Felix Julian Lechleitner, BSc

Data driven modeling of highly cross-linked production plants

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

submitted to

Graz University of Technology

Supervisors

Assoc.Prof. Dipl.-Ing. Dr.techn. Thomas Wallek
Dipl.-Ing. Fabian Zapf, BSc

Institute of Chemical Engineering and Environmental Technology

Graz, November 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Kurzfassung

In dieser Arbeit wurde eine sequenziell-modulare Simulationsumgebung in Wolfram Mathematica in Form eines Packages implementiert. Zu den wesentlichen Eigenschaften der Anwendung zählen unter anderem eine grafische Benutzeroberfläche sowie eine automatisierte Handhabung von im Kreis geführten Materialströmen. Desweiteren wurde während der Entwicklung darauf geachtet, die Implementierung neuer verfahrenstechnischer Operationen so einfach wie möglich zu gestalten. Die Verwendung von künstlichen neuronalen Netzwerken in *grey-box*-Modellen für verfahrenstechnische Anwendungen wurde ausführlich getestet. Es wurde festgestellt, dass die Leistung dieser Modelle stark von der Qualität der zugrunde liegenden Daten abhängt. Als alternative Herangehensweise wurden Standard Gauß-Prozesse verwendet. Diese leiden jedoch unter einem hohen Rechenaufwand, sowie unter extrem hohen Speicheranforderungen (sowohl für den Trainingsprozess als auch für das Speichern der Modelle). Die Simulationsumgebung wurde anhand des Beispiels einer Entspannungstrommel mit rückgeführter Gasphase, und an einer Anlage zur Produktion von Ethyl-tert-butylether (ETBE) getestet. Die Anlagen aus den Beispielen wurden in PetroSIM modelliert, und die Ergebnisse stimmen gut mit jenen der entwickelten Simulationsumgebung überein. Neue Methoden zur Modellierung der Ungewissheit von Vorhersagen neuronaler Netzwerke wurden mit Messdaten einer realen Anlage getestet. Für kleine Datensätze entspricht der Einfluss der Hyperparameter auf die resultierende Vorhersage dem der Literatur. Dies ist nicht der Fall für große, verrauschte Datensätze.

Abstract

In this work, a sequential-modular simulation framework was implemented in Wolfram Mathematica in form of a package. This package features a graphical user interface, automated handling of recycle streams and was designed to make the implementation of new unit operations as easy as possible. In-depth testing of artificial neural networks in grey-box models for chemical engineering applications was conducted. It was found that the performance of these models strongly depends on the quality of the underlying data. Standard Gaussian processes were tried as an alternative approach, but they suffer from high computational effort as well as extremely high memory requirements (for training and storing the models). The simulation framework was tested on a flash drum with a recycle and on an Ethyl tert-butyl ether (ETBE) production plant. The same plants were modelled in PetroSIM, and the results are in good agreement with those of the simulation framework developed in this work. Novel methods for modelling uncertainty with artificial neural networks were tested using real plant data. For small datasets, the effect of the hyperparameters on the resulting prediction is in good agreement with the literature. This is not the case for large datasets which feature a lot of noise.

Contents

1 Introduction

Data-driven modelling of process plants is an emerging discipline in the field of process simulation. The underlying motivation is the developement of new methods, which produce predictions for the performance of single units and whole production plants that are closer to reality than those made by classical, theoretical models.

In this work, the possibility of utilizing machine learning methods with real-world measurement data for process simulation is explored. This approach aims to be an improvement over existing mixed-integer linear programming (MILP) modelling methods. This work focuses on artificial neural networks (ANN) to build the models, but Gaussian Processes (GP) are also adressed.

A sequential modular process simulation framework has been built in Wolfram Mathematica. Notable features include a fully automated approach for handling recycled streams, as well as a simple graphical user interface for processing user input. The procedure for introducing new units and unit functions into the framework has been kept simple to facilitate further developement on the application.

Also, novel approaches for modelling and quantifying uncertainties with artificial neural networks using homo- and heteroscedastic approaches have been implemented and tested on real plant data.

An interface between the simulation framework and an genetic algorithm for optimization, namely the nondominated sorting genetic algorithm II (NSGA-II), has been implemented. The genetic algorithm was used to maximize the monetary feasibility while minimizing the environmental impact of a sub-part of a real oil refinery.

The main focus of this work is to provide a robust simulation framework and to test the implementation of artificial neural networks in grey-box unit operations.

2 Theory

This chapter aims at explaining the concepts and methods used in this work. Because some of these concepts have their roots in the fields of mathematics or computer science, they will be explained in more detail.

2.1 Flowsheet Solver

The term flowsheet solver refers to a framework for solving process engineering simulation problems. It provides methods and routines for reading and processing user input, building a (simulation) flowsheet, and stepping through and solving the flowsheet. There are two main types of flowsheet solving techniques:

Equation oriented (EO): With this approach, every unit is basically represented as a set of equations. The solver then builds a large system of equations from all the units and tries to solve it. Depending on the system in question, this approach is susceptible to stability/convergence problems or requires a large computational effort. [1]

Sequential modular (SM): With this approach, every type of unit includes a sub-model and a procedure for solving said model. Units are evaluated one after the other, passing their output to the next unit, hence the adjective sequential. The solvers' main purpose in this case is to determine the order of evaluation for the single units. If recycle streams are present, the solver enters a loop until convergence is reached. [1]

In this work, the sequential modular approach is implemented.

2.2 Data-driven modeling

2.2.1 Types of models

There are three different basic types of models: black box models, white box models and grey box models.

Black box models do not contain any information about the underlying mechanisms which are used to obtain the outputs for the supplied inputs. Black box models are always based on data, and the model itself and its parameter have no physical significance. Examples

for black box models are artificial neural networks, support-vector machines, Gaussian processes or linear regression. [2]

In white box models (or glass box models), the behaviour of the system, how predictions are made and how all the different variables affect the outcome is clear. White box models are deterministic, and are modeled entirely based on physical principles. [3] There are many examples for white box models in process engineering, as a lot of calculations in this field are based on ordinary or partial differential equation which are deterministic, rather than stochastic. An example for that is the calculation of the conversion along a plug-flow reactor.

Grey box models are a combination of white and black box models, which means some part of the system is modeled by a data-driven approach, whereas the other part is modelled by applying expert knowledge about the process in the form of equations [4]. An example for a grey-box model would be a neural network used to estimate the K-values for a flash calculation. The mass balance along with the summation condition are then used to calculate the liquid and vapor fractions.

In this work, all models are either white or grey box models. For simple units which are not relying on any estimated parameters, like a mixing unit, a white box approach is used. Units which use neural networks usually have their output modified in some way to guarantee conservation of mass.

2.2.2 Data preprocessing

During data preprocessing, the raw measurement data taken from the refinery, which is described in section 4.5, is filtered for steady-state, outliers are removed and the data is fixed to fulfill the mass balance for the single units. The last step is the sampling of the data into a training and validation dataset.

The preprocessing is important, because bad data with a lot of noise negatively affect the prediction capability of the final model. [5]

2.2.2.1 Filter

The first preprocessing step is the application of a data filter. Measurement data from timeframes where the plant is starting up, shutting down or already shut down are not beneficial to model training process, because the aim is to model normal and steady state operation.
To get usable and meaningful data a min-/max filter and a filter based on the relative change between data points over time is used. The first one can be used to utilize knowledge about the process in an effective way. For example, if it is known that a certain temperature in a reactor is needed for normal operation, all data points where the temperature is below

that threshold are removed. The second filter removes data points based on the relative change between consecutive measurements. The threshold for this value was usually set between 5 % and 20 %, depending of the quality of the data. Filtering too strictly can shrink the size of the dataset significantly, thus a trade-off between data quality and data quantity has to be chosen.

2.2.2.2 Outliers

To identify outliers, the **Lo**cal **O**utlier **P**robabilites algorithm, in short LoOP, by Kriegel et al. [6] is used. The algorithm combines local, density-based outlier scores with probabilistic concepts. Whereas other outlier algorithms result in a binary label (outlier or no outlier) or an outlier score, this algorithm gives a probability (between 0 and 1) for a data point to be an outlier.

Let $\mathcal{D}$ be a set of n objects and $d(p_1, p_2)$ a function to determine the Euclidian distance between the objects p_1 and p_2. Furthermore, let $S(x)$ be a context set $S \subseteq \mathcal{D}$ which contains the k nearest neighbors of x. Assuming that x is the center of S and the set of distances of $s \in S$ to x is approximately a folded normal distribution, the standard distance σ can be calculated as follows [6]:

$$\sigma(x, S) = \sqrt{\frac{\sum_{s \in S} d(x, s)^2}{|S|}} = \sqrt{\frac{\sum_{s \in S} d(x, s)^2}{k}} \tag{2.1}$$

$|S|$ is the cardinality of the set S (i.e. the number of elements in the set) and is equal to k. The *probabilistic set distance* pdist of x to S is then defined as: [6]

$$\text{pdist}(\lambda, x, S) = \lambda \cdot \sigma(x, S) \tag{2.2}$$

The parameter λ is a normalization factor and gives control over the aproximation of the density. It affects the contrast between the resulting outlier probabilites but does not change the ranking. [6]

Using pdist, the *Probabilistic Local Outlier Factor* PLOF with respect to a significance λ and a context set S can be calculated as follows: [6]

$$\text{PLOF}_{\lambda,S}(x) = \frac{\text{pdist}(\lambda, x, S(x))}{E_{s \in S(x)}[\text{pdist}(\lambda, x, S(s))]} - 1 \tag{2.3}$$

The PLOF value of an object x is the ratio of the estimation for the density around x and the expected value (denoted by the symbol E, which means the mean value in this context) of the estimation for the densities around all objects in the context set $S(x)$. A PLOF

value below 0 is not an outlier, whereas higher values indicate an increasing chance for the object to be an outlier. To get to a probability, the PLOF value needs to be normalized. For normalization, the aggregate value nPLOF is used: [6]

$$\text{nPLOF} = \lambda \cdot \sqrt{E\left[(\text{PLOF})^2\right]} \tag{2.4}$$

Again, the E operator stands for the mean value. So $E\left[(\text{PLOF})^2\right]$ is the mean of the squared PLOF values of all objects in $\mathcal{D}$.

Finally, the *Local Outlier Probability* LoOP can be calculated as follows, where erf is the Gaussian error function: [6]

$$\text{LoOP}_S(x) = \max\left\{0, \text{erf}\left(\frac{\text{PLOF}_{\lambda,S}(x)}{\text{nPLOF} \cdot \sqrt{2}}\right)\right\} \tag{2.5}$$

Figure 2.1 shows a two-dimensional dataset (Fisher's Iris data set [7]) with the local outlier probabilities for points where LoOP ≥ 0.26.

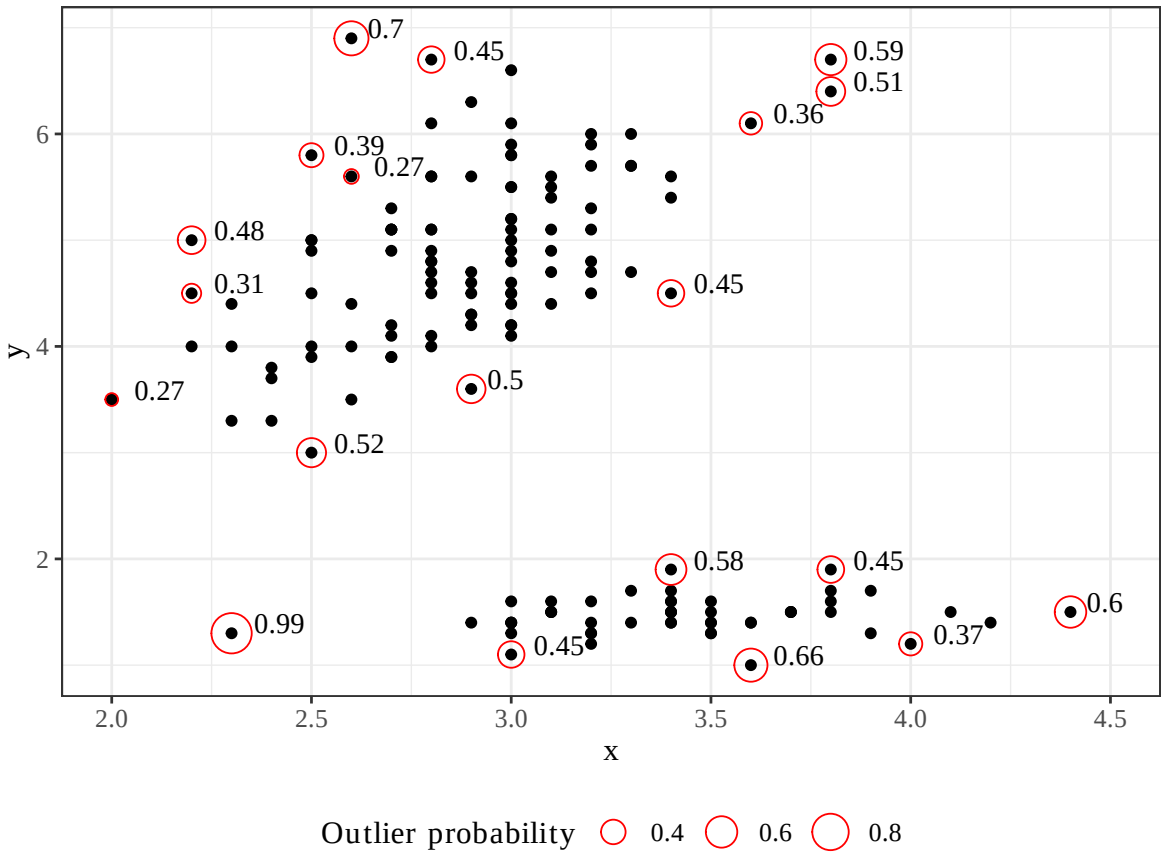


Figure 2.1: Visualization of local outlier probabilities

In this work, the parameters $k = 10$ and $\lambda = 3$ have been used for outlier detection. Points with a local outlier probability above 20 % have been removed, which usually excluded about 2 % to 5 % of the dataset after filtering.

2.2.2.3 Data reconciliation

Data reconciliation is a two-step process where the mass balance for a unit is enforced, because most of the time there is some sort of mass defect when dealing with real-life and real-time measurements [8]. The first step is to define which streams are input, and which are output streams. Based on this information, a relative error in the mass balance with respect to the input can be calculated for every datapoint. The modal value in this relative error distribution is determined, and data points below and above a certain threshold (5 %, 10 % or 15 % for most cases) are removed. See figure 2.2 for a visual representation.

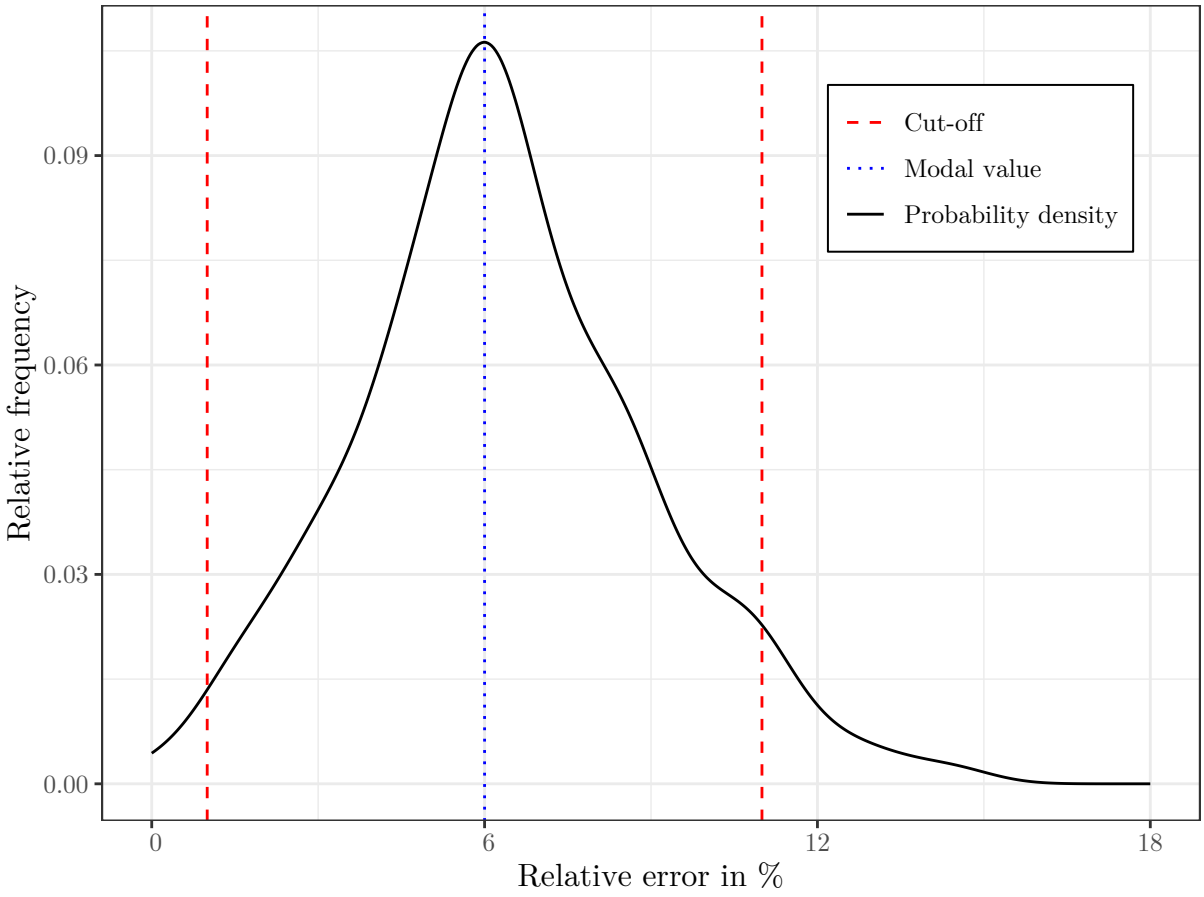


Figure 2.2: Visualization of the mass balance filter

The blue, dotted line marks the modal value, and the red, dashed lines mark the cut-off

points.

After filtering the data, a weighted approach is used to scale the input and output streams so that the mass defect is eliminated.

2.2.2.4 Sampling

During data sampling, the data are divided into a training data set and a validation training set. There are a lot of methods for data sampling, and the best choice is usually depending on the nature of the problem in question. Commonly used data sampling techniques include random sampling, stratified sampling [9] (with or without Neymann allocation [10]) and Editted or Condensed Nearest Neighbor sampling [11]. The data sampling process is often called instance selection in related literature.

Previous investigations of different instance selection methods at TU Graz have shown that a combination of convex hull and self information based sampling yields the best results.

2.2.2.4.1 Convex Hull The motivation behind the usage of a convex hull algorithm is to ensure that the training data set encloses the whole range of the data in all dimensions [12]. Artificial neural networks are not suited for extrapolation [13], which means reliable predictions can only be made for input data that falls inside the range of the training data set.

Figure 2.3 shows an extreme example for bad (random) sampling. 50 % of the available data should be used for training, where the randomly sampled data points cover only a fraction of the available data. The remaining data points are used for validation. As expected, the neural network fails to correctly extrapolate the data, at least for predictions far away from the training data. For small x values, the prediction is acceptable even though the net is extrapolating, most likely because of the high density of validation data points in this area.

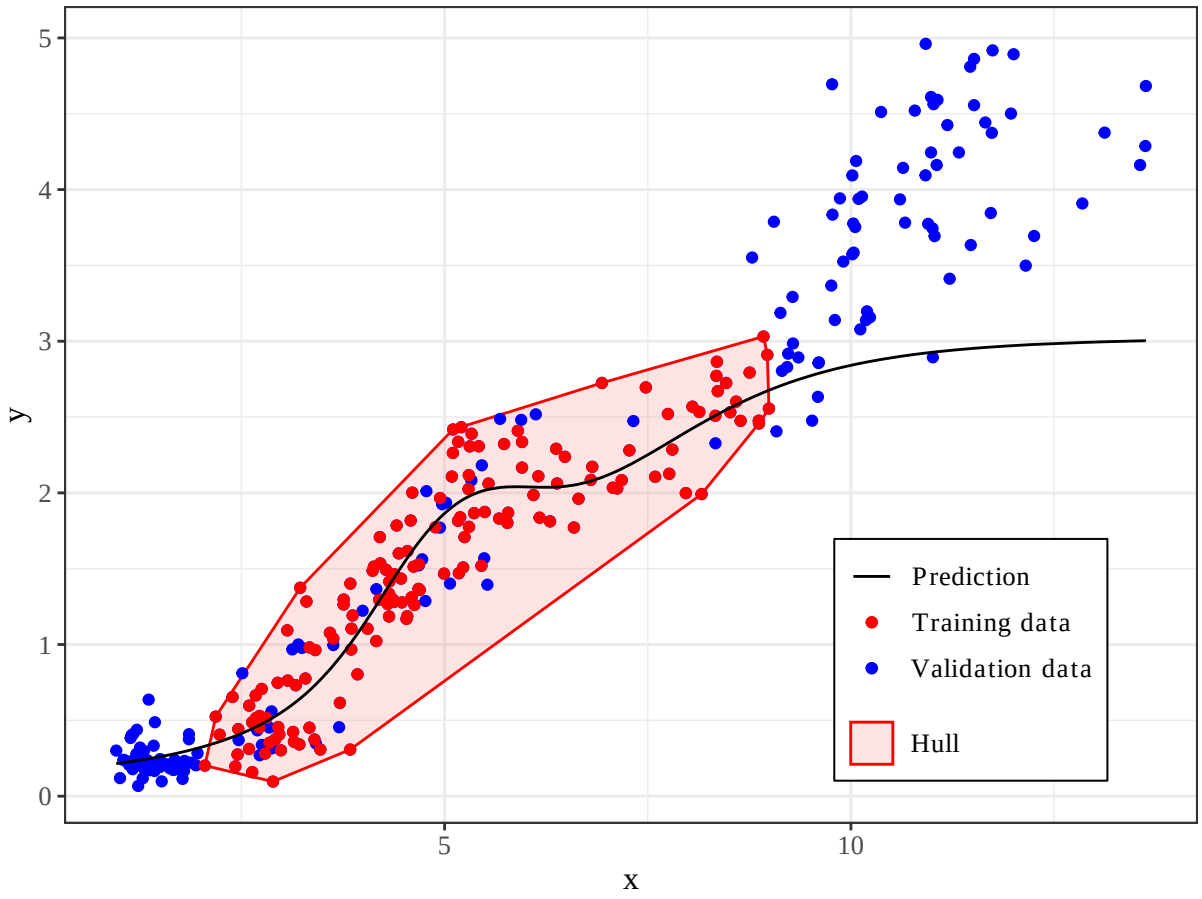


Figure 2.3: Example for bad sampling - the red data points have been sampled for training

In figure 2.4, a combination of random sampling and a convex hull algorithm has been used. As the training data encloses all available data points, the neural network can make reliable predictions across the entire data range.

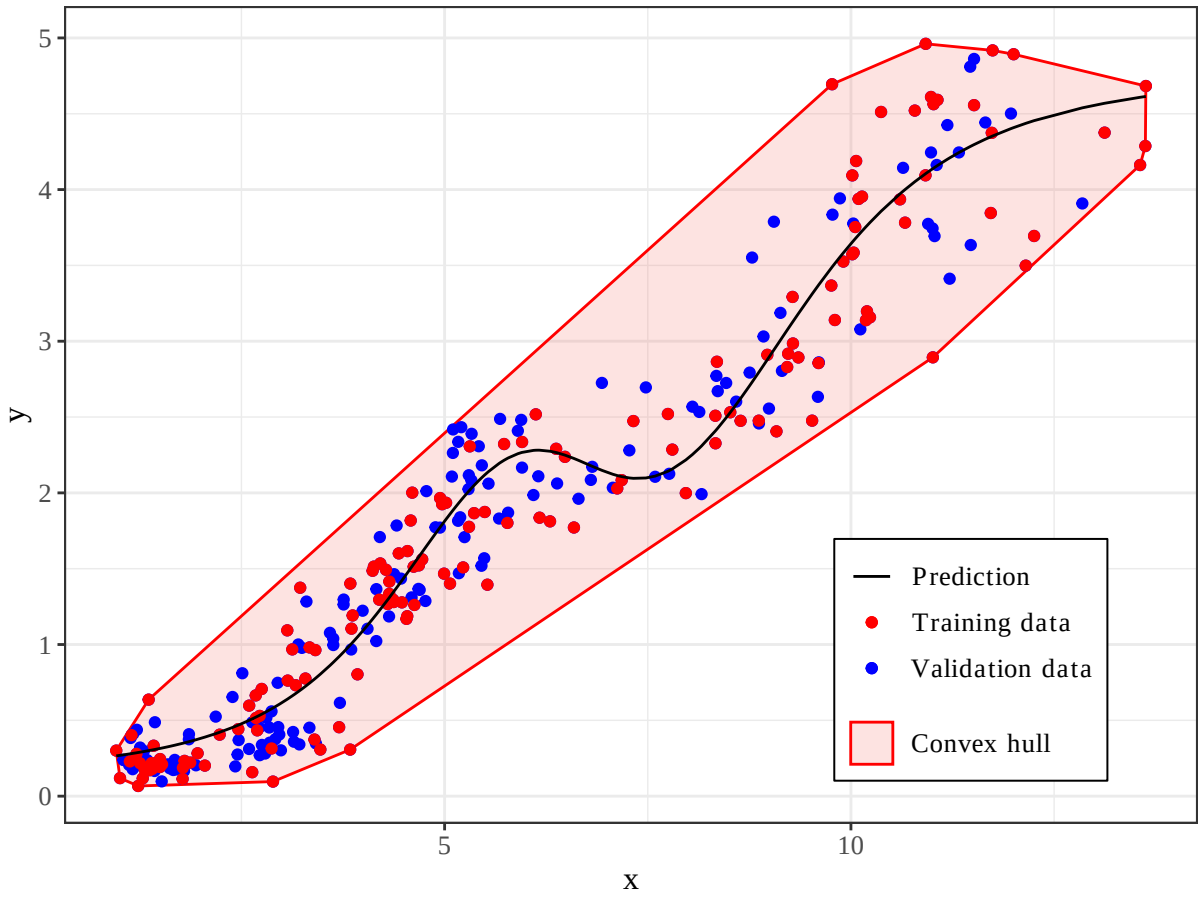


Figure 2.4: Example for good sampling - the red data points have been sampled for training

There are many different algorithms for computing the convex hull of a dataset [14]. This work uses an implementation of the quickhull algorithm by Barber et al. [15] in C. Although Mathematica features a built-in function for computing convex hulls, it is not usable for large datasets because of extremely long computation times. Thus, a low-level implementation has been chosen.

The time complexity for two- and three-dimensional[1] datasets is $\mathcal{O}(n \log v)$, where n is the number of input points and v the number of output vertices. For higher dimensions, the time complexity is $\mathcal{O}(nf_v/v)$, where f_v is the maximum number of facets for a convex hull of v vertices. The function f_v increases rapidly with growing dimensions, it is $\mathcal{O}(v^{\lfloor d/2 \rfloor}/\lfloor d/2 \rfloor !)$. Furthermore, the memory requirements rise rapidly with growing dimension of the dataset. [15]

As a result, the application of the complex hull for datasets with many variables becomes impractical, either because of very long computation times or the computer running

[1] each variable in the dataset adds another dimension

out of available memory. In this work, this has been the case for the modelling of the ethylene cracking furnaces, because the corresponding datasets consist of about 25 variables, depending on the type of furnace.

2.2.2.4.2 Self-information-based subset selection The method of self-information-based subset selection (SIBSS) has been developed by Zapf et al. [16] and is a variation of the entropy-based subset selection method (EBSS) proposed by Ferreira et al. [17].

The EBSS method utilizes Shannon's entropy of information [18] to conduct a weighted random data selection. First, the contribution of each data point to the overall entropy is calculated. A data point can be expressed as a vector z_i, containing all variables.

$$z_i = [x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,d}]^T \tag{2.6}$$

The probability density $p(z)$ is defined as follows [19]:

$$p(z) = \frac{1}{N} \cdot \sum_{i=1}^{N} \left[\prod_{j=1}^{d} K_{h_j}(z_j - z_{i,j}) \right] \tag{2.7}$$

Where $K_{h_j}(x)$ is the Gaussian kernel function:

$$K_{h_j}(x) = \frac{1}{\sqrt{2\pi h_j^2}} \cdot \exp\left[-\frac{x^2}{2h_j^2} \right] \tag{2.8}$$

h_j is the bandwidth parameter, which takes the different scaling of the data in each dimension into consideration. The parameter can be estimated using the mean and standard deviation for a dimension j [19]:

$$\overline{z_j} = \frac{1}{N} \cdot \sum_{i=1}^{N} z_{i,j} \tag{2.9}$$

$$\sigma_j = \sqrt{\frac{1}{N-1} \cdot \sum_{i=1}^{N} \left(z_{i,j} - \overline{z_j}\right)^2} \tag{2.10}$$

$$h_j = \sigma_j \cdot N^{\left(-\frac{1}{d+4}\right)} \tag{2.11}$$

For SIBSS, the self-information $I(z)$ is used as a weighting function [16]:

$$I(z) = -\log_2(p(z)) \tag{2.12}$$

For the actual sampling, the Stochastic Universal Sampling algorithm (SUS), also known as roulette wheel selection, is used. As mentioned before, this is a weighted, random selection process, which is commonly used in genetic algorithms. Every data point occupies a segment on a wheel, where the size of the segment is proportional to the weight of the data point. Around the wheel pointers are placed, where the number of pointers is equal to the number of points to be selected. The first pointer is placed randomly and the rest are distributed evenly around the wheel [20].

Figure 2.5 depicts the concept of the sampling method.

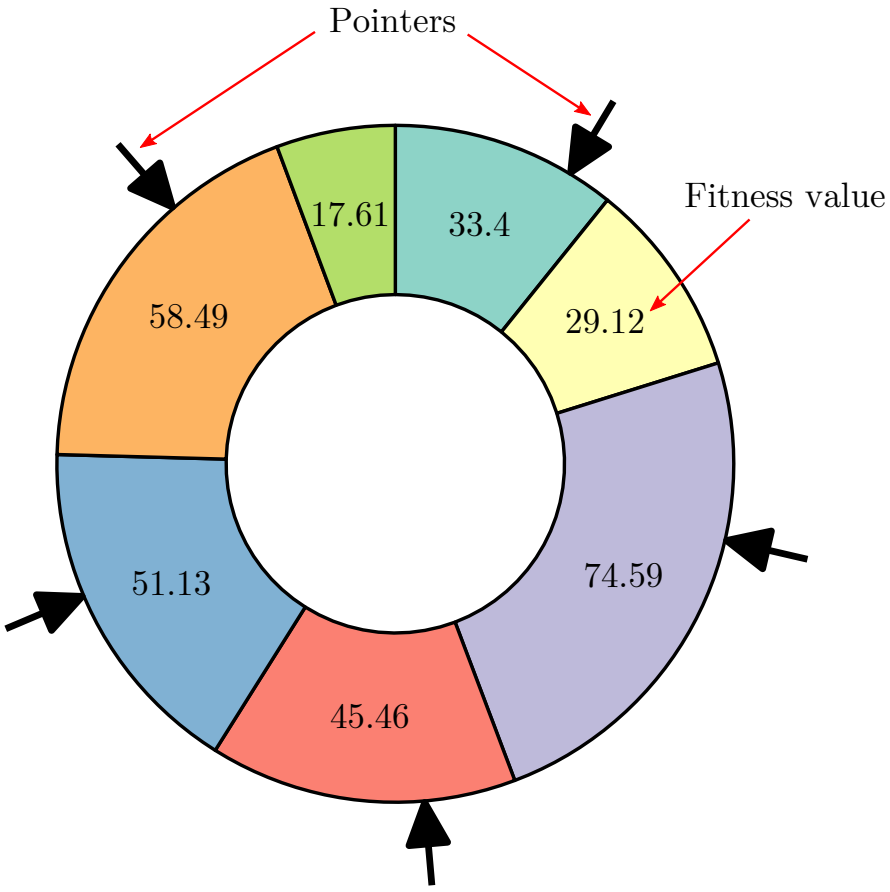


Figure 2.5: Visualization of the roulette wheel selection

The difference between the weighting functions of EBSS and SIBSS is that the self-information based weighting function favours data points in regions with low density, as illustrated in figure 2.6. The idea behind this approach is to sample more unique points and thereby increase the generalization capability of the neural network by creating a more uniform probability distribution. Because outliers are very likely to be sampled, a strict removal of outliers is essential for obtaining a high quality training data set. [16]

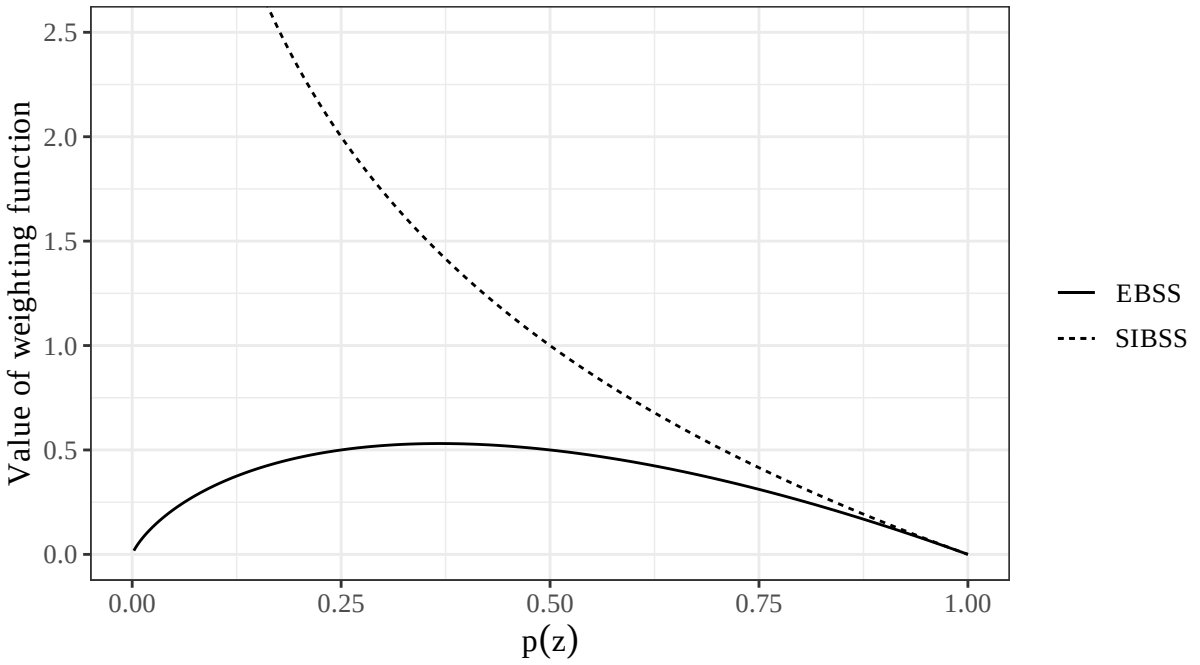


Figure 2.6: Comparison of the weighting functions used in EBSS and SIBSS

2.2.3 Artificial neural networks

Neural networks are powerful tools used for classification and regression problems. Some of their main advantages are that they implicitly detect complex nonlinear relationships between dependent and independent variables, as well as interactions between predictor variables. They also require less knowledge about statistics to be used efficiently, than, for example, logistic regression. [21]

The biggest disadvantage of neural networks is their black-box nature. No information about the underlying relationship between predictor and dependent variables can be extracted from the network. They also require a higher computational effort than other, classical statisctical regression methods. [21]

2.2.3.1 Working principle

A neural network consists of a specified number of layers, which in turn are made up of nodes. Each node in a layer connects to every node in the following layer, except for the layer of the elementwise activation function, which only takes the output of the corresponding node in the previous layer as input. Furthermore, each node in a linear layer has a bias. [22]

See figure 2.7 for an example of a neural network, which takes two input values to predict a single output value utilizing one hidden layer.

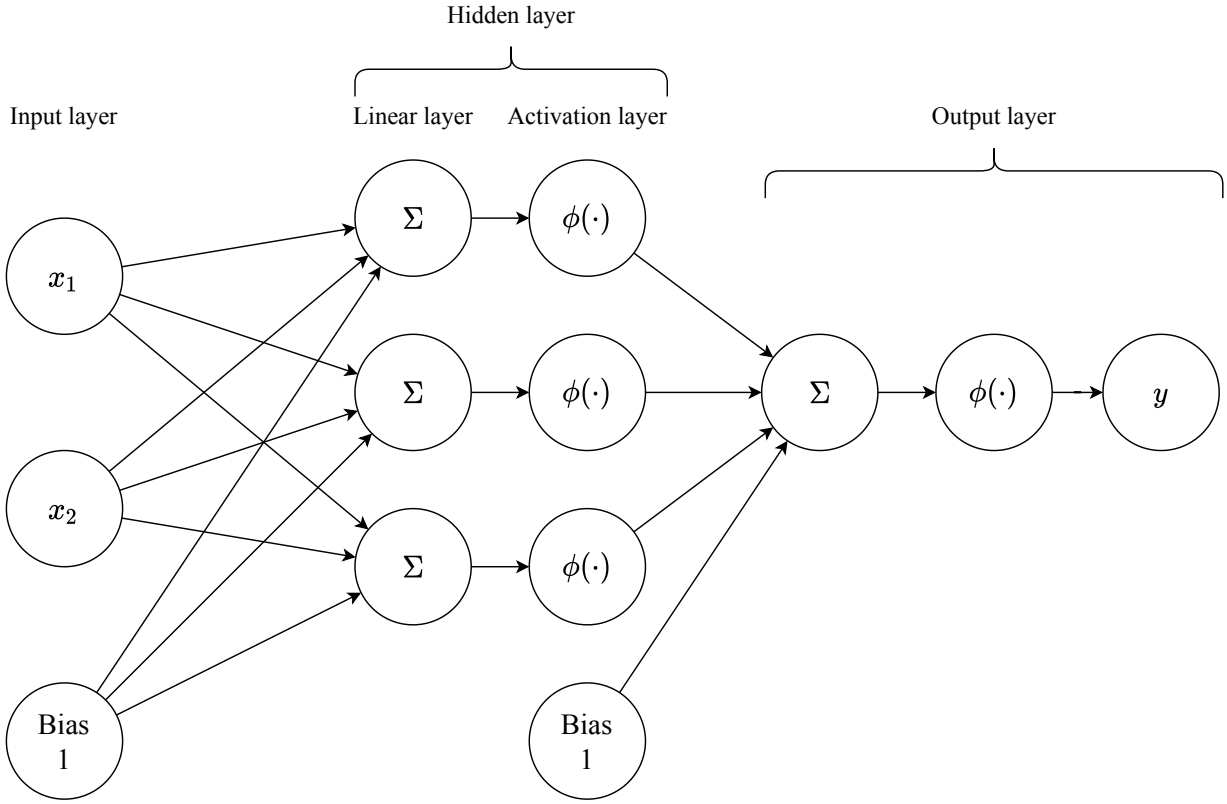


Figure 2.7: A basic exemplary artificial neural network

The training process consists of two steps. The first one is the forward propagation. During this step, values from the training data set are fed through the network via the input ports and every node applies it's transformation along the way. At the end of the network, the predicted output is compared to the expected output and the loss is calculated. [22]

The next step is the back propagation. The resulting loss is transferred back to the neurons in the network, where each neuron receives a fraction of the loss based on its relative contribution to the output. Then, the weights and biases are adjusted to reduce the loss using gradient descent. For this, the gradient of the loss function is calculated to show in

which direction the weights have to be adjusted. The change of the weights is damped by the learning rate, which is specified by the user. Too large learning rates can lead to divergence. [22]

The updating of the network parameters is usually done in batches, i.e. a certain number of samples is propagated through the network before the loss is calculated and the parameters are adjusted. Common batch sizes are 32, 64 or 128 samples. [22]

One training round (also called an epoch) is one pass of the whole training data set. A common way to monitor the training process is to draw the loss over the training rounds. [22] Commonly used loss functions are described in chapter 2.2.3.3.

2.2.3.2 Types of layers

There are many different types of layers, therefore only the ones used in this work will be discussed.

Linear layer: The linear layer consists of neurons, which build the weighted sum from the outputs of the previous layer, as depicted in figure 2.8.

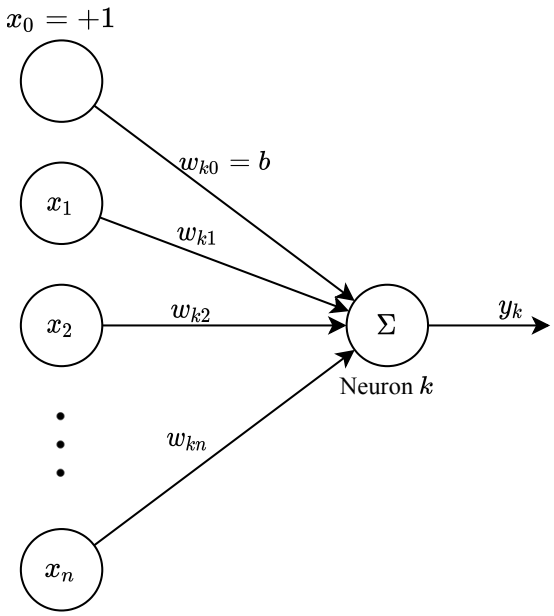


Figure 2.8: Example for a neuron in a linear layer

For the k-th neuron in a layer with $n+1$ inputs x_0 to x_n and the weights w_{k0} to w_{kn}, the output signal y_k is calculated as follows:

$$y_k = \sum_{j=0}^{n} w_{kj} x_j \tag{2.13}$$

The input x_0 is called the bias input and holds the value of +1 and its weight is usually written as b_k. [22]

Activation layer: The activation layer is used to introduce non-linearity into the neural network, or acts as a threshold function. It is a elementwise layer, which means that each neuron only takes one input signal, as can be seen in figure 2.7. Thus, the number of neurons in the activation layer is the same as in the previous layer. Figure 2.9 shows a selection of commonly used activation functions. [23]

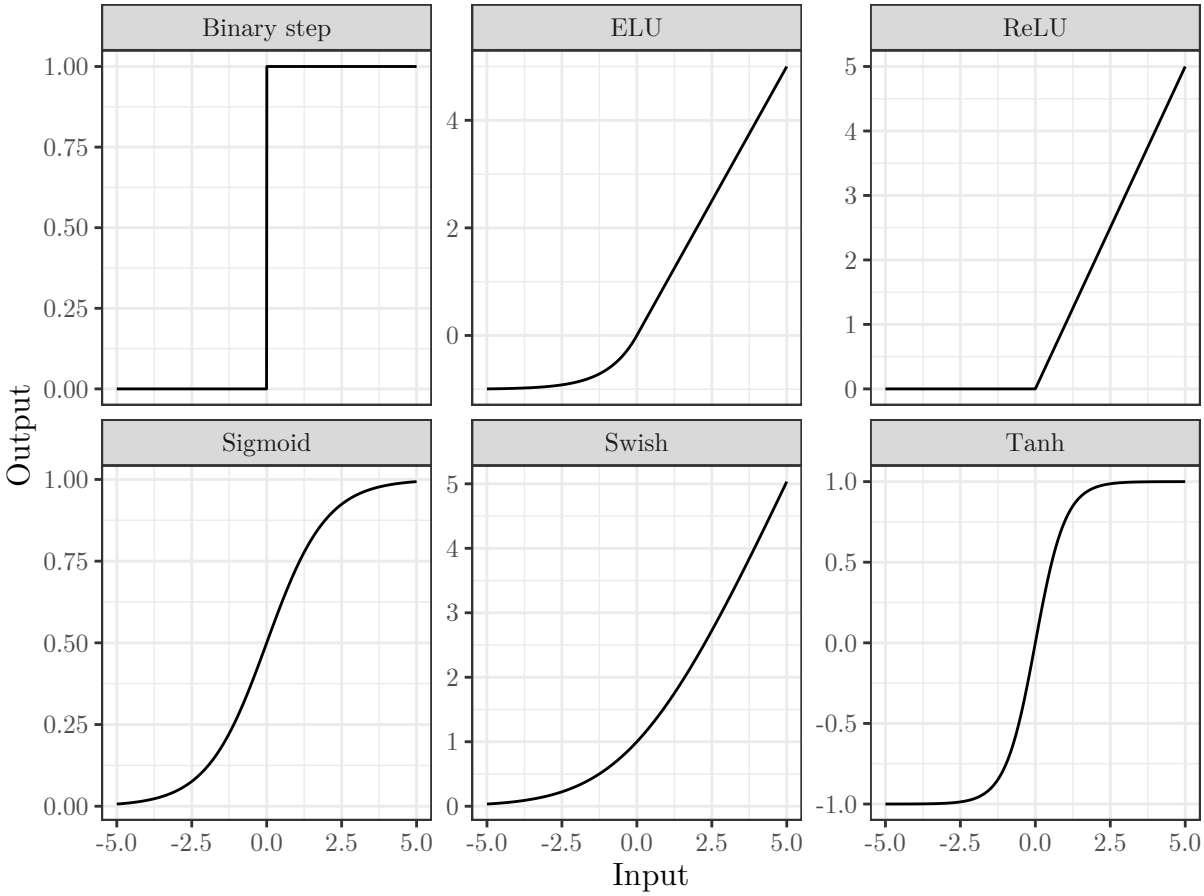


Figure 2.9: Frequently used activation functions

The binary step is mostly used in classification problems, but not in regression problems. In this work, the hyperbolic tangent Tanh$(\cdot)$ has been used as activation function, as previous

work has shown to yield the best results with the available data. ReLU is amongst the most commonly used activation functions for both classification and regression problems. [22]

Swish is a new activation function developed by Google, and it is special as it includes an additional, trainable or pre-definable parameter [24].

Hidden layer: The term hidden layer frequently occurs in literature and describes the combination of a linear layer with an activation layer between the input and output layers, as indicated in figure 2.7.

Dropout layer: The dropout layer deactivates neurons in the subsequent layer with a pre-defined probability. This is a way to prevent over-fitting. Research suggests that neurons may adapt in a way that they compensate the mistakes of previous layers. This leads to complex co-adaptions between neurons, which leads to overfitting because these co-adaptions do not generalize to new data. By disabling a fraction of neurons each pass, the building of co-adaptions is inhibited. [22]

The dropout layers are usually only active during training, except when they are used to stochastically sample the predictions of a neural network to model uncertainty, as described in section 2.2.6.

2.2.3.3 Loss functions

There are a lot of different loss functions for both regression and classification problems. The standard loss function for regression networks is the Mean Squared Error (MSE), which is the mean of the squared differences between the predicted and actual values. [22]

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{2.14}$$

In the equation above, y is the actual value, $\hat{y}$ is the predicted value and N is the number of data points in a batch. The MSE is always positive, and larger differences are penalized heavier than small ones due to the squaring of the difference.

For some applications, the penalization of large differences may not present an advantage. For these cases, the Mean Squared Logarithmic Error (MSLE) can be used as a loss function. [22]

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} \left(\log \left(\frac{y_i + 1}{\hat{y}_i + 1} \right) \right)^2 \tag{2.15}$$

This loss function decreases the penalty for large differences by taking the logarithm first. The +1 is added to prevent undefined expressions, as both y and $\hat{y}$ may take the value 0, for which the logarithm is not defined. [22]

These two are the most important loss function for continuous, single-point regression using neural networks. For predicting probabilities, the log-likelihood is commonly use which is described in chapter 2.2.6.

For classification problems, the loss functions are usually based on cross-entropy. [25]

2.2.4 Gaussian process

A Gaussian process is a supervised, non-parametric, bayesian machine learning method. Its main advantage is that it predicts a distribution instead of a single point estimate. The working principle behind it is based on multivariate gaussian distributions. These distributions are defined by a vector of mean values $\boldsymbol{\mu}$ and a correlation matrix Σ for n variables x: [26]

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{bmatrix}$$

Where σ_{ij} is the covariance between the variables x_i and x_j. The matrix is symmetric, so $\sigma_{ij} = \sigma_{ji}$.

The left side of figure 2.10 shows a multivariate Gaussian distribution for two variables with $\mu_1 = 0.5$, $\mu_2 = -0.5$, $\sigma_{11} = \sigma_{22} = 1$ and $\sigma_{12} = \sigma_{21} = 0.75$.

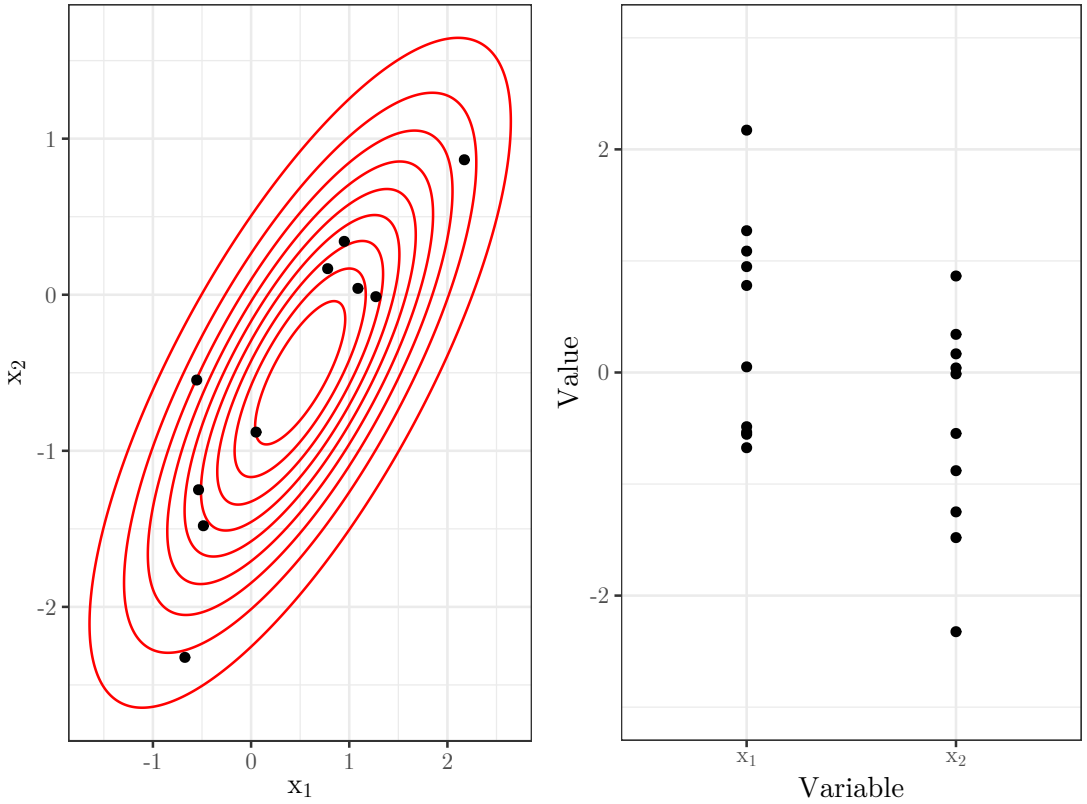


Figure 2.10: Sampling from a multivariate Gaussian distribution - 2 variables

This is a common visualization technique for these distributions, where the contour lines are lines of constant probability. The shape of the contour is an ellipsoid, and for the special case where the covariance is zero, it becomes a circle. A number of points has been sampled from the distribution. The sampled points are plotted again on the right side, where the variable index is on the x-axis, and the value on the y-axis.

A Gaussian process is basically a multivariate Gaussian distribution with an infinite number of distributions [26]. In figure 2.11 the number of distributions is increased to 25 and was sampled 500 times. The resulting curve is already resembling a continuous function.

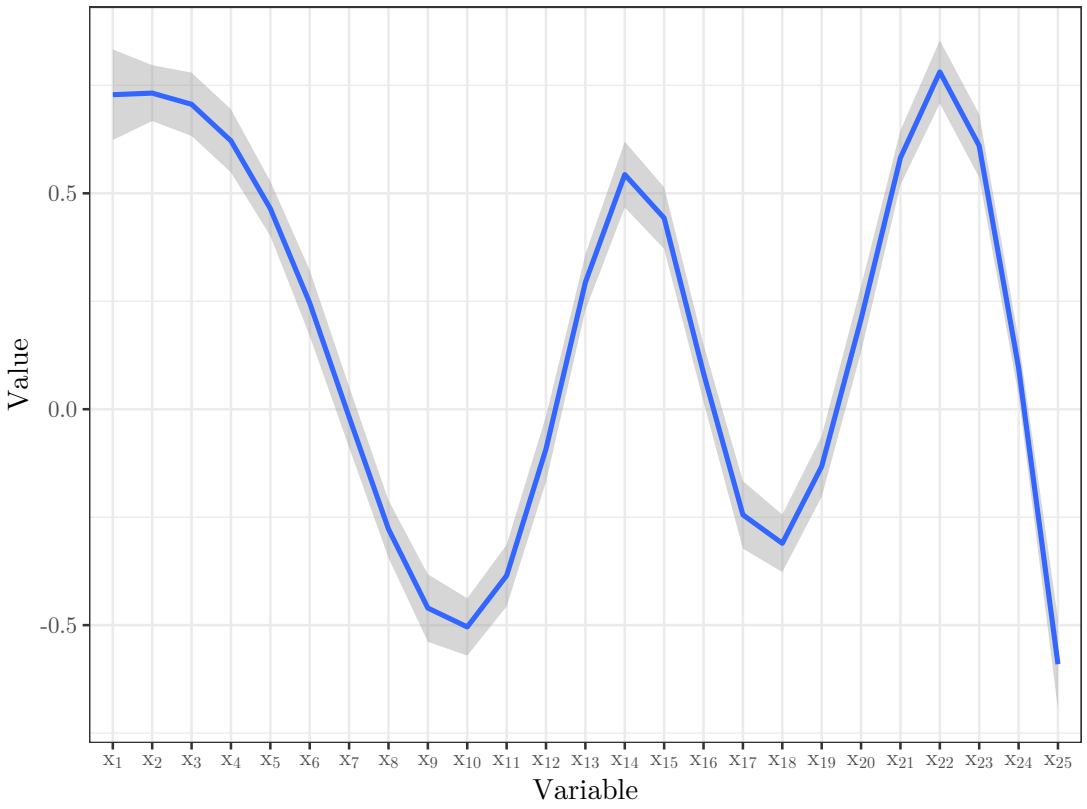


Figure 2.11: Sampling from a multivariate Gaussian distribution - 25 variables

To implement an infinite multivariate distribution, functions are introduced for the mean and the covariance matrix, because a function is basically a vector of infinite length. For most GP applications the mean function has little to no importance, and is simply set to zero most of the time. [26] For the covariance matrix, a kernel function is used. There are many kernel functions available, and they can also be combined by multiplication or addition. The best choice is depending on the data to which the GP should be fitted. For example, the exponentiated quadratic kernel function k is defined as follows: [27]

$$k(x_a, x_b) = \sigma^2 \cdot \exp\left(-\frac{|x_a - x_b|}{2l^2}\right) \tag{2.16}$$

Where σ and l are hyper-parameters which can be tuned to improve the prediction. Different kernel functions have a different number of parameters.

The actual training process delves deep into Bayesian statistics and especially Bayesian inference, which will not be further discussed in this work. One aspect of the training process which is relevant for this work is that during the training process the inversion of the covariance matrix takes place. This is a complex operation with a time complexity of

$\mathcal{O}(n^3)$, leading to long computation times and large memory requirements for datasets with $n \geq 10000$. [28]

Finally, figure 2.12 shows how the available number of data points and their positioning affect the prediction of the GP. The feint, black lines show different predictions made by the multivariate Gaussian distribution, and for the case with no data points (which is called the prior) they are uniformly distributed. Adding more and more points, the posterior distribution starts to follow the underlying function of the data points, and the uncertainty of the prediction is reduced.

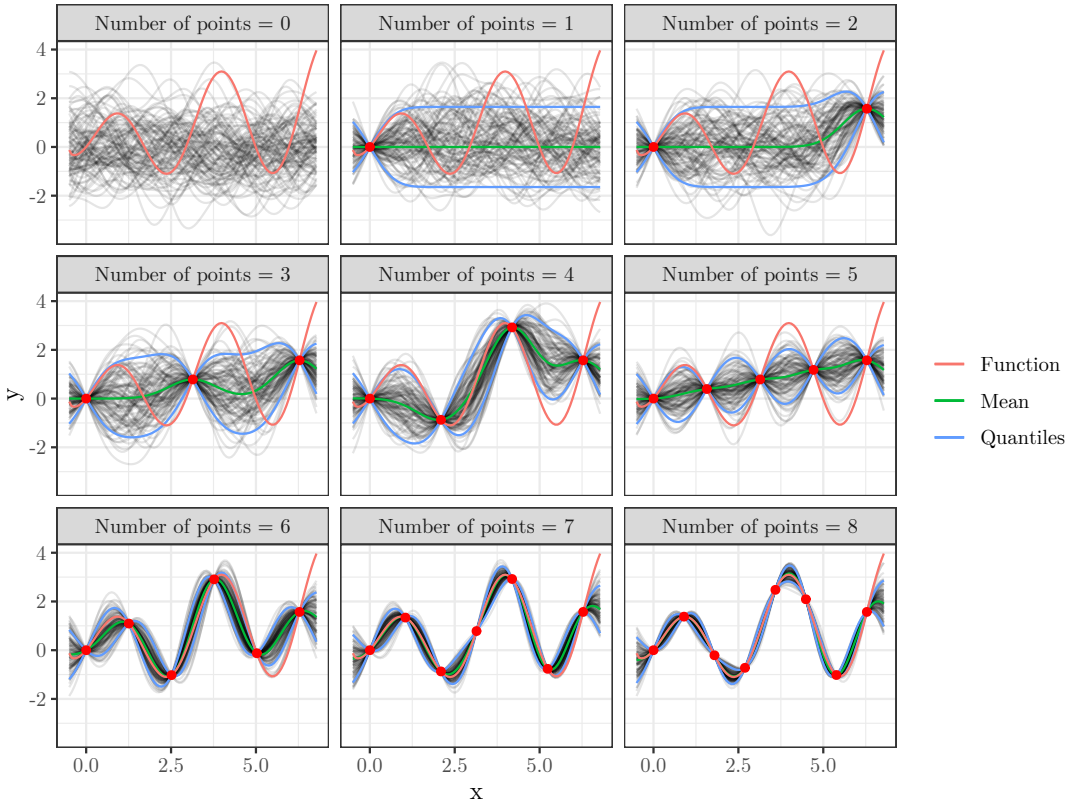


Figure 2.12: Predictions of a Gaussian Process

2.2.5 Linear regression

Linear regression is used in this work for a comparison of the prediction performance of a neural network. Given a dataset of n observations of a dependent variable y and a number of p predictor variables x, as well as an error term ϵ, the regression model can be expressed in matrix form as follows: [29]

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2.17}$$

with

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \qquad \boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} \qquad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \qquad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

Fitting a linear regression model to data is the process of estimating model parameters $\boldsymbol{\beta}$ for which the error term $\boldsymbol{\epsilon} = \boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}$ is minimal. A common method for estimating these parameters is the least-squares estimation. The sum of the error squares is introduced as a loss function $\mathcal{L}$: [29]

$$\mathcal{L} = ||\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Y}||^2 \tag{2.18}$$

This loss function is convex, thus the optimal solution is found where the gradient is zero. Thus, the gradient is calculated, set to zero and solved for $\boldsymbol{\beta}$ to get the optimal parameters $\hat{\boldsymbol{\beta}}$: [29]

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -2\boldsymbol{Y}^T\boldsymbol{X} + 2\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{X} = 0 \tag{2.19}$$

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{2.20}$$

2.2.6 Prediction intervals

Neural networks used in regression predict point estimates, which yield no information about the (un-)certainty of the result. Other methods, like the Gaussian process, predict a probability distribution instead of a point estimate. There are ways to construct neural networks which estimate a distribution, but this is still a very recent developement. A lot of research in this area is done by Yarin Gal, who presents three different approaches to model uncertainty with neural networks.

Homoscedastic regression: In this method it is assumed that the noise of the data is constant across the space of the predictor variables. The network is constructed like any other feed-forward network, but with dropout layers added before the linear layers of the model (except the first one), see figure 2.13 for a visualization of the network used for this example. [30]

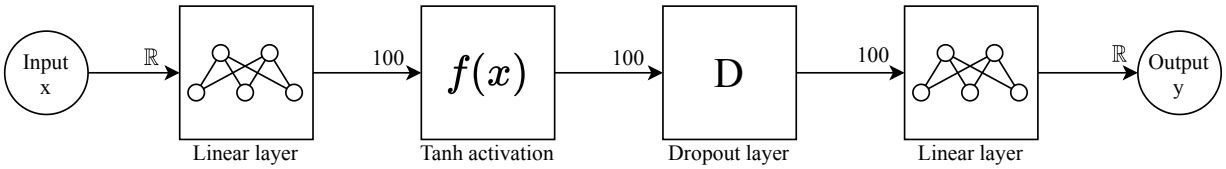


Figure 2.13: Example network for homoscedastic regression

The network is then trained and repeatedly evaluated while the dropout layer is active. This opens up the possibility to calculate the sample variance σ^2_{sample}, to which a constant variance term depending on parameters describing the data and the network training process is added. [30]

$$\sigma^2_{\text{total}} = \sigma^2_{\text{sample}} + \frac{2 \cdot \lambda_2 \cdot N}{l^2 \cdot (1-p)}$$

In this case, λ_2 is the L2 regularization parameter, N is the total number of training data points, p is the dropout probability and l is a correlation parameter. All these parameters - with the exception of N - are free to define and are used to tune the network performance (along with the structure of the network). Figure 2.14 shows the prediction with a $\pm\sigma$ errorband for $l = 2$, $\lambda_2 = 0.01$, $p = 0.1$ and a two-dimensional dataset containing 15 points.

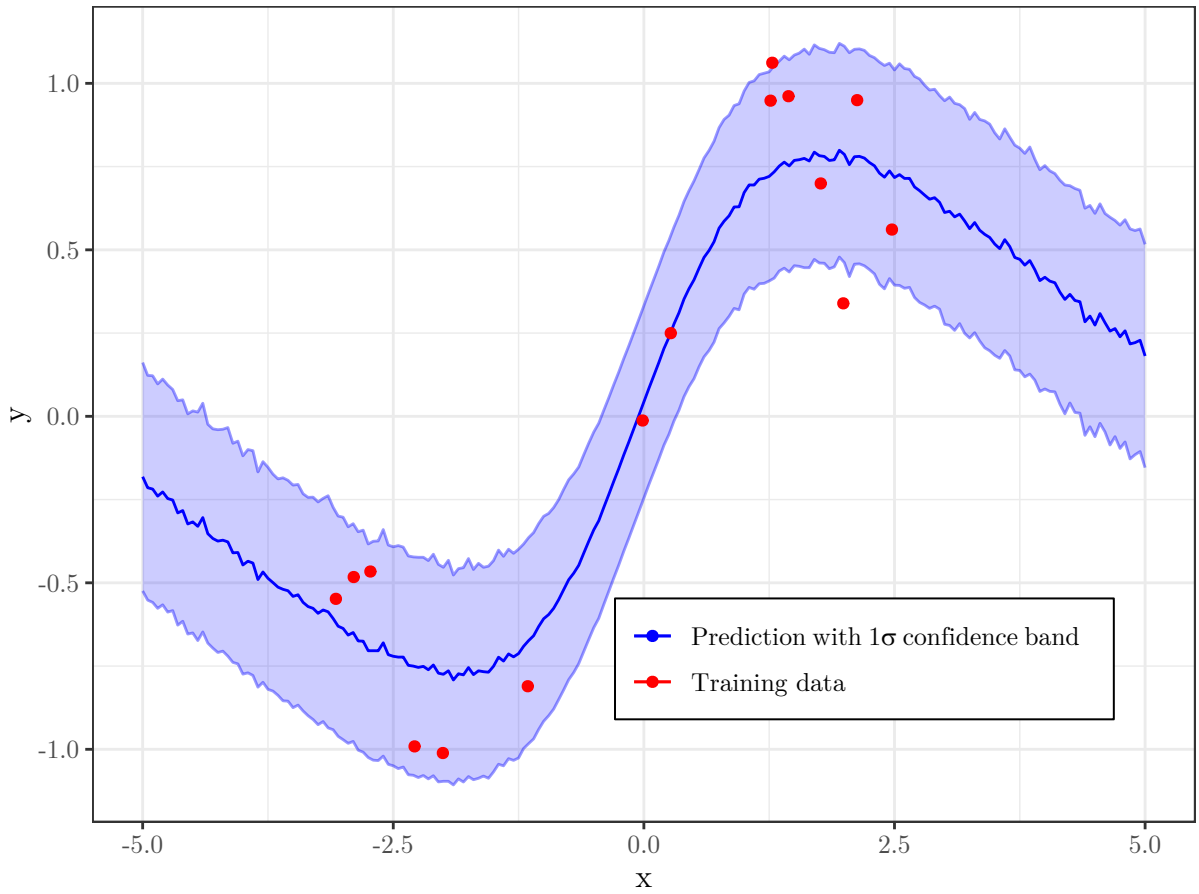


Figure 2.14: Neural network prediction using heteroscedastic regression

Heteroscedastic regression: Heteroscedastic regression estimates the noise in the data by itself. This is achieved by constructing a neural network which fits both the mean μ and the log-precision $\log \tau$ by minimizing the negative log-likelihood of the observed data [30]. The structure of the network is shown in figure 2.15.

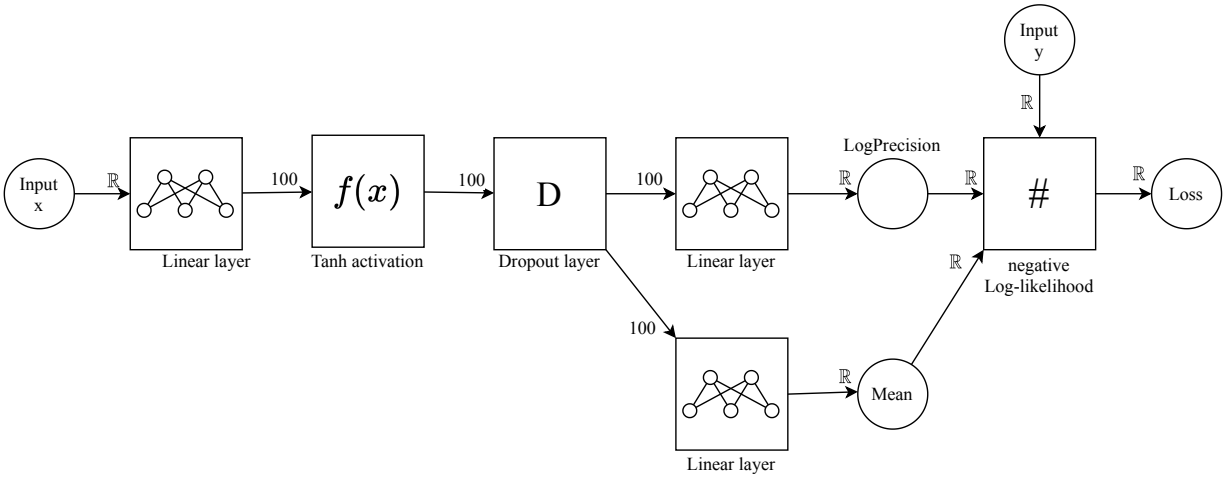


Figure 2.15: Example network for heteroscedastic regression

The loss function $\mathcal{L}$, which is the negative log-likelihood in this case, is defined as follows, where the constant term is omitted: [30]

$$\mathcal{L} = -\log \tau + e^{\log \tau} \left(y_{\text{obs}} - \mu\right)^2 + \cancel{\log\left(2\pi\right)} \tag{2.21}$$

After training, the regression part of the network, which is the network chain from the x-input to the log precision and mean output, is extracted from the whole network. For the actual prediction, the network is again sampled multiple times with the dropout layer active. For each x, the actual mean is then the mean of all mean values gathered during sampling. The total standard deviation consists of the standard deviation of the returned mean values and the mean of the log precision values, which are transformed into standard deviation values during sampling using the following relationship:

$$\sigma = e^{-\log \tau} \tag{2.22}$$

Thus, the total variance is the sum of the variance of the predicted mean and the mean of the predicted variance. Figure 2.16 shows the prediction with a $\pm\sigma$ errorband for $\lambda_2 = 0.01$ and $p = 0.1$. In contrast to the homoscedastic regression, the correlation parameter is no longer required, which leaves two parameters for tuning.

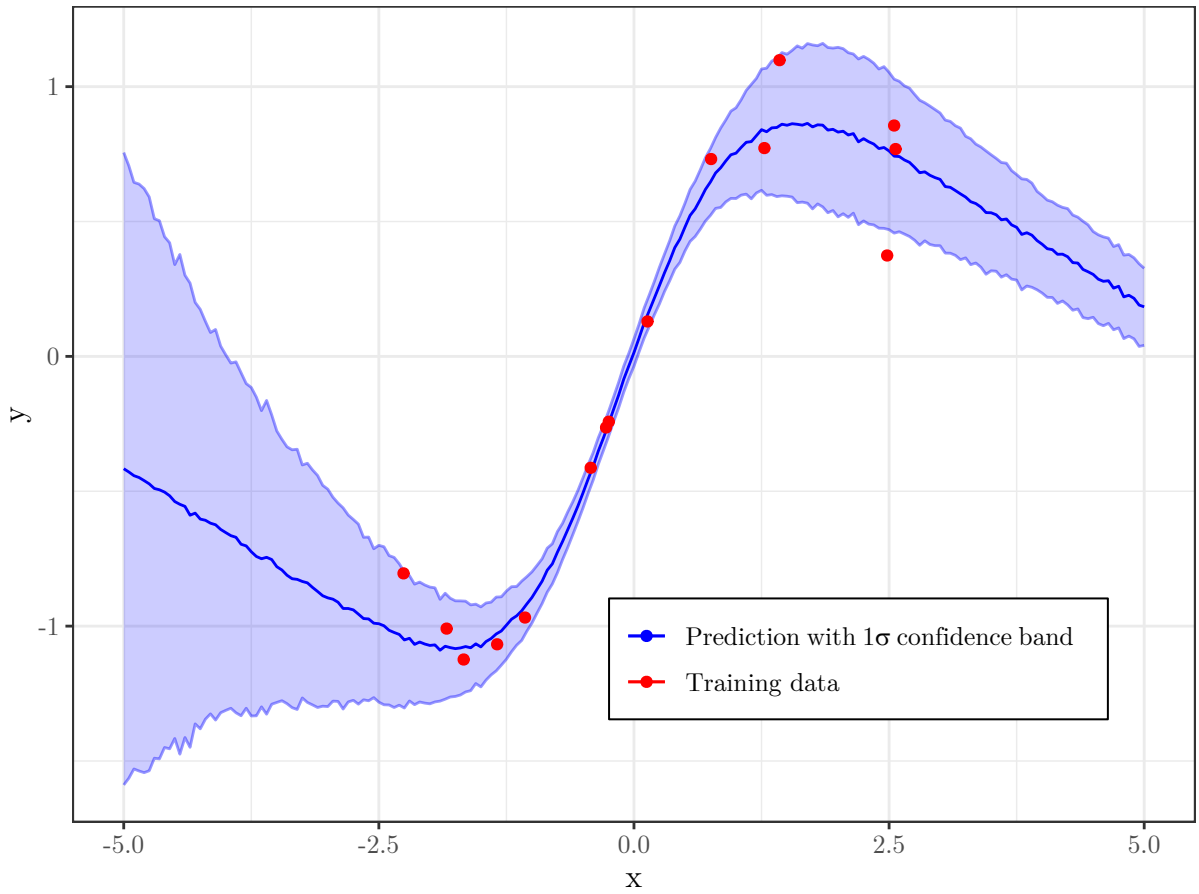


Figure 2.16: Neural network prediction using heteroscedastic regression

The Alpha-Divergence Loss Function: This method represents an extension of the heteroscedastic regression. The standard loss function tends to underfit the posterior which leads to overly optimistic predictions. To address this problem, Li and Gal propose a modified loss function: [31]

$$\mathcal{L}(y_n, \theta, \alpha) = -\frac{1}{\alpha} \sum_{n=1}^{N} \underset{k=1}{\overset{K}{\text{LogSumExp}}} \left[-\alpha \cdot l(\tilde{y}_n^k, y_n) \right] + \text{L}_2(\theta) + \cancel{\frac{N \log(K)}{\alpha}} \tag{2.23}$$

The last term of the equation can be omitted since it is constant. During training, the inputs x_n (with $1 \leq n \leq N$) are fed through the network K times to sample the outputs $\tilde{y}_n^k$, which are compared with the training outputs y_n using the negative log-likelihood loss function l that was defined in equation (2.21). L_2 is the regularization function for the weights θ, and the LogSumExp operator, which is also called RealSoftMax, is a smooth approximation to the maximum function and is defined as follows: [32]

$$\underset{i=1}{\overset{n}{\text{LogSumExp}}}(x_i) = \log(e^{x_1} + e^{x_2} + \cdots + e^{x_n}) = \log\left(\sum_{i=1}^{n} e^{x_i}\right) \tag{2.24}$$

The parameter α is the divergence parameter, and controls the "pessimism" of the neural network. The higher α is, the more cautious the network and the larger the error estimates will be [31]. The parameter is typically tuned to be between 1 and 0, where 0.5 is a good starting value for the tuning process.

Figure 2.17 shows the network structure for heteroscedastic regression using the alpha divergence loss function.

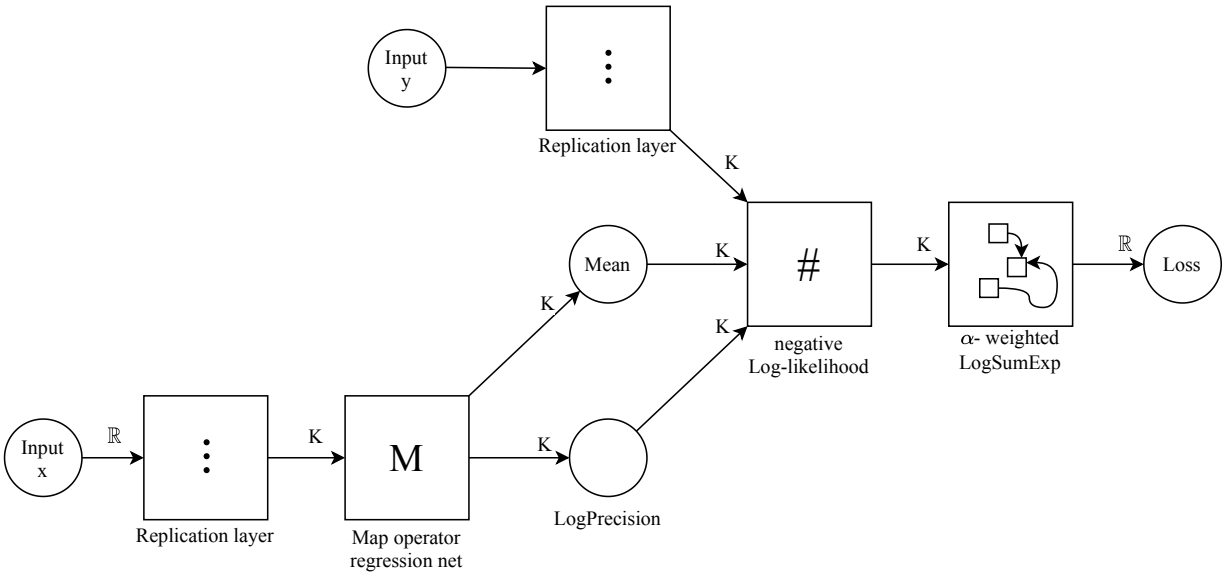


Figure 2.17: Example network for heteroscedastic regression with alpha divergence

The first step is to replicate the in- and outputs K times using replication layers. Then the regression net, which is the same net as in the heteroscedastic regression, is mapped over the K inputs. The resulting log precisions and means are fed into the negative log-likelihood loss function along with the replicated outputs. The resulting losses are then fed into the alpha-weighted LogSumExp layer, which multiplies the input vector with $-\alpha$ and factors out the largest term before feeding the vector into the exponent function to increase numerical stability. [31]

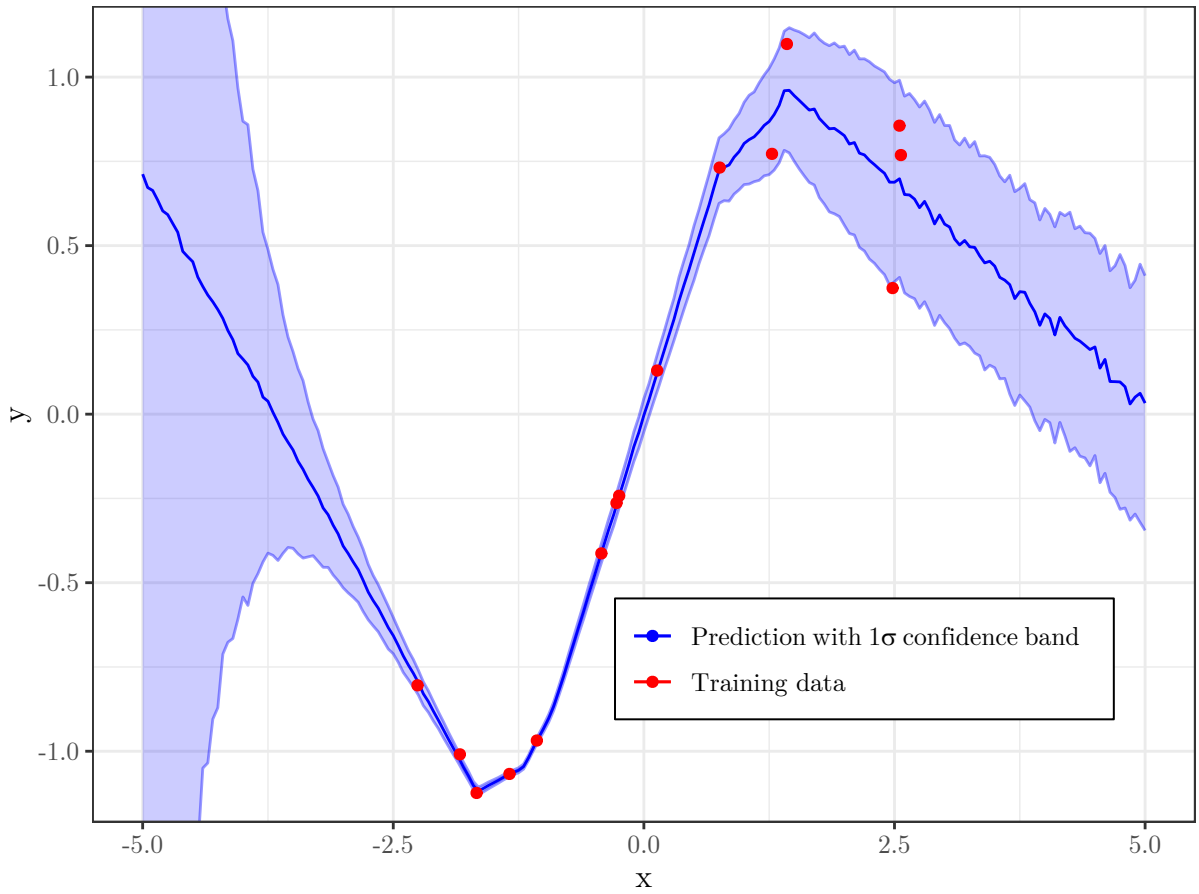


Figure 2.18: Neural network prediction using the Alpha-Divergence loss function

2.3 Multi objective optimization - NSGA-II

For optimization, the multi objective genetic algorithm NSGA-II (Non-dominated Sorting Genetic Algorithm II) is used. This chapter aims at giving a basic understanding of the algorithm, for a more detailed explanation consult other sources like [33] or [34]. This work uses an implementation by the author of this work [35], on which this chapter is based.

Multi objective optimization algorithms can minimize (or maximize, by taking the negative) any number of objective functions in dependence of any number of variables. For process engineering applications two to three objectives are usually sufficient.

The working principle of the algorithm is inspired by the theory of evolution and is based on genetic operators like mutation, crossover and jumping genes.

The nomenclature in genetic algorithms is also inspired by the theory of evolution. To better convey the working principle of the algorithm, a few definitions need to be explained.

Population: A population contains a certain number of individuals. Every individual represents a solution of the optimization problem.

Sorted population: A population, in which the individuals are organized into ranks (i.e. pareto fronts).

Rank: A rank or pareto front consists exclusively of non-dominated solutions.

Individual: An individual is made up of one or more chromosomes (the genotype), the corresponding values of the target functions (the phenotype) and the crowding distance.

Chromosome/Genotype: A chromosome is the binary representation of a variable. It consists of a certain number of genes, which may vary between different chromosomes.

Gene: The gene is the smallest building block of a genetic algorithm. It is a single digit in the binary representation of a variable, and its value is either 1 or 0.

Elitism: The best individuals of one generation are allowed to pass into the next generation without any alteration. This ensures that the solution quality will not decrease during the optimization.

Figure 2.23 on page 33 shows the fundamental operating principle of the algorithm. First, N_P individuals are randomly generated and sorted into ranks. Then the crowding distance is calculated. Using tournament selection, N_P individuals are selected for reproduction by crossover. After reproduction, the resulting offspring is mutated and the jumping gene operator is applied. The offspring is then recombined with the initial population and again sorted into ranks. From the combined population with a size of $2N_P$, the best N_P individuals are selected. The process is then repeated until the algorithm terminates, either by achieving convergence or reaching the maximum number of allowed generations.

2.3.1 Binary encoding

The variables occuring in the target functions are represented in binary form to allow for easy application of the genetic operators. Each variable i is characterized by a value range, specified by the lower bound x_i^L and the upper bound x_i^U, and the length l_{string} which is the number of bits used to represent the variable.

The following equation is used to convert a binary string to it's corresponding real value:

$$x_i = x_i^L + \frac{x_i^U - x_i^L}{2^{l_{\text{string}}} - 1} \cdot \left(\sum_{i=0}^{l_{\text{string}}-1} 2^i \cdot S_i \right) \tag{2.25}$$

S_i is the binary value of the gene at position i, where $i = 0$ is the rightmost gene and $i = l_{\text{string}} - 1$ is the leftmost gene. For example, the conversion of the string 001101 with $x^L = 1$ and $x^U = 20$ yields the following result:

$$x = 1 + \frac{20 - 1}{2^6 - 1} \cdot \left(2^0 \cdot 1 + 2^1 \cdot 0 + 2^2 \cdot 1 + 2^3 \cdot 1 + 2^4 \cdot 0 + 2^5 \cdot 0\right) \approx 4,921$$

Figure 2.19 shows the binary encoding of a variable x over the interval $[0, 11]$ with $l_{\text{string}} = 4$.

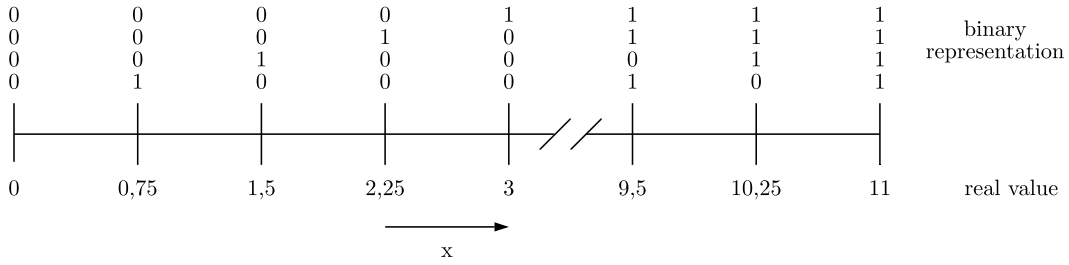
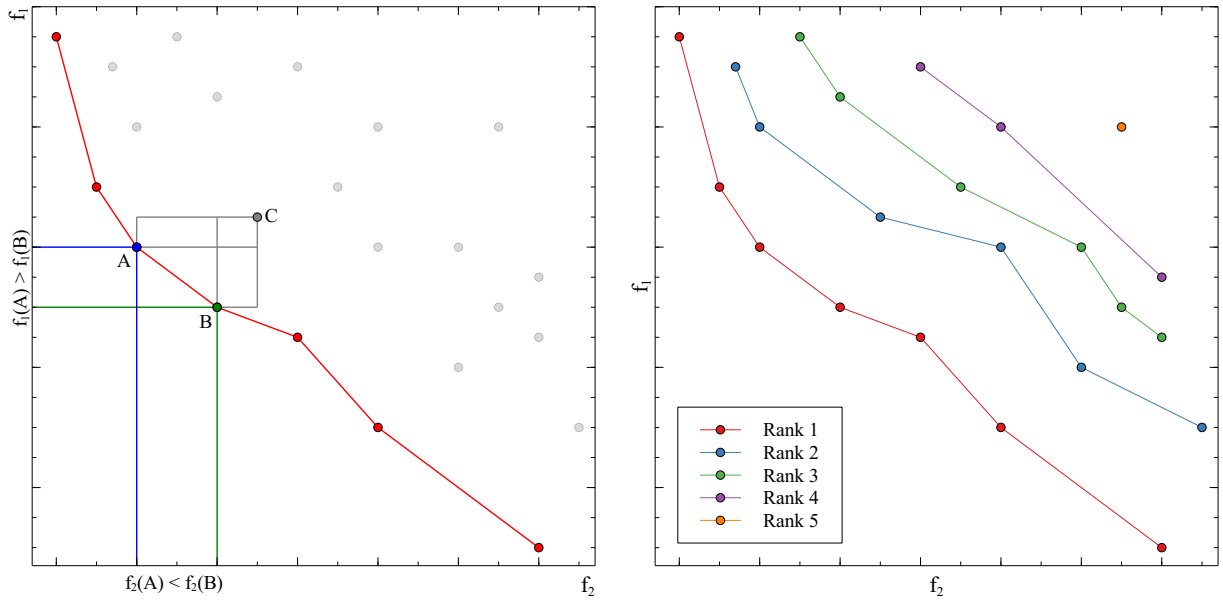


Figure 2.19: Binary encoding of a variable x over the interval $[0, 11]$

2.3.2 Pareto frontiers

Figure 2.20 illustrates the concept of the pareto front for a two-dimensional optimization problem. A solution $\boldsymbol{x^P}$ is pareto optimal or non-dominated, if no other $\boldsymbol{x}$ exists for which $f_1(\boldsymbol{x}) \leq f_1(\boldsymbol{x^P})$ and $f_2(\boldsymbol{x}) \leq f_2(\boldsymbol{x^P})$ are true, where strict inequality must apply for at least one function. [36]

In figure 2.20a, point C is dominated by both A and B and is therefore not part of the pareto front. A and B are not dominated by any other point, i.e. there is no point where both f_1 and f_2 are lower. Thus, they are part of the pareto front. The sorting procedure is repeated until no more unsorted solutions remain, as depicted in figure 2.20b.



(a) A single pareto front

(b) A fully sorted population

Figure 2.20: The concept of pareto frontiers

2.3.3 Crowding distance

Another important measure used in NSGA-II is the crowding distance. It is a measure for the distance between a solution and its neighboring solutions and is used to retain solution diversity. During the reduction from the combined population (Box P$'''$ in figure 2.23), the best ranks are picked first. If a rank contains more solutions than needed to get to N_P solutions, the ones with the highest crowding distance are selected. It is infinite for solutions at the boundary of a pareto front. The crowding distance CD for a solution i is calculated as follows, where n_f is the number of objective functions:

$$\mathrm{CD}_i = \frac{1}{2} \cdot \sum_{j=1}^{n_f} (f_{j,i+1} - f_{j,i-1}) \tag{2.26}$$

Figure 2.21 shows the crowding distance for a point i in the first rank of the previously sorted population. The crowding distance for this particular solution is calculated as follows:

$$\mathrm{CD}_i = \frac{1}{2} \cdot \left[\underbrace{|f_{1,i+1} - f_{1,i-1}|}_{b} + \underbrace{|f_{2,i+1} - f_{2,i-1}|}_{a} \right]$$

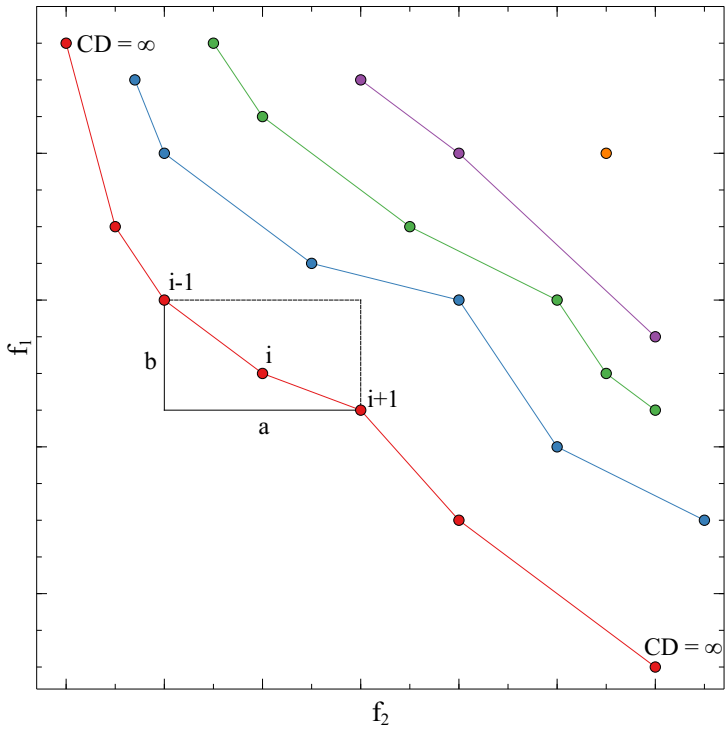


Figure 2.21: Visualization of the crowding distance for a point i in rank 1

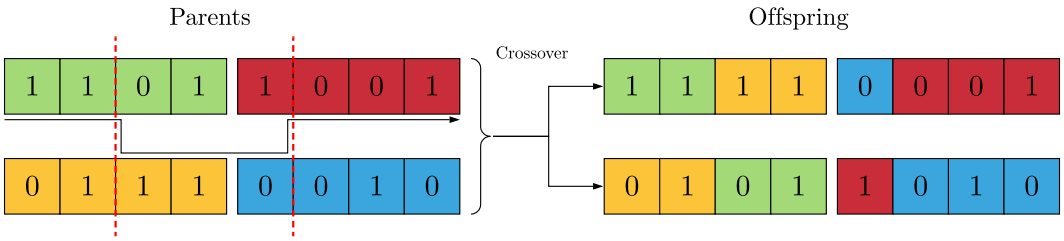
2.3.4 Genetic operators

The three types of genetic operators used in this algorithm are crossover, mutation and the jumping gene.

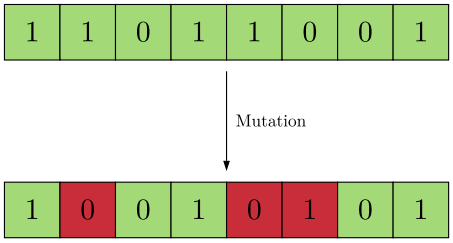
The crossover is used to combine the genetic information of two parents to create an offspring. Two random individuals are selected from the population that has been created using tournament selection. Whether crossover happens or they are passed to the offspring population unmodified is determined randomly with a certain crossover probability P_{Cross}. In this work, a k-point crossover has been used, where k is the number of randomly chosen crossover points. The procedure is illustrated in figure 2.22a.

During the mutation step, each gene of every chromosome in every individual of the offspring population is randomly mutated with a certain probability P_{Mut}. If a gene mutates, its value is flipped, thus a 1 becomes a 0 and vice versa. The procedure is depicted in figure 2.22b.

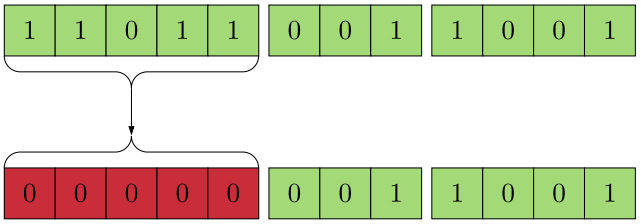
The last operator is the jumping gene. There exist many different variations of this operator, but previous research [35] has shown that the modified jumping gene (mJG) is the most



(a) 2-point crossover of parents with two chromosomes containing four genes



(b) Mutation of the second, fifth and sixth gene in a chromosome



(c) application of the mJG operator

Figure 2.22: Visualization of the genetic operators

effective one. Every chromosome in every individual of the offspring population is randomly set to its maximum (all genes become 1) or minimum (all genes become 0) value with a probability P_{JG}. Figure 2.22c shows the application of the mJG operator.

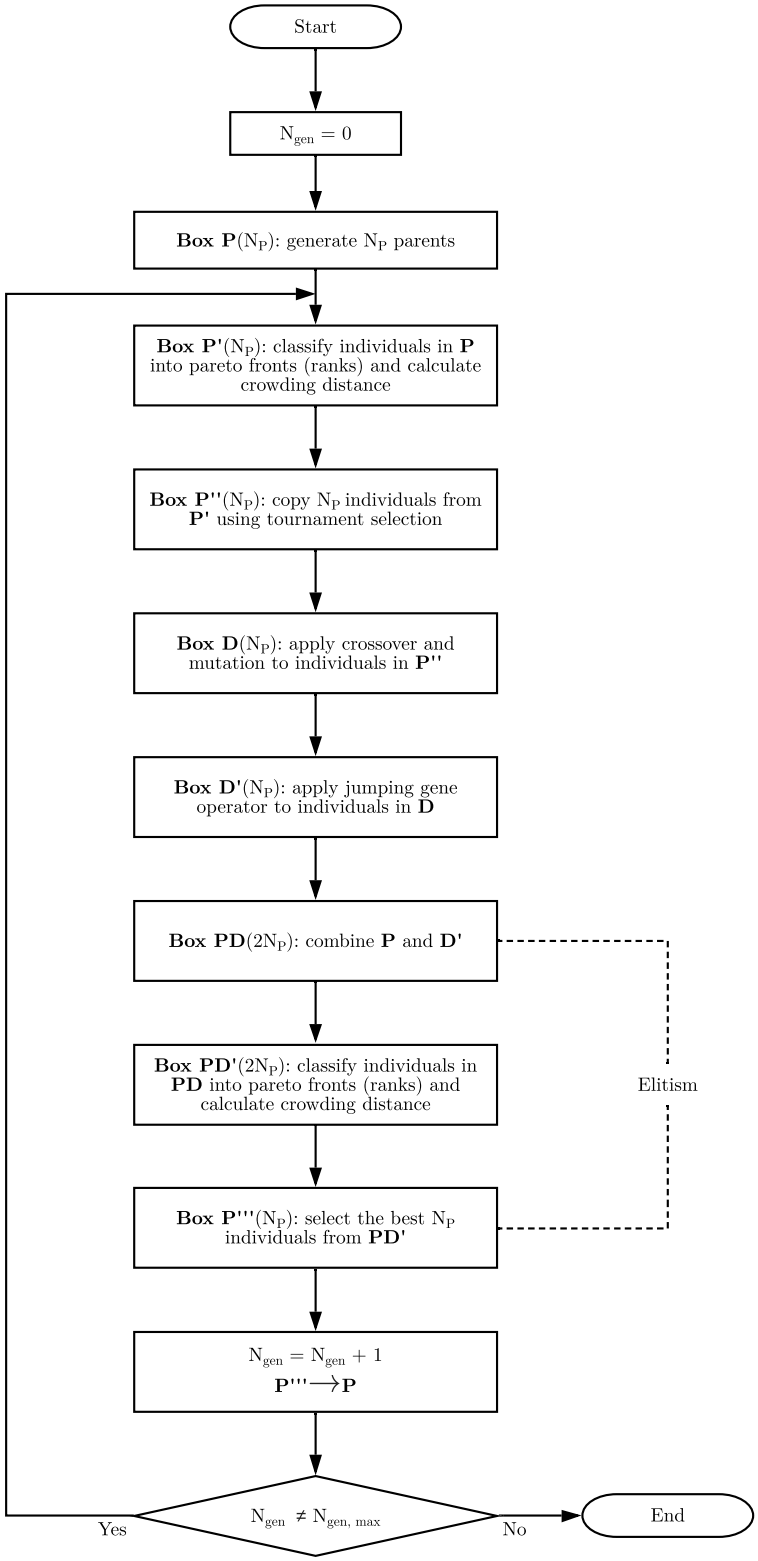


Figure 2.23: Flowchart for NSGA-II

3 Implementation

3.1 Flowsheet Solver

This chapter gives insight into the inner workings of the newly developed FlowsheetSolver-package, i.e. what is it doing and why. For a more detailed description, the package documentation can be consulted.

The procedure of solving a simulation problem can be divided into three major steps:

1. Gathering and processing input data.
2. Identifying recycled streams and simplification of the flowsheet.
3. Sequential evaluation of the flowsheet units.

3.1.1 User input

To completely specify a simulation case, four different sets of information and parameters need to be supplied by the user:

Stream specification: This dataset contains the names of streams, their origins and their destinations. By the definition of the streams, the units in the flowsheet are implicitly defined. A skeleton flowsheet can be drawn from the stream definition alone. Every stream needs to have an origin and a destination, thus, a dummy unit needs to be defined to properly specify feed streams. This could be a separate unit for every feed stream, which leads to a less cluttered flowsheet in case of many streams, or a single unit which is the origin for every feed stream.

Stream properties: This dataset contains information about convergence, flow rates and composition of all streams. Initial conditions for the simulation case are set by supplying information about streams in this dataset.

Unit functions: This dataset specifies what calculation routine is used for which unit. The stream specification along with the unit functions can be used to draw a more sophisticated flowsheet like one may know from KBC PetroSIM® or Aspen HYSYS®.

Unit parameters: Necessary information that is specific for a certain unit function is stored in this dataset. This includes a plethora of things, ranging from temperature and pressure specification for flash units to the filepaths for neural networks for grey-box units.

Consider the act of dragging and dropping a flash unit into the flowsheet in a process simulation software featuring a GUI, as well as adding a feed, top and bottom stream. This is the equivalent of adding the three streams to the stream specification, specifying that the unit created uses a flash calculation, identifying which of the product streams contains the liquid and which one contains the vapour phase in the unit parameters and specifying a flow rate and composition in the stream properties.

User interface: A basic graphical user interface for setting up a simulation case has been implemented. It is called using the `SimulationInterface[]` command in Mathematica.

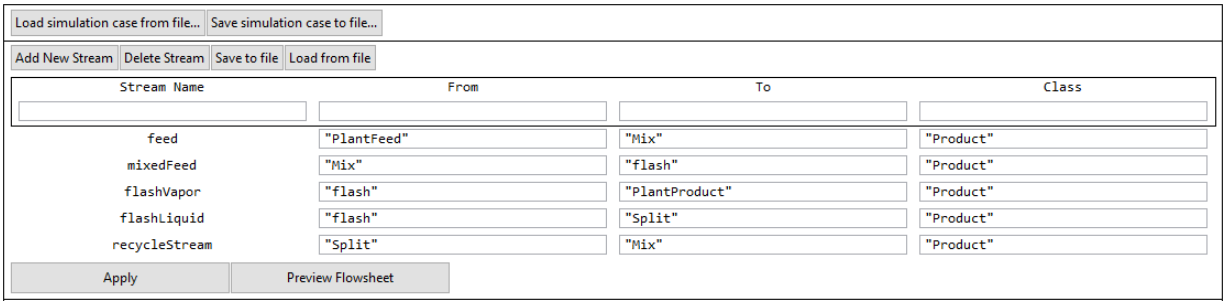


Figure 3.1: Input mask for the streams dataset

Figure 3.1 shows the user interface for defining streams. Positioned at the top are buttons for loading and saving whole simulation cases, i.e. streams, stream properties, unit functions and unit parameters, at once. Note that a simulation case will only be saved correctly after the unit parameters (i.e. the last information) have been specified. The text fields are used to enter all the necessary information for a stream definition, and the button *Add New Stream* appends the information to the list below. The *From*, *To* and *Class* information can be edited using the text corresponding text fields, but the name can not be changed. To remove a stream, the name is entered in the *Stream Name*-field and the *Delete Stream*-button is pressed. When first defining a stream, all information has to be entered **without** quotation marks.

All the information shown to the user is stored in a temporary data structure which is not accessible for the FlowsheetSolver package until the data is transferred to the package context by pressing the *Apply*-button.

The *Preview Flowsheet*-button opens a new Mathematica notebook which contains a graph of the flowsheet. This makes it easier to verify if all the streams have been entered correctly.

The *Save to file* and *Load from file* buttons are used to export or import just the streams dataset, but not the rest of the required information like stream properties or unit functions and parameters.

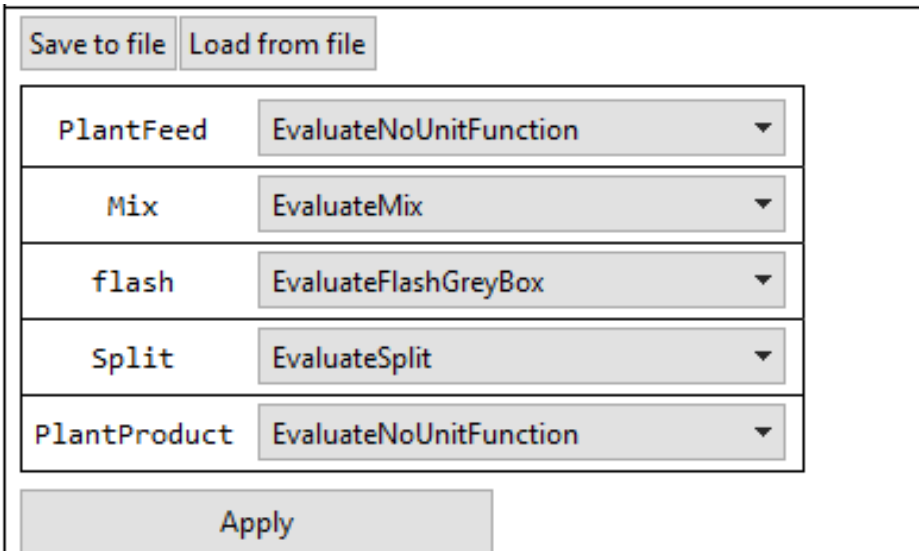


Figure 3.2: Input mask for the unit functions

Figure 3.2 shows the interface for defining unit functions. For every unit in the flowsheet, a function is selected using the corresponding dropdown menu. The data can also be saved and loaded, and needs to be transferred to the package context via the *Apply*-button.

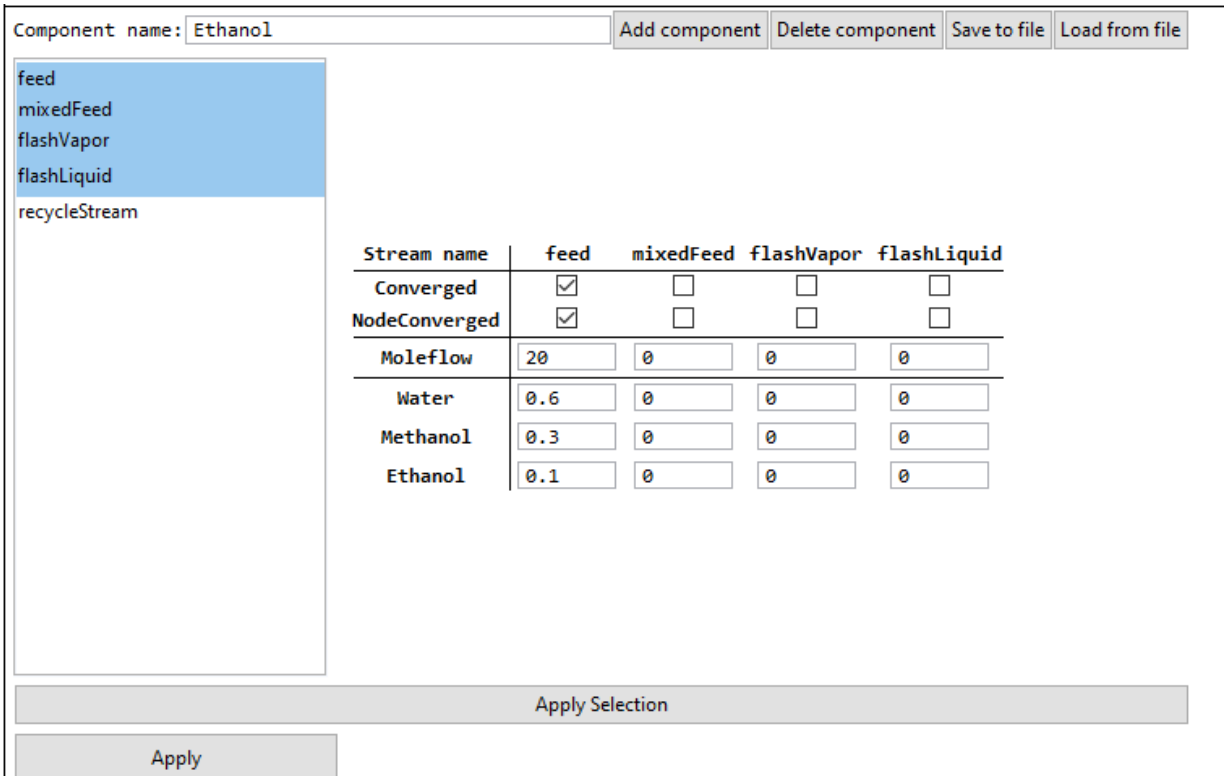


Figure 3.3: Input mask for the stream properties

Figure 3.3 shows the interface for defining stream properties. On the left, the user can select multiple streams which are then displayed in the table after pressing the *Apply Selection*-button. In the table, each column corresponds to one stream. The checkboxes for *Converged* and *NodeConverged* are actually a relic from previous versions, where the framework did not automatically detect feed streams. The boxes can be checked for input streams as a visual guide for the user, but it is not required for the calculation as the information is overwritten during the initialization of the algorithm. The *Moleflow* specifies the flow rate of a stream, and below that are fields to specify the fractions of all available components. Components are added or removed via the text field and the buttons at the top of the interface. Again, the information needs to be transferred to the package using the *Apply*-button.

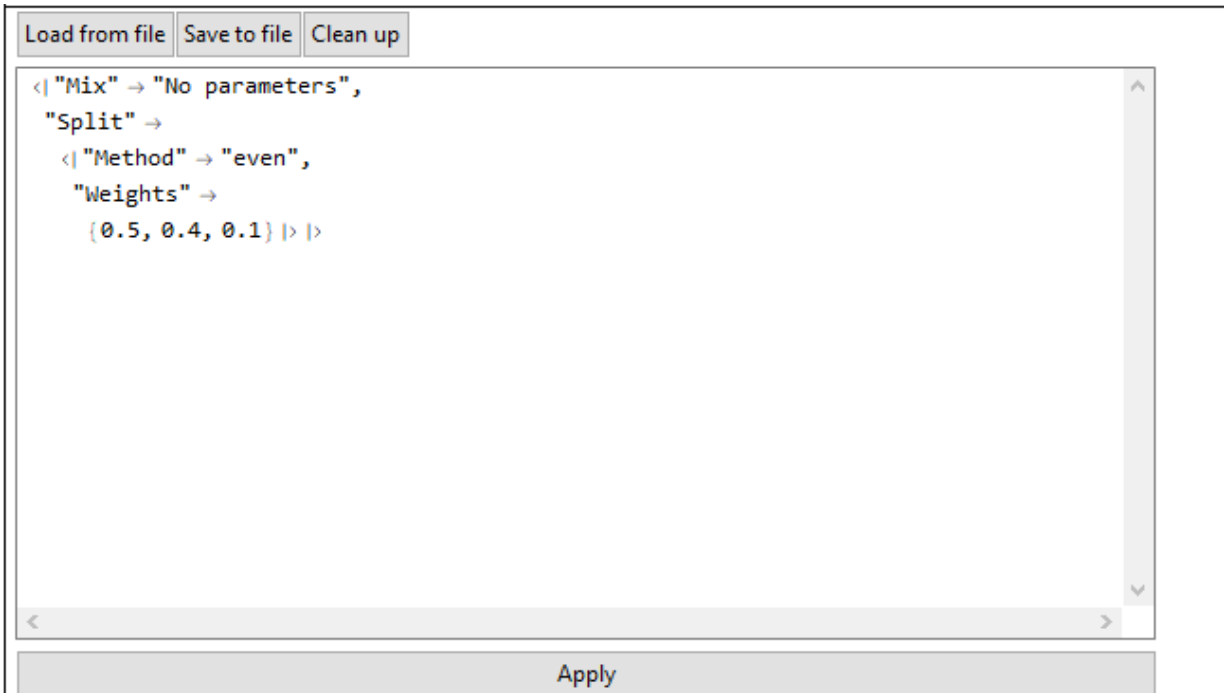


Figure 3.4: Input mask for the unit parameters

Figure 3.4 shows the interface for specifying unit parameters. This is a large text input field, which already contains an expression that is generated automatically from the unit names and their designated functions. The user needs to read through the expression and edit the parameters as needed for the simulation problem at hand.

The *Clean up* button at the top removes units which do not require parameters from the expression.

After entering the parameters and pressing the *Apply*-button, the flowsheet is fully specified and the simulation case can be saved.

Starting the simulation: After fully specifying the flowsheet, the user needs to execute the `RunSimulation[]` command to start the solving algorithm.

3.1.2 Recycled streams & flowsheet simplification

The flowsheet simplification, or computation path optimization, is partially based on the work of Laguitton et al. [37]. In this step, cycles in the flowsheet are detected and condensed into complex nodes. The general procedure is as follows:

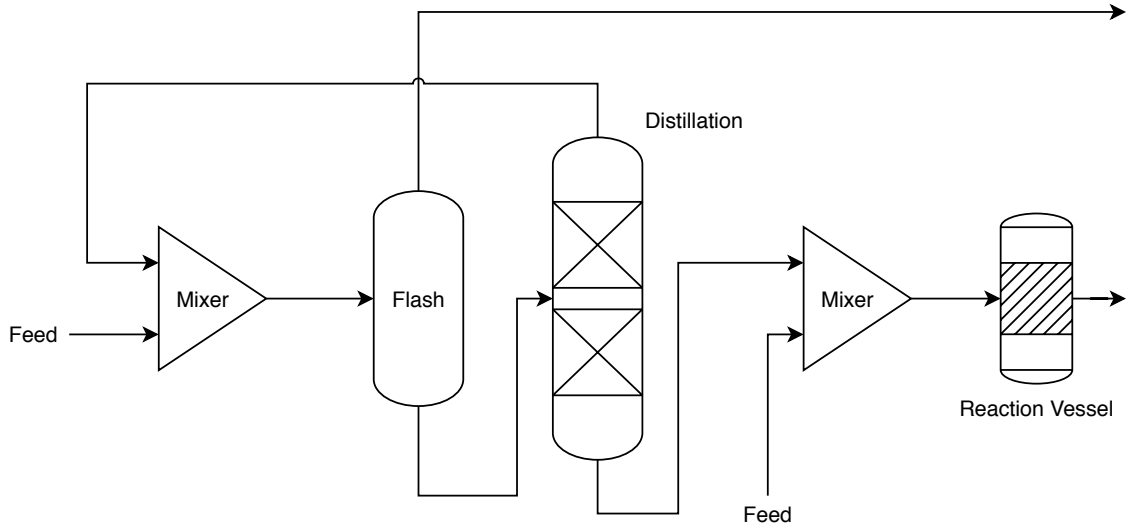
1. Identify all cycles in the flowsheet.

2. Check if any cycles share a common stream. If yes, merge the cycles into a new cycle.
3. Pick a random stream from each cycle, tear it and insert a recycle unit.
4. Replace all units in a cycle with a single complex node.

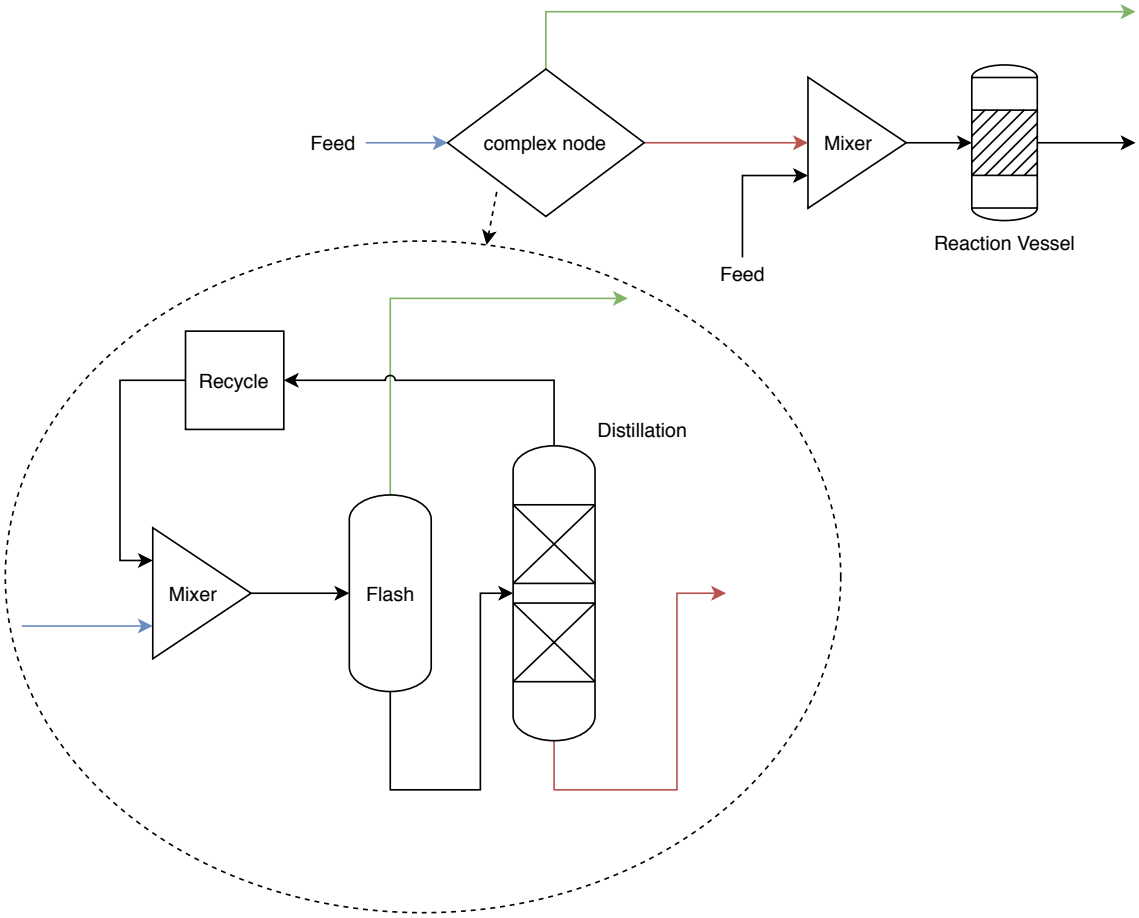
The algorithm automatically identifies recycled streams using Mathematicas graphing capabilities, as the flowsheet is in essence a directed graph.

Figure 3.5 shows a basic example flowsheet and its simplified version to illustrate the concept.

The main reason for the usage of complex nodes is that the recycles are separated from the main flowsheet. The complex nodes are treated like any other unit, with inputs, outputs and an associated function. This makes the implementation of the sequential modular solving routine considerably easier. Also, the resulting reduced flowsheet is completely sequential.



(a) A basic flowsheet with a recycled stream



(b) The simplification of the flowsheet in (a)

Figure 3.5: Comparison between a flowsheet and its simplification

3.1.3 Solving routine

After the flowsheet simplifcation by introducing complex nodes, the flowsheet is completely linear (or sequential) as all cycles are contained within the complex nodes. Before the algorithm starts to step through the units, the feed streams are automatically detected and they are declared as converged. To find theses streams, the algorithm is looking for streams, which come from an origin unit that has no inputs and thus, only outputs.

The algorithm is then looking for units, which are ready for evaluation. This is done by checking if all input streams for a unit are converged. These units are then evaluated, after which their output streams are set to converged and the procedure is repeated until the algorithm fails to find any units to evaluate. The algorithm keeps track of already evaluated units and prevents them from being evaluated twice.

3.1.3.1 Complex nodes

A special calculation routine handles the evaluation of complex nodes. As seen in figure 3.5, each complex node is essentially a sub-flowsheet.

All streams have a property (or status), to track convergence during the evaluation of a complex node, similar to the convergence status for the main flowsheet. During initialization, the user is asked to specify the starting point for each complex node, which can be any unit in the node with an input stream from the main flowsheet. For the selected unit, the node convergence of all input streams is set to true, so that it is guaranteed to be identified as a unit ready for evaluation. Also, output streams of units which are not part of a complex nodes have their node convergence status set to true along with the convergence status for the main flowsheet.

Similar to the routine for the main flowsheet, the algorithm is looking for units which are ready for evaluation by checking if the node convergence of all input streams is true.

Furthermore, the algorithm keeps a list of units which have not yet been evaluated during the current iteration. An iteration is finished, when every unit in the complex node has been evaluated once. The list of units ready for evaluation is compared to the list containing the unevaluated units, and only units, which have not been evaluated during the current iteration are retained. This ensures that each unit is evaluated only once during an iteration. After a unit has been evaluated, the node convergence of the input streams which originate from a unit that belongs to the complex node are set to false. Input streams coming from outside the complex node retain their node convergence status. Figure 3.6 shows how the status of the streams within a complex node changes during the first iteration, with unit #1 being the starting point for the calculation.

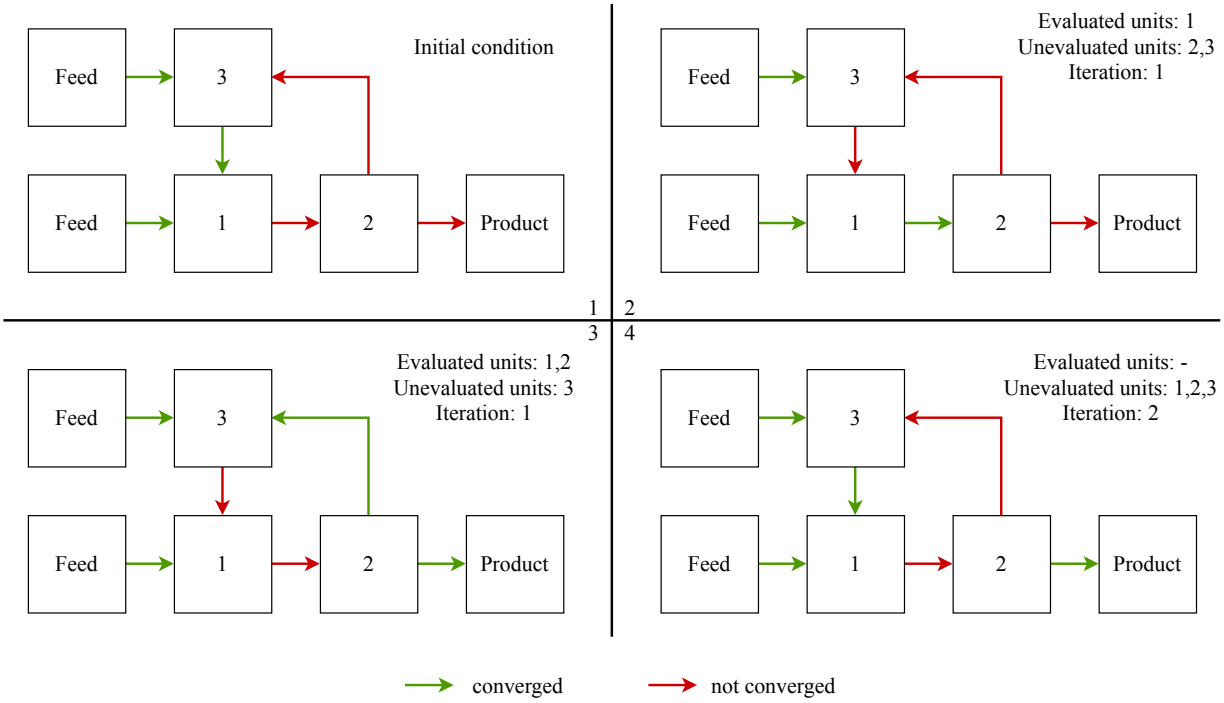


Figure 3.6: Convergence status of the streams during an iteration

The product stream leaving the complex node from unit #2 does not change its status after the first evaluation of the unit. This is intended, as it has no effect on the calculation routine for the complex node.

The calculation continues in a loop until the recycle unit reports that the difference between the torn streams is sufficiently small, which means that the calculation is converged. When this happens, the convergence status of all streams within the complex node is set to true and the calculation of the main flowsheet continues with the next unit.

A problem that can occur during the first iteration is that the algorithm finds, that no units are ready for evaluation. One such possible case is illustrated in figure 3.7. The left side shows the initial condition of the complex node, and the right side the status after the first unit has been evaluated.

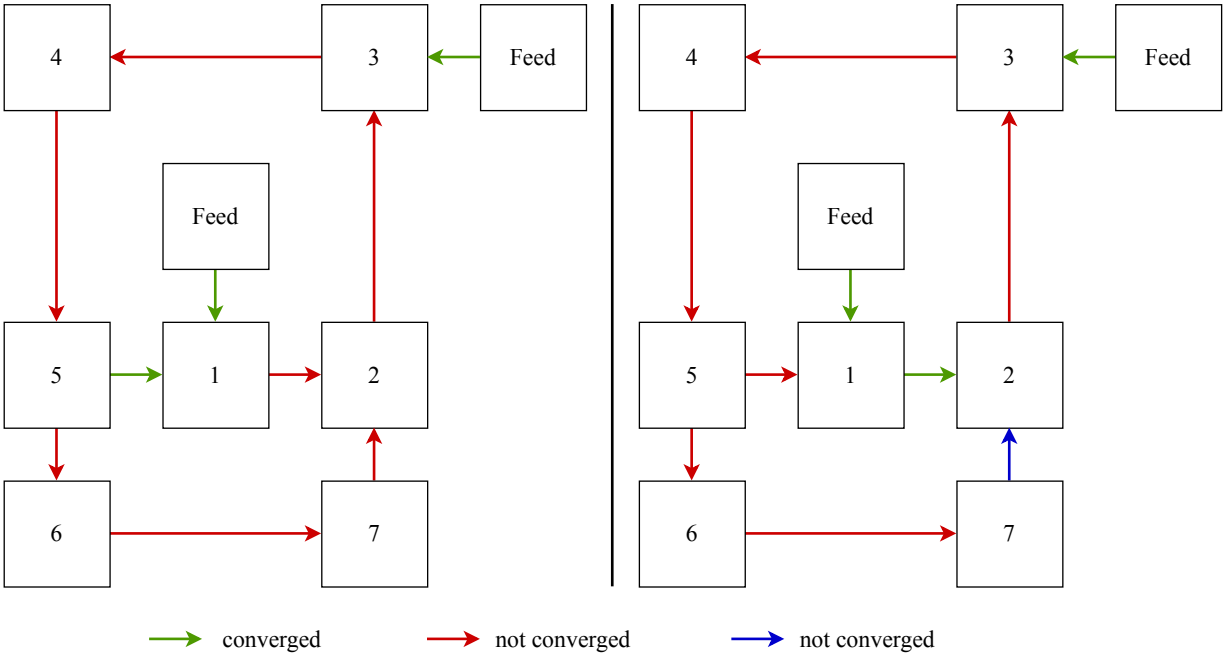


Figure 3.7: A computation which fails to continue after the first unit

The calculation can not continue, because the input stream for unit #2 from unit #7 (marked in blue) is not converged. To prevent the algorithm from getting stuck, the following procedure has been implemented: If the algorithm fails to find any units for evaluation during the first iteration, it is forced to evaluate the next units based on the connections of the output streams of the previously evaluated units. Once every unit has been evaluated, the algorithm will no longer be stuck at the second unit during the second iteration because the stream from #7 to #2 will be converged.

It should be noted that in this case the issue could be circumvented by choosing unit #3 as starting point. Nevertheless, the procedure described above is needed for more complex production plants with many overlapping cycles for which a suitable starting point might not exist.

4 Applications

4.1 Comparison of ANN, Gaussian process and linear regression

In this chapter, the prediction capabilities of an artificial neural network, a Gaussian process and a linear regression for the thermal power of the burner (BWL) in an ethylene cracking furnace are compared.

The predictor variables are:

- The outlet temperature of the cracked gas
- The ratio of steam to hydrocarbons in the furnace
- The mass flow of ethane to the furnace
- The mass flow of a C_4/LPG mixture to the furnace
- The mass flow of residual gas to the furnace
- The volume fraction of O_2 in the flue gas

This adds up to a total of six predictor variables for a single dependent variable.

Figure 4.1 shows the predictions of the different models alongside the measured data for the burner power.

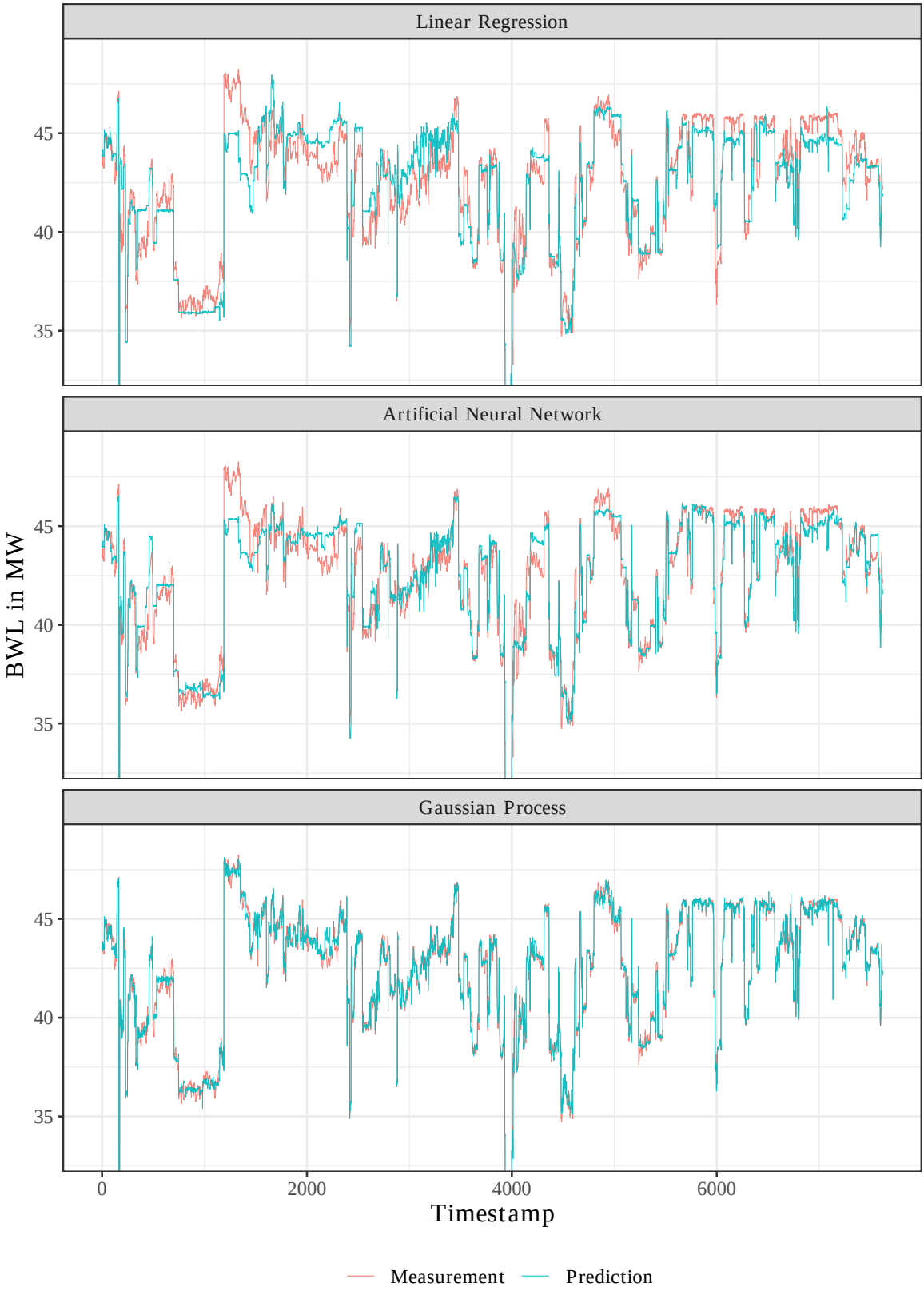


Figure 4.1: Comparison of the different models

The Gaussian process seems to deliver the most accurate prediction. Both, the ANN and the linear regression fail, to correctly predict large parts of the data, especially the sharp increase in power after approximately 1000 timesteps, as well as the jagged plateaus near the end of the data.

Figure 4.2 shows the cumulative deviation from the measurement data for the three models. The prediction of the Gaussian process has the smallest deviation overall, and the cumulative deviation is relatively smooth, which means that the prediction fits quite well across the whole data range, whereas the linear regression shows quite a lot of jumps.

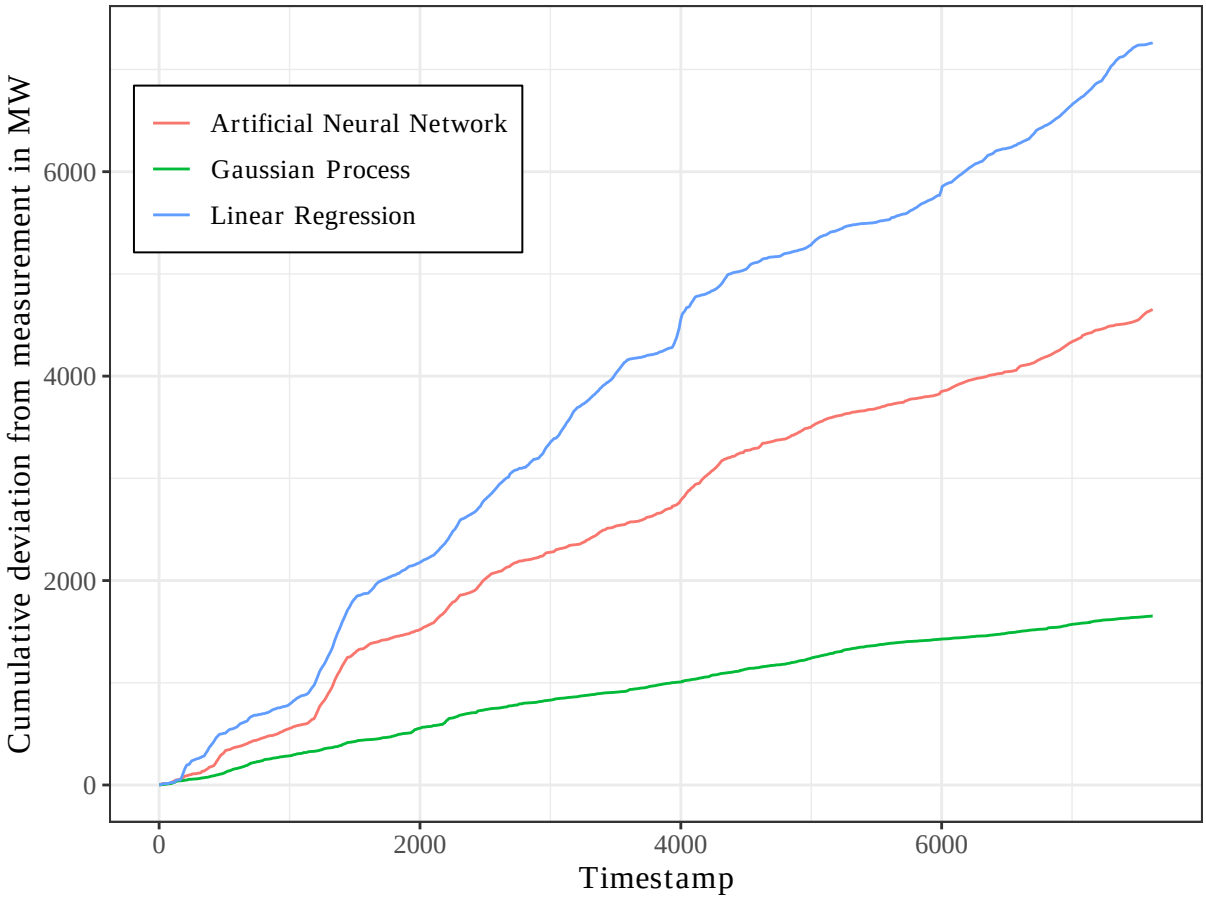


Figure 4.2: Deviation from measurement data for the different models

Although the results favor the Gaussian process in this case in terms of the quality of prediction, there are two severe disadvantages to justify the usage of artificial neural networks instead:

- The time complexity for Gaussian processes is $\mathcal{O}(n^3)$, where n is the number of data points, which makes it almost unusable for big data applications.

- Multi-Output Gaussian processes are not built into Mathematica, and the implementation effort is significant.

4.2 Prediction intervals

The methods for performing homoscedastic and heteroscedastic regression discussed in chapter 2.2.6 are applied to real world data. The following data and predictions describe a product stream of a part of a refinery, consisting of two interlinked distillation columns, which is modelled as a black-box. This chapter aims to investigate how the alpha divergence parameter and the dropout probability affect the prediction of the confidence interval.

Figure 4.3 shows the prediction for different dropout probabilities p with $\alpha = 0.5$ and $K = 20$, where *alpha* is the divergence parameter and K the number of times the the network is sampled.

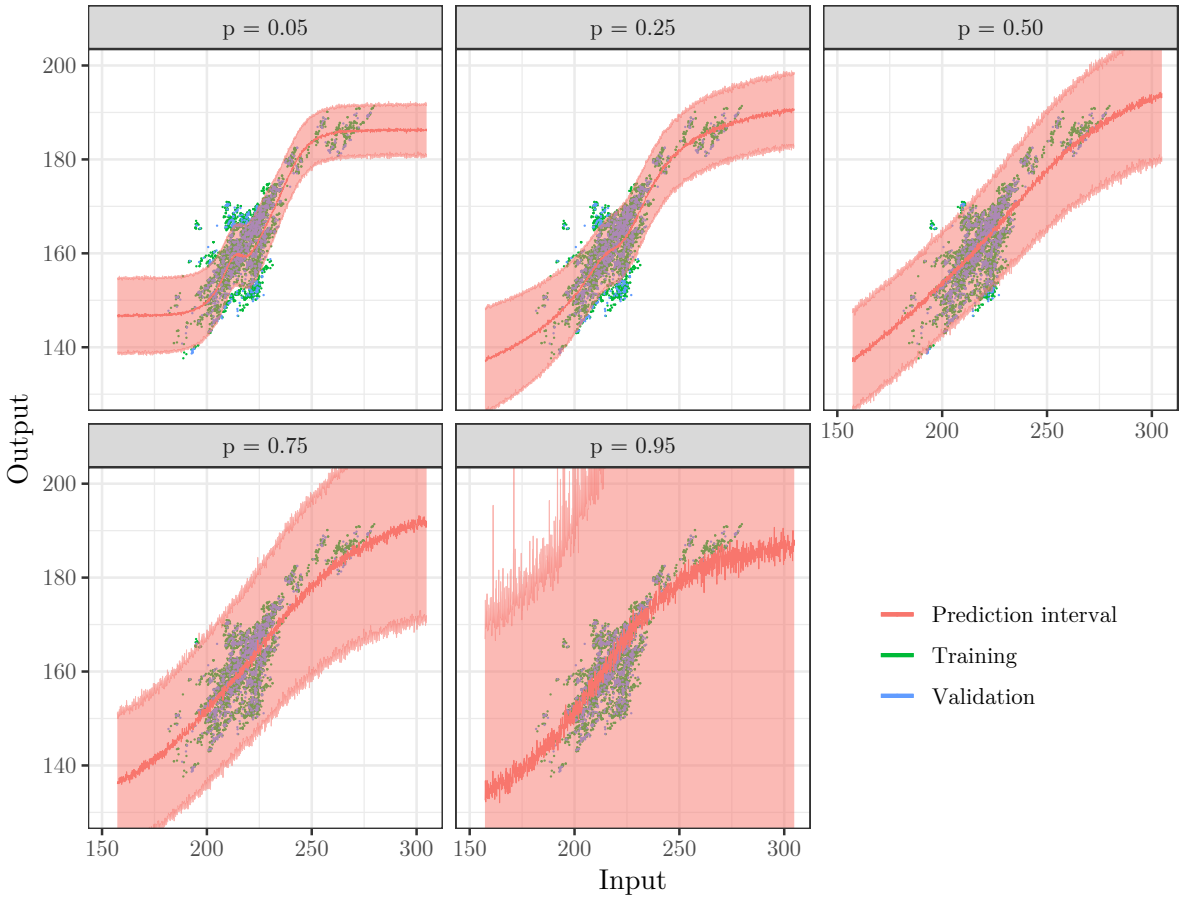


Figure 4.3: Effect of different dropout probabilities p using the α-divergence loss function

Higher dropout probabilities lead to wider error bands. This happens because the high dropout probability disables a lot of neurons during every evaluation, which significantly increases the variance of the prediction. Furthermore, higher dropout probabilities seem to linearize the trend of the prediction, at least for the data used in this example. Another difficulty with high dropout probabilities is the training process. Due to the high deactivation ratio of the nodes in the linear layers, the network is constantly and radically adjusting its weights. While some dropout is beneficial to prevent overfitting, high probabilities reduce the quality of the network.

Figure 4.4 shows a side-by-side comparison of the three different prediction methods with standard parameters found in literature.

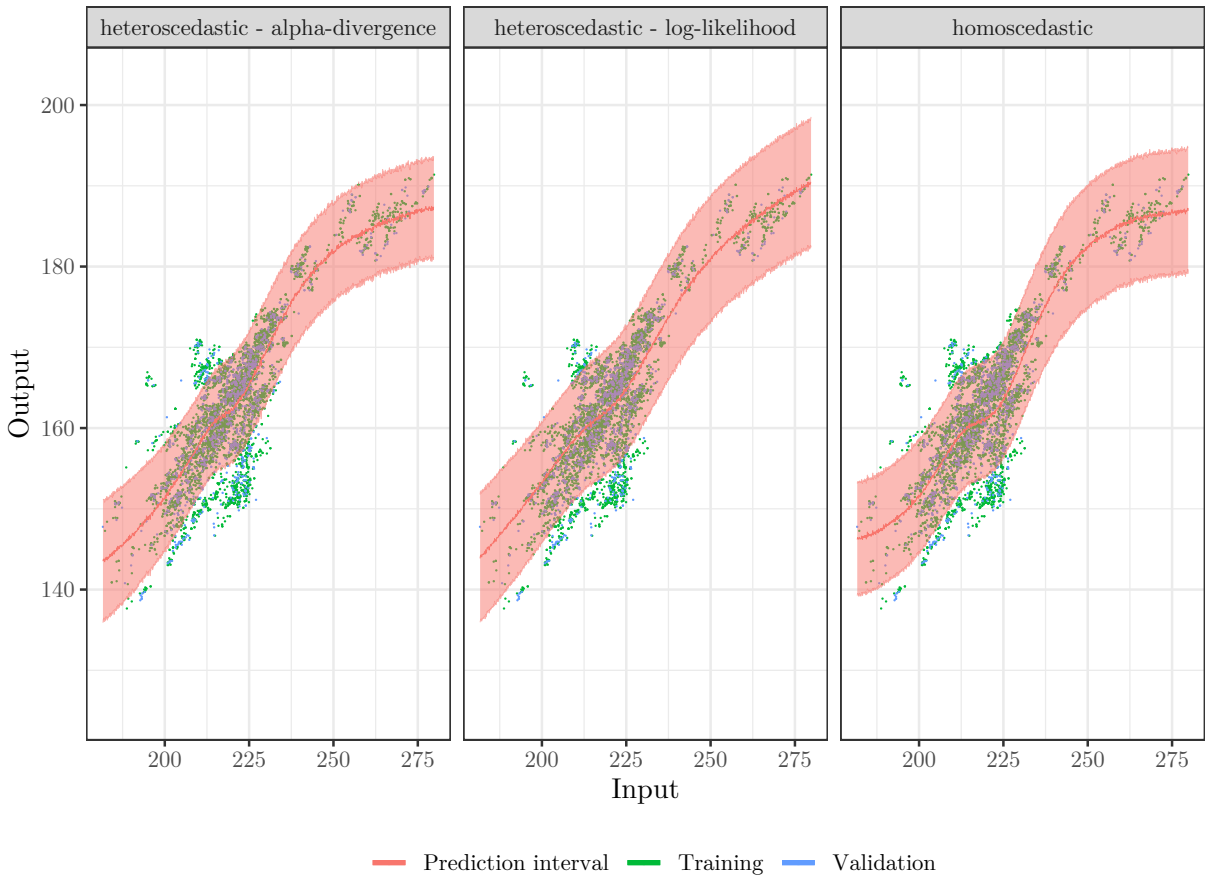


Figure 4.4: Comparison of different methods for modelling prediction intervals

The difference between the prediction intervals seems to be insignificant. Although the α-divergence loss function should lead to a more cautious prediction, the error band is narrower than the one produced by the standard log-likelihood loss function, and is very similar to the one produced by the homoscedastic regression.

In figure 4.5, the prediction for different values of α is shown.

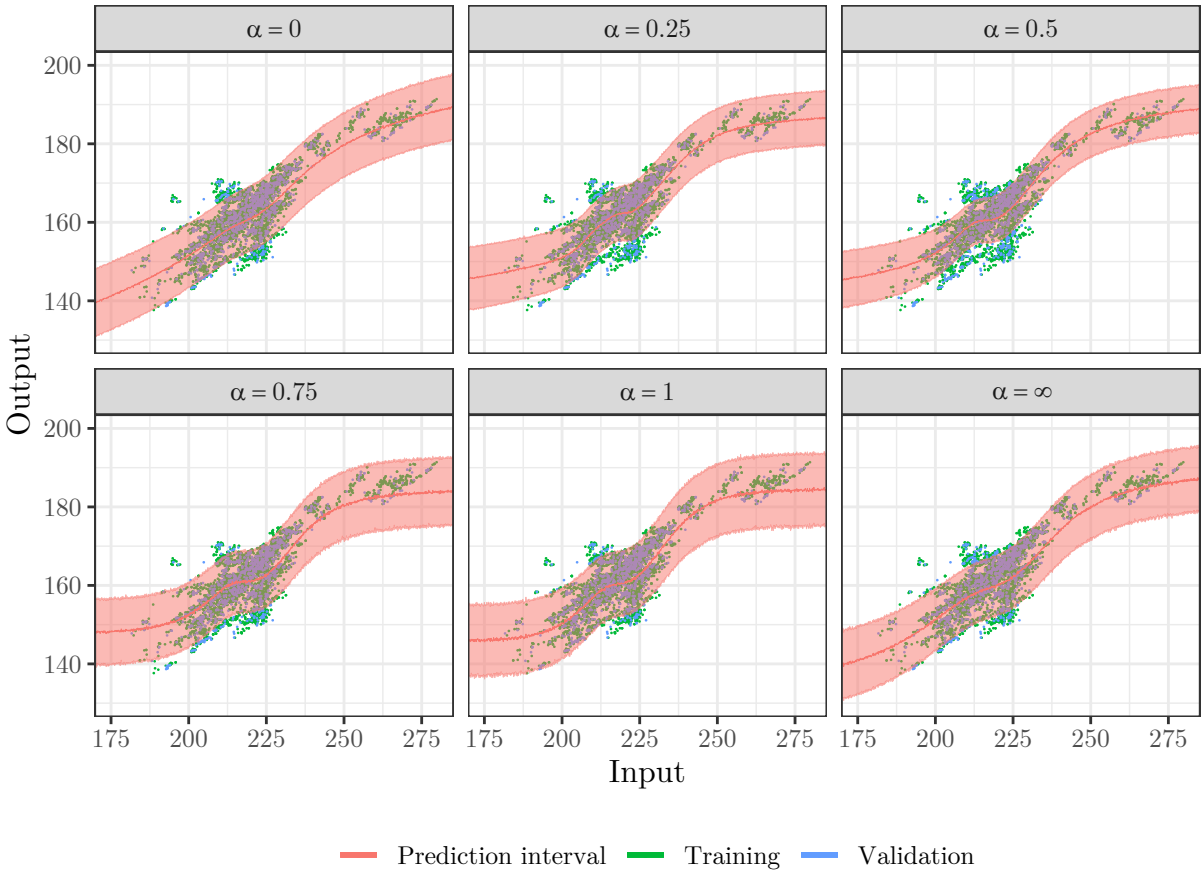


Figure 4.5: Effect of different values for α using the α-divergence loss function

The results do not indicate what would be expected according to theory. Increasing the α-parameter should lead to wider error bands, but they all appear to be very similar. Increasing the value from 0.25 to 0.5 even yields a slightly narrower error band.

A comparison of the different prediction methods and parameter variation shows behaviour that is not in accord with the underlying theory, except for the effect of the dropout probability. This is most likely due to the high variance of the data used in this example. Another possible reason could be that the network structure is sub-optimal. For point estimates in this work, a layer height of about 50 neurons and one to three layers was sufficient. For the prediction interval networks the layer height had to be set to 400 neurons to get reasonable predictions. As there is no tuning procedure, optimal parameters and network structure need to be found by conducting an extensive, comparative study which would require a lot of computational power to be conducted within a reasonable time frame.

4.3 Flash - Unittest

The first application or test case is a simple flash drum, where half of the liquid product stream is recycled. This example is inspired by Biegler et al., where a similar flowsheet featuring multiple components has been used to demonstrate the feasible path optimization. The flowsheet is shown in figure 4.6.

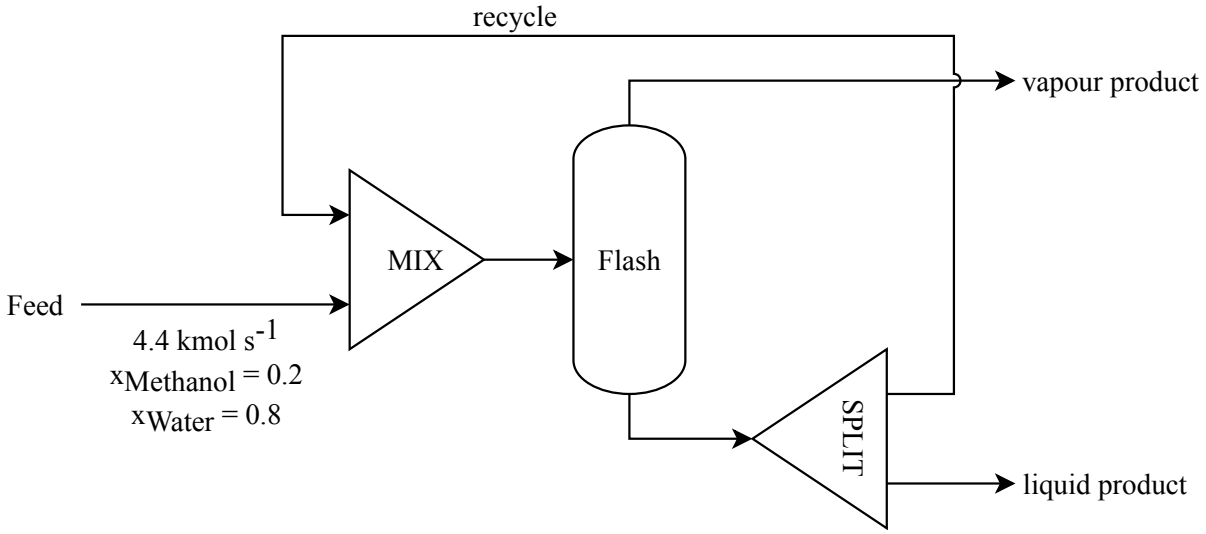


Figure 4.6: Flowsheet for a flash drum with a recycle stream

As can be seen in the figure above, $4.4\,\mathrm{kmol\,s^{-1}}$ of a mixture with a mole fraction of 0.2 for methanol (1) and 0.8 for water (2) is mixed with a recycle stream before being fed into a flash drum.

The flash unit utilizes a neural network, which predicts the mole fraction of methanol in the liquid and vapor phase based on the temperature with a constant pressure of 1 atm. VLE data generated using the NRTL model were used to train the network, where 80 % of the generated data were randomly sampled as training data, and the remaining 20 % were used for cross-validation.

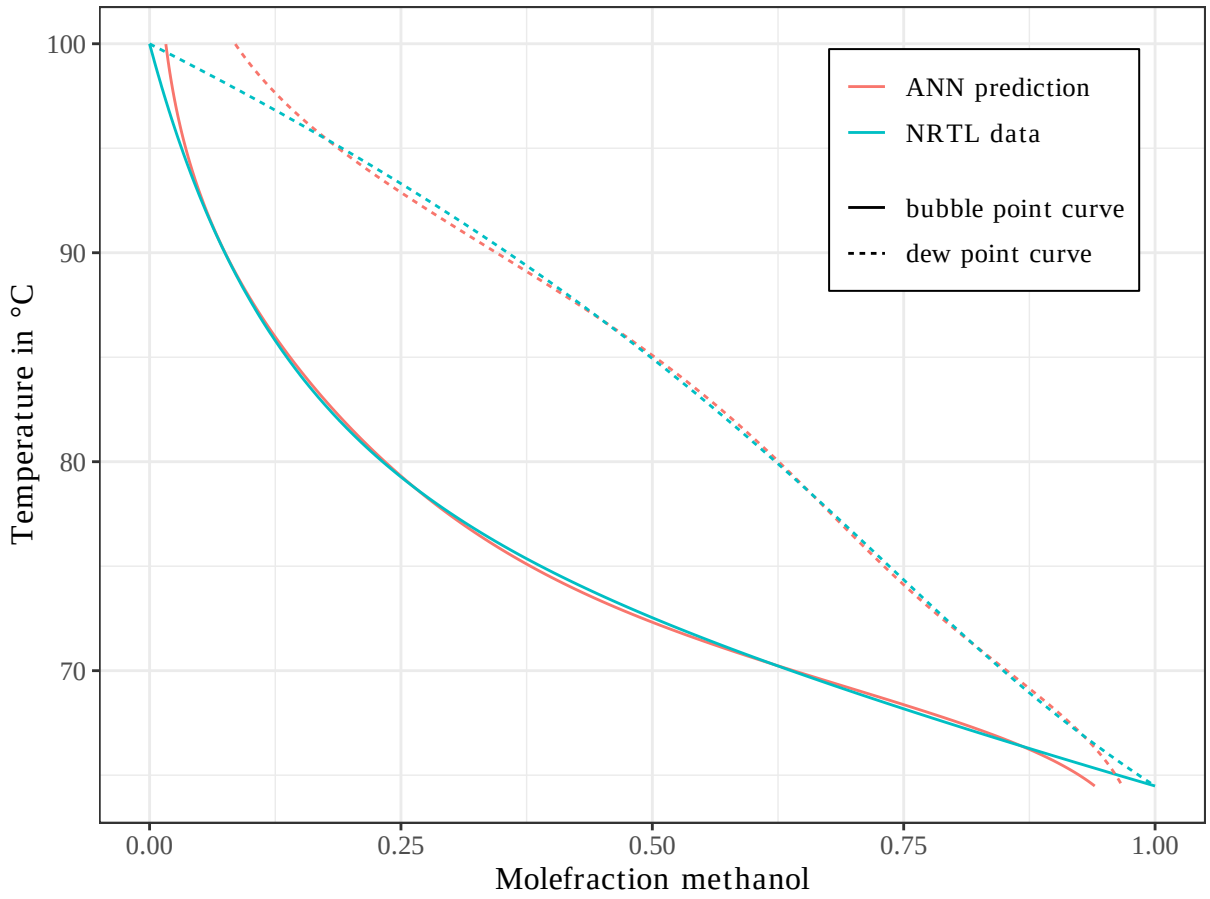


Figure 4.7: T-x diagram for water-methanol at 1 atm

Figure 4.7 shows the prediction of the neural network alongside the NRTL data. The neural network fails to correctly predict the behavior of the system at the low and high ends of the temperature range, namely between 95 to 100 °C and between 64 to 66 °C. Other than that, the bubble and dew curves are in good agreement with only minor deviations.

A simple mass balance is then used to calculate the vapor and liquid streams, as depicted in figure 4.8 and equations 4.1 and 4.2: [38]

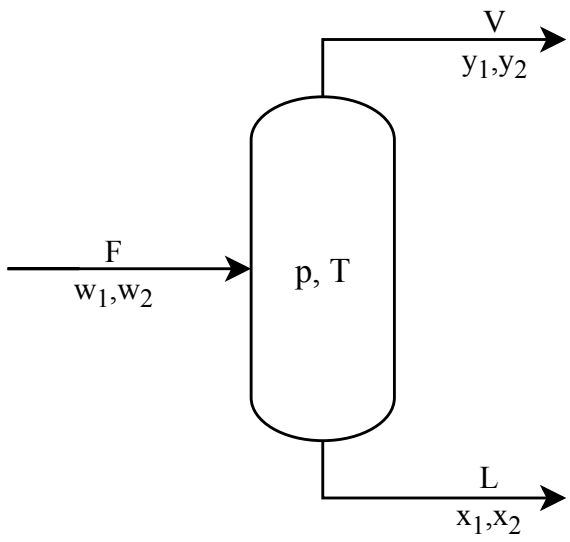


Figure 4.8: Mass balance around the flash drum

$$L = F \cdot \frac{w_1 - y_1}{x_1 - y_1} \tag{4.1}$$

$$V = F - L \tag{4.2}$$

Where w_1 is the mole fraction of methanol in the feed, F is the feed flowrate and L and V are the flow rates of the liquid and vapor stream respectively. The equations 4.1 and 4.2 show how to obtain the liquid and vapor streams from the feed stream and the predicted mole fractions in the vapor (y_1) and liquid (x_1) phase. [38]

The same flowsheet was set up in KBC PetroSIM®, and the results of the simulations at different temperatures are listed in the following table:

Table 4.1: Flash - molar flow rates in $\mathrm{kmol\,s^{-1}}$

	PetroSIM				Neural Network Sim			
	Methanol		Water		Methanol		Water	
Temperature	liquid	vapor	liquid	vapor	liquid	vapor	liquid	vapor
80	0.880	0.000	3.520	0.000	0.880	0.000	3.520	0.000
85	0.494	0.385	3.133	0.388	0.513	0.367	3.156	0.364
90	0.183	0.697	2.260	1.260	0.177	0.703	2.186	1.334
95	0.000	0.880	0.000	3.520	0.000	0.880	0.000	3.520

As table 4.1 shows, the results from both simulations are in good agreement. At 80 °C, the whole product is in liquid form, and at 95 °C it is in vapor form. There are some minor deviations between the results at 85 and 90 °C, which are most likely caused by the deviation of the neural network prediction from the NRTL model.

This test case also shows, that the algorithm for evaluating units, as well as the algorithm for handling recycled streams are working correctly.

4.4 ETBE production plant

During developement, a complex production plant for **E**thyl **t**ert-**b**utyl **e**ther has been modelled as depicted in figures 4.9 and 4.10. The plant features mixers, splitters, reactors and separation (i.e. distillation) units. For this example, no neural network units have been utilized.

The reactors are conversion reactors, therefore for every reaction occuring the stoichiometry, a base component, as well as the corresponding conversion rate needs to be specified . The reactions are evaluated consecutively and the conversion rate is adjusted if a reaction would consume more product than available. The concept is illustrated by the following example:

In a conversion reactor two reactions share one educt component but are based on different components:

$$\begin{aligned} \text{I:} \quad & \mathrm{A} + \mathrm{B} \longrightarrow \mathrm{C} \quad \text{with} \quad X_A = 0.5 \\ \text{II:} \quad & \mathrm{A} + \mathrm{D} \longrightarrow \mathrm{E} \quad \text{with} \quad X_D = 0.2 \end{aligned}$$

Depending on the available ammount of substance for the products, the reactions could consume more product than available and lead to negative molar flows, as depicted in table 4.2.

Table 4.2: Molar flows during the evaluation of the conversion reactor

	$\dot{n}_A$ in $\mathrm{mol\,s^{-1}}$	$\dot{n}_B$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_C$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_D$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_E$ $\mathrm{mol\,s^{-1}}$
initial condition	1	2	0	3	0
after reaction I	0.5	1.5	0.5	3	0
after reaction II	-0.1	1.5	0.5	2.4	0.6

To prevent this from happening, the unit checks for negative flow after every reaction and adjusts the conversion rate. In this case, X_D would be changed from 0.2 to $0.1\dot{6}$ to prevent $\dot{n}_A$ from becoming negative. The results for the adjusted conversion rate are depicted in table 4.3.

Table 4.3: Molar flows during the evaluation of the conversion reactor with adjusted conversion rate

	$\dot{n}_A$ in $\mathrm{mol\,s^{-1}}$	$\dot{n}_B$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_C$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_D$ $\mathrm{mol\,s^{-1}}$	$\dot{n}_E$ $\mathrm{mol\,s^{-1}}$
initial condition	1	2	0	3	0
after reaction I	0.5	1.5	0.5	3	0
after reaction II	0	1.5	0.5	2.5	0.5

It should be mentioned that the basis for the conversion rate is always the flow rate before any reaction occured.

The distillation units are implemented as simple component splitters. For every component, the user specifies the fraction of the component in the feed that should go to the top stream. The remaining ammount is then assigned to the bottom stream.

The parameters for the reactors and separation units, as well as the initial conditions of the feed streams have been taken from an existing PetroSIM simulation case, depicted in figures 4.9 and 4.10.

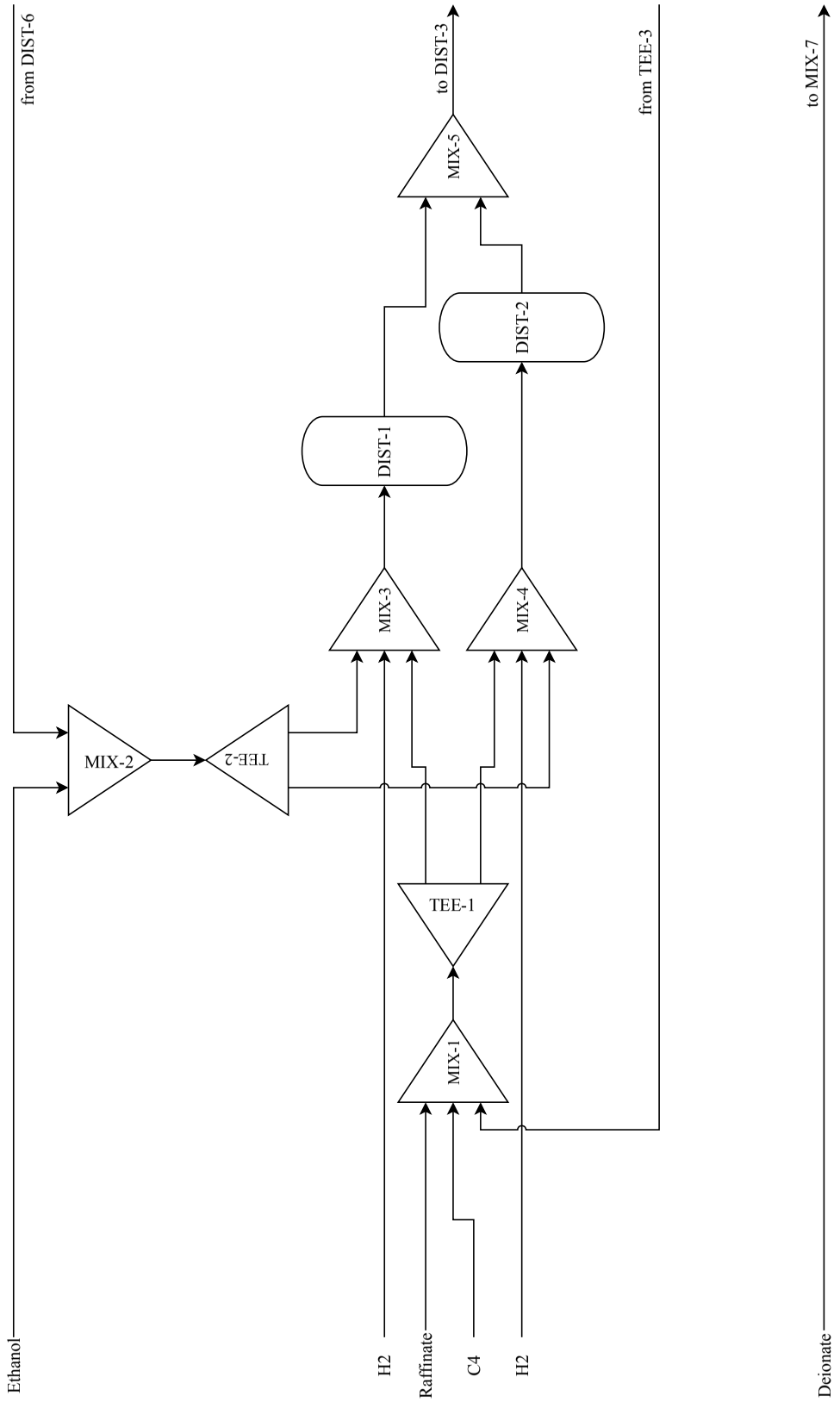


Figure 4.9: ETBE flowsheet - part 1

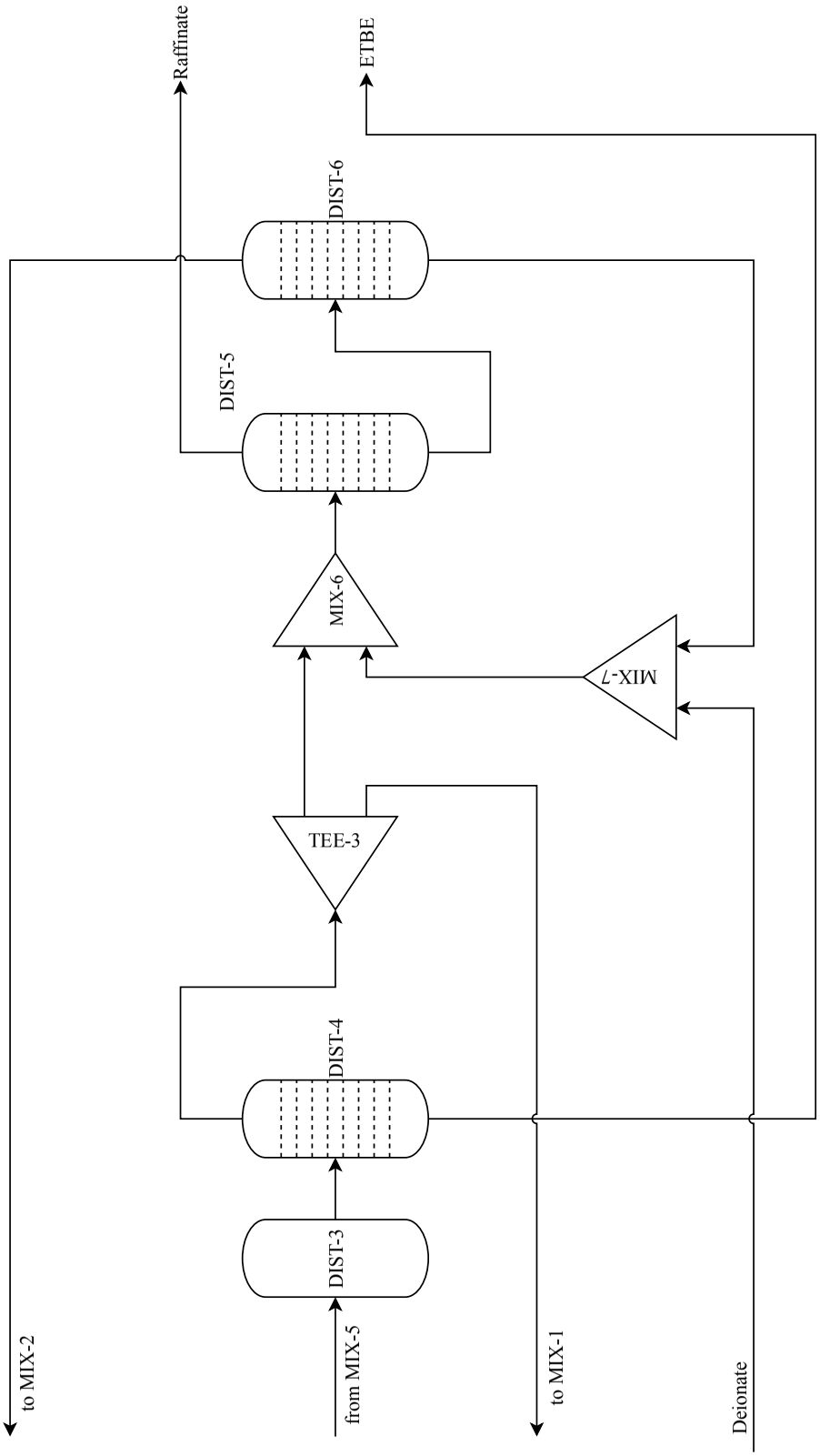


Figure 4.10: ETBE flowsheet - part 2

4.4.1 Comparison with KBC PetroSIM®

The simulation results from this work and the existing KBC PetroSIM® simulation are in good agreement. For the ETBE product stream, KBC PetroSIM® returns a molar flow of $165.10\,\mathrm{kmol\,h^{-1}}$ with a molar ETBE fraction of 0.9537, and the results using the simulation framework presented in this work are 165.99 with a molar fraction of 0.9535. This equals a 0.54 % deviation for the flow rate, and a 0.021 % deviation for the molar fraction.

4.5 Refinery

Parts of a refinery are modeled using neural networks. The plant consists of multiple, distinct areas which are shown in figure 4.11:

SPL1: This is a separation area consisting of a main separation column and a side column, which is fed with a side cut of the main column. One reboiler is a heat exchanger powered by low pressure steam, and the other one uses a methane burner. The whole area has two outlets, one connects to the ethylene cracking (SC) part, and one is the feed for the NHT part.

NHT: This area contains a reactor and a downstream separation column. The main input is coming from SPL1, and some excess gas from REF1 and low pressure steam condesate are mixed into the feed. After the reactor, the excess hydrogen is fed back into the refinery's hydrogen system. The bottom of the separation column is the feed for the SPL2 area. The top stream is used in another part of the refinery, which is not part of this work. This area includes two methane burners, one to preheat the mixture before the reaction and one for the reboiler of the column.

SPL2: This area consists of a separation column. The bottom is the feed for the REF1 part, while the head is split up between a fuel depot and the ISOM part of the refinery. The reboiler is a heat exchanger using medium pressure steam.

REF1: This area consists of five reactors along with two separation columns. The only input is coming from SPL2. The main product/output is reformate. Apart from that, the unit produces stabilizing gas and liquid gas. Excess gas is fed back into the NHT part.

ISOM: This unit takes the light naphta from SPL2 along with a external hydrogen stream. It has several outputs of different kinds of fuel.

SC: Here, the naphta from SPL1 is mixed with several different additional feed streams and distributed to 10 steam cracking furnaces. The product gas mixture is then fed into the gas separation unit. The furnaces run on methane burners and generate a large ammount of high pressure process steam.

Gas separation: The product gas is separated and parts of the C_4 streams are fed back into the SC part. This process step consumes a lot of energy in form of high pressure process steam.

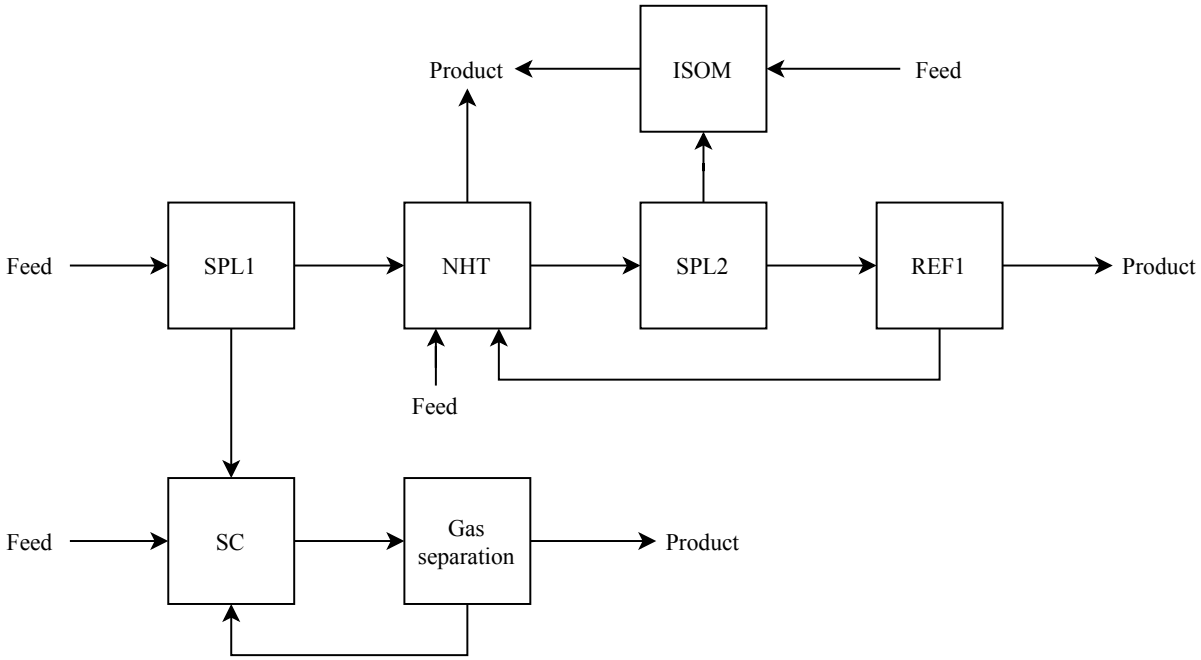


Figure 4.11: Process flow diagram for the refinery

4.5.1 Models

For the fuel line units (SPL1, NHT, SPL2, REF1, ISOM), the relevant measurement points have been pulled from the piping and instrumentation diagrams. Data were not available for all the required streams, either due to absent measurement points or faulty data, but the missing streams could be calculated using mass balances and neglection of minor streams. Unfortunately, measurement points for composition/quality are sparse and thus could not be included in the models. After the calculation of the missing streams, the raw data were filtered for steady states before points with an outlier probability above 20 % according to the LoOP method are removed. The filtered data were then prepared for the mass balance reconciliation by applying a mass balance filter. This filter finds the mode of the mass defect percentage and removes data points that stray further than 5 % from the modal

value. Then, the mass balance reconciliation is applied which uses a weighted approach to modify the flow rates in a way, that the mass balance for the unit in question is fulfilled. The last step is the sampling of the data. Using the SIBSS method and - if possible - the concept of the convex hull, 75 % of the data were sampled into the training set, 20 % were sampled into the validation set and 5 % remain for additional testing if necessary.

For the ethylene cracking part of the refinery, every furnace and splitter has been modelled individually. The composition of the feed streams, as well as the product composition is known from the available data and included in the models. Mass balance reconciliation is not necessary, as the furnaces only have one inlet and one outlet stream. Besides the quantity and composition of the input, the coil outlet temperature is used as a predictor variable. Depending on the type of furnace, the feed mixture consists of one to four different components, thus the number of predictor variables varies between different types of furnaces. The dependent variables are the mass fractions of the 17 products, the thermal power of the burners, the volume fraction of O_2 in the flue gas, the mass-based ratio of steam to hydrocarbons and the amount of produced high pressure process stream. The separation efficiency of the gas separation is almost perfect in real-life, but the energy demand is very high. For that reason, the actual separation is modeled as a component splitter, which completely separates the incoming mixture into several pure component streams. The energy demand is modeled using a neural network, which predicts the amount of imported high and medium pressure steam and the amount of low pressure steam that is exported based on the amount and composition of the product gas stream.

4.5.2 Multi-objective optimization

For the optimization demonstration, the objective functions are based on the material and energy balances. Every species has been assigned a monetary rating, and based on the mass flow and composition of the input and output streams a monetary plant performance rating can be calculated. This also includes the import and export of steam, as well as the energy consumption in form of heating gas (CH_4). The performance rating is to be maximized, but the NSGA-II can only minimize objectives. This limitation is circumvented by taking the negative of the performance rating, this means that after optimization, the sign of the objective needs to be switched again.

All energy that is imported and exported has been transformed into a CO_2 equivalent mass flow, which makes the second objective a measure for the environmental impact that needs to be minimized.

The independent variables are the coil outlet temperatures for all of the ethylene cracking furnaces, as well as the mass flow of the input streams. The furnace temperature can take values between 800 and 850 °C, and the input mass flows can take values from 10 % below to 10 % above their respective median values.

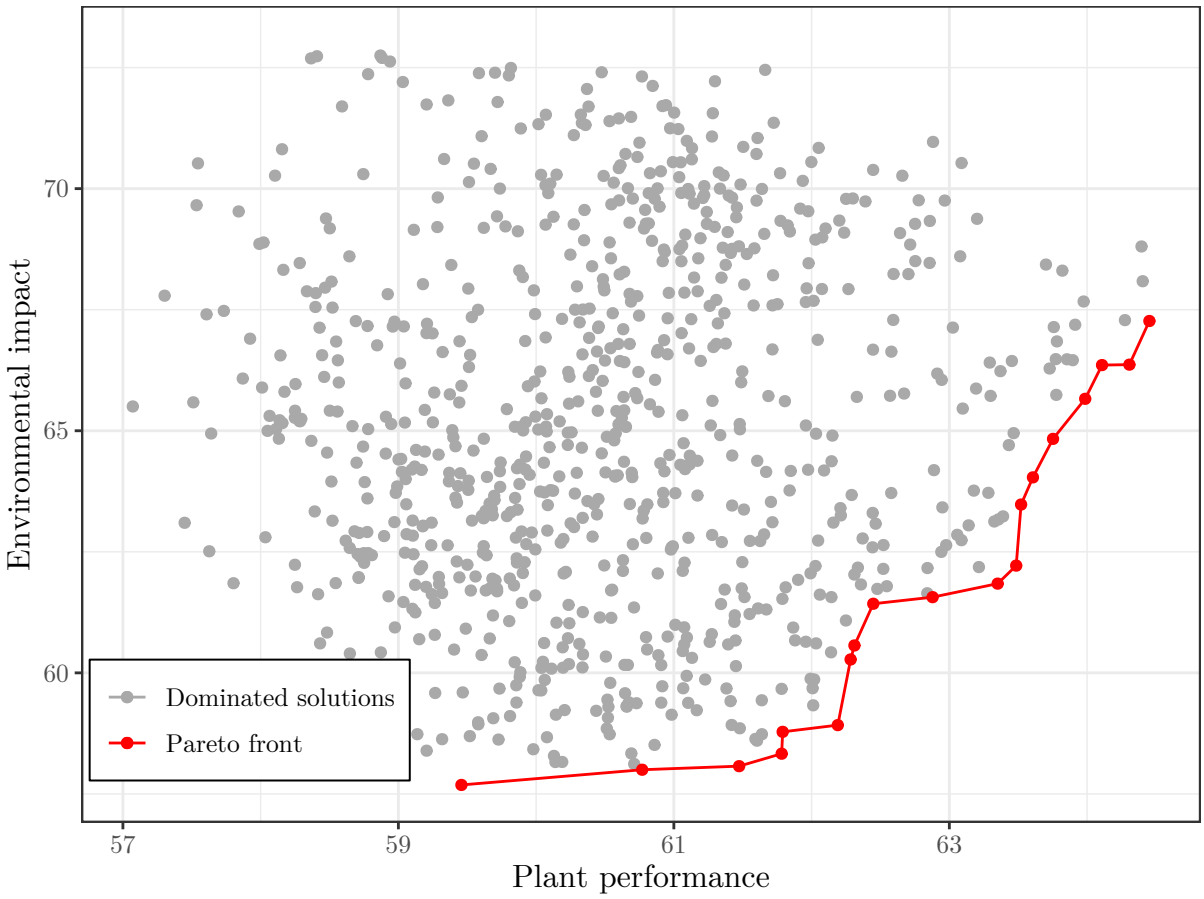


Figure 4.12: Optimization results utilizing NSGA-II

Figure 4.12 shows the result of the optimization, which contains 822 unique solutions of which 19 are pareto-optimal.

Figures 4.13 to 4.16 show the values of the target functions over the input variable values. The red circles mark the pareto-optimal solutions. The first five furnaces all have their optimal solutions either at the upper or lower temperature bound, while the optimal solutions for the remaining five furnaces are scattered across the temperature range.

As for the refinery input streams, pareto-optimal solutions can be found across the whole range of all inputs, except input #4 and #9, which seem to favour a lower mass flow. Also, mass input #1 shows a strong correlation with the environmental impact of the plant. This is to be expected, because it represents the largest input and by that the downstream energy demand is affected most.

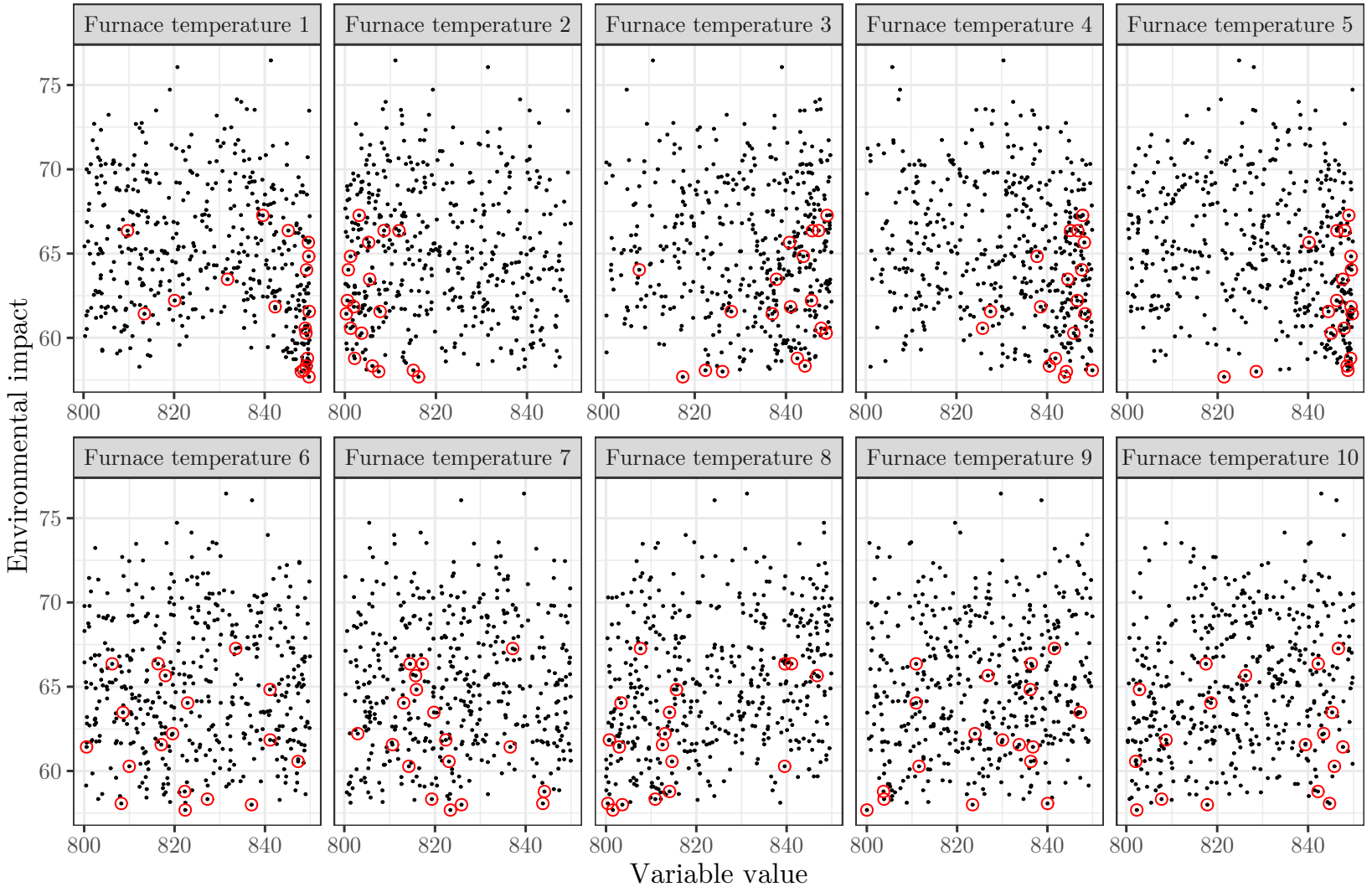


Figure 4.13: Environmental impact over furnace temperatures

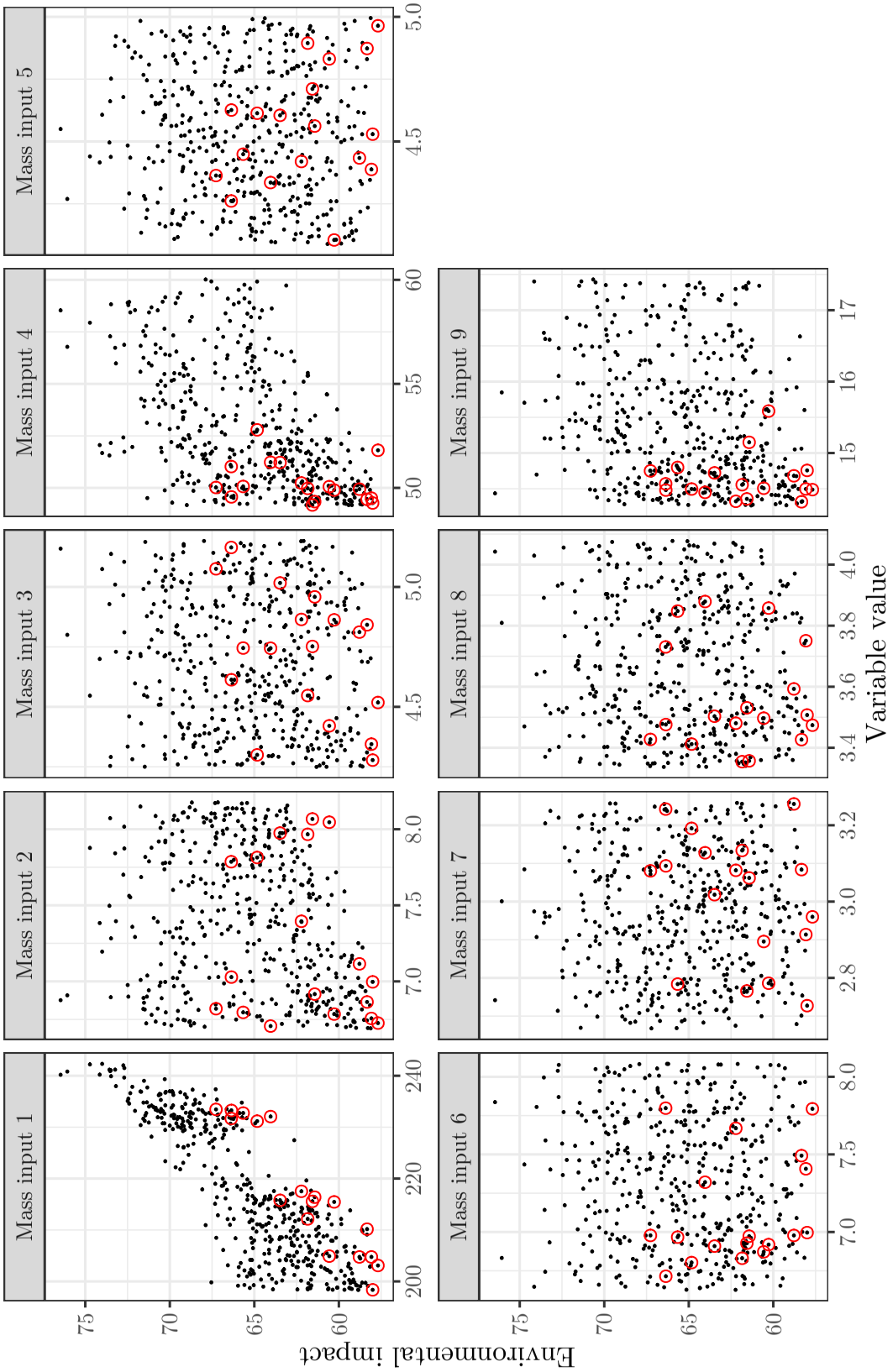


Figure 4.14: Environmental impact over feed rates

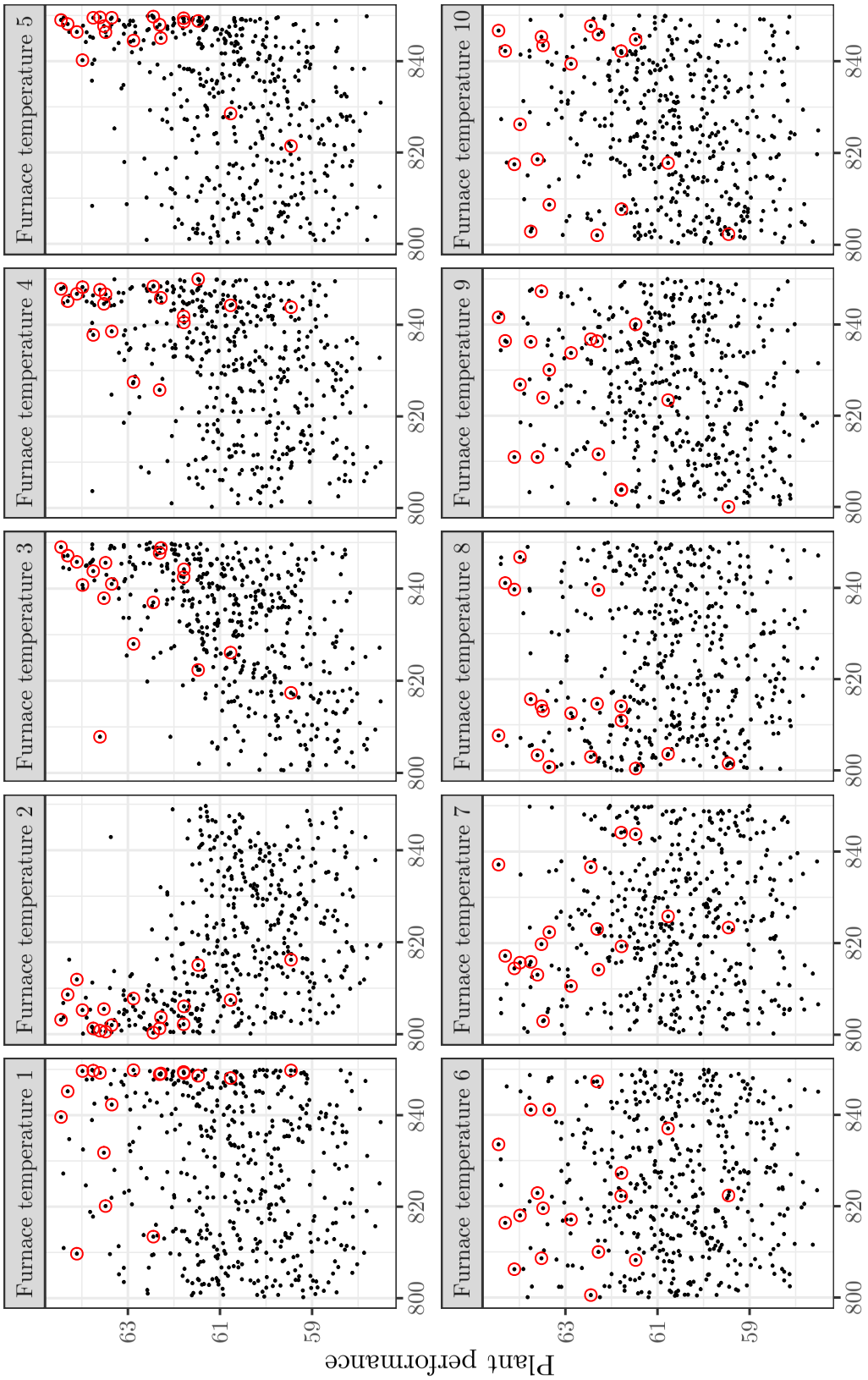


Figure 4.15: Plant performance over furnace temperatures

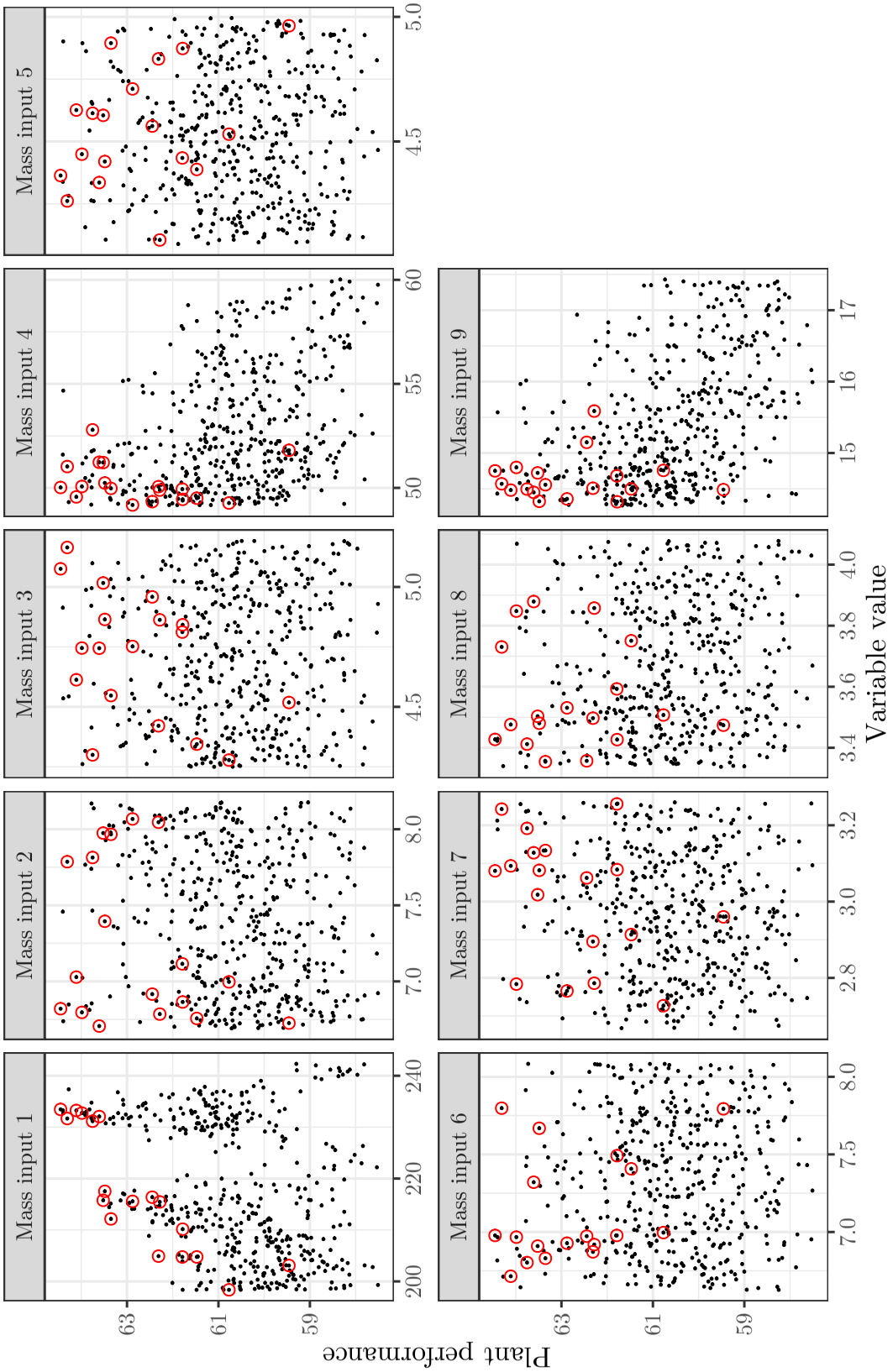


Figure 4.16: Plant performance over feed rates

4.5.3 Comparison with measurement data

To compare the simulation against real data, only parts of the fuel line units are used, i.e. SPL1, NHT and SPL2. The reduced model has only two input streams, the feed for SPL1 and a condensate flow to the NHT part.

The unit models are trained and validated on reconciled measurement data. The input data for the simulation as well as the output data which were used for comparison with the simulation results have also been reconciled. The reconciliation is of high importance, as the measurement data is found to have mass defects of up to 20 %.

The comparison shows, that streams with higher mass flows can be predicted more accurately, as the median of the deviation for the main product line lies between 2 % and 5 %. The median of the deviation from the simulation data to the measurement has been chosen as a measurement to quantify the accuracy of the predictions. The reason for the usage of the median instead of the mean or a cumulative deviation is, that some streams have their flow rate reduced to almost 0 for single data points during reconciliation. The deviation D is calculated with respect to the measurement data:

$$D_i = \frac{|\hat{y}_i - y_i|}{y_i} \tag{4.3}$$

In the equation above, $\hat{y}_i$ is the predicted value from the simulation and y_i the measurement value. If a single data point has its measurement value greatly reduced to fulfill the mass balance, the relative deviation is greatly increased, up to values of 10^7 %. The mean value would be influenced significantly by these points, thus the median value has been chosen for comparison.

Figure 4.17 shows the comparison between the simulation results and the measurement data for three exemplary streams. The streams with the highest and lowest deviation, as well as a mid-range stream have been selected for this purpose.

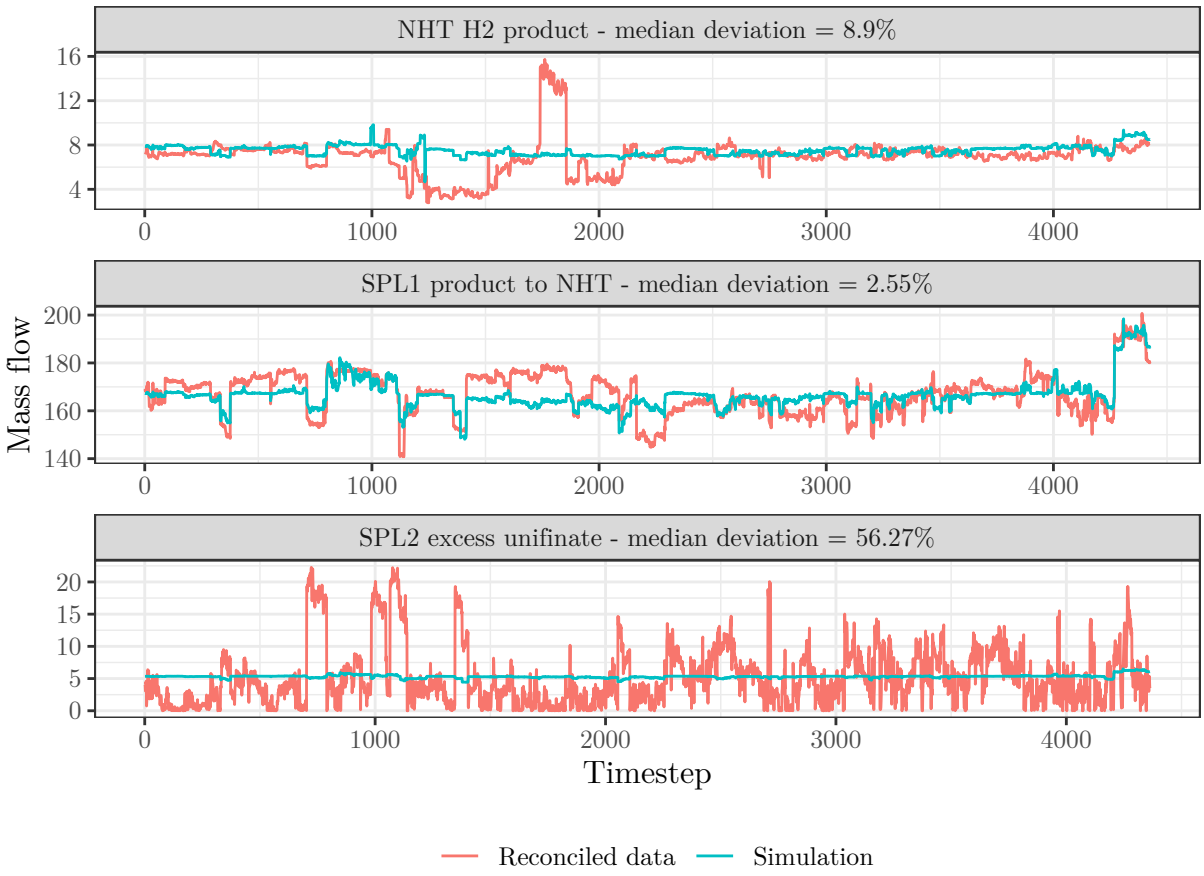


Figure 4.17: Comparison between measurement data and simulation results for three exemplary streams

The differences in prediction quality are most likely due to differing qualities of the underlying data and the structure of the models themselves. For example, the SPL2 unit uses the feed mass flow as a single predictor variable to predict four output streams as well as the steam consumption of the reboiler. Furthermore, the data used to train the model contained a higher number of outliers as well as larger mass defects than the other units.

4.5.4 Comparison with MILP

The simulation results of the FlowsheetSolver-package are compared to the results of an existing mixed integer linear programming approach. First, the prediction qualities of the standalone units, i.e. just one singular unit and not the whole flowsheet, are compared. The same input values are fed into the neural network and the linear regression function for the corresponding unit. The output is then compared to the expected value from the reconciliated dataset.

The data used for the training of the MILP and ANN models come from different datasets. The neural networks are trained independently, so the units do not need a common dataset. The MILP model is trained on data for the whole plant, so all the units are trained on data coming from the same dataset.

During development of the neural networks, two separate data sets were used. Data for the fuel line, i.e. every unit except SC and the gas separation, was taken from a dataset from 2018. This dataset did not include any data for the steam cracking line, so a different dataset from 2016 had to be used for training.

The MILP models are trained on a different dataset, containing data for the fuel line and the stream cracking line from the year 2016.

An overview of the datasets is given in table 4.4. The comparison for the single units is based on the dataset which has been used to train the artificial neural networks, as it is the largest available dataset. The comparison for the whole plant needs to be based on the dataset which was used to train the MILP models, as it is the only available dataset which contains both product lines.

Table 4.4: Overview of the datasets used for training and comparison

	Content	Year	ANN training	MILP training	Comparison
Dataset 1	Fuel line w/o SC	2018	yes	no	Single units
Dataset 2	SC only	2016	yes	no	-
Dataset 3	Fuel line and SC	2016	no	yes	Whole plant

Single units: Figure 4.18 shows the LPG output of the NHT unit. To make the plot more readable, the value triplets have been sorted by the expected output value in ascending order instead of ascending timestamps.

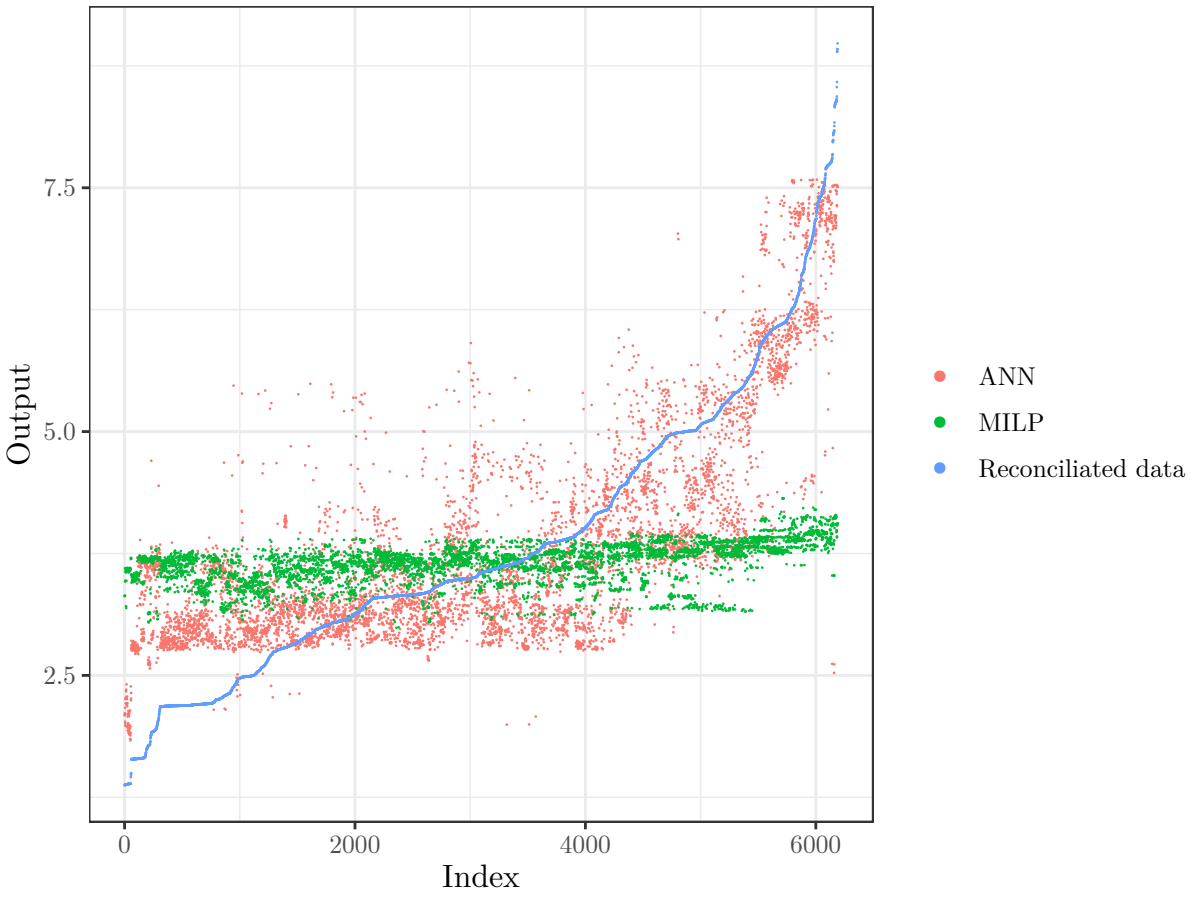


Figure 4.18: LPG output of the NHT unit - comparison between ANN, MILP and real data

The prediction of the MILP approach remains fairly constant over the whole data range, and so it fails to correctly predict the output below index 2000 and above index 4000. The artificial neural network also fails for data below index 2000, but it correctly predicts the higher output values starting at index 4000. In this case, the ANN is superior to the MILP model.

Figure 4.19 shows the comparison of the outputs from the REF1 models.

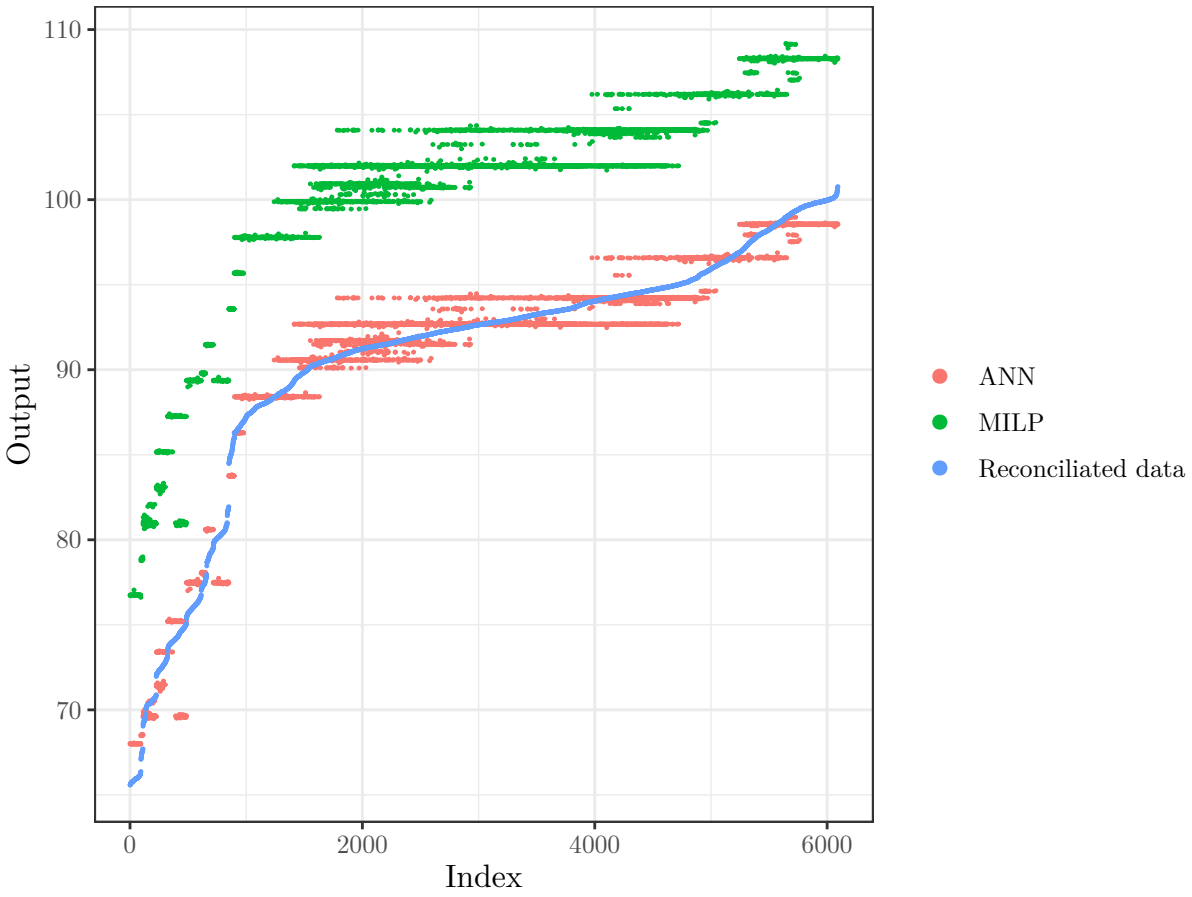


Figure 4.19: Output of the REF1 unit - comparison between ANN, MILP and real data

The plot shows areas, where the output of both models is constant over several indices, which manifests as horizontal lines. The consistency of these data points comes from constant input values. The general shape of the predictions of both models is similar, but the MILP values are constantly overpredicting the output, so the curve seems shifted upwards. Due to the overprediction of the MILP, the ANN is superior for this unit.

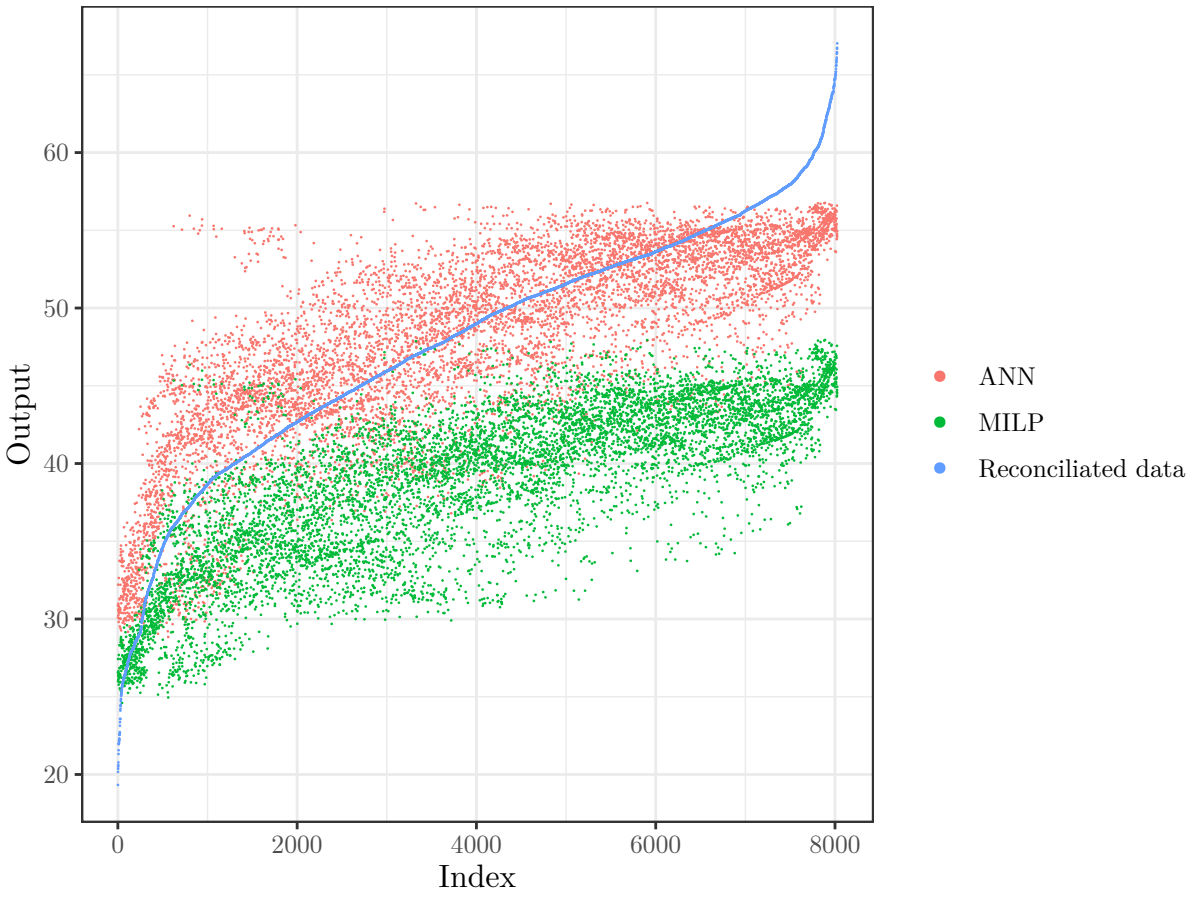


Figure 4.20: Output of the SPL2 unit - comparison between ANN, MILP and real data

Figure 4.20 shows the comparison of the outputs from the SPL2 models. In this plot, the predicted values show a lot more variance for both models than the previous two examples. This is due to a higher variance in the input values for the models. Both models fail to correctly predict the output for the data points at the upper bound of the index range. The MILP model is underpredicting the output of this unit across all data points, except at the lower indices. Again, the shape of the predictions of the models is very similar, and the MILP model predictions seem to have an offset. For this unit, the prediction capability of the ANN is again superior to the MILP model.

The comparison of the prediction quality of the artificial neural networks and the mixed integer linear programming models for several single units shows, that the artificial neural network is the superior model in most cases. One needs to consider that the data to which the models are compared comes from the same data set that was used to train the ANN, but not the MILP.

Production network: Figure 4.21 shows the comparison between the predictions and the reconciliated data for the platforming product (output of REF1), the ethylene product (output of SC) and the stream from SPL2 to NHT. In this case, the whole refinery has been simulated, which means that errors propagate from unit to unit.

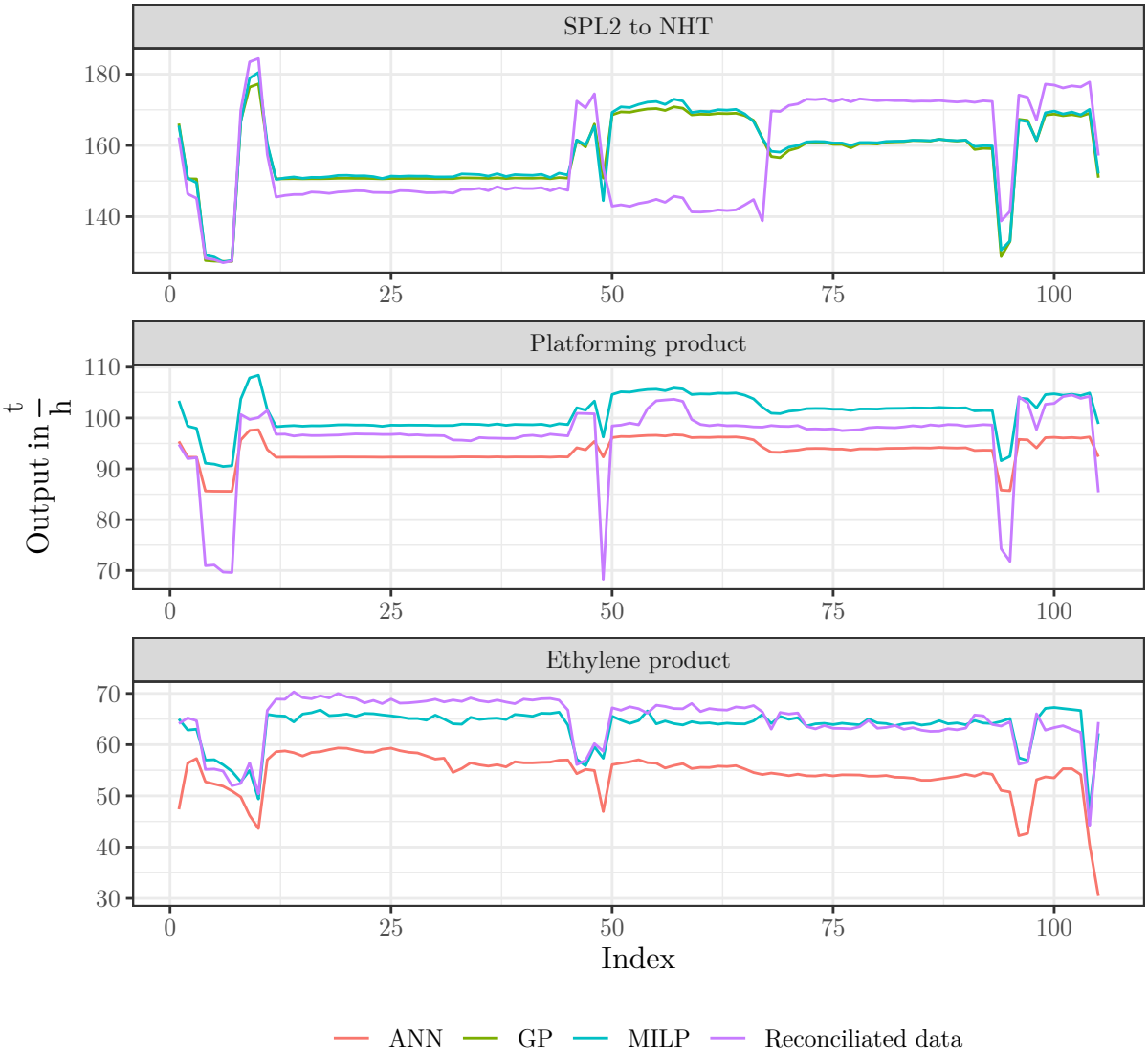


Figure 4.21: Whole plant simulation - comparison between ANN, MILP and real data

The SPL2 unit has been modelled as a Gaussian Process using the same data the MILP was trained on. The predictions of the GP and the MILP model are almost identical for the SPL2 unit. For the platforming product and the ethylene product, the MILP model seems to predict the data more accurately than the ANN. The data that the model output is being compared to stems from the same dataset that was used to train the MILP model.

5 Conclusion & Outlook

The simulation framework presented in this work lays the foundation for future research in the field of data driven modelling for chemical engineering applications. The implementation of new units is straightforward and allows for the utilization of Wolfram Mathematicas' extensive modelling toolkit.

A simple comparison between the prediction capabilities of classical linear regression, an artificial neural network and a Gaussian process using the example of the burner power of a steam cracking furnace in dependence of six predictor variables was conducted. This comparison has shown, that the Gaussian process is superior to both the ANN and the linear regression, with the linear regression showing the worst accuracy of the three. Further studies comparing the prediction capabilities of ANNs and GPs in different chemical engineering applications are necessary to determine the preferable modelling aproach.

The investigation of the prediction intervals of artificial neural networks with real measurement data has shown behaviour, that is only partially in agreement with the underlying theory. This is most likely due to the high amount of noise in the data or an insufficient height or depth of the regression network. Comparative studies would be required to find the optimal network structure and hyperparameters to model uncertainty.

The modelling of the ETBE production plant and the comparison of the simulation results to those of an equivalent simulation case in PetroSIM have shown, that the simulation framework and the automated handling of recycle streams are implemented correctly and work as expected. Furthermore, the successful simulation of a flash drum with a recycle stream using artificial neural networks proves the feasibility of artificial neural networks in grey-box modelling, provided that the quality of the available data is sufficient.

Comparing the simulation results of a complex refinery to the expected output shows, that the prediction of some units is superior to others in terms of accuracy. The reason for that, is that the quality and quantity of the data used to train the models varies from unit to unit. The data of some units shows a relatively large quantity of outliers, as well as higher and more frequent mass defects. The effects of bad data can be reduced by data preprocessing, but they still effect the quality of the models as the comparison shows.

By combining the simulation framework with the NSGA-II genetic algorithm, a new way of using data-driven modelling for plant optimization has been demonstrated.

Although Gaussian Processes suffer from high computational effort and memory requirements, they could prove to be a worthwhile replacement for artificial neural networks. Gaussian Processes have not been tested extensively in this work, but the prediction quality surpassed those of artificial neural networks. Another advantage is that they provide a full probabilistic model, which also quantifies the uncertainty of the predictions. By using sparse Gaussian Processes, which rely on lower-dimensional representations defined by a

smaller set of "inducing points" to represent the full Gaussian Process, the computational effort and memory requirements can be significantly reduced [39]. It is also possible for one Gaussian Process to model multiple output variables [40]. Unfortunately, this functionality is missing from Mathematica and would require some effort to implement.

6 Appendix

List of Tables

List of Figures

Bibliography

[1] Nishanth G. Chemmangattuvalappil, Chien Hwa Chon, and Denny Ng Kok Sum. *Chemical Engineering Process Simulation*. Elsevier Science, San Diego, CA, USA, 2017.

[2] Christoph Molnar. *Interpretable Machine Learning*. Lulu.com, 2020.

[3] Guillaume Dubois. *Modeling and Simulation*. CRC Press, 2018.

[4] Anne Katrine Duun-Henriksen, Signe Schmidt, Rikke Meldgaard Røge, Jonas Bech Møller, Kirsten Nørgaard, John Bagterp Jørgensen, and Henrik Madsen. Model identification using stochastic differential equation grey-box models in diabetes. *Journal of diabetes science and technology*, 7(2):431–440, 2013.

[5] Nazri Mohd Nawi, Walid Hasen Atomi, and M. Z. Rehman. The Effect of Data Preprocessing on Optimized Training of Artificial Neural Networks. *Procedia Technology*, 11:32–39, 2013.

[6] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. LoOP. In David Cheung, Il-Yeol Song, Wesley Chu, Xiaohua Hu, and Jimmy Lin, editors, *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, page 1649, New York, New York, USA, 2009. ACM Press.

[7] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[8] Michael J. Grimble, Michael A. Johnson, Daniel Sbárbaro, and René del Villar, editors. *Advanced Control and Supervision of Mineral Processing Plants*. Advances in Industrial Control. Springer London, London, 2010.

[9] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. On Sampling Strategies for Neural Network-based Collaborative Filtering. In Stan Matwin, Shipeng Yu, and Faisal Farooq, editors, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 767–776, New York, NY, USA, 08132017. ACM.

[10] M. G. M. Khan and Jacek Wesołowski. Neyman-type sample allocation for domains-efficient estimation in multistage sampling. *AStA Advances in Statistical Analysis*, 103(4):563–592, 2019.

[11] Mirosław Kordos. Data Selection for Neural Networks. *Schedae Informaticae*, 1/2016, 2017.

[12] H. Khosravani, A. Ruano, and P. M. Ferreira. A Comparison of Four Data Selection Methods for Artificial Neural Networks and Support Vector Machines. *IFAC-PapersOnLine*, 50(1):11227–11232, 2017.

[13] P. J. Haley and D. Soloway. Extrapolation limitations of multilayer feedforward neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, pages 25–30. IEEE, 7-11 June 1992.

[14] David Avis, David Bremner, and Raimund Seidel. How good are convex hull algorithms? *Computational Geometry*, 7(5-6):265–301, 1997.

[15] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[16] Fabian Zapf and Thomas Wallek. Comparison of data selection methods for modeling chemical processes with artificial neural networks.

[17] Pedro M. Ferreira. Unsupervised entropy-based selection of data sets for improved model fitting. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3330–3337. IEEE, 24.07.2016 - 29.07.2016.

[18] Claude Elwood Shannon and Warren Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1999.

[19] David W. Scott and Stephan R. Sain. Multidimensional density estimation. *Handbook of statistics*, 24:229–261, 2005.

[20] Tania Pencheva, Krassimir Atanassov, and Anthony Shannon. Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. In *Proceedings of the tenth international workshop on generalized nets, Sofia*, pages 1–7. 2009.

[21] Jack V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, 1996.

[22] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer International Publishing, Cham, 2018.

[23] Tomasz Szandała. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In Akash Kumar Bhoi, Pradeep Kumar Mallick, Chuan-Ming Liu, and Valentina E. Balas, editors, *Bio-inspired Neurocomputing*, volume 903 of *Studies in Computational Intelligence*, pages 203–224. Springer Singapore, Singapore, 2021.

[24] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a Self-Gated Activation Function. *Neural and Evolutionary Computing*, 2017.

[25] Katarzyna Janocha and Wojciech Marian Czarnecki. On Loss Functions for Deep Neural Networks in Classification. *Schedae Informaticae*, 1/2016, 2017.

[26] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71, 2003.

[27] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pages 1067–1075, 2013.

[28] James Hensman, Nicolo Fusi, and Neil D. Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

[29] Mathieu Rouaud. Probability, statistics and estimation. *Propagation of uncertainties*, 2013.

[30] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[31] Yingzhen Li and Yarin Gal. Dropout inference in Bayesian neural networks with alpha-divergences. *arXiv preprint arXiv:1703.02914*, 2017.

[32] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *Unpublished Draft. Retrieved*, 19:2019, 2019.

[33] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[34] Santosh K. Gupta and Manojkumar Ramteke. Applications of Genetic Algorithms in Chemical Engineering I: Methodology. In Jayaraman Valadi and Patrick Siarry, editors, *Applications of Metaheuristics in Process Engineering*, volume 86, pages 39–59. Springer International Publishing, Cham, 2014.

[35] Felix Lechleitner. Multikriterielle Optimieralgorithmen in der Anlagensimulation: Konstruktionsübung.

[36] A. Ben-Israel, A. Ben-Tal, and A. Charnes. Necessary and sufficient conditions for a pareto optimum in convex programming. *Econometrica*, 45(4):811–820, 1977.

[37] D. Laguitton and E. Ter Heijden. Computation Path Optimization for Process Simulation by the Sequential Modular Approach. *IFAC Proceedings Volumes*, 16(15):239–249, 1983.

[38] Kevin D Dahm and Donald P Visco. *Fundamentals of Chemical Engineering Thermodynamics*. Nelson Education, 2014.

[39] Mitchell McIntire, Daniel Ratner, and Stefano Ermon. Sparse Gaussian Processes for Bayesian Optimization. In *UAI*, 2016.

[40] Haitao Liu, Jianfei Cai, and Yew-Soon Ong. Remarks on multi-output Gaussian process regression. *Knowledge-Based Systems*, 144:102–121, 2018.