Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

José Nicolás Espinoza Guzmán

Study on the Performance of Machine Learning Algorithms as Surrogate Models for a Representative Model

Master's thesis in RAMS
Supervisor: Shen Yin
Co-supervisor: Siegfried Eisinger
June 2023

NTNU
Norwegian University of Science and Technology

RAMS
Reliability, Availability, Maintainability, and Safety

José Nicolás Espinoza Guzmán

Study on the Performance of Machine Learning Algorithms as Surrogate Models for a Representative Model

Master's thesis in RAMS
Supervisor: Shen Yin
Co-supervisor: Siegfried Eisinger
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Preface

This thesis is conducted as the final work to achieve the Master's degree in Reliability, Availability, Maintenance and Safety (RAMS) given by the Department of Mechanical and Industrial Engineering of NTNU in Norway. It represents the culmination of interesting research, analysis, and reflection in pursuit of a deeper understanding of how machine learning can be used as surrogate models in production environments.

This thesis journey started with a topic proposed by DNV Group AS with Siegfried Eisinger in contact with the supervisor Shen Yin from the RAMS department at NTNU. The initial step was a literature review and research related ended up in my specialization project called "Study on AI-based surrogate modeling technique and its efficiency in a production environment" delivered in December 2022. It continued with a definition of the research followed by a review of more academic papers and relevant case studies, intellectual discourse with mentors, and the performance of different tests and experiments with programming. The objective was not to engage in an exhaustive review of every aspect of the subject but rather to identify key elements and present them concisely and coherently.

This thesis work aims at readers who have background knowledge or understanding of system modeling and simulations as well as basic knowledge and interest in statistics and machine learning and want to apply it to implement in a real productive environment or anything similar.

José Nicolás Espinoza Guzmán

Trondheim, June 2023

Acknowledgment

I would like to take this opportunity to express my deepest gratitude to all those who have contributed to the completion of this master's thesis. Their support, guidance, and encouragement have been instrumental in my academic journey, and I am profoundly indebted to their contributions.

First, I am immensely grateful to my thesis supervisor and co-supervisor, Shen Yin and Siegfried Eisinger, for their constant guidance, expertise, and availability throughout this long journey. Their mentorship has influenced my thinking, challenged my ideas, and pushed me to reach new heights. I am truly fortunate to have had the opportunity to work with such a dedicated and inspiring team.

Lastly, I would like to thank my family, friends, and colleagues that have also helped with ideas, support, shared experience, and motivation when needed. These key contributors have, to a certain extent, made the realization of this project possible and as a consequence, contributed to the future development of my career.

J.N.E.

Executive Summary

Nowadays, in the engineering areas, modeling and simulation are widely used for productive and research purposes. They represent a very important discipline, especially in the scientific and design fields. The complexity of the models and their simulation has surpassed the physical capacity of computers to perform such requirements in an acceptable time. A possible solution for this problem is the development of simpler models for the already complex models. This answer is called surrogate models or metamodels and represents another abstract layer over reality. These surrogate models act like "black-box" functions, which imitate the outcomes based only on inputs. The approach is data-driven so it matches perfectly with the development of machine learning since those methodologies synergize harmoniously.

The objective of this thesis is to evaluate the performance of using Machine learning algorithms as surrogate models for a simple bouncing ball model. The results can be used as a guide for future research regarding the use of these techniques in production environments. All the concepts related to Modeling, Surrogate Modeling, and Machine Learning are reviewed and presented in the theoretical background in addition to other topics needed to understand the research completely.

The research methodology was based on the classical approach and the Design of Experiments steps were followed to establish the experiments. Five different Machine Learning algorithms were chosen to train surrogate models. Support Vector Machines, Kernel Ridge, Decision Tree, K-Nearest Neighbors, and Gaussian Process. Each type of these techniques was used to train multiple models varying the hyperparameter selection and comparing the results with two different setups. Then, after selecting the best representative of every type, they were measured on the performance of solving a specific task: Finding the angle that maximizes the horizontal distance traveled by the ball before the first bounce.

The results show how different the different algorithms create the surrogate models to mimic the original model behavior. In this opportunity, Support Vector Machine models did not perform well but all the others did. When evaluated on the range of the same parameters as trained the models were very accurate, particularly the Gaussian Process model with very impressive results. The Kernel Ridge model consistently presented accurate predictions and behavior, becoming one of the most robust models. K-Nearest Neighbors and Decision Tree models showed also good results, with the particularity that the out function was similar to a step function in mathematics, with multiple values giving the same results, which can be very practical when using these techniques to narrow down possibilities of possible simulations. Overall, the results demonstrate that Machine Learning algorithms can be a powerful tool to be used as surrogate models and can assist expensive models in their task as predictors. Of course, more experiments and research are needed and some recommendations are suggested to continue this research.

Contents

Chapter 1

Introduction

This introductory section presents the context, background, and motivation for this thesis problem, followed by a description of the objectives pursued. In addition, the work's scope and limitations are presented, as well as the approach used to solve the objectives. Lastly, a structural overview of the complete document is given.

1.1 Background

The massive use of computer simulations caused a revolution in all development fields, especially in engineering and design. Bundling implementations of material, mechanical and physical properties into a software package, simulating the desired aspects of a system, and performing the tests and experiments virtually greatly decreased the cost of researching and increased the efficiency of the outcomes. This radical decrease in the number of prototypes or physical experiments needed to be able to validate any new ideas in conjunction with the possibility of simulating thousands of attempts with a small fraction of the resources invested was a major achievement accomplished in the 90s. However, their computational cost grew enormously as simulation software became more accurate and gained precision over the last years. This growth of the computational virtual cost was, in fact, faster than the growth of the physical power of computers and, in consequence, produced very slow and lengthy simulations for very detailed and powerful models, even in high-performance environments (regarding computational power). These results are due mainly to the never-ending drive for finer time scales, more detail, and general algorithmic complexity. For instance, the simulation of climate situations can take several hours to complete [7], and a computational fluid dynamics (CFD) simulation of a cooling system can take days to complete [18], not even mention particle simulations in the field of micro and nanomechanics. Considering all the previous context, nowadays we face a new type of problem. The use of this computationally expensive simulation for evaluation with intensive tasks such as optimization and sensitivity analysis has become impractical and should be avoided. The actual demand for virtual experiments for modern product development processes can get out of control due to fine resolution and detail and to counter this enormous cost, an additional layer of abstraction between the complex system in the real world and the engineer was proposed, more specifically between the simulation and the engineer. Rather than interacting directly with the simulator, a cheaper approximation is constructed. The possible solution is surrogate models, meta-models, or emulators [49].

Surrogate models offer an enticing data-driven approach to achieve analogous objectives as simulations in various applications, such as design space exploration, optimization, visualization, or sensitivity analysis, while upholding predetermined quality standards and requirements. The fundamental concept behind surrogate modeling involves constructing a globally accurate model across the entire design space, devoid of any involved parameters, while abstaining from explicit consideration of the underlying phenomena being represented, often called black-box functions, because there is no interest in knowing what happens inside the model but instead, the focus primarily lies on capturing the relationships between inputs and outputs. Surrogate models or metamodels can be defined as the process of building mathematical models capable of capturing the input/output relationship of the simulation (or a previous model), generating predictions for the value of the response functions faster than simulation models [46]. Therefore, the metamodels seek to establish a cause-and-effect relationship between the input and output variables of the model. At this point is when the advances in artificial intelligence come to assist the generation of these kinds of solutions. In particular, with the machine learning techniques and methods, since they are built as data-driven approaches just the same way surrogate models are. The best of using machine learning is that the surrogate models produced are not expensive at all. The whole process of producing a good surrogate model is not an easy task by the results are promising. There are many examples of using machine learning for this specific proposes, but due to the multiple options existing these days, there is still not clear how these methods perform generally, that is the reason to start with this work and conduct a review of some applications to inspire a future work for a project master thesis.

1.2 Objectives

The main objective of this master's thesis is to investigate the performance of existing machine learning algorithms to generate surrogate models for a simple model, as a representation of any generic productive model. These surrogate models are a simple, neat, and cheap option when detailed models, simulations, or experimentation are very expensive. The specific objectives set to achieve the main objective can be summarized as follows:

1. To clearly explain the key concepts of modeling and surrogate modeling.

2. To research Machine Learning algorithms and their use as surrogate models.
3. To train surrogate models for a simple model using different Machine Learning algorithms.
4. To measure the general performance of these surrogate models.
5. To establish guidelines and recommendations for the use of these algorithms in real applications

1.3 Approach

The thesis continues a specialization report called *"Study on AI-based surrogate modeling technique and its efficiency in a production environment"*. The theoretical background is a summary of that work that is based on an extensive literature review covering all the necessary topics. Thus, some of the sections are taken from there. The sources used to assess the literature review were books, articles, scientific papers, and, websites found with the engines such as ORIA, Scopus, Science Direct, and Google Scholar.

The following research is an experiment performed in Python using a model developed by the co-supervisor of this thesis, Siegfried Eisinger, from the company DNV. All the codes are provided in the appendix and were developed by me to achieve this thesis's objectives.

1.4 Scope & Limitations

The scope of the basic concepts of modeling and surrogate modeling are thoroughly reviewed since they are the underlying base for the whole purpose of this study. The concepts and the process are exposed even though is expected that the reader may have some basic knowledge about it. On the other hand, Artificial Intelligence and Machine Learning concepts and topics are only presented as their basis and mostly bounded to the specific matters of this thesis.

The research and experiment limitations come from the construction of the surrogate models. This is not a study on how to properly train and fit models but instead, find general lines, patterns, or such to evaluate the performance of some of these techniques applied in the constructions of these emulators. Also, the number of techniques used, the performance metrics, and the experiments' ideas are narrow and, therefore, susceptible to being replicated, complemented, or improved in further research and experiments.

1.5 Outline

This section serves as a quick guide for the thesis. It highlights the main sections and subsections that are covered, giving a clear understanding of the flow and progression of this work. The following is the outline of this thesis.

- **Chapter 1 - Introduction:** This chapter presents the background and context of this thesis in conjunction with the objectives and limitations.
- **Chapter 2 - Theoretical Background:** The state of the art is presented in this chapter. The necessary knowledge of the topics of interest for the complete understanding of this work is presented, mainly structured in modeling, machine learning, and some complementary subjects.
- **Chapter 3 - Research:** The chapter presents the specific work and experiments. Describing the model of study, the problem, and the methodology used to tackle it.
- **Chapter 4 - Results:** This chapter shows the results obtained through the experimentation. It is divided into two experiments, and the machine learning technique groups the results.
- **Chapter 5 - Conclusions:** This is the last chapter of the thesis. It summarizes the work done and the conclusions withdrawn from it. Also, a section dedicated to the discussion and recommendations for further work is added.
- **Appendix and Bibliography:** As with any other research, in the end, all the references used and complementary information, such as code lines, are presented.

Chapter 2

Theoretical background

This chapter presents a series of concepts, methodologies, and a literature review of matters necessary to know and understand to explore this thesis further. Some of the topics reviewed are strongly limited or simplified to give only the necessary information and approach that this study concerns and, therefore, should be considered this work objective as context for the use of certain definitions outside this environment.

The chapter starts with the theory related to modeling to understand the following, and more specifically, the topic of surrogate modeling. It continues with a small review of Artificial Intelligence, machine learning, and examples of the different techniques used to create the surrogate models. Finally, complementary topics are presented to provide the necessary information and context to cover all actions performed in this research fully.

2.1 Modeling Theory

Modeling is an integral and indispensable component of all intellectual tasks. In simple terms, scientific models serve as representations of reality, adaptable to varying levels of complexity and detail. Depending on the context, these models are often described as simplifications or abstractions of reality. In different fields, formal definitions of "modeling" and "model" may vary to align with specific objectives and limitations, but they fundamentally share the same core concept.

The scope of model applications is extensive. They are employed to depict realities imperceptible to the naked eye, such as atoms and cells, and those that transcend human scale, like planets. Models also aid in representing phenomena occurring at different speeds, ranging from rapid lightning strikes to gradual erosion. Furthermore, models can even encapsulate potential realities and ideas that have yet to materialize, such as weather forecasting.

The main objective of models is to use them to solve problems. Many of these problems are too complex to be solved by commonsense rules of thumb or by intuitive reasoning, and the use of words by themselves is only sometimes a satisfactory way of describing relationships. Models reduce ambiguity because they describe complexity with the maximum parsimony [14].

2.1.1 Type of Models

Models are essential and common in many fields of knowledge. They can be found in many different disciplines of science, such as philosophy, communication, ecology, engineering, medicine, etc., and can be classified in various ways. For simplicity, this thesis considers a basic approach common approach for classification:

1. **Iconic:** These are also known as physical models, which are representations that visually show what real things look at the physical and escalated representation of the target reality.

2. **Symbolic:** These are mathematical models. They represent the entities and their relations in terms of their functioning and mathematical functions and equations. Very often these models consider time and space. These are the models this work is focused on for the research chapter.

3. **Analogue:** These models are between the other two. They usually take a representation or function from one model and apply it to another.

A formal definition used by John Jeffers in his modeling book [14], can describe in simple terms a symbolic model, and it is the definition used for this work. The definition states:

"A model is a formal expression of the relationship between defined entities in physical or mathematical terms"

An important point to clarify is that modeling is not a mathematics branch and therefore, a purely theoretical task. Modeling is the application of logic and mathematics to reality. As a result of this process, the formal expression, must be capable of predicting reality and be tested against it [5].

2.1.2 The Modeling Process

Building a model in science resembles a scientific investigation. Thus, the starting point is the definition and delimitation of a problem. Next, a sequence of different steps needs to be completed for the majority of the modeling projects. Some of these steps are usually repetitive and cyclical due to the intrinsic fact of testing and confirmation activities involved. To illustrate the process, Ian Cameron's model development framework [11] is represented in Figure 2.1.

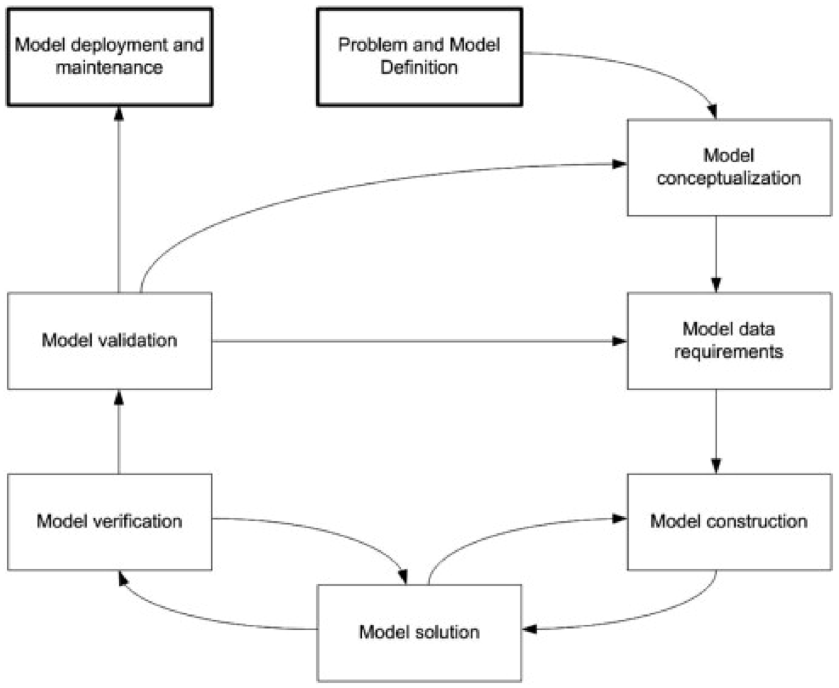


Figure 2.1: Model development framework [11]

Problem and Model Definition

The starting point of any modeling process is the definition of a problem. It is key to clearly articulate the problem subject to solve or the question trying to answer through the model. In this step, understanding the objectives, scope, and requirements of the project is extremely important.

Model Conceptualization

The conceptualization of the model is one of the most important steps in the whole process because boundaries and assumptions are defined, and the systems, sub-systems, different phases, and constraints of consideration are identified.

In this phase of the process is important to understand and criticize the assumptions of a model from the circumstance that the most mathematically sophisticated member of the team brings quantitative judgment to bear even on problems where the other team members understand the workings of the system better [45].

Model Data Requirements

In simple words, every model needs data to be constructed. Even though this step is rather simple to understand, this is one of the most challenging tasks to complete. Plenty of issues to be identified and multiple considerations depend particularly on the on-process model (and the previous conceptualization step). Some examples are the inaccessibility of the data, which will lead to experiments or simulation activities to generate it. Inappropriate data, due to poor choices, carelessness on treatment, etc, may lead to disastrous outcomes. This concept is so relevant in science in general that the expression *"garbage in, garbage out"* [33], which explains the previous point, is very popular among all the disciplines in science.

Model Construction

This step refers to transferring the abstract description from the previous phases to something tangible, through an appropriate tool or approach. The challenge, though, is to take the outcome from the goal discussions and combine it with the model conceptualization and data requirements to create a model that can encompass both the conceptualization and the objective.

Model Solution

This is the execution or run of the model. Most of the models are solved through numerical methods. The most common methods are algebraic equations, differential equations, integrals, hybrid systems, stochastic systems, time series, etc. [11].

Verification

This step refers to checking the implementation of the model and contrasting it against the original or initial model. Simply put, to determine if the model developed accurately represents the model concept output on the second step (model conceptualization) through comparison. Thacker [48] defines this concept as a "debugging" activity. He explains how to carry out this phase and addresses some possible implementation issues and ways to fix them. This review needs to go further into these details.

Validation

Validation is a step closely related to verification, yet it is different. To start with an analogy theories can be proved to be wrong but can never be proved to be right; therefore, there is always a risk of false confidence in model-based outcomes. Models that can provide accurate predictions of the performance of the corresponding real system for some restricted circumstances do not mean that the model can give good predictions in all cases [38]. The validation process requirement should establish the model's predictive accuracy and underscores the key role of uncertainty quantification in this process. A simple question to explain this stage proposed in [11] is: "*is the model a reasonable representation of the actual system?*". Sensitivity analyses and tests can be performed to ensure the quality of assumptions, inputs, parameters, etc., and validate the model. Here the connection between the second and third steps is completely clear. If the model is not validated it is necessary to review the conceptualization and the data requirements.

Deployment and Maintenance

The final step in the modeling cycle is the deployment and posterior maintenance of the models. Some authors [11] [21] will indicate that this is the beginning of the model's life. There are a bunch of good practices to ensure a long and healthy life of models, but, to be consistent with the scope of the thesis, this step represents the implementation of a verified and validated model.

2.1.3 Computational Simulation and Modeling

Computer modeling has emerged as a powerful tool in various fields, revolutionizing how we understand, analyze, and predict complex systems and phenomena. These techniques utilize the computational power of computers to create virtual representations of real-world systems, allowing researchers, engineers, and scientists to simulate and experiment with these systems in a controlled and cost-effective manner. Simulation involves running the computer model to replicate the behavior of the real-world system over time. Through simulation, it can be observed how the system responds to different inputs, study its dynamics, and evaluate the impact of various factors and variables. This allows for a deeper understanding of complex phenomena that may be challenging or impractical to study directly in the real world.

Computer modeling and simulation find applications in diverse domains, including physics, engineering, biology, economics, social sciences, etc. They enable scientists and engineers to study intricate systems that were impossible before. The advantages of computer modeling and simulation are numerous. They provide a controlled and repeatable environment for experimentation, reducing the need for expensive and time-consuming physical prototypes or experiments. They also offer the ability to explore various scenarios and "what-if" analyses, aiding in decision-making processes and risk assessments. Additionally, computer models and simulations can uncover insights and reveal hidden patterns that might not be immediately apparent in real-world observations. One of the main motivations is that simulation eliminates approximations. Usually, to treat a problem analytically one needs to resort to some approximation; for example, a mean-field-type approximation. With a computer simulation, we can study systems not yet tractable with analytical methods. The computer simulation approach allows one to study complex systems and gain insight into their behavior. Indeed, the complexity can go far beyond the reach of present analytic methods[24]. Even with such a, there is an important limitation: "*you can program what you want but you cannot compute what you want*" [37]. It's clear that there is an important difference in the concept of programming and computing, and there is always good to remember that computers are built machines with limitations; therefore, they are not always capable of carrying out what is in our minds. However, it is crucial to acknowledge that computer models and simulations simplify reality and inherently have limitations. The accuracy and reliability of the results depend on the fidelity of the model, the quality of input data, and the assumptions made during the modeling process. Validation and verification techniques are employed to assess the credibility of the models and ensure that they capture the system's essential features accurately.

There is plenty of literature to go deep into this topic. Some interesting sources can be read [36] [12] [24] [9] however the deepness of their work belongs to the computer science discipline and the information and discussions presented on their studies are far from the scope of this study.

2.2 Surrogate Modelling

Surrogate models are a special type of model. They are also called **metamodels, emulators, or response surface models**. Surrogate models are Data-driven models. This means that the models assume that absolutely nothing is known about the inner workings of the simulator. The literature commonly uses the "black box" concept, where the information about the response is collected from evaluations and the phenomena that happen inside are disregarded. Finally, from these data, an approximation is derived.

To understand the development and importance of surrogate models it is important to understand how they become a solution. The introduction of computer simulations caused a revolution. Simulating the desired aspects of a system and performing the tests and experiments virtually drastically reduced the cost of prototypes, experiments, etc. However, as simulation software became more precise and gained accuracy over the years, its computational cost grew tremendously. The growth of computational cost was so fast that it had beaten the growth in computational power, resulting in lengthy simulations on state-of-the-art machines and high-performance computing environments, mainly due to the never-ending drive for finer time scales, more detail, and general algorithmic complexity [12]. This is the problem that surrogate models try to solve. To this end, surrogate models can be considered a cheap alternative to evaluate mathematical expressions that can replace the simulator (when this is very expensive).

2.2.1 Formalism

As surrogate models are one of the main topics of interest in this thesis, it is appropriate to formally define the surrogate modeling process. Equation 2.1 presents the formality. Where given an expensive function f and a collection of data samples with corresponding evaluations represented by D we seek to find an approximation function $\widetilde{f}$:

$$\begin{aligned} arg\ max_{t \in T}\ arg\ min_{\theta \in \Theta} - \ \Lambda(\kappa, \widetilde{f}_{t,\nu}, D) \\ subject\ to\ \Lambda(\kappa, \widetilde{f}_{t,\nu}, D) \leq \tau \end{aligned} \tag{2.1}$$

κ represents an error function and τ the target value for the quality as expressed by the error function, all under the operation of the model quality estimator Λ. The quality estimator drives the optimization of both the model type t out of the set of available model types T and its hyperparameters θ. Therefore, the choice of Λ is crucial to obtain a satisfying surrogate model at the end of the process as it is the metric driving the search for θ. This often leads to requiring several iterations of the process to obtain satisfactory results.

2.2.2 Approximation Models

Approximation-based models constitute the largest and most popular category of surrogate models. The reason for the popularity of these approaches is mainly the benefits or advantages that these category offers, which can be summarized in the next points [47]:

- Can be constructed without prior knowledge about the system of interest and provide interpretability. By analyzing the surrogate model's coefficients or feature importance, insights into the underlying relationships and mechanisms driving the system's behavior can be obtained.

- They are mostly based on algebraic models integrated with optimization algorithms. This integration allows for various optimization techniques (like gradient-based methods, or surrogate-based optimization), to find optimal designs or parameter settings.

- They are very cheap or easy to evaluate. Using surrogate models can minimize the need for expensive physical prototypes or extensive testing. This leads to significant cost savings in research, development, and manufacturing processes.

- They are generic and thus easily transferable. These models can often be transferred to similar problem domains or related systems, allowing for knowledge transfer and reuse of modeling efforts. This adaptability and transferability make surrogate models valuable tools in various applications.

Regardless of the technique selected for the constructions of the surrogate model, most of them can follow the flow shown in Figure 2.2. We can appreciate the similarities and differences with a classic flow or framework for models presented in the previous point in Figure 2.1. Consider that these models are constructed from the training samples obtained from the high-fidelity simulation model. The main stages for developing a surrogate model are described as follows.

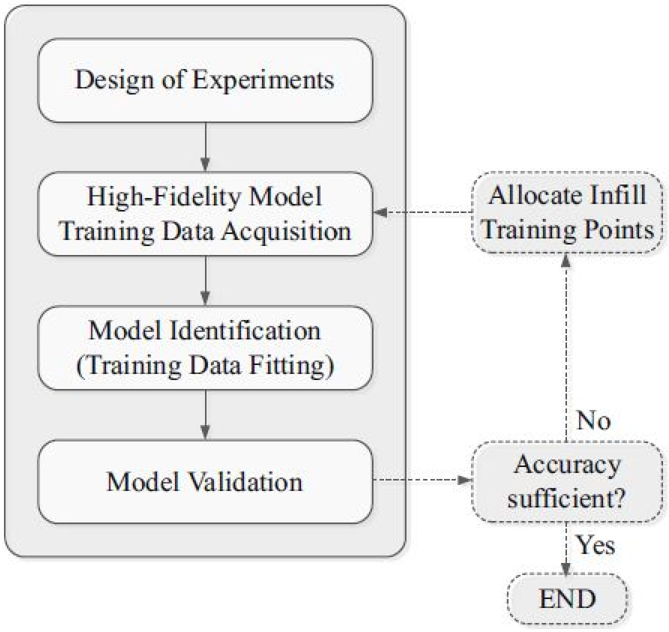

Design of Experiments
High-Fidelity Model Training Data Acquisition
Model Identification (Training Data Fitting)
Model Validation
Allocate Infill Training Points
No
Accuracy sufficient?
Yes
END

Figure 2.2: Generic surrogate model construction [34]

Design of Experiments

The design of experiments (usually abbreviated as DOE) is the first stage for constructing a surrogate model. Is the strategy for allocating the training points into the space[34]. Typically, the number of samples is limited due to budget. When the training data comes from computer simulations, it's commonly used the space-filling technique. The most common techniques for space filling in DOE are factorial design **factorial designs, random sampling, uniform grid sampling, and Latin hypercube sampling**. DOE is also a whole new subject of study, and plenty of literature exhaustively covers this topic. To go deeper, some recommendations for review are [15], [6], and [41].

Data Acquisition

This step involves gathering the data that has been trained and generated during the preceding phase.

Identification

This step defines and develops a model using approximation techniques or algorithms. In most cases, determining the surrogate model parameters requires solving a suitably defined minimization problem; however, in some situations, the model parameters can be found using explicit formulas by solving an appropriate regression problem [34].

Validation

This stage is to verify the model's accuracy. Usually, the generalization capability of the surrogate is of major concern [8]. The predictive power of the designs is not seen during the identification stage. Consequently, the model testing should involve a separate set of testing samples. Maintaining a proper balance between surrogate approximation and generalization (accuracy at the known and the unknown data sets) is also of interest.

Those are the main stages for one iteration of constructing a surrogate model based on approximations. Specifically, a new set of samples and the corresponding high-fidelity model evaluations may be added to the training set upon model validation and utilized to re-identify the model until the target accuracy is reached. This thesis presents a generic study mainly in the Identification and Validation stages.

2.3 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a discipline within the computer science field that aims to replicate and develop human intelligence and its processes through computers. There is not a formal agreement about the definition of this concept but in this study, we are not going to provide any specific definition but adopt the general idea of it. Artificial Intelligence is a wide field of knowledge dedicated to the design, modeling, and implementation of intelligent systems so that they automatically give a response to complex problems arising in the real world. In this regard, several subfields can be found in this broader paradigm. Machine Learning (ML) and Optimization are the ones that stand out [50]. Furthermore, this study will focus on the Machine Learning branch of the IA and how it is used for modeling.

2.3.1 Machine Learning

Machine learning (ML) is currently widely used in many modern artificial intelligence applications. The breakthrough of the computation ability has enabled the system to compute complicated different ML algorithms in a relatively short time, providing real-time human-machine interaction and making multiple applications possible [13]. ML comprises those algorithms targeted to extract knowledge from data, relying on fundamental concepts in computer science, statistics, or probability. Apart from that, ML goes one step further, being capable of unveiling additional features from the data, such as causality or advanced cognitive reasoning. Thus, ML techniques are meant to properly represent raw data featuring experience and render it into a model able to gain insights and make either decisions or predictions. ML is closely related to data mining, although the latter fundamentally concentrates on exploratory analysis, while the former draws upon other artificial intelligence disciplines such as computational statistics or pattern recognition [50].

All machine learning algorithms have 3 main components: the data, the model, and the loss function.

The Data

Arguably the most important of the components for an ML method. The collection of data points or data sets is singular points which are atomic units of "information containers". This collection of points could be everything of interest. As there are no restrictions for the types of data, one practical requirement for a useful definition of data points is that we should have access to *many of them*. Many ML methods construct estimates for a quantity of interest (such as a prediction or forecast) by averaging over a set of reference (or training) data points. [30]. Another requirement is the number of features contained in the set. A feature is any variable used to define every individual point. If the sample size is called m and the number of features n, the behavior of ML methods often depends crucially on the ratio m/n. The performance of ML methods typically improves with increasing m/n. As a rule of thumb, we should use data sets for which $m/n >> 1$.

The Model

Practical ML methods can search and evaluate only a (tiny) subset of all possible hypothesis maps. This subset of computationally feasible ("affordable") hypothesis maps is called the hypothesis space or model underlying an ML method. The preference for a particular hypothesis space often depends on the available computational infrastructure available to an ML method. Different computational infrastructures favor different hypothesis spaces. ML methods implemented in a small embedded system might prefer a linear hypothesis space, resulting in algorithms requiring a small number of arithmetic operations.[30]. As an informal objective, and the way ML is strongly connected with surrogate modeling, can be represented as shown in equation 2.2.

$$y \approx h(x) \quad or \quad y \approx \hat{y} \tag{2.2}$$

where the ML will learn a hypothesis map $h : X \longrightarrow Y$ with X features and Y labels ($x \in X$ and $y \in Y$).

The Loss Function

The loss function, also known as the cost function or objective function, is implemented to answer the question: *Which predictor map out of all the maps in the hypothesis space is the best for the ML problem at hand?*. The value quantifies the discrepancy or error between the predicted output of a model and the true target values in the training data. A small (close to zero) value indicates a low discrepancy between the predicted label and the true label of a data point, which is the desired target. It's worth noting that different loss functions have distinct properties and implications. Some loss functions may be more sensitive to outliers or have different gradients, which can influence the learning dynamics and convergence of the model. Thus, understanding the characteristics of different loss functions is crucial for designing effective machine learning models. More details about this topic can be found in [30]. Still, essentially every method of ML surrogate models has associated with a loss function and can change the performance of the same algorithm [25].

2.3.2 Machine Learning Categories

Here are four classical main classical categories to classify Machine Learning algorithms based on how the data is collected and the learning style applied to the model generation. These categories are:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

The next short definitions for these categories are taken from [50].

Supervised Learning

In this category, labeled input data feeds a learning algorithm in the training phase. The model or inferred function will be generated under the premise of minimizing an error function or, on the contrary, maximizing the precision. These systems are intended to correctly map unseen examples. Mostly addressed problems, in this case, are classification and regression

Unsupervised Learning

No label for any input vector is provided. In this case, the objective is to find the structure behind the patterns with no supervisory or reward signal. These models analyze and deduce peculiarities or common traits to discover similarities and associations among the samples. Example problems are clustering and latent variable models.

Semi-supervised Learning

Labeled and Unlabeled instances feed the algorithm, falling between the previously mentioned categories. The acquisition of labeled data is fairly expensive and often requires human skills while unlabeled data can be of great practical value to surpass the performance of previous learning approaches. The goal of this kind of system can be oriented towards transudative learning (deriving the labels of the unlabeled data by searching for analogies) or inductive learning (inferring the mapping from initially labeled vectors to their corresponding categories).

Reinforcement Learning

In this last category, the system interacts with its environment by producing actions and receiving either a positive or a negative stimulus from the events in response. These stimuli prompt the translation of that feedback into a learning process aiming at minimizing the punishment or maximizing the gained reward. This sort of learning is typical of robotics and its realistic environments which require algorithms for identifying relevant peripheral events in the stream of sensory inputs.

In addition to these categories, we can define 2 new categories, due to the latest technological advances and the development of more complex and sophisticated paradigms. The inner complexity of these new models, which try to represent complex knowledge is usually called **Deep Learning** [32], and the newer ways for training a model via online and shared learning is called **Federated Learning** [29].

Regardless of the categories of ML, there are two fundamental types of machine learning problems: Regression and classification.

- **Regression Problems:** Regression problems involve predicting a continuous or numerical output variable. The goal is to model the relationship between input variables (also known as features or independent variables) and the output variable. Regression techniques aim to estimate the best-fitting mathematical function that describes the relationship between the variables. Examples of regression problems include predicting housing prices based on features like size and location, forecasting stock prices, or estimating sales revenue based on marketing expenditures.

- **Classification Problems:** Classification problems involve assigning input instances to predefined classes or categories. The goal is to learn a decision boundary or function to classify new, unseen instances based on their features accurately. Classification tasks can be binary, where there are two classes (e.g., spam detection or disease diagnosis), or multiclass, with more than two classes (e.g., image recognition or sentiment analysis).

In summary, regression problems predict continuous values, while classification problems deal with assigning instances to predefined categories. The choice between regression and classification depends on the nature of the problem and the type of output variable you are trying to predict or classify [26].

2.3.3 Machine Learning Techniques

Machine learning encompasses a wide range of techniques that allow computers to model through data and make predictions or take actions without explicit programming. These models act as a model of a model and thus can replace an expensive simulation model by approximating its input and output responses [28]. A compact table can be found in figure 2.3 and it shows a sample of techniques in parallel with DOE and fitting alternatives.

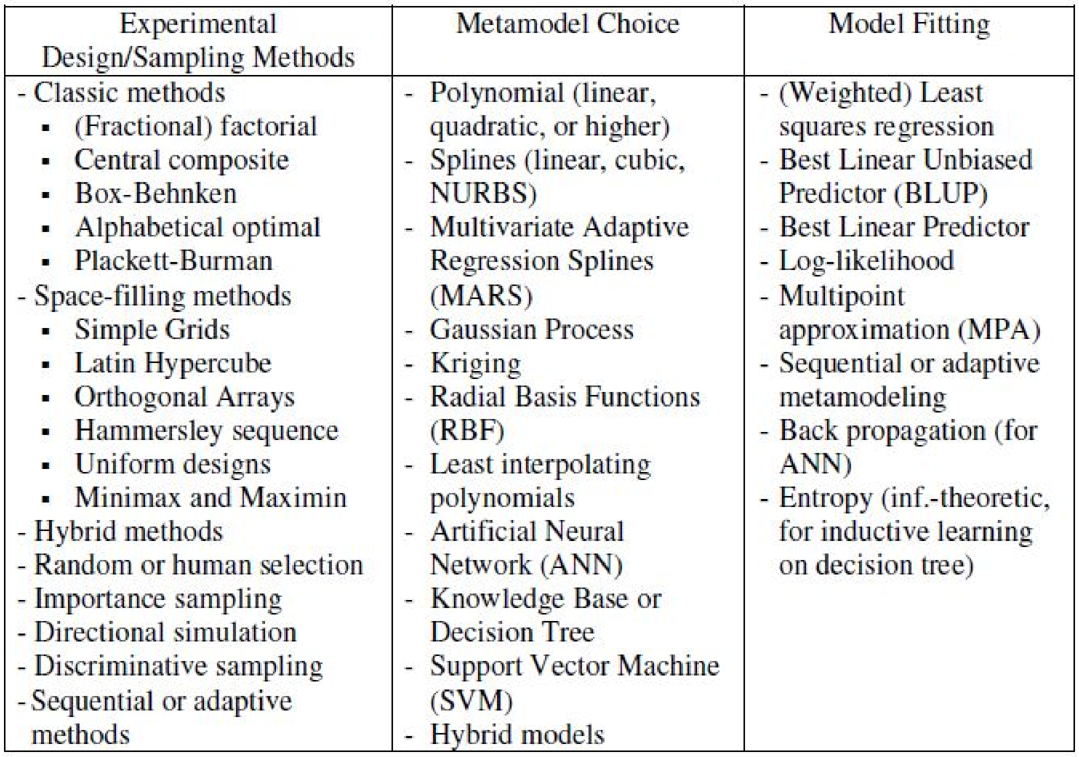
Some of the most commonly used machine learning techniques for constructing surrogate models are presented in the next lines.

Linear Regression

This technique is often used where a surrogate system of interest is represented as a linear combination of the variables. The general expression can be found in equation 2.3

$$\hat{f}(x) = w_0 + \sum_{j=1}^{d} x_j w_j \tag{2.3}$$

where x is a vector of size d, d is the number of variables and w is the vector of length $d+1$. To obtain the weight vector, the sum of squared errors between the actual data and the surrogate predicted value is minimized. Linear regression can often describe the behavior of functions



Experimental Design/Sampling Methods	Metamodel Choice	Model Fitting
- Classic methods ▪ (Fractional) factorial ▪ Central composite ▪ Box-Behnken ▪ Alphabetical optimal ▪ Plackett-Burman - Space-filling methods ▪ Simple Grids ▪ Latin Hypercube ▪ Orthogonal Arrays ▪ Hammersley sequence ▪ Uniform designs ▪ Minimax and Maximin - Hybrid methods - Random or human selection - Importance sampling - Directional simulation - Discriminative sampling - Sequential or adaptive methods	- Polynomial (linear, quadratic, or higher) - Splines (linear, cubic, NURBS) - Multivariate Adaptive Regression Splines (MARS) - Gaussian Process - Kriging - Radial Basis Functions (RBF) - Least interpolating polynomials - Artificial Neural Network (ANN) - Knowledge Base or Decision Tree - Support Vector Machine (SVM) - Hybrid models	- (Weighted) Least squares regression - Best Linear Unbiased Predictor (BLUP) - Best Linear Predictor - Log-likelihood - Multipoint approximation (MPA) - Sequential or adaptive metamodeling - Back propagation (for ANN) - Entropy (inf.-theoretic, for inductive learning on decision tree)

Figure 2.3: Common DOE, ML and Fitting techniques used for surrogate models construction [51]

over small ranges but eventually becomes too limited to describe data from real systems. The most basic linear regression model (when X is dimension 1) is given by equation 2.4

$$\hat{f}(x) = b + a \times x \tag{2.4}$$

Polynomial Regression

Polynomial regression is a statistical technique that uses regression analysis and analysis of variance to determine the relationship between design variables and responses. A linear polynomial is used to approximate the implicit limit state equation. The surrogate model is defined in equation 2.5.

$$\hat{f}(x) = \sum_{j=1}^{k} \beta_j v_j(x) \tag{2.5}$$

where β are the unknown coefficients and v the polynomial functions. Once again, the parameters can be estimated by the least square solution of the system. The final expression of the model is given by the equation 2.6 where the ϵ represents the statistical error.

$$\hat{f}(x) = \beta_0 + \sum_{j=1}^{n} \beta_j x_j + \sum_{i=1}^{n} \sum_{j \leq i}^{n} \beta_{ij} x_i x_j + \ldots + \epsilon \tag{2.6}$$

Radial Basis Function

To start with this method, a radial function is one where the Euclidean distance between the point to be measured and the sample point is used as the independent variable. Knowing that a radial basis function is a model constructed via the linear superposition with radial functions. In the equation 2.7 we have β as the vector of the model parameters and $\phi(x,c)$ as the radial functions vector.

$$\hat{f}(x) = \sum_{j=1}^{k} \beta_j \phi(||x - c^j||) \tag{2.7}$$

The calculation of the parameters is obtained with the solution of the equation shown in equation 2.8 where Φ is the matrix of $p \times k$ with the entries defined as shown in 2.9.

$$\lambda = \Phi^+ f = (\Phi^T \Phi)^{-1} \Phi^T f \tag{2.8}$$

$$\Phi_{k,l} = \phi(||c^k - c^l||) \tag{2.9}$$

Is worth mentioning that the radial function basis properties vary depending on the chosen radial function selected $\phi(||x - c^j||)$. The most common ones are Gauss, Multi-quadric, inverse Multi-quadric, and thin-plate splines functions. More details are found in reference [28]

Kriging

Kriging is based on the idea that a surrogate can be represented as a realization of a stochastic process [10]. It is a special type of Gaussian process and in its most general form can be represented as follows (see 2.10).

$$\hat{f}(x) = \sum_{j=1}^{k} \beta_j f_j(x) + \epsilon(x) \tag{2.10}$$

In the expression, $f_j(x)$ are the k known independent functions that define the trend of the location of x, as follows, β_j are the unknown parameters and $\epsilon(x)$ the random error at location x. The predictor is written as shown in 2.11:

$$\hat{f}(x) = f(x)^T \beta^* + r(x)^T \gamma^* \tag{2.11}$$

In the previous equation, β^* is the vector of generalized least-square estimates of β, $r(x)$ is the correlation vector (between $\epsilon(x)$ and $\epsilon(x_j)$) and γ^* is given by the covariance matrix. Having a random error term allows Kriging surrogates to provide an estimate of the uncertainty in addition to the predicted value at a specific location.

Support Vector Regression Models

This form of the surrogate is similar to Kriging (see 2.3.3) and usually are referred to as SVR. However, the way to calculate unknown parameters for this surrogate differs significantly from that. The unknown parameters in the model are obtained by formulating a mathematical optimization problem shown in 2.12 s.t. 2.13

$$min \quad \frac{1}{2}|w|^2 + C\frac{1}{n}\sum_{i=1}^{n}(\zeta^{+(i)} + \zeta^{-(i)}) \tag{2.12}$$

$$\begin{aligned} w.x^i + \mu - y^i &\leq \epsilon + \zeta^{-(i)} \\ y^i - w.x^i - \mu &\leq \epsilon + \zeta^{+(i)} \\ \zeta^{+(i)}, \zeta^{-(i)} &\geq 0 \end{aligned} \tag{2.13}$$

Support Vectors regressions transform the input data into m-dimensional space and attempt to construct a set of hyper-planes so that the distance from it to the nearest data point on each side of the plane is maximized using kernel functions. The kernel functions to transform the data into a higher dimensional feature space to make it possible to perform the linear separation.[52] An approximate graphical representation can be found in figure 2.4

Kernel Ridge Regression Model

Kernel Ridge Regression (KRR) is a machine learning technique used for regression tasks. It is an extension of Ridge Regression that incorporates kernel methods to handle nonlinear relationships between the input features and the target variable in a similar way that support vector machine. KRR combines the strengths of ridge regression, which provides regularization to prevent over-fitting, with the flexibility of kernel methods, which can capture complex patterns in the data.

The goal is to learn a function that maps input features to the target variable. The resultant function is a linear combination of basis functions transformed by a kernel function. The kernel function is responsible for computing the similarity between pairs of input samples and plays a crucial role in capturing nonlinear relationships[44].

This method includes the solution to overcome over-fitting or ill-posed problems in the regression analysis is so-called ridge regularization, also known as Tikhonov regularization, in

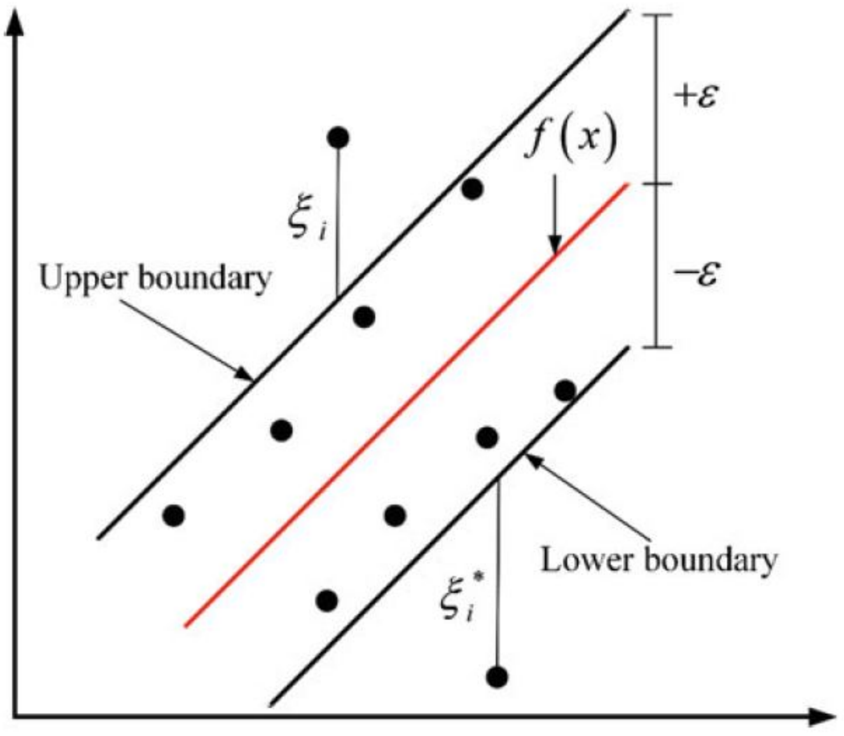


Figure 2.4: Representation of support vector regression [28]

which a ridge penalty term is imposed on the objective function in the optimizer to keep the regressor coefficients under control. This leads to the classic ridge regression (RR) methods which are commonly used in many regularization problems [35]. Given a finite training dataset, the objective of a linear regressor is to minimize the following cost function:

$$E_{RR}(w) = \sum_{i=1}^{n}(\epsilon_i^2) + \rho\|w\|^2, \tag{2.14}$$

where $\epsilon_i \equiv w^T x_i - y_i$. It can be shown that the optimal decision vector is:

$$w = [XX^T + \rho I]^{-1}Xy = [S + \rho I]^{-1}Xy \tag{2.15}$$

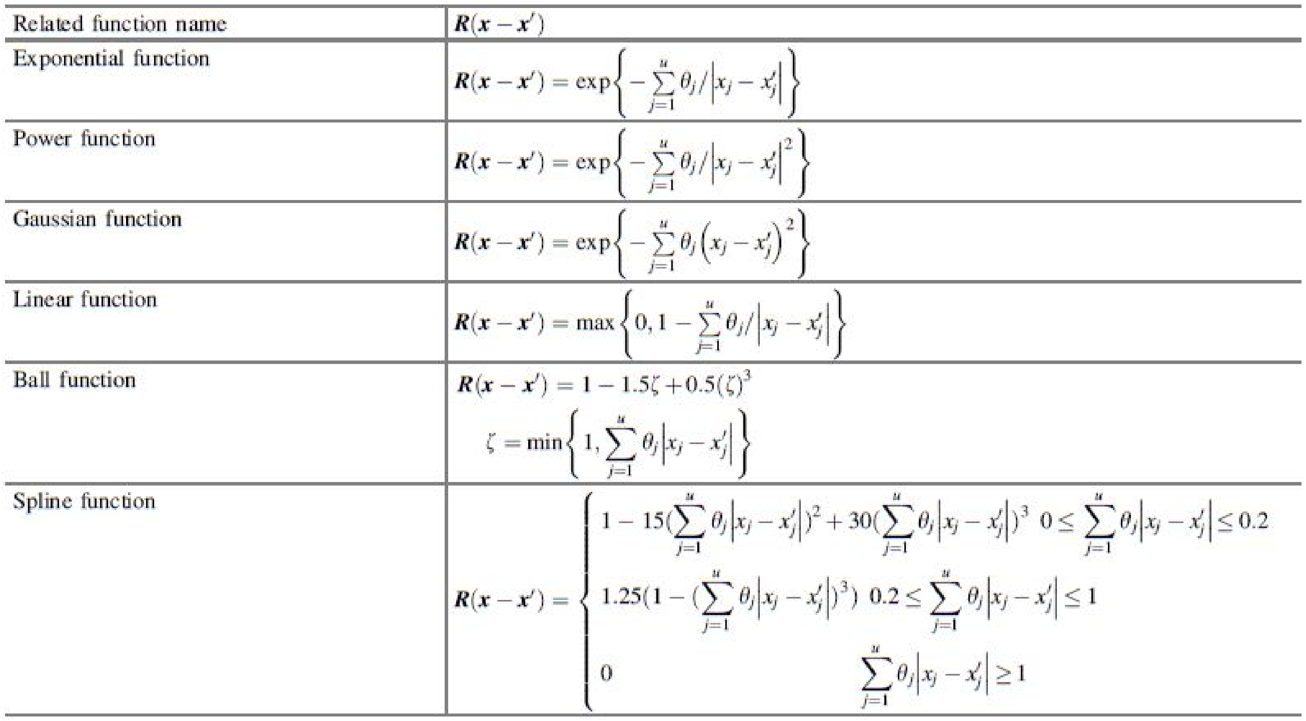
Gaussian Process

This non-parametric Bayesian technique provides the regression of the target function as a distribution over functions. This process uses a linear combination of inputs to predict the outputs. To achieve the goal, the Gaussian process (GP) uses a kernel function as a measure of similarity between points. Predictions are also characterized by a mean value and a standard deviation. The mean values of the predictions are given by equation 2.16.

$$\hat{f}(x) = \sum_{j=1}^{N_t} \beta_j k(x_i, x) \tag{2.16}$$

$$k(x_i, x) = \sigma^2 R(x - x_i) \tag{2.17}$$

x_i are the parameters used for the training and N_t is the total number of observations. β_j represents the weights (calculated with the knowledge of the kernel function) and $k(x_i, x)$ is the kernel function of election to generate the covariance matrix. The kernel is expressed as shown in 2.17, which leads to analysis $R(x - x_i)$ as the correlation between the points and the samples. There is not only one way to tackle this function, and in the figure 2.5 we can see some of the most common functions. The θ are roughness parameters indicating the rate at which the correlation between the output responses at the design point and the sample point. A maximum likelihood method is used to estimate the parameters but, due to the difficulty of solving partial derivatives, usually an optimization algorithm is the common approach to find the solution of the method. The prior distribution is a GP in which the mean function is quadratic. When the information at the five sample points is added to the GP, the posterior distribution of the design space is obtained. The shaded areas in the prior and posterior distributions represent the 95% confidence intervals. A graphic illustration is presented in 2.6.



Related function name	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}')$
Exponential function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = \exp\left\{-\sum_{j=1}^{u} \theta_j / \left\|x_j - x'_j\right\|\right\}$
Power function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = \exp\left\{-\sum_{j=1}^{u} \theta_j / \left\|x_j - x'_j\right\|^2\right\}$
Gaussian function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = \exp\left\{-\sum_{j=1}^{u} \theta_j \left(x_j - x'_j\right)^2\right\}$
Linear function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = \max\left\{0, 1 - \sum_{j=1}^{u} \theta_j / \left\|x_j - x'_j\right\|\right\}$
Ball function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = 1 - 1.5\zeta + 0.5(\zeta)^3$ $\zeta = \min\left\{1, \sum_{j=1}^{u} \theta_j \left\|x_j - x'_j\right\|\right\}$
Spline function	$\boldsymbol{R}(\boldsymbol{x}-\boldsymbol{x}') = \begin{cases} 1 - 15(\sum_{j=1}^{u} \theta_j \|x_j - x'_j\|)^2 + 30(\sum_{j=1}^{u} \theta_j \|x_j - x'_j\|)^3 & 0 \leq \sum_{j=1}^{u} \theta_j \|x_j - x'_j\| \leq 0.2 \\ 1.25(1 - (\sum_{j=1}^{u} \theta_j \|x_j - x'_j\|)^3) & 0.2 \leq \sum_{j=1}^{u} \theta_j \|x_j - x'_j\| \leq 1 \\ 0 & \sum_{j=1}^{u} \theta_j \|x_j - x'_j\| \geq 1 \end{cases}$

Figure 2.5: Common correlation function types [28]

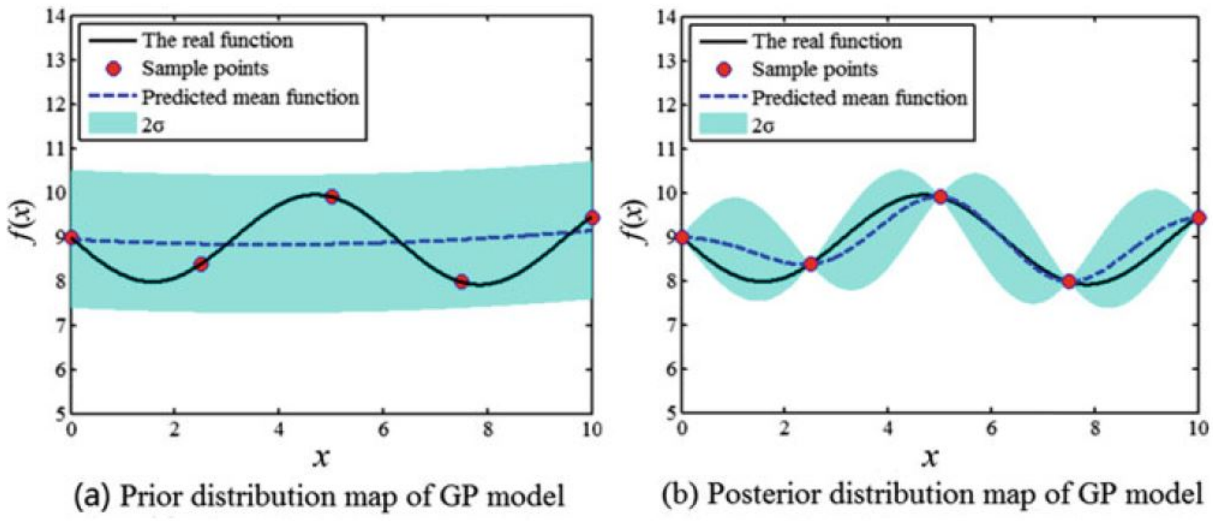


Figure 2.6: GP model examples [28]

Decision Trees

The Decision Trees methodology works by dividing the dataset into small subsets that serve as guides to develop the decision tree nodes. The nodes can be either decision nodes or leaf nodes, where the former represents a question or decision, and the latter represents the decisions made or the outcome. Decision trees continually split the dataset according to the parameters defined in the decision nodes. Decision nodes have branches coming out of them, and each decision node can have two or more branches. The branches represent the different possible answers that define how the data is split [42].

To perform the prediction, once the decision tree is built, the model takes each instance and follows the sequence that matches the instance's features until it reaches a final leaf. According to this, the classification process starts at the root node (the one on top) and continues along the branch that describes the instance. This process continues until a leaf node is reached, which represents the prediction for that instance

Random Forest

In a random forest regression, the response is obtained from a combination of the predictions of several decision trees, which are trained on random subsets of the complete data set[19]. The decision trees are flowchart structures in which each internal node represents a "decision" on a particular characteristic or attribute, and each branch represents a possible outcome of the decision. The technique has become very popular for nonlinear regression, scaling well to many input parameters and large amounts of data. It also allows for an inspection of the importance of individual features in the results, which can be useful for interpretability. A graphical scheme can be found in figure 2.7

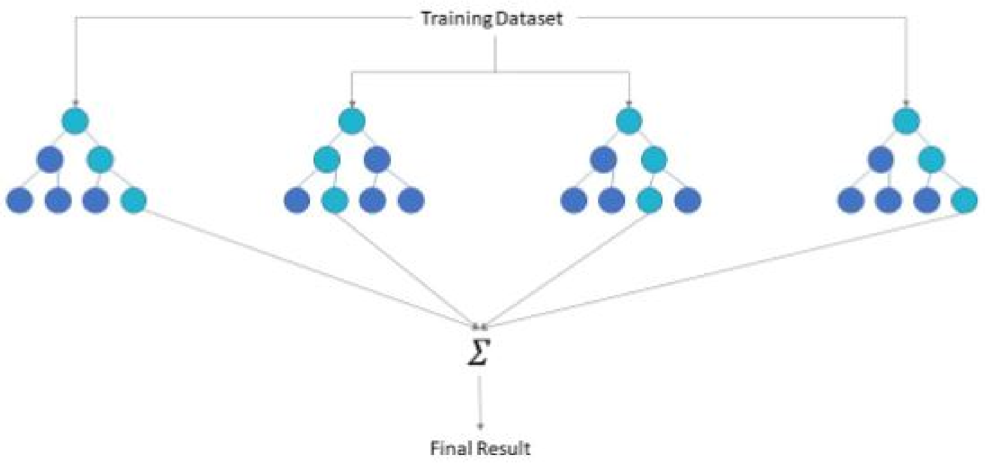


Figure 2.7: Random forest diagram

K - Nearest Neighbors

The k-Nearest Neighbors (KNN) is a nonparametric statistical algorithm [53]. This means that there is no explicit training phase before classification. KNN algorithm uses the entire dataset as the training set, i.e., it does not split the dataset into training and testing sets. KNN algorithm assumes that similar entities exist nearby. In other words, similar entities have features that are close to each other. Figure 2.5 shows similar data points that are close to each other. KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN can be used to predict loan approval, calculate credit ratings, speech recognition, handwriting detection, image recognition, intrusion detection, etc. KNNs are used in real-life scenarios where non-parametric algorithms are required. These algorithms do not make any assumptions about how the data is distributed[20]. Figure 2.8 show a graphical representation of the KNN algorithm.

Artificial Neural Networks

Artificial neural networks (ANNs) attempt to mimic the behavior of neurons in the brain. The models consist of an input and an output layer that are connected by several hidden layers in between [52]. For this project, ANNs can be considered as just another way of approximating sampled model data to create a surrogate model, since this is a large area of research with multiple approaches. The basic component of any ANN is the neuron and these interact with each other through different layers. The illustration shown in figure **??** can represent a neuron and

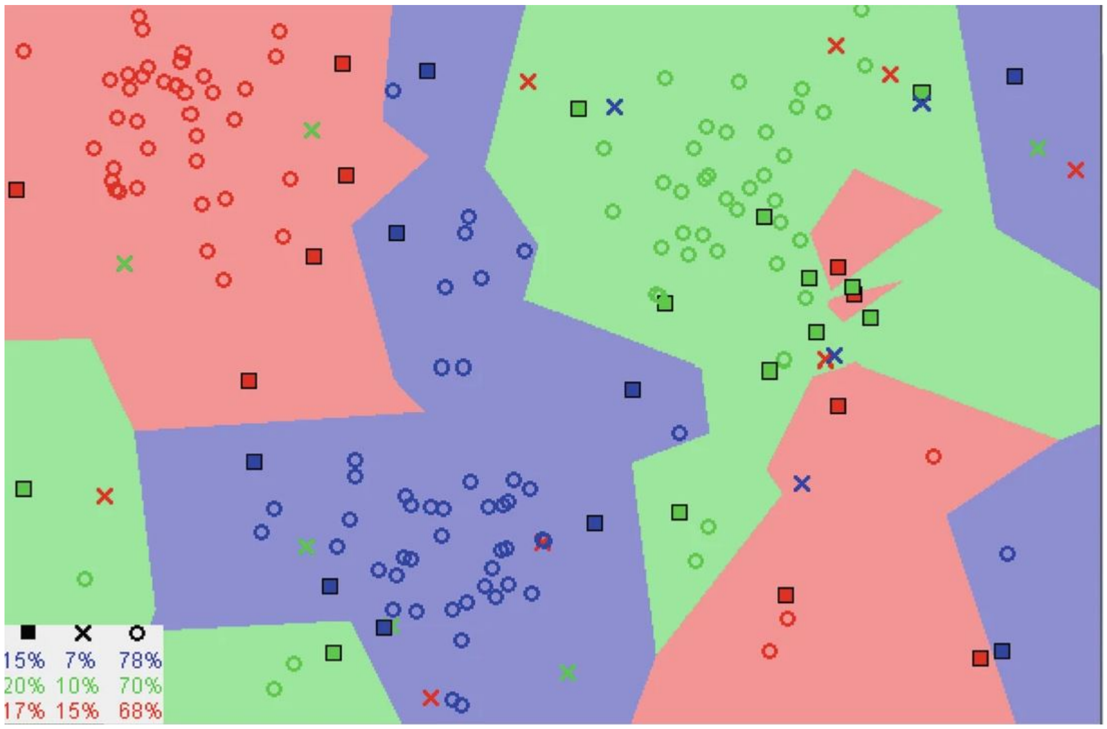


Figure 2.8: Graphical representation of KNN classification

figure 2.10 the interaction they make. In those figures, w_i represent the weights or regression coefficients, β the bias of each neuron.

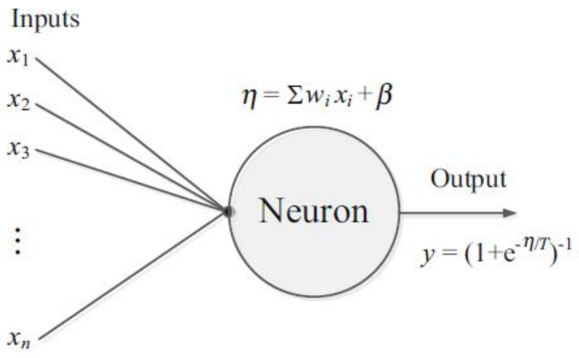


Figure 2.9: Basic structure of a neuron [34]

The construction of a neural network model involves the selection of its architecture selection and the training, i.e., the assignment of the values to the regression parameters. The network training can be formulated as a nonlinear least-squares regression problem. A popular technique for solving this regression problem is the error back-propagation algorithm.[34]. Training of a neural network refers to the process that identifies the values of the weights and

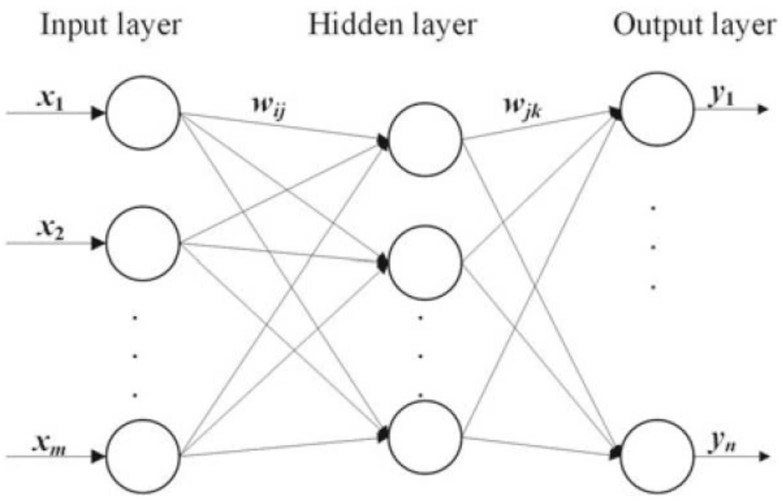


Figure 2.10: Basic representation of a 3 layer ANN [28]

biases, for this maximum number of iterations, a maximum error threshold, and a maximum training time are set. When any of these conditions are met, the training ends, and by comparing the training time, the number of training iterations, and the training error, it is judged whether the trained neural network meets the established requirements.

The steps to implement a neural network[28]:

1. **Neural Network initialization:** The connection weights and thresholds of the input layer, the hidden layer, and the output layer are randomly assigned values in the range of $(-1, 1)$.

2. **Input of sample data:** The t-th sample data $X^t = (x_1^t, x_2^t, ..., x_m^t)$ and the corresponding output data $Y^t = (y_1^t, y_2^t, ..., y_n^t)$ are extracted using a random method to fed the network.

3. **Layer connection:** The inputs and outputs of each neuron among the hidden layers and the output layer are calculated using the connection weights and thresholds.

4. **Training Error calculus:** Using the expected and actual outputs, calculate the error for each sample. The empirical formula is shown in equation 2.18.

$$E^t = \frac{1}{2}\sum_{k=1}^{n}(y_k^t - o_k^t) \tag{2.18}$$

5. **Correction for the hidden-output:** The error found in each neuron is used to correct and adjust the connection weights and thresholds between the hidden and output layers. The

following formula is used to continuously improve the weights of the output layer: (see equations 2.19 and 2.20)

$$\Delta w_{kj} = -\eta \frac{\partial E^t}{\partial w_{kj}} \tag{2.19}$$

$$\frac{\partial E^t}{\partial w_{kj}} = -(y_k^t - o_k^t) f^{'}(l_k) c_j \tag{2.20}$$

where η is called learning efficiency, $f^{'}(l_k)$ is the partial derivative of the excitation function concerning the input l_k to the output layer and c_j are the output layer values. Therefore, the adjustment of the weights is given by 2.21

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj} = w_{kj}(t) + \eta(y_k^t - o_k^t) f^{'}(l_k) c_j \tag{2.21}$$

6. **Correction for the input-hidden:** The iterative formula for adjusting the weights is the same previous idea, but here the partial derivatives contain 2 functions. Thus, the final equation for adjusting the weights is shown in 2.22.

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji} = w_{ji}(t) + \eta(y_k^t - o_k^t) f^{'}(l_k) w_{kj} f^{'}(b_j) x_i \tag{2.22}$$

7. **Continue the iteration:** Random selection of one of the remaining samples within the training data and repeat from step 3 until the training of N samples is completed. The final error can be easily computed using the formula given in 2.23.

$$E = \frac{1}{N} \sum_{t=1}^{N} E^t \tag{2.23}$$

2.4 Others

2.4.1 Functional Mock-up Interface

The Functional Mock-up Interface or FMI is a standard for exchanging dynamical simulation models between different tools in a standardized format[16]. The intention is to simplify the creation, storage, exchange, and (re-) use of dynamic system models of different simulation systems for model/software/hardware-in-the-loop simulation, for cyber-physical systems, and other applications. It is an open-source published under a creative commons license (Creative Common Attribution-Share Alike 4.0 International). Currently, the latest version is 3.0 and it is supported for more than 160 simulation tools [31].

Every model is distributed in a zip file with extension .fmu (functional mock-up unit) that contains:

- **Model description**, an XML file with the definition of the variables
- **Functions**, a set of all model equations, called C-functions,
- **Extras**, any kind of extra information, documentation, tables, maps, etc.

2.4.2 Computational Tools

Python language

Python is one of the most popular programming languages for data science and therefore enjoys a large number of useful add-on libraries developed by its great developer and open-source community [43]. Python is not a compiled language, meaning that it does not precompile the code into binary. Instead, a software environment, Python interpreter, translates the script into binary during the execution of the code in real-time. With its distribution, Python comes with some basic functionality but relies on external packages to perform almost all numerical computations. After the natural selection process over the past 10 years, a small set of packages that provide some fundamental computing capabilities have received wide acceptance in the Python community [22].

Sci-kit Learn

Scikit-learn is the most comprehensive and open-sourced machine learning package in Python. As machine learning is often a component of a more general application (such as a Web service), it is desirable to have it furnished using the same programming language as the other parts of the application for seamless integration [40].

In summary, Scikit-learn includes a collection of efficiently implemented machine learning methods and is well-documented and maintained by the community. However, readers should also be aware that it is possible that Scikit-learn may not include some methods that are used in specialized applications. As such, we recommend readers consider Scikit-learn as long as the needed methods are available but look for other software for those methods not included. In the next section, we will show some specific examples of how to use the machine learning methods in Scikit-learn [22].

The Scikit-learn package covers four main topics related to machine learning. They are data transformation, supervised learning, unsupervised learning, and model evaluation and selection. The details in each topic can be found in the user guide on the Scikit-learn website (see ref [3].

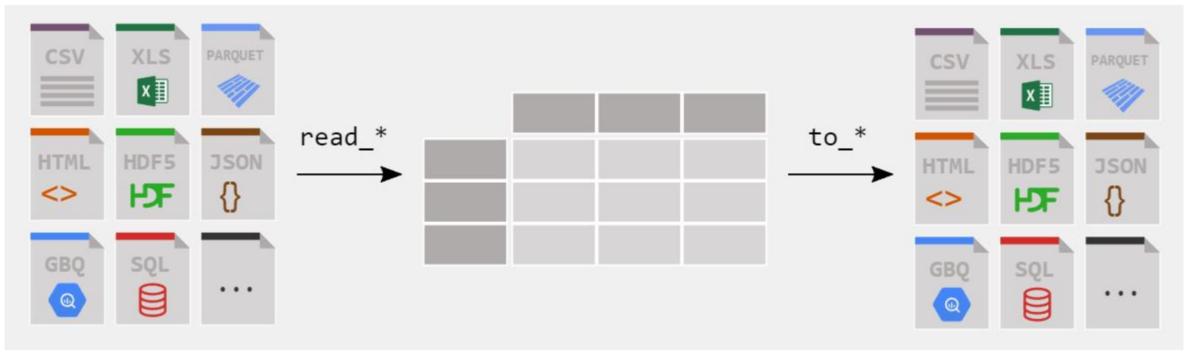


Figure 2.11: Pandas work scheme [39]

Pandas

Pandas is an open-source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical real-world data analysis in Python. It can read practicable all popular data files format (CSV, Excel, SQL, JSON, parquet,...) and present them into a neat framework. Similarly, it can save data in the same multiple file extensions. A small structure can be found in figure 2.11

Pyplot and Seaborn

Pyplot and Seaborn are Python data visualization libraries based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. A brief description of each one is stated:

- Pyplot provides a MATLAB-like interface for creating various types of plots, charts, and figures. Pyplot offers a wide range of functions and methods to customize plots, control axes, add labels, titles, legends, and more. It is a versatile library used for creating basic to advanced visualizations and is well-suited for scientific and technical plotting.

- Seaborn provides a higher-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex plots, such as scatter plots, line plots, bar plots, box plots, heatmaps, and more. It offers various built-in themes, color palettes, and statistical functionalities to enhance the visual representation of data. Seaborn is particularly useful for exploratory data analysis and visualizing relationships and distributions in datasets.

More details can be found on their web pages [2] and [4]

FMPy and PythonFMU

FMPy is a free Python library to simulate Functional Mock-up Units (FMUs). It supports FMI 1.0 and 2.0 and the Co-Simulation and Model Exchange features. Also, it has a graphical user interface that facilitates the work with simulations [1]. It has been developed at Dassault Systèmes and released under the BSD 3-clause license.

PythonFMU is a lightweight framework that enables the packaging of Python3.x code as co-simulation FMUs. The framework consists of a set of helper classes and a command line utility for transforming compliant Python sources into ready-to-use cross-platform FMUs [23].

2.4.3 Error Measurements - Metrics

When evaluating regression models, it is important to consider the specific context and requirements of the problem. Different metrics have different strengths and weaknesses, and the choice of evaluation metric depends on the specific goals and characteristics of the regression task at hand [17],[27]. Some of the most popular metrics can be used to assess their performance and in this thesis, the focus is on the metrics for regression problems, therefore, classification metrics are not covered.

Coefficient of Determination or R^2

It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well-unseen samples are likely to be predicted by the model, through the proportion of explained variance. It indicates how well the model fits the data, with higher values (closer to 1) indicating better fit. Equation 2.24 shows the estimated calculus of the coefficient.

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{2.24}$$

Mean Absolute Error

Mean Absolute Error, often also called MAE, is a risk metric corresponding to the expected value of the absolute error loss or l-norm loss. It calculates the average absolute differences between the predicted values and the true values. It provides a measure of the average absolute deviation of the predictions from the actual values. The formula for its calculation is shown in equation 2.25.

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \times \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i| \tag{2.25}$$

where $\hat{y}_i$ is the predicted valued of the i-th sample and y is the true value.

Mean Squared Error

Mean Squared Error or MSE, is the average of the squared differences between the predicted values and the true values. It provides a measure of the average squared deviation of the predictions from the actual values. Lower MSE values indicate better model performance. It is very similar to MAE but this measurement is more sensitive to outliers. Equation 2.26 shows the formula.

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \times \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2 \tag{2.26}$$

Median Absolute Error

The Median Absolute Error is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction, in other words, it calculates the median of the absolute differences between the predicted values and the true values. The metric is calculated as equation 2.27 shows.

$$Median\ Absolute\ Error\ (y, \hat{y}) = median(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, ..., |y_n - \hat{y}_n|) \tag{2.27}$$

Max Error

The Max error calculates the maximum residual error, which means, it finds the largest difference between the predicted value and the true value. It serves as a measure of the worst-case error encountered by the regression model. In an ideal scenario where the model perfectly fits the training set, the Max Error would be 0. This metric provides valuable insights into the extent of error exhibited by the fitted model. The equation 2.28 shows the formality of its calculus.

$$Max\ Error\ (y, \hat{y}) = \max(|y_i - \hat{y}_i|) \tag{2.28}$$

Chapter 3

Research

This research chapter aims to show the investigation of the performance of different machine learning algorithms as surrogate models. The research's methodology involves the construction of surrogate models using a range of machine learning algorithms, which are, support vector machines, Kernel Ridge, Decision Trees, and Gaussian Processes from a simple simulation model that will represent a productive process. The performance of these surrogate models is evaluated using some performance metrics, including accuracy and robustness. The findings of this study provide valuable insights into the performance of machine learning algorithms in surrogate modeling and offer guidance on the selection of appropriate algorithms, methods, and considerations for general engineering applications or production processes.

3.1 Model of Study

The constructions of surrogate models need a prior model to be based on. For this purpose, a model that simulates the trajectory of a simple ball that bounces in the earth with the possibility to set energy loss during bounce and energy loss from drag was chosen. The model selection was based on its simplicity, prior knowledge of the physics involved, and the possibility of gathering as much data as needed without expending a huge amount of resources.

This particular model was developed by Siegfried Eisinger from DNV company, co-supervisor of this work. It is written in Python and built it using FMU standards with the help of PythonFMU and FMPY libraries. The whole code for the model can be found in the appendix. It is important to mention once again that this model tries to act as a symbolic representation of a productive process.

The model needs a set of initial values, also known as parameters, to perform the simulation. These parameters have to be given to the model as initial values. The list of the parameters is following:

- **Height** y_0 **:** Represents the ball's height at the simulation's beginning. Also considered as the vertical distance. It has dimension $[m]$.
- **Velocity** v_0 **:** The ball's total velocity at the simulation's beginning. Also called speed. It has dimension $[m/s]$.
- **Angle** α_0 **:** Represents the initial angle of the velocity regarding the horizontal axis. It is set in degrees.
- **Bouncing Factor** Bf **:** Represents the speed change factor while the ball bounces. It has no dimension and can be considered as a multiplicative energy factor during the bounce.
- **Drag Factor** Df **:** Represents the drag deceleration factor. It has dimension $[1/m]$.

Given the initial parameters, the model will simulate the behavior of the ball, tracking the following variables:

- **Horizontal distance** x **:** Keeps track of the ball's position on the horizontal axis.
- **Vertical distance** y **:** Keeps track of the ball's position on the vertical axis.
- **Horizontal speed** v_x **:** Keeps track of the velocity or speed on the horizontal axis.
- **Vertical speed** v_y **:** Keeps track of the velocity or speed on the vertical axis.
- **Time** t **:** Keeps track of the time in the simulation.

The model will simulate the trajectory of a ball thrown with the initial parameters and register the tracking variables at every time step (in this setup every $0.04[s]$). Figure 3.1 shows an example of the output of one simulation generated with initial parameters: $y_0 = 0$, $v_0 = 3$, $Bf = 1$, $Df = 0$ and $\alpha_0 = 45^o$. On the horizontal axis, the movement is a Rectilinear Uniform Motion, due to a constant v_x (drag=0), and Uniform Accelerated Motion in the vertical axis, due to a constant slope in v_y. Also, it is easy to identify when the ball bounces because the v_y describes a sudden change from a negative value to a positive (direction of the motion). Those times also correspond with the variable y reaching a value close to 0.

3.2 Definition of the problem

Now that the model is described and known, the next step is to define the particular objective of this experiment. Taking into consideration that the main objective of this study is to assess the performance of machine learning algorithms as surrogate models generally and not finding the best algorithm or parameters optimization to fit the surrogate models for a particular

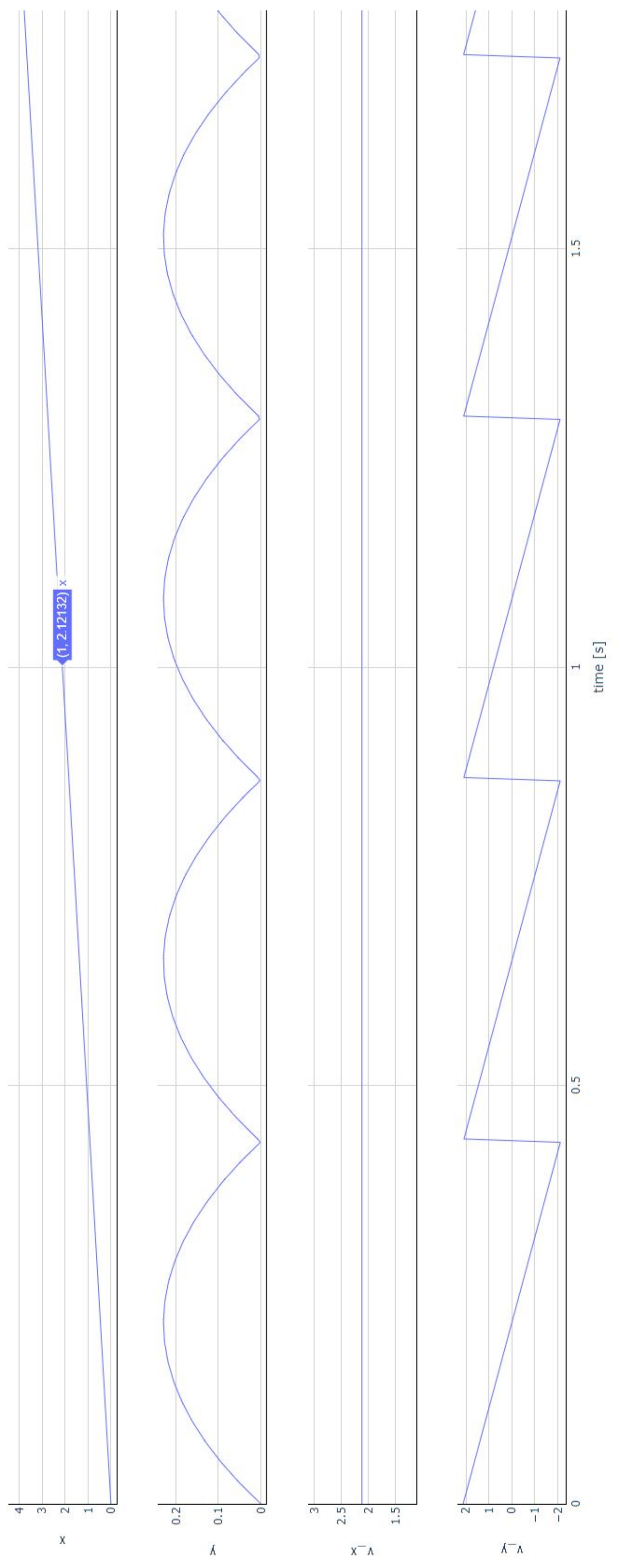


Figure 3.1: Graphical output of one simulation of the model of study

model, the problem cannot be only to predict some results but to work with these predictions. The problem of this research is, therefore, as follows:

"To build a surrogate model that helps to find the angle that maximizes the horizontal distance traveled by the ball before its first bounce, given a set of initial parameters".

This problem is suitable for this study mainly for two reasons. The first one is because it represents a problem that the original model cannot solve on its own unless that iterate and perform multiple simulations. Given the assumptions that are carrying out simulations, it may result in expensive either in terms of time and/or resources involved the alternative solution is to construct a surrogate model that is able to produce similar results inexpensively, in this particular case, using the help of artificial intelligence.

The second reason is the link between this and a potential future problem in a productive environment. For instance, it can be imagined as the parallel with a model that simulates a productive plant under a certain configuration that looks for the production optimization for all the possible configurations of variables.

3.3 Methodology

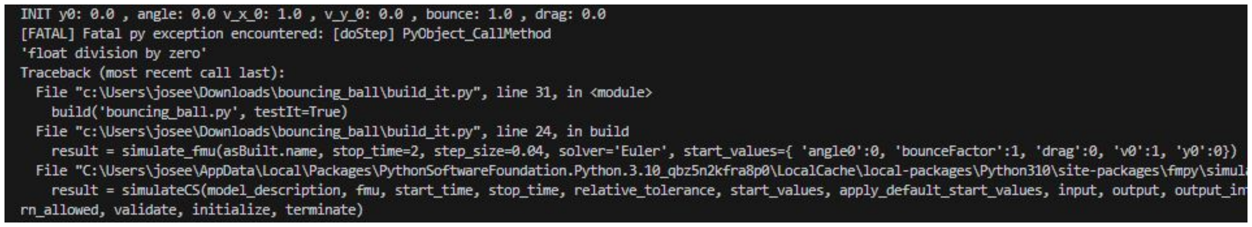
Now that the model subject of study is clear and the problem is defined, the next lines describe the general methodology of the study of this thesis. It consists in an experiment that mixes classical experimentation with artificial intelligence tools, so logically, the methodology starts with the design of the experiment, followed by the machine learning preparation and finalizing with the analysis of the results.

3.3.1 Design of the Experiment

The first phase of the thesis methodology is the experiment design. This has been done based partially on the Design of Experiments methodology to keep formality and give a robust base on the thesis results.

Problem Definition

As it is mentioned in the previous section, the problem to solve is to build surrogate models that can emulate the behavior of a more complex model and use the outcomes to optimize another certain task. To achieve this, it is necessary to establish the variables involved.



```
INIT y0: 0.0 , angle: 0.0 v_x_0: 1.0 , v_y_0: 0.0 , bounce: 1.0 , drag: 0.0
[FATAL] Fatal py exception encountered: [doStep] PyObject_CallMethod
'float division by zero'
Traceback (most recent call last):
  File "c:\Users\josee\Downloads\bouncing_ball\build_it.py", line 31, in <module>
    build('bouncing_ball.py', testIt=True)
  File "c:\Users\josee\Downloads\bouncing_ball\build_it.py", line 24, in build
    result = simulate_fmu(asBuilt.name, stop_time=2, step_size=0.04, solver='Euler', start_values={ 'angle0':0, 'bounceFactor':1, 'drag':0, 'v0':1, 'y0':0})
  File "C:\Users\josee\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\fmpy\simula
    result = simulateCS(model_description, fmu, start_time, stop_time, relative_tolerance, start_values, apply_default_start_values, input, output, output_in
rn_allowed, validate, initialize, terminate)
```

Figure 3.2: Terminal Simulation Error with $\alpha_0 = 0$

Factor Definition

Following the Design of Experiments methodology, it is necessary to identify the factors, or independent variables, involved. This particular model has five variables that can be considered for the study. Nevertheless, it has been decided to work only with three. Three factors are enough to find interactions or interference among them but still not too many to require extensive analysis that may lead to wrong conclusions. In addition, three variables allow using graphical tools to visualize the outcomes.

Since the **Bouncing Factor** (Bf) does not interfere with the objective of this study, because the useful data will be only until the ball reaches its first bounce, it is immediately the first factor to stay off the study. This variable was set as 1 and it is mentioned for control purposes only.

From the rest of the variables, the **Height** was selected to remain constant and equal to 0 ($y_0 = 0$). The reason for this decision is purely convenience since from the theory it is well known that without the consideration of any drag forces, $Df = 0$, we know that the angle that maximizes the distance is 45^o for any velocity values, so it is easy to establish a control output with these parameters.

Consequently, the factors that are considered in the study are the **Velocity, Drag Factor**, and **Angle**.

Level of the Factors

The next step is time to define the level of the selected factors. Starting with the **Angle** factor the levels chosen range from 1^o to 90^o. This decision is supported by the objective defined and the factors decision. The lower value is 1 because the model does not support $\alpha_0 = 0$ while $y_0 = 0$. It does not run. Figure 3.2 shows the terminal output. This behavior corresponds with reality because the ball will never bounce due to its starting position at 0 and no vertical speed. The maximum value was chosen as 90^o because from that value the vertical movement is symmetric to the previous ones ($\sin(90-x)^o = \sin(90+x)^o \ \forall x \in [0,90]$) and the horizontal movement only changes direction thus, in combination, these values will provide only redundant information.

For the Velocity and the Drag Factor the range of the values were chosen arbitrarily. There are no other criteria for the selection other than maintaining the common idea of having positive values. For the **Velocity** factor, the the range from 1 to 5 [m/s] was chosen; for the **Drag Factor** factor, the range level was 0 to 30 [m^{-1}].

Response Variable Definition

Since the study's objective is to find an angle that maximizes the distance traveled, it is evident that the response variable has to be the **horizontal distance** x.

Experimental Data Collection

The experimental design type chosen for collecting the data is the uniform grid sampling or leveling experimental design. In this, the levels of the factors are chosen such that the experimental points are uniformly spaced along the range of the factors. This allows for better estimation of the response between the levels of the factors and reduces the possibility of extrapolation errors.

One simulation with the combination of the following parameters was carried out to gather the initial data.

1. Height $y_0 = [0]$
2. Bouncing Factor $Bf = [1]$
3. Velocity $v_0 = [1,2,3,4,5]$
4. Drag Factor $Df = [0,5,10,15,20,25,30]$
5. Angle $\alpha_0 = [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90]$

3.3.2 Machine Learning set up

This machine learning problem lies in the category of Supervised Learning because it involves labeled data. This means the algorithms learn to map input data (that include the factors) to predict output using the "labeled training dataset" obtained from the experimental data collection. Further on identifying the task, the surrogate models must predict the variable numerical value, therefore, a regression algorithm is applied.

The programming language used is Python, and some libraries were used to facilitate the coding. Scikit-learn is the library used for the artificial intelligence approach, allowing the application of machine learning algorithms and training models easily. Pandas is the library for data manipulation and Seaborn with Matplot.Pyplot for creating the graphics.

Machine learning Algorithms

Scikit-learn has implemented several methods under the supervised learning regression category. In this study, only five of them are implemented. The models used are listed as follows:

- Kernel ridge regression (KRR)
- Support Vector Machine regression (SVM)
- Neighbors-based regression (KNN)
- Gaussian Processes for regression (GP)
- Decision Trees applied to regression (DT)

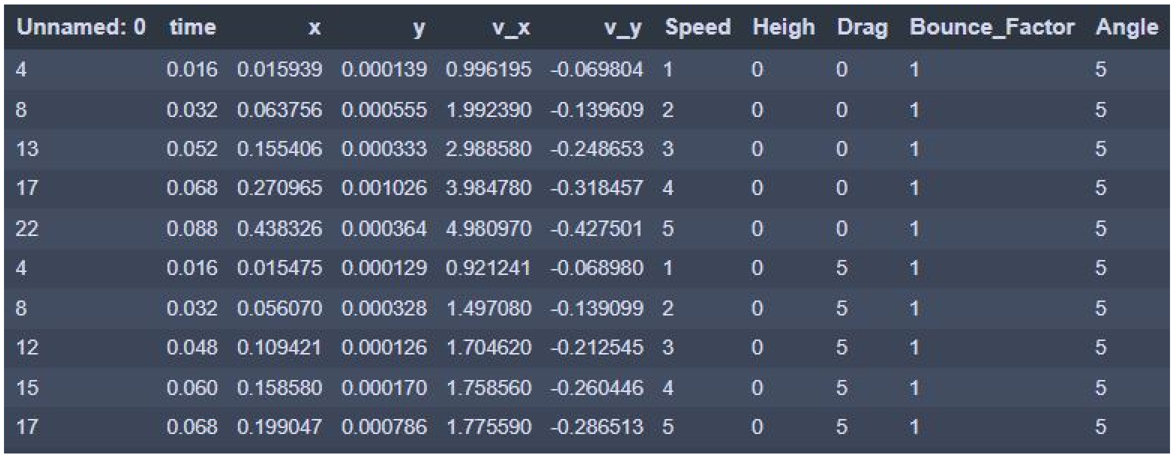
All linear algorithms were set aside due to their limitations regarding the prediction of nonlinear processes.

Training

The basis for the Machine Learning algorithms is the necessity of data to be trained and tested, also called dataset. In this experiment, to construct the dataset, a simple approach was taken and the steps used to obtain it were the following:

1. Perform the simulations for every combination of the parameters established in the experimental data collection. During this step, three combinations ($\alpha_0 = 5$, $v_0 = 1$ and $Df = 20, 25, 30$) failed to run due to model limitations regarding step size, so 627 files with data were generated.
2. Run an algorithm to find the line position when the variable v_y (vertical speed) changes its direction (changes from a negative value to a positive one) and the vertical position y is very close to 0. This indicates the first bounce.
3. Take the values of the line that contains the first bounce, add the correspondent fields of the initial parameters, also called factors in the experiment, and add it to a new file, the dataset.

The full code can be found in the appendix. Figure 3.3 shows the head of the Dataset generated. It has all the tracking variables during the first bounce for the setup of the five parameters. In addition, data frames normally have an index column ('Unnamed: 0' in this case) but are not relevant to this work. The last step before training is splitting the information into the input variables (columns with the factors) and the output variable i.e. column **x**.



Unnamed: 0	time	x	y	v_x	v_y	Speed	Heigh	Drag	Bounce_Factor	Angle
4	0.016	0.015939	0.000139	0.996195	-0.069804	1	0	0	1	5
8	0.032	0.063756	0.000555	1.992390	-0.139609	2	0	0	1	5
13	0.052	0.155406	0.000333	2.988580	-0.248653	3	0	0	1	5
17	0.068	0.270965	0.001026	3.984780	-0.318457	4	0	0	1	5
22	0.088	0.438326	0.000364	4.980970	-0.427501	5	0	0	1	5
4	0.016	0.015475	0.000129	0.921241	-0.068980	1	0	5	1	5
8	0.032	0.056070	0.000328	1.497080	-0.139099	2	0	5	1	5
12	0.048	0.109421	0.000126	1.704620	-0.212545	3	0	5	1	5
15	0.060	0.158580	0.000170	1.758560	-0.260446	4	0	5	1	5
17	0.068	0.199047	0.000786	1.775590	-0.286513	5	0	5	1	5

Figure 3.3: Original dataset overview

3.3.3 Analysis of the Results

The last phase in the methodology is to explain how to analyze the results. The procedure to analyze the results starts with comparing multiple models for each machine learning methodology, studying their performance through different sets of values. Then a comparison of the best of each model identifies its strength and weaknesses.

For the selection of the best models, two approaches are used. Graphical analysis in conjunction with a mathematical comparison of the errors of each model. The metric used for accuracy purposes is the Mean Absolute Error (MAE), and for robustness, is the Median Absolute Error.

The reasons for choosing MAE as the accuracy indicator are:

1. Robustness to outliers: MAE is less sensitive to outliers than MSE, and since the experiments are not focused on optimizing the models is better to use this metric.
2. Intuitive interpretation: MAE provides a clear interpretation of the average error magnitude.
3. Simplicity: MAE is a straightforward metric, easy to understand and also is computationally more efficient since it does not involve squaring the errors.

The reasons for choosing Median Absolute Error as the robustness indicator are:

1. Robustness: This is the least sensitive measurement to outliers that every other metric for regression in the Scikit learn package.

2. Insensitivity to error magnitude: is insensitive to error magnitudes and only considers their relative ordering. This property can be desirable in situations where the exact magnitude of errors is less important than their rank or order.

3. Intuitive interpretation: Similar to MAE, the Median absolute error provides a clear interpretation of the typical size of errors made by the model.

Chapter 4

Results

In this chapter, we will delve into the results obtained from the data analysis, which was conducted by the established methodology outlined in the preceding chapters, by organizing and comparing the outcomes of every model. These findings will contribute to the existing knowledge in the field and offer insights for future investigations.

The presentation of results will be structured logically, starting with the comparison of models trained with the same methodology but different hyperparameters and their performance as surrogate models with the same configurations as the dataset and after the same comparison with different configurations. Subsequently, the results obtained from each experiment will be presented, accompanied by appropriate visual aids, such as tables and graphs, to facilitate comprehension and enhance the clarity of the findings.

Ultimately, the results chapter aims to provide a comprehensive understanding of the empirical evidence generated from the study. It is the foundation upon which subsequent discussions, interpretations, and conclusions are built, enabling others to draw inferences and make informed decisions based on the study's outcomes.

4.1 General performance with the same level of factors as the trained

Subsequently, the results obtained from each experiment will be presented, accompanied by appropriate visual aids, such as tables and graphs, to facilitate comprehension and enhance the clarity of the findings. It is expected that the surrogate models should be able to predict relatively well since the points are very close to the original data. In the experiment, a prediction of the distance x for every 1-degree angle is performed to catch how the different algorithms perform extrapolating the points. (trained data only has responses for every 5 degrees). The conditions for the first comparison, in consequence, are the following:

1. Height $y_0 = [0]$
2. Bouncing Factor $Bf = [1]$
3. Velocity $v_0 = [1, 2, 3, 4, 5]$
4. Drag Factor $Df = [0, 5, 10, 15, 20, 25, 30]$
5. Angle $\alpha_0 = [1, 2, 3, ..., 88, 89, 90]$

This experiment is addressed as "First Comparison" in further references.

4.1.1 Support Vector Machine Regression Models

One of the main characteristics of SVM regression is that it is less sensitive to outliers than other regression methods. This feature will affect the performance in this experiment since the data does not contain natural out layers or noise. In addition, careful tuning of hyperparameters is required, especially the regularization parameter (denoted **C**) and the kernel function, which can affect the performance of the model. Since this is not a thorough study of how to properly fit a model only general advice was used to select the hyperparameters. Two are the most important hyperparameters that need to be decided. For the kernel, the choice was Radial Basis Function (RBF), since linear kernels were discarded due to the nature of the process itself (it is known a priory that a ball trajectory with acceleration is not a linear behavior) and also polynomial kernels are more computationally expensive. Then, following the literature recommendations, the **C** hyperparameter is set between 10^{-3} and 10^{3}, allowing the training of seven different models. Figure 4.1 shows the results of predicting the seven models for the parameters

It can be appreciated that models with the lowest **C** value (see pink and brown points) tend to stay very linear, since they are more tolerant of errors and allow more points to surpass the margins. On the other extreme, the blue and orange points, which represent the models with the highest value of **C** show curves that, in some cases, are very accurate in comparison to the real model, for example in the cases with $Df = 0$ but in other are completely far from the original model behavior. This can easily be spotted on $Df = 15$ and lower speeds in which the models behave opposite, presenting a "U" shape.

On the numerical analysis, it is no surprise that the models with high **C** values obtained the lowest value of mean absolute error value. In fact, the higher the **C** value on the model, the lower the error is. Nevertheless, analyzing the median absolute error results, the models with the best score, i.e., the lowest value of error, change. These results indicate that the most accurate model

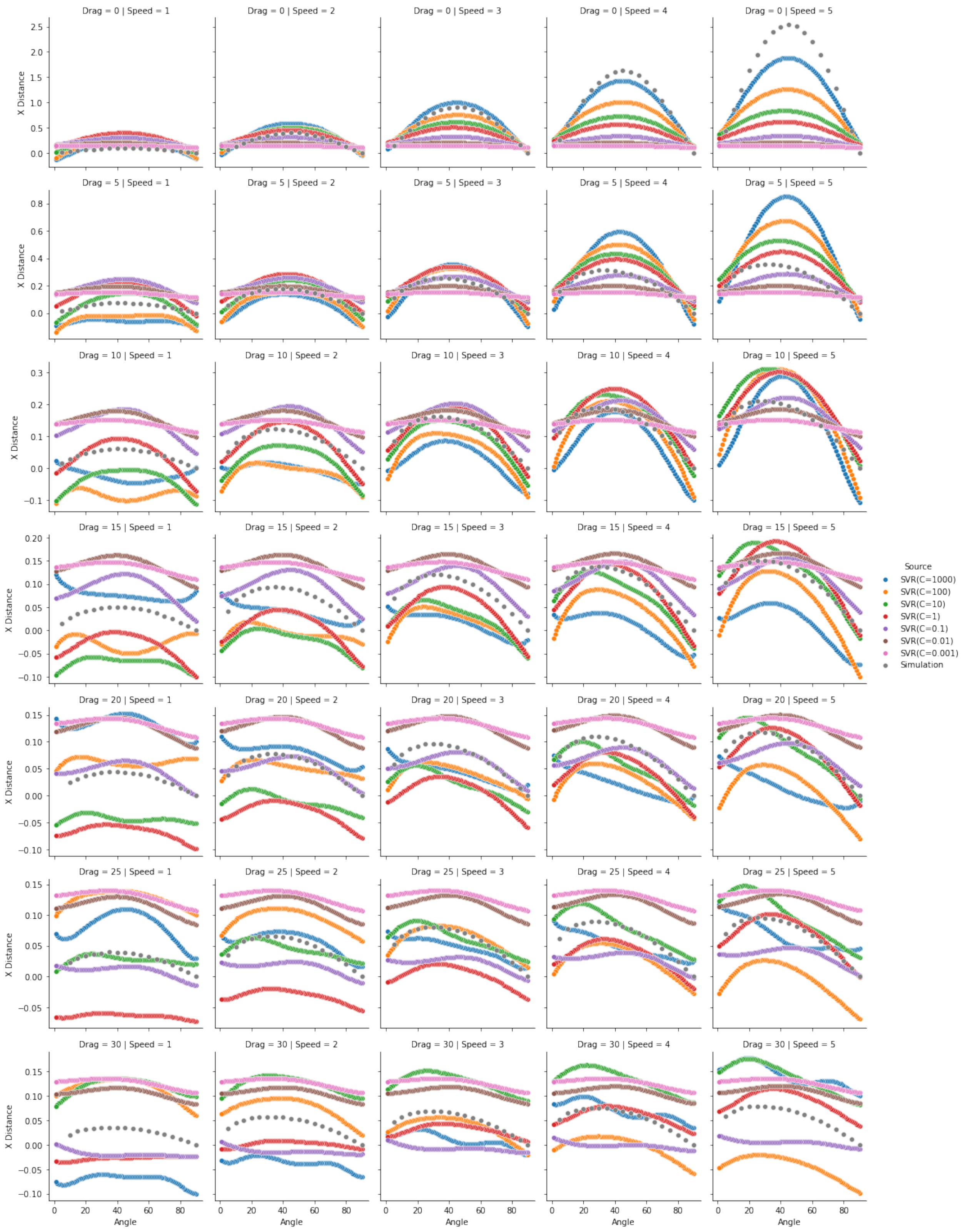
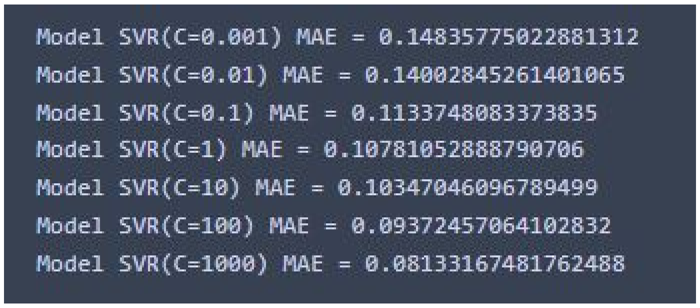


Figure 4.1: Performance of SVM regression models, first comparison

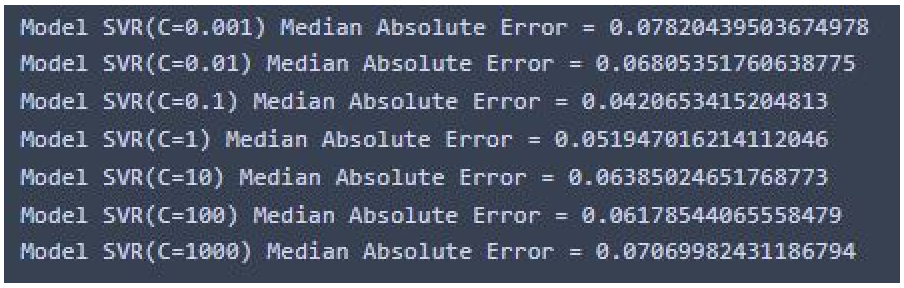
is the SVR(C=1000) but is the second worst in robustness and the model SVR(C=0.1) is the most robust but the third worst in accuracy.

Figures 4.2 and 4.3 show the mean absolute error and the median absolute error values of the models. The graphical and numerical analysis combination suggests that these SVM models do not surrogate properly the original model in this first comparison experiment.



```
Model SVR(C=0.001) MAE = 0.14835775022881312
Model SVR(C=0.01) MAE = 0.14002845261401065
Model SVR(C=0.1) MAE = 0.1133748083373835
Model SVR(C=1) MAE = 0.10781052888790706
Model SVR(C=10) MAE = 0.10347046096789499
Model SVR(C=100) MAE = 0.09372457064102832
Model SVR(C=1000) MAE = 0.08133167481762488
```

Figure 4.2: Mean absolute error for the SVM models, first comparison



```
Model SVR(C=0.001) Median Absolute Error = 0.07820439503674978
Model SVR(C=0.01) Median Absolute Error = 0.06805351760638775
Model SVR(C=0.1) Median Absolute Error = 0.0420653415204813
Model SVR(C=1) Median Absolute Error = 0.051947016214112046
Model SVR(C=10) Median Absolute Error = 0.06385024651768773
Model SVR(C=100) Median Absolute Error = 0.06178544065558479
Model SVR(C=1000) Median Absolute Error = 0.07069982431186794
```

Figure 4.3: Median absolute error for the SVM models, first comparison

4.1.2 Kernel Ridge Regression Models

Similar to SVM, Kernel Ridge Regression (KKR) also is based on a kernel function that aims to find a higher dimensional space to map the data and applied ridge regressions in that space. Computationally speaking, is less expensive than SVM based on the algorithms needed to train models. The same criteria apply in this set of models for the election of the kernel, a Radial Basis Function (RBF), and the regularization parameter in this methodology is called **alpha** and also is the most important hyperparameter. In this opportunity, the hyperparameter **gamma** that determines the weighting of every point was set with the value 0.1 to reduce the risk of overfitting and, once again, is one of the general tips found in the literature. Figure 4.4 shows the graphical results of the different KRR models.

The graphical analysis for the models evidences the behavior of the different models and compares them with the behavior of the original model. Similarly to the SVM regression models,

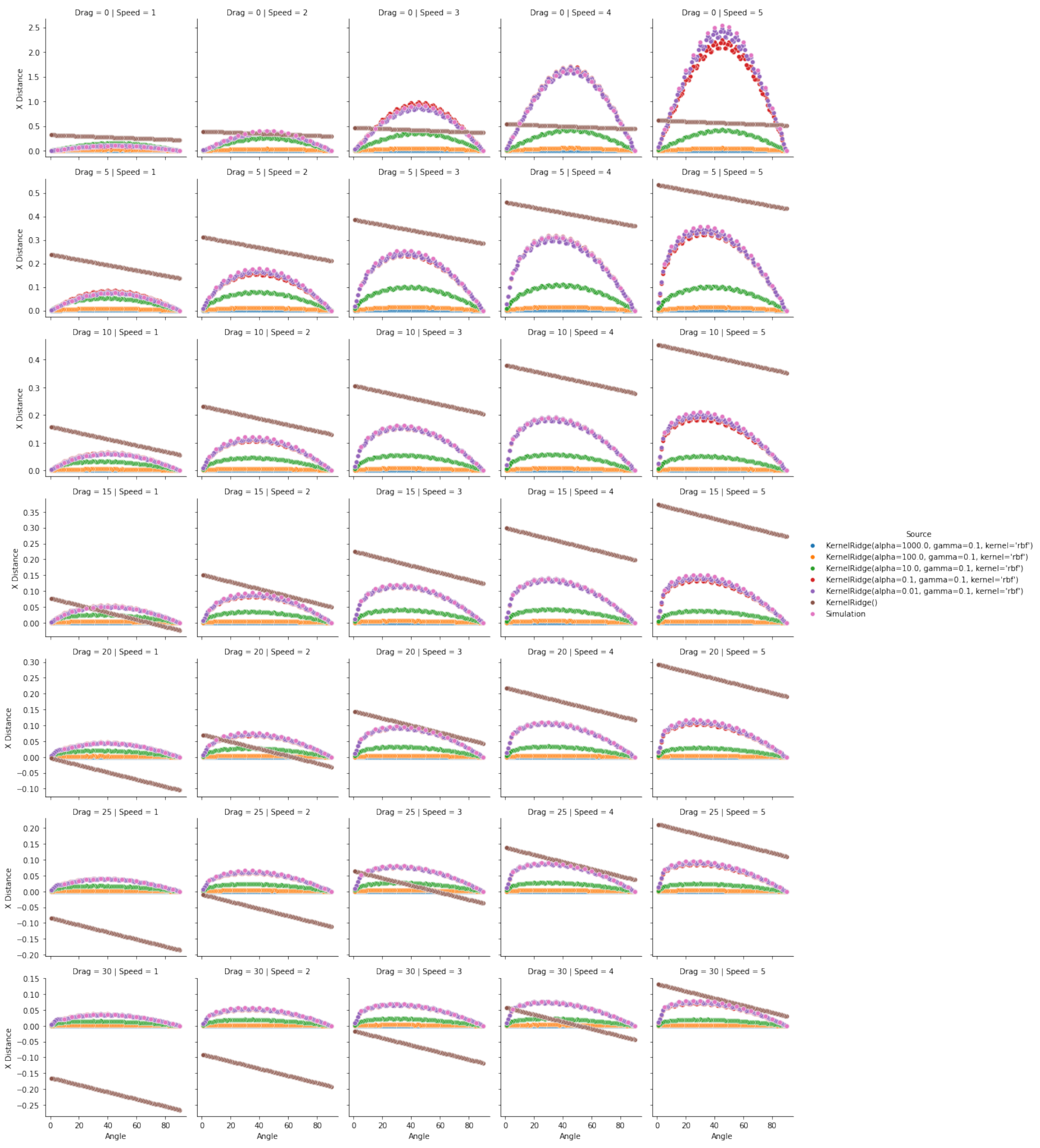
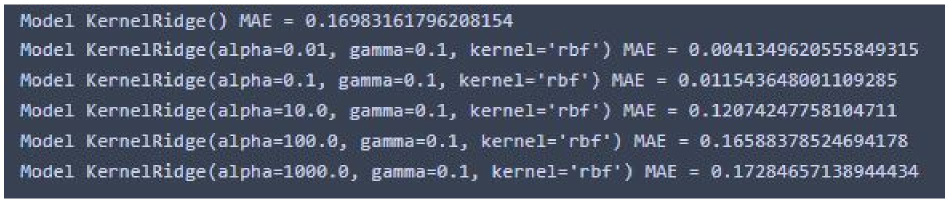


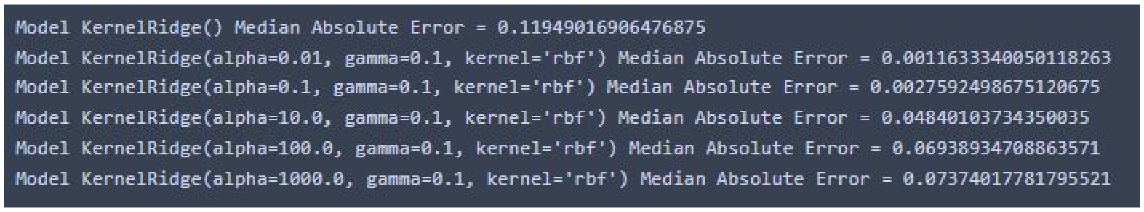
Figure 4.4: Performance of KR regression models, first comparison

the models with more strict regularization parameters tend to predict better. KR(alpha=0.01) and KR(alpha=0.1) seem to represent in a good way the original model and keep it constant through the different levels of the factors. Furthermore, reviewing the mathematical scores of the errors, the same models present lower values in accuracy and robustness. The figures 4.5 and 4.6 show the mean of these values and indicates a great performance of these models to surrogate the original.



```
Model KernelRidge() MAE = 0.16983161796208154
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') MAE = 0.0041349620555849315
Model KernelRidge(alpha=0.1, gamma=0.1, kernel='rbf') MAE = 0.011543648001109285
Model KernelRidge(alpha=10.0, gamma=0.1, kernel='rbf') MAE = 0.12074247758104711
Model KernelRidge(alpha=100.0, gamma=0.1, kernel='rbf') MAE = 0.16588378524694178
Model KernelRidge(alpha=1000.0, gamma=0.1, kernel='rbf') MAE = 0.17284657138944434
```

Figure 4.5: Mean absolute error for the KRR models, first comparison



```
Model KernelRidge() Median Absolute Error = 0.11949016906476875
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') Median Absolute Error = 0.0011633340050118263
Model KernelRidge(alpha=0.1, gamma=0.1, kernel='rbf') Median Absolute Error = 0.0027592498675120675
Model KernelRidge(alpha=10.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.04840103734350035
Model KernelRidge(alpha=100.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.06938934708863571
Model KernelRidge(alpha=1000.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.07374017781795521
```

Figure 4.6: Median absolute error for the KRR models, first comparison

4.1.3 K Nearest Neighbors Regression Models

The next set of models to analyze is the K Nearest Neighbors technique. One of the peculiarities of this methodology is that is a non-parametric algorithm. It makes predictions by finding the "K" closest data points (also called neighbors) in the training set to a given test data point and using their average (or weighted average) value as the predicted output. A very simple algorithm that is mainly used for classification problems, but in this case adapted to predict a numerical value.

The only hyperparameter needed to build a KNN model is ’K’, which indicates the number of neighbors to be considered. In Scikit Learn, this parameter is called ’n’. In this work, four different models were trained, with $n = \{3, 5, 7, 10\}$. By default, the ’n’ is set by 5 in Scikit Learn, so in the figures usually, this model only shows KNeighborsRegression().

Figure 4.7 shows the behavior of the model, and it can be notated that these models can consistently represent the behavior of the original model, however for some of the combinations of variables the models with the largest K value (blue and orange on the figure) tends to present a slightly chaotic behavior and scatter away from the rest. For instance, Figure 4.7 for $Df = 5$ and $V = 5$. There is also an interesting behavior where all models tend to overestimate the distance traveled for smaller velocities and sub-estimate for higher velocities. This lead to thinking that the best estimations are done for the middle ground level of the factors.

The figures 4.8 and 4.9 that show the error scores, confirm that the model KNeighborRegressor(n=3) is the most accurate and robust of the four trained which seems consistent with the graphical analysis.

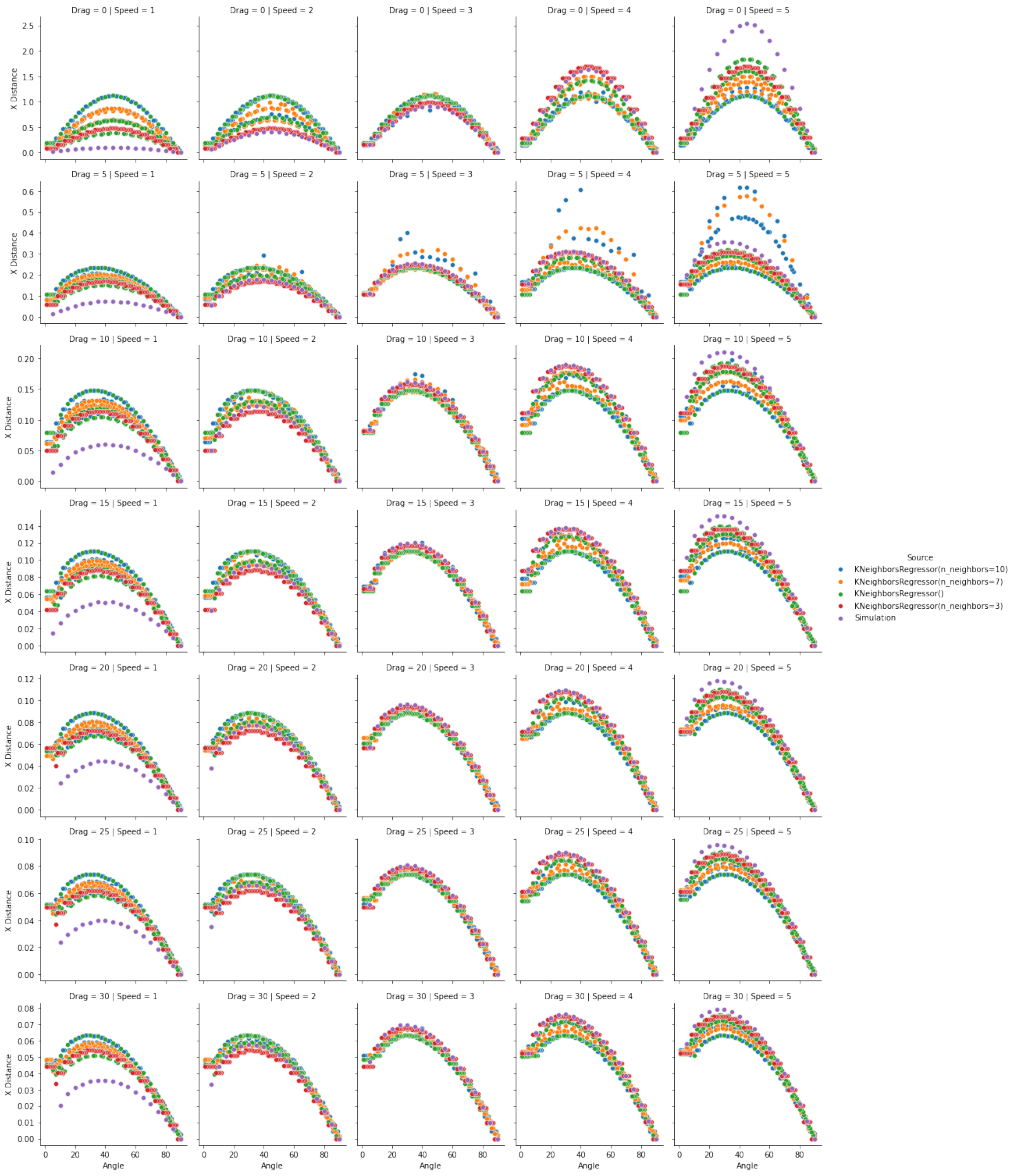
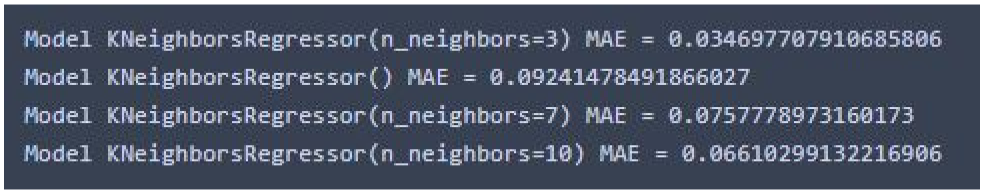
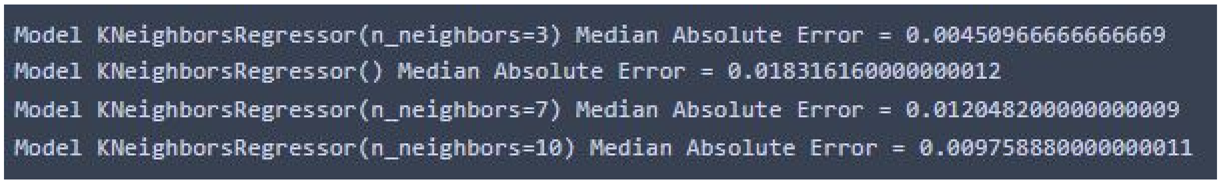


Figure 4.7: Performance of KNN regression models, first comparison



```
Model KNeighborsRegressor(n_neighbors=3) MAE = 0.034697707910685806
Model KNeighborsRegressor() MAE = 0.09241478491866027
Model KNeighborsRegressor(n_neighbors=7) MAE = 0.0757778973160173
Model KNeighborsRegressor(n_neighbors=10) MAE = 0.06610299132216906
```

Figure 4.8: Mean absolute error for the KNN models, first comparison



```
Model KNeighborsRegressor(n_neighbors=3) Median Absolute Error = 0.004509666666666669
Model KNeighborsRegressor() Median Absolute Error = 0.018316160000000012
Model KNeighborsRegressor(n_neighbors=7) Median Absolute Error = 0.012048200000000009
Model KNeighborsRegressor(n_neighbors=10) Median Absolute Error = 0.009758880000000011
```

Figure 4.9: Median absolute error for the KNN models, first comparison

4.1.4 Decision Trees Regression Models

Decision trees are another technique mainly used for classification problems but can also be extended to predict numerical values in their regression form. There is only one important parameter for the algorithm to be indicated and it is the 'max depth', which indicates the maximum amount of nodes that the model should create. By default, this value is blank, which indicates that the model reaches all the possible branches. In conjunction with the default model, the other four models were trained with "max depth"$= 2, 5, 6, 10$ to compare outcomes. Figure 4.10 show these outcomes.

As expected, lower "max depth" models seem very limited in their ability to predict accurately, in contrast, the model with no restriction tends to follow very closely the behavior of the original model simulations in all of the level combinations. Once again, these observations are corroborated with the error values presented in figure 4.11 and figure 4.12 where the values of both types of errors decrease while the max depth increases, reaching practically 0 error for the unrestricted model. This is not a major surprise, due to this algorithm including the points trained as outcomes, defining the branches.

4.1.5 Gaussian Processes Regression Models

The last method of machine learning to compare is the Gaussian Process (GP). As stated in the theoretical background, this algorithm provides a non-parametric approach that models the distribution over functions rather than explicit function approximations which allow for handling complex relationships and provides uncertainty estimates for predictions.

Training a GP model is challenging since choosing the kernel, also known as the covariance

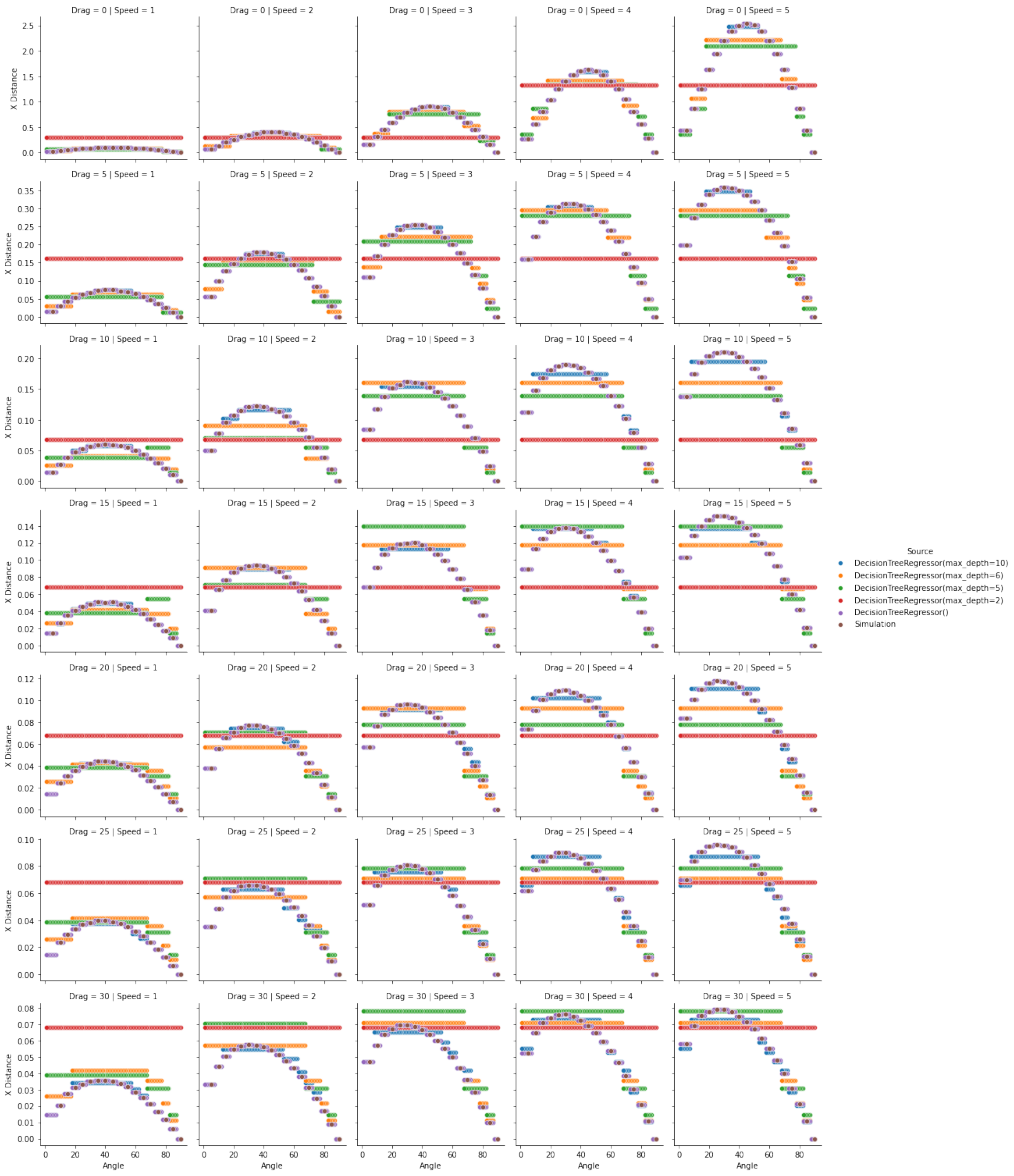
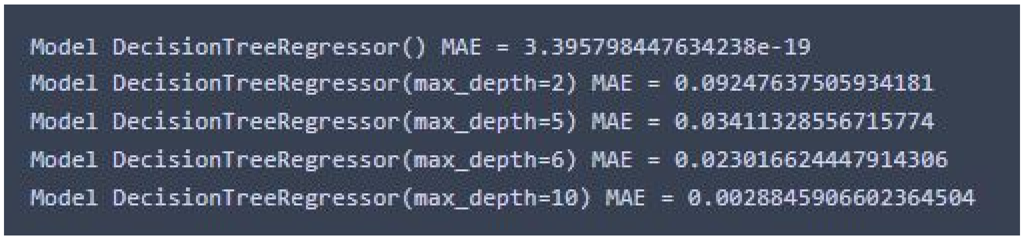
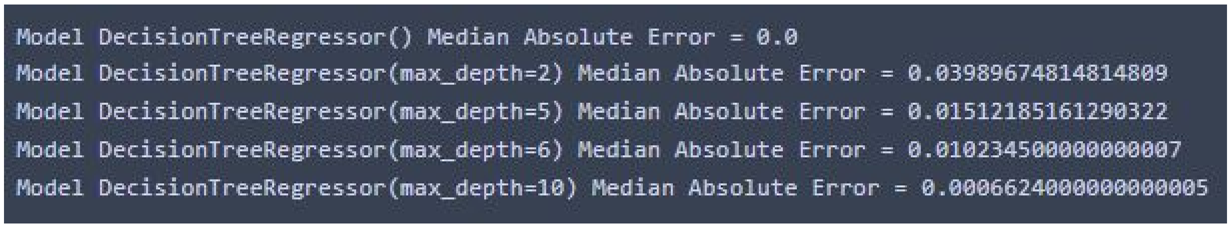


Figure 4.10: Performance of DT regression models, first comparison



```
Model DecisionTreeRegressor() MAE = 3.395798447634238e-19
Model DecisionTreeRegressor(max_depth=2) MAE = 0.09247637505934181
Model DecisionTreeRegressor(max_depth=5) MAE = 0.03411328556715774
Model DecisionTreeRegressor(max_depth=6) MAE = 0.023016624447914306
Model DecisionTreeRegressor(max_depth=10) MAE = 0.0028845906602364504
```

Figure 4.11: Mean absolute error for the DT models, first comparison



```
Model DecisionTreeRegressor() Median Absolute Error = 0.0
Model DecisionTreeRegressor(max_depth=2) Median Absolute Error = 0.03989674814814809
Model DecisionTreeRegressor(max_depth=5) Median Absolute Error = 0.01512185161290322
Model DecisionTreeRegressor(max_depth=6) Median Absolute Error = 0.010234500000000007
Model DecisionTreeRegressor(max_depth=10) Median Absolute Error = 0.0006624000000000005
```

Figure 4.12: Median absolute error for the DT models, first comparison

function or correlation function, is crucial when using Gaussian Processes. The kernel determines the assumptions about the underlying data and the functional relationships between input variables. The selection of an appropriate kernel depends on the specific problem domain, the nature of the data, and the desired characteristics of the GP model. The choice of kernel should be guided by the characteristics of the data, such as smoothness, periodicity, linearity, and prior knowledge about the problem. In practice, it is often beneficial to try multiple kernels and select the one that yields the best model performance through techniques like cross-validation or marginal likelihood optimization, nevertheless, that work requires substantial knowledge and experience working with GP and it is outside of this thesis objective. Therefore, in this experiment the GP models trained were using 3 different kernel functions independently: Radial Basis Function or RBF, Dot Product, and Rational Quadratic. Figure 4.13 shows the outcomes of these models.

The models with RBF and Rational Quadratic kernel mimic the behavior of the original model very consistently, only with a small variation for small angles, where the Rational Quadratic Model tends to deviate from predicting higher values than the original model simulations. On the other hand, the pure dot product kernel has a very low attachment to the original. This was expected due to its nature as a covariance function and specialization in capturing linear relationships.

The error values, confirming the graphical analysis, show values near 0 for both of the models, but if it only considered the magnitude of these results the best model, in terms of accuracy and robustness is the rational quadratic model, an unexpected result considering the graphical observations done regarding the prediction of this for smaller angles for all factor combinations.

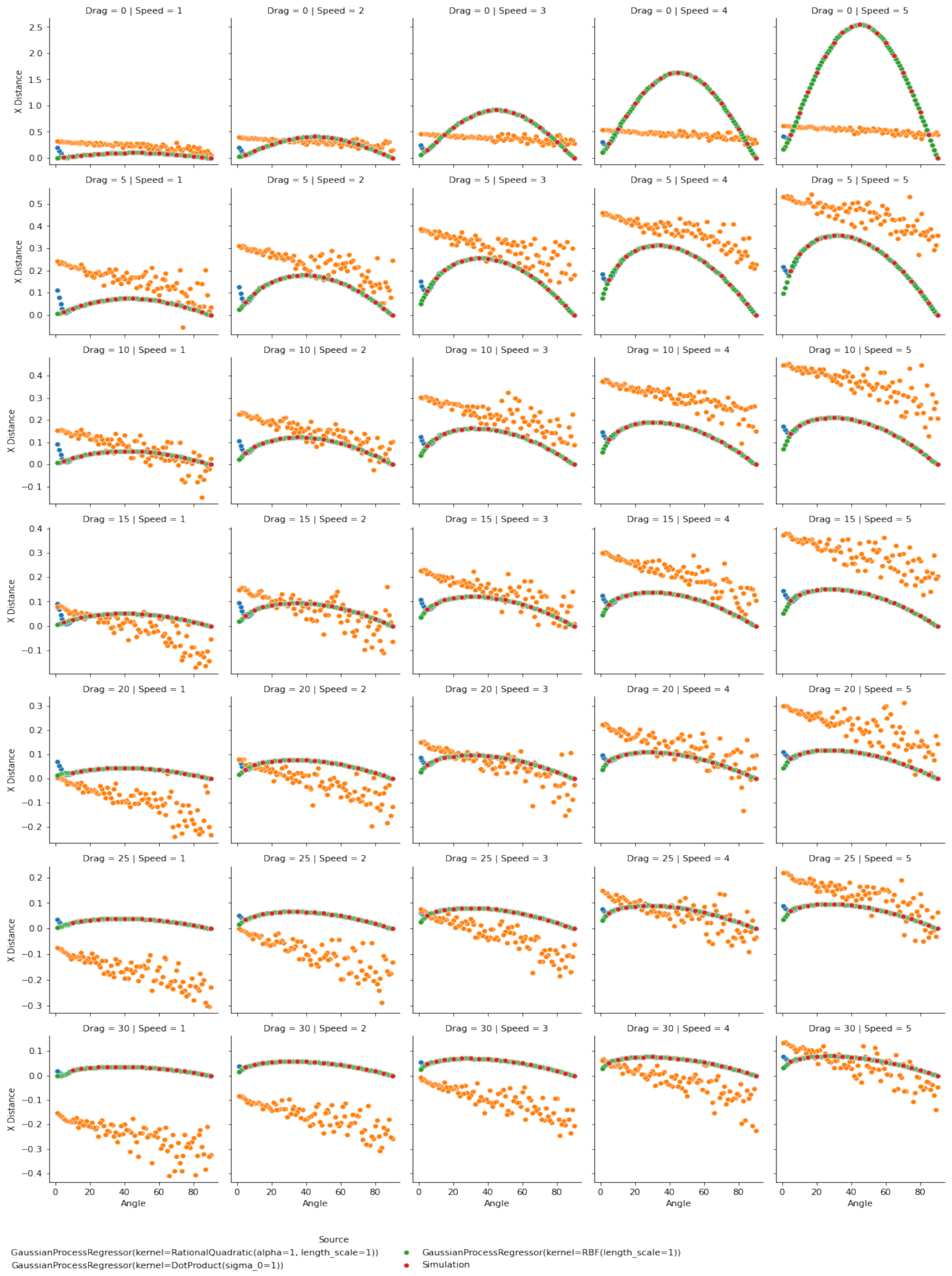
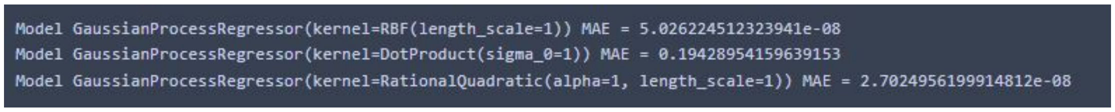


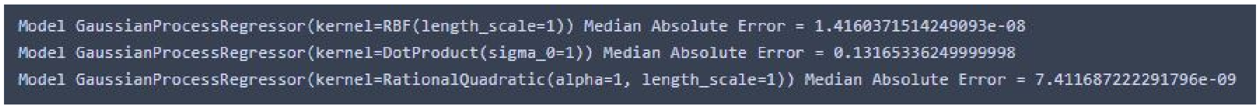
Figure 4.13: Performance of GP regression models, first comparison

The detailed values of the Mean and Median Absolute Errors can be found in figures 4.14 and 4.15 respectively.



```
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) MAE = 5.026224512323941e-08
Model GaussianProcessRegressor(kernel=DotProduct(sigma_0=1)) MAE = 0.19428954159639153
Model GaussianProcessRegressor(kernel=RationalQuadratic(alpha=1, length_scale=1)) MAE = 2.7024956199914812e-08
```

Figure 4.14: Mean absolute error for the GP models, first comparison



```
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) Median Absolute Error = 1.4160371514249093e-08
Model GaussianProcessRegressor(kernel=DotProduct(sigma_0=1)) Median Absolute Error = 0.13165336249999998
Model GaussianProcessRegressor(kernel=RationalQuadratic(alpha=1, length_scale=1)) Median Absolute Error = 7.411687222291796e-09
```

Figure 4.15: Median absolute error for the GP models, first comparison

4.2 General performance with an interpolated level of factors

For a second experiment, the levels of the factors differ from those used for training. The original model simulated all the combinations for the middle points on the factor levels than the original dataset. This means that the data to compare results was carried out with the following values:

- Height $y_0 = [0]$
- Bouncing Factor $Bf = [1]$
- Velocity $v_0 = [1.5, 2.5, 3.5, 4.5]$
- Drag Factor $Df = [2.5, 7.5, 12.5, 17.5, 22.5, 27.5]$
- Angle $\alpha_0 = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90]$

For references purposes, this comparison is addressed as "Second Comparison" and the same patterns of analysis in the previous comparison are used here, which means that the models are predicting outcomes for the same factor levels listed before, but for every 1 whiting the range of the angle level, i.e:

- Angle $\alpha_0 = [1, 2, 3, ..., 88, 89, 90]$

4.2.1 Support Vector Machine Regression Models

The performance for the second comparison of the SVM regression models is shown in Figure 4.16. Once again the graphical analysis shows that SVM models cannot reply consistently to the behavior of the original model regardless of the value of the C parameter. There is not a single model that can predict accurately nor robustly the simulation's behavior.

The error values in this occasion present different results compared with the first comparison. Regarding accuracy, now the middle values C models present the best performance, being the model SVR(C=0.1) the one with the less score. Similar behavior is presented in the robust measurement, where once again model SVR=(0.1) presents the best value even when all the models score alike. Details on the score values can be found in Figure 4.17 for Mean Absolute Error and 4.18 for Median Absolute Error. These results only come to reaffirm the previous analysis that these types of algorithms (with only the variation of the C hyperparameter) do not properly surrogate this model.

4.2.2 Kernel Ridge Regression Models

Kernel Ridge regression models behave very similarly to the first comparison for this second experiment. The Model KR(alpha=0.01) presents the best scores regarding errors and also the model seems to mimic better the behavior of the simulations. Details on the graphical performance of these models are shown in Figure 4.19. The figure evidence that the models with the small alpha (purple and red) follow very closely the behavior of the original model in various setups but tend to be less precise for the combinations with small Drag Factor and High values of Speed.

The error metrics are displayed in Figures 4.20 and 4.21. These are consistent with the first comparison experiment. Also, it can be noticed that the models KR(alpha=0.01) and KR(alpha=0.1) present very similar results, being the models with the lower values in accuracy and robustness.

4.2.3 K Nearest Neighbors Regression Models

Based on the graphical performance of KNN models shown in Figure 4.22 once again, models tend to represent fairly accurate simulation behavior. Maintaining the tendency of the model KNeighbors(n=10) to overestimate for lower values of factors and estimate for higher combinations, nevertheless, the other models seem to estimate better among this range of variables even though the curve that presents is more "spiky" than smooth.

Figure 4.23 shows the Mean Absolute Error for the models. In this opportunity, the model KNeighbors(n=5) is the one that presents the lower score, and Figure 4.24 shows the Median Ab-

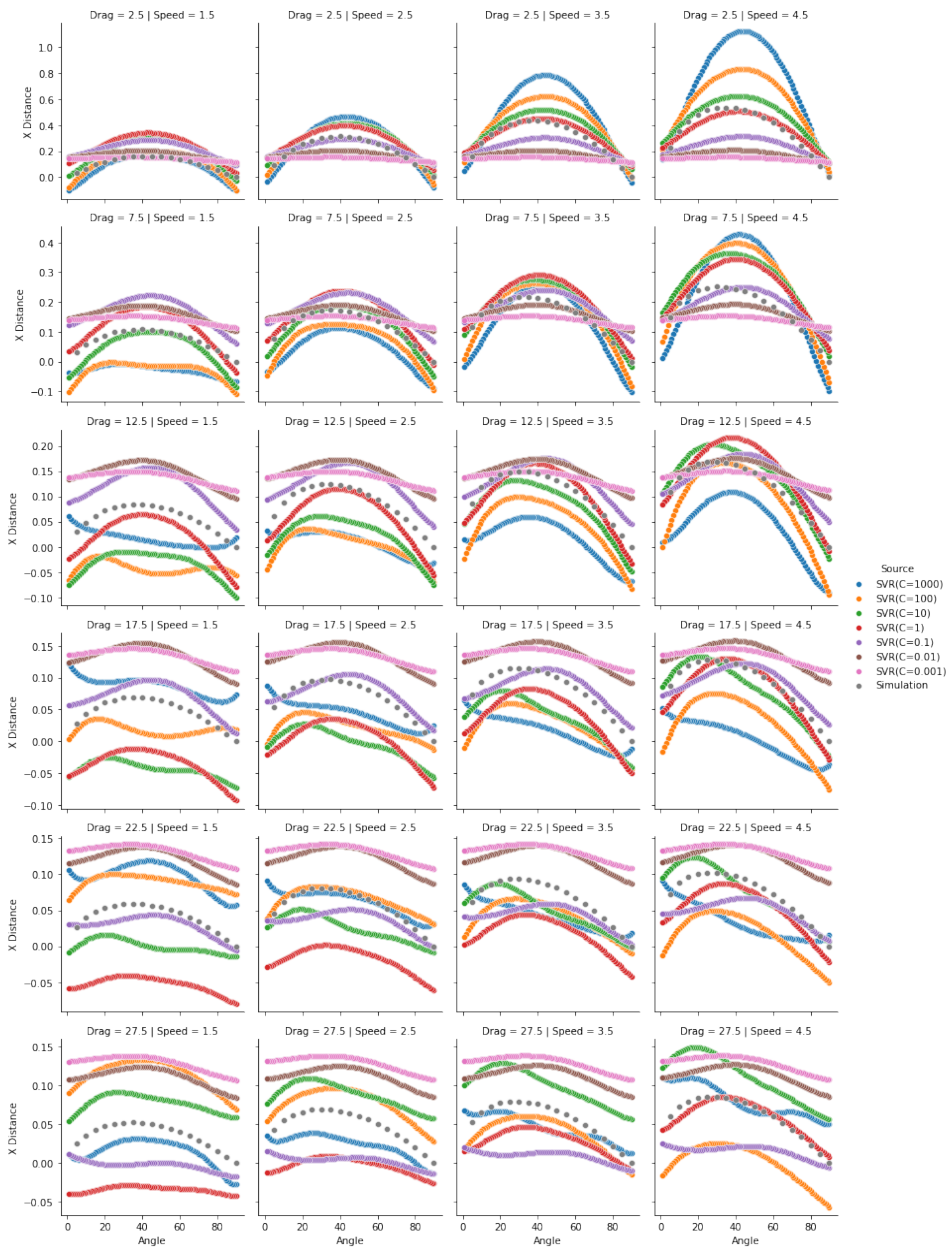
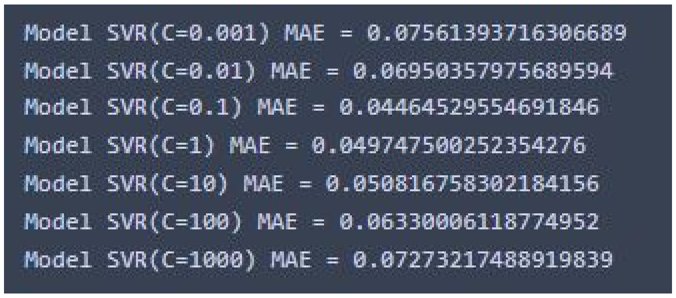
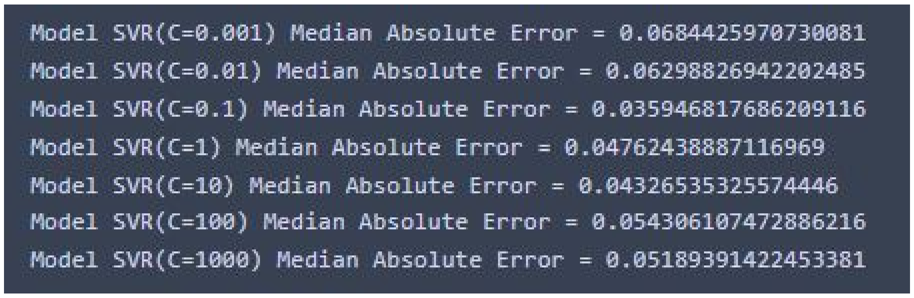


Figure 4.16: Performance of SVM regression models, second comparison



```
Model SVR(C=0.001) MAE = 0.07561393716306689
Model SVR(C=0.01) MAE = 0.06950357975689594
Model SVR(C=0.1) MAE = 0.04464529554691846
Model SVR(C=1) MAE = 0.049747500252354276
Model SVR(C=10) MAE = 0.050816758302184156
Model SVR(C=100) MAE = 0.06330006118774952
Model SVR(C=1000) MAE = 0.07273217488919839
```

Figure 4.17: Mean Absolute Error for the SVM models, second comparison



```
Model SVR(C=0.001) Median Absolute Error = 0.0684425970730081
Model SVR(C=0.01) Median Absolute Error = 0.06298826942202485
Model SVR(C=0.1) Median Absolute Error = 0.035946817686209116
Model SVR(C=1) Median Absolute Error = 0.04762438887116969
Model SVR(C=10) Median Absolute Error = 0.04326535325574446
Model SVR(C=100) Median Absolute Error = 0.054306107472886216
Model SVR(C=1000) Median Absolute Error = 0.05189391422453381
```

Figure 4.18: Median Absolute Error for the SVM models, second comparison

solute Error for the models. For this metric, the model KNeighbors(n=3) is the one that presents the lower score.

4.2.4 Decision Trees Regression Models

The performance for decision tree models keeps presenting a good mimic of the original model even though the accuracy for some combinations (i.e. $Df = 2.5$ and $V_0 = 4.5$) was very distant from the simulations. Once again the model without the depth restriction seems to be the one with the closest behavior. The overall performance of these models in the second comparison is shown in Figure 4.25, and it can be appreciated that the output function can be associated with a step function.

For the accuracy measure, the unrestricted model remains to present the lowest value (check Figure 4.26) and also is keeping the first place in the Median Absolute Error as Figure 4.27 shows.

4.2.5 Gaussian Processes Regression Models

GP models remain among the best-performing models in imitating the original model behavior. However, it can be appreciated that the quality of the predictions for these combinations is inferior to the one shown in the first comparison. See, for example, the combination $Df = 7.5$ and $V_0 = 4.5$ in Figure 4.28 how the behavior of the surrogate models differs from the original model. In addition to being away from the numerical values the curve that the surrogate describes for

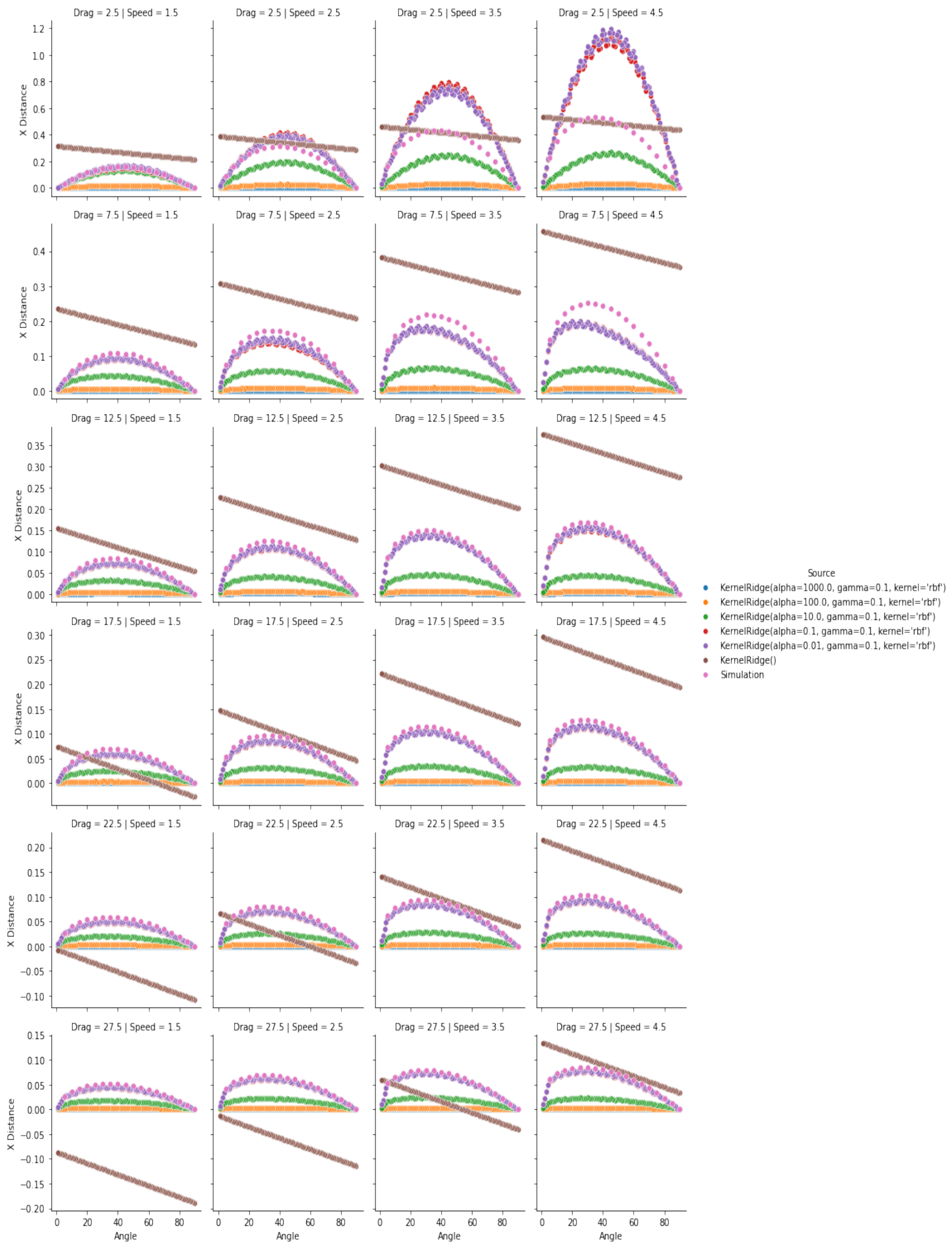
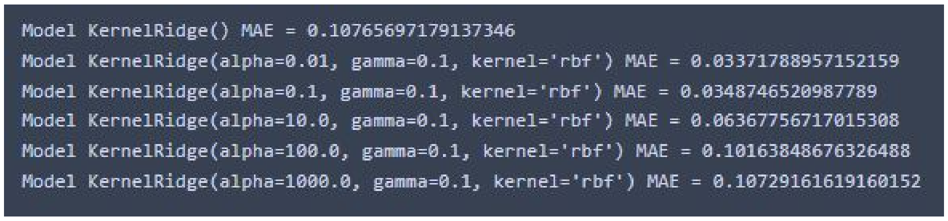
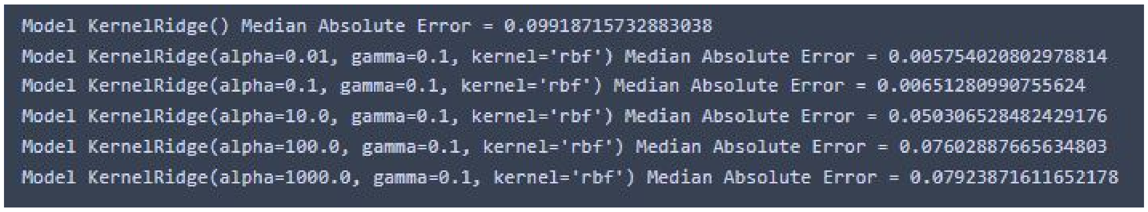


Figure 4.19: Performance of KR regression models, second comparison



```
Model KernelRidge() MAE = 0.10765697179137346
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') MAE = 0.03371788957152159
Model KernelRidge(alpha=0.1, gamma=0.1, kernel='rbf') MAE = 0.0348746520987789
Model KernelRidge(alpha=10.0, gamma=0.1, kernel='rbf') MAE = 0.06367756717015308
Model KernelRidge(alpha=100.0, gamma=0.1, kernel='rbf') MAE = 0.10163848676326488
Model KernelRidge(alpha=1000.0, gamma=0.1, kernel='rbf') MAE = 0.10729161619160152
```

Figure 4.20: Mean Absolute Error for the KR models, second comparison



```
Model KernelRidge() Median Absolute Error = 0.09918715732883038
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') Median Absolute Error = 0.005754020802978814
Model KernelRidge(alpha=0.1, gamma=0.1, kernel='rbf') Median Absolute Error = 0.00651280990755624
Model KernelRidge(alpha=10.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.050306528482429176
Model KernelRidge(alpha=100.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.07602887665634803
Model KernelRidge(alpha=1000.0, gamma=0.1, kernel='rbf') Median Absolute Error = 0.07923871611652178
```

Figure 4.21: Median Absolute Error for the KR models, second comparison

that combination is very different from the original one.

Despite the graphical analysis, the errors for these models keep scoring very low and similar values. Detailed scores can be checked in Figures 4.29 and 4.30.

4.3 Performance on the specific problem

Based on the general performance analyzed in the previous comparisons, one model for each machine learning methodology was chosen except for the SVM model due to the discrepancies observed. The surrogate models analyzed here are listed below:

- GaussianProcessRegressor(kernel=RBF())
- DecisionTreeRegressor()
- KNeighborsRegressor(3)
- KernelRidge(kernel=RBF, gamma=0.1, alpha=0.01)

These models now are measured with the study's objective of finding the angle that maximizes the horizontal distance traveled before the first bounce. To start with this step, the performance of these models is put into perspective. Therefore, Figure 4.31 shows the graphical performance of the models on the first comparison setup and Figure 4.32 on the second comparison.

It can be noticed that on the first comparison, the models generally are very close to each other and also close to the values of the simulation; the KNN model nevertheless, seems to be

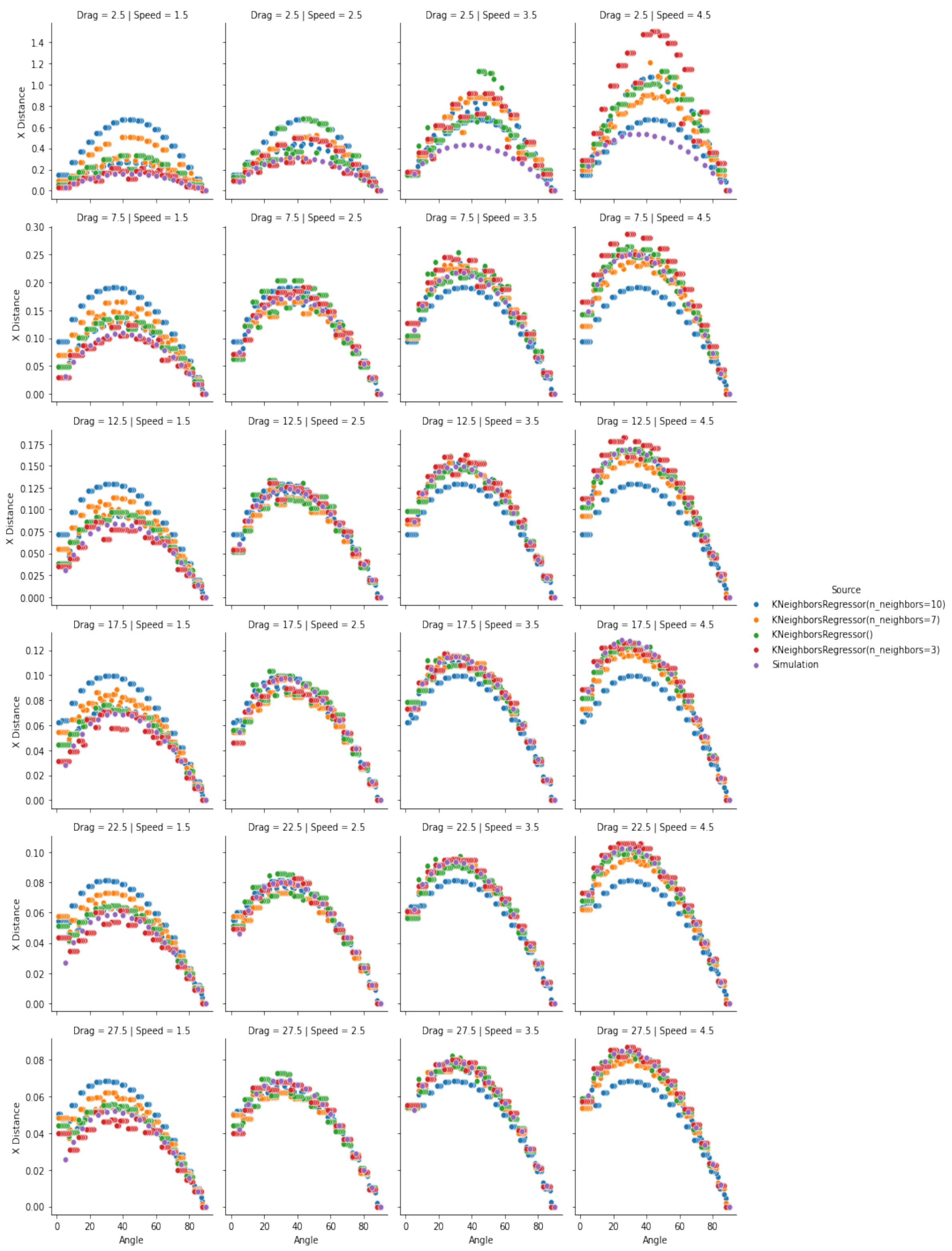
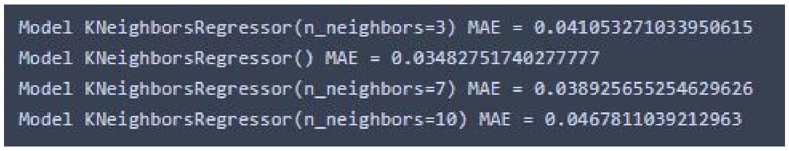
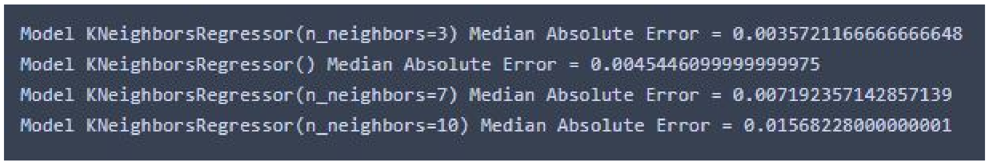


Figure 4.22: Performance of KNN regression models, second comparison



```
Model KNeighborsRegressor(n_neighbors=3) MAE = 0.041053271033950615
Model KNeighborsRegressor() MAE = 0.03482751740277777
Model KNeighborsRegressor(n_neighbors=7) MAE = 0.03892565525462962 6
Model KNeighborsRegressor(n_neighbors=10) MAE = 0.0467811039212963
```

Figure 4.23: Mean Absolute Error for the KNN models, second comparison



```
Model KNeighborsRegressor(n_neighbors=3) Median Absolute Error = 0.003572116666666648
Model KNeighborsRegressor() Median Absolute Error = 0.004544609999999975
Model KNeighborsRegressor(n_neighbors=7) Median Absolute Error = 0.007192357142857139
Model KNeighborsRegressor(n_neighbors=10) Median Absolute Error = 0.01568228000000001
```

Figure 4.24: Median Absolute Error for the KNN models, second comparison

one with less accuracy in the prediction of the value presenting a constant tendency to overestimate at speed $v = 1$ and sub estimate at speed $v = 5$.

For the comparison of the second experiment, it can be appreciated how the models struggle to represent certain combinations, the same that was already checked in the previous section.

4.3.1 Specific problem algorithm

To solve the specific problem a simple algorithm was programmed. It consists of predicting the x value for each angle in the range with step 1 degree and keeping the predictions in a variable as Data Frame, then looking through this variable and finding the maximum value and selecting the angle that produced that outcome. In order to produce more data to withdraw more robust conclusions, the surrogates predicted the target value for every one of the combinations from the first and second comparison factor values. This means that every surrogate model predicted a total of 10,530 values. (combination of 90 degrees, 7 levels of velocity, and 13 levels of Drag Factor). Figure 4.33 shows the errors for the models on the predictions of these values. The numbers evidence that the most accurate of the four models is the Kernel Ridge, followed closely by the Gaussian Process. Then, for the median absolute error, is the Gaussian Process model the most robust, placing in the second position the Kernel Ridge model.

Since the error calculated only take into consideration the predicted values, a manual approach was used in addition to the previous tools. This consisted of finding the real answer for the angle through the simulation using the original model. The same algorithm applied for the surrogate models was used with the original information finding the angle for all the configurations. This angle was compared with the angle finding in the surrogates and calculated as an error. These errors are shown as "Average Deviation" which considers the average value of the differences and also the "Max Deviation" which shows the maximum difference through all the combinations. These values have a dimension equal to a degree.

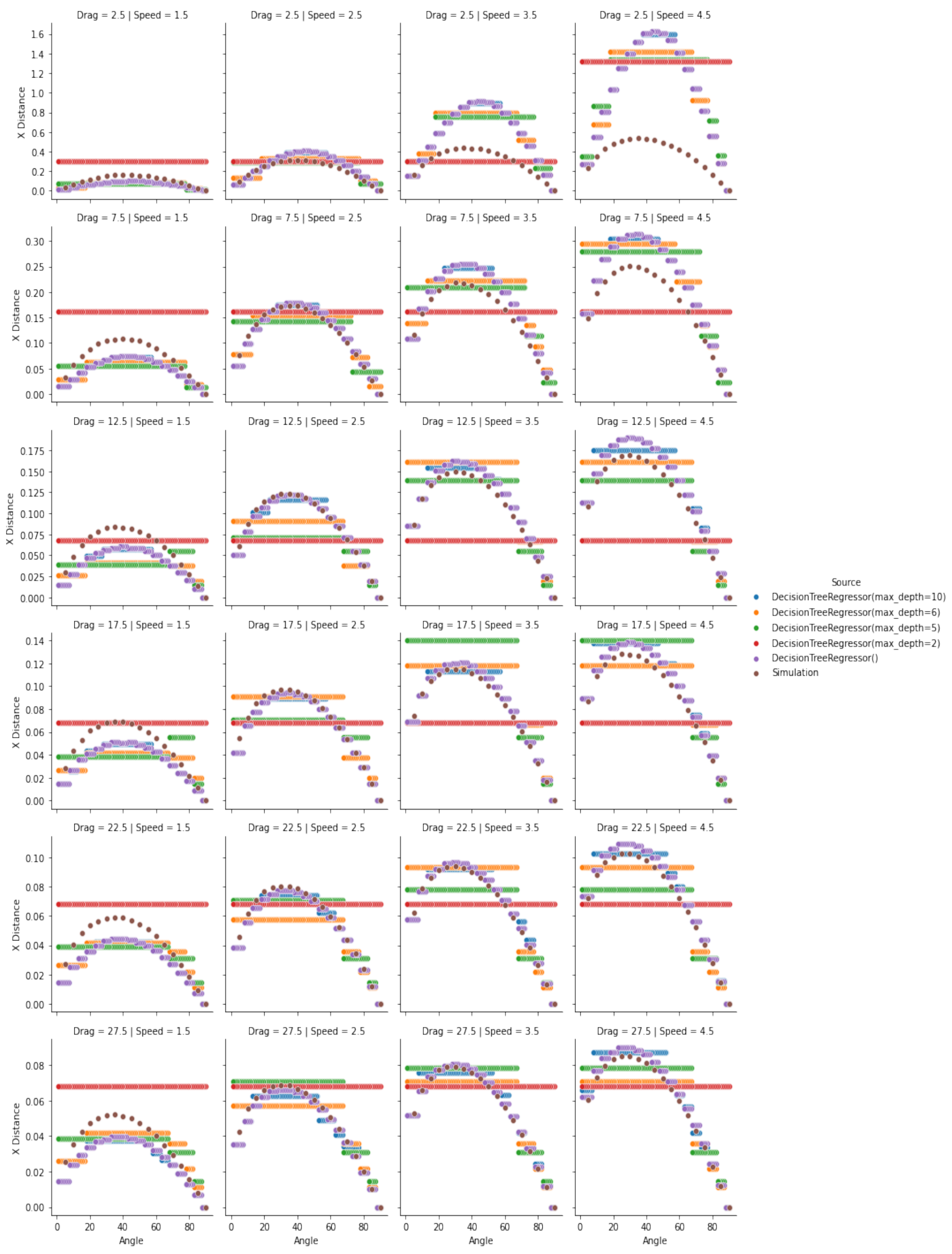
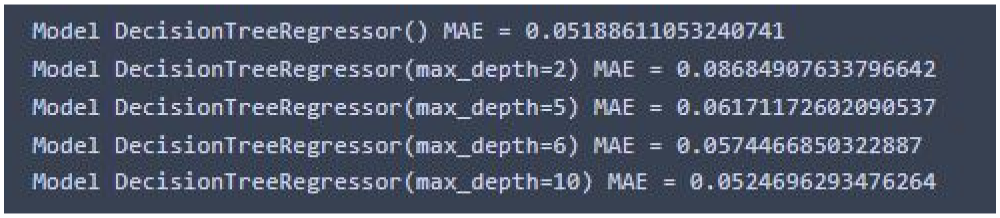
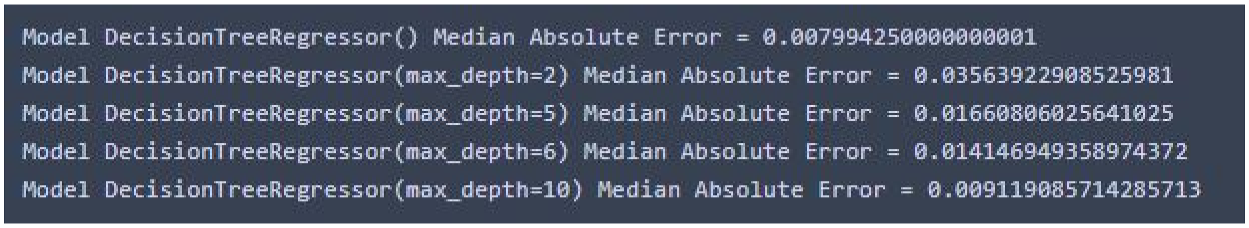


Figure 4.25: Performance of DT regression models, second comparison



```
Model DecisionTreeRegressor() MAE = 0.05188611053240741
Model DecisionTreeRegressor(max_depth=2) MAE = 0.08684907633796642
Model DecisionTreeRegressor(max_depth=5) MAE = 0.06171172602090537
Model DecisionTreeRegressor(max_depth=6) MAE = 0.0574466850322887
Model DecisionTreeRegressor(max_depth=10) MAE = 0.0524696293476264
```

Figure 4.26: Mean Absolute Error for the DT models, second comparison



```
Model DecisionTreeRegressor() Median Absolute Error = 0.007994250000000001
Model DecisionTreeRegressor(max_depth=2) Median Absolute Error = 0.03563922908525981
Model DecisionTreeRegressor(max_depth=5) Median Absolute Error = 0.01660806025641025
Model DecisionTreeRegressor(max_depth=6) Median Absolute Error = 0.014146949358974372
Model DecisionTreeRegressor(max_depth=10) Median Absolute Error = 0.009119085714285713
```

Figure 4.27: Median Absolute Error for the DT models, second comparison

Following the methodology applied so far, the comparison was applied to the outcomes with the parameters regarding the first comparison. Table 4.1 shows the deviation associated with this configuration.

As expected, the Gaussian Process model is the most accurate under these factor values, presenting only 0.057 degrees of deviation on average from the simulated solution. Also, the model has the lowest maximum error, with a value of 5 degrees.

Table 4.2 shows the deviation on the second comparison parameters. In this opportunity, the Gaussian Process keeps the first place according to accuracy, with an average deviation of -0.458 degrees. However, it is the worst score regarding the max error, presenting a difference of 20 degrees for some combinations. The Decision Tree model has the least of the max errors and the second best in accuracy.

Finally, Table 4.3 shows the deviations considering all the combinations. This includes the 91 combinations of Speed and Drag Factor. Here, it can be seen that the Gaussian Process model has the lowest average deviation, hence the most accurate, but also presents the worst deviation, with a max error of 21 degrees. On the other extreme, the Decision Tree model has the least max deviation, only with a value of 8 degrees, however, is the model that scores the worst on the

Model	Average Deviation	Max Deviation
DecisionTreeRegressor()	-1.971	6
GaussianProcessRegressor(kernel=RBF(length scale=1))	0.057	5
KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf')	0.314	6
KNeighborsRegressor(n neighbors=3)	-1.686	8

Table 4.1: Angle deviations for first comparison combinations

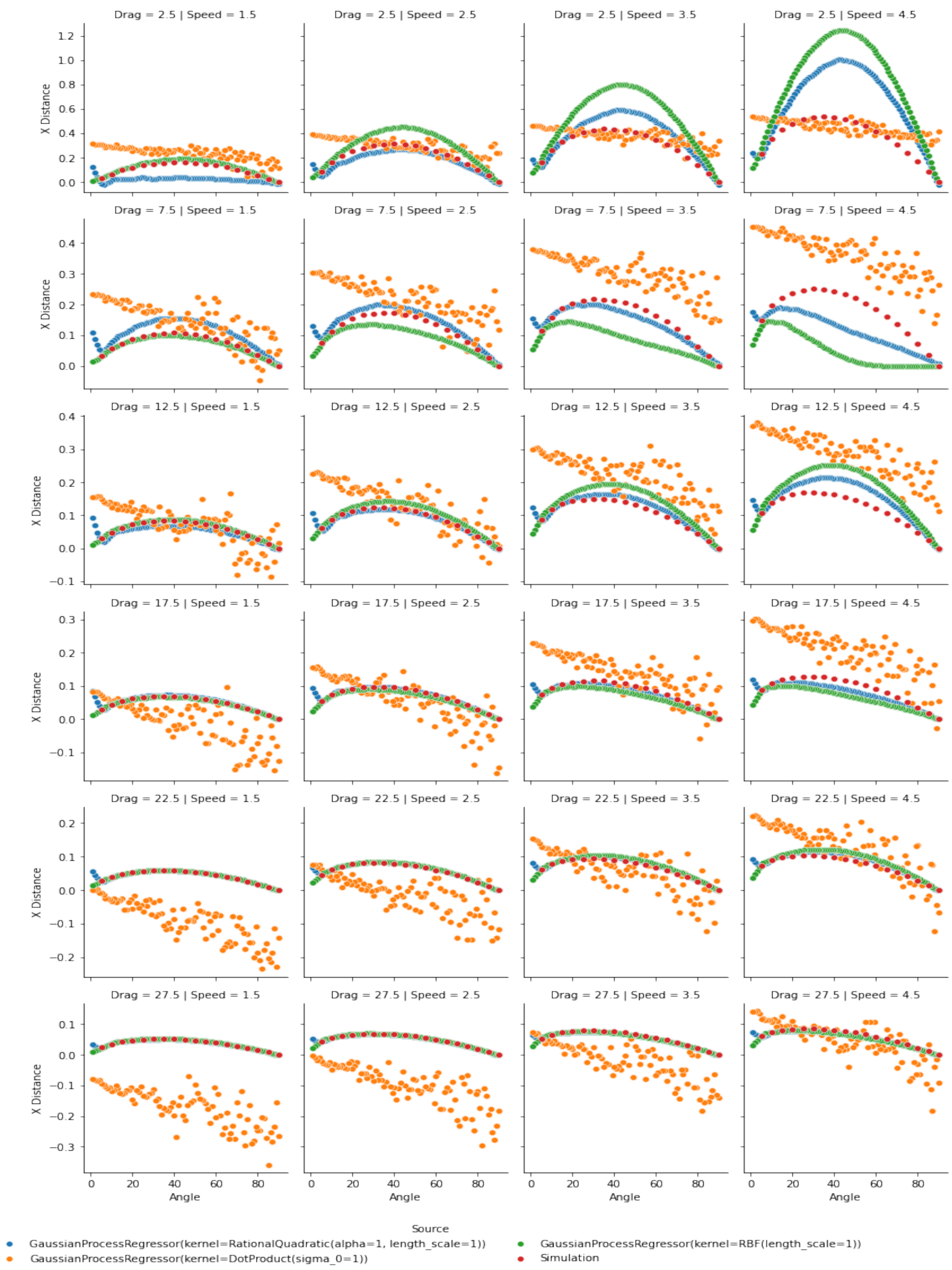
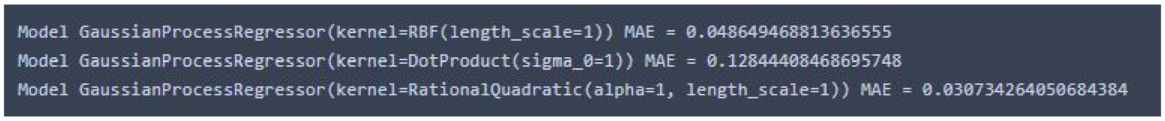
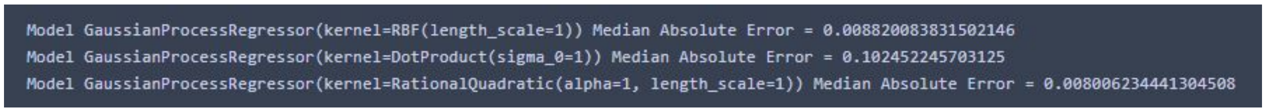


Figure 4.28: Performance of GP regression models, second comparison



```
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) MAE = 0.048649468813636555
Model GaussianProcessRegressor(kernel=DotProduct(sigma_0=1)) MAE = 0.12844408468695748
Model GaussianProcessRegressor(kernel=RationalQuadratic(alpha=1, length_scale=1)) MAE = 0.030734264050684384
```

Figure 4.29: Mean Absolute Error for the GP models, second comparison



```
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) Median Absolute Error = 0.008820083831502146
Model GaussianProcessRegressor(kernel=DotProduct(sigma_0=1)) Median Absolute Error = 0.102452245703125
Model GaussianProcessRegressor(kernel=RationalQuadratic(alpha=1, length_scale=1)) Median Absolute Error = 0.008006234441304508
```

Figure 4.30: Median Absolute Error for the GP models, second comparison

average precision.

It checked the combinations where the models performed the worst on the task. These configurations are:

- For the GP model, the worst scores were in the combinations Df=7.5 with v= 4, 4.5 and 5
- For the DT model, the worst scores were in the combinations Df=2.5 with v= 4, 4,5 and 5
- For the KR model, the worst scores were in the combinations Df=2.5, 7.5 with v= 4, 4.5 and 5
- For the KNN model, the worst scores were in the combinations Df= 12.5, 22.5, and 27.5 with v= 1.5 and 3

The most repetitive levels of the factors are the higher values of velocity and the intermediate values of drag. One possible explanation is the distribution of the data on the training dataset. Figure 4.34 shows the map of points that were used to train the models. It can be appreciated that the majority of the points are in the bottom left quarter of the graphic. Furthermore, the three curves that can be easily appreciated over the rest there correspond to the values of $Df = 0$ and $v = 3, 4, 5$.

Model	Average Deviation	Max Deviation
DecisionTreeRegressor()	0.958	7
GaussianProcessRegressor(kernel=RBF(length scale=1))	-0.458	20
KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf')	1.083	9
KNeighborsRegressor(n neighbors=3)	-2.000	14

Table 4.2: Angle deviations for second comparison combinations

Model	Average Deviation	Max Deviation
DecisionTreeRegressor()	-0.786	8
GaussianProcessRegressor(kernel=RBF(length scale=1))	-0.427	21
KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf')	0.444	10
KNeighborsRegressor(n neighbors=3)	-0.581	15

Table 4.3: Summary of the angle deviations for the models

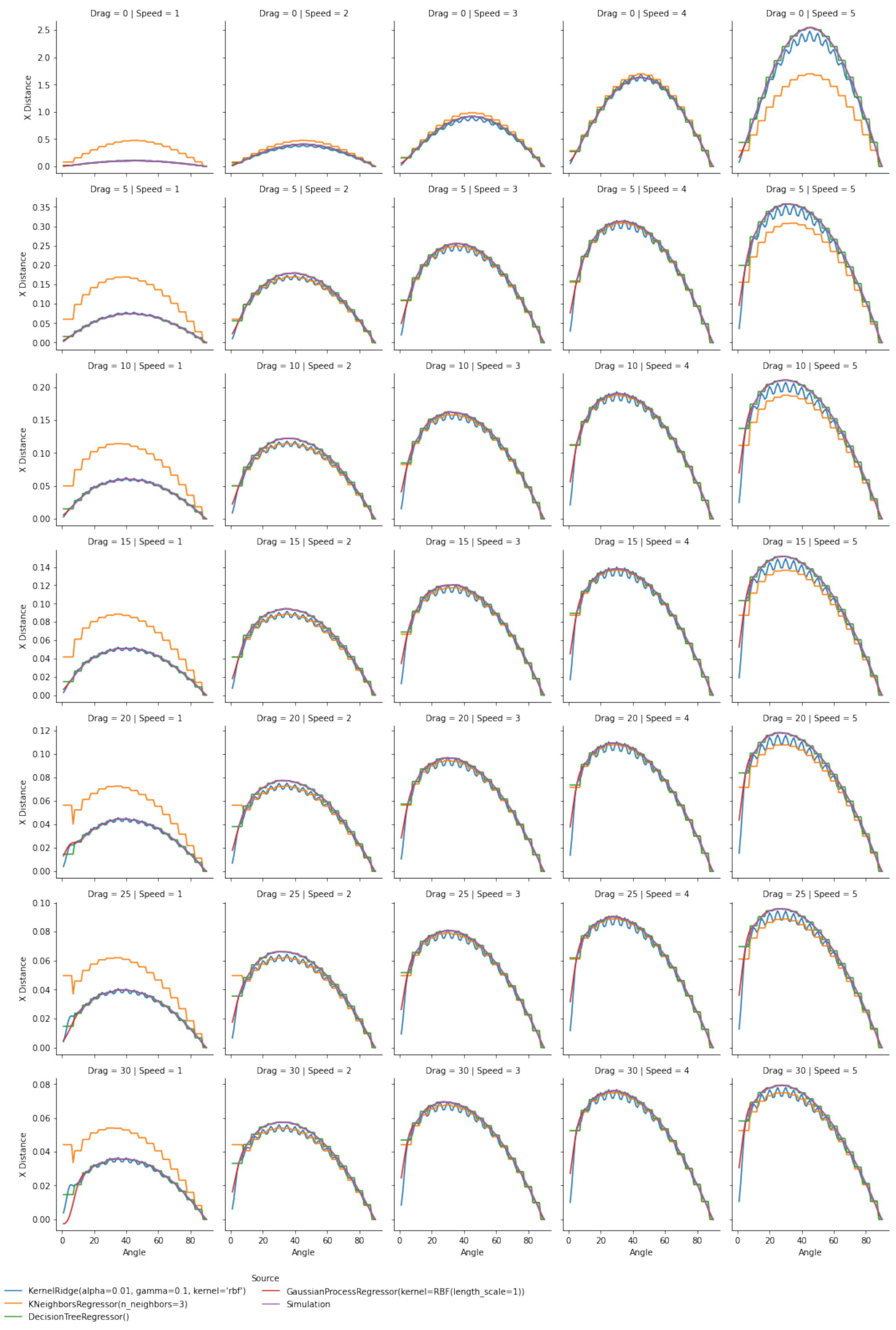


Figure 4.31: Performance of the best models, first comparison

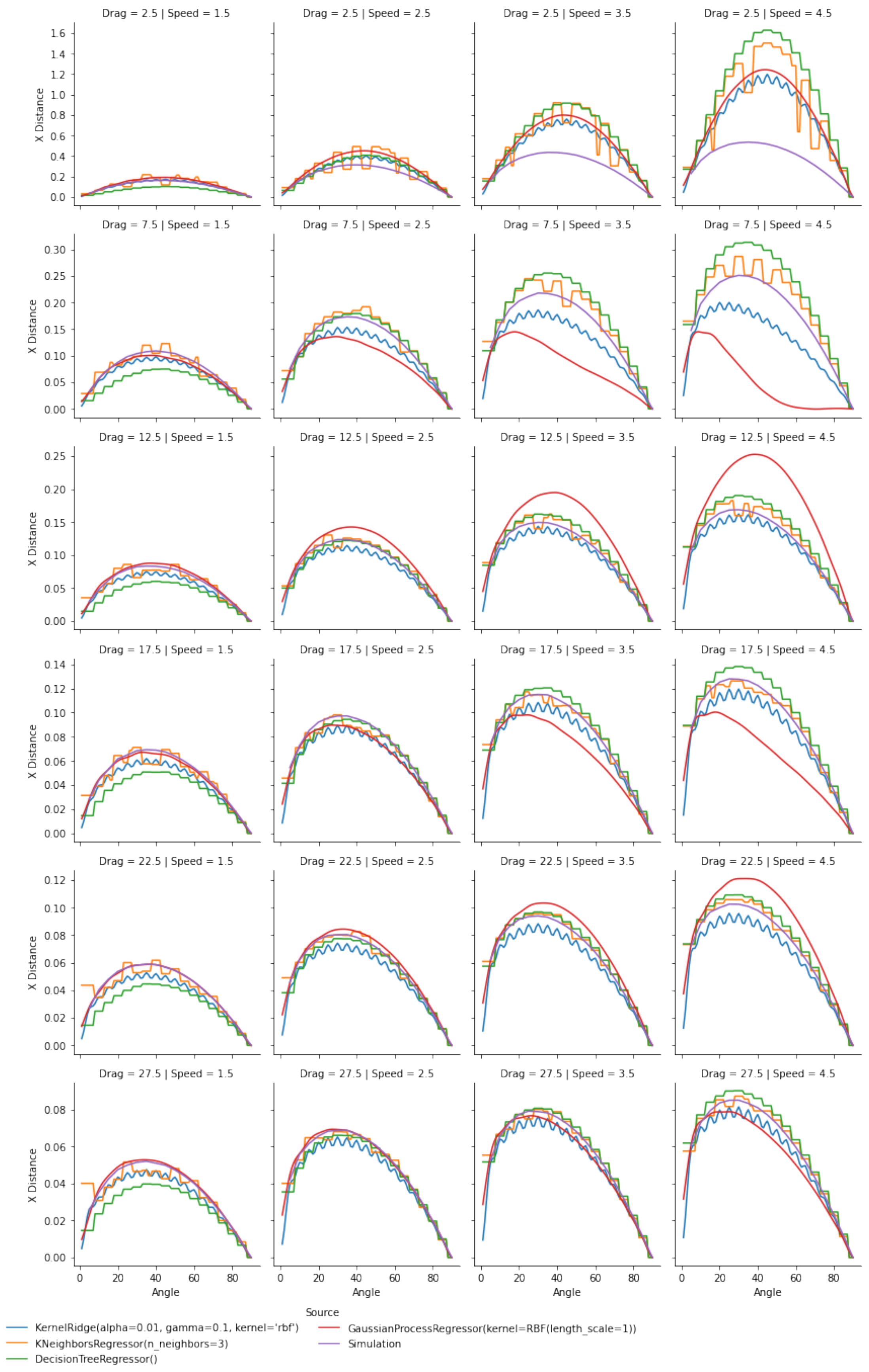
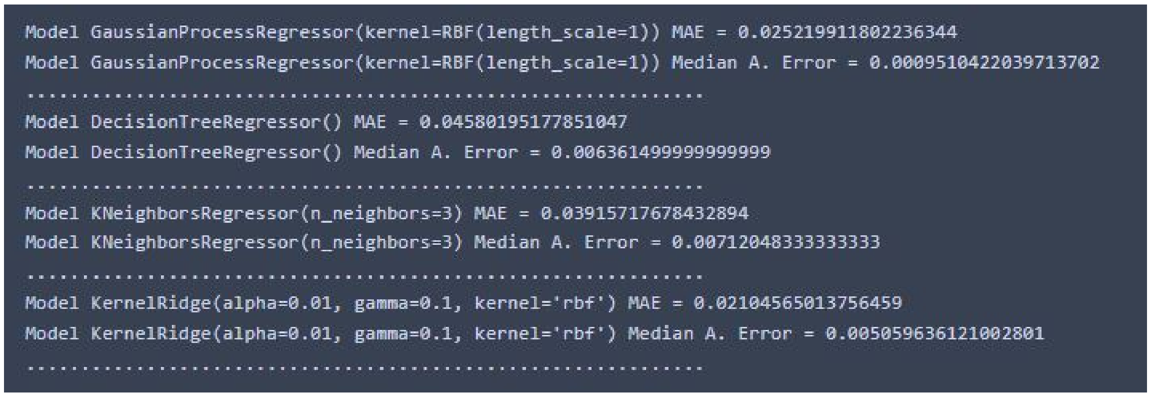


Figure 4.32: Performance of the best models, second comparison



```
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) MAE = 0.025219911802236344
Model GaussianProcessRegressor(kernel=RBF(length_scale=1)) Median A. Error = 0.0009510422039713702
..............................................................
Model DecisionTreeRegressor() MAE = 0.04580195177851047
Model DecisionTreeRegressor() Median A. Error = 0.006361499999999999
..............................................................
Model KNeighborsRegressor(n_neighbors=3) MAE = 0.03915717678432894
Model KNeighborsRegressor(n_neighbors=3) Median A. Error = 0.007120483333333333
..............................................................
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') MAE = 0.02104565013756459
Model KernelRidge(alpha=0.01, gamma=0.1, kernel='rbf') Median A. Error = 0.005059636121002801
..............................................................
```

Figure 4.33: Mean and Median absolute errors for the models

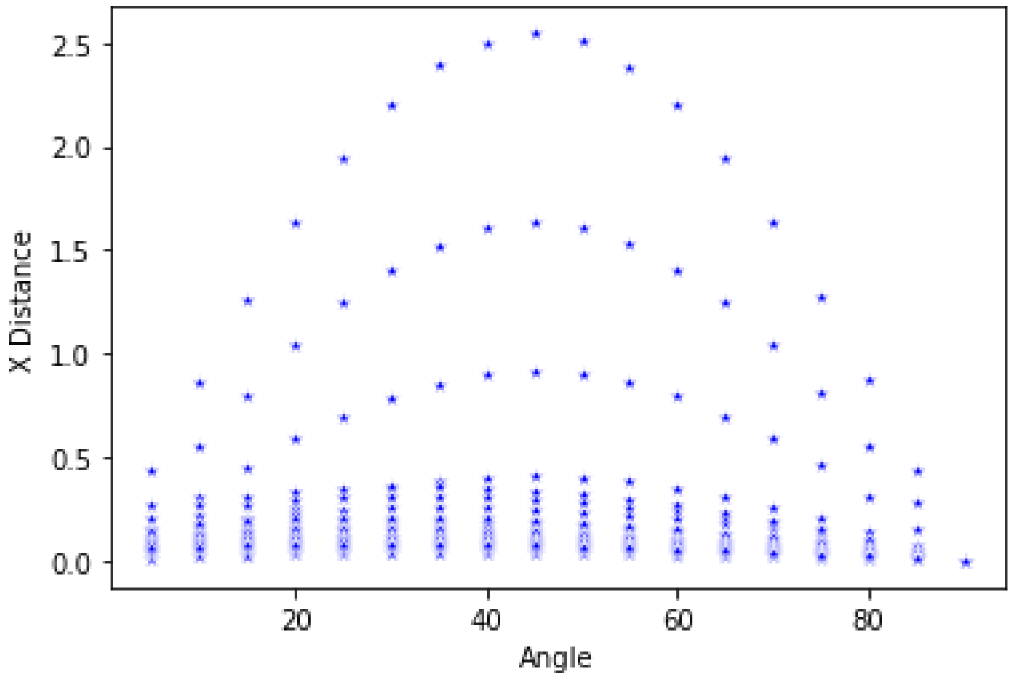


Figure 4.34: Scatter plot of training data

Chapter 5

Conclusions and Recommendations for Further Work

5.1 Summary and Conclusions

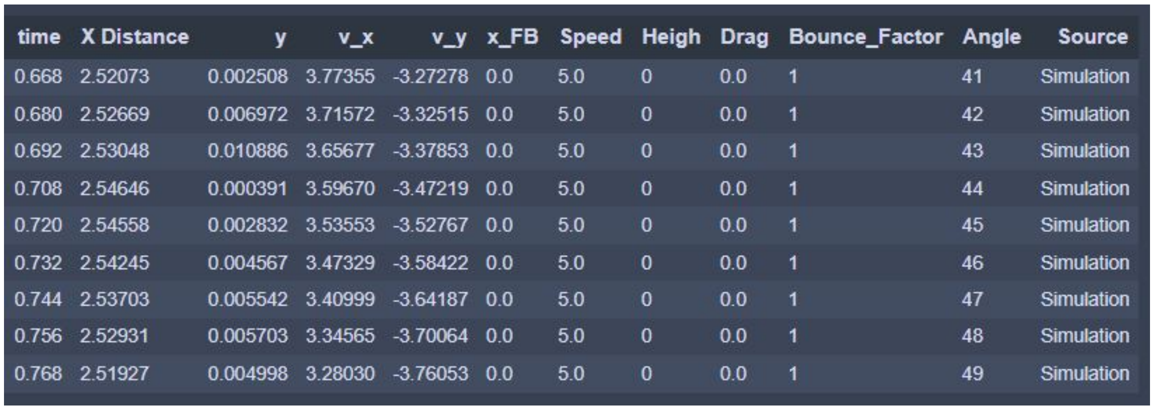
The general objective of this thesis is to investigate and study the performance of machine learning techniques used as surrogate models. In this thesis, the model to be surrogate was a simple bouncing ball simulation model and five different machine learning techniques were used to train different models to experiment with. Two major comparisons were carried out to measure the quality of the predictions made by the models. Also, the models were measured regarding a specific problem: finding the angle that maximizes the horizontal distance traveled before the first bounce. The findings of this study allow us to conclude the following.

In general, machine learning techniques can be really good tools to act as surrogate models. The results show that four of the five techniques used on average emulated the original model's behavior very accurately and consistently. The best performance on all the models was when evaluating with the same level of parameters that the data used for training them. The worst performance they presented was when the speed values were high and the drag was relatively small. This bad performance was due to the amount of points or data. Figure 4.34 in the previous chapter shows the density of the training points. The models cannot correctly interpret this drastic change which is the reason for this performance. The performance of the models on the specific problem of finding the angle that maximizes the distance was very good. All models presented on average less than 1 degree of deviation from the original simulation model considering 91 different levels of factor combinations.

Support Vector Machine Regression models could have performed better in this study. This does not mean that SVM models are not good to act like surrogate models but probably was due to the training settings and more research on these models is needed before concluding something. The literature also indicates that SVM is normally used for classification problems rather than regression ones. On the other hand, Kernel Ridge Regression models performed very solid in all scenarios and even presented the best score on the accuracy measurement. Even when Decision Tree and K Nearest Neighbors techniques are mostly used for classification problems, the algorithm to make them predict numerical outcomes in regression problems showed very decent results. KNN models presented good results when the combinations of the factors were in the middle values. Still, the model tends to overestimate or underestimate the predictions at near-the-border combination values. This is not a surprise considering the algorithm used for this technique. The DT model presented an excellent performance when the values to predict matched the trained data but started to fail on the predictions whit interpolated values, nevertheless for the specific problem was the model with the worst score of average deviation ($-0,786^o$) but the best score on the max deviation. Gaussian Process models can be extremely good at emulating the behavior of the models, performing very well in almost all scenarios and also presenting the least average deviation on the problem. The drawback is that this model has the worst maximum deviation, reaching 21^o for the combination of high speed and low drag.

The definition of the objective and the requirements and characteristics of the problem plays a huge role in the selection of the technique for the model. For instance, if a problem needs to replicate the "shape" of the predictor function GP is the only type of model that replicates the smoothness of the curve in this case. In contrast, KRR presented a "spiky" shape, and DT and KNN showed essentially a step function. But these characteristics can also be used in different ways. Imagine the objective is to use the surrogate models to narrow down a possible solution, then the DT and KNN can provide a range of values since the step shape. GP can also give the standard deviation, a prediction, and a range of possible outcomes associated with a probability that can narrow down options.

As a final conclusion, always remember that the surrogate models will emulate the behavior of the models they are surrogating, including mimicking their failures and limitations. In the original model used in this work, when the parameters are set as $V_0 = 5$ and $Df = 0$. Theoretically, the angle that maximizes is 45^o. Nevertheless, due to approximations, steps, and other models' limitations, the angle that shows the maximum distance x is 46^o. This can be corroborated in Figure 5.1



time	X Distance	y	v_x	v_y	x_FB	Speed	Heigh	Drag	Bounce_Factor	Angle	Source
0.668	2.52073	0.002508	3.77355	-3.27278	0.0	5.0	0	0.0	1	41	Simulation
0.680	2.52669	0.006972	3.71572	-3.32515	0.0	5.0	0	0.0	1	42	Simulation
0.692	2.53048	0.010886	3.65677	-3.37853	0.0	5.0	0	0.0	1	43	Simulation
0.708	2.54646	0.000391	3.59670	-3.47219	0.0	5.0	0	0.0	1	44	Simulation
0.720	2.54558	0.002832	3.53553	-3.52767	0.0	5.0	0	0.0	1	45	Simulation
0.732	2.54245	0.004567	3.47329	-3.58422	0.0	5.0	0	0.0	1	46	Simulation
0.744	2.53703	0.005542	3.40999	-3.64187	0.0	5.0	0	0.0	1	47	Simulation
0.756	2.52931	0.005703	3.34565	-3.70064	0.0	5.0	0	0.0	1	48	Simulation
0.768	2.51927	0.004998	3.28030	-3.76053	0.0	5.0	0	0.0	1	49	Simulation

Figure 5.1: Results of simulation for V=5 and Drag 0

5.2 Recommendations for Further Work

This section aims to give some recommendations and ideas that outline potential avenues for future research based on the findings and limitations of the current study. The objective is to be a small guide for others interested in expanding the existing research or addressing the identified gaps and unanswered questions. These recommendations have been divided into short-term and medium/long-term recommendations.

5.2.1 Short-term Recommendations

In this category, the first recommendation is to repeat the experiments performed in this thesis but with better knowledge and experience on how to properly train and fit models, with all the specific techniques. Use complementary tools such as preprocessing the training data, dimensionality reduction, parameter optimization, etc. Focusing on only one technique at a time but performing an optimized surrogate model will give more complementary information about the performance of the IA.

A second short-term recommendation is to include and explore using of Deep Learning techniques such as Artificial Neural Networks. A small review of these techniques was presented in Chapter 2, but since the training using ANNs requires more expertise, it was impossible to include them in the experiments. Nevertheless, deep learning seems more powerful than some of the techniques used and should be considered for future analysis.

5.2.2 Medium/Long-term Recommendations

The next step would be experimenting with a detailed or expensive real production model. This is the ultimate goal of the research. This is at least a medium-term recommendation thought since is necessary to have more knowledge about the performance of the techniques to be able to choose the best algorithm for the particular case, etc. This thesis already presents some general conclusions but further specific research is needed. Real production models are expensive, so it is recommended to continue researching those once the short-term recommended studies have been carried out to save resources.

Chapter 6

Appendix

6.1 Bouncing Ball Model

6.1.1 Code for the Model

```python
# These imports are only there to be able to override more of the fmi2slave class in order to avoid error messages (see below). Can be removed when pythonfmu is updated
import datetime
from typing import Dict#, Any, ClassVar, List, Optional
from xml.etree.ElementTree import Element, SubElement
from pythonfmu._version import __version__ as VERSION
from pythonfmu.variables import Boolean, Integer, Real, ScalarVariable, String
from pythonfmu.fmi2slave import FMI2_MODEL_OPTIONS
# end special imports

from pythonfmu import Fmi2Causality, Fmi2Initial, Fmi2Variability, Fmi2Slave, Boolean, Integer, Real, String, DefaultExperiment
#from .default_experiment import DefaultExperiment

from math import radians, degrees, sin, cos, sqrt

class BouncingBall(Fmi2Slave):
    '''Fmi2Slave implementation of a model, which can be compiled into an FMU through buildFMU

    The following variables can be set and transferred to Fmi2Slave.__init__:

```

```
    guid, author, license, version, copyright, modelName, description, default_experiment

    In the overridden class the interface variables should be defined.
    '''
    instance_name = 'Bouncing Ball'

    def __init__(self, **kwargs):
#         argDict = { 'modelName'           : 'Bouncing Ball',
#                     'author'              : 'Siegfried Eisinger, DNV',
#                     'description'         : 'A simple bouncing ball model for experimenting with PythonFMU',
#                     'default_experiment' : DefaultExperiment( start_time=0, stop_time=10, step_size=0.1),
        self.modelName = 'BouncingBall'
        self.author    = 'Siegfried Eisinger, DNV'
        self.description = 'A simple bouncing ball model for experimenting with PythonFMU'
        self.start_time = 0
        self.stop_time  =10#, step_size=0.1)
        self.step_size = 0.1
        self.default_experiment = DefaultExperiment( start_time=0, stop_time=10)#, step_size=0.1)

        super().__init__( **kwargs)
        self.y0 = 0.0 # the (initial)) height in [m]
        self.angle0 = 45 # the initial angle (in degrees!)
        self.v0 = 1.0 # speed in angle direction
        self.bounceFactor = 0.95 # factor on speed when bouncing
        self.drag = 0.0 # drag decelleration factor defined as a = self.drag* v^2 with dimensin 1/m
        self.energy = None
        self.period = None
        self.enter_initialization_mode() # ensure that the variables get correct values
        self.register_variable(Real( name="y0", causality=Fmi2Causality.parameter, description="y position at time 0", initial=Fmi2Initial.exact, variability=Fmi2Variability.fixed))
        self.register_variable(Real( name="angle0", causality=Fmi2Causality.parameter, description="angle at time 0", initial=Fmi2Initial.exact, variability=Fmi2Variability.fixed))
        self.register_variable(Real( name="v0", causality=Fmi2Causality.parameter, description="speed at time 0", initial=Fmi2Initial.exact, variability=Fmi2Variability.fixed))
        self.register_variable(Real( name="bounceFactor", causality=Fmi2Causality.parameter, description="factor on speed when bouncing", initial=Fmi2Initial.exact, variability=Fmi2Variability.fixed))
        self.register_variable(Real( name="drag", causality=Fmi2Causality.parameter, description="drag decelleration factor defined as a = self.drag* v^2 with dimensin 1/m", initial=Fmi2Initial.exact, variability=Fmi2Variability.fixed))
        self.register_variable(Real( name="x", causality=Fmi2Causality.output, description="x position at time", initial=Fmi2Initial.exact, variability=Fmi2Variability.continuous))
        self.register_variable(Real( name="y", causality=Fmi2Causality.output, description="y position at time", initial=Fmi2Initial.exact, variability=Fmi2Variability.continuous))
        self.register_variable(Real( name="v_x", causality=Fmi2Causality.output, description="speed in x-direction at time", initial=Fmi2Initial.exact, variability=Fmi2Variability.continuous))
        self.register_variable(Real( name="v_y", causality=Fmi2Causality.output, description="speed in y-direction at time", initial=Fmi2Initial.exact, variability=Fmi2Variability.continuous))
        self.register_variable(String("mdShort", causality=Fmi2Causality.local))

        # Note:
        # it is also possible to explicitly define getters and setters as lambdas in case the variable is not backed by a Python field.
        # self.register_variable(Real("myReal", causality=Fmi2Causality.output, getter=lambda: self.realOut, setter=lambda v: set_real_out(v))

    def enter_initialization_mode(self):
        a0 = radians( self.angle0)
        self.x = 0.0 # start always at x=0
        self.y = 1.0*self.y0
        self.v_x = self.v0* cos( a0)
        self.v_y = self.v0* sin( a0)
        self.energy = 9.81*self.y + 0.5*self.v_y*self.v_y
        self.period = 2* self.v_y/ 9.81 # may change when energy is taken out of the system
        print("INIT y0:", self.y0,", angle:", self.angle0, "v_x_0:", self.v_x, ", v_y_0:", self.v_y, ", bounce:", self.bounceFactor, ", drag:", self.drag)
        return( True)

    def do_step(self, current_time, step_size):
        def bounce_loss( v0):
            if self.bounceFactor == 1.0:
                return( v0)
            v0 *= self.bounceFactor # speed with which it leaves the ground
            self.energy = v0*v0/2
            self.period = 2*v0/9.81
            return( v0)

        self.x += self.v_x* step_size

        y = self.y + self.v_y* step_size - 9.81/ 2* step_size*step_size

        if y <= 0: # bounce
            t0 = self.v_y/9.81* (1 - sqrt( 1 + 2*self.y*9.81/ self.v_y/self.v_y)) # time when it hits the ground
            v0 = sqrt(2*self.energy) #more exact than self.v_y - 9.81* t0 # speed when jumps off the ground (without energy loss)
            v0 = bounce_loss( v0) # check energy loss during bouncing
            #print("BOUNCE", current_time, '(', self.x, ',', self.y,')', t0, v0)
            tRest = step_size-t0
            while True:
                if tRest < self.period: # cannot do a whole bounce in the remaining time
                    break
                if self.drag != 0: raise NotImplementedError("Bouncing a whole period is not implemented when drag is involved. Try choosing smaller time steps.")
                v0 = bounce_loss( v0)
                tRest -= self.period

            self.y = v0* tRest  - 9.81/ 2* tRest*tRest # height end of step
            self.v_y = v0 - 9.81* tRest # speed end of step
        else:
            self.v_y -= 9.81* step_size
            self.y = y
        if self.drag != 0:
            fac = 1 - self.drag* sqrt( self.v_x*self.v_x + self.v_y*self.v_y)* step_size
            self.v_x *= fac
            self.v_y *= fac
            self.energy = 9.81*self.y + 0.5*self.v_y*self.v_y
            #print("FAC", fac, self.v_x, self.v_y, self.energy)
        e = 9.81*self.y + 0.5*self.v_y*self.v_y
        if  abs( e-self.energy) > 1e-6: # and  and
```

```
            print("Energy leak", current_time, e, self.energy)
            self.energy = e
        #print(current_time, self.x, self.y, self.v_x, self.v_y)
        return True

    # =================
    # Note: It should not be necessary to include this, but there is an error in the function which needs to be fixed in the package
    #       Until then, the function is overridden
    #       See location of DefaultExperiment within modelDescription and the fact that the default variables must be converted to strings
    def to_xml(self, model_options: Dict[str, str] = dict()) -> Element:
        """Build the XML representation of the model.

        Args:
            model_options (Dict[str, str]) : FMU model options

        Returns:
            (xml.etree.TreeElement.Element) XML description of the FMU
        """

        t = datetime.datetime.now(datetime.timezone.utc)
        date_str = t.isoformat(timespec="seconds")

        attrib = dict(
            fmiVersion="2.0",
            modelName=self.modelName,
            guid=f"{self.guid!s}",
            generationTool=f"PythonFMU {VERSION}",
            generationDateAndTime=date_str,
            variableNamingConvention="structured"
        )
        if self.description is not None:
            attrib["description"] = self.description
        if self.author is not None:
            attrib["author"] = self.author
        if self.license is not None:
            attrib["license"] = self.license
        if self.version is not None:
            attrib["version"] = self.version
        if self.copyright is not None:
            attrib["copyright"] = self.copyright

        root = Element("fmiModelDescription", attrib)

```

```python
        options = dict()
        for option in FMI2_MODEL_OPTIONS:
            value = model_options.get(option.name, option.value)
            options[option.name] = str(value).lower()
        options["modelIdentifier"] = self.modelName
        options["canNotUseMemoryManagementFunctions"] = "true"

        SubElement(root, "CoSimulation", attrib=options)

        if len(self.log_categories) > 0:
            categories = SubElement(root, "LogCategories")
            for category, description in self.log_categories.items():
                categories.append(
                    Element(
                        "Category",
                        attrib={"name": category, "description": description},
                    )
                )

        if self.default_experiment is not None:
            attrib = dict()
            if self.default_experiment.start_time is not None:
                attrib["startTime"] = str(self.default_experiment.start_time)
            if self.default_experiment.stop_time is not None:
                attrib["stopTime"] = str(self.default_experiment.stop_time)
            if self.default_experiment.tolerance is not None:
                attrib["tolerance"] = str(self.default_experiment.tolerance)
            SubElement(root, "DefaultExperiment", attrib)

        variables = SubElement(root, "ModelVariables")
        for v in self.vars.values():
            if ScalarVariable.requires_start(v):
                self.__apply_start_value(v)
            variables.append(v.to_xml())

        structure = SubElement(root, "ModelStructure")
        outputs = list(
            filter(lambda v: v.causality == Fmi2Causality.output, self.vars.values())
        )

        if outputs:
            outputs_node = SubElement(structure, "Outputs")
            for i, v in enumerate(self.vars.values()):
                if v.causality == Fmi2Causality.output:
                    SubElement(outputs_node, "Unknown", attrib=dict(index=str(i + 1)))

        return root

    def __apply_start_value(self, var: ScalarVariable):
        vrs = [var.value_reference]

        if isinstance(var, Integer):
            refs = self.get_integer(vrs)
        elif isinstance(var, Real):
            refs = self.get_real(vrs)
        elif isinstance(var, Boolean):
            refs = self.get_boolean(vrs)
        elif isinstance(var, String):
            refs = self.get_string(vrs)
        else:
            raise Exception(f"Unsupported type!")

        var.start = refs[0]
```

6.1.2 Code for Python FMU Simulator of the model

```python
import tempfile
import zipfile
from pathlib import Path

from pythonfmu.builder import FmuBuilder #builder, csvbuilder, deploy
from pythonfmu._version import __version__

from fmpy import dump, simulate_fmu, plot_result, write_csv



if __name__ == '__main__':
    def build( scriptFile, testIt=False):
        with tempfile.TemporaryDirectory() as documentation_dir:
            doc_dir = Path(documentation_dir)
            license_file = doc_dir / "licenses" / "license.txt"
            license_file.parent.mkdir()
            license_file.write_text("Dummy license")
            index_file = doc_dir / "index.html"
            index_file.write_text("dummy index")
            asBuilt = FmuBuilder.build_FMU( scriptFile, dest='.', documentation_folder=doc_dir)

        if testIt:
            result = simulate_fmu(asBuilt.name, stop_time=2, step_size=0.04, solver='Euler', start_values={ 'angle0':45, 'bounceFactor':1.0, 'drag':0, 'v0':3, 'y0':0})

            print( dump( asBuilt.name))
            plot_result(result)
            write_csv('res.csv', result)
        return( asBuilt)
#     build('basic_example.py')
    build('bouncing_ball.py', testIt=True)
```

6.2 Gathering the Data

6.2.1 Code for performing the initial simulations - FMU modified

```
import tempfile
import zipfile
import pandas as pd
import numpy as np
from pathlib import Path

from pythonfmu.builder import FmuBuilder #builder, csvbuilder, deploy
from pythonfmu._version import __version__

from fmpy import dump, simulate_fmu, plot_result, write_csv


bouncelist=[1]
ylist=[0]
draglist=np.linspace(0,30,13)
anglelist = list(range(1,91,1))
vlist=np.linspace(1,5,9)

failedlist=[]

for i in anglelist:
    for j in bouncelist:
        for k in draglist:
            for l in ylist:
                 for m in vlist:

                     try:

                         if __name__ == '__main__':
                             def build( scriptFile, testIt=False):
                                 with tempfile.TemporaryDirectory() as documentation_dir:
                                     doc_dir = Path(documentation_dir)
                                     license_file = doc_dir / "licenses" / "license.txt"
                                     license_file.parent.mkdir()
                                     license_file.write_text("Dummy license")
                                     index_file = doc_dir / "index.html"
                                     index_file.write_text("dummy index")
                                     asBuilt = FmuBuilder.build_FMU(scriptFile, dest='.', documentation_folder=doc_dir)

                                 if testIt:
                                     result = simulate_fmu( asBuilt.name, stop_time=2, step_size=0.001, solver='Euler', start_values={'angle0':i, 'bounceFactor':j, 'drag':k, 'y0':l, 'v0':m})
                                     print( dump( asBuilt.name))
                                     #plot_result(result)
                                     write_csv('Single_Experiments3/BB-%dangle-%dbf-%2fdrag-%dy-%2fv.csv' %(i,j,k,l,m), result)
                                 return( asBuilt)
                         #    build('basic_example.py')
                             build('bouncing_ball_Multiple_Experiments.py', testIt=True)
                     except:
                         print('combination {},{},{},{},{} not possible'.format(i,j,k,l,m))
                         failedlist.append('BB-%dangle-%dbf-%ddrag-%dy-%dv.csv' %(i,j,k,l,m))

faildf = pd.DataFrame(failedlist)
faildf.to_csv('failed_experiments.csv')
```

6.2.2 Code for generating the initial dataset

```
import pandas as pd
import numpy as np

```

```
anglelist=[1,10,20,30,40,50,60,70,80,90]
bouncelist=[1,2,3]
draglist=[0,5,10,15,20,25,30]
ylist=[0, 1, 2, 3]
vlist=[0, 1, 2]

final = pd.DataFrame()
errores = []
for i in anglelist:
    for j in bouncelist:
        for k in draglist:
            for l in vlist:
                for m in ylist:
                    try:
                        df = pd.read_csv('BB-{}angle-{}bf-{}drag-{}y-{}v.
    csv'.format(i,j,k,l,m))
                        if l ==0:
                            varray = [0.1]
                        else:
                            varray = [l]
                        if m == 0:
                            yarray = [0.1]
                        else:
                            yarray= [m]
                        df[['Speed']]=varray
                        df[['Heigh']]=yarray
                        df[['Drag']]=[k]
                        df[['Bounce_Factor']]=[j]
                        df[['Angle']]=[i]
                        final = pd.concat([final, df])

                    except:
                        errores.append('BB-{}angle-{}bf-{}drag-{}y-{}v.csv
    '.format(i,j,k,l,m))


final.to_csv('final_data.csv')
```

6.3 Code for the Experiments

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:

```

```python




# In[1]:


#Setting up libraries and loading Dataframe
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel,
    ExpSineSquared, RationalQuadratic, DotProduct
from sklearn.tree import DecisionTreeRegressor
from sklearn import neighbors
from sklearn.kernel_ridge import KernelRidge

from sklearn.metrics import *

df = pd.read_csv('Data_First_Bounce.csv')
x = df.iloc[:,6:].values
y_x = df.iloc[:,2]
y_time = df.iloc[:,1]

df2 = pd.read_csv('Data_First_Bounce2.csv')
x2 = df2.iloc[:,6:11]
y_x2 = df2.iloc[:,2]

df.head(10)


# In[2]:


#Train the models
GP = GaussianProcessRegressor(kernel=RBF())
DT = DecisionTreeRegressor()
KNN = neighbors.KNeighborsRegressor(3)
KR = KernelRidge(kernel = 'rbf', gamma=0.1, alpha=0.01)

models=[GP, DT, KNN, KR]
for i, model in enumerate(models):
    model.fit(x,y_x)


# In[3]:


#creating q dataframe with the predictions and ploting against the original model
totalresults =pd.DataFrame()
for s in range(1,6):
    for d in [0,5,10,15,20,25,30]:
        test_point = [s,0,d,1]  #[v0, y0, drag, BF, a0]
        data_points = df.loc[(df['Speed'] == test_point[0]) & (df['Heigh'] == test_point[1]) & (df['Drag'] == test_point[2]) & (df['Bounce_Factor'] == test_point[3])]
        pred = pd.DataFrame()

        for i, model in enumerate(models):
            for a in range(1,91):
                param = [[test_point[0], test_point[1], test_point[2], test_point[3],a]]
                #print(param)
                re = pd.DataFrame(model.predict(np.array(param)))
                re[['Angle']]=a
                re[['model']]=model
                re[['Drag']]=d
                re[['Speed']]=s
                pred = pd.concat([pred,re])
                totalresults = pd.concat([re, totalresults])

        colors=['r', 'y', 'g', 'm', 'c']
        plt.plot(data_points['Angle'],data_points['x'], color='blue', linewidth=2.5)
        for i, model in enumerate(models):
            plt.scatter(pred.loc[pred['model']==model, ['Angle']], pred.loc[pred['model']==model, [0]], s=3, color=colors[i])

        plt.plot(data_points['Angle'],data_points['x'], color='blue', linewidth=2.5)
        plt.show()


# In[4]:


```

```python
### mergin prediction with original points in one DataFrame
df[['Source']]='Simulation'
df.rename(columns={'x':'X Distance'}, inplace=True)

totalresults.rename(columns={0:'X Distance', 'model':'Source'}, inplace=True)
plotdata = pd.concat([totalresults, df],ignore_index=True, sort=False )
#plotting the results for every level of factor
plt=sns.FacetGrid(plotdata, col='Speed', row='Drag', hue='Source', sharey='row')
plt.map(sns.lineplot,'Angle', 'X Distance')
plt.add_legend()
sns.move_legend(plt, "lower left", bbox_to_anchor=(0, -0.05), ncol=2)


# In[5]:


#calculating errors for first comparison
ypred = {}
points = x
for i, k in enumerate(models):
    ypred['p{0}'.format(i)] = k.predict(np.array(points))
    print('Model {} MAE ='.format(k),mean_absolute_error(y_x, ypred['p{}'.format(i)]))
    print('Model {} Median A. Error ='.format(k),median_absolute_error(y_x, ypred['p{}'.format(i)]))


# In[6]:


#Solving the specific problem of angle that maximizes the X distance for every model
totalresults.reset_index(drop=True, inplace=True)

for d in range(0,31,5):
    for s in range(1,6):
        for i, model in enumerate(models):
            trcopy = totalresults.loc[(totalresults['Source']==model)&(totalresults['Speed']==s)&(totalresults['Drag']==d)]
            value_max = trcopy['X Distance'].idxmax()
            print('With Drag={} and V0={}; {} gives Max Angle={}'.format(d,s,model,totalresults.loc[value_max]['Angle']))
            #print(df)
        orcopy = df.loc[(df['Speed']==s)&(df['Drag']==d)]
        ormax = orcopy['X Distance'].idxmax()
        print('With Drag={} and V0={}; Simulation Model gives Max Angle={}
    '.format(d,s,df.loc[ormax]['Angle']))
        print('------------------------------------------------')
    print('------------------------------------------------')
print('------------------------------------------------')


# In[7]:


totalresults2 = pd.DataFrame()
for s in [1.5,2.5,3.5,4.5]:
    for d in [2.5,7.5,12.5,17.5,22.5,27.5]:
        test_point2 = [s,0,d,1]  #[v0, y0, drag, BF]
        data_points = df2.loc[(df['Speed'] == test_point[0]) & (df['Heigh'
    ] == test_point[1]) & (df['Drag'] == test_point[2]) & (df['
    Bounce_Factor'] == test_point[3])]
        pred2 = pd.DataFrame()

        for i, model in enumerate(models):
            for a in range(1,91):
                param2 = [[test_point2[0], test_point2[1], test_point2[2],
    test_point2[3],a]]
                #print(param)
                re2 = pd.DataFrame(model.predict(np.array(param2)))
                re2[['Angle']]=a
                re2[['model']]=model
                re2[['Drag']]=d
                re2[['Speed']]=s
                pred2 = pd.concat([pred2,re2])
                totalresults2 = pd.concat([re2, totalresults2])


# In[8]:


#plot the results
df2[['Source']]='Simulation'
df2.rename(columns={'x':'X Distance'}, inplace=True)

totalresults2.rename(columns={0:'X Distance', 'model':'Source'}, inplace=
    True)
plotdata2 = pd.concat([totalresults2, df2],ignore_index=True, sort=False )
```

```

#plotting for the second comparisson
plt2=sns.FacetGrid(plotdata2, col='Speed', row='Drag', hue='Source',
    sharey='row')
plt2.map(sns.lineplot,'Angle', 'X Distance')
plt2.add_legend()
sns.move_legend(plt2, "lower left", bbox_to_anchor=(0, -0.05), ncol=2)


# In[9]:


#calculating errors for second comparison
ypred2 = {}
points2 = x
for i, k in enumerate(models):
    ypred2['p{0}'.format(i)] = k.predict(np.array(points))
    print('Model {} MAE ='.format(k),mean_absolute_error(y_x, ypred2['p{}'
    .format(i)]))
    print('Model {} Median A. Error ='.format(k),median_absolute_error(y_x
    , ypred2['p{}'.format(i)]))


# In[10]:


allpredicts = pd.DataFrame()
for s in np.arange(1,5.5,0.5):
    for d in np.arange(0,31,2.5):
        for i, model in enumerate(models):
            for a in range(1,91):
                p = [[s,1,d,1,a]]
                re2[['X Distance']] = pd.DataFrame(model.predict(np.array(
    p)))
                re2[['Angle']]=a
                re2[['model']]=model
                re2[['Drag']]=d
                re2[['Speed']]=s
                allpredicts = pd.concat([allpredicts,re2])


# In[11]:


#generating the new Dataframe with the data combined
```

```
df3 = pd.read_csv('Data_First_Bouncefull.csv')
x3 = df3.iloc[:,7:12]
y_x3 = df3.iloc[:,2]

df3[['Source']]='Simulation'
df3.rename(columns={'x':'X Distance'}, inplace=True)


# In[12]:





# In[13]:


#check the algorithm
temporal=df3.loc[(df3['Speed']==5)&(df3['Drag']==0)]
temporal.loc[(df3['Angle']>40)&(df3['Angle']<50)]


# In[14]:


#plotting the density of the training points
simplot=sns.scatterplot(data=df, x='Angle', y='X Distance', marker='*',
    color='b')


# In[15]:


#generating the dataframe with the solution of the specific problem for
    all 91 combinations of speed and drag
allpredicts.reset_index(drop=True, inplace=True)
df3.reset_index(drop=True, inplace=True)
new=pd.DataFrame()
temp=pd.DataFrame()
temp2=pd.DataFrame()
for d in np.arange(0,31,2.5):
    for s in np.arange(1,5.5,0.5):
        for model in models:
            trcopy2 = allpredicts.loc[(allpredicts['model']==model)&(
    allpredicts['Speed']==s)&(allpredicts['Drag']==d)]
```

```python
            #print(trcopy2)
            value_max2 = trcopy2['X Distance'].idxmax()
            #print(value_max2)
            print('With Drag={} and V0={}; {} gives Max Angle={}'.format(d,s,model,allpredicts.loc[value_max2]['Angle']))
            p = [[s,1,d,1,a]]
            #temp[['X Distance']] = pd.DataFrame(model.predict(np.array(p)))
            temp[['Model']]=model
            temp[['Speed']]=s
            temp[['Drag']]=d
            temp[['Max Angle']]=allpredicts.loc[value_max2]['Angle']
            new = pd.concat([new,temp])

        orcopy2 = df3.loc[(df3['Speed']==s)&(df3['Drag']==d)]
        ormax2 = orcopy2['X Distance'].idxmax()
        #temp2[['X Distance']] = pd.DataFrame(model.predict(np.array(p)))
        temp2[['Model']]='Simulation'
        temp2[['Speed']]=s
        temp2[['Drag']]=d
        temp2[['Max Angle']]=df3.loc[ormax2]['Angle']
        new = pd.concat([new,temp2])
        print('With Drag={} and V0={}; Simulation Model gives Max Angle={}'.format(d,s,df3.loc[ormax2]['Angle']))
        print('------------------------------------------------')
    print('------------------------------------------------')
print('------------------------------------------------')


# In[16]:


#saving the results of the specific problem
new.to_csv('performance.csv')


# In[17]:


## FOR DT regressor model, use the extreme angles for the new simulations

param =[5, 2.5] #speed & Drag
inter = allpredicts.loc[(allpredicts['model']==models[1])&(allpredicts['Speed']==param[0])&(allpredicts['Drag']==param[1])]
maxim = inter['X Distance'].idxmax()
```

```
inter[['Angle']].loc[inter['X Distance']==inter.loc[maxim]['X Distance']]


# In[18]:


# FOR KNN Regressor, use the extreme angles for the new simulations
param =[5, 2.5] #speed & Drag
inter2 = allpredicts.loc[(allpredicts['model']==models[2])&(allpredicts['Speed']==param[0])&(allpredicts['Drag']==param[1])]
maxim2 = inter2['X Distance'].idxmax()
inter2[['Angle']].loc[inter2['X Distance']==inter2.loc[maxim2]['X Distance']]


# In[19]:


param =[2, 0] #speed & Drag
inter3 = allpredicts.loc[(allpredicts['model']==models[0])&(allpredicts['Speed']==param[0])&(allpredicts['Drag']==param[1])]
maxim3 = inter3['X Distance'].idxmax()

a,b = models[0].predict(np.array([[param[0],1,param[1],1,inter3.loc[maxim3]['Angle']]]), return_std=True) #[v0, y0, drag, BF, a0]
print(a)
print(b)
print(a-b)
#inter3[['Angle']].loc[(a-b) < inter3['X Distance'] < (a+b)]

inter3.head(60)

##tells me that is a lot of uncertanty because the std dev is way bigger than the predicted value


# In[20]:


#calculating the overall error for every model
y_final=df3.iloc[:,2]
x_final=df3.iloc[:,7:12]
ypredfinal = {}
for i, k in enumerate(models):
    ypredfinal['p{0}'.format(i)] = k.predict(np.array(x_final))
    print('Model {} MAE ='.format(k),mean_absolute_error(y_final, ypredfinal['p{}'.format(i)]))
    print('Model {} Median A. Error ='.format(k),median_absolute_error(y_final, ypredfinal['p{}'.format(i)]))
    print('.............................................................')


# In[ ]:
```

Bibliography

[1] FMPy library for fmu simulations. https://fmpy.readthedocs.io/en/latest/. Accessed: 2023-05-20.

[2] Matplot.Pyplot pyplot: a collection of functions that make matplotlib work like matlab. https://matplotlib.org/stable/index.html. Accessed: 2023-05-22.

[3] scikit-learn simple and efficient tools for predictive data analysis. https://scikit-learn.org/stable/index.html. Accessed: 2023-05-22.

[4] Seaborn seaborn: statistical data visualization. https://seaborn.pydata.org/index.html. Accessed: 2023-05-22.

[5] Systems modelling : theory and practice, 2004.

[6] Design of experiments : Applications, 2013.

[7] Solving computationally expensive engineering problems: Methods and applications, 2014.

[8] Model validation and uncertainty quantification, volume 3 : Proceedings of the 37th imac, a conference and exposition on structural dynamics 2019, 2020.

[9] Computational modeling in industry 4.0: a sustainable resource management perspective, 2022.

[10] Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers chemical engineering*, 108:250–267, 2018.

[11] Ian Cameron and Rafiqul Gani. Chapter 2 - modelling practice. In Ian Cameron and Rafiqul Gani, editors, *Product and Process Modelling*, pages 19–32. Elsevier, Amsterdam, 2011.

[12] Dragan Cvetkovic. *Computer Simulation*. IntechOpen, Rijeka, Jun 2017.

[13] Li Du and Yuan Du. Hardware accelerator design for machine learning. In Hamed Farhadi, editor, *Machine Learning*, chapter 1. IntechOpen, Rijeka, 2017.

[14] George M Dunnet, Charles H Gimingham, and John N. R Jeffers. *Modelling*. Outline Studies in Ecology. Springer Netherlands, Dordrecht, 1982.

[15] Ronald A Fisher. The design of experiments, 1951.

[16] FMI. Functional mock-up interface, 2022.

[17] Raúl Garreta. Scikit-learn : machine learning simplified., 2017.

[18] Kim Goethals, Ivo Couckuyt, Tom Dhaene, and Arnold Janssens. Sensitivity of night cooling performance to room/system design: Surrogate models based on cfd. *Building and environment*, 58:23–36, 2012.

[19] Gabriel F. N. Gonçalves, Assen Batchvarov, Yuyi Liu, Yuxin Liu, Lachlan R. Mason, Indranil Pan, and Omar K. Matar. Data-driven surrogate modeling and benchmarking for process equipment. *Data-Centric Engineering*, 1, 2020.

[20] Pramod Gupta. Introduction to machine learning in the cloud with python : concepts and practices, 2021.

[21] K.M Hangos. Process modelling and model analysis, 2001.

[22] Jiangang Hao and Tin Kam Ho. Machine learning made easy: A review of "scikit-learn" package in python programming language. *Journal of Educational and Behavioral Statistics*, 44(3):348–361, 2019.

[23] Lars Ivar Hatledal, Frederic Collonval, and Houxiang Zhang. Enabling python driven co-simulation models with pythonfmu. 2020.

[24] Dieter W. Heermann. *Computer-Simulation Methods*, pages 8–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[25] Anders Hjort, Johan Pensar, Ida Scheel, and Dag Einar Sommervoll. House price prediction with gradient boosted trees under different loss functions. *Journal of property research*, 39(4):338–364, 2022.

[26] Alan J Izenman. Modern multivariate statistical techniques : Regression, classification, and manifold learning, 2008.

[27] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning: With applications in R*, volume 103 of *Springer texts in statistics*. Springer, New York, 2013.

[28] Ping Jiang. Surrogate model-based engineering design and optimization, 2020.

[29] Yaochu Jin. Federated learning : fundamentals and advances, 2023.

[30] Alexander Jung. *Introduction*, pages 1–18. Springer Nature Singapore, Singapore, 2022.

[31] Andreas Junghanns, Cláudio Gomes, Christian Schulze, Klaus Schuch, Pierre R., Matthias Blaesken, Irina Zacharias, Andreas Pillekeit, Karl Wernersson, Torsten Sommer, Christian Bertsch, Torsten Blochwitz, and Masoud Najafi. The functional mock-up interface 3.0 - new features enabling new applications. *Linköping Electronic Conference Proceedings*, 09 2021.

[32] John D. Kelleher. Deep learning, 2019.

[33] Monique F Kilkenny and Kerin M Robinson. Data quality: "garbage in - garbage out". *Health Information Management Journal*, 47(3):103–105, 2018.

[34] Slawomir Koziel and Leifur Leifsson. *Simulation-Driven Design by Knowledge-Based Response Correction Techniques*. Springer International Publishing AG, Cham, 2016.

[35] S. Y. Kung. *Kernel ridge regressors and variants*, page 219–220. Cambridge University Press, 2014.

[36] Vladimír Kuti, Vladimír Kompi, Vladimír Kompi, Vladimír Kuti, and Justín Murín. *Computational Modelling and Advanced Simulations*, volume 24 of *Computational methods in applied sciences*. Springer Science + Business Media, Dordrecht, 1. aufl. edition, 2011.

[37] Marten Lohstroh, Patricia Derler, and Marjan Sirjani. *Principles of Modeling*, volume 10760 of *LNCS sublibrary. SL 2, Programming and software engineering*. Springer International Publishing AG, Cham, 2018.

[38] David J. Murray-Smith. *Concepts of Simulation Model Testing, Verification and Validation*, pages 19–33. Springer International Publishing, Cham, 2015.

[39] NumFOCUS. Pandas documentation, 2022.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] Luc Pronzato and Werner G. Müller. Design of computer experiments: space filling and beyond. *Statistics and computing*, 22(3):681–701, 2012.

[42] Hyatt Saleh. *Machine Learning Fundamentals*. Packt Publishing, 2018.

[43] Raschka Sebastian and Mirjalili Vahid. *Python Machine Learning - Second Edition : Unlock Modern Machine Learning and Deep Learning Techniques with Python by Using the Latest Cutting-edge Open Source Python Libraries.*, volume 2nd ed. Packt Publishing, 2017.

[44] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, 2004.

[45] William Silvert. Modelling as discipline. *International journal of general systems*, 30(3):261–282, 2001.

[46] João Victor Soares do Amaral, José Arnaldo Barra Montevechi, Rafael de Carvalho Miranda, and Wilson Trigueiro de Sousa Junior. Metamodel-based simulation optimization: A systematic literature review. *Simulation modeling practice and theory*, 114:102403, 2022.

[47] András Sóbester, Alexander I. J. Forrester, David J. J. Toal, Es Tresidder, and Simon Tucker. Engineering design applications of surrogate-assisted optimization techniques. *Optimization and engineering*, 15(1):243–265, 2014.

[48] Ben H Thacker, Scott W Doebling, Francois M Hemez, Mark C Anderson, Jason E Pepin, and Edward A Rodriguez. Concepts of model verification and validation. 2004.

[49] Joachim van der Herten, Tom Van Steenkiste, Ivo Couckuyt, and Tom Dhaene. Surrogate modelling with sequential design for expensive simulation applications. In Dragan Cvetkovic, editor, *Computer Simulation*, chapter 8. IntechOpen, Rijeka, 2017.

[50] Esther Villar, Eneko Osaba, Jesus L. Lobo, and Ibai Laña. Introductory chapter: Artificial intelligence - latest advances, new paradigms and novel applications. In Eneko Osaba, Esther Villar, Jesús L. Lobo, and Ibai Laña, editors, *Artificial Intelligence*, chapter 1. IntechOpen, Rijeka, 2021.

[51] G. Gary Wang and S Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of mechanical design (1990)*, 129(4):370–380, 2007.

[52] Bianca Williams and Selen Cremaschi. Selection of surrogate modeling techniques for surface approximation and surrogate-based optimization. *Chemical engineering research design*, 170(C):76–89, 2021.

[53] Shichao Zhang, Xuelong Li, Ming Zong, Xiaofeng Zhu, and Debo Cheng. Learning k for knn classification. *ACM transactions on intelligent systems and technology*, 8(3):1–19, 2017.