NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for naturvitenskap og teknologi
Institutt for kjemisk prosessteknologi

SPECIALIZATION PROJECT 2016

TKP4580

PROJECT TITLE:

Investigation of surrogate model generation for the reaction section of an ammonia synthesis loop

By

Kun Wang

Supervisors for the project: **Sigurd Skogestad**
Julian Straus

Date: 20/12/2016

Preface

This project investigated a methodology for dimension reduction of variables for surrogate modelling. The problem comes from the optimization for an ammonia synthesis loop modelled in HYSYS. Since this is a highly integrated process, it is not feasible to optimize it directly in the HYSYS model. In order to make the optimization possible, the whole process is firstly divided into subunits, such as the reaction section and the separation section, and then surrogate models are generated for each subunit. After the model generation, the surrogate models for the whole process can be obtained by combining all the models of the subunits, and hence the ammonia plant can be optimized using the new models. The objective of this project is to generate the surrogate model for the reaction section and validate its the performance. The work of this project is achieved in MATLAB based on the dimension reduction methodology.

The project is quite interesting since it includes some frontier concepts for modelling chemical processes. It was not easy for me to get into the topic but it has been worked out fine because I got a lot of help during the project work. I would like to thank my supervisor, Sigurd Skogestad, and co-supervisor, Julian Straus, for letting me work on this project and for their guidance in this project. I am grateful to Julian for the kindly sharing of his knowledge and answering my questions. And I appreciate the corporation of Yara International ASA for providing the HYSYS model.

Trondheim, 2016-12

Kun Wang

Contents

Preface i

1 Introduction 2

2 Description of the process 4

3 Surrogate model generation 6

3.1 Sampling for both input and output variables 6

3.2 Dimension reduction for input variables 9

3.3 Dimension reduction for output variables 10

3.4 Using artificial neural network for surrogate model generation 11

4 Validation of the surrogate models 14

5 Overview of the approach 17

6 Conclusions and discussions 18

A MATLAB scripts **19**

1 Sampling 19

2 Fitting 24

Reference **31**

1 Introduction

The optimization of highly integrated chemical processes using commercial steady-state simulators, like Aspen HYSYS, SimSci PRO/II, or UniSim Design Suite, is generally difficult [1]. The reasons is that these simulators use sequential-modular approach to solve the flowsheet and each unit operation is solved sequentially [2].

In this project, an ammonia plant in HYSYS model is aimed to be optimized. However, there are several nested loops in the process, leading to convergence issues when it is optimized in the HYSYS model. In order to make the optimization feasible, the whole process is divided into subunits and then surrogate models are generated for each subunit. After model generation for the subunits, the complete model of the plant can be obtained by combining the subunit models, and hence the whole plant can be optimized based on the new combined model. The surrogate models are generated based on the input and output variables of the subunits. These subunits are connected with each other by connecting variables. For instance, some variables are inputs in one subunit while they are also outputs of another subunit. Hence based on the connecting variables, the whole process can be represented by combining the subunit models without consistent problems or issues of information loss. In this project, the main objective is to generate surrogate models for the reaction section of the ammonia plant and its performance is evaluated. The work is achieved in MATLAB.

In surrogate modelling, the objective process is considered as a black box with input data and output data, and the surrogate models are generated based on the sample data. To obtain the sample data, the data of input variables are imported into the HYSYS model and the corresponding output data are obtained. These input and output data form the sampling space for generation of surrogate models. The number of variables corresponds to the dimension of the sampling space. There are many different methods for surrogate model generation, such as Kriging models, artificial neural networks, and simple table look up method [1]. In this project, artificial neural networks is use to generate the surrogate models.

The complicity of surrogate models are related to the dimension of the sampling space, which indicates that more variables used for model generation lead to more complicated models and take more computation expense. Since the main goal of using surrogate modelling is to

yield a simple model which is easy to handle and feasible for optimization with enough reliability, the reduction of variables for surrogate modelling is implemented to reduce the dimension of the sampling space. The methodology for reduction of variables is based on a three-step procedure illustrated by Straus and Skogestad [1]. First, partial least square (PLS) regression of the input sampling space is performed to obtain new independent variables. These new independent variables, also known as components in PLS [1], are linear combinations of the initial input variables which yield a reduced-dimension sampling space representing the whole sampling space. Second, the the linear material balances are introduced to reduce the number of output variables. Third, surrogate models are generated based on the dimension-reduced input and output variables.

The surrogate models are evaluated after generation. A new validation sampling space of input variables is required for validation, which is different from the sampling space used for model generation. The input data for validation is imported to the HYSYS model to obtain the corresponding output data as exact values. Then the same input data is feed to the surrogate models yielding output data of the new model. The performance of the surrogate models are evaluated by calculating the relative errors of these two set of output data.

2 Description of the process

The objective process is the reaction section of an ammonia synthesis loop in the HYSYS model which is provided by Yara International ASA. For confidential issue, the exact data in the process are not presented in this report. The flowsheet of the reaction section in the ammonia plant is shown in Figure 1.

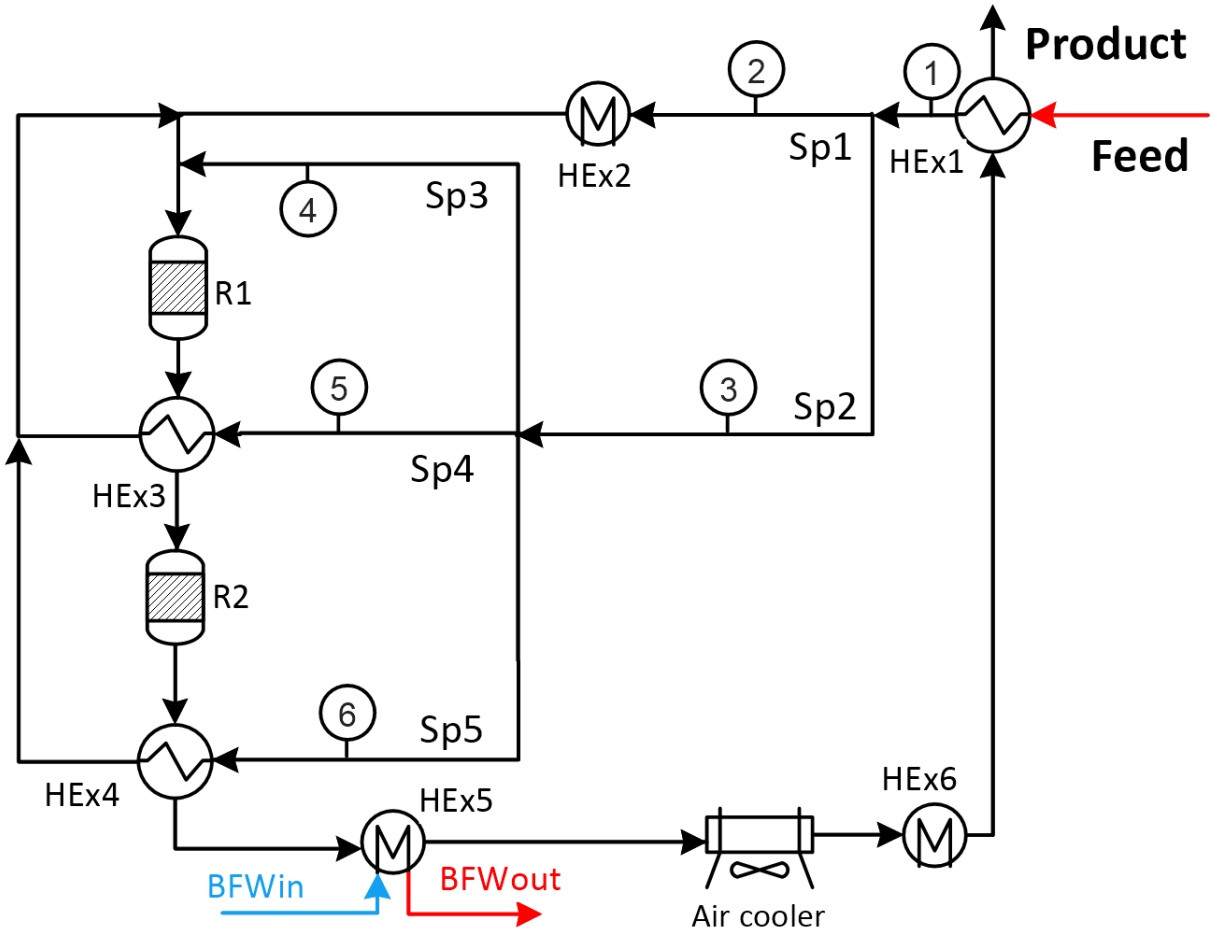


Figure 1: The flowsheet of the reaction section in the ammonia synthesis loop

From Figure 1, it can be observed that the process contains 2 plug flow reactors, $R1$ and $R2$, 6 heat exchangers, $HEx1 \sim HEx6$, and 1 air cooler. The feed flow to the process contains 5 compositions which are H_2, N_2, NH_3, CH_4, and Ar, with corresponding molar flows as $N_{H_2,in}$, $N_{N_2,in}$, $N_{NH_3,in}$, $N_{CH_4,in}$ and $N_{Ar,in}$ respectively. There are also 5 split ratios, $Sp_1 \sim Sp_5$, specified in the process.

The input variables are the ones which can be modified in the process. In this case, these

input variables are identified as molar flows of 5 compositions in the feed flow, $N_{H_2,in}$, $N_{N_2,in}$, $N_{NH_3,in}$, $N_{CH_4,in}$ and $N_{Ar,in}$, 2 split ratios, Sp_4 and Sp_5, the temperature of the feed flow, T_{in}, the pressure of the feed flow, p_{in}, the flow of the boiler feed water (BFW), N_{BFW} and the rotational speed of the fan inside the air cooler, n_{fan}. Hence, there are 11 input variables for this process, the number of which is defined as n_u.

The output variables in this process are the ones which can effect the downstream processes or/and provide important information to identify the process. Therefore, the output variables are identified as the pressure of product flow, p_{out}, the temperature of the product flow, T_{out}, molar flows of 5 compositions in the product flow, $N_{H_2,out}$, $N_{N_2,out}$, $N_{NH_3,out}$, $N_{CH_4,out}$ and $N_{Ar,out}$, and the temperature in the outlet of the BFW, $T_{BFW,out}$. The total number of output variables is 8 and it is defined as n_y.

Hence for surrogate modelling ,the reaction section can be regarded as a black box with $n_u = 11$ input variables and $n_y = 8$ output variables. The output variables are connecting variables which are input variables for the downstream processes.

3 Surrogate model generation

The surrogate models are generated based on the variables defined in the previous section. In this case, there are 11 input variables and 8 output variables in the process. These many variables may lead to complicated surrogate models, thus the surrogate models would be difficult to handle and may cause complexity for optimization after it is combined with other subunit models. Hence, before the surrogate model generation, dimension reduction is performed preliminarily for both input and output variables to reduce the number of variables.

For dimension reduction, sampling spaces are firstly created for both input and output variables using Latin hyper cube sampling [3], based on the exact data stored in the HYSYS model. Next, the PLS regression [4] is applied to the input sample data for dimension reduction, and output variables are reduced by mass balance. Then the surrogate models are generated based on the dimension-reduced data using artificial neural networks (ANNs) [5]. The work is achieved in MATLAB and the MATLAB scripts are shown in Appendix A.

3.1 Sampling for both input and output variables

For dimension reduction, a sampling space must be created beforehand. A sampling space contains a subset of the data which can represent the whole data space. The data space is defined based on the real initial data stored in the HYSYS model. The exact values of the initial data in the HYSYS model are not included in this report due to confidential issue.

A data space is firstly designed for the input variables. Based on the initial data stored in the HYSYS model, different variation ranges are set to the corresponding input variables, which are specified by considering the possible maximum and minimum values for each input variable. The defined variation ranges for the input variables are shown in Table 1.

Table 1: Variation ranges for input variables

Input variables	T_{in} [°C]	p_{in} [barg]	Sp_4 [%]	Sp_5 [%]
Variation range	+-10°C	+-5 barg	+-2%	+-5%
Input variables	N_{BFW} [kmole/h]	n_{fan} [rpm]	$N_{H_2,in}$ [kmole/h]	$N_{N_2,in}$ [kmole/h]
Variation range	+-10%	+-20%	+-12.5%	+-15%
Input variables	$N_{NH_3,in}$ [kmole/h]	$N_{CH_4,in}$ [kmole/h]	$N_{Ar,in}$ [kmole/h]	
Variation range	+100%/-50%	+50%/-40%	+50%/-40%	

With the variation ranges, corner points in the data space can be identified by all the possible combinations of the maximum and minimum values, which can represent the extreme conditions for the input variables. The number of all the possible combinations of the maximum and minimum values is calculated as $2^{11} = 2048$. Each corner point contains 11 (n_u) values corresponding to one possible combination of the maximum and minimum values of the input variables, hence the number of corner points is $n_{CP} = 2048$. In MATLAB, the data of corner points is stored in a 2048-by-11 matrix, $\mathbf{A_{CP}}$.

Corner points only contain the extreme conditions for the input variables, and hence they can not represent all the data space. To solve this problem, additional sample points are defined besides the corner points. The additional sample points are generated using Latin Hypercube Sampling (LHS) method within the variation range. LHS is a statistical method for generating a near-random sample space of parameter values from a multidimensional distribution [6], which can be viewed as a multidimensional extension of Latin square sampling [3].

A Latin square is a square grid containing sample positions where there is only one sample in each row and each column [6]. For example, there is a data space with two dimensions defined as the temperature, T, and the pressure, p, and one wants to creat a sample space with 4 sample points to represent the whole data space. Then a Latin square can be designed as in Figure 2, where each dimension is divided into 4 equal intervals and the sample points are placed to satisfy the Latin square requirements. The symbols, $\times$, in the cells represent the sample points which are distributed in different rows and columns in the square, so that they contain different values for T and p. Hence they can represent and explain the whole data space in an effective way. A Latin hypercube is the generalisation of this Latin square sampling concept to an arbitrary number of dimensions, where each sample is the only one in each axis-aligned hyperplane containing it.

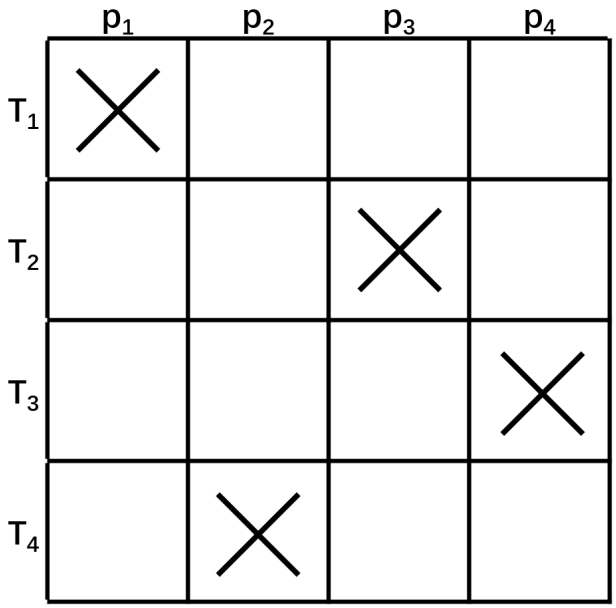


Figure 2: An example of a Latin square sample space with 4 sample points, where the rows are defined as temperatures, T, and columns are defined as pressures, p. The symbols, $\times$, represent sample positions.

In this project, the LHS is achieved by using the MATLAB function `X = lhsdesign(n,p)`, which returns a n-by-p matrix, $\mathbf{X}$, containing a Latin hypercube sample of n values on each of p variables. For each column of $\mathbf{X}$, the n values are distributed with one from each interval $(0, \frac{1}{n})$, $(\frac{1}{n}, \frac{2}{n})$, ..., $(1-\frac{1}{n}, 1)$, and in this case they are placed at the midpoints of the above intervals: $\frac{0.5}{n}$, $\frac{1.5}{n}$, ..., $1-\frac{0.5}{n}$. Hence, the result matrix $\mathbf{X}$ returned by `X = lhsdesign(n,p)` defines the sample points positions in the Latin hypercube space with a range from 0 to 1. Rescaling the LHS matrix, $\mathbf{X}$, using the variation ranges for input variables yields the desired input sample data, defined as $\mathbf{A_{LHS}}$ in MATLAB. Since the input variables have a dimension of $n_u = 11$ and 5000 sample points are desired in this case, n is specified as 5000 while p is specified as 11 in the function, `X = lhsdesign(n,p)`, and hence $\mathbf{A_{LHS}}$ is a 5000-by-11 matrix. Combining $\mathbf{A_{LHS}}$ with the matrix of corner points samples, $\mathbf{A_{CP}}$, a 7048-by-11 matrix, $\mathbf{A_{grid}}$, is obtained, which contains all the required sample data for input variables.

Next, the sample data of input variables, $\mathbf{A_{grid}}$, are imported into the HYSYS model to obtain the corresponding output sample data. The data transformation between the MATLAB and the HYSYS model is achieved by a library, `Hysyslib toolbox`, developed in MATLAB by Olaf T. Berglihn, which is an $activeX/COM$ controller for HYSYS [7]. The data of output variables

obtained from the HYSYS model is stored in a matrix, $\mathbf{X_{grid}}$, in MATLAB. With the number of output variables, $n_y = 8$, $\mathbf{X_{grid}}$ is therefore a 7048-by-8 matrix.

The data of input and output variables in matrices, $\mathbf{A_{grid}}$ and $\mathbf{X_{grid}}$ are used for dimension reduction and the subsequent generation of surrogate models. The surrogate models should also be validated after generation, thus a validation space for both input and output variables are also yielded using the same approach as described above. In MATLAB, the sample data of input variables for validation are stored in the matrix, $\mathbf{A_{val}}$ and the corresponding output data obtained from the HYSYS model is stored in the matrix $\mathbf{X_{val}}$. With a validation space with 1000 points designed in this project, $\mathbf{A_{val}}$ is a 1000-by-11 matrix and $\mathbf{X_{val}}$ is a 1000-by-8 matrix. And these matrices containing the validation samples are used for validation of the surrogate models.

3.2 Dimension reduction for input variables

For dimension reduction of the input variables, partial least square (PLS) regression is applied to the input sample data. PLS is used to find the fundamental relations between two matrices (**X** and **Y**) by finding a linear regression model by projecting the predicted variables (also known as predictors) ,**X**, and the observable variables (also known as response or observers),**Y**, to a new space. A PLS model will try to find the multidimensional direction in the **X** space that explains the maximum multidimensional variance direction in the **Y** space [4]. The dimension of the new space can be defined by different values depending on the the number of components required.

In this case, the predicted variables are the input variables, $\mathbf{A_{grid}}$, and the observable variables are the output variables, $\mathbf{X_{grid}}$. The PLS regression in MATLAB is achieved by the function `[XL,YL] = plsregress(X,Y,ncomp)`, which computes a partial least-squares (PLS) regression of **Y** on **X**, using $ncomp$ PLS components, and returns the predictor and response loadings in **XL** and **YL**, respectively. **X** is an n-by-p matrix of predictor variables, with rows corresponding to observations and columns to variables. **Y** is an n-by-m response matrix. **XL** is a p-by-$ncomp$ matrix of predictor loadings, where each row contains coefficients that define a linear combination of PLS components that approximate the original predictor variables. **YL** is an m-by-$ncomp$ matrix of response loadings, where each row contains coefficients that define a linear combination of PLS components that approximate the original response variables. In this case, for the

function, `[XL,YL] = plsregress(X,Y,ncomp)`, **X** corresponds to $\mathbf{A_{grid}}$ and **Y** corresponds to $\mathbf{X_{grid}}$. The number of components, $ncomp$, is defined as equal to the dimension of the input variables, $n_u = 11$. By multiplying $\mathbf{A_{grid}}$ with the loading **XL**, we can get the new sample space for the input data, defined as $\mathbf{A_{PLS}}$ in MATLAB. The loadings for input variables, **XL**, is also utilized for the later dimension reduction of validation sample data. For the subsequent surrogate model generation, we use the first main k ($1 \leqslant k \leqslant 11$) components defined in the loadings, **XL**, for fitting the model and evaluate the performance with different numbers of components. Usually, better performance can be achieved with larger k, because more components can keep more information of the initial data.

In summary, the dimension reduction for input variables is achieved by obtaining new sample data, $\mathbf{A_{PLS}}$, with the first main k components yielded by PLS regression to represent all the input variable samples, $\mathbf{A_{grid}}$. The $\mathbf{A_{PLS}}$ is used as input data to generate the surrogate models and the models are also evaluated with different k components.

3.3 Dimension reduction for output variables

Instead of PLS regression, mass balance is used to reduce the dimension of the output variables. The advantage of using mass balance is that all the information of initial data can be kept in the new dimension-reduced data by this approach. As mentioned in Section 2, there are $n_y = 8$ output variables in the process, which are the pressure of product flow, p_{out}, the temperature of the product flow, T_{out}, molar flows of 5 compositions in the product flow, $N_{H_2,out}$, $N_{N_2,out}$, $N_{NH_3,out}$, $N_{CH_4,out}$ and $N_{Ar,out}$ and the temperature in the outlet of the BFW, $T_{BFW,out}$. The variables, p_{out}, T_{out} and $T_{BFW,out}$, can not be reduced using mass balance and they are kept as the new variables. Since CH_4 and Ar have no reactions in the process, the molar flows of them in the product stream are the same as the feed stream based on mass conservation, meaning that $N_{CH_4,out} = N_{CH_4,in}$ and $N_{Ar,out} = N_{Ar,in}$. Therefore they can be removed from the output variables. As for compositions of H_2, N_2 and NH_3, they are included in the same reaction as

$$3H_2 + N_2 \longrightarrow 2NH_3$$

The extent of reaction can be calculated in order to describe the relations of the molar flows of these compositions. The extent of reaction is a quantity that measures the extent in which the

reaction proceeds, denoted as ξ. The extent of reaction is defined as [8]

$$\xi = \frac{\Delta N_i}{\nu_i} \tag{1}$$

where ΔN_i denotes the changes of amount for the i-th reactant which are ΔN_{H_2}, ΔN_{N_2} and ΔN_{NH_3} in this case, and ν_i is the stoichiometric number of the i-th reactant identified as $\nu_{H_2} = -3$, $\nu_{N_2} = -1$, and $\nu_{NH_3} = 2$.

In this process, ΔN_i is the difference of amounts of composition molar flows in product stream and feed stream, hence the extent of reaction can be rewritten as

$$\xi = \frac{N_{i,out} - N_{i,in}}{\nu_i} \tag{2}$$

The values of $N_{H_2,in}$, $N_{N_2,in}$ and $N_{NH_3,in}$ are known as input data, hence the variables $N_{H_2,out}$, $N_{N_2,out}$ and $N_{NH_3,out}$ can be calculated with the same extent of reaction, ξ, using the equation above. Therefore, the three variables, $N_{H_2,out}$, $N_{N_2,out}$ and $N_{NH_3,out}$, can be reduced to one variable, ξ.

After dimension reduction, the 8 output variables are reduced to 4 variables which are the pressure of product flow, p_{out}, the temperature of the product flow, T_{out}, the temperature in the outlet of the BFW, $T_{BFW,out}$, and the extent of reaction, ξ. The data of the dimension-reduced output variables is stored in the matrix $\mathbf{X_{PLS}}$.

3.4 Using artificial neural network for surrogate model generation

Based on the input data and output data obtained previously, the approach of artificial neural networks (ANNs) [5] is implemented to generate surrogate models. The advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism that "learns" from observed data [9], therefore they are applicable to this project for surrogate model generation. The artificial neural networks are essentially mathematical models defining a function $f : \mathbf{X} \rightarrow \mathbf{Y}$ to describe the relations between two data space, $\mathbf{X}$ and $\mathbf{Y}$. A neuron's network function f is defined as a class of functions g_i, which can further be defined as a composition of other functions.

The artificial neural networks typically consist of multiple layers. For instance, Figure 3

shows an example of cascade-forward networks which contain 3 layers including 1 input layer, 1 output layer and 1 hidden layer. The layers consist of multiple neurons and each neural unit is connected with many others. The first layer has input neurons which send data via synapses to the second hidden layer of neurons, and then via more synapses to the third layer of output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations [10]. The "weights" in Figure 3 are notated as **W** and **b**. Each individual neural unit may have a summation function, g_i, which combines the values of all its inputs together. More complex systems will have more layers of neurons, some having increased layers of input neurons and output neurons [10]. In summary, a neuron's network function f is a class of functions, g_i, where members of the class, g_i, are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity [5].

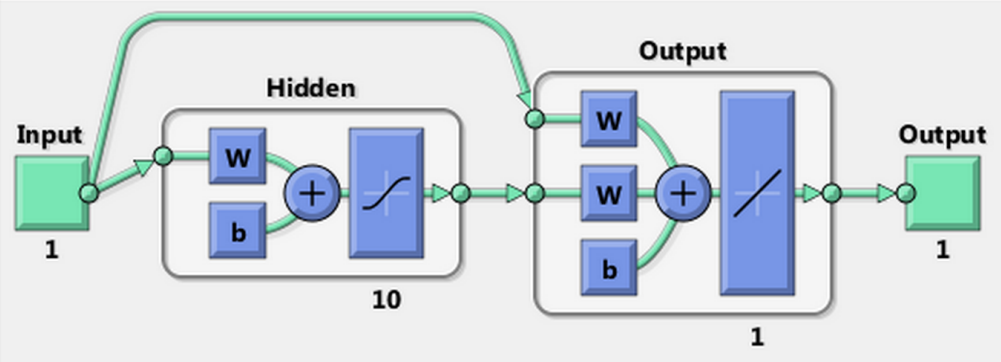


Figure 3: An example of cascade-forward networks with 1 input layer, 1 output layer and 1 hidden layer containing 10 neurons

In this project, the function, f, which defines the artificial neural networks are obtained by the function `cascadeforwardnet(hiddenSizes,trainFcn)` in MATLAB, which returns a cascade-forward neural network. The variable $hiddenSizes$ defines the row vector of hidden layer sizes which in this case is specified as 3 hidden layers with 2, 5 and 5 hidden neurons in each layer respectively. The function `cascadeforwardnet` develops a cascade-forward networks with a similar structure as shown in Figure 3 with corresponding $hiddenSizes$ specification. This structure consist of a series of layers. The first layer has a connection from the network input and each subsequent layer has a connection from the previous layer. The final layer produces the network's output. Moreover, all the layers include a connection from the input and every previous layer to following layers, which is a main difference from common feedforward

networks.

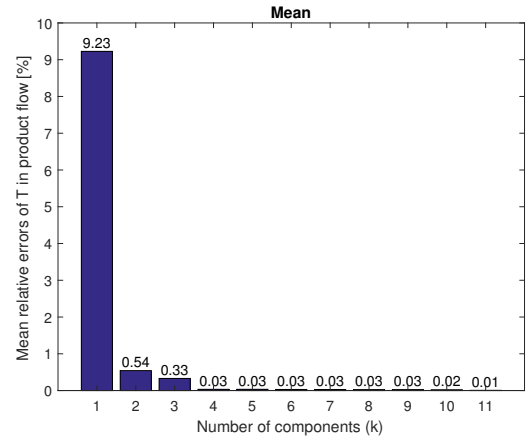
The cascade-forward neural network obtained by the function `cascadeforwardnet` is stored in a variable named *net* in MATLAB, which essentially represents the desired surrogate models. The *net* can be used as a function, `Y = net(X)`, where X is the input data. The function `Y = net(X)` returns Y as the output data.

4 Validation of the surrogate models

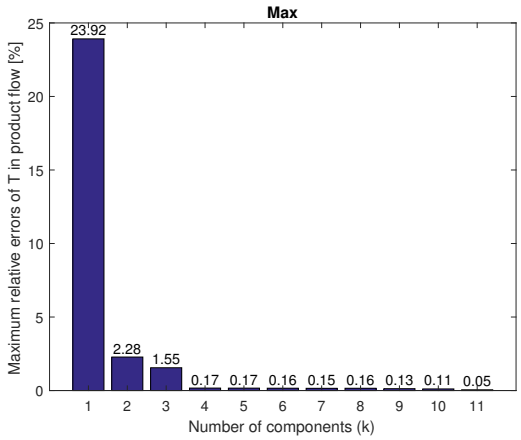
After surrogate model generation, the models are evaluated with different k components obtained from the PLS regression. The sampling space for validation is created using the approach mentioned in Section 3.1. The sample data for input variables are stored in the matrix, $\mathbf{A_{val}}$. Importing $\mathbf{A_{val}}$ into the HYSYS model yields the output data stored in the matrix, $\mathbf{X_{val}}$, which is considered as the exact values to evaluate the surrogate models.

To get the validation data of output exported by the surrogate models, the input data for validation, $\mathbf{A_{val}}$, is multiplied with the loadings, **XL**, obtained by the PLS regression as described in Section 3.2. The multiplication results in the dimension-reduced input data to generate output data for validation using surrogate models. This action ensures the dimension-reduced validation data of input contains the same components as the input data used for model generation, making the comparison of the two sets of output data meaningful. The input data is stored in the matrix, $\mathbf{A_{valPLS}}$. Using the function of ANNs, net, the output data exported by the surrogate models can be obtained by $\texttt{net}(A_{valPLS})$, which is stored in the matrix, $\mathbf{X_{valPLS}}$. Hence the performance of the surrogate models can be evaluated by calculating the relative errors of $\mathbf{X_{valPLS}}$ and $\mathbf{X_{val}}$.

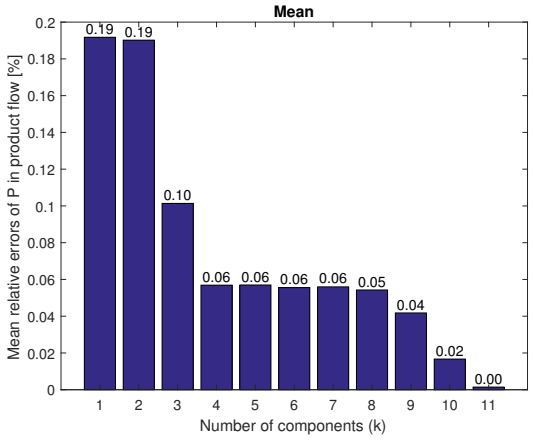
The maximum and mean relative errors for the dimension-reduced output variables, T_{out}, p_{out}, $T_{BFW,out}$ and ξ, are shown in Figure 4 and Figure 5.



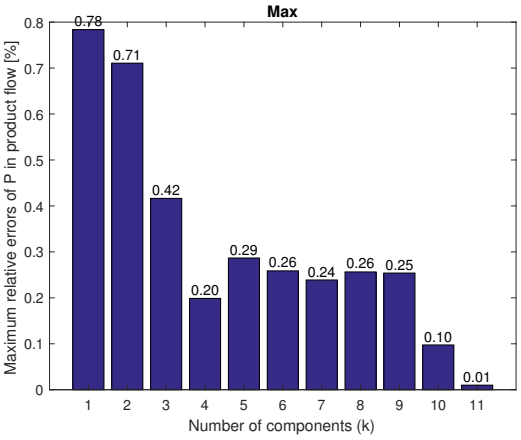
(a) The mean relative errors of T_{out}



(b) The maximum relative errors of T_{out}



(c) The mean relative errors of p_{out}

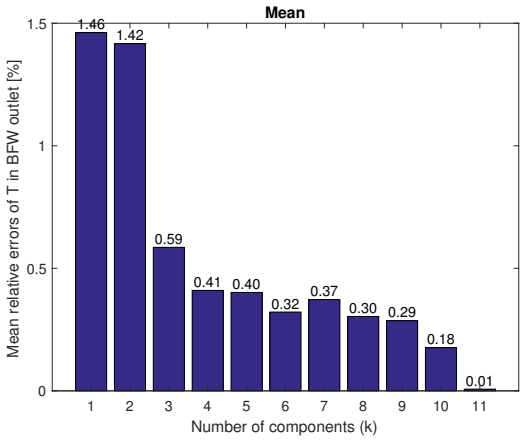


(d) The maximum relative errors of p_{out}

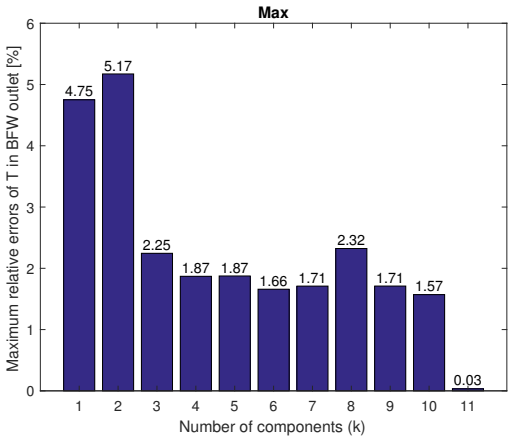
Figure 4: Mean and maximum relative errors for the output variables, T_{out} and p_{out}

From the Figure 4, it can be seen that as components (k) yielded by PLS regression increases, both maximum and mean relative errors for T_{out} and p_{out} decrease. The reason is that the dimension-reduced input variables with more components can keep more information from the initial data. Hence the surrogate models generated by the input data with more components can describe the HYSYS model more accurately. For T_{out}, input data with more than 4 components can generate surrogate models with the mean relative error less than 0.03% and the maximum relative error less than 0.17%, which can be considered as acceptable deviations. For p_{out}, the models generated by more than 4 components also achieves a good performance, with mean relative error less than 0.06% and maximum relative error less than 0.3%. Therefore, the surrogate models generated with 4 components can yield the output data for T_{out} and p_{out}

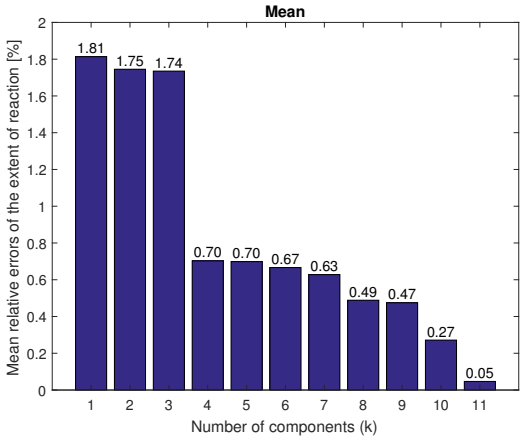
with acceptable deviations.



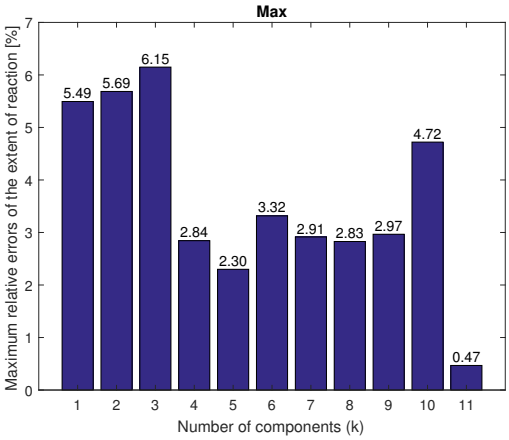
(a) The mean relative errors of $T_{BFW,out}$



(b) The mean relative errors of $T_{BFW,out}$



(c) The mean relative errors of ξ



(d) The maximum relative errors of ξ

Figure 5: Mean and maximum relative errors for the output variables, $T_{BFW,out}$ and ξ

In comparison to T_{out} and p_{out}, $T_{BFW,out}$ and ξ can not result in a similar good performance as shown in Figure 5. The tendency of decreasing for both mean relative errors can be observed from the plots but the maximum relative errors no longer decrease with more than 2 or 3 components. Although the mean relative errors with 4 components for $T_{BFW,out}$ and ξ are 0.41% and 0.70% respectively, the maximum errors are more than 2% for both variables which is not acceptable. Actually, we care more about the maximum relative errors because we want the surrogate models to have a good performance in any possible conditions. To reduce the errors, the further investigation could be conducted to generate better surrogate models by using other regression methods and improving the structure of the ANNs.

5 Overview of the approach

An overview of the approach used in this project is shown in Figure 6. The black boxes represent the data without dimension reduction and the blue boxes represent the dimension-reduced data. The red boxes represent the two models. Meanwhile, The boxes with solid frames and the solid arrows represent the section for surrogate model generation; whereas the boxes with dash frames and dash arrows represent the section for model validation. The approach starts with the initial data provided in the HYSYS model. By the defined variation ranges for input variables and implementation of Latin hypercube sampling, the input sampling space for model generation, $\mathbf{A_{grid}}$ and the input samples for model validation, $\mathbf{A_{val}}$ are obtained. Importing the input data yields the corresponding output data, $\mathbf{X_{grid}}$ and $\mathbf{X_{val}}$, via HYSYS model. Applying PLS regression to $\mathbf{A_{grid}}$ yields the dimension-reduced input data, $\mathbf{A_{PLS}}$, for the model generation. The loadings of components, $\mathbf{XL}$ is also obtained and applied to $\mathbf{A_{val}}$ to get the dimension-reduced input data, $\mathbf{A_{valPLS}}$ with the same K components for model validation. The dimension of $\mathbf{X_{grid}}$ is reduced by mass balance, yielding $\mathbf{X_{PLS}}$ for model generation. Then the surrogate model, net, is generated based on $\mathbf{A_{PLS}}$ and $\mathbf{X_{PLS}}$. Using the model, net, with input data, $\mathbf{A_{valPLS}}$, yields the output data, $\mathbf{X_{valPLS}}$, exported by the surrogate models. Calculating the relative errors of $\mathbf{X_{val}}$ and $\mathbf{X_{valPLS}}$ with different k components, the performance of the surrogate models can be evaluated for the output variables.

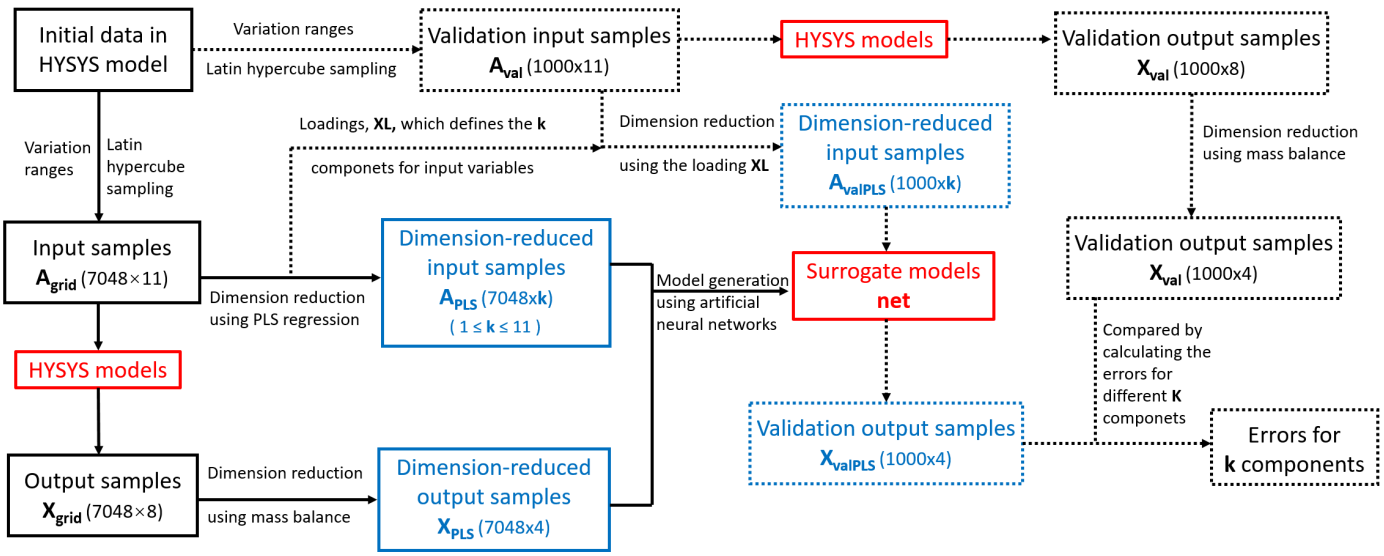


Figure 6: Overview of the approach for surrogate model generation and validation

6 Conclusions and discussions

This project generated surrogate models for the reaction section of an ammonia plant in the HYSYS model, and evaluated the performance of the new models. The approach is based on the dimension reduction methodology illustrated by Straus and Skogestad [1]. Latin hypercube sampling is used to design the sample space. For dimension reduction of the variables, PLS regression is applied to input variables and mass balanced is implemented to get the dimension-reduced output variables. The surrogate models are generated based on the dimension-reduced variables using artificial neural networks. The performance of the surrogate models are evaluated by calculating the relative errors.

It is concluded that the surrogate models have good performance with 4 components for the temperature of the product flow, T_{out} and the pressure in the product flow, p_{out}. However, the relative errors for the temperature in the BFW outlet, $T_{BFW,out}$ and the extent of reaction, ξ, are not acceptable. The reason could probably be the difficulty of finding the linear combinations of input variables using PLS regression or the non-optimal structure of ANNs. To improve the performance of the surrogate models, different regression methods could be implemented and the structure of the ANNs could be optimized in the further investigation. Different approaches, like Kriging models, for surrogate model generation could also be investigated in the future work.

Appendix A

MATLAB scripts

In this project, the work is achieved in MATLAB. All the MATLAB scripts are attached below, except the library for data transformation between HYSYS and MATLAB which can be found in the author's homepage [7]. The MATLAB scripts contain two parts. In the Sampling part, the Latin hypercube sampling is implemented to obtain the sample space. And the Fitting part realizes the surrogate model generation and validation.

1 Sampling

```
% Clearing of the workspace and the memory, closing of all open
% windows/figures
clc, clear variables
close all
format long

% Definition of global variables
global hy hycase

hy = actxserver('Hysys.Application');
hycase = hy.Activedocument;

% Definition of the initial values
```

```
A.init = % Tin(C),Pin(barg)
         % H2 NH3 Ar N2 CH4(Nm3/h)
         % Sp1(-) Sp2(-) Tref(rpm) BFW(Nm3/h)

% Time lhs
time.lhs.on = tic;

% Indices for input variables
indVar = 1:11;

% Number of input variables
nVar = length(indVar);

% Variation ranges for the input values
% T+-10(C),P+-5(barg),
% H2+-12.5%,NH3-50%+100%,Ar-40%+50%,N2+-15%,CH4-40%+50%,
% Sp1+-0.02,Sp2+-0.05,Fan+-20%,BFW+-10%

lbmub(1,:)  = A.init.*[1   1  0.875 0.5 0.6 0.85 0.6 1     1     0.8 ...
    0.9]...
                    +[-10 -5 0     0   0   0    0   -0.02 -0.05 0   0];
lbmub(2,:)  = A.init;
lbmub(3,:)  = A.init.*[1   1  1.125 2   1.5 1.15 1.5 1     1     1.2 ...
    1.1]...
                    +[+10 +5 0     0   0   0    0   +0.02 +0.05 0   0];

%nvar = length(indin);
nVar = size(lbmub,2);

% Number of corner points
nCP = 2^nVar;

time.total.on = tic;

%% Corner points sampling
% Sampling type: Corner Points
```

```
opt.type = 'CP';

% Call GridDef function for sampling
[grid.CP] = GridDef(lbmub,indVar,opt);
A.CP.HYSYS = grid.CP;
nSamp.CP = size(A.CP.HYSYS(:,1),1);

% HYSYS sampling
[x.CP.HYSYS] = HYSYS(nSamp.CP,A.CP.HYSYS);
x.CP.SI = x.CP.HYSYS*[1 0 0 0;0 1000 0 0;
    0 0 1/1000 0;0 0 0 1];

%% Input sapce sampling except corner points
% Sampling type: LHS
opt.type = 'LHS';
% Number of samples
opt.Np = 5000;

% Call GridDef function for sampling
[grid.LHS] = GridDef(lbmub,indVar,opt);
A.LHS.HYSYS = grid.LHS;
nSamp.LHS = opt.Np;

% HYSYS sampling
[x.LHS.HYSYS] = HYSYS(nSamp.LHS,A.LHS.HYSYS);
x.LHS.SI = x.LHS.HYSYS*[1 0 0 0;0 1000 0 0;
    0 0 1/1000 0;0 0 0 1];

%% Validation space sampling
% Sampling type: LHS
opt.type = 'LHS';
% Number of samples
opt.Np = 1000;

% Call GridDef function for sampling
[grid.val] = GridDef(lbmub,indVar,opt);
```

```
84
85  A.val.HYSYS = grid.val;
86
87  nSamp.val = opt.Np;
88
89  % HYSYS sampling
90  [x.val.HYSYS] = HYSYS(nSamp.val,A.val.HYSYS);
91
92  x.val.SI = x.val.HYSYS*[1 0 0 0;0 1000 0 0;
93      0 0 1/1000 0;0 0 0 1];
94
95  %% HYSYS simulation for corner points
96
97  function [output] = HYSYS(nsamp,input)
98
99  % Grabbing of the Hysys object
100 hy = actxserver('Hysys.Application');
101 hycase = hy.Activedocument;
102
103 % Definition of the spreadsheet
104 spread.Input = hyspread(hy,'Input');
105 spread.Output = hyspread(hy,'Output');
106
107 % Stop the Solver of AspenHysys
108 hycase.solver.CanSolve = 0;
109
110 % Calculation in HYSYS
111 output = zeros(nsamp,4); %size(cell.Output,2));
112
113     for i = 1:nsamp
114     % Definition of the cells in the spreadsheets
115     cell.Input = ...
            hycell(spread.Input,{'B1','B2','B3','B4','B5','B6','B7','B8','B9','B10','B11'});
116     cell.Output = hycell(spread.Output,{'B1','B2','B3','B4'});
117
118     % Set inputs
```

```
119     hyset(cell.Input,input(i,:)); % Sp1 Sp2 Tref BFW
120
121     % Start the Solver of AspenHysys
122     hycase.solver.CanSolve = 1;
123
124     % Solve for the new input
125         while hycase.Solver.issolving ~= 0
126             % Wait and see
127         end
128
129     % Stop the solver
130     hycase.solver.CanSolve = 0;
131
132     % Get output values
133     output(i,:) = hyvalue(cell.Output);
134
135     end
136 end
137
138
139 %% Define the function GridDef for sampling
140 % Including cornerpoints sampling and LHS sampling
141
142 function [grid] = GridDef(lbmub,indice,opt)
143
144 nvar = length(indice);
145 ntot = size(lbmub,2);
146
147 switch opt.type
148
149     case 'CP'
150         % Predefinition of the cell vector for the varied values
151         vec = cell(1,ntot);
152
153         for k = indice
154             vec{k} = linspace(lbmub(1,k), lbmub(3,k),2);
```

```
155         end
156
157         grid_comb = cell(1,ntot);
158         [grid_comb{indice}] = ndgrid(vec{indice});
159
160         for k = indice
161             grid_comb{k} = grid_comb{k}(:);
162         end
163
164         grid = cell2mat(grid_comb(indice));
165
166     case 'LHS'
167
168         % Definition of the number of samples
169         nsamp = opt.Np;
170
171         % Performing the LHS
172         lhs = lhsdesign(nsamp,nvar,'criterion','correlation');
173
174         % Rescaling of the problem
175         grid = [lhs.*(ones(nsamp,1)*diff(lbmub([1 ...
                3],indice)))+ones(nsamp,1)*lbmub(1,indice)];
176
177     otherwise
178
179     end
180 end
```

2 Fitting

```
1 %% PLS Regression
2 clear all
3 load sampling.mat;
```

```
4
5  % Combine the corner points and LHS samples as a complete sampling space
6  A.grid.HYSYS = [A.CP.HYSYS;A.LHS.HYSYS];
7  x.grid.HYSYS = [x.CP.HYSYS;x.LHS.HYSYS];
8
9  nVar = 11;
10 nCP = 2^11;
11
12 % Define the value, PLS should be fitted to (as row vector)
13 %   1 = Outlet p
14 %   2 = Outlet T
15 %   3 = Extent of reaction
16 %   4 = BFW outlet T
17 nFit = 4;
18
19 % Column vector for sacling use
20 nSamp.val = 1000;
21 OnN1.val = ones(nSamp.val,1);
22
23 % The number of fitting networks
24 for nfit = 1:nFit
25
26 % The number of principle components defined by PLS regeression
27     for ncomp = 1:nVar
28
29     % Scaling the input and output data
30             [A.grid.zscore,Var.muA,Var.stdA] = zscore(A.grid.HYSYS);
31     % Same scaling for the validation points
32             A.val.zscore = ...
                   (A.val.HYSYS-OnN1.val*Var.muA)./(OnN1.val*Var.stdA);
33
34             [x.grid.zscore,Var.mux,Var.stdx] = zscore(x.grid.HYSYS);
35     % Same scaling for the validation points
36             x.val.zscore = ...
                   (x.val.HYSYS-OnN1.val*Var.mux)./(OnN1.val*Var.stdx);
37
```

```
38      % Calculation of the Partial least square regression
39      [Aload,xload,Ascores,xscores,...
40                  par.PLS,pctvar,¬,stats] = ...
                        plsregress(A.grid.zscore,x.grid.zscore,ncomp);
41
42      % The cumulated sums
43      expvar = cumsum(pctvar,2);
44
45  %% Create a Fitting Network
46
47      display('FItting ANN');
48
49      % time.ANN =tic;
50      % Calculation of the new components
51      switch ncomp
52          case {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
53              mult = Aload;
54          case 11
55              mult = 1;
56      end
57
58      % Main component of scaled input samples
59      A.grid.PLS = A.grid.zscore*mult;
60
61      % Number of validation points
62      nsamp_val = size(A.val.HYSYS,1);
63      A.val.PLS = A.val.zscore*mult;
64
65      % Definition of network parameter
66      hiddenLayerSize = [2 ones(1,2)*5];
67      net = cascadeforwardnet(hiddenLayerSize,'trainbr');
68      net.trainParam.max_fail = 200;
69      net.trainParam.epochs = 1e5;
70      net.trainParam.min_grad = 1e-9;
71
72      % Calculation of the indices for training, validation and testing
```

```
73      trainR = 70/100;
74      valR   = 15/100;
75      testR  = 15/100;
76      [trainInd,valInd,testInd] = dividerand(nSamp.val,trainR,valR,testR);
77
78      % Set up Division of Data for Training, Validation, Testing
79      net.divideFcn = 'divideind';
80      net.divideParam.trainInd = [1:nCP trainInd+nCP];
81      net.divideParam.valInd = valInd+nCP;
82      net.divideParam.testInd = testInd+nCP;
83      % net.trainFcn = 'trainbr'; 'trainlm'; 'trainscg'
84
85      % Dimension of the output remains the same
86      x.grid.PLS = x.grid.zscore;
87
88      % Train the Network
89      [net,tr] = ...
            train(net,A.grid.PLS',x.grid.PLS(:,nfit)','UseParallel','yes'); ...
            % );% ,'UseGPU','yes');%
90
91      % Inverse zscore, get the unsaled output values
92      x.val.PLS = net(A.val.PLS')';
93      x.val.ANN(:,nfit) = ...
            x.val.PLS.*(OnN1.val*Var.stdx(nfit))+OnN1.val*Var.mux(nfit);
94
95      % Calculation relative error[%] for validation points
96      err.val{nfit,ncomp} = ...
            (x.val.HYSYS(:,nfit)-x.val.ANN(:,nfit))./x.val.HYSYS(:,nfit)*100;
97
98      end
99  end
100
101 % Plot and compare the errors for different output variables with different
102 % number of main components
103 for nfit = 1:nFit
104     for ncomp = 1:nVar
```

```
105
106 err.max(nfit,ncomp) = max(abs(err.val{nfit,ncomp}));
107 err.mean(nfit,ncomp) = mean(abs(err.val{nfit,ncomp}));
108 err.median(nfit,ncomp) = median(abs(err.val{nfit,ncomp}));
109
110     end
111 end
112
113 %% Plot
114
115 figure(1)
116 bar(ncomp,err.max(1,:));
117 title('Max');
118 ylabel('Relative errors of T in product flow [%]');
119 xlabel('Number of components (k)');
120 for n = 1:11
121 text(ncomp(n), err.max(1,n), num2str(err.max(1,n), '%0.2f'), ...
122     'HorizontalAlignment', 'center', ...
123     'VerticalAlignment', 'bottom')
124 end
125
126 figure(2)
127 bar(ncomp,err.max(2,:));
128 title('Max');
129 ylabel('Relative errors of P in product flow [%]');
130 xlabel('Number of components (k)');
131 for n = 1:11
132 text(ncomp(n), err.max(2,n), num2str(err.max(2,n), '%0.2f'), ...
133     'HorizontalAlignment', 'center', ...
134     'VerticalAlignment', 'bottom')
135 end
136
137 figure(3)
138 bar(ncomp,err.max(3,:));
139 title('Max');
140 ylabel('Relative errors of the extent of reaction [%]');
```

```
xlabel('Number of components (k)');
for n = 1:11
text(ncomp(n), err.max(3,n), num2str(err.max(3,n), '%0.2f'), ...
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end

figure(4)
bar(ncomp,err.max(4,:));
title('Max');
ylabel('Relative errors of T in BFW outlet [%]');
xlabel('Number of components (k)');
for n = 1:11
text(ncomp(n), err.max(4,n), num2str(err.max(4,n), '%0.2f'), ...
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end

figure(5)
bar(ncomp,err.mean(1,:));
title('Mean');
ylabel('Relative errors of T in product flow [%]');
xlabel('Number of components (k)');
for n = 1:11
text(ncomp(n), err.mean(1,n), num2str(err.mean(1,n), '%0.2f'), ...
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end

figure(6)
bar(ncomp,err.mean(2,:));
title('Mean');
ylabel('Relative errors of P in product flow [%]');
xlabel('Number of components (k)');
for n = 1:11
text(ncomp(n), err.mean(2,n), num2str(err.mean(2,n), '%0.2f'), ...
```

```
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end

figure(7)
bar(ncomp,err.mean(3,:));
title('Mean');
ylabel('Relative errors of the extent of reaction [%]');
xlabel('Number of components (k)');
for n = 1:11
text(ncomp(n), err.mean(3,n), num2str(err.mean(3,n), '%0.2f'), ...
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end

figure(8)
bar(ncomp,err.mean(4,:));
title('Mean');
ylabel('Relative errors of T in BFW outlet [%]');
xlabel('Number of components (k)');

for n = 1:11
text(ncomp(n), err.mean(4,n), num2str(err.mean(4,n), '%0.2f'), ...
    'HorizontalAlignment', 'center', ...
    'VerticalAlignment', 'bottom')
end
```

Reference

[1] J. Straus and S. Skogestad. Minimizing the complexity of surrogate models for optimization. *Computer Aided Chemical Engineering*, 38(1):289–294, 2016.

[2] Lorenz T Biegler, Ignacio E Grossmann, and Arthur W Westerberg. Systematic methods for chemical process design. 1997.

[3] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[4] Herman Wold. Partial least squares. *Encyclopedia of statistical sciences*, 1985.

[5] IA Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[6] Latin hypercube sampling. https://en.wikipedia.org/wiki/Latin_hypercube_sampling. Accessed: 2016-12.

[7] Hysyslib toolbox. http://www.pvv.org/~olafb/software/hysyslib/. Accessed: 2016-12.

[8] Peter Atkins and Julio De Paula. Atkins' physical chemistry. *Oxford University Press, Oxford*, 2006.

[9] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016.

[10] Artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network. Accessed: 2016-12.