



Politecnico di Torino

Technische Universität München

Master Thesis in Automotive Engineering

Course of Vehicle System Development

**Creation of a database concept for expandable  
parametric vehicle architectures**

Ranocchia Gabriele, M.Sc.

A.Y. 2019/2020

Thesis Coordinator: Prof. A. Tonoli

Thesis Supervisors: Prof. M. Lienkamp

Ph.D. candidate L. Nicoletti



# Geheimhaltungsverpflichtung

Herr/Frau: Ranocchia, Gabriele

Gegenstand der Geheimhaltungsverpflichtung sind alle mündlichen, schriftlichen und digitalen Informationen und Materialien, die der Unterzeichner vom Lehrstuhl oder von Dritten im Rahmen seiner Tätigkeit am Lehrstuhl erhält. Dazu zählen vor allem Daten, Simulationswerkzeuge und Programmcode sowie Informationen zu Projekten, Prototypen und Produkten.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere:

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des Betreuers an Dritte weiterzuleiten,
- ohne Zustimmung eines Mitarbeiters keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an den Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Besondere Sorgfalt gilt im Umgang mit digitalen Daten:


- Für den Dateiaustausch dürfen keine Dienste verwendet werden, bei denen die Daten über einen Server im Ausland geleitet oder gespeichert werden (Es dürfen nur Dienste des LRZ genutzt werden (Lehrstuhlserver, Sync&Share, GigaMove).
- Vertrauliche Informationen dürfen nur in verschlüsselter Form per E-Mail versendet werden.
- Nachrichten des geschäftlichen E-Mail Kontos, die vertrauliche Informationen enthalten, dürfen nicht an einen externen E-Mail Anbieter weitergeleitet werden.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 02.12.2020

Unterschrift: \_\_\_\_\_

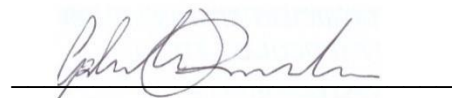




## Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 02.12.2020



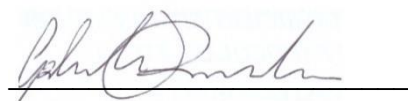
Gabriele Ranocchia, M. Sc.



## Declaration of Consent, Open Source

Hereby I, **Ranocchia, Gabriele**, born on 24.10.1996, make the software I developed during my Master Thesis available to the Institute of Automotive Technology under the terms of the license below.

Garching, 02.12.2020



Gabriele Ranocchia, M. Sc.

Copyright 2020 **Ranocchia, Gabriele**

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





# Abstract

The general contest of the doctorate project, in which the thesis is developed, aims to create a complete parametric vehicle architecture model in the early development phase of a new electric car.

The computation software used for this scope needs inputs such as requirements given by manufacturer, performance targets, and type of vehicle. Furthermore, it also needs several other factors and statistical models that cannot be defined by users but are necessary to obtain the vehicle architecture. These necessary external parameters, that are not defined by user input, are obtained in a pre-processing phase that is the core of the developed work.

The pre-processing phase that has been developed in this work, has the objective to automatize the process, reducing manual errors and maintaining an actualized structure. The creation of a database is mandatory. An interface can then interact with data contained in the database and completely retrieve all needed values for the parametric vehicle architecture model. Those values are defined as models and variables in the context of the work.

The aim of the project is the creation and organization of the whole database, the use of management keys to properly connect data with solutions to reduce the propagation of errors. Moreover, the creation of a MATLAB interface for the complete automatization of the computation of models has been carried out with the use of statistical tools. The whole implementation is devoted also to user-friendliness, with the aim to minimize manual actions. In this way, the pre-processing can be performed also by not skilled users.

The work gives a complete analysis of the database creation and implementation and the working principle of the code interface. Models and variables are used to create the whole vehicle structure. The final part gives an overview of the application of the pre-processing phase and its importance in the whole computation of the vehicle model.



# Contents

<b>List of Acronyms .....</b>	<b>III</b>
<b>List of Symbols.....</b>	<b>V</b>
<b>1 Introduction.....</b>	<b>1</b>
1.1 Tool presentation .....	2
1.2 Database integration.....	3
1.3 Thesis structure .....	3
<b>2 State of the art .....</b>	<b>5</b>
2.1 Parametric vehicle modelling .....	5
2.2 Fixed parameters .....	8
2.3 Normal distribution analysis.....	9
2.4 Linear regression analysis.....	13
2.4.1 Statistical evaluation coefficients .....	16
2.4.2 Further statistical tests .....	17
2.5 Database theory .....	20
2.5.1 Identify the entity sets and relations .....	20
2.5.2 Identify relationship types .....	21
2.5.3 Define the value of the sets and attributes .....	21
2.5.4 Organize the data and choose the primary keys .....	22
<b>3 SQL and Database implementation.....</b>	<b>25</b>
3.1 Database structure.....	25
3.2 Data and models .....	27
3.2.1 General data tables.....	28
3.2.2 Data tables.....	33
3.2.3 Additional tables.....	35
3.2.4 Calculation tables .....	36
3.3 SQL language .....	41
3.4 Creation and management of the database .....	44

3.5	Brief explanation of views.....	49
3.6	Criticalities .....	50
4	MATLAB pre-processing phase.....	53
4.1	MATLAB codes for table integration.....	53
4.1.1	Get IDs for model series connection .....	53
4.1.2	Set NULL values .....	56
4.2	MATLAB codes for complete pre-processing .....	57
4.2.1	Create the Database connection .....	58
4.2.2	Generate the MATLAB file to store results .....	59
4.2.3	Assign fixed parameters .....	59
4.2.4	Assign constant values computed from normal distribution .....	61
4.2.5	Assign catalogues.....	63
4.2.6	Assign regression models.....	64
4.2.7	Structure changes complete check .....	68
5	Spring design for dimensional chain of the rear axle .....	71
5.1	Dimensional chain of the rear axle.....	71
5.2	Spring dimensioning procedure.....	73
5.2.1	Spring catalogue .....	74
5.2.2	Geometrical check .....	77
5.2.3	Frequency check.....	79
5.2.4	Travel check.....	81
5.2.5	Resistance check.....	83
5.2.6	Buckling check .....	85
5.3	Evaluation of results .....	88
6	Conclusions and considerations.....	91
7	List of Figures .....	93
8	List of Tables .....	95
9	Appendix .....	97
9.1	Appendix A .....	97
9.2	Appendix B .....	101
9.3	Appendix C .....	103
10	References .....	105

# List of Acronyms

ADAC	Allgemeiner Deutscher Automobil-Club
adj	Adjusted
AWD	All Wheel Drive
BEV	Battery Electric Vehicle
BOF	Ball of Foot
ERM	Entity Relationship Model
fix	Fixed Parameters
FWD	Front Wheel Drive
HEV	Hybrid Electric Vehicle
ISO	International Organization for Standardization
KS Test	Kolmogorov–Smirnov Test
LDS	Longitudinal Dynamic Simulation
MAE	Mean Absolute Error
max	Maximum
min	Minimum
MSE	Means Square Error
NaN	Not A Number
nMAE	Normalized Mean Absolute Error
norm	Normal Distribution
OOST	Out of Sample Test
RDBMS	Relational Database Management System
regr	Regression
RMSE	Root Mean Square Error
RWD	Rear Wheel Drive
SAE	Society of Automotive Engineers
SgRP	Seating Reference Point
SQL	Structured Query Language
VIF	Variance Inflation Factor



# List of Symbols

-	Referred to dataset
$\hat{\phantom{x}}$	Estimated
$\mu$	Mean
$b$	Length of the suspension lower axis
$b_i$	Coefficients of linear regression model
$c$	Elastic rate
$C_v$	Coefficient of variation
$D$	Mean diameter of the spring
$d$	Wire diameter of the spring
$D_{ext}$	External diameter of the spring
$D_{int}$	Internal diameter of the spring
$d_{sa}$	Piston diameter of the shock absorber
$d_{tire}$	Diameter of the tire
$f$	Frequency of oscillation
$F$	Force
$G$	Shear modulus
$g$	Gravitational acceleration
$h_{test}$	Conventional height of loaded spring
$i$	Transmission ratio
$k$	Correction factor
$L$	Length of the coil
$m$	Mass
$m_{rear}$	Load on the rear wheels
$n$	Number of elements
$n$	Number of windings
$R^2$	Coefficient of determination
$R_m$	Maximum resistance
$s$	Displacement
$S_a$	Deformation margin between coils

## List of Symbols

---

$w$	Winding ratio
$X_{ij}$	Independent regression variable
$y_{coil\_out}$	Position in Y of the outer side of the spring
$y_{fixed\_arm}$	Position in Y of the fixing point of suspension to body with respect to the centre of the vehicle
$Y_i$	Dependent regression variable
$y_{inside\_surface\_sa}$	Position in Y of the critical point for contact with the shock absorber
$y_{sa}$	Position in Y of the lower hinge of the shock absorber with respect to the centre of the vehicle
$Z_{wh}$	Position in Z of the upper point of the wheelhouse with respect to the centre of the wheel in empty weight conditions
$\alpha$	Significance level
$\delta$	Deformation
$\varepsilon$	Residual
$\theta_{sa}$	Inclination angle of the shock absorber with respect to the Z-axis
$\lambda$	Slenderness factor
$v$	Constraint factor
$\xi$	Retained spring deflection
$\sigma$	Standard deviation
$\tau$	Shear stress
$\tau_{lim}$	Limit of shear stress



# 1 Introduction

The process of designing a new vehicle concept is a complex set of procedures and it is influenced by a wide range of factors. Those constraints are the results of the legislative restrictions, the reference market, the car manufacturer, and its brand image. Moreover, the work concerns development of electric vehicle and the electromobility branch does not have a consolidated technical history that can be relied upon for new models [1][2].

At the beginning of the vehicle design, a set of starting parameters needs to be empirically estimated on the base of a given set of inputs. As mentioned in [1] concerning internal combustion engine vehicles (ICEVs), previous model series or models from competitors are a good base to start the development of new models. On the other hand, battery electric vehicles (BEVs) do not have a consolidated model history and the number of competitor's electric vehicle on the market is still low. Moreover, most of the parameters cannot be derived from ICEVs, since BEV powertrains and their requirements are completely different [1][3]. Due to this lack of parameters, the derivation of new BEVs needs to rely on empirical models.

To do so, for modelling each vehicle component, a set of data regarding already existing vehicles is needed to create the model. Based on the set of data, empirical models can be obtained and employed to derive a vehicle architecture. Nevertheless, over time the originally collected data loses validity due to the presence of new vehicles on the market. This means, that also the models derived from this data, lose their validity. To avoid this issue, it is important to periodically update data and recompute the empirical models [2]. For this purpose, a database structure was implemented in the scope of this thesis. The hereby derived structure can be applied to store and calculate the data necessary for a vehicle architecture model. In this way, by progressively updating the data within the database, an actualised structure is always maintained and it is used as the starting point for the models calculation [2].

The aim of this work, which is presented in the following elaboration, is the creation and management of the aforementioned data structure and the development of a MATLAB interface that allows the pre-processing phase (introduced in section 1.1) to be carried out autonomously on the data contained in the database.

## 1.1 Tool presentation

The aim of the project, in which this work is developed is the creation of a tool for deriving BEV architecture in the early development design. The tool is implemented in MATLAB and operates with a limited amount of inputs. These are general vehicle characteristics, dimensions, and performances requirements.

The main problem related to the process is that to compute the whole vehicle model, many different factors are needed, and they are not all given by user input. The scope of the current work is to try to solve this problem creating an automatic procedure that is able to retrieve all the needed information without requiring any user intervention. Moreover, the project aims at the creation of a way to maintain the computed values always actualized, with an eye on user friendliness and modularity.

The flow diagram in Figure 1.1 reports the main steps of the complete process.

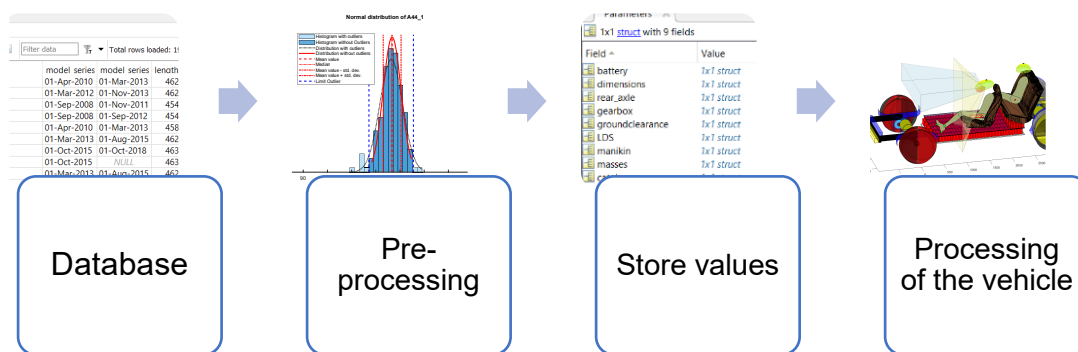


Figure 1.1: Flow of the general process

The procedure starts with the need of a database. The need of the database is better analysed in chapter 2. The database is then used to make computations in a pre-processing phase that is the core of the developed work. All computed values are then stored in a dedicated variable that is then used as an additional input for the main tool to compute the complete vehicle model.

The current work is dedicated to the first three steps of the flow. The main problem is not related to the calculation of all the needed parameters. The issue is to create a way to have an automatic computation process that needs no user intervention. Moreover, there is the need to have always updated values to not lose significance in the results of the whole procedure. As will be presented in the text, these tasks can be achieved with the use of a structured database.

## 1.2 Database integration

The database is the core element for the automatic update of empirical models. Its implementation must be structured to fulfil the following tasks:

- Have always up to date data sets to obtain the empirical models: the database can be easily updated by any user adding new rows to the existing tables or importing new tables.
- Avoid collisions, interferences, and repetitions between tables: at this purpose, an organized structure is needed.
- Let the MATLAB interface autonomously read data: some specific tables are dedicated to the MATLAB interface, they do not contain any numerical vehicle data but they are crucial to give information on how to retrieve needed parameters in the database.
- Avoid data dispersion [2]: By collecting all data sets for the models in a database, they can be all easily accessible at any moment.

The solutions that have been adopted to fulfil these requirements will be treated in the following chapters.

## 1.3 Thesis structure

The thesis is composed by six chapters (Figure 1.2).

In chapter 1, an introduction to the main tool objectives and features and a first insight to database management is performed.

In chapter 2, statistical tools for data management are analysed. Moreover, database theory and functioning are shown, including the main critical aspects of the work. Finally, an explanation of the working principle of dimensional chains is shown.

Chapter 3 describes the implementation of the database, an in-depth analysis of its structure, the explanation and representation of the models and an explanation of the main criticalities and solutions.

Chapter 4 describes the implementation of the MATLAB interface for the pre-processing phase, with a complete explanation of the interaction with the database, the technical solutions and optimizations that have been adopted and the representation and analysis of the results of the code.

Chapter 5 shows an analysis on how the dimensional chains of the tool work and how obtained models are used. Particularly, the work exploits the primary design procedure for the rear spring element in the rear axle dimensional chain. In the whole discussion of the subjects, the main attention will be focused on the models and their contents, their

importance to the main tool and analysis of results, anyway without omitting the importance of the practical work for the proper data management and further improvements that can be implemented in the future.

Finally, Chapter 6 offers an overview of the work and its results and a brief outlook on the possible future improvements.

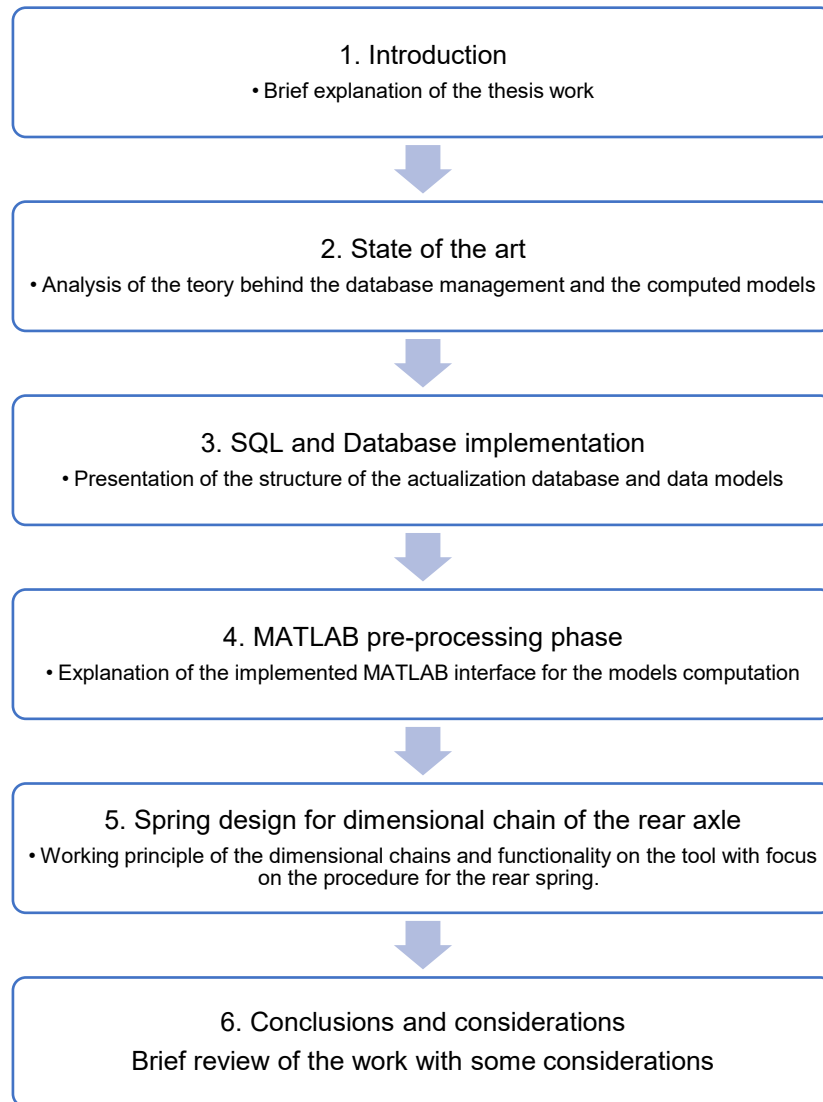


Figure 1.2: Thesis structure

## 2 State of the art

In the pre-processing phase, it is necessary to obtain from the database all the data required by the tool to perform the further calculations. These data range over different categories and types that are presented in detail in the following sections. Depending on the type of data or model, different approaches are used to obtain the results. Specifically, they can be divided into three categories: fixed parameters (section 2.2), parameters obtained with a normal distribution analysis (section 2.3) and models obtained through a linear regression (section 2.4).

This chapter presents the theory behind the three type of categories and an explanation of the MATLAB code functions to calculate normal distribution analysis and linear regression models with particular attention to their statistical properties, useful to give a quantitative evaluation of the results.

Moreover, the sequent part of the chapter (section 2.4) is dedicated to the exposition of the database theory, fundamental part for the proper data management.

Finally, in section 2.5, an explanation of the concept behind dimensional chains is made, considering the importance they have for the tool and the implementation of the computed models.

### 2.1 Parametric vehicle modelling

To define the complete architecture of a BEV vehicle, there are four different features: the dimensional concept, the powertrain topology, the component models and the dimensional chains. [1][2].

The dimensional concept is the description of internal and external dimensions, including, for the interior, the definition of seat rows and passengers. The powertrain topology describes the general position and characteristics of the powertrain components, such as the electric motor type, the gearbox type, and the battery position. Component empirical models are essential to estimate the weight and volume of the vehicle components. Finally, the dimensional chains describe the geometrical interdependencies between different components in X-, Y- and Z-direction [1][2].

By modelling the four architectural features, the tool can output a vehicle architecture (Figure 2.1).

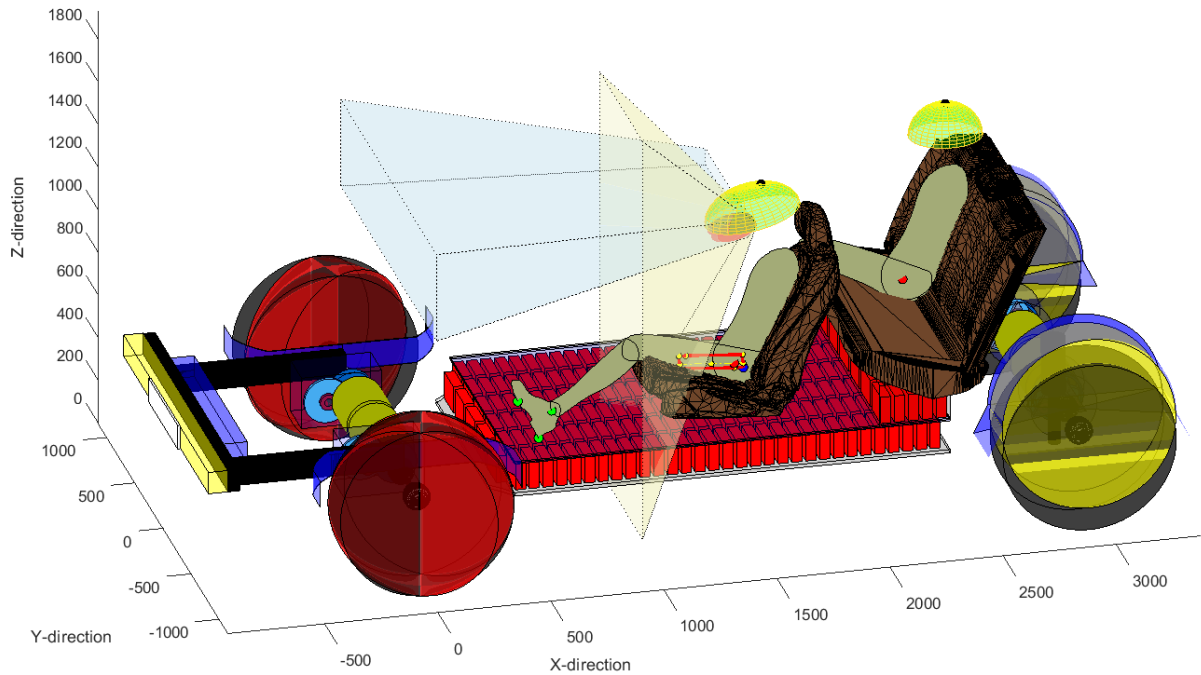


Figure 2.1: Typical graphical output of the vehicle architecture tool

The approach uses data and models for the proper computation. These are required since not all necessary values for the vehicle architecture parameters are available as input parameters.

To make a practical example, the turning circle is neither an input, nor a requirement of the car manufacturer. Anyway, it is required to estimate the maximum steering angle [4, pp. 241-242], that is in turn crucial to derive the required space for the wheel and to define the front wheel-arch in Y-direction.

For this reason, there is a set of independent inputs, which cannot be asked to the users of the tool, that are required for modelling the vehicle architecture. The aim of the current work is to obtain all those values and statistical models through a structured computation process during the pre-processing phase.

The dimensional chain procedure is one of the possible ways to determine the dimensions of a whole component or system. It consists in the subdivision of subcomponents and their addition results in the complete dimension [5].

They are the base in the early development phase to test the feasibility of the vehicle architecture. They are the tools that allow for parametrical vehicle modelling. [6]

According to [2], dimensional chains have two different tasks: position the components and evaluate the available space. To properly evaluate the available room, different constraints are considered, for example the vehicle general dimensions. Knowing the space, main components can be placed. After that, comparing the available space with the required installation space for the different components, allows the feasibility of the vehicle architecture to be checked.

Referring to Figure 2.2, it is the graphical representation of a dimensional chain. Specifically it refers to the dimensional chain to estimate the vehicle height [2][7].

The procedure starts from the ground clearance of the vehicle (H156) [8]. Knowing the battery height (BH), the height of the passenger compartment is obtained. Then, if H30 is known, the SgRP can be obtained. To the computation purposes, a 95%ile manikin has been used, with dimensions defined in [7][9].

The manikin is placed in the H30 at the lowest rearmost position, that is determined from the SgRP with the seat travel path. From that point, using dimension H62-1 and the torso angle A40-1 [7], it is possible to compute the height of the passenger compartment.

From that condition, adding the value of the roof thickness (RT), one can obtain the total vehicle height that is the purpose of the dimensional chain.

The dimensional process is reported in equation 2.1.

$$H100 = H156 + BH + H30_{min} + H62 \cdot \cos(A40) + RT \quad (2.1)$$

The general process is not monodirectional. If the vehicle height is known (H100), the process can be inverted to calculate other missing dimensions, for example to retrieve the passenger compartment available space.

This is only one example to show the main characteristics of dimensional chains. The main tool uses 15 different dimensional process that are able to fully describe the vehicle architecture, positioning all relevant components and modules and deriving available components space [2][10].

The different terms used by the tool to compute dimensional chains are known or retrieved in some way. Taking again the example of equation 2.1, in the early development phase, one of the known factors is the height of the vehicle. The general vehicle dimensions are known and used as input. Instead, some other values are unknown and not imposed as input, as for example the roof thickness.

It is in this contest that the integration with database is crucial. The storage of vehicle models, also contains information about the roof thickness of many vehicles. In this way, the real value is not known, but it can be retrieved computing an average distribution on known values (or in some cases a regression model).

Component models are based on empirical models that can be constant values derived from normal distribution and regression models. In the scope of the work referring to normal distribution or regression we refer to empirical models.

The pre-processing tool developed in the thesis can autonomously compute all the parameters which cannot be asked as an input. Moreover, it is important that all empirical models and parameters are always actualized. To this purpose, it is necessary to store data in a database [2]. The pre-processing tool interacts with the database, using a MATLAB-SQL interface, to autonomously retrieve empirical models and stores it in a structure variable that is given as input for the main tool. This gives the possibility to automatize the procedure and maintain an updated structure.

The pre-processing phase has the task to obtain all the empirical models and values that are used in the main tool for the dimensional chains.

Moreover, adding and updating data in the database, keeps all values and models automatically updated and significant for the proper computation of the parametric vehicle model.

Empirical models are presented in the following sections (fixed parameters 2.2, normal distributions 2.3, and regression models 2.4)

## 2.2 Fixed parameters

Concerning the designation of fixed parameters, the analysis is not complex because, as the name suggests, they are already established single values that as such, do not require calculations or models but are obtained and used as they are. They derive from constant values imposed by legislation, which therefore cannot be modified or do not require interpretation, or from internal procedures conventions.

Table 2.1 shows some fixed values. The ones denoted with LDS refer to the Longitudinal Dynamic Simulation, that is used for the computation of the powertrain components. Other terms are manikin dimensions that are part of the dimensional concept for the design of the passenger compartment.

Table 2.1: Examples of fixed parameters

Parameter	Value	Unit of measurement
A40	25	°
Manikin foot length	306	mm
LDS $\eta$ differential	1	-
LDS $\eta$ power electronics	0.95	-
Weight max	3500	Kg

The values of Table 2.1 are taken as they are in the table and they derive from regulations or they are imposed as conventions for the calculations. The A40 and the manikin foot length are related to the dimensional concept of manikin. They are the torso angle of the manikin and the length of the foot (FL) [11][12], respectively. Figure 2.2 shows an example of the dimensions [2][6].



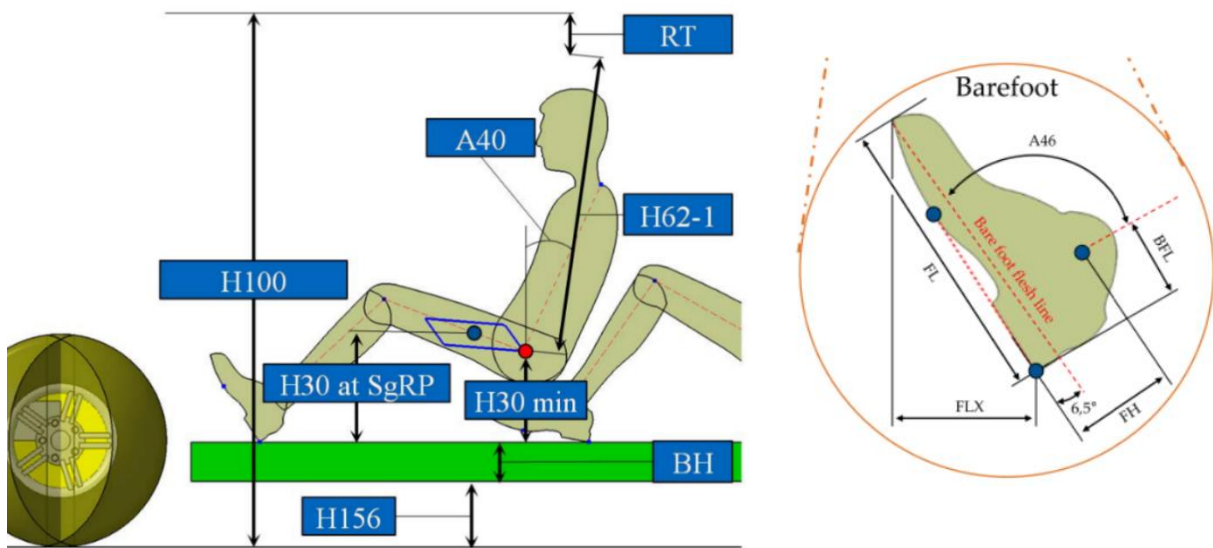


Figure 2.2: Example of manikin dimensions [2][6]

The two factors of the LDS are related to the dynamic simulation. The  $\eta$  differential is the efficiency of the differential [13] while  $\eta$  power electronics is the efficiency of the electronics from battery to motor [14]. The last term is the maximum legal mass for a vehicle [15].

Fixed parameters can be implemented by collecting them in tables of the database (and not in the tool code). Doing so, if a fixed parameter value needs to be updated, it can be easily changed in the table and the user does not need to search it through the code. After the explanation of the fixed parameters, the next section deals with the presentation of the statistical tool for the normal distribution.

## 2.3 Normal distribution analysis

Normal analysis are important in statistics to represent random variables whose distributions are not known [16]. It is usually suitable to get a constant value when no particular influence on other dimensions is present [5]. The information of the dataset can be represented graphically using a histogram (Figure 2.3).

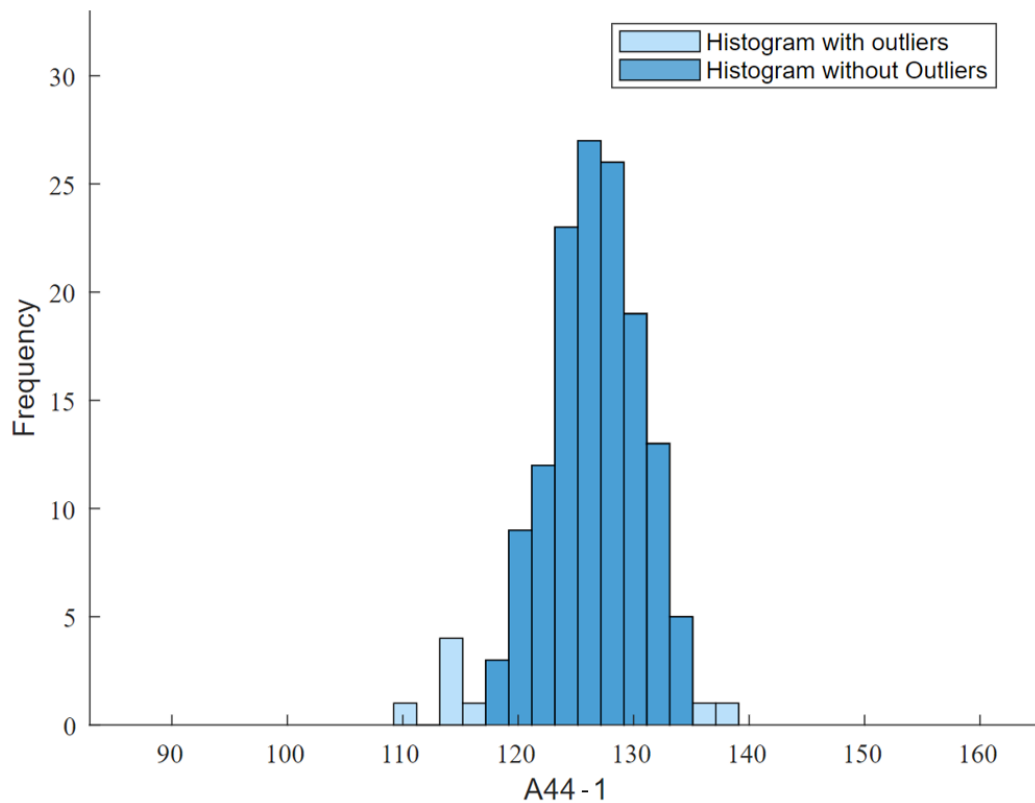


Figure 2.3: Histogram example of A44 dimension of the first row

Figure 2.3 shows the distribution of the A44 dimension of the first row. It is the angle of the manikin between the lower and upper leg, as reported in Figure 2.4 [6]. The definition of the name A44-1 states that the dimension is referred to the first row. For the second row, the same dimension is indicated with A44-2.

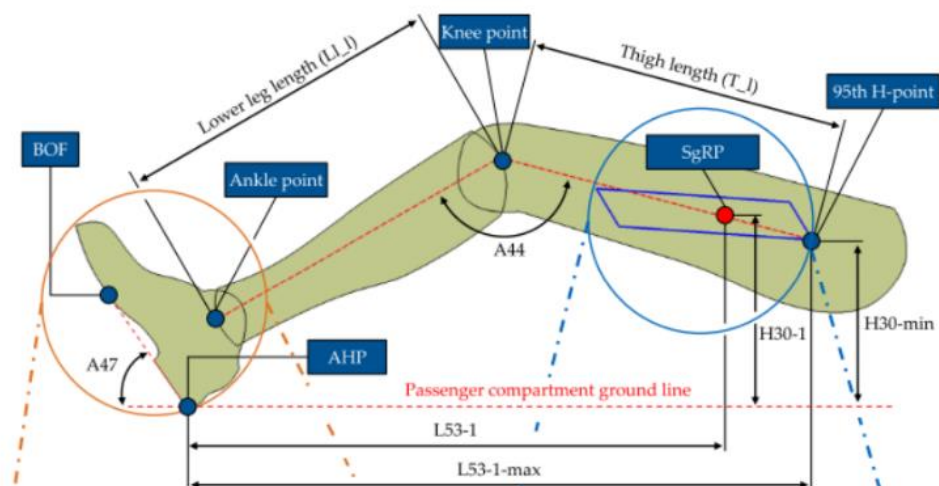


Figure 2.4: Leg dimensions [6]

On the x-axis the range of data is divided in different categories with equal width, each category is represented by a bin. The y-axis depicts the associated probabilities to pick a data in the considered category.

This type of visualization allows to graphically show a continuous set of data. In the range between the minimum and the maximum value of the sample there are infinite possible results. Dividing them into categories, they are discretized, and the infinite possible set is represented by a finite number of classes.

The following parameters give a quantitative numerical description of the data:

- Mean  $\mu$ : it is the most common measure of central tendency [17][18], calculated by the sum of all values of the data set divided by number of elements of the set of data.
- Median: it is also a measure of central tendency but defined differently. Values are sorted by their size and then divided into two equally sized groups. The value that is dividing the groups of data is the median.
- Standard deviation  $\sigma$ : it measures the dispersion of a set of data. A set of data with wide distribution has a higher value of the standard deviation. Another important characteristic are the probability intervals around the mean value defined by the standard deviation. In the function it is of particular interest the 95% interval characterized by  $1.96 \sigma$  [19, pp. 43-45]. Values outside this interval are called outliers [17].
- Coefficient of variation: A standardized measure of variation of the distribution of the probability density function. It can be interpreted as a unit independent deviation and is calculated by the standard deviation divided by the mean. The coefficient of variation has no unit and represents a percentage value. A low coefficient of variation indicates a low data dispersion [20]. The equation 2.2 shows the formulation of the coefficient.

$$c_v = \frac{\sigma}{\mu} [\%] \quad (2.2)$$

As mentioned above, the representation in categories is discretized, but it is useful to analyse the phenomena also through continuous distributions.

The normal distribution, or Gauss distribution, is a continuous distribution representing the probability density function associated with a specific data range. With this type of distribution, it is not possible to calculate the exact probability value associated with a given datum, but it is possible to derive the probability of a datum falling in a certain interval, integrating the function in that interval.

Moreover, the central limit theorem states that a process, which is the sum of many independent processes, tends to the normal distribution. The higher the number of processes and the closer the mean is to the average of the normal distribution [17][21].

Due to this extremely important statistical statement, it is often possible to use the normal distribution for a theoretical approximation of a random physical phenomenon.

The probability density function of the normal distribution in general form is presented in equation 2.3.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.3)$$

The probability density function is characterized by the specific  $x$  taken from the set of data, the mean value  $\mu$ , and the standard deviation  $\sigma$ . For the normal distribution, the mean and the median value coincide because the normal distribution is axis-symmetric with the mirror axis on the mean [22].

By performing a normal distribution analysis, two further statistical values can be obtained. The first one is the mean absolute error (MAE) as shown in equation 2.4.

$$MAE = \frac{\sum_{k=1}^n |y_k - \mu|}{n} \quad (2.4)$$

Where  $n$  is the number of data in the data set,  $y_k$  is the  $k$ -th resulting value from the normal distribution and  $\mu$  is the mean of the distribution.

The second one is the normalized value of the mean absolute error (nMAE) obtained dividing the previous value by the mean of the data set  $\bar{\mu}$ . By definition, this factor is dimensionless (equation 2.4).

$$nMAE = \frac{MAE}{\bar{\mu}} \quad (2.5)$$

The pre-processing tool performing the analysis of the set of data, carries out also statistical tests, with the goal of verifying the assumptions on the behaviour of the data set.

The hypothesis test has two evaluations: the null hypothesis and the alternative hypothesis which are one the opposite of the other [23].

A null hypothesis gives information about the possible correlation between two given sets of data. This hypothesis is rejected if the probability value, indicated by p-value, stays below an established critical level, called significance level  $\alpha$ . The typical values of  $\alpha$  are 1% or 5% [5].

If the null hypothesis is refused, it means that there is no statistical relation between the two considered data sets.

Another test that is performed is the Kolmogorov–Smirnov test, or KS test. It performs a comparison between two probability distributions or a probability distribution and a sample distribution (usually of greater interest). In many cases the KS test can be used to verify if a normal distribution fits a given set of data; in this case, the difference between the normal distribution and the sample distribution is computed for each value and the maximum of this differences is compared with the maximum allowed value according to a certain significance level  $\alpha$ . If the maximum exceeds the limit value, the results is the rejection of the similarity between the data set and the normal distribution. This means that the analysed data is not normally distributed.

The MATLAB function computes an approximate normal distribution of a given set of values and, if needed, generates a plot. On the same plot of the normal distribution, also

a histogram of the given data values is generated. Taking again the example for the A44 of the first row, the example of the complete plot is reported in Figure 2.5.

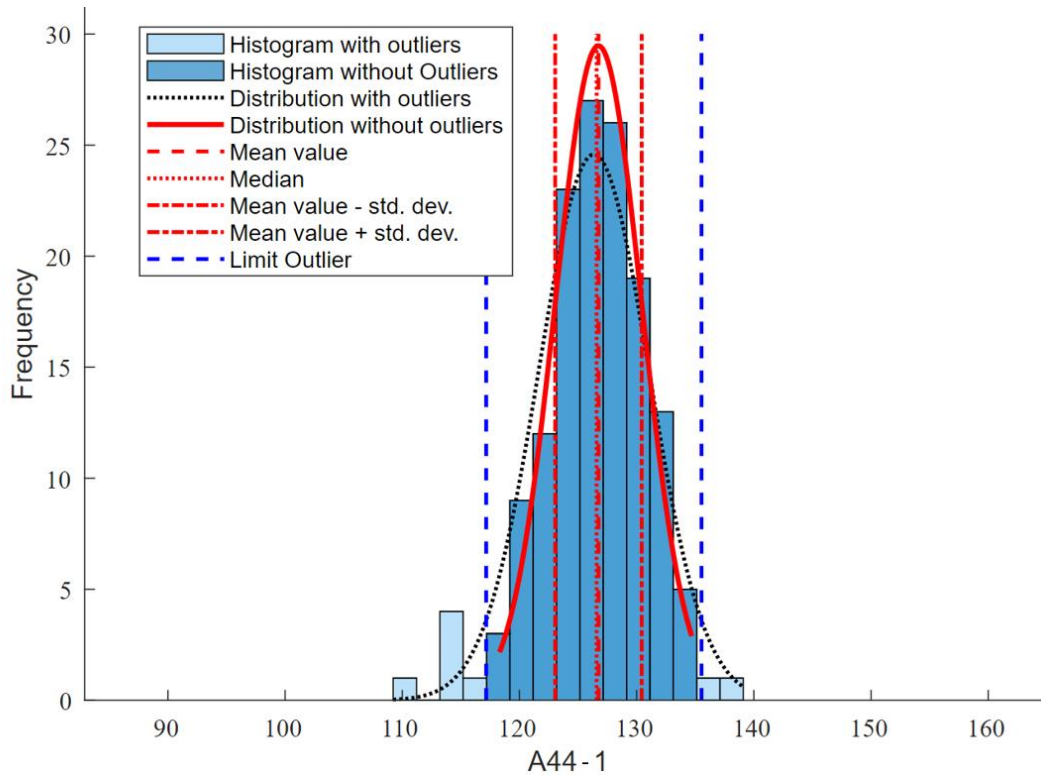


Figure 2.5: Histogram plot with normal distribution of A44 dimension of the first row

The data analysis with the normal distribution can be done including or excluding outliers.

The generation of the normal distribution is implemented in a MATLAB code that can generate all the discussed results and plots [18].

Moreover, it is also possible to generate an Excel file that contains the results of the normal distribution analysis and the plot. It is based on a fixed template that is directly filled from the MATLAB code. The main aspects of the file are reported in the Appendix A.

So, this function can make and store all the necessary evaluations of a given sample of data and it is widely used to retrieve some results during the pre-processing phase.

## 2.4 Linear regression analysis

Another way to obtain information from a data set is the computation of a linear regression model. Typically, this type of evaluation is applied to obtain a model capable of describing the correlation between different variables.

The objective of the regression analysis is to estimate the relation between a set of one or more independent variables  $x_{ij}$  and a dependent variable  $y_i$  and to statistically evaluate how robust this relation is [5]. To estimate the regression model, a set of already observed  $x_{ij}$  and  $y_i$  is needed. The terms indicate that  $x_{ij}$  is the  $i$ -th observation of the

j-th independent variable. This means that j represents the number of independent variables.  $y_i$  is the i-th response [24]. The i-th term states the specific value of the data set.

In the scope of this thesis only linear regression models are implemented in the tool. Linear regression means that the relation between the independent variables and the dependent one is a linear combination of the  $x_{ij}$  set. Nevertheless, since the database is built as a modular tool, it is possible to extend it to model also other regression models. For example, instead of using a linear equation, one can increase the order of the estimation. In this way the general regression becomes a polynomial.

Regarding the linear regression, it is possible to distinguish between two types: simple linear regression and multiple linear regression.

In the former, the model is a linear relationship between a dependent variable and an independent variable. The relation describing this type of model is given by equation 2.6.

$$\hat{y}_i = b_0 + b_1 x_{i1} \quad (2.6)$$

In the equation, the dependent variable is indicated with  $\hat{y}_i$ , since it represents an estimation of the real value  $y_i$ . The term  $b_0$  represents the intersection term while the parameters  $b_j$  ( $j \neq 0$ ) are called regression coefficients [17]. The data set from the database contains  $x_{ij}$  and  $y_i$ . They are used to get the model linear equation. After the computation,  $b_0$  and  $b_j$  are available and can be used to estimate new values of the dependent variable. The data set and the obtained model can be plotted on a cart, that in this case is a simple straight line that tries to fit the data. The example of the plot is reported in Figure 2.6.

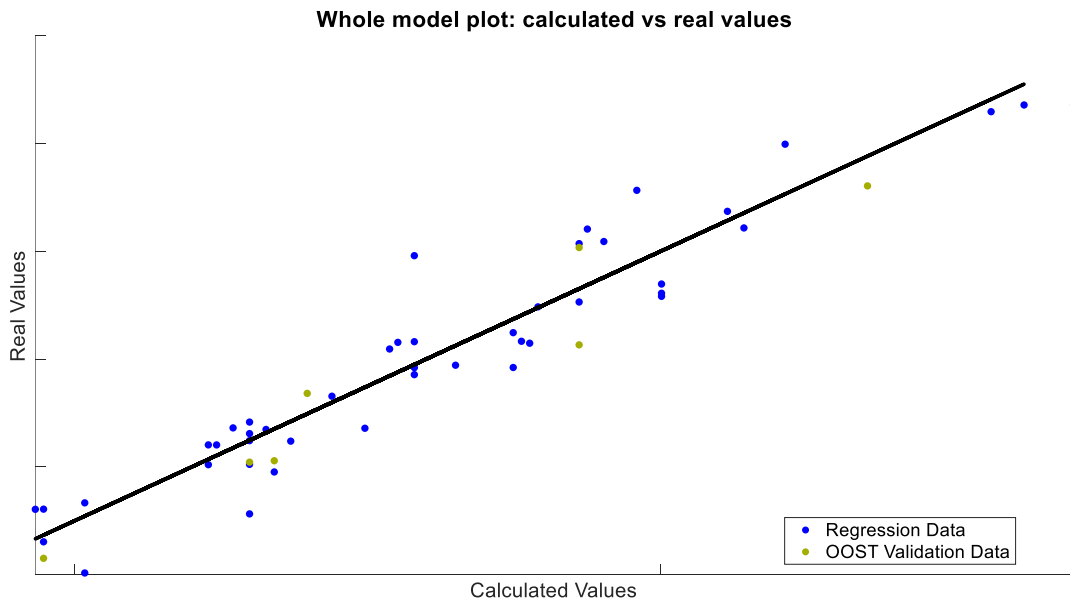


Figure 2.6: Calculated vs Real values

The example shows the whole model plot for the mass of the brake disc using the relation with the diameter of the brake disc. The real values on the axis cannot be shown due to the secrecy of data of the catalogue [25].

There is a more generalized type of linear regression with respect to the one of equation 2.6. It is the multiple linear regression. It involves a higher number of independent factors  $x_{ij}$ .

The relation describing this type of model is given by equation 2.7.

$$\hat{y}_i = b_0 + b_1x_{i1} + b_2x_{i2} + \dots + b_nx_{in} \quad (2.7)$$

To obtain the model formula an established approach is the method of least squares. It works minimizing the distance between the estimated value  $\hat{y}_i$  and the observation value  $y_i$ .

The distance is called residual  $\varepsilon_i$  and it is one of the most important statistical factors [26]. It is calculated as in equation 2.8.

$$\varepsilon_i = y_i - \hat{y}_i \quad (2.8)$$

By definition, the residual can be positive or negative. Being the regression line the average of the data set, the sum of all residuals is always 0 [26], no matter whether they are large or small.

Therefore, since the sum of the residuals cannot be used to derive the model, the regression is calculated by minimizing the quadratic value of the residuals as reported in equation 2.9.

$$\min \sum \varepsilon_i^2 = \min \sum (y_i - \hat{y}_i)^2 \quad (2.9)$$

Once the theoretical models have been introduced, it is important to explain how the MATLAB code for the regression function works and how it is used during the pre-processing phase.

As already mentioned, the model has to be obtained from an already present set of observation data containing both the values of  $x_{ij}$  and the values of the dependent variable  $y_i$ , practically taken from the database (in-depth analysis will be done in the following chapters).

In some cases, some of the data worsen the regression model. Without these values, the linear regression would be better, so these data has to be flagged as outliers and removed from the computation before the model is obtained. There is not an established method to evaluate outliers, but one of the most used is the calculation of Cook's distances [27].

The Cook's distance is evaluated comparing each model without the  $i$ -th value to the model computed with the whole data set and scaling them according to the mean square error (MSE, equation 2.14) and the number of data. The formulation is reported in equation 2.10 [28].

$$\text{Cook's distance} \rightarrow D_i = \frac{\sum_{j=1}^n (\hat{y}_s - \hat{y}_{s(j)})^2}{MSE \cdot n} \quad (2.10)$$

Where  $n$  is the total number of observations,  $\hat{y}_{s(j)}$  is the value of the predicted dependent variable with the model derived without observation  $j$ -th,  $\hat{y}_s$  is the value of the predicted dependent variable with the model derived on the whole set of data and MSE is the mean square error.

The result  $D_i$  is a vector with  $n$  elements containing the Cook's distance for each of the observed variables. If the value of Cook's distance is small, the impact on the regression model of the observation is small as well. On the other hand, a big value of  $D_i$  means a high impact of the observation. So, a critical value can be set to filter out the critical observations. A typical suggested value is around 3 to 5 times the mean of the Cook's distances [18]. Subsequently, the model is recomputed after filtering out the outliers.

Once the model has been computed, it is fundamental to calculate some statistical parameters that can give information about how well the model behaves related to the data sample.

### 2.4.1 Statistical evaluation coefficients

Using previous formulas, filtering outliers, and applying the least square method the regression model is obtained. It is important to be able to evaluate it. There are some statistical coefficients that are useful to quantify the quality of the computed model.

The most important factors are the following [29][30]:

- The Coefficient of determination  $R^2$  represents the percentage of accuracy with which the independent variables predict the dependent variable [30]. The explicit formula is given in equation 2.11.

$$R^2 = \frac{\sigma_{\hat{y}}^2}{\sigma_y^2} = \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (2.11)$$

Where  $\sigma_{\hat{y}}$  is the predicted variance,  $\sigma_y$  is the total variance,  $\hat{y}$  is the predicted value with the model,  $y_i$  is the observed value and  $\bar{y}$  is the mean value of the observations.

The perfect prediction gives  $R^2 = 100\%$  (predicted and observed data are equal).

- By increasing the number of independent variables to describe the estimation,  $R^2$  increases. Nevertheless, this does not give a real improvement of the model. To take the number of independent variables into account, the adjusted coefficient of determination is used [31]. It is formulated according to equation 2.12.

$$adj. R^2 = \hat{R}^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1} \quad (2.12)$$

Where  $n$  is the total number of observations (the data set dimension) and  $p$  is the number of independent variables.



- The mean absolute error (MAE) is the average absolute distance between the predicted value  $\hat{y}_i$  from the regression analysis and the observed value  $y_i$ , divided by the number of observations. For a perfect model, the MAE is zero. It is different with respect to the MAE computed for normal distribution. In this case, there is not the mean of the data set but the value of the  $j$ -th estimation. Moreover, the mean of the distributions is substituted by the real value of the data set. The equation 2.13 reports it.

$$MAE = \frac{\sum_{j=1}^n |y_j - \hat{y}_j|}{n} \quad (2.13)$$

- A normalized (dimensionless) version of the mean absolute error on the mean of the data set is used to deal with dimensionless quantity. It is expressed in equation 2.14.

$$nMAE = \frac{MAE}{\bar{y}} \quad (2.14)$$

- The mean square error (MSE) is calculated like the MAE but uses the squared errors, as reported in equation 2.15.

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (2.15)$$

- The root mean square error (RMSE), is obtained by taking the square root of the MSE, as in equation 2.16. Due to the square terms, this error is sensitive to outliers.

$$RMSE = \sqrt{MSE} \quad (2.16)$$

## 2.4.2 Further statistical tests

In some cases, two or more terms in the independent variable set depend on each other. This event is called multicollinearity and reduces the model accuracy. The variance inflation factor (VIF) is used to verify the presence of multicollinearities [32]. After eliminating the outliers, in the case of multiple regression models, a test for multicollinearity is required.

The computation is carried on by calculating a new regression model using all the independent variables except for one. This step is performed for every independent variable  $x_{ij}$ . To be clear, supposing a model with  $x_{i1}$ ,  $x_{i2}$  and  $x_{i3}$ , three different regressions are computed. One with  $x_{i2}$  and  $x_{i3}$ , one with  $x_{i1}$  and  $x_{i3}$ , and one with  $x_{i1}$  and  $x_{i2}$ . For each regression,  $R_j^2$  is computed. The VIF can be obtained using equation 2.17.

$$VIF_j = \frac{1}{1 - R_j^2} \quad (2.17)$$

ranging between 1 (no collinearity) to  $\infty$  (perfect collinearity). The test should give values of VIF below 10 to consider the regression model as acceptable [33].

The residuals  $\varepsilon_i$  of the regression model should be distributed according to a normal distribution [17][34].

The code can also plot the distribution of the residuals for a better investigation. An example is reported in Figure 2.7, for the same regression used in Figure 2.6.

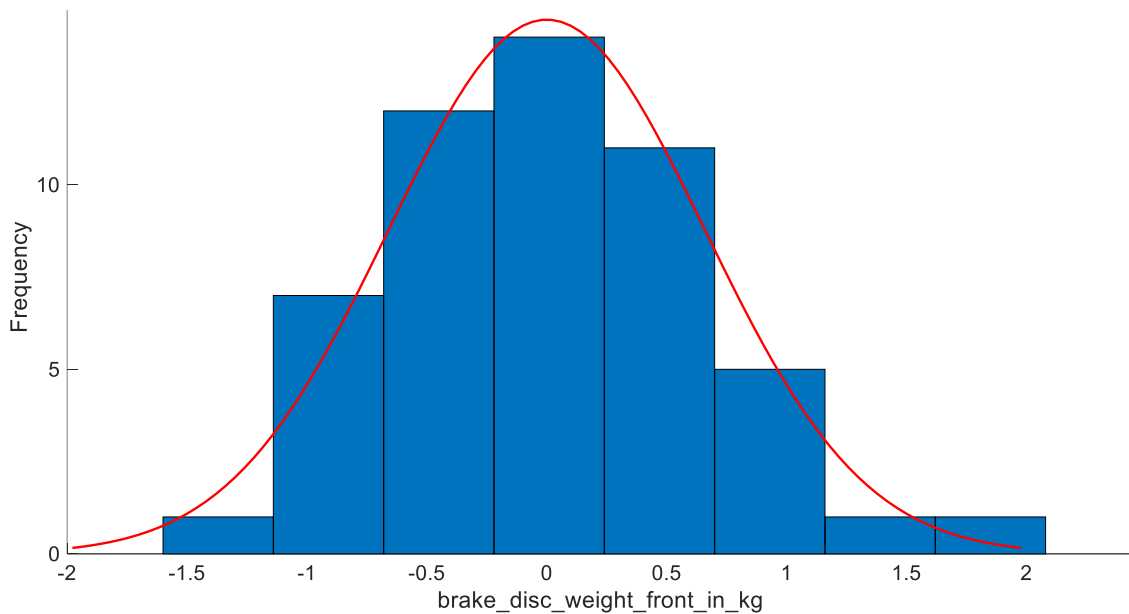


Figure 2.7: Distribution of residuals represented as bar plot

This is an important condition to execute the F-Test and the t-Test. They are tests used when comparing statistical models that have been fitted to a data set. They are useful to identify the model that best fits the population from which the data were taken [35]. The t-Test is used to test the hypothesis whether the given mean is significantly different from the sample mean or not, while F-test is used to compare the variances of two samples.

These two types of tests are hypothesis tests to evaluate the influence of the terms of the model equation on the dependent variable that are computed by obtaining the p-value and comparing it to the significance level  $\alpha$ . The p-value is the probability to obtain results of the test that exceed the limits of the observed sample [36]. A very small p-value means that an outcome would be very unlikely.

The F-Test is used to evaluate if the chosen set of independent variables is relevant (xi) in describing the dependent variable (y) behaviour. If the resulting p-value is below the significance level  $\alpha$ , usually 5% [37][17], the test is passed.

If the F-Test fails, it practically means that there is no term in the set of independent variables that can predict the estimation. If it happens, new different independent variables have to be chosen to obtain the regression model.

The t-Test is used to assess the impact that each single term of the regression equation has on the dependent variable. Also, in this case, the test is passed if the p-value keeps below the significance level  $\alpha$  of 5%, and it means that the considered term is relevant

for the model. If the test fails, the term is not significant for the regression and it can be removed from the computations.

Each time one or more terms, starting from the one with highest p-value, are removed from the model, the regression has to be recomputed and tests are performed again repeating this process until all terms falls below the significance level. At this point the final version of the regression model is obtained.

To clarify, the Figure 2.8 shows a schematized version of the described process.

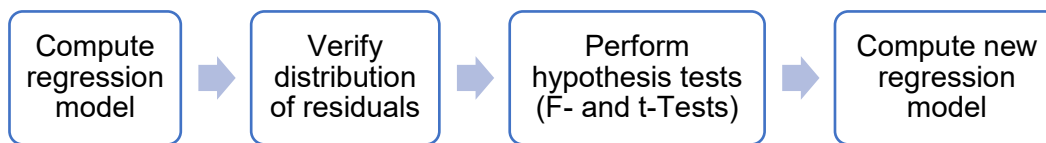


Figure 2.8: Flow of the process for linear regression models

Once this stage has been reached, it is important to perform a check on the model to verify that the relationship found between independent variables and the dependent variable is not due to a random coincidence caused by the data set. To do so the Out of Sample Test (OOST) is performed.

In this test the original data set on which the model has been computed, that has to be a minimum of 15 values, is iteratively divided in two parts containing approximatively the 15% and the 85% of the data respectively.

The smaller sample is called validation sample, the bigger one is the examination sample. The examination sample is used to compute and obtain the regression model while the validation sample is used to check the validity of the equation. For every iteration of the process the deviation between the model and the validation is computed using the normalized mean absolute error ( $nMAE_i$ ) and the result of the OOST is given by the average of all the calculated  $nMAE_i$ .

The MATLAB function used during the project performs all the needed computations and tests presented above, giving the user the possibility to get the models and also all the relevant statistical parameters useful for a correct analysis of the results.

As for the previous function of the normal distribution, if needed, the code can produce as results some plots of the relevant characteristics of the model and compile an Excel file to store all the results. The example of the file report is in the Appendix A.

The method explained in this chapter is bundled in a single MATLAB function. This function is able to obtain the proper regression model for a certain set of independent and dependent variables and is widely used in the tool.

### 2.5 Database theory

To implement the architecture features, different data sets need to be stored. The data sets are used to get fixed values and compute regression and distributions models. The data sets must be adequately structured to implement the computation feature in an automatic way.

To properly create a database structure, the used approach is the one reported in [38], following the creation of an Entity Relationship Model (ERM), a graphical approach to database design [39].

According to [2][38], the process of the ERM creation is divided in four steps:

1. Identify the entity sets and relations
2. Identify relationship types
3. Define the value of the sets and attributes
4. Organize the data and choose the primary keys

The following sections describe the single steps in detail.

#### 2.5.1 Identify the entity sets and relations

First of all, it is important to define an entity that, according to [2][40, p. 213], is an independent unit that can be identified in a unique way within the framework of a model. To be clearer, it is the specific name of a categorical set as for example the name of a manufacturer or a vehicle model.

All the information that describes a certain entity is saved in a specific entity set. The entity set can be for example a table. Each row of the table describes different entities with the related information. So, for example, the table called “manufacturer” is an entity set that contain information about car makers. Each row of the table corresponds to a different manufacturer, with its specific name and, if needed, some additional information as the address of the headquarter and the country.

Each element of the entity set, is a different entity. So, each row is uniquely identified by an identifier that is called “primary key”. The primary key can be a numerical or string field. The only requirement is that it is strictly associated to one single row. The primary key can be used to indicate the specific entity and, through that identifier, associate all other information of the entity.

To store all data in the database in a structured way, it is important to identify which are the relevant entities (indicated as rectangles in Figure 2.9 [2]). In the created database, entities are manufacturers, model series, models, dimensional concept, and all single components that need to be modelled by the tool. For each type of entity, a different set (table) must be stored. Moreover, not all entities have the same priority. Model series and model are the central tables. All other entities, in some way, are connected to the main entities, since each data set is associated to a specific model series or model [2].

### 2.5.2 Identify relationship types

As mentioned in the previous section, entity sets have relations between them. In the representation of Figure 2.9, relationships are indicated with rhombuses. They describe how many elements of a certain set are related to how many elements of another set. The use of relations is important to correlate data and avoid writing the same value in different entity sets.

There are three different types of relation:

1.  $1:n \rightarrow$  This type of relationship indicates that to one entity of an entity set, more different values of another table can be associated. For example, this is the category for the relation between the manufacturer and the model series. The same car maker produces more vehicle series that are associated to it. Another case is the relation between model series and model: different models belong to the same model series. For example, both models “159 1.8 TBi 16V” and “159 2.0 JTDM 16V” are part of the model series “159 (939) Limousine (03/08 - 11/11)”.
2.  $1:1 \rightarrow$  This type of relationship links one element of an entity set to another single element of a different table. An example is the link between the model series and the dimensional concept. The different model variations of a series (i.e. change of motor capacity) have no influence on the external dimensions or on the disposition of passenger and seats. So, for each model series, a single dimensional concept is associated.
3.  $n:m \rightarrow$  The last type of relation is a generic case in which a certain number of elements a table is associated to more fields of another entity set. It is the relation between model and battery. Taking the specific example of [2], Tesla Model S 60 and Tesla Model S 60D are two different models with the same battery, but they have a different number of motors [41], so a different topology of the powertrain.

A proper relationship management structures the database, gives no repetitions between entity sets and avoids data dispersion.

### 2.5.3 Define the value of the sets and attributes

Up to now, the ERM gave a definition of entities and corresponding sets. Inside each entity set (table), there are different fields with the information about the entity. The different fields stored in the table are called entity attributes. They are the single data of the table organized in columns. In Figure 2.9 they are represented with ellipses.

According to [2], data are divided in two main categories: attributes for the general vehicle and attributes for the components.

The general vehicle data are the ones saved in the manufacturer, model series, model, and dimensional concept tables. They contain general information at vehicle level, such as external dimensions and frame form for the model series, or weight and topology of

the powertrain for the models. The values of the dimensional concept, as for example the H30, height of the hip point, at SgRP (seating reference point) [8][7], are stored in the entity set for the dimensional concept.

Instead, components attributes are saved inside specific entity sets. So, for each element necessary for the computations of the models, there is a different table. For example, the entity set for the battery or for the headlights. As reported in section 2.5.2, each of this entity set is related to the main tables of the model series or model.

### 2.5.4 Organize the data and choose the primary keys

As already introduced in section 2.5.1, data stored in an entity set must be uniquely identified by a primary key. The single attributes used as primary keys are represented in Figure 2.9 as ellipses (because they are part of the data) with underlined text.

Taking the main reference table, model series, data is stored in a way that each model series has a specific name from [41], that is unique. No other line contains the same model series name. For this reason, the name of the model series can be a suitable primary key. The dimensional concept set can also use the model series name as primary key (since it has a 1:1 connection as reported in 2.5.2). Moreover, for the entity set of the models, if each row has a different model name, it can be used as primary key. Practically this is not suitable because the number of attributes is extremely large, and error can easily occur. To avoid problems, autoincrement is used for the primary keys. Whenever a new row is added in the set, the ID is automatically incremented. In this way, also for big tables, it is sure that no repetition occurs between rows.

For each entity set, the proper key must be selected evaluating every case. Specifically, it is important to check if attributes are model-series-dependent, model-dependent or none of them. In the first two cases, the entity set must be related to the main table using the reference primary key. The idea is to assign a key that takes the value of the primary key of the parent table. This type of key is the foreign key.

In case, no relation with another table is suitable, a dedicated primary key is set with autoincrement, as in the case of tables model series and model.

The general structure of the database according to the principle of ERM is reported in Figure 2.9 [2].

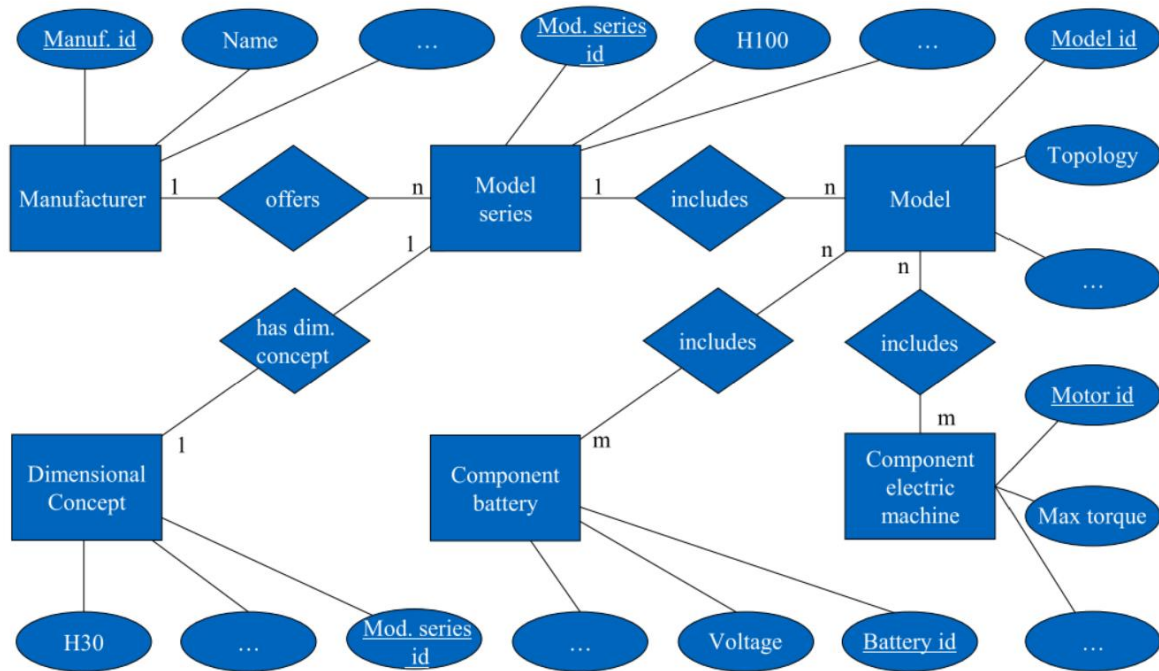


Figure 2.9: Structure of the database according to the ERM [2]

The presented structure and principles give the possibility to save all data for the empirical models for the pre-processing phase in a single database. Furthermore, the centralization of all data sets in a single database gives faster model update and reduction of dispersion. Moreover, all entity sets are related to the main tables of the model series and models. In this way, it is possible to merge columns of different entity sets to generate new tables. For example, one can create a new table containing information about the dimensional concept and data of a component, as the battery. This procedure is particularly useful for linear regression models since they often relate general vehicle data and specific component data.

With the presented structure, there is a clear division between the architecture features and the empirical models [2]. Thanks to that, the architecture entities can be updated adding new data to the database when new vehicles enter the market. The updating process is simple, it consists in adding rows and writing values as for normal calculation sheets (such as Excel). This makes the update process accessible also for people with no programming or database skills.

Easy update of the database avoids the problem of model aging, that with time makes calculated models no more suitable to real vehicles.





## 3 SQL and Database implementation

The database is crucial for the management of the models and for the proper actualisation during the pre-processing phase.

This chapter is dedicated to the deep analysis of the database structure (section 3.1), the explanation of the implemented models and tables (section 3.2), a brief overview of the main SQL language useful commands and solutions (section 3.3) and the explanation of the main practical activities on the software (section 3.4). Section 3.5 gives a brief explanation of the view feature. A final part (section 3.6) is dedicated to the discussion of some criticalities.

### 3.1 Database structure

The database has been built according to the structure theory presented in chapter 2.5. It contains several tables, that in turn store different data and have different tasks. Some of the most relevant aspects are the relations between tables.

In the current section all the relations and constitution of the structure is assessed. The specific content of the tables is explained in section 3.2.

The table in the database can be categorized based on their functionality:

- General data tables: Contain general information about vehicles' characteristics.
- Data tables: Contain data to generate the models.
- Calculation tables: Contain information for the MATLAB interface on how to interact with the database.
- Additional tables: Contain additional information.

If for some computations the wheelbase measures of the vehicles are needed to derive an empirical model, the tool knows where exactly take data. Moreover, if many computations require the wheelbase, having it in one table only, the reference to the wheelbase column is always the same, which avoids data dispersion. Finally, if a vehicle's wheelbase has to be added or changed, it can be edited in one place only, automatically updating all the computed empirical models.

The task is performed creating relations between tables so that data can be connected and retrieved from different locations. To build a connection between tables, as shown in section 2.5.2, a unique row identifier, the so called "primary key", is necessary for one of the entities involved in the relation.

The table that must be connected to the main one, usually the model series table, is a subordinated table. To understand the information stored in there, it is important to mention what a model series is.

A model series is a set of vehicles produced by the carmaker which share some characteristics and can be further divided in different models. A model, instead, is a precise variant of a vehicle in a model series. For example, it contains a variation of engine dimension or transmission topology.

A key is needed as well, but not as a unique identifier for the rows (more rows can have the same key), that refers to the primary key of the main table. This type of key is the already known “foreign key”. Creating the relation between tables, the only reference is made through the just mentioned keys without repetition of any other column. Taking again the example presented above for the wheelbase, there is a table that needs that value but does not contain it. It has a foreign key that refers to a primary key of another table (in this case the one containing the wheelbase values). Through that key, it can look for the wheelbase in the main table, associating it with the row containing the same key value. This can be done in the same way for all the tables that need wheelbase. It is important to notice that tables can only have one primary key, used as field identifier, while foreign keys can be more than one, referring to different primary keys of other tables. Software commands and procedures to set keys and get relations and connection between tables are explained in sections 3.3 and 3.4.

First, it is important to set a table, or some of them, that are the core of the database. These tables are important because they are used to set the main primary keys and contain the basic information to be associated with the computation tables.

Since in this specific case the tool is used for empirical models related to previous vehicles, the main tables for the database must be the ones containing all previous vehicle’s data. The core of this database is the model series table.

To give an overview of the structure, a graphical representation of the database is shown in Figure 3.1. It is different from the one of Figure 2.9 because it is devoted to classifying the various categories of tables.

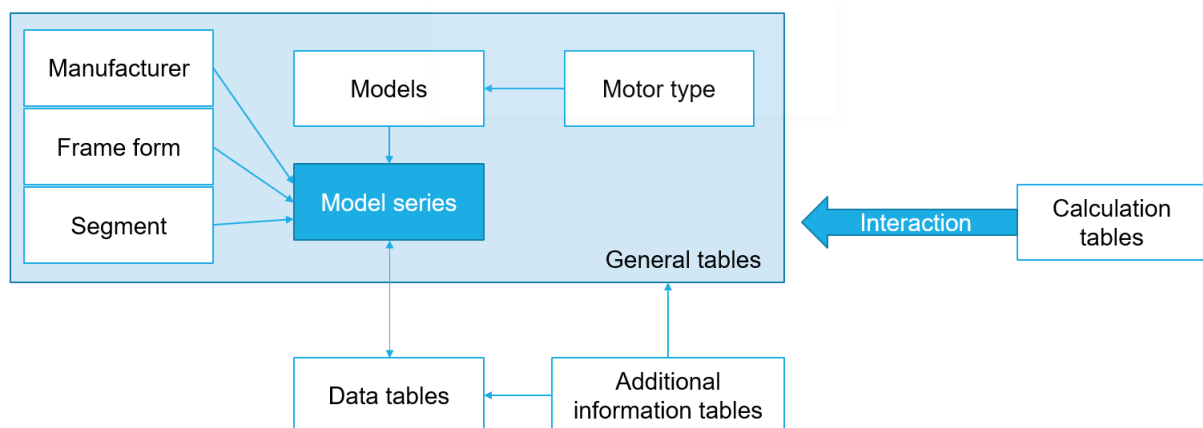


Figure 3.1: Structure of database tables

## 3.2 Data and models

As abovementioned, the current section is dedicated to the complete presentation and explanation of the database tables and their content.

A general set of guidelines has been followed to create the tables in the clearest possible way:

- The name of the tables is set with no uppercase (i.e. “model\_series”).
- The name of the columns is set with no uppercase (i.e. “manufacturer”).
- The only exception for the previous rule is the word “ID”, that is used to identify a key (i.e. “ID\_model\_series”).
- If the column is a primary key, the word “ID” is placed before the column name (i.e. “ID\_model\_series”).
- If the column is a foreign key, the word “ID” is placed after the column name (i.e. “manufacturer\_ID”).
- If the table or column name has more than one word, they are separated by underscore (“\_”), no spaces between them due to problems with SQL language (i.e. “number\_seats”).
- If the data has a unit of measurement, it is added at the end of the name (i.e. “wheelbase\_in\_mm”).
- If the table stores general vehicle information, the name is simply the content of the table (i.e. “dimensional\_concept”).
- If the table is a data table, the word “data” is placed before the table name (i.e. “data\_gearbox”).
- If the table is a calculation table, the word “calc” is placed before the table name (i.e. “calc\_masses”).

The used software for the database management, SQLite Studio (version 3.2.1), that will be presented more in detail in section 3.4, does not give the possibility to organize the tables in folders. To this purpose, it has been decided to add a word in front of the name of the tables that underlines their category. In this way, all the tables belonging to a certain type, appear all together due to the alphabetical order that is automatically used by the software.

### 3.2.1 General data tables

#### Model series table

The core table containing data on vehicle characteristic is the model series table. The model series table contains all variables that are model series-related, so that remain constant independently on the variant of model.

For example, general dimensions as width or length, are proper for a model series. Then there are different vehicles belonging to the same model series, with a variation of the engine type or topology. Every variant is a different model with its weight due to different engines, but all have same length that is related to the model series.

To build the model series table, and in general all tables containing the main vehicles' data, the reference has been the catalogue of ADAC [41], the Allgemeiner Deutscher Automobil-Club, the biggest automobile club of Europe. More precisely, all data of models and model series are taken with no distinction from ADAC, and they have been then divided in the ones that are model-dependent and the ones that are model series-dependent.

This table is the central entity of the database. To be consistent with the task of the project, so that to get models for the parameters that are actualized and significant for current vehicles, a filtering action has been performed, considering only vehicles produced after 2010. In this way, the models are computed on relatively new vehicles.

The model series table has the columns structure reported in Table 3.1.

Table 3.1: Structure of model series table

Column name	Example	Description
ID_model_series	41	Integer number that is the unique identifier for each row (primary key of the table). The column "model_series", containing the complete name, can be also used as primary key, but using a numerical identifier is more intuitive and robust.
manufacturer_ID	3	Integer number that is used as a foreign key with reference to another table, the "manufacturer" table, containing the list of all the manufacturers
model_series	Giulietta (940) (05/10- 04/16)	The name of the model series as reported on the ADAC catalogue. The name is a string, and it includes both the whole name of the series and the start and end dates (only the start date if the model series is still in progress).
model_series_start	01-May- 2010	The date of the beginning of the model series. It is in the format of dd-Mmm-yyyy where the month is indicated by the thirt three letters of the word.
model_series_end	01-Apr- 2016	The date of the end of the model series. It is in the format of dd-Mmm-yyyy where the month is indicated by the thirt three letters of the word. It is not always present in case the model series is still in progress.
length_in_mm	4643	The length of the vehicle expressed in mm.
width_in_mm	1860	The width of the vehicle expressed in mm.

height_in_mm	1436	The height of the vehicle from the ground expressed in mm.
ground_clearance_in_mm	-	The free distance from ground (ground clearance) of the vehicle expressed in mm.
wheel-base_in_mm	2820	The wheelbase of the vehicle expressed in mm.
turning_circle_in_m	10.8	The value of the minimum turning circle of the vehicle expressed in m.
frame_form_ID	12	Integer number that is used as a foreign key with reference to another table, the "frame_form" table, containing the list of all the possible frame forms of the vehicles.
segment_ID	4	Integer number that is used as a foreign key with reference to another table, the "segment" table, containing the list of all the possible segments to categorize a vehicle.
number_seats	5	Integer number representing the seats.
spring_front	Helical	Type of primary spring mounted on the front suspension (helical or air spring).
spring_rear	Helical	Type of primary spring mounted on the rear suspension (helical or air spring).
brake_front	Disc	Type of braking system mounted on the front wheels (disc or drum brakes).
brake_rear	Disc	Type of braking system mounted on the rear wheels (disc or drum brakes).
source_ID	16	Integer number that is used as a foreign key with reference to another table, the "source" table, that is an additional information table, containing the list of the sources from which data have been retrieved

Whenever there is the word "ID" in the column name, that column is a key. It is clear from the table that there is one only primary key, selected as the unique number for the model series, but in the same table there can be more than one foreign key that refer to the primary key of other tables. Some relevant categorical variables have been selected to be foreign keys. The idea to use numerical values instead of the strings is a more robust solution to avoid mistakes (as explained in section 2.5.4): the user must select the number associated to the word, avoiding typing the word in the wrong way. Moreover, dealing with numbers is easier and more intuitive when working with SQL queries.

The related tables of categorical variables, in this case frame form, segment and manufacturer, are composed by two columns. The first one containing the primary key and the second one containing the corresponding categorical value. More columns can be added (i.e. for the table of manufacturers additional information about the producer can be introduced as for example the location of the headquarter).

An example of categorical table is reported in Table 3.2, with reference to the table of segments.

Table 3.2: Segment table

ID_segment	segment
1	mini_cars
2	small_cars
3	micro_cars
4	medium_cars
5	large_cars
6	executive_cars
7	small_medium_cars

## Model table

Another table containing information about vehicle's general characteristics is the model table. The table contains different models, that are specific variations of a model series. All models of a certain model series have some common features, that are peculiar of the model series itself, as for example the external dimensions. The model table contains all the relevant data that are proper of a certain model.

The data have been collected from the ADAC catalogue [41].

The model table has the columns structure reported in Table 3.3.

Table 3.3: Structure of model table

Column name	Example	Description
model_series_ID	41	Integer number that is used as a foreign key with reference to the "model_series" table presented in the previous section. This is the key that let the model to be linked to a precise model series (being the model a specific variation of a model series).
model	Giulietta 1.4 TB 16V	The specific name of the model. This table has no primary key because the connection with data is always created through the model series, but this column could be also used as primary key
model_type	Turismo	It is still part of the model definition, defining a specific type or variation of the model (i.e. it often contains the specific setup of the vehicle, for example "business", or the type of traction distribution or propulsion).
model_start	01-May-2010	The date of the beginning of production of the model. It is in the format of dd-Mmm-yyyy where the month is indicated by the thirist three letters of the word. This information is required since not always the model life corresponds to the model series start.
model_end	01-May-2011	The date of the end of production of the model. It is in the format of dd-Mmm-yyyy where the month is indicated by the thirist three letters of the word.

price	21700	The starting price of the specific model. It is expressed in euro (€).
motor_type_ID	5	Integer number that is used as a foreign key with reference to another table, the "motor_type" table, containing the list of all the possible propulsion systems that can be employed.
volume_trunk_in_l	350	The volume of the luggage compartment measured in normal use conditions (the minimum available volume). It is expressed in litres.
volume_trunk_at_window_folded_seat_in_l	-	The volume of the luggage compartment measured with folded seat up to the belt line. It is expressed in litres. This value is often not available in the catalogue because only the maximum value to the roof is usually measured.
volume_trunk_at_roof_folded_seat_in_l	1045	The maximum volume of the luggage compartment measured with folded seat up to the roof (using all the available space). It is expressed in litres.
weight_eu_in_kg	1355	The weight of the vehicle measured with neither passengers nor baggage. It is measured in kg.
max_weight_in_kg	1785	The maximum weight that can be reached by the vehicle. It is measured in kg.
payload_in_kg	430	The variation of weight between the maximum and minimum condition. It is therefore measured in kg.
number_seats_max	5	The integer number representing the maximum number of seats that can be mounted. It is slightly different from the one in the model series table because for some models there can be some configurations with additional seats.
tire_type	205/55R16	The standard tire that can be mounted on the model. It is expressed with a string with the standard form of the tire codes.
source_ID	16	Integer number that is used as a foreign key with reference to another table, the "source" table, that is an additional information table, containing the list of the sources from which data have been retrieved.

In this table also, a categorical variable is present. It is the case of the motor type that is typical of the model characteristic. Again, a numerical value has been chosen with reference to an external table containing the list of motor types. It is reported in the Table 3.4.

Table 3.4: Motor type table

ID_motor_type	Motor_type
1	Diesel
2	Full electric
3	Gas
4	Hybrid
5	Gasoline
6	Plug-In-Hybrid

#### Dimensional concept table

The content of the dimensional concept table is not taken from the ADAC catalogue. It makes reference to the A2Mac1 benchmarking catalogue [25]. It contains benchmarking data across different models on every part and component of the vehicles. Moreover, the catalogue gives the possibility to get 3D models of parts and modules and pictures of real components. The access to the portal is granted to authorized users only.

This table contains a reduced selection of vehicles with respect to the ones presented in section 3.2.1. For each vehicle, the complete set of dimensions of the exterior and of the internal passenger compartment is reported.

The data contained in this table are standard dimensions according to SAE norms [8]. They are useful for the computation of regression models, especially dealing with body dimension and the passenger compartment.

The complete list of columns is not reported being extremely large. Anyway, It The set contains also relevant angles both for the internal configuration and the external. Typical example is the definition of the passenger compartment and manikin dimensions as reported in Figure 3.2 [6].



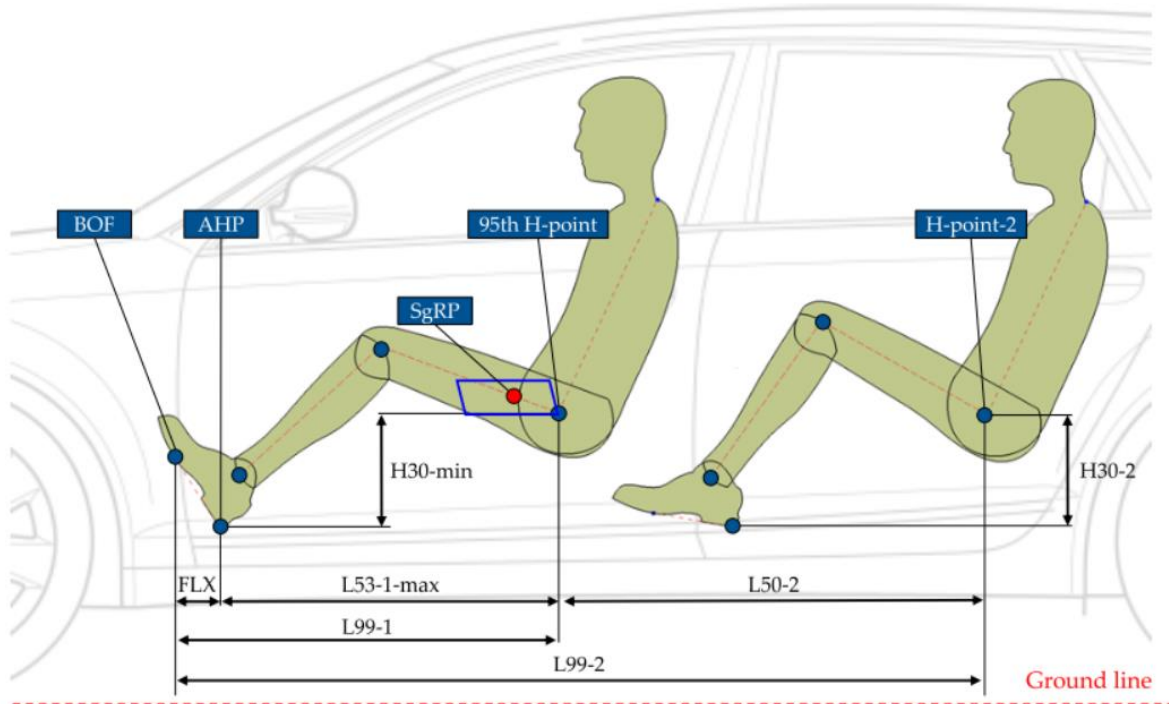


Figure 3.2: Example of internal manikin dimensions [6]

In many cases, some of the variables are also used to compute regression models.

### 3.2.2 Data tables

The previously presented tables were the general data tables with information about the vehicle that can be used both for empirical model computation and simply to store past series information.

There is another category of tables that have been implemented in the database that are the tables specifically containing data for the computation of models. These are the data tables.

Those tables are specifically defined for different vehicle's components and used by the tool to get necessary models according to regression computations or normal distribution (discussed in Chapter 2).

This tables, with the carry-on of the project, will increase because new models will be added to the tool.

Data table can be used to get constant values (from a normal distribution analysis) or regression models

For the data tables used to compute normal distribution models, the example is the table for the mass of the driver airbag. The specific name, according to the nomenclature conventions, is "data\_airbag\_driver". The name, after the designation "data\_" contains the specific element of the vehicle. The structure is reported in Table 3.5.

In the following tables, the reported data are not the real values. They are simply a representation of suitable numbers. Real specific values are internal and cannot be shared in the current work.

Table 3.5: Example data\_airbag\_driver table

model_series_ID	segment	market	driver_airbag_weight_in_kg
123	D	Europe	1.250

The general structure contains the reference to the main table. It is the foreign key with reference to the model series. It is always the first column of the entity set. In this way airbag information are strictly related to a specific vehicle with its characteristics. There can be some additional information (segment and market in this case), but what is important is the value of the mass of the component. All the list of masses is the data set for the computation of the normal distribution.

Different case for tables used to compute the regression models. The example of Table 3.6 reports the data table for the empirical mass model computation of the tires.

Table 3.6: Example data\_tires table

model_series_ID	tire_diameter_in_mm	tire_width_in_mm	tyre_sidewall_in_mm	tyre_weight_in_kg
123	740	255	100	15.5

As in the previous case, the first column refers to the model series. It contains the reference to the ID to correlate the tire information to a specific model series.

There must be a column containing the dependent variable for the regression. Since the table is useful to compute the weight of the tire, the estimation column is the one containing the mass. There are then additional columns that can be used as independent variables. In some cases the same table contains more columns that are used for the computation of different regression models.

Moreover, there is the possibility to have are tables with only one column, that are anyway used to get regression models. This is the case in which the independent variables are taken from another table, using the connections through keys. It is the example of the regression for the computation of the mass of front driver door panel. The Table 3.7 reports it.

Table 3.7: Example data\_door\_panel\_driver table

model_series_ID	segment	market	door_complete_weight_in_kg
123	D	Europe	32.250

There is only one value, the dependent variable, because the independent ones are the height and the wheelbase of the vehicle. In this situation, the model\_series\_ID value is used to read the corresponding needed dimensions in the table of the model series.

## Catalogues

In some cases, the tool needs component catalogues to do its computations. In those conditions, the whole catalogue table is needed, without calculating distributions or regression models.

Some catalogues are implemented in the current version: for tires and for different types of bearings. It depends on the catalogue, but generally the table contains all information about technical properties and codes.

Inside the database catalogues are saved, indicating them with the word “ctlg” before the component name (i.e. “ctlg\_ball\_bearings”).

They are useful because, for example, the main tool computes the loads on the bearings of a certain component and then selects the proper bearing that can sustain that load.

They are not included in the list of data tables because the code makes no calculations on them.

### 3.2.3 Additional tables

The source table belongs to the category of additional tables. It has no values for vehicle measurements, and it is not used for the pre-processing phase, but it is crucial for the completeness of the database.

It contains the list of the references from which data stored in the database have been obtained. In every table, one or more columns referring to the “ID\_source” primary key are present so that data origin can be retrieved.

The structure with an example is reported in the Table 3.8.

Table 3.8: Structure of source table

Column name	Example	Description
ID_source	16	The numerical ID representing the primary key of the table. It is used to create the reference in the data sets. It is automatically generated by the table adding new elements.
author	ADAC	The name of the people or agency that wrote the document.
fullname_title	Allgemeiner Deutscher Automobil-Club Catalogue	The complete title of the work. It can be the name of the web page or the title of an article, book, norm...
document_type	Internet page	The type of document of reference.
place_of_publication	-	The place in which the document has its publication. It can be missing, especially for internet pages or normative.
year	-	The year in which the document has been published. It can be missing, especially for internet pages.
link	<a href="https://www.adac.de/rund-ums-fahrzeug/auto-katalog/marken-modelle/">https://www.adac.de/rund-ums-fahrzeug/auto-katalog/marken-modelle/</a>	The link to the page or document. It can also be missing in case there is not a direct online link, especially in case of books.

#### 3.2.4 Calculation tables

Last category of tables is the so-called set of calculation tables.

This tables build the MATLAB interface. They do not contain information about vehicles, but rather information on how to derive the empirical models from the data

Using those tables, the MATLAB code can generate the proper queries and derive the constant values and regression models

As already mentioned, four different types of data or models can be obtained: fixed parameters, catalogues, normal distributions, and linear regression. For each one of these categories, a different query structure (and therefore a different calculation table) is necessary. To distinguish between the categories, a special word is added at the end of the name:

- “\_fix” for the calculation table of fixed parameters.
- “\_catalogue” for the calculation table of catalogues.
- “\_norm” for the calculation table to get normal distributions.
- “\_regression” for the calculation of the linear regression models.

## Calculate fixed parameters and catalogues

Different approach has been used for regression models. The regression studies relations between different measures. For this reason it is difficult to categorize them according to components or modules. Anyway, the MATLAB interface has been designed considering the possibility to change this approach also dividing regressions for modules or components (the detail of MATLAB side will be investigated in chapter 4). Specifically, it means that in the current version there is only one calculation table containing all the regressions. If needed, more calculation regression tables can be introduced with no variation in the MATLAB code (for example dividing them according to regression for masses, dimensions...).

The table for the fixed parameters, has a simple structure. In the first column, as also for all other categories, there is the name of the variable in the MATLAB code. The code can read it and to create a workspace variable with the prescribed name. A second column contains the value to be assigned to the variable.. There are other columns in which additional information for the user is contained to clarify the meaning of the variable. Specifically, there is the unit of measurement of the measure, a brief description of the parameter and two final columns with a reference to the source table (shown in section 3.2.6).

An example of the structure of the fixed parameters table is reported in Table 3.9.

Table 3.9: Structure of calculation table for fixed parameters

Column name	Example	Description
parameter-name	Parameters.masses.weight_max_permitted	The name of the structure path in which to save the parameter.
value	3500	The value of the parameter.
unit	kg	The unit of measure of the dimension. It is given for the user clarity.
description	Maximum weight permitted for a vehicle	Brief description of the parameter
source_ID	25 [15]	Integer number that is used as a foreign key with reference to another table, the "source" table, that is an additional information table, containing the list of the sources from which data have been retrieved.

The MATLAB interface follows simple instructions (they will be presented in chapter 4.2.3) and can read the value of the table and assign it to the corresponding variable. In this way the fixed parameter is saved in the Parameters structure and used as input for the tool.

An update of fixed values can be easily made in the database (for example due to change of regulation) overwriting the previous value or renaming the MATLAB name variable.

Similar discussion regards the catalogues. In the database, the catalogues are saved in tables and they must be saved in the way they appear. The first column contains the name of the variable in which to save the catalogue. Since the whole table must be stored, only table name is needed, and it is given in the second column. A final column is present: it contains a brief description of the catalogue for a better comprehension for user.

The structure of the calculation table for catalogues is reported in Table 3.10.

Table 3.10: Structure of calculation table for catalogues

Column name	Example	Description
parametername	Parameters.catalogue.tires	The name of the structure path in which to save the catalogue.
tablename	ctlg_tires	The name of the catalogue to be saved.
description	Catalogue of the tires	A brief description of the catalogue.

## Calculate normal distributions

For normal distributions, the pre-processing tool needs to get a precise data set on which to compute the distribution model, and not a single value as it is the case for the fix parameters. The structure for the normal distribution table is reported in the Table 3.11.

Table 3.11: Structure of calculation table for normal distributions

Column name	Example	Description
parametername	Parameters.dimensions.CY.axis_length_rear.torsion_beam	The name of the structure path in which to save the constant values obtained with the normal distribution.
variablename	axle_length_in_mm	The name of the column containing the data set.
tablename	rear_axle	The name of the table that contains the previous column.
table_category_1	rear_axle	The name of the table containing the column for categorization.
column_category_1	axle_type	The name of the column used for the categorization.
filter_type_ID_1	Consider only	The type of filter that code must apply.
category_1	torsion_beam	The value(s) used for the filtering.

The first column contains the name of the parameters variable to be used in the MATLAB code. In this variable, the mean value of the normal distribution must be saved. In some exceptional cases, for a more complete analysis, more fields are added to better evaluate the statistical properties (it will be discussed in chapter 4.2.4). The second and third columns contain information on where to take data from the database, so specifically the name of the column and table in which necessary data are contained. Theoretically, this

is sufficient to get a column of values to be used as input for the statistical tool of the normal distribution. But, in many conditions, not all data from a column need to be taken. For this reason, a set of additional fields is used to impose certain conditions on how to retrieve data. For example, one can be interested in computing the mean value of the wheelbase of all the SUVs of the database. In this case, the query needs to get wheelbase value only for the frame form "SUV". Basically, this part is used to create a query condition to get only a certain selection of data from the specified column. Query conditional parts are explained in the next section (3.3).

The conditional columns are 4 for each filter. The first two specify the table and the column in which the variable to be used for the filter is stored. The third one contains a string explaining which type of filter is applied. On the base of that, the MATLAB code can perform different tasks and select the needed data. The type of filters implemented up to now are:

- "Filter out": used to get all data of a certain column, excluding one or some categories (i.e. considering all values but not for "Vans" and "Motorhomes").
- "Consider only": it is the opposite of the one above. It is used to get only data which fulfil one or some specified categories (i.e. considering only values of "Sedans").
- "Higher": used for non-categorical values. As the name suggests, it retrieves only data on which the specified column has a value greater than the selected one.
- "Lower": used for non-categorical values. As the name suggests, it retrieves only data on which the specified column has a value smaller than the selected one.

The last column for the filters contains the real value on which to apply the condition. Every group of categorization table contains a suffix number to identify the filter.

## Calculate regressions

The last table for calculations is the one for regressions. As already mentioned, there is a single table containing all regressions. Due to dependency on many variables, belonging to different categories, a categorization is more difficult, avoiding the possibility to create many regression tables. Anyway, if needed, one can divide them. The tool interface works independently on the number of tables.

The structure example of the regression calculation table is reported in Table 3.12.

### 3 - SQL and Database implementation

Table 3.12: Structure of calculation table for regression models

Column name	Example	Description
parametername	Parameters.regr.turning_circle	The name of the structure path in which to save the regression model.
independent_var	wheelbase_in_mm; width_in_mm	The name of the column containing the data sets of the independent variables. If more than one, they are separated by “;”.
dependent_var	turning_circle_in_m	The name of the column containing the data set of the dependent variable.
tablename	model_series	The name of the table containing the previous columns. If more than one, they are separated by “;”.
description	Turning circle as function of wheelbase and width	A brief description of the model.
table_category_1	frame_form	The name of the table containing the column for categorization.
column_category_1	frame_form	The name of the column used for the categorization.
filter_type_ID_1	Filter out	The type of filter that code must apply.
category_1	Bus; Motorhome	The value(s) used for the filtering.

The first column, as in previous cases, contains the name of the MATLAB variable in which to save the regression model. The structure path is like the previous ones, it only contains the word “regr” that specifies the model belongs to regression tab. In this way in the output variable structure all regressions are collected inside this folder.

Computing linear regressions, one or more independent variables, and a dependent one, are needed. Immediate solution is the creation of a column for each independent variable and for the dependent one, but regressions can have different number of independent variables. For example if three columns are needed for a regression, the empirical model with only one will have two empty columns. Moreover, if a new regression computation needs four independent variables, a new column needs to be added. This solution is neither versatile nor user-friendly. Some knowledge of the database and SQL is needed and adding new columns is time consuming.

To avoid this problem, the table has only two columns for the model variables: one for independent variables and one for the response. In this way the number of independencies is not relevant. The user simply needs to write them in the second column, separating the name with “;”. The MATLAB code can read the field and automatically retrieve the different independent variables through the separator.

The next column contains the name of the tables in which to retrieve data. Since more columns must be taken, more tables can be needed. The solution is like the one used for independent variable. If there are more tables, they are all written in this field with a “;” separator. The MATLAB tool can build a particular piece of query to link all tables together (it will be presented in chapter 4).



In the following field, there is a space to add a brief description of the regression model. It is necessary just to clarify it to the user.

After this part, more columns are present, they are needed to add the possibility to apply filters. They contain table and column on which look for the filter, the type of filter is implemented and the value (or values if more than one, separated by “;”) to be used as category. This block of four columns can be repeated to add more filters. No further detail is explored because the structure of categorization is the same as the one presented for normal distributions.

### 3.3 SQL language

To make operations in the database environment, SQL language is needed. The SQL, or completely Structured Query Language [42], is a standard language to program and manage data contained in a RDBMS (relational database management system). This type of language is useful to store, access, and modify data in a way that is organized and efficient [43]. The database was created using the platform SQLiteStudio in the version 3.2.1 [44]. The software offers an interface to control all operations without the need to use the query language. Nevertheless, in many cases, the code editor is needed to implement complex procedures. For example, to properly select a set of values to be updated according to a filtering condition, the use of the query editor is faster than the use of the manual interface. To learn the SQL language, the complete course of w3school [45] has been followed.

The most relevant query structures used to implement the MATLAB interface are now analysed. Before starting with the discussion, it is important to underline that commands in this text (and in books and online courses) are always written in capital letters to be clearer. Anyway, SQL language is not case sensitive, meaning that no difference is made if letters are capital or not in any position of the query.

- `SELECT column(s) FROM table`

The SELECT command is used to get a certain set of data from a table. As the structure suggests, the command let the user select one or more columns of a certain table. If more than one column is needed, as for the case of the regression variables, their name must be separated by “,” (not “;” that is used as statement ending). In the table field only one value is accepted. If data comes from many different tables, the command JOIN must be used.

An example can be “SELECT width\_in\_mm, height\_in\_mm FROM model\_series”.

If the complete table needs to be taken, it is not necessary to write the list of all columns, but a simple “\*” in the columns space (i.e. “SELECT \* FROM model\_series”).

- `SELECT MAX/MIN(column) FROM table`

A particular case of SELECT sentence, is made when the maximum or minimum of a dataset is needed. The current command retrieves a single value that is the maximum or minimum of the selected column. The word "MAX" ("MIN") is used before the column name written between parenthesis and it simply gets the maximum (minimum) value of that data set.

- INNER JOIN *table2* ON *table1.key1=table2.key2*

The JOIN command is quite articulated, but it is crucial for the tables' connection. There are four types of join command: INNER, LEFT, RIGHT and FULL JOIN. In the current implementation, only INNER JOIN is used. They can be thought in a graphical way with Eulero-Venn's figures as reported in the Figure 3.3 [46].

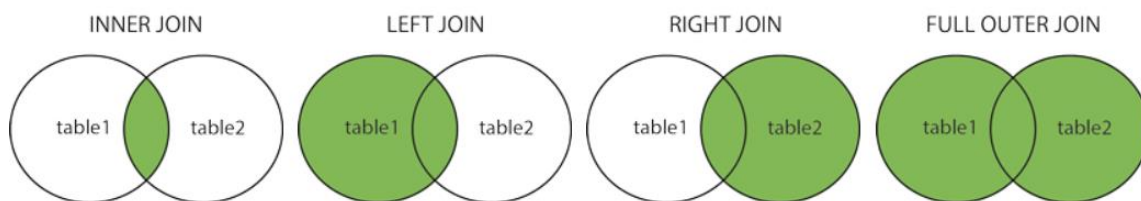


Figure 3.3: Type of joins [46]

Since in the models, intersections are needed, the relevant command is the INNER JOIN. Retrieving data of the same vehicle from different entities, only the one that are present in both tables must be considered. The word "INNER" can be omitted, simply writing JOIN refers to this type of connection.

As visible in the structure, a key matching is needed to get the connection between data. It is in this case that primary and foreign keys become crucial because they are the link between tables on which to get the intersection.

Making a practical example, supposing to take the width of the vehicles (present in model series table) to get the model for the headlights mass (present in data headlights table) the query results in:

```
"SELECT width_in_mm, headlights_weight_in_kg FROM model_series INNER
JOIN data_headlights_halogen ON
model_series.ID_model_series=data_headlights_halogen.model_series_ID".
```

In this way the connection is created. The sentence let the code get the data of vehicle width for the vehicles that are also used for the calculation of headlights. This command is the key to avoid repeating columns in different tables improving robustness and reducing error propagation.

- WHERE *condition*

The WHERE clause is used to set conditions. In some cases, not all data of a certain column need to be taken, so a filtering action is necessary. The condition after the where is used to apply this control.

The condition can be related to values, so using comparison operators (“=”, “!=”, “<” and “>”) or strings, using the “LIKE” or “NOT LIKE” commands. For example, one can get the value of the vehicle width for models with a wheel-base higher than 2800 mm using the following query:

```
“SELECT width_in_mm FROM model_series WHERE wheel-  
base_in_mm>2800”.
```

Or it is possible to select the width of vehicles that are sedans using:

```
“SELECT width_in_mm FROM model_series WHERE frame_form LIKE  
‘sedan’”.
```

In many versions of SQL database interface, the command “=” also works for strings in the place of “LIKE”.

In case more than one filter needs to be applied, more conditions are added after the WHERE statement using common logic operators (“AND” or “OR”). If width of vehicle of sedans with wheelbase higher than 2800 mm is needed the query becomes:

```
“SELECT width_in_mm FROM model_series WHERE wheel-  
base_in_mm>2800 AND frame_form LIKE ‘sedan’”.
```

Finally, the categorization made with this part must be written after the JOIN section if there are more tables, because data needs to be linked through the key first.

- **UPDATE** *table* SET *column=new value* WHERE *column=old value*

This part is differently used with respect to the previous commands. The former structures are used to get data from the database. The actual command instead, is used to modify some fields of a table.

If single values must be changed, a manual action on the involved table is easier and faster, while if many fields with a common condition needs to be updated, it can be automatically done.

The query lets the user set a certain column to a defined new value where this column satisfies a precise condition. An example easily explains the importance of this command. For a better working of the MATLAB interface in model computation, NaN (not a number) values needs to be substituted with NULL values. It can be easily done with the following query statement:

```
“UPDATE data_brakes SET brake_pad_weight_in_kg = NULL where  
brake_pad_weight_in_kg = ‘NaN’”.
```

The only problem related to that command is that only one column at a time can be modified. For big tables, it is still time consuming to apply the previous phrase for each column (some tables can have hundreds of columns). To this purpose, a further automatization is made with a dedicated MATLAB code that can cyclically apply the command to a predefined table.

## 3.4 Creation and management of the database

This section is dedicated to show the use of the SQLiteStudio interface. Specifically, the main scope of the interface is creating tables, managing table structure, and updating single fields. All other operations regarding computations and models are automatically performed through queries and the MATLAB tool.

The interface is structured in a simple way (Figure 3.4). On the left side of the screen, once a database has been opened or created, there is the complete list of the stored tables (blue rectangle, Figure 3.4).

The central panel represents the current opened entity (red rectangle, Figure 3.4), it can be the data of a tables or a table structure with the list of its columns or the panel to write and run queries. A bottom panel called “status” (green rectangle, Figure 3.4) gives instant feedbacks for every operation the user performs.

Finally, on the lowest part there is the list of opened tabs (black rectangle, ), containing every table or SQL query editor has been opened.

The complete interface is reported in Figure 3.4.

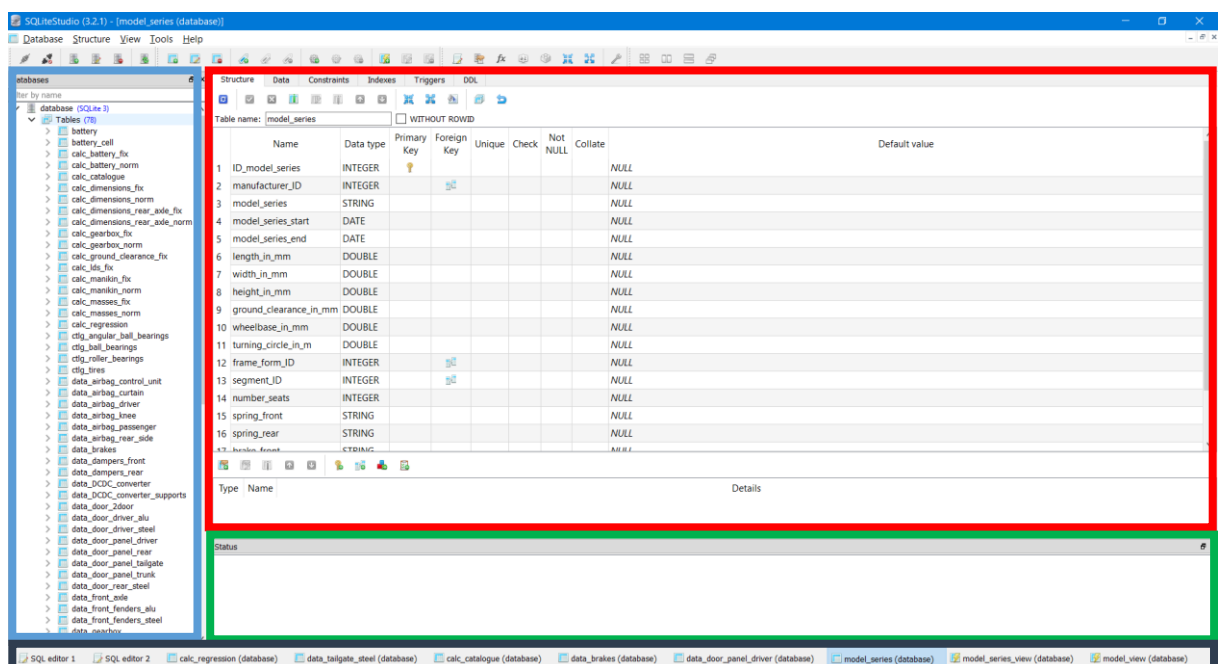


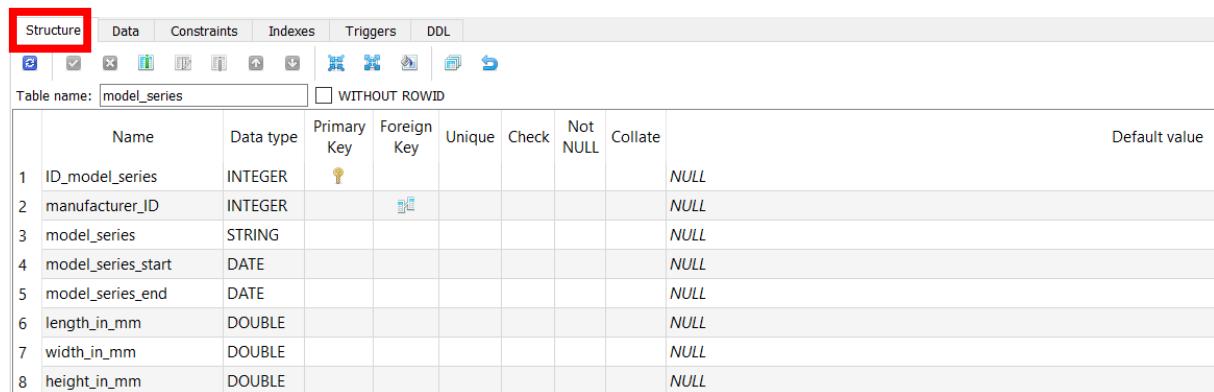
Figure 3.4: SQLiteStudio general interface

One of the most important operations in the database management, is the creation of a new table. For this scope, two options can be employed: manual creation of a new table or import of an existing table from another platform.

The manual creation of a new table to be filled can be done through specific commands on the SQL query editor for more expert users, or it can be easily made using the software interface. From the branch section showing the list of tables, user can create a new object. An empty structure is shown in the central panel.

Subsequently, the user can assign a name to the table and add columns one by one, defining their names and properties (the way to assign properties to columns is shown in the next part).

The structure contains the list of all the columns and an easy looking overview of all the column properties (Figure 3.5).

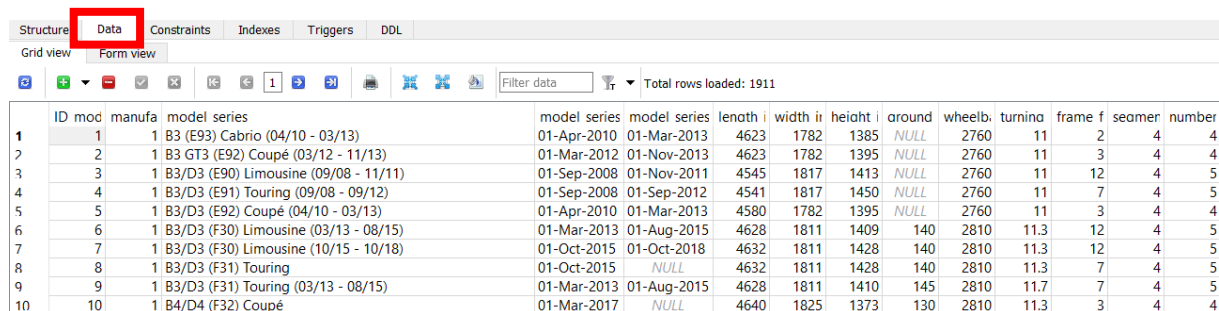


	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ID_model_series	INTEGER	Primary Key						NULL
2	manufacturer_ID	INTEGER		Foreign Key					NULL
3	model_series	STRING							NULL
4	model_series_start	DATE							NULL
5	model_series_end	DATE							NULL
6	length_in_mm	DOUBLE							NULL
7	width_in_mm	DOUBLE							NULL
8	height_in_mm	DOUBLE							NULL

Figure 3.5: Structure tab

Once the structure is ready, one can select the data tab and start adding data row by row, in a similar way as in calculation sheets.

Data section has a grid interface with rows and columns. In each cell, values are present. The view of the data tab is reported in the Figure 3.6.



	ID mod	manufa	model series	model series	model series	length i	width i	height i	around	wheelb	turnino	frame f	seamen	number
1	1	1	B3 (E93) Cabrio (04/10 - 03/13)	01-Apr-2010	01-Mar-2013	4623	1782	1385	NULL	2760	11	2	4	4
2	2	1	B3 GT3 (E92) Coupé (03/12 - 11/13)	01-Mar-2012	01-Nov-2013	4623	1782	1395	NULL	2760	11	3	4	4
3	3	1	B3/D3 (E90) Limousine (09/08 - 11/11)	01-Sep-2008	01-Nov-2011	4545	1817	1413	NULL	2760	11	12	4	5
4	4	1	B3/D3 (E91) Touring (09/08 - 09/12)	01-Sep-2008	01-Sep-2012	4541	1817	1450	NULL	2760	11	7	4	5
5	5	1	B3/D3 (E92) Coupé (04/10 - 03/13)	01-Apr-2010	01-Mar-2013	4580	1782	1395	NULL	2760	11	3	4	4
6	6	1	B3/D3 (F30) Limousine (03/13 - 08/15)	01-Mar-2013	01-Aug-2015	4628	1811	1409	140	2810	11.3	12	4	5
7	7	1	B3/D3 (F30) Limousine (10/15 - 10/18)	01-Oct-2015	01-Oct-2018	4632	1811	1428	140	2810	11.3	12	4	5
8	8	1	B3/D3 (F31) Touring	01-Oct-2015	NULL	4632	1811	1428	140	2810	11.3	7	4	5
9	9	1	B3/D3 (F31) Touring (03/13 - 08/15)	01-Mar-2013	01-Aug-2015	4628	1811	1410	145	2810	11.7	7	4	5
10	10	1	B4/D4 (F32) Coupé	01-Mar-2017	NULL	4640	1825	1373	130	2810	11.3	3	4	4

Figure 3.6: Data tab

Nevertheless, it is unusual that user is requested to do so in the current work. Tables can have lots of columns, thousands of data and they are usually already generated by other colleagues which works on single models.

To this purpose, the import command is more useful. Tables are usually obtained from Excel files or after some MATLAB elaborations. In any case, to import them in the database, they must be saved in “csv” files. Once the user has the table in the proper format, it enters the “Tools” folder, and makes the import procedure. He enters the table name he wants to generate and then select the file from the directory.

The only case in the current work in which the table needs to be manually created and filled is for calculation tables. User must set all the instructions for the MATLAB interface on how to get data and where to save them. Anyway, the structure is always the same (different only from the calculation category “fix”, “norm” or “regression”) so the operator can copy structure from an existing table of the same type and manually add data inside.

The imported table appears in the left branch. At this point, the table contains only the data and the column names. No properties or variable types are assigned. In the specific, user must define keys, constraints and the type of values contained in each column.

First, the referencing to the model series table is the core operation. From the structure tab, the user creates a new column that will contain the IDs that refer to the model series primary key. The column is called “model\_series\_ID” or a different name if it links to another primary key. Double clicking on the just created column, a dialogue window opens, showing column properties.

An example of that window is reported in Figure 3.7.

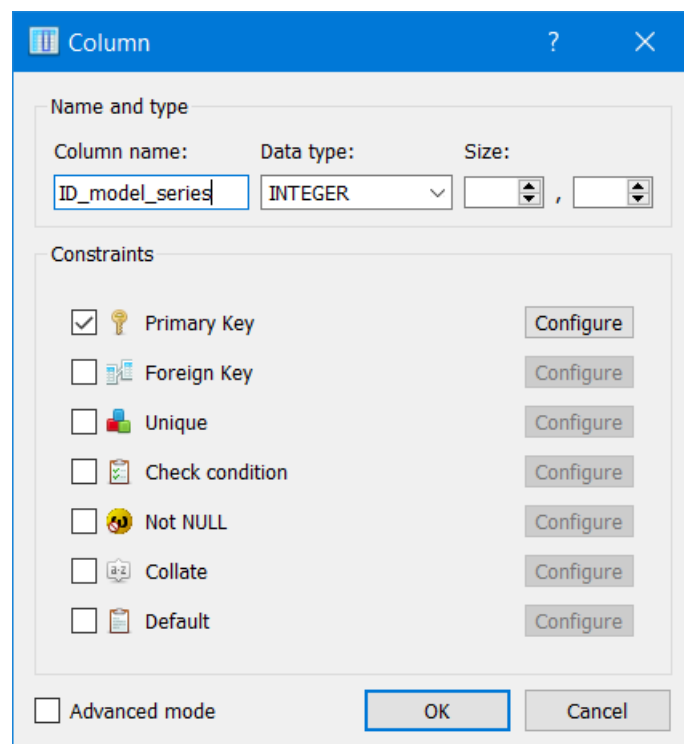


Figure 3.7: Column properties window

From this window, the user can further select all the properties of the column, assigning the column name and the type of data it contains (most common are Integer, Double and String).

Subsequently, the column becomes a key, by entering the corresponding section. When setting a primary key, one highlights the first field and enters the “Configure” panel. In the current project, primary keys are always integer numbers with autoincrement for every row is added. The configuration settings are reported in Figure 3.8.

**Primary key**

☒ Autoincrement

☒ Sort order: ASC

☐ Named constraint:

☐ On conflict: ROLLBACK

Apply Cancel

Figure 3.8: Primary key configuration

In rare cases, users need to set a primary key, while for almost every table, one or more foreign keys must be set. As already mentioned, every data table for model computations contains the reference to the ID of the model series table. After creating the “model\_series\_ID” column, operator enters the column properties and sets the column to be a foreign key. The configuration panel of the foreign key is reported in Figure 3.9.

**Foreign key**

Foreign table: model\_series

Foreign column: ID\_model\_series

**Reactions**

☒ ON UPDATE CASCADE

☐ ON DELETE NO ACTION

☐ MATCH SIMPLE

**Deferred foreign key**

☐ Named constraint Constraint name

Apply Cancel

Figure 3.9: Foreign key configuration

The foreign key must be linked with the primary key of another table. In the panel, the user selects the table for the linkage and the column in which the primary key is contained. One further operation is needed. In the Reactions section, it is important to select the Cascade action on update. This passage is crucial because makes the foreign key automatically change if a primary key value is modified. For example, suppose to have a certain row of the foreign key that is linked with ID=198 on its primary key. Due to some modifications in the primary key, the previous row with ID=198 now has identifier ID=200. In the foreign table that referred to that row, now the value ID=200 is reported.

After these operations, the structure has been set, with all column properties and keys. The new created foreign key column (usually “model\_series\_ID”) is empty. Each value referring to the primary key needs to be added.

In some cases, user manually enters the ID values looking for them in the mother table. For big entities, this is time consuming and inefficient. The person that prepares the model table to be integrated within the database, if possible, introduces the model series name. In this way, a dedicated MATLAB code can be run, and it is able to automatically assign the correct ID values. Only some single values that are not found need to be added manually. Operations are so faster and more efficient.

A final step is related to empty or NaN values in the columns. As already shown in section 3.3, there are some criticalities in the MATLAB interface when reading empty or NaN cells. For this reason, user has to replace all cells with these values with NULL cells.

This operation is manually almost impossible. No time can be wasted updating thousands of cells. The operation can be done column by column with dedicated SQL query command. This way is still time consuming, especially for large tables) and requires some SQL language knowledge. The fastest way is the use of a dedicated MATLAB code that needs the table name only. The script automatically acts on each column and substitute all required cells.

In all the operations, no SQL programming should be used. Anyway, it is possible to operate with SQL editor for more expert users. The editor shows an empty panel in which programmer can write SQL code. He can run queries and get the results in the lower panel. Finally, in the bottom part, in the status panel, he can get immediate feedback on the operations.

An example of the editor is reported in Figure 3.10, in which the command gets ID, manufacturer and model series of vehicles with wheelbase higher than 2800 mm.



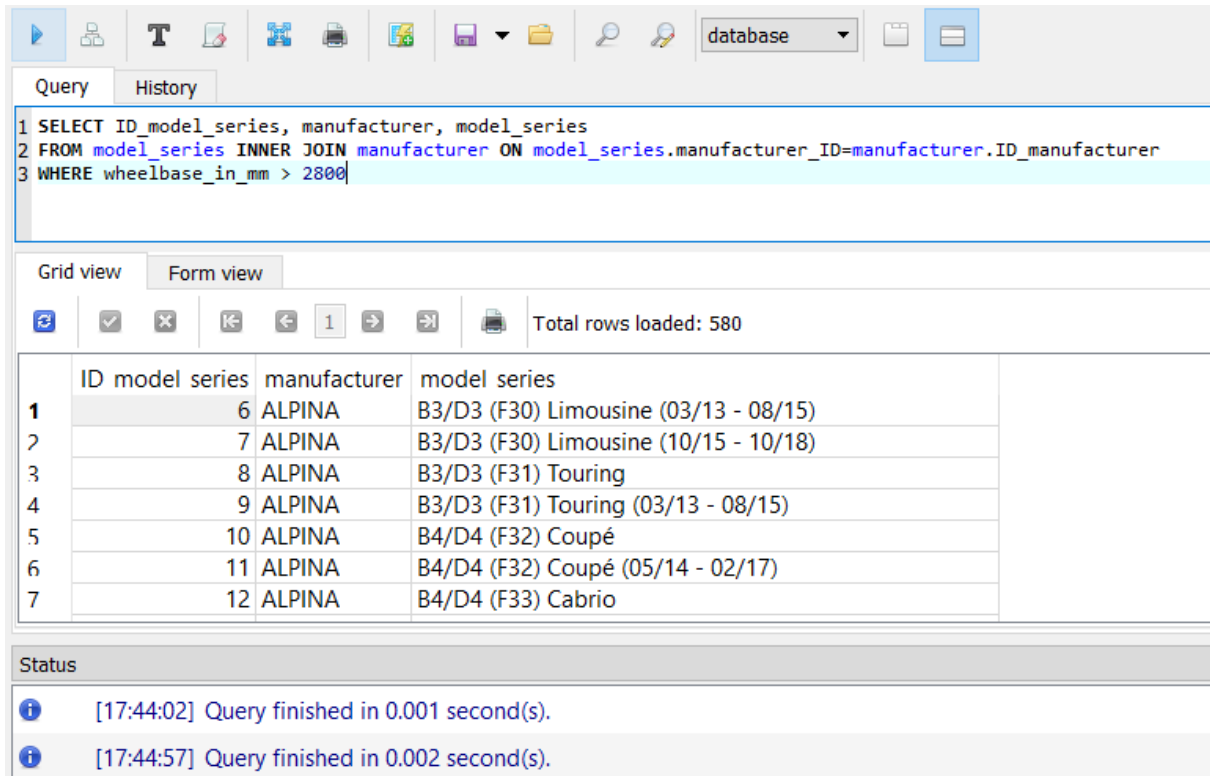


Figure 3.10: SQL query editor

Nevertheless, the complete integration of new tables in the database is possible with the software interface and some dedicated MATLAB codes only and requires almost no knowledge of SQL language for the operator. This is the core of the integration with MATLAB and use of a user-friendly interface as the one offered by SQLiteStudio.

### 3.5 Brief explanation of views

The SQLite interface is also useful to create views. A view is a particular representation of data that is not strictly related to table's structures. In the left side of the screen, below the list of the database tables, there is the list of views.

Using views, the user can visualize a table in a different way, usually with a more user-friendly representation. An example is useful to clarify this concept.

The model series table, contains the primary key "ID\_model\_series" and the foreign keys for the manufacturer ("manufacturer\_ID"), for the frame form ("frame\_form\_ID") and for the segment ("segment\_ID"). The primary key is the identifier for each row, needed to the other tables, but it is not a useful vehicle data. Moreover, the foreign keys contain information about the manufacturer in a numerical way, more robust and useful when doing automatic operations, but not user-friendly. A user who reads "manufacturer\_ID"=28 does not know what manufacturer corresponds to the number 28. To answer this question he must look into the "manufacturer" table, which is time consuming.

In this case, a view can be useful since it offers a user-friendlier representation of the table. The view for the table model series contains the name of the manufacturer instead of the ID number and the same for other foreign keys.. User can enter the data tab of the view, that has the same structure has the data tab for the real table, and work on data from there.

The structure tab is replaced by a query tab, which contains the list of the visualized columns from different tables and a lower panel contains the query command to obtain the view (Figure 3.11).

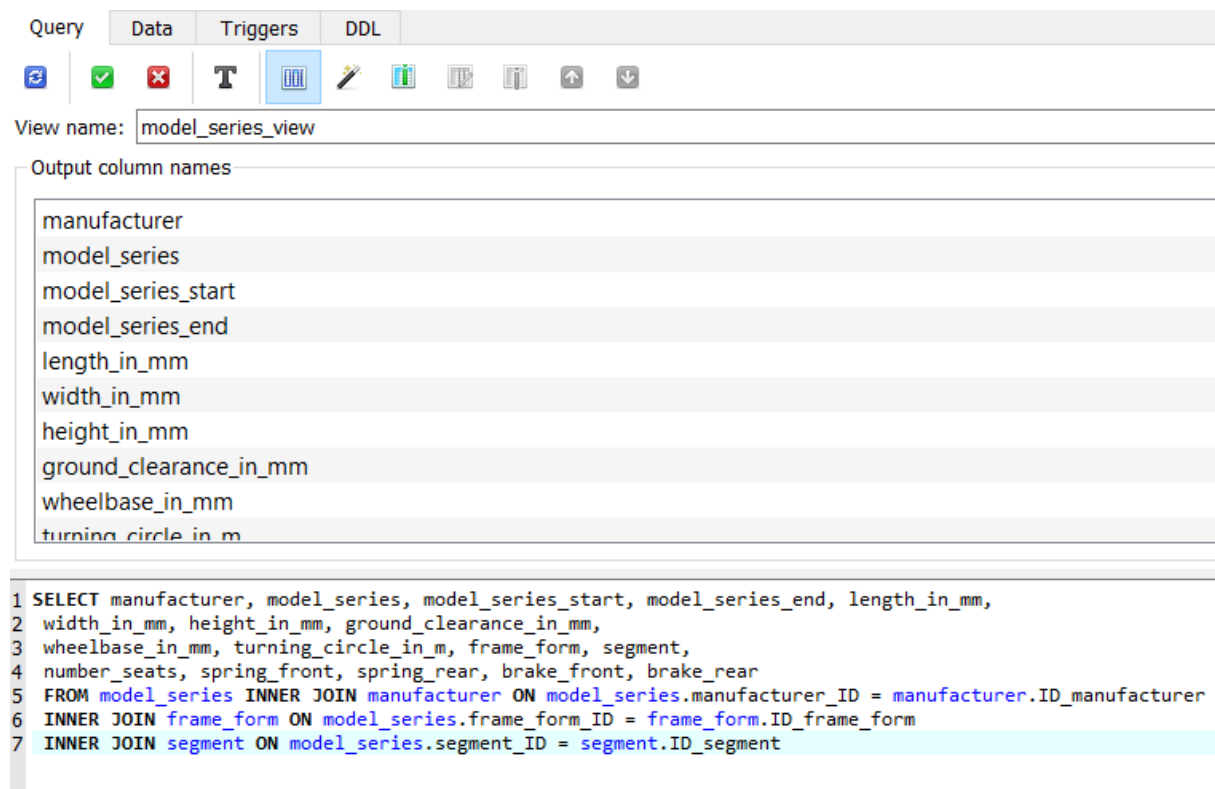


Figure 3.11: Query tab of views

The only problem related to views is that they can be created only with query commands . Therefore , some SQL knowledge is required to add new views, especially when the representation needs to get columns from different tables, in which a JOIN command is necessary.

## 3.6 Criticalities

The software offers the possibility to do many operations without having any knowledge of SQL language. Some knowledge is required, if the database has to be extended or new views have to be created.

Another problem is, that the very basic interface of SQLite can easily reach its limits. The left panel with the list of the tables is useful to move inside the different entities of

the database, but for big structures, as the one developed in this work, the list can become quite long. At this stage, the tables are 78, and the number has still to increase in the future. Anyway, the software does not offer the possibility to group tables in folders.

For example, it could be useful to collect all data tables together in a dedicated “data” folder or the same for catalogues or calculation table, which would make the list would more accessible.

Furthermore, when the user modifies a table structure or data, the changes are irreversible, since there is no way to retrieve previously deleted structures or values. The user must think before applying changes and accidental or superficial modifications can have bad effects.

Further problems are related to the import of NaN or empty cells from “csv” files. As already mentioned, when reading data with the MATLAB interface problems arise. This is because those cells are automatically set as strings. Even if the user sets a column to be integer or double, The NaN or empty cell is read as a string. This is the reason why it needs to be substituted with NULL value.

Finally, the last problem of the program is related to data tab. When manually adding values in a table, user can write inside cells, but no formulas or relations to other cells can be inserted (as in common calculation sheets as Excel), making manual data addition inefficient.



## 4 MATLAB pre-processing phase

The database is the core for the collection and management of all data. Nevertheless, the project needs to use those data to get all the input parameters it needs for the generation of the vehicle parametric model.

To this purpose, as already mentioned, several MATLAB functions have been implemented. In this chapter, the MATLAB interface which was created to interact with the database are analysed.

The first part (section 4.1) analyses the scripts dedicated to table integration in the database, to be used when the user introduces a new data table inside the list. Subsequently, in section 4.2 the main code is explained, together with its functioning and interaction with the database and all solutions adopted to obtain the Parameters structure.

### 4.1 MATLAB codes for table integration

When a new table is added to the database structure, user performs some common operations. Data tables for model's computation need two relevant steps: create the connection to the model series table through the foreign key containing IDs and set all NaN or empty imported cells to NULL value. MATLAB codes for the automatization of those procedures are presented in section 4.1.1 and 4.1.2 respectively.

#### 4.1.1 Get IDs for model series connection

When integrating a new data table in the database, one of the first things to operate is the creation of the connection with the vehicle's model series on the mother table. The column called "model\_series\_ID" is the foreign key to create the linkage.

To reduce required time to create the connection, the implementation of a dedicated automatic code was necessary. The script (called "get\_ID.m") does not completely replace some manual operations, but it significantly reduces the time needed to complete the procedure.

The only requirement to apply this code is that the person that prepares the model table to be integrated writes the exact name of the model series for each row as reported on the ADAC catalogue [41]. If this condition is fulfilled, the script can be executed. The

operator must write in the input section the name of table to which IDs need to be assigned.

For a faster comprehension, a flow chart representing the code steps is reported in .

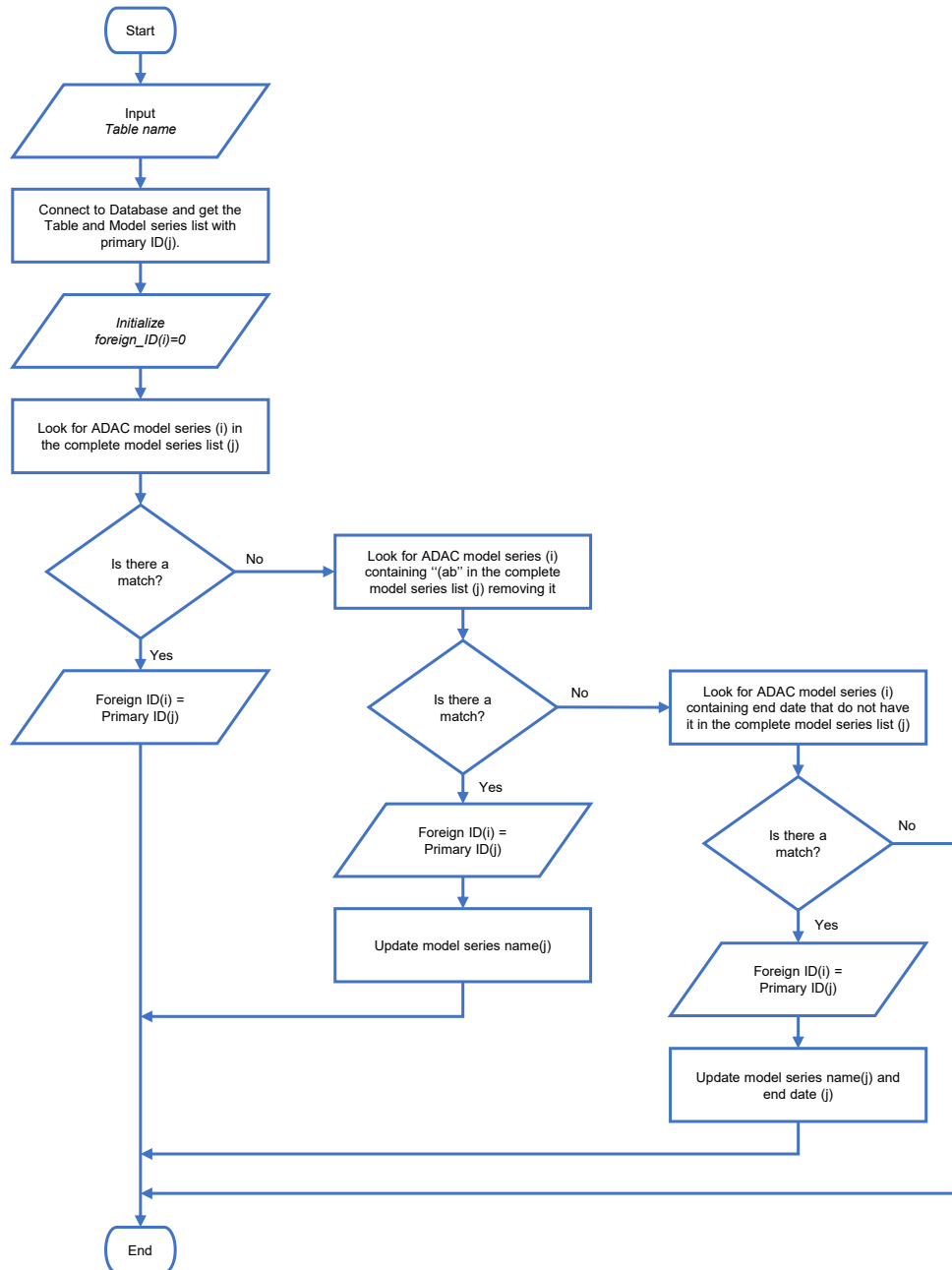


Figure 4.1: Flow chart of the MATLAB function to get model series IDs

The first operation that needs to be done, as almost for every code, is the connection with the database. Since codes are used to make operations on the tables, MATLAB needs to know which the database is. First, to make all operations on a database, the Database Explorer App of MATLAB must be installed. This packet lets the program connect to database, investigate data inside and take them to workspace [47]. The standard section of commands used to create the complete connection is presented in paragraph 4.2.

To get IDs, the code reads the complete list of model series and corresponding identifications. Through the connection, the script runs a query in the database and assigns the output of the command to a workspace variable.

In the same way, through the input name given by the user, code executes a query to obtain the table to which assign IDs.

In the workspace, script initializes a vector variable with the same dimension of the involved table to give IDs. The default value is set to 0.

At this point, for each row, code reads the ADAC model series name, and it looks for it in the complete model series list. If it finds it, the corresponding ID is assigned to the output variable at the current row.

The name from ADAC has a standard structure: there is the name of the model series followed by a parenthesis containing its initial and final dates in the format “(mm/yy – mm/yy)”. In some cases, the model series have not final date yet. For these values, the name is followed by a parenthesis with the initial date only in the following format: “(ab mm/yy)”. There is the German word “ab”, meaning “from”, as reported in the ADAC site.

This last convention was introduced in the current year. Before that, model series with no end date, did not have any parenthesis. Since the filling of the complete model series table started before introduction of this rule, many model series have no date part. This can give problems in finding the IDs.

For example, in the new introduced table, there is the ADAC name of “Stelvio (949) (ab 04/17)” but in the model series table there is “Stelvio (949)” because introduced before the date addition. MATLAB code takes care of that: if it finds “(ab” it looks for the name without accounting for the date part and if it finds it, it assigns the corresponding ID. Moreover, to progressively align the complete table to the new convention, it updates the name in the database automatically adding the parenthesis with the date.

Another problem is given by the progressive end of model series. There are certain vehicles that had no end date when introduced in the list, but the model series is now ended. To clarify, suppose to look for “Tiguan (II) (04/16 - 06/20)”. The model series ended in June 2020, but it was introduced in the list before that time. In the table it can figure as “Tiguan (II)” or “Tiguan (II) (ab 04/16)”. The MATLAB considers this possibility. It checks if the end date is after the creation of the list (in the current work if “/19” or “/20”). After removing the date, if it finds the name in the list, the script assigns the ID. Moreover, the name is automatically updated adding the end date and filling the field “model\_series\_end” with the correct value.

As visible, many possible mismatching problems are automatically found and corrected, also giving an automatic procedure to maintain the model series list up to date.

At this point, the column containing IDs is full. It can be copied and assigned to the database table. Since the column was initialized to 0, the rows containing a 0 value are the one where there is no matching. There are two possible situations in which this can happen:

1. The model series name is written in a different way or it contains an error.
2. The model series does not exist yet in the list.

In any case, for these single rows, user must manually write the ID and, if the case is the number 2, it needs to add the new model series to the list.

Anyway, this happens in minor cases and only some single values are not found.

The general procedure offered by the code is faster than manual implementation, more reliable and gives a good progressive automatic actualization of the model series list.

### 4.1.2 Set NULL values

When MATLAB interface gets columns for the computation of models, some errors can occur if the cell contains a NaN (not a number) or an empty value. The common error is that when those values are part of an integer or double type data set, they must be read as numbers, so a 0-default quantity is automatically assigned to the variable in the workspace. The problem arises because these zeros become part of the dataset and influence the result of the distribution or regression giving wrong responses.

This problem can be avoided setting NULL values. In this way the code removes the NULL rows from the dataset, without misleading the calculation.

The operator can manually perform those substitutions from the data tab or with the query editor in SQLiteStudio (as presented in section 3.3) acting on single columns. The issue is that this operation is time consuming, especially for large tables with many columns, and it requires some SQL language knowledge.

The MATLAB code to set NULL values, called “set\_null.m”, solves these problems and requires no SQL knowledge. The user gives the name of the table to which assign NULL values in the input section of the code. Then, the script operates the connection with the database. The detail of this step is presented in section 4.2.

The code retrieves the complete table to update from the input name given by the user, running a query and saving the computed output in a workspace variable of table type.

From the properties of this variable, the program can obtain the list of column names. For each column, an UPDATE-SET-WHERE query command is performed, and all NaN or empty cells are substituted by NULL values.

There is no output from the run since the Database Explorer App directly performs the operations on the database tables. If user updates the data view in SQLiteStudio, can directly see the applied changes.

As in previous case, a graphical flow of the code is reported in Figure 4.2.



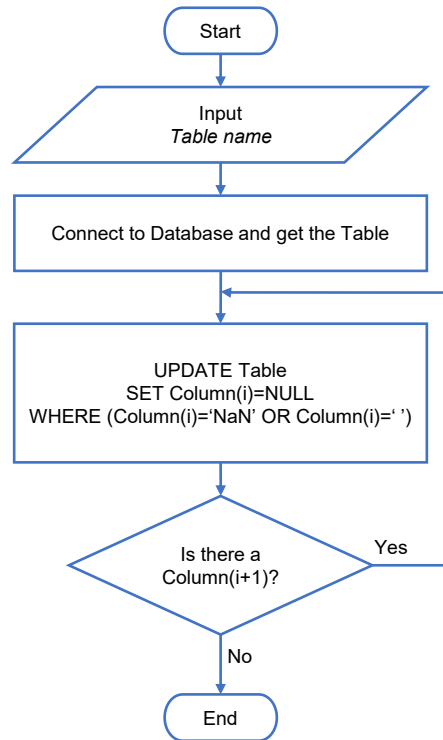


Figure 4.2: Flow chart of the MATLAB function to set NULL values

The presented codes are not used to get the real models yet, but they are crucial to the proper preparation of the data sets. User must use them to implement the tables in the correct way. No connection to model series' values can be created if there are no IDs in the foreign key and results are not coherent if data are not retrieved in the correct way.

## 4.2 MATLAB codes for complete pre-processing

When the database is actualized and all current tables are correctly integrated, it can be used to retrieve all needed empirical models.

To do so, a main for the pre-processing phase was created. The objective of this script is to read data in the proper way and save all models, fixed values, distributions, regressions, and catalogues in a structured variable called "Parameters". A structure variable is type of element that groups data inside sort of folders that are called "fields" [48]. Each field can store any type of data , and can further contain other fields.

With this approach, all values and models are organized and saved in a single entity. The complete variable containing all actualized calculations can be used as input variable to make all needed steps to obtain the vehicle architecture.

The aim of the developed code is the complete creation of the abovementioned Parameters variable. The chapter exploits the detailed process of calculation of the variable and all the additional solutions adopted to reduce the occurrence of errors.

Any time a new parametric model is needed, the Parameters variable is necessary before running the tool. So, user runs the following pre-processing script to obtain the most

recent and actualized version of the Parameters structure from the database. In this way, empirical models always refer to the latest version of the data sets contained in the database. Sections from 4.2.1 to 4.2.7 contain the specific steps of the process to completely retrieve all empirical models and check solutions.

### 4.2.1 Create the Database connection

The code interacts with data contained in the database. The connection is possible with the MATLAB Database Explorer App [47]. Through this add-on, a connection can be created with a database and script can perform calculations and run queries. Nevertheless, the code needs to know which is the database, where it is and all standard inputs for the connection. The operation can be manually performed through the app, but it is time wasting and it requires specific knowledge from the operator.

To avoid those problems, the following standard section substitutes the manual operations. Every time a script needs to connect to a database, this set of commands is required at the beginning of the flow.

In the section there is a first part related to the setting of some database preferences. It contains information on the standard values of empty cells of the tables.

After that, there are crucial information. Specifically, the path and the name of the database are set. There can be different solutions in retrieving the path. The simplest one is to write the complete local path of the database location. This solution is not possible in the current work. The folder in which codes and database are saved is shared among different operators, so that different users can run the pre-processing code to obtain the Parameters variable from different devices. For any PC, the path is different up to the shared folder.

To overcome this problem, the script retrieves the location of the running MATLAB file and gets the path of the shared folder. Then the constant part is added, and it is always the same.

Then, there are some further inputs that must not be changed. They contain some technical settings for the connection.

At this point the linkage is ready and the result is stored in a particular connection variable. The code uses this variable to interact and make operation on the database.

To end this section, a table variable is stored in the workspace. It contains the basic information of the database structure. Particularly relevant, it contains the list of all the table names that are used to automatize the whole process.

A flow chart of the section is reported in Figure 4.3.

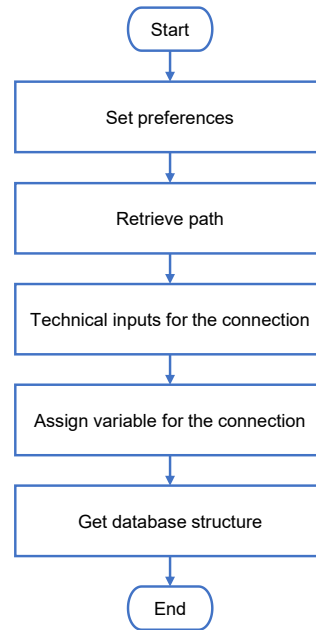


Figure 4.3: Flow chart of the section to create the Database connection

### 4.2.2 Generate the MATLAB file to store results

The code saves the results of every pre-processing phase. There is a dedicated folder that contains all “Parameters.mat” variables that are progressively generated. This step is useful to have an historic recording of the generated models. In this way, every previous version of the input file for the tool can be retrieved. This can be done to go back to a previous version of the code if some errors occurred or to make comparisons with older models.

Moreover, this step is crucial for the tool presented in section 4.2.7, dedicated to the complete comparison of the new generated variable with the previous one.

The variables have a numerical order that the code can automatically assign. If, for example, the last saved output is “Parameters\_16.mat”, MATLAB creates the file “Parameters\_17.mat”.

Up to now, code does all preliminary steps for the proper running. Everything is now ready for the real computation of values and models.

### 4.2.3 Assign fixed parameters

At the end of the database connection step (section 4.2.1), the complete structure of the database was saved. As already mentioned, this variable contains information about the whole structure of the database, and it is particularly important because it contains the complete list of table names.

This list is relevant to have a complete automatization of the pre-processing phase. In this way, it has no importance if new tables are added because the new names are autonomously got in the following run. The only requirement is that every table that is

added to the database fulfils the nomenclature requirements: use prefix “data\_” for tables containing data for the models and “calc\_” for calculation tables. Moreover, the suffix “\_fix”, “\_catalogue”, “\_norm” or “\_regression” must be present for this second category.

It is important to underline that the table must not contain the abovementioned prefixes or suffixes inside their specific names. This can give problems in running the different sections.

The code cyclically reads every table name and runs this section whenever it finds the “\_fix” strings.

When it finds a calculation table for fixed values, the code generates a query instruction and runs it in the database through the connection.. The procedure works as follows:

1. Create the query string.

```
SQL_query= ['SELECT * FROM ',char(Database_structure{i,'Table'})];
```

The command selects the complete table for the fixed parameters.

2. Execute the query in the database through the connection. The execution is saved in a variable called curs that is an object containing information about the running command. The arguments required for this step are the variable of the connection and the query. For this scope the function “fetch” is used.

```
curs = fetch(exec(Database, SQL_TableOutput));
```

3. Assign the output of the calculation on the database to a workspace variable of MATLAB.

```
t_fixed_parameters = curs.Data;
```

4. Close the cursor object. It is required to improve running time. If not closed, this causes a stacking of curs objects thus requiring more memory.

```
close(curs)
```

At this point, the workspace contains the table of fixed parameters to be assigned. The table contains the MATLAB variable name to which assign the result and a second column with the value. The code further enters inside a dedicated function called “retrieve\_fixed\_parameters.m”.

Being single values, no queries or further calculations need to be performed. The MATLAB command “eval” is used. This instruction contains a string as argument. The function is used to run the argument string inside the command window of MATLAB. In this way the variable name and value are not explicitly written, and the procedure is automatic.

To clarify, Table 4.1 shows an example. The code must assign the value of the maximum permitted weight for passenger cars, imposed by law at 3500 kg [15].

The created table in the workspace (retrieved from the database fixed calculation table) contains the name of the variable to be assigned and the value.

Table 4.1: Argument table (Fixed table)

parametername	value
Parameters.masses.weight_max_permitted	3500

To manually assign the result the eval function must run the following command:

```
Parameters.masses.weight_max_permitted=3500;
```

Which can be achieved with the following code:

```
eval([Table.parametername{i}, '=', num2str(Table.value(i)), ';']);
```

Whatever the variable name and the value are, they are never written in the code and the same single line of code can be cyclically applied for every value. The string that is generated is the same as the manual one and the code saves the proper value in the defined structure path.

The flow scheme of the section is reported in Figure 4.4.

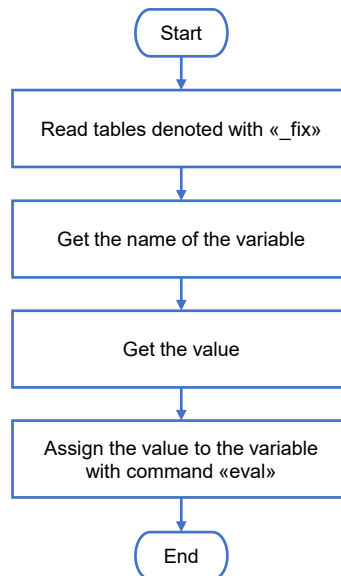


Figure 4.4: Flow chart of the section to assign fixed parameters

#### 4.2.4 Assign constant values computed from normal distribution

The following part is related to the computation of normal distribution models. The code enters this section whenever it finds the word “\_norm” in the table name taken from the complete list of tables of the database.

When it finds a calculation table for normal distribution values, the code generates a query instruction, and it runs it through the connection to retrieve the required data for the derivation of the constant value.

After this step, the code has all the needed instructions to compute the normal distributions and it enters a dedicated function called “retireve\_norm\_dist\_parameters.m”.

Differently from fixed values, the calculation table does not contain single values to be directly assigned, but a set of columns used to build an articulated query to get the proper results.

As already presented in chapter 3.2.4, the table contains the MATLAB variable name in which to store results, table and column names from which to take the data set and a list of fields to apply some filtering. So, before computing any model, the script must generate a query query to retrieve the data for the constant value calculation.

To this purpose, the code enters a dedicated function called “load\_data\_from\_database.m”. It completely automatizes the generation of the query and it gets the data set on which normal distribution analysis will be performed.

First, it takes the column and table names from which to take data and it starts building the query with the standard SELECT-FROM command. If the complete column needs to be taken, no further operations are needed. But, in many cases, the code must consider some filtering action. To do so, the function must add a WHERE section in the query.

The filters are contained in the second part of the calculation table and they can be more than one with a standard structure composed by the column and table for the category, the type of filter and the value that is used.

The function automatizes the categorization counting how many the filters are and adding a condition for each of them. The only requirement for the proper working is that operator follows the guideline instructions presented in 3.2.4.

The code gets the values of the four filtering columns and for each category enters a further function to complete the query generation. The function is called “calc\_SQL\_string.m”. The output of this function is the complete query to get data.

The procedure is completely automatic depending on the type of filter. The specific explanation of the dedicated function is reported in Appendix C.

At this stage, the complete query is in the workspace and can be used to get data from the database. An example of the query with applied filters is:

```
“SELECT axle_length_in_mm FROM data_rear_axle WHERE axle_length_in_mm != ""  
AND rear_axle.axle_type LIKE "trapezoidal_link" ”
```

It takes the values of axle lengths of rear axle considering only trapezoidal link suspensions.

With the usual procedure, through the connection, the code executes the query on the database and saves the obtained data in the workspace. In case the script gets no data from the execution, it informs the user with a warning that an error occurred. Otherwise, data has been successfully retrieved and the function ends.

The data set becomes the input for the statistical tool presented in chapter 2.3. The function performs the normal distribution analysis and stores the results in the selected

path using “eval” commands. The Parameters variable needs the mean of the distribution without outliers that is automatically assigned.

The process for the normal distributions model is schematized in Figure 4.5.

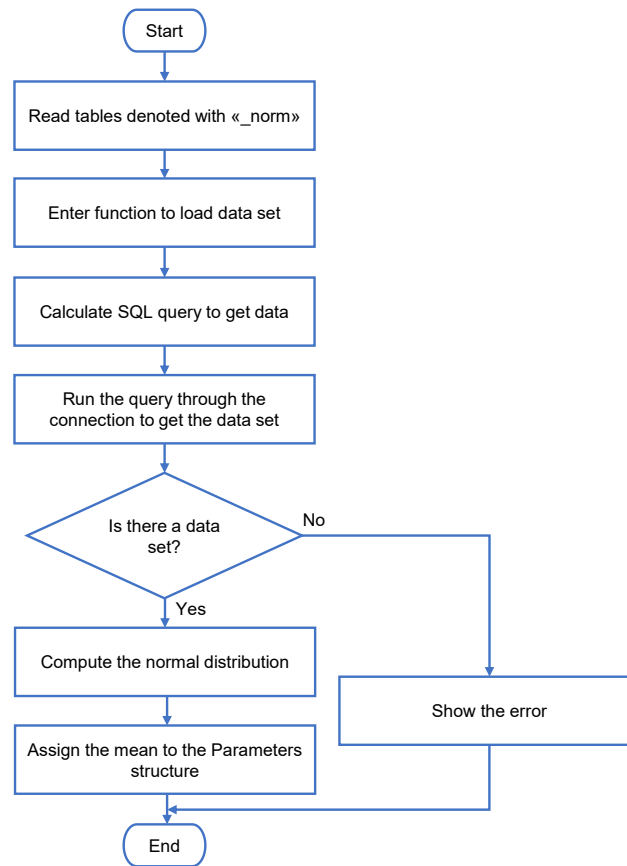


Figure 4.5: Flow chart of the section to assign constant values from normal distribution

### 4.2.5 Assign catalogues

The catalogues are reported in the database as they need to be saved. The code makes no modifications or calculations in this section.

When it finds a calculation table with suffix “\_catalogue”, it runs the current section. It uses the usual command with the database connection to get the calculation table containing the list of catalogues.

As already mentioned, the code needs the name of the variable in which to store results and the catalogue name in the database. No other factors are needed because the whole table must be taken as it is in the database. For each catalogue, a SELECT \* FROM command is used. It gets all column of a defined table.

Using an “eval” command, the code stores the complete catalogue in the correct variable name and it repeats it for each catalogue.

The simple process is reported in Figure 4.6.

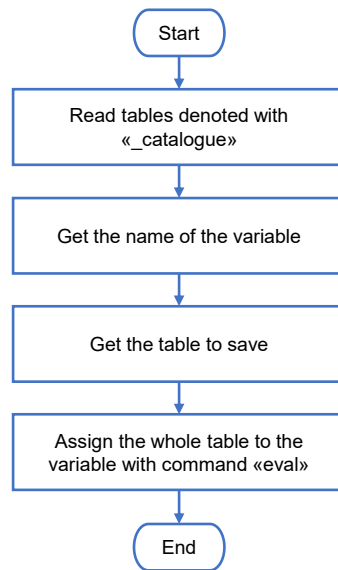


Figure 4.6: Flow chart of the section to assign catalogues

### 4.2.6 Assign regression models

At this stage, the code enters the last computation section. This part is dedicated to the storage of regression models exploiting the interaction between different dimensions. It is the most complex part for computation and automatization.

The section regards all calculation tables with the suffix “\_regression” in the name. In the current version, there is only one regression table containing all the models to be computed, but the use of the suffix automatizes the process. If more calculation tables containing regression will be added, the script executes this part for each of them.

As in all previous blocks, the first operation is the running of the query through the database connection to obtain the calculation table in the MATLAB workspace.

When the calculation table is ready, the code enters a dedicated function for the computation that is called “retireve\_regression\_parameters.m”.

To compute regression models, the statistical tool needs independent variables and the dependent variable. The independent variables can be more than one, and, in that case, they are all listed in the same cell divided by “;”. The use of this symbol is necessary for the automatization of the process. In this way, only one cell contains all independent factors, no matter how many they are. The code autonomously divides them using the “;” separator.

The same solution is used for the table names. Since the model needs different columns that can come from different tables, they are listed with separator and automatically divided. The list of table names is necessary to perform the JOIN command in the query.

Moreover, as for normal distributions, the tool gives the possibility to apply filters. The categorization is performed in the same way, adding WHERE conditions in the query command. See section 4.2.4 for the explanation of categories.



Being the generation of the query different from the one of normal distribution, there is a dedicated function called “function\_regression\_query.m”. It creates the complete query to get the data set for the statistical tool.

First, it uses the list of independent variables and the dependent one to fill a SELECT-FROM command. The response must be written as last column because the statistical regression function automatically reads the last column as the dependent variable.

The list of needed columns can be in different tables of the database. So, if there are more table names in the calculation cell, the function adds JOIN commands to the query. A different join is necessary for each different table.

When doing this step, the code automatically creates the connection with keys using the model series IDs.

The following block then adds the conditions for the filters. The procedure is like the one for normal distributions with the “switch” command to distinguish different category types. There is an additional part at the beginning that checks if the table for the categorization is already part of the joined tables. If not, it adds another JOIN command to the query.

At this point, the query is complete, and it can be used by the program. An example of the query is reported. The query gets the data for the regression model of the turning circle of the vehicle, using wheelbase and width of the vehicle as independent variables. The command example filters out the values for vans and motorhomes.

```
“SELECT wheelbase_in_mm, width_in_mm, turning_circle_in_m FROM model_series  
INNER JOIN frame_form ON frame_form_ID=ID_frame_form INNER JOIN segment ON  
segment_ID=ID_segment WHERE (frame_form NOT LIKE "Bus" AND frame_form NOT  
LIKE "Wohnmobil)”
```

The code runs the query through the database connection and gets the data set for the computation. The obtained table is the input for the statistical tool presented in 2.4.

The result of the computation is saved in a struct variable. It contains all information related to the computed model: the table, the statistical properties, significance test results, and an object that is called “Regression” by default that is of “Linear model” type. This last element contains useful information as the coefficients of the linear equation, the linear formula, the residuals, and many other elements.

Clearly, not all the values of this structure are useful for the current work. For this reason, the script takes only the main relevant statistical results and the coefficients of the linear equation. They give a clear overview of the model and the possibility to make an evaluation. If the Excel report is needed, the name of the variable is generated with “function\_complete\_name.m” and results of significance tests are also taken.

The result of normal distribution was a single value, that is the mean of the data set. For this reason, the Parameters fields for normal analysis contain one single value. For regression models instead, the main result is a linear equation that is used to estimate the value of the dependent variable. The main tool needs to retrieve this equation in some

way, assign the independent variables to the model, and get the estimation knowing the coefficients of the formula.

To automatize the process the code uses MATLAB function handles. It is a particular MATLAB variable which can store a function. Function handles can be input arguments to other functions that evaluate mathematical expressions [49].

In this way, when calling the regression model from the Parameters variable, there is no need to explicitly write the equation because it is contained as function handle in the Parameter field. The code gives the input independent variables and the handle that contains the linear equation, automatically computes the result. With this solution, even if the number of independent variables changes or the user updates the data sets in the database, there is not the explicit formula and no changes must be done in the MATLAB code. It is another great solution towards user-friendliness and modularity of the tool.

An example can clarify the solution the model for the turning is used. The model estimates the turning cycle as function of wheelbase and width of the vehicle.

The computation of the model gives the coefficients of the linear regression equation. The result of the formula is given in the equation 4.1.

$$turning\_circle = -0.35983 + 0.0018479 \cdot wheelbase + 0.0036078 \cdot width \quad (4.1)$$

The formula is not written in the code in explicit form because new data can change the coefficients and the equation should be manually updated from the user.

The pre-processing code takes the coefficients and builds the equation, then it creates the function handle to store the linear equation in the Parameters variable. The command for the function handle is assigned with the “@” symbol, saving the list of input arguments of the formula and then the linear equation. Then the script assigns the function object to the correct path of the Parameters variable in the field that is called “.eq”.

The command to get the function handle is “*function=@(var\_1, var\_2, ..., var\_n)equation;*” where the argument in the parenthesis is the list of the variables and the equations contains the formula to be assigned. For the current example, the command is:

*“function=@(wheelbase, width)-0.03598+0.0018479\*wheelbase+0.0036078\*width;”*

The code then assigns the object “function” to the proper variable path that in this example is “Parameters.regr.turning\_circle.eq”. It simply takes the name of the variable from the calculation table of the database and it adds the “.eq” field.

The main tool can use the object to compute an estimation of the turning circle of the vehicle, using as input arguments the wheelbase and the width. The script needs to call the field with the function and, using input arguments, it computes the result.

Supposing that wheelbase “wb=2700” mm and width “w=1700” mm.

*“tc=Parameters.regr.turning\_circle.eq(wb, w)”* gives the estimation of the turning circle with the just calculated linear regression model and it assigns it to the variable “tc” (“tc=10.7628” m).

The linear equation never appears in the main code. If, for some reasons, user adds or removes an independent variable no changes are needed. If data are updated and the model coefficients change, the pre-processing automatically assigns the new equation to the function handle from the coefficients and the main code directly uses the new version.

The path of the function handle is the only element that must be kept constant or that must be updated in the code if it changes in the database.

This solution is a good improvement of the code in terms of complete automatization and user-friendliness of the interface.

The process for the computation of regression models is represented in Figure 4.7.

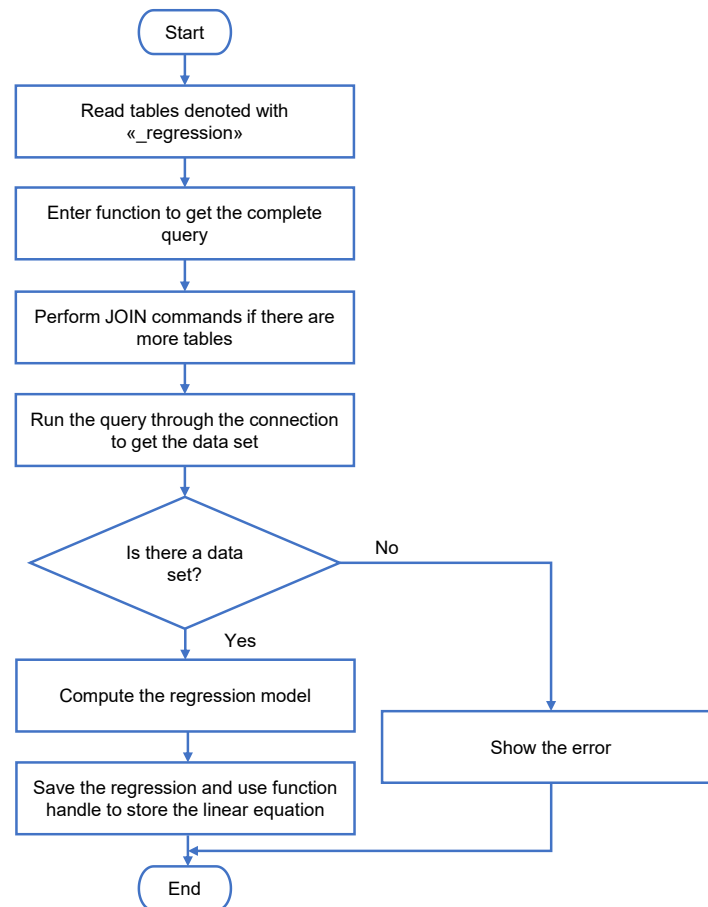


Figure 4.7: Flow chart of the section to assign empirical regression models

At this stage, all necessary values and empirical models are in the Parameters variable. The real pre-processing computation ends, and the object can be used as input for the main tool. Nevertheless, some additional steps have been developed, to improve the robustness of the whole process and reduce the occurrence of errors caused by unwanted modifications.

### 4.2.7 Structure changes complete check

At the beginning of the pre-processing code, the script unpacks the previously generated Parameters variable, and it lists all the fields with their values in a table.

It makes the same procedure with the just generated structure, obtaining another table with the whole list of fields. To do so, there is a dedicated function that is called “loop-structure.m”.

It takes a structure type variable as input and it gives back a list of all the fields, and corresponding values, in a table with two columns.

For each layer, it reads all the fields and controls if they are still structured variables or single cells. Whenever it finds a structure, it enters in a deeper layer and performs the same control. It does the same up to when it reaches the end of the branch and it gets a single field. At this point, it saves the entire field path and its value in the first and second column of the output variable.

The current version of the function arrives to a seventh layer of the structure. It is sufficient for Parameters. If it finds further layers, the code informs the user that a deeper computation is needed.

At this point, the two tabular variables can be used for the comparison.

There is a dedicated function (“compare\_structure.m”) that performs the complete analysis. It is useful to check for different type of changes that occurred in the pre-processing phase.

First, the script compares the number of fields of the previous and new Parameters variable. The size of the tables gives the number of fields. The code writes a message for the user, informing him if there is the same number of fields or if one of the two has more fields.

The second section controls if there are fields in the old structure that cannot be found in the new one. It takes each name of the old structure and it looks for it in the list of new fields. This control is useful to check if some variables have been removed or renamed.

After that, the script checks if there are fields in the new structure that were not in the old one. It takes each name of the new structure and it looks for it in the list of old fields. This feature is needed to know if some variables have been added or renamed.

Then, user needs to know which the changed fields are. To this purpose, the code shows in the command window the list of field names in two columns. One for the fields in the previous structure that are not in the new one, and one for the names that are in the new one but not in the old version. In case one of the two conditions is not verified (for example there are only fields that have been removed or added), it shows one column only.

In this way, the operator can see changes and check if they are wanted or not. If, by mistake, he removed a field from the database, it disappears from the variable and it can be aware of the error.

The last check is related to the values. The control is performed on all the fields that are equal in the previous and new structure.

For each of them, it takes the value in the second column and it compares them. The comparison is not performed for function handles or table fields because not suitable for “isequal” command. If the code finds that a field value changed, it informs the user telling which is the field and shows the previous and new values. In this way the user can check if there are unwanted changes in the data of the database. The values change due to variations in the data sets.

The comparison process is schematized in Figure 4.8.

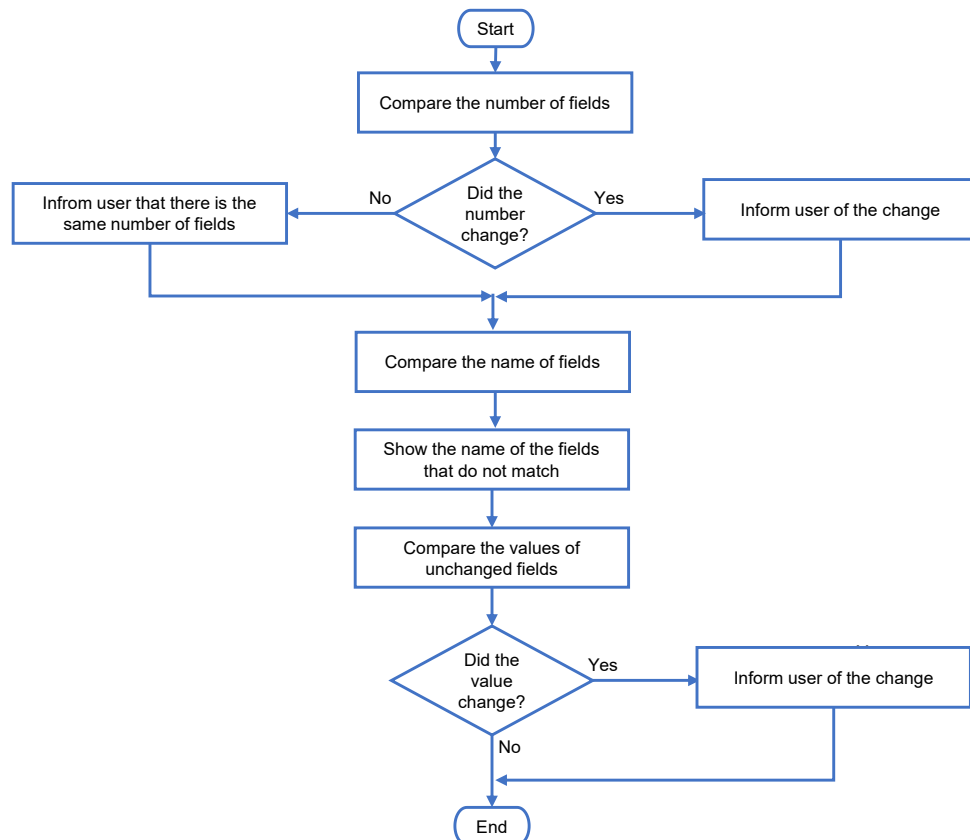


Figure 4.8: Flow chart of the section to compare structures

The block ends saving the Parameters variable in the archive folder.

This function is fundamental for the complete control of changes. Saving all the generated Parameters variable gives the possibility to retrieve older versions of the pre-processing and overcome the occurrence of unwanted mistakes.



## 5 Spring design for dimensional chain of the rear axle

The last chapter of the work presents an application on the tool models. Specifically, the dimensional chain for the evaluation of the available space in the rear axle (section 5.1). The following part (section 5.2) is devoted to the computation of the primary elastic member [4]. It is useful for the presented dimensional chain. In the application, some of the data contained in the database are used to make the design procedure. The final section 5.3, is dedicated to the results of the developed design process.

### 5.1 Dimensional chain of the rear axle

As already presented in chapter 1, dimensional chains are the core of the main tool of the project. The rear axle dimensional chain is composed by different steps. In this specific application, the developer is interested in the Y-direction of the space.

First, the reference frame is defined. The X-axis is directed towards the back of the vehicle. The Z-axis exits from the ground. The Y-axis is consequently defined with the centre on the symmetry plane of the vehicle and the positive side on the rear axis towards the left side. The reference frame of the rear axle is reported in Figure 5.1 [50].

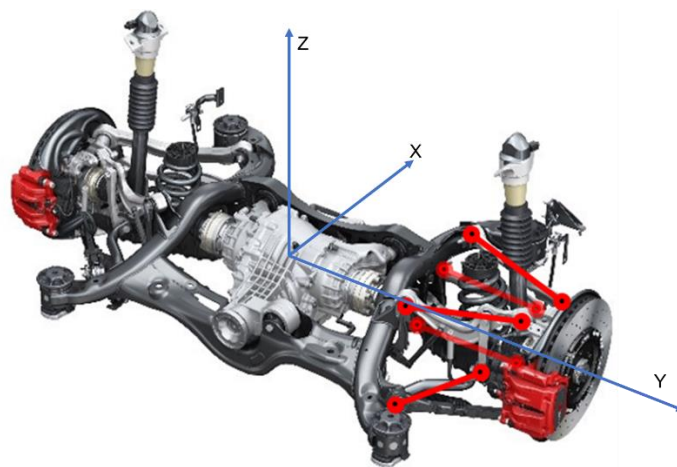


Figure 5.1: Rear axle reference frame [50]

It is important to underline that the main elements influencing the current dimensional chain are the wheel, the damper, and the spring.

For this reason, according to [50], the main steps to evaluate the available space are the followings:

1. Calculation of tire positions.
2. Calculation of the wheel arch.
3. Determination of damper dimensions and damper position.
4. Determination of the spring dimensions and spring position.

The tire dimensions and positions are required to design the wheel arch both in XZ plane and YZ plane.

Other vehicle components must avoid any type of collision with the wheel arch. After the definition of the wheelhouse, the damper is positioned. It is placed on the lower axis of the suspension system and it can have an angle orientation. The angle is designed exactly to avoid contact with the wheel arch.

Subsequently, the spring must be positioned. It can have different layouts with respect to the damper. It can be positioned in X direction with respect to the damper, in Y direction or coaxial (Figure 5.2 [50]). For the dimensional chain of the rear axle in Y direction, the most critical is the layout of spring in Y that is the one considered in the design phase of section 5.2.

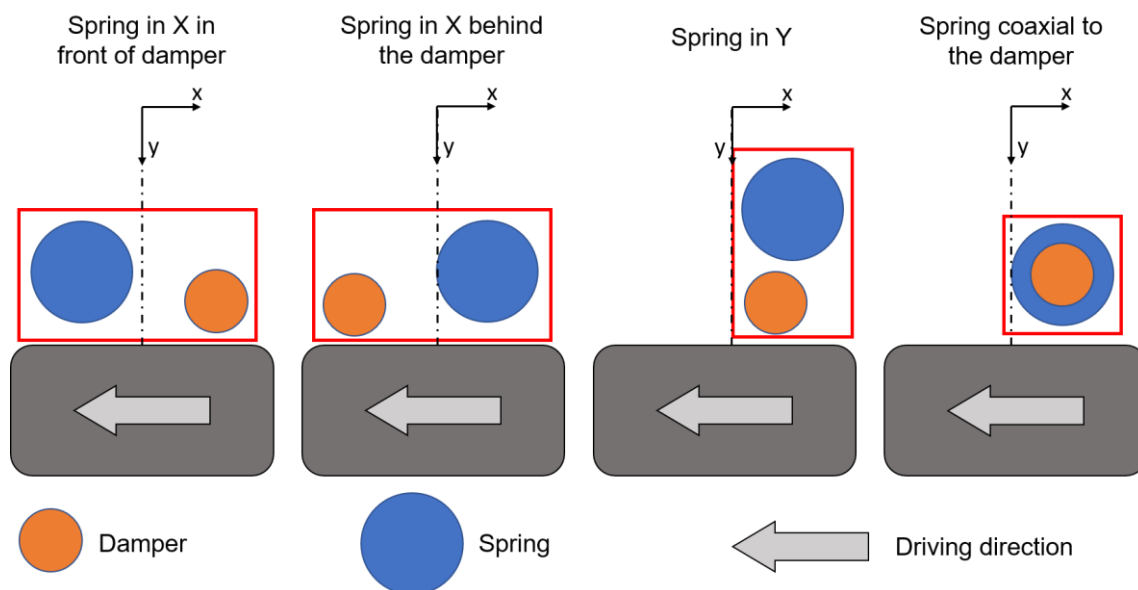


Figure 5.2: Possible Spring-damper layouts [50]

It is visible that the configurations of spring in X in front and behind the damper optimize the available space in Y direction but not in X direction. The spring in Y configuration optimizes the X, but it is the most restrictive in Y direction. Finally, the coaxial reduces the needed space, but it requires more complex solutions for design and assembly.

Determining the spring dimension and its position in Y direction, from the complete width of the vehicle it is possible to know the available space between the axles of the two vehicle sides. The complete process is detailed and already defined up to the step 3 (page 72) [50].



The missing step is the definition of the spring dimension. There is the need of a procedure that uses the available data, that is quite limited in this early development phase, and tries to get an estimation of the spring dimensions (section 5.2).

## 5.2 Spring dimensioning procedure

The procedure to derive the dimensions of the rear axle spring bases on the following assumptions:

1. Consider only helical coil steel springs (no air springs).
2. The spring is supposed perfectly cylindrical, meaning that the external diameter is constant for its entire length.
3. The behaviour of the spring is linear in all the deformation range.

This assumption are due to the few input parameters available at the early design phase.. At this stage, the tool computes a preliminary model, and the measures are useful just to get an estimation of the space occupied by single components and modules. The abovementioned conditions are acceptable at this level of the design process.

On the base of the hypothesis, an estimation procedure has been designed. The main idea of the process is to create a catalogue of possible springs. Subsequently, some controls are performed on each coil and only springs that satisfy all the conditions can be considered as acceptable for the specific vehicle.

The procedure begins with the definition of the necessary inputs that are available from the user inputs or from already executed computations. The Table 5.1 shows the list of available factors.

Table 5.1: Available inputs

Input	Unit of measure	Description
Axle type	-	Type of rear suspension mounted on the vehicle
Layout	-	Positioning layout if spring and damper of the suspension
$m_{\text{rear}}$	kg	Part of the weight load that acts on the rear wheels
$y_{\text{sa}}$	mm	Position in Y of the lower hinge of the shock absorber with respect to the centre of the vehicle
$m_{\text{axle}}$	kg	Mass of the rear axle
$\theta_{\text{sa}}$	°	Inclination angle of the shock absorber with respect to the Z-axis
$d_{\text{sa}}$	mm	Piston diameter of the shock absorber
$b$	mm	Length of the suspension lower axis
$y_{\text{fixed\_arm}}$	mm	Position in Y of the fixing point of suspension to body with respect to the centre of the vehicle
$z_{\text{wh}}$	mm	Position in Z of the upper point of the wheelhouse with respect to the centre of the wheel in kerb weight conditions
$d_{\text{tire}}$	mm	Diameter of the biggest tire that can be fitted on the vehicle

The limited list of available data, does not give the possibility to design the spring but is enough to estimate its dimensions.

The process uses a reverse approach. Instead of designing a coil suitable for the vehicle, the program builds a catalogue of springs and then, suitable ones are selected through a filtering action.

### 5.2.1 Spring catalogue

To prepare a catalogue, the main spring properties must be defined.

A perfectly cylindrical coil spring can be defined by some dimensions that are schematized in Figure 5.3 [51, p. 499] and explained in Table 5.2.

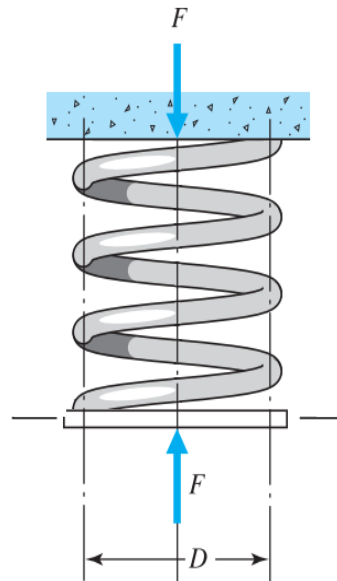


Figure 5.3: Helical coil compression spring [51, p. 499]

Table 5.2: Spring properties

Dimension	Unit of measure	Description
$D$	mm	Mean diameter
$D_{\text{ext}}$	mm	External diameter
$D_{\text{int}}$	mm	Internal diameter
$d$	mm	Wire diameter
$w$	-	Winding ratio
$n$	-	Number of windings
$i$	-	Transmission ratio
$L$	mm	Length of the coil

The winding ratio is defined by equation 5.1 [51, p. 500].

$$w = \frac{D}{d} \quad (5.1)$$

By using some of the parameters listed above, the catalogue is derived. The catalogue assumes a tabular structure with different properties of the springs. Each row of the table represents a different coil (Table 5.3).

Table 5.3: Spring catalogue layout

	D [mm]	d [mm]	w [-]	i [-]	n [-]
Coil 1	190.4	17	11.2	0.53	3
...	...	...	...	...	...
Coil n	85.2	12	7.1	0.65	7

The database is useful in this first part of the process. The catalogue is not directly available so the script must build it. The data table dedicated to the rear suspension, contains information about different rear axle elements. Rear springs are also part of the list. The idea to create the catalogue is to take the maximum and minimum of the variable ranges from real vehicles and vary with constant step inside the interval.

For this reason, the maximum and minimum value for the wire diameter and the winding ratio are taken from the database. Generally, using available data, the wire diameter is an integer number. So, the variation in the interval is set to 1 mm. Winding ratios vary with a 0.1 step.

So, by inverting the equation 5.1 for each couple of winding ratio and wire diameter, a value of mean diameter  $D$  can be computed.

Subsequently, the database is used to obtain the range of possible values of transmission ratios. This value gives information about the positioning of the spring. It is the transmission factor that correlates the movement of the wheel with the movement of the spring attachment.

To better understand the concept, the length of the arms is reported Figure 5.4 [50].

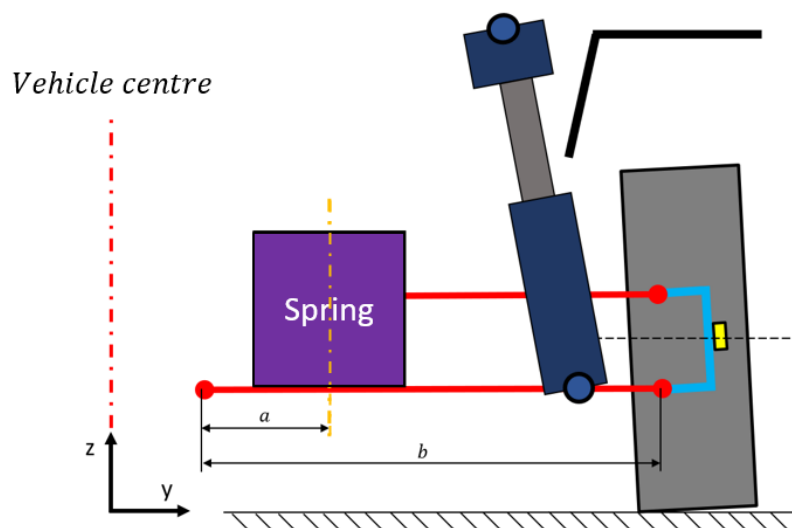


Figure 5.4: Scheme for the calculation of transmission ratio [50]

The measure of  $b$  is known, and it is the length of the lower axis of the suspension. The transmission ratio of the spring is given by the formula 5.2.

$$i = \frac{a}{b} \quad (5.2)$$

As previously done, the range is used to get the values of  $i$ , with a variation step of 0.01.

### 5.2.2 Geometrical check

This section considers only the critical case where the springs is mounted in Y direction with respect to the shock absorber (configuration Spring in Y, Figure 5.2).

To position the spring, the inner point of the damper piston must be computed to avoid interference. This can be done because in the dimensional chain, the spring is evaluated after the definition of the shock absorber.

To evaluate the point, the height of the spring (that is not known) is taken with a conventional value of 100 mm ( $h_{test}$ ). The value derives from an estimation of more empirical measurements of coils in loaded conditions. Since the shock absorber can have an inclination, the most internal point is the one at the top height of the spring.

The Y coordinate of the inner point (critical point) of the damper is given by the formula 5.3.

$$y_{inside\_surface\_sa} = y_{sa} - h_{test} \cdot \tan(\theta_{sa}) - \frac{1}{2} d_{sa} \cdot \cos(\theta_{sa}) \quad (5.3)$$

An easy visualization of the point is given in Figure 5.5 [50].

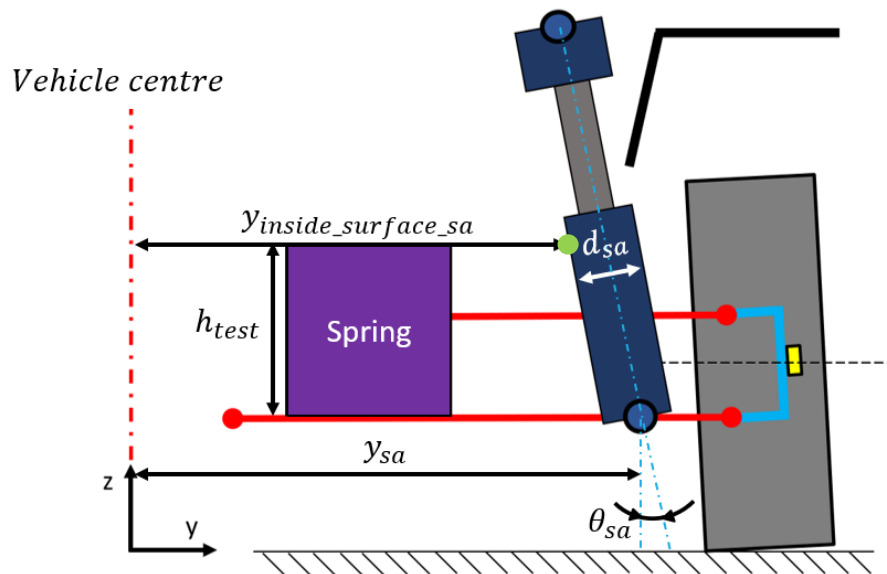


Figure 5.5: Shock absorber position and critical point (green dot) [50]

To avoid interference between the spring and the critical point of the shock absorber (green dot in Figure 5.5) there must be a safety distance between the two elements.

At the same time, the spring cannot be too far from the damper. The inner it is, the higher the loads on the spring are. Moreover, a more internal position, reduces the available space in Y direction for the battery pack and the luggage compartment.

The distance target between spring and damping element is set to 40 mm with an acceptable variation of 5%. The value is an estimation derived from an empirical data analysis. At this point, to check the distance, the external point of the spring must be calculated.

The information of the location of the spring axis is given by the transmission ratio. The inverse of the equation 5.2 is useful to get the value of  $a$ . It is the Y dimension of the position of the centre axis of the spring with respect to the lower bolt joint of the suspension.

The position of the spring axis is fixed for a determined transmission ratio. The outer coordinate, instead, depends on the mean and wire diameters. To compare the location of the coil with the position of the damper, it is computed with respect to the centre of the vehicle.

The equation is given by 5.4.

$$y_{coil\_out} = a + \frac{D + d}{2} + y_{fixed\_arm} \quad (5.4)$$

The scheme for the positioning of the coil is represented in Figure 5.6 [50].

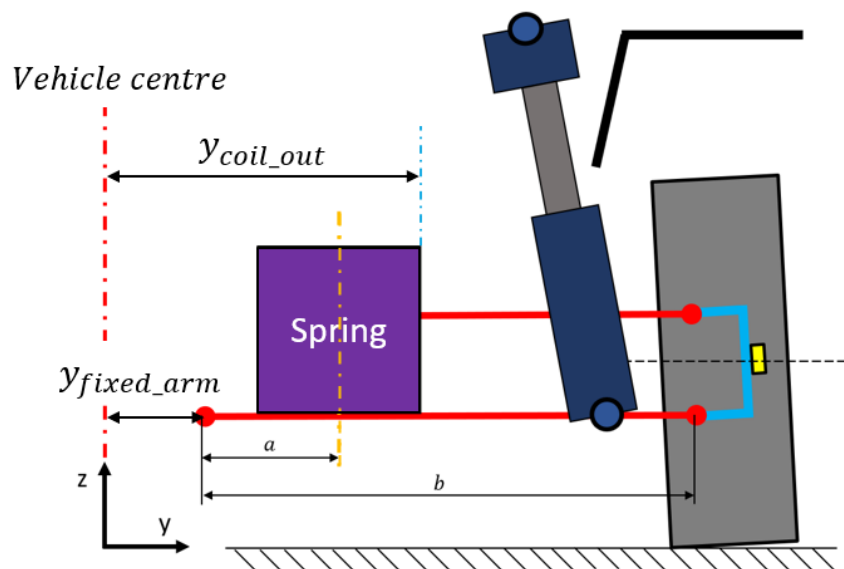


Figure 5.6: Spring positioning [50]

The computation is performed for each coil of the list, for every transmission ratio that gives the value of  $a$ .

At this stage, Y coordinate of the external point of the spring and internal point of the damper piston are known. The code computes the distance.

Only springs that give a difference in positioning in the range of 5% of 40 mm can be mounted. Coils with distance higher than 42 mm are too far from the damper. They must sustain higher loads and they reduce available space in Y direction. Coils with distance lower than 38 mm can give problems of interference with the other element.

The springs that are acceptable are maintained inside the coil catalogue.

At this point, only springs that can be mounted are in the catalogue. Then, the code adds a new parameter, the number of windings  $n$ . The script expands the list and associates each coil to every possible number of windings. The range of variation is taken from the database and rounded to slightly increase the interval. The  $n$  ranges from 3 to 7 with a step of 0.5 (only multiples of half turn are possible).

### 5.2.3 Frequency check

The second check is related to the oscillation properties. Specifically, it regards the frequency of the rear axle.

To compute the frequency, different factors are needed. Although the process regards an estimation with some simplification due to the reduced number of elements, different components are considered anyway. To know the natural frequency, a mass and a stiffness are necessary.

The value of the frequency is given by equation 5.5 [52, p. 325][50].

$$f = \sqrt{\frac{c \cdot 1000 \cdot 2}{m}} \cdot 9.55 \left[ \frac{1}{min} \right] \quad (5.5)$$

That is the natural frequency formula for oscillating mass. The factor of 1000 is used to convert the value of  $c$  from [N/mm] to [N/m], the value of 2 doubles the rate because the computations refer to the whole axis and the value 9.55 is necessary for the conversion in oscillations per minute. It is expressed with this unit for an easier visualization for the user.

Although the expression is simple,  $c$  and  $m$  consider different factors.

The rate  $c$  refers to the stiffness of the complete oscillating axis. It considers the presence of the main spring, the secondary elastic elements, and the tire. Moreover, the formulation also considers a different position of the elements that is expressed through the transmission ratio  $i$ . The formula gives a parallel connection between secondary elastic elements and primary spring and a series connection to the tire. It can be obtained with equivalent spring evaluation. The expression is reported in 5.6.

$$c = \frac{1}{\left( \frac{1}{c_{sec\_spring} + c_{spring} \cdot i^2} + \frac{1}{c_{tire}} \right)} \quad (5.6)$$

## 5 - Spring design for dimensional chain of the rear axle

Since the exact values of secondary elastic members and tire stiffnesses are not available, code assigns a representative value. The values are taken and kept constant for every coil with reference to internal design procedures. Table 5.4 reports these values.

Table 5.4: Conventional elastic rates

Stiffness	Value	Unit of measure
$C_{sec\_spring}$	12	N/mm
$C_{tire}$	300	N/mm

The transmission ratio is known, so the only missing element of equation 5.6 is the stiffness of the helical spring.

The rate of a helical coil compression spring can be computed from its properties. The formulation considers dimensional factors, the mean diameter, the wire diameter, and the number of coils, but also material factors, expressed through the shear modulus.

The equation 5.7 gives the explicit expression [51, p. 503][53, p. 16].

$$c_{spring} = \frac{G \cdot d^4}{8 \cdot D^3 \cdot n} \quad (5.7)$$

The spring rate expresses the ratio between the spring force and the deflection [54, pp. 321-322].

In this way, for each coil of the matrix, the spring rate can be computed. The value of shear modulus  $G$  is constant considering only steel spring with a value of  $G=80000$  MPa [55].

So, all elements of expression 5.6 are now known and the code can obtain the value of  $c$ .

To compute also the second variable of equation 5.5, different mass contributions are considered. The mass to be considered is the one of design condition. In this configuration, two occupants are considered. The weight of the occupants is evenly distributed between front and rear axle. Each of them has a mass of 68 kg [52, p. 343]. A 50 kg luggage is also accounted with a 110% distribution on the rear axle due to the leverage considered [52, p. 343][54, p. 36].

The mass is given by equation 5.8.

$$m = (m_{rear} - m_{axle}) + 0.5 \cdot (2 \cdot m_{passenger}) + 1.1 \cdot m_{luggage} \quad (5.8)$$

At this stage, the frequency can be computed for each coil of the matrix. The value of  $m$  is constant for every spring, while the value of  $c$  changes due to the variation of  $c_{spring}$ . Each element of the list has a different frequency that can be acceptable or not.

The acceptability range is expressed by equation 5.9.

$$60 \frac{1}{min} \leq f \leq 90 \frac{1}{min} \quad (5.9)$$



Only coils that satisfy the condition of 5.9 are maintained inside the matrix. Moreover, two new columns are added to show the value of the spring rate and the derived frequency. The new structure is reported in Table 5.5.

Table 5.5: Updated spring catalogue layout

	D [mm]	d [mm]	w [-]	i [-]	n [-]	$C_{spring}$ [N/mm]	f [1/min]
Coil 1	180.2	17	10.6	0.54	3	47.578	63.625
...	...	...	...	...	...	...	...
Coil n	85.2	12	7.1	0.65	7	47.897	75.848

The number of coils in the table reduces, so that inside the matrix only coils that succeeded the frequency test and the previous geometrical check remain. After that user gets a final message with information on the number of springs that passed the check.

## 5.2.4 Travel check

Finally, the code enters a third section for the control of the proper travel. Since the body is an hyperstatic structure, the suspension is a deformable system that allows to guarantee the proper touching with ground with all four contact points [4, p. 133].

To assure the correct working of the module, in every working condition, the suspension spring must push towards the ground. This means that even in the lowest position of the wheel, the spring must still be in compression to ensure a residual pushing towards the ground.

The test presented in this section controls if the spring in the most extended position is still in a compressed condition.

The deformation of the helical coil spring is composed by different contributions. At this stage, not all information is available, so an estimation of the deflection is necessary. The different deformation steps are reported in Figure 5.7 [51, p. 510].

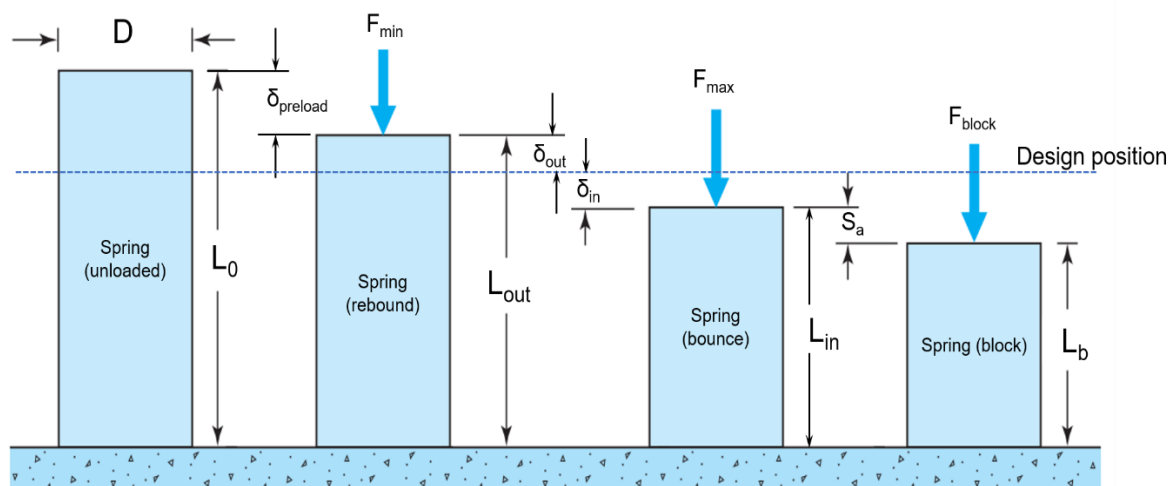


Figure 5.7: Spring deflections [51, p. 510]

The out displacement is set to a default value of 120 mm. The value is taken from empirical evaluations of existing data in the database. User can decide to slightly modify this value. The spring in design condition, is in an intermediate deflection between the minimum and maximum force. The target of 120 mm refers to the sum of the deflection due to the rebound with the addition of the preload deformation ( $\delta_{out} + \delta_{preload}$ ). The reference position is the unloaded spring length  $L_0$ . Due to the hypothesis of complete linearity, the relation of force and deformation is always constant and it is given by the spring rate.

The first operation the code performs is the computation of the acting force in steady conditions. The force on the spring can be obtained from the weight force on the wheel through the transmission ratio. The load considers the weight of the vehicle on the rear part reduced by the weight of the whole axle and of the wheel assembly.

The equation for the acting force is given by 5.10.

$$F_{weight} = \frac{(m_{rear} - m_{axle} - m_{tire}) \cdot g}{2} \quad (5.10)$$

The whole mass acts on both wheels, so the force on one single side is obtained dividing by 2. This is the force that acts on the wheel position. The necessary dimension is the one in spring position that can be obtained through the transmission ratio according to equation 5.11.

$$F_{spring} = \frac{F_{weight}}{i} \quad (5.11)$$

The force on the wheel is fixed for a defined vehicle while the force on the spring changes for each coil of the matrix due to the different values of  $i$ .

With the force and the spring properties, the deformation in loaded condition can be computed with 5.12 [51, p. 503] [53, p. 16].

$$\delta_{out} + \delta_{preload} = \frac{8 \cdot F_{spring} \cdot D^3 \cdot n}{G \cdot d^4} \quad (5.12)$$

The obtained deflection refers to the unloaded condition of the spring, so to the difference with the maximum length  $L_0$ . This value can be compared with the imposed target of 120 mm. A tolerance range of 1% around the design value is acceptable.

The script performs the control and maintains in the matrix only the coils that satisfy the condition. The tolerance range can be modified according to the necessity of the designer.

As previously mentioned, this section controls if the spring in rebound condition is still in compression, so that the travel is not too small. At the same time, the travel of the spring cannot be too high. This can lead to an excessive compression of the coils.

The script, then, shows a message to the user to inform about the number of coils that succeeded the travel test and are kept in the matrix.

### 5.2.5 Resistance check

At this stage, the code did not make any control about the structural properties of the spring. There are checks about the fitting on the axle, about the comfort properties associated to the oscillations of the mass and check for the proper suspension functionality in rebound condition. Now, the opposite movement must be controlled. In bounce the wheel moves toward the wheelhouse and gives a further compression to the coil.

In the maximum travel-in condition, the force on the coil is the highest. The spring must be able to work without damage in this limit case.

First, the highest position of the tire is needed. At this point of the process, wheelhouse and tire have already been defined. The position of the wheel arch is given by the travel-in of the wheel as reported in Figure 5.8 [50].

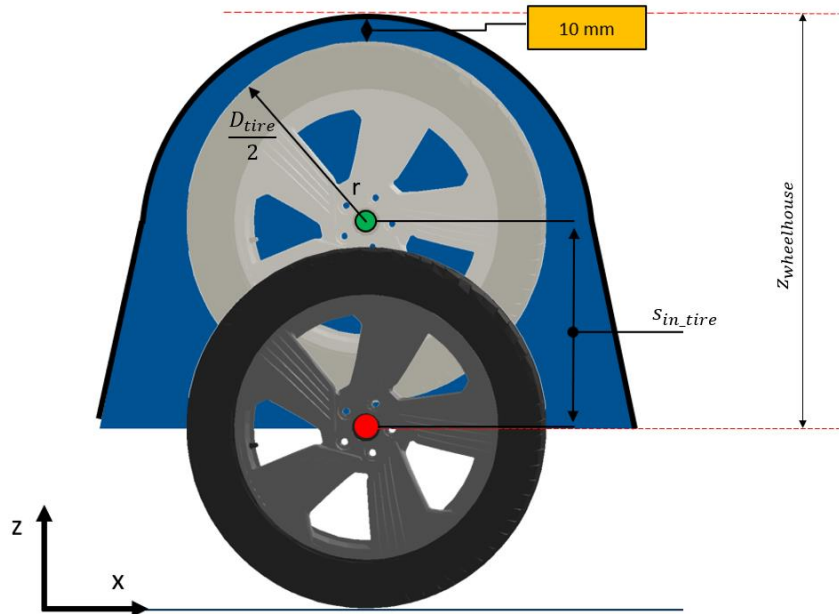


Figure 5.8: Wheel travel-in definition [50]

The top point of the wheel arch is defined from the centre of the wheel in steady condition with  $Z_{wheelhouse}$ . The envelope is set supposed to be at distance 10 mm with respect to the highest positioning of the biggest wheel that can be fitted on the axis. The value of 10 mm is a fixed parameter of the database.

Knowing the radius of the biggest tire and the wheel arch dimensions, the movement of the centre of the wheel is given by equation 5.13.

$$S_{in\_tire} = Z_{wheelhouse} - 10 - \frac{D_{tire}}{2} \quad (5.13)$$

The movement of the wheel causes a compression of the spring, that can be calculated given by the transmission ratio as reported in equation 5.14.

$$\delta_{in} = S_{in\_tire} \cdot i \quad (5.14)$$

The value of  $\delta_{in}$  deformation refers to the steady position of the wheel, so it gives the amount of additional compression given to the spring. From that value, according to the hypothesis of complete linearity in the whole range (page 73), the code can compute the supplementary force that adds to the basic condition (equation 5.15).

$$F_{in} = \delta_{in} \cdot c_{spring} \quad (5.15)$$

Adding this contribution to the previously computed force of the coil (equation 5.11), it gives the maximum value of the force the spring must sustain. The formula is reported in 5.16.

$$F_{max} = F_{in} + F_{spring} \quad (5.16)$$

The complete length of the wire of helical spring is subjected to torsion [51, p. 499]. The shear stress along the coil can be obtained from the spring properties according to equation 5.17 [51, p. 499].

$$\tau = \frac{8 \cdot F_{max} \cdot D}{\pi \cdot d^3} \quad (5.17)$$

The torque is distributed along a curved torsion bar. This causes a non-uniform gradient that is higher on the inside surface of the coil. The lower is the winding ratio, the higher is the severity of this effect [51, pp. 499-500]. Moreover, there is also a worsening factor due to the dynamic conditions in which the spring must work [53, p. 17].

For these reasons, a correction factor must be added, considering the winding ratio  $w$ . The correction factor  $k$  is given by the equation 5.18 [53, p. 14].

$$k = \frac{w + 0.5}{w - 0.75} \quad (5.18)$$

The equation for the computation of the shear stress becomes the one in 5.19 [53, p. 16].

$$\tau = k \frac{8 \cdot F_{max} \cdot D}{\pi \cdot d^3} \quad (5.19)$$

At this point, for each coil of the matrix, a different value of shear stress is associated. This value must not exceed the maximum allowable stress in the coil.

To perform the check, the code must compute the limit value. Being a control on the resistance, it is strictly related to the material of the spring. The limit value of the shear stress can be obtained by the relation with the maximum resistance of the material reported in 5.20 [51, p. 506] for steel compression coils with preload.

$$\tau_{lim} = 0.56 \cdot R_m \quad (5.20)$$

The value of  $R_m$  can be considered fix. For a more accurate analysis, different maximum resistances apply for different ranges of wire diameter. For each value, a different resistance limit can be taken looking at the reference table in [56, pp. 9-13]. The table is implemented in the code.

Anyway, for each coil of the matrix, the value of  $\tau_{lim}$  is computed. In this way, it can be compared to the previously calculated value with equation 5.19.

Only coils that give a shear stress lower than the limit value are suitable. The others would not sustain the load caused by the maximum bound of the wheel. The condition that coils must satisfy is the 5.21.

$$\tau \leq \tau_{lim} \quad (5.21)$$

The code gives information to the user about the number of coils of the matrix that succeeded the resistance test.

### 5.2.6 Buckling check

A helical spring loaded in compression can act as columns and the phenomenon of buckling can occur. It occurs especially for slender bodies, in which there is a predominant dimension along which the compression is applied. In some conditions, if the compression load exceeds a critical value, the result is an eccentric bending that is not restored by internal elastic moments of the material and the body collapses [57, p. 228]. This is buckling.

Since, as reported, this event causes collapsing of the element, it must be avoided.

To evaluate buckling condition, two different factors are necessary, as reported in Figure 5.9 [53, p. 21].

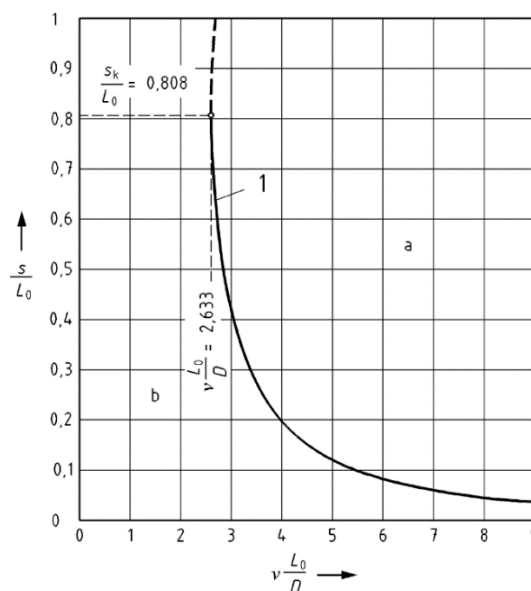


Figure 5.9: Buckling curve [53, p. 21]

The points that falls in the region “a” can lead to buckling. The “b” zone, instead, is a safe region. The two factors are the ones that characterize the X- and Y-axis.

On the X-axis, there is a term that is called “slenderness” factor ( $\lambda$ ). It is expressed according to 5.22 [53, p. 19].

$$\lambda = v \frac{L_0}{D} \quad (5.22)$$

It expresses a ratio between the main length of the body and its transversal dimension that in this specific case are the free length of the spring and the mean diameter. An additional correction factor  $v$  considers the constraint conditions at the extremities of the coil. According to normative [53, p. 20], the different values are reported in Figure 5.10.

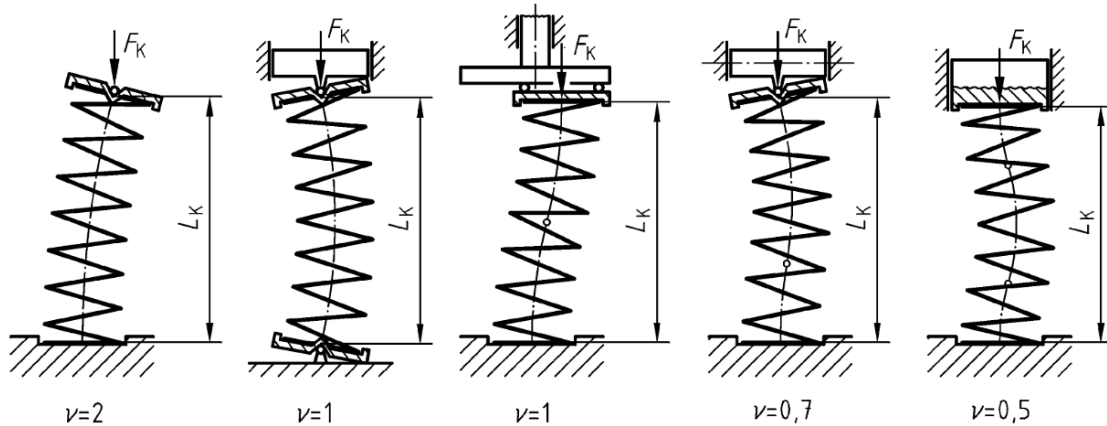


Figure 5.10: Constraint factors [53, p. 20]

In this project, the value is  $v=0.5$ .

On the Y-axis, the other factor considers the loading conditions. The dimension is called “retained spring deflection” ( $\xi$ ) and it is expressed with equation 5.23 [53, p. 19].

$$\xi = \frac{\delta}{L_0} \quad (5.23)$$

As visible, it is the ratio between the applied deformation and the whole length of the element. In the specific spring case, they are the maximum deformation (when the spring is fully compressed) and the unloaded length.

For each spring, the two presented dimensions must be evaluated.

The dimension of the free length of the coil appears in both formulas 5.22 and 5.23. To evaluate the length of the unloaded spring, the procedure begins with the block length. It is the limit condition in which all coils are in contact and the length is the minimum.

It is computed in different ways, according to the manufacturing procedure, as reported in [53, pp. 17-18].

For cold formed coils:

$$L_b = n_t \cdot d \quad (5.24)$$

$$n_t = n + 2 \quad (5.25)$$

For hot formed coils:

$$L_b = (n_t - 0.3) \cdot d \quad (5.26)$$

$$n_t = n + 1.5 \quad (5.27)$$

This is the minimum dimension the spring can have. This limit condition is never reached. There is always a certain safety margin to ensure that different coils never touch in operating condition. So, in the maximum deflected position, there is still a tolerance. The minimum distance is also computed according to the manufacturing procedure and it is reported in 5.28 and 5.29 [53, p. 17].

For cold formed coils:

$$S_a = \left( 0.0015 \cdot \frac{D^2}{d} + 0.1 \cdot d \right) \cdot n \quad (5.28)$$

For hot formed coils:

$$S_a = 0.02 \cdot (D + d) \cdot n \quad (5.29)$$

The spring in the maximum compressed condition is obtained with the wheel in the top height.

With respect to the design position, the travel-in of the spring is already available from equation 5.13. From this condition, also the deformation with respect to the unloaded length has already been computed. It is given by 5.7, recalling that it expresses the contribution of the out travel of the wheel and the preload.

With this procedure, going from the block length, all deformations are added to reach the unloaded length. The value can now be computed using 5.30 [50, p. 61].

$$L_0 = L_b + S_a + \delta_{in} + \delta_{out} + \delta_{preload} \quad (5.30)$$

For each coil, the retained spring deflection (equation 5.23) can be computed. Only contributions of the in-bound and out-bound travel are considered being the operative range of the spring (it does not reach the block length), as reported in 5.31.

$$\xi = \frac{\delta_{in} + \delta_{out}}{L_0} \quad (5.31)$$

Moreover,  $L_0$  is available so the slenderness factor can be also computed with 5.22.

At this point, the limit curve of Figure 5.9 is used to evaluate in which area the coil falls.

Only coils that give a couple of value below the curve can be maintained in the matrix. In opposite case, there is the risk of buckling so they cannot be used.

It is visible that two elements give contribution to buckling. One is the deformation of the spring: even very slender bodies can fall in safe region if the amount of deformation is small. The other is the distribution of the dimensions: if the body is little slender, also high amount of compressive deformation does not cause any problem.

At this stage, all checks are done, and the matrix contains only springs that succeeded all the different tests.

Depending on the different inputs, a different number of coils can be found by the discussed procedure. Anyway, starting from the list of thousands of springs, only few units remain. If the coil is only one, no further selection process is necessary. Instead, if there are more coils, one of them must be selected.

It is out of this work, but an idea is to use an optimization process to properly select the coil. For example, user, according to project tasks, can drive the procedure to reduce as much as possible the occupied space in Y direction to increase the available space for the battery pack or to select the lowest coil in Z direction, to reduce the attachment points.

The described procedure has many approximations and relies on simplified hypothesis, but, for the early development phase, is useful to get a first estimation of the rear axle spring, letting the tool the possibility to compute the dimensional chain for the parametric model.

### 5.3 Evaluation of results

The process gives a certain set of coils that has dimensions in the selected range of properties. It is sure that the low number of coils that are available succeeded all the tests and, according to the hypothesis, are suitable for the vehicle.

To evaluate results, different vehicles that are available in the database are taken. The set of models has the layout characteristics imposed for the geometrical check. The graphical evaluation represents the different properties of the spring, plotting the points of the real spring and the one obtained with the estimation process.

Being interested in the measures of the element, only dimensions are taken. Specifically, the wire diameter, the mean diameter, the number of windings, and the unloaded length. In this way, a graphical representation gives an idea if the real value falls near or inside the variation range of the obtained feature.

The plot is reported in Figure 5.11, taking as example the Toyota Corolla 2.0 Hybrid Collection (2019).



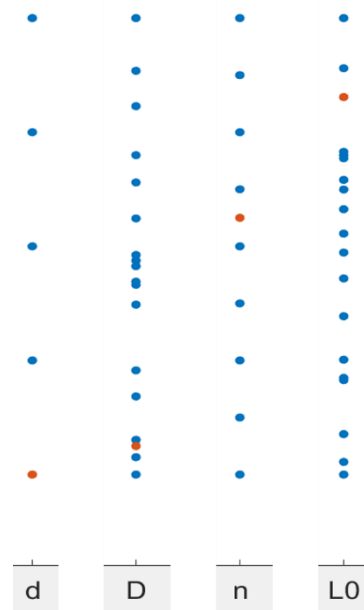


Figure 5.11: Dimensions evaluation

For each variable, the orange dot represents the value of the real spring of the vehicle taken from the database. The set of blue dots shows the different values of the variation of the coils in the matrix. The specific values on the Y-axis cannot be reported. They are part of a private set of data of A2Mac1 [25].

It is visible that, for matrices with more springs, the real value falls inside the variation range in most of the cases. When it is not, the real value is anyway near the selected range.

This procedure shows that the process is not perfect. It never gets the dimensions of the real spring. Nevertheless, the real values usually fall inside the variation range.

Moreover, it is important to notice that, even in the case in which selected coils do not contain the real dimension, the variation is in the order of units of millimetres for the wire diameter and some centimetres for the mean diameter.

The deviation causes an error of estimation of dimensions in the order of some tenths of millimetres. It is not a tolerable error for the design of the spring for the production, but for the early phase in which operations are done, it is a good estimation. The dimensional chain for the rear axle regards the whole width of the vehicle. The width of the car depends on the characteristics of the model, but it is in the order of 1700 mm.

This means that, taking the example of Figure 5.11, the maximum deviation is 70 mm for the diameter dimension. It causes an estimation error on the vehicle width that is quite small. The width of the vehicle of the example is of 1790 mm.

This is the maximum error caused by the procedure on the dimensional chain. The optimization process to select the coil can get the central values of the range, reducing the magnitude of the error.

In general, looking at the results, it is visible that the process generally selects springs with higher diameters, both for the wire and for the mean, and a lower free length. The computed coils are stiffer and bigger than the real one.

This is due to different approximations. The hypothesis of linear behaviour and perfectly cylindrical shape are far from the reality. As visible in Figure 5.12 [58], the coils have different diameters along the length and the distribution of windings is not regular.



Figure 5.12: Example of spring [58]

Moreover, computations are made also using some values that are set constant, but they are not always the same, as for example the elastic rate of the tire or the stiffness of the secondary bushing. Also, the position of the coils can vary. In the process, the spring is in Y direction, in the real case it can be slightly deviated and at a different distance than the imposed target.

The procedure, anyway, gives a method to select suitable coils that make sense and controls for different requirements. User can vary some parameters and try to find better results, if necessary. It can play on the variations range or the step inside the range. Try to change fix values of stiffness or resistances, vary the target values or tolerance ranges for the different checks.

In conclusion, as already mentioned, the procedure is satisfactory to get an estimation of the element dimensions, also considering the early phase of the development.

## 6 Conclusions and considerations

The main objective of the work was to create a tool that would allow the pre-processing phase to be carried out automatically.

The process started with the creation of the main tool to achieve the objective, i.e. the creation of the database. It was carried out keeping in mind the real objective that it had, which was to allow the automation of the process.

To do this, it was fundamental to establish rules for the structure and nomenclature of all the entities contained in it. It is very important that these structure rules are respected by everyone who acts on the database, so as not to compromise the success of the calculation process.

The result is a complex apparent structure, which is, however, very effective because of its modularity.

This feature has made it possible to create the MATLAB interface according to the objective of the work. The tool can interact with the database and to perform all the necessary operations to obtain the empirical models used by the main tool to calculate the parametric model of the vehicle. This interaction takes place in a completely autonomous way. The instructions are themselves written in the database and are read automatically.

This is a very important step, as no intervention is necessary. The pre-process phase can therefore be performed by any user, regardless of their level of knowledge of MATLAB or SQL. It does not have to perform any intervention, as long as all the rules of modularity and structure have been respected.

Moreover, by performing pre-process calculations every time the parametric model has to be obtained, empirical models based on the last saved version of the database can be obtained. It is only necessary to update the data progressively to ensure that the calculated empirical models are always current. Therefore, the only necessary operation is to update the data, which does not require any kind of programming language knowledge. Through the software interface, the fields are updated as in a normal spreadsheet. The only precaution is to always respect the established conventions.

The work has required different solutions to minimize any kind of intervention. The Project also offers interesting solutions to control the possible appearance of errors or unwanted variations in the database. In this way, it becomes more difficult to propagate empirical models that are incorrect over time. The user can at any time retrieve a previous version of the pre-process phase.

As far as the structure is concerned, the modularity is adequate for the purpose of the process. A possible solution for the future could be to divide regression models into several tables, with a categorization according to the type of data contained. In any case, no intervention is necessary on the MATLAB code, as it is designed in a modular way, regardless of the number of tables into which the database is divided.

Finally, the work done has made it possible to come into contact with data and design procedures of the different modules of the vehicle. The creation of the database has therefore gained the author in the knowledge of the numerical values associated with the different modules. Moreover, it has allowed to acquire and enhance knowledge of different programming languages for the management of the whole work, crucial for the engineering design.

## 7 List of Figures

Figure 1.1:	Flow of the general process.....	2
Figure 1.2:	Thesis structure .....	4
Figure 2.1:	Typical graphical output of the vehicle architecture tool.....	6
Figure 2.2:	Example of manikin dimensions [2][6] .....	9
Figure 2.3:	Histogram example of A44 dimension of the first row .....	10
Figure 2.4:	Leg dimensions [6].....	10
Figure 2.5:	Histogram plot with normal distribution of A44 dimension of the first row .....	13
Figure 2.6:	Calculated vs Real values .....	14
Figure 2.7:	Distribution of residuals represented as bar plot .....	18
Figure 2.8:	Flow of the process for linear regression models.....	19
Figure 2.9:	Structure of the database according to the ERM [2] .....	23
Figure 3.1:	Structure of database tables.....	26
Figure 3.2:	Example of internal manikin dimensions [6] .....	33
Figure 3.3:	Type of joins [46].....	42
Figure 3.4:	SQLiteStudio general interface.....	44
Figure 3.5:	Structure tab .....	45
Figure 3.6:	Data tab.....	45
Figure 3.7:	Column properties window .....	46
Figure 3.8:	Primary key configuration .....	47
Figure 3.9:	Foreign key configuration .....	47
Figure 3.10:	SQL query editor.....	49
Figure 3.11:	Query tab of views .....	50
Figure 4.1:	Flow chart of the MATLAB function to get model series IDs .....	54
Figure 4.2:	Flow chart of the MATLAB function to set NULL values .....	57
Figure 4.3:	Flow chart of the section to create the Database connection.....	59

Figure 4.4:	Flow chart of the section to assign fixed parameters .....	61
Figure 4.5:	Flow chart of the section to assign constant values from normal distribution .....	63
Figure 4.6:	Flow chart of the section to assign catalogues .....	64
Figure 4.7:	Flow chart of the section to assign empirical regression models .....	67
Figure 4.8:	Flow chart of the section to compare structures .....	69
Figure 5.1:	Rear axle reference frame [50] .....	71
Figure 5.2:	Possible Spring-damper layouts [50] .....	72
Figure 5.3:	Helical coil compression spring [51, p. 499] .....	75
Figure 5.4:	Scheme for the calculation of transmission ratio [50] .....	76
Figure 5.5:	Shock absorber position and critical point (green dot) [50] .....	77
Figure 5.6:	Spring positioning [50] .....	78
Figure 5.7:	Spring deflections [51, p. 510] .....	81
Figure 5.8:	Wheel travel-in definition [50] .....	83
Figure 5.9:	Buckling curve [53, p. 21] .....	85
Figure 5.10:	Constraint factors [53, p. 20] .....	86
Figure 5.11:	Dimensions evaluation .....	89
Figure 5.12:	Example of spring [58] .....	90
Figure 9.1:	Example of Excel report for normal distribution .....	97
Figure 9.2:	Example of Excel report for regression model .....	98
Figure 9.3:	Example of Excel report plots for regression model .....	99
Figure 9.4:	Report of the pre-processing structure .....	101

## 8 List of Tables

Table 2.1:	Examples of fixed parameters .....	8
Table 3.1:	Structure of model series table .....	28
Table 3.2:	Segment table .....	30
Table 3.3:	Structure of model table .....	30
Table 3.4:	Motor type table .....	32
Table 3.5:	Example data_airbag_driver table .....	34
Table 3.6:	Example data_tires table .....	34
Table 3.7:	Example data_door_panel_driver table .....	34
Table 3.8:	Structure of source table .....	36
Table 3.9:	Structure of calculation table for fixed parameters .....	37
Table 3.10:	Structure of calculation table for catalogues .....	38
Table 3.11:	Structure of calculation table for normal distributions .....	38
Table 3.12:	Structure of calculation table for regression models .....	40
Table 4.1:	Argument table (Fixed table) .....	61
Table 5.1:	Available inputs .....	74
Table 5.2:	Spring properties .....	75
Table 5.3:	Spring catalogue layout .....	76
Table 5.4:	Conventional elastic rates .....	80
Table 5.5:	Updated spring catalogue layout .....	81





# 9 Appendix

## 9.1 Appendix A

### Normal distribution report

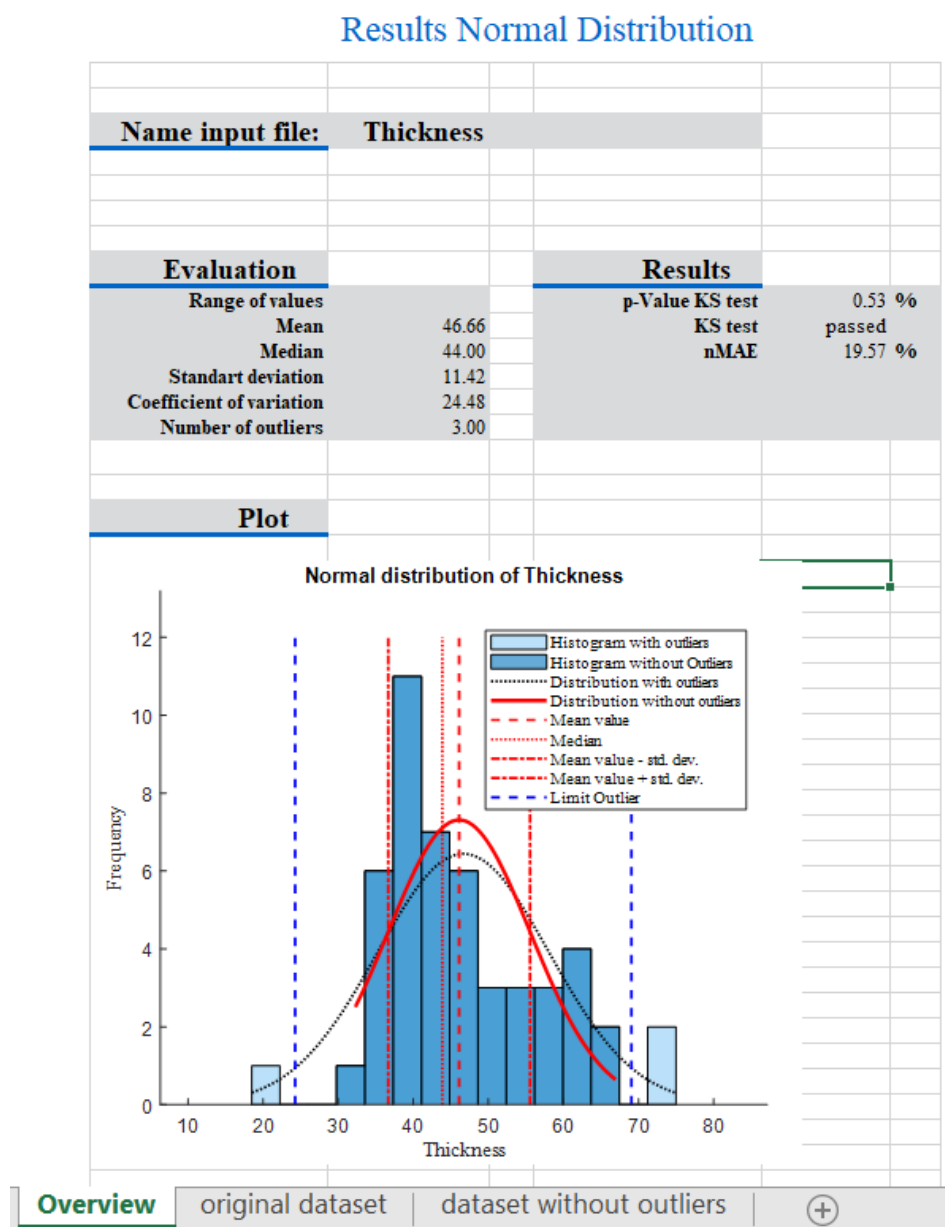


Figure 9.1: Example of Excel report for normal distribution

## Regression report

### Results Regression Analysis

Input			
Input File	Table		
Typ of regression function	linear		
Equation	1 + Top_Speed		
Dependent variable			
Name	Acceleration		
Range	4.7-14.4		
Term	Estimted coefficients	p-value t-Test in %	VIF
(Intercept)	25.870	5.5979E-101	-
Top_Speed	-0.081	9.59817E-72	1
Result			
Rsquared (%)	86.21	Critical Cook's distance	0.055089922
adj. Rsquared (%)	86.12	Robuste regression	Off
RMSE	0.81	p-Value F-Test in %	9.59817E-72
nRMSE (%)	8.62	Elements orgianl dataset	169
MAE	0.63	Number of outliers	3
nMAE (%)	6.64	OOST	On
OOST - MAE	0.63		
OOST - nMAE (%)	0.07		

Figure 9.2: Example of Excel report for regression model

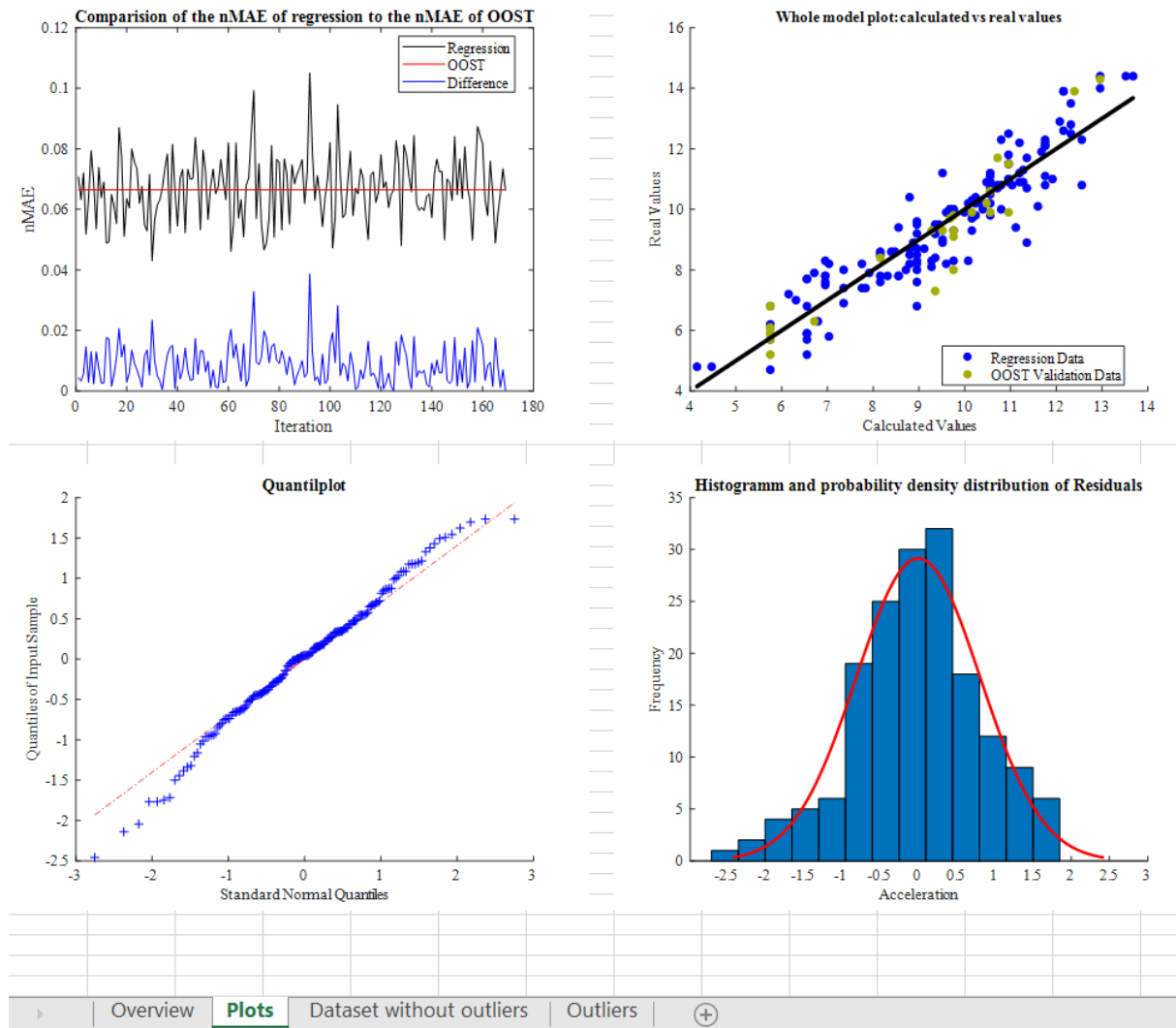


Figure 9.3: Example of Excel report plots for regression model



## 9.2 Appendix B

### Excel report of the pre-processing phase

	A	B	C
1	MATLABname	value	
2	Parameters.battery.battery_voltage_min		
3	Parameters.battery.battery_voltage_max		
4	Parameters.battery.cell.dimensions_x.cylindrical		
5	Parameters.battery.cell.dimensions_x.pouch		
6	Parameters.battery.cell.dimensions_x.prismatic		
7	Parameters.battery.cell.dimensions_y.pouch		
8	Parameters.battery.cell.dimensions_y.prismatic		
9	Parameters.battery.cell.dimensions_z.cylindrical		
10	Parameters.battery.cell.dimensions_z.pouch		
11	Parameters.battery.cell.dimensions_z.prismatic		
12	Parameters.battery.cell_energy_density_grav.cylindrical		
13	Parameters.battery.cell_energy_density_vol.cylindrical		
14	Parameters.dimensions.EX.battery_front_axle_min		
15	Parameters.dimensions.EX.battery_rear_axle_min		
16	Parameters.dimensions.EZ.pass_compartement_battery		
17	Parameters.dimensions.EZ.tunnel_battery		
18	Parameters.dimensions.EY.tunnel_battery		
19	Parameters.dimensions.CZ.batt_cooling		
20	Parameters.dimensions.CZ.batt_top_cover		
21	Parameters.dimensions.CZ.batt_bottom_cover		
22	Parameters.dimensions.GZ.H156.flatfloor		
23	Parameters.dimensions.GZ.H156.highfloor		
24	Parameters.dimensions.CX.min_offset		
25	Parameters.dimensions.rear_axle.EX.wheel_wheelhouse		
26	Parameters.dimensions.rear_axle.EY.wheel_wheelhouse		
27	Parameters.dimensions.rear_axle.EZ.wheel_wheelhouse		
28	Parameters.dimensions.rear_axle.MP.middle_camber_angle		
29	Parameters.dimensions.rear_axle.MP.deflected_camber_angle		
30	Parameters.dimensions.rear_axle.MP.deflected_camber_angle		
	Constant_values	Norm_distr_with_outliers	Normal_distr_no_outliers

	A	B	C	D	E	F	G	H	I
1	MATLABname	mean	std dev	median	variance	KS_test	p_KS_test	MAE	nMAE
2	Parameters.battery.cell_energy_density_grav.pouch								
3	Parameters.battery.cell_energy_density_grav.prismatic								
4	Parameters.battery.cell_energy_density_vol.pouch								
5	Parameters.battery.cell_energy_density_vol.prismatic								
6	Parameters.battery.cell_voltage								
7	Parameters.dimensions.EZ.H30_1.Hatchback								
8	Parameters.dimensions.EZ.H30_1.SUV								
9	Parameters.dimensions.EZ.H30_1.Sedan								
10	Parameters.dimensions.EZ.H30_2.Hatchback								
11	Parameters.dimensions.EZ.H30_2.SUV								
12	Parameters.dimensions.EZ.H30_2.Sedan								
13	Parameters.dimensions.EZ.H61_2.Hatchback								
14	Parameters.dimensions.EZ.H61_2.SUV								
15	Parameters.dimensions.EZ.H61_2.Sedan								
16	Parameters.dimensions.EZ.H62.Hatchback								
17	Parameters.dimensions.EZ.H62.SUV								
18	Parameters.dimensions.EZ.H62.Sedan								
19	Parameters.dimensions.EY.W20								
20	Parameters.dimensions.EY.seat_f_door								
21	Parameters.dimensions.rear_axle.CX.piston_diameter_r_five_link								
22	Parameters.dimensions.rear_axle.CX.piston_diameter_r_max_weig								
23	Parameters.dimensions.rear_axle.CX.piston_diameter_r_sword_arr								
24	Parameters.dimensions.rear_axle.CX.piston_diameter_r_torsion_be								
25	Parameters.dimensions.rear_axle.CX.piston_diameter_r_torsion_be								
26	Parameters.dimensions.rear_axle.CX.piston_diameter_r_trapezoidi								
27	Parameters.dimensions.rear_axle.CX.upper_bearing_diameter_sho								
28	Parameters.dimensions.rear_axle.CY.axis_length_rear_five_link								
29	Parameters.dimensions.rear_axle.CY.axis_length_rear_sword_arm								
30	Parameters.dimensions.rear_axle.CY.axis_length_rear_sword_arm								
	Constant_values	Norm_distr_with_outliers	Normal_distr_no_outliers	Regression					

Figure 9.4: Report of the pre-processing structure

The values are hidden for secrecy reasons.



## 9.3 Appendix C

### Explanation of the function to apply filters

Through a “switch” command, it contains different blocks for every filter type. In this way, it performs different operation according to the selected category.

The working principle is similar for each block. For “Filter out” and “Consider only” sections, the code checks if the values for the category are more than one. There is the possibility to consider only or exclude more than one value. The convention is that, when more than one element is used for filtering, they must be separated by “;”. This symbol is useful for the automatization of this step. The code splits the string contained in the cell whenever it finds the “;” and it knows if there are more values. Then for each category, it adds a condition to the WHERE statement of the query.

For “Higher” and “Lower” filters instead, only one value can be present, and it is a numerical value. The query condition simply contains the control to take data that are “>” or “<” than the selected one.

Finally, for “Max” and “Min” filters, as already discussed, they do not give a real data set but a single value only, the maximum or minimum of the column. The script does not add a WHERE condition, but it adds the needed command before the column name.

If the filter type does not fall in one of the presented blocks, the code gives a feedback telling the user that the type of filter is not implemented yet and no data can be loaded. In this way, the operator can immediately check if this is due to an error in writing the name or if there is the need to add a new type of filter. This second case is the only one in which a correction on the MATLAB code is needed and it requires some knowledge.





# 10 References

- [1] L. Nicoletti, S. Mayer, M. Brönnert, F. Schockenhoff, and M. Lienkamp, "Design Parameters for the Early Development Phase of Battery Electric Vehicles," *WEVJ*, vol. 11, no. 3, p. 47, 2020, doi: 10.3390/wevj11030047.
- [2] L. Nicoletti, W. Schmid, and M. Lienkamp, Eds., *Databased Architecture Modeling for Battery Electric Vehicles*, 2020.
- [3] A. Kampker, D. Vallée, and A. Schnettler, *Elektromobilität*. Berlin/Heidelberg, Germany: Springer, 2018.
- [4] G. Genta and L. Morello, *The automotive chassis*. Dordrecht: Springer, 2009.
- [5] M. Felgenhauer, C. Angerer, R. Marksteiner, F. Schneider, and M. Lienkamp, Eds., *Geometric substitute models for efficient scaling of dimensions during vehicle architecture design*, 2018.
- [6] L. Nicoletti, S. Mirti, F. Schockenhoff, A. König, and M. Lienkamp, "Derivation of Geometrical Interdependencies between the Passenger Compartment and the Traction Battery Using Dimensional Chains," *WEVJ*, vol. 11, no. 2, p. 39, 2020, doi: 10.3390/wevj11020039.
- [7] *H-Point Machine (HPM-II) Specifications and Procedure for H-Point Determination: SAE J4002*, 2010.
- [8] *Motor vehicle dimensions: J1100*, 2009.
- [9] *Devices for Use in Defining and Measuring Vehicle Seating Accommodation: SAE J826*, 2015.
- [10] A. Krohn, "Designing Geometric Substitute Models for Automated Concept Development," Master Thesis, Automotive Technology, Technical University of Munich, Munich, 2017.
- [11] *ECE R 125*, 2010. [Online]. Available: <https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A42010X0731%2803%29>
- [12] S. Mirti, "Erstellung eines parametrischen Modells zur Herleitung von Maßkonzepten für Elektrofahrzeuge [Creation of a parametric model to derive dimensional concepts for electric vehicles]," Automotive technology, Technical University of Munich, Munich, 2019.
- [13] M. Hoppert, "Analytische und experimentelle Untersuchungen zum Wirkungsgradverhalten von Achsgetrieben [Analytical and experimental investigations on the

- efficiency behaviour of axle drives],” Dissertation, Technical University of Ilmenau, Ilmenau, 2015.
- [14] H. Tschöke, *Die Elektrifizierung des Antriebsstrangs [The electrification of the powertrain]: Basiswissen*. Wiesbaden: Springer Vieweg, 2015.
- [15] *Directive 2006/126/EC*, 2006. [Online]. Available: <https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A32006L0126>
- [16] G. Casella and R. L. Berger, *Statistical inference*, 2nd ed. Pacific Grove, CA: Brooks/Cole, 2002.
- [17] G. Barbato, G. Genta, and A. Germak, *Misurare per decidere: Misure e statistica di base*, 3rd ed. Bologna: Società Editrice Esculapio, 2014.
- [18] K. Lagemann, *Function\_Normal\_Distribution*.
- [19] J. H. Zar, *Biostatistical analysis*, 4th ed. Upper Saddle River, N.J.: Prentice Hall; London : Prentice Hall International (UK), 1999.
- [20] B. Everitt, *The Cambridge dictionary of statistics*. Cambridge: Cambridge University Press, 1998.
- [21] D. C. Montgomery and G. C. Runger, *Applied statistics and probability for engineers*. Hoboken NJ: John Wiley and Sons Inc, 2014.
- [22] J. K. Patel and C. B. Read, *Handbook of the normal distribution*, 2nd ed. New York: M. Dekker, 1996.
- [23] J. A. Rice, *Mathematical statistics and data analysis*. [New Delhi]: Cengage Learning/Brooks/Cole, 2007, reimp. 2014.
- [24] A. C. Rencher and W. F. Christensen, *Methods of multivariate analysis*. Hoboken New Jersey: Wiley, 2012.
- [25] A2Mac1, *A2Mac1 catalogue*. [Online]. Available: <https://portal.a2mac1.com/>
- [26] G. B. Wetherill, *Intermediate statistical methods*. London: Chapman and Hall, 1981.
- [27] W. Mendenhall, T. Sincich, and W. S. c. i. b. s. Mendenhall, *A second course in statistics: Regression analysis / William Mendenhall, Terry Sincich*, 5th ed. Upper Saddle River, N.J.: Prentice Hall; London : Prentice-Hall International, 1996.
- [28] MathWorks, *Cook's Distance*. [Online]. Available: <https://www.mathworks.com/help/stats/cooks-distance.html>
- [29] K. Lagemann, *Function\_Regression\_Analysis*.
- [30] N. R. Draper and H. Smith, *Applied regression analysis*, 3rd ed. New York, Chichester: Wiley, 1998.
- [31] A. J. H. Hallet and J. Marquez, *Henri Theil's Contributions to Economics and Econometrics: Econometric Theory and Methodology*. Springer Netherlands, 1992.

- 
- [32] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning: With applications in R*. New York: Springer, 2013.
  - [33] M. H. Kutner, C. Nachtsheim, and J. Neter, *Applied linear regression models*, 4th ed. Boston, New York: McGraw-Hill/Irwin, 2004.
  - [34] Statistics Solutions, *Testing Assumptions of Linear Regression*. [Online]. Available: <https://www.statisticssolutions.com/testing-assumptions-of-linear-regression-in-spss/#:~:text=In%20order%20to%20make%20valid,variable%20and%20the%20predicted%20value>.
  - [35] R. G. Lomax, *Statistical concepts: A second course*, 3rd ed. Mahwah N.J.: Lawrence Erlbaum Associates, 2007.
  - [36] R. L. Wasserstein and N. A. Lazar, "The ASA Statement on p -Values: Context, Process, and Purpose," *The American Statistician*, vol. 70, no. 2, pp. 129–133, 2016, doi: 10.1080/00031305.2016.1154108.
  - [37] D. Colquhoun, "The reproducibility of research and the misinterpretation of p-values," *Royal Society open science*, vol. 4, no. 12, p. 171085, 2017, doi: 10.1098/rsos.171085.
  - [38] P. P.-S. Chen, "The entity-relationship model toward a unified view of data," *ACM Transaction Database System*, vol. 1, pp. 9–36, 1976.
  - [39] Wikipedia, *Entity–relationship model*. [Online]. Available: [https://en.wikipedia.org/wiki/Entity–relationship\\_model#:~:text=An%20entity–relationship%20model%20\(or,instances%20of%20those%20entity%20types\)](https://en.wikipedia.org/wiki/Entity–relationship_model#:~:text=An%20entity–relationship%20model%20(or,instances%20of%20those%20entity%20types)).
  - [40] M. Unterstein and G. Matthiessen, *Relationale Datenbanken und SQL in Theorie und Praxis*. Berlin/Heidelberg, Germany: Springer, 2012.
  - [41] ADAC, *ADAC catalogue*. [Online]. Available: <https://www.adac.de/rund-ums-fahrzeug/autokatalog/>
  - [42] Wikipedia, *SQL*. [Online]. Available: [https://en.wikipedia.org/wiki/SQL#cite\\_note-oed-US-6](https://en.wikipedia.org/wiki/SQL#cite_note-oed-US-6)
  - [43] Microsoft, *Structured Query Language (SQL)*. [Online]. Available: <https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?redirectedfrom=MSDN&view=sql-server-ver15>
  - [44] SQLiteStudio, *SQLiteStudio Download*. [Online]. Available: <https://www.sqlite.org/download.html>
  - [45] w3schools, *SQL Tutorial*. [Online]. Available: <https://www.w3schools.com/sql/default.asp>
  - [46] w3schools, *Different Types of SQL JOINS*. [Online]. Available: [https://www.w3schools.com/sql/sql\\_join.asp](https://www.w3schools.com/sql/sql_join.asp)
  - [47] MathWorks, *Database Explorer*. [Online]. Available: <https://www.mathworks.com/help/database/ug/databaseexplorer-app.html>

- [48] MathWorks, *struct*. [Online]. Available: <https://www.mathworks.com/help/matlab/ref/struct.html>
- [49] MathWorks, *Function Handles*. [Online]. Available: <https://www.mathworks.com/help/matlab/function-handles.html>
- [50] M. Spreng, “Maßkettenanalyse am Hinterwagen zur Erstellung von Ersatzmodellen [Dimensional chain analysis on the rear end of the vehicle to create replacement models],” Bachelor thesis, Automotive technology, Technical University of Munich, Munich, 2020.
- [51] R. C. Juvinall and K. M. Marshek, *Fundamentals of machine component design*, 5th ed. Hoboken NJ: John Wiley & Sons, 2012.
- [52] J. Reimpell and J. W. Betzler, *Grundlagen: Fahrwerk und Gesamtfahrzeug, Radaufhängungen und Antriebsarten, Achskinematik und Elastokinematik, Lenkanlage, Federung, Reifen, Konstruktions- und Berechnungshinweise*, 4th ed. Würzburg: Vogel Buchverlag, 2000.
- [53] *Zylindrische Schraubenfedern aus runden Drähten und Stäben – Berechnung und Konstruktion – Druckfedern [Cylindrical helical springs made from round wire and bar – Calculation and design – Compression springs]: DIN EN 13906-1*, 2013.
- [54] H. Wittel, D. Jannasch, J. Voßiek, and C. Spura, *Roloff/Matek Maschinenelemente: Normung, Berechnung, Gestaltung*, 24th ed. Wiesbaden: Springer Gabler. in Springer Fachmedien Wiesbaden GmbH, 2019.
- [55] R. R. Archer and T. J. Lardner, *An introduction to the mechanics of solids*, 2nd ed. New York, London: McGraw-Hill, 1978.
- [56] *Stahldraht für Federn – Patentiert gezogener unlegierter Federstahldraht [Steel wire for mechanical springs – Patented cold drawn unalloyed spring steel wire]: DIN EN 10270-1*, 2017.
- [57] R. C. Juvinall and K. M. Marshek, *Juvinall's fundamentals of machine component design: SI version / Robert C. Juvinall, Kurt M. Marshek*. Hoboken, New Jersey: John Wiley & Sons, 2017.
- [58] Car-Bock Automotive parts, *Coil spring rear axle*. [Online]. Available: <https://www.car-bock.de/Coil-spring-rear-VW-Passat-3B-HD>