Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Michael A. Lindbak

Gaussian process-based grey-box modelling of heat exchanger networks

Machine learning applied to disturbance prediction in a heat exchanger network

Master's thesis in Chemical process engineering
Supervisor: Sigurd Skogestad
Co-supervisor: Lucas Ferreira Bernardino
June 2021

NTNU
Norwegian University of
Science and Technology

Michael A. Lindbak

Gaussian process-based grey-box modelling of heat exchanger networks

Machine learning applied to disturbance prediction in a heat exchanger network

Master’s thesis in Chemical process engineering
Supervisor: Sigurd Skogestad
Co-supervisor: Lucas Ferreira Bernardino
June 2021

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Dedicated to my grandfather,

Thank you for always supporting me throughout my studies.

Abstract

Supervised learning is a field that is rapidly growing within the world of science as the popularity of machine learning (ML) is ever growing. The potential these methods hold, not only in simpler classification methods but also within surrogate models for process optimization, is potentially limitless, and it is a appealing topic of investigation that will be considered in this paper. This paper considers a heat exchanger network using a three-way parallel split, where the goal of the process optimization is to adjust the split ratio in such a manner that the greatest outlet temperature is achieved. This was approached by two ML models, one with a direct classic black-box approach, and one that uses a intermediate prediction step of a set of process parameters. These two models were then used to predict the optimal valve configuration for the split configuration. The performance of the models was evaluated by comparing the obtained plant inputs with analytically calculated optimal values, and see how much the final outlet temperature deviated from its optimal as a result of the predicted inputs. The goal of this project was to evaluate whether the intermediate prediction step in the grey-box model produced any significant benefit or disadvantage compared to a direct black-box approach, and to evaluate any additional benefits of using such a model structure. From the simulation results it was found that when opting for ideal priors for each structure, both methods had fairly equal performance. During further comparisons between the two structures, it was found that the black-box model has more predictions closer to the optimal values, while the grey-box model seemed to excel at keeping the overall temperature loss reduced. This was seen by how the overall spread of the temperature loss of the black-box model was greater than the grey-box. Due to this, no clear winner could be determined as each method seemed to posses its own fair share of strengths and weaknesses. Regardless, the presented work shows that there is potential within grey-box surrogate optimization approaches, and that with further investigation it could grow to outperform the traditional black-box model, as the obtained model seemed to have a better understanding of the dynamics in place. It is strongly believed that with further investigations and improvements to the approach, such as a deeper evaluation of measurement and disturbance parameter selection, a more adequate model can be developed. Other suggestions for improvements include Gaussian process regression networks (GPRN), which is shown to have strong empirical performance for finding correlations between disturbance parameters.

Preface

This report is the result of a chemical process masters degree given to graduate students at the Norwegian University of Science and Technology (NTNU) as they finish their graduation term at the university. The project falls under the course ”TKP4900 Chemical process technology masters”. While most students taking this course opt to continue work from their previous semester from a masters project, the work presented in this paper is work resulting of a semesters worth of work from the author. Having made the decision to switch research topic mid-year, the gratitude towards my supervisor cannot be understated as surely I would not have made it this far without our weekly meetings and guidance sessions over mail. Since the subject switched mid-year it is understandable that progress started somewhat slow, and it was clear that a lot of learning was due in order to catch up on the relevant subject before proper work could commence.
Nonetheless, the author is proud to present the resulting work in this paper and I truly believe that something of value has been created throughout the restless hours of work in the final hours of putting this together. I can only hope that the value the reader sees can measure up to that extent as I proudly present the reader with my study on Gaussian process-based grey-box modelling of heat exchanger networks.

List of Symbols

Variable	Description	Unit
T_0	The inlet temperature of the cold stream in the case study	[K]
w_0	The heat capacity and mass flow of the inlet cold stream of the case study	[kWK^{-1}]
α_1, α_2	The valve openings, also labelled u_1, u_2 as the inputs of the plant	[-]
UA_{1-3}	The overall heat transfer coefficient for each of the heat exchangers in the system	[kWK^{-1}]
wh_{1-3}	The heat capacities of the hot inlets of the heat exchangers in the system	[kWK^{-1}]
Th_{1-3}	The inlet temperatures of the hot streams in the heat exchanger network	[K]
$Th_{1-3}e$	The final temperature of the hot stream of the respective heat exchangers	[K]
T_{1-3}	The resulting temperature of the cold stream of each of the splits	[K]
T	The final outlet temperature of the heat exchanger network	[K]
ttratio	A ”test training ratio” used to adjust the generated dataset for model predictions	[-]
LMTD	Logarithmic mean temperature difference	[K]
q	Energy transfer resulting from heat transfer	[Wm^{-2}]
F_j	Mass flow of a stream j	[kmol s^{-1}]
Q_j	Energy from heat transfer for stream j	[W m^{-2}]
YD, **DU**. **YU**	Labelled machine learning models for their respective inputs and outputs	[-]

List of Figures

Table of Contents

1 Introduction

Mathematical optimization is a classic computer science and engineering problem which has existed for centuries. In its simplest form, an optimization problem consists of the process of maximizing or minimizing (finding the greatest or smallest value) a real *objective function*. The objective function is typically set as either a loss/cost function (in which minimization is the preferred optimization) or a utility or fitness function (vice versa).[7] The goal is thus to systematically choose input values from within a feasible set and computing the value of the objective function in order to find the best possible set of inputs. Typically, these optimization cases come packaged with some sort of restraints on the inputs, which is what was referred to when talking about a feasible set.[8]

Machine learning (ML) is a field of study under artificial intelligence that can be specifically tailored to perform regression analysis and is used frequently in model predictions. ML is typically exposed to a set of training data sets, which is used for the machine to ”learn” regression pattern and model structure, in order to be able to predict optimal values. One method which excels at this front is neural network trained ML. These networks consists of a set of *nodes* that are paired in smaller subsets called *layers*. All nodes in layer i, are connected to all nodes in layer $i+1$ through a set of signal wires, and the strength of the signals is represented by the *weight* of the nodes in layer i. During operation, a ”input” will stimulate nodes in the first layer, which then passes through all i layers, before it produces a output determined by how the input is affected throughout by all the weights within the various layers.[9]

While still a underdeveloped field, it is strongly agreed upon that, the usage of ML for chemical engineering is a field that has great promise.[10] This is because the method is especially good for processes where parameters can be difficult to measure, or the existing models are found lacking. By today, there are already multiple implemented existing algorithms in the chemical industry such as artificial neural networks, fuzzy logic, genetic algorithms and evolution strategy which are used in processes with parameters that are difficult to measure.[11]

One particular model, which showed great promise within regression analysis, was the aforementioned model using a neural network. This method was designed to be able to utilize adaptive basis functions, which allowed them to *learn* ”hidden features” (hidden here meaning not directly obvious when analyzing the data) within the modelling problem.

While often seen hand-in-hand, the study of statistics and that of machine learning can be argued to vary greatly. The goal of statistics is to obtain a comprehensible model for a set of data and be able to obtain relationships and dependencies for the data. On the contrary, the field of study that is machine learning is mainly concerned with making output predictions with the highest accuracy possible. The difference here lies in the fact that the machine learning community is rarely concerned with the actual ”shape” or ”form” of the model, and is mostly satisfied with using a black-box model. In some cases, these black-box models can be adjusted slightly if some trend of dependencies is expected, by which we will have created a grey-box model. A grey-box model is any model that combines elements of a white-box model and black-box model. A white-box model is here defined as a purely theoretical one, while a black-box model has no model form and prior presumptions.[12] While these types of models may often bring outstanding prediction results, many statisticians would doubtlessly argue that these models are substandard as they do not provide a thorough enough breakdown of the data and thus unsubstantial understanding of the matter.

While this was traditionally the case for statistics and machine learning, orienting the problems towards *Gaussian processes* (GP) seemed to be a solution that would fit both camps. While the model is computationally simple to implement, the model is also comprehensible on a logical scale - thus making it a good middle ground between the two fields. A GP is obtained through a generalization of a Gaussian probability distribution.[3] One of the main strengths of the Gaussian process framework is the fitting capability it possesses. Additionally, as long as the training sets are not remarkably large, these processes also managed to retain a fairly decent level of computational tractability. Gaussian processes can be visualized in a simplified way by thinking of a function as a very long vector containing every possible function value, $f(x)$, for every desired x value. By leaving out the infinite possible values for x, and only considering a set finite number of points for it, we obtain the same answer through inference in the Gaussian process as if we had ignored infinitely many other points. In doing so, we can assign a probability for all our function values $f(x)$, on how likely they are to fit our data, and by taking the mean of our probability distribution we can obtain a "most likely" characterization of our data.[3]

Usage of GPs has been popularized strongly within the world of ML as the method has several useful properties that make them ideal. One of the big strengths that lie in GPs is that of the model being *non-parametric*. This means that when adjusting a GP to new data, we do not have to worry about the pre-existing mean representation not being able to fit the data. Despite this, the model also has good *analytic inference* in that the predictive posterior distribution can be computed exactly in its closed form. As we will see, adjusting the prior knowledge of the model is a fairly straightforward task, as one can combine kernels in any desired manner, based on the process to be modelled, which allows for great flexibility in evaluating different approaches for the model.

GPs are however not omnipotent, as they also hosts a fair share of weaknesses. One of the greatest is that the process has a relatively slow inference, as computing the $n\times$ n matrix inverse of the predictive distribution equation (Equation 3.13) takes $\mathcal{O}(N^3)$ operations. While also mentioned as a strength of GP that one can chose kernels in order to obtain various GP approaches and thus more flexibility in model approach, this can also be viewed as a down-side as substantial pre-requisite knowledge is required on the subject in order to make a good kernel suggestion. This has been improved on in later years as one can set a computer to automatically find the proper set of kernels by maximizing marginal likelihood (Abdessalem et al.[13]).

Heat exchanger network (HEN) is a design technique based on coupling together heat exchangers in a optimal set of series and/or parallel to achieve the greatest energy saving within the operation. One possibility for HEN synthesis is to use *pinch technology* in order to find what combinations gives most ideal heat transfer between all streams involved.[14] In this paper, a slightly different approach is considered as the design of the network (explained further under Section 4.2) is defined as a fixed structure of three heat exchangers in parallel setup. The strategy of the optimization is thus to utilize ML in order to find a optimal set of valve openings in each parallel for the output temperature to be maximized. The most straight-forward way to approach this would be to take a set of measurements and use these to predict the optimal set of valve openings, with the goal being to achieve the greatest outlet temperature. The main emphasis for this report, however, is on the usage of a grey-box model, and thus the main objective will be to compare the efficacy of a direct black-box approach to a customized grey-box approach that utilizes a set intermediate parameter predictions. The idea is based on the potential advantages of knowing the intermediate values, and to evaluate if by doing this we can create a model that has a better understanding of the system, and thus can give better predictions.

1.1 Structure of Report

The thesis will proceed with a further delve into Machine Learning in Section 2. Here a couple of key concepts will be introduced and explained before moving onto a more extensive look at Gaussian processes in Section 3. This section will further introduce all the general concepts and mathematical concepts that lie behind the Gaussian processes and ML models and elaborate how we are able to achieve the results we have. We then move on to a further description of the system that is considered in Section 4, before doing a breakdown of the methodology used to solve the optimization problem in Section 5. The results will then be presented under Section 6 where all case studies will have graphs illustrating the performance of the various methods employed. These results will then be further discussed and evaluated under Section 7, before finally concluding under Section 8. The appendix will feature the result graphs in a side-by-side manner for easier comparisons in Appendix A, as well as all code used in the thesis itself in Appendix B.

2 Machine Learning

One major factor that differentiates humans from machines is the way machines simply follow a set of instructions, while we are able to learn from experiences and adapt. While machines most often are used to solve explicit problems but computationally heavy problems, no learning is typically required. Machine Learning (ML) comes in handy when there exist no fully satisfactory algorithm which we can program a computer to execute. We will therefore have the machine ”learn” through training datasets in order to be able to best predict a set of outputs given a set of inputs. ML is typically divided into three categories depending on the amount of human intervention required for it to operate:

- **Supervised learning:** The computer is given a set of inputs which is to be mapped to a set of ”desired” outputs given by a teacher or supervisor.
- **Unsupervised learning:** The computer is given non-labeled, non-structured data which it is to find a structure on its own.
- **Reinforcement learning:** The computer is reinforced of its outputs based on a feedback system that either penalizes ”wrong” behaviour or rewards ”good” behaviour.[15]

For this paper, the topic of supervised learning will be considered. This is the kind of learning where input-output mapping is considered from a set of training data. Based on the nature of the output, the learning is either considered as *classification* (in the case of discrete outputs), or *regression* (in the case of continuous outputs).

2.1 Classification learning

The most basic form of machine learning can be argued to lie within the basic decision tree type of learning that operate with classification. These type of decision trees are a set of algorithm pathways that predicts a output through a set of Boolean (true or false) conditions. The tree could be as simple as attempting to discern gender of a person on the basis of height and weight, where the tree would look something like Figure 2.1.

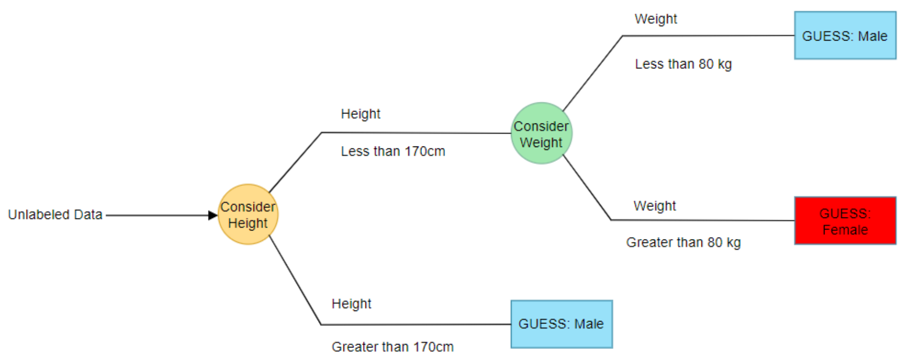


Figure 2.1: A simple and possibly completely inaccurate decision tree of a hypothetical scenario of determining gender based on the parameters weight and height. The tree is just made to be an example of how one can classify unlabeled data into a predetermined group based on certain characteristics of said data.

The figure was created arbitrarily for the sake of the report and is based on no scientific evaluations. In this example, the input of the classification algorithm is mass and height. By using labelled training data, we can improve our model by modifying the fitted parameters such that the mass and heights rules correlate to a better likelihood of correct output prediction.[16]

When working with real-life engineering problems however, simply relying on a decision tree like this has obvious flaws in that the prediction can often be inaccurate and thus unreliable. Additionally these kinds of trees need tweaking and are not easy to visualize with when the amount of discriminatory rules increases. One way that was looked into to bettering this was in terms of using multiple decision trees and combining the outputs to provide a more accurate prediction, which later on was cleverly dubbed forests.[17] These types of structures were later on evaluated to be closely resembling the way neurons work in the human brain, and through years of research an approach of artificial neural networks was discovered. For this ”new” field of study, the term ”classification branches” along the trees were replaced by nodes, and the ”trees” were replaced by layers. The concept remains the same, in that we have labelled data used for the input-to-output tweaking of the system. However the algorithm now uses these layers and a natural *bias* to adjust numbers through each layer. All of which, in the end, results in a likelihood prediction in the case of classification.[18] A classic example of a artificial neural network like this is illustrated below in Figure 2.2.

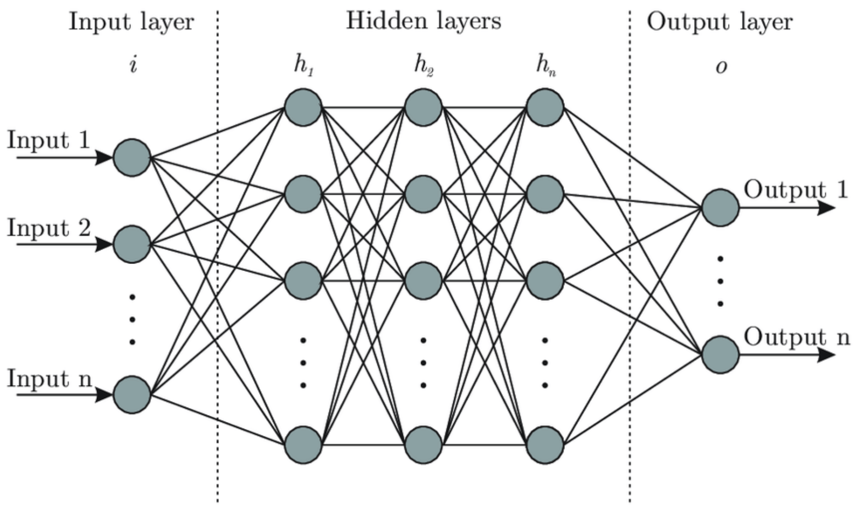


Figure 2.2: A simple illustration of a artificial neural network, images rights are credited to *Bre et al.*[1]. The image shows how a set of inputs is passed through *n* layers before becoming a specific output. The signal is being passed as a numerical value and transformed by a *weight* for each of the lines in the network. The final value of the numerical value in the output layer is the numerical values that the network computes.

2.2 Regression

While classification is characterized by the application of discrete labels to objects, regression is concerned with prediction of continuous quantities. For example, a socioeconomist might wish to examine how the usage of cigarettes correlates with various demographic factors such as age, location, education, cigarette pricing and income.

The simplest form for regression models are arguably those of linear nature, where we have either a single variable or multiple variables that are scaled to form a linear result. While simple to understand and implement, the method is not particularly useful aside from a handful of cases, or situations where the inputs are pre-treated through iterative feature engineering. In this paper, a more comprehensive method is used, namely that of Gaussian processes. In short, this class of methods predicts a normal distribution for any point of interest. This will be looked into further in Section 3.

In general, when working with regression cases, the input is labelled as x, while the targeted output is labelled as y. When considering a training dataset for the regression, we typically classify this as the set $\mathscr{D}$, which contains the n training points $\{(x_i, y_i) \,|\, i = 1, ..., n\}$. It is important to note here that depending on the case in question, both x and y can be multivariate. In the earlier socioeconomic example, the input, x, would be all the factors of age, location, education, cigarette pricing and income - while the output, y, could be cigarette consumption. After successful training, the goal of the model would be to be able to predict a unknown output, y^*, for a given set of input variables, x^*, with acceptable accuracy. The accuracy of the model would ideally be tested with a prediction from a input x^* from which we already have a expected output value y^* and then we measure the deviancy of the prediction with relation to this expected output. In most real cases, we cannot expect the model to operate with a 100% accuracy for these kinds of problems, due to possible measurement noise or lack of data in the training.

2.3 Surrogate modelling of process systems

A major bottleneck in the implementation of real-time optimization in process systems is that detailed models, while accurate, require extensive computational power. This is not only limited by the amount of processing power a modern computer can work with, but also the fact that these calculations can be slow for more intricate process systems. If the time needed to compute input parameters for a plant arrives by the time the plant already has moved on to a significantly different state, we would naturally get a deviation from optimal operation. Surrogate optimization is the ideology of finding an alternative model that can solve this optimization problem within an acceptable margin of error.[19]

Ideally we can imagine this as having a set of real world influences that affects our process. The first step of surrogate optimization is therefore to discover the set of influences that actually have causality over our process. From this, we attempt to create a virtual representation of our problem, sometimes referred to as a digital twin model. This digital twin model is the part of the model that acts as a surrogate for which we can find optimal solutions to our process. If we find that the model is insufficient in its predictions, we can utilize a feedback system where the optimal predictions then fed back into our model such that we can modify it to give better predictions.[20]

In terms of this paper, a surrogate model will be applied for a heat exchanger network where a cold stream is split into three parallel streams that each pass through a heat exchanger (more thoroughly explained under Section 4). While this sort of case can be solved completely analytically given the right parameters, this is not always feasible for real world cases. For example, it can be extremely hard to measure the overall heat transfer coefficient for a given heat exchanger, and one must rely on model-based disturbance estimation techniques, which are costly on their own. Since the situation becomes more difficult to solve for these cases, a surrogate model is implemented to bypass these issues that arise from lack of knowledge about the system.[21]

2.4 ML evaluation

When evaluating ML models, there is a wide array of possibilities to chose from and typically one has to chose depending on what type of prediction the model is doing in order to find a proper evaluation. For classification models, the most commonly used metrics to evaluate model performance are confusion matrices or a receiver operating characteristics (ROC) curve.[22] On the other hand, evaluation of regression methods, which is the class of models considered in this paper, is commonly assessed by mean squared error or scatter plots.[23] This work has mainly considered scatter plots, as well as a boxplot, as these were deemed most relevant for the case. Scatter plots are often great visualization tools for model evaluation as they show the true, desired value of the output variable and compares these to the ML predictions in a plot. An example of a scatter plot is illustrated below in Figure 2.3. As these plots contain the predicted values and the actual ones, we have that points closer to the prediction line (linear $y = x$ line) indicate better predictions. Often when working with figures like these, terms like ”scatter cloud” are frequently used. This is used to talk about the general shape of all the plotted data points, as once the amount of them increases they tend to resemble a cloud. By visualizing it as this we can also use terms like cloud density or sparsity to classify how concentrated the points are.

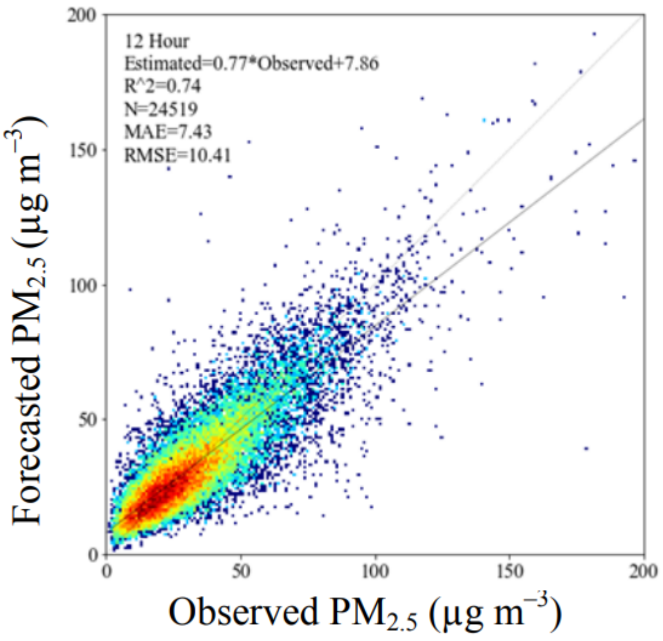


Figure 2.3: Example of a scatter plot used to evaluate the efficacy of a ML model. The image is from the Karimian et al.[2] and shows the forecasted (predicted) $PM_{2.5}$ concentrations to determine air pollution in a local area.

3 Gaussian Processes

Gaussian processes (GP) is named after its creator Carl Friedrich Gauss based on his notion of a Gaussian distribution (also known as a normal distribution).[24] Simply speaking, for any set $S \in R^D$, a GP on S is a set of multi-dimensional random variables $(Z_t : t \in S)$ such that Equation 3.1 is satisfied. In this expression, $N(\vec{\mu}, K)$ represents a normal distribution with mean vector $\vec{\mu}$ and covariance matrix K.

$$\forall n \in N \quad , \quad \forall t_1, t_2, ..., t_n \in S \quad , \quad (z_{t_1}, z_{t_2}, ..., z_{t_n}) \sim N(\vec{\mu}, K) \tag{3.1}$$

In lingual terms, this means that the created GP on S, $(z_{t_1}, ..., z_{t_n})$, is a set of finite dimensional distributions that also are Gaussian distributed. From this we can describe a GP as a distribution over functions, meaning $f \sim GP(\mu, k)$. Before conditioning a GP to any data, the function is thus specified by its mean function $E[f(x)] = \mu(x)$ and its covariance function (often referred to as a *kernel*) $\mathrm{Cov}\big(f(x), f(x')\big) = k(x, x')$ which is in the domain R^D.

The prior mean of a GP is typically set to be 0 for all $x \in S$, as any prior mean can be accounted for at a later point by simple manipulation of the regression values. In doing so, it follows that the structure of the GP model is entirely determined by its kernel.

One of the big benefits of GPs is the possibility of writing the *marginal likelihood* of our constructed GP model.[25] By evaluating the marginal likelihood we are able to compare the performance of various models in terms of adequacy to the data. The marginal likelihood for a GP prior using a set of function values $f(\mathbf{X}) = [f(x_1), f(x_2), ..., f(x_N)]$ at positions $\mathbf{X}$ is defined below in Equation 3.2.

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|\mathbf{f}, X) p(\mathbf{f}|X)\, d\mathbf{f} \tag{3.2}$$

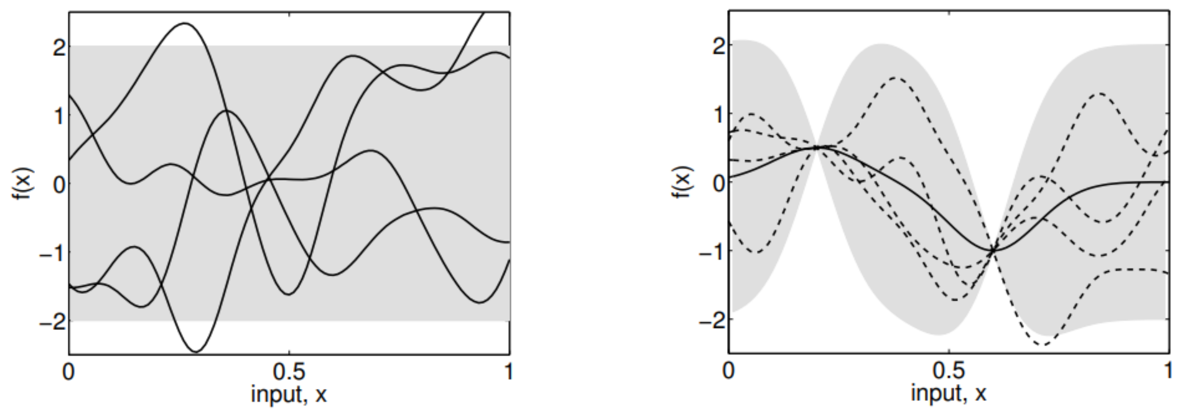
The term *marginal* refers to the implicit integration (marginalization) over all function values of f. Through some derivations (see Seeger[31]), we can solve the integral and obtain a log marginal likelihood shown in Equation 3.3.

$$\log p(y|X) = -\frac{1}{2} y^T (K + \sigma_n^2 I)^{-1} y - \frac{1}{2} \log|K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi \tag{3.3}$$

3.1 Priors

A GP is operated through a baseline known as a *prior*. A prior is the base assumption used before any kind of model approaches are applied. This is, as the name suggest, a *prior* assumption of what we expect to observe from our function.[26] In a case where we have no knowledge of our model beforehand, this prior would usually be a function that returns zero for any value of x. Priors also usually has a uncertainty region, which represents how likely it is to reject the prior assumptions.

When combining the prior with a set of measured datapoints, we obtain what is called a *posterior distribution*. This is illustrated below in Figures 3.1, where we can see how the distribution is changed with the introduction of new knowledge, and how the new realization of the Gaussian process (the lines in the figure) adjust to the data points.



(a) A figure showing the prior of a possible GP. **(b)** A figure of the posterior resulting from inputting two data points.

Figure 3.1: Figures showing how we achieve a posterior from a prior and its set data points. The mean prediction is represented by the solid line, while the dashed lines represent four sample posterior functions. The shaded region is the uncertainty in the predictions, which is found by two times the standard deviation of each input, x. Image credits goes to Seeger[3].

By supplementing more datapoints into a posterior, we can imagine how the shaded region becomes significantly smaller as it collapses near all the datapoints to give a proper fit for the GP function. Since GP also is a non-parametric model, we do not have to worry about the model not being able to pass through all the points. This is however not the case when working with GP models where data noise is possible. For these models, it is desired that the model to not pass through *all* points as it only means it is trying to fit the measurement noise in a deterministic manner, which in turn gives poorer predictions. In the event where we adjust a GP model to adjust for all training points and it results in poor prediction due to uncertainty and errors, we have a case called *overfitting*.[27] Poor predictions from overfitting is especially sensible when either datasets are too small, or there is a substantial lack of prior knowledge of the modelling problem.

Selecting the proper prior is another important aspect of the pre-processing work that goes into creating a GP grey-box model. This is also utilized in this paper where we apply a total of three different priors to both the black-box approach and the grey-box approach. As we will see in Section 5.3, these prior assumptions about a model are quite important and can greatly impact the performance of a model. The properties of a prior are mainly consisting of the uncertainty area and the *realizations of the GP* (the lines in Figures 3.1).

3.2 Covariance functions

The covariance function $k(x,x')$ is a measure of the similarity between two measurement points x and x'. By assuming similar data points to have similar function values, we attain inference in our data set. By definition, we have that the covariance function needs to be symmetric ($k(x,x') = k(x',x)$) and it needs to be positive semidefinite.[28] A function is defined as positive semidefinite if the inequality in Equation 3.4 is satisfied. For this equation, $f(x)$ is a short-hand way of writing $GP(m(x),k(x,x'))$, and $\mu(x)$ is a mean function used on x.

$$\int k(x,x')f(x)f(x')d\mu(x)d\mu(x') \geq 0 \tag{3.4}$$

A *kernel* is a function which maps pairs of inputs $x \in S$ and $x' \in S$ into $\mathbf{R}$.[3] One of the most commonly used kernel is the squared exponential kernel (SEK),[29] which is defined below in Equation 3.5 for the single-input case. Some examples of other kernels commonly used include periodical and linear kernels, and we pick the desired kernel based on the type of problem we wish to model.

$$k(x,x') = \sigma^2 \exp - \frac{(x-x')^2}{2l^2} \tag{3.5}$$

In this expression, l varies the length (or width) of the kernel, while σ adjusts the height of the normal distribution curve. These are known as the hyperparameters of the kernel and the effects of adjusting these parameters is illustrated below in Figure 3.2, where the mean is set as zero.

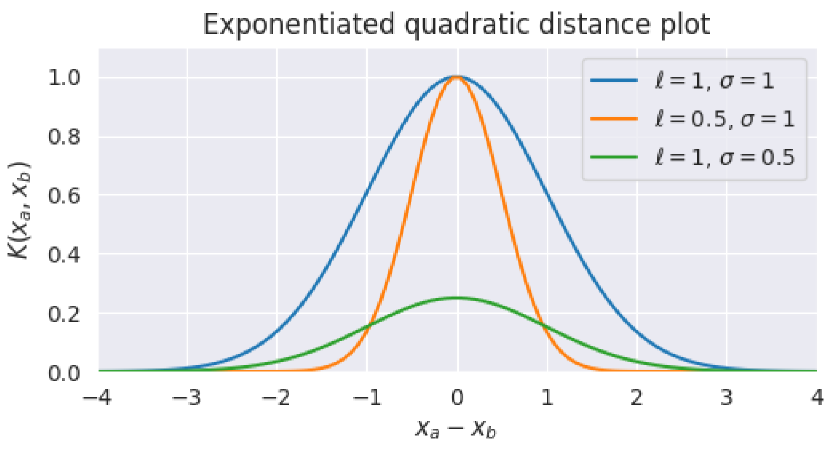


Figure 3.2: Figure showing a exponential quadratic distance plot (a example of a SEK kernel). The figure shows comparisons the effect of varying l and σ on the normal distribution. The figure shows how l varies the width of the curve, while σ adjust the height of the curve. The image is credited to Roelants[4].

When this kernel is applied as a function however, these parameters take on a different interpretation. The length-scale l is used to describe the smoothness of the function. When using a small lengthscale, the function values are allowed to change quickly, while greater values make for a smoother graph with less steep changes. The signal variance σ^2 is used as a scaling factor. Smaller values of this can be recognized as functions that stay close to their mean values, whereas larger values of this value lets the function chase outliers. Another term that was omitted from Equation 3.5 is a *noise variance* term, which is included when there is additional noise in the data that needs to be considered. This parameter is used to specify how much noise that is expected within the data.[30]

3.2.1 Combining covariance functions

One downside about the SEK however is that it is one-dimensional. Regardless, one major benefit of kernels is the possibility of combining the kernels to get the desired properties of the model we wish to construct. This allows us to combine kernels with the right properties to customize our surrogate model to better fit our process. One can assemble multiple kernels together in order to produce a higher level structure with specific properties.[3]

As an example of kernel combination through multiplication, we can see Equation 3.6 below, where we combine two separate SEKs which gives us a two-dimensional radial-basis functions. The input-parameters of this two-dimensional kernel is therefore a input vector meaning that $\big(\mathbf{x},\ \mathbf{x'}\big) = \big((x_1,x_2),(x'_1,x'_2)\big)$.

$$k_{ij}\big(\mathbf{x},\mathbf{x'}\big) \propto \sigma_1^2\sigma_2^2 \exp\Big(-\frac{(x_1-x'_1)^2}{2l_1^2}\Big)\exp\Big(-\frac{(x_2-x'_2)^2}{2l_2^2}\Big) \tag{3.6}$$

Following the example of combining SEKs, we can construct a kernel using a SEK for each of the inputs of the dataset where each kernel has a different lengthscale parameter. In doing so we construct what is commonly referred to a *squared exponential automatic relevance determination* (SE-ARD) which is one of the most commonly used kernels in most GP applications.[29] The individual lengthscale parameters here are important because the input dimensions have different effects on the output. By having a individual lengthscale for each input we can therefore scale them accordingly.

For this paper however, the goal is to create a single surrogate model for the full dataset, and thus we do not desire to use SE-ARD. In order to ensure that the dimensions of the inputs do not affect the output unevenly, we use data normalization for the training data.

When constructing a single model that accounts for multiple outputs, we use a coregionalized model that takes the form shown in Equation 3.7. Using this kernel allows the model to express the outputs as combinations of independent random functions. The kernel also has the benefit of ensuring that all resulting covariance functions are valid positive semidefinite functions.[31]

$$B \otimes K = \begin{pmatrix} B_{1,1}\times K(X_1,X_1) & \dots & B_{1,D}\times K(X_1,X_D) \\ \vdots & \ddots & \vdots \\ B_{D,1}\times K(X_D,X_1) & \dots & B_{D,D}\times K(X_D,X_D) \end{pmatrix} \tag{3.7}$$

By using this definition we have that the covariance function is the chosen kernel applied ith function at X and the jth junction at X', multiplied with the (i,j)th entry in the matrix of B. To ensure that this is a valid kernel we must uphold the criteria of the matrix being positive semidefinite. We do this by defining the coregionalization matrix, B, as shown in Equation 3.8.

$$B = WW^T + \mathrm{diag}(\kappa) \tag{3.8}$$

The definition is defined for some matrix W and vector κ such that the coregionalized matrix is positive semidefinite. This is regarded as the *intrinsic model of coregionalization* (ICM).[32] One major benefit of this model is that is allows for shared information across outputs, a feature which independent models cannot. In cases like this, if there is a region where a lack of training data is present, the independent models tend to collapse to their prior assumptions. In the case of an ICM, if the models have associated patterns, the overall fit is better.[33] When working with noise-free data, the ICM model is mathematically equivalent to predicting each output independently. This property is known as *autokrigeability*.[34]

3.3 Gaussian Process Regression

When using Gaussian Process for regression the methodology follows a set of steps. This includes having a training data set and a test data set which are the inputs for the GP. The output of said process can be either a single value, or a set of values. Since ML models possess fundamental limitations, they posses critical flaws that can prove detrimental for the predictions. If the models are tuned poorly, and we need predictions far outside the training region of the models, we can end up with exceptionally poor results.[35] It is due to this that evaluation of the confidence interval is an additional point that is of grave importance when working with predictions for ML, and it has become a increasingly attractive field of study.

GP regression uses a Bayesian approach and infers a probability distribution over the output space, given any test input in the considered set S. In parametric Bayesian modelling, this is done by first specifying a prior distribution, $p(w)$, on a parameter w. From this a posterior is constructed by relocating the probability distributions based on training data using Bayes' theorem.[36] The prior can be obtained my modifying Equation 3.2 to obtain the posterior distribution, shown in Equation 3.9. The posterior is defined as shown in Equations 3.10, which is the combination of prior and the dataset.

$$p(w|y,X) = \frac{p(y|X,w)p(w)}{p(y|X)} \tag{3.9}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}} \tag{3.10}$$

The posterior probability distribution of the parameters can be further propagated to the output predictions, as shown in Equation 3.11.

$$p(f^*|X^*,y,X) = \int_w p(f^*|x^*,w)p(w|y,X)dw \tag{3.11}$$

3.3.1 Predicting with Noise-free Data

Using the aforementioned kernels $k(x,x')$, we can create a *covariance matrix* $K_{ij} = k(x_i,x_j)$ that has all the covariances for all the points of our inputs $x_i \,\|\, i = 1,2,...,n$. Note however, that we can only create a covariance matrix if our choice of kernel is positive semidefinite.[37] By drawing samples from the distribution of all the functions evaluated in the covariance functions, we can see that the specifications of the covariance functions implies distribution over functions. We do this by selecting a set of prediction points, X^* and write out the corresponding covariance matrix, using our previously defined SEK (Equation 3.5). Additionally, we generate a Gaussian vector with the created covariance matrix $(K(X^*,X^*))$ and obtain Equation 3.12 for the prior distribution of the prediction, assuming a zero prior mean.

$$f_* \sim \mathcal{N}\big(0, K(X^*,X^*)\big) \tag{3.12}$$

In most cases, including this, we assume that the likelihood and the prior are Gaussian, thus making the predictive distribution a Gaussian distribution which we can solve to obtain a prediction for our points X^*. This is done by finding the hyperparameters of the distribution, namely the mean and variance which is done using the *maximum a posteriori* probability of y. This is also commonly referred to evidence maximization or empirical Bayes.[38]

Regression predictions are obtained using prediction points X^*, and a predictive distribution around said point $f^* := f(X^*)$. We start by *conditioning* on the joint Gaussian prior distribution on the observations[3] to construct the following *predictive distribution*, shown in Equation 3.13, for f^*.

$$f^*|X^*,X,f \sim \mathcal{N}\left(K(X^*,X)X(X,X)^{-1}f, K(K(X^*,X^*) - K(X^*,X)K(X,X)^{-1}K(X,X^*)\right) \tag{3.13}$$

From this equation, we have that function values of f^* can be sampled from the joint posterior distribution through evaluation of the covariance matrix and mean. We do this by deriving the conditional distribution from Equation 3.13. In doing so, we obtain the key predictive equations when using Gaussian process regression, namely Equations 3.14 and 3.15 for the mean and the covariance respectively.[3]

$$\bar{f}^* = E\left[f^*|X,y,X^*\right] = K(X^*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1} y \tag{3.14}$$

$$\mathrm{Cov}(f^*) = K(X^*,X^*) - K(X^*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1} K(X,X^*) \tag{3.15}$$

4 Case Study - Parallel Heat Exchanger Network

Energy saving and optimization is a big topic of interest within chemical process technology. Since these plants commonly operate at big scales, any energy that can be recovered is worth looking into, lest it be wasted and thus losing out on a major cost reducing factor. This can be accomplished by heat exchangers, and it was during the energy crisis in the 1970s that sparked a great research field within further optimizing heat exchangers through the introduction of *pinch technology*.[39] This design methodology is based on minimizing energy consumption in a chemical process by calculating the best configurations for heat exchangers to decide pairings, and positions which maximizes feasible energy transfer. In the operation of a heat exchanger network that presents parallel configurations, such as this case study, the streams can instead of a regular split utilize selective vents by which we can control the exact amount sent through each parallel. The case study for this work is illustrated below in Figure 4.1.

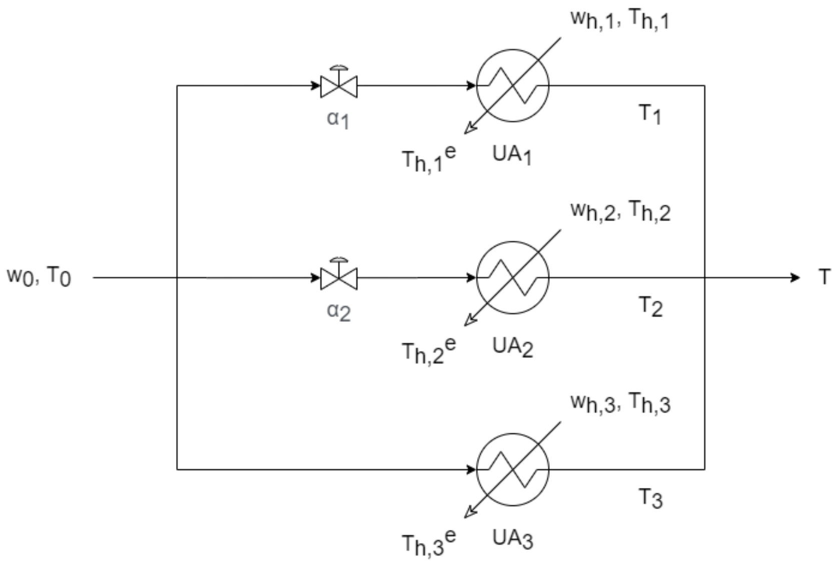


Figure 4.1: A figure illustrating the case study for this thesis. The case study consists of a heat exchanger network where the inlet stream, with heat capacity w_0 and temperature T_0, is split in three. The three resulting streams are then fed into three separate heat exchangers with separate heat capacities and temperatures, $w_{h,1-3}$ and $T_{h,1-3}$, and overall heat transfer coefficients UA_{1-3}. The stream is then merged for a final stream with temperature T. The image is credited to Chen[5].

The system entails a cold stream being sent into a three-way split using the vents α_1 and α_2. Each of the parallel streams are then fed to a different heat exchanger. These heat exchangers have their respective heat transfer coefficient UA_{1-3} and hot streams with parameters $w_{h,1-3}$ and $T_{h,1-3}$ for their heat capacities and inlet temperatures respectively. The parameters for the heat capacities are simplified to entail the product of the mass flow with the heat capacities of the streams. During operation, we also obtain the parameters $T_{h1-3}e$ for resulting exit temperatures of the heat exchanger streams. Additionally, as the cold streams are heated, we obtain the temperatures T_{1-3} before they are merged again to obtain a final outlet stream with temperature T. As this is a heat exchanger network, the goal is then to be able to implement a control of the vents, α_{1-2}, such that the outlet temperature, T, is the highest possible.

Since this system is completely modelled and readily available from a preset of code, it could be tempting to consider every available possible parameter for the prediction algorithm. Since this case would leave a lot to be desired from a research perspective as its lacking a lot of realistic limitations, the model parameters selected has been consciously selected such as to resemble a real-life case as closely as possible. The goal of the model is to be able to take a set of measurement parameters from the system, labeled y's, and use these to predict a set of process parameters labeled d's. Part of the y's is a set of current valve calculations $\alpha_{1-2,estimated}$ included, as these showed to provide a lot of necessary information about the system. The measurements were used in some sort of manner to try and predict u^*, for α_{1-2} that grants the highest possible outlet temperature for T. The parameters y's, d's and u^* are defined as the following process parameters shown below in Equation 4.1.

$$\vec{y} = \begin{bmatrix} \alpha_{1,current} \\ \alpha_{2,current} \\ T_1 \\ T_2 \\ T_3 \\ T_{h,1}e \\ T_{h,2}e \\ T_{h,3}e \end{bmatrix} \qquad \vec{d} = \begin{bmatrix} wh_1 \\ wh_2 \\ wh_3 \\ UA_1 \\ UA_2 \\ UA_3 \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} \alpha_{1,optimal} \\ \alpha_{2,optimal} \end{bmatrix} \tag{4.1}$$

This method will also be evaluated by the more direct approach of measurements, y, straight to valve configurations, u^*. This will be done to see if there is any benefit to having the disturbances, d's, as a buffer calculation step before the configurations are calculated. The benefit of calculating this buffer step is that they provide slightly more insight into what is actually going on in the grey-box model. When working with grey box models like this, it is not highly unusual for some set of data points to give very inaccurate predictions. While the reasons for this can be many, by providing these sets of buffer points for the model, we can not only investigate these data points further to see if there is a trend in the model - but the points also act as a sort of ”anchor” to keep the model from going astray.

4.1 Model assumptions and simplification

In order to model this system we also make some simplifications in order to not overcomplicate the calculations. One of these assumptions is that we keep a single phase throughout the heat exchanger network. This implies that none of the streams involved will have a phase change occurring during the heat transfer, thus neglecting effects of latent energy.

The next assumption made is that of *constant heat capacity*. As we know, heat capacity is realistically not constant as it depends on the temperature, pressure and volume of the system considered. In this experiment, the temperature change for the streams is not considerably large. This allows us to set this assumption without much worry, as the heat capacity generally does not change much when the temperature change is small overall.[40]

The final assumption made is that we use a logarithmic mean to determine the driving force for the heat transfer. This approximation was derived by Chen[5] and used as the logarithmic mean temperature difference (LMTD) in this paper. This can be done by assuming a countercurrent flow for the heat exchanger network. And we can define the logarithmic mean temperature difference between ΔT_1 and ΔT_2 as shown in Equation 4.2. These parameters will, from the case study, be represented by $(T_j - T_0)$ for ΔT_1, and $(T_{h,j} - T_{h,j}e)$ for ΔT_2.

$$\text{LMTD} = \frac{\Delta T_1 - \Delta T_2}{ln(\frac{\Delta T_1}{\Delta T_2})} \approx \left\{ \Delta T_1 \cdot \Delta T_2 \cdot \left(\frac{\Delta T_1 + \Delta T_2}{2} \right) \right\}^{\frac{1}{3}} \tag{4.2}$$

4.2 System model

When modeling the system, we start with energy balances on both hot and cold sides, and with expressions to the heat transfer between the streams, aiming to obtain a final set of equations we can use for computational calculations. The system to be modelled is shown in Figure 4.1.

$$\begin{aligned} w_j &= F_j c_{p,0} \\ wh_j &= F_{h,j} c_{ph,j} \end{aligned} \tag{4.3}$$

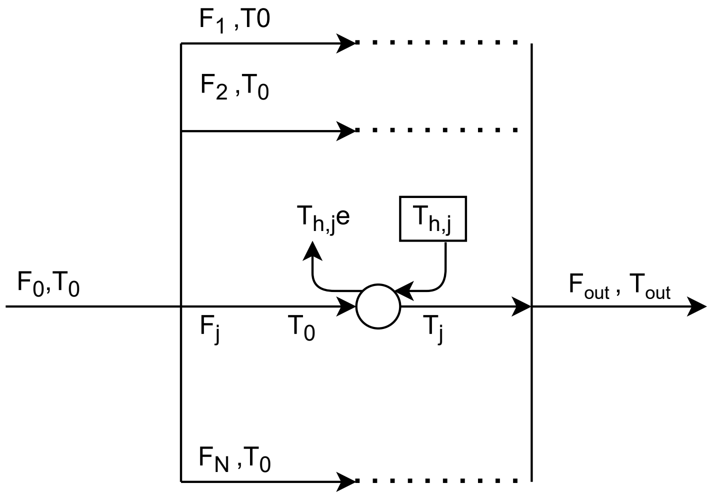


Figure 4.2: A simple illustration showing a heat exchanger system where a inlet stream is split into N parallels. The line j is focused for calculations and is heated using a heat exchanger with temperature $T_{h,j}$ and heat capacity (and mass flow product) wh_j. After being fed to the heat exchanger the stream on line j has its temperature changed to T_j. The figure illustrates a design method for heat exchanger networks where splitting a stream, and then unifying them later on can give great results for the final outlet temperature of the system T. The figure was created using a diagram software developed by Benson[6].

$$\begin{aligned} Q_{j,\text{cold}} &= w_j \cdot (T_j - T_0) \\ Q_{j,\text{hot}} &= wh_j \cdot \left(T_{h,j} - T_{h,j}e\right) \end{aligned} \tag{4.4}$$

The notation here uses w_j and wh_j defined earlier in Equation 4.3 as the product of the mass flows and heat capacities of the respective streams. We see that Q_j, which is the heat transfer occurring in heat exchanger on line j, is given by the total heat transferred shown in Equation 4.5.

$$Q_j = UA_j \cdot \Delta T_{LMTD,j} \tag{4.5}$$

Equation 4.5 has been extended with subscripts to account for the various parameters that we have at the different heat exchangers in the parallel system. $\Delta T_{LMTDi,j}$ here refers to the logarithmic mean temperature, which is shown in Equation 4.2 and is the driving force of the heat transfer.

Finally, we have a overall mass balances and overall energy balances for the system which yields the following Equations shown below in 4.6 and 4.7.

$$\begin{aligned} 1 &= \alpha_1 + \alpha_2 + \alpha_3 \\ \Rightarrow \alpha_3 &= 1 - (\alpha_1 + \alpha_2) \end{aligned} \tag{4.6}$$

$$\begin{aligned} T &= \alpha_1 \cdot T_1 + \alpha_2 \cdot T_2 + \alpha_3 \cdot T_3 \\ \Rightarrow T &= \alpha_1 \cdot T_1 + \alpha_2 \cdot T_2 + (1 - \alpha_1 - \alpha_2) \cdot T_3 \end{aligned} \tag{4.7}$$

Combining all of these equations for the case study, shown in Figure 4.1, we get the following set of equations shown in Equations 4.8. Solving these computationally was done using CasADi's packages for optimization and root-finding problems. All the variables present were set as individual symbolic parameters and the system would solve a optimization problem when finding the optimal valve configurations u^*. The optimization was set to maximize the outlet temperature T, meaning the objective function J was set as $-T$ as the optimizer is, by default, implemented to solve minimization problems. Additionally, the y's were found using a root finder for the system given the initial parameters of the d's.

$$\begin{aligned} Q_j &= w_j \cdot \alpha_j \cdot (T_j - T_0) \\ Q_j &= UA_j \cdot \Delta T_{LMTD,j} \\ \Delta T_{LTMD,j} &= \left\{ (T_j - T_0) \cdot (T_{h,j} - T_{h,j}e) \cdot \frac{(T_j - T_0) + (T_{h,j} - T_{h,j}e)}{2} \right\}^{\frac{1}{3}} \\ Q_j &= wh_j \cdot (T_{h,j} - T_{h,j}e) \quad \forall \quad j = 1,2,3 \end{aligned} \tag{4.8}$$

5 Methodology

Since the problem involved machine learning, it should come to no surprise that the first step is to train the machine learning models (ML models). This step is called the training step, and requires the biggest time investment for the overall prediction process. The flipside being that once these ML models are trained, they do not have to be re-trained before repeated usage. The process of initializing these models was found to be very quick and part of the main bottleneck for the predictions is getting the correct measurements from the CasADi optimization. As the size of the disturbance-set grew however, it was quickly seen that prediction using GPs for multiple inputs and outputs, the time needed for training grew drastically. For further elaboration on this, the reader is referred to the discussion in Section 7. The CasADi optimization employed to find optimal valve-predictions used the software package Ipopt, developed by Wächter and Biegler[41].

5.1 Dataset generation

All of these models were trained with randomly generated data using the code in Appendix B.1. This function takes the desired amount of datapoints to be generated (1200 in the case of training) and generates a random set of data, which includes all disturbances, measurements and valve openings. The set of disturbances and valve openings were uniformly distributed inside a given interval. The only parameters with a interval length $\neq 0$ were the disturbances wh_{1-3} and UA_{1-3}, which had individual ”reasonable” disturbance intervals set to each parameter. This function also has the ability to take in a variable dubbed ”ttratio”, from test-training ratio. This parameter proportionally increases the size of the intervals from which data is uniformly drawn, and thus generating a dataset from a larger ”hypercube” that we are working with initially. The term ”nominal hypercube” in this paper, refers to the cases where the ttratio is set as 1.0. Thus, testing values on the nominal hypercube refers to testing within the same range the models were trained for. The ttratio was later used when testing the models' ability to predict for datapoints outside of its training range to see how well it would hold up to possible extreme disturbances, which can be seen under Section 6.2.

Additionally it is important to note that all the data used for training the ML models were normalized before training. This accounts for both the input variables and the output variables. The values were later readjusted after prediction to show their actual value. The scaling was done using Scikit-learn's package for scaling, to which all work is attributed to Pedregosa et al.[42]. Normalization of training data is a pre-processing step frequently done in machine learning as it helps neutralize effects of data scaling. These undesired effects are often caused by variables having different units or representing different physical entities. By normalizing all the data, the ML is able to focus its optimization more on the relationship between the input-output rather than how the inputs relate to each other.[43] Additionally, working with normalized data has been shown to give more effective kernel performance for high-dimensional models as seen in Schölkopf et al.[44].

In order to find the optimal valve parameters for the system, CasADi's optimization package was used on Equations 4.6 - 4.8. These equations formed a non-linear problem, which could be optimized using the outlet temperature, T, as the objective function. In doing so, the optimal valve configurations were found, and these were used to both train the ML models and when evaluating their performance later on. The optimization was set up following the model derivations shown in Section 4.2.

5.2 Machine learning model generation and evaluation

For the machine training step, a total of 3 separate ML models were trained for a set of 1200 training points each. In order to be able to distinguish these machines from one another, they were dubbed accordingly to their input-output configuration. **YD** is the part of the process that takes the measurements, y, and uses these to predict the disturbances d. **DU** it the proceeding ML model that was intended to take the predicted disturbances and predict a set of valve configurations u^*, which would give the optimal outlet temperature T. It is important to note here, that **DU** was trained using a generated set of disturbances d and not using the predicted set from the **YD** ML model. The final ML model is a **YU** model, which takes the measurements, y, and predicts the optimal valve configurations u^* directly. This final model was trained in order to be able to compare the two approached $y \rightarrow d \rightarrow u^*$, which uses **YD** and **DU**, and $y \rightarrow u^*$ which only uses the **YU** model. All the parameters used under y, d and u can be seen in Equation 4.1 and in the setup of the three ML models, which is illustrated below in Figure 5.1.

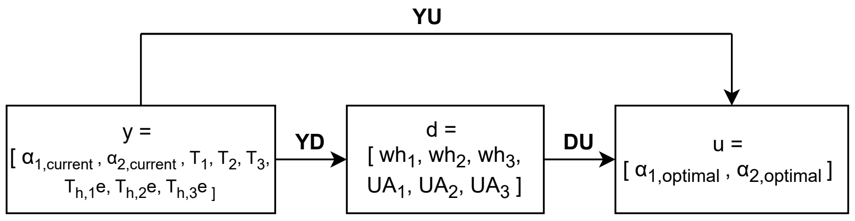

YU
y = [$\alpha_{1,current}$, $\alpha_{2,current}$, T_1, T_2, T_3, $T_{h,1}e$, $T_{h,2}e$, $T_{h,3}e$]
YD
d = [wh_1, wh_2, wh_3, UA_1, UA_2, UA_3]
DU
u = [$\alpha_{1,optimal}$, $\alpha_{2,optimal}$]

Figure 5.1: Figure showing a block diagram of the ML structure and which parameters are used. The goal is to find a suitable set of values for the prediction u^* by using the **YU** model for the $y \rightarrow u^*$ approach, and **YD** combined with **DU** for the $y \rightarrow d \rightarrow u^*$ approach. The diagram was created using a diagram software developed by Benson[6].

Evaluation of the code was done by generating a new set of 1000 datapoints. 500 of the generated datapoints had a ttratio of 1.0, meaning that we test for values in the range that the ML had been trained for, but not the exact same points it is trained for. The latter 500 points were generated using a ttratio of 1.3, which was to test the ML model's ability to predict outside of its training region. The results for the datapoints inside the training region is included under Section 6.1, while the analysis done outside the training region is included under Section 6.2.

5.3 Prior adjustment

Following the $y \rightarrow d \rightarrow u^*$ and $y \rightarrow u^*$ approaches, a separate analysis was performed by adjusting the priors of the ML models. This can be done by first making more simplistic regression predictions for some of the values, and subtracting the prediction values from the training values. This method is illustrated below in Equations 5.1. In these equations, y_{pr} refers to the predicted value from the simple regression, x_{tr} and y_{tr} refers to the original training dataset for x and y, y^*_{tr} refers to the adjusted training value for y, and y_{GP} refers to the final predicted

y-value from the ML models.

$$\begin{aligned} y_{pr} &= f_{\text{simple regression}}(x_{tr}, y_{tr}) \\ y^*_{tr} &= y_{tr} - y_{pr}(x_{tr}) \\ y_{GP} &= \mathbf{GP}(x_{tr}, y^*_{tr}) \end{aligned} \tag{5.1}$$

For this experiment, two prior types other that the default zero prior were tested, namely a linear prior and a quadratic prior. From this, we are able to compare how selecting different priors affect the performances of the ML models. Typically, we could expect that a linear prior would be optimal for these kinds of predictions as a GP will always return to their prior for results they have not been trained for. By enforcing a linear *trend* for the training values, we can have the predictions that land outside the prediction hypercube, land somewhat closer to their desired value compared to them approaching zero. As quadratic fits are typically used as priors for cost functions and the like, it is hypothesized that using a linear prior will produce a better fit for this process model.

In doing adjustments like this, it it typically known that higher degrees of polynomials lead to overfitting of the training data, which is why no higher order polynomial was tested than a quadratic one.[3] In this sense, we expect the linear fit to predict slightly better than the quadratic, as quadratic priors are typically better suited towards optimization problems where the predictions are focused towards some sort of objective function, and there might not be enough complexity in the data trend to justify a quadratic fit.

5.4 GPy

GPy is a Gaussian process framework that is written in Python. The framework was made by the Sheffield machine learning group and published in 2012 on the popular coding website GitHub.[45] The framework is one of many modern data-driven types of programming approaches of which machine learning falls under.[46] The package covers a wide range of popular machine learning algorithms that are based on GPs, and is the main machine learning tool that is employed in this paper.

The models were built using a standard radial basis function kernel (often referred to as a RBF kernel or SEK). This function is defined as shown in Equation 3.5, and further explored under Section 3.2. This kernel was combined with a coregnionalized regression model by using a Intrinsic Coregionalization Model (ICM), which is shown in Equation 3.7.

5.5 Algorithm Summary

To summarize, a set of datapoints was generated using a piece of code, shown in Appendix B.1, following a uniform distribution within a set region in the disturbance space, which then was used to train three separate ML models. Two of the aforementioned models, **YD** and **DU**, were meant to work together to predict a set of valve predictions, and a third model **YU** was developed for comparison. The goal of this comparison is to evaluate whether a grey-box approach using a set of intermediate prediction of d's (see Equation 4.1) would produce any beneficial results. Both of the approaches, $y \rightarrow d \rightarrow u^*$ and $y \rightarrow u^*$ were tested both inside and outside their training region to measure how well they perform when predicting for unfamiliar data. The approaches were also tested using a set of three different prior adjustments to analyze how any prior knowledge of the system affects the final outlet temperature.

6 Results

All results included are tested for randomly generated data within a set variance. The data will first be presented when using a ttratio of 1.0 in Section 6.1.1. This will be to focus mainly on the differences in the approaches between $y \rightarrow d \rightarrow u^*$ and $y \rightarrow u^*$, from here on dubbed **YDU** and **YU** approaches. As mentioned in Section 5, the ML models were tested on 500 training points for both ttratio's of 1.0 and 1.3.

The first set of results included will compare the predicted optimal values for u to the analytically calculated optimal ones. These will be illustrated in separate scatters where a condensed linear trend of the scatter clouds suggests better predictions. Following this, histograms for both approaches will be included to evaluate how the valve predictions perform to the overall plant optimization. The histograms will show how much each final outlet temperature T deviates from the maximum attainable temperature. The amount of deviation will be plotted versus the frequency of said deviation and it is used show the accuracy of the predictions. Naturally, this entails that better predictions lead to more frequent cases close to 0°C deviation.

As a final step of the analysis, the three priors were compared using boxplots. These plots included the overall temperature loss that all the various priors achieved from their valve predictions. Since ML has a slight tendency for some extreme predictions, this can lead to a fair share of outliers which makes the overall visual presentation messy. To compensate for this and make sure the plots were visually clear, the whiskers were set to cover the 98^{th} percentile of the data.

From all these plots, the three separate prior adjustments will be analysed to see how these affect the final results for both the YDU and the YU approach. The differences between the priors are explained computationally in Section 5.3 and theoretically under Section 3.1, but in short three regression approaches are applied to the training data before the ML model is trained. These three are namely no prior regression, linear regression and quadratic regression.

Alongside this section, some general performances differences will be brought up between all the approaches. The goal from all of this is to be able to see what prior adjustment works best and to be able to determine the value of a grey-box approach (YDU) compared to a direct black-box approach (YU). As this is more nuanced than declaring a winner and a loser, the pros and cons of each approach will be evaluated.

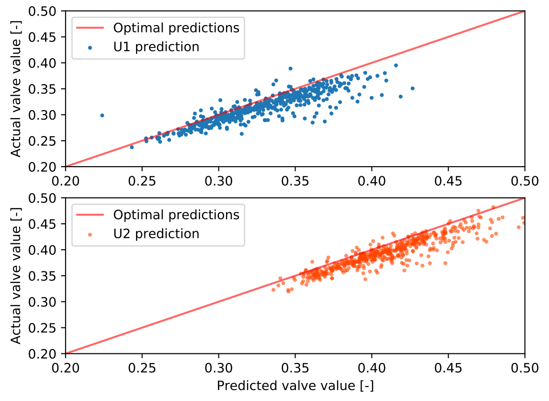
6.1 Testing inside the training region

First off, an extensive analysis was be performed for the nominal hypercube, where we test with a set ttratio of 1.0. This entails that the ML models are being tested for the same kind of scenarios they were trained for, but not the exact same data points.

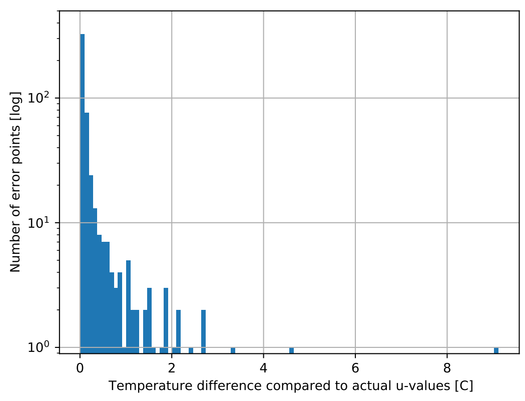
6.1.1 Nominal hypercube using YDU approach

The results obtained when using a nominal hypercube with the YDU approach are illustrated in the following section. As the models are being tested in the same region they were trained for, we expect somewhat good results from this if the model is to hold any value.

The obtained valve predictions for YDU using the **default prior**, are illustrated below in Figures 6.1.



(a) Figure showing the scatter plot of all the predicted values of u compared to their actual optimal prediction value, when using no prior adjustment.



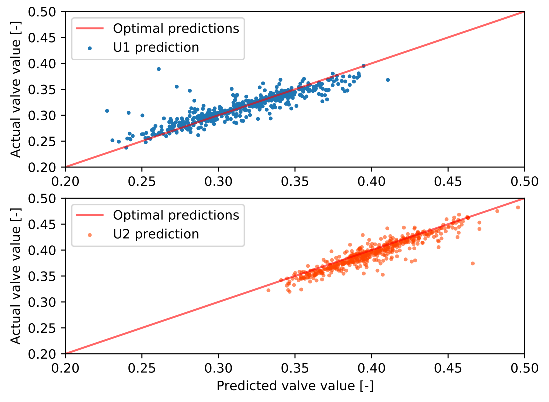
(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.1a.

Figure 6.1: Figures illustrating the prediction efficacy of the YDU approach with no prior adjustment, i.e. **default prior**.

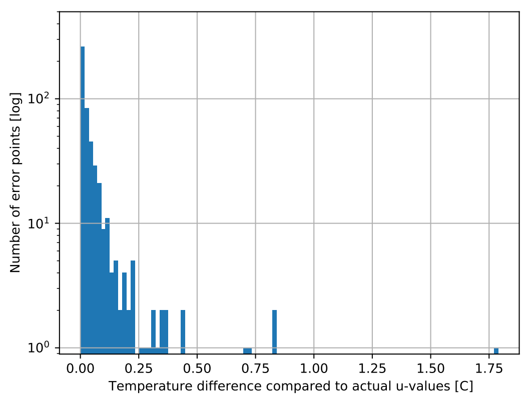
From the scatter plots we can see that most of the predictions land close to their optimal, with some slight deviations in the predictions. This is especially true for the greater values of u^* as they seem to lean greatly under their optimal prediction. We also notice that the general predictions for both u_1 and u_2 are slightly lower than that of their optimal prediction.

This is somewhat reflected in the final temperature results, which can be seen in Figure 6.1b, where we see that there are some deviancies (from optimally achievable temperature) as great as 9°C. Most of the resulting temperatures do however seem to range from [0, 2]°C range, which is reasonable given the lack of prior knowledge of the model.

The obtained valve predictions for YDU using a **linear prior**, are illustrated below in Figures 6.2.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a linear prior.



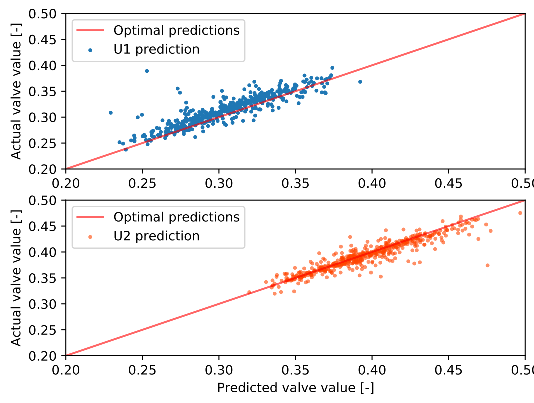
(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.2a.

Figure 6.2: Figures illustrating the obtained valve predictions u^* and their resulting outlet temperatures, when using a **linear prior** for the YDU approach.

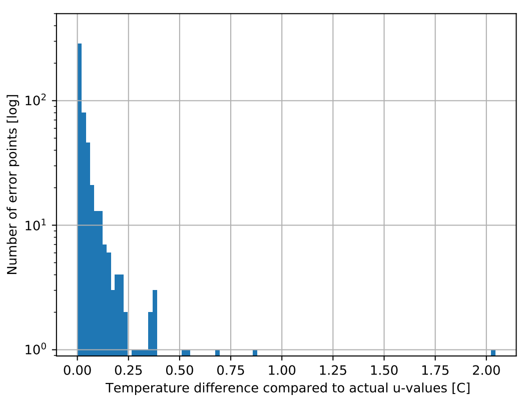
From this figure we can see that the results seem to be overall closer to their overall predictions compared to using the default prior (shown in Figure 6.1a). This is especially true for the predictions for u_1 as they seem to be shifted in a manner that makes the overall scatter cloud seem better aligned with the line for optimal predictions. We do however still see a trend for some uncertainty in the predictions with a fair amount of outliers scattered about, but as we can see in Figure 6.2b, the overall resulting outlet temperatures seem to be much closer to 0°C compared to those in 6.1b.

The overall temperature deviancies in Figure 6.2b seem to mainly lie in the range of [0, 0.2]°C, which is a big improvement over the default prior approach (Figure6.1b). We do however see that the general scatter cloud for u_2 still seem to be having a trend shifted slightly below the line for optimal predictions, which indicated a somewhat still poor fit for the data.

The obtained valve predictions when using a **quadratic prior** are illustrated below in Figures 6.3. For this approach, we initially expected the results to possibly be slightly overfitted and thus performing slightly worse than the linear prior.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value.



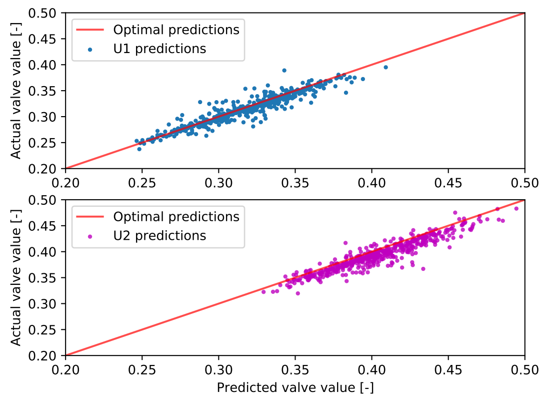
(b) Figure showing the resulting histogram when using the valve predictions shown in Figure 6.3a.

Figure 6.3: Figures showing the resulting data when using a **quadratic prior** for the YDU approach.

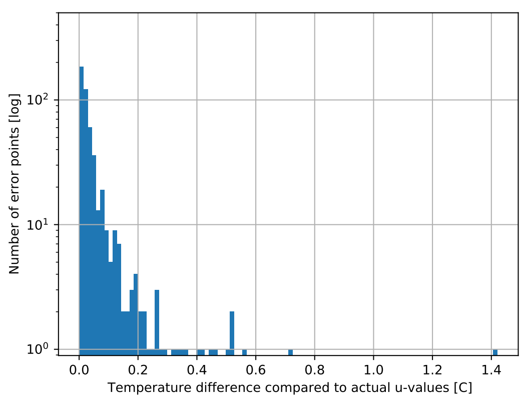
From these figures we can see that the trained model predicts values of u_1 that are more often greater than their optimal values, as the scatter cloud seems shifted above the optimal predictions line. The predictions for u_2, however, seem to be fitted fairly accurately around the line showing the optimal predictions. From the histogram in Figure 6.3b we can also see that the overall performance of the graph seems to be fairly similar to the linear prior, with most temperature deviancies laying in the range of [0, 0.2]°C. But as visual evaluation of the histograms alone can be difficult, this will be further evaluated in Section 6.1.3, where we utilize boxplots for more readily comparison of the performance of the two methods.

6.1.2 Nominal hypercube using YU approach

The results obtained when using the YU approach (explained under Section 5) within the training region are illustrated in this section. The data is presented in the same order as the previous section, meaning that the first set of data contains the obtained results when using a **default prior**. This is shown below in Figures 6.4.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using no prior adjustment.

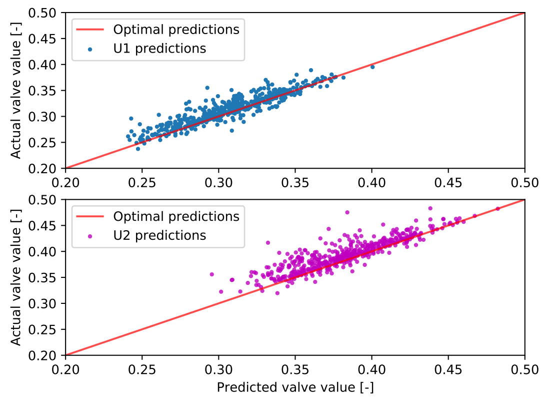


(b) Figure showing the resulting histogram when using the valve predictions shown in Figure 6.4a.

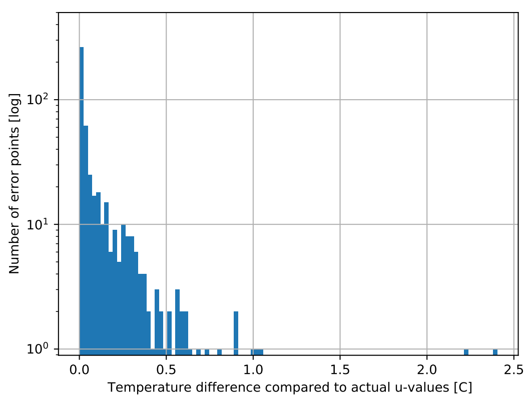
Figure 6.4: Figures showing the resulting data when using a **default prior** for the YU approach.

From these graphs, we can see that the valve prediction scatter results seem to be more focused around their optimal values compared to the default prior YDU approach (Figure 6.1a). While there are some outliers, these are not as extreme as the ones for the YDU approach. This naturally leads to a better outlet temperature, which can be seen in the histograms presented in Figures 6.1b and 6.4b for the YDU and YU approach respectively. We can see that not only is the maximum temperature difference greatly different (from 9°C to 1.4°C), but the overall distribution also varies greatly. The main distribution for the YU approach seems to be concentrated around the [0, 0.15]°C range, compared to YDU's [0, 0.2]°C range (for both the linear and quadratic prior).

Next up was the analysis of using a **linear prior** for the YU approach. The results from this is illustrated below in Figures 6.5.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a **linear prior**.



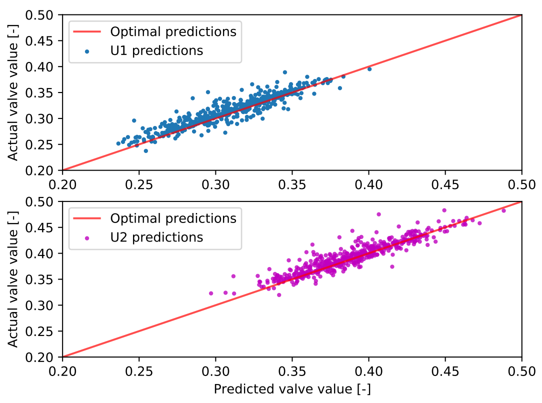
(b) Figure showing the resulting histogram when using the valve predictions shown in Figure 6.5a.

Figure 6.5: Figures showing the resulting data when using a **linear prior** for the YU approach.

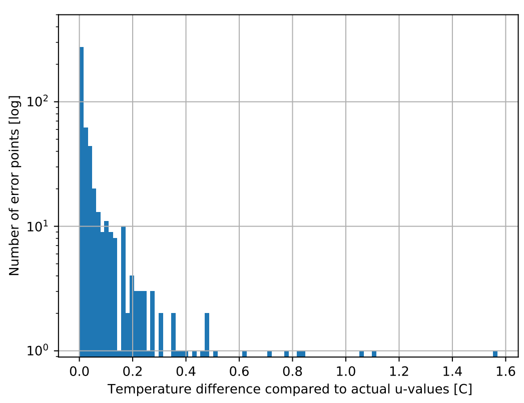
Contrary to the expectations stated under Section 5.3, we can see from Figures 6.5 that the overall YU predictions for a linear prior performs worse than when using no prior adjustment. By a quick glance at the u-prediction scatter plots (in Figures 6.4a and 6.5a) we can see that the results are overall less concentrated around their optimal values for the linear prior. This combined with the fact that the overall scatter cloud seems to be shifted slightly above the optimal prediction line makes the overall results for the linear prior somewhat poor.

In the YU approach, we already have six variables used to predict two, which already leads to a quite exceptional fit as can be seen in the histogram for the default prior (Figure 6.4b). When applying a linear trend onto the prior, we thus add an extra layer of inaccurate information that seems to lower the overall performance. Unlike the default prior, where we see that the results are even so good that they seem to rival the quadratic prior adjustment for the YDU approach (seen when comparing the histogram Figures 6.3b and 6.4b). This will be evaluated further under Section 6.1.3, but since the linear prior already seems to clutter the YU predictions - we can already somewhat expect the quadratic prior to overfit as well and give somewhat poor results.

The obtained predictions and their resulting outlet temperatures when using a **quadratic prior** are plotted below in Figures 6.6.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a **quadratic prior**.



(b) Figure showing the resulting histogram when using the valve predictions shown in Figure 6.6a.

Figure 6.6: Figures showing the resulting data when using a **quadratic prior** for the YU approach.

It is not directly clear, from looking at the u-predictions for the default and quadratic prior (Figures 6.4a and 6.6a), which one of the two performs better as both clouds seem fairly scattered. At a closer glance however, it seems to be a trend for the predictions that the data is more concentrated around the true optimal values. This observation is verified when looking at the resulting outlet temperatures as the overall predictions seem way more concentrated around the [0, 0.15]°C range, with fewer outliers (seen from Figures 6.4b and 6.6b).

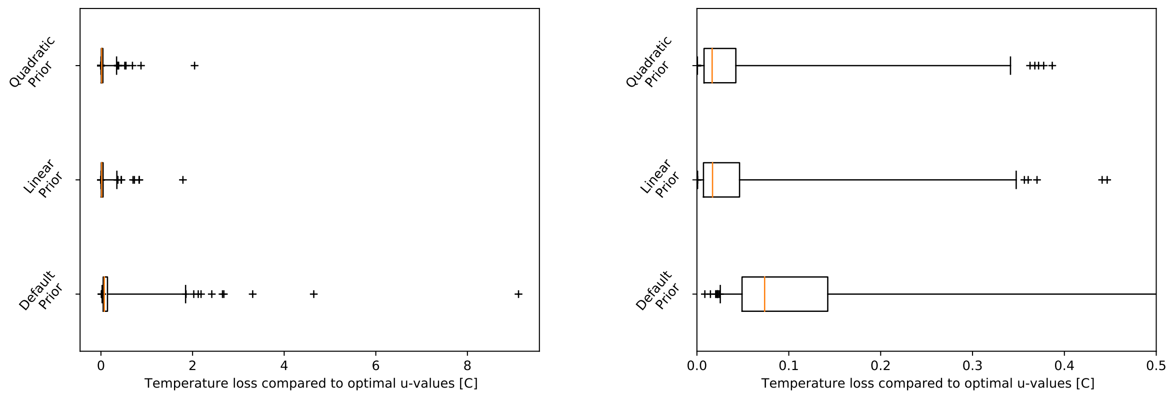
When comparing the performance of the quadratic prior and the linear prior however, we can see that the scatter cloud seems to be concentrated way better around the optimal prediction line (seen from Figures 6.5a and 6.6a). This is especially noticeable for the predictions for u_2. On the contrary, for the predictions of u_1, the overall predictions seem fairly similar and no major differences are noted.

When looking at the histograms (Figures 6.5b and 6.6b), we can see that the overall performance seems to strongly favor the quadratic prior, as the temperature deviancy range is almost half to that of the linear prior (from [0, 0.5]°C to [0, 0.2]°C. The overall amount of outliers is also greatly decreased and seem much closer to the 0°C.

6.1.3 Boxplot evaluation of the approaches

In order to compare the different approaches, it was decided to construct a boxplot from the data in the histograms as this would give best visual representation of how the performances of the different approaches. The boxplots were constructed to find out how the performance of the various priors vary, in order to better be able to pick a winner among the three. All three prior adjustment methods were also evaluated outside the training region to see if some held their performance better than others. This is evaluated in Section 6.2.3.

The first boxplot shows the combined results of the histograms for the YDU approach (shown in Figures 6.1b, 6.2b and 6.3b) and is shown below in Figures 6.7.

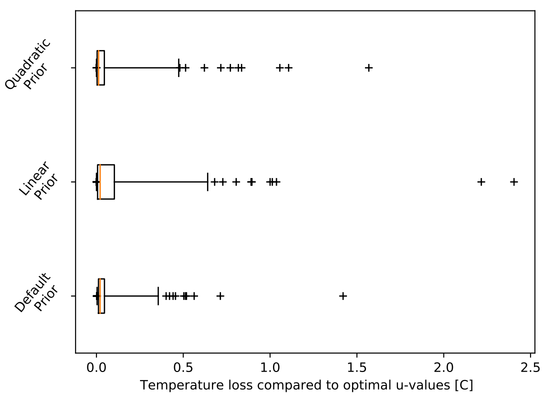


(a) Figure showing the obtained boxplot for the YDU approach. **(b)** Figure showing a zoomed rendition of the box plots for visual clarity.

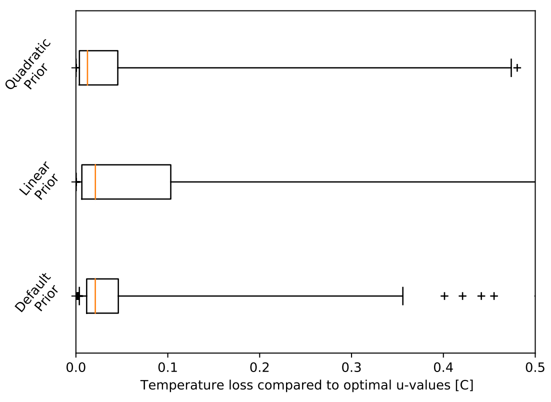
Figure 6.7: Figures showing the obtained boxplot for the various prior selections for the **YDU** approach. The box is set as the default interquartile range (IQR), the whiskers are set to 2_{nd} and 98^{th} and the outliers are marked with ”+” crosses. Both a normal and a zoomed version is included for visual clarity as some extreme outliers cause the full picture to become a bit hard to evaluate visually.

From the boxplots one can immediately tell that the default prior performs way worse in its overall resulting temperatures than those with the prior adjustment. Between the quadratic and the linear prior however, the trend of improvement seem to diminish drastically. The box and the whiskers only show a slightly more accurate trend for the quadratic prior compared to the linear prior. As this improvement seems relatively minor, the change is evaluated as insignificant. But we can still say that when using a grey-box approach, having some prior adjustment seems to be like the most optimal choice.

Moving on to the YU approach, the resulting boxplots from the histograms of the selected priors (Figures 6.4b, 6.5b and 6.6b) the following boxplots were created, illustrated below in Figures 6.8.



(a) Figure showing the obtained boxplot for the YU approach.



(b) Figure showing a zoomed rendition of the box plots for visual clarity.

Figure 6.8: Figures showing the obtained boxplot for the various prior selections for the **YU** approach. Both a normal and a zoomed version is included for visual clarity as some extreme outliers cause the full picture to become a bit hard to evaluate visually. The boxplots show the overall temperature loss, compared to analytically optimal values, resulting from the valve predictions.

Similarly to what was seen from the histograms for the YDU approach, it seems that the linear prior performs considerably poor for the YU approach. Unlike the YDU approach, it seems like there is a much closer call between the quadratic and default prior for the YU approach. While the overall 50% distribution (IQR) of the quadratic prior is closer to 0°C temperature loss, it also has a wider span of its whiskers. Either way it seems that both the quadratic prior and the default prior seems to outperform the linear prior for the YU approach.

As to why the linear prior performs so poorly compared to the quadratic prior adjustment (and even no prior adjustment), this will be further discussed and evaluated under Section 7.3.

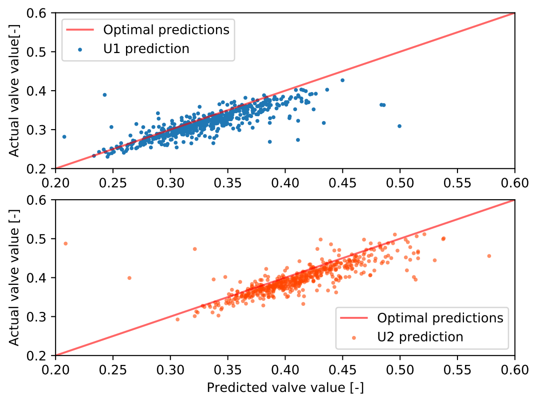
6.2 Testing outside the training region

From Figures 6.7 and 6.8, we can see that both the YDU approach and YU approach have two prior options that outperform the third. For the YDU approach, this was the linear and quadratic priors - while for the YU approach the quadratic and default priors seem to perform best. All priors will still be tested for both approaches to see how well the models can operate when working outside of their training region. This is investigated during this section, with results summarized under Section 6.2.3 in the boxplot Figures 6.15 and 6.16.

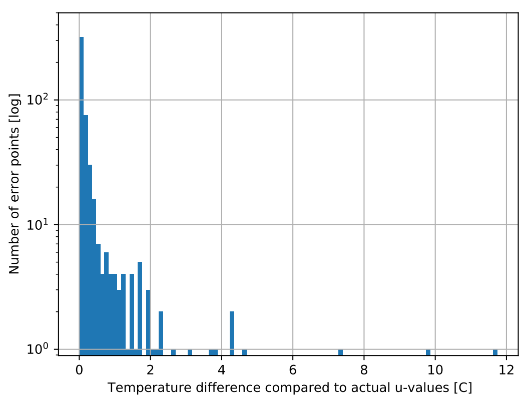
Since comparisons will be done sporadically with results from inside the training region, the reader is referred to Appendix A, where side-by-side graphs are attached for more readily comparisons.

6.2.1 YDU approach outside the training region

In a similar fashion to Section 6.1, the results in this part will be presented in the same order of *default prior*, *linear prior* and finally *quadratic prior*. The results for the **default prior** are illustrated in Figure 6.9.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a default prior.

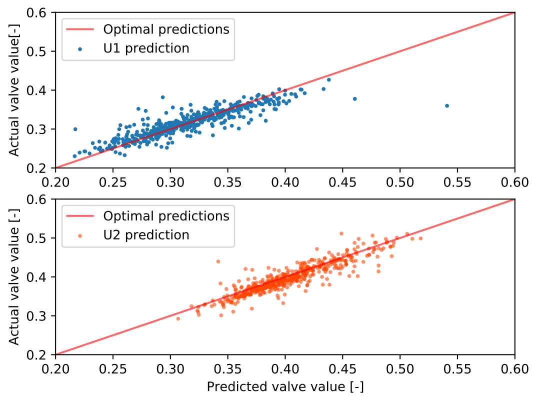


(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.9a.

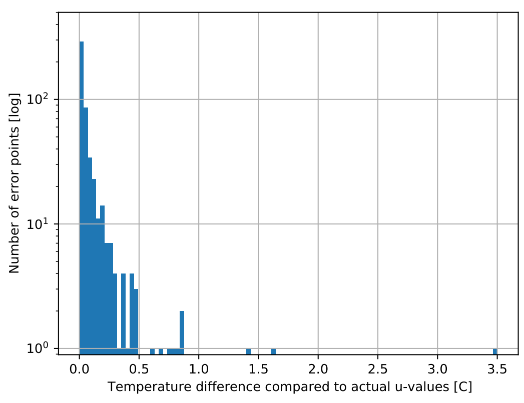
Figure 6.9: Figures showing the resulting data when using a **default prior** for the YDU approach.

From a quick glance at the figure for the valve predictions, in comparison to the nominal hypercube (Figure 6.1a), we can quickly see that the clouds are way more scattered for this case. These clouds also show way more frequent appearances of extreme outliers, where a much greater amount of points are separated from the main scatter cloud. This leads to more extreme max temperature discrepancies too, as the worst case scenario (as seen in Figure 6.9b) has outliers as far as 12°C. Aside from these few extremes, the overall prediction still seems to lie within the [0, 2]°C range however, so the model seem to be able to adapt fairly well to predicting outside its training region.

Next up is the **linear prior** approach, which has its results illustrated below in Figure 6.10.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a default prior.

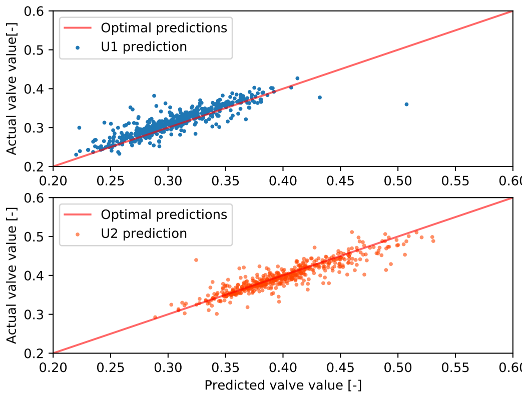


(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.10a.

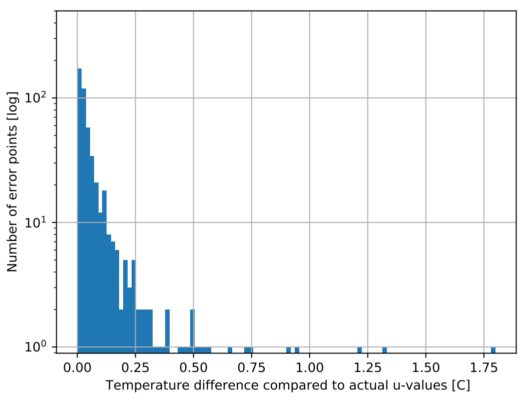
Figure 6.10: Figures showing the resulting data when using a **linear prior** for the YDU approach, with a ttratio of 1.3.

Similarly to the case for the default prior, we see that the scatter cloud for the u-predictions in Figure 6.10a is generally less concentrated than when testing inside the training region. The linear prior does however not have as many extreme outliers as the case for default prior. This is also reflected in the histogram in Figure 6.10b, where we see that even though the general prediction range now includes more cases for predictions in the [0.25, 0.5]°C range, these are overall not considerably frequent. The extrapolation capability of the linear prior is therefore deemed as good, and even better than the case for default prior. This can be said as for 500 prediction points, only a single point exceeded the maximum extreme found from the analysis inside the training region, which was found to be 3.5°C.

Finally there is the results obtained when using a **quadratic prior**, are shown below in Figures 6.11.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a quadratic prior.



(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.11a.

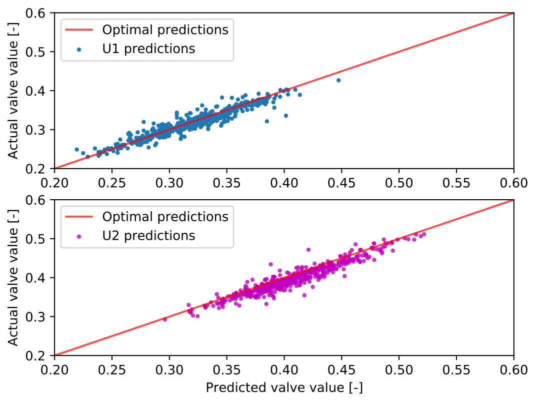
Figure 6.11: Figures showing the resulting data when using a **quadratic prior** for the YDU approach, with a ttratio of 1.3.

Similarly to the previous two cases, the scatter cloud for the u-prediction is more sparse when comparing with the results from training inside the training region (Figure 6.3). This is especially apparent for the higher end of the u_2 predictions, and the lower end of the u_1 predictions. Again, like in the previous cases the overall temperature difference histogram does not seem to be too affected by this, as the main temperature prediction lie in the same range of [0, 0.2]°C as the nominal ttratio. This prediction is fairly similar to the predictions for the linear approach (Figure 6.10a), but the different maximum outlier masks it in the graph. Further comparisons are made under Section 6.2.3 using boxplots.

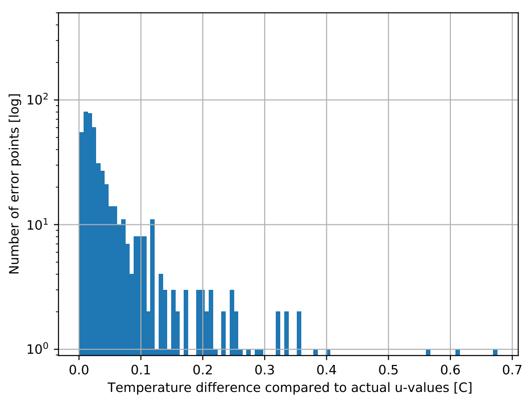
One relatively important observation that needs to be stated for the YDU approach however, is that regardless of how seemingly scattered the u-predictions are, the overall final temperature results are not all that much affected by it. It would seem that the loose points in the u_2 scatter cloud seems to be paired with a ”good match” from the u_1 cloud (and vice versa), which overall gives a decent final temperature. ”Good match” here meaning any pairing of the two valve predictions that result in a final outlet temperature close to the maximum possible attainable temperature. This will be further discussed under Section 7.

6.2.2 YU approach outside the training region

When testing outside the training region for the YU approach, it was found a lot of similarities with how the YDU approach reacted to the new testing environment. Still, the two ML models seemed to have slightly varying ways of adapting to the new data, which we will see in the following section. The first set of results show the data obtained for the **default prior**, and is illustrated below in Figures 6.12a.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a default prior.

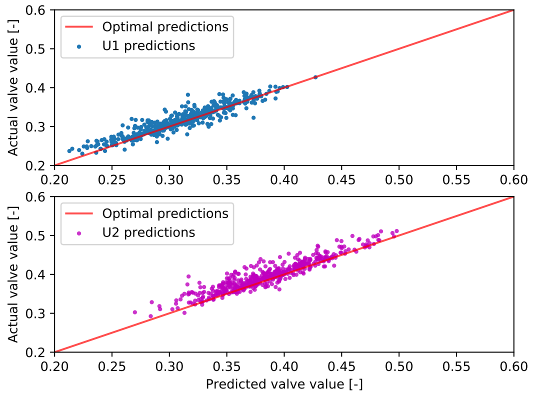


(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.12a.

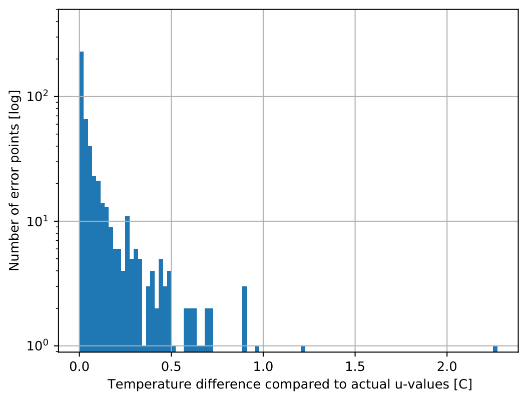
Figure 6.12: Figures showing the resulting data when using a **default prior** for the YU approach, with a ttratio of 1.3.

When evaluating the effect of the increased ttratio on the YU approach, it is interesting to compare the adjustment of the YU approach to the YDU approach. This can be valuable as we get to see how the different approaches adapt to the new testing environment. Thus, Figures 6.12 are comparable with the figures for a nominal hypercube (using the YU approach), shown in Figures 6.4, and with the nominal and expanded hypercube for the YDU approach shown in Figures 6.1 and 6.9 respectively. As aforementioned, the reader is referred to Appendix A as all these plots are attached side-by-side for more readily visual comparison between them. When comparing these, we can see that the scatter cloud for the YU approach seems to be way less affected by the increased testing region. The plots are concentrated far better around their desired position than that of the YDU approach. Still, we see that the overall scatter cloud for the u_2 predictions are shifted slightly below the optimal prediction line, albeit not as extreme as the YDU approach with a default prior (Figure 6.9). This does at first glance not seem to be very well reflected in the histograms as most predictions result in temperatures ranging from [0, 0.15]°C, but again it is important to note how visually deceptive these graphs can appear as due to the variations in the x-axis. Comparisons on that aspect will therefore be more focused on under Section 6.2.3.

Next up is the plots for the results when using a **linear prior**. These are shown below in Figures 6.13.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a linear prior.

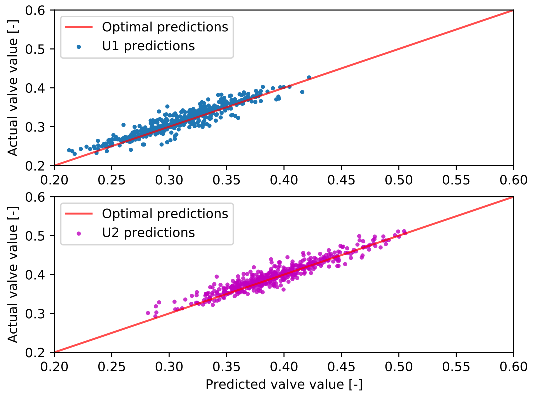


(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.13a.

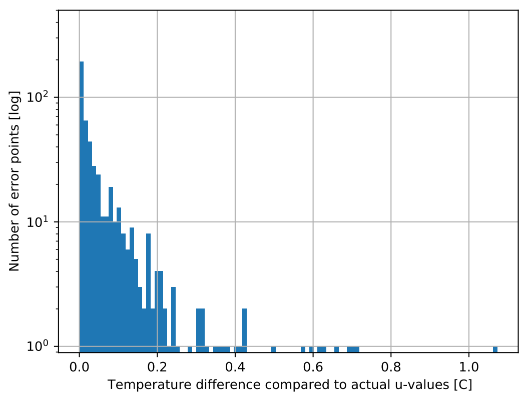
Figure 6.13: Figures showing the resulting data when using a **linear prior** for the YU approach, with a ttratio of 1.3.

Similarly to the case for the default prior, the scatter cloud for the linear prior seems to overall be less scattered when comparing with the YDU approach using a linear prior, for both nominal and extended hypercubes (in Figures 6.2a and 6.10a respectively). This is less apparent for the linear prior for the YU approach, but as discussed under Section 6.1.3, the linear prior seems to be the worst fit of the three priors for this approach. When evaluating the final resulting temperatures (Figures 6.5b and 6.13b), not much change is observed and the overall accuracy of the model seems to remain the same with both histograms having a prediction range of [0, 0.5]°C for most of their points.

Finally, is the plots for the results when using a **quadratic prior**. These are shown below in Figures 6.14.



(a) Figure showing the scatter plot of all the predicted u-values to their actual optimal prediction value, when using a default prior.



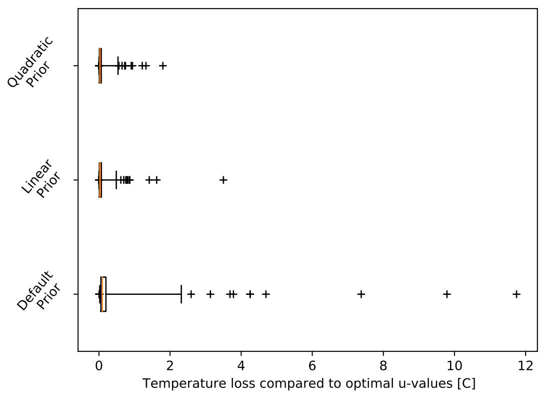
(b) Graph showing the frequency of various temperature deviations that resulted from the valve predictions in 6.14a.

Figure 6.14: Figures showing the resulting data when using a **quadratic prior** for the YU approach, with a ttratio of 1.3.

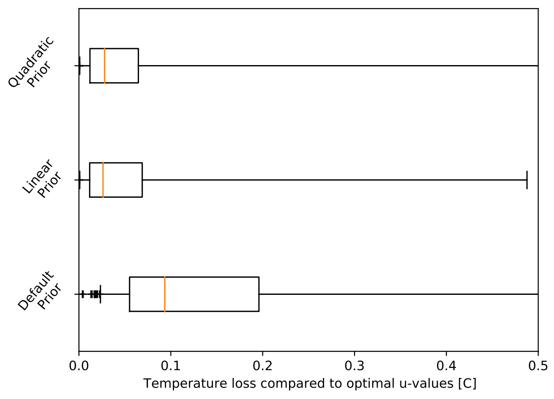
When evaluating the scatter cloud of the YU predictions using a quadratic prior, we see a similar behaviour to the default prior (Figure 6.12). From Figure 6.14a, we can see that the scatter cloud is slightly more scattered compared to the nominal hypercube (Figure 6.6a). The effect is less apparent than for the case with the YDU approach (Figure 6.3a). Nonetheless, the overall accuracy of the temperature prediction does not seem to be too affected by this. From a quick glance at the results from the nominal hypercube (Figure 6.6b) we can see that the main prediction range seems to remain unchanged for this case (Figure 6.14b). The main distribution however is shifted more towards 0.2°C, which is more apparent in the following comparison boxplots in Figure 6.16.

6.2.3 Boxplot evaluation of the approaches

The final step of the evaluation is that of the boxplot analysis. In this section the boxplots of both the YDU approach and YU approach are included to see the performance variations between the nominal hypercube (Section 6.1.3) and this case, using an expanded hypercube. These results can be seen below in Figures 6.15 and 6.16.



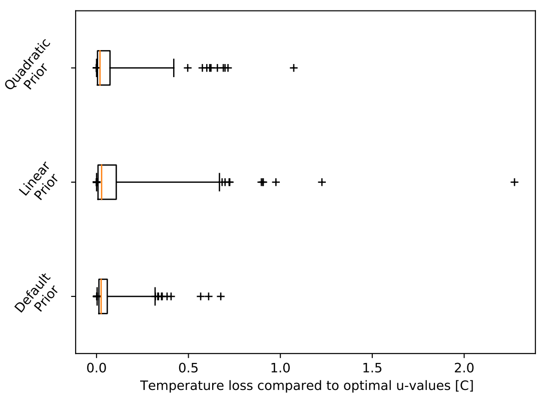
(a) Figure showing the obtained boxplot for the YDU approach.



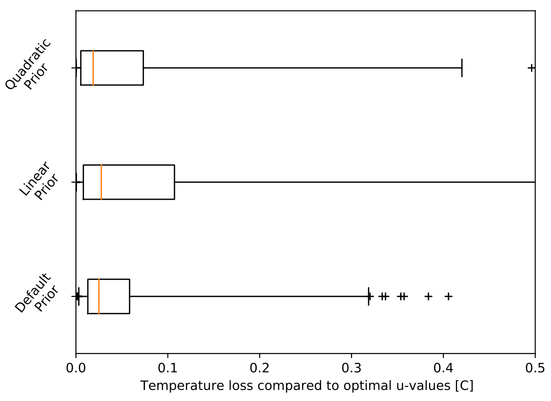
(b) Figure showing a zoomed rendition of the box plots for visual clarity.

Figure 6.15: Figures showing the obtained boxplot for the various prior selections for the **YDU** approach. Both a normal and a zoomed version is included for visual clarity as some extreme outliers cause the full picture to become a bit hard to evaluate visually.

When comparing this boxplot to the same approach over the training region (Figure 6.7), we see that the boxes are stretched a bit, as expected. Similarly to the case with a nominal hypercube, testing over a region greater than the training set does not seem to show any significant difference between the linear and the quadratic prior. And when using a ttratio of 1.3 it is deemed that both of these methods work equally effective at predicting values that give adequate final outlet temperature. The YU approach was evaluated in a similar manner, and the results can be seen in Figures 6.16.



(a) Figure showing the obtained boxplot for the YU approach.



(b) Figure showing a zoomed rendition of the box plots for visual clarity.

Figure 6.16: Figures showing the obtained boxplot for the various prior selections for the **YU** approach. Both a normal and a zoomed version is included for visual clarity as some extreme outliers cause the full picture to become a bit hard to evaluate visually.

When comparing this boxplot to the analysis done within the training region (Figures 6.8) it is a bit more visually clear that the quadratic prior performs marginally better than the default prior. Even thought the whiskers are slightly shorter for the default prior, with a difference of approximately 0.1°C, the overall IQR box is slightly closer to 0°C for the quadratic prior. It is worth mentioning that the median of all the approaches is fairly close. This is especially noticeable for the default and the quadratic prior. But still, while the resulting temperature loss is more concentrated for default prior, the overall temperature loss distribution seems to slightly favor the quadratic prior. This would of course depend on the type of plant that is operated on and if the cost function is highly sensitive to small temperature loss or not.

7 Discussion

It becomes apparent when evaluating the u-predictions, that only a scatter plot of the predictions does not tell the entire story. There have been presented multiple cases where we can clearly see that an increased scatter cloud does not necessarily mean a direct correlation to worse predictions in terms of overall loss. This was especially apparent for the YDU case studies, where for all priors the scatter appeared as being far worse than the actual resulting temperature was. For the YU study, most of the predictions seemed to land fairly close to their optimal predictions, but with the overall temperature not necessarily being better for it. Regardless, it is interesting to see how the grey-box YDU model managed to pair seemingly off values of u_1 with seemingly off values of u_2, to create something that is close to the actual optimal.

The reasoning for the YDU approach being able to create better pairs of u_1 and u_2 is speculated to lie within the cost function, namely the outlet temperature. To evaluate this further we can analyze a Hessian of the input-parameters as well as a Taylor expansion of the cost function around the optimum. A Hessian is frequently used to evaluate the local curvature of some point in a multivariable function. The function is defined below in Equation 7.1, and is usually evaluated at some point $((x_i, x_j) = (x_i^*, x_j^*)$ to evaluate the sensitivity of the function f to disturbances in its variables.[47]

$$(\mathbf{H}_f)_{i,j} = \frac{\delta^2 f}{\delta x_i \delta x_j} \tag{7.1}$$

By taking a Taylor expansion of the cost function around the optimum, we obtain Equation 7.2 which is shown below.

$$\begin{gathered} J(u^* + \Delta u) = J(u^*) + J_u(u^*)\Delta u + \Delta u^T \mathbf{H}(u^*)\Delta u \\ J_u(u^*) = 0 \rightarrow J(u^* + \Delta u) - J(u^*) = \Delta u^T \mathbf{H}(u^*)\Delta u \end{gathered} \tag{7.2}$$

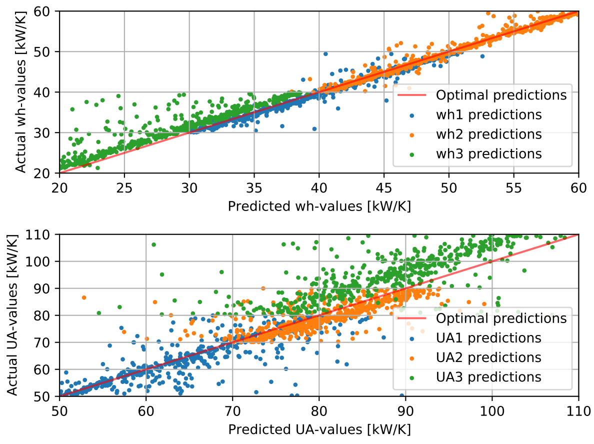
As the gradient is zero at the optimum, we can see that the Hessian carries all the information about the gradients at the optimum. These gradients will indicate the overall temperature loss we have for the cost function. As the Hessian is a matrix, we naturally know that some directions have higher gain, thus leading to a greater temperature loss. It is speculated that even though the overall scatter clouds for the YDU predictions look worse than the YU approach (seen in Figures 6.3a and 6.6a respectively), the YDU approach seems to have a better idea of the Hessian for the optimum. This allows it to pair better combinations of u_1 and u_2 that gives overall lower temperature loss (as seen in Figures 6.7 and 6.8).

As to why, not only the results, but mainly the scatters for YDU and YU look so different; this is hypothesized to lie in the nature of the models. The $y \rightarrow u^*$ approach is more direct, presenting six inputs and two outputs. This means the overall prediction of each point should land closer to the actual optimal values and thus give smaller scatter clouds. On the other hand, the YDU approach, uses the more comprehensive grey-box route of $y \rightarrow d \rightarrow u^*$. This can lead to the ML model somehow getting a better understanding of the system and how these parameters interact, and thus being able to predict values for u^* that, while ultimately incorrect and ultimately further from their actual optimal values, were able to create a different interpretation of the the system that had reasonable performance in terms of system optimization.

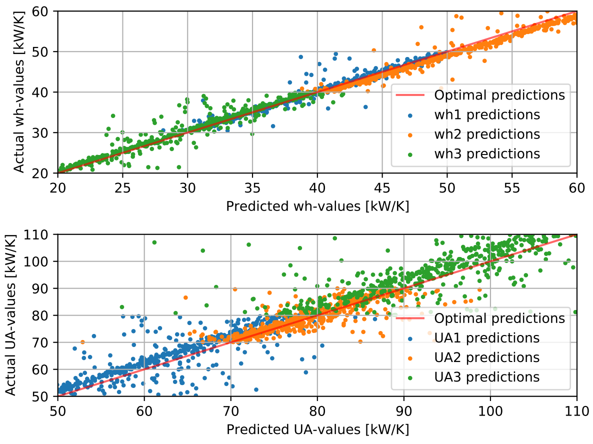
When considering all the results presented, both for testing in the nominal and in an expanded region, it becomes apparent that some approaches performed considerably worse than others. This is in regards to the default prior for the YDU approach and the linear prior for the YU approach. Both of these methods showed considerably greater uncertainty in their prediction and an overall bigger inaccuracy. For the other two cases, for both approaches, picking a single winner becomes a bit more ambiguous. For example, when evaluating the YU approach for a nominal and increased ttratio, we can see that while the default prior has an overall shorter range for its outer whiskers of about 0.1°C the IQR for the quadratic prior is slightly closer to 0. Thus the question becomes if the cost function for a process utilizing this HEN network is more sensitive to bigger temperature deviations (from optimal temperature) or if some deviations are acceptable as long as it means more situations overall run closer to optimal temperature. Similarly for the YDU approach, we can see that the performance of the methods is fairly similar. For the nominal hypercube, it can be seen from Figure 6.7b that the quadratic prior has a slightly smaller error in its predictions. For an increase in ttratio however (seen in Figure 6.15a), while the IQR for both priors are similar, the linear prior has a smaller prediction error range overall. Thus the question a plant engineer, considering to implement one of these approaches, would have to ask is whether we expect most operation cases to be within the training range or if more extreme cases are prone to show up. Regardless of the situation, it can be stated very confidently that a default prior for the YDU approach and a linear prior for the YU approach seems to be a poor choice.

7.1 Evaluating the intermediate values for the YDU approach

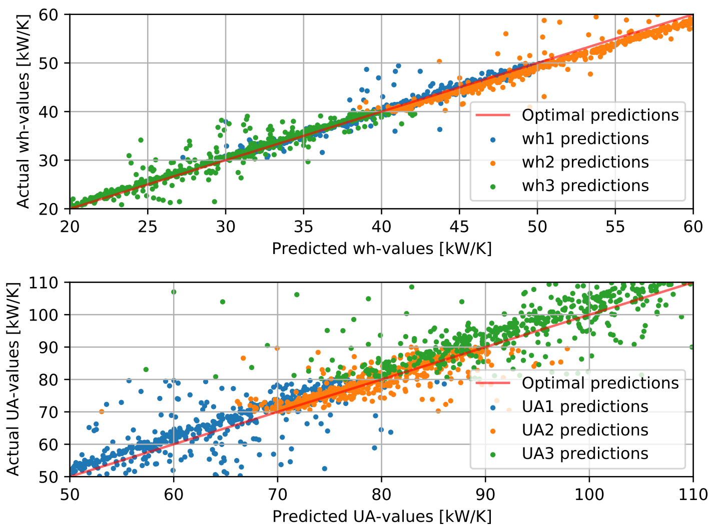
One of the main attractions of a grey-box model is the added possibility of troubleshooting for extreme points. This was evaluated by comparing the predicted disturbance values from the YD model and comparing them with actual analytical values. The results are illustrated below in Figure 7.1.



(a) Figure showing predicted against actual d-values, using a **default prior**.



(b) Figure showing predicted against actual d-values, using a **linear prior**.



(c) Figure showing predicted against actual d-values, using a **quadratic prior**.

Figure 7.1: Figures showing the predicted d-values against their actual analytical values from the YDU approach with the various priors

While the overall predictions for wh_{1-3} is fairly adequate, the predictions for UA_{1-3} is very chaotic. During evaluation of this method the predictions that gave the greatest temperature loss were manually labelled and scrutinized. From this analysis, it is was found that, while not immediately apparent from the values of u alone, the disturbance values for wh_{1-3} could prove very helpful in spotting predictions that have gone off course. In some of the extreme cases, one could immediately tell from the values for wh, that something was wrong as one of the values would either be considerably low or fairly high. This trend was, however, found to not be particularly consistent and it was mainly found that extreme values for UA_{1-3} were hard to spot.

When building up the machine learning model, it was found that N inputs to N outputs seemed to lose its value when approaching greater values for amount of output predictions. This was fairly well reflected in the YU approach as it was fairly decent at predicting the two u-values even when there were only two measurement inputs used for training. When constructing the YDU approach however simply giving training the ML model with six inputs (for six outputs) was not enough, as the overall performance of the model was exceptionally poor. It was quickly seen that the measurements did not carry the necessary information required to predict the disturbances, and so the current valve openings α_{1-2} were included in the model.

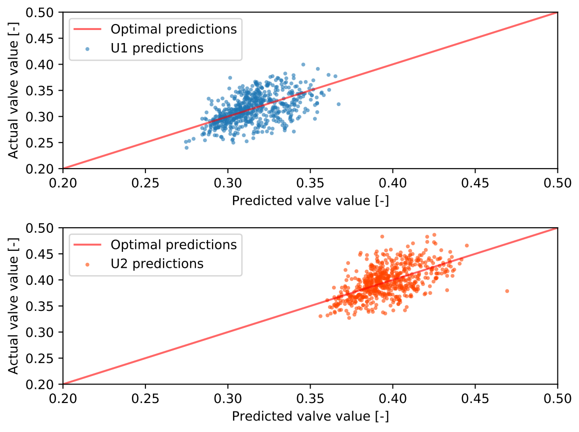
7.2 Training dataset size

When it comes to finding the ideal amount of training points for the ML models, no extensive testing was done, but during the early phases of the coding it was found that the scatter clouds were very loose if exceptionally few training points were being used (∼100). However, after a certain amount of training points (∼ 500) the amount of time required for the hyperparameter optimization rapidly increased, and the prediction results would remain somewhat the same. The drastic increase in training time was particularly noticeable for the **YD** model, as this model accounted for the greatest amount of variables. While a set of training points of 500 points *could* do equally well as a set of 1200, which was the considered amount in this paper, the training dataset size was set as big as reasonably possible to be sure that further training would not drastically change the observations made in this work.

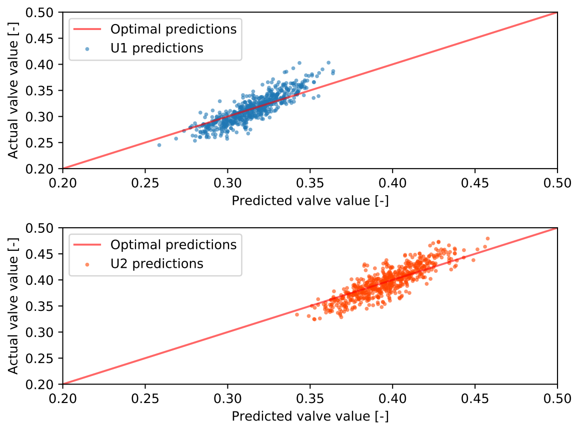
When using this amount of training points, the training was already fairly time-consuming and is easily determined as the most time-consuming part of the machine learning process. Once the models are trained, loading up their hyperparameters for prediction in a later run is very swift and efficient. Training the ML models for points above 1200-1500 would drastically increase the training time, and it was decided that setting a reasonable training size for all models were better than having part of the ML modules be trained far better than the YD model.

7.3 Performance of priors in YU approach

When evaluating the performance of the YU model, it is speculated that applying a linear prior only seemed to ”confuse” the model and hinder its ability to successfully comprehend the process using a GP. Since this was not the case for the case applying quadratic regression, this was evaluated further by looking at the performance for linear and quadratic regression alone. This was coded and tested, and the obtained results are shown below in Figures 7.2.



(a) Linear regression applied to the YU model.



(b) Quadratic regression applied to the YU model.

Figure 7.2: Evaluation of simple regression applied to the YU model. The figures show the valve predictions against their calculated analytical value in order to explain why using a linear prior performs so poorly for the YU model.

It was originally hypothesized that the reason the prior had a negative influence over the performance of the YU model was that the approach was already fairly direct, having only two prediction parameters using eight inputs. By applying any prior to such a approach, it seems like the prior's individual performance does seemingly need to be somewhat good on its own in order to not hamper the performance of the ML model. From the figure we can see that simple quadratic regression actually manages to produce a noticeable trend for the model. These predictions are by no means exceptional, but it does manage to display a trend in its predictions. This trend is fairly adequate compared to the predictions obtained from linear regression, in which the predictions are not well correlated with their true values. This can explain much of why we saw the performance of the linear prior being rather poor in its performance (seen in Figures 6.8 and 6.16 for nominal and extended testing regions).

7.4 Improving the model: Regression networks

In recent years there has been a rapid development of interest within the field of Gaussian process regression frameworks that accounts for fixed correlations between output variables (Byron et al.[48], Osborne et al.[49], Williams et al.[50], Alvarez and Lawrence[51], Alvarez and Lawrence[52]). One of the more promising models utilize a regression network, that is designed to account for correlation between outputs. In doing so, they created a multiple output model (often referred to as 'multi-task' models) that can achieve better prediction results. This could be especially applicable for a grey-box model like the YDU model, considered in this paper. By training a regression network like this, we can help the model better understand the correlations between the disturbance parameters and thus achieve better plant inputs to minimize temperature loss. Other methods include Gaussian process regression networks (GPRN). These models have been found to show "strong empirical performance" on several various datasets (Wilson et al.[53]).

7.5 Improving the model: Disturbance selection and ML formulations

As was seen from Figure 7.1, the grey-box evaluation had a rather poor performance for its disturbance predictions. The prediction efficacy for the heat capacity parameterers wh_{1-3} were fairly adequate, but the prediction for the overall heat transfer coefficients UA_{1-3} were exceptionally poor. It is speculated that when constructing a model like this, a more thorough evaluation of the measurement and disturbance parameter selection *could* have improved this. Additionally, when working with larger disturbance (or measurement) sets it can be an advantage to incorporate other function approximators over GPs. A possibility is the use of spline models, which have often been shown to work better for problems where the problem scale is too big for a GP to handle, and the models are in many instances, regarded as mathematically equivalent.[54] The scaling was already noticeable and slightly problematic when increasing the size of the training sets. For sets over $\sim$1500 training points, training the YD model took up to 8 hours to complete, even with a fairly high-end computer being used.

8 Conclusion

While more traditional black-box models have been the main go-to method for the use of machine learning in most studies, this paper made an attempt at the results of using a grey-box approach. Grey-box in this sense indicates that a set of intermediate process disturbance parameters were predicted and used to predict the optimal process inputs. This not only allows a operator to more readily investigate any peculiar predictions without having to fully break down the black-box model, but has been shown to also give room for the ML models to understand the process better and thus give *alternative* parameter predictions that lead to surprisingly adequate process optimization.

In terms of the simulations performed in this work, the grey-box model was found to display some shortcomings when predicting outside the dedicated training area, to which it was found that the traditional black-box model is superior (Figures 6.15 and 6.16 for YDU and YU approach respectively). In terms of predicting within the training region however, the grey-box model presented less overall temperature loss and thus slightly better predictions. On the other hand, the black-box model showed a higher amount of prediction points that yield zero (or close to zero) temperature loss. It can therefore be said that when opting between a grey-box and a black-box model for a plant considering a case study like this, one would have to evaluate the specific case in order to best be able to select a model.

From further evaluation of the grey-box model it was found that it does not necessarily respect the intermediate disturbance values. While the predictions for the disturbances wh_{1-3} (seen in Figure 7.1) were fairly accurate, the predictions for the parameters UA_{1-3} were not very indicative and useful for estimating when a input prediction had gone amiss. Despite this shortcoming, we can say that utilizing a grey-box model certainly has its fair share of benefits compared to a traditional black-box model, but more research into this field is required. One of the suggested methods by which one can achieve this is by using GPRN or other regression networks that can help in accounting for correlations between the output variables of the process. By doing this we can possibly design machine learning models that better understand the underlying structure of the system and is thus able to give better input predictions.

References

[1] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017. doi: 10.1016/j.enbuild.2017.11.045.

[2] Hamed Karimian, Qi Li, Chunlin Wu, Yanlin Qi, Yuqin Mo, Gong Chen, Xianfeng Zhang, and Sonali Sachdeva. Evaluation of different machine learning approaches to forecasting PM2.5 mass concentrations. *Aerosol and Air Quality Research*, 19(6):1400–1410, 2019. doi: 10.4209/aaqr.2018.12.0450. URL `https://doi.org/10.4209%2Faaqr.2018.12.0450`.

[3] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02): xiv–xv, 1–4, 7–20, 69–106, 112–116, 2004.

[4] Peter Roelants. Gaussian processes (3/3) - exploring kernels. `https://peterroelants.github.io/posts/gaussian-process-kernels/`, 2015.

[5] JJJ Chen. Logarithmic mean: Chen's approximation or explicit solution? *Computers & Chemical Engineering*, 120:1–3, 2019.

[6] David Benson. drawio: Diagram software. `https://github.com/jgraph/drawio-desktop/releases/tag/v14.6.13`, 2021.

[7] W Erwin Diewert, Paul Schreyer, SN Durlauf, and LE Blume. Capital measurement. *New Palgrave Dictionary of Economics*, 2008.

[8] B. Beavis and I. Dobbs. *Optimisation and Stability Theory for Economic Analysis*. Cambridge University Press, 1990. ISBN 9780521336055. URL `https://books.google.no/books?id=L7HMACFgnXMC`.

[9] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993. pp. 15-20.

[10] Timothy E. H. Allen. Chapter 1 computers as scientists. In *Machine Learning in Chemistry: The Impact of Artificial Intelligence*, pages 1–15. The Royal Society of Chemistry, 2020. ISBN 978-1-78801-789-3. doi: 10.1039/9781839160233-00001. URL `http://dx.doi.org/10.1039/9781839160233-00001`.

[11] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349 (6245):255–260, 2015.

[12] Bill Whiten. Model completion and validation using inversion of grey box models. In Scott McCue, Tim Moroney, Dann Mallet, and Judith Bunder, editors, *Proceedings of the 16th Biennial Computational Techniques and Applications Conference, CTAC-2012*, volume 54 of *ANZIAM J.*, pages C187–C199, May 2013. `http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/6125` [May 21, 2013].

[13] Anis Ben Abdessalem, Nikolaos Dervilis, David J. Wagg, and Keith Worden. Automatic kernel selection for gaussian processes regression with approximate bayesian computation and sequential monte carlo. *Frontiers in Built Environment*, 3:52, 2017. ISSN 2297-3362. doi: 10.3389/fbuil.2017.00052. URL `https://www.frontiersin.org/article/10.3389/fbuil.2017.00052`.

[14] Takeru Kono et al. Application of strategy switching mechanism with improved strategy for heat exchanger network design. In Mario R. Eden, Marianthi G. Ierapetritou, and Gavin P. Towler, editors, *13th International Symposium on Process Systems Engineering (PSE 2018)*, volume 44 of *Computer Aided Chemical Engineering*, pages 949–954. Elsevier, 2018. doi: https://doi.org/10.1016/B978-0-444-64241-7.50153-1. URL `https://www.sciencedirect.com/science/article/pii/B9780444642417501531`.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN 978-0-387-31073-2.

[16] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285, 2004.

[17] Lior Rokach. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2015.06.005. URL `https://www.sciencedirect.com/science/article/pii/S1566253515000561`.

[18] Yung-Yao Chen, Yu-Hsiu Lin, Chia-Ching Kung, Ming-Han Chung, I Yen, et al. Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. *Sensors*, 19(9):2047, 2019.

[19] A. Forrester, A. Sobester, and A. Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. Wiley, 2008. ISBN 9780470770795. URL `https://books.google.no/books?id=ulMHmeMnRCcC`.

[20] Xue Jiang, Wenxi Lu, Jin Na, Zeyu Hou, Yanxin Wang, and Baoming Chi. A stochastic optimization model based on adaptive feedback correction process and surrogate model uncertainty for dnapl-contaminated groundwater remediation design. *Stochastic Environmental Research and Risk Assessment*, 32(11):3195–3206, 2018.

[21] S. Koziel and A. Bekasiewicz. *Multi-objective Design Of Antennas Using Surrogate Models*. World Scientific Publishing Company, 2016. ISBN 9781786341495. URL `https://books.google.no/books?id=OMSkDQAAQBAJ`. p.45-46.

[22] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[23] Guy S. Handelman and Hong Kuan Kok et al. *Peering Into the Black Box of Artificial Intelligence: Evaluation Metrics of Machine Learning Methods*. American Journal of Roentgenology, 2019.

[24] Oscar Sheynin. *History of Statistics*. NG Verlag Berlin, 2012. p. 81.

[25] David John Cameron Mackay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[26] O. Martin. *Bayesian Analysis with Python*. Packt Publishing, 2016. ISBN 9781785889851. URL `https://books.google.no/books?id=t6PcDgAAQBAJ`.

[27] K.P. Hellwig. *Overfitting in Judgment-based Economic Forecasts: The Case of IMF Growth Projections*. IMF Working Papers. INTERNATIONAL MONETARY FUND, 2018. ISBN 9781484386187. URL `https://books.google.no/books?id=EKEZEAAAQBAJ`.

[28] Felipe Cucker and Ding Xuan Zhou. *Learning theory: an approximation theory viewpoint*, volume 24. Cambridge University Press, 2007. pp.59.

[29] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[30] Iain Murray. Gaussian processes and kernels. `https://www.inf.ed.ac.uk/teaching/courses/mlpr/2016/notes/w7c_gaussian_process_kernels.pdf`, 2016.

[31] Mauricio A. Alvarez, Lorenzo Rosasco, and Neil D. Lawrence. Kernels for vector-valued functions: a review, 2012.

[32] Edwin V Bonilla, Kian Chai, and Christopher Williams. Multi-task gaussian process prediction. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL `https://proceedings.neurips.cc/paper/2007/file/66368270ffd51418ec58bd793f2d9b1b-Paper.pdf`.

[33] Ricardo Andrade-Pacheco. Coregionalized regression model (vector-valued regression). `https://nbviewer.jupyter.org/github/SheffieldML/notebook/blob/master/GPy/coregionalized_regression_tutorial.ipynb`, 2015.

[34] Hans Wackernagel. *Multivariate geostatistics: an introduction with applications*. Springer Science & Business Media, 2013.

[35] Hefin I Rhys. *Machine Learning with R, the tidyverse, and mlr*. Manning Publications, 2020. pp.472-475.

[36] Giulio D'Agostini. A multidimensional unfolding method based on bayes' theorem. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 362(2-3):487–498, 1995.

[37] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophicol Trinsdictions ofthe Rogyal Society*, 1909. pp.415-446.

[38] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[39] Knut Wiig Mathisen. Integrated design and control of heat exchanger networks, 1994.

[40] Z. S. Spakovszky. Thermodynamics and propulsion. `http://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/notes.html`, 2002.

[41] A. Wächter and L. T. Biegler. *On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming*. Springer, 2006. pp. 25-27.

[42] 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.

[43] G. Bonaccorso. *Machine Learning Algorithms*. Packt Publishing, 2017. ISBN 9781785884511. URL `https://books.google.no/books?id=_-ZDDwAAQBAJ`.

[44] B. Schölkopf, B.S.A.J. Smola, A.J. Smola, F. Bach, MIT Press, and M.D.M.P.I.B.C.T.G.P.B. Scholkopf. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press, 2002. ISBN 9780262194754. URL `https://books.google.no/books?id=y8ORL3DWt4sC`. pp. xiv, 73, 89-81, 129-133.

[45] GPy. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[46] Michael Stutz. Get started with gawk: Awk language fundamentals. `https://web.archive.org/web/20110520232835/http://www.ibm.com/developerworks/aix/tutorials/au-gawk/section2.html`, 2006.

[47] K. Binmore and J. Davies. *Calculus: Concepts and Methods*. Cambridge University Press, 2001. ISBN 9780521775410. URL `https://books.google.no/books?id=sZOTrsHARlEC`.

[48] M. Yu Byron, John P. Cunningham, Gopal Santhanam, Stephen I. Ryu, Krishna V. Shenoy, and Maneesh Sahani. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. In *Advances in neural information processing systems*, pages 1881–1888, 2009.

[49] M. A. Osborne, S. J. Roberts, A. Rogers, S. D. Ramchurn, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, pages 109–120, 2008. doi: 10.1109/IPSN.2008.25.

[50] Chris Williams, Edwin V Bonilla, and Kian M Chai. Multi-task gaussian process prediction. *Advances in neural information processing systems*, pages 153–160, 2007.

[51] Mauricio A Alvarez and Neil D Lawrence. Sparse convolved gaussian processes for multi-output regression. In *NIPS*, volume 21, pages 57–64, 2008.

[52] Mauricio A Alvarez and Neil D Lawrence. Computationally efficient convolved multiple output gaussian processes. *The Journal of Machine Learning Research*, 12:1459–1500, 2011.

[53] Andrew Gordon Wilson, David A. Knowles, and Zoubin Ghahramani. Gaussian process regression networks, 2011.

[54] Gabriel Riutort-Mayol, Paul-Christian Bürkner, Michael R. Andersen, Arno Solin, and Aki Vehtari. Practical hilbert space approximate bayesian gaussian processes for probabilistic programming, 2020.

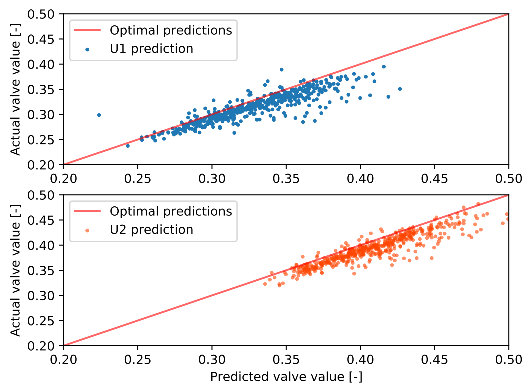
A Side-by-Side results

This appendix has attached to it all of the plots shown in the results (section 6), but in a different order. This makes comparison of the efficacy of the various approaches better, especially when evaluating the performance of the various ttratios.

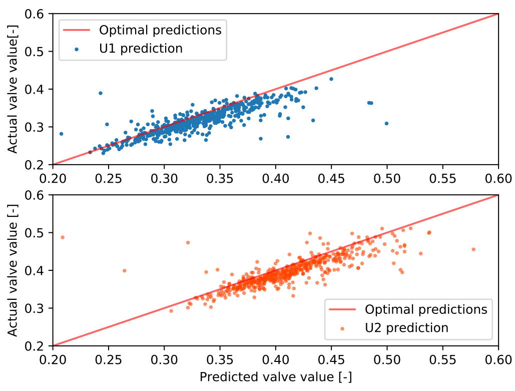
A.1 Default prior

Attached below (figures A.1 and A.2) is a comparison figures showing all the created scatter plots, and their respective histograms, for the YDU and the YU approach using a **default prior** for both ttratios.

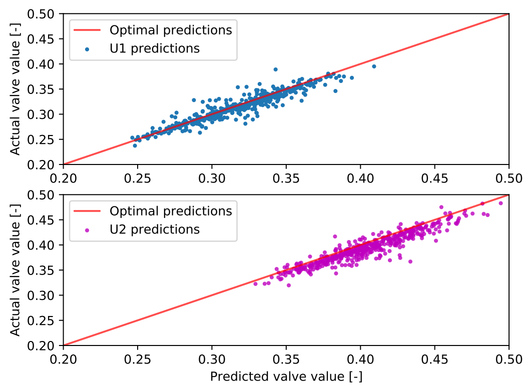
A.1.1 U-prediction comparison plots



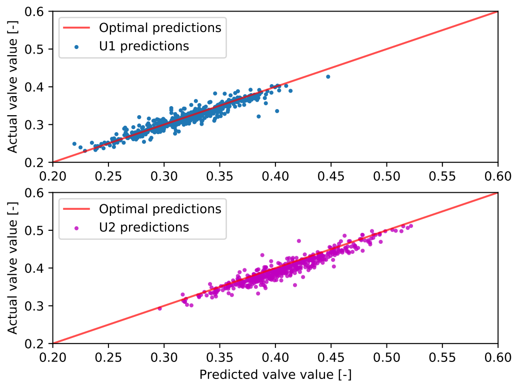
(a) Scatter plot of the predicted u-values for the **YDU** approach.



(b) Scatter plot of the predicted u-values for the **YDU** approach with ttratio as 1.3.



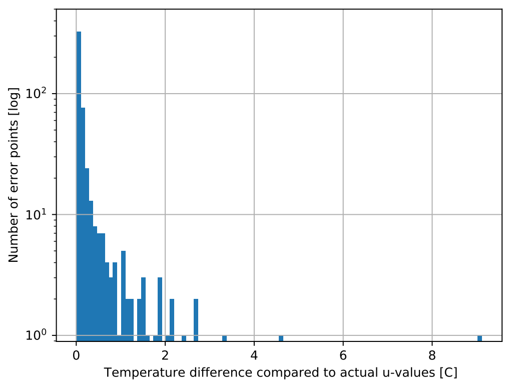
(c) Scatter plot of the predicted u-values for the **YU** approach.



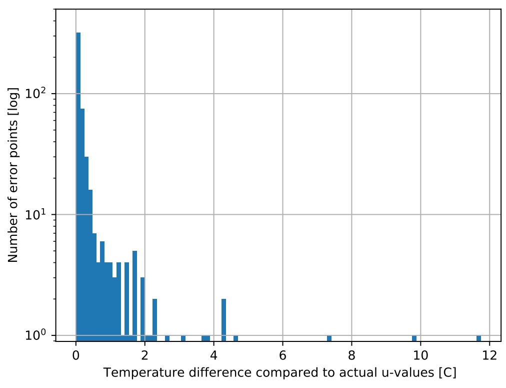
(d) Scatter plot of the predicted u-values for the **YU** approach with ttratio as 1.3.

Figure A.1: Comparison scatter plot for all the predicted u-values for the YDU and YU approach using a **default prior.**

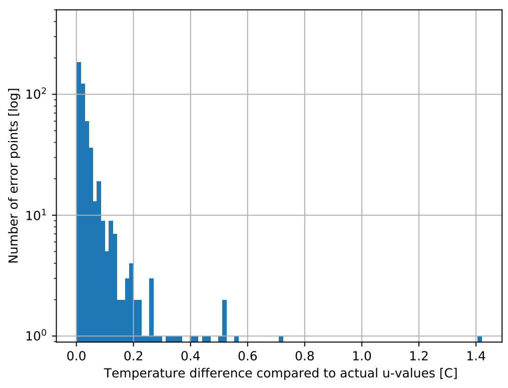
A.1.2 Histogram comparison plots



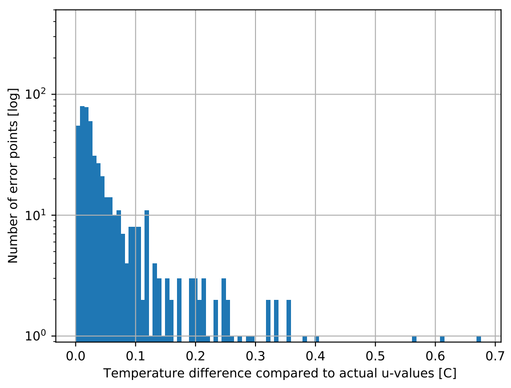
(a) Histogram of the resulting T-differences for the **YDU** approach.



(b) Histogram of the resulting T-differences for the **YDU** approach with ttratio as 1.3.



(c) Histogram of the resulting T-differences for the **YU** approach.



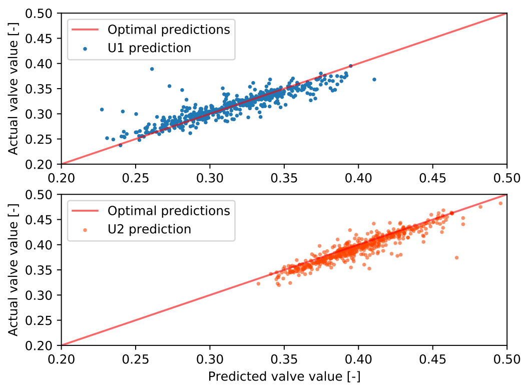
(d) Histogram of the resulting T-differences for the **YU** approach with ttratio as 1.3.

Figure A.2: Comparison histograms for all the resulting T-differences for the YDU and YU approach using a **default prior.**

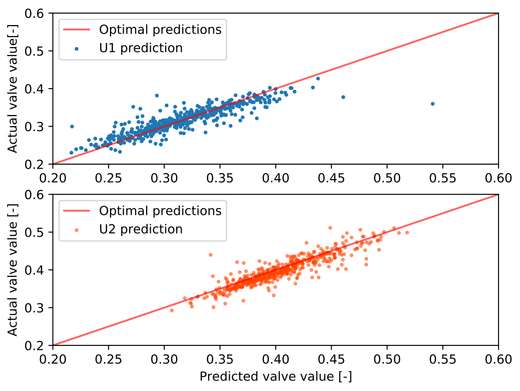
A.2 Linear prior adjustment

The plots for the YDU and YU approach using a **linear prior** can be seen below in figures A.3 and A.4.

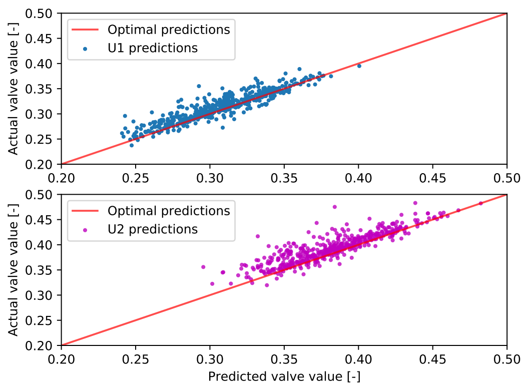
A.2.1 U-prediction comparison plots



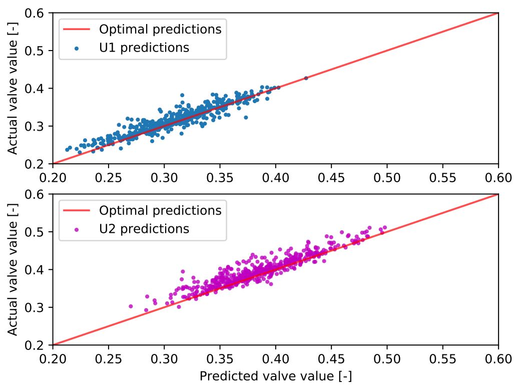
(a) Scatter plot of the predicted u-values for the **YDU** approach.



(b) Scatter plot of the predicted u-values for the **YDU** approach with ttratio as 1.3.



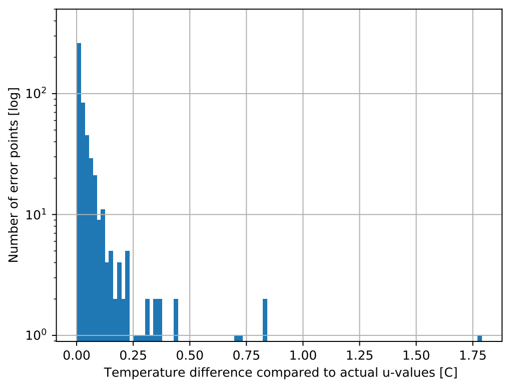
(c) Scatter plot of the predicted u-values for the **YU** approach.



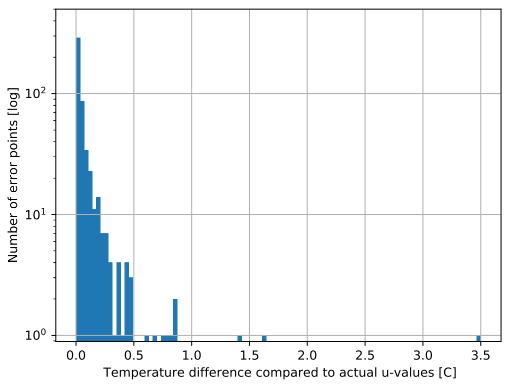
(d) Scatter plot of the predicted u-values for the **YU** approach with ttratio as 1.3.

Figure A.3: Comparison scatter plot for all the predicted u-values for the YDU and YU approach using a **linear prior.**

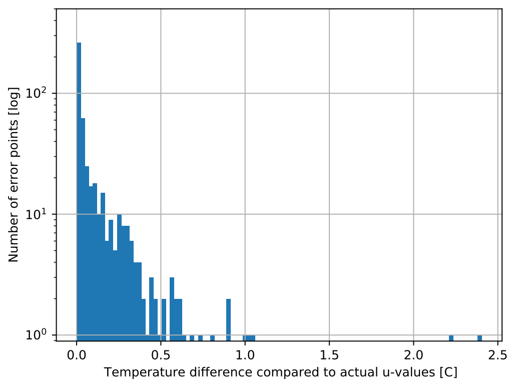
A.2.2 Histogram comparison plots



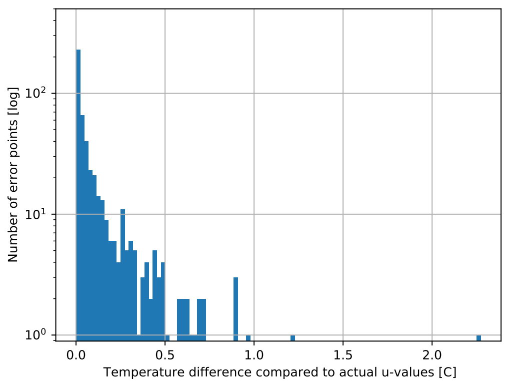
(a) Histogram of the resulting T-differences for the **YDU** approach.



(b) Histogram of the resulting T-differences for the **YDU** approach with ttratio as 1.3.



(c) Histogram of the resulting T-differences for the **YU** approach.



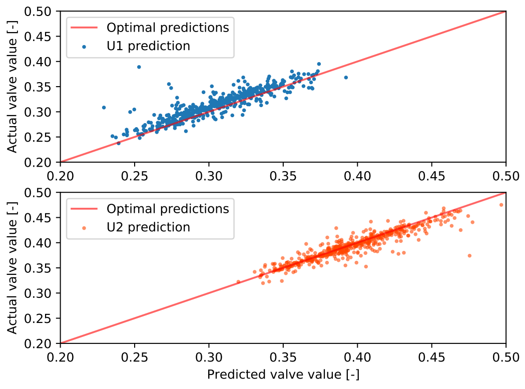
(d) Histogram of the resulting T-differences for the **YU** approach with ttratio as 1.3.

Figure A.4: Comparison histograms for all the resulting T-differences for the YDU and YU approach using a **linear prior.**

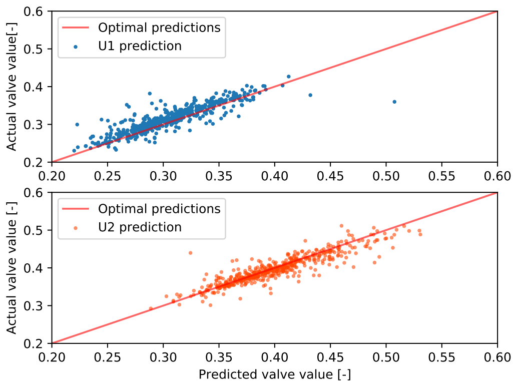
A.3 Quadratic prior adjustment

And finally, comparison plots for the **quadratic prior** are included below in figures A.5 and A.6.

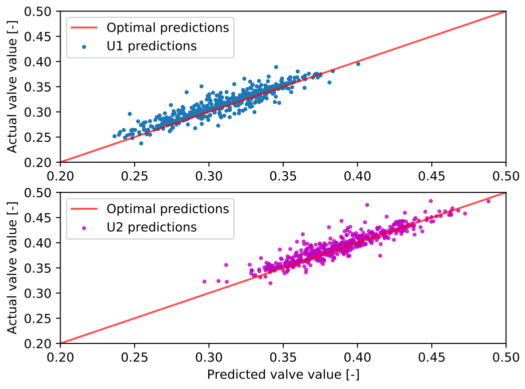
A.3.1 U-prediction comparison



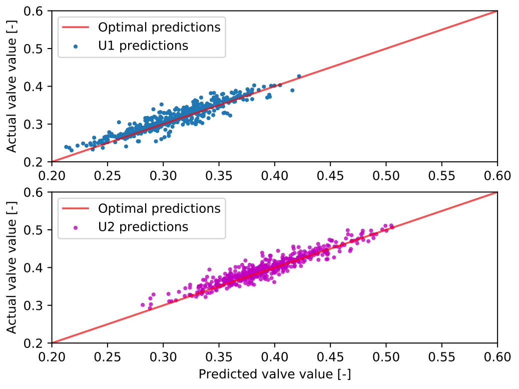
(a) Scatter plot of the predicted u-values for the **YDU** approach.



(b) Scatter plot of the predicted u-values for the **YDU** approach with ttratio as 1.3.



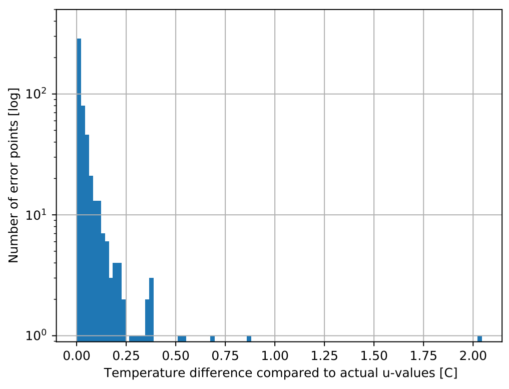
(c) Scatter plot of the predicted u-values for the **YU** approach.



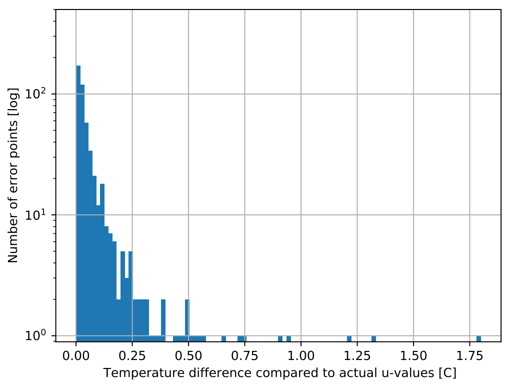
(d) Scatter plot of the predicted u-values for the **YU** approach with ttratio as 1.3.

Figure A.5: Comparison scatter plot for all the predicted u-values for the YDU and YU approach using a **quadratic prior.**

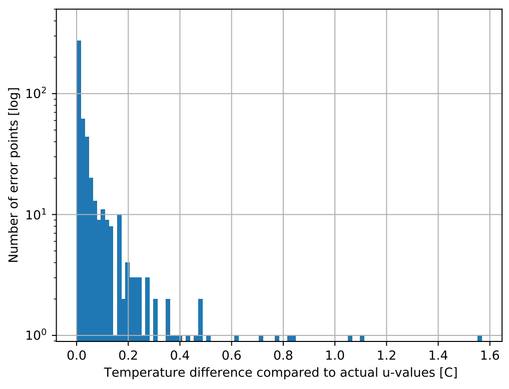
A.3.2 Histogram comparison plots



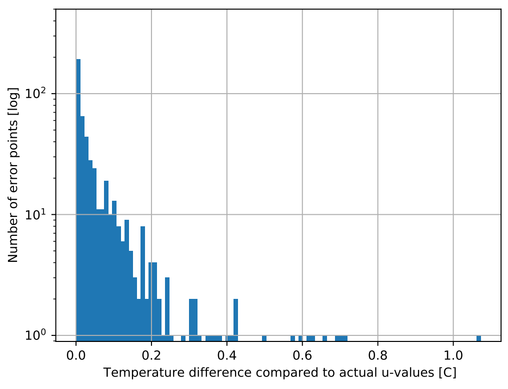
(a) Histogram of the resulting T-differences for the **YDU** approach.



(b) Histogram of the resulting T-differences for the **YDU** approach with ttratio as 1.3.



(c) Histogram of the resulting T-differences for the **YU** approach.

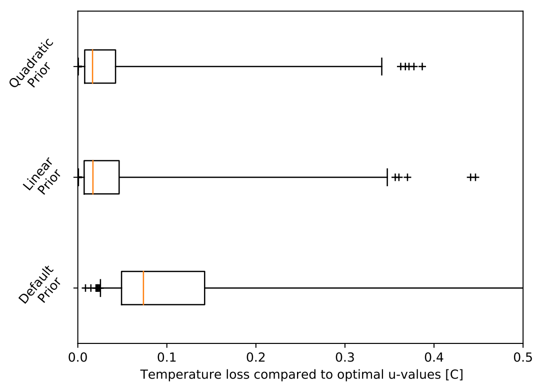


(d) Histogram of the resulting T-differences for the **YU** approach with ttratio as 1.3.

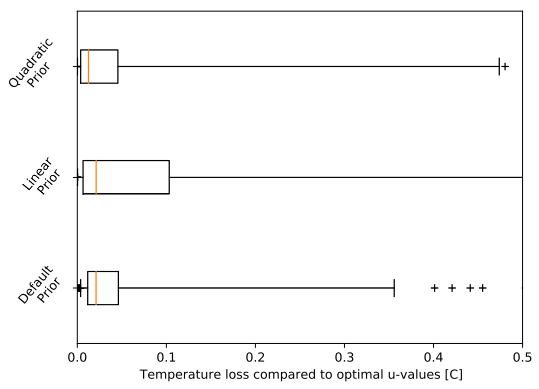
Figure A.6: Comparison histograms for all the resulting T-differences for the YDU and YU approach using a **quadratic prior.**

A.4 Boxplot Comparisons

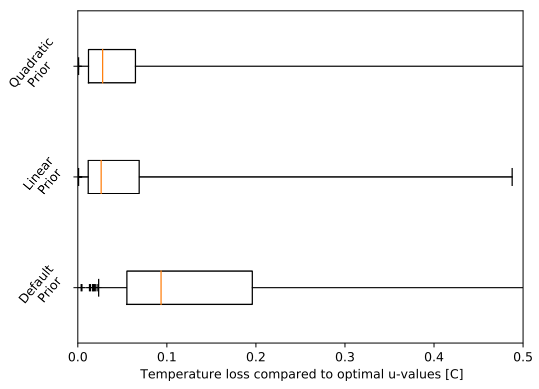
This appendix includes all the zoommed versions of the boxplots for easier visual comparison betweent he performance of the various approaches.



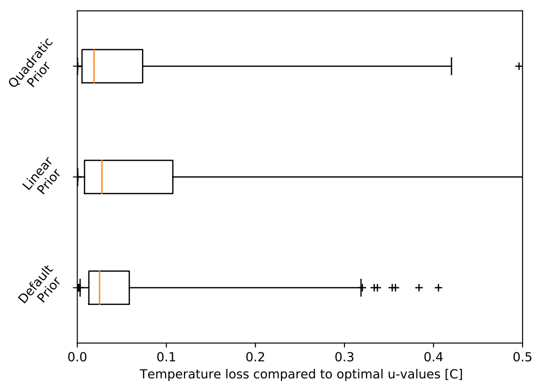
(a) Zoomed boxplot of the temperature loss and frequency for the **YDU** approach.



(b) Zoomed boxplot of the temperature loss and frequency for the **YU** approach.



(c) Zoomed boxplot of the temperature loss and frequency for the **YDU** approach outside the training region.



(d) Zoomed boxplot of the temperature loss and frequency for the **YU** approach outside the training region.

Figure A.7: Comparison boxplots for the resulting temperature loss and their frequencies. All versions are zoomed with a consistent axis for better visual clarity.

B Code Attachment

B.1 hex3_gendist.py

```python
import numpy as np

#np.random.seed(1) ## important when working with random numbers!

def gen_dataset(N, ttratio=1.0):

    parspan = {}
    # Defining disturbance box [center, variability]
    parspan['T0'] = [60, 0]  # C
    parspan['w0'] = [105, 0]  # kW/K
    parspan['wh1'] = [40, 10]  # kW/K
    parspan['wh2'] = [50, 10]  # kW/K
    parspan['wh3'] = [30, 10]  # kW/K
    parspan['Th1'] = [150, 0]  # C
    parspan['Th2'] = [150, 0]  # C
    parspan['Th3'] = [150, 0]  # C
    parspan['UA1'] = [65, 15]  # kW/K
    parspan['UA2'] = [80, 10]  # kW/K
    parspan['UA3'] = [95, 15]  # kW/K

    # Copied from transfer learning
    parspan['Ts'] = [0, 0]  # C
    parspan['h1'] = [0, 0]  # kW/K
    parspan['h2'] = [0, 0]  # kW/K
    parspan['h3'] = [0, 0]  # kW/K

    randmatrix = np.random.rand(len(parspan), N)
    parvec = {}
    for i, parname in enumerate(parspan.keys()):
        parvec[parname] = parspan[parname][0] + ttratio * (2 * randmatrix[i] - 1) * (parspan[
            parname][-1])

    par0 = [{key: value[i] for key, value in parvec.items()} for i in range(N)] #Convert from
        dict->array for GPy

    # Random input values
    # Smith, Noah A., and Roy W. Tromble. "Sampling uniformly from the unit simplex." Johns
        Hopkins University, Tech. Rep 29 (2004).
    dim = 3
    x = np.sort(np.random.rand(dim - 1, N), axis=0)
    x = np.concatenate([np.zeros((1, N)), x, np.ones((1, N))], axis=0)
    alpha = x[1:] - x[:-1]

    # # Checking uniformity
    # ax = plt.axes(projection='3d')
    # ax.plot(alpha[0], alpha[1], alpha[2] ,'b.')
    # plt.show()

    alpha = alpha[:-1]

    return par0, alpha
```

B.2 hex3_chen.py

```python
from casadi import *
import numpy as np

nlpopts = {'ipopt': {'print_level':0}, 'print_time':False}
x_vars = ['alpha3','T','Tstar1','Tstar2','Tstar3','The1','The2','The3','Q1','Q2','Q3','Qloss1'
    ,'Qloss2','Qloss3','T1','T2','T3']
u_vars = ['alpha1','alpha2']

meas_sets = {
    #1: ['T0', 'T1', 'T2', 'T3', 'The1', 'The2', 'The3', 'alpha1', 'alpha2'],
    1: ['alpha1', 'alpha2', 'T1', 'T2', 'T3', 'The1', 'The2', 'The3'],
    2: ['wh1', 'wh2', 'wh3', 'Th1', 'Th2', 'Th3'],
    3: ['T0', 'w0', 'wh1', 'wh2', 'wh3', 'Th1', 'Th2', 'Th3', 'UA1', 'UA2', 'UA3', 'Ts', 'h1',
        'h2', 'h3']
}

Ti_max = 1500

def model(par):
    T = SX.sym('T')
    Tstar1 = SX.sym('Tstar1')
    Tstar2 = SX.sym('Tstar2')
    Tstar3 = SX.sym('Tstar3')
    The1 = SX.sym('The1')
    The2 = SX.sym('The2')
    The3 = SX.sym('The3')
    Q1 = SX.sym('Q1')
    Q2 = SX.sym('Q2')
    Q3 = SX.sym('Q3')
    Qloss1 = SX.sym('Qloss1')
    Qloss2 = SX.sym('Qloss2')
    Qloss3 = SX.sym('Qloss3')
    T1 = SX.sym('T1')
    T2 = SX.sym('T2')
    T3 = SX.sym('T3')
    alpha1 = SX.sym('alpha1')
    alpha2 = SX.sym('alpha2')
    alpha3 = SX.sym('alpha3')

    T0 = par['T0']
    w0 = par['w0']
    Th1 = par['Th1']
    Th2 = par['Th2']
    Th3 = par['Th3']
    wh1 = par['wh1']
    wh2 = par['wh2']
    wh3 = par['wh3']
    UA1 = par['UA1']
    UA2 = par['UA2']
    UA3 = par['UA3']

    Ts = par['Ts']
    h1 = par['h1']
    h2 = par['h2']
    h3 = par['h3']

    dTlm1 = ( (Th1 - Tstar1) * (The1 - T0) * ( (Th1 - Tstar1) + (The1 - T0) )/2)**(1/3)
    dTlm2 = ( (Th2 - Tstar2) * (The2 - T0) * ( (Th2 - Tstar2) + (The2 - T0) )/2)**(1/3)
    dTlm3 = ( (Th3 - Tstar3) * (The3 - T0) * ( (Th3 - Tstar3) + (The3 - T0) )/2)**(1/3)

    f0 = - T + alpha1*T1 + alpha2*T2 + alpha3*T3
    f01 = alpha1 + alpha2 + alpha3 - 1
    f11 = - Q1 + w0*alpha1*(Tstar1 - T0)
    f12 = - Q2 + w0*alpha2*(Tstar2 - T0)
    f13 = - Q3 + w0*alpha3*(Tstar3 - T0)
    f21 = - Q1 + UA1*dTlm1
    f22 = - Q2 + UA2*dTlm2
    f23 = - Q3 + UA3*dTlm3
    f31 = - Q1 + wh1*(Th1 - The1)
    f32 = - Q2 + wh2*(Th2 - The2)
    f33 = - Q3 + wh3*(Th3 - The3)
    f41 = - Qloss1 + w0*alpha1*(T1 - Tstar1)
    f42 = - Qloss2 + w0*alpha2*(T2 - Tstar2)
    f43 = - Qloss3 + w0*alpha3*(T3 - Tstar3)
    f51 = - Qloss1 + h1*(Ts - T1)
    f52 = - Qloss2 + h2*(Ts - T2)
    f53 = - Qloss3 + h3*(Ts - T3)

    x = vertcat(alpha3,T,Tstar1,Tstar2,Tstar3,The1,The2,The3,Q1,Q2,Q3,Qloss1,Qloss2,Qloss3,T1,
        T2,T3)
    f = vertcat(f0,f01,f11,f12,f13,f21,f22,f23,f31,f32,f33,f41,f42,f43,f51,f52,f53)
    u = vertcat(alpha1,alpha2)
    J = -T

    return {'x': x, 'u': u, 'f': f, 'J': J}

def output(u, par, x0=None):

    m = model(par)
    nx = np.prod(m['x'].shape)
    nu = np.prod(m['u'].shape)
    nf = np.prod(m['f'].shape)

    if x0 is None:
        x0 = np.array( [0.33]*(nu+1) + [(par['T0']+par['Th1'])/2, (par['T0']+par['Th2'])/2, (
            par['T0']+par['Th3'])/2]*2 + [par['T0']] * (nx - 2*(nu+1) - 1) )

    nlp = {}                              # NLP declaration
    nlp['x'] = vertcat(m['u'],m['x'])     # decision vars
    nlp['f'] = m['J']                     # objective
    nlp['g'] = m['f']                     # constraints

    # Create solver instance
    F = nlpsol('F','ipopt',nlp,nlpopts)

    # Solve the problem using a guess
    lbx = np.array([*u]+[0]*(1)+[-inf]*(nx-1))  # constraint on inputs and first state (last
        flow split)
    ubx = np.array([*u]+[1]*(1)+[+inf]*(nx-1))  # upper limit on splits is not necessary, but
        will automatically be satisfied

    lbx[(nu+2):(nu+2+3*2)] = par['T0']  # constraint on temperatures
    ubx[(nu+2):(nu+2+3*2)] = np.array([par['Th1'], par['Th2'], par['Th3'], par['Th1'], par['
        Th2'], par['Th3']])  # constraint on temperatures


    r = F(x0=x0, lbg=np.zeros(nf), ubg=np.zeros(nf), lbx=lbx, ubx=ubx)

    sol = r['x'].full().reshape(-1)

    return {'x': sol[-nx:], 'success': F.stats()['success']}

def cost(u,par):
    m = model(par)
    F = Function('F',[m['x'], m['u']],[m['J']], ['x','u'], ['J'])
    out = output(u,par)  #np.zeros(nx);
    J = F(out['x'], u)
    return {'J': J.full().reshape(-1), 'success': out['success']}


def grad(u,par):
    m = model(par)

    F = Function('F',[m['x'], m['u']],[m['f'], m['J']], ['x','u'], ['f','J'])
    G = rootfinder('G','newton',F)
    out = output(u,par)  #np.zeros(nx);
    Jufun = G.factory('Ju', ['x','u'], ['jac:J:u'])

    delta = 0
    Ju = Jufun(out['x']+delta, u).full().reshape(-1)
    # while not G.stats()['success']:
    #     delta = delta*10;
    #     Ju = Jufun(xguess+delta, u).full().reshape(-1)
    return {'grad': Ju, 'success': True}


def output_meas(meas_set, u, par):
    meas_vars = meas_sets[meas_set]
    y = np.zeros((len(meas_vars),))
    out = output(u, par)
    x = out['x']
    for i, var in enumerate(meas_vars):
        if var in par:
            y[i] = par[var]
        elif var in u_vars:
            y[i] = u[u_vars.index(var)]
        elif var in x_vars:
            y[i] = x[x_vars.index(var)]
        else:
            y[i] = np.nan
    return {'y': y, 'success': out['success']}

def optim(par, x0=None):

    m = model(par)
    nx = np.prod(m['x'].shape)
    nu = np.prod(m['u'].shape)
    nf = np.prod(m['f'].shape)

    nlp = {}                               # NLP declaration
    nlp['x'] = vertcat(m['u'],m['x'])      # decision vars
    nlp['f'] = m['J']                      # objective
    nlp['g'] = m['f']                      # constraints

    # Create solver instance
    F = nlpsol('F','ipopt',nlp,nlpopts)

    Tbackoff = 1
    # Trand = 1;
    alphabackoff = 1e-3

    if x0 is None:
        # x0 = np.zeros(nx+nu);
        # x0[:nu+1] = 1/(nu+1);
        # x0[(nu+2):(nu+2+3*2)] = par['T0'] + Tbackoff #+ Trand*np.random.rand(3*2); # [Tstar1
            ,Tstar2,Tstar3,The1,The2,The3]
        x0 = np.array( [0.33]*(nu+1) + [(par['T0']+par['Th1'])/2, (par['T0']+par['Th2'])/2, (
            par['T0']+par['Th3'])/2]*2 + [par['T0']] * (nx - 2*(nu+1) - 1) )

    # Solve the problem using first guess

    lbx = np.array([alphabackoff]*(nu+1)+[-inf]*(nx-1))  # constraint on inputs and first
        state (last flow split)
    ubx = np.array([1]*(nu+1)+[+inf]*(nx-1))  # upper limit on splits is not necessary, but
        will automatically be satisfied

    lbx[(nu+2):(nu+2+3*2)] = par['T0']  # constraint on temperatures
    ubx[(nu + 2):(nu + 2 + 3)] = np.array([min(Ti_max, t) for t in [par['Th1'], par['Th2'],
        par['Th3']] ])  # constraint on temperatures
    ubx[(nu + 2 + 3):(nu + 2 + 3 * 2)] = np.array([par['Th1'], par['Th2'], par['Th3']])  #
        constraint on temperatures

    r = F(x0=x0, lbg=np.zeros(nf), ubg=np.zeros(nf), lbx=lbx, ubx=ubx)

    # while not F.stats()['success']:
    #     Trand = Trand + 1;
    #     x0[(nu+2):(nu+2+3*2)] = par['T0'] + Tbackoff + Trand*np.random.rand(3*2);
    #     r = F(x0=x0, lbg=np.zeros(nf), ubg=np.zeros(nf), lbx=lbx, ubx=ubx);

    sol = r['x'].full().reshape(-1)

    return {'u': sol[:nu], 'x': sol[-nx:], 'success': F.stats()['success']}
```

B.3 YU_training.py

```
##########################PREFACE##########################
# Code by Michael Lindbak
# For academic masters degree in Chemical Engineering Spring 2021

# Importing tools for the code
import GPy                                                  #ML
import numpy as np                                          #Numpy
import matplotlib.pyplot as plt                             #Plotting
from sklearn.preprocessing import MinMaxScaler              #Scaling
from sklearn.linear_model import LinearRegression           #Prior adjustment
import pandas as pd                                         #Data-management
from sklearn.preprocessing import PolynomialFeatures        #Prior adjustment
from sklearn.pipeline import make_pipeline                  #prior adjustmtnt
from sklearn.linear_model import Ridge                      #prior adjustment
import joblib                                               #Saving/Exporting Scaler

#Import pre-made code
import hex3_gendist as gendist        #Code for generating random data-points
import hex3_chen as ch                #Code for data calculations of the system

# Import prediction code from Supervisor
def predict_all(m: GPy.models.GPCoregionalizedRegression, X):
    ny = len(np.unique(m.output_index))
    y = []
    covy = []
    Xaug = np.hstack((X, 0.0 * np.ones_like(X[:, 0:1])))
    for iy in range(ny):
        Xaug[:, -1:] = iy
        y_i, covy_i = m.predict(Xaug, Y_metadata={'output_index': Xaug[:, -1:].astype(int)})
        y.append(y_i)
        covy.append(covy_i)
    return np.hstack(y), np.hstack(covy)


##########################CODING##########################

## Generating dataset
gen_data = gendist.gen_dataset(1200)

## Separate data into GPy standards
# y-data
y_train = np.array([ch.optim(x_pt)['u'] for x_pt in gen_data[0]])
yscale = MinMaxScaler()
y_train = np.array(yscale.fit_transform(y_train))
joblib.dump(yscale, 'yu_uscale.gz')
Y_train = np.array(list(zip(*y_train)))
Y_train = np.array([i[:, None] for i in Y_train])

# x-data
x_train = np.array([ch.output_meas(1, [gen_data[1][0][i], gen_data[1][1][i]], gen_data[0][i])[
    'y']
                    for i in range(len(gen_data[0]))])
xscale = MinMaxScaler()
x_train = np.array(xscale.fit_transform(x_train))
joblib.dump(xscale, 'yu_yscale.gz')              #Exports the scaler

# u-pr
#reg = LinearRegression().fit(x_train, y_train)                              #Linear Prior (
    uncomment line)
#reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(x_train, y_train)    #Polynomial prior
    (uncomment line)
#u_pr = y_train - reg.predict(np.array(x_train))                              #Prior adjustment
     (uncomment line)
u_pr = y_train

U_pr = np.array(list(zip(*u_pr)))
U_pr = np.array([i[:, None] for i in U_pr])


#Saving data for later use
#Note that this data is not sorted into the respective folders with priors and the like
x_yutrain = pd.DataFrame(x_train)
y_yutrain = pd.DataFrame(y_train) #NOTE: NOT Y_train !!!!!
x_yutrain.to_csv('x_yutrain.csv', index=False)
y_yutrain.to_csv('y_yutrain.csv', index=False)



# Making a ML Kernel
K = GPy.kern.RBF(input_dim=x_train.shape[1])
icm = GPy.util.multioutput.ICM(input_dim=x_train.shape[1], num_outputs=Y_train.shape[0],
    kernel=K)
m = GPy.models.GPCoregionalizedRegression([x_train]*U_pr.shape[0], U_pr, kernel = icm)
m['.*rbf.lengthscale'].constrain_bounded(0.1,200)
m['mixed_noise.Gaussian_noise_.*.variance'].constrain_bounded(1e-6,1e-3)
```

```
m['.*rbf.variance'] = 50.
m.optimize_restarts(2)
print(m)
print("\n\n", m.ICM.B.W)
print(m.ICM.B.kappa)
np.save('ML_parameters/YtoU.npy', m.param_array)
```

B.4 YD_training.py

```
##########################PREFACE##########################
# Code by Michael Lindbak
# For academic masters degree in Chemical Engineering Spring 2021

# Importing tools for the code
import GPy                                              #ML
import numpy as np                                      #Numpy
import matplotlib.pyplot as plt                         #Plotting
from sklearn.preprocessing import MinMaxScaler          #Scaling
from sklearn.linear_model import LinearRegression       #Prior adjustment
import pandas as pd                                     #Data-management
from sklearn.preprocessing import PolynomialFeatures    #Prior adjustment
from sklearn.pipeline import make_pipeline              #prior adjustmtnt
from sklearn.linear_model import Ridge                  #prior adjustment
import joblib                                           #Saving/Exporting Scaler

#Import pre-made code
import hex3_gendist as gendist        #Code for generating random data-points
import hex3_chen as ch                #Code for data calculations of the system

# Import prediction code from Supervisor
def predict_all(m: GPy.models.GPCoregionalizedRegression, X):
    ny = len(np.unique(m.output_index))
    y = []
    covy = []
    Xaug = np.hstack((X, 0.0 * np.ones_like(X[:, 0:1])))
    for iy in range(ny):
        Xaug[:, -1:] = iy
        y_i, covy_i = m.predict(Xaug, Y_metadata={'output_index': Xaug[:, -1:].astype(int)})
        y.append(y_i)
        covy.append(covy_i)
    return np.hstack(y), np.hstack(covy)


##########################CODING##########################

## Generating dataset
gen_data = gendist.gen_dataset(1200)

## Separate data into GPy standards
# y-data
y_train = pd.DataFrame(gen_data[0])
y_train = y_train.drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', 'Th2', 'Th3']) #
    Estimating wh1-3, UA1-3
y_train = np.array(y_train)
yscale = MinMaxScaler()                       #Class used for scaling
y_train = yscale.fit_transform(y_train)
Y_train = np.array(list(zip(*y_train)))
Y_train = np.array([i[:, None] for i in Y_train])    #"Flips" the y-train in a manner GPy likes
     (I think)
joblib.dump(yscale, 'Masters_yscale.gz')             #Exports the scaler


# x-data
x_train = np.array([ch.output_meas(1, [gen_data[1][0][i], gen_data[1][1][i]], gen_data[0][i])[
    'y']
                   for i in range(len(gen_data[0]))])
xscale = MinMaxScaler()
x_train = np.array(xscale.fit_transform(x_train))
joblib.dump(xscale, 'Masters_xscale.gz')             #Exports the scaler

# y-pr
#reg = LinearRegression().fit(x_train, y_train)                              #Linear Prior (
    uncomment line)
#reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(x_train, y_train)    #Polynomial prior
    (uncomment line)
#y_pr = y_train - reg.predict(np.array(x_train))                              #Prior adjustment
     (uncomment line)
y_pr = y_train
Y_pr = np.array(list(zip(*y_pr)))
Y_pr = np.array(([i[:, None] for i in Y_pr]))


#Exporting data
#Note that this data is not sorted into the respective folders with priors and the like
x_ttrain = pd.DataFrame(x_train)
y_ttrain = pd.DataFrame(y_train) #NOTE: NOT Y_train !!!!!
x_ttrain.to_csv('x_ttrain.csv', index=False)
y_ttrain.to_csv('y_ttrain.csv', index=False)
```

```
# Making a ML Kernel
K = GPy.kern.RBF(input_dim=x_train.shape[1])
icm = GPy.util.multioutput.ICM(input_dim=x_train.shape[1], num_outputs=y_train.shape[1],
     kernel=K)
m = GPy.models.GPCoregionalizedRegression([x_train]*y_train.shape[1], Y_train, kernel = icm)
m['.*rbf.lengthscale'].constrain_bounded(0.1,200)
m['mixed_noise.Gaussian_noise_.*.variance'].constrain_bounded(1e-6,1e-3)
m['.*rbf.variance'] = 50.
m.optimize_restarts(2)
#Display hyperparameters
print(m)
print("\n\n", m.ICM.B.W)
print(m.ICM.B.kappa)
np.save('ML_parameters/YtoD.npy', m.param_array)



## Control values
control_data = gendist.gen_dataset(600,1)
control_dataX = np.array([ch.output_meas(1, [control_data[1][0][i], control_data[1][1][i]],
                                         control_data[0][i])['y'] for i in range(len(
                                              control_data[0]))])
control_dataX = xscale.transform(control_dataX)

predict_control = []
for i in range(len(control_dataX)):
    predict_control.append(predict_all(m, np.array([control_dataX[i]]))[0][0])    #Predicting
         w0 wh1-3 Th1-3

#predict_control += reg.predict(control_dataX)
predict_control = np.array(yscale.inverse_transform(predict_control))
actual_control = pd.DataFrame(control_data[0])
actual_control = actual_control.drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', 'Th2'
     , 'Th3'])
actual_control = np.array(actual_control)#yscale.inverse_transform(y_train))#actual_control)
        #If prior is adjusted
control_dataX = xscale.inverse_transform(control_dataX)

print("Predict", predict_control)
print("\n\nActual", actual_control)


## Plotting
fig, (ax1, ax2) = plt.subplots(2)

ax1.scatter(actual_control[:,0], predict_control[:,0],
              s=5, label="wh1 predictions")
ax1.scatter(actual_control[:,1], predict_control[:,1],
              s=5, label="wh2 predictions")
ax1.scatter(actual_control[:,2], predict_control[:,2],
              s=5, label="wh3 predictions")
ax1.set(xlabel= "Actual wh-values [kW/K]", ylabel="Predicted Th-values [kW/K]")
ax1.set_title("wh 1-3 Control, ttratio=1")
ax1.set_xlim(20, 60)
ax1.set_ylim(20, 60)
ax1.plot(np.linspace(20,60), np.linspace(20,60),
          color="red", alpha=0.25, label="Optimal predictions")
ax1.legend()
ax1.grid()

ax2.scatter(actual_control[:,3], predict_control[:,3],
              s=5, label="UA1 predictions")
ax2.scatter(actual_control[:,4], predict_control[:,4],
              s=5, label="UA2 predictions")
ax2.scatter(actual_control[:,5], predict_control[:,5],
              s=5, label="UA3 predictions")
ax2.set(xlabel="Actual UA-values [kW/K]", ylabel="Predicted UA-values [kW/K]")
ax2.set_title("UA 1-3 Control, ttratio=1")
ax2.set_xlim(50,110)
ax2.set_ylim(50,110)
ax2.plot(np.linspace(50,110), np.linspace(50,110),
          color="red", alpha=0.25, label="Optimal predictions")
ax2.legend()
ax2.grid()

plt.show()

## Extreme values
extreme_data = gendist.gen_dataset(600,1.25)
extreme_dataX = np.array([ch.output_meas(1, [extreme_data[1][0][i], extreme_data[1][1][i]],
                                         extreme_data[0][i])['y'] for i in range(len(
                                              extreme_data[0]))])
extreme_dataX = xscale.transform(extreme_dataX)

predict_extreme = []
for i in range(len(extreme_dataX)):
    predict_extreme.append(predict_all(m, np.array([extreme_dataX[i]]))[0][0])    #Predicting
         w0 wh1-3 Th1-3

predict_extreme = np.array(yscale.inverse_transform(predict_extreme))
```

```
actual_extreme = pd.DataFrame(extreme_data[0])
actual_extreme = actual_extreme.drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', 'Th2'
    , 'Th3'])
actual_extreme = np.array(actual_extreme)#yscale.inverse_transform(y_train))#actual_control)
       # if prior adjusted
extreme_dataX = xscale.inverse_transform(extreme_dataX)

print("Predict", predict_extreme)
print("\n\nActual", actual_extreme)


## Plotting
fig, (ax1, ax2) = plt.subplots(2)

ax1.scatter(actual_extreme[:,0], predict_extreme[:,0],
            s=5, label="wh1 predictions")
ax1.scatter(actual_extreme[:,1], predict_extreme[:,1],
            s=5, label="wh2 predictions")
ax1.scatter(actual_extreme[:,2], predict_extreme[:,2],
            s=5, label="wh3 predictions")
ax1.set(xlabel= "Actual wh-values [kW/K]", ylabel="Predicted wh-values [kW/K]")
ax1.set_title("wh 1-3 Extreme, ttratio=1.25")
ax1.set_xlim(10, 70)
ax1.set_ylim(10, 70)
ax1.plot(np.linspace(10,70), np.linspace(10,70),
         color="red", alpha=0.25, label="Optimal predictions")
ax1.legend()
ax1.grid()

ax2.scatter(actual_extreme[:,3], predict_extreme[:,3],
            s=5, label="UA1 predictions")
ax2.scatter(actual_extreme[:,4], predict_extreme[:,4],
            s=5, label="UA2 predictions")
ax2.scatter(actual_extreme[:,5], predict_extreme[:,5],
            s=5, label="UA3 predictions")
ax2.set(xlabel="Actual UA-values [kW/K]", ylabel="Predicted UA-values [kW/K]")
ax2.set_title("UA 1-3 Extreme, ttratio=1.25")
ax2.set_xlim(40,110)
ax2.set_ylim(40,110)
ax2.plot(np.linspace(40,110), np.linspace(40,110),
         color="red", alpha=0.25, label="Optimal predictions")
ax2.legend()
ax2.grid()


plt.show()
```

B.5 DU_training.py

```
###########################PREFACE##########################
# Code by Michael Lindbak
# For academic masters degree in Chemical Engineering Spring 2021

# Importing tools for the code
import GPy
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression       #Prior adjustment
from sklearn.preprocessing import PolynomialFeatures    #Prior adjustment
from sklearn.pipeline import make_pipeline              #prior adjustmtnt
from sklearn.linear_model import Ridge                  #prior adjustment
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import joblib

#Import pre-made code
import hex3_gendist as gendist       #Code for generating random data-points
import hex3_chen as ch               #Code for data calculations of the system

# Import prediction code from supervisor
def predict_all(m: GPy.models.GPCoregionalizedRegression, X):
    ny = len(np.unique(m.output_index))
    y = []
    covy = []
    Xaug = np.hstack((X, 0.0 * np.ones_like(X[:, 0:1])))
    for iy in range(ny):
        Xaug[:, -1:] = iy
        y_i, covy_i = m.predict(Xaug, Y_metadata={'output_index': Xaug[:, -1:].astype(int)})
        y.append(y_i)
        covy.append(covy_i)
    return np.hstack(y), np.hstack(covy)


###########################CODING##########################

##Generating dataset##
gen_data = gendist.gen_dataset(1200, 1)

## Separate data into GPy standards
```

```
# x-data
x_train = pd.DataFrame(gen_data[0])
x_train = x_train.drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', 'Th2', 'Th3']) #wh1
    -3 and UA1-3
x_train = np.array(x_train)
xscale = MinMaxScaler()
x_train = np.array(xscale.fit_transform(x_train))
joblib.dump(xscale, 'Starting_xscale.gz')

# y-data
y_train = np.array([ch.optim(x_pt)['u'] for x_pt in gen_data[0]])
yscale = MinMaxScaler()
y_train = yscale.fit_transform(y_train)
joblib.dump(yscale, 'Starting_yscale.gz')

Y_train = np.array(list(zip(*y_train)))
Y_train = np.array([i[:, None] for i in Y_train])


# y-pr
#reg = LinearRegression().fit(x_train, y_train)                         #Linear Prior (
    uncomment line)
#reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(x_train, y_train)  #Polynomial Prior
    (uncomment line)
#u_pr = y_train - reg.predict(np.array(x_train))                        #Prior adjustment
    (uncomment line)
u_pr = y_train
U_pr = np.array(list(zip(*u_pr)))
U_pr = np.array([i[:, None] for i in U_pr])


#Saving data for re-use
#Note that this data is not sorted into the respective folders with priors and the like
x_strain = pd.DataFrame(x_train)
y_strain = pd.DataFrame(y_train) #NOTE: NOT Y_train
y_strain.to_csv('y_strain.csv', index=False)
x_strain.to_csv('x_strain.csv', index=False)


##Making a ML kernel
K = GPy.kern.RBF(input_dim=x_train.shape[1])
icm = GPy.util.multioutput.ICM(input_dim=x_train.shape[1], num_outputs=y_train.shape[1],
    kernel=K)  #should be 2 if fail
m = GPy.models.GPCoregionalizedRegression([x_train]*U_pr.shape[0], U_pr, kernel=icm)
m['mixed_noise.Gaussian_noise_.*.variance'].constrain_bounded(1e-6,1e-3)
m['.*rbf.variance'] = 100.
m.optimize_restarts(2)
#Display hyperparameters
print("\n\n", m.ICM.B.W)
print(m.ICM.B.kappa)
np.save('ML_parameters/DtoU.npy', m.param_array)


##Control values
control_data = gendist.gen_dataset(600, 1)
control_dataX = pd.DataFrame(control_data[0]).drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0'
    , 'Th1', 'Th2', 'Th3'])
control_dataX = np.array(control_dataX)
control_dataX = xscale.transform(control_dataX)

predict_controlU = []
actual_controlU = []
for i in range(len(control_dataX)):
    predict_controlU.append(yscale.inverse_transform(predict_all(m, np.array([control_dataX[i
        ]]))[0][0].reshape(1, -1))[0])# +
                                                        #reg.predict(control_dataX[i].reshape(1,
                                                            -1)))[0])
    actual_controlU.append(ch.optim(control_data[0][i])['u'])

predict_controlU = np.array(predict_controlU)
#predict_controlU = predict_controlU + reg.predict(control_dataX)   #If prior adjusted
actual_controlU = np.array(actual_controlU)

##Plotting
fig, (ax1, ax2) = plt.subplots(2)

ax1.scatter(actual_controlU[:,0], predict_controlU[:,0],
            s=5, label="Predictions")
ax1.set(xlabel= "Actual U1-value", ylabel="Predicted U1-value")
ax1.set_title("U1 Control, ttratio=1")
ax1.set_xlim(0.2, 0.5)
ax1.set_ylim(0.2, 0.5)
ax1.plot(np.array(np.linspace(0,1)), np.array(np.linspace(0,1)),
         color="red", alpha=0.25, label="Optimal predictions")
ax1.legend()
ax1.grid()

ax2.scatter(actual_controlU[:,1], predict_controlU[:,1],
            s=5, label="Predictions")
ax2.set(xlabel= "Actual U2-value", ylabel="Predicted U2-value")
```

```
ax2.set_title("U2 Control, ttratio=1")
ax2.set_xlim(0.2, 0.5)
ax2.set_ylim(0.2, 0.5)
ax2.plot(np.array(np.linspace(0,1)), np.array(np.linspace(0,1)),
         color="red", alpha=0.25, label="Optimal predictions")
ax2.legend()
ax2.grid()

plt.show()

##Extreme testing >:3
ex_data = gendist.gen_dataset(300, 1.4)
ex_dataX = pd.DataFrame(ex_data[0]).drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', '
    Th2', 'Th3'])
ex_dataX = np.array(ex_dataX)
ex_dataX = xscale.transform(ex_dataX)

predict_exU = []
actual_exU = []
for i in range(len(ex_dataX)):
    #predict_controlU.append(yscale.inverse_transform(predict_all(m, np.array([control_dataX[i
        ]]))[0][0].reshape(1, -1) +
    #                                                   reg.predict(control_dataX[i].reshape(1,
        -1)))[0])  #If prior adjusted
    predict_exU.append(yscale.inverse_transform(predict_all(m, np.array([ex_dataX[i]]))[0][0].
        reshape(1, -1))[0])# +
                                 #reg.predict(ex_dataX[i].reshape(1,-1)))[0])
    actual_exU.append(ch.optim(ex_data[0][i])['u'])
predict_exU = np.array(predict_exU)
actual_exU = np.array(actual_exU)

##Plotting
fig, (ax1, ax2) = plt.subplots(2)
ax1.scatter(actual_exU[:,0], predict_exU[:,0],
            s=5, label="Predictions")
ax1.set(xlabel= "Actual U1-value", ylabel="Predicted U1-value")
ax1.set_title("U1 Extreme, ttratio=1.4")
ax1.set_xlim(0.2, 0.6)
ax1.set_ylim(0.2, 0.6)
ax1.plot(np.array(np.linspace(0,1)), np.array(np.linspace(0,1)),
         color="red", alpha=0.25, label="Optimal predictions")
ax1.legend()
ax1.grid()

ax2.scatter(actual_exU[:,1], predict_exU[:,1],
            s=5, label="Predictions")
ax2.set(xlabel= "Actual U2-value", ylabel="Predicted U2-value")
ax2.set_title("U2 Extreme, ttratio=1")
ax2.set_xlim(0.2, 0.6)
ax2.set_ylim(0.2, 0.6)
ax2.plot(np.array(np.linspace(0,1)), np.array(np.linspace(0,1)),
         color="red", alpha=0.25, label="Optimal predictions")
ax2.legend()
ax2.grid()

plt.show()
```

B.6 Comparison.py

```
###########################PREFACE##########################
# Code by Michael Lindbak
# For academic masters degree in Chemical Engineering Spring 2021

# Importing tools for the code
import GPy                                               #ML
import numpy as np                                       #Numpy
import matplotlib.pyplot as plt                          #Plotting
from sklearn.preprocessing import MinMaxScaler           #Scaling
import pandas as pd                                      #Data-management
import joblib                                            #Scaler-management
from sklearn.linear_model import LinearRegression        #Prior adjustment
from sklearn.preprocessing import PolynomialFeatures     #Prior adjustment
from sklearn.pipeline import make_pipeline               #Prior adjustmtnt
from sklearn.linear_model import Ridge                   #Prior adjustment
#Import pre-made code
import hex3_gendist as gendist        #Code for generating random data-points
import hex3_chen as ch                #Code for data calculations of the system

# Import prediction code from Supervisor
def predict_all(m: GPy.models.GPCoregionalizedRegression, X):
    ny = len(np.unique(m.output_index))
    y = []
    covy = []
    Xaug = np.hstack((X, 0.0 * np.ones_like(X[:, 0:1])))
    for iy in range(ny):
        Xaug[:, -1:] = iy
        y_i, covy_i = m.predict(Xaug, Y_metadata={'output_index': Xaug[:, -1:].astype(int)})
        y.append(y_i)
        covy.append(covy_i)
    return np.hstack(y), np.hstack(covy)

def plot_hist(predicted, target, title, fp):
    # histogram
    abs_value_error = np.abs(predicted - target)
    print(min(abs_value_error))
    print(max(abs_value_error))
    #plt.title(title)
    plt.yscale('log')
    plt.ylim([0,500])
    plt.hist(abs_value_error, bins=100)
    plt.ylabel('Number of error points [log]')
    plt.xlabel('Temperature difference compared to actual u-values [C]')
    plt.grid()
    plt.savefig(f'figs\\{fp}.pdf')
    plt.show()
    plt.close()

def plot_box(predicted, target, title, fp):
    fig = plt.figure()
    predicted = np.abs(predicted-target)
    ax = fig.add_subplot()
    ax.boxplot([predicted[i] for i in range(len(predicted))], vert=0, whis=(2, 98), sym='+')
    ax.set_yticklabels(['y->d->u*', 'y->u*'])
    #plt.title(title)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
    plt.xlabel("Temperature loss compared to optimal u-values [C]")
    plt.savefig(f'figs\\{fp}.pdf')
    plt.show()

run_types = ['NO PRIOR', 'LINEAR PRIOR', 'QUADRATIC PRIOR']
run_type = run_types[2] #Used to decide what run to do
np.random.seed(420)

#Set up prior re-adjustment flags
prior_adjustment = False
if run_type != 'NO PRIOR':
    prior_adjustment = True


##SETUP y->d
#Prepearing data
x_ttrain = np.genfromtxt(f'ML_parameters\\{run_type}\\x_ttrain.csv', delimiter=',',
    skip_header=1)
y_ttrain = np.genfromtxt(f'ML_parameters\\{run_type}\\y_ttrain.csv', delimiter=',',
    skip_header=1)
if run_type == 'NO PRIOR':
    y_pr = y_ttrain
elif run_type == 'LINEAR PRIOR':
    YD_reg = LinearRegression().fit(x_ttrain, y_ttrain)
    y_pr = y_ttrain - YD_reg.predict(np.array(x_ttrain))
else:
    YD_reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(x_ttrain, y_ttrain)
    y_pr = y_ttrain - YD_reg.predict(np.array(x_ttrain))


Y_pr = np.array(list(zip(*y_pr)))
Y_pr = np.array([i[:, None] for i in Y_pr])

#Setting up ML
k = GPy.kern.RBF(input_dim=8)
icm = GPy.util.multioutput.ICM(input_dim=8, num_outputs=6, kernel=k)
YD = GPy.models.GPCoregionalizedRegression(np.array([x_ttrain]*Y_pr.shape[0]), Y_pr, kernel=
    icm, initialize=False)
YD.update_model(False)
YD.initialize_parameter()
para = np.load(f'ML_parameters\\{run_type}\\YtoD.npy')
YD.param_array[:] = para[:]
YD.update_model(True)
print(YD)
print("\n\n", YD.ICM.B.W, "\n", YD.ICM.B.kappa)
print(YD.ICM.rbf.lengthscale)


##SETUP d->u*
#Prepearinging data
x_strain = np.genfromtxt(f'ML_parameters\\{run_type}\\x_strain.csv', delimiter=',',
    skip_header=1)
y_strain = np.genfromtxt(f'ML_parameters\\{run_type}\\y_strain.csv', delimiter=',',
    skip_header=1)
if run_type == 'NO PRIOR':
    u_pr = y_strain
elif run_type == 'LINEAR PRIOR':
    DU_reg = LinearRegression().fit(x_strain, y_strain)
    u_pr = y_strain - DU_reg.predict(np.array(x_strain))
else:
```

```python
    DU_reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(x_strain, y_strain)
    u_pr = y_strain - DU_reg.predict(np.array(x_strain))

U_pr = np.array(list(zip(*u_pr)))
U_pr = np.array([i[:, None] for i in U_pr])


#Setting up ML
kern = GPy.kern.RBF(input_dim=x_strain.shape[1])
icm2 = GPy.util.multioutput.ICM(input_dim=x_strain.shape[1], num_outputs=U_pr.shape[0], kernel=kern)
DU = GPy.models.GPCoregionalizedRegression(np.array([x_strain]*U_pr.shape[0]), U_pr, kernel=icm2, initialize=False)
DU.update_model(False) #Ignores call of expensive underlaying algebra
DU.initialize_parameter() #Initialize parameters
parr = np.load(f'ML_parameters\\{run_type}\\DtoU.npy')
DU.param_array[:] = parr[:] #Loads all parameters to assigned slots
DU.update_model(True) #Calls the underlying algebra once

##SETUP y->u*
#Prepearing YU data
YU_y = np.genfromtxt(f'ML_parameters\\{run_type}\\x_yutrain.csv', delimiter=',', skip_header=1)
YU_u = np.genfromtxt(f'ML_parameters\\{run_type}\\y_yutrain.csv', delimiter=',', skip_header=1)
if run_type == 'NO PRIOR':
    yu_pr = YU_u
elif run_type == 'LINEAR PRIOR':
    YU_reg = LinearRegression().fit(YU_y, YU_u)
    yu_pr = YU_u - YU_reg.predict(np.array(YU_y))
else:
    YU_reg = make_pipeline(PolynomialFeatures(), Ridge()).fit(YU_y, YU_u)
    yu_pr = YU_u - YU_reg.predict(np.array(YU_y))

YU_pr = np.array(list(zip(*yu_pr)))
YU_pr = np.array([i[:, None] for i in YU_pr])

#Setting up YU ML
YU_k = GPy.kern.RBF(input_dim=YU_y.shape[1])
YU_icm = GPy.util.multioutput.ICM(input_dim=YU_y.shape[1], num_outputs=YU_pr.shape[0], kernel=YU_k)
YU = GPy.models.GPCoregionalizedRegression([YU_y]*YU_pr.shape[0], YU_pr, kernel=YU_icm, initialize=False)
YU.update_model(False)
YU.initialize_parameter()
yu_par = np.load(f'ML_parameters\\{run_type}\\YtoU.npy')
YU.param_array[:] = yu_par[:]
YU.update_model(True)




##GENERATING DATA // TTratio = 1
gen_data = gendist.gen_dataset(500, 1)
y_data = np.array([ch.output_meas(1, [gen_data[1][0][i], gen_data[1][1][i]], gen_data[0][i])['y']
                    for i in range(len(gen_data[0]))])
yscale = joblib.load(f'ML_parameters\\{run_type}\\Masters_xscale.gz')#MinMaxScaler()
dscale = joblib.load(f'ML_parameters\\{run_type}\\Masters_yscale.gz')
uscale = joblib.load(f'ML_parameters\\{run_type}\\yu_uscale.gz')
start_uscale = joblib.load(f'ML_parameters\\{run_type}\\Starting_yscale.gz')
start_dscale = joblib.load(f'ML_parameters\\{run_type}\\Starting_xscale.gz')
y_data = np.array((yscale.transform(y_data)))


##PREDICTING y->d && d->u
d_pred = []
u_pred = []
u_actual = []
u_comp = []     #Comparing the y->d->u* model to y->u*
d_actual = []

for i in range(len(y_data)):
    if prior_adjustment:
        d_pred.append(np.array(predict_all(YD, np.array([y_data[i]]))[0][0] +
                                YD_reg.predict(y_data[i].reshape(1, -1))[0]))
        u_pred.append(uscale.inverse_transform(predict_all(DU, np.array([d_pred[i]]))[0][0].reshape(1, -1) +
                                DU_reg.predict(np.array(d_pred[i]).reshape(1, -1)))[0])
        u_comp.append(uscale.inverse_transform(predict_all(YU, np.array([y_data[i]]))[0][0].reshape(1, -1) +
                                YU_reg.predict(y_data[i].reshape(1, -1)))[0])
    else:
        d_pred.append(np.array(predict_all(YD, np.array([y_data[i]]))[0][0]))
        #d_pred.append(np.array(dscale.inverse_transform(predict_all(YD, np.array([y_data[i]]))[0][0].reshape(1, -1)[0])))  #If prior was adjusted
        u_pred.append(uscale.inverse_transform(predict_all(DU, np.array([d_pred[i]]))[0][0].reshape(1, -1))[0])
        u_comp.append(uscale.inverse_transform(predict_all(YU, np.array([y_data[i]]))[0][0].
            reshape(1, -1))[0])

    u_actual.append(ch.optim(gen_data[0][i])['u'])


#Used to unscale d_pred
d_pred = np.array([dscale.inverse_transform(d_pred[i].reshape(1, -1))[0] for i in range(len(
    d_pred))])
u_pred = np.array(u_pred)
u_actual = np.array(u_actual)
u_comp = np.array(u_comp)
d_actual = pd.DataFrame(gen_data[0])
d_actual = d_actual.drop(columns=['Ts', 'h1', 'h2', 'h3', 'T0', 'w0', 'Th1', 'Th2', 'Th3'])
d_actual = np.array(d_actual)



#Plotting U prediction vs Real
fig, (ax1, ax2) = plt.subplots(2)
ax1.scatter(u_pred[:,0], u_actual[:,0], label = "U1 prediction", s=4)
ax1.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.6, color='
    red')
ax1.set_xlim(0.2, 0.5)
ax1.set_ylim(0.2, 0.5)
ax1.legend()
#ax1.set_xlabel("Predicted valve value [-]")    #Removed for visual clarity
ax1.set_ylabel("Actual valve value [-]")

ax2.scatter(u_pred[:,1], u_actual[:,1], label = "U2 prediction", alpha=0.6, color='orangered',
     s=4)
ax2.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.6, color='
    red')
ax2.legend()
ax2.set_xlim(0.2, 0.5)
ax2.set_ylim(0.2, 0.5)
ax2.set_xlabel("Predicted valve value [-]")
ax2.set_ylabel("Actual valve value [-]")
plt.savefig(f'figs/U-predictions vs actual_{run_type}.pdf')

plt.show()


#Plotting temperature difference
T_pred = []
T_actual = []
T_comp = []

for i in range(len(y_data)):
    T_pred.append(ch.output(u_pred[i], gen_data[0][i])['x'][1])
    T_actual.append(ch.output(u_actual[i], gen_data[0][i])['x'][1])
    T_comp.append(ch.output(u_comp[i], gen_data[0][i])['x'][1])
T_pred = np.array(T_pred)
T_actual = np.array(T_actual)
T_comp = np.array(T_comp)

plot_hist(T_pred, T_actual, "Temperature difference y->d->u* \n ttratio = 1.0", f"ydu
    histogram_{run_type}")
plot_hist(T_comp, T_actual, 'Temperature difference y->u* \n ttratio = 1.0', f'yu histogram_{
    run_type}')
plot_box([T_pred, T_comp], T_actual, 'Temperatures in box-plot, ttratio = 1.0', f"Boxplot_{
    run_type}")
pd.DataFrame(T_pred).to_csv(f'T_pred_{run_type}.csv', index=False)
pd.DataFrame(T_comp).to_csv(f'T_comp_{run_type}.csv', index=False)
pd.DataFrame(T_actual).to_csv(f'T_actual_{run_type}.csv', index=False)

##GENERATING DATA ttratio = 1.2
gen_dataEX = gendist.gen_dataset(500, 1.3)
y_dataEX = np.array([ch.output_meas(1, [gen_dataEX[1][0][i], gen_dataEX[1][1][i]], gen_dataEX
    [0][i])['y']
                    for i in range(len(gen_dataEX[0]))])
#yscale = joblib.load('Masters_xscale.gz')#MinMaxScaler()
#dscale = joblib.load('Masters_yscale.gz')
y_dataEX = np.array((yscale.transform(y_dataEX)))

##PREDICTING y->d && d->u
d_predEX = []
u_predEX = []
u_actualEX = []
u_compEX = []
for i in range(len(y_dataEX)):
    if prior_adjustment:
        d_predEX.append(np.array(predict_all(YD, np.array([y_dataEX[i]]))[0][0] +
                        YD_reg.predict(y_dataEX[i].reshape(1, -1))[0]))
        u_predEX.append(uscale.inverse_transform(predict_all(DU, np.array([d_predEX[i]]))
            [0][0].reshape(1, -1) +
                         DU_reg.predict(np.array(d_predEX[i]).reshape(1, -1)))[0])
        u_compEX.append(uscale.inverse_transform(predict_all(YU, np.array([y_dataEX[i]]))
            [0][0].reshape(1, -1) +
                        YU_reg.predict(y_dataEX[i].reshape(1, -1)))[0])
    else:
        d_predEX.append(np.array(predict_all(YD, np.array([y_dataEX[i]]))[0][0]))
        u_predEX.append(uscale.inverse_transform(predict_all(DU, np.array([d_predEX[i]]))
            [0][0].reshape(1, -1))[0])
        u_compEX.append(uscale.inverse_transform(predict_all(YU, np.array([y_dataEX[i]]))
            [0][0].reshape(1, -1))[0])

    u_actualEX.append(ch.optim(gen_dataEX[0][i])['u'])


u_predEX = np.array(u_predEX)
u_actualEX = np.array(u_actualEX)
u_compEX = np.array(u_compEX)



#Plotting U prediction vs Real
fig2, (ax1, ax2) = plt.subplots(2)
ax1.scatter(u_predEX[:,0], u_actualEX[:,0], label = "U1 prediction", s=4)
ax1.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.6, color='
    red')
ax1.set_xlim(0.2, 0.6)
ax1.set_ylim(0.2, 0.6)
ax1.legend()
#ax1.set_xlabel("Predicted valve value [-]")   #Removed for visual clarity
ax1.set_ylabel("Actual valve value[-]")

ax2.scatter(u_predEX[:,1], u_actualEX[:,1], label = "U2 prediction", alpha=0.6, color='
    orangered', s=4)
ax2.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.6, color='
    red')
ax2.legend()
ax2.set_xlim(0.2, 0.6)
ax2.set_ylim(0.2, 0.6)
ax2.set_xlabel("Predicted valve value [-]")
ax2.set_ylabel("Actual valve value [-]")
plt.savefig(f'figs/U-predictions vs actualEX_{run_type}.pdf')

plt.show()

#Plotting temperature difference
T_predEX = []
T_actualEX = []
T_compEX = []

for i in range(len(y_data)):
    T_predEX.append(ch.output(u_predEX[i], gen_dataEX[0][i])['x'][1])
    T_actualEX.append(ch.output(u_actualEX[i], gen_dataEX[0][i])['x'][1])
    T_compEX.append(ch.output(u_compEX[i], gen_dataEX[0][i])['x'][1])
T_predEX = np.array(T_predEX)
T_actualEX = np.array(T_actualEX)
T_compEX = np.array(T_compEX)
pd.DataFrame(T_predEX).to_csv(f'T_predEX_{run_type}.csv', index=False)
pd.DataFrame(T_compEX).to_csv(f'T_compEX_{run_type}.csv', index=False)
pd.DataFrame(T_actualEX).to_csv(f'T_actualEX_{run_type}.csv', index=False)

plot_hist(T_predEX, T_actualEX, "Temperature difference y->d->u* \n ttratio = 1.2", f"ydu
    histogramEX_{run_type}")
plot_hist(T_compEX, T_actualEX, 'Temperature difference y->u* \n ttratio = 1.2', f'yu
    histogramEX_{run_type}')
plot_box([T_predEX, T_compEX], T_actualEX, 'Boxplot, ttratio = 1.2', f"BoxplotEX_{run_type}")

#Plotting comparison graphs between YD-DU and YU
fig3, (ax1, ax2) = plt.subplots(2)
ax1.scatter(u_comp[:,0], u_actual[:,0], label="U1 predictions", s=5, color="tab:blue")
ax1.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.7, color="
    red")
ax1.set_xlim(0.2, 0.5)
ax1.set_ylim(0.2, 0.5)
ax1.legend()
#ax1.set_xlabel("Predicted valve value [-]")   #Removed for visual clarity
ax1.set_ylabel("Actual valve value [-]")

ax2.scatter(u_comp[:,1], u_actual[:,1], label="U2 predictions", s=5, alpha=0.8, color='m')
ax2.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.7, color="
    red")
#ax2.grid()
ax2.set_xlim(0.2, 0.5)
ax2.set_ylim(0.2, 0.5)
ax2.legend()
ax2.set_xlabel("Predicted valve value [-]")
ax2.set_ylabel("Actual valve value [-]")
plt.savefig(f'figs/YU-predictions vs actual_{run_type}.pdf')

plt.show()

#Plotting EX comparisons

```

```
fig4, (ax1, ax2) = plt.subplots(2)
ax1.scatter(u_compEX[:,0], u_actualEX[:,0], label="U1 predictions", s=5, color='tab:blue')
ax1.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.7, color="
    red")
ax1.set_xlim(0.2, 0.6)
ax1.set_ylim(0.2, 0.6)
ax1.legend()
#ax1.set_xlabel("Predicted valve value [-]")    #Removed for visual clarity
ax1.set_ylabel("Actual valve value [-]")

ax2.scatter(u_compEX[:,1], u_actualEX[:,1], label="U2 predictions", s=5, alpha=0.8, color='m')
ax2.plot(np.linspace(0,1), np.linspace(0,1), label="Optimal predictions", alpha=0.7, color="
    red")
ax2.set_xlim(0.2, 0.6)
ax2.set_ylim(0.2, 0.6)
ax2.legend()
ax2.set_xlabel("Predicted valve value [-]")
ax2.set_ylabel("Actual valve value [-]")
plt.savefig(f'figs/YU-predictions vs actualEX_{run_type}.pdf')

plt.show()

##Evaluating d-pred efficacy
fig5, (ax1, ax2) = plt.subplots(2)
ax1.scatter(d_pred[:,0], d_actual[:,0], s=5, label="wh1 predictions")
ax1.scatter(d_pred[:,1], d_actual[:,1], s=5, label="wh2 predictions")
ax1.scatter(d_pred[:,2], d_actual[:,2], s=5, label="wh3 predictions")
ax1.set(xlabel= "Actual wh-values [kW/K]", ylabel="Predicted wh-values [kW/K]")
ax1.set_xlim(20, 60)
ax1.set_ylim(20, 60)
ax1.plot(np.linspace(20,60), np.linspace(20,60),
         color="red", alpha=0.6, label="Optimal predictions")
ax1.legend()
ax1.grid()


ax2.scatter(d_pred[:,3], d_actual[:,3], s=5, label="UA1 predictions")
ax2.scatter(d_pred[:,4], d_actual[:,4], s=5, label="UA2 predictions")
ax2.scatter(d_pred[:,5], d_actual[:,5], s=5, label="UA3 predictions")
ax2.set(xlabel="Actual UA-values [kW/K]", ylabel="Predicted UA-values [kW/K]")
ax2.set_xlim(50,110)
ax2.set_ylim(50,110)
ax2.plot(np.linspace(50,110), np.linspace(50,110),
         color="red", alpha=0.6, label="Optimal predictions")
ax2.legend()
ax2.grid()
plt.tight_layout()
plt.savefig(f'figs/d-predictions_{run_type}.pdf', dpi=400)

plt.show()

##Evaluating the outlier datapoints
abs_value_error = np.abs(T_pred-T_actual)
weird_spot = np.where(abs_value_error == max(abs_value_error))[0][0]
next_spot = np.where(abs_value_error == np.sort(abs_value_error)[-2])[0][0]
control_spot = np.where(abs_value_error == np.sort(abs_value_error)[0])[0][0]
print("D-predictions for max T-difference: ", d_pred[weird_spot], #dscale.inverse_transform(
    d_pred[weird_spot].reshape(1,-1))[0],
      "\nCompared to actual d-values of: ", d_actual[weird_spot])
print("\nWhich was the following u*: ", u_pred[weird_spot], "\nCompared to actual u of: ",
    u_actual[weird_spot])

print("\n\nD-predictions for second max T-difference: ", d_pred[next_spot], #dscale.
    inverse_transform(d_pred[next_spot].reshape(1,-1))[0],
      "\nCompared to actual d-values of: ", d_actual[next_spot])
print("\nWhich was the following u*: ", u_pred[next_spot], "\nCompared to the actual u of: ",
    u_actual[next_spot])

print("\n\nD-predictions for min T-difference: ", d_pred[control_spot], #dscale.
    inverse_transform(d_pred[next_spot].reshape(1,-1))[0],
      "\nCompared to actual d-values of: ", d_actual[control_spot])
print("\nWhich was the following u*: ", u_pred[control_spot], "\nCompared to the actual u of:
    ", u_actual[next_spot])


print("u1 pred ex max: ", max(u_predEX[:,0]), "u2 pred ex max: ", max(u_predEX[:,1]),
      "\nu1 comp max: ", max(u_compEX[:,0]), "u2 comp max: ", max(u_compEX[:,1]))
print("\n\nu1 pred ex #2: ", np.sort(u_predEX[:,0])[-2], "u2 pred ex #2: ", np.sort(u_predEX
    [:,1])[-2],
      "\nu1 comp #2: ", np.sort(u_compEX[:,0])[-2], "u2 comp #2: ", np.sort(u_compEX[:,1])
          [-2])
```

B.7 Boxplot comparisons.py

```
#Lazily made code for creating comparison boxplots for the approaches
import GPy
import numpy as np
import matplotlib.pyplot as plt                    #Plotting

def predict_all(m: GPy.models.GPCoregionalizedRegression, X):
    ny = len(np.unique(m.output_index))
    y = []
    covy = []
    Xaug = np.hstack((X, 0.0 * np.ones_like(X[:, 0:1])))
    for iy in range(ny):
        Xaug[:, -1:] = iy
        y_i, covy_i = m.predict(Xaug, Y_metadata={'output_index': Xaug[:, -1:].astype(int)})
        y.append(y_i)
        covy.append(covy_i)
    return np.hstack(y), np.hstack(covy)

def plot_box(predicted, target, title, fp):
    fig = plt.figure()
    predicted = np.abs(predicted-target)
    ax = fig.add_subplot()
    b = ax.boxplot([predicted[i] for i in range(len(predicted))], vert=0, whis=(2, 98), sym='+
        ')
    ax.set_yticklabels(['Default \nPrior  ', 'Linear \nPrior  ', 'Quadratic \nPrior    '],
        va='center')
    #plt.title(title)
    plt.xlim([0, 2])
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
    plt.xlabel("Temperature loss compared to optimal u-values [C]")
    plt.yticks(rotation=50)
    plt.savefig(f'figs\\{fp}.pdf')

    plt.show()
run_types = ['EX', ""]  #Either expanded testing region or default
run_type = run_types[1] #Pick 0 or 1 depending on desired run-type


T_pred = np.genfromtxt(f'T_pred{run_type}_NO PRIOR.csv', delimiter=',', skip_header=1)
T_pred_2 = np.genfromtxt(f'T_pred{run_type}_LINEAR PRIOR.csv', delimiter=',', skip_header=1)
T_pred_3 = np.genfromtxt(f'T_pred{run_type}_QUADRATIC PRIOR.csv', delimiter=',', skip_header
    =1)
T_comp = np.genfromtxt(f'T_comp{run_type}_NO PRIOR.csv', delimiter=',', skip_header=1)
T_comp_2 = np.genfromtxt(f'T_comp{run_type}_LINEAR PRIOR.csv', delimiter=',', skip_header=1)
T_comp_3 = np.genfromtxt(f'T_comp{run_type}_QUADRATIC PRIOR.csv', delimiter=',', skip_header
    =1)
T_actual = np.genfromtxt(f'T_actual{run_type}_NO PRIOR.csv', delimiter=',', skip_header=1)
T_actual_2 = np.genfromtxt(f'T_actual{run_type}_LINEAR PRIOR.csv', delimiter=',', skip_header
    =1)
T_actual_3 = np.genfromtxt(f'T_actual{run_type}_QUADRATIC PRIOR.csv', delimiter=',',
    skip_header=1)
T_tot = T_actual
np.append(T_tot, T_actual_2)
np.append(T_tot, T_actual_3)

plot_box([T_pred, T_pred_2, T_pred_3], T_tot, 'redundant title', f'comp_boxplots{run_type}_YDU
    ')
plot_box([T_comp, T_comp_2, T_comp_3], T_tot, 'redundant title', f'comp_boxplots{run_type}_YU'
    )
```