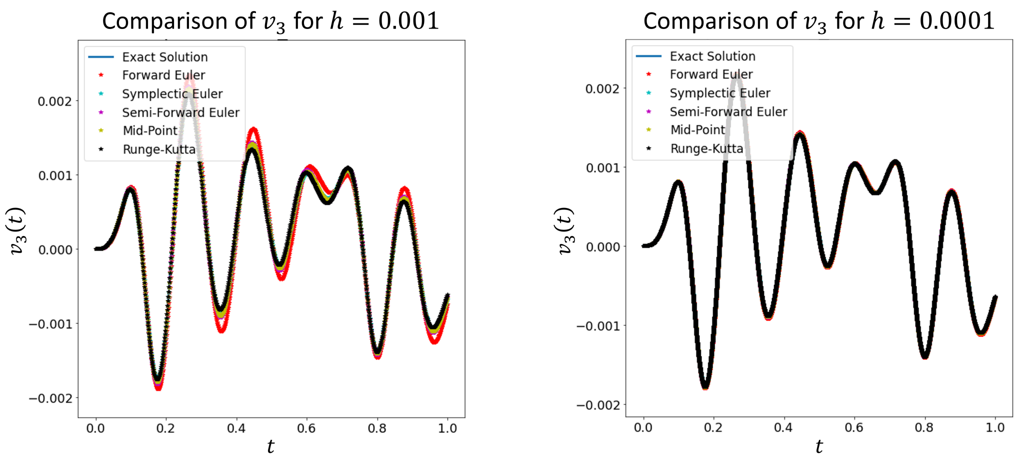


FIGURE 4.4: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $v_3$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

*schemes, are shown in Figure 4.4 - 4.6 for* $h = 10^{-3}$ *on the l.h.s and* $h = 10^{-4}$ *on the r.h.s. Obviously, we recognize that all methods converge to the true solution for small step-sizes, as*

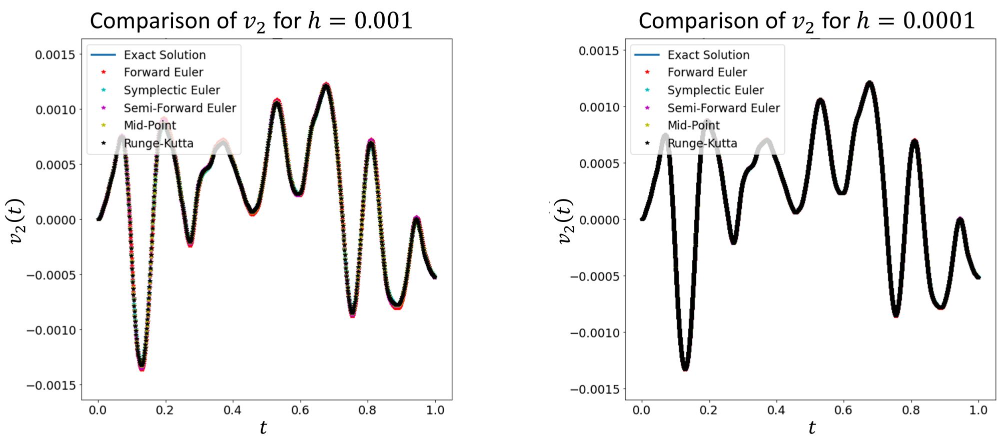


FIGURE 4.5: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $v_2$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

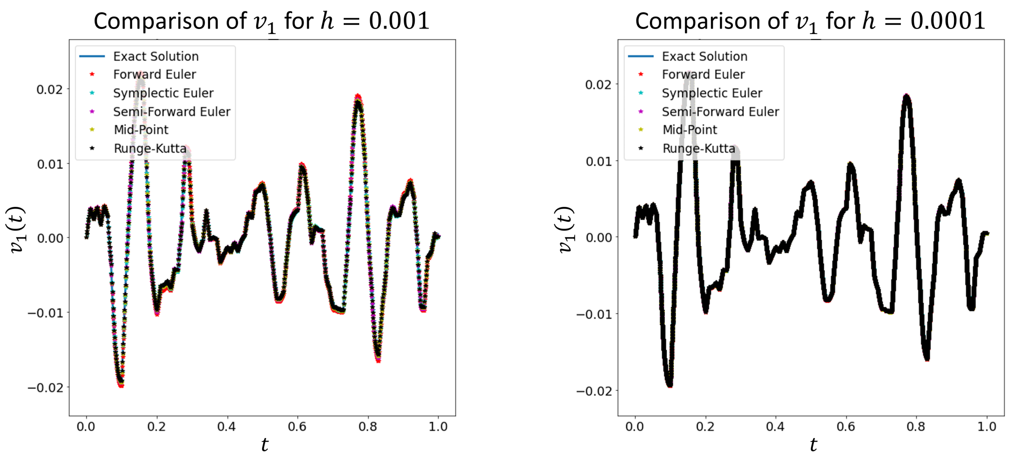


FIGURE 4.6: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $v_1$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

*it has been analysed in the previously shown tables. Due to the stability of the Forward Euler for small step-sizes, we can further recognize larger deviations for $h = 10^{-3}$. Nevertheless, the method converges for an appropriate step-size of $h = 10^{-4}$ to the true solution. Similar to the approximate velocities, we regard the approximate states of the system components in Figure 4.7 - 4.9. The results are consistent with those of the approximate velocities. It is therefore possible to use several different numerical schemes to generate appropriate results for the first order reduction of the Quarter-Car-Model.*

We have shown how to compute the solution of a family of coupled dynamical differential systems explicitly, using the solution theory of ordinary differential equations as well as the numerical solutions for different Euler methods. We have further found out that structure preserving methods (Semi-Implicit / Semi-Explicit) give a more appropriate scheme to efficiently compute approximations of coupled dynamical systems, even when choosing

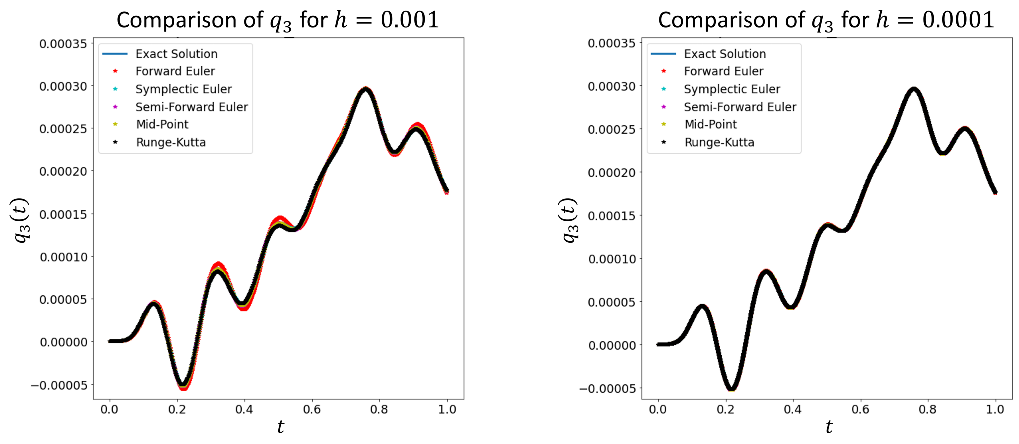


FIGURE 4.7: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $q_3$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

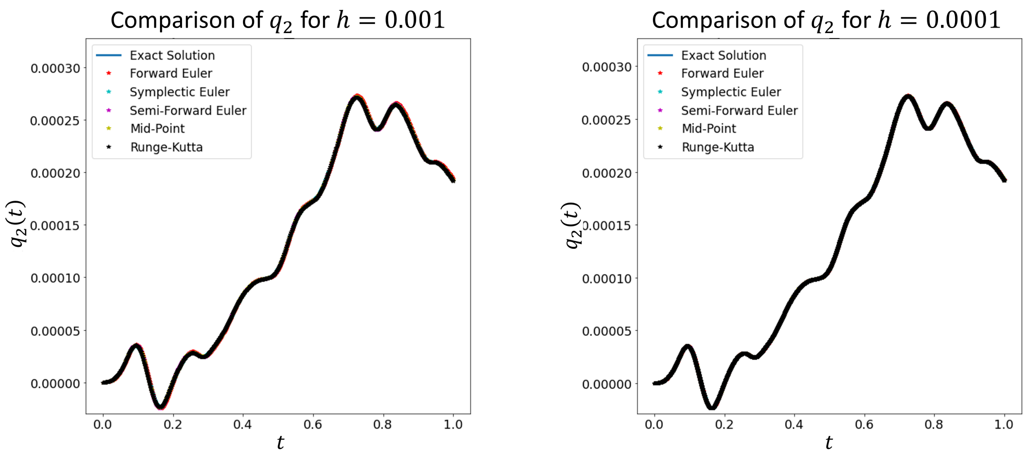


FIGURE 4.8: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $q_2$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

a smaller step-size, where a state-of-the-art Forward-Euler method fails. Finally, we have shown in very detail, how to compute these solutions and can now in a last step compute appropriate acceleration data of a dynamical system, given the numerical solutions of the Quarter-Car-Model, introduced in the previous sections.

## 4.3 Data Generation from Randomly Perturbed Models

We know that the discussed second order ordinary differential equations can be used to describe the acceleration of a coupled dynamical system, using the velocities and displacements with their corresponding spring - and damping - coefficients. Since velocities and displacements are both partial solutions of the first order differential system, one can in a first step

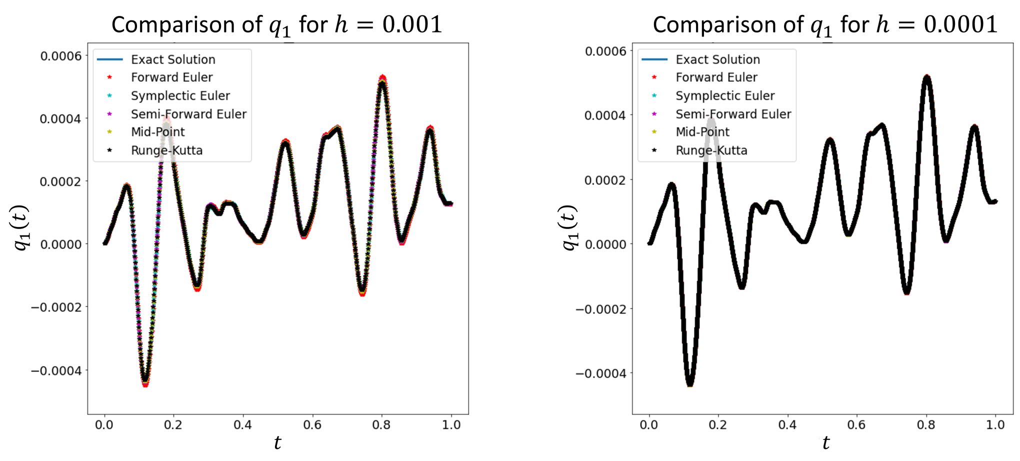


FIGURE 4.9: Exact solution (blue) of the example, compared to the Forward-Euler (red), Symplectic Euler (cyan), Semi-Forward (magenta), Midpoint (yellow) and Runge-Kutta (black) integration schemes for $q_1$ on interval $[0,1]$ and step-size $h = 0.001$ (left) and $h = 0.0001$ (right).

compute the approximation of the discrete solution with an appropriate numerical integration method and then in a second step, compute the l.h.s. of the ordinary differential equations to acquire discrete values of the acceleration of the system.

We can then assume that this artificially created sequential data, could in principle also be taken from a real sensing device, while a vehicle is driving on a road. Therefore, we are going to focus for the generation of artificial data on randomly generated road-profiles, following the principle in Chapter 2 based on [106]. In order to acquire a large amount of data, we use random variations of the coefficients of the system matrix, based upon the explicit parameters given in the vehicle model of [60].

### 4.3.1 Definition of the System

Let us shortly recapture the main components to define a separable differential system to apply numerical solvers to it for the data generation process. Therefore, we have already seen in Chapter 2, that the displacement of a road-profile was defined for a position $s \in [s_0, s_n] \subset \mathbb{R}$ for a road with length $|s_n - s_0|$ by

$$u(s) = \sum_{i=1}^{N} A_i \sin\left(\Omega_i s - \varphi_i\right), \tag{4.97}$$

where we have $N = 1000$ realizations of the phase shift, resulting from uniformly distributed random phase shifts $\varphi_i \sim \mathscr{U}\left([0, 2\pi)\right)$ for $i \in \{1, 2, ..., N\}$. Further we need to use an obvious state-time shift, since the differential equations are defined to depend upon a time variable $t \in [t_0, t_n]$. In this case, we make use of $tv_c = s$ for $t \in [t_0, t_n] \subset \mathbb{R}$, where $v_c \in \mathbb{R}$ is the constant velocity of a car. For instance, let us assume that we have a track of 500 meters for $[s_0, s_n] = [0, 500]$ , where the car is supposed to constantly drive a speed of $30\frac{\text{km}}{\text{h}}$. It then holds that the car needs a total time of 60 seconds and we therefore get $t \in [t_0, t_n] = [0, 60]$, such that

$$u(t) = \sum_{i=1}^{N} A_i \sin(\Omega_i t v_c - \varphi_i) \tag{4.98}$$

with $\Omega_0 = 1 \cdot \frac{1}{\mathrm{m}}$, $\Omega_1 = 0.02\pi \cdot \frac{1}{\mathrm{m}}$, $\Omega_N = 6\pi \cdot \frac{1}{\mathrm{m}}$ and $\Omega_{i+1} = \Omega_i + \Delta\Omega$ for $i \in \{1, 2, ..., N-1\}$ and $\Delta\Omega = \frac{\Omega_N - \Omega_1}{N-1}$.

We now want to investigate the degree of roughness for the above description of the road-displacement. It is therefore necessary to again regard the definition of the amplitude [106] by

$$A_i = \sqrt{\Phi(\Omega_i) \frac{\Delta\Omega}{\pi}} \tag{4.99}$$

with

$$\Phi(\Omega_i) = \Phi(\Omega_0) \left( \frac{\Omega_1^{-2}}{\Omega_0} \right) \quad \text{for} \quad \Omega_1 < \Omega_i \leq \Omega_N, \tag{4.100}$$

where the degree of roughness then depends upon the value of $\Phi(\Omega_0)$. According to the ISO 8606 road-profile classification [106], we have a road of class **A**, if $\Phi(\Omega_0) = 4^0 \cdot 10^{-6} m^3$, a road of class **B** for $\Phi(\Omega_0) = 4^1 \cdot 10^{-6} m^3$ and a road of class **C**, if $\Phi(\Omega_0) = 4^2 \cdot 10^{-6} m^3$. We restrict our investigations to these three ISO classes, where it is obvious to see, that the classification scheme is dependent upon a scaling of the amplitude of the road displacement for all time-steps.

Let us now further assume that we have a fixed system matrix of our system, defined by

$$A_\mu := \begin{bmatrix} D_\mu & S_\mu \\ I & O \end{bmatrix}, \tag{4.101}$$

where

$$D_\mu := \begin{bmatrix} -\frac{C_3}{m_3} & \frac{C_3}{m_3} & 0 \\ \frac{C_3}{m_2} & -\frac{C_2+C_3}{m_2} & \frac{C_2}{m_2} \\ 0 & \frac{C_2}{m_1} & -\frac{C_2+C_1}{m_1} \end{bmatrix}, \quad S_\mu := \begin{bmatrix} -\frac{K_3}{m_3} & \frac{K_3}{m_3} & 0 \\ \frac{K_3}{m_2} & -\frac{K_2+K_3}{m_2} & \frac{K_2}{m_2} \\ 0 & \frac{K_2}{m_1} & -\frac{K_2+K_1}{m_1} \end{bmatrix}, \tag{4.102}$$

$I \in \mathbb{R}^{3\times3}$ is the identity matrix and $O \in \mathbb{R}^{3\times3}$ is the null matrix. In this case, we refer to [60] and use $C_3 = 615\frac{\mathrm{Ns}}{\mathrm{m}}$, $C_2 = 4761\frac{\mathrm{Ns}}{\mathrm{m}}$, $C_1 = 0$, $K_3 = 98935\frac{\mathrm{N}}{\mathrm{m}}$, $K_2 = 149171\frac{\mathrm{N}}{\mathrm{m}}$, $K_1 = 40000\frac{\mathrm{N}}{\mathrm{m}}$, $m_3 = 80\mathrm{kg}$, $m_2 = 2160\mathrm{kg}$ and $m_1 = 145\mathrm{kg}$ as realizations of the parameters for the Quarter-Car-Model.

We have inserted the value $C_1 = 0$, because in a more generic case, we could also investigate the parameters of a dynamical system, which has also a damping component for the wheel-suspensions. It is further obvious that one can find a more compact expression for the system parameters, since although we have $A_\mu \in \mathbb{R}^{6\times6}$, we recognize that there are only 10 parameters to distinguish in the system, such that we can define a bijection of the parameter vector

$$p_\mu := \left(\frac{C_3}{m_3}, \frac{K_3}{m_3}, \frac{C_3}{m_2}, \frac{C_2}{m_2}, \frac{K_3}{m_2}, \frac{K_2}{m_2}, \frac{C_2}{m_1}, \frac{C_2+C_1}{m_1}, \frac{K_2}{m_1}, \frac{K_2+K_1}{m_1}\right)^T \tag{4.103}$$

and the system matrix $A_\mu$, using the short notation $p_\mu \leftrightarrow A_\mu$.

A Semi-Explicit Euler method will be used in the following section to get approximate solutions of the Quarter-Car-Model and to generate discrete acceleration values of the individual components of the system. Therefore, we make use of a step-size $h = 0.01$, which equals the sampling frequency of a common sensing device. Moreover, the computation of the road-profile is also restricted to $h = 0.01$, as we do not intend to produce interpolation errors of the road-profile for synthetic data generation. The Semi-Explicit Euler method has shown to return appropriate approximations of the system. Besides, it requires less computational time as for instance the implicit Method of the Runge-Kutta method and does not require the computation of a matrix inverse.

### 4.3.2 Variations of Systems

We have defined the general terms of the main system in the previous section. It is now further possible to use a variation of road-profiles and system matrices to generate arbitrarily large datasets for Neural Network training. We want to give a short overview now, with the help of the following scheme, to show the data generation process.

1. We generate a total number of $N_{train}$ + $N_{test} \in \mathbb{N}$ road-profiles, which satisfies that we can compute sufficiently large training and test datasets. We need to ensure that most road-profiles belong to ISO class **A** ($N_{A;train} + N_{A;test}$ profiles), some less to class **B** ($N_{B;train} + N_{B;test}$ profiles) and a few to class **C** ($N_{C;train} + N_{C;test}$ profiles). It should then hold that $N_{train} = N_{A;train} + N_{B;train} + N_{C;train}$ and $N_{test} = N_{A;test} + N_{B;test} + N_{C;test}$. We choose a time-horizon $[t_0, t_n] \subset \mathbb{R}$ and a time-step $h > 0$. Therefore, each road-profile has a total number of $N = \frac{t_n - t_0}{h}$ discrete time-steps per road-profile. The following steps are explained for the training dataset, but are analogously done for the test dataset.

2. We choose a standard deviation of $\sigma_{veh} > 0$ for the vehicle parameters, including chassis and wheel-suspensions. For each road-profile, we want to compute a total number of $M \in \mathbb{N}$ samples with random parameter realization. Therefore, we compute a dataset of a total number of $N_S = MN_{train}$ samples. Let one training sample be indexed by $m \in \{1, 2, ..., N_S\}$. Let the system matrix for sample $m$ be denoted by $A_m$, let the corresponding parameter vector be denoted analogously by $p_m$. Then for all elements of parameter $p_m$, denoted by $p_{m;l}$ with $l \in \{3, 4, ..., 10\}$ we randomly choose the parameter value for each sample $k \in \{1, 2, ..., N_S\}$, where each realization of the parameter

values follows a Gaussian distribution $\mathcal{N}\left(p_{\mu;l},\sigma_{veh}^2\right)$. Since we have $p_{\mu;1}=\frac{C_3}{m_3}$ and $p_{\mu;2}=\frac{K_3}{m_3}$, we assume that the spring - and damping-constants of the seat are the same, but the occupants mass of the system should vary. Therefore we assume that $m_3$ varies for all samples, following another distribution $\mathcal{U}\left([\underline{m},\overline{m}]\right)$, where $\underline{m}>0$ is the lower limit for the mass, while $\overline{m}>0$ is the upper limit for the mass. This means that the parameter values described in the previous section, are the mean values for the choice of the random samples.

3. We denote by $u_m\in\mathbb{R}^N$ the road-profile belonging to sample $m$. Then for initial conditions $v_m^0=q_m^0=0$ for all $m\in\{1,2,...,MN_{train}\}$, we compute

$$\begin{aligned}v_m^{k+1}&=v_m^k+h\left(D_mv_m^k+S_mq_m^k+b_m^k\right)\\ q_m^{k+1}&=q_m^k+hv_m^{k+1},\end{aligned}$$

with $b_m^k=\left(0,0,\left(p_{m,10}-p_{m,9}\right)u_m^k\right)^T$ for $k\in\{0,1,...,N-1\}$where $p_{m,10}-p_{m,9}$ is the coefficient of the spring, which scales the displacement of the road-profile, that acts on the dynamical system at each time-step. We store $v_m\in\mathbb{R}^{N\times 3}$ and $q_m\in\mathbb{R}^{N\times 3}$.

4. For all time-steps $k\in\{0,1,...,N-1\}$, we compute

$$a_m^k=D_mv_m^k+S_mq_m^k+b_m^k, \tag{4.104}$$

which is the l.h.s. of the differential equation and is therefore the discrete value of the acceleration for each time-step. We store $a_m\in\mathbb{R}^{N\times 3}$.

5. Repeat 2.-4. for the $N_{test}$ road-profiles for the test dataset.

Let then

$$\mathcal{S}=\{\left(\left(a_m,v_m,q_m,u_m\right),p_m\right)\}_{m=1}^{N_S} \tag{4.105}$$

be the artificially generated training dataset. This means, that the training samples are given by acceleration $a_m$, velocity $v_m$, displacement $q_m$ and road-displacement $u_m$ for all $m=\{1,2,...,N_S\}$. For the training data, according to the above scheme, we choose $N_{A;train}=200$, $N_{B;train}=140$, $N_{C;train}=60$, $[t_0,t_n]=[0,60]$, $h=0.01$ and $\sigma_{veh}=0.1$. The choice of $\sigma_{veh}$ then means that the parameters of the system, including chassis and wheel-suspensions should be normally distributed around the mean value of the Quarter-Car-Model presented before, with a standard deviation of 10%.

In a similar way, we can define a test dataset with

$$\mathcal{T}=\{\left(\left(a_m,v_m,q_m,u_m\right),p_m\right)\}_{m=N_S+1}^{N_S+N_T} \tag{4.106}$$

with the same type of data, where the road-profiles generated for the test data, are independent on the generation of the road-profiles for the training data. We use $N_{A;test}=100$, $N_{B;test}=70$ and $N_{C;test}=30$, such that $N_T=MN_{test}$ with $N_S=2\cdot N_T$, meaning that the size of the training set is double the size of the test set. Further, we choose $\underline{m} = 30$ and $\overline{m} = 150$, meaning that the masses of the seat, including the occupant, vary uniformly distributed from 30 kg to 150 kg, where 30 kg should be the rear mass of the seat without a person on it.

Since we have in the above scheme described the data generation process of training and test samples, resulting from the numerical solution of randomly generated Quarter-Car-Models, we want to show now one arbitrary sample of the training set in the following section.

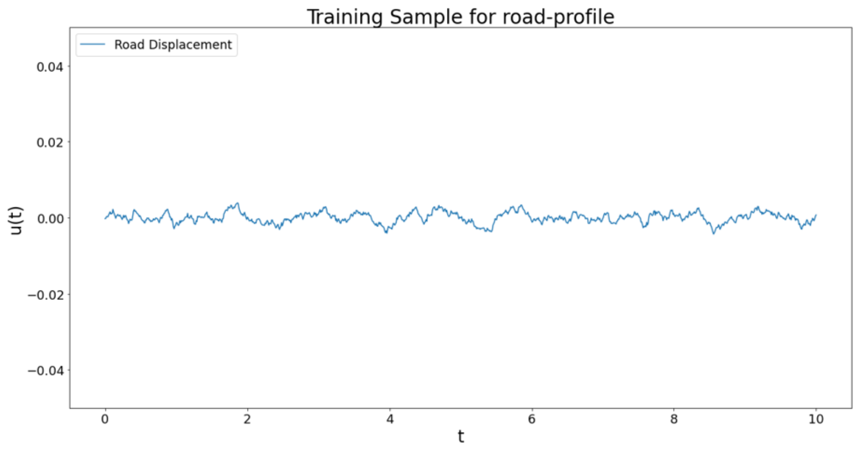


FIGURE 4.10: Example of a road profile for step-size $h = 0.01$ and a time horizon of $[0, 10]$

An example of a randomly generated road-profile is shown in Figure 4.10 for a time horizon of 10 seconds and a step-size of $h = 0.01$. The herein shown road-profile is used for the following generation of random training samples of the Quarter-Car-Model.

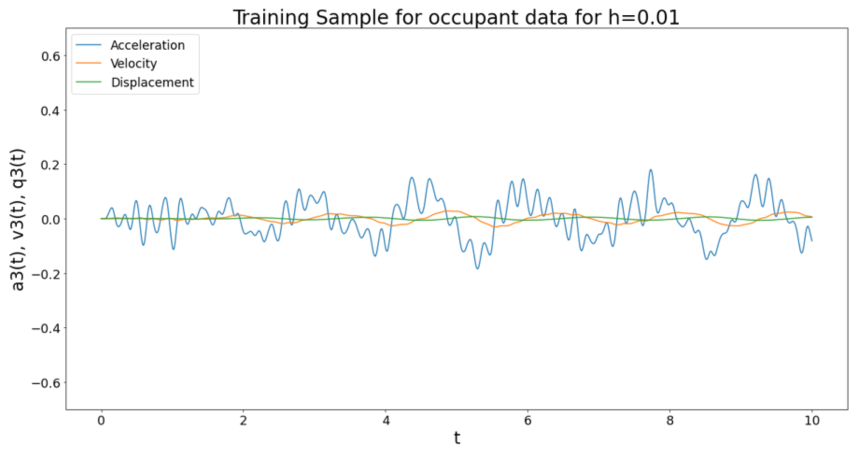


FIGURE 4.11: Example of acceleration (green), velocity (red) and displacement (blue) of the occupant and seat data for step-size $h = 0.01$ and a time horizon of $[0, 10]$

Since the solution of the differential system returns three trajectories of displacement $q_{m;i}$, velocity $v_{m;i}$ and acceleration $a_{m;i}$ with $m \in \{1,2,...,N_S+N_T\}$ and $i \in \{1,2,3\}$ for the Quarter-Car-Model, we can see one example for the training dataset, resulting from the road-profile of Figure 4.10. The data samples for the motion of the seat, including the occupant, is shown in Figure 4.11. We can see from Eq. (4.104) that the acceleration depends upon a scaling of velocity and displacement with the corresponding parameters. Thus one can recognize larger amplitudes of the acceleration, where we have a more flat shape of the velocity and the displacement for this sample.

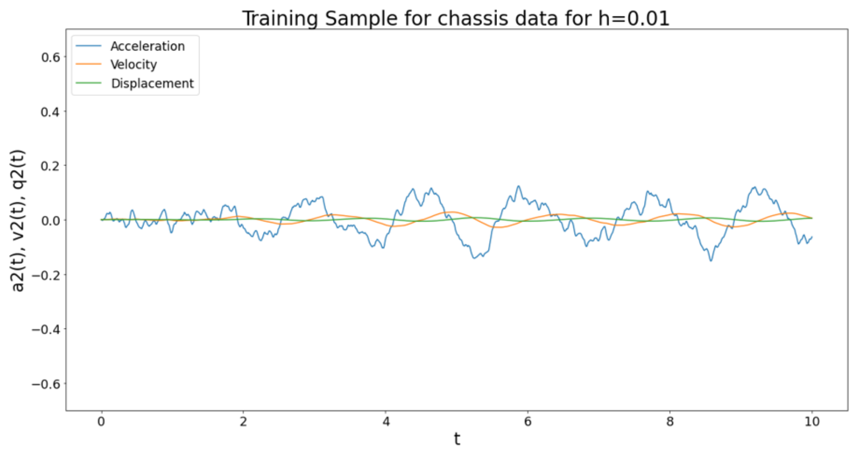


FIGURE 4.12: Example of acceleration (green), velocity (red) and displacement (blue) of the chassis data for step-size $h = 0.01$ and a time horizon of $[0,10]$

Figure 4.12 is a similar example as given in Figure 4.11, where we can recognize the trajectories of motion for the chassis of the Quarter-Car-Model. The shape of the returned acceleration, velocity and displacement is similar to the sample shown for the motion of the occupant. Nevertheless, the shape of the acceleration is less volatile compared to the acceleration for the occupant, resulting from a significantly larger damping coefficient for the chassis.

Figure 4.13 finally shows the trajectories of motion for the wheel-suspensions of the Quarter-Car-Model, analogously to the chassis components in Figure 4.12 and the seat and occupant components of Figure 4.11, resulting from the road-profile of Figure 4.10. Although the displacement and velocity of the herein shown trajectories are comparably flat to the other component trajectories, one can recognize a highly volatile shape of the acceleration data, resulting from setting the damping coefficient $C_1 = 0$.

Finally, since we have assumed the parameters to be randomly generated from Uniform and Gaussian distributions for the training and test data, we can verify this assumption by having a look at the histograms of the parameter values for all training and test samples.

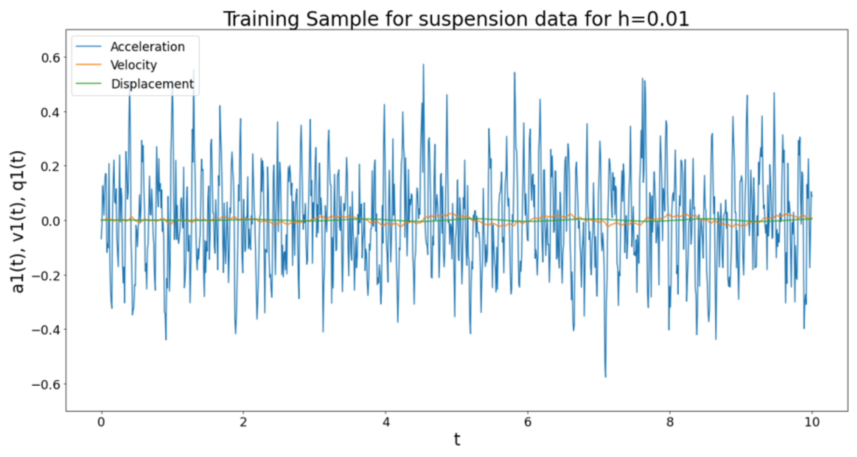


FIGURE 4.13: Example of acceleration (green), velocity (red) and displacement (blue) of the wheel-suspension data for step-size $h = 0.01$ and a time horizon of $[0,10]$

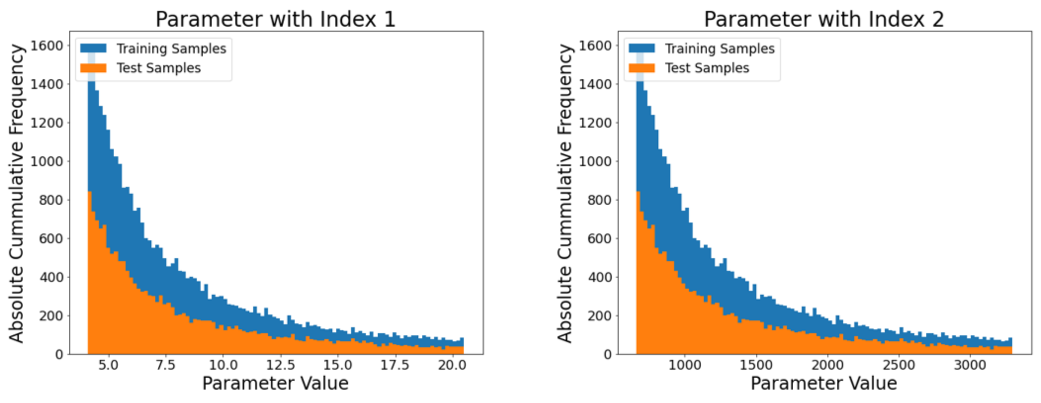


FIGURE 4.14: Histogram of parameters 1 and 2 for the training data (blue) and the test data (orange) for bin-width 1. The realized parameter values are shown on the horizontal axis, while the absolute cumulative frequency is shown on the vertical axis.

We have assumed that the masses of the occupant component are randomly generated, following a uniform distribution. Since the damping - and spring - coefficients of the first equation are scaled by the realization of the mass, one can recognize in Figure 4.14 that the resulting parameter values are not uniformly distributed, which is an obvious investigation if one considers a function $f(m) = \frac{K}{m}$ for $K > 0$ and $m \in [\underline{m}, \overline{m}]$ with $\underline{m} > 0$ and $\overline{m} > 0$. The histogram on the l.h.s. according to the definition of the parameter vector in Eq. (4.103), shows the distribution for the training and for the test samples of the damping coefficient, while the r.h.s. shows the same for the spring coefficient. Since, we have $N_S = 2 \cdot N_T$ it is further obvious that there are less realizations of the test samples compared to the training samples.

The remaining parameters of the Quarter-Car-Model have been assumed to be Gaussian distributed, since we want to have minor vehicle specific deviations of the model. Therefore, one can recognize the realization of the parameters for the chassis in Figure 4.15. We can

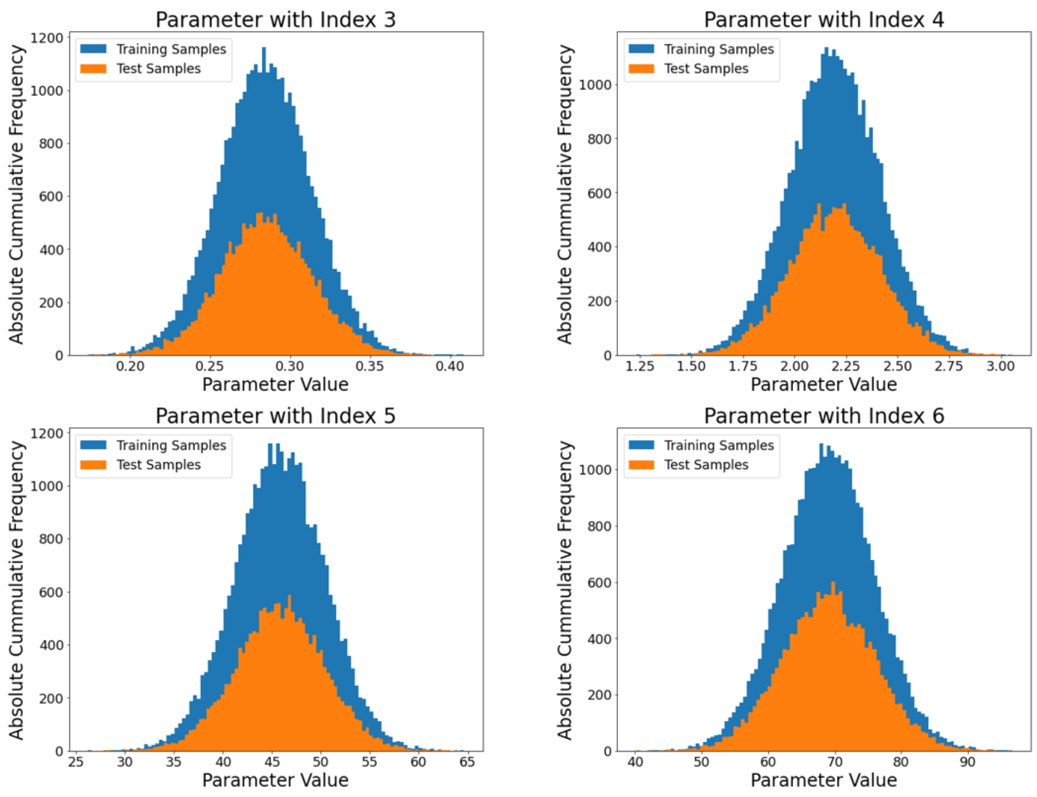


FIGURE 4.15: Histogram of parameters 3, 4, 5 and 6 for the training data (blue) and the test data (orange) for bin-width 1.The realized parameter values are shown on the horizontal axis, while the absolute cumulative frequency is shown on the vertical axis.

easily verify that the realization of the parameters for the training as well as for the test samples do indeed follow a Gaussian distribution with mean values specified in Eq. (4.103).

Finally, the realization of the parameter for the wheel-suspension in Figure 4.16 is similar to those discussed in Figure 4.15. Again, one can recognize a discrete Gaussian distribution for the training and test samples.

We have given a detailed introduction to solve ordinary differential equations of coupled dynamical systems in this section. Further we have shown that arbitrarily large datasets of random road-profiles and random parameter realizations can be used to specify a training and a test dataset. For the final chapter, we will show that this dataset can then be used, to identify the (then assumed to be unknown) parameters of the system. Therefore, the following chapter will give a short overview about how parameters can be estimated from sequential data for optimal systems. In a further step, we will investigate that a combination of commonly used parameter estimation methods and data-driven methods can successfully be used to even estimate the parameters of an uncertain system. These **Hybrid Models** show success for the estimation problem, where one component of the model would fail to solve the task for the uncertain system.

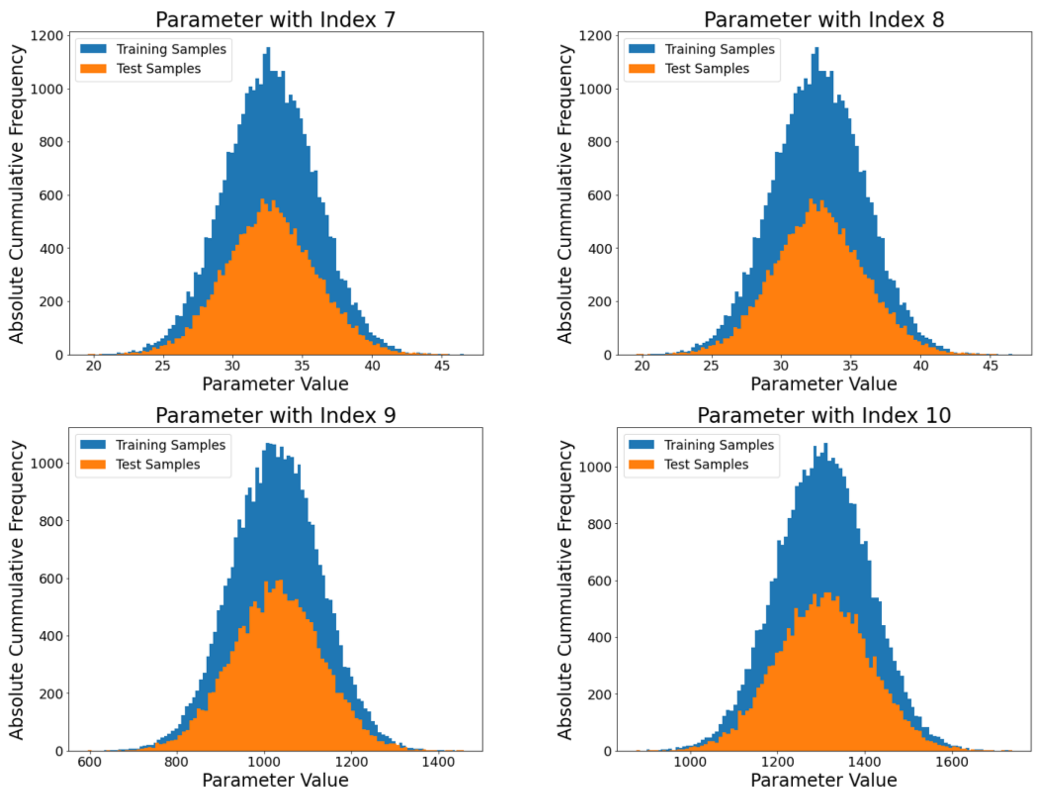


FIGURE 4.16: Histogram of parameters 7, 8, 9 and 10 for the training data (blue) and the test data (orange) for bin-width 1. The realized parameter values are shown on the horizontal axis, while the absolute cumulative frequency is shown on the vertical axis.

# Chapter 5

# Parameter Estimation for Dynamical Systems

**Parameter Estimation** [6], in most cases also analogously called **System Identification** [68], deals with the problem of fitting a hypothesis or candidate model to a given dataset of input sources and output observations, minimizing the difference of estimated candidate model states and observable measurements. Therefore, one needs to make use of the following terms: **Control** (model input), **System** (model) and **Response** (model output). The herein described structure of those three basic components leads to different problems in a vast field of mathematical research. Without any detailed mathematical definition of the components, one can for simplicity reasons denote the input of a system by $u$, the model by $G$ and the output by $y$, such that $G(u) = y$.

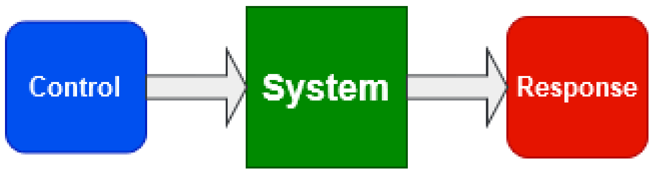


FIGURE 5.1: Input (control) is passed through the model (system) and generates an observation (system response). The scheme is crucial for all following experiments of the section.

**First Problem**: Assuming that the input and the model are known, one can compute the underling true system response by solving the **forward problem** $y = G(u)$. If one has real measurements of such a system, denoted by $\hat{y}$, one can estimate the measurement error $\varepsilon = ||G(u) - \hat{y}||$. Computing the optimal states $y$ of the forward problem for an optimal system, has for instance already been investigated in Chapter 4 when solving differential systems to approximate the discrete accelerations of the Quarter-Car-Model states with the help of appropriate numerical schemes. As one never has continuous measurements, due to sensing devices are restricted to given sampling frequencies, a sufficiently large finite set of true states is appropriate to estimate the measurement noise for such problems.

**Second Problem**: Assuming that we have knowledge about the underlying true system and a finite set of responses, we want to estimate the corresponding value of the unknown input control. If the system is linear, such that $G$ is an invertible matrix, one can find the solution of this **inverse problem** [6] by $u = G^{-1}y$. Since it is most often the case, that the system is in general not invertible, a more common method is finding an approximation of $u$ by minimizing the distance of given candidate controls $\hat{u}$ for $||G(\hat{u}) - y||$. Problems herein occur, if the true system response it not exactly known, such that one needs to use the, probably noisy, measurements $\hat{y}$ as target values. The described second problem is also called **reconstruction problem**, as the now following last problem is sometimes also referred to as an inverse problem in some sources.

**Third Problem**: Assuming that we have full knowledge about the input controls and the corresponding system responses, we want to identify the model, which satisfies a mapping from input to output. The problem is also called **system identification** [6], as we want to estimate the system from this input-output relation. Since most systems are parametrized functions or matrices, characterizing the given shape of measurements, one can instead also use the term **parameter estimation**. One cannot explicitly solve $G(u) = y$ for the system $G$, since input and output do not necessarily need to have equal dimension. Therefore, similar to the previously described reconstruction problem, one needs to have candidate systems $\hat{G}$ to find such a system that minimizes $||\hat{G}(u) - y||$. Again, intense problems can occur, if the true underlying system response $y$ is unknown, but only noisy or incomplete measurements $\hat{y}$.

The last described problem, system identification, as it has already been intensively studied in the field of system theory, for instance by the mathematical description of dynamical systems [51], is critical, when dealing with systems under uncertainties. Finding a unique solution of the system then depends upon the terms **controllability** and **observability** of the system [2, 101], which then ensures in most cases the **identifiability** of the system parameters. Uncertainties, as for example measurement noise or partial observability, then lead to massive problems, such that identifiability of the parameters within an acceptable confidence interval cannot be ensured with commonly well-studied estimation methods.

System identification has thus become of intense interest, as **Hybrid Models** [15, 84] were efficiently applied to overcome model or measurement uncertainties. Hybrid Models in general describe the combination of at least two principles, for instance combining two mathematical methods to solve a certain problem. One possible approach is then to combine mathematical models to solve, for instance least-squares problems, to estimate the parameters of a system and combine these methods with Neural Networks, which have experienced robustness against anomalies of the observed measurements. **Prior knowledge** [85] about the underlying data structure can efficiently be exploited to define a given structure of a Hybrid Model.

The scope of Chapter 5 is therefore to make use of the previously achieved knowledge of this work and efficiently apply the knowledge to define robust estimation models:

- We **know** the structure of the Quarter-Car-Model, therefore we can simplify the estimation problem to the relevant parameters of the model, since some entries of the system matrix only differ in the sign or can be described as the negative sum of neighboured entries.
- We **know** that Neural Networks can solve various data fitting problems for arbitrary dimension of the data, where it is in principle possible to map the input data to any output dimension.
- We **know** that a Semi-Explicit Euler method has been used to generate the data samples of our training and test set. Therefore, as it is commonly used for least-squares problems, we assume that this method describes the exact behaviour of the real world, to have full knowledge about the structure of the optimal model. We can then appropriately evaluate the estimated system responses and the corresponding true observations.

The now following section can therefore be separated in a general problem description of the system identification task, where the properties on parameter identifiability are shown, following the structure of the transfer-function of the system. We then make use of own developed Hybrid Models, on the one hand for a relatively simple, data-driven, parameter estimation problem and on the other hand for an identification problem of the Quarter-Car-Model, where a Hybrid Gauss-Newton method will be derived and explained.

## 5.1 Identifiability of Linear Time-Invariant Systems

We are interested in finding a sufficiently good approximation of the true underlying parameter values of given acceleration data when trying to solve system identification tasks. Regarding a coupled dynamical system as the Quarter-Car-Model, the input control has been described as a spring-force scaled representation of the given road-displacement, divided by the mass of the suspensions, where there are output measurements given by the vertical acceleration profiles of wheel-suspensions, car-body / chassis and occupant. Since the individual equations of the differential system, as described by Eq. (4.102), do not share any of their parameters, as they depend upon the component specific mass, it could therefore occur, given less than full observability of the output measurements, that the entire set of parameters cannot be estimated within an acceptable confidence interval.

There is a wide area in research about system theory [68, 69] and identifiability of dynamical systems, delivering different approaches to find theoretical criteria to validate whether a system is completely identifiable or not. Therefore, the terms controllability and observability have become of immense interest since both properties are necessary conditions on the identifiability of the system. Both terms will be described in the next session, as it is stated by [51]. Further explanations for the identifiability of continuous systems, as discussed by [101] or [2] then define the property of observability and controllability by corresponding matrices and their rank. Nevertheless, it can be shown, that computation of the rank for these matrices is not trivial, since it requires computation of moments of the system matrix. We therefore use a more common approach, considering the transfer-function and the Impulse-Response [2, 9, 51] to find properties on identifiability. This then leads to insights, when a dynamical system, as for instance the Quarter-Car-Model, should be identifiability, nevertheless [87] has stated, that there is a gap between structural identifiability and practical identifiability, since the identifiability also depends on the quality of available data.

### 5.1.1 Linear Systems and Laplace-Transform

We first of all consider **Linear Time-Invariant** (LTI) systems, as they have in principle already been defined by the general differential systems in Chapter 4. So far, we have considered the system matrix and a source vector for the non-homogeneous term. It should nevertheless be obvious that the non-homogeneous term can analogously be expressed by a given control matrix and a source vector of multi-dimensional input signals. Furthermore, we in addition make use of an observability term, since we consider a generalized expression for the system response, which leads to the following expression, as stated by [2, 51, 101].

**Definition 24** (Linear Time-Invariant Systems)
*Let us consider the system*

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{5.1}$$

$$y(t) = Cx(t), \tag{5.2}$$

*for a finite time-horizon $t \in [t_0, t_n] \subset \mathbb{R}$, where $x \in \mathscr{C}^1\left([t_0, t_n], \mathbb{R}^d\right)$ is the state, $u \in \mathscr{C}^0\left([t_0, t_n], \mathbb{R}^p\right)$ is the control and $y \in \mathscr{C}^0\left([t_0, t_n], \mathbb{R}^m\right)$ is the system response. Furthermore, we have the constant **system matrix** $A \in \mathbb{R}^{d \times d}$, the **control matrix** $B \in \mathbb{R}^{d \times p}$ and the **observation matrix** $C \in \mathbb{R}^{m \times d}$ where $d, m, p \in \mathbb{N}$. The system by Eq. (5.1) and Eq. (5.2) is then called a **Linear Time-Invariant System**.*

**Definition 25** (Controllable System [51])
*A Linear Time-Invariant system is **completely controllable** at time $t_0 \in \mathbb{R}$, if we cannot find a separation of the state variables $x \in \mathscr{C}^1\left([t_0, t_n], \mathbb{R}^d\right)$ with $x := (x_1, x_2)^T \in \mathbb{R}^d$, such that the LTI system is algebraically equivalent for $t \geq t_0$ to a system of type*

$$\dot{x}_1(t) = A_{11}x_1(t) + A_{12}x_2(t) + B_1u(t) \tag{5.3}$$

$$\dot{x}_2(t) = A_{22}x_2(t) \tag{5.4}$$

$$y(t) = C_1x_1(t) + C_2x_2(t). \tag{5.5}$$

Controllability, as stated above, in other words mean: It is not possible to find an equivalent coordinate system, in which the state variables are separated into $x_1(t) \in \mathbb{R}^m$ and $x_2(t) \in \mathbb{R}^{d-m}$ for $t \in [t_0, t_n]$ and $d$, $m \in \mathbb{N}$, such that the second group is not affected either by the first group or the inputs of the system.

**Definition 26** (Observable System System [51])
*A Linear Time-Invariant system is **completely observable** at time $t_0 \in \mathbb{R}$, if we cannot find a separation of the state variables $x \in \mathscr{C}^1\left([t_0,t_n],\mathbb{R}^d\right)$ with $x := (x_1,x_2)^T \in \mathbb{R}^d$, such that the LTI system is algebraically equivalent for $t \geq t_0$ to a system of type*

$$\dot{x}_1(t) = A_{11}x_1(t) + B_1u(t) \tag{5.6}$$

$$\dot{x}_2(t) = A_{21}x_1(t) + A_{22}x_2(t) + B_2u(t) \tag{5.7}$$

$$y(t) = C_1x_1(t). \tag{5.8}$$

Observability, similar to controllability, means: It is not possible to find an equivalent coordinate system, in which the state variables are separated into $x_1(t) \in \mathbb{R}^m$ and $x_2(t) \in \mathbb{R}^{d-m}$ for $t \in [t_0,t_n]$ and $d$, $m \in \mathbb{N}$, such that the second group is not affected either by the first group or the inputs of the system. A dynamical system is **identifiable**, if it is completely controllable and completely observable.

Since the above definitions can be used to in general understand the terminology of controllability, observability and identifiability, but give no insights about how to show the property explicitly, we need to make use of another approach to define criteria on the identifiability of systems. Therefore, we introduce the Laplace-Transform [53, 82, 94, 96, 105] to get deeper insights, how to define identifiable systems with the help of the Transfer-Function, which is precisely described below.

**Definition 27** (Laplace-Transform [94])
*Given a function $f \in \mathscr{C}^2\left([0,t_n],\mathbb{R}^d\right)$, the (point-wise) **Laplace-Transform** is defined by*

$$\mathfrak{L}\{f\}(z) := \int_0^\infty e^{-zt} \odot f(t)\mathrm{d}t, \tag{5.9}$$

*where $t \in [0,t_n]$ is the time variable of the original function, $d \in \mathbb{N}$ the dimension of the function mapping and we assume that $\mathfrak{L}\{f\} = 0$ if $t < 0$. Moreover, the complex variable $z \in \mathbb{C}$ is the frequency variable of the Laplace-Transform.*

Given a function $g \in \mathscr{C}^2\left([0,t_n],\mathbb{R}^d\right)$ of the same class, the following properties of the Laplace-Transform hold [94].

- The Laplace-Transform is linear, meaning that for two scalar-valued terms $a,b \in \mathbb{R}$, we have
$$\mathfrak{L}\{af + bg\} = a\mathfrak{L}\{f\} + b\mathfrak{L}\{g\}. \tag{5.10}$$

- It holds for the Laplace-Transform of the first order derivative $f'(t) := \frac{\mathrm{d}}{\mathrm{d}t}f(t)$ for $t \geq 0$, that
$$\mathfrak{L}\{f'\}(z) = z\mathfrak{L}\{f\}(z) - f(0). \tag{5.11}$$

- It holds for the Laplace-Transform of the second order derivative $f''(t) = \frac{\mathrm{d}^2}{\mathrm{d}t^2} f(t)$ that

$$\mathfrak{L}\{f''\}(z) = z^2 \mathfrak{L}\{f\}(z) - zf(0) - f'(0). \tag{5.12}$$

Making use of the above properties of the Laplace-Transform, one can in some cases efficiently use the transformation to the frequency domain, to find a solution to a differential equation, as for instance given in [94], without making use of the techniques stated in Chapter 4, but using other methods like partial fractions to find the original function of the transform.

**Example 17**
*Consider a non-homogeneous ordinary differential equation of the following type*

$$\ddot{y}(t) - y(t) = e^{5t}$$

*for a time variable $t \in [t_0, t_n] \subset \mathbb{R}$ and $y \in \mathscr{C}^2([t_0, t_n], \mathbb{R})$ with $t_0 = 0$ and initial conditions $y(0) = 0$, $\dot{y}(0) = 0$. Then, making use of the following equation with $f(t) = e^{at}$,*

$$\mathfrak{L}\{f\}(z) = \int_0^\infty e^{-zt} e^{at} \mathrm{d}t = \frac{1}{z-a} \tag{5.13}$$

*if $\mathfrak{Re}(z) > \mathfrak{Re}(a)$ for $a \in \mathbb{C}$, one obtains by applying the Laplace-Transform on the l.h.s. as well as on the r.h.s., that*

$$\begin{aligned} & \mathfrak{L}\{\ddot{y}\}(z) - \mathfrak{L}\{y\}(z) = \frac{1}{z-5} \\ \Leftrightarrow \quad & z^2 \mathfrak{L}\{y\}(z) - \mathfrak{L}\{y\}(z) = \frac{1}{z-5} \\ \Leftrightarrow \quad & \mathfrak{L}\{y\}(z) = \frac{1}{(z-5)(z^2-1)}. \end{aligned}$$

*Using partial fractions, we use the approach*

$$\mathfrak{L}\{y\}(z) = \frac{c_1}{z-5} + \frac{c_2}{z-1} + \frac{c_3}{z+1},$$

*with constants $c_1, c_2, c_3 \in \mathbb{C}$. Obviously, it needs to hold that*

$$1 = (z^2-1)c_1 + (z-5)(z+1)c_2 + (z-5)(z-1)c_3,$$

*which then results to the solution*

$$\mathfrak{L}\{y\}(z) = \frac{1}{24(z-5)} - \frac{3}{24(z-1)} + \frac{2}{24(z+1)}$$

*and since we have already considered Eq. (5.13), we conclude that*

$$y(t) = \frac{1}{24} e^{5t} - \frac{3}{24} e^{t} + \frac{2}{24} e^{-t}. \tag{5.14}$$

*It should easily be verifiable that $y(t)$ is indeed a solution to the above defined non-homogeneous second order ordinary differential equation.*

Applying the Laplace-Transform to a LTI-system, as stated in Definition 24, where we now assume to have no system or observation noise and $t_0 = 0$, $x(0) = 0$ and $\dot{x}(0) = 0$, we directly obtain that

$$\begin{aligned} & \mathfrak{L}\{\dot{x}\}(z) = A\mathfrak{L}\{x\}(z) + B\mathfrak{L}\{u\}(z) \\ \Leftrightarrow \quad & \mathfrak{L}\{x\}(z) = (zI - A)^{-1} B\mathfrak{L}\{u\}(z), \end{aligned}$$

where for the identity matrix $I \in \mathbb{R}^{d\times d}$, the matrix term $(zI - A)$ is assumed to be invertible.

The **Transfer-Function** [2] of the system is then given by

$$\mathfrak{L}\{y\}(z) = C\mathfrak{L}\{x\}(z) = C(zI - A)^{-1} B\mathfrak{L}\{u\}(z). \tag{5.15}$$

Moreover, the input-output relation, or **Impulse-Response** of the dynamical system, can then be defined for a LTI-system by

$$G_1(z; \{A, B, C\}) := C(zI - A)^{-1} B \tag{5.16}$$

as it is also stated for instance in [2, 9, 101].

Since we want to find criteria on the identifiability of the system parameters for coupled dynamical models, we need to analyse the invariance of the impulse-response for LTI-systems [101] for a specific transformation. Let us therefore w.l.o.g. assume, that we have a specified transformation of an LTI system with square matrices $A$, $B$, $C \in \mathbb{R}^{d\times d}$ with $d \in \mathbb{N}$ and a regular transformation matrix $T \in \mathbb{R}^{d\times d}$, such that we consider

$$\{A, B, C\} \mapsto \left\{T^{-1}AT, T^{-1}B, CT\right\}. \tag{5.17}$$

One can then show, that the Impulse-Response $G_1(z; \{A, B, C\})$ is invariant, for instance under a transformation as given by Eq. (5.17), due to

$$\begin{aligned} G_1\left(z; \left\{T^{-1}AT, T^{-1}B, CT\right\}\right) &= CT\left(zI - T^{-1}AT\right)^{-1} T^{-1}B \\ &= CT\left(T^{-1}(zI - A)T\right)^{-1} T^{-1}B \\ &= CTT^{-1}\left(T^{-1}(zI - A)\right)^{-1} T^{-1}B \\ &= CTT^{-1}(zI - A)^{-1} TT^{-1}B \\ &= C(zI - A)^{-1} B \\ &= G_1(z; \{A, B, C\}). \end{aligned}$$

Since we have considered a three dimensional dynamical system of second order ordinary differential equations for the Quarter-Car-Model, we need to apply the herein shown results to such systems in the following section.

### 5.1.2 Systems of Second Order and Impulse Response

Similar to Definition 24, we can define a second order system with the help of the following definition.

**Definition 28** (Second Order Linear-Time Invariant System)
*We consider the **second order dynamical system***

$$\ddot{x}(t) = D\dot{x}(t) + Sx(t) + Bu(t) \tag{5.18}$$

$$y(t) = Cx(t), \tag{5.19}$$

*for a finite time-horizon $t \in [t_0, t_n] \subset \mathbb{R}$, a state function $x \in \mathscr{C}^2\left([t_0,t_n], \mathbb{R}^d\right)$, a control $u \in \mathscr{C}^0\left([t_0,t_n], \mathbb{R}^p\right)$ and the system response $y \in \mathscr{C}^0\left([t_0,t_n], \mathbb{R}^m\right)$ with $d,p,m \in \mathbb{N}$. Further, let $D \in \mathbb{R}^{d\times d}$ be the damping-coefficient matrix, $S \in \mathbb{R}^{d\times d}$ the spring-coefficient matrix and similar to the first order system, let us denote by $B \in \mathbb{R}^{d\times p}$ the control matrix and by $C \in \mathbb{R}^{m\times d}$ the observation matrix.*

Let us w.l.o.g. assume that there is a transformation for a second order dynamical system, given by

$$\{D,S,B,C\} \mapsto \left\{T^{-1}DT, T^{-1}ST, T^{-1}B, CT\right\}, \tag{5.20}$$

where again, $T \in \mathbb{R}^{d\times d}$ is a regular matrix with inverse $T^{-1}$. In this case, we again have the damping-matrix $D \in \mathbb{R}^{d\times d}$, the spring-matrix $S \in \mathbb{R}^{d\times d}$, the control matrix $B \in \mathbb{R}^{d\times d}$ as well as the observation matrix $C \in \mathbb{R}^{d\times d}$. It is obvious that the **impulse-response of the second order system**, is also invariant under the above transformation, yielding that

$$G_2\left(z; \{D,S,B,C\}\right) := C\left(z^2 I - zD - S\right)^{-1} B \tag{5.21}$$

$$= CT\left(z^2 I - zT^{-1}DT - T^{-1}ST\right)^{-1} T^{-1}B \tag{5.22}$$

$$= G_2\left(z; \left\{T^{-1}DT, T^{-1}ST, T^{-1}B, CT\right\}\right). \tag{5.23}$$

The above invariance of the impulse response shows that for a transformation matrix $T \in \mathbb{R}^{d\times d}$, the spring-matrix $S \in \mathbb{R}^{d\times d}$ as well as the damping matrix $D \in \mathbb{R}^{d\times d}$ are corrupted in an equal sense, such that $\{D,S\} \mapsto \left\{T^{-1}DT, T^{-1}ST\right\}$. Therefore, if one wants to find criteria on the identifiability of the system parameters, as shown in the next section, it is sufficient to either compare the entries of $D$ and $\tilde{D} := T^{-1}DT$ or $S$ and $\tilde{S} := T^{-1}ST$. If $D = \tilde{D}$ and therefore also $S = \tilde{S}$, we can assume that the system parameters can uniquely be determined, independent on a transformation $T$, if the matrices are not equal, we can conclude that there are different parameter matrices, which give the same impulse-response under distinguishable realizations of the system parameters.

### 5.1.3 Identifiability of Observable Systems

We want to show explicit criteria on the structural identifiability for the parameters of the Quarter-Car-Model for partially observable systems. Therefore, the damping-coefficient matrix $D$ and the spring-coefficient matrix $S$ can be defined by

$$D = \begin{bmatrix} -d_1 & d_1 & 0 \\ d_2 & -(d_2+d_3) & d_3 \\ 0 & d_4 & -d_5 \end{bmatrix}, \quad S = \begin{bmatrix} -s_1 & s_1 & 0 \\ s_2 & -(s_2+s_3) & s_3 \\ 0 & s_4 & -s_5 \end{bmatrix}. \tag{5.24}$$

Due to the hierarchical structure of the Quarter-Car-Model, the road-profile does directly affect only the wheel-suspension's state, such that we can assume that with $B \in \mathbb{R}^{3\times 3}$, we get

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.25}$$

Since the control directly or indirectly affects all states of the system, the system cannot be decoupled, such that there is a group of states not affected by the control or the remaining group of states. Following this definition of [51] as stated in the beginning of this section, we can conclude that the Quarter-Car-Model is controllable. Therefore, the identifiability property of the system parameters, obviously depends upon the observation property of the system. We therefore need to make assumptions to the realization of $C$ in a similar way as for the matrix $B$ with the help of the now stated examples.

Assume that the Quarter-Car-Model is invariant under the transformation $T \in \mathbb{R}^{3\times 3}$, such that it does not affect the output with $CT = C$ and the input with $T^{-1}B = B$ as given by the impulse-response. If the similarity matrices, as stated in the impulse response, are different from the original matrices, namely $\tilde{D} = T^{-1}DT \neq D$ and $\tilde{S} = T^{-1}ST \neq S$, then the system is not uniquely identifiable, since $\{D,S\}$ differs from$\{\tilde{D},\tilde{S}\}$.

Since we want to investigate properties on the identifiability of the Quarter-Car-Model, we have $d = 3$ as row and column size for the transformation matrix. Let us therefore in general assume that the transformation matrix and its inverse are given by the general expression

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}, \quad T^{-1} = \begin{bmatrix} \tau_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & \tau_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & \tau_{33} \end{bmatrix}. \tag{5.26}$$

Using the general description of the transformation and its inverse is helpful to step by step investigate the structure in the now following examples.

**Example 18** (Full-Observability)
*Let us assume that the observation matrix is given by the identity*

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.27}$$

*which means that all states of the system can be observed from the impulse response. Since we assume that $T^{-1}B = B$, it follows from*

$$T^{-1}B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \tau_{33} \end{bmatrix} \tag{5.28}$$

*that $\tau_{33} = 1$. We have further assumed that $CT = C$. It is obvious that there is only one possible transformation, since $C$ is equal to the identity matrix $I \in \mathbb{R}^{3\times3}$ and from $IT = I$ it follows that $T = I$. Therefore, for the inverse, it holds that $T^{-1} = I$ and for the transformation of the parameter matrices, we have $T^{-1}DT = IDI = D$ and $T^{-1}ST = ISI = S$.*

*In summary, this investigation yields, that if one has full-observability of all states of the system, then there is only the identity matrix as possible transformation which satisfies the invariance property of the impulse-response. Since the identity matrix does not change the parameters of the damping-matrix and the spring-matrix, there is only one possible realization for the parameters. Therefore, the system should be identifiable from the full-observation property. As we have investigated that the system is completely controllable and completely observable, the identifiability of the system follows from the theoretical statements as given by [2, 51, 101].*

**Example 19** (One Unobservable State)
*Let us assume that the observation matrix is given by*

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{5.29}$$

*which means that all states, but the wheel-suspension state, of the system can be observed from the impulse response. Since we assume that $T^{-1}B = B$, it follows from*

$$T^{-1}B = \begin{bmatrix} 0 & 0 & \tau_{13} \\ 0 & 0 & \tau_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix} \tag{5.30}$$

*that $\tau_{13} = 0$, $\tau_{23} = 0$ and $\tau_{33} = 1$. We have further assumed that $CT = C$. It is therefore obvious from computation of the transformation*

$$CT = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 0 \end{bmatrix} \tag{5.31}$$

*that $t_{11} = t_{22} = 1$ and $t_{12} = t_{13} = t_{21} = t_{23} = 0$. Then, one can finally observe from $TT^{-1} = I$ that $\tau_{11} = \tau_{22} = 1$, $\tau_{12} = \tau_{13} = \tau_{21} = \tau_{23} = 0$, $t_{33} = \tau_{33} = 1$ and $\tau_{31} = -t_{31}$, $\tau_{32} = -t_{32}$, such that we get the structure of the transformation matrices by*

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}, \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_{31} & -t_{32} & 1 \end{bmatrix}. \tag{5.32}$$

*Computation of the transformed damping-matrix $\tilde{D} = T^{-1}DT$, then yields, that*

$$\tilde{D} = \begin{bmatrix} -d_1 & d_1 & 0 \\ d_2 + t_{31}d_3 & -d_2 + (t_{32} - 1)d_3 & d_3 \\ t_{31}d_1 - t_{32}d_2 - t_{31}t_{32}d_3 - t_{31}d_5 & -t_{31}d_1 + t_{32}d_2 + (t_{32} - t_{32}^2)d_3 + d_4 - t_{32}d_5 & -t_{32}d_3 - d_5 \end{bmatrix}, \tag{5.33}$$

*which shows that $\tilde{D} \neq D$ and it is trivial that for $\tilde{S} = T^{-1}ST$, we have the same type of transformation structure.*

*In summary, this investigation yields, that one can choose the entries of the transformation, namely $t_{31} \in \mathbb{R}$ and $t_{32} \in \mathbb{R}$ arbitrarily, nevertheless all possible transformations satisfy $CT = C$, and $T^{-1}B = B$, due to having only partial observability of the system. Therefore, we can suggest that if we have less than full-observability of the system, that it is then not possible to uniquely determine the parameters of the model, since the impulse-response is equal for the true parameter matrices $\{D, S\}$ and all of the possible transformations $\{T^{-1}DT, T^{-1}ST\}$. It is obvious that there are an infinite number of possible transformations for this case.*

**Example 20** (Two Unobservable States)
*Let us assume that the observation matrix is given by*

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{5.34}$$

*which means that only the state of the occupant of the system can be observed from the impulse response. Since we assume that $T^{-1}B = B$, it follows from*

$$T^{-1}B = \begin{bmatrix} 0 & 0 & \tau_{13} \\ 0 & 0 & \tau_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix} \tag{5.35}$$

*that $\tau_{13} = 0$, $\tau_{23} = 0$ and $\tau_{33} = 1$. We have further assumed that $CT = C$. It is therefore obvious from*

$$CT = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.36}$$

*that $t_{11} = 1$ and $t_{12} = t_{13} = 0$. It obviously follows from $TT^{-1} = I$ that we then require $\tau_{11} = 1$ and $\tau_{12} = \tau_{13} = 0$. Furthermore, we can derive the following equations from $T^{-1}T$ and similar from $T^{-1}T$ that*

$$\begin{array}{llcl} (i) & \tau_{21} + \tau_{22}t_{21} & = & 0 \\ (ii) & \tau_{22}t_{22} & = & 1 \\ (iii) & \tau_{31} + \tau_{32}t_{21} + t_{31} & = & 0 \\ (iv) & \tau_{32}t_{22} + t_{32} & = & 0. \end{array}$$

*It is obvious, from $(i) - (iv)$, that $\tau_{22} = \frac{1}{t_{22}}$, $\tau_{21} = -\frac{t_{21}}{t_{22}}$, $\tau_{32} = -\frac{t_{32}}{t_{22}}$ and $\tau_{31} = \frac{t_{32}t_{21} - t_{31}t_{22}}{t_{22}}$ if we assume that $t_{22} \neq 0$. The class of transformations and their corresponding inverses then have the following general structure*

$$T = \begin{bmatrix} 1 & 0 & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}, \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{t_{21}}{t_{22}} & \frac{1}{t_{22}} & 0 \\ \frac{t_{32}t_{21} - t_{31}t_{22}}{t_{22}} & -\frac{t_{32}}{t_{22}} & 1 \end{bmatrix}. \tag{5.37}$$

*It can be verified, without explicitly computing the transformation $\tilde{D} = T^{-1}DT$ in detail, that we have*

$$\tilde{D} = \begin{bmatrix} -d_1 & d_1 & 0 \\ \tilde{d}_{21} & \tilde{d}_{22} & \tilde{d}_{23} \\ \tilde{d}_{31} & \tilde{d}_{32} & \tilde{d}_{33} \end{bmatrix}. \tag{5.38}$$

*Therefore, we can again notice that the first row of the parameter matrix is not transformed, due to the condition of partial observability. Nevertheless, we recognize that in contrast to the previous example, we have not only two transformation variables to choose a specific acceptable transformation, but four variables, namely $t_{21} \in \mathbb{R}$, $t_{22} \in \mathbb{R} \setminus \{0\}$, $t_{31} \in \mathbb{R}$ and $t_{32} \in \mathbb{R}$. Thus we have again an infinite number of possible transformation matrices, which satisfy the given observability and controllability condition of the example. We can therefore conclude that for this specific problem, all parameters cannot be uniquely determined from the impulse-response of the system.*

Given the above stated examples, we can conclude that for partially observable systems, the parameters cannot be uniquely determined since there exist a family of possible parameter matrices, induced by transformations of the shown types, which give the same impulse-response of the system. Nevertheless, we have seen that for all cases, the first row of the damping-matrix and the spring-matrix, are invariant under all possible transformations. As a consequence, we now want to deal with the following problems:

- The parameters of the occupant's differential equation should be identifiable. We are therefore going to make investigations from several experiments, if this is indeed the case.

- The parameters of the Quarter-Car-Model are not all identifiable if we only have partial observability of the system. We are therefore going to make investigations from several experiments, where we change the grade of observability.
- We are going to develop Hybrid Models, including several Neural Networks structures, that overcome the problems of the above stated experiments.

The now following section therefore deals with several investigations, we have done as far as parameter estimation for optimal systems and for systems under uncertainties is concerned. A more specific description of each problems is given in the corresponding problem specification.

## 5.2 Parameter Estimation for Ordinary Differential Equations

We have observed, that for the identifiability condition of the Quarter-Car-Model parameters, if the state of the occupant is observable, that also the corresponding parameters of the ordinary differential equation, are not varied for a transformation of the system matrix. Regarding the training and test dataset, that we have created in Chapter 4, denoted by

$$\mathcal{S} = \{((a_m, v_m, q_m, u_m), p_m)\}_{m=1}^{N_S}$$

for the training data with $N_S = M \cdot N_{train} = 100 \cdot 340 = 34000$ samples and

$$\mathcal{T} = \{((a_m, v_m, q_m, u_m), p_m)\}_{m=N_S+1}^{N_S+N_T}$$

for the test data with $N_T = M \cdot N_{test} = 100 \cdot 170 = 17000$ samples, resulting from the number of generated random road-profiles and random parameter realization for each set. Resuming the terminology of the dataset elements, we herein give a short definition as follows:

- $a_m \in \mathbb{R}^{N \times 3}$ is the discrete acceleration of the Quarter-Car-Model for the three observable components, with length $N \in \mathbb{N}$. The length is specified for each experiment later on.
- $v_m \in \mathbb{R}^{N \times 3}$ is the discrete velocity of the Quarter-Car-Model with the same dimension as given by the definition of the acceleration profile $a_m$.
- $q_m \in \mathbb{R}^{N \times 3}$ is the discrete state or displacement of the Quarter-Car-Model.
- $u_m \in \mathbb{R}^{N}$ is the generated road-profile to the corresponding displacement, velocity and acceleration.
- $p_m \in \mathbb{R}^{10}$ is the vector of relevant Quarter-Car-Model parameters, such that we can simplify the problem to the estimation of the parameter vector, instead of estimation of the system matrix $A_m \in \mathbb{R}^{6 \times 6}$. As we already know, the system matrix contains

four blocks, namely the damping-coefficient matrix $D_m \in \mathbb{R}^{3\times 3}$, the spring coefficient matrix $S_m \in \mathbb{R}^{3\times 3}$, the identity matrix $I \in \mathbb{R}^{3\times 3}$ and the null matrix $O \in \mathbb{R}^{3\times 3}$, resulting from reduction of the second order system to a first order system. As it is inefficient to estimate the identity and the null matrix and since we already know that each, the damping- and spring-coefficient matrices, contain 5 distinguishable parameters, it is efficient to estimate these 10 parameters, given by $p_m$. It is then possible to switch from $p_m$ to $A_m$ and vice versa, using a bijection $p_m \leftrightarrow A_m$.

- $m \in \mathbb{N}$ describes the index of the training and test samples. We have $m \in \{1, 2, ..., N_S\}$ for the training samples and $m \in \{N_S + 1, N_S + 2, ..., N_S + N_T\}$ for the test samples.

For the sake of simplicity, let us assume that the parameter vector $p_m$ can be described by

$$p_m := \left(p_{m;1}, p_{m;2}, p_{m;3}, p_{m;4}, p_{m;5}, p_{m;6}, p_{m;7}, p_{m;8}, p_{m;9}, p_{m;10}\right)^T, \tag{5.39}$$

where the definition of the elements $p_{m;i}$ for $i \in \{1, 2, ..., 10\}$ follows the random variation of the mean parameters of Eq. (4.103) for sample $m \in \{1, 2, ..., N_S + N_T\}$. Further, let us define the scaling coefficient of the road displacement by $p_{m;u} := p_{m;10} - p_{m;9}$. In a similar way, the multi-dimensional acceleration profile $a_m$ can be distinguished in $a_{m;3} \in \mathbb{R}^N$ for the occupant, $a_{m;2} \in \mathbb{R}^N$ for the chassis and $a_{m;1} \in \mathbb{R}^N$ for the wheel-suspensions. Analogously, one can define the individual velocities $v_{m;i}$ and displacements $q_{m;i}$ for $i \in \{1, 2, 3\}$ such that the acceleration of the occupant for each sample follows the equation

$$a_{m;3} = -p_{m;1}\left(v_{m;3} - v_{m;2}\right) - p_{m;2}\left(q_{m;3} - q_{m;2}\right) \in \mathbb{R}^N, \tag{5.40}$$

which corresponds to Eq. (2.54) of the Quarter-Car-Model. We can therefore observe, that the acceleration of the occupant depends upon the damping parameter $p_{m;1}$, the spring parameter $p_{m;2}$, the velocity $v_{m;3}$ and the displacement $q_{m;3}$ of the occupant and the velocity $v_{m;2}$ and the displacement $q_{m;2}$ of the chassis. The effect of the road-displacement $u_m$ is implicitly given by the velocity and displacement of the chassis, as it is not explicitly shown in Eq. (5.40). If we therefore consider this sub-system of the Quarter-Car-Model for the parameter estimation problem, the following setting is used for a first series of experiments.

- The input (control) of the system can be described by $q_{m;2}$, $v_{m;2}$ and $a_{m;2}$ as the chassis is coupled to the occupant's seat for the Quarter-Car-Model.
- The system is described by the parameters $p_{m;1}$ and $p_{m;2}$.
- The output (system response) is given by $q_{m;3}$, $v_{m;3}$ and $a_{m;3}$.

The problem of the now following experiments for the section, can be described with the help of Figure 5.2: We want to estimate the parameters of the described sub-system, given by $p_{m;1}$ and $p_{m;2}$ for each sample of the training data $\mathcal{S}$ and the test data $\mathcal{T}$, using a data-driven approach. Therefore, a Convolutional Neural Network is used to process the control and the system response to predict the unknown parameter values of the system. Several approaches will be investigated, using a labelled as well as an unlabelled training method, where we

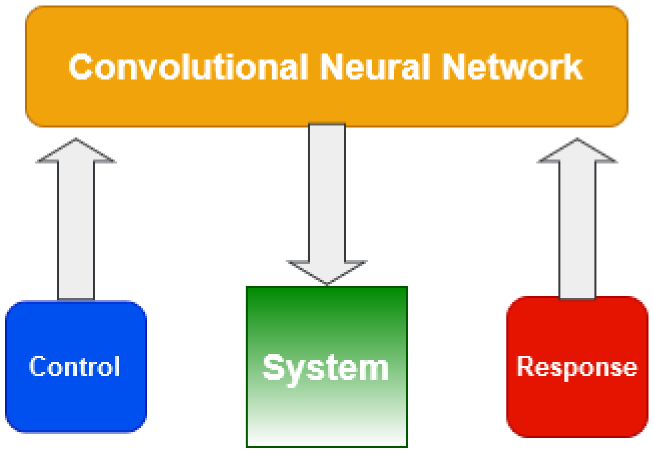


FIGURE 5.2: Data driven parameter estimation: Control and System Response are processed via a Convolutional Neural Network to predict the values of the system parameters of an ordinary differential equation.

compare the trained Neural Networks w.r.t. the generalization error and robustness against noise. All experiments of Chapter 5 were executed, using an **NVIDIA Tesla V100 SMX2 32GB** graphics card.

### 5.2.1 Experiment 1: Data Driven Parameter Estimation

#### Problem

We assume that we have access to the reduced training dataset

$$\mathcal{S}^{(1)} = \{((a_{m;3}, a_{m;2}), (p_{m;1}, p_{m;2}))\}_{m=1}^{N_S},$$

and a reduced test dataset

$$\mathcal{T}^{(1)} = \{((a_{m;3}, a_{m;2}), (p_{m;1}, p_{m;2}))\}_{m=N_S+1}^{N_S+N_T},$$

which means that the acceleration profile $a_{m;3} \in \mathbb{R}^N$ of the occupant, as well as the acceleration profile $a_{m;2} \in \mathbb{R}^N$ of the chassis are observable and the corresponding labels are given by the damping-coefficient $p_{m;1} \in \mathbb{R}$ and the spring-coefficient $p_{m;2} \in \mathbb{R}$. The parameters are characteristic for the acceleration of the occupant, as can be verified from Eq. (5.40). As it has been defined in Chapter 4, we consider a training dataset with $N_S = MN_{train} = 100 \cdot 340 = 34000$ individual samples and a test dataset with $N_T = MN_{test} = 100 \cdot 170 = 17000$ samples.

The **problem**, we want to investigate is: **Given the acceleration profiles of the defined training dataset, can the unknown parameters of the occupant's acceleration be appropriately estimated with a data driven model, as for instance a Neural Network?**

**Method**

For the sake of simplicity, we denote by $a_3 \in \mathbb{R}^N$ with $N \in \mathbb{N}$ a general acceleration profile of the occupant of the training data. Analogously, $a_2 \in \mathbb{R}^N$ is the acceleration profile for the chassis, $p_1 > 0$ the damping-parameter and $p_2 > 0$ the spring-parameter. We now want to solve the above described problem with the help of a Convolutional Neural Network, which processes the observable acceleration profiles, targeting at the values of the underlying true parameter labels. More precisely, we can define the network with the help of the general function description

$$\begin{aligned} f_\theta^{(1)} &: \mathbb{R}^{N\times 2} \to \mathbb{R}^2 \\ (a_3, a_2) &\mapsto \begin{pmatrix} \tilde{p}_1(\theta) \\ \tilde{p}_2(\theta) \end{pmatrix}, \end{aligned}$$

where we can define the prediction for the damping-coefficient and the spring-coefficient in terms of the Neural Network's output as

$$\tilde{p}_i(\theta) = f_\theta^{(1)}\left((a_3, a_2)\right)_i \in \mathbb{R}$$

with index $i = 1$ for the damping-coefficient and $i = 2$ for the spring-coefficient. As it has been described in Chapter 3, the network parameters, which should be optimized in the course of the training process, are denoted by $\theta$.

Finally, we define the loss-function of the Neural Network training for all samples $(a_{m;3}, a_{m;2}) \in \mathcal{S}^{(1)}$ as follows:

$$\min_\theta \frac{1}{N_S} \sum_{m=1}^{N_S} \mathscr{L}\left(\theta, (a_{m;3}, a_{m;2}), (p_{m;1}, p_{m;2})\right) = \frac{1}{N_S} \sum_{m=1}^{N_S} \sum_{i=1}^{2} |p_{m;i} - \tilde{p}_{m;i}(\theta)|, \tag{5.41}$$

where $N_S = 34000$ is the size of the training dataset.

The loss-function is minimized, using the following training specifications:

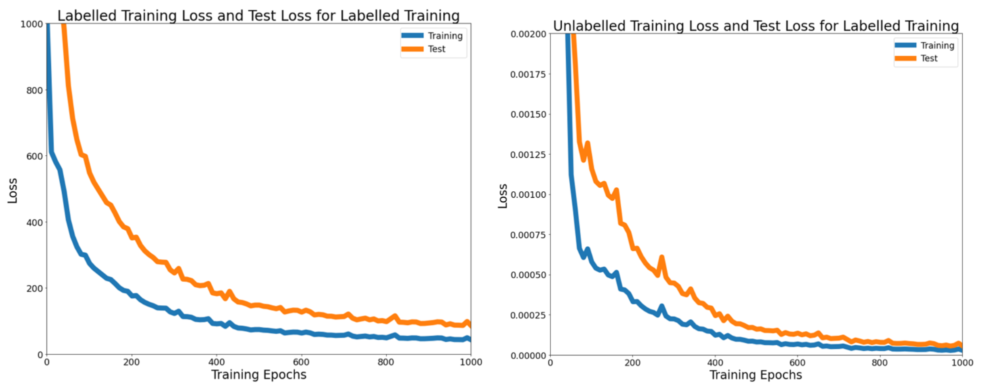
- Optimization method: Adaptive Momentum (ADAM) Optimization
- Learning-rate: $\eta = 0.005$
- Batch-size: $B = 1000$
- Series-length of the input samples: $N = 500$
- Training-epochs: $E = 1000$

Since the size of the training set, according to the data generation description of Chapter 4, is $N_S = 34000$, one training epoch contains $\frac{N_S}{B} = 34$ optimization steps. Having a total number of $E = 1000$ epochs, then results in 34000 optimization steps for the total training process. ADAM optimization [33] has shown to be one of the most successful optimization techniques for training of Deep Neural Networks, using an averaged Gradient-Descent step, compared to the state-of-the-art Stochastic Gradient Descent (SGD) method [33]. After finishing the optimization process, we want to compare the performance of the pre-trained network denoted by $f^{(1)}_{\theta_\alpha}$, comparing the performance for the data samples and corresponding labels of $\mathcal{S}^{(1)}$ with $N_S = 34000$ pairs and $\mathcal{T}^{(1)}$ with $N_T = 17000$ pairs. With $\theta_\alpha \in \Theta$, we denote the realization of the network parameters after $K = E \cdot \frac{N_S}{B} = 34000$ optimization steps, therefore $\theta_\alpha = \theta^K$.

A detailed description of the network's layer structure is given by Figure *B*.1 and the source code is shown by Figure *A*.1 and Figure *A*.2. Implementation of the training is given by Figure *A*.3 and Figure *A*.4.

## Results

The first investigation we want to make, is to analyse the minimization of the loss-function described in the previous section for a total number of 1000 training epochs. If the loss-function has been chosen appropriately, this should be verifiable from a visualization of the loss-function's shape.



(a) Loss in Parameter Space.

(b) Loss in Data Space.

FIGURE 5.3: Shape of the loss-functions for labelled training approach. The horizontal axis shows the number of training epochs. The loss-functions have been evaluated for the entire datasets at each epoch. The vertical axis shows the value of the loss-function, as it has previously been defined. The shape of the training loss is visualized in blue, the test loss in orange. We differ between parameter loss in Figure 5.3*a* and data loss in Figure 5.3*b*. The parameter loss describes the value of the labelled loss-function in **Experiment 1**, the data loss describes the value of the unlabelled loss-function as defined in **Experiment 2**.

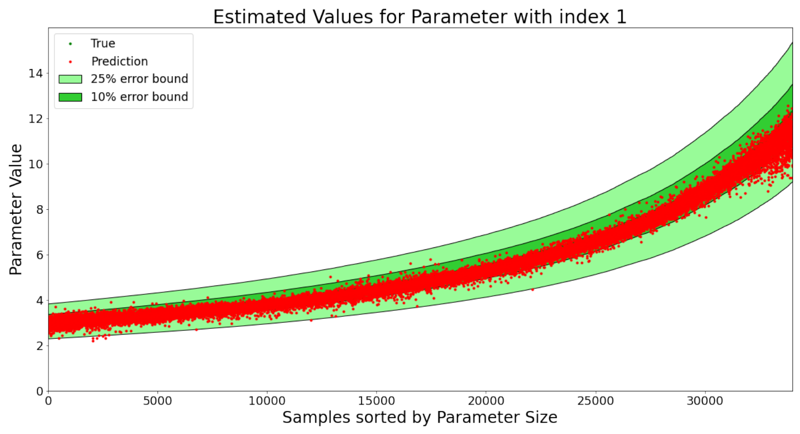
The shape of the loss-functions are shown in Figure 5.3. As it is described in the caption of the function, we observe the parameter loss on the l.h.s., as well as the data loss on the r.h.s. The data loss describes the distance in the data space, where the estimated parameter values are used to reconstruct the acceleration profile of the occupant. Then the difference between predicted profile and true profile can be estimated. This function will more precisely be defined in **Experiment 2**. The parameter loss corresponds to the value of the loss-function described in the method of this experiment. We can recognize that for both losses, the optimization is done appropriately. The loss-function is minimized, the longer we train our network. Furthermore, it seems that there is still the potential to minimize the loss after having trained the network for 1000 epochs. Nevertheless, this training procedure requires a significant amount of time. We can further observe, that the value of the test loss is larger than the value of the training loss. The network therefore probably has a higher accuracy on the training data, but since the shape of training loss and test loss correlate, we should have an acceptable performance for the test samples as well.

We want to analyse the numerical results of this first problem with the help of the following figures, summarizing the predictions of the trained Neural Network $f_{\theta_\alpha}^{(1)}$ for all training and test samples. Furthermore, we can have a more detailed view on the error distribution, observing the results given in the table described at the end of this experiment.

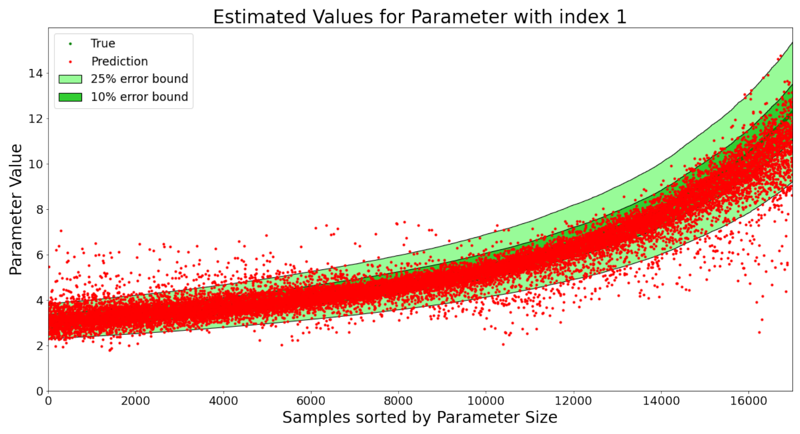
The results in Figure 5.4*a* show, that nearly all training predictions for parameter $p_1$ lie within the 10% error range, while a minor set of the estimates can be located in the larger 25% range. Furthermore, it seems that the parameters of the Neural Network have been adjusted to underestimate the parameter values, since the red point cloud does not have its center in the middle of the green area. As far as the test results are concerned, we can observe in Figure 5.4*b*, that there is a higher mean deviation of the predictions, compared to those of the training set. Additionally, we recognize a tendency, that smaller parameters are more probable to be overestimated, where in contrast larger parameters are underestimated. We can even recognize relative deviations of more than 25% on the plot for the test samples.

The results of the second parameter estimate in Figure 5.5 are mainly comparable to those of Figure 5.4. One difference can be observed in Figure 5.5*a*, where we now have more precise predictions within the 10% boundary, broadly lying in the center of the green area. Again, the test results in Figure 5.5*b* show larger deviations from the true underlying labels, where we also have a tendency of overestimation for small values and underestimation for large values.

The observations of Figure 5.4 and Figure 5.5 can be verified using Table 5.1: We have observed that for both cases the most training predictions lie within the error bound of 25%. We can recognize from the last column of the table that nearly 0% of the training estimates have a deviation of more than 25%. Furthermore, we have maximum relative errors of 32% for $p_1$ and 26% for $p_2$, which again verifies the previously made investigations. We have further recognized, that the predictions for $p_1$ are not centred in the green area, in contrast to the training predictions of $p_2$. This can also be recognized from the values of $0-1\%$ deviation and $0-5\%$ deviation. For the second parameter, we can observe that more than 40% of the training estimates are below 1% relative deviation, while for the first parameter, we have less than 8%. The same holds for the 5% area, where we have approximately 94% for parameter $p_2$ and around 73% for parameter $p_1$.



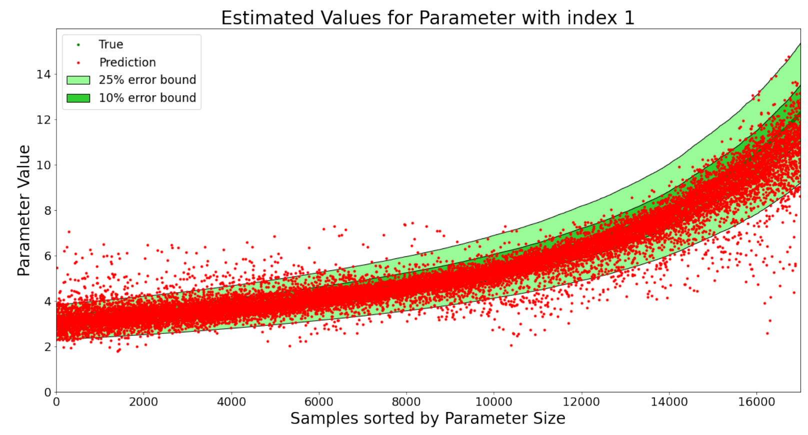
(a) Training results for estimation of parameter 1 using a labelled objective without noise.



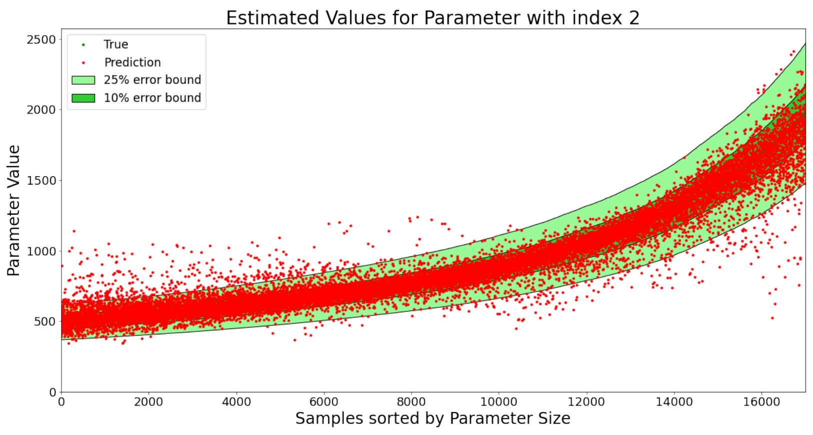
(b) Test results for estimation of parameter 1 using a labelled objective without noise.

FIGURE 5.4: Experiment 1: Labelled estimation of parameter $p_1$. The results are sorted on the horizontal axis, from 1 to 34000 or 17000, where 1 corresponds to the sample with the smallest parameter value and contrarily, 34000 (training set) and 17000 (test set) to the largest parameter value. The vertical axis describes the value of the parameter, which corresponds to the sorted indices on the horizontal axis. The true label values are shown in **green** and the predictions of the Neural Network are shown in **red**. Furthermore, a relative deviation of $0-10\%$ is visually shown with the help of the **green** area, where a relative deviation of $10-25\%$ is shown by the **lime green** area. Figure 5.4*a* shows the results for the training set, where Figure 5.4*b* shows the results for the test set.

For the test results of this first experiment, we can observe more homogeneous observation for both parameters. Summarizing, it is obvious that the performance of the trained Neural Network $f_{\theta_\alpha}$ on the test samples is worse compared to the training samples. Although 95% of the parameter estimates lie within the 25% error range, we ave around $4.1-4.7\%$ predicted values above 25%. In addition the maximum relative deviation is approximately 127% for the test data in contrast to 32% for the training data. It can be verified from all columns of the table, that the prediction quality on the test data is below the training performance.



(a) Training results for estimation of parameter 2 using a labelled objective without noise.



(b) Test results for estimation of parameter 2 using a labelled objective without noise.

FIGURE 5.5: Experiment 1: Labelled estimation of parameter $p_2$. The description of the plots is analogous to Figure 5.4, where we now consider the results of the second parameter of the acceleration, denoted by $p_2$.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.32057 | 0.04095 | 0.02617 | 0.07829 | 0.72485 | 0.96859 | 0.99974 | 0.00026 |
| Training | 2 | 0.26229 | 0.01805 | 0.01890 | 0.40865 | 0.93985 | 0.99194 | 0.99997 | 0.00003 |
| Test | 1 | 1.26533 | 0.08186 | 0.08818 | 0.09482 | 0.44076 | 0.73388 | 0.95265 | 0.04735 |
| Test | 2 | 1.27169 | 0.07241 | 0.08651 | 0.12129 | 0.52582 | 0.78576 | 0.95906 | 0.04094 |

TABLE 5.1: Experiment 1: Error distribution of labelled parameter estimation. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. **"Data"**: Differ between training and test dataset. **"Parameter"**: Differ between parameter $p_1$ and $p_2$. **"Max Error"**: Shows the maximum relative deviation between predicted and true parameter value of the entire dataset. **"Mean Error"**: Shows the mean relative error of the entire dataset. **"Std Error"**: Shows the mean standard deviation of the entire dataset. **"0-1%"**: Percentage of predictions below 1% relative deviation. **"0-5%"**: Percentage of predictions below 5% relative deviation. **"0-10%"**: Percentage of predictions below 10% relative deviation. **"0-25%"**: Percentage of predictions below 25% relative deviation. **">25%"**: Percentage of predictions above 25% relative deviation.

We have chosen a convolutional layer structure as Neural Network architecture, since it easily processes the acceleration data of the occupant and the chassis by using 1D convolutions, as it has been described by Eq. (3.27) and Eq. (3.28). Furthermore, Convolutional Neural Networks have shown to generalize well, since they naturally require less trainable parameters, due to weight sharing of the convolutional kernels, compared to other structures like Fully-Connected Neural Networks.

Comparable experiments to adjust the optimization algorithm and the hyperparameters are not shown here, but the training configuration has shown to perform well, when trying to solve the training problem. In addition, we have included the acceleration of the chassis as a source term for the road-displacement, since we have also made several investigations in which a Neural Network, mapping solely from occupant's acceleration to parameter space, is more sensitive with respect to the input data.

**Summary**

Resuming the above results, investigated in this first experiment, we can conclude that it is in principle possible to estimate the parameters of the occupant's acceleration from discrete data samples of a specific time horizon. Nevertheless, we can further recognize that the deviation of the estimated parameter values for the test dataset is above the deviation for the training data. The model, as it is described here, therefore leads to overfitting of the training data. A variation of the network architecture, meaning increasing the depth of the network or the number of parameters for each layer, does not significantly lead to a more precise prediction of the training labels. In contrast, a more shallow network with less parameters does not necessarily lead to a lower generalization error.

### 5.2.2 Experiment 2: Hybrid Training Methods

**Problem**

We assume that we have access to the reduced training dataset

$$\mathcal{S}^{(2)} = \{((a_{m;3}, a_{m;2}, v_{m;3}, v_{m;2}, q_{m;3}, q_{m;2}))\}_{m=1}^{N_S} \tag{5.42}$$

and a reduced test dataset

$$\mathcal{T}^{(2)} = \{((a_{m;3}, a_{m;2}, v_{m;3}, v_{m;2}, q_{m;3}, q_{m;2}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.43}$$

where the general definitions can be verified from the description shown in **Experiment 1**, without having the need to explicitly resume the terms again at this point of the section. Therefore the acceleration, velocity and displacement for the occupant of a specific element of training or test data can be denoted by $a_3$, $v_3$, $q_3 \in \mathbb{R}^N$ and analogously, we have $a_2$, $v_2$, $q_2 \in \mathbb{R}^N$ as dynamical data of the chassis. It can be recognized that in contrast to the datasets of the previous experiments, the true underlying parameter values $p_1 > 0$ and $p_2 > 0$ cannot be observed. Further, although we have access to velocity and displacement for the training data, we assume that for the relevant data to test, we only have access to the acceleration profiles. It is therefore obvious, that the estimation method should be described as a function, which uses the acceleration samples as input and maps to the parameter space, similar to the first experiments. Nevertheless, we cannot use the true parameter values to iteratively train such a model. We now want to investigate if the parameters can anyhow be identified from acceleration data following a hybrid optimization method, similar to [5, 38, 79, 84]. In this case, the exact parameters are not known, but we have appropriate prior knowledge about the acceleration's structure, using the corresponding second order ordinary differential equation, as it is stated by Eq. (5.40).

The **problem**, we want to investigate is: **Given the acceleration profiles of the defined training dataset and the connected velocities and displacements, can the unknown parameters be appropriately estimated with a Hybrid Neural Network, which uses knowledge about the data structure for the definition of an unlabelled loss function?**

**Method**

Again, let us assume that we have a Convolutional Neural Network, with the identical structure as defined in **Experiment 1**, such that we can define the network for the second experiment by

$$\begin{aligned} f_\theta^{(2)} &: \mathbb{R}^{N\times 2} \to \mathbb{R}^2 \\ (a_3, a_2) &\mapsto \begin{pmatrix} \tilde{p}_1(\theta) \\ \tilde{p}_2(\theta) \end{pmatrix}. \end{aligned}$$

The aim of defining a second function $f_\theta^{(2)}$ is to not confuse the reader, when comparing the results of the first and the second experiment in a latter part of the section. In addition, the predicted parameter values are defined in terms of the network's output by

$$\tilde{p}_i(\theta) = f_\theta^{(1)}\left((a_3, a_2)\right)_i$$

with $i \in \{1, 2\}$. It should be clear, that following Eq. (5.40), a prediction of the acceleration profile for the occupant can be computed, using the output of the Neural Network as

$$\tilde{a}_3(\theta) := -\tilde{p}_1(\theta)\,(v_3 - v_2) - \tilde{p}_2(\theta)\,(q_3 - q_2) \in \mathbb{R}^N, \tag{5.44}$$

since velocity and displacement are by definition assumed to be accessible for the training data. Otherwise, given the acceleration profile, one can also use common numerical integration schemes to compute the profiles of velocity and displacement. Since we want to investigate the idea of identifying the parameter values without having access to the true labels, we use the exact profiles to prevent our observations from being dependent on the approximation error of the integrated profiles.

Then, we can define the following optimization problem for the training of the Neural Network, to adjust the values of the parameters implicitly, via minimization of the underlying true acceleration profiles and the predicted data by

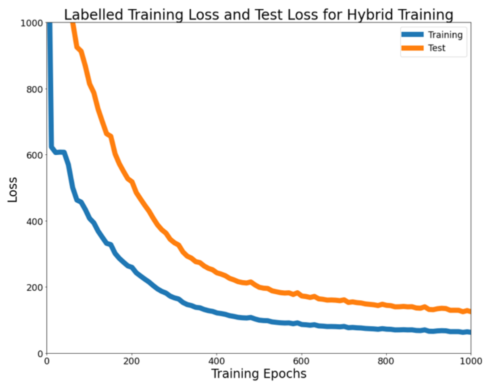
$$\min_{\theta} \frac{1}{N_S} \sum_{m=1}^{N_S} \mathscr{L}\left(\theta, (a_{m;3}, a_{m;2}), a_{m;3}\right) = \frac{1}{N_S} \sum_{m=1}^{N_S} \sum_{k=1}^{N} \left| a_{m;3}^{k} - \tilde{a}_{m;3}^{k}(\theta) \right|^2, \tag{5.45}$$

where $a_{m;3}^{k} \in \mathbb{R}$ for $k \in \{1, 2, ..., N\}$ describes the $k$-th entry of the vector-valued acceleration profile $a_{m;3} \in \mathbb{R}^N$. The same holds for the entries of the predicted acceleration $\tilde{a}_{m;3}^{k}(\theta) \in \mathbb{R}$.

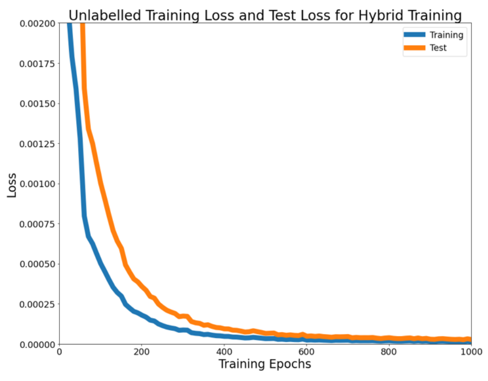
The training specifications, for instance optimization method, batch-size and learning-rate, are equal to those described in **Experiment 1** and the implementation details by Figure *A*.1, Figure *A*.2, Figure *A*.3 and Figure *A*.4. As the Neural Network has the same layer-structure as for the first experiment, a detailed description is again shown by Figure *B*.1. Similar to **Experiment 1**, we denote the trained network by $f_{\theta_\beta}^{(2)}$, where $\theta_\beta \in \Theta$ is the realization of the network parameters after $K = E\frac{N_S}{B} = 34000$ optimization steps. In summary, we use the same setting, but change the loss-function for this unlabelled optimization approach. The following results, showing the same structure as for the first experiment can therefore be used to directly compare the unlabelled and the labelled approach.

## Results

Again, we want to have a look at the shape of the loss-functions for the training of the unlabelled approach with 1000 training epochs. If the shape of the training and the test loss is comparable to those of the labelled approach, the unlabelled training should also be an appropriate method for the parameter estimation problem.



(a) Loss in Parameter Space.

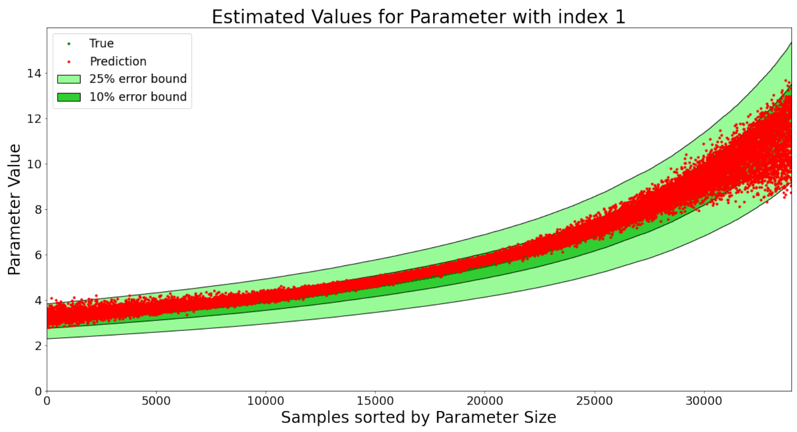


(b) Loss in Data Space.

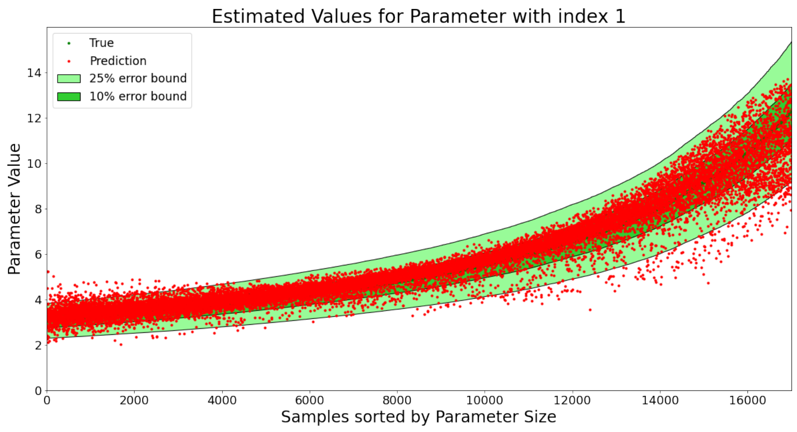
FIGURE 5.6: Shape of the loss-functions for labelled training approach. The horizontal axis shows the number of training epochs. The loss-functions have been evaluated for the entire datasets at each epoch. The vertical axis shows the value of the loss-function, as it has previously been defined. The shape of the training loss is visualized in blue, the test loss in orange. We differ between parameter loss in Figure 5.6*a* and data loss in Figure 5.6*b*. The parameter loss describes the value of the labelled loss-function in **Experiment 1**, the data loss describes the value of the unlabelled loss-function in **Experiment 2**.

As we can observe in Figure 5.6, the shapes of training and test loss, for the parameter space as well as for the data space look similar to those of Figure 5.3. The training loss is again below the test loss, we should therefore have a better performance for the training data. Nevertheless, we can notice that the shape of the loss-functions for the unlabelled approach is more smooth as those of the labelled training. Most often, a smooth shape of the loss-functions indicates a robust training method for Neural Networks.

Let us now analyse the results of the previously described unlabelled training method analogously to those of the first method in **Experiment 1**. Therefore, let us investigate the results shown in the following figures.



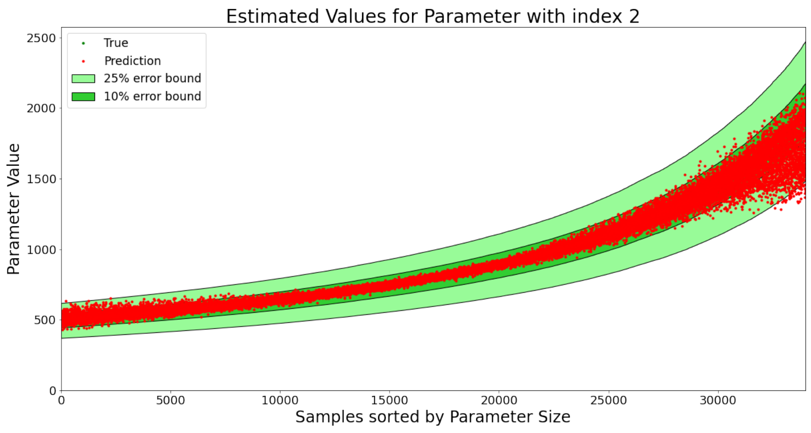
(a) Training results for estimation of parameter 1 using an unlabelled objective without noise.



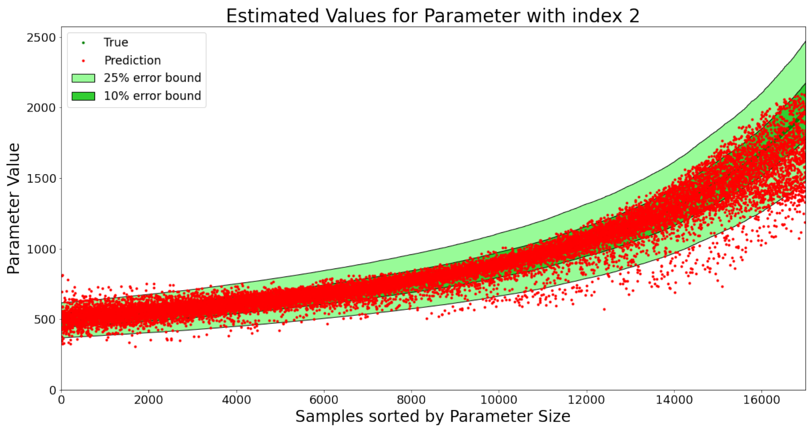
(b) Test results for estimation of parameter 1 using an unlabelled objective without noise.

FIGURE 5.7: Experiment 2: Unlabelled estimation of parameter $p_1$. A precise description of the plots is described by the caption of Figure 5.4. The training results are shown in Figure 5.7*a* and the test results in Figure 5.7*b*.

The results, shown in Figure 5.7, visualize the prediction of the trained Neural Network for parameter $p_1$ of the training dataset (Figure 5.7*a*) at the top and those of the test dataset (Figure 5.7*b*) at the bottom. Similar to the results of the first experiment for parameter $p_1$, we can observe that most parameter estimates for the training samples lie within the 10% error bound, while nearly all lie within the 25% range. In addition we can observe for this case, that in contrast to the labelled approach, there is a tendency to overestimate the parameter prediction, since the red point cloud is close to the upper bound of the 10% error range area. The last observation is that for large values of the parameter, it seems to be more probable that the Neural Network $f_{\theta_\beta}$ estimates smaller values, since we can recognize a significant deviation of the point cloud from the center of the green areas. The shape of the point cloud for the test data, is similar to the training cloud. Again, the results show a tendency of overestimation, while the largest parameters seem to be the most difficult ones to predict. We can further observe that the Neural Network seems to overfit the training data, since some estimates of the test data lie outside of the 25% area. Nevertheless, it is also worth mentioning, that the deviations are more close to the green areas, as it is for instance for the labelled approach in **Experiment 1**. We should therefore observe in the error distribution table, that the maximal deviation is (significantly) smaller as the deviation for the labelled approach.



(a) Training results for estimation of parameter 2 using an unlabelled objective without noise.



(b) Test results for estimation of parameter 2 using an unlabelled objective without noise.

FIGURE 5.8: Experiment 2: Unlabelled estimation of parameter $p_2$. A precise description of the plots is described by the caption of Figure 5.4. The training results are shown in Figure 5.8*a* and the test results in Figure 5.8*b*.

The results of the second parameter $p_2$ for the training and the test data are visualized in Figure 5.8. The results are almost similar to those of the first parameter. Similar to the labelled approach, we cannot observe an over - or underestimation of the predictions. The estimates are mostly distributed along the center of the green area, for the training data and for the test data. We can therefore conclude, since this effect occurs for the labelled and the unlabelled approach, that this effect is caused by the different value range of $p_1$ and $p_2$. The training of the Neural Network is more robust for the larger parameter $p_2$, while a small deviation of the network parameters seems to lead to a general tendency of over - or underestimation of all system parameters $p_1$. Furthermore, we can also recognize that larger parameter values are more likely to be underestimated. A large value of parameter $p_1$ or $p_2$ is equal to a small mass of the passenger and the seat, since $p_1 = \frac{C}{m}$ and $p_2 = \frac{K}{m}$, where $C$ is a general damping-constant, $K$ a spring-constant and $m$ the mass variable. The larger the quotient, the smaller the mass, since for the data generation only the mass has been randomly generated and not the quotient. Therefore, we can furthermore conclude that a larger mass leads to a more parameter-characteristic shape of the acceleration profile, while it is more difficult to evaluate the shape of acceleration profiles connected to small masses.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.32562 | 0.05214 | 0.03009 | 0.04403 | 0.52968 | 0.94165 | 0.99938 | 0.00062 |
| Training | 2 | 0.30742 | 0.02576 | 0.03233 | 0.33391 | 0.87688 | 0.96147 | 0.99865 | 0.00135 |
| Test | 1 | 0.70204 | 0.07552 | 0.06339 | 0.07400 | 0.41400 | 0.74294 | 0.97606 | 0.02394 |
| Test | 2 | 0.64694 | 0.06173 | 0.06623 | 0.14594 | 0.57724 | 0.80941 | 0.97335 | 0.02665 |

TABLE 5.2: Experiment 2: Error distribution of unlabelled parameter estimation. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

Analysing the error distribution values as they are shown in Table 5.2, again verifies the observations of the previously regarded figures. We have observed that there are smaller deviations for the test dataset from the 25% area, compared to the labelled results of **Experiment 1**. Therefore, we can observe that the maximal deviation for parameter $p_1$ is approximately 70% and for parameter $p_2$ around 65%, compared to 127% for both parameters of the labelled training method of **Experiment 1**. The general overestimation of the first parameter for training and test data can be observed from the 1% error range, where only 4.4% of the training predictions lie within this range, while we have 33.4% for the larger scaled parameter $p_2$. The same observation can be made for the test samples.

## Summary

In summary, we can compare Table 5.2 of the unlabelled training method and Table 5.1 of the labelled training method as follows: For the 5% area and for the 10% area, we can observe that for the training data, we have better estimates using the labelled approach. Where the unlabelled approach, for instance for the 5% range of parameter $p_1$, reaches a value of around 53%, we have 72% for the labelled approach. The same holds for the second parameter, where 94% of the training predictions lie within the 5% range, contrarily to a value of 88% for the unlabelled training approach. The most significant observation is therefore given, if we have a look at the error distribution of the test data for both approaches. We can observe that for the 10% and the 25% range, the unlabelled approach is on average 2% better, compared to the first method. Finally, $4-5\%$ of the test predictions in **Experiment 1** have an error larger than 25%, where we have around 2.5% for the unlabelled approach in **Experiment 2**.

We therefore conclude, that the unlabelled training method is in general more robust for an unknown test dataset, compared to the training method of the first experiment, although it is just a small improvement of around 2%. The now following example will give more obvious results concerning robustness against noise for the two training methods.

### 5.2.3 Experiment 3: Robust Parameter Estimation for Noisy Data

#### Problem

We have investigated a labelled and an unlabelled training method for an estimation problem of two parameters, containing the damping-coefficient $p_1$ and the spring-coefficient $p_2$, which are significant for the occupant's acceleration profile of the Quarter-Car-Model. Often, Neural Networks are capable to achieve great performances on a test dataset that is similar to an experienced training dataset. Nevertheless, it is still of immense research interest to make networks also robust against noise of the input data.

Let us therefore assume that we have one dataset to test the Neural Networks of the previous examples, based upon Gaussian disturbances of the training samples, denoted by

$$\mathcal{S}^{(3)} = \{((\widehat{a}_{m;3}, \widehat{a}_{m;2}))\}_{m=1}^{N_S} \tag{5.46}$$

and a second noisy test dataset based upon disturbances of the original test data given by

$$\mathcal{T}^{(3)} = \{((\widehat{a}_{m;3}, \widehat{a}_{m;2}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.47}$$

where $N_S \in \mathbb{N}$ and $N_T \in \mathbb{N}$ are again the sizes of the training and the test dataset. We assume that for each element $m \in \{1,2,...,N_S+N_T\}$, for all data points with $k \in \{1,2,...,N\}$, for the occupant's acceleration indexed by $i=3$ as well as for the chassis' acceleration indexed by $i=2$, the disturbed data is described by

$$\widehat{a}_{m;i}^{k} = a_{m;i}^{k}\left(1+\eta_{m;i}^{k}\right) \tag{5.48}$$

where the random noise term is supposed to follow a Gaussian distribution with $\eta_{m;i}^{k} \sim \mathcal{N}\left(0,\sigma^2\right)$. The noisy data points $\widehat{a}_{m;i}^{k} \in \mathbb{R}$ therefore deviate relatively from the clean points $a_{m;i}^{k} \in \mathbb{R}$ by multiplication of the noise term $(1+\eta_{m;i}^{k}) \in \mathbb{R}$. The standard deviation $\sigma$ will be specified, when observing the results of this experiment.

The **problem**, we want to investigate is: **Given the noisy acceleration profiles for the chassis and the occupant, how robust are the pre-trained Neural Networks to estimate the parameter values, as it has been described in the previous experiment, from these noisy samples?**

**Method**

Let us assume that the trained Neural Network of **Example 1** can uniquely be described by a realization of the network parameters, denoted by $\theta_\alpha$, where the network of **Example 2** is specified by $\theta_\beta$. This means, that $\theta_\alpha$ and $\theta_\beta$ are the values of the network parameters, that result from ADAM optimization with 34000 iterations for the minimization problem of Eq. (5.41) for $\theta_\alpha$ and Eq. (5.45) for $\theta_\beta$.

Given the noisy datasets $\mathcal{S}^{(3)}$ and $\mathcal{T}^{(3)}$, we not want to evaluate the robustness against Gaussian disturbances of the Neural Networks $f^{(1)}_{\theta_\alpha}$ and $f^{(2)}_{\theta_\beta}$ for the task to predict the parameters $p_1$ and $p_2$.

Therefore, we want to observe the predictions, resulting from

$$\begin{pmatrix} \tilde{p}_1(\theta_\alpha) \\ \tilde{p}_2(\theta_\alpha) \end{pmatrix} = f^{(1)}_{\theta_\alpha}\left((\widehat{a}_3, \widehat{a}_2)\right) \tag{5.49}$$

and compare them to the results of the output

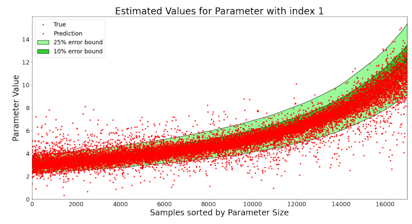
$$\begin{pmatrix} \tilde{p}_1(\theta_\beta) \\ \tilde{p}_2(\theta_\beta) \end{pmatrix} = f^{(2)}_{\theta_\beta}\left((\widehat{a}_3, \widehat{a}_2)\right) \tag{5.50}$$

for elements $\widehat{a}_3, \widehat{a}_2$ of the datasets $\mathcal{S}^{(3)}$ and $\mathcal{T}^{(3)}$. The now following results show the performance of the labelled and unlabelled approach, using in a first step noisy samples with $\sigma = 0.01$ and in a second step $\sigma = 0.05$.

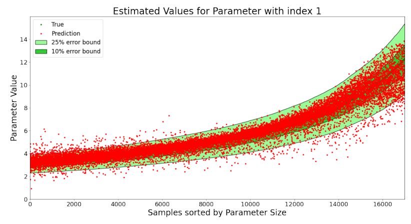
**Results**

We separate the results, by an **Experiment 3a**, which describes the evaluation of the pre-trained Neural Networks using noisy datasets with a deviation of $\sigma = 0.01$ for the random Gaussian noise of the data. The second part, **Experiment 3b**, then describes the same evaluation, using $\sigma = 0.05$. As a starting point, let us analyse **Experiment 3a**.

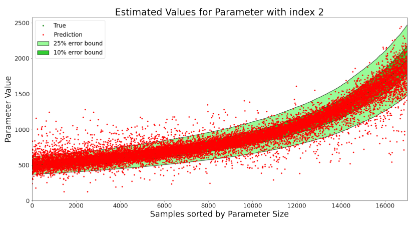
The visualized results in Figure 5.9 are restricted to the estimated parameter values of noisy elements of $\mathcal{T}^{(3)}$. The results for the pre-trained Neural Network $f^{(1)}_{\theta_\alpha}$ are shown on the l.h.s., the results for $f^{(2)}_{\theta_\beta}$ on the r.h.s. of the figure. Obviously, the prediction is relatively robust for $\sigma = 0.01$ for both training approaches. Nevertheless, one can recognize, that similar to the test results of **Experiment 1** and **Experiment 2**, the unlabelled method is more robust, since there are more larger deviations for the labelled approach compared to the unlabelled



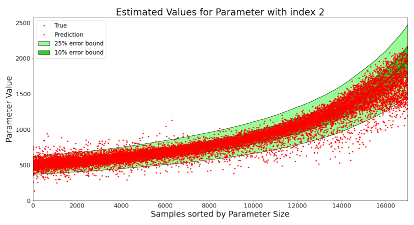
(a) Test results for labelled objective with $\sigma = 0.01$ for parameter 1.



(b) Test results for unlabelled objective with $\sigma = 0.01$ for parameter 1.



(c) Test results for labelled objective with $\sigma = 0.01$ for parameter 2.



(d) Test results for labelled objective with $\sigma = 0.01$ for parameter 2.

FIGURE 5.9: Experiment 3: Noise level $\sigma = 0.01$, labelled vs unlabelled training. A precise description of the plots is described by the caption of Figure 5.4. The test results for the labelled objective are shown in Figure 5.9*a* for $p_1$ and in Figure 5.9*c* for $p_2$. The test results for the unlabelled objective are shown in Figure 5.9*b* for $p_1$ and in Figure 5.9*d* for $p_2$.

training. A more detailed analyse is shown with the now following tables for both methods.

| **Data** | **Parameter** | **Max Error** | **Mean Error** | **Std Error** | **0-1%** | **0-5%** | **0-10%** | **0-25%** | **>25%** |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.98915 | 0.06331 | 0.06592 | 0.10376 | 0.53153 | 0.82976 | 0.97903 | 0.02097 |
| Training | 2 | 0.95878 | 0.04922 | 0.05990 | 0.18032 | 0.67512 | 0.88147 | 0.98465 | 0.01535 |
| Test | 1 | 1.45144 | 0.09748 | 0.10591 | 0.08029 | 0.38618 | 0.66259 | 0.93053 | 0.06947 |
| Test | 2 | 1.42642 | 0.08714 | 0.10268 | 0.10147 | 0.45424 | 0.71724 | 0.93906 | 0.06094 |

TABLE 5.3: Experiment 3a: Error distribution of labelled parameter estimation with noise $\sigma = 0.01$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is explicitly shown in the caption of Table 5.1.

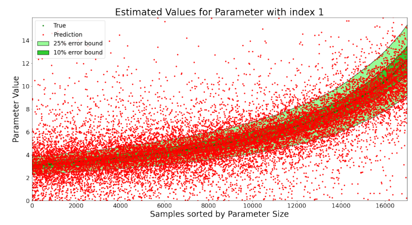
| **Data** | **Parameter** | **Max Error** | **Mean Error** | **Std Error** | **0-1%** | **0-5%** | **0-10%** | **0-25%** | **>25%** |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.93651 | 0.06238 | 0.04897 | 0.07938 | 0.46412 | 0.84091 | 0.99271 | 0.00729 |
| Training | 2 | 0.90251 | 0.04405 | 0.04664 | 0.18468 | 0.69488 | 0.90606 | 0.99391 | 0.00609 |
| Test | 1 | 0.93830 | 0.08323 | 0.07453 | 0.07882 | 0.39188 | 0.70112 | 0.96335 | 0.03665 |
| Test | 2 | 0.84944 | 0.07139 | 0.07516 | 0.12112 | 0.51247 | 0.76994 | 0.96347 | 0.03653 |

TABLE 5.4: Experiment 3a: Error distribution of unlabelled parameter estimation with noise $\sigma = 0.01$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

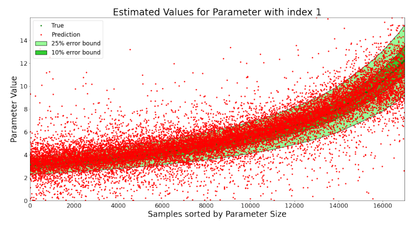
We directly compare the results for the noisy dataset of the pre-trained network $f^{(1)}_{\theta_\alpha}$ in Table 5.3 with the results of $f^{(2)}_{\theta_\beta}$ in Table 5.4. The results of the noisy set $\mathcal{S}^{(3)}$ are similar for both approaches, as we can recognize from comparing the tables. Let us therefore restrict to the results of the noisy test set $\mathcal{T}^{(3)}$. Concerning the maximal deviation of the approaches, we can recognize that for the labelled approach, we have values of 145.1% for parameter

$p_1$ and 142.6% for parameter $p_2$, while for the unlabelled method, the maximal deviation is significantly lower with 94% for $p_1$ and 85% for $p_2$. Besides, we recognize a mean standard deviation of approximately 10% for the first approach, while the hybrid / unlabelled objective leads to values of 7.5%. The unlabelled network is therefore more close to the previously defined green error boundaries. It can also be observed that the 10% error bound is $4-5\%$ larger and the 25% error bound is approximately 3% larger for the unlabelled approach. Using a deviation of $\sigma = 0.01$ therefore results in a more robust performance for $f^{(2)}_{\theta_\beta}$. This unlabelled approach has also been observed to be more robust for clean test samples in **Experiment 2**. As we can only find small improvements for the unlabelled approach, we are now going to focus on a second experiment, increasing the noise rate of the disturbed dataset.

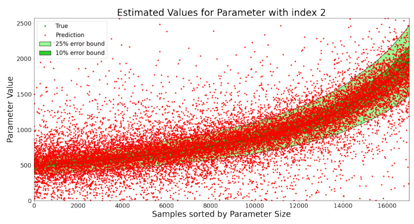
We continue with **Experiment 3b**, where we generate datasets $\mathcal{S}^{(3)}$ and $\mathcal{T}^{(3)}$, as defined by the problem description, with a standard deviation of $\sigma = 0.05$. The now following results are analogously structured to those of **Experiment 3a**.



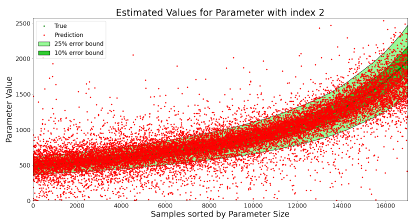
(a) Test results for labelled objective with $\sigma = 0.05$ for parameter 1.



(b) Test results for unlabelled objective with $\sigma = 0.05$ for parameter 1.



(c) Test results for labelled objective with $\sigma = 0.05$ for parameter 2.



(d) Test results for labelled objective with $\sigma = 0.05$ for parameter 2.

FIGURE 5.10: Experiment 3: Noise level $\sigma = 0.05$, labelled vs unlabelled training. A precise description of the plots is described by the caption of Figure 5.4. The test results for the labelled objective are shown in Figure 5.10*a* for $p_1$ and in Figure 5.10*c* for $p_2$. The test results for the unlabelled objective are shown in Figure 5.10*b* for $p_1$ and in Figure 5.10*d* for $p_2$.

Analogous to the previously shown Figure 5.9 for $\sigma = 0.01$, we can now make use of Figure 5.10 for $\sigma = 0.05$ to visualize the predictions of the pre-trained Neural Network $f^{(1)}_{\theta_\alpha}$ on the l.h.s. and those of $f^{(2)}_{\theta_\beta}$ on the r.h.s. of the figure. Again, we observe the results for the labelled noisy test data on the l.h.s., while the r.h.s. of the figure shows the estimates of the unlabelled pre-trained Neural Network. We can recognize that a deviation of $\sigma = 0.05$, significantly leads to a much higher deviation of the parameter estimation problem, compared to the previously shown plots for $\sigma = 0.01$. Similar to that, it is possible to observe, that the results for $f^{(2)}_{\theta_\beta}$ are more dense to the green areas, compared to $f^{(1)}_{\theta_\alpha}$.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 6.31576 | 0.22820 | 0.27531 | 0.04021 | 0.19950 | 0.37888 | 0.70021 | 0.29979 |
| Training | 2 | 5.97519 | 0.21388 | 0.26473 | 0.04785 | 0.22385 | 0.40747 | 0.72368 | 0.27632 |
| Test | 1 | 3.56003 | 0.25308 | 0.28616 | 0.03694 | 0.17376 | 0.33165 | 0.65841 | 0.34159 |
| Test | 2 | 3.38591 | 0.23831 | 0.27607 | 0.03953 | 0.19024 | 0.35900 | 0.68265 | 0.31735 |

TABLE 5.5: Experiment 3b: Error distribution of labelled parameter estimation with noise $\sigma = 0.05$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 3.30868 | 0.16372 | 0.19006 | 0.05291 | 0.25556 | 0.46888 | 0.80979 | 0.19021 |
| Training | 2 | 3.19051 | 0.15188 | 0.17979 | 0.05991 | 0.27974 | 0.50747 | 0.83088 | 0.16912 |
| Test | 1 | 3.22332 | 0.17836 | 0.19840 | 0.04806 | 0.23159 | 0.42718 | 0.78182 | 0.21818 |
| Test | 2 | 2.99078 | 0.16712 | 0.18897 | 0.05065 | 0.25153 | 0.46241 | 0.80053 | 0.19947 |

TABLE 5.6: Experiment 3b: Error distribution of unlabelled parameter estimation with noise $\sigma = 0.05$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

Table 5.5 shows the results for $f^{(1)}_{\theta_\alpha}$, Table 5.6 shows the corresponding results for $f^{(2)}_{\theta_\beta}$, evaluated for all samples of the noisy datasets $\mathcal{S}^{(3)}$ and $\mathcal{T}^{(3)}$ with $\sigma = 0.05$. All in all, the comparison of the two tables shows more significant differences as **Experiment 3a**. As far as $f^{(1)}_{\theta_\alpha}$ is concerned, the results are in a comparable range for the noisy training dataset and the noisy test dataset. The same holds for the error values of $f^{(2)}_{\theta_\beta}$. Obviously, we have smaller maximal deviations, a smaller mean deviation and a smaller mean standard deviation for the unlabelled approach in contrast to the labelled method. Furthermore, we have at least 23% in the 5% range for unlabelled training, compared to at least 17% for the labelled training. The same observation can be found for the 10% and 25% areas, where the unlabelled method shows approximately 10% larger areas throughout all types of noisy data. This can also be verified from analysing the percentage, that shows a relative deviation of more than 25%, where we have $16.9-21.8\%$ for the unlabelled method and $27.6-34.1\%$ for the labelled one. It is therefore obvious that the more sophisticated, hybrid learning method of **Experiment 2** leads to significantly more robust results, throughout all error estimation values shown in the tables.

## Summary

The discussed results have shown, that the unlabelled trained Neural Network $f^{(2)}_{\theta_\beta}$, although the training performance is worse compared to the labelled $f^{(1)}_{\theta_\alpha}$, considering **Experiment 1** and **Experiment 2**, is significantly more robust against disturbances of the input data. This holds for the disturbed training data $\mathcal{S}^{(3)}$ as well as for the disturbed test data $\mathcal{T}^{(3)}$ for $\sigma = 0.01$ and $\sigma = 0.05$.

We can therefore summarize the observations as follows: It is preferable to use an unlabelled training approach for the parameter estimation problem of the occupant-related coefficients, if we want to develop a robust model. The robust unlabelled training shows stable predictions for clean test data and additionally less sensitivity to noisy samples, compared to a straight forward labelled Neural Network data fitting approach.

### 5.2.4 Experiment 4 : Denoising via Neural Networks

#### Problem

Up to this point, we have analysed a labelled and an unlabelled Neural Network training approach, regarding robust prediction for unknown test data and sensitivity of Gaussian relative noise of the input data. We have further recognized that even for deviations with $\sigma = 0.05$, the unlabelled approach is significantly more robust compared to the labelled pre-trained Neural Network. Nevertheless, the prediction quality for the noisy data in **Experiment 3** is much worse to those of **Experiment 1** and **Experiment 2**. Therefore, one usually uses data pre-processing algorithms to denoise the data, instead of directly using the raw noisy data to estimate the parameter values.

At a last investigation for this two-dimensional parameter estimation problem, we want to analyse, if we can develop another Hybrid Model, which maps the noisy data samples to a denoised version, as a first step, and then uses the pre-processed samples to estimate the parameters of the occupant's acceleration profile. We assume, that a Convolutional Auto-Encoder (CAE) structure, is capable to solve this problem.

#### Method

Let us again assume that we consider a noisy training dataset of the type

$$\mathcal{S}^{(4)} = \{((\widehat{a}_{m;3}, \widehat{a}_{m;2}), (a_{m;3}, a_{m;2}))\}_{m=1}^{N_S} \tag{5.51}$$

and a corresponding test dataset

$$\mathcal{T}^{(4)} = \{((\widehat{a}_{m;3}, \widehat{a}_{m;2}), (a_{m;3}, a_{m;2}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.52}$$

where we have observable noisy data points $\widehat{a}_{m;i}^{k}$ and the clean underlying true data points $a_{m;i}^{k}$, where $m \in \{1,2,...,N_S + N_T\}$, $k \in \{1,2,...,N\}$ and $i \in \{2,3\}$, now serving as labels for the denoising network. Let us therefore consider a function of the form

$$f_{\theta}^{(4)} : \mathbb{R}^{N\times 2} \to \mathbb{R}^{N\times 2} \tag{5.53}$$

$$(\widehat{a}_3, \widehat{a}_2) \mapsto (\tilde{a}_3(\theta), \tilde{a}_2(\theta)), \tag{5.54}$$

such that the predicted clean data samples, as output of the Neural Network, are defined by

$$\tilde{a}_3(\theta) = f_{\theta}^{(4)} (\widehat{a}_3, \widehat{a}_2)_1 \in \mathbb{R}^N \tag{5.55}$$

and analogously

$$\tilde{a}_2(\theta) = f_\theta^{(4)}\left(\widehat{a}_3, \widehat{a}_2\right)_2 \in \mathbb{R}^N. \tag{5.56}$$

We can then define the training problem of the denoising task as follows:

$$\min_\theta \sum_{m=1}^{S} \sum_{k=1}^{N} \sum_{j=2}^{3} \left| \tilde{a}_{m;j}^k(\theta) - a_{m;j}^k \right|^2, \tag{5.57}$$

where $\tilde{a}_{m;j}^k \in \mathbb{R}$ and $a_{m;j}^k \in \mathbb{R}$ describe the $k$-th element of the predicted data and the clean data. Since the network $f_\theta^{(4)}$ has a different structure compared to the previously defined networks in **Experiment 1** and **Experiment 2**, the network parameters $\theta$ have a different shape, as it can be seen from the layer structure in Figure *B*.2. The task of the Neural Network can therefore be described to map the noisy data to the clean underlying samples.

The now following results can be separated into two steps. First, we train the Neural Network, using the following training specifications:

- Optimization method: Adaptive Momentum (ADAM) Optimization
- Learning-rate: $\eta = 0.0001$
- Batch-size: $B = 100$
- Series-length of the input samples: $N = 500$
- Training-epochs: $E = 100$
- Noise rate of the samples: $\sigma = 0.1$

We can then compare, if the denoising operation is appropriately done for training samples as well as for test samples. Afterwards, the trained Neural Network, which we are going to denote by $f_{\theta_\gamma}^{(4)}$, where $\theta_\gamma$ is the value of the network parameters, after $K = E\frac{N_S}{B} = 100 \cdot 340 = 34000$ training steps, can be used to denoise the corrupted data as a pre-processing step. Implementation details can be found in Figure *A*.5, Figure *A*.6, Figure *A*.7 and Figure *A*.8. Then, the pre-trained Neural Networks for parameter estimation, $f_{\theta_\alpha}^{(1)}$ for the labelled approach and $f_{\theta_\beta}^{(2)}$ for the unlabelled approach, can be used to estimate the parameters, similar to the previous experiments, taking this pre-processed data as input.

In detail, we can compare the predictions for the noisy data for $f_{\theta_\alpha}^{(1)}$, given by

$$f_{\theta_\alpha}^{(1)}\left(\hat{a}_3, \hat{a}_2\right) \tag{5.58}$$

with the predictions using the denoising network, described by

$$f_{\theta_\alpha}^{(1)}\left(\tilde{a}_3(\theta_\gamma), \tilde{a}_2(\theta_\gamma)\right). \tag{5.59}$$

The same is then also done for the noisy predictions of the unlabelled pre-trained network $f_{\theta_\beta}^{(2)}$ and the predictions of a denoised input.

### Results

The section is separated into two steps: First, we want to investigate the training loss and test loss of the Convolutional Auto-Encoder, similar to **Experiment 1** and **Experiment 2**, to evaluate if the optimization is appropriately done. Then we want to give an overview about the approximation error of the predictions for the test data, visualized by histograms and finally shows some examples of noisy input data, clean true data and denoised data by the trained Neural Network.

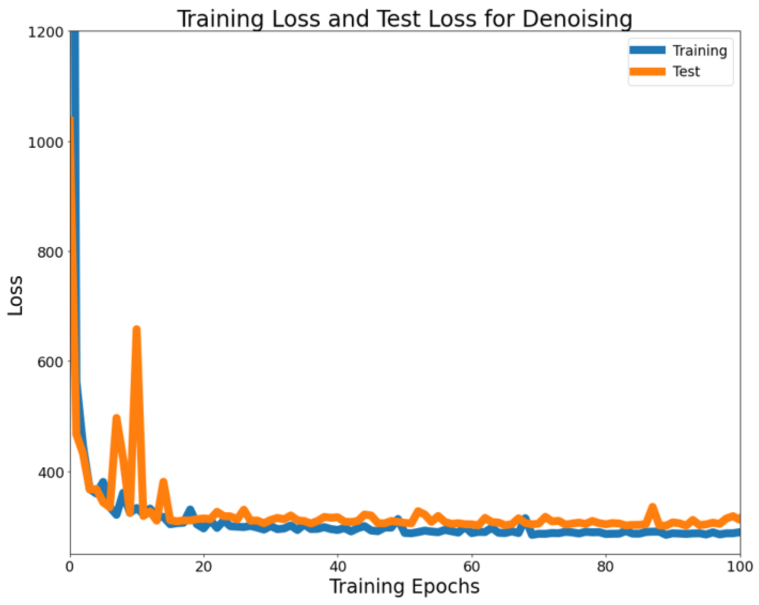
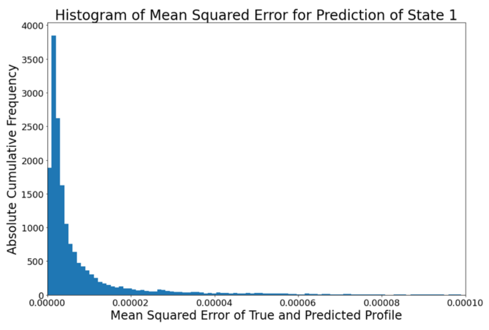


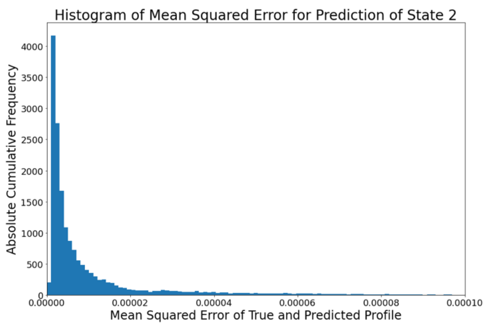
FIGURE 5.11: Training Loss and Test Loss of Training the Convolutional Auto-Encoder for denoising of the input data. The shape of the training loss is visualized in blue, the test loss in orange. The horizontal axis shows the number of training epochs, the vertical axis the values of the loss-function. The loss has been evaluate after each epoch for the entire datasets.

The shape of the training loss and the test loss is shown in Figure 5.11. Similar to Figure 5.3 and Figure 5.6, the training and the test loss are appropriately reduced for a large number of training epochs. It is significant that there are some peaks for the area around 10 epochs for the test loss. Nevertheless, the shapes of the training loss and the test loss are close to each other from epoch 20 to the termination of the training after reaching 100 epochs. There is obviously no improvement in the performance for this range, since the loss remains in the same area and is not further minimized, if we continue the training.

The histograms in Figure 5.12, which show the Mean-Squared-Error (MSE) for the test dataset yield, that we have the same range of the approximation error for the denoising of the occupant's acceleration profile as well as for the denoising of the chassis acceleration profile. Most errors are close to zero, nearly all are below a value of $4 \cdot 10^{-5}$. We need to verify, if this is a sufficiently low approximation error, with the help of some examples.

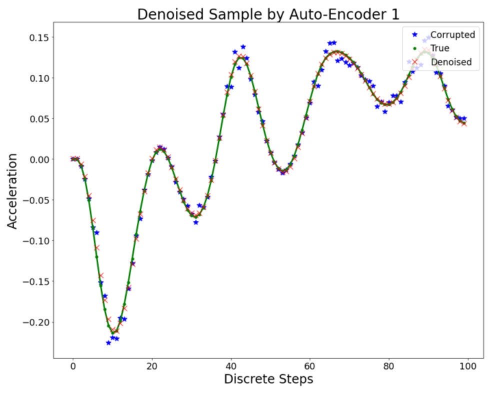


(a) Error Distribution of Occupant Acceleration.

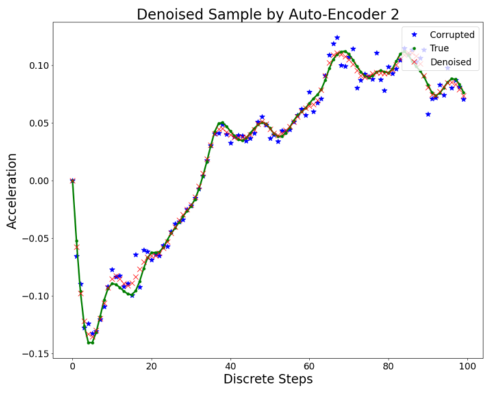


(b) Error Distribution of Chassis Acceleration.

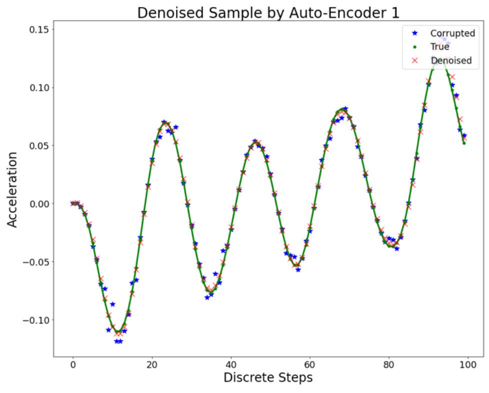
FIGURE 5.12: Histograms of the Mean-Squared-Error for the predicted cleaned data of the test dataset. The horizontal axis shows the Mean-Squared-Error for each denoised sample, according to the true underlying clean test sample. The vertical axis shows the absolute cumulative frequency of the error values that have been computed for the test samples.



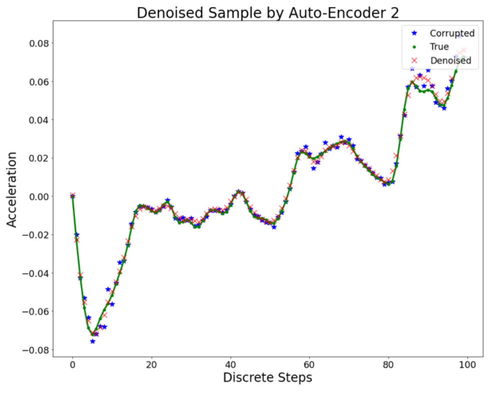
(a) Denoising of Occupant Acceleration.



(b) Denoising of Chassis Acceleration.



(c) Denoising of Occupant Acceleration.

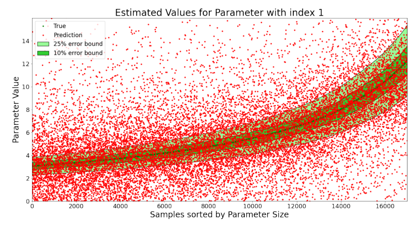


(d) Denoising of Chassis Acceleration.

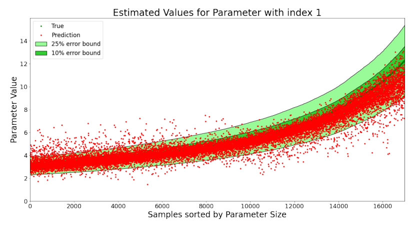
FIGURE 5.13: Examples of Denoising for Test Samples. The figure shows the prediction of the Convolutional Auto-Encoder for noisy input samples of the test dataset. The horizontal axis for each plot describes the discrete step, we therefore have 100 discrete points for each sample. The vertical axis shows the value of the acceleration. The plot shows the values of the noisy input, true clean sample and the predicted denoised output.

Obviously, the denoised samples are close to the true clean samples, as it can be seen in Figure 5.13. This holds true for the denoising of the occupant's acceleration on the l.h.s. as well as for the chassis profile on the r.h.s. Moreover, it is worth mentioning that the noisy input data is also close to the true clean acceleration value. Nevertheless, we have already seen that even smaller deviations of $\sigma = 0.05$, lead to a significant performance reduction for the pre-trained Neural Networks. Therefore, we want to observe in the second part of the results, if the herein shown denoised data, is sufficiently close to the true data, such that the accuracy of the parameter prediction is improved.

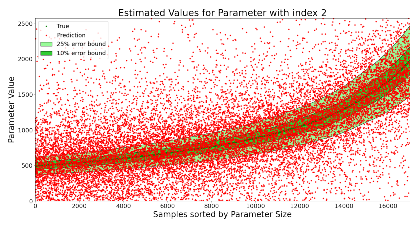
We can easily recognize the efficient use of a pre-processing algorithm, like it is our pre-trained denoising Convolutional Auto-Encoder, with the help of the now following figures and tables. We therefore directly compare the parameter estimation results for the labelled pre-trained Neural Network $f^{(1)}_{\theta_\alpha}$ and the unlabelled approach $f^{(2)}_{\theta_\beta}$, predicting the corresponding parameters from input samples with noise level $\sigma = 0.1$ and in contrast from denoised samples using $f^{(4)}_{\theta_\gamma}$.



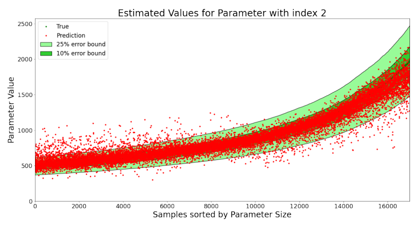
(a) Test results for labelled objective with $\sigma = 0.1$ for parameter 1.



(b) Test results for labelled objective with $\sigma = 0.1$ and denoising Auto-Encoder for parameter 1.



(c) Test results for labelled objective with $\sigma = 0.1$ for parameter 2.



(d) Test results for labelled objective with $\sigma = 0.1$ and denoising Auto-Encoder for parameter 2.

FIGURE 5.14: Experiment 4: Noise level $\sigma = 0.1$, estimation for raw noisy data vs. estimation for pre-processed data for labelled pre-trained network. A precise description of the plots is given by the caption of Figure 5.4. The estimation results for noisy input data of the test dataset, using the labelled pre-trained network, are shown in Figure 5.14*a* for $p_1$ and in Figure 5.14*c* for $p_2$. Contrarily, Figure 5.14*b* shows the estimation for pre-processed test samples for $p_1$ and finally Figure 5.14*d* shows the same for $p_2$.

We can observe the visualization of the parameter estimation problem for the noisy test dataset with $\sigma = 0.1$ in Figure 5.14, which shows the predictions of the two-dimensional problem for the noisy input samples, using the labelled, pre-trained Convolutional Neural Network $f^{(1)}_{\theta_\alpha}$. The l.h.s. of the figure, shows the result of the test dataset $\mathcal{T}^{(4)}$, using the noisy samples $\widehat{a}_3$ and $\widehat{a}_2$, where the r.h.s. in contrast shows the parameter estimation of the same network, using the de-noised samples $\tilde{a}_3(\theta_\gamma)$ and $\tilde{a}_2(\theta_\gamma)$, resulting from the pre-trained denoising Convolutional Auto-Encoder. Obviously, taking the raw noisy data samples produces estimates, which deviate significantly from the target values, where the pre-processed samples lead to more robust predictions, which are nearly comparable to the observed results from **Experiment 1**.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 8.70219 | 0.40582 | 0.48987 | 0.02121 | 0.10824 | 0.21465 | 0.47247 | 0.52753 |
| Training | 2 | 8.32585 | 0.38620 | 0.46855 | 0.02482 | 0.11776 | 0.22785 | 0.49391 | 0.50609 |
| Test | 1 | 9.32839 | 0.43248 | 0.52005 | 0.02059 | 0.09941 | 0.19847 | 0.44906 | 0.55094 |
| Test | 2 | 8.95942 | 0.41321 | 0.49810 | 0.01959 | 0.10818 | 0.21353 | 0.47000 | 0.53000 |

TABLE 5.7: Experiment 4: Error distribution of labelled parameter estimation with noise $\sigma = 0.1$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.68530 | 0.06077 | 0.04827 | 0.08838 | 0.48462 | 0.84374 | 0.99291 | 0.00709 |
| Training | 2 | 0.70370 | 0.04555 | 0.04767 | 0.18147 | 0.67679 | 0.89515 | 0.99356 | 0.00644 |
| Test | 1 | 1.02250 | 0.08830 | 0.08890 | 0.07765 | 0.39165 | 0.70206 | 0.94800 | 0.05200 |
| Test | 2 | 1.03719 | 0.07617 | 0.09009 | 0.11976 | 0.51294 | 0.76224 | 0.95335 | 0.04665 |

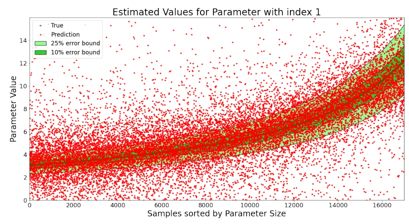
TABLE 5.8: Experiment 4: Error distribution of labelled parameter estimation with noise $\sigma = 0.1$, using a pre-trained Convolutional Auto-Encoder for denoising. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

The observations, already made with respect to Figure 5.14, can further be verified, comparing Table 5.7, describing the already known error structure for the noisy test data of $f_{\theta_\alpha}^{(1)}$, where Table 5.8, shows the result of the pre-processed denoising operation of $f_{\theta_\gamma}^{(4)}$ on $\mathcal{S}^{(4)}$ and $\mathcal{T}^{(4)}$. Without having the need to compare all columns separately, we can recognize that the pre-processing significantly improves the performance of the Neural Network. We can therefore, for instance, observe, that for the raw data in Table 5.7, more than 50% of the parameter estimates, have a relative deviation of more than 25% from the target values, where we have less than 5.2% for the pre-processed data in Table 5.8.

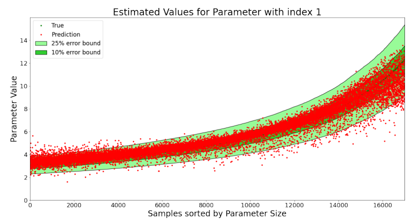
We observe similar results for the pre-trained unlabelled Neural Network $f_{\theta_\beta}^{(2)}$ in Figure 5.15, compared to the previously seen results for $f_{\theta_\alpha}^{(1)}$ in Figure 5.14. It is again obvious, that the pre-processing of the noisy test samples, shows a significant improvement for the accuracy of the estimated parameter values.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 5.05091 | 0.28981 | 0.33701 | 0.03185 | 0.15476 | 0.29694 | 0.60488 | 0.39512 |
| Training | 2 | 4.82886 | 0.27441 | 0.31822 | 0.03453 | 0.16253 | 0.31021 | 0.62765 | 0.37235 |
| Test | 1 | 5.15793 | 0.30893 | 0.35039 | 0.02900 | 0.14306 | 0.26900 | 0.57182 | 0.42818 |
| Test | 2 | 4.89862 | 0.29352 | 0.33027 | 0.02853 | 0.14547 | 0.28294 | 0.59429 | 0.40571 |

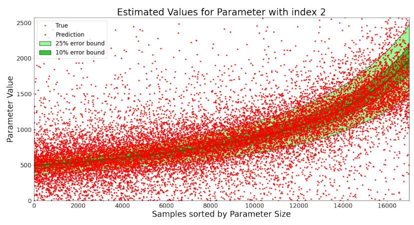
TABLE 5.9: Experiment 4: Error distribution of unlabelled parameter estimation with noise $\sigma = 0.1$. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.



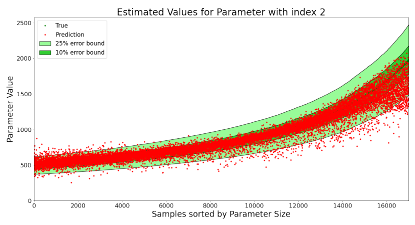
(a) Test results for unlabelled objective with $\sigma = 0.1$ for parameter 1.



(b) Test results for unlabelled objective with $\sigma = 0.1$ and denoising auto-encoder for parameter 1.



(c) Test results for unlabelled objective with $\sigma = 0.1$ for parameter 2.



(d) Test results for unlabelled objective with $\sigma = 0.1$ and denoising auto-encoder for parameter 2.

FIGURE 5.15: Experiment 4: Noise level $\sigma = 0.1$, estimation for raw noisy data vs. estimation for pre-processed data for unlabelled pre-trained network. A precise description of the plots is described by the caption of Figure 5.4. The estimation results for noisy input data of the test dataset, using the unlabelled pre-trained network, are shown in Figure 5.15*a* for $p_1$ and in Figure 5.15*b* for $p_2$. Contrarily, Figure 5.15*c* shows the estimation for pre-processed test samples for $p_1$ and finally Figure 5.15*d* shows the same for $p_2$.

| Data | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| Training | 1 | 0.68789 | 0.06354 | 0.05211 | 0.08891 | 0.48088 | 0.82229 | 0.99068 | 0.00932 |
| Training | 2 | 0.62953 | 0.04755 | 0.04954 | 0.17841 | 0.66959 | 0.88012 | 0.99271 | 0.00729 |
| Test | 1 | 0.82857 | 0.07567 | 0.06666 | 0.08829 | 0.42641 | 0.74629 | 0.97153 | 0.02847 |
| Test | 2 | 0.76016 | 0.06459 | 0.06796 | 0.13188 | 0.56094 | 0.79953 | 0.96976 | 0.03024 |

TABLE 5.10: Experiment 4: Error distribution of unlabelled parameter estimation with noise $\sigma = 0.1$, using a pre-trained Convolutional Auto-Encoder for denoising. The table shows the error distribution and more statistical values of the observed estimation results. The whole datasets are considered for the computation. The description of the columns is shown in the caption of Table 5.1.

Finally, we can make use of Table 5.9 for the noisy estimates of $f^{(2)}_{\theta_\beta}$ and Table 5.10 for the denoised estimates. Similar to the results shown in Table 5.7 and Table 5.8 for the labelled approach, we can observe that the denoised experiment, delivers throughout a more robust estimation of the strongly noise-corrupted input samples. More than 37% of the noisy samples lead to a relative deviation of more than 25%, where we have around less than 3% for the denoised samples. Furthermore, we can recognize that again the hybrid, unlabelled approach is significantly more robust, when processing the raw noisy data samples for parameter estimation.

## Summary

It is obvious, that a denoising Neural Network can efficiently be trained to improve the prediction quality of Neural Networks to solve parameter estimation tasks. Summarizing the shown results, we have investigated that unlabelled, hybrid objective functions deliver appropriate robust estimations for test samples and disturbed samples of a relatively small Gaussian noise term. For larger deviations, another Hybrid Model can be used, considering a pre-trained denoising Convolutional Auto-Encoder to estimate a cleaned version of the corrupted input data. Then, the pre-processed samples can be used to achieve good parameter estimation results, which are comparable to the results seen in **Experiment 1** and **Experiment 2** for clean data samples.

We have investigated several experiments for a two-dimensional parameter estimation problem to predict the parameters of occupant-related acceleration profiles. A Hybrid Neural Network can therefore on the one hand efficiently be used to train a robust estimation model, using an unlabelled objective, while on the other hand an additional Hybrid Model can be defined, using a denoising Convolutional Auto-Encoder network as a pre-processing algorithm, to significantly improve the estimation quality for noisy samples. The next section therefore investigates a more difficult problem, namely estimating all parameters of the Quarter-Car-Model and not only the parameters of a single ordinary differential equation of the entire system.

## 5.3 System Identification of the Quarter-Car-Model

The previous section has shown, that the parameters, corresponding to the acceleration profile of the occupant of the Quarter-Car-Model, can efficiently be estimated, using two different training methods for parameter estimation and an additional denoising network, such that the problem can even be solved for strongly corrupted noisy test samples.

It has further been shown in Section 5.1, that varying the grade of observable states leads to more possible variations of the system matrices of the Quarter-Car-Model, which deliver the same output as the true underlying matrix. We therefore want to investigate in this section, if all parameters of the Quarter-Car-Model can uniquely be determined from a given set of acceleration profiles, if we assume all states to be observable.

We therefore want to follow the idea stated in Figure 5.16 in the now following section. Again, we assume that there is knowledge about the input (control) of the system and the output (system response). As we now want to estimate all parameters of the Quarter-Car-Model, the input is given by the scaled road-displacement, namely the dissipative spring-force acting on the coupled system, where for a full-observable system, we assume that the acceleration profiles of the wheel-suspensions, the chassis and the occupant are all accessible for a given test dataset.

Furthermore, the system is assumed to be partially unknown, as we do not know the true values of the distinguishable relevant spring - and damping - coefficients of the system matrix, but have knowledge about the coupled structure of the system. Following [28, 35, 41, 77, 78], we can define a non-linear least-squares minimization problem, basically comparable to

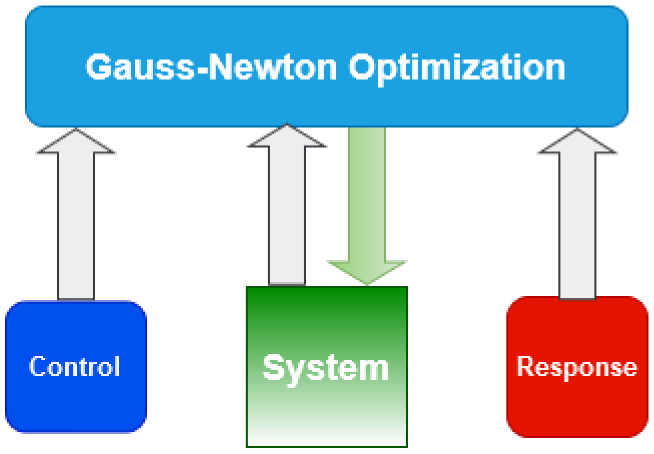


FIGURE 5.16: Identification of the Quarter-Car-Model parameters using a Gauss-Newton optimization method. The algorithm requires an estimate of the System and the Control to reproduce the estimated values of the System Response. A least-squares fit then results in an updated estimate of the System.

the unlabelled training approach in **Experiment 2**, to indirectly estimate the parameters of the entire Quarter-Car-Model.

Therefore, we will make use of a Gauss-Newton optimization technique, where control, estimated system and resulting system response are used to firstly estimate the predicted responses to the estimated system, with the help of an appropriate state estimation model. Then, as a second step, we can compute the difference of predicted and true system responses to update the estimated system parameters. An efficient decision rule to adjust a sufficient step-size of the optimization scheme, should then lead to acceptable estimates of the true system within a few steps. A more detailed description is now given in the course of the following section.

### 5.3.1 Non-Linear Least Squares Problem

#### Setting

For the sake of simplicity, let us consider the following test dataset

$$\mathcal{T}^{(5)} = \{((a_m, u_m))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.60}$$

where again $N_S \in \mathbb{N}$ is the size of the previously used training dataset for the Neural Networks and $N_T \in \mathbb{N}$ is the actual size of the test dataset. Let us further assume that in consistency with the previously discussed experiments, we define for a general acceleration element $a \in \mathcal{T}^{(5)}$,

$$a = (a_3, a_2, a_1) \in \mathbb{R}^{N\times 3}, \tag{5.61}$$

where $a_3$ is the acceleration profile of the occupant, $a_2$ of the chassis and $a_1$ of the wheel-suspensions. The term $u \in \mathbb{R}^N$ then again describes the road-profile with length $N \in \mathbb{N}$, specified in the latter experiments. Let us therefore clarify the following terms, which are necessary to derive an appropriate method for system identification of the Quarter-Car-Model:

For a general sample $(a, u) \in \mathcal{T}^{(5)}$, we consider

- $a \in \mathbb{R}^{N \times 3}$: True observable acceleration profiles of the system (multi-dimensional measurements).
- $u \in \mathbb{R}^N$: Road-profile as input to the system.
- $p \in \mathbb{R}^{10}$: General vector of the estimated relevant / unknown parameters of the Quarter-Car-Model.
- $p^* \in \mathbb{R}^{10}$: True values of the system parameters, as target values for the problem.
- $A(p) \in \mathbb{R}^{6 \times 6}$: Estimation of the system matrix, with $A(p) = \begin{bmatrix} D(p) & S(p) \\ I & O \end{bmatrix}$, where $D(p) \in \mathbb{R}^{3 \times 3}$ is the damping-coefficient matrix, $S(p) \in \mathbb{R}^{3 \times 3}$ the spring-coefficient matrix, $I \in \mathbb{R}^{3 \times 3}$, the identity matrix and $O \in \mathbb{R}^{3 \times 3}$ the null matrix. It is therefore assumed, that there is always a bijection $p \leftrightarrow A(p)$, such that we can uniquely map $p$ to the system matrix $A(p)$ and vice versa.

We can further note, that the data points $a^k \in \mathbb{R}^3$ correspond to the $k$-th element of the complete measurement sample $a$ with $k \in \{1, 2, ..., N\}$, resulting from an equidistant discrete time-horizon $\mathcal{D}_h([t_0, t_n])$ with $h = 0.01$. Besides, we have assumed zero-valued initial acceleration for each sample, denoted by $a^0 = 0 \in \mathbb{R}^3$, which also holds for the initial velocity $v^0 = 0 \in \mathbb{R}^3$ and the initial displacement / state $q^0 = 0 \in \mathbb{R}^3$.

### Least-Squares Problem

The following section shows, how one can derive an appropriate update-scheme to solve iteratively solve a non-linear least-squares problem, following [77, 83]. We define a non-linear least-squares problem as an optimization problem to minimize a loss-function $\mathcal{L} \in \mathcal{C}^2(\mathbb{R}^{10}, \mathbb{R})$ by

$$\min_p \mathcal{L}(p) = \frac{1}{2} \sum_{k=1}^{N} \sum_{i=1}^{3} \left( r_i(t^k, p) \right)^2 \tag{5.62}$$

for a twice continuously differentiable residual function $r \in \mathcal{C}^2([t_0, t_n] \times \mathbb{R}^{10}, \mathbb{R}^3)$, evaluated at $N \in \mathbb{N}$ time-steps, resulting from an equidistant time-discretization $\mathcal{D}_h([t_0, t_n])$ with $t^k \in \mathcal{D}_h([t_0, t_n])$ for all $k \in \{1, 2, ..., N\}$ . A Gradient-Descent optimization technique could be used to iteratively find an approximation of the solution $p^*$ of Eq. (5.62). Nevertheless, updating the parameters using Gradient-Descent would independently update the parameters, without considering the dependencies between the system parameters. Since we regard a coupled dynamical system, it is necessary to use an appropriate optimization method, which makes use of the interrelations of the individual parameters.

We therefore need to make use of second order derivatives, since for the gradient of the loss-function $\mathscr{L}(p)$, it holds that the gradient with respect to the parameters is defined as

$$\nabla_p \mathscr{L}(p) = \left( \frac{\partial \mathscr{L}(p)}{\partial p_1}, \frac{\partial \mathscr{L}(p)}{\partial p_2}, ..., \frac{\partial \mathscr{L}(p)}{\partial p_{10}} \right)^T \in \mathbb{R}^{10} \tag{5.63}$$

with partial derivatives being explicitly given by

$$\frac{\partial \mathscr{L}(p)}{\partial p_j} = \sum_{k=1}^{N} \sum_{i=1}^{3} \frac{\partial r_i(t^k, p)}{\partial p_j} r_i(t^k, p) \tag{5.64}$$

for parameters, indexed by $j \in \{1,2,...,10\}$. Therefore, the derivatives with respect to the individual parameters $p_j$ are not related to different parameters of the system.

For a second order optimization technique, including computation of the Hessian $H(p) := \nabla_p^2 \mathscr{L}(p) \in \mathbb{R}^{10\times 10}$, we can recognize that for elements $H(p)_{nj} = \frac{\partial^2 \mathscr{L}(p)}{\partial p_n \partial p_j} \in \mathbb{R}$ for $n$, $j \in \{1,2,...,10\}$, it holds that

$$\begin{aligned} \frac{\partial^2 \mathscr{L}(p)}{\partial p_n \partial p_j} &= \sum_{k=1}^{N} \sum_{i=1}^{3} \left( \frac{\partial}{\partial p_n} \left( \frac{\partial r_i(t^k, p)}{\partial p_j} r_i(t^k, p) \right) \right) \\ &= \sum_{k=1}^{N} \sum_{i=1}^{3} \left( \frac{\partial r_i(t^k, p)}{\partial p_n} \frac{\partial r_i(t^k, p)}{\partial p_j} + r_i(t^k, p) \frac{\partial^2 r_i(t^k, p)}{\partial p_n \partial p_j} \right). \end{aligned} \tag{5.65}$$

We recognize that the entries of the Hessian contain the product of the first order derivatives $\frac{\partial r_i(t^k,p)}{\partial p_n} \frac{\partial r_i(t^k,p)}{\partial p_j}$ as well as the second order derivatives $\frac{\partial^2 r_i(t^k,p)}{\partial p_n \partial p_j}$, which describes the dependencies of the individual parameters on each other.

Regarding the Semi-Explicit Euler method, it is already challenging to compute the first order derivatives, as for large $k \in \{1,2,...,N\}$, gradient-computation requires applying the chain-rule of differentiation multiple times. It is therefore obvious, that computation of second order derivatives is even more challenging and results in high time-consumption. As we want to develop an efficient method for parameter estimation of the Quarter-Car-Model, let us therefore consider the following.

Making use of Eq. (5.80), we can define the vector of residuals for all steps $k \in \{1,2,...,N\}$ by

$$R(p) := \begin{pmatrix} r(t^1, p) \\ r(t^2, p) \\ \vdots \\ r(t^N, p) \end{pmatrix} \in \mathbb{R}^{3N}. \tag{5.66}$$

The Jacobian matrix, which we denote by $J(p) \in \mathbb{R}^{3N\times 10}$, is therefore given as

$$J(p) := \begin{bmatrix} J_1(p) \\ J_2(p) \\ \vdots \\ J_N(p) \end{bmatrix}, \tag{5.67}$$

where

$$J_k(p) := \begin{bmatrix} \nabla_p^T r_3(t^k,p) \\ \nabla_p^T r_2(t^k,p) \\ \nabla_p^T r_1(t^k,p) \end{bmatrix} = \begin{bmatrix} \frac{\partial r_3(t^k,p)}{\partial p_1} & \dots & \frac{\partial r_3(t^k,p)}{\partial p_{10}} \\ \frac{\partial r_2(t^k,p)}{\partial p_1} & \dots & \frac{\partial r_2(t^k,p)}{\partial p_{10}} \\ \frac{\partial r_1(t^k,p)}{\partial p_1} & \dots & \frac{\partial r_1(t^k,p)}{\partial p_{10}} \end{bmatrix}, \tag{5.68}$$

which contains all partial derivatives of the computed residuals, using the Semi-Explicit Euler method as state estimation model. Since

$$(J^T R(p))_j = \sum_{k=1}^{N}\sum_{i=1}^{3} \frac{\partial r_i(t^k,p)}{\partial p_j} r_i(t^k,p) = \frac{\partial \mathscr{L}(p)}{\partial p_j}, \tag{5.69}$$

it holds that the loss-function can also be computed by $\nabla_p \mathscr{L}(p) = J^T(p)R(p) \in \mathbb{R}^{10}$. following from Eq. (5.64) for parameter with index $j \in \{1,2,...,10\}$. As a last step, the Fisher-matrix, as approximation of the Hessian, can be computed by $F(p) = J^T(p)J(p) \in \mathbb{R}^{10\times 10}$. Thus, the matrix entries $F(p)_{nj}$ are given by

$$F(p)_{nj} = \sum_{k=1}^{N}\sum_{i=1}^{3} \left( \frac{\partial r_i(t^k,p)}{\partial p_n} \frac{\partial r_i(t^k,p)}{\partial p_j} \right) \tag{5.70}$$

and as a consequence, by comparing Eq. (5.65) and Eq. (5.70), it further holds that

$$H(p)_{nj} = F(p)_{nj} + \sum_{k=1}^{N}\sum_{i=1}^{3} \left( r_i(t^k,p) \frac{\partial^2 r_i(t^k,p)}{\partial p_n \partial p_j} \right). \tag{5.71}$$

If

$$\sum_{k=1}^{N}\sum_{i=1}^{3} \left( r_i(t^k,p) \frac{\partial^2 r_i(t^k,p)}{\partial p_n \partial p_j} \right) \approx 0 \tag{5.72}$$

for all $n$, $j \in \{1,2,...,10\}$, then

$$F(p) \approx H(p). \tag{5.73}$$

An iterative optimization scheme to solve the non-linear least-squares problem, can therefore be derived, using the Fisher-matrix as approximation to the Hessian. The update rule can then be described by

$$p^{l+1} = p^l - \alpha^l F(p^l)^{-1} J^T(p^l) R(p^l), \tag{5.74}$$

where $\alpha^l > 0$ is a step-size for iteration step $l \in \mathbb{N}$. The above shown update-scheme of Eq. (5.74) can further be derived, if one tries to solve a linearized least-squares problem, which does not require computation of second order derivatives, as shown in [35] with an adjustable step-size.

### State Estimation Model

Following Eq. (4.93) and Eq. (4.94), the Semi-Explicit Euler scheme for the data generation process can be used to define an appropriate state estimation model, using the parameter dependent system matrices and a time-step $h = 0.01$ by

$$v^{k+1} = v^k + h\left(D(p)v^k + S(p)q^k + b^k\right) \tag{5.75}$$

$$q^{k+1} = q^k + hv^{k+1} \tag{5.76}$$

$$y^{k+1}(p) = D(p)v^{k+1} + S(p)q^{k+1} + b^{k+1}, \tag{5.77}$$

where $b^k = \left(0, 0, p_u u^k\right)^T$ describes the non-linear driving term for $k \in \{1, 2, ..., N\}$ with corresponding spring-coefficient $p_u := p^*_{10} - p^*_9 \in \mathbb{R}$. In consistency with $a^k = (a^k_3, a^k_2, a^k_1)^T \in \mathbb{R}^3$, let the model estimates be indexed by $y^k(p) = (y^k_3(p), y^k_2(p), y^k_1(p))^T \in \mathbb{R}^3$.
It then holds, that for the true value of the system parameters, $p^*$, the exact value of the acceleration is given by

$$a^k = y^k(p^*) \tag{5.78}$$

for all $k \in \{1, 2, ..., N\}$. If in contrast, we have parameter estimates $p \neq p^*$, the **discrete residuals**, in our case a three dimensional vector for each time-step, can then be defined as

$$r^k(p) = \left(y^k(p) - a^k\right) = \begin{pmatrix} y^k_3(p) - a^k_3 \\ y^k_2(p) - a^k_2 \\ y^k_1(p) - a^k_1 \end{pmatrix} = \begin{pmatrix} r^k_3(p) \\ r^k_2(p) \\ r^k_1(p) \end{pmatrix} \in \mathbb{R}^3. \tag{5.79}$$

The herein described setting then leads to a non-linear least-squares problem, trying to find the optimal parameter value $p^*$ by minimizing the discrete residuals for each time-step. Replacing the residuals resulting from the state estimation, $r^k(p)$, with the evaluation of the continuously differentiable residual function of the previous section, $r(t^k, p)$ for $k \in \{1, 2, ..., N\}$, one can apply the Gauss-Newton algorithm with the help of the Semi-Explicit Euler, used as a state estimation method.

### Gauss-Newton Algorithm

We can now give an overview about a simple Gauss-Newton optimization technique, considering the previously described Fisher-matrix as approximation to the Hessian. The resulting iteration scheme, which is stated below, then serves as an approximative second order method, without requiring computation of the second order partial derivatives.

As inputs to the algorithm, we require a starting point for the system parameters, denoted by $p^0 \in \mathbb{R}^{10}$, the acceleration data $a \in \mathbb{R}^{N\times 3}$, and the road-profile $u \in \mathbb{R}^N$. Further, we need to predefine a constant step-size $\alpha > 0$ for the Wolfe-Condition to apply the Armijo-Rule [30] for control of the step-size. Furthermore, we need a boundary value of $\varepsilon > 0$ as termination criterion for the algorithm.

### Algorithmic Scheme

1. **Set** the current value of parameters for iteration step $l \leq T$ as $p^l$, where $T \in \mathbb{N}$ is the maximum iteration number. Further choose an initial step-size $a > 0$.
2. **Compute** the residuals $r_i^k(p^l)$ for all $i \in \{1,2,...,10\}$ and $k \in \{1,2,...,N\}$. Then, the resulting residual vector $R(p^l)$ can be defined.
3. **Compute** the Jacobian $J(p^l)$, the Fisher-matrix $F(p^l) = J^T(p^l)J(p^l)$ and the inverse $F(p^l)^{-1}$, if it exists. Otherwise, a Pseudo-Inverse of the Fisher-matrix is required.
4. **Set** $g^l = (F(p^l))^{-1}J^T(p^l)R(p^l)$ as direction of optimization.
5. **Test** if $\mathscr{L}(p^l - \alpha g^l) \leq \mathscr{L}(p^l) - \alpha \left(g^l\right)^T J^T(p^l)R(p^l)$.
   **If not**, we **set** $\alpha = \frac{\alpha}{2}$ and test the condition again, until it is satisfied. If the condition is tested for $\nu \in \mathbb{N}$ times, until it is satisfied, set $\alpha^l = \frac{\alpha}{2^{\nu-1}}$.
6. **Set** $p^{l+1} = p^l - \alpha^l g^l$.
7. **Compute** $\Delta p = \frac{1}{10}\sum_{i=1}^{10} \left|p_i^{l+1} - p_i^l\right| = \frac{1}{10}\sum_{i=1}^{10} \left|\alpha^l g_i^l\right|$.
   **If** $\Delta p < \varepsilon$ or $l > T$, the optimization is **finished**.
   **If not**, we **set** $p^l = p^{l+1}$ and **continue** at 2., until the termination criterion is satisfied.

The Tensorflow source code is given in Figure A.16, Figure A.17, Figure A.18 and Figure A.19, where we can observe that it is possible to implement a Gauss-Newton method, using Tensorflow variables, such that the entire test dataset can simultaneously be used to estimate the corresponding parameters. Nevertheless, the above described algorithmic scheme gives a short simplified overview, about the general steps, we have to consider.

It is obvious that for this specific Gauss-Newton method for the parameter estimation of the Quarter-Car-Model, no Neural Network is required. Thus, we are going to investigate, if for the described optimal case, having full-observability of the unnoisy measurements, the parameters can uniquely be determined from the acceleration profiles, by iteratively solving the non-linear least-squares problem.

### 5.3.2 Experiment 5 : System Identification of Full-Observable Systems

#### Problem

We consider the previously defined dataset with

$$\mathcal{T}^{(5)} = \{((a_m, u_m))\}_{m=N_S+1}^{N_S+N_T},$$

such that the acceleration and the road-profile can uniquely be described by the index $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$ as elements of the test dataset, different from the samples belonging to the training data with $m \in \{1,2,...,N_S\}$, which are not being considered here. Further, let the corresponding vector of true parameter values be described by $p_m \in \mathbb{R}^{10}$, where the individual parameters are denoted by $p_{m;i}$ for $i \in \{1,2,...,10\}$.

The **problem**, we want to investigate is: **Given a randomly chosen initial value of the system parameters, is it possible to achieve a sufficiently good approximation of the true parameter values, using the above described Gauss-Newton method for full-observability of the acceleration profiles?**

**Method**

We want to solve

$$\min_p \mathscr{L}(p) = \frac{1}{2} \sum_{k=1}^{N} \sum_{i=1}^{3} \left( r_i^k(p) \right)^2,$$

where

$$r^k(p) = \left( y^k(p) - a^k \right) = \begin{pmatrix} y_3^k(p) - a_3^k \\ y_2^k(p) - a_2^k \\ y_1^k(p) - a_1^k \end{pmatrix} \tag{5.80}$$

for all acceleration profiles $a \in \mathfrak{T}^{(5)}$.

Let further the estimated parameter value at iteration step $l \in \mathbb{N}$ for each true underlying value $p_{m;i}$ for $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$ and $i \in \{1,2,..,10\}$ be denoted by $p_{m;i}^l$. Therefore, let us recall, that the parameters of a reference model have been described in Eq. (4.103) by

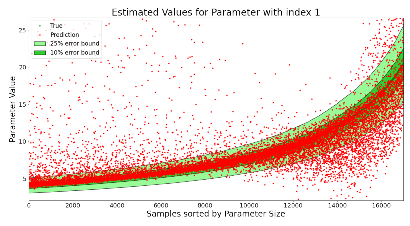
$$p_\mu = \left( \frac{C_3}{m_3}, \frac{K_3}{m_3}, \frac{C_3}{m_2}, \frac{C_2}{m_2}, \frac{K_3}{m_2}, \frac{K_2}{m_2}, \frac{C_2}{m_1}, \frac{C_2 + C_1}{m_1}, \frac{K_2}{m_1}, \frac{K_2 + K_1}{m_1} \right)^T,$$

such that the elements, denoted by $p_{\mu;i}$ for all $i \in \{1,2,...,10\}$ have fixed values. We can then define a randomly drawn deviation for all elements and all parameters of the test dataset, denoted by $\delta_{m;i}$ with $\delta_{m;i} \sim \mathcal{N}\left(0, 0.1^2\right)$. Then the initial parameter values are chosen, following $p_{m;i}^0 = p_{\mu;i}\left(1 + \delta_{m;i}\right)$. The previously described Gauss-Newton method can then be used to iteratively solve the non-linear least-squares problem. Furthermore, we choose a length of the acceleration profiles of $N = 100$, which has been experienced from several experiments to be an appropriate size for the Gauss-Newton method. Smaller realizations of $N$ have shown to cause problems when computing the inverse of the Fisher-matrix, larger realizations are inefficient, as the computation of the chained partial derivatives for large $N$ are time-consuming.

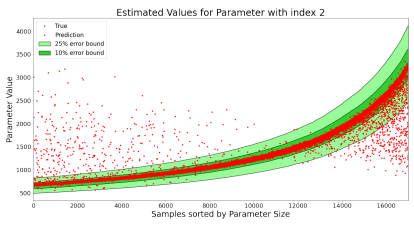
**Results**

The here shown results are taken from a Gauss-Newton method with initial step-size $\alpha = 0.1$. Although, we use the Armijo-Rule to regularize the step-size, random initialization causes problems to compute the inverse Fisher-matrix, although the condition is satisfied. This problem obviously occurs from the poor random initialization of the system parameters, as we compute the inverse Fisher-matrix for all samples of the test dataset simultaneously. This means computation of the inverse for $T = 17000$ components.

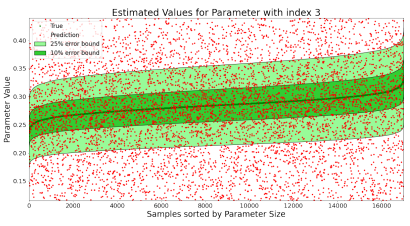
Nevertheless, let us investigate the parameter estimation results for a smaller step-size, such that the method does not cause numerical issues.



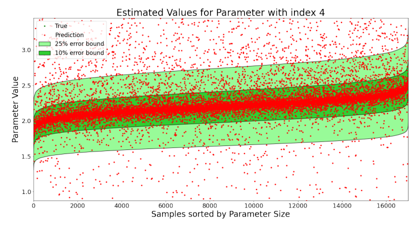
(a) Estimation of parameter 1.



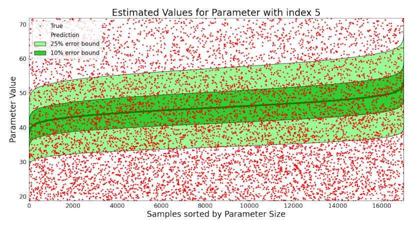
(b) Estimation of parameter 2.



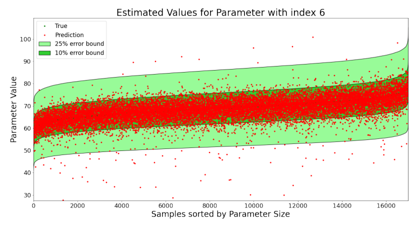
(c) Estimation of parameter 3.



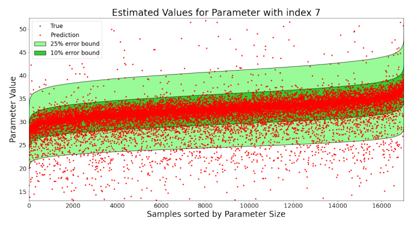
(d) Estimation of parameter 4.



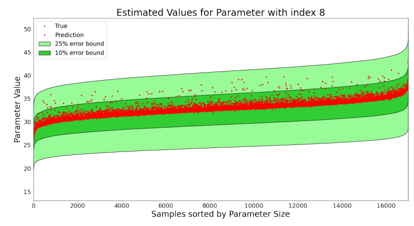
(e) Estimation of parameter 5.



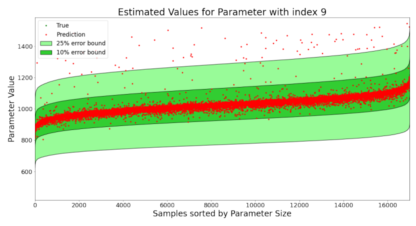
(f) Estimation of parameter 6.



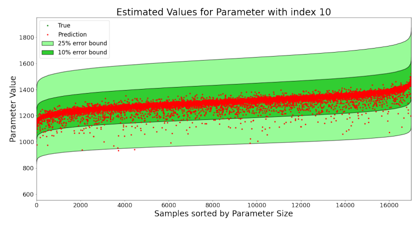
(g) Estimation of parameter 7.



(h) Estimation of parameter 8.



(i) Estimation of parameter 9.



(j) Estimation of parameter 10.

FIGURE 5.17: Gauss-Newton parameter estimation for full-observability and random parameter initialization. A precise description of the plots is described by the caption of Figure 5.4. We recognize the estimation of the Quarter-Car-Model parameters, after 100 optimization steps with initial step-size $\alpha = 0.1$ in Figure 5.17$a$ - Figure 5.17$j$. The visualized estimates correspond to the error distribution given in Table 5.11.

It can be observed with the help of Figure 5.17, that there are large deviations for the parameter estimation of the developed Gauss-Newton method throughout nearly all parameters. Obviously, we have overestimations of small parameter values for $p_1$ and $p_2$ and underestimates of larger values (Figure 5.17*a* and Figure 5.17*b*). Furthermore, there is a significant tendency to overestimate the values of $p_4$, as can be seen in Figure 5.17*d*, though the most parameters are within the 10% error boundary. More robust estimates can be seen for parameters $p_6$ - $p_{10}$, although we can also recognize underestimations of $p_7$ in Figure 5.17*g*. Apparently, following Figure 5.17*c* and Figure 5.17*e*, it is not possible in this setting, to achieve an appropriate estimation of neither $p_3$ nor $p_5$.

| Step | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 1 | 50.88248 | 0.38816 | 2.17220 | 0.24735 | 0.57012 | 0.68435 | 0.83188 | 0.16812 |
| 100 | 2 | 4.60309 | 0.05236 | 0.18841 | 0.28641 | 0.89941 | 0.93441 | 0.96171 | 0.03829 |
| 100 | 3 | 268.96414 | 7.22228 | 16.27473 | 0.01371 | 0.06088 | 0.11312 | 0.22976 | 0.77024 |
| 100 | 4 | 2.45321 | 0.09652 | 0.21055 | 0.38694 | 0.71618 | 0.80047 | 0.88871 | 0.11129 |
| 100 | 5 | 5.46888 | 0.73327 | 0.66495 | 0.00918 | 0.04594 | 0.09394 | 0.23471 | 0.76529 |
| 100 | 6 | 0.85683 | 0.04338 | 0.04304 | 0.17094 | 0.66388 | 0.93235 | 0.99453 | 0.00547 |
| 100 | 7 | 1.51068 | 0.06089 | 0.11921 | 0.27976 | 0.69871 | 0.85141 | 0.95118 | 0.04882 |
| 100 | 8 | 0.19972 | 0.00725 | 0.01417 | 0.81041 | 0.98482 | 0.99453 | 1.00000 | 0.00000 |
| 100 | 9 | 1.89653 | 0.01080 | 0.03369 | 0.72571 | 0.98188 | 0.99247 | 0.99506 | 0.00494 |
| 100 | 10 | 0.26131 | 0.00946 | 0.01882 | 0.77118 | 0.96082 | 0.99112 | 0.99988 | 0.00012 |

TABLE 5.11: Experiment 5: Error distribution of Gauss-Newton parameter estimation for full-observability and random parameter initialization. The table shows the error distribution and more statistical values of the observed estimation results. The whole test dataset $\mathcal{T}^{(5)}$ is considered for the computation. The description of the columns is shown in the caption of Table 5.1. The table corresponds to the visualization of Figure 5.17.

We have already seen in Figure 5.17, that given random initialization of the parameters for the Gauss-Newton method, several problems occur, except for a subset of the relevant parameters. The same investigation can be observed from the corresponding Table 5.11, where it is for instance obvious that more than 75% of the estimates for $p_3$ and $p_5$ lie out of the 25% error bound. Furthermore, the most robust estimates can be observed for parameters $p_6$ to $p_{10}$, as more than 85% lie within the 10% error bound, where a relatively large mean standard deviation of 0.11921 for $p_7$ is characteristic for the significant underestimation we have already observed in Figure 5.17*g*.

### Summary

The Gauss-Newton method for parameter estimation of the Quarter-Car-Model with random initialization shows, that it is difficult to achieve appropriate results in this setting. As we can observe a strong tendency to underestimate small values of $p_1$ and $p_2$, where large values are underestimated, this could occur from poor initialization of the occupant-related coefficients. It has already been investigated in Figure 4.14 for the data generation of the training and test dataset, that parameters $p_1$ and $p_2$ are generated, using a different distribution, compared to the remaining coefficients of the coupled dynamical system. The mass of the seat and the occupant have been generated from a uniform distribution, which makes it difficult to randomly choose an initial value for the Gauss-Newton method. Fortunately, **Experiment 1** - **Experiment 4** have shown, that the parameter values can robustly be predicted from data driven models. We can therefore use one of the pre-trained Neural Networks of the previously investigated problems to initialize the parameter values to solve the non-linear least-squares problem. We then want to observe, if the algorithm is stabilized using this more sophisticated initialization. Furthermore, we will compare, if the problems stated in **Experiment 5** vanish in this case.

### 5.3.3 Experiment 6 : Parameter Initialization via Neural Networks

#### Problem

As it has already been summarized in **Experiment 5**, we now want to observe, if initialization by a Neural Network for parameters $p_1$ and $p_2$ for the data samples contained in $\mathcal{T}^{(5)}$, improves the Gauss-Newton method in multiple senses.

The **problem**, we want to investigate is: **Given a better initialization of the first and second parameter, as a starting point for the Gauss-Newton method, do we achieve significantly better results for the entire parameters of the system?**

#### Method

The method differs from that described in **Experiment 5** only by choosing the initial values for parameter $p_1$ and $p_2$ with the help of a pre-trained Neural Network. We have already investigated, that the unlabelled training approach for parameter estimation is more robust for unknown test samples. Let us further notice, that since we again assume to have a full-observable system, the acceleration profiles $a \in \mathbb{R}^{N \times 3}$ with $a = (a_3, a_2, a_1)$ contain the occupant's acceleration $a_3$, as well as the chassis acceleration $a_2$. We made use of these data combination in **Experiment 1** - **Experiment 4**. It is therefore obvious, that we can use the pre-trained Neural Network of **Experiment 2** with

$$\begin{aligned} f_{\theta_\beta}^{(2)} &: \mathbb{R}^{N\times 2} \to \mathbb{R}^2 \\ &(a_3, a_2) \mapsto \begin{pmatrix} \tilde{p}_1(\theta_\beta) \\ \tilde{p}_2(\theta_\beta) \end{pmatrix}. \end{aligned}$$

Obviously, we can choose the following initial value for each sample of the test dataset by

$$p_{m;1}^0 = \left(f_{\theta_\beta}^{(2)}(a_{m;3}, a_{m;2})\right)_1, \quad p_{m;2}^0 = \left(f_{\theta_\beta}^{(2)}(a_{m;3}, a_{m;2})\right)_2 \tag{5.81}$$

and for the remaining parameters again use random initialization with $p_{m;i}^0 = p_{\mu;i}(1 + \delta_{m;i})$ for $i \in \{3, 4, ..., 10\}$ for all $m \in \{N_S + 1, N_S + 2, ..., N_S + N_T\}$.

We apply the Gauss-Newton method analogously to the first experiment of this section, where we now use a Neural Network based initialization of parameter $p_1$ and $p_2$. Again, we choose a vector length of $N = 100$ for the input data, which will remain the same for the entire section, as far as Gauss-Newton methods for parameter estimation are concerned. Furthermore, we try an initial step-size of $\alpha = 1.0$ and apply the Armijo-Rule to adjust it throughout the optimization process.

### Results

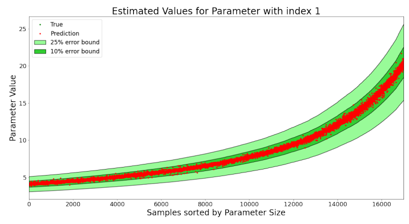
In order to have comparable results, we use the common plots and tables, we have experienced in the course of the last sections. We note, that in contrast to the 100 iterations of the previously random initialization in **Experiment 5**, we only have 30 iterations, until the algorithm terminates. The more sophisticated initialization of parameters $p_1$ and $p_2$ leads therefore to an increasing stability of the algorithm such that we can apply larger step-sizes for each optimization step.

The estimation results after 30 iterations of the Gauss-Newton method for a Neural Network supported initialization of the first two parameters, can easily be summarized, if we regard Figure 5.18. Obviously, the parameter estimates significantly lie within the 10% error bound, where we can even recognize very small deviations for parameter $p_2$ and $p_8$ to $p_{10}$. Furthermore, if we consider the results for the test dataset in **Experiment 5**, we can even recognize that the estimates given by the Neural Network for the test dataset can be significantly improved using the Gauss-Newton method. Nevertheless, there is still no possibility to appropriately estimate parameters $p_3$ and $p_5$ as can be seen in Figure 5.18*c* and Figure 5.18*e*. Apparently, the value of those parameters is not as sensitive to the approximation quality of the model states, compared to the remaining parameters.

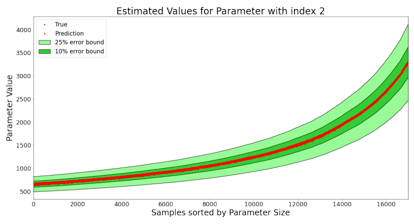
| Step | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| 30 | 1 | 0.11898 | 0.01177 | 0.01193 | 0.56400 | 0.98482 | 0.99941 | 1.00000 | 0.00000 |
| 30 | 2 | 0.04901 | 0.00545 | 0.00528 | 0.84659 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 30 | 3 | 6.87345 | 0.45187 | 0.51629 | 0.03400 | 0.13776 | 0.23665 | 0.45700 | 0.54300 |
| 30 | 4 | 0.08483 | 0.00602 | 0.00678 | 0.83176 | 0.99706 | 1.00000 | 1.00000 | 0.00000 |
| 30 | 5 | 1.18007 | 0.15325 | 0.12715 | 0.04647 | 0.21759 | 0.41547 | 0.81076 | 0.18924 |
| 30 | 6 | 0.14083 | 0.01011 | 0.01070 | 0.64100 | 0.98971 | 0.99971 | 1.00000 | 0.00000 |
| 30 | 7 | 0.20535 | 0.00830 | 0.01253 | 0.76329 | 0.98318 | 0.99735 | 1.00000 | 0.00000 |
| 30 | 8 | 0.00535 | 0.00053 | 0.00057 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 30 | 9 | 0.02713 | 0.00152 | 0.00206 | 0.98947 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 30 | 10 | 0.01132 | 0.00065 | 0.00080 | 0.99982 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |

TABLE 5.12: Experiment 6: Error distribution of Gauss-Newton parameter estimation for full-observability and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. The table shows the error distribution and more statistical values of the observed estimation results. The whole test dataset $\mathcal{T}^{(5)}$ is considered for the computation. The description of the columns is shown in the caption of Table 5.1. The table corresponds to the visualization of Figure 5.18.

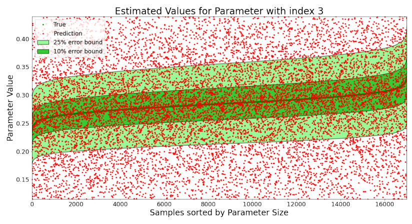
The visualized results in Figure 5.18 can be verified with Table 5.12. If we compare the results for the apparently identifiable parameters with index $i \in \{1,2,4,6,7,8,9,10\}$ and the practically non-identifiable remaining parameters with $i \in \{3,5\}$, we can recognize that more than 98.3% of the identifiable parameters lie within the 5% error bound, while we have



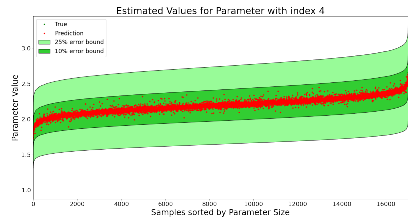
(a) Estimation of parameter 1.



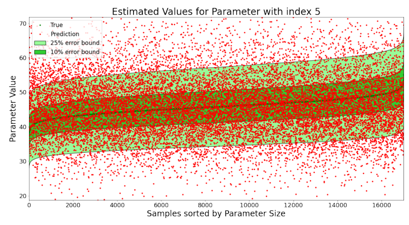
(b) Estimation of parameter 2.



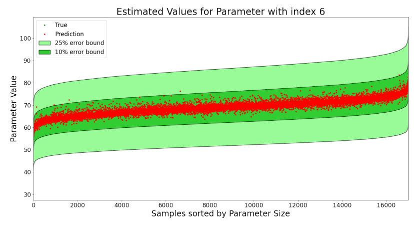
(c) Estimation of parameter 3.



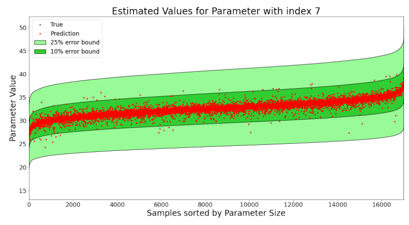
(d) Estimation of parameter 4.



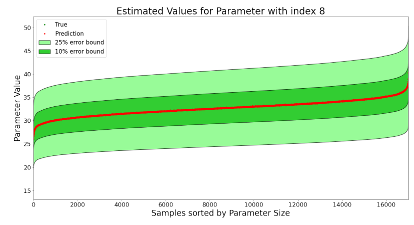
(e) Estimation of parameter 5.



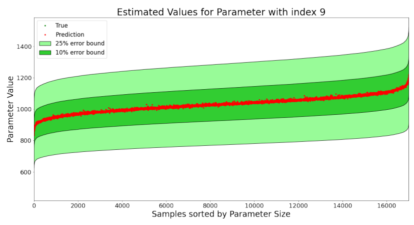
(f) Estimation of parameter 6.



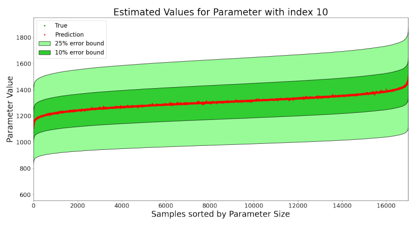
(g) Estimation of parameter 7.



(h) Estimation of parameter 8.



(i) Estimation of parameter 9.



(j) Estimation of parameter 10.

FIGURE 5.18: Gauss-Newton parameter estimation for full-observability and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. A precise description of the plots is described by the caption of Figure 5.4. We recognize the estimation of the Quarter-Car-Model parameters, after 30 optimization steps with initial step-size $\alpha = 1.0$ in Figure 5.18$a$ - Figure 5.18$j$. The visualized estimates correspond to the error distribution given in Table 5.12.

13.78% for $p_3$ and 21.76% for $p_5$. Furthermore, maximal deviation, mean relative deviation and mean standard deviation show significantly larger values for $p_3$ and $p_5$, compared to the remaining parameters of the Quarter-Car-Model. Comparing this to the random initialization of $p_1$ and $p_2$, we can investigate that now 100% of the identifiable parameters lie within the 25% error bound, where 54% of the estimates for $p_3$ have deviations larger than 25% and 19% for $p_5$. Nevertheless, this deviation, even for the non-identifiable parameters is much better compared to the results of **Experiment 5**.

### Summary

Summarizing the observed results, we need to define the index set of practically identifiable parameters by $\{1,2,4,6,7,8,9,10\}$ and the index set of practically non-identifiable parameters by $\{3,5\}$. It has been shown in [87], that although all parameters of a model can be structurally identifiable for full-observable systems, it can happen that there exist practically non-identifiable parameters resulting from the given type of data. Further experiments, to overcome this problem have also been made, by using loss-functions similar to **Experiment 1** and **Experiment 2** for the acceleration profile of the chassis. The observations have shown the same as for the Gauss-Newton method: The parameters $p_3$ and $p_5$ are not identifiable within an appropriate error range for the test dataset. Although we can increase the depth of the used Neural Network to guarantee a minimization of the training error, several Neural Network architectures do not lead to a good generalization for the test samples.

We can therefore conclude, that in this setting or for this type of data, it is not possible to identify the parameters $p_3$ and $p_5$ from the observed data. Nevertheless, poor estimates are not sensitive to the stability of the optimization method and a sophisticated initialization of $p_1$ and $p_2$ with the help of a pre-trained Convolutional Neural Network for parameter estimation leads to a faster and more robust Hybrid Gauss-Newton estimation method.

## 5.3.4 Experiment 7 : System Identification of Identifiable Parameters

### Problem

The previous experiment has stated, that there is a set of practically identifiable parameters and a complementary set of unidentifiable parameters. We want to solve again Eq. (5.62) for a full-observable system, but assume now, that the true values of the unidentifiable parameters, or at least an acceptable range of the true values, are a priori known. The observable test dataset is therefore given by

$$\mathcal{T}^{(7)} = \{((a_m, u_m), (p_{m;3}, p_{m;5}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.82}$$

which means that the unidentifiable parameters are not considered as target values for the iterations of the Gauss-Newton method.

The **problem**, we want to investigate is: **Given an appropriate estimate of the unidentifiable parameters $p_3$ and $p_5$, can the results shown in the previous experiment, be improved by eliminating this uncertainty?**

### Method

Instead of applying the Gauss-Newton method to the entire parameter vector $p_m \in \mathbb{R}^{10}$, for all $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$, let us now consider the reduced parameter vector of identifiable coefficients by

$$\overline{p} := (p_1, p_2, p_4, p_6, p_7, p_8, p_9, p_{10})^T \in \mathbb{R}^8. \tag{5.83}$$

One can then verify, that for the Gauss-Newton method, we can not consider the partial derivatives $\frac{\partial \mathscr{L}(p)}{\partial p_j}$ for $j \in \{3,5\}$. As a consequence, the gradient reduces its dimension, such that $\nabla_p \mathscr{L}(\overline{p}) \in \mathbb{R}^8$. Furthermore, we have a reduction of the Jacobian with $J(\overline{p}) \in \mathbb{R}^{3N \times 8}$ and consequently the Fisher-matrix of the reduced parameters is given by $F(\overline{p}) \in \mathbb{R}^{8 \times 8}$.

Having a current estimate of the reduced, identifiable parameters for iteration $l \in \mathbb{N}$, given by $\overline{p}^l$, one can update the estimated parameter values with the Gauss-Newton step

$$\overline{p}^{l+1} = \overline{p}^l - \alpha^l \left(F(\overline{p}^l)\right)^{-1} J^T(\overline{p}^l) R(\overline{p}^l). \tag{5.84}$$

Using the updated value of the identifiable parameters, then requires to map $\overline{p}^{l+1}$ to the vector of all relevant parameters $p^{l+1}$, where

$$p^{l+1} = \left(\overline{p}_1^{l+1}, \overline{p}_2^{l+1}, p_3, \overline{p}_4^{l+1}, p_5, \overline{p}_6^{l+1}, \overline{p}_7^{l+1}, \overline{p}_8^{l+1}, \overline{p}_9^{l+1}, \overline{p}_{10}^{l+1}\right)^T, \tag{5.85}$$

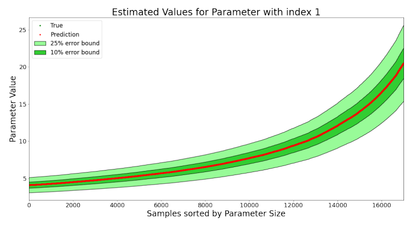
such that the system matrix $A(p^{l+1})$ can be defined for the model to estimate the updated states $y(p^{l+1})$ of the system. Again, as it has shown to improve robustness of the method, we use the Neural Network initialization of parameters $p_1$ and $p_2$ and a series-length of $N = 100$.

### Results

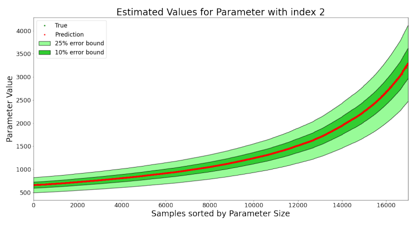
We can directly compare the herein shown results to the previous **Experiment 7**, where we have considered the entire parameter vector for the Gauss-Newton method. The herein shown results for the reduced parameter estimation problem, have been observed after 25 Gauss-Newton steps with application of the Armijo-Rule to adjust the step-size.

It is more than obvious, that the estimates of the parameters for the identifiable set are almost optimal, if we consider Figure 5.19. Most of the estimates for the reduced problem with the shown Gauss-Newton method, result in values close to the optimal value. We can only observe small deviations for parameter $p_7$, which has already in previous experiments been experienced to be more difficult to identify compared to other coefficients.

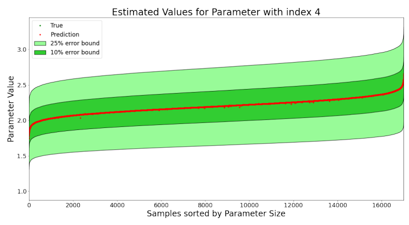
The apparently optimal results for this setting can be verified using Table 5.13. Where we had values of more than 56% in the 1% error range for the identifiable parameters in **Experiment 6**, we now end up at more than 99.63% for all identifiable parameters in the same range.



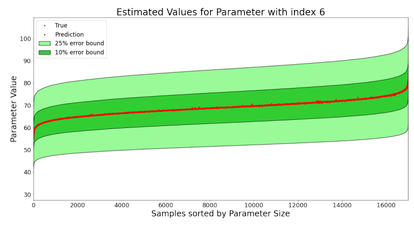
(a) Estimation of parameter 1.



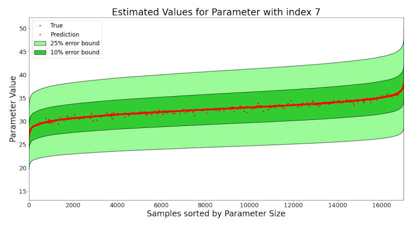
(b) Estimation of parameter 2.



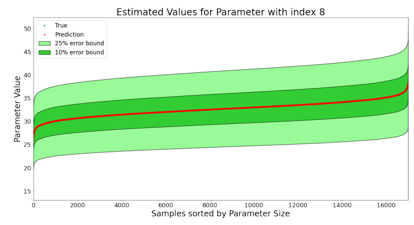
(c) Estimation of parameter 4.



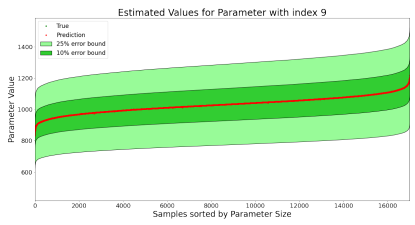
(d) Estimation of parameter 6.



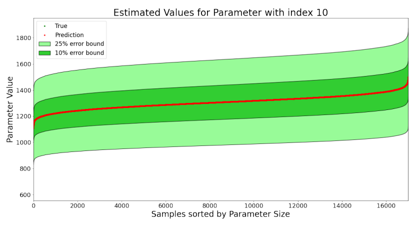
(e) Estimation of parameter 7.



(f) Estimation of parameter 8.



(g) Estimation of parameter 9.



(h) Estimation of parameter 10.

FIGURE 5.19: Gauss-Newton parameter estimation for full-observability, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. A precise description of the plots is described by the caption of Figure 5.4. We recognize the estimation of the Quarter-Car-Model parameters, after 10 optimization steps with initial step-size $\alpha = 1.0$ in Figure 5.19*a* - Figure 5.19*h*. The visualized estimates correspond to the error distribution given in Table 5.13.

Besides, we have 100% of the estimates in the 5% range. Moreover, the small values of the maximal deviation, for instance 3.7% for $p_7$, compared to 20.5% for **Experiment 6** and mean relative deviations smaller than $10^{-3}$, show that the method is capable to uniquely determine the parameter values of the reduced problem .

## Summary

We have observed optimal parameter estimation results for a problem, where we have full-observability of the measurements and a partial initialization of the parameters via a pre-trained Convolutional Neural Network. In addition, we have excluded the non-identifiable

| Step | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 1 | 0.01079 | 0.00004 | 0.00024 | 0.99994 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 2 | 0.02499 | 0.00005 | 0.00046 | 0.99959 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 4 | 0.02229 | 0.00004 | 0.00037 | 0.99976 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 6 | 0.01121 | 0.00006 | 0.00044 | 0.99982 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 7 | 0.03714 | 0.00017 | 0.00129 | 0.99635 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 8 | 0.00155 | 0.00001 | 0.00006 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 9 | 0.00680 | 0.00003 | 0.00020 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |
| 25 | 10 | 0.00239 | 0.00001 | 0.00009 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.00000 |

TABLE 5.13: Experiment 7: Error distribution of Gauss-Newton parameter estimation for full-observability, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. The table shows the error distribution and more statistical values of the observed estimation results. The whole test dataset $\mathcal{T}^{(7)}$ is considered for the computation. The description of the columns is shown in the caption of Table 5.1. The table corresponds to the visualization of Figure 5.19.

parameters, which we have observed in the previous examples to be most difficult to estimate. The Gauss-Newton method is capable to solve the non-linear least-squares problem, if we consider the best possible setting for the estimation problem.

Finally, we will consider another setting, where the conditions are not optimal at all. We have already clarified that there is a difference between structurally and practically identifiability of the system parameters. The practically non-identifiable parameters have therefore been excluded in **Experiment 7** and we have restricted the problem to those parameters, which are practically and structurally identifiable for a full-observable coupled dynamical systems. Nevertheless, we have seen that the structurally identifiability property is hurt, if we only consider partially observable systems. Therefore, the next section deals with such systems, in which only the acceleration profile of the occupant is accessible for the Gauss-Newton estimation method.

## 5.4 System Identification of Uncertain Systems

The last investigations, we are going to consider, is system identification of uncertain systems. It has already been shown, that Hybrid Models [15, 52], can in general be described as a combination of individual methods, which improve the performance of the combined model, in comparison to the performance of the individual models. Moreover, we have already experienced several experiments, where we used such Hybrid Models for a two-dimensional parameter estimation problem. We have shown the training of an unlabelled, hybrid Convolutional Neural Network in **Experiment 2**, where we have combined the method of parameter estimation via least-squares minimization with generic Neural Network training methods. A second Hybrid Model has been experienced in **Experiment 4**, where a pre-trained Convolutional Auto-Encoder has been used as a preprocessing step to significantly improve the performance for noise-corrupted input samples. A third model has been shown in **Experiment 6**, where a pre-trained Convolutional Neural Network has been used to initialize the parameters of the occupant related acceleration profile. Then a Gauss-Newton method has been used to estimate the parameters of the entire Quarter-Car-Model.

Since the previous experiments, using Gauss-Newton methods for parameter estimation, have only considered full-observable systems, we are now going to investigate the estimation accuracy for uncertain systems, where we only have partially observable states.

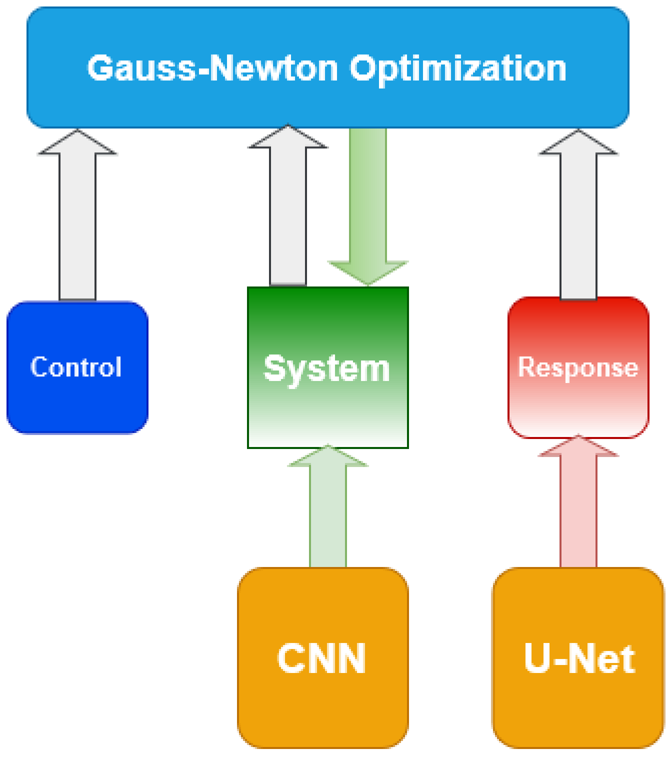


FIGURE 5.20: Identification of the Quarter-Car-Model parameters using a Hybrid Gauss-Newton optimization method. The initial values of the system are partially initialized by the prediction of a pre-trained Convolutional Neural Network. The missing observations of the system response are predicted with a pre-trained U-Net. Then the Gauss-Newton algorithm can be applied, since all required components are predicted with several Neural Networks.

We use the scheme presented in Figure 5.20 to show the idea of the now following experiments. Again, we use a Gauss-Newton method for optimization of the system coefficients of a given test dataset. As already known, the method requires input control and the system matrix to apply an appropriate state estimation method, by a Semi-Explicit Euler scheme. Then the Gauss-Newton step can be applied, minimizing the difference between estimated states and measurements.

Up to now, we have already improved the method by including a pre-trained Neural Network to initialize a subset of the system parameters within an appropriate range. The problem, we now investigate is, what happens to our method, if not all system responses are accessible. This is then crucial for the non-linear least-squares problem, since the measurements are required as reference points for estimation of the residuals for each Gauss-Newton optimization step. To overcome this problem, we will observe, if a U-Net Neural Network structure is able to artificially complete the system responses, using the input of the system and the partially observable system responses.

### 5.4.1 Experiment 8 : Parameter Estimation for Incomplete Systems

#### Problem

The problem of parameter estimation for incomplete systems can easily be clarified, if we consider the following accessible test dataset in consistency with the previously shown experiments. Let us assume that the test dataset is given by

$$\mathcal{T}^{(8)} = \{((a_{m;3}, u_m), (p_{m;3}, p_{m;5}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.86}$$

which can be interpreted, that only the acceleration profile of the occupant is observable. We have access to the input control and do not need to estimate the practically non-identifiable parameters, as they are assumed to be known for the test samples.

The **problem**, we want to investigate is: **Given the partial observations of the system, is it anyhow possible to identify the practically identifiable parameters of the Quarter-Car-Model, if we use the initialization network for parameters $\mathbf{p_1}$ and $\mathbf{p_2}$?**

#### Method

Since there are no observable measurements for the chassis and the wheel-suspensions, it is only possible to solve the following non-linear least-squares problem

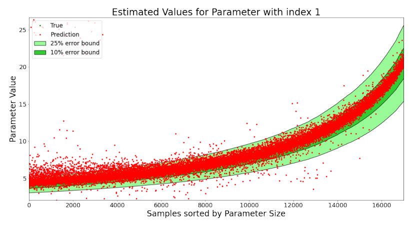
$$\min_p \mathscr{L}(p) = \frac{1}{2}\sum_{k=1}^{N}\left(r_3^k(p)\right)^2, \tag{5.87}$$

where $r_3^k(p) = y_3^k(p) - a_3^k$ for all elements of the test dataset. Obviously, this significantly causes a dimension reduction of the Jacobian, since there is a lack of the partial derivatives $\frac{\partial r_i^k(p)}{\partial p_j}$ for $i \in \{1,2\}$ and $j \in \{1,2,4,6,7,8,9,10\}$ for all steps $k \in \{1,2,...,N\}$. The Jacobian of the reduced parameter vector $\overline{p}$ is therefore given by $J(\overline{p}) \in \mathbb{R}^{N\times 8}$. Therefore, we have no information how a deviation of a current parameter value changes the acceleration profiles of the chassis or the wheel-suspensions. We solely have information, how a variation of the parameters changes the observable acceleration profile of the occupant.

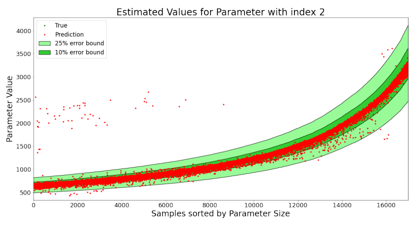
Again, we choose a series-length of $N = 100$ points for the now presented results. Since we have again a large rate of uncertainty in our problem setting, the developed algorithm causes problems for an initial step-size of $\alpha = 1.0$. It is therefore only possible to achieve results, if we choose a smaller step-size $\alpha = 0.1$ and let the algorithm run for a significantly larger number of iterations, compared to the experiments with a larger step-size.

## Results

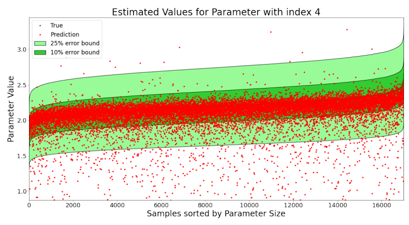
We show the parameter estimation results of a Gauss-Newton method for an incomplete system after 100 optimization steps with step-size $\alpha = 0.1$. Therefore, a visualization of estimated and true parameter values can be considered, as well as the error distribution table, we have broadly used for the previous experiments.



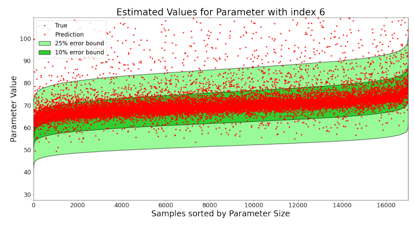
(a) Estimation of parameter 1.



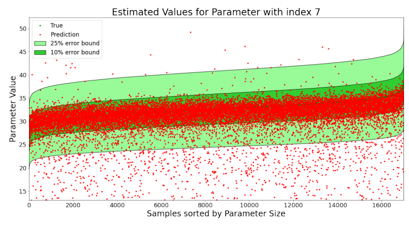
(b) Estimation of parameter 2.



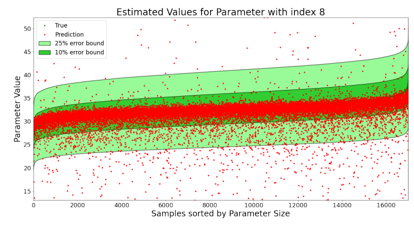
(c) Estimation of parameter 4.



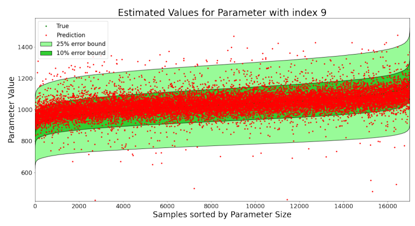
(d) Estimation of parameter 6.



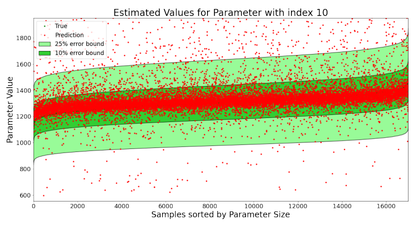
(e) Estimation of parameter 7.



(f) Estimation of parameter 8.



(g) Estimation of parameter 9.



(h) Estimation of parameter 10.

FIGURE 5.21: Gauss-Newton parameter estimation for partial observability, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. A precise description of the plots is described by the caption of Figure 5.4. We recognize the estimation of the Quarter-Car-Model parameters, after 100 optimization steps with initial step-size $\alpha = 0.1$ in Figure 5.21*a* - Figure 5.21*h*. The visualized estimates correspond to the error distribution given in Table 5.14.

If we shortly want to summarize the visualizations which can be observed in Figure 5.21 for uncertain system identification, it is obvious, that we have a significant tendency to underestimate parameters $p_4$, $p_7$ and $p_8$, where parameters, denoted by $p_6$ and $p_{10}$ are mainly overestimated if the approximation of the true value does not lie within the green area. Moreover, we have a central deviation for parameter $p_9$ and relatively good estimates for parameters $p_1$ and $p_2$. Nevertheless, we recognize a large point cloud for small values of $p_1$ and some outliers for the estimates of $p_2$. In conclusion, although we only have partial observability of the system, it is possible to estimate the parameters of a larger subset of the test dataset within the 10% error boundary. Nevertheless, if we require robust estimates of all system coefficients, the herein used model is critical, since there are large deviations beyond 25%.

| Step | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 1 | 6.02052 | 0.09647 | 0.27948 | 0.12253 | 0.49329 | 0.73282 | 0.93935 | 0.06065 |
| 100 | 2 | 2.87366 | 0.02875 | 0.10758 | 0.35618 | 0.88641 | 0.96771 | 0.99553 | 0.00447 |
| 100 | 4 | 3.58796 | 0.12843 | 0.33188 | 0.15688 | 0.62276 | 0.81506 | 0.90994 | 0.09006 |
| 100 | 6 | 3.71102 | 0.12381 | 0.33492 | 0.20576 | 0.69941 | 0.84488 | 0.91406 | 0.08594 |
| 100 | 7 | 3.97295 | 0.12139 | 0.27335 | 0.12994 | 0.55453 | 0.79582 | 0.89653 | 0.10347 |
| 100 | 8 | 0.92186 | 0.05128 | 0.08479 | 0.21165 | 0.74488 | 0.88541 | 0.96588 | 0.03412 |
| 100 | 9 | 1.74747 | 0.04371 | 0.05131 | 0.16635 | 0.69794 | 0.92535 | 0.99188 | 0.00812 |
| 100 | 10 | 3.89406 | 0.12055 | 0.34068 | 0.22518 | 0.72359 | 0.84459 | 0.91288 | 0.08712 |

TABLE 5.14: Experiment 8: Error distribution of Gauss-Newton parameter estimation for partial observability, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. The table shows the error distribution and more statistical values of the observed estimation results. The whole test dataset $\mathcal{T}^{(8)}$ is considered for the computation. The description of the columns is shown in the caption of Table 5.1. The table corresponds to the visualization of Figure 5.21.

We again need to consider the error distribution of this example, given in Table 5.14, as precise values cannot be investigated from Figure 5.21. The main points, we can investigate are therefore the following. We have investigated, that it is nevertheless possible to estimate most of the parameters within an appropriate range. As we can see on the table, more than 73.28% of the parameter estimates lie within the 10% error bound of the system. In contrast, deviations of more than 25% are recognizable for the estimates of $p_1$ with 6%, $p_4$ with 9%, $p_6$ with 8.6%, $p_7$ with 10.3% and $p_{10}$ with 8.7%. Additionally, the maximal deviations are around 92% for parameter $p_8$ and 174% for parameter $p_9$, between 287% and 397% for most of the remaining parameters, except for $p_1$ with even maximal 602% relative deviation from the target value.

## Summary

Estimating the parameters of an uncertain, incomplete system, using the Gauss-Newton method, is difficult in terms of robustness for larger step-sizes, as well as robustness of the estimated values. Although, most of the estimates lie within the 10% error bound, there are significantly large deviations of the parameter values. If we for instance consider the maximal relative deviation of 600% for parameter $p_1$, this is indeed a critical value for occupant safety within the car interior. Since the parameter only considers a variation of the occupant's mass, this could for instance lead to classification of a child as an adult, enabling the airbag-function of the system, and killing the child in case of a crash. What we therefore need are robust models, that do not lead to large false estimates of the parameters.

### 5.4.2 Experiment 9 : Data Completion via Neural Networks

#### Problem

It is critical to develop robust optimization methods, when not all states are observable. Therefore, we want to investigate, if again Neural Networks can be used to overcome such problems. Appropriate network structures, as for instance U-Net [89], have been created to solve the problem of data segmentation for tomography images. We therefore assume, that such structures can also be used to solve segmentation tasks to estimate unobservable sequential acceleration profiles.

Let us consider the training dataset

$$\mathcal{S}^{(9)} = \{((a_{m;3}, \overline{u}_m), (a_{m;2}, a_{m;1}))\}_{m=1}^{N_S} \tag{5.88}$$

and for latter evaluation of the generalization property, let us further define the corresponding test dataset as

$$\mathcal{T}^{(9)} = \{((a_{m;3}, \overline{u}_m), (a_{m;2}, a_{m;1}))\}_{m=N_S+1}^{N_S+N_T}, \tag{5.89}$$

where the task almost results from investigating the structure of the training dataset. Obviously, we assume that the acceleration profiles of the occupant, $a_{m;3} \in \mathbb{R}^N$ are observable, as well as the input control $\overline{u}_m \in \mathbb{R}^N$, the randomly generated road-profile of the sample, scaled by the spring-coefficient $p_u$ of the wheel-suspensions, such that $\overline{u}_m = p_u u_m$. The target values are therefore given by the acceleration profiles of the chassis, $a_{m;2} \in \mathbb{R}^N$ and the profile of the wheel-suspensions, $a_{m;1} \in \mathbb{R}^N$.

The **problem**, we want to investigate is. **Given the accessible acceleration of the occupant and the scaled road-profile, can a U-Net structure be used to appropriately predict the unobservable states of the coupled dynamical system for the training samples and more important also generalizes for the test samples?**

#### Method

For common elements of the training or the test data, denoted by $a_3 \in \mathbb{R}^N$ and $\overline{u} \in \mathbb{R}^N$, let us define a general U-Net structure as a function

$$f_\theta^{(9)} : \mathbb{R}^{N\times 2} \to \mathbb{R}^{N\times 2} \tag{5.90}$$

$$(a_3, \overline{u}) \mapsto (\tilde{a}_2(\theta), \tilde{a}_1(\theta)), \tag{5.91}$$

such that in terms of the network's output, the predictions of the unobservable profiles are given by

$$\tilde{a}_2(\theta) = \left(f_\theta^{(9)}(a_3, \bar{u})\right)_1, \quad \tilde{a}_1(\theta) = \left(f_\theta^{(9)}(a_3, \bar{u})\right)_2. \tag{5.92}$$

The training of the Neural Network to map the observable information to the unobservable states, can then be defined as the optimization problem

$$\min_\theta \frac{1}{N_S} \sum_{m=1}^{N_S} \sum_{k=1}^{N} \left( \left(a_{m;2}^k - \tilde{a}_{m;1}^k(\theta)\right)^2 + \left(a_{m;1}^k - \tilde{a}_{m;2}^k(\theta)\right)^2 \right), \tag{5.93}$$

which is obviously a similar training definition as the denoising problem in **Experiment 4**, yet we do not want to map corrupted data to cleaned versions, but different measurement profiles on each other.

The training specifications for this problem are given by

- Optimization method: Adaptive Momentum (ADAM) Optimization
- Learning-rate: $\eta = 0.0003$
- Batch-size: $B = 100$
- Series-length of the input samples: $N = 512$
- Training-epochs: $E = 200$

The layer structure of the U-Net, used to achieve our results, is given by Figure *B*.3. Implementation details are shown in Figure *A*.9, Figure *A*.10, Figure *A*.11 and Figure *A*.12. The following results will mainly focus on the observations made for the test samples, as they are stated in the parameter estimation problems, using the Gauss-Newton method in **Experiment 10**.

**Results**

We want to quickly evaluate the training and the generalization performance for the U-Net to identify the unobservable states of the system. Therefore, we evaluate the loss-functions of training and test error, the Mean-Squared-Error (MSE) of predicted state and true unobservable state and finally show some examples for the test dataset. The results shown here, are therefore structurally comparable to the denoising investigations in **Experiment 4**.

The shapes of the loss-functions in Figure 5.22 show, that the network parameters are appropriately trained. The longer the training, the lower the value of the loss-functions. This holds for the training loss as well as for the test loss. Again, the training loss is slightly below the test loss, but we cannot observe a significant overfitting of the training data, since the loss-functions of training and test are obviously close to each other.

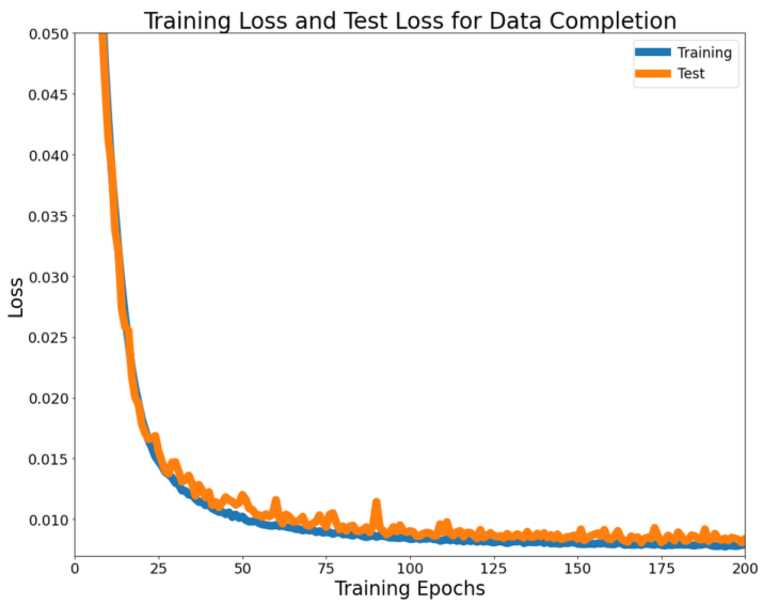
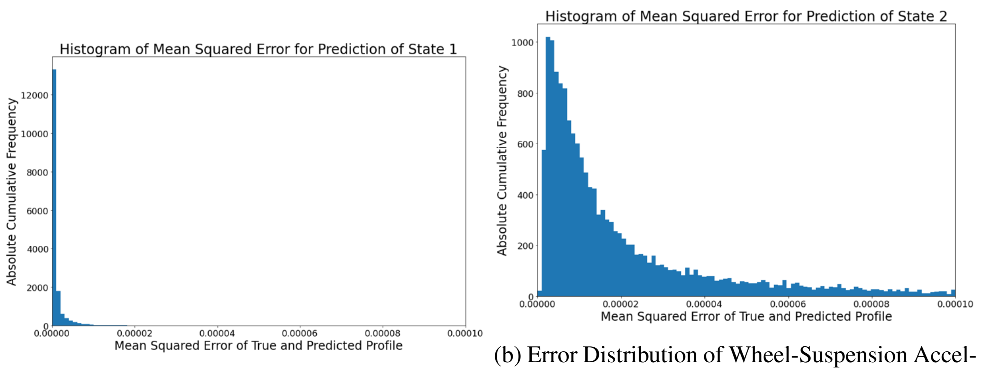


FIGURE 5.22: Training Loss and Test Loss for the Data Completion U-Net. The shape of the training loss is visualized in blue, the test loss in orange. The horizontal axis describes the number of training epochs, the vertical axis the values of the loss function.



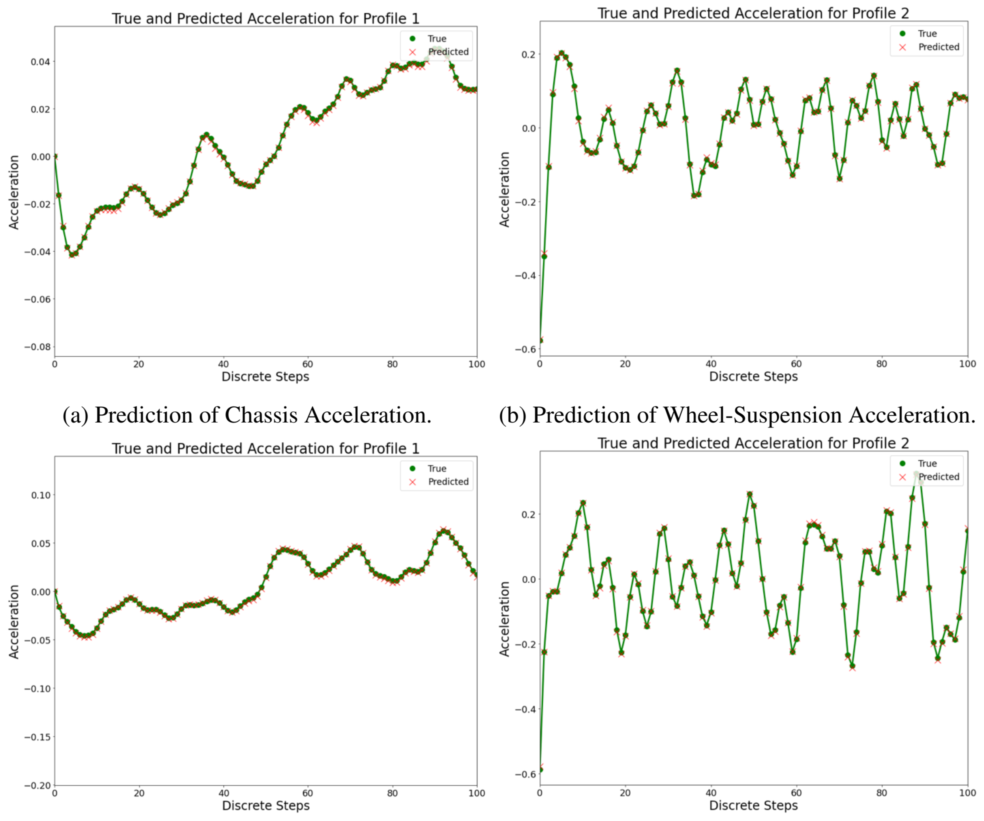
(a) Error Distribution of Chassis Acceleration.

(b) Error Distribution of Wheel-Suspension Acceleration.

FIGURE 5.23: Histograms of the Mean-Squared-Error for the predicted unobservable data of the test dataset. The horizontal axis shows the Mean-Squared-Error for each predicted sample, according to the true underlying unobservable test sample. The vertical axis shows the absolute cumulative frequency of the error values that have been computed for the test samples.

The evaluation of the MSE in Figure 5.23 yields that the error range for the prediction of the chassis profile on the l.h.s. and that of the wheel-suspensions on the r.h.s. differ. Most errors for the chassis are below $10^{-5}$, while we have an upper bound of approximately $10^{-4}$ for the wheel-suspension. Let us finally consider some examples of the test dataset to evaluate the prediction accuracy of the pre-trained U-Net, which we denote in consistency with the already discussed networks by $f_{\theta_\delta}^{(9)}$ after $K = E\frac{N_S}{B} = 200 \cdot 340 = 68000$ optimization steps.

There can be two examples observed in Figure 5.24. The predictions of the unobservable chassis acceleration is given on the l.h.s. while the wheel-suspension's profile is shown on the r.h.s. It is not possible to recognize large deviations of the predicted estimates for the test dataset. Most estimated values are close to the true, unknown values. This means, that



(a) Prediction of Chassis Acceleration. (b) Prediction of Wheel-Suspension Acceleration.

(c) Prediction of Chassis Acceleration. (d) Prediction of Chassis Acceleration.

FIGURE 5.24: Examples of Data Completion for Test Samples.

given the acceleration of the chassis and the road-profile, it is possible for unknown test data to complete the uncertain system with the help of a pre-trained U-Net for segmentation of sequential data.

### Summary

The training of a U-Net has shown promising results to handle the problem of an incomplete system. It is remarkable, that even for samples, which have not been used to update the network parameters in the training process, the network is capable to give very close estimates of the true states. Therefore, as a logical final step, we will make use of this pre-trained network in **Experiment 10**, to complete the uncertain system, such that the Gauss-Newton method of **Experiment 7** can be applied to the predicted system.

## 5.4.3 Experiment 10 : Parameter Estimation for Hybrid Models

### Problem

We have seen that it is indeed possible to train a U-Net in **Experiment 9**, which delivers appropriate results on the test dataset for prediction of the unobservable acceleration profiles. Let us therefore now consider again the test dataset of **Experiment 8**, by

$$\mathcal{T}^{(8)} = \{((a_{m;3}, u_m), (p_{m;3}, p_{m;5}))\}_{m=N_S+1}^{N_S+N_T},$$

which has previously already been defined in detail. We have further investigated, that trying to solve parameter estimation problems for incomplete systems is not robust against large deviations of the approximated coefficient values.

The **problem** we want to investigate is: **Given the acceleration of the occupant, the road-profile and the value of the non-identifiable parameters, can we use the pre-trained U-Net to complete the system and with the help of the initialization network for $\mathbf{p_1}$ and $\mathbf{p_2}$ develop a more robust Hybrid Model for system identification of the Quarter-Car-Model coefficients?**

**Method**

Instead of directly defining the optimization problem to apply the Gauss-Newton method, we need to execute some steps beforehand. Let us therefore consider the following:

1. The pre-trained U-Net for data completion in **Experiment 9** can be denoted with the parameter realization $\theta_\delta$ by

$$\begin{aligned} f^{(9)}_{\theta_\delta} &: \mathbb{R}^{N\times 2} \to \mathbb{R}^{N\times 2} \\ &(a_3, \bar{u}) \mapsto (\tilde{a}_2(\theta_\delta), \tilde{a}_1(\theta_\delta)), \end{aligned}$$

, with $\bar{u} = p_u u$ the source term of the Quarter-Car-Model. Therefore, for all $a_{m;3}$ and $u_m$ of the test dataset, we assume that the corresponding spring-coefficient $p_{m;u}$ is known, such that we can compute $\bar{u}_m = p_{m;u} u_m$ for all samples of the set. Then, the incomplete system can be approximately completed, using $f^{(9)}_{\theta_\delta}$, such that we get

$$\tilde{a}_{m;2}(\theta_\delta) = \left(f^{(9)}_{\theta_\delta}(a_{m;3}, \bar{u}_m)\right)_1, \quad \tilde{a}_{m;1}(\theta_\delta) = \left(f^{(9)}_{\theta_\delta}(a_{m;3}, \bar{u}_m)\right)_2 \tag{5.94}$$

for all $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$, with $\tilde{a}_{m;1}(\theta_\delta)$, $\tilde{a}_{m;2}(\theta_\delta) \in \mathbb{R}^N$. The system is therefore artificially completed.

2. The pre-trained Convolutional Neural Network in **Experiment 2** can be denoted with the parameter realization $\theta_\beta$ by

$$\begin{aligned} f^{(2)}_{\theta_\beta} &: \mathbb{R}^{N\times 2} \to \mathbb{R}^2 \\ &(a_3, a_2) \mapsto \begin{pmatrix} \tilde{p}_1(\theta_\beta) \\ \tilde{p}_2(\theta_\beta) \end{pmatrix}. \end{aligned}$$

Since execution of the pre-trained Neural Network for parameter estimation requires acceleration data of the occupant as well as acceleration data of the chassis, we can now use the completed system, to get initial parameter values $p^0_{m;1}$ and $p^0_{m;2}$ for all $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$. It therefore holds that

$$p^0_{m;i} = \left(f^{(2)}_{\theta_\beta}\begin{pmatrix} a^T_{m;3} \\ \tilde{a}^T_{m;2}(\theta_\delta) \end{pmatrix}\right)_i \tag{5.95}$$

for $i \in \{1,2\}$. The initialization network therefore uses the predicted acceleration profile of the chassis to estimate the initial values of the occupant related parameters. The initialization of the parameters is already a function composition of two pre-trained Neural Networks.

3. The pre-trained U-Net has already been used to complete the system, the pre-trained Convolutional Neural Network has been used to initialize the parameter vector. As a final step, one can define the predicted residuals for the Gauss-Newton method to start, for any iteration step $l \in \mathbb{N}$ at any point $k \in \{1,2,...,N\}$ for all samples with index $m \in \{N_S+1, N_S+2, ..., N_S+N_T\}$ by

$$\tilde{r}_m^k(p^l) = \begin{pmatrix} y_{m;3}^k(p^l) - a_{m;3}^k \\ y_{m;2}^k(p^l) - \tilde{a}_{m;2}^k(\theta_\delta) \\ y_{m;1}^k(p^l) - \tilde{a}_{m;1}^k(\theta_\delta) \end{pmatrix}. \tag{5.96}$$

We can then define the non-linear least-squares problem for parameter estimation of the artificially completed system by

$$\min_p \mathscr{L}(p) = \frac{1}{2} \sum_{k=1}^{N} \sum_{i=1}^{3} \left( \tilde{r}^k(p) \right)^2, \tag{5.97}$$

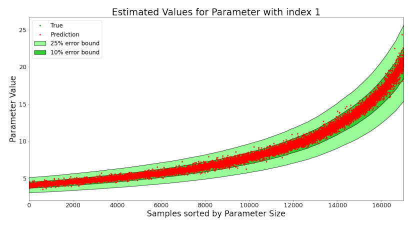
which needs to be solved for all samples of the test dataset individually.

Choosing a series-length of $N = 100$, and trying to apply the Armijo-Rule with $\alpha = 1.0$ for the now completed system, we can compare the results of the applied Gauss-Newton method with those of the incomplete system in **Experiment 8**. Implementation details for the entire model, are given by Figure $A.13$ - Figure $A.19$.

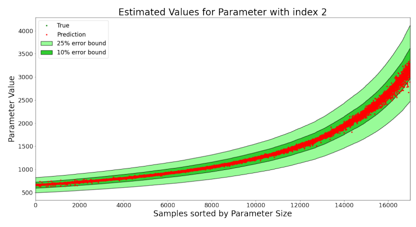
## Results

We now present the last results as far as this work is concerned. It is now possible to compare, if there are any advantages to use the artificially completed system instead of the incomplete uncertain system in **Experiment 8**.

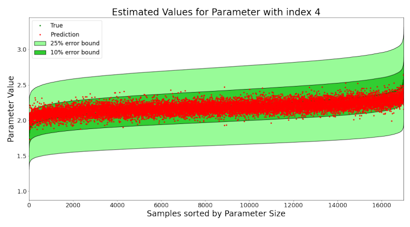
The visualization of the estimates in Figure 5.25 for the Hybrid Gauss-Newton method show, that after 10 iterations, the results are significantly more stable, compared to the incomplete system. There is obviously no tendency of underestimation or overestimation of a given parameter. Most estimates are observable within the 10% error bound, while we can furthermore recognize a few outliers in the 25% area at least for $p_6$ and $p_7$. In addition, the most difficult parameter to estimate still remains $p_7$, as it is the only parameter, where we recognize outliers beyond the 25% error boundary. Another significant observation is that for the remaining parameters, which correspond to the acceleration profile of the wheel-suspensions, the Gauss-Newton method estimates the approximative values within the same area. It is not



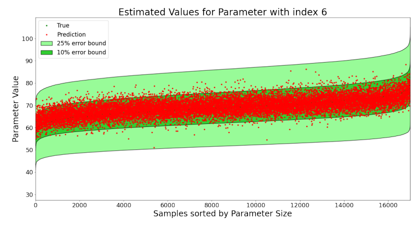
(a) Estimation of parameter 1.



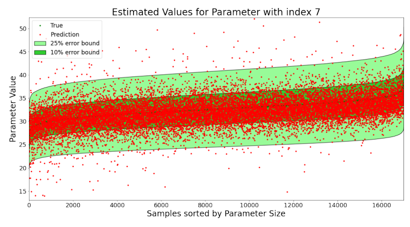
(b) Estimation of parameter 2.



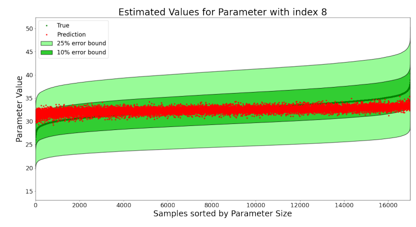
(c) Estimation of parameter 4.



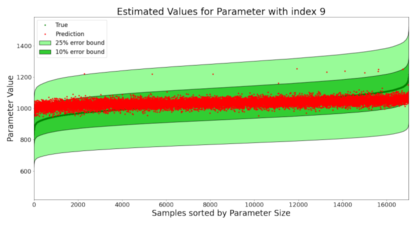
(d) Estimation of parameter 6.



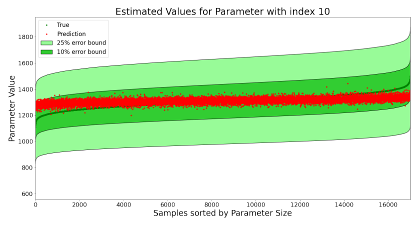
(e) Estimation of parameter 7.



(f) Estimation of parameter 8.



(g) Estimation of parameter 9.



(h) Estimation of parameter 10.

FIGURE 5.25: Gauss-Newton parameter estimation for completed system via U-Net, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. A precise description of the plots is described by the caption of Figure 5.4. We recognize the estimation of the Quarter-Car-Model parameters, after 10 optimization steps with initial step-size $\alpha = 1.0$ in Figure 5.25*a* - Figure 5.25*h*. The visualized estimates correspond to the error distribution given in Table 5.15.

possible to investigate a tendency for smaller or larger values of $p_8$, $p_9$ and $p_{10}$. Nevertheless, the estimated value is on average close to the mean of the parameters. Therefore, no large deviations can be observed for those parameters. This effect, apparently occurs from the data completion of the acceleration profile of the wheel-suspensions, since **Experiment 7** has already shown, that it is indeed possible to uniquely determine the parameters from a full-observable system with exact measurements.

Finally, the investigations done with the help of Figure 5.25 can be verified with the well-known error distribution of Table 5.15. In comparison with Table 5.14, it is worth mentioning

| Step | Parameter | Max Error | Mean Error | Std Error | 0-1% | 0-5% | 0-10% | 0-25% | >25% |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 1 | 0.20629 | 0.02544 | 0.02155 | 0.27018 | 0.87965 | 0.99065 | 1.00000 | 0.00000 |
| 10 | 2 | 0.17763 | 0.00928 | 0.01119 | 0.69765 | 0.98565 | 0.99882 | 1.00000 | 0.00000 |
| 10 | 4 | 0.20689 | 0.02945 | 0.02228 | 0.20829 | 0.82759 | 0.99241 | 1.00000 | 0.00000 |
| 10 | 6 | 0.24163 | 0.03541 | 0.02809 | 0.17912 | 0.74806 | 0.96829 | 1.00000 | 0.00000 |
| 10 | 7 | 1.09824 | 0.06551 | 0.05899 | 0.10288 | 0.48176 | 0.79929 | 0.98618 | 0.01382 |
| 10 | 8 | 0.20682 | 0.03309 | 0.02536 | 0.18853 | 0.77418 | 0.98141 | 1.00000 | 0.00000 |
| 10 | 9 | 0.25447 | 0.03409 | 0.02683 | 0.19094 | 0.76059 | 0.97588 | 0.99994 | 0.00006 |
| 10 | 10 | 0.16866 | 0.02641 | 0.02069 | 0.24547 | 0.86765 | 0.99529 | 1.00000 | 0.00000 |

TABLE 5.15: Experiment 10: Error distribution of Gauss-Newton parameter estimation for completed system via U-Net, restricted to the identifiable parameters and parameter initialization of $p_1$ and $p_2$ via Convolutional Neural Network. The table shows the error distribution and more statistical values of the observed estimation results. The whole test dataset $\mathcal{T}^{(8)}$ is considered for the computation. The description of the columns is shown in the caption of Table 5.1. The table corresponds to the visualization of Figure 5.25.

that more than 98.6% of all parameter estimates lie within the 25% boundary for the artificially completed system, while we had a value of 89.6% for the incomplete system. Analogously, more than 79.9% of the completed system lie within the 10% boundary in contrast to 73.2% for **Experiment 8**. Moreover the maximal relative deviation is 109% for parameter $p_7$ and below 26% for the remaining ones, while we had at least 92% for the incomplete system and deviations, reaching from about 200% to a top value of 600%. This large deviations do not occur for the artificially completed system.

## Summary

We have used several Neural Networks to eliminate uncertainties that have been described in **Experiment 8**. Where we had large deviations for the incomplete system, the data completion using U-Net now leads to a more robust estimation with relatively small maximal deviations from the true target values. Furthermore, initialization of parameters $p_1$ and $p_2$ via Convolutional Neural Networks, result in stability of the algorithm to converge to the true optimal area of the parameter values.

Before ending this section, we want to briefly give an overview about the computational time of the main shown Neural Networks and algorithms.

The computational time to train the considered Neural Networks is given in Table 5.16. Since individual batch-sizes and training epochs have been used, it is not possible to compare the time-consumption of all individual networks. Nevertheless, we can recognize that the unlabelled training approach, since it uses the same training specifications as for the labelled approach, requires double the time compared to the labelled training approach for the Convolutional Neural Networks. Obviously, this effect occurs from computation of the predicted acceleration for the unlabelled optimization of the acceleration parameters. All in all, the networks need less than 1936.36 seconds for training, which equals around 33 minutes. In total, all trainings require 4704 seconds of pure training time, which means that less than 80 minutes of pre-training for the Neural Networks is required to achieve the herein shown

| Network | Time per Opt Step | Time per Epoch | Total Training Time |
|---|---|---|---|
| **CNN (labelled)** | 0.03097 | 1.0074 | 1023.2019 |
| **CNN (unlabelled)** | 0.05704 | 1.98851 | 1966.36042 |
| **CAE** | 0.01556 | 5.28485 | 528.39478 |
| **U-Net** | 0.01742 | 5.95131 | 1187.01905 |

TABLE 5.16: Computational time of the training Neural Networks in seconds, which have been used in the section. We estimate the time of the Convolutional Neural Network with labelled optimization (CNN labelled) and unlabelled optimization (CNN unlabelled) for 1000 training epochs with batch-size 1000 for the training dataset. Further, we have the Convolutional Auto-Encoder for Denoising (CAE) for 100 training epochs and batch-size 100 and finally the U-Net for state estimation with 200 training epochs and batch-size 100. **Time per Opt Step**: Total time to apply the ADAM-Optimization algorithm for one training step. **Time per Epoch**: Total time to apply the ADAM-Optimization algorithm for the entire training-dataset (one training epoch). **Total Training Time**: Total time used to train the Neural Network, with respect to the described training epochs and batch-sizes.

results. One can find Deep Neural Networks, which have been trained for hours or days to achieve a required accuracy for a given problem.

| | State | Gradient | Fisher Inversion | Model | Update | Total Step |
|---|---|---|---|---|---|---|
| **Abs. Time (sec.)** | 0.20 | 27.82 | 0.11 | 29.25 | 0.01 | 29.61 |
| **Rel. Time (%)** | 0.69 | 93.97 | 0.37 | 98.78 | 0.03 | 100.00 |

TABLE 5.17: Computational time of the Gauss-Newton algorithm to estimate the parameters of the entire test dataset in seconds. The rows show the mean absolute computational time for one step of the Gauss-Newton algorithm and the mean relative time of the sub-routines. **State**: Total time to estimate the model states with the Semi-Explicit Euler scheme for 100 time-steps. **Gradient**: Total time to compute the partial derivatives, which are necessary for computation of the Jacobian for 100 time steps. **Fisher inversion**: Time to compute the inverse of the Fisher matrix with the pre-implemented Tensorflow inversion function. **Model**: Time from first Explicit-Euler step to the inversion of the Fisher matrix, time to compute the model states and return the optimization direction for 100 time steps. **Update**: Time to apply the Gauss-Newton iterative scheme to update the system parameters and to reshape the updated values to the updated system matrix. **Total Step**: Total time, which is used for one step of the Gauss-Newton method.

At last, we want to shortly discuss, what components of the Gauss-Newton method are the most time-consuming ones. From Table 5.17, it is obvious that one Gauss-Newton step requires approximately 30 seconds on the described **NVIDIA Tesla V100 SMX2 32GB** graphics card. This means for the Hybrid Gauss-Newton method of **Experiment 10**, that only 3 minutes are required for the optimization, where we have around 30 minutes for the results of the uncertain systems, for instance in **Experiment 8**. Furthermore, we can recognize that almost 94% of the time for one Gauss-Newton step is required for the computation of the derivatives. Nevertheless, the Tensorflow computation of the Gradient for the model states is highly efficient, since it uses automatic differentiation and does not need to exactly compute the derivatives individually for each component.

Closing the last main section of this work, we can summarize as follows: Hybrid Models, including well-studied mathematical methods and data driven models, can therefore efficiently be used to eliminate uncertainties of a given dynamical system. It is worth mentioning that this works at least for several different Neural Networks, which solve smaller individual tasks. We cannot assume that there is one specific network structure to easily solve the parameter estimation problem. Further experiments have shown, that it is indeed not possible to directly estimate the network parameters with the help of a Neural Network for a test dataset with comparable estimation quality as presented by the shown Gauss-Newton parameter estimation.

# Chapter 6

# Conclusion and Discussion

This thesis has contained several interdisciplinary fields, reaching from mathematical modelling of mechanical systems, from Statistical Learning Theory connected to Neural Networks and solving ordinary differential equations for data generation to hybrid parameter estimation of dynamical systems. Closing this work, we want to briefly summarize the main investigations, we have observed.

We have motivated the thesis by discussing modern individual traffic, where vehicles are nowadays equipped with a broad set of smart sensing devices to ensure safety for interior occupants as well as exterior traffic participants. As one cannot easily make theoretical investigations, since mostly, there is no full vehicle model accessible, we need to use approximative vehicle models, which describe a simplification of the world. Therefore, we have focused on the Quarter-Car-Model, which is in general a coupled Mass-Spring-Damper model, including wheel-suspension, chassis and occupant (and seat) specific components, driven by a non-linear source term, the synthetic road-profile. A general coupled dynamical model, described as a system of differential equations can be derived from the Euler-Lagrange formalism, which requires definition of the energy terms of an observable system. The shown principle can easily be extended to more complex Half-Car-Models or Full-Car-Models, which contain a larger amount of degrees of freedom. Neural Networks can be used as universal function approximations, since there exist layer structures, which are capable to theoretically map any input dimension to another target dimension. Given an accessible training and a test dataset to solve a stated problem, one always tries to find a set of parameter values, for which the Neural Network delivers comparable results for all samples of the training dataset as well as for all samples of the test dataset, such that it solves the problem within an acceptable error range. We have focused on three different Neural Network structures, namely Convolutional Neural Networks connected to Fully-Connected layers, Convolutional Auto-Encoders and the so-called U-Net as special type of a Convolutional Neural Network. The Quarter-Car-Model has shown to be mathematically described as a system of second order ordinary differential equations, which can equally be transferred into a system of first order ordinary differential equations. It is therefore in principle possible to compute the exact acceleration of a theoretical dynamical system with the help of general solution theory for non-homogeneous ordinary differential equations. Since the road-profile is a complex non-linear construct, due to the summation of random sine-waves, it is non-trivial to compute the exact solution, such that we need efficient, structure preserving numerical integration schemes as the Symplectic Euler scheme. Using an appropriate integration scheme, one can assume that the discrete approximated solution is a sufficiently good estimation of the true world. Random generation of the parameters of the system matrix and random road-profiles, offers the opportunity to get training and test datasets of arbitrary size, following pre-defined parameter distributions.

The theoretical concepts, which have been described up to this point, are necessary to clarify the general setting of parameter estimation. The initial question has stated, whether it is possible to estimate the parameters of a dynamical system, from broadly accessible sensing devices like accelerometers. Assuming that the approximative vehicle model and the numerical solution of the integration schemes are a sufficiently good approximation of the real world, one can assume to have full-knowledge about the entire system. Therefore, one can now try to answer the question, what information can be computed out of the artificially created acceleration profiles. Furthermore, it has been shown that the property of estimating all parameters of the dynamical system is restricted to the grade of observable states of the system. It is therefore more probable, that there are several distinguishable system matrices, which satisfy the same input and output relation, if only one component of the acceleration profile of the entire system can be observed.

Our own investigations can roughly be separated into three sections: A two-dimensional parameter estimation problem, if the acceleration data of the occupant and the chassis is observable for different training methods of Neural Networks, a ten-dimensional (and later eight-dimensional) parameter estimation problem of the Quarter-Car-Model, if all acceleration profiles and the source term are accessible and an eight-dimensional parameter estimation problem under uncertainties, if we assume that only one acceleration profile of the Quarter-Car-Model is observable.

The main investigations can be summarized as follows:

1. **Convolutional Neural Networks can efficiently be used to combine acceleration profiles of different states, due to the convolutional layer structure, and map the data to the two-dimensional parameter space for the occupant-related coefficients. Convolutional layer structures have shown to be appropriate models to process sequential data of a coupled dynamical system, since the size of the kernels makes it possible to weight the shape of the acceleration profiles for a large time-frame.**

2. **Unlabelled training approaches are more robust for parameter estimation compared to labelled, state-of-the-art, approaches. Although the training of an unlabelled loss-function requires a significantly larger amount of training-time, the unlabelled trained Neural Network, is more robust for unknown test samples and for relative Gaussian disturbances of the input data.**

3. **Convolutional Auto-Encoders can successfully be trained to serve as data pre-processing algorithms for parameter estimation. Pre-processing by a trained Convolutional Auto-Encoder and estimation of the denoised samples then leads to accuracies, comparable to results of the true underlying clean acceleration data.**

4. **A Gauss-Newton method to numerically solve non-linear least-squares problems can be implemented to estimate all parameters of the Quarter-Car-Model. Though, each of the ten individual parameters of the Quarter-Car-Model are structurally identifiable, practical investigations show, that a subset of two parameters highly deviates from the true parameter values and is therefore practically non-identifiable. Restricting the problem to the structurally and practically identifiable parameters, the Gauss-Newton method delivers unique results for the parameter values of the entire test dataset.**

5. **The Gauss-Newton method can be stabilized and accelerated, if one uses a pre-trained Neural Network to estimate the initial parameter values for the first two parameters of the Quarter-Car-Model. The remaining parameters can be initialized using a Gaussian distribution under a specific standard deviation around the mean parameter values of a reference model.**

6. **If we only have partial observability of the states, it is not possible to ensure a stable Gauss-Newton estimation method. One therefore needs to choose a sufficiently low step-size to get at least any results for the unobservable system. After a significantly higher time for estimation of the system parameters, one reaches results with high maximal deviations from the true values. It is therefore not possible to reach robust estimates of the parameters.**

7. **A U-Net structure can be used to map the observable acceleration of the occupant and the road-profile to the remaining unobservable states. This is possible for significantly small deviations for the training data as well as for nearly all unknown samples of the test dataset.**

8. **A Hybrid Gauss-Newton method can be defined, using a pre-trained Convolutional Neural Network for initialization of a subset of the parameters and a pre-trained U-Net for data completion for the non-linear least-squares problem. The created model is robust compared to the uncertain problem and gives estimates for all parameters within a small error bound after only a few optimization steps.**

We have learnt, that Neural Networks are not universally applicable algorithms to solve any task sufficiently good. In contrast, Neural Networks are probably capable to solve any task within an appropriate error bound, but fail to generalize this property for another set of unknown test samples. Furthermore, very Deep Neural Networks to solve complex task, mostly require hours and days for efficient training of millions of network parameters. As far as our investigations are concerned, we have restricted our research to mainly shallow Neural Networks, which need less than one hour of training time to solve a specific sub-task of a more complex problem. It is remarkable that neither a pure state-of-the-art Gauss-Newton method can be used to appropriately estimate the parameters of an uncertain coupled dynamical system, nor a sufficiently Deep Neural Network, since it suffers from overfitting. Only the combination as a Hybrid Model is capable to deliver results within an appropriate error range. We can therefore conclude that as far as this work is concerned, Neural Networks can efficiently be used to solve smaller sub-task, which are essential to support a sophisticated mathematical model to solve a highly complex task.

We can therefore conclude, that the main new achievements of this work are the following:

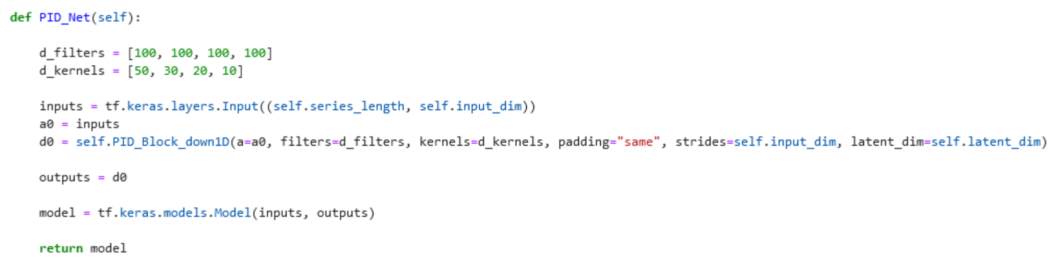
1. **It is possible to use discrete acceleration profiles of a passenger-based Quarter-Car-Model to estimate mass-specific parameters of the system. Therefore, under optimal conditions, this could theoretically offer the opportunity to process data of several g-sensors in a car interior to detect the mass of the occupant for safety critical applications. Until now, there is no approach for such an application.**

2. **It is possible to use Convolutional Neural Networks to efficiently process time-series. When dealing with time-series processing, for instance for time-series forecasting, one in the most common cases uses Recurrent Neural Networks, as for instance LSTM. It is not the usual case to use convolutional architectures for such tasks. Furthermore, the specific U-Net structure has never before been used to process time-series data, as the main applications are restricted to data-segmentation for tomography images. We have shown, that the architecture, due to the skip-connections of the network, is sufficient to also approximate unknown acceleration profiles, when input and output of the system are known.**

3. **It is possible to overcome the problem of non-identifiability for incomplete observations with the help of the presented Hybrid Gauss-Newton method. By making use of a full-observable training dataset, one can fit the network parameters of U-Net to generalize for unknown samples of the test data. The approximation quality of the pre-trained network is sufficient to stabilize the Gauss-Newton method. In contrast, parameter estimation for incomplete observations is unstable and delivers significant errors for a large amount of the predicted parameter values. With the help of a Hybrid Gauss Newton method, outliers are eliminated. There is no comparable approach that combines a Gauss-Newton method for parameter estimation tasks in combination with several pre-trained Convolutional Neural Networks to increase the prediction quality of the method.**

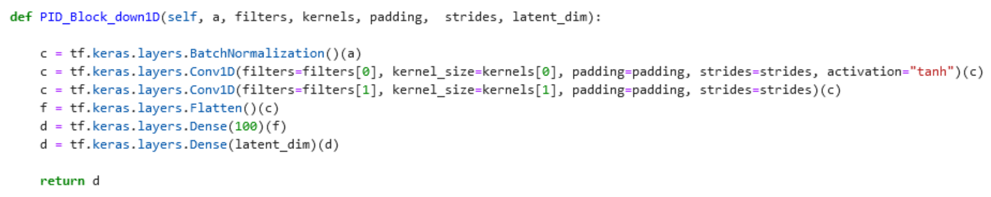
# Appendix A

# Tensorflow and Python Source Code



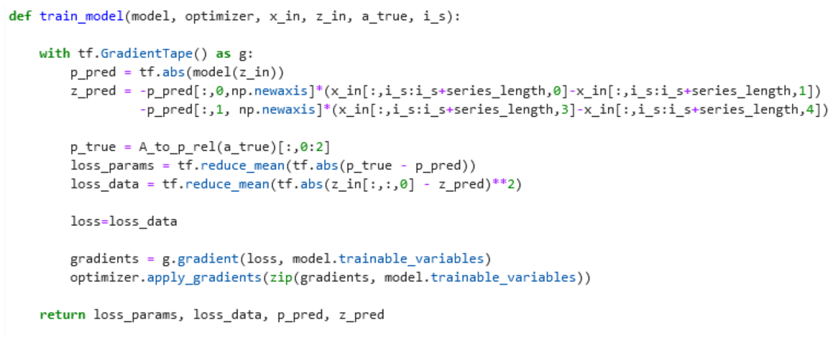
```python
def PID_Net(self):

    d_filters = [100, 100, 100, 100]
    d_kernels = [50, 30, 20, 10]

    inputs = tf.keras.layers.Input((self.series_length, self.input_dim))
    a0 = inputs
    d0 = self.PID_Block_down1D(a=a0, filters=d_filters, kernels=d_kernels, padding="same", strides=self.input_dim, latent_dim=self.latent_dim)

    outputs = d0

    model = tf.keras.models.Model(inputs, outputs)

    return model
```

FIGURE A.1: Algorithm: Definition of CNN Network Function. Layer structure and kernel sizes shown in the code.



```python
def PID_Block_down1D(self, a, filters, kernels, padding,  strides, latent_dim):

    c = tf.keras.layers.BatchNormalization()(a)
    c = tf.keras.layers.Conv1D(filters=filters[0], kernel_size=kernels[0], padding=padding, strides=strides, activation="tanh")(c)
    c = tf.keras.layers.Conv1D(filters=filters[1], kernel_size=kernels[1], padding=padding, strides=strides)(c)
    f = tf.keras.layers.Flatten()(c)
    d = tf.keras.layers.Dense(100)(f)
    d = tf.keras.layers.Dense(latent_dim)(d)

    return d
```

FIGURE A.2: Algorithm: Definition of CNN Network Function. Structure of the layer functions are shown in the code. Are used to define the model as shown in the source coder above.



```python
def train_model(model, optimizer, x_in, z_in, a_true, i_s):

    with tf.GradientTape() as g:
        p_pred = tf.abs(model(z_in))
        z_pred = -p_pred[:,0,np.newaxis]*(x_in[:,i_s:i_s+series_length,0]-x_in[:,i_s:i_s+series_length,1])
                 -p_pred[:,1, np.newaxis]*(x_in[:,i_s:i_s+series_length,3]-x_in[:,i_s:i_s+series_length,4])

        p_true = A_to_p_rel(a_true)[:,0:2]
        loss_params = tf.reduce_mean(tf.abs(p_true - p_pred))
        loss_data = tf.reduce_mean(tf.abs(z_in[:,:,0] - z_pred)**2)

        loss=loss_data

        gradients = g.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss_params, loss_data, p_pred, z_pred
```

FIGURE A.3: Algorithm: Training Function of Parameter Estimation Network. For labelled approach, we use the parameter loss for application of the gradient, for unlabelled approach, we us the data loss.

```
batch_size=1000
series_length=512
optimizer = tf.keras.optimizers.Adam(1e-3)
opt_steps=1100

for i in range(opt_steps):
    for z_train, x_train, a_train, m_train in train_dataset:
        i_s = 0
        train_model(model=model_test, optimizer=optimizer,
                    x_in = x_train, z_in = z_train[:,i_s:i_s+series_length,:],
                    a_true=a_train, i_s=i_s)
```

FIGURE A.4: Algorithm: Optimization Function of Parameter Estimation Network. The training is done for the number of optimization steps for all training samples. Hyperparameter specification as batch-size, optimizer and learning-rate are stated in the header.

```
def ID_Net(self):

    d_filters = [100, 100, 100, 100]
    d_kernels = [50, 30, 20, 10]

    u_filters = [100, 50, 2]
    u_kernels = [50, 50, 50]

    inputs = tf.keras.layers.Input((self.series_length, self.input_dim))
    a0 = inputs
    d0 = self.ID_Block_down(a=a0, filters=d_filters, kernels=d_kernels, padding="same", strides=2, latent_dim = self.latent_dim)
    a1 = self.ID_Block_up(d=d0, filters=u_filters, kernels=u_kernels, padding="same", strides=2, latent_dim=self.latent_dim)
    outputs = a1
    model = tf.keras.models.Model(inputs, outputs)

    return model
```

FIGURE A.5: Algorithm: Definition of CAE Network Function. Layer structure and kernel sizes shown in the code.

```
def ID_Block_down(self, a, filters, kernels, padding,  strides, latent_dim):


    c = tf.keras.layers.Conv1D(filters=filters[0], kernel_size=kernels[0], padding=padding, strides=strides)(a)
    c = tf.keras.layers.Conv1D(filters=filters[1], kernel_size=kernels[1], padding=padding, strides=strides)(c)

    return c

def ID_Block_up(self, d, filters, kernels, padding, strides, latent_dim):

    t = tf.keras.layers.Conv1DTranspose(filters=filters[0], kernel_size=kernels[0], padding=padding, strides=strides)(d)
    t = tf.keras.layers.Conv1DTranspose(filters=filters[1], kernel_size=kernels[1], padding=padding, strides=strides)(t)
    t = tf.keras.layers.Conv1DTranspose(filters=filters[2], kernel_size=kernels[2], padding=padding, strides=1)(t)

    return t
```

FIGURE A.6: Algorithm: Definition of CAE Network Function. Structure of the layer functions are shown in the code. Are used to define the model as shown in the source coder above.

```
def train_model(model, optimizer, x_true, x_in):

    with tf.GradientTape() as g:
        x_pred = model(x_in)
        loss = tf.reduce_sum(tf.math.squared_difference(x_pred , x_true))
        gradients = g.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss, x_pred
```

FIGURE A.7: Algorithm: Training Function of Denoising Network. Noisy data is input of the model, clean samples are the target values of the loss-function.

```
batch_size=100
series_length=500
optimizer = tf.keras.optimizers.Adam(1e-4)
opt_steps = 150

for i in range(opt_steps):

    for z_train, y_train, a_train, m_train, r_train in train_dataset:

        x_in = y_train
        train_model(model=model, optimizer=optimizer, x_true=z_train, x_in=x_in)
```

FIGURE A.8: Algorithm: Optimization Function of Denoising Network. The training is done for the number of optimization steps for all training samples. Hyperparameter specification as batch-size, optimizer and learning-rate are stated in the header.

```
def U_ConvNet3(self):

    f = [4, 8, 16, 32, 64]
    inputs = tf.keras.layers.Input((self.series_length, self.input_dim, 1))

    p0 = inputs
    c1, p1 = self.down_block(p0, f[0])
    c2, p2 = self.down_block(p1, f[1])
    c3, p3 = self.down_block(p2, f[2])

    bn = self.bottleneck(p3, f[3])

    u1 = self.up_block(bn, c3, f[2])
    u2 = self.up_block(u1, c2, f[1])
    u3 = self.up_block(u2, c1, f[0])

    outputs = tf.keras.layers.Conv2D(1, (1,1), padding="same")(u3)
    model = tf.keras.models.Model(inputs, outputs)

    return model
```

FIGURE A.9: Algorithm: Definition of U-Net Network Function. Layer structure and kernel sizes shown in the code.

```
def down_block(self, x, filters, kernel_size=(3,3), padding="same", strides=1):
    c = tf.keras.layers.BatchNormalization()(x)
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="tanh")(c)
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides)(c)
    p = tf.keras.layers.MaxPool2D((2,1), (2,1))(c)

    return c, p

def up_block(self, x, skip, filters, kernel_size=(3,3), padding="same", strides=1):
    us = tf.keras.layers.UpSampling2D((2,1))(x)
    concat = tf.keras.layers.Concatenate()([us, skip])
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="tanh")(concat)
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides)(c)

    return c

def bottleneck(self,  x, filters, kernel_size=(3,3), padding="same", strides=1):
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="tanh")(x)
    c = tf.keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides)(c)

    return c
```

FIGURE A.10: Algorithm: Definition of U-Net Network Function. Structure of the layer functions are shown in the code. Are used to define the model as shown in the source coder above.

```
def train_model(model, optimizer, x_true, x_in):

    with tf.GradientTape() as g:
        x_pred = model(x_in)
        loss = tf.reduce_mean(tf.abs(x_pred[:,:,:,0] - x_true)**2)
        gradients = g.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss, x_pred
```

FIGURE A.11: Algorithm: Training Function of U-Net for State Estimation. Observable states are input of the model, unobservable are target value of the loss-function.

```
batch_size=100
series_length=512
optimizer = tf.keras.optimizers.Adam(3e-4)
opt_steps = 300

for i in range(opt_steps):

    for z_train, y_train, a_train, m_train, r_train in train_dataset:

        x_in = y_train
        train_model(model=model_test, optimizer=optimizer, x_true=z_train, x_in=x_in)
```

FIGURE A.12: Algorithm: Optimization Function of U-Net for State Estimation. The training is done for the number of optimization steps for all training samples. Hyperparameter specification as batch-size, optimizer and learning-rate are stated in the header.

```
def A_to_p_rel(A):

    k=0
    l=0
    p = np.zeros([A.shape[0], 2*A.shape[1]-2]).astype(np.float32)
    z, s = rel_indizes(size=size)
    for j in range(size):
        if j == 0:
            p[:,l]=-A[:,z[k], s[k]].numpy()
            p[:,l+1]=-A[:, z[k+2], s[k+2]].numpy()
            k+=4
            l+=2

        elif 0<j<size-1:
            p[:,l]=A[:, z[k], s[k]].numpy()
            p[:,l+1]=A[:, z[k+2], s[k+2]].numpy()
            p[:,l+2]=A[:, z[k+3], s[k+3]].numpy()
            p[:,l+3]=A[:, z[k+3+2], s[k+3+2]].numpy()
            k+=6
            l+=4

        else:
            p[:,l]=A[:, z[k], s[k]].numpy()
            p[:,l+1]=-A[:, z[k+1], s[k+1]].numpy()
            p[:,l+2]=A[:, z[k+2], s[k+2]].numpy()
            p[:,l+3]=-A[:, z[k+3], s[k+3]].numpy()

    p_out = tf.constant(p)

    return p_out
```

FIGURE A.13: Algorithm: Mapping System Matrix to Parameter Vector.

```
def data_completion23(model, x_obs, x_true):

    x_pred = model(x_obs)

    x_full = np.zeros([x_obs.shape[0], x_obs.shape[1], x_obs.shape[2]+1]).astype(np.float32)

    x_full[:,:,0] = x_obs[:,:,0].numpy()
    x_full[:,:,1:3] = x_pred[:,:,:,0].numpy()

    return tf.constant(x_full)
```

FIGURE A.14: Algorithm: Data Completion for Gauss-Newton Optimization with pre-trained U-Net.

```
def ann_init_system_matrix(p_init, A):

    A[:,0,0] = -p_init[:,0]
    A[:,0,1] = p_init[:,0]
    A[:,0,3] = -p_init[:,1]
    A[:,0,4] = p_init[:,1]


    return A
```

FIGURE A.15: Algorithm: Parameter Initialization for $p_1$ and $p_2$ with Convolutional Neural Network.

```python
def gauss_newton(A, a, y, B, u, h, n, d, input_dim, del_index, stepsize, time_steps, rel_size, rel_r_red, rel_c_red):
    R = np.zeros([batch_size, input_dim*time_steps, 1]).astype(np.float32)
    J = np.zeros([batch_size, input_dim*time_steps, rel_size]).astype(np.float32)
    with tf.GradientTape(persistent=True) as gv:

        loss = tf.constant(0.0)
        v = tf.constant(np.zeros([y.shape[0], size, 1]), dtype=np.float32)
        x = tf.constant(np.zeros([y.shape[0], size, 1]), dtype=np.float32)
        r_step = 0

        for i in range(time_steps):

            v_neu = v + h*(tf.matmul(A[:,0:n, 0:n], v)+tf.matmul(A[:,0:n,n:d], x)+tf.matmul(B, u[:,i,:,:])[:,0:n])
            x_neu = x + h*(tf.matmul(A[:,n:d, 0:n], v_neu))
            z = (tf.matmul(A[:,0:n, 0:n],  v_neu)+tf.matmul(A[:,0:n,n:d], x_neu))[:,:,0]+tf.matmul(B, u[:,i+1,:,:])[:,0:n,0]
            x = x_neu
            v = v_neu

            d0 = (z[:,0]-y[:,i+1,0])
            d1 = (z[:,1]-y[:,i+1,1])
            d2 = (z[:,2]-y[:,i+1,2])

            gradients_0 = gv.gradient(z[:,0], [A])
            gradients_1 = gv.gradient(z[:,1], [A])
            gradients_2 = gv.gradient(z[:,2], [A])

            J[:,r_step,:]=gradients_0[0].numpy()[:, rel_r_red, rel_c_red]
            J[:,r_step+1,:]=gradients_1[0].numpy()[:, rel_r_red, rel_c_red]
            J[:,r_step+2,:]=gradients_2[0].numpy()[:, rel_r_red, rel_c_red]

            R[:,r_step,0]=d0.numpy()
            R[:,r_step+1,0]=d1.numpy()
            R[:,r_step+2,0]=d2.numpy()

            r_step+=input_dim

    Jacobian = tf.constant(J)
    Residuum = tf.constant(R)
    Jacobian_t = tf.transpose(Jacobian, perm=[0,2,1])
    Gradient = tf.matmul(Jacobian_t, Residuum)
    Fisher = tf.matmul(Jacobian_t, Jacobian)
    Fisher_inv = tf.linalg.inv(Fisher)

    K = correlation_matrix(C)
    Direction = tf.matmul(C, Gradient)

    return A, R, Fisher_inv, Gradient
```

FIGURE A.16: Algorithm: State Estimation Model for Gauss-Newton Method. Semi-Explicit Euler is used for State Estimation. Derivatives and Residuals are computed for each step.

```python
def efficient_update3(A, A_true, del_index, gradients, stepsize):

    size = int(A.shape[1]/2)
    rel_r, rel_c = rel_indizes2(size=size)

    del_r = rel_r[del_index]
    del_c = rel_c[del_index]

    rel_r_red = np.delete(rel_r, del_index)
    rel_c_red = np.delete(rel_c, del_index)

    #1 Gauss Newton Update
    for i in range(rel_r_red.shape[0]):
        A[:,rel_r_red[i], rel_c_red[i]].assign(A[:,rel_r_red[i], rel_c_red[i]]-stepsize*gradients[:,i,0])

    #2 Ground Truth Projection
    for j in range(del_r.shape[0]):
        A[:,rel_r[del_index[j]], rel_c[del_index[j]]].assign(tf.constant(A_true[:, rel_r[del_index[j]], rel_c[del_index[j]]]))

    #3 Restructure System Matrix
    for k in range(2):

        if rel_r[k]==0:
            A[:, rel_r[k], rel_c[k]+1].assign(-A[:, rel_r[k], rel_c[k]])

    for l in range(1, size-1):

        A[:, l, l].assign(-(A[:, l, l-1]+A[:, l, l+1]))
        A[:, l, l+size].assign(-(A[:, l, l-1+size]+A[:, l, l+1+size]))

    return A
```

FIGURE A.17: Algorithm: Optimization Step for System Matrix. Gradient direction and parameter vector are used to update the new values of the parameters. Then the parameter vector is mapped to the system matrix.

```python
def system_identification_reg(size, a, y, y_in, r, u, B, time_steps, opt_steps, generator):

    prefix = "GN_ARMIJO_EXP_FULL"
    d = size*2
    n = size
    stepsize=1.0
    h = 0.01
    u = tf.reshape(u,  [u.shape[0], u.shape[1], u.shape[2], 1])

    A_init = system_init(batch_size=batch_size, size=size)
    p_init = model_init(y).numpy()
    A_init = ann_init_system_matrix(p_init=p_init, A=A_init, A_true=a.numpy())

    A = tf.Variable(A_init)
    p_true = A_to_p_rel(a)

    rel_r, rel_c = rel_indizes2(size=size)
    del_index = [2,4]

    rel_r_red = np.delete(rel_r, del_index)
    rel_c_red = np.delete(rel_c, del_index)
    rel_size = rel_r_red.shape[0]
    input_dim = 3

    for j in range(opt_steps):

        A_armijo, Residual, Direction, Grdient = gauss_newton(A=A, a=a, y=y, B=B, u=u, h=h,del_index=del_index,
                                                      n=n, d=d, input_dim=input_dim, stepsize=stepsize,
                                                      time_steps=time_steps, rel_size=rel_size,
                                                      rel_r_red=rel_r_red, rel_c_red=rel_c_red)

        dp = tf.matmul(Gradient, Direction)
        m = tf.reduce_sum(dp).numpy()
        res_sum0 = np.abs(Residual).sum()
        grad_dir = res_sum0 - stepsize*m
        res_sum = res_sum0

        while res_sum > grad_dir:
            stepsize = stepsize/2

            A_armijo, Residual, Direction, Grdient = gauss_newton(A=A, a=a, y=y, B=B, u=u, h=h,del_index=del_index,
                                                      n=n, d=d, input_dim=input_dim, stepsize=stepsize,
                                                      time_steps=time_steps, rel_size=rel_size,
                                                      rel_r_red=rel_r_red, rel_c_red=rel_c_red)

            A_armijo = efficient_update3(A=A_armijo, A_true=a, del_index=del_index, gradients=Direction, stepsize=stepsize)
            dp = tf.matmul(Gradient, Direction)
            m = tf.reduce_sum(dp).numpy()
            res_sum = np.abs(Residual).sum()
            grad_dir = res_sum - stepsize*m

        res_sum0 = res_sum
        A = A_armijo
        G_sum = tf.reduce_mean(stepsize*tf.abs(Direction))
        p_pred = A_to_p_rel(A)

        if G_sum.numpy() < 0.001:
            print("Optimization finished")
            break
        else:
            continue

    return Residual, A
```

FIGURE A.18: Algorithm: Gauss-Newton Optimization Step. Complete optimization algorithm, including Armijo rule, for estimation of the relevant system parameter. Method is terminated, if there is no sufficient change in the updated parameter value.

```python
for z_set, x_set, y_set, u_set, r_set, b_set, a_set, m_set in train_dataset:

    y_pred = data_completion23(model=model, x_obs=y_set, x_true=z_set)
    system_identification_reg(size=int(a_set.shape[1]/2),a=a_set, y=y_pred,
                              y_in=y_set,r=r_set, u=u_set,  B=b_set,
                              time_steps=100, opt_steps=100, generator=generator)
```

FIGURE A.19: Algorithm: Main Function of the optimization. For all elements of the test-dataset, the data completion is done. Then the parameter estimation algorithm is executed.

# Appendix B

# Neural Network Structures

```
Convolutional Neural Network Structure
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 500, 2)]          0
_________________________________________________________________
conv1d (Conv1D)              (None, 250, 100)          10100
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 125, 100)          100100
_________________________________________________________________
flatten (Flatten)            (None, 12500)             0
_________________________________________________________________
dense (Dense)                (None, 100)               1250100
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22
=================================================================
Total params: 1,361,332
Trainable params: 1,361,332
Non-trainable params: 0
=================================================================
```

FIGURE B.1: Layer Structure of a Convolutional Neural Network

```
Convolutional Auto-Encoder Structure
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 500, 2)]          0
_________________________________________________________________
conv1d (Conv1D)              (None, 250, 50)           5050
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 125, 100)          250100
_________________________________________________________________
conv1d_transpose (Conv1DTran (None, 250, 100)          500100
_________________________________________________________________
conv1d_transpose_1 (Conv1DTr (None, 500, 50)           250050
_________________________________________________________________
conv1d_transpose_2 (Conv1DTr (None, 500, 2)            5002
=================================================================
Total params: 1,010,302
Trainable params: 1,010,302
Non-trainable params: 0
=================================================================
```

FIGURE B.2: Layer Structure of a Convolutional Auto-Encoder

```
U-Net Layer Structure
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 512, 2, 1)]  0
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 512, 2, 1)    4           input_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 512, 2, 4)    40          batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 512, 2, 4)    148         conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 256, 2, 4)    0           conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 256, 2, 4)    16          max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 256, 2, 8)    296         batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 256, 2, 8)    584         conv2d_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 128, 2, 8)    0           conv2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 128, 2, 8)    32          max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 128, 2, 16)   1168        batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 128, 2, 16)   2320        conv2d_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 64, 2, 16)    0           conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 64, 2, 32)    4640        max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 64, 2, 32)    9248        conv2d_6[0][0]
__________________________________________________________________________________________________
up_sampling2d (UpSampling2D)    (None, 128, 2, 32)   0           conv2d_7[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 128, 2, 48)   0           up_sampling2d[0][0]
                                                                 conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 128, 2, 16)   6928        concatenate[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 128, 2, 16)   2320        conv2d_8[0][0]
__________________________________________________________________________________________________
up_sampling2d_1 (UpSampling2D)  (None, 256, 2, 16)   0           conv2d_9[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256, 2, 24)   0           up_sampling2d_1[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 256, 2, 8)    1736        concatenate_1[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 256, 2, 8)    584         conv2d_10[0][0]
__________________________________________________________________________________________________
up_sampling2d_2 (UpSampling2D)  (None, 512, 2, 8)    0           conv2d_11[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 512, 2, 12)   0           up_sampling2d_2[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 512, 2, 4)    436         concatenate_2[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 512, 2, 4)    148         conv2d_12[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 512, 2, 1)    5           conv2d_13[0][0]
==================================================================================================
Total params: 30,653
Trainable params: 30,627
Non-trainable params: 26
__________________________________________________________________
```

FIGURE B.3: Layer Structure of a U-Net for sequential data segmentation

# Bibliography

[1] Wael Abbas et al. “Optimal Seat and Suspension Design for a Half-Car with Driver Model Using Genetic Algorithm”. In: *Intelligent Control and Automation* 04 (Jan. 2013), pp. 199–205. DOI: 10.4236/ica.2013.42024.

[2] Luis Aguirre. “Controllability and observability of linear systems: some noninvariant aspects”. In: *Education, IEEE Transactions on* 38 (Mar. 1995), pp. 33 –39. DOI: 10.1109/13.350218.

[3] Md Zahangir Alom et al. “The history began from alexnet: A comprehensive survey on deep learning approaches”. In: *arXiv preprint arXiv:1803.01164* (2018).

[4] Ervin Alvarez-Sánchez. “A Quarter-Car Suspension System: Car Body Mass Estimator and Sliding Mode Control”. In: *Procedia Technology* 7 (2013). 3rd Iberoamerican Conference on Electronics Engineering and Computer Science, CIIECC 2013, pp. 208–214. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2013.04.026. URL: https://www.sciencedirect.com/science/article/pii/S2212017313000273.

[5] Emmanuel Assidjo et al. “A Hybrid Neural Network Approach for Batch Fermentation Simulation”. In: *Australian Journal of Basic and Applied Sciences* 3 (Oct. 2009), pp. 3930–3936.

[6] Richard Aster, Brian Borchers, and Clifford Thurber. “Parameter Estimation and Inverse Problems”. In: *Recherche* 67 (Jan. 2012), p. 02.

[7] Jan Awrejcewicz. “Mathematical and Physical Pendulum”. In: May 2012, pp. 69–106. ISBN: 978-1-4614-3739-0. DOI: 10.1007/978-1-4614-3740-6_2.

[8] A. Bartsch, F. Fitzek, and R. Rasshofer. “Pedestrian recognition using automotive radar sensors”. In: *Advances in Radio Science* 10 (Sept. 2012), pp. 45–55. DOI: 10.5194/ars-10-45-2012.

[9] Ror Bellman and Karl Johan Åström. “On structural identifiability”. In: *Mathematical biosciences* 7.3-4 (1970), pp. 329–339.

[10] Julius Berner et al. “The modern mathematics of deep learning”. In: *arXiv preprint arXiv:2105.04026* (2021).

[11] Alexandr Borovkov. *Stochastic processes in queueing theory*. Vol. 4. Springer Science & Business Media, 2012.

[12] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186.

[13] Martin Braun. *Differential equations and their applications*. Vol. 1. Springer.

[14] Charles-Edouard Bréhier. "Introduction to numerical methods for Ordinary Differential Equations". Licence. Lecture - Cours donné à Pristina (Kosovo), en décembre 2016, dans le cadre de l'école de recherche Franco-Kosovarde en Mathématiques. Pristina, Kosovo, Serbia, Dec. 2016. URL: https://hal.archives-ouvertes.fr/cel-01484274.

[15] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[16] G. D. Buckner and K. T. Schuetze. "Intelligent Estimation of System Parameters for Active Vehicle Suspension Control". In: *SAE Transactions* 108 (1999), pp. 1257–1263. ISSN: 0096736X, 25771531. URL: http://www.jstor.org/stable/44667996 (visited on 06/17/2022).

[17] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[18] Claudio G. Carvalhaes and Patrick Suppes. "Approximations for the period of the simple pendulum based on the arithmetic-geometric mean". In: *American Journal of Physics* 76.12 (2008), pp. 1150–1154. DOI: 10.1119/1.2968864. eprint: https://doi.org/10.1119/1.2968864. URL: https://doi.org/10.1119/1.2968864.

[19] Peter J Collins. *Differential and integral equations*. Oxford University Press, 2006.

[20] Antonia Creswell et al. "Generative adversarial networks: An overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65.

[21] Silvia Curteanu and Florin Leon. "Hybrid neural network models applied to a free radical polymerization process". In: *Polymer-Plastics Technology and Engineering* 45.9 (2006), pp. 1013–1023.

[22] Frederik Michel Dekking et al. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.

[23] Georg Dorffner. "Neural Networks for Time Series Processing". In: *Neural Network World* 6 (1996), pp. 447–468.

[24] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[25] W P Eaton and J H Smith. "Micromachined pressure sensors: review and recent developments". In: *Smart Materials and Structures* 6.5 (1997), pp. 530–539. DOI: 10.1088/0964-1726/6/5/004. URL: https://doi.org/10.1088/0964-1726/6/5/004.

[26] W.J. Fleming. "Overview of automotive sensors". In: *IEEE Sensors Journal* 1.4 (2001), pp. 296–308. DOI: 10.1109/7361.983469.

[27] Andronic Florin, Manolache-Rusu Ioan-Cozmin, and Pătuleanu Liliana. “Passive suspension modeling using MATLAB, quarter-car model, input signal step type”. In: *New technologies and products in machine manufacturing technologies* (2013), pp. 258–263.

[28] Henri P. Gavin. “The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems c ©”. In: 2013.

[29] Izrail Moiseevitch Gelfand, Richard A Silverman, et al. *Calculus of variations*. Courier Corporation, 2000.

[30] Jean Charles Gilbert. “On the Realization of the Wolfe Conditions in Reduced Quasi-Newton Methods for Equality Constrained Optimization”. In: *Siam Journal on Optimization* 7 (Aug. 1997), pp. 780–813. DOI: 10.1137/S1052623493259604.

[31] Vladimír Goga and Marian Kl’účik. “Optimization of Vehicle Suspension Parameters with use of Evolutionary Computation”. In: *Procedia Engineering* 48 (Dec. 2012), 174–179. DOI: 10.1016/j.proeng.2012.09.502.

[32] Herbert Goldstein, Charles Poole, and John Safko. *Classical mechanics*. 2002.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[34] Loukas Grafakos. *Classical fourier analysis*. Vol. 2. Springer, 2008.

[35] Serge Gratton, Amos Lawless, and Nancy Nichols. “Approximate Gauss–Newton Methods for Nonlinear Least Squares Problems”. In: *SIAM Journal on Optimization* 18 (Jan. 2007), pp. 106–132. DOI: 10.1137/050624935.

[36] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. In: *Pattern Recognition* 77 (2018), pp. 354–377.

[37] Rahmi Güçlü. “Active control of seat vibrations of a vehicle model using various suspension alternatives”. In: *Turkish Journal of Engineering and Environmental Sciences* 27.6 (2003), pp. 361–374.

[38] Bing Guo and Youting Shen. “Modeling approach to coal gasification using hybrid neural networks”. In: 37 (Feb. 1997), pp. 11–15.

[39] Ernst Hairer and Gerhard Wanner. “Euler methods, explicit, implicit, symplectic”. In: *Encyclopedia of Applied and Computational Mathematics* 1 (2015), pp. 451–455.

[40] Ernst Hairer et al. “Geometric numerical integration”. In: *Oberwolfach Reports* 3.1 (2006), pp. 805–882.

[41] Herman O Hartley and Aaron Booker. “Nonlinear least squares estimation”. In: *The Annals of mathematical statistics* 36.2 (1965), pp. 638–650.

[42] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[43] Robert Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[44] C. J. Himmelberg and F. S. Van vleck. “Some Selection Theorems for Measurable Functions”. In: *Canadian Journal of Mathematics* 21 (1969), 394–399. DOI: 10.4153/CJM-1969-041-7.

[45] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[46] Fabian Höflinger et al. “A wireless micro inertial measurement unit (IMU)”. In: *IEEE Transactions on instrumentation and measurement* 62.9 (2013), pp. 2583–2595.

[47] Scott Ikenaga et al. “Active suspension control of ground vehicle based on a full-vehicle model”. In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 6. IEEE. 2000, pp. 4019–4024.

[48] Wu Jih-Huah, Pen Cheng-Chung, and Joe-Air Jiang. “Applications of the Integrated High-Performance CMOS Image Sensor to Range Finders – from Optical Triangulation to the Automotive Field”. In: *Sensors* 8 (Mar. 2008). DOI: 10.3390/s8031719.

[49] Felipe Jiménez et al. “Advanced Driver Assistance System for Road Environments to Improve Safety and Efficiency”. In: *Transportation Research Procedia* 14 (2016). Transport Research Arena TRA2016, pp. 2245–2254. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2016.05.240. URL: https://www.sciencedirect.com/science/article/pii/S2352146516302460.

[50] Jack Judy. “Microelectromechanical systems (MEMS): Fabrication, design and applications”. In: *Smart Materials and Structures* 10 (Dec. 2001), pp. 1115–1134. DOI: 10.1088/0964-1726/10/6/301.

[51] Rudolf Emil Kalman. “Mathematical description of linear dynamical systems”. In: *Journal of the Society for Industrial and Applied Mathematics, Series A: Control* 1.2 (1963), pp. 152–192.

[52] Halil Karahan, Gurhan Gurarslan, and Zong Woo Geem. “Parameter estimation of the nonlinear Muskingum flood-routing model using a hybrid harmony search algorithm”. In: *Journal of Hydrologic Engineering* 18.3 (2013), pp. 352–360.

[53] Suheil Khuri. “A Laplace decomposition algorithm applied to a class of nonlinear differential equations”. In: *Journal of Applied Mathematics* 1 (Jan. 2001). DOI: 10.1155/S1110757X01000183.

[54] Tom Kibble and Frank H Berkshire. *Classical mechanics*. world scientific publishing company, 2004.

[55] Myoungsoo Kim et al. “A Hybrid Neural Network Model for Power Demand Forecasting”. In: *Energies* 12.5 (2019). ISSN: 1996-1073. DOI: 10.3390/en12050931. URL: https://www.mdpi.com/1996-1073/12/5/931.

[56] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[57] Diederik P Kingma and Max Welling. “An introduction to variational autoencoders”. In: *arXiv preprint arXiv:1906.02691* (2019).

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[59] Ambarish Kulkarni, Sagheer Ranjha, and Ajay Kapoor. "A quarter-car suspension model for dynamic evaluations of an in-wheel electric vehicle". In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 232 (Oct. 2017), p. 095440701772716. DOI: 10.1177/0954407017727165.

[60] Vivek Kumar. "Modelling and Simulation of a Passenger Car for Comfort Evaluation". In: *International Journal for research in Applied Science and Engineering Technology* Vol. 6 (Apr. 2018). DOI: 10.22214/ijraset.2018.4662.

[61] Rafał Kwiatkowski, Tadeusz J. Hoffmann, and Andrzej Kołodziej. "Dynamics of a Double Mathematical Pendulum with Variable Mass in Dimensionless Coordinates". In: *Procedia Engineering* 177 (2017). XXI Polish-Slovak Scientific Conference Machine Modeling and Simulations MMS 2016.September 6-8, 2016, Hucisko, Poland, pp. 439–443. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2017.02.242. URL: https://www.sciencedirect.com/science/article/pii/S1877705817307506.

[62] Twan van Laarhoven. "L2 Regularization versus Batch and Weight Normalization". In: *CoRR* abs/1706.05350 (2017). arXiv: 1706.05350. URL: http://arxiv.org/abs/1706.05350.

[63] Tara Larrue, Xiaoxu Meng, and Changyoung Han. "Denoising Videos with Convolutional Autoencoders A Comparison of Autoencoder Architectures". In: 2018.

[64] Yann LeCun et al. "A theoretical framework for back-propagation". In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988, pp. 21–28.

[65] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[66] Hyo Jong Lee. "Neural network approach to identify model of vehicles". In: *International Symposium on Neural Networks*. Springer. 2006, pp. 66–72.

[67] Daniel Liberzon. *Calculus of variations and optimal control theory*. Princeton university press, 2011.

[68] Lennart Ljung. "System identification". In: *Signal analysis and prediction*. Springer, 1998, pp. 163–173.

[69] Lennart Ljung. "Perspectives on system identification". In: *Annual Reviews in Control* 34.1 (2010), pp. 1–12.

[70] Wei Lu and Namrata Vaswani. *The Wiener-Khinchin Theorem for Non-wide Sense stationary Random Processes*. 2009. arXiv: 0904.0602 [math.ST].

[71] Piotr Mackowiak et al. "Development and fabrication of a very High-g sensor for very high impact applications". In: *Journal of Physics: Conference Series* 757 (Oct. 2016), p. 012016. DOI: 10.1088/1742-6596/757/1/012016.

[72] K Manoj Mahala, Prasanna Gadkari, and Anindya Deb. “Mathematical models for designing vehicles for ride comfort”. In: *ICORD 09: Proceedings of the 2nd International Conference on Research into Design, Bangalore, India 07.-09.01. 2009*. 2009.

[73] K.R. Meyer and G.R. Hall. *Introduction to Hamiltonian Dynnamical Systems and the N-Body Problem*. Vol. 2. Springer-Verlag New York, 2009. DOI: 10.1007/978-0-387-09724-4.

[74] Mahesh P Nagarkar, Gahininath J Vikhe Patil, and Rahul N Zaware Patil. “Optimization of nonlinear quarter car suspension–seat–driver model”. In: *Journal of advanced research* 7.6 (2016), pp. 991–1007.

[75] Mark Nagurka and Shuguang Huang. “A mass-spring-damper model of a bouncing ball”. In: *Proceedings of the 2004 American control conference*. Vol. 1. IEEE. 2004, pp. 499–504.

[76] G. Nagy. *Ordinary Differential Equations.* 2016. URL: https://simiode.org/resources/2681.

[77] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[78] Jorge Nocedal and Ya-xiang Yuan. “Combining trust region and line search techniques”. In: *Advances in nonlinear programming*. Springer, 1998, pp. 153–175.

[79] Rui Oliveira. “Combining first principles modelling and artificial neural networks: A general framework”. In: *Computers & Chemical Engineering* 28 (May 2004), pp. 755–766. DOI: 10.1016/j.compchemeng.2004.02.014.

[80] Luis Perez and Jason Wang. “The effectiveness of data augmentation in image classification using deep learning”. In: *arXiv preprint arXiv:1712.04621* (2017).

[81] Thilo Pfau, Thomas H Witte, and Alan M Wilson. “A method for deriving displacement data during cyclical movement using an inertial sensor”. In: *Journal of Experimental Biology* 208.13 (2005), pp. 2503–2514.

[82] Igor Podlubny. “The Laplace Transform Method for Linear Differential Equations of the Fractional Order. UEF-02-94, The Academy of Scien”. In: *Inst. of Exp. Phys., Kosice, Slovak Republic* (1994), p. 20.

[83] R. Gerhard Pratt, Changsoo Shin, and G. J. Hick. “Gauss–Newton and full Newton methods in frequency–space seismic waveform inversion”. In: *Geophysical Journal International* 133.2 (May 1998), pp. 341–362. ISSN: 0956-540X. DOI: 10.1046/j.1365-246X.1998.00498.x. eprint: https://academic.oup.com/gji/article-pdf/133/2/341/1550344/133-2-341.pdf. URL: https://doi.org/10.1046/j.1365-246X.1998.00498.x.

[84] Dimitris C Psichogios and Lyle H Ungar. “A hybrid neural network-first principles approach to process modeling”. In: *AIChE Journal* 38.10 (1992), pp. 1499–1511.

[85] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[86] Jitendra R Raol, Gopalrathnam Girija, and Jatinder Singh. *Modelling and parameter estimation of dynamic systems*. Vol. 65. Iet, 2004.

[87] A. Raue et al. “Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood”. In: *Bioinformatics* 25.15 (June 2009), pp. 1923–1929. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp358. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/15/1923/16889623/btp358.pdf. URL: https://doi.org/10.1093/bioinformatics/btp358.

[88] Martin Riedmiller and Thomas Gabel. “On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup”. In: *2007 IEEE Symposium on Computational Intelligence and Games*. IEEE. 2007, pp. 17–23.

[89] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[90] Santiago Royo and Maria Ballesta-Garcia. “An Overview of Lidar Imaging Systems for Autonomous Vehicles”. In: *Applied Sciences* 9.19 (2019). ISSN: 2076-3417. DOI: 10.3390/app9194093. URL: https://www.mdpi.com/2076-3417/9/19/4093.

[91] David E Rumelhart et al. “Backpropagation: The basic theory”. In: *Backpropagation: Theory, architectures and applications* (1995), pp. 1–34.

[92] Shibani Santurkar et al. “How does batch normalization help optimization?” In: *Proceedings of the 32nd international conference on neural information processing systems*. 2018, pp. 2488–2498.

[93] Werner Schiehlen and Peter Eberhard. *Technische dynamik*. Springer, 2017.

[94] Joel Schiff. *The Laplace Transform: Theory and Applications*. Jan. 1999. ISBN: 978-1-4757-7262-3. DOI: 10.1007/978-0-387-22757-3.

[95] Tasnim Shaikh, Dr. Satyajeet Chaudhari, and Hiren Rasania. “Air Bag: A Safety Restraint System of an Automobile”. In: *Int. Journal of Engineering Research and Application* 3 (Oct. 2013), pp. 615–621.

[96] Dongwoo Sheen, Ian H. Sloan, and Vidar Thomée. “A Parallel Method for Time-Discretization of Parabolic Problems Based on Contour Integral Representation and Quadrature”. In: *Mathematics of Computation* 69.229 (2000), pp. 177–195. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2584835.

[97] Isaac Skog and Peter Händel. “Calibration of a MEMS inertial measurement unit”. In: *XVII IMEKO world congress*. Citeseer. 2006, pp. 1–6.

[98] Foo Chong Soon et al. “PCANet-Based Convolutional Neural Network Architecture For a Vehicle Model Recognition System”. In: *IEEE Transactions on Intelligent Transportation Systems* PP (June 2018), pp. 1–11. DOI: 10.1109/TITS.2018.2833620.

[99] Nathan A Spielberg et al. “Neural network vehicle models for high-performance automated driving”. In: *Science robotics* 4.28 (2019), eaaw1975.

[100] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[101] Shelby Stanhope, J Rubin, and David Swigon. “Identifiability of Linear and Linear-in-Parameters Dynamical Systems from a Single Trajectory”. In: *SIAM Journal on Applied Dynamical Systems* 13 (Jan. 2014), pp. 1792–1815. DOI: 10.1137/130937913.

[102] Andrew Stuart and Anthony R Humphries. *Dynamical systems and numerical analysis*. Vol. 2. Cambridge University Press, 1998.

[103] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[104] Gerald Teschl. *Ordinary differential equations and dynamical systems*. Vol. 140. American Mathematical Soc., 2012.

[105] Vidar Thomée. “A high order parallel method for time discretization of parabolic type equations based on Laplace transformation and quadrature”. In: *International Journal of Numerical Analysis and Modeling* 2 (2005), pp. 85–96.

[106] Feng Tyan et al. “Generation of random road profiles”. In: *Journal of Advanced Engineering* 4.2 (2009), pp. 1373–1378.

[107] Michael Unterreiner. “Modellbildung und Simulation von Fahrzeugmodellen unterschiedlicher Komplexität”. Duisburg, Essen, Univ., Diss., 2013. PhD thesis. Duisburg, Essen, 2014. URL: https://duepublico.uni-due.de/servlets/DocumentServlet?id=34562.

[108] Deepak Unune and Suhas Mohite. “Ride Analysis of Quarter Vehicle Model”. In: Nov. 2011.

[109] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[110] Cong Wang et al. “A vehicle rollover evaluation system based on enabling state and parameter estimation”. In: *IEEE Transactions on Industrial Informatics* 17.6 (2020), pp. 4003–4013.

[111] Yasi Wang, Hongxun Yao, and Sicheng Zhao. “Auto-encoder based dimensionality reduction”. In: *Neurocomputing* 184 (2016), pp. 232–242.

[112] Robert Weinstock. *Calculus of variations: with applications to physics and engineering*. Courier Corporation, 1974.

[113] Yijia Zhang et al. “A hybrid model based on neural networks for biomedical relation extraction”. In: *Journal of Biomedical Informatics* 81 (2018), pp. 83–92. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2018.03.011. URL: https://www.sciencedirect.com/science/article/pii/S1532046418300534.

[114] Youqi Zhang et al. “Vibration-based structural state identification by a 1-dimensional convolutional neural network”. In: *Computer-Aided Civil and Infrastructure Engineering* 34.9 (2019), pp. 822–839.

[115] Wei Zhao et al. “Real-Time Vehicle Motion Detection and Motion Altering for Connected Vehicle: Algorithm Design and Practical Applications”. In: *Sensors* 19.19 (2019). ISSN: 1424-8220. DOI: 10.3390/s19194108. URL: https://www.mdpi.com/1424-8220/19/19/4108.

[116] L. Zimmermann et al. “Airbag application: a microsystem including a silicon capacitive accelerometer, CMOS switched capacitor electronics and true self-test capability”. In: *Sensors and Actuators A: Physical* 46.1 (1995), pp. 190–195. ISSN: 0924-4247. DOI: https://doi.org/10.1016/0924-4247(94)00888-O. URL: https://www.sciencedirect.com/science/article/pii/092442479400888O.