HAW HAMBURG

BACHELOR THESIS
Enrico Uhlenberg

Development of a procedure for extraction of multilinear parameters from standardised nonlinear models

Faculty of Life Sciences
Department Environmental Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Enrico Uhlenberg

Development of a procedure for extraction of multilinear parameters from standardised nonlinear models

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science Environmental Engineering*
at the Department Environmental Engineering
at the Faculty of Life Sciences
at University of Applied Science Hamburg

Supervisor: Prof. Dr.-Ing. Gerwald Lichtenberg
Supervisor: M.Sc. Marius Block

Submitted on: 30. August 2022

Enrico Uhlenberg

Title of Thesis

Development of a procedure for extraction of multilinear parameters from standardised nonlinear models

Keywords

Multilinear systems, tensor decompsition, heating systems

Abstract

Multilinear systems are an extension of linear systems and are an active field of research in control systems. Tensors algebra enables compact and efficient computation of multilinear systems analogous to how linear algebra is used in the context of linear systems. This work develops a method of extracting multilinear parameters in canonical polyadic decomposition from standardised nonlinear systems in MATLAB. This procedure is applied to an existing model of a heating system of a non-residential building and subsequently evaluated for accuracy of the resulting model.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Models of complex physical systems are a vital tool in control engineering for reliable and energy-efficient operation, i.e. when being used in model predictive controllers which rely heavily on both the accuracy and performance of the supplied model.
Within the field of control systems, linear models have always played a significant role as the tools of linear algebra are paramount to problem-solving in this field. [6]

Multilinear models are a superset of linear models, while still offering tools to be efficiently computed via tensor algebra analogously to linear functions and linear algebra. This makes them a interesting field of research for control as they have the potential to enable efficient computation of systems previously categorised as non-linear and thereby computationally expensive. [2, p. 17]

This thesis aims to ease the process of finding possible multilinear representations of existing standardised models and thereby lowering the barrier of entry for this active field of research.

2 Background

2.1 Mathematical background

2.1.1 Multilinear functions

Multilinear functions are a subclass of non-linear functions, extending the class of linear functions. Figure 2.1 shows a visualisation of different function classes, with multilinear functions being a superset of linear and binary functions.

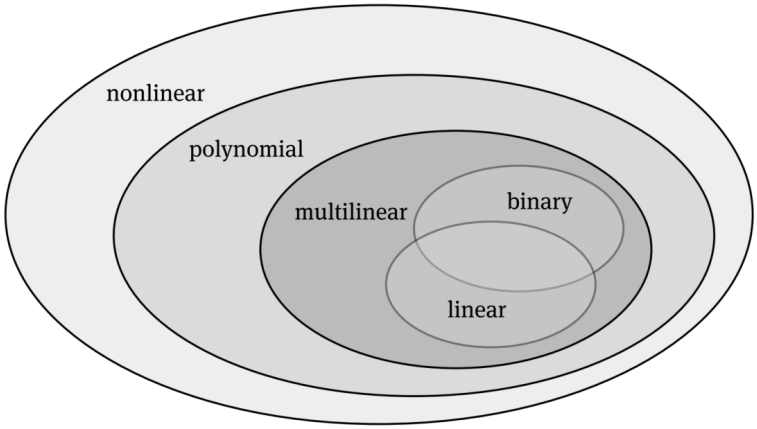


Figure 2.1: Classes of functions
Source:[3, p. 2]

A function $f(\mathrm{x})$ with $\mathrm{x} = (x_1, \ldots, x_n)$ is called multilinear, if the function is linear in each individual variable x_i, meaning it behaves linearly when all other variables are held constant. From this follows, in a multilinear function all variables can be multiplicatively combined while this property still holds true.

All possible combinations of scalars can be generated by a so called monomial vector m(x), which is given by

$$m(x) = \begin{pmatrix} 1 \\ x_n \end{pmatrix} \otimes ... \otimes \begin{pmatrix} 1 \\ x_1 \end{pmatrix} \tag{2.1}$$

where $\otimes$ denotes the Kronecker product. [3, p. 3]

2.1.2 Tensor representation

Tensors are n-dimensional arrays to store numerical data. Matrices, used in linear algebra and many other fields of mathematics and engineering are a subset of tensors with two dimensions, vectors are thereby one-dimensional tensors.
The multi dimensionality of tensors is what makes them useful in representing multilinear functions, as the possibility of every variable of interacting with any other variable in itself is a property of multidimensional space. [2, p. 13]

With the introduction of tensors as mathematical constructs corresponding mathematical operations need to be defined. An overview of the field of tensor algebra can be found in [2, p. 16-21], the operation used in this thesis is the contracted product of two tensors A and B notated by $\langle A|B \rangle$. When A contains all parameters of the model and B is given by the monomial tensor such as (2.4), this operation can be used to represent a state transition equation using tensors. In depth information about this operation can be found in [5, p. 86].

Tensors can have very high storage demands as the number of elements increase exponentially with additional dimensions. To mitigate this issue several tensor decomposition methods have been developed over the years. [1]
One of those decomposition methods is the canonical polyadic decomposition. Using this method the elements of the tensor are computed by summing the outer products of the column vectors of so called "factor matrices" $X_i \in \mathbb{R}^{i \text{ x } r}$. Introducing a weighting vector $\lambda \in \mathbb{R}^r$ allows normalization of the column vectors. The variable r is the rank of the resulting tensor. A visualisation of a CP decomposition of a 5 x 4 x 3 tensor can be seen in Fig. 2.2.

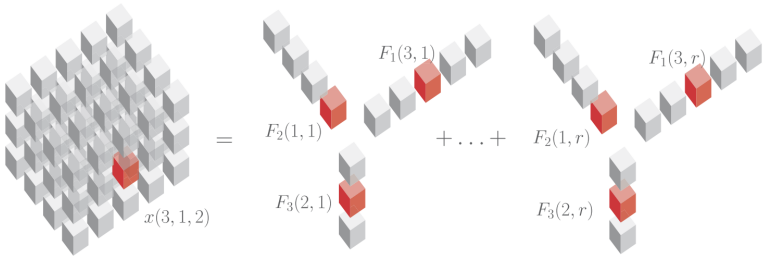


Figure 2.2: CP Decomposition of a tensor with dimensions 5 x 4 x 3
Source:[3, p. 8]

2.1.3 Multilinear models

When modelling real world systems, multilinear functions represented by tensors can be a useful tool. A state space model

$$\dot{x} = \langle \mathrm{F} | \mathrm{M}(x, u) \rangle \tag{2.2}$$

$$y = \langle \mathrm{G} | \mathrm{M}(x, u) \rangle \tag{2.3}$$

can be formulated, where $x \in \mathbb{R}^n$ represents the states of the system and $u \in \mathbb{R}^m$ the inputs. The monomial tensor M has the decomposition

$$\mathrm{M}(x, u) = \left[\begin{pmatrix} 1 \\ x_1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ x_n \end{pmatrix}, \begin{pmatrix} 1 \\ u_1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ u_m \end{pmatrix} \right]. \tag{2.4}$$

The transition tensors can be decomposed into its factor matrices $\mathrm{F}_i, \mathrm{G}_i \in \mathbb{R}^{2 \times r}$ with $i = 1, \ldots, (n + m)$ and scaling matrices $\mathrm{F}_\phi \in \mathbb{R}^{n \times r}$ and $\mathrm{G}_\phi \in \mathbb{R}^{p \times r}$, where p is the number of outputs of the system. [3, p. 20-21]

Determining the values within these matrices, is a crucial part in describing the dynamics of the system and the main focus of this thesis.

2.2 Technical background

2.2.1 Simulink S-Function

The well known programming language MATLAB and its graphical programming environment Simulink are widely used tools within science and engineering. [4]
Within Simulink it is possible to write custom models which integrate seamlessly into the Simulink environment using the programming languages C, C++, Fortran or MATLAB. This is advantageous, as it allows users to implement arbitrarily complex models, which can interact with native graphical Simulink components.
This integration is achieved by enforcing the model to communicate with the Simulink environment via callback methods that will be invoked by Simulink. Some of these callback methods are mandatory to implement such as *mdlInitializeSizes* for initialisation of variable inputs and outputs. Other callback are optional an can be implemented if the model calls for it - such as *mdlDerivatives* for calculating a current derivative of a time-continuous system.
A diagram containing possible callback methods and their order of invocation during simulation can be seen in Fig. 2.3.

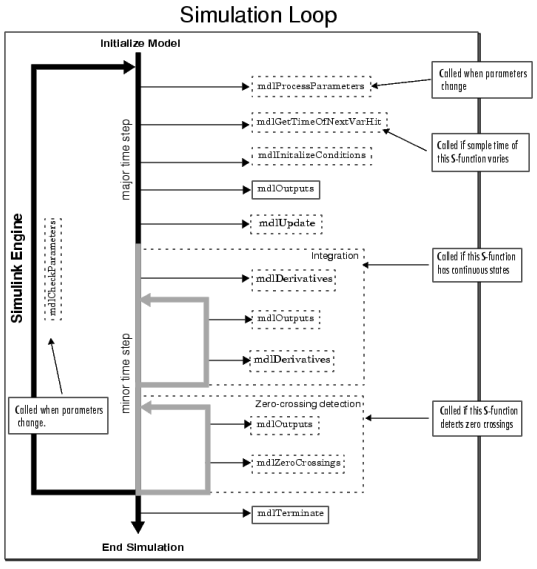


Figure 2.3: Order of invocation of S-Function callbacks
Source:[7]

3 Procedure

3.1 Overview

The procedure described in this chapter is designed to determine whether a given set of differential equations can be represented multilinearly, determine all multilinear terms present in the set and subsequently extract the parameters of each multilinear term. A simple flow diagram illustrating the process can be seen in Fig. 1.1.

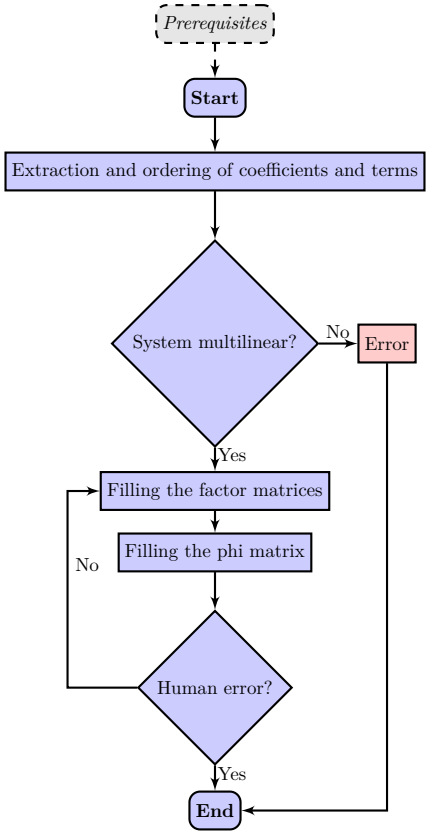


Figure 3.1: Flow diagram of the procedure

The results of this procedure are then organised and stored in CP representation of a tensor. To illustrate the working principle of the procedure, a simple example will be used in this chapter to show the application of the procedure.

3.2 Implementation

This sections contains a sequentially ordered description of the procedure. The singular steps are labelled with roman numerals for easier reference.

Prerequisites

The starting point for this procedure is a nonlinear state space model of the system implemented in MATLAB with the symbolic toolbox as seen in Lst. 3.2 .

This procedure does not require the differential equations to be in any particular form. It is advised to keep logical connections such as physical constants within their logical context, as this can enable more intuitive adjustment even if the multilinear parameters are already extracted. The system

$$\dot{x}_1 = ax_1 + bx_2 + cu \tag{3.5}$$

$$\begin{aligned} \dot{x}_2 &= x_1 x_2 (du + \frac{e}{x_1}) \mid \text{for } x_1 \neq 0 \\ &= dx_1 x_2 u + ex_2 \end{aligned} \tag{3.6}$$

is comprised of the states x_1, x_2 and one input u. The parameters chosen here are symbolic for readability, they can be substituted for numeric values at any point during the procedure.

Eqn. 3.6 is intentionally represented in a form from which the multilinear terms and parameters can't be extracted directly by visual inspection (in contrast to Eqn. 3.5).

Listing 3.1: Example differential equations

```
1 % Define the differential equations
2 syms x1 x2 u a b c d e
3 dT_x1 = a*x1+b*x2+c*u;
4 dT_x2 = x1*x2*(d*u+e/x1);
```

I Extraction and ordering of coefficients and terms

The Matlab command *coeffs* returns all occurring terms and coefficients in respect to a given set of independent variables in a polynomial. By combining the terms of all differential equations into one array and applying the *unique* command, an ordered list of all unique combinations of states and inputs can be obtained. This order should stay unchanged, as it will be used as a reference to order the columns of the factor- and phi-matrices.

Listing 3.2: Defining states and inputs ; Extracting terms and coefficients

```
1     % Define states and input
2  states = [x1 x2];
3  inputs = [u];
4
5  % Get the coefficients of the multilinear terms
6  [x1_coeffs,x1_terms] = ...
7              coeffs(expand(dT_x1),[states inputs]);
8  [x2_coeffs,x2_terms] = ...
9              coeffs(expand(dT_x2),[states inputs]);
10
11 % Get multilinear terms from all diff equations and
      order them
12 u_seq = unique([x1_terms x2_terms]);
```

II Checking for non-linearities

When executing the code above, an error "*Polynomial expression expected.*" will occur if one of the equations can not be parsed as a polynomial i.e. when one of the states or inputs occur in the denominator of the polynomial. This is seemingly the case for (3.6) since x_1 is present in a denominator, but after all terms are expanded by using *expand* as seen in line 10 of Lst. 3.2, x_1 is only present in the numerator of the polynomial. This simplification is trivial in this case but goes to show how this procedure offers a way to ease the evaluation of the multi-linearity of systems programmatically.

By manually inspecting the *u_seq* array, further non-linearities can be identified such as higher powers of states or inputs. If the resulting terms of the equations are exclusively linear multiplications of states and inputs proceed to the next step.

This can be confirmed by visual inspection of the terms for smaller systems, for larger systems programmatic solution should be considered.

III Manually setting the factor matrices

A row vector $F \in \mathbb{B}^j$ representing the minimal normalised form of the factor matrix for each state and input should now be created, j being the length of *u_seq*. The array *u_seq* contains all multilinear terms of the system, as seen in the following table. Each column of the factor matrices should be set to 1 if the corresponding

Index	1	2	3	4
Value	u	x1	x2	u*x1*x2

Table 3.1: Contents of *u_seq*

column of *u_seq* contains the respective term. All other values should be 0. The resulting vectors for the example system can be seen here:

Listing 3.3: Factor matrices in MATLAB

```
% Manually set the factor matrices according to u_seq
F_X1 =     [0 1 0 1];
F_X2=      [0 0 1 1];
F_U =      [1 0 0 1];
```

IV Assembling the Phi matrix

For reliable assembly of the phi matrix, a function was developed in order to assign the extracted factors according to the terms occurring in the differential equations. This function makes use of logical indexing to find pairs of corresponding terms and coefficients and subsequently assigns them along the vectors according to the original equations. The matrix F_ϕ is a n times j matrix, where $n = length(states)$ and j as previously defined. The matrix F_ϕ is implemented in MATLAB as a symbolic array and subsequently filled with either 0 or coefficients of the respective terms.

Listing 3.4: Function for assembling the phi matrix

```
%Create empty symbolic phi matrix
F_Phi = sym('phi',[length(states) length(u_seq)]);

% Fill the phi matrix
```

```
F_Phi = fillPhi(F_Phi,{x1_terms,x2_terms}, ...
                      {x1_coeffs,x2_coeffs},u_seq);

function r = fillPhi(phi,terms,coeffs,seq)
    for j=1:size(phi,1)
        for i=1:length(phi(j,:))
            coeff = coeffs{j}(terms{j} == seq(i));
            if isempty(coeff)
                phi(j,i) = 0;
            else
                phi(j,i) = coeff;
            end
        end
    end
    r = phi;
end
```

V Review the generated data

Due to the high degree of user interaction with raw data in this procedure it is advised to review generated data at this point. Especially the factor matrices are prone to errors but can easily be checked for oversights.
The command *table* is used here, to present the data in a visually accessible way. The array u_{seq} is used to label the columns, the arrays *states* and *inputs* for labelling the rows.

Listing 3.5: Table creation for review

```
% Convert to Tables for inspection
F_Phi_table = array2table(F_Phi,'RowNames',string(states
   ),'VariableNames',string(u_seq));
F_table = array2table([F_X1;F_X2;F_U],'RowNames',string
   ([states inputs]),'VariableNames',string(u_seq));
```

The *F_Table* representing the minimal normalised factor matrices is deemed valid in this context if every column contains a one in each row corresponding to a term in the columns name. The rest of the table should be zero.

Listing 3.6: Review of factor matrices

```
F_table =

3x4 table

          u     x1     x2     u*x1*x2
          -     --     --     -------

   x1     0     1      0          1
   x2     0     0      1          1
   u      1     0      0          1
```

The *F_Phi_Table* represents the differential equations the procedure started with. By multiplying each cell with its column name and adding them together, the original differential equations

$$\dot{x_1} = cu + ax_1 + bx_2 + 0ux_1x_2 = ax_1 + bx_2 + cu \tag{3.7}$$

$$\dot{x_2} = 0u + 0x_1 + ex_2 + dux_1x_2 = x_1x_2(du + \frac{e}{x_1}) \tag{3.8}$$

can be reconstructed.

Listing 3.7: Review of Phi matrix

```
F_Phi_table =

  2x4 table

          u     x1     x2     u*x1*x2
          -     --     --     -------

   x1     c     a      b          0
   x2     0     0      e          d
```

3.2.1 Higher powers of binary variables

As mentioned in Sec. 2.1.1 multilinear systems can include binary variables. (See also [3, p. 3]) Binary variables $\mathbb{B} \in \{0, 1\}$ stay unchanged when multiplied by itself. When the differential equations contain binary variables b^m with $m > 1$, they can be substituted by b^1. This can lead to substantially simplified multilinear terms and should be considered when higher powers of binary variables occur.

The function in Lst. 3.8 is included in the template of the procedure, automating the substitution of higher powers of a symbolic variable.

Listing 3.8: Substite powers of symbolic variable

```
function s = substitute_powers(eq,term,powers)
    arguments
        eq (1,1) sym
        term (1,1) sym
        powers (1,:) int32
    end
    for i=1:length(powers)
        eq = subs(expand(eq),term^powers(1,i),term);
    end

    s = eq;
end
```

4 Application Example

In this section an existing model of the heating system of the CC4E, implemented in Simulink, will first be converted into a system of differential equations. Afterwards the

procedure described in Chapter 3 will be applied in order to represent the system as a tensor in CP decomposition.

4.1 Existing thermal model of the CC4E campus

The heating system of the CC4E Campus is modelled in Simulink, consisting of a consumer of energy, a combined heat and power plant (CHP) and a water heat storage element. The CHP and the consumer are modelled using native Simulink building blocks, whereas the heat storage is implemented in C and embedded within the model using the Matlab Executable (MEX) file format.
The system is laid out as such, that the only inputs of the system are the heat transferred out of the heating system and the control signal to the CHP.

4.2 Extracting differential equations from vanilla Simulink

In order to formulate the equation describing the change of a state with a model containing one integrator at the output, it needs to be converted into its mathematical representation. Each node represents either a variable or an operator, the arrows determine their relationship to each other. By following each path of the model from the integrator in direction of the inputs an equation can be extracted. A simple example of the Simulink and mathematical representation can be seen in Fig. 4.1 and Lst. 4.1.

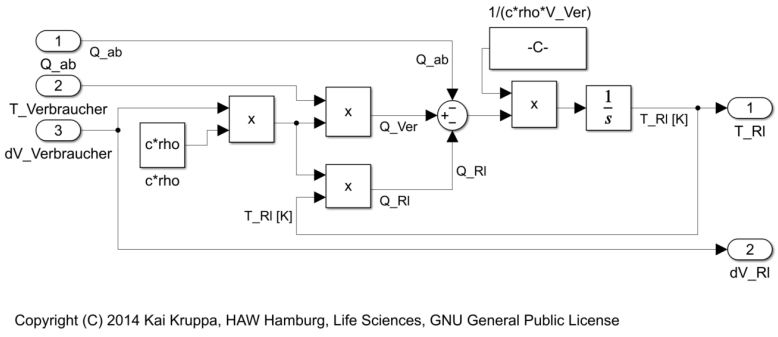


Figure 4.1: Simulink model of the consumer

Listing 4.1: Differential equation of consumer block

```
1 syms Q_ab T_Verbraucher  T_Rl
```

```
dV_Verbraucher = 5.5e-05;
c = 4182;
rho = 1000;
V_Verbraucher = 0.054;

dT_R1 = (c*rho*dV_Verbraucher*T_Verbraucher - Q_ab - T_R1*c*
   rho*dV_Verbraucher)/(c*rho*V_Verbraucher);
```

The example seen above is simple because the only operations within are addition, subtraction, muliplication and division. Other models containing non-linearities such as saturation or binary switches require the introduction of binary variables conditionals.

4.3 Extracting differential equations from C S-Function

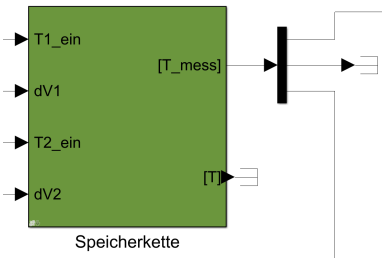


Figure 4.2: Simulink Block containing the storage S-Function

In this section the process of extracting a nonlinear state space model from a Simulink S-Function will be described. The S-Function is embedded in a Simulink block, which receives two volume flows (dV_1, dV_2) and a temperature for each volume flow ($T1_{ein}$,$T2_{ein}$) as input. The Simulink block with inputs and outputs can be seen in Fig. 4.2.

S-Functions in MATLAB can be written in native MATLAB code, Fortran, C or C++. In either case the S-Function implements a set of callback methods, one of which can be "*mdlDerivatives*". This callback method will be called by the Simulink engine if the model is registered as a continuous-time function.

As this is the case for the model of the storage, it is the location from which the state equation of the state space model can be extracted from. This particular model is written in a fairly generic manner, so in order to extract concrete equations this genericity must in turn be concretised. This model is designed to model a storage tank in up to 100 layers, the concrete implementation at CC4E only uses three layers. Those three layers

will subsequently be referred to as Top, Mid and Bot. When inspecting *mdlDerivatives*, the equations

$$dT_{Top} = aT_{Top} + cT_{Mid} + d \tag{4.9}$$

$$dT_{Mid} = aT_{Mid} + bT_{Top} + cT_{Bot} + d \tag{4.10}$$

$$dT_{Top} = aT_{Bot} + bT_{Mid} + d \tag{4.11}$$

for the uppermost, lowermost and interjacent layers can be extracted.
The coefficients a,b,c,d, are defined as

$$a = -\frac{\mathrm{m1out} + \mathrm{mBdwn} + \mathrm{mTup} + \frac{\mathrm{As}\,U}{\mathrm{Cp}} + \frac{\mathrm{Ac_{up}}\,K_{\mathrm{eff}}}{\mathrm{Cp}\,H} + \frac{\mathrm{Ac_{dwn}}\,K_{\mathrm{eff}}}{\mathrm{Cp}\,H}}{M} \tag{4.12}$$

$$b = \frac{\mathrm{mTdwn} + \frac{\mathrm{Ac_{up}}\,K_{\mathrm{eff}}}{\mathrm{Cp}\,H}}{M} \tag{4.13}$$

$$c = \frac{\mathrm{mBup} + \frac{\mathrm{Ac_{dwn}}\,K_{\mathrm{eff}}}{\mathrm{Cp}\,H}}{M} \tag{4.14}$$

$$d = \frac{T_{\mathrm{in}}\,\mathrm{m1in} + \frac{\mathrm{As}\,T_{\mathrm{env}}\,U}{\mathrm{Cp}}}{M}. \tag{4.15}$$

These coefficients are calculated for every call of *mdlDerivatives* and for every layer separately. Note that, in the source code the layers (i.e. Top,Mid,Bot) are implemented using arrays. The necessary indexing is omitted here for readability and replaced with representative subscripts in the case of the states and derivatives. All variables in the coefficients are also stored in arrays per layer, but since they always reference the layer currently being calculated, the subscripts are omitted for readability.

The variables introduced in (4.12),(4.13),(4.14) and (4.15) can be categorised as constant (set by parameters), linearly and nonlinearly dependent on inputs.
A majority of nonlinear behaviour of the model can be attributed to the transition between charging and discharging of the tank, i.e. when dV_1 surpasses dV_2 or vice versa. This behaviour can be encapsulated within the variable *charge*, which is defined by

$$charge = \begin{cases} 1 \text{ for } dV_1 \geq dV_2 \\ 0 \text{ for } dV_1 < dV_2 \end{cases} . \tag{4.16}$$

An overview of how the variables for the coefficients are calculated can be found in Table 4.1.

Table 4.1: Dependencies of variables in differential coefficients in S-Function source code

Variable	Linear relationship	Nonlinear relationship	Subroutine
As	Const. Htank, Vtank, N		
$\mathrm{Ac_{dwn}}$	Const. $\mathrm{Ac_{dwn}}$		
$\mathrm{Ac_{up}}$	Const. $\mathrm{Ac_{up}}$		
$\mathrm{K_{eff}}$	Const. $\mathrm{K_{eff}}$		
T_{env}	Const. T_{env}		
Cp		T	getCp()
M	Const. VTank	T	getRho()
m1in	dV_1, dV_2	charge, T	setInputFlows()
m1out	dV_1, dV_2	charge, T	setInputFlows()
mBup	dV_1, dV_2	charge, T	setFlowDirection()
mBdwn	dV_1, dV_2	charge, T	setFlowDirection()
mTup	dV_1, dV_2	charge, T	setFlowDirection()
mBdwn	dV_1, dV_2	charge, T	setFlowDirection()
T_{in}	$T1_{ein}$ $T2_{ein}$	charge	setInletTemperatures()

The nonlinearities in relation to temperature are due to the density (rho) and specific heat capacity (Cp) being approximated using third and fourth degree polynomials respectively. In order to simplify a multilinear representation a fixed value from within the operating range for Cp and Rho is chosen. (See Appendix A.2) This eliminates all nonlinear dependencies on temperature.

The binary variable *charge* is implemented as

$$charge = \frac{dV_1}{2dV_2}. \tag{4.17}$$

Using an external quantising routine before every evaluation it shall be set to 1 if the numerical value is equal or above 0.5. Otherwise it shall be set to 0. Although this seemingly adds complexity to the implementation it separates the the binary and non-binary logic and thereby enables the multilinear tensor-based representation of the system.

4.4 Building the Factor matrices

4.4.1 Finding non-multilinearities

With the differential equations now extracted from the existing model the procedure introduced in chapter 3 can now be applied.
After following step I (and thereby also step II) for all components of the system (CHP,consumer and storage) it becomes apparent, that the CHP is not multilinearly representable.
After investigating through the use of the *numden()* as seen in Lst. 4.2 command it becomes apparent, that the differential equation describing the CHP contains variables in a denominator which disqualifies it for multilinear representation.
By examining the terms present in the denominator of the polynomial in question as seen in Lst 4.2 and subsequently Tbl. 4.2.

Listing 4.2: Examining the denominator of the CHP for nonzero terms

```
1 [N, D] = numden(dT_BHKW);
2 [BHKW_Dc,BHKW_Dt] = coeffs(D);
3 [BHKW_Nc,BHKW_Nt] = coeffs(N);
```

Note: *BHKW_an_above_half* is a binary variable introduced during the manual extraction of the differential equation. It represents the *Switch3* seen in Fig. 4.3. The signal

Table 4.2: Contents of *BHKW_Dt*

Index	1	2	3	4
Value	BHKW_an_above_half*T_Rl	BHKW_an_above_half	T_Rl	1

path making the model non-multilinear can be seen highlighted in blue in Fig. (4.3). In the highlighted pathway the signal *T_Rl* goes through the divisor node *Divide1* and afterwards through threshold-switch *Swich1*, explaining the denominator terms found in the differential equation.

A MATLAB script documenting the process up to this point can be found in appendix A.4. As this thesis is focussed on viable multilinear models, modelling the CHP will not be investigated further.

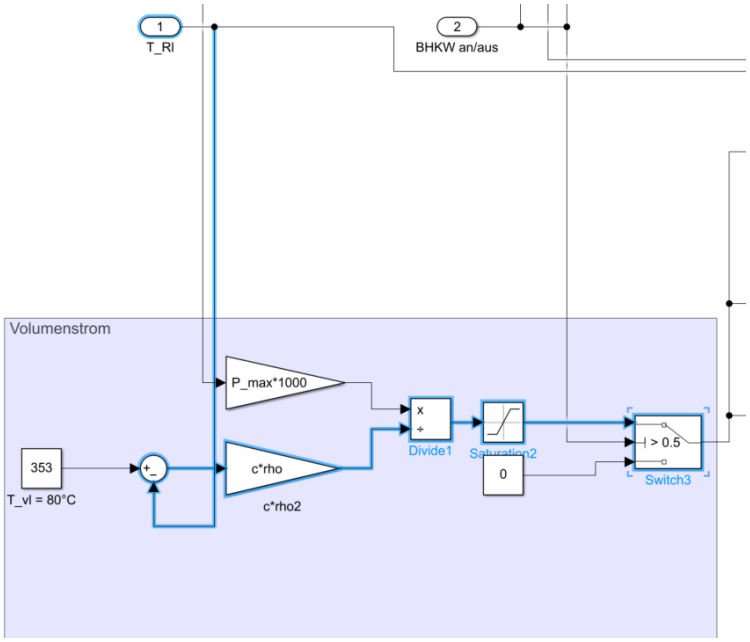


Figure 4.3: Non-multilinear part of the CHP model

4.4.2 Detailed modelling of the storage in isolation

With the aspiration to model the complete system multilinearly put aside, the focus remains on modelling the storage. In order to compare the multilinear model with the original a Simulink model is created with only the storage as can be seen in Fig. 4.4.

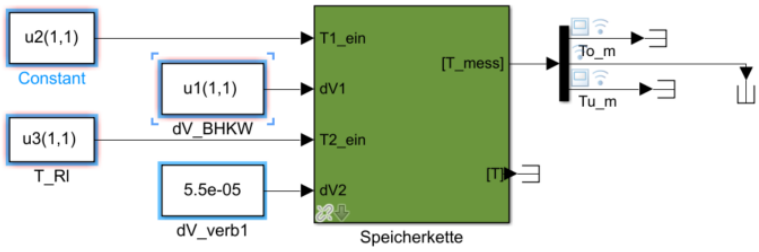


Figure 4.4: Storage in isolation in Simulink

The inputs are the two input flows and the associated temperatures, the states are the temperatures of the three layers of the storage and the additional *charge*-state as defined in (4.16).

Listing 4.3: Defining the inputs and states of the storage system

```
1
2  % Get the differential equations
3  dT_StoTop = getStorageEq('Top');
4  dT_StoMid = getStorageEq('Middle');
5  dT_StoBot = getStorageEq('Bottom');
6
7  % Declare the states and inputs
8  syms charge T_Top T_Mid T_Bot dV_BHKW T_BHKW T_Rl dV_Rl rho
       Cp
9  states = [T_Top T_Mid T_Bot charge];
10 inputs = [dV_BHKW T_BHKW T_Rl];
```

When inspecting the differential equations of the storage it is apparent, that *charge* occurs in powers up to three. To simplify the resulting terms the helper function described in 3.2.1 is applied. The function calls can be seen in Lst. 4.4.

Listing 4.4: Substitution of higher powers of *charge*

```
1 syms charge
2 dT_StoTop = ...
3     substitute_powers(dT_StoTop,charge,[int32(2) int32(3)]);
4 dT_StoMid = ...
5     substitute_powers(dT_StoMid,charge,[int32(2) int32(3)]);
6 dT_StoBot = ...
7     substitute_powers(dT_StoBot,charge,[int32(2) int32(3)]);
```

When going through the remaining steps of the procedure no further problems arise and the tensor in minimal CP representation can be obtained.

4.5 Testing the model

To compare the model against the original S-Function identical time-bases and inputs and a similar solver is chosen. (Ode45 for the tensor based model, Ode8 Dormand-Prince within Simulink) When plotting different responses of both models with static inputs, it becomes apparent that the resulting model behaves similar to the S-Function but the

final stable values differ as well as the responsetime of the system. Different inputs and the respective responses can be seen in Fig 4.5. Depicted are the temperatures of the three layers for the two respective models.

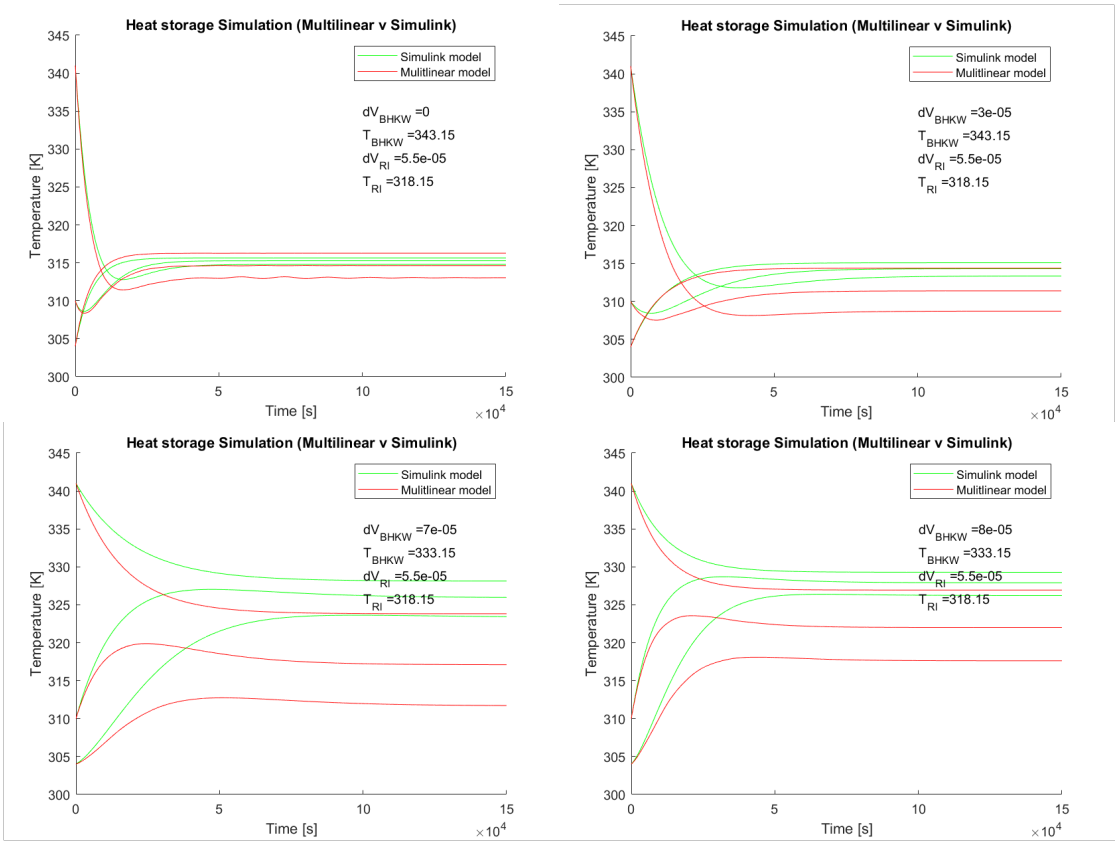


Figure 4.5: Comparison of multilinear model and Simulink model

4.6 Troubleshooting

To achieve a better understanding of the discrepancy between the multilinear model and the Simulink model a manual review of the C code was carried out and some internal variables were inspected during simulation via console outputs. This lead to the following two observations:

Error in absolute value calculation

The C code contains a line of code which calculates the absolute value of the

difference between input flows. Since the information of the relation between these flows is already known through the *charge* variable, this is seemingly trivial to implement. Nonetheless an error occurred here, which was subsequently fixed, see Lst. 4.6. This fix improved performance of the model but a discrepancy remained.

Listing 4.5: Absolute value of floating number in C

```
m = fabs(m1-m2); /* work only with positive flow
   internally */
```

Listing 4.6: Corrected error in differential calculation in MATLAB

```
% m = charge*m1 + (1-charge)*m2; This was wrong

m = m1-m2;
m = charge*m - (1-charge)*m;
```

Inconsistent temperatures around the S-Function

After inspecting values within the C code during the simulation, it became apparent that the mass flow itself is much lower than expected in the C file. The C function calculates the density and specific heat capacity of the water being stored using polynomials with temperature in °C as the independent variable. The model of the CC4E only handles temperatures in Kelvin and thereby these polynomials evaluate to significantly divergent values as the assumptions made during this work(See Section 4.2). A table showing the discrepancies for Cp and Rho and a fix to the isolated model can be seen in Tbl. 4.3 and Fig. 4.6.

Table 4.3: Discrepancy between °C and K in Cp and Rho Polynomials

Input temp	Cp	(|Cp - 4200|/ 4200)*100	Rho	(|Rho - 1000| / 1000)*100
70	4189	0.25%	977.75	2.22%
70 + 273.15	14971	256.45%	871.10	12.89%

4.7 Limitations

Due to time constraints during this thesis the resulting model could not be further refined to exactly match the original Simulink model. Additionally a few limitations to this work should be stated:

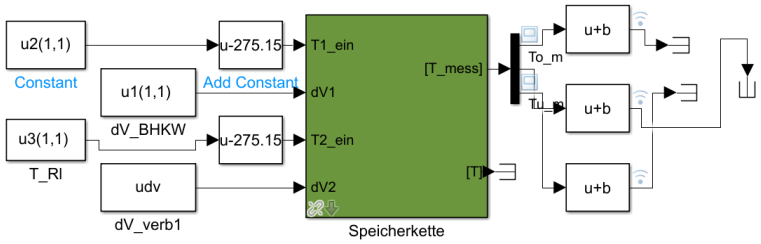


Figure 4.6: Fixing the temperature inputs and outputs of the storage model

Output of the Simulink model

The Simulink model contains a function in *mdlOutput*, which simulates mixing of the layers. This is equivalent of a having a tensor other than the identity tensor for G in (2.3) and was intentionally omitted. It was therefore removed from the C model during the comparisons in section 4.5.

Binary States

As mentioned at the end of section 4.2, an external routine is needed to evaluate the real-valued intermediary values of the binary state to a binary value. As this was not implemented due to time constraints, the *charge* variable was implemented as a constant in the multilinear model and initiated with the valid value. Due to the tests not varying the input values and *charge* only depending on inputs this does not compromise the results.

Static input values

The validation with static input values only allows for limited insight in a system's behaviour. With more time, dynamic analysis would yield more detailed information about the system.

5 Conclusion

The procedure described in this thesis is intended to ease the evaluation of system for multilinearity, enable programmatic extraction of multilinear parameters while keeping the original systems editable. By applying the procedure to the existing model of the CC4E it was shown, that not only evaluation of multilinearity is possible, but concrete inferences on the variables causing the incompatibility can be made (See Sec. 4.4.1).
Keeping the original system equations editable in their human comprehensible form, while evaluating the multilinear model proved useful in tracing errors. The possibility of being able to tweak physical constants while evaluating the resulting model proved especially useful in Sec. 4.6.

Regarding the accuracy of the resulting model, the application example did not yield conclusive results within the time constraints of this work. The resulting multilinear model did increasingly resemble the original, as errors from the manual reading of the source code were found and remedied.
This, however, can not be seen as proof for the accuracy of the proposed procedure. The multilinear sample system introduced in Chapter 3 contrawise could be fully reconstructed after application of the procedure (See (3.7) and (3.8)). The proof of reconstruction of state space models from CP decomposed tensors is given in [5, p. 72-76].
In conclusion the procedure proposed in this can aid in evaluating the multilinearity of standardised non-linear models and extracting multilinear parameters from said systems. Future refinements of this procedure such as automated detection of different kinds of multilinearities and the automation of manual steps could further simplify this process.

Bibliography

[1] Cichocki, Andrzej ; Zdunek, Rafal ; Phan, Anh H. ; Amari, Shun-ichi: Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. John Wiley & Sons, 2009

[2] Kruppa, Kai: Multilinear Design of Decentralized Controller Networks for Building Automation Systems, HafenCity Universität Hamburg, doctoralThesis, 2018. – URL https://repos.hcu-hamburg.de/handle/hcu/496

[3] Lichtenberg, Gerwald ; Pangalos, Georg ; Yáñez, Carlos C. ; Luxa, Aline ; Jöres, Niklas ; Schnelle, Leona ; Kaufmann, Christoph: Implicit multilinear modeling. In: at - Automatisierungstechnik 70 (2022), Nr. 1, S. 13–30. – URL https://doi.org/10.1515/auto-2021-0133

[4] MathWorks: Company Overview. Online Company Report. 2019. – URL https://www.mathworks.com/content/dam/mathworks/fact-sheet/company-fact-sheet-8282v19.pdf. – Zugriffsdatum: 2022-08-28

[5] Pangalos, Georg ; Eichler, Annika ; Lichtenberg, Gerwald: Hybrid Multilinear Modeling and Applications. In: Simulation and Modeling Methodologies, Technologies and Applications. Cham : Springer International Publishing, 2015, S. 71–85. – ISBN 978-3-319-11457-6

[6] Skelton, R.E. ; Iwasaki, T.: Increased roles of linear algebra in control education. In: IEEE Control Systems Magazine 15 (1995), Nr. 4, S. 76–90

[7] University, NorthWestern: Writing S-Functions. Online Learning Material. 2019. – URL http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/simulink/sfg/sfunc_c11.html. – Zugriffsdatum: 2022-08-28

A Appendix

A.1 Example Code

```
1    % Define the differential equations
2  syms x1 x2 u a b c d e
3
4  dT_x1 = a*x1+b*x2+c*u;
5  dT_x2 = x1*x2*(d*u+e/(x1));
6
7
8  % Define states and input
9  states = [x1 x2];
10 inputs = [u];
11
12 % Get the coeffieicients of the multilinear terms
13
14 [x1_coeffs,x1_terms] = coeffs(expand(dT_x1),[states inputs
       ]);
15 [x2_coeffs,x2_terms] = coeffs(expand(dT_x2),[states inputs
       ]);
16
17 % Get multilinear terms from all diff equations and order
       them
18 u_seq = unique([x1_terms x2_terms]);
19
20 % Optional: If the differentials contain binary states,
       you can substitute all powers of it with a power of one
21
22
```

```
23
24
25
26  % Manually set the factor matrices according to u_seq
27  F_U =      [1 0 0 1];
28  F_X1 =     [0 1 0 1];
29  F_X2=      [0 0 1 1];
30
31
32  %Create empty symbolic phi matrix
33  F_Phi =     sym('phi',[length(states) length(u_seq)]);
34
35  % Fill the phi matrix
36  F_Phi = fillPhi(F_Phi,{x1_terms,x2_terms},{x1_coeffs,
        x2_coeffs},u_seq);
37
38
39  % Convert to Tables for inspection
40  F_Phi_table = array2table(F_Phi,'RowNames',string(states),
        'VariableNames',string(u_seq));
41  F_table = array2table([F_X1;F_X2;F_U],'RowNames',string([
        states inputs]),'VariableNames',string(u_seq));
```

A.2 Storage differential equations in MATLAB

```
1  function eq = getStorageEq(type)
2  % GETSTORAGEEQ Returns any of the three Storage
       differential equations
3  arguments
4      type (1,:) char {mustBeMember(type,{'Top','Middle','
           Bottom'})}
5  end
6      T_env = 295;
7      switch type
```

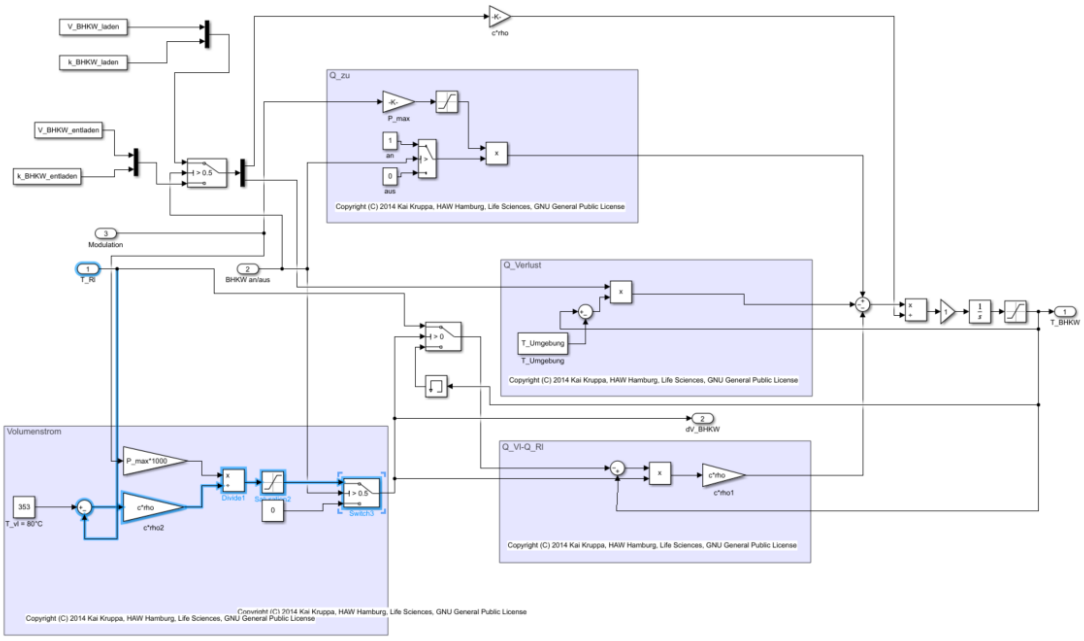
```
8           case 'Top'
9               syms dV_BHKW T_BHKW T_Rl T_Top T_Mid charge
                    rho %dV_Rl Cp
10              dV_Rl = 5.5e-05;
11              Cp = 4182;
12              m1 = dV_BHKW * rho;
13              T1 = T_BHKW;
14              m2 = dV_Rl * rho;
15              T3 = T_Rl;
16              V_TANK = 0.946;
17              H_TANK = 2;
18              U = 2.62;
19              K_eff = 0.003;
20              n = 3;
21              H = H_TANK / n;
22              T_in= T1*charge+(1-charge)*T3;
23
24
25              AC_UP = V_TANK/H_TANK;
26              Ac_dwn = AC_UP;
27              PER = sqrt(4*pi*AC_UP);
28              H_NODE = 3;
29              AS = PER*H_NODE;
30              As_cap = AS + AC_UP;
31              M = V_TANK*rho/n;
32              m = m1-m2;
33              m = charge*m - (1-charge)*m;
34
35
36
37              m1out = (1-charge)*m;
38              m1in = charge*m;
39              mBup = (1-charge)*m;
40              mBdwn = charge*m;
41
42
```

```
43              a = (-m1out-mBdwn-K_eff*Ac_dwn/(Cp*H)-U*As_cap
                    /Cp)/M;
44              c = (mBup+K_eff*Ac_dwn/(Cp*H))/M;
45              d = (m1in*T_in+U*As_cap*T_env/Cp)/M;
46
47              eq = a*T_Top+c*T_Mid+d;
48
49          case 'Middle'
50              syms charge T_Mid T_Top T_Bot dV_BHKW T_BHKW
                    T_Rl dV_Rl %rho Cp
51              dV_Rl = 5.5e-05;
52              m1out =0;
53              m1in = 0;
54              rho = 857.89;
55              Cp = 4182;
56              K_eff = 0.003;
57              V_TANK = 0.946;
58              H_TANK = 2;
59              Ac_up = V_TANK/H_TANK;
60              Ac_dwn = Ac_up;
61              n= 3;
62              H = H_TANK / n;
63              U = 2.62;
64              PER = sqrt(4*pi*Ac_up);
65              H_NODE = 3;
66              As = PER*H_NODE;
67              M = V_TANK*rho/n;
68
69              m1 = dV_BHKW * rho;
70              m2 = dV_Rl * rho;
71              T1 = T_BHKW;
72              T3 = T_Rl;
73
74
75              m = m1-m2;
76              m = charge*m - (1-charge)*m;
```

```
77              T_in= T1*charge+(1-charge)*T3;
78              mTup = (1-charge)*m;
79              mTdwn = charge*m;
80              mBup = (1-charge)*m;
81              mBdwn = charge*m;
82              a = (-m1out-mTup-mBdwn-K_eff*Ac_up/(Cp*H)-
                    K_eff*Ac_dwn/(Cp*H)-U*As/Cp)/M;
83              b = (mTdwn+K_eff*Ac_up/(Cp*H))/M;
84              c = (mBup+K_eff*Ac_dwn/(Cp*H))/M;
85              d = (m1in*T_in+U*As*T_env/Cp)/M;
86
87              eq = a*T_Mid + b*T_Top + c*T_Bot + d;
88          case 'Bottom'
89              syms T_Bot T_Mid charge dV_BHKW T_BHKW T_Rl %
                    dV_Rl rho Cp
90              dV_Rl = 5.5e-05;
91              rho = 857.89;
92              Cp = 4182;
93              K_eff = 0.003;
94              V_TANK = 0.946;
95              H_TANK = 2;
96              Ac_up = V_TANK/H_TANK;
97              n= 3;
98              H = H_TANK / n;
99              U = 2.62;
100             PER = sqrt(4*pi*Ac_up);
101             H_NODE = 3;
102             As = PER*H_NODE;
103             As_cap = As + Ac_up;
104             M = V_TANK*rho/n;
105
106             m1 = dV_BHKW * rho;
107             m2 = dV_Rl * rho;
108             T1 = T_BHKW;
109             T3 = T_Rl;
110
```

```
111
112             m = m1-m2;
113             m = charge*m - (1-charge)*m;
114             T_in= T1*charge+(1-charge)*T3;
115             m1out = charge*m;
116             m1in = (1-charge)*m;
117             mTup = (1-charge)*m;
118             mTdwn = charge*m;
119
120             a = (-m1out-mTup-K_eff*Ac_up/(Cp*H)-U*As_cap/
                    Cp)/M;
121             b = (mTdwn+K_eff*Ac_up/(Cp*H))/M;
122             d = (m1in*T_in+U*As_cap*T_env/Cp)/M;
123
124             eq = a*T_Bot + b*T_Mid + d;
125
126         end
127     end
```

A.3 Full Simulink model of the CHP



A.4 Script modelling the system including storage and CHP

```
1
2  %[sys,x,y,str]= minimalmodell_V7(0,[],[],0);
3  dT_BHKW = getBHKWEq();
4  dT_Rl = getConsumerEq() ;
5  dT_StoTop = getStorageEq('Top');
6  dT_StoMid = getStorageEq('Middle');
7  dT_StoBot = getStorageEq('Bottom');
8
9  [N, D] = numden(dT_BHKW);
10 [BHKW_Dc,BHKW_Dt] = coeffs(D);
11 [BHKW_Nc,BHKW_Nt] = coeffs(N);
12
```

```
13 [Rl_coeffs,Rl_terms] = coeffs(dT_Rl);
14
15 [StoTop_coeffs,StoTop_terms] = coeffs(dT_StoTop);
16
17 [StoMid_coeffs,StoMid_terms] = coeffs(dT_StoMid);
18
19 [StoBot_coeffs,StoBot_terms] = coeffs(dT_StoBot);
20
21 Fx = 3;
22 Fy= 2;
23 kombis = 19;
24 % REIHENFOLGE:  T_RL Q_ab T_Verbraucher(=T_Top?) T_Mid T_Bot
      (=T_Rl_BHKW?) dV_BHKW charge
25
26 syms T_Rl Q_ab T_Top T_Verbraucher T_Mid T_Bot dV_BHKW
      charge T_BHKW BHKW_an_above_half BHKW_an_above_zero
      T_BHKW_above_low T_BHKW_above_upper Vstrom_above_low
      Vstrom_above_upper
27
28
29 F_Rl = zeros(Fy,kombis);
30 F_Qab = zeros(Fy,kombis);
31 F_Top = zeros(Fy,kombis);
32 F_Mid = zeros(Fy,kombis);
33 F_Bot = zeros(Fy,kombis);
34 F_dV_BHKW = zeros(Fy,kombis);
35 F_T_BHKW = zeros(Fy,kombis);
36 F_charge = zeros(Fy,kombis);
37 F_mod = zeros(Fy,kombis);
38
39
40
41 F_Rl(2,1:end) =      [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
      0];
42 F_Top(2,1:end) =     [0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
      0];
```

```
43 F_Mid(2,1:end) =      [0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0
       0];
44 F_Bot(2,1:end) =      [0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0
       0];
45 F_dV_BHKW(2,1:end) = [0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0
       0];
46 F_T_BHKW(2,1:end) =   [0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
       0];
47 F_charge(2,1:end) =   [0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
       0];
48 F_mod(2,1:end) =      [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
       0];
49 F_Qab(2,1:end) =      [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       1];
50
51
52 %                      T_RL
       T_BHKW*charge

       T_Top/T_Verbraucher                          T_Mid
                                          T_bot
                                         dV_BHKW
                               T_BHKW
       charge                             mod
                                  T_Mid*charge

       T_Mid*charge*dV_BHKW

       T_BHKW*charge*dV_BHKW

       T_Top*charge

       T_Top*charge*dV_BHKW
       T_Bot*charge

       T_Bot*charge*dV_BHKW
```

```
   T_Rl*charge*dV_BHKW

   T_Rl*charge

   Q_ab
F_phi(1,1:kombis) = [Rl_coeffs(Rl_terms == T_Rl)
   0

   Rl_coeffs(Rl_terms == T_Verbraucher)   0
                                          0
                                         0
                                  0
                                  0
                                  0
                                  0

   0

   0

   0

   0                                                                    0


   0

   0

   0

   Rl_coeffs(Rl_terms == Q_ab)];
F_phi(2,1:kombis) = [0
   0
```

```
   StoTop_coeffs(StoTop_terms == T_Top)   StoTop_coeffs(
   StoTop_terms == T_Mid)   0
                                          0
                               0
                               0
                               0
                               StoTop_coeffs(StoTop_terms ==
    T_Mid*charge) + StoTop_coeffs(StoTop_terms == T_Mid*
   charge^2)   StoTop_coeffs(StoTop_terms == T_Mid*charge^2*
   dV_BHKW) +  StoTop_coeffs(StoTop_terms == T_Mid*charge*
   dV_BHKW)    StoTop_coeffs(StoTop_terms == T_BHKW*charge
   ^3*dV_BHKW)

   StoTop_coeffs(StoTop_terms == T_Top*charge)

   StoTop_coeffs(StoTop_terms == T_Top*charge*dV_BHKW)     0


   0

   0

   0

   0];
F_phi(3,1:kombis) = [0
   0

   StoMid_coeffs(StoMid_terms == T_Top)   StoMid_coeffs(
   StoMid_terms == T_Mid)   StoMid_coeffs(StoMid_terms ==
   T_Bot)  0                                 0
                               0
                               0
                               StoMid_coeffs(StoMid_terms ==
    T_Mid*charge)
```

```
   StoMid_coeffs(StoMid_terms == T_Mid*charge*dV_BHKW)

   0

   StoMid_coeffs(StoMid_terms == T_Top*charge) +
   StoMid_coeffs(StoMid_terms == T_Top*charge^2)
   StoMid_coeffs(StoMid_terms == T_Top*charge^2*dV_BHKW)
   StoMid_coeffs(StoMid_terms == T_Bot*charge) +
   StoMid_coeffs(StoMid_terms == T_Bot*charge^2)
   StoMid_coeffs(StoMid_terms == T_Bot*charge^2*dV_BHKW) +
   StoMid_coeffs(StoMid_terms == T_Bot*charge*dV_BHKW)   0

   0

   0];
F_phi(4,1:kombis) = [StoBot_coeffs(StoBot_terms == T_Rl)
   StoBot_coeffs(StoBot_terms == T_BHKW*charge^3) +
   StoBot_coeffs(StoBot_terms == T_BHKW*charge^2) +
   StoBot_coeffs(StoBot_terms == T_BHKW*charge)      0
                                        StoBot_coeffs(
   StoBot_terms == T_Mid)   StoBot_coeffs(StoBot_terms ==
   T_Bot)  0                              0
                              0
                              0
                              StoBot_coeffs(StoBot_terms ==
    T_Mid*charge) + StoBot_coeffs(StoBot_terms == T_Mid*
   charge^2)   StoBot_coeffs(StoBot_terms == T_Mid*charge^2*
   dV_BHKW)

   StoBot_coeffs(StoBot_terms == T_BHKW*charge^3*dV_BHKW) +
   StoBot_coeffs(StoBot_terms == T_BHKW*charge^2*dV_BHKW)
   0

   0
   StoBot_coeffs(StoBot_terms == T_Bot*charge)
```

```
   StoBot_coeffs(StoBot_terms == T_Bot*charge*dV_BHKW)

   StoBot_coeffs(StoBot_terms == T_Rl*charge*dV_BHKW) +
   StoBot_coeffs(StoBot_terms == T_Rl*charge^2*dV_BHKW) +
   StoBot_coeffs(StoBot_terms == T_Rl*charge^3*dV_BHKW)
   StoBot_coeffs(StoBot_terms == T_Rl*charge) +
   StoBot_coeffs(StoBot_terms == T_Rl*charge^2) +
   StoBot_coeffs(StoBot_terms == T_Rl*charge^3)    0];


F_Rl = invertsecondrow(F_Rl);
F_Qab = invertsecondrow(F_Qab);
F_Top = invertsecondrow(F_Top);
F_Mid = invertsecondrow(F_Mid);
F_Bot = invertsecondrow(F_Bot);
F_dV_BHKW = invertsecondrow(F_dV_BHKW);
F_T_BHKW = invertsecondrow(F_T_BHKW);
F_charge = invertsecondrow(F_charge);
F_mod = invertsecondrow(F_mod);

function prod = easykron(in)
    res = 1;
    for i=1:length(in)
        col(2,1) = in(1,i);
        col(1,1) = 1;
        res = kron(res,col);
    end
    prod = res;
end

function mat = invertsecondrow(in)
    for i=1:length(in)
        in(1,i) = abs(in(2,i)-1);
    end
    mat = in;
end
```

```
85
86  function [dT_Rl] = getConsumerEq()
87  syms Q_ab T_Verbraucher  T_Rl
88  dV_Verbraucher = 5.5e-05;
89  c = 4182;
90  rho = 1000;
91  V_Verbraucher = 0.054;
92
93  dT_Rl = (c*rho*dV_Verbraucher*T_Verbraucher - Q_ab - T_Rl*c*
        rho*dV_Verbraucher)/(c*rho*V_Verbraucher);
94  end
95
96  function dT_BHKW = getBHKWEq()
97
98  syms T_Rl T_BHKW BHKW_an_above_half BHKW_an_above_zero
        T_BHKW_above_low T_BHKW_above_upper Vstrom_above_low
        Vstrom_above_upper
99  mod = sym('mod');
100 V_BHKW_laden = 0.052;
101 V_BHKW_entladen = 0.075;
102 P_min = 19;
103 P_max = 36;
104 k_BHKW_laden = 50.46;
105 k_BHKW_entladen = k_BHKW_laden;
106 c = 4182;
107 rho = 1000;
108
109 T_Umgebung = 295;
110
111 % Saturation Boundaries
112 Vstrom_min = 1.38e-4;
113 Vstrom_max = 5.55e-4;
114 T_BHKW_min = 273;
115 T_BHKW_max = 353;
116 %
117
```

```
118 % BHKW_an_aus
119 %BHKW_an_above_half = heaviside(BHKW_an - 0.5);
120 %BHKW_an_above_zero = heaviside(BHKW_an - 1.0e-6); % TODO
       more beautiful solution
121
122
123 V_BHKW =((1-BHKW_an_above_half)*V_BHKW_entladen + (
       BHKW_an_above_half)*V_BHKW_laden );
124 k_BHKW = ((1-BHKW_an_above_half)*k_BHKW_entladen + (
       BHKW_an_above_half)*k_BHKW_laden );
125
126 Q_zu= BHKW_an_above_zero * ((P_max-P_min)*mod + P_min)*1000;
127 %
128
129 %Saturation T_BHKW
130 %T_BHKW_above_low = heaviside(T_BHKW- T_BHKW_min);
131 %T_BHKW_above_upper = heaviside( T_BHKW  - T_BHKW_max);
132
133 T_BHKW_Term = ((1 - T_BHKW_above_low) * T_BHKW_min + ...   %
       Lower limit
134     T_BHKW_above_low * (1 - T_BHKW_above_upper) * T_BHKW +
          ...%  Gain 1 Zone
135     T_BHKW_above_upper * T_BHKW_max); % Upper limit
136 %
137
138 %Saturation VStrom
139 Vstrom = (mod*P_max*1000) / ...
140     ((353-T_Rl)*c*rho) ;
141
142 %Vstrom_above_low = Vstrom/(2*Vstrom_min);  %% binary
143 %Vstrom_above_upper = Vstrom/(2*Vstrom_max); %% binary
144
145 Vstrom = ((1 - Vstrom_above_low) * Vstrom_min + ...   %
       Lower limit
146     Vstrom_above_low * (1 - Vstrom_above_upper) * Vstrom +
          ...%  Gain 1 Zone
```

```
147     Vstrom_above_upper * Vstrom_max) * ... ; % Upper limit
148     BHKW_an_above_half; % ON/OFF
149
150 %QVl-Q_Rl
151 Q_VLQ_Rl = ((T_BHKW_Term - T_Rl) * Vstrom)*c*rho;
152
153
154 Q_Verlust = (T_BHKW_Term - T_Umgebung) * k_BHKW;
155
156  dT_BHKW = ( Q_zu - Q_VLQ_Rl - Q_Verlust) / (V_BHKW*c*rho);
157
158
159 end
```

A.5 Model of storage in isolation

Listing A.1: Model of the storage in isolation, simulation in comparison with simulink

```
1
2 % Get the differential equations
3 dT_StoTop = getStorageEq('Top');
4 dT_StoMid = getStorageEq('Middle');
5 dT_StoBot = getStorageEq('Bottom');
6
7 % Declare the states and inputs
8 syms charge T_Top T_Mid T_Bot dV_BHKW T_BHKW T_Rl dV_Rl rho
     Cp
9 states = [T_Top T_Mid T_Bot charge];
10 inputs = [dV_BHKW T_BHKW T_Rl];
11
12 % Optional: If the differentials contain binary states, you
     can substitute all powers of it with a power of one
13 syms charge
14 dT_StoTop = ...
15     substitute_powers(dT_StoTop,charge,[int32(2) int32(3)]);
```

```
dT_StoMid = ...
    substitute_powers(dT_StoMid,charge,[int32(2) int32(3)]);
dT_StoBot = ...
    substitute_powers(dT_StoBot,charge,[int32(2) int32(3)]);

% Get the coeffieicients of the multilinear terms
[StoTop_coeffs,StoTop_terms] = coeffs(dT_StoTop,[states
   inputs]);
[StoMid_coeffs,StoMid_terms] = coeffs(dT_StoMid,[states
   inputs]);
[StoBot_coeffs,StoBot_terms] = coeffs(dT_StoBot,[states
   inputs]);

% Get multilinear terms from all diff equations and order
   them
u_seq = unique([StoTop_terms StoMid_terms StoBot_terms]);


% Manually set the factor matrices according to u_seq
F_Top =      [0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1];
F_Mid=       [0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0];
F_Bot =      [0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0];
F_charge =   [0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1];
F_dV_BHKW =  [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1];
F_T_BHKW =   [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0];
F_T_Rl =     [0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0];


%Create empty symbolic phi matrix
Phi =    sym('phi',[length(states) length(u_seq)]);
% Automatically fill the phi matrix
F_Phi = fillPhi(Phi,{StoTop_terms,StoMid_terms,StoBot_terms
   ,[]},{StoTop_coeffs,StoMid_coeffs,StoBot_coeffs,[]},u_seq
   );
udv = 5.5e-5;
F_Phi = subs(F_Phi,[dV_Rl rho Cp],[udv 1000 4200]);
```

```
% Convert to Table for inspection
F_Phi_table = array2table(F_Phi,'RowNames',string(states),'
    VariableNames',string(u_seq));

% Simulation Inputs and Time-vector
u0 = 0:20:15e4;
u0 = u0.';
u1 = zeros(length(u0),1);
u1(:)=7e-05;

u2 = zeros(length(u0),1);
u2(:)=273.15+60;

u3 = zeros(length(u0),1);
u3(:)=273.15+45;


u = [u0 u1 u2 u3];

ntype=1; %norm 1
x0=[341;310;304;u1(1,1)>=5.5e-05]; %initial states
Ts=0;   %continous time system= 0, discret time=1
n=4;
m=3;

F=[F_Top;F_Mid;F_Bot;F_charge;F_dV_BHKW;F_T_BHKW;F_T_Rl;
    F_Phi];
F = double(F);
mod=mticpn(F,x0,Ts,ntype,n,m);
si1=simulateMTI(mod,u);
si2 = sim('only_storage');

tic;
simContOde45Mti(si1,u(:,1)); %for discrete simDiscMTI(sys,
    Tend)
tsi=toc;
```

```
%plotSim(si1)
clf;
hold on;
xlabel('Time [s]')
ylabel('Temperature [K]')
title('Heat storage Simulation (Multilinear v Simulink)')
p1 = plot(u0,si2.logsout{2}.Values.Data,'green');
plot(u0,si2.logsout{1}.Values.Data,'green');
plot(u0,si2.logsout{3}.Values.Data,'green');
p2 =plot(u0,si1.xsim(:,1),'red');
plot(u0,si1.xsim(:,2),'red');
plot(u0,si1.xsim(:,3),'red');
hold off;
legend([p1 p2],{'Simulink model','Mulitlinear model'},'
   Location','northeast')
str = {strcat('dV_{BHKW} = ',string(u1(1))),strcat('T_{BHKW}
    = ',string(u2(1))),strcat('dV_{Rl} = ',string(udv)),
   strcat('T_{Rl} = ',string(u3(1)))};
text(10e4,330,str);
function r = fillPhi(phi,terms,coeffs,seq)
    for j=1:size(phi,1)
        for i=1:length(phi(j,:))
            coeff = coeffs{j}(terms{j} == seq(i));
            if isempty(coeff)
                phi(j,i) = 0;
            else
                phi(j,i) = coeff;
            end
        end
    end
    r = phi;
end

function s = substitute_powers(eq,term,powers)
    arguments
        eq (1,1) sym
```

```
112         term (1,1) sym
113         powers (1,:) int32
114     end
115     for i=1:length(powers)
116         eq = subs(expand(eq),term^powers(1,i),term);
117     end
118
119     s = eq;
120 end
```

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original