ulm university universität
uulm

Multi-Objective Optimization of Simple and Multiple Dividing Wall Columns and their Operational Flexibility Close to the Optimum

Supplementary Information for Dissertation

to obtain the academic title of

Doktor-Ingenieur
(Dr.-Ing.)

at the Faculty of Engineering,
Computer Science and Psychology at the University of Ulm

by

Lena-Marie Ränger
from Hamburg

First examiner:	Prof. Dr.-Ing. Thomas Grützner
Second examiner:	Prof. Dr.-Ing. Hans Hasse
Acting Dean:	Prof. Dr.-Ing. Maurits Ortmanns

Ulm, November 16, 2021

Contents

1 Introduction

The following file provides additional information for the PhD Thesis with the title "Multi-Objective Optimization of Simple and Multiple Dividing Wall Columns and their Operational Flexibility Close to the Optimum". The manuscript is organized as follows. First fundamentals of the calculation of $\dot{V}_{min}$ diagrams and its implementation in MATLAB is presented in Section 2. Afterwards, the visual basic for applications (VBA) code used for the data exchange between Aspen Plus and Excel is presented in Section 3. Section 4 shows the VBA code used for the automated screening. Last, the MATLAB code used for the calculation of self-organizing patch plots (SOPPs) can be found in Section 5. Note that the sandwiching algorithm used for optimization is intellectual property of the Fraunhofer Gesellschaft, thus it will not be provided here.

2 $\dot{V}_{min}$ diagrams

In the main document of this work the general appearance of $\dot{V}_{min}$ diagrams has already been described. Accordingly, only the main facts are summarized here:

- There are maxima and minima describing the vapor demand for sharp splits between the components
- For a multi component feed mixture (n = number of components) there are $\frac{n\cdot(n-1)}{2}$ sharp splits, thereof $(n-1)$ without distributing component, $(n-2)$ with one distributing component etc. and $\frac{(n\cdot(n-1))}{2}+2$ significant points including:
  - First boundary condition: no vapor results in no distillate (0,0)
  - Second boundary condition: distillate is at least the vapor fraction of the feed $(1{,}1-q)$
- Infeasible region: $\dot{V} < \dot{D}$ and $\dot{V} < (1-q)\cdot\dot{F}$
- Preferred split: Split between the lightest and heaviest component fed at the corresponding minimum vapor flow

In the following, first the theoretical calculation of $\dot{V}_{min}$ diagrams is presented (Section 2.1) followed by the implementation of the procedure in MATLAB (Section 2.2).

2.1 Theoretical calculation

$\dot{V}_{min}$ diagrams are based on Underwood equations [1–4]. The methodology was adapted for the calculation of $\dot{V}_{min}$ diagrams by Halvorsen [5–7]. The analytical procedure for their calculation is summarized in this section. For this purpose, first the Underwood roots will be introduced briefly. Afterwards the calculation steps are presented in the sections 2.1.1 to 2.1.3. Note that most of the text was already published elsewhere [8]. To understand the calculation procedure, first Underwood Roots have to be introduced. For this purpose a mass balance is performed around a simple distillation column with a feed stream at its middle, with the rectifying section above the feed stage and the stripping section below. An infinite number of stages, a constant molar flow, and constant relative volatilities of the components are assumed. The latter one is denoted as α_i, which is the distribution ratio of component i K_i compared to the distribution ratio of the overall heavy boiling component K_{HB} in the original feed mixture, as shown in Equation 2.1.

$$\alpha_i = \frac{\frac{y_i}{x_i}}{\frac{y_{HB}}{x_{HB}}} = \frac{K_i}{K_{HB}} \tag{2.1}$$

where x_i, is the molar fraction of component i in the liquid phase and y_i, is the molar fraction of component i in the vapor phase. With these assumptions, Underwood [4] defined equations 2.2 and 2.3 [9].

$$\sum_{i=1}^{n} \frac{\alpha_i \cdot x_i^T}{\alpha_i - \Phi} = R^T + 1 \tag{2.2}$$

$$\sum_{i=1}^{n} \frac{\alpha_i \cdot x_i^B}{\alpha_i - \Psi} = R^B \tag{2.3}$$

The superscript T indicates the top of the column and B is the bottom; accordingly, R^T is the reflux ratio, R^B is the boil up ratio, and Φ and Ψ are the Underwood roots. There are different roots above and below the feed stage, which are called actual roots:

- Φ is the actual root in the rectifying section
- Ψ is the actual root in the stripping section

A pinch point analysis results in the following definitions for the actual roots: Equation 2.4 for the rectifying section and Equation 2.5 for the stripping section [9].

$$\Phi = \frac{\dot{L}^T}{\dot{V}^T \cdot K_{HB}} \tag{2.4}$$

$$\Psi = \frac{\dot{L}^B}{\dot{V}^B \cdot K_{HB}} \tag{2.5}$$

$\dot{L}$ is the molar liquid stream and $\dot{V}$ the molar vapor stream. These internal streams differ depending on the performed product split. Hence, there are as many actual roots as components n in each section and their values are between the values of the relative volatilities of each component α_i (Equation 2.6).

$$\begin{aligned} &\alpha_1 > \Phi_1 > \alpha_2 > \Phi_2 > .... > \Phi_n \\ &\Psi_1 > \alpha_1 > \Psi_2 > \alpha_2 > .... > \alpha_n \end{aligned} \tag{2.6}$$

A decreasing vapor flow results in increasing Ψ and decreasing Φ, which means that there is an intersection of the top and bottom roots. At this value, the minimum vapor flow $\dot{V}_{min}$ is reached and the roots are called common roots θ. There are $n-1$ common roots fulfilling Equation 2.7.

$$\alpha_1 > \Phi_1 = \Psi_2 = \theta_1 > \alpha_2 > .... > \alpha_n \tag{2.7}$$

The common roots are called active roots θ^* if their value lies between the values of relative volatilities of the components, which are split as shown in Figure 2.1a. With

n_{dist}, the number of distributing components (components with $r_i^T \neq 1$ or 0) there is $n_{dist} + 1$ active roots:

1.) For sharp splits (no distributing component), there is always only one active root, which is the one with a value between the volatilities of the components split

2.) For sharp splits with one distributing component, there are two active roots

3.) For sharp splits with two distributing components, there are three active roots etc.

Figure 2.1b symbolically shows all kinds of Underwood roots in a distillation column for a sharp AC split with four components fed. $\dot{V}_{min}$ diagrams are based on these

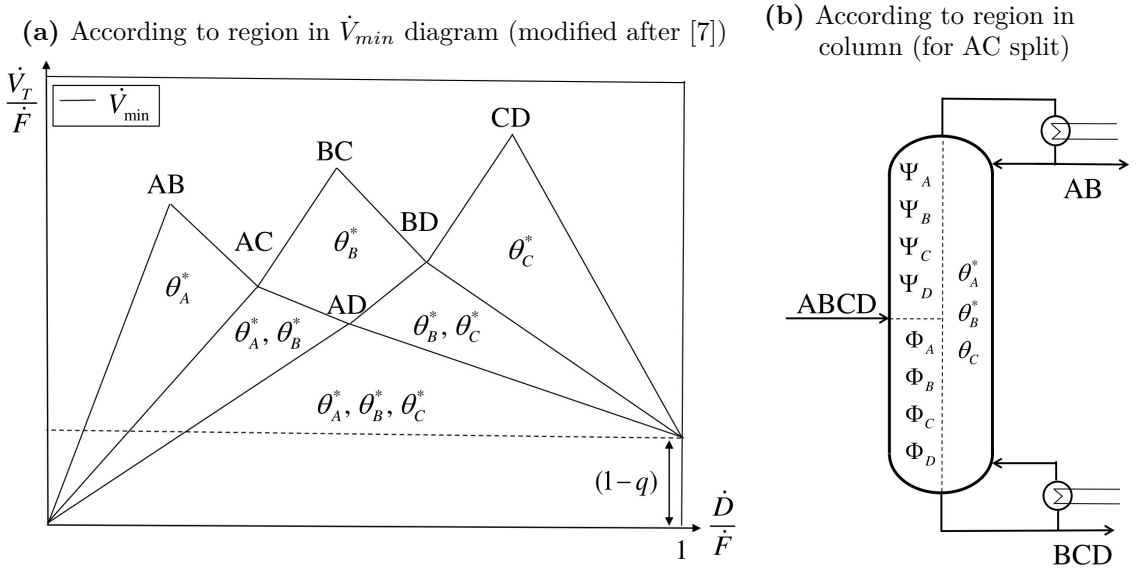

Figure 2.1: Active Underwood roots for several cases (modified after [8]).

Underwood roots. The general calculation procedure is always the same, independent of the number of components. First, the Underwood roots of the feed stream have to be calculated (Section 2.1.1). Second, in the case of sharp splits with distributing components, the recoveries of the distributing components have to be determined. If there are no splits with distributing components, this step is canceled. Third, the minimum vapor flow, as well as the distillate streams, can be calculated with the active roots and the recoveries. The second and third step are performed according to Section 2.1.2. Note that only the extreme points of the $\dot{V}_{min}$ diagram are calculated, which can be connected by straight lines. The resulting $\dot{V}_{min}$ diagram is valid for dividing wall columns as such. However, if one of the columns does not work at its preferred split, an adaptation has to be performed in the diagram according to the method from Section 2.1.3.

2.1.1 Calculation of the common roots

The common roots of the feed mixture are calculated with the first Underwood equation which is also called the feed equation (Equation 2.12). In the following, the derivation of the equation is presented. First, a vapor flow balance around the column is performed, as shown in Figure 2.2. $\dot{F}$ is the feed stream and $\dot{V}$ is the vapor stream; the superscripts

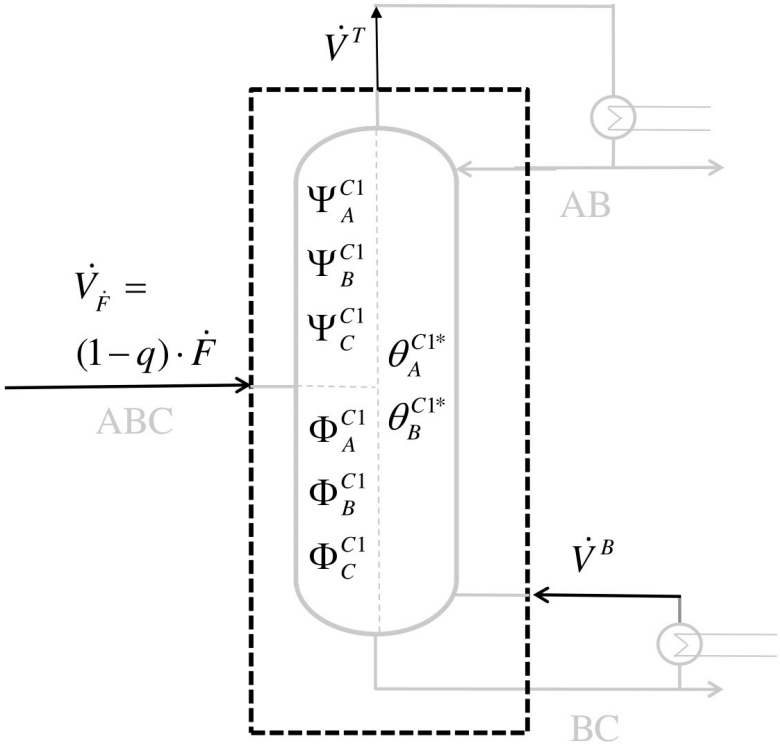

Figure 2.2: Vapor balance around a column C1 for the separation of a three component mixture ABC (modified after [8]).

B, T, and $\dot{F}$ indicate the streams at the bottom and top of the column and the feed stream, respectively; and q is the liquid fraction of the feed. The vapor balance results in Equation 2.8.

$$(1-q)\cdot\dot{F}+\dot{V}^B=\dot{V}^T \tag{2.8}$$

Based on the Underwood roots, the vapor stream in the top of the column is defined by Equation 2.9 (for the derivation, see [10]).

$$\dot{V}^T=\sum_{i=1}^{n}\frac{\alpha_i\dot{D}_i}{\alpha_i-\Phi}=\sum_{i=1}^{n}\frac{\alpha_i\cdot\dot{F}\cdot z_i\cdot r_i^T}{\alpha_i-\Phi} \tag{2.9}$$

$\dot{D}_i$ is the molar flow of component i in the distillate stream, z_i is the molar concentration of component i in the feed stream, and r_i^T is the recovery of the component in the distillate stream (Equation 2.10).

$$r_i^T = \frac{\dot{D}_i}{\dot{F}_i} \tag{2.10}$$

The vapor flow at the bottom of the column is defined by Equation 2.11.

$$\dot{V}^B = \sum_{i=1}^{n} \frac{\alpha_i \cdot \dot{B}_i}{\alpha_i - \Psi} = \sum_{i=1}^{n} \frac{\alpha_i \cdot \dot{F} \cdot z_i \cdot r_i^B}{\alpha_i - \Psi} \tag{2.11}$$

$\dot{B}_i$ is the molar flow of component i in the bottom stream and r_i^B is the corresponding recovery related to the fed stream of the component. The minimum vapor flow is reached at $\Phi = \Psi = \theta$, with $r_i^B + r_i^T = 1$. A combination of Equation 2.8 to 2.11 results in the so called feed equation, which is used to calculate the common roots (Equation 2.12). Note that r_i^B is defined for the outgoing bottom product, whereas the bottom vapor stream is incoming, which results in a negative sign that causes the recoveries to be canceled out.

$$\dot{V}^{\dot{F}} = (1 - q) \cdot \dot{F} = \sum_{i=1}^{n} \frac{\alpha_i \cdot \dot{F} \cdot z_i}{\alpha_i - \theta} \tag{2.12}$$

2.1.2 Calculation of minimum vapor flow

The second Underwood equation Equation 2.13) is used to calculate $\dot{V}_{min}$ depending on the active root θ^*. Note that Equation 2.13 represents Equation 2.9 for the $\dot{V}_{min}$ case, meaning $\Phi = \theta$. Note that in most literature the dot above $\dot{V}_{min}$ is not shown, instead it is written as V_{min} .

$$\dot{V}_{min} = \sum_{i=1}^{n} \frac{\alpha_i \cdot \dot{D}_i}{\alpha_i - \theta^*} = \sum_{i=1}^{n} \frac{\alpha_i \cdot z_i \cdot \dot{F}}{\alpha_i - \theta^*} \cdot r_i^T \tag{2.13}$$

The index $*$ means that only the active root of the split is used for the calculation. The corresponding distillate stream is calculated according to Equation 2.14.

$$\dot{D} = \sum_{i=1}^{n} z_i \cdot r_i^T \cdot \dot{F} \tag{2.14}$$

The obtained data for $\dot{V}_{min}$ and $\dot{D}$ divided by the feed stream are plotted in the $\dot{V}_{min}$ diagram.

2.1.3 Adaptations of $\dot{V}_{min}$ diagrams

The $\dot{V}_{min}$ diagram can be applied to dividing wall columns without changes. The only condition for its validity is that all column sections perform their preferred split. The corresponding preferred splits for the column sections are shown in the main document of this work. It was stated, that the simplified multiple dividing wall column has a section, in which not the preferred split is performed. The necessary adaptation of the $\dot{V}_{min}$ diagram can be performed according to the scheme presented in Figure 2.3.

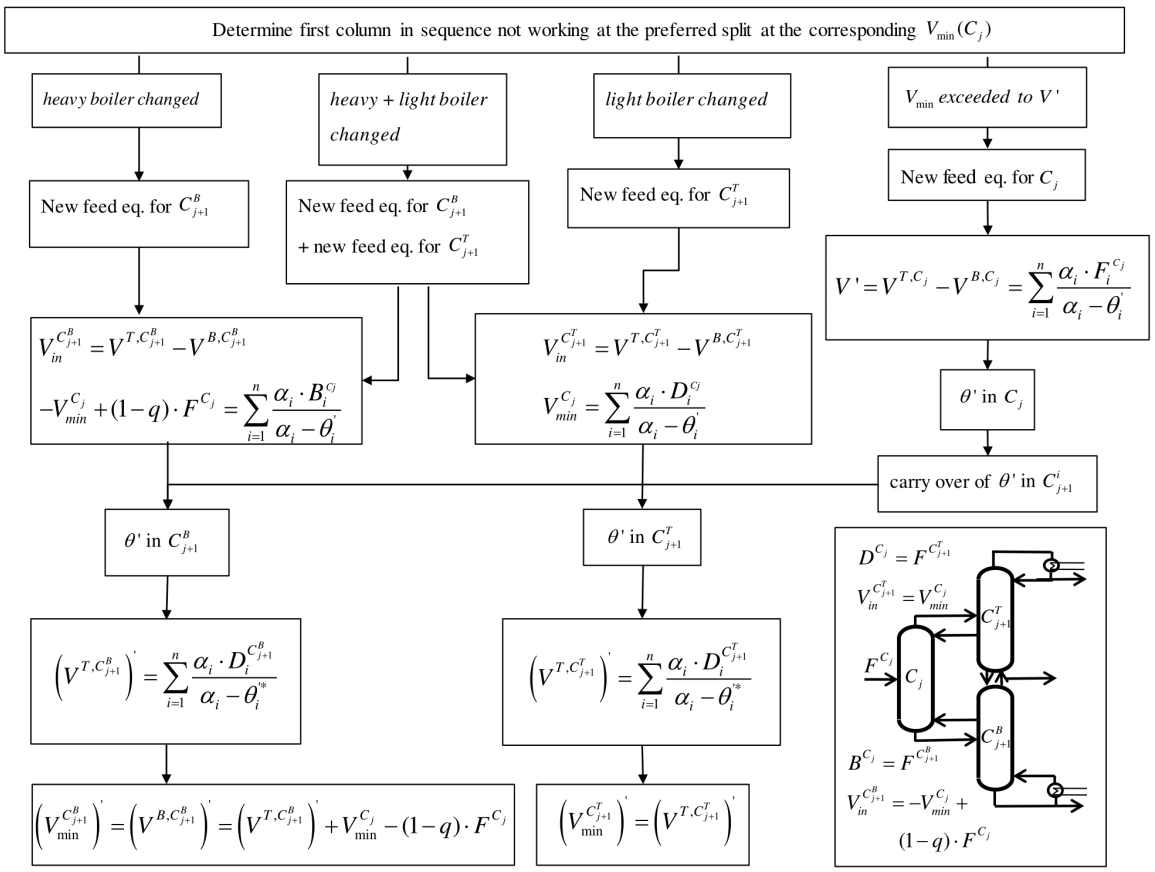

Figure 2.3: Proceeding for adaptations of $\dot{V}_{min}$ diagrams in case of non-preferred splits of thermally and materially coupled columns [8].

2.2 Automated calculation in Matlab

For the calculation of $\dot{V}_{min}$ diagrams for two, three or four component systems a Matlab code is implemented. Note that the symbolic toolbox is required.

```
n_comp = 4; %number of components; works for 2, 3 and 4

% Available Versions for 4 component dividing wall columns:
% Simplified = 0: three dividing walls
```

```
% Simplified = 1: two dividing walls, one in the middle and the top one
% behind
% Simplified = 2: two dividing walls, one in the middle and the bottom one
% behind
% Simplified = 3: Kaibel column
Simplified = 1;

% Available column sequences
DirectSplitOption = 1; %For quaternarya direct-direct
IndirectSplitOption = 1; %for quaternary indirect-indirect
if n_comp ==4
DirectIndirectSplitOption = 1;
IndirectDirectSplitOption = 1;
end

% Plot diagram? 0=no, 1=yes
PlotDiagram = 1;

q = 1; % liquid fraction of feed
F=100; % Total feed flow

alpha = ones (1,n_comp); % vector of relative volatility of components
z_Feed = ones(1,n_comp); % vector of composition of feed

%Ki (x/y) of all components
K(1,1) = 1.9597;
K(1,2) = 0.9089;
K(1,3) = 0.6667;
K(1,4)= 0.4985;

%Alpha (Ki/KHK) for all components
for k = 1 : n_comp
alpha(1, k) = K(1,k)/K(1,n_comp);
end
clear k;

%or type in given alphas
% alpha(1, 1) = 4.7;
% alpha(1, 2) = 2.2;
% alpha(1, 3) = 1.5;
% alpha(1, 4) = 1;

%Molar fraction of components in feed
%either equimolar
```

```
% z_Feed(1,:)=1/n_comp;
%or not (choose one)
z_Feed(1,1) = 0.25;
z_Feed(1,2) = 0.25;
z_Feed(1,3) = 0.2;
z_Feed(1,4) = 0.3;

%% Calculate Roots
n_roots = n_comp-1;
com_roots = ones(1,n_roots); %vector for common roots

%calculating active roots
eqn = 0;
%Sum of feed equation
for j = 1:n_comp
syms roots;
eqn = eqn + alpha(1,j)*z_Feed(1,j)/(alpha(1,j)-roots);
end
eqn = eqn - (1-q);
%solving feed equation (numerically)
com_roots(1,:) = vpasolve(eqn,roots,[alpha(1,1) alpha(1,n_comp)]);
com_roots(1,:)=sort(com_roots(1,:),'descend');
clear eqn roots

%Matrix Comp describes alpha*z/(alpha-root) (for all cases!)
Comp = ones(n_roots,n_comp);

for p=1:n_roots
for r=1:n_comp
Comp(p,r)=alpha(1,r)*z_Feed(1,r)/(alpha(1,r)-com_roots(1,p));
end
end
clear p r;

% Creation of Vmin/F and D/F vectots
VF=zeros(1,n_comp*(n_comp-1)/2+2);
DF=zeros(1,n_comp*(n_comp-1)/2+2);
VF(1,1)=0;
DF(1,1)=0;
VF(1,n_comp*(n_comp-1)/2+2)=(1-q);
DF(1,n_comp*(n_comp-1)/2+2)=1;

%% for 2 components
if n_comp == 2
```

```
% Recovery matrix describes the recoveries for every split option
% Lines describe components
% Rows describe the splits
% order of rows: AB

Rec = zeros(n_comp,n_comp*(n_comp-1)/2);
Rec(1,1)=1;
VF(1,2) = Comp(1,1) * Rec (1,1);
VF(1,3) = (1-q);
DF(1,2:n_comp*(n_comp-1)/2+1)=z_Feed*Rec;
DF(1,3)=1;

if PlotDiagram ==1
figure('DefaultAxesFontSize', 15);
plot(DF, VF,':ok');
xlabel('D/F');
ylabel('V/F');
end
%% for 3 components
elseif n_comp == 3
% Recovery matrix describes the recoveries for every split option
% Lines describe components
% Rows describe the splits
% order of rows: AB, AC (B dist), BC

Rec = zeros(n_comp,n_comp*(n_comp-1)/2); %rc is always zero
Rec(1,:)=1; % ra is always one
Rec(2,3)=1; % BC split
% AC split
syms rb
eq=Comp(1,1)*1+Comp(1,2)*rb; %ra is 1, rc is 0
eq0=Comp(2,1)*1+Comp(2,2)*rb;
Rec(2,2)=solve(eq==eq0,rb);
%clear rb eq eq0;

VF(1,2)=Comp(1,:)*Rec(:,1);
VF(1,3)=Comp(1,:)*Rec(:,2);
VF(1,4)=Comp(2,:)*Rec(:,3);
DF(1,2:n_comp*(n_comp-1)/2+1)=z_Feed*Rec;
if PlotDiagram ==1
figure('DefaultAxesFontSize', 15);
plot(DF, VF,':ok');
xlabel('D/F');
ylabel('V/F');
```

```
end
%% for 4 components
elseif n_comp == 4
% Recovery matrix describes the recoveries for every split option
% Lines describe components
% Rows describe the splits
% order of rows: AB, AC (B dist), BC, AD (BC dist), BD (C dist), CD
% if simplified = 1, row 7 is AC', row 8 AB'
% if simplified = 2, row 7 is BD', row 8 CD'
Rec = zeros(n_comp,n_comp*(n_comp-1)/2); %rd is always zero
Rec(1,:)=1; %ra is always one
Rec(2,3)=1; %sharp BC split (row 3)
Rec(2,6)=1; %sharp CD split (row 6)
Rec(3,6)=1; %sharp CD split (row 6)
syms rb rc
eq1=Comp(1,1)*1+Comp(1,2)*rb+Comp(1,3)*rc; %ra is 1, rd is 0
eq2=Comp(2,1)*1+Comp(2,2)*rb+Comp(2,3)*rc;
eq3=Comp(3,1)*1+Comp(3,2)*rb+Comp(3,3)*rc;

% Recovery matrix for distributing cases (row 2,4,5)
% AC (root 1 and 2 active) (row 2)
eq4=subs(eq1,rc,Rec(3,2));
eq5=subs(eq2,rc,Rec(3,2));
Rec(2,2)=solve(eq4==eq5,rb);
clear eq4 eq5;
% BD (root 2 and 3 active) (row 5)
Rec(2,5)=1;
eq6=subs(eq2,rb,Rec(2,5));
eq7=subs(eq3,rb,Rec(2,5));
Rec(3,5)=solve(eq6==eq7,rc);
clear eq6 eq7;
% AD (root 1, 2, 3 active) (row 4)
[A,B]=equationsToMatrix([eq1==eq2, eq2==eq3], [rb, rc]);
X=linsolve(A,B);
Rec(2,4)=X(1);
Rec(3,4)=X(2);
clear A B;
clear eq1 eq2 eq3 rb rc X

% VF and DF vectors
VF(1,2)=Comp(1,:)*Rec(:,1);
VF(1,3)=Comp(1,:)*Rec(:,2);
VF(1,4)=Comp(2,:)*Rec(:,3);
VF(1,5)=Comp(2,:)*Rec(:,4);
```

```
VF(1,6)=Comp(3,:)*Rec(:,5);
VF(1,7)=Comp(3,:)*Rec(:,6);
DF(1,2:n_comp*(n_comp-1)/2+1)=z_Feed*Rec;

%% if simplified version is calculated, this is done here
% two dividing walls, only the upper one in the second row
if Simplified == 1
% new root A
eq8=0;
syms rootAnew

for r=1:n_comp
eq8 = eq8 + alpha(1,r)*z_Feed(1,r)*Rec(r,5)/(alpha(1,r)-rootAnew);
end
eq8=eq8-VF(1,6);
com_roots(1,n_comp)= vpasolve(eq8,rootAnew,[alpha(1,1) alpha(1,2)]);
clear rootAnew eq8 r;

for k=1:n_comp
Comp(4,k)=alpha(1,k)*z_Feed(1,k)/(alpha(1,k)-com_roots(1,n_comp));
end
clear k;
% new recovery B of AC split
% Rec matrix column 7 is AC' split
syms rbnew
eq9 = Comp(2,1)*1+Comp(2,2)*rbnew;
eq10 = Comp(4,1)*1+Comp(4,2)*rbnew;
Rec(1,7) = 1;
Rec(2,7)=solve(eq9==eq10,rbnew);
clear eq9 eq10 rbnew
%neues VminAC'
VF(1,9) = Comp(2,1)*1+Comp(2,2)*Rec(2,7);
DF(1,9) = z_Feed (1,1) * Rec(1,7) + z_Feed (1,2) * Rec(2,7);

%neues Vmin AB'
VF(1,10) = Comp(4,1);
DF(1,10) = z_Feed(1,1);

% two dividing walls, only the lower one in the second row
elseif Simplified == 2

% new Root C
eq9=0;
syms rootCnew
```

```
for r=1:n_comp
eq9 = eq9 + alpha(1,r)*z_Feed(1,r)*(1-Rec(r,2))/(alpha(1,r)-rootCnew);
end
eq9=eq9+VF(1,3)-(1-q);
com_roots(1,n_comp)= vpasolve(eq9,rootCnew,[alpha(1,3) alpha(1,4)]);
clear rootCnew eq9
% Comp um Zeile mit neuer Root C ergï¿½nzen (Zeile 4)

for k=1:n_comp
Comp(4,k)=alpha(1,k)*z_Feed(1,k)/(alpha(1,k)-com_roots(1,n_comp));
end

% new recovery of C in BD split
% Rec matrix column 7 is BD' split
syms rcnew
eq11 = Comp(2,2)*(1-Rec(2,2))+Comp(2,3)*rcnew;
eq12 = Comp(4,2)*(1-Rec(2,2))+Comp(4,3)*rcnew;
Rec(1,7) = 1;
Rec(2,7) = 1;
Rec(3,7)=solve(eq11==eq12,rcnew);
clear eq11 eq12 rcnew
% new Vmin BD'
VF(1,9) = VF(1,3)+Comp(2,2)*(1-Rec(2,2))+Comp(2,3)*Rec(3,7);
DF(1,9) = z_Feed (1,1) * Rec(1,7) + z_Feed (1,2) * Rec(2,7) + z_Feed (1,3) * Rec
    ↪ (3,7);

% new Vmin CD'
VF(1,10) = VF(1,9) + Comp(4,3)*(1-Rec(3,7));
DF(1,10) = z_Feed(1,1) + z_Feed(1,2) + z_Feed(1,3);
% Kaibel column
elseif Simplified == 3

% new root A
eq13=0;
syms rootAnew

for r=1:n_comp
eq13 = eq13 + alpha(1,r)*z_Feed(1,r)*Rec(r,5)/(alpha(1,r)-rootAnew);
end
eq13=eq13-VF(1,4);%-(1-q);
com_roots(1,n_comp)= vpasolve(eq13,rootAnew,[alpha(1,1) alpha(1,2)]);
clear rootAnew eq13 r;
```

```
for k=1:n_comp
Comp(4,k)=alpha(1,k)*z_Feed(1,k)/(alpha(1,k)-com_roots(1,n_comp));
end
clear k;

% new Vmin AB'
VF(1,9) = Comp(4,1);
DF(1,9) = z_Feed (1,1);

% new root C
eq14=0;
syms rootCnew

for r=1:n_comp
eq14 = eq14 + alpha(1,r)*z_Feed(1,r)*(1-Rec(r,2))/(alpha(1,r)-rootCnew);
end
eq14 = eq14+VF(1,4)-(1-q);
com_roots(1,n_comp+1)= vpasolve(eq14,rootCnew,[alpha(1,3) alpha(1,4)]);
clear rootCnew eq14 r;

for k=1:n_comp
Comp(4,k)=alpha(1,k)*z_Feed(1,k)/(alpha(1,k)-com_roots(1,n_comp+1));
end
clear k;
% new Vmin CD'
VF(1,10) = VF(1,4) + Comp(4,3);
DF(1,10) = z_Feed(1,1) +z_Feed(1,2) + z_Feed(1,3);

end

%% plots
if PlotDiagram ==1
figure ('DefaultAxesFontSize', 15);
plot([DF(1, 1:4),DF(1, 6:8)], [VF(1, 1:4),VF(1, 6:8)],':ok');
hold on
plot([DF(1,3), DF(1,5), DF(1,6)], [VF(1,3), VF(1,5), VF(1,6)],':ok');
xlabel('D/F');
ylabel('V/F');

if Simplified == 1
hold on
plot([DF(1,1), DF(1,10), DF(1,9)] , [VF(1,1),VF(1,10), VF(1,9)], ':or');
elseif Simplified == 2
hold on
```

```
plot([DF(1,9), DF(1,10), DF(1,8)] , [VF(1,9),VF(1,10), VF(1,8)], ':or');
elseif Simplified == 3
hold on
plot([DF(1,1), DF(1,9), DF(1,4), DF(1,10), DF(1,8)],[VF(1,1), VF(1,9), VF(1,4), VF
    ↪ (1,10), VF(1,8)], ':or');
end

else
end
end
%% Column sequences
Vmin_DWC = max(VF)*F;
if DirectSplitOption == 1
[Vmin_DS]=DirectSplit(n_comp, z_Feed, alpha, VF, com_roots, F);
end

if IndirectSplitOption == 1
[Vmin_IS]=IndirectSplit(n_comp, z_Feed, alpha, VF, com_roots, F);
end

if n_comp ==4
if DirectIndirectSplitOption == 1
[Vmin_DSIS]=DirectIndirectSplit(z_Feed, alpha, VF, com_roots,F);
end
if IndirectDirectSplitOption == 1
[Vmin_ISDS]=IndirectDirectSplit(z_Feed, alpha, VF, com_roots,F);
end
end
```

The function Vmin_DS calculating the vapor demand of a direct split sequence is shown in the following.

```
function [Vmin_DS] = DirectSplit(n_comp, z_Feed, alpha, VF, com_roots,F)

V1 = VF(1,2);
Feed=ones(3,1);
Feed(1,1)=F;
n_roots = n_comp-1;
com_rootsDS = ones(3,n_roots); %vector for common roots
if n_comp ==3
com_rootsDS(1,:) = com_roots(1,:);
elseif n_comp == 4
com_rootsDS(1,:) = com_roots(1,1:3);
end
```

```
%% Column C2
z_Feed_C2 =0;
for k=2:n_comp
z_Feed_C2 = z_Feed_C2 + z_Feed(1,k);
end
z_Feed(2,2:end)=z_Feed(1,2:end)./z_Feed_C2;
Feed(2,1) = z_Feed_C2*F;
eqn = 0;
%Sum of feed equation
for j = 2:n_comp
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(2,j)/(alpha(1,j)-roots2);
end
%solving feed equation (numerically)
com_rootsDS(2,2:n_roots) = vpasolve(eqn,roots2,[alpha(1,2) alpha(1,n_comp)]);
com_rootsDS(2,2:n_roots)=sort(com_rootsDS(2,2:n_roots),'descend');
clear eqn roots2
V2 = alpha(1,2)*z_Feed(2,2)/(alpha(1,2)-com_rootsDS(2,2));

if n_comp == 3
[Vmin_DS]= V1*Feed(1,1) + V2*Feed(2,1);
elseif n_comp == 4

%% Column C3
z_Feed_C3 =0;
for k=3:n_comp
z_Feed_C3 = z_Feed_C3 + z_Feed(1,k);
end
z_Feed(3,3:end)=z_Feed(1,3:end)./z_Feed_C3;
Feed(3,1) = z_Feed_C3*F;
%calculating active roots
eqn = 0;
%Sum of feed equation
for j = 3:n_comp
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(3,j)/(alpha(1,j)-roots2);
end
%eqn = eqn - (1-q);
%solving feed equation (numerically)
com_rootsDS(3,3) = vpasolve(eqn,roots2,[alpha(1,3) alpha(1,n_comp)]);
clear eqn roots
V3 = alpha(1,3)*z_Feed(3,3)/(alpha(1,3)-com_rootsDS(3,3));

Vmin_DS= V1*Feed(1,1) + V2*Feed(2,1) + V3*Feed(3,1);
```

```
end

end
```

The code to calculate the vapor demand of an indirect split sequence is Vmin_IS .

```
function [Vmin_IS] = IndirectSplit(n_comp, z_Feed, alpha, VF, com_roots, F)

Feed=ones(3,1);
Feed(1,1)=F;
n_roots = n_comp-1;
com_rootsIS = ones(3,n_roots); %vector for common roots
if n_comp ==3
com_rootsIS(1,:) = com_roots(1,:);
elseif n_comp == 4
com_rootsIS(1,:) = com_roots(1,1:3);
end

%% Column C2 feed eq
z_Feed_C2 =0;
for k=1:n_comp-1
z_Feed_C2 = z_Feed_C2 + z_Feed(1,k);
end
z_Feed(2,1:end-1)=z_Feed(1,1:end-1)./z_Feed_C2;
Feed(2,1) = z_Feed_C2*F;

eqn = 0;
for j = 1:n_comp-1
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(2,j)/(alpha(1,j)-roots2);
end
if n_comp == 3
V1 = VF(1,4);
com_rootsIS(2,1) = vpasolve(eqn,roots2,[alpha(1,1) alpha(1,n_comp-1)]);
clear eqn roots2
V2 = alpha(1,1)*z_Feed(2,1)/(alpha(1,1)-com_rootsIS(2,1));
[Vmin_IS]= V1*Feed(1,1) + V2*Feed(2,1);
elseif n_comp == 4

V1 = VF(1,7);

com_rootsIS(2,1:n_roots-1) = vpasolve(eqn,roots2,[alpha(1,1) alpha(1,n_comp-1)
    ↪ ]);
com_rootsIS(2,1:n_roots-1)=sort(com_rootsIS(2,1:n_roots-1),'descend');
```

```
clear eqn roots2
V2 = alpha(1,1)*z_Feed(2,1)/(alpha(1,1)-com_rootsIS(2,2))+alpha(1,2)*z_Feed
    ↪ (2,2)/(alpha(1,2)-com_rootsIS(2,2));

%% Column C3
z_Feed_C3 =0;
for k=1:n_comp-2
z_Feed_C3 = z_Feed_C3 + z_Feed(1,k);
end
z_Feed(3,1:end-2)=z_Feed(1,1:end-2)./z_Feed_C3;
Feed(3,1) = z_Feed_C3*F;

eqn = 0;
for j = 1:n_comp-2
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(3,j)/(alpha(1,j)-roots2);
end
com_rootsIS(3,1:n_roots-2) = vpasolve(eqn,roots2,[alpha(1,1) alpha(1,n_comp-2)
    ↪ ]);
clear eqn roots
V3 = alpha(1,1)*z_Feed(3,1)/(alpha(1,1)-com_rootsIS(3,1));

Vmin_IS= V1*Feed(1,1) + V2*Feed(2,1) + V3*Feed(3,1);
end
end
```

For a sequence of two direct splits the following code is called.

```
function [Vmin_DS_DS] = DirectIndirectSplit(z_Feed, alpha, VF, com_roots,F)
n_comp = 4;

V1 = VF(1,2);

Feed=ones(3,1);
Feed(1,1)=F;
n_roots = n_comp-1;
com_rootsDSIS = ones(3,n_roots); %vector for common roots
com_rootsDSIS(1,:) = com_roots(1,1:3);

%% Column C2
z_Feed_C2 =0;
for k=2:n_comp
z_Feed_C2 = z_Feed_C2 + z_Feed(1,k);
end
```

```
z_Feed(2,2:end)=z_Feed(1,2:end)./z_Feed_C2;
Feed(2,1) = z_Feed_C2*F;
eqn = 0;
%Sum of feed equation
for j = 2:n_comp
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(2,j)/(alpha(1,j)-roots2);
end
%solving feed equation (numerically)
com_rootsDSIS(2,2:n_roots) = vpasolve(eqn,roots2,[alpha(1,2) alpha(1,n_comp)]);
com_rootsDSIS(2,2:n_roots)=sort(com_rootsDSIS(2,2:n_roots),'descend');
clear eqn roots2
V2 = alpha(1,2)*z_Feed(2,2)/(alpha(1,2)-com_rootsDSIS(2,3))+alpha(1,3)*z_Feed
    ↪ (2,3)/(alpha(1,3)-com_rootsDSIS(2,3));

%% Column C3
z_Feed_C3 =0;
for k=2:n_comp-1
z_Feed_C3 = z_Feed_C3 + z_Feed(1,k);
end
z_Feed(3,2:end-1)=z_Feed(1,2:end-1)./z_Feed_C3;
Feed(3,1) = z_Feed_C3*F;
%calculating active roots
eqn = 0;
%Sum of feed equation
for j = 2:n_comp-1
syms roots2;
eqn = eqn + alpha(1,j)*z_Feed(3,j)/(alpha(1,j)-roots2);
end
com_rootsDSIS(3,2) = vpasolve(eqn,roots2,[alpha(1,2) alpha(1,3)]);
clear eqn roots
V3 = alpha(1,2)*z_Feed(3,2)/(alpha(1,2)-com_rootsDSIS(3,2));

Vmin_DS_DS= V1*Feed(1,1) + V2*Feed(2,1) + V3*Feed(3,1);

end
```

For sequences indirect-direct splits another code is called.

```
function [Vmin_ISDS] = IndirectDirectSplit(z_Feed, alpha, VF, com_roots, F)

n_comp = 4;
Feed=ones(3,1);
Feed(1,1)=F;
```

```
n_roots = n_comp−1;
com_rootsIS = ones(3,n_roots); %vector for common roots
com_rootsIS(1,:) = com_roots(1,1:3);

V1 = VF(1,7);
%% Column C2 feed eq
z_Feed_C2 =0;
for k=1:n_comp−1
z_Feed_C2 = z_Feed_C2 + z_Feed(1,k);
end
z_Feed(2,1:end−1)=z_Feed(1,1:end−1)./z_Feed_C2;
Feed(2,1) = z_Feed_C2∗F;
eqn = 0;
for j = 1:n_comp−1
syms roots2;
eqn = eqn + alpha(1,j)∗z_Feed(2,j)/(alpha(1,j)−roots2);
end

com_rootsIS(2,1:n_roots−1) = vpasolve(eqn,roots2,[alpha(1,1) alpha(1,n_comp−1)
    ↪ ]);
com_rootsIS(2,1:n_roots−1)=sort(com_rootsIS(2,1:n_roots−1),'descend');
clear eqn roots2
V2 = alpha(1,1)∗z_Feed(2,1)/(alpha(1,1)−com_rootsIS(2,1));

%% Column C3
z_Feed_C3 =0;
for k=2:3
z_Feed_C3 = z_Feed_C3 + z_Feed(1,k);
end
z_Feed(3,2:3)=z_Feed(1,2:3)./z_Feed_C3;
Feed(3,1) = z_Feed_C3∗F;

eqn = 0;
for j = 2:3
syms roots2;
eqn = eqn + alpha(1,j)∗z_Feed(3,j)/(alpha(1,j)−roots2);
end
com_rootsIS(3,2) = vpasolve(eqn,roots2,[alpha(1,2) alpha(1,3)]);
clear eqn roots
V3 = alpha(1,2)∗z_Feed(3,2)/(alpha(1,2)−com_rootsIS(3,2));

Vmin_ISDS= V1∗Feed(1,1) + V2∗Feed(2,1) + V3∗Feed(3,1);
end
```

3 Interface Excel - Aspen Plus (VBA)

In the following chapter the code used for the interface between Excel ans Aspen Plus is presented. The user interface is always organized according to the same template of the Workbook "FlowsheetAccess", which is specifically designed for every column version. All versions of the worksheet are presented in the following sections. However, some parts of the worksheet are similar in any case.
There are five buttons to click on, the macros connected to the buttons are summarized in Table 3.1. The macro connected to the buttons "Load Aspen RunID" and "Push variables to Aspen" are specific for the column (sequence) in the Aspen Plus file. The four versions considered are dividing wall column (DWC), simplified multiple dividing wall column (mDWC) and two direct split sequences (DSS-DSS) for the conventional quaternary split. Simpler column sequences were developed analogue to the latter example. In Section 3.1 the column specific worksheet "FlowsheetAccess" and codes are

Table 3.1: Functions called by click on the buttons.

Button name	Assigned Macro	see section		
		DWC	mDWC	DSS-DSS
Load Aspen RunID	GetSimulationVariables	3.1.1.1	3.1.2.1	3.1.3.1
Unload Aspen RunID	UnloadAspenPlusRunID		3.2.2	
Push Variables to Aspen	SendVariablesToAspen	3.1.1.2	3.1.2.2	3.1.3.2
Run Simulation	RunSimulation		3.2.4	
Save Simulation	SaveAspenPlusSimulation		3.2.5	

presented. Afterwards, general subs that are similar independent of the used column are summarized in Section 3.2.

3.1 Column specific interfaces

In this section the procedures used to read and send data between Aspen Plus and Excel. Since the required data depend on the column model, the following sections show the code for the simple dividing wall column (Section 3.1.1), the multiple dividing wall column (Section 3.1.2) and column sequences (Section 3.1.3).

3.1.1 Simple dividing wall column

The internal and external flows in Aspen Plus are named as shown in Figure 3.1. Figure 3.2 shows the Worksheet "FlowsheedAccess" in Excel applied as user interface for a simple dividing wall column. In the code of the simple dividing wall column presented

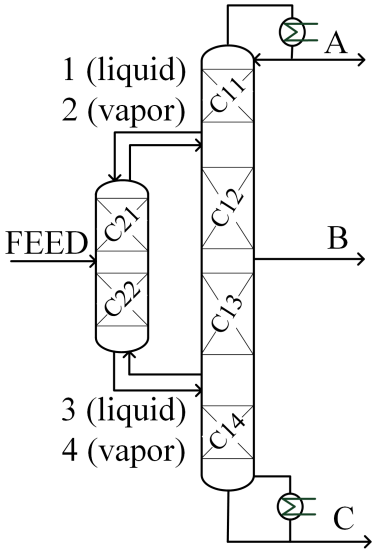

Figure 3.1: Names of flows used for simple dividing wall column in Aspen Plus simulation

Table 3.2: Parameters used for ternary systems

System	*hest*	$TC1top$ [°C]	$TC1bot$ [°C]
3.1	10.3	80	138
3.2	9.61	68	126
3.3	12.07	80	138
3.4	7.123	-1	36

in the following sections some system specific variables have to be inserted. These are the estimated enthalpy of the bottom product *hest*, the temperature at the top of the main column $TC1top$ and the one at the bottom $TC1bot$ which are summarized in Table 3.2 for the ternary systems. The enthalpy is used to estimate the internal vapor and liquid split fractions since it is used to calculate the bottom vapor flow from the reboiler duty. The enthalpy is read from an initial converging run in which the bottom product purity is above 98 mol %. The temperatures are required for the estimated temperature profile which is an input variable for the *Petlyuk* model. For the temperature the boiling points of the low and high boiling component is assumed. Note that also the names of the components had to be adjusted depending on the components name in the Aspen Plus simulation.

Load Aspen RunID | Unload Aspen RunID | Push variables to Aspen | Run simulation | Save Simulation

Current simulation:

\\cifs.uni-ulm.de\data\cheming\uwp20\03 Projekte\01 Fraunhofer\01_Optimization\03_3 components\01_BTX\04_TWK\ErrorHandlerV02\NQ\D1_S095_fix\95B\Petlyuk_BTX.bkp

Inputs

Block	Variable	Strom	Wert	Einheit
Directly read values				
Prefrac C2	N_{C2}		12	
	Feed stage	F	6	
Main C1	N_{C1}		28	
	Coupling Stage top		7	
	Side stream stage	B	13	
	Coupling Stage bot		21	
	Distillate Stream Flow	A	1	kmol/h
	Side Stream Flow	B	0.95	kmol/h
	Reboiler Duty		40.0	kW
Liquid Split	$V_{C2\ to\ C1}$		2.5	kmol/h
	$L_{C1\ to\ C2}$		1.1	kmol/h
	Vapor Split Pseudo		0.4	
	Liquid Split Pseudo		0.6	
Resulting configuration				
	N_{C11}		7	
	N_{C21}		6	
	N_{C22}		6	
	N_{C12}		6	
	N_{C13}		8	
	N_{C14}		7	

Outputs

Product stream composition

	Flow [kmol/h]	x_{Benzol}	x_{Toluol}	x_{pXylol}
A	1	98.04%	1.96%	0.00%
B	0.95	2.07%	95.55%	2.38%
C	1.05	0.00%	6.92%	93.08%

Internal stream outputs

	Stream to split [kmol/h]	Stream right [kmol/h]	Fraction right
Vapor Split Real	4.11	1.6587	0.40
Liquid Split Real	3.28	2.1738	0.66

Evaporated Vapor 4.0 kmol/h

	Flow [mol/h]	x_{Benzol}	x_{Toluol}	x_{pXylol}
1	1.1	34.76%	60.68%	4.56%
2	2.5	57.68%	41.08%	1.23%
3	4.0	1.28%	58.73%	39.99%
4	2.5	3.53%	73.72%	22.75%

Resulting split streams [kmol/h]

	AC	AB	BC
F	3.0	1.4	1.6
V	2.5	1.7	1.2
L	1.1	3.5	4.0

Run Status	Error message	Error Count
9345	No Errors	0
Success		
9345		
Success		

Figure 3.2: Layout of the Interface in Excel for simple DWC: Worksheet "FlowsheetAccess"

3.1.1.1 Read Aspen Plus Variables

For the sub "CheckSimulationConvergence" see section 3.2.6.

```
Sub GetSimulationVariables()
If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot␣access␣simulation␣Variables" & Chr(13) & _
"because␣no␣Aspen␣Plus␣file␣is␣loaded." & Chr(13) & _
"Load␣a␣RunID␣first.")
End
End If

Dim firstTime As Boolean: firstTime = True

CheckSimulationConvergence

PullVariables:
On Error GoTo ErrorHandler

Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim Root As IHNode: Set Root = AspenPlusRunID.Tree
Dim Streams As IHNode: Set Streams = Root.Data.Streams
Dim UnitOps As IHNode: Set UnitOps = Root.Data.Blocks
Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
Dim count As Integer: count = 0

If SimulationErrorCount = 11 Then
GoTo ErrorHandler

Else

Dim ComponentA As String: ComponentA = "BENZENE" 'Adapt according to
    ↪ system
Dim ComponentB As String: ComponentB = "TOLUENE" 'Adapt according to
    ↪ system
Dim ComponentC As String: ComponentC = "XYLENE" 'Adapt according to
    ↪ system
Dim hest As Double: hest = 10.3 'Adapt according to system, also in push

' ------- Inputs --------
Dim NC1 As Integer: NC1 = UnitOps.B1.Subobjects.Columns.Elements("1").Input
    ↪ .Elements("COLSP_NSTAGE").Elements("1").Value
Dim NC2 As Integer: NC2 = UnitOps.B1.Subobjects.Columns.Elements("2").Input
    ↪ .Elements("COLSP_NSTAGE").Elements("2").Value
```

```
Dim Nf As Integer: Nf = UnitOps.B1.Input.FEED_STAGE.FEED.Value
Dim Ns As Integer: Ns = UnitOps.B1.Input.PROD_STAGE.B.Value
Dim NC1top As Integer: NC1top = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("1").Input.Elements("SOURCE_STAGE").Elements("1").
    ↪ Value
Dim NC1bot As Integer: NC1bot = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("3").Input.Elements("DEST_STAGE").Elements("3").
    ↪ Value
Dim VapToC2 As Double: VapToC2 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("4").Input.Elements("BASIS_CSFLOW").Elements("4").
    ↪ Value
Dim LiqToC2 As Double: LiqToC2 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("1").Input.Elements("BASIS_CSFLOW").Elements("1").
    ↪ Value
Dim Q As Double: Q = UnitOps.B1.Subobjects.Columns.Elements("1").Input.
    ↪ Elements("QN").Elements("1").Value
Dim Dist As Double: Dist = UnitOps.B1.Subobjects.Columns.Elements("1").Input.
    ↪ Elements("BASIS_D").Elements("1").Value

' Transform splits
Dim VapToSplit As Double: VapToSplit = Q / hest 'estimated,
Dim VapSplitPseudo As Double: VapSplitPseudo = (VapToSplit − VapToC2) /
    ↪ VapToSplit
Dim LiqSplitPseudo As Double: LiqSplitPseudo = (VapToSplit − Dist − LiqToC2)
    ↪ / (VapToSplit − Dist)

' Transform stage setup
Dim NC11 As Integer: NC11 = NC1top
Dim NC21 As Integer: NC21 = Nf
Dim NC22 As Integer: NC22 = NC2 − Nf
Dim NC12 As Integer: NC12 = Ns − NC1top
Dim NC13 As Integer: NC13 = NC1bot − Ns
Dim NC14 As Integer: NC14 = NC1 − NC1bot

' Column specs
fs.Cells(9, 4) = NC2
fs.Cells(11, 4) = NC1
fs.Cells(15, 4) = Dist
fs.Cells(17, 4) = Q


' Inlet outlet specs
fs.Cells(10, 4) = Nf
fs.Cells(13, 4) = Ns
```

```
fs.Cells(16, 4) = UnitOps.B1.Input.Elements("PROD_FLOW").Elements("B").Value
    ↪ 'Seitenabzug

' Connecting streams specs
fs.Cells(12, 4) = NC1top 'Flüssigsplit und Dampfmix Stufe in top C1 zu top C2
fs.Cells(14, 4) = NC1bot 'Flüssigmix und Dampfsplit Stufe in bot C1 zu bot C2
fs.Cells(18, 4) = VapToC2 'Wert Dampfstrom zu bot C2
fs.Cells(19, 4) = LiqToC2 ' Wert Flüssigstrom zu top C2
fs.Cells(20, 4) = VapSplitPseudo
fs.Cells(21, 4) = LiqSplitPseudo



' Resulting column stages in each section
fs.Cells(24, 4).Value = NC11
fs.Cells(25, 4).Value = NC21
fs.Cells(26, 4).Value = NC22
fs.Cells(27, 4).Value = NC12
fs.Cells(28, 4).Value = NC13
fs.Cells(29, 4).Value = NC14

'------- Outputs ----------
' Splits

Dim NtopStage As String: NtopStage = NC1top
Dim NbotStage As String: NbotStage = NC1bot

Dim VapTotal As Double: VapTotal = UnitOps.B1.Subobjects.Columns.Elements("1
    ↪ ").Output.Elements("VAP_FLOW").Elements("1").Elements(CStr(NC1bot)).
    ↪ Value ' UnitOps.FindNode("\B1\Subobjects\Columns\1\Output\
    ↪ VAP_FLOW\1\N1top").Value
Dim VaporSplit As Double: VaporSplit = (VapTotal - VapToC2) / VapTotal
Dim LiqTotal As Double: LiqTotal = UnitOps.B1.Subobjects.Columns.Elements("1")
    ↪ .Output.Elements("LIQ_FLOW2").Elements("1").Elements(CStr(NC1top)).
    ↪ Value
Dim LiqSplit As Double: LiqSplit = (LiqTotal - LiqToC2) / LiqTotal

' Flows
fs.Cells(10, 8) = Dist
fs.Cells(11, 8) = UnitOps.B1.Input.Elements("PROD_FLOW").Elements("B").Value

' Product purities
fs.Cells(10, 9) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
```

```
fs.Cells(10, 10) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(10, 11) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(11, 9) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(11, 10) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(11, 11) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(12, 9) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(12, 10) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(12, 11) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value


' Internal streams
fs.Cells(20, 8) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.Elements("
    ↪ VAP_FLOW").Elements("1").Elements(CStr(NC1)).Value

fs.Cells(17, 8) = VapTotal
fs.Cells(18, 8) = LiqTotal
fs.Cells(17, 10) = VaporSplit
fs.Cells(18, 10) = LiqSplit

' Connecting streams
fs.Cells(23, 8) = LiqToC2
fs.Cells(24, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("2").
    ↪ Output.Elements("VAP_FLOW").Elements("2").Elements("1").Value
fs.Cells(25, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("3").
    ↪ Output.Elements("LIQ_FLOW2").Elements("2").Elements(CStr(NC2)).Value
fs.Cells(26, 8) = VapToC2

fs.Cells(23, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("X
    ↪ ").Elements("2").Elements("1").Elements(ComponentA).Value
fs.Cells(23, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements("1").Elements(ComponentB).Value
fs.Cells(23, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements("1").Elements(ComponentC).Value

fs.Cells(24, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("Y
    ↪ ").Elements("2").Elements("1").Elements(ComponentA).Value
```

```
fs.Cells(24, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements("1").Elements(ComponentB).Value
fs.Cells(24, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements("1").Elements(ComponentC).Value

fs.Cells(25, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("X
    ↪ ").Elements("2").Elements(CStr(NC2)).Elements(ComponentA).Value
fs.Cells(25, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements(CStr(NC2)).Elements(ComponentB).Value
fs.Cells(25, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements(CStr(NC2)).Elements(ComponentC).Value

fs.Cells(26, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("Y
    ↪ ").Elements("2").Elements(CStr(NC2)).Elements(ComponentA).Value
fs.Cells(26, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements(CStr(NC2)).Elements(ComponentB).Value
fs.Cells(26, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements(CStr(NC2)).Elements(ComponentC).Value


' Flows for Vmin diagrams
fs.Cells(31, 8) = Streams.FEED.Input.TOTFLOW.MIXED.Value
fs.Cells(32, 9) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ VAP_FLOW.Elements("1").Elements(CStr(Ns)).Value
fs.Cells(33, 9) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ LIQ_FLOW2.Elements("1").Elements("2").Value
fs.Cells(32, 10) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ LIQ_FLOW2.Elements("1").Elements(CStr(Ns + 1)).Value


If Worksheets("FlowsheetAccess").Cells(13, 2).Value = 0# Then
If firstTime Then
firstTime = False
'FailSave
'GoTo PullVariables
End If
End If
End If
Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
'MsgBox "Error in GetSimulationVariables"
```

```
fs.Cells(7, 14).Value = "Error in GetSimulationVariables"
If SimulationErrorCount = 11 Then
fs.Cells(8, 14).Value = "Data was not updated, too many fails!"
SimulationErrorCount = 0
End If
ReloadAspenPlusRunIDWithoutVariables
Exit Sub
End Sub
```

3.1.1.2 Send variables to Aspen Plus

The following code is used to send variables back to Aspen Plus.

```
Sub SendVariablesToAspen()
If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot access simulation Variables" & Chr(13) & _
"because no Aspen Plus file is loaded." & Chr(13) & _
"Load a RunID first.")
End
End If

Dim fs As Worksheet
Dim Root As IHNode
Dim Streams As IHNode
Dim UnitOps As IHNode
Dim Estimate1 As IHNode
Dim Estimate2 As IHNode

PushVariables:
On Error GoTo ErrorHandler

Set fs = Worksheets("FlowsheetAccess")
Set Root = AspenPlusRunID.Tree
Set Streams = Root.Data.Streams
Set UnitOps = Root.Data.Blocks
Set Estimate1 = Root.Data.Blocks.Elements("B1").Subobjects.Columns.Elements("1"
    ↪ ).Input.Elements("TEMP_EST").Elements(Format$(1, "0"))
Set Estimate2 = Root.Data.Blocks.Elements("B1").Subobjects.Columns.Elements("2"
    ↪ ).Input.Elements("TEMP_EST").Elements(Format$(2, "0"))

'−−−−−−−− Transform variables to Aspen Plus notation −−−−−−−−−
' Read new inputs from Excel workbook
Dim TC1top As Double: TC1top = 80 'Adapt according to system
Dim TC1bot As Double: TC1bot = 138 'Adapt according to system
```

```
Dim hest As Double: hest = 10.3 'Adapt according to system
Dim NC11 As Integer: NC11 = fs.Cells(24, 4).Value
Dim NC21 As Integer: NC21 = fs.Cells(25, 4).Value
Dim NC22 As Integer: NC22 = fs.Cells(26, 4).Value
Dim NC12 As Integer: NC12 = fs.Cells(27, 4).Value
Dim NC13 As Integer: NC13 = fs.Cells(28, 4).Value
Dim NC14 As Integer: NC14 = fs.Cells(29, 4).Value
Dim Qout As Double: Qout = fs.Cells(17, 4).Value
Dim Distout As Double: Distout = fs.Cells(15, 4).Value
Dim Sideout As Double: Sideout = fs.Cells(16, 4).Value
Dim VaporSplitPseudo As Double: VaporSplitPseudo = fs.Cells(20, 4).Value
Dim LiqSplitPseudo As Double: LiqSplitPseudo = fs.Cells(21, 4).Value

' Calculate resulting stages and splits
Dim NC2out As Integer: NC2out = NC21 + NC22
Dim NC1out As Integer: NC1out = NC11 + NC12 + NC13 + NC14
Dim NC1topout As Integer: NC1topout = NC11
Dim NC1sout As Integer: NC1sout = NC11 + NC12
Dim NC1botout As Integer: NC1botout = NC11 + NC12 + NC13
Dim VapToSplit As Double: VapToSplit = Qout / hest
Dim VapEst As Double: VapEst = VapToSplit * (1 - VaporSplitPseudo)
Dim LiqEst As Double: LiqEst = (VapToSplit - Distout) * (1 - LiqSplitPseudo)

fs.Cells(9, 4).Value = NC2out
fs.Cells(10, 4).Value = NC21
fs.Cells(11, 4).Value = NC1out
fs.Cells(12, 4).Value = NC1topout
fs.Cells(13, 4).Value = NC1sout
fs.Cells(14, 4).Value = NC1botout
fs.Cells(18, 4).Value = VapEst
fs.Cells(19, 4).Value = LiqEst

'------- Send variables to Aspen Plus -------
' Remove all stage imputs first (otherwise errors occur)...
UnitOps.FindNode("\B1\Subobjects\Columns\2\Input\COLSP_NSTAGE\2").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\COLSP_NSTAGE\1").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = Nothing
UnitOps.FindNode("\B1\Input\PROD_STAGE\B").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\3\Input\SOURCE_STAGE
    ↪ \3").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\4\Input\DEST_STAGE\4").
    ↪ Value = Nothing
```

```
UnitOps.FindNode("\B1\Subobjects\Connect Streams\1\Input\SOURCE_STAGE
    ↪ \1").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\2\Input\DEST_STAGE\2").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\DEST_STAGE\3").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\SOURCE_STAGE
    ↪ \4").Value = Nothing

Estimate1.RemoveAll
Estimate2.RemoveAll

' ...then specify new
' Column specs
UnitOps.FindNode("\B1\Subobjects\Columns\2\Input\COLSP_NSTAGE\2").
    ↪ Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\COLSP_NSTAGE\1").
    ↪ Value = NC1out
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\BASIS_D\1").Value =
    ↪ Distout
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\QN\1").Value = Qout

' Inlets outlet
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = NC21
UnitOps.FindNode("\B1\Input\PROD_STAGE\B").Value = NC1sout
UnitOps.FindNode("\B1\Input\PROD_FLOW\B").Value = Sideout
UnitOps.FindNode("\B1\Input\PROD_STAGE\C").Value = NC1out

' Connecting streams
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\SOURCE_STAGE
    ↪ \3").Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\DEST_STAGE\4").
    ↪ Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\1\Input\SOURCE_STAGE
    ↪ \1").Value = NC1topout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\2\Input\DEST_STAGE\2").
    ↪ Value = NC1topout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\DEST_STAGE\3").
    ↪ Value = NC1botout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\SOURCE_STAGE
    ↪ \4").Value = NC1botout


' Estimated vapor and liquid split
```

```
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\BASIS_CSFLOW
    ↪ \4").Value = VapEst
UnitOps.FindNode("\B1\Subobjects\Connect Streams\1\Input\BASIS_CSFLOW
    ↪ \1").Value = LiqEst


' Estimated temperature profile
Dim j As Integer
Dim k As Integer

Dim dTC1 As Double: dTC1 = (TC1bot − TC1top) / NC1out
Dim TC2top As Double: TC2top = TC1top + dTC1 ∗ NC11
Dim TC2bot As Double: TC2bot = TC1top + dTC1 ∗ (NC11 + NC12 + NC13)
Dim dTC2 As Double: dTC2 = (TC2bot − TC2top) / NC2out


'Column 1
For j = 1 To NC1out
Call Estimate1.Elements.InsertRow(0, 0)
Estimate1.Elements.Label(0, 0) = Format$(j, "0")
Estimate1.Elements.Item(0, 0).Value = Format$(TC1top + dTC1 ∗ (j − 1), "0")
Next j

'Column 2
For k = 1 To NC2out
Call Estimate2.Elements.InsertRow(0, 0)
Estimate2.Elements.Label(0, 0) = Format$(k, "0")
Estimate2.Elements.Item(0, 0).Value = CDbl(TC2top + dTC2 ∗ (k − 1))
Next k
Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
MsgBox "Error in SendVariablesToAspen"
ReloadAspenPlusRunIDWithoutVariables
fs.Cells(8, 14) = "Error in Send Variables"
fs.Cells(8, 16) = AspenErrorCounter
Resume PushVariables

End Sub
```

3.1.2 Simplified multiple dividing wall column

The internal and external flows required for the simplified multiple dividing wall column model in Aspen Plus are named as shown in Figure 3.3. Figure 3.4 shows the Worksheet

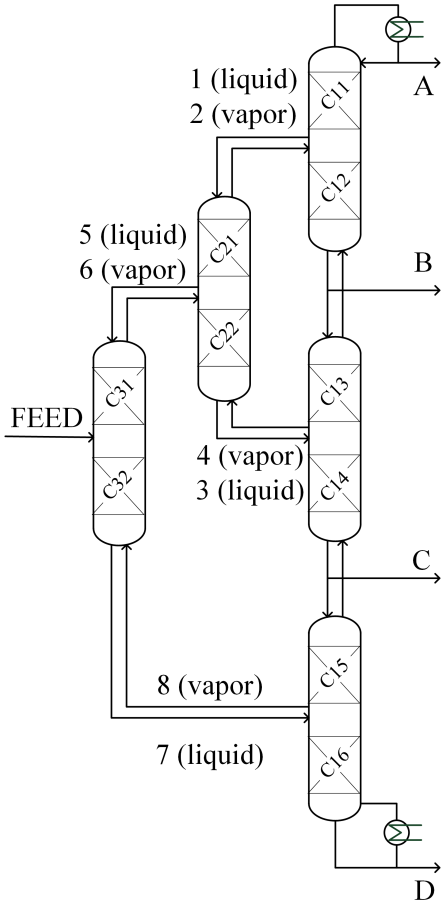

Figure 3.3: Names of flows used for simplified multiple dividing wall column in Aspen Plus simulation

"FlowsheetAccess" from the Excel user interface. Similar as for the simple dividing wall column, the bottom product enthalpy and the temperature at top and bottom of the column have to be specified according to the system. These are summarized in Table 3.3 for the quaternary systems.

Table 3.3: Parameters used for ternary systems

System	*hest*	$TC1top$ [°C]	$TC1bot$ [°C]
4.1	85.2708	78	118
4.2	85.2708	58	118
4.3	84.023	75	118
4.4	84.023	58	118
4.5	86.31	58	108
4.6	96.93	80	152

Load Aspen RunID | Unload Aspen RunID | Push variables to Aspen | Run simulation | Save simulation

Current simulation:

\\cifs.uni-ulm.de\data\cheming\uwp20\03 Projekte\01 Fraunhofer\01_Optimization\04_4 components\02_Uli\mDWC\03_NQ_98\AllVar\Uli_initial6.bkp

Inputs

Block	Variable	Strom	Wert	Einheit
Directly read values				
Prefrac 1 C3	N_{C3}		26	
	Feed stage	F	11	
Prefrac2 C2	N_{C2}		19	
	Feed stage	6	15	
Main C1	N_{C1}		82	
	Coupling Stage top		10	
	Side stream 1 stage	B	23	
	Coupling Stage mid		38	
	Side stream 2 stage	C	53	
	Coupling Stage bot		68	
	Distillate Stream Flow	A	25.03	mol/h
	Side Stream 1 Flow	B	25.12	mol/h
	Side Stream 2 Flow	C	24.55	mol/h
	Reboiler Duty		5.33	kW
Vapor Stream C1C3	$V_{C1\ to\ C3}$		328.64	mol/h
Vapor Stream C1C2	$V_{C1\ to\ C2}$		34.50	mol/h
Liquid Stream C1C2	$L_{C1\ to\ C2}$		327.13	mol/h
Liquid Stream C2C3	$L_{C2\ to\ C3}$		256.26	mol/h
Vapor Split C1C3 Pseudo			0.28	
Vapor Split C1C2 Pseudo			0.73	
Liquid Split C1C2 Pseudo			0.24	
Liquid Split C2C3 Pseudo			0.22	
Resulting configuration				
	N_{C11}		10	
	N_{C21}		15	
	N_{C22}		4	
	N_{C12}		13	
	N_{C13}		15	
	N_{C31}		11	
	N_{C32}		15	
	N_{C14}		15	
	N_{C15}		15	
	N_{C16}		14	

Outputs

Product stream composition

	Flow [mol/h]	$x_{Ethanol}$	$x_{Propanol}$	$x_{Isobutanol}$	$x_{Butanol}$
A	25.03	99.56%	0.44%	0.00%	0.00%
B	25.12	0.30%	97.91%	1.79%	0.00%
C	24.55	0.00%	1.20%	97.89%	0.91%
D	25.29	0.00%	0.00%	2.04%	97.96%

Internal stream outputs

	Stream to split [mol/h]	Stream right [mol/h]	Fraction right
Vapor Split C1C3 Real	475.68	147.0465	0.31
Vapor Split C1C2 Real	143.74	109.2342	0.76
Liquid Split C1C2 Real	468.78	141.6488	0.30
Liquid Split C2C3 Real	307.83	51.5693	0.17

Evaporated Vapor	447.8184	kmol/h

	Flow [mol/h]	$x_{Ethanol}$	$x_{Propanol}$	$x_{Isobutanol}$	$x_{Butanol}$
1	327.13	29.54%	68.12%	2.34%	0.00%
2	375.74	47.12%	51.72%	1.15%	0.00%
3	51.42	0.48%	39.22%	60.05%	0.24%
4	34.50	1.10%	48.27%	50.51%	0.12%
5	256.26	5.27%	25.49%	68.70%	0.54%
6	321.79	11.89%	31.08%	56.75%	0.28%
7	363.10	0.00%	0.61%	63.80%	35.59%
8	328.64	0.00%	0.80%	71.03%	28.17%

Resulting split streams [mol/h]

	BD	AC	AB	BC	CD
F	100.00	65.53	48.62	16.92	34.46
V	328.64	34.50	109.12	143.18	447.82
L	256.26	327.13	465.81	107.53	134.12

V/F	0	3.29	3.76	4.85	4.85	4.48	0
D/F	0	0.66	0.49	0.25	0.50	0.75	1

Run Status	Error message	Error Count
9345	No Errors	0
Success		
9345		
Success		

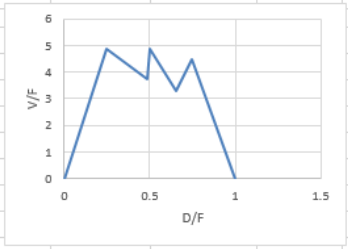

Figure 3.4: Layout of the Interface in Excel for simplified mDWC: Worksheet "FlowsheetAccess"

3.1.2.1 Read Aspen Plus Variables

For the sub "CheckSimulationConvergence" see section 3.2.6. The code is valid for the parameter of system 4.1.

```
Sub GetSimulationVariables()

If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot␣access␣simulation␣Variables" & Chr(13) & _
"because␣no␣Aspen␣Plus␣file␣is␣loaded." & Chr(13) & _
"Load␣a␣RunID␣first.")
End
End If

Dim firstTime As Boolean: firstTime = True

CheckSimulationConvergence

PullVariables:
On Error GoTo ErrorHandler

Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim Root As IHNode: Set Root = AspenPlusRunID.Tree
Dim Streams As IHNode: Set Streams = Root.Data.Streams
Dim UnitOps As IHNode: Set UnitOps = Root.Data.Blocks
Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪  1).Value
Dim count As Integer: count = 0

If SimulationErrorCount = 11 Then
GoTo ErrorHandler
Else
Dim ComponentA As String: ComponentA = "ETHANOL" 'Adapt according to
    ↪ system
Dim ComponentB As String: ComponentB = "PROPANOL" 'Adapt according to
    ↪ system
Dim ComponentC As String: ComponentC = "ISOBUTAN" 'Adapt according to
    ↪ system
Dim ComponentD As String: ComponentD = "NBUTAN" 'Adapt according to
    ↪ system
Dim hest As Double: hest = 85.2708 'Adapt according to system, also in push

' −−−−−−−− Inputs −−−−−−−−−
Dim NC1 As Integer: NC1 = UnitOps.B1.Subobjects.Columns.Elements("1").Input
    ↪ .Elements("COLSP_NSTAGE").Elements("1").Value
```

```
Dim NC2 As Integer: NC2 = UnitOps.B1.Subobjects.Columns.Elements("2").Input
    ↪ .Elements("COLSP_NSTAGE").Elements("2").Value
Dim NC3 As Integer: NC3 = UnitOps.B1.Subobjects.Columns.Elements("3").Input
    ↪ .Elements("COLSP_NSTAGE").Elements("3").Value
Dim NfC3 As Integer: NfC3 = UnitOps.B1.Input.FEED_STAGE.FEED.Value
Dim NfC2 As Integer: NfC2 = UnitOps.B1.Subobjects.Elements("Connect␣Streams
    ↪ ").Elements("6").Input.DEST_STAGE.Elements("6").Value
Dim Ns1 As Integer: Ns1 = UnitOps.B1.Input.PROD_STAGE.B.Value
Dim Ns2 As Integer: Ns2 = UnitOps.B1.Input.PROD_STAGE.C.Value
Dim NC1top As Integer: NC1top = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("2").Input.DEST_STAGE.Elements("2").Value 'identisch
    ↪  UnitOps.B1.Subobjects.Connect Streams.2.Input.DEST_STAGE.2.Value '
    ↪ Dampfmix Stufe aus top C2 in C1
Dim NC1mid As Integer: NC1mid = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("3").Input.DEST_STAGE.Elements("3").Value
Dim NC1bot As Integer: NC1bot = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("7").Input.DEST_STAGE.Elements("7").Value 'identisch
    ↪  UnitOps.B1.Subobjects.Connect Streams.4.Input.SOURCE_STAGE.4.Value
Dim LiqToC2 As Double: LiqToC2 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("1").Input.Elements("BASIS_CSFLOW").Elements("1").
    ↪ Value
Dim VapToC2 As Double: VapToC2 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("4").Input.Elements("BASIS_CSFLOW").Elements("4").
    ↪ Value
Dim LiqToC3 As Double: LiqToC3 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("5").Input.Elements("BASIS_CSFLOW").Elements("5").
    ↪ Value
Dim VapToC3 As Double: VapToC3 = UnitOps.B1.Subobjects.Elements("Connect␣
    ↪ Streams").Elements("8").Input.Elements("BASIS_CSFLOW").Elements("8").
    ↪ Value
Dim DistStream As Double: DistStream = UnitOps.B1.Subobjects.Columns.
    ↪ Elements("1").Input.Elements("BASIS_D").Elements("1").Value
Dim SideStream1 As Double: SideStream1 = UnitOps.B1.Input.Elements("
    ↪ PROD_FLOW").Elements("B").Value
Dim SideStream2 As Double: SideStream2 = UnitOps.B1.Input.Elements("
    ↪ PROD_FLOW").Elements("C").Value
Dim Q As Double: Q = UnitOps.B1.Subobjects.Columns.Elements("1").Input.
    ↪ Elements("QN").Elements("1").Value

' Transform splits
Dim VapC1C3ToSplit As Double: VapC1C3ToSplit = Q * hest 'estimated
Dim VapSplitC1C3Pseudo As Double: VapSplitC1C3Pseudo = (VapC1C3ToSplit −
    ↪ VapToC3) / VapC1C3ToSplit
Dim VapC1C2ToSplit As Double: VapC1C2ToSplit = VapC1C3ToSplit − VapToC3
```

```
Dim VapSplitC1C2Pseudo As Double: VapSplitC1C2Pseudo = (VapC1C2ToSplit −
    ↪ VapToC2) / VapC1C2ToSplit

Dim LiqC1C2ToSplit As Double: LiqC1C2ToSplit = (VapC1C3ToSplit −
    ↪ DistStream)
Dim LiqSplitC1C2Pseudo As Double: LiqSplitC1C2Pseudo = (LiqC1C2ToSplit −
    ↪ LiqToC2) / LiqC1C2ToSplit
Dim LiqC2C3ToSplit As Double: LiqC2C3ToSplit = LiqToC2
Dim LiqSplitC2C3Pseudo As Double: LiqSplitC2C3Pseudo = (LiqC2C3ToSplit −
    ↪ LiqToC3) / LiqC2C3ToSplit

' Transform stage setup
Dim NC11 As Integer: NC11 = NC1top
Dim NC21 As Integer: NC21 = NfC2
Dim NC22 As Integer: NC22 = NC2 − NfC2
Dim NC12 As Integer: NC12 = Ns1 − NC1top
Dim NC13 As Integer: NC13 = NC1mid − Ns1
Dim NC31 As Integer: NC31 = NfC3
Dim NC32 As Integer: NC32 = NC3 − NfC3
Dim NC14 As Integer: NC14 = Ns2 − NC1mid
Dim NC15 As Integer: NC15 = NC1bot − Ns2
Dim NC16 As Integer: NC16 = NC1 − NC1bot

fs.Cells(9, 4) = NC3
fs.Cells(10, 4) = NfC3
fs.Cells(11, 4) = NC2
fs.Cells(12, 4) = NfC2
fs.Cells(13, 4) = NC1
fs.Cells(14, 4) = NC1top
fs.Cells(15, 4) = Ns1
fs.Cells(16, 4) = NC1mid
fs.Cells(17, 4) = Ns2
fs.Cells(18, 4) = NC1bot
fs.Cells(19, 4) = DistStream
fs.Cells(20, 4) = SideStream1
fs.Cells(21, 4) = SideStream2
fs.Cells(22, 4) = Q
fs.Cells(23, 4) = VapToC3
fs.Cells(24, 4) = VapToC2
fs.Cells(25, 4) = LiqToC2
fs.Cells(26, 4) = LiqToC3
fs.Cells(27, 4) = VapSplitC1C3Pseudo
fs.Cells(28, 4) = VapSplitC1C2Pseudo
fs.Cells(29, 4) = LiqSplitC1C2Pseudo
```

```
fs.Cells(30, 4) = LiqSplitC2C3Pseudo

' Resulting column stages in each section
fs.Cells(33, 4).Value = NC11
fs.Cells(34, 4).Value = NC21
fs.Cells(35, 4).Value = NC22
fs.Cells(36, 4).Value = NC12
fs.Cells(37, 4).Value = NC13
fs.Cells(38, 4).Value = NC31
fs.Cells(39, 4).Value = NC32
fs.Cells(40, 4).Value = NC14
fs.Cells(41, 4).Value = NC15
fs.Cells(42, 4).Value = NC16

' -------- Outputs ---------

Dim VapTotalC16 As Double: VapTotalC16 = UnitOps.B1.Subobjects.Columns.
    ↪ Elements("1").Output.Elements("VAP_FLOW").Elements("1").Elements(
    ↪ CStr(NC1bot)).Value ' UnitOps.FindNode("\B1\Subobjects\Columns\1\
    ↪ Output\VAP_FLOW\1\N1top").Value 'UnitOps.B1.Subobjects.Columns.1.
    ↪ Output.VAP_FLOW.1.9
Dim VaporSplitC1C3 As Double: VaporSplitC1C3 = (VapTotalC16 - VapToC3) /
    ↪ VapTotalC16
Dim VapTotalC14 As Double: VapTotalC14 = UnitOps.B1.Subobjects.Columns.
    ↪ Elements("1").Output.Elements("VAP_FLOW").Elements("1").Elements(
    ↪ CStr(NC1mid)).Value ' UnitOps.FindNode("\B1\Subobjects\Columns\1\
    ↪ Output\VAP_FLOW\1\N1top").Value 'UnitOps.B1.Subobjects.Columns.1.
    ↪ Output.VAP_FLOW.1.9
Dim VaporSplitC1C2 As Double: VaporSplitC1C2 = (VapTotalC14 - VapToC2) /
    ↪ VapTotalC14

Dim LiqTotalC11 As Double: LiqTotalC11 = UnitOps.B1.Subobjects.Columns.
    ↪ Elements("1").Output.Elements("LIQ_FLOW2").Elements("1").Elements(
    ↪ CStr(NC1top)).Value ' UnitOps.FindNode("\B1\Subobjects\Columns\1\
    ↪ Output\LIQ_FLOW2\1\N1bot").Value
Dim LiqSplitC1C2 As Double: LiqSplitC1C2 = (LiqTotalC11 - LiqToC2) /
    ↪ LiqTotalC11
Dim LiqTotalC21 As Double: LiqTotalC21 = UnitOps.B1.Subobjects.Columns.
    ↪ Elements("2").Output.Elements("LIQ_FLOW2").Elements("2").Elements(
    ↪ CStr(NfC2)).Value ' UnitOps.FindNode("\B1\Subobjects\Columns\1\Output
    ↪ \LIQ_FLOW2\1\N1bot").Value
Dim LiqSplitC2C3 As Double: LiqSplitC2C3 = (LiqTotalC21 - LiqToC3) /
    ↪ LiqTotalC21
```

```
' Product purities
fs.Cells(10, 8) = UnitOps.B1.Subobjects.Columns.Elements("1").Input.BASIS_D.
    ↪ Elements("1").Value
fs.Cells(11, 8) = UnitOps.B1.Input.PROD_FLOW.B.Value
fs.Cells(12, 8) = UnitOps.B1.Input.PROD_FLOW.C.Value

fs.Cells(10, 9) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(10, 10) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(10, 11) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(10, 12) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(11, 9) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(11, 10) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(11, 11) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(11, 12) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(12, 9) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(12, 10) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(12, 11) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(12, 12) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(13, 9) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(13, 10) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(13, 11) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(13, 12) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value

' Real Splits
fs.Cells(23, 8) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ VAP_FLOW.Elements("1").Elements(CStr(NC1)).Value
```

```
fs.Cells(18, 8) = VapTotalC16
fs.Cells(19, 8) = VapTotalC14
fs.Cells(20, 8) = LiqTotalC11
fs.Cells(21, 8) = LiqTotalC21
fs.Cells(18, 10) = VaporSplitC1C3
fs.Cells(19, 10) = VaporSplitC1C2
fs.Cells(20, 10) = LiqSplitC1C2
fs.Cells(21, 10) = LiqSplitC2C3

' Connecting streams
fs.Cells(26, 8) = LiqToC2
fs.Cells(27, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("2").
    ↪ Output.Elements("VAP_FLOW").Elements("2").Elements("1").Value
fs.Cells(28, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("3").
    ↪ Output.Elements("LIQ_FLOW2").Elements("2").Elements(CStr(NC2)).Value
fs.Cells(29, 8) = VapToC2
fs.Cells(30, 8) = LiqToC3
fs.Cells(31, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("6").
    ↪ Output.Elements("VAP_FLOW").Elements("3").Elements("1").Value
fs.Cells(32, 8) = UnitOps.B1.Subobjects.Elements("Connect␣Streams").Elements("7").
    ↪ Output.Elements("LIQ_FLOW2").Elements("3").Elements(CStr(NC3)).Value
fs.Cells(33, 8) = VapToC3

fs.Cells(26, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("X
    ↪ ").Elements("2").Elements("1").Elements(ComponentA).Value
fs.Cells(26, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements("1").Elements(ComponentB).Value
fs.Cells(26, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements("1").Elements(ComponentC).Value
fs.Cells(26, 12) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements("1").Elements(ComponentD).Value

fs.Cells(27, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("Y
    ↪ ").Elements("2").Elements("1").Elements(ComponentA).Value
fs.Cells(27, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements("1").Elements(ComponentB).Value
fs.Cells(27, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements("1").Elements(ComponentC).Value
fs.Cells(27, 12) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements("1").Elements(ComponentD).Value

fs.Cells(28, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("X
    ↪ ").Elements("2").Elements(CStr(NC2)).Elements(ComponentA).Value
```

```
fs.Cells(28, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements(CStr(NC2)).Elements(ComponentB).Value
fs.Cells(28, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements(CStr(NC2)).Elements(ComponentC).Value
fs.Cells(28, 12) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ X").Elements("2").Elements(CStr(NC2)).Elements(ComponentD).Value

fs.Cells(29, 9) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("Y
    ↪ ").Elements("2").Elements(CStr(NC2)).Elements(ComponentA).Value
fs.Cells(29, 10) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements(CStr(NC2)).Elements(ComponentB).Value
fs.Cells(29, 11) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements(CStr(NC2)).Elements(ComponentC).Value
fs.Cells(29, 12) = UnitOps.B1.Subobjects.Columns.Elements("2").Output.Elements("
    ↪ Y").Elements("2").Elements(CStr(NC2)).Elements(ComponentD).Value

fs.Cells(30, 9) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("X
    ↪ ").Elements("3").Elements("1").Elements(ComponentA).Value
fs.Cells(30, 10) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements("1").Elements(ComponentB).Value
fs.Cells(30, 11) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements("1").Elements(ComponentC).Value
fs.Cells(30, 12) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements("1").Elements(ComponentD).Value

fs.Cells(31, 9) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("Y
    ↪ ").Elements("3").Elements("1").Elements(ComponentA).Value
fs.Cells(31, 10) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements("1").Elements(ComponentB).Value
fs.Cells(31, 11) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements("1").Elements(ComponentC).Value
fs.Cells(31, 12) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements("1").Elements(ComponentD).Value

fs.Cells(32, 9) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("X
    ↪ ").Elements("3").Elements(CStr(NC3)).Elements(ComponentA).Value
fs.Cells(32, 10) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements(CStr(NC3)).Elements(ComponentB).Value
fs.Cells(32, 11) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements(CStr(NC3)).Elements(ComponentC).Value
fs.Cells(32, 12) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ X").Elements("3").Elements(CStr(NC3)).Elements(ComponentD).Value
```

```
fs.Cells(33, 9) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("Y
    ↪ ").Elements("3").Elements(CStr(NC3)).Elements(ComponentA).Value
fs.Cells(33, 10) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements(CStr(NC3)).Elements(ComponentB).Value
fs.Cells(33, 11) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements(CStr(NC3)).Elements(ComponentC).Value
fs.Cells(33, 12) = UnitOps.B1.Subobjects.Columns.Elements("3").Output.Elements("
    ↪ Y").Elements("3").Elements(CStr(NC3)).Elements(ComponentD).Value

' Flows for Vmin diagrams
fs.Cells(38, 8) = Streams.FEED.Input.TOTFLOW.MIXED.Value
fs.Cells(39, 10) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ VAP_FLOW.Elements("1").Elements(CStr(Ns1)).Value
fs.Cells(40, 10) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ LIQ_FLOW2.Elements("1").Elements("2").Value
fs.Cells(39, 11) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ VAP_FLOW.Elements("1").Elements(CStr(Ns2)).Value
fs.Cells(40, 11) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ LIQ_FLOW2.Elements("1").Elements(CStr(Ns1 + 1)).Value
fs.Cells(40, 12) = UnitOps.B1.Subobjects.Columns.Elements("1").Output.
    ↪ LIQ_FLOW2.Elements("1").Elements(CStr(Ns2 + 1)).Value


If Worksheets("FlowsheetAccess").Cells(13, 2).Value = 0# Then
If firstTime Then
firstTime = False
'FailSave
'GoTo PullVariables
End If
End If

End If
Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
'MsgBox "Error in GetSimulationVariables"

fs.Cells(7, 15).Value = "Error in GetSimulationVariables"
If SimulationErrorCount = 11 Then
fs.Cells(8, 15).Value = "Data was not updated, too many fails!"
SimulationErrorCount = 0
End If
ReloadAspenPlusRunIDWithoutVariables
```

```
Exit Sub

End Sub
```

3.1.2.2 Send variables to Aspen Plus

The following code is used to send variables to Aspen Plus.

```
Sub SendVariablesToAspen()
If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot access simulation Variables" & Chr(13) & _
"because no Aspen Plus file is loaded." & Chr(13) & _
"Load a RunID first.")
End
End If

Dim fs As Worksheet
Dim Root As IHNode
Dim Streams As IHNode
Dim UnitOps As IHNode
Dim Estimate1 As IHNode
Dim Estimate2 As IHNode
Dim Estimate3 As IHNode

PushVariables:
On Error GoTo ErrorHandler

Set fs = Worksheets("FlowsheetAccess")
Set Root = AspenPlusRunID.Tree
Set Streams = Root.Data.Streams
Set UnitOps = Root.Data.Blocks
Set Estimate1 = Root.Data.Blocks.Elements("B1").Subobjects.Columns.Elements("1"
    ↪ ).Input.Elements("TEMP_EST").Elements(Format$(1, "0"))
Set Estimate2 = Root.Data.Blocks.Elements("B1").Subobjects.Columns.Elements("2"
    ↪ ).Input.Elements("TEMP_EST").Elements(Format$(2, "0")) 'hinter Format
    ↪ ist column number!
Set Estimate3 = Root.Data.Blocks.Elements("B1").Subobjects.Columns.Elements("3"
    ↪ ).Input.Elements("TEMP_EST").Elements(Format$(3, "0")) 'hinter Format
    ↪ ist column number!



'-------- Transform variables to Aspen Plus notation ----------
' Read new inputs from Excel workbook
Dim TC1top As Double: TC1top = 78 'Adapt according to system
```

```
Dim TC1bot As Double: TC1bot = 108 'Adapt according to system
Dim hest As Double: hest = 85.2708 'Adapt according to system
Dim NC11 As Integer: NC11 = fs.Cells(33, 4).Value
Dim NC21 As Integer: NC21 = fs.Cells(34, 4).Value
Dim NC22 As Integer: NC22 = fs.Cells(35, 4).Value
Dim NC12 As Integer: NC12 = fs.Cells(36, 4).Value
Dim NC13 As Integer: NC13 = fs.Cells(37, 4).Value
Dim NC31 As Integer: NC31 = fs.Cells(38, 4).Value
Dim NC32 As Integer: NC32 = fs.Cells(39, 4).Value
Dim NC14 As Integer: NC14 = fs.Cells(40, 4).Value
Dim NC15 As Integer: NC15 = fs.Cells(41, 4).Value
Dim NC16 As Integer: NC16 = fs.Cells(42, 4).Value

Dim Distout As Double: Distout = fs.Cells(19, 4).Value
Dim Side1out As Double: Side1out = fs.Cells(20, 4).Value
Dim Side2out As Double: Side2out = fs.Cells(21, 4).Value
Dim Qout As Double: Qout = fs.Cells(22, 4).Value

Dim VaporSplitC1C3Pseudo As Double: VaporSplitC1C3Pseudo = fs.Cells(27, 4).
    ↪ Value
Dim VaporSplitC1C2Pseudo As Double: VaporSplitC1C2Pseudo = fs.Cells(28, 4).
    ↪ Value
Dim LiqSplitC1C2Pseudo As Double: LiqSplitC1C2Pseudo = fs.Cells(29, 4).Value
Dim LiqSplitC2C3Pseudo As Double: LiqSplitC2C3Pseudo = fs.Cells(30, 4).Value

' Calculate resulting stages and splits
Dim NC3out As Integer: NC3out = NC31 + NC32
Dim NC2out As Integer: NC2out = NC21 + NC22
Dim NC1out As Integer: NC1out = NC11 + NC12 + NC13 + NC14 + NC15 +
    ↪ NC16
Dim NC1topout As Integer: NC1topout = NC11
Dim NC1s1out As Integer: NC1s1out = NC11 + NC12
Dim NC1midout As Integer: NC1midout = NC11 + NC12 + NC13
Dim NC1s2out As Integer: NC1s2out = NC11 + NC12 + NC13 + NC14
Dim NC1botout As Integer: NC1botout = NC11 + NC12 + NC13 + NC14 +
    ↪ NC15

Dim VapBotToSplit As Double: VapBotToSplit = Qout * hest 'estimated in mol/h
Dim VapMidToSplit As Double: VapMidToSplit = VapBotToSplit *
    ↪ VaporSplitC1C3Pseudo
Dim LiqTopToSplit As Double: LiqTopToSplit = (VapBotToSplit - Distout)
Dim LiqMidToSplit As Double: LiqMidToSplit = LiqTopToSplit * (1 -
    ↪ LiqSplitC1C2Pseudo)
```

```
Dim VapEstC1C3 As Double: VapEstC1C3 = VapBotToSplit * (1 -
    ↪ VaporSplitC1C3Pseudo)
Dim VapEstC1C2 As Double: VapEstC1C2 = VapMidToSplit * (1 -
    ↪ VaporSplitC1C2Pseudo)
Dim LiqEstC1C2 As Double: LiqEstC1C2 = LiqTopToSplit * (1 -
    ↪ LiqSplitC1C2Pseudo)
Dim LiqEstC2C3 As Double: LiqEstC2C3 = LiqEstC1C2 * (1 -
    ↪ LiqSplitC2C3Pseudo)

fs.Cells(9, 4).Value = NC3out
fs.Cells(10, 4).Value = NC31
fs.Cells(11, 4).Value = NC2out
fs.Cells(12, 4).Value = NC21
fs.Cells(13, 4).Value = NC1out
fs.Cells(14, 4).Value = NC1topout
fs.Cells(15, 4).Value = NC1s1out
fs.Cells(16, 4).Value = NC1midout
fs.Cells(17, 4).Value = NC1s2out
fs.Cells(18, 4).Value = NC1botout

fs.Cells(23, 4).Value = VapEstC1C3
fs.Cells(24, 4).Value = VapEstC1C2
fs.Cells(25, 4).Value = LiqEstC1C2
fs.Cells(26, 4).Value = LiqEstC2C3

'-------- Send variables to Aspen Plus --------
' Remove all stage imputs first (otherwise errors occur)...
UnitOps.FindNode("\B1\Subobjects\Columns\3\Input\COLSP_NSTAGE\3").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Columns\2\Input\COLSP_NSTAGE\2").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\COLSP_NSTAGE\1").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = Nothing
UnitOps.FindNode("\B1\Input\PROD_STAGE\B").Value = Nothing
UnitOps.FindNode("\B1\Input\PROD_STAGE\C").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\1\Input\SOURCE_STAGE
    ↪ \1").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\3\Input\SOURCE_STAGE
    ↪ \3").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\4\Input\SOURCE_STAGE
    ↪ \4").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect␣Streams\5\Input\SOURCE_STAGE
    ↪ \5").Value = Nothing
```

```
UnitOps.FindNode("\B1\Subobjects\Connect Streams\7\Input\SOURCE_STAGE
    ↪ \7").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\8\Input\SOURCE_STAGE
    ↪ \8").Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\2\Input\DEST_STAGE\2").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\DEST_STAGE\3").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\DEST_STAGE\4").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\6\Input\DEST_STAGE\6").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\7\Input\DEST_STAGE\7").
    ↪ Value = Nothing
UnitOps.FindNode("\B1\Subobjects\Connect Streams\8\Input\DEST_STAGE\8").
    ↪ Value = Nothing

Estimate1.RemoveAll
Estimate2.RemoveAll
Estimate3.RemoveAll

' ...then specify new
' Column specs
UnitOps.FindNode("\B1\Subobjects\Columns\3\Input\COLSP_NSTAGE\3").
    ↪ Value = NC3out
UnitOps.FindNode("\B1\Subobjects\Columns\2\Input\COLSP_NSTAGE\2").
    ↪ Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\COLSP_NSTAGE\1").
    ↪ Value = NC1out
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\BASIS_D\1").Value =
    ↪ Distout
UnitOps.FindNode("\B1\Subobjects\Columns\1\Input\QN\1").Value = Qout

' Inlets outlet
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = NC31
UnitOps.FindNode("\B1\Input\PROD_STAGE\B").Value = NC1s1out
UnitOps.FindNode("\B1\Input\PROD_FLOW\B").Value = Side1out
UnitOps.FindNode("\B1\Input\PROD_STAGE\C").Value = NC1s2out
UnitOps.FindNode("\B1\Input\PROD_FLOW\C").Value = Side2out
UnitOps.FindNode("\B1\Input\PROD_STAGE\D").Value = NC1out

' Connecting streams
UnitOps.FindNode("\B1\Subobjects\Connect Streams\1\Input\SOURCE_STAGE
    ↪ \1").Value = NC1topout
```

```
UnitOps.FindNode("\B1\Subobjects\Connect Streams\2\Input\DEST_STAGE\2").
    ↪ Value = NC1topout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\SOURCE_STAGE
    ↪ \3").Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\DEST_STAGE\4").
    ↪ Value = NC2out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\3\Input\DEST_STAGE\3").
    ↪ Value = NC1midout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\SOURCE_STAGE
    ↪ \4").Value = NC1midout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\5\Input\SOURCE_STAGE
    ↪ \5").Value = NC21
UnitOps.FindNode("\B1\Subobjects\Connect Streams\6\Input\DEST_STAGE\6").
    ↪ Value = NC21
UnitOps.FindNode("\B1\Subobjects\Connect Streams\7\Input\SOURCE_STAGE
    ↪ \7").Value = NC3out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\8\Input\DEST_STAGE\8").
    ↪ Value = NC3out
UnitOps.FindNode("\B1\Subobjects\Connect Streams\7\Input\DEST_STAGE\7").
    ↪ Value = NC1botout
UnitOps.FindNode("\B1\Subobjects\Connect Streams\8\Input\SOURCE_STAGE
    ↪ \8").Value = NC1botout


'Estimated vapor and liquid splits
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\BASIS_CSFLOW
    ↪ \4").Value = VapEstC1C2
UnitOps.FindNode("\B1\Subobjects\Connect Streams\8\Input\BASIS_CSFLOW
    ↪ \8").Value = VapEstC1C3
UnitOps.FindNode("\B1\Subobjects\Connect Streams\1\Input\BASIS_CSFLOW
    ↪ \1").Value = LiqEstC1C2
UnitOps.FindNode("\B1\Subobjects\Connect Streams\5\Input\BASIS_CSFLOW
    ↪ \5").Value = LiqEstC2C3


' Estimate temperature profile
Dim j As Integer
Dim k As Integer
Dim l As Integer

Dim dTC1 As Double: dTC1 = (TC1bot − TC1top) / NC1out
Dim TC2top As Double: TC2top = TC1top + dTC1 ∗ NC11
Dim TC2bot As Double: TC2bot = TC1top + dTC1 ∗ (NC11 + NC12 + NC13)
Dim dTC2 As Double: dTC2 = (TC2bot − TC2top) / NC2out
```

```
Dim TC3top As Double: TC3top = TC1top + dTC1 * NC11 + dTC2 * NC21
Dim TC3bot As Double: TC3bot = TC1top + dTC1 * (NC11 + NC12 + NC13 +
    ↪ NC14 + NC15)
Dim dTC3 As Double: dTC3 = (TC3bot - TC3top) / NC3out

' Column 1
For j = 1 To NC1out
Call Estimate1.Elements.InsertRow(0, 0)
Estimate1.Elements.Label(0, 0) = Format$(j, "0")
Estimate1.Elements.Item(0, 0).Value = Format$(TC1top + dTC1 * (j - 1), "0") '
    ↪ CDbl(Cells(13, 2).Offset(i, 1).Value)
Next j

' Column 2
For k = 1 To NC2out
Call Estimate2.Elements.InsertRow(0, 0)
Estimate2.Elements.Label(0, 0) = Format$(k, "0") 'Format$(Cells(13, 2).Offset(i, 0).
    ↪ Value, "0")
Estimate2.Elements.Item(0, 0).Value = CDbl(TC2top + dTC2 * (k - 1)) ' CDbl(
    ↪ Cells(13, 2).Offset(i, 1).Value)
Next k

' Column 3
For l = 1 To NC3out
Call Estimate3.Elements.InsertRow(0, 0)
Estimate3.Elements.Label(0, 0) = Format$(l, "0") 'Format$(Cells(13, 2).Offset(i, 0).
    ↪ Value, "0")
Estimate3.Elements.Item(0, 0).Value = CDbl(TC3top + dTC3 * (l - 1)) ' CDbl(
    ↪ Cells(13, 2).Offset(i, 1).Value)
Next l

Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
MsgBox "Error in SendVariablesToAspen"
ReloadAspenPlusRunIDWithoutVariables
fs.Cells(8, 14) = "Error in Send Variables"
fs.Cells(8, 16) = AspenErrorCounter
Resume PushVariables

End Sub
```

3.1.3 Conventional column sequences

Figure 3.5 shows the user interface in Excel for a direct split sequence consisting of three columns. All following codes apply to this example. However, the proceeding for different column sequences is performed analogue and not presented in detail here.

Load Aspen RunID | Unload Aspen RunID | Push variables to Aspen | Run simulation | Save simulation

Current simulation:

\\cifs.uni-ulm.de\data\cheming\uwp20\03 Projekte\01 Fraunhofer\01_Optimization\04_4 components\02_Ulldirect Split\FourComponent_DS_UlliSystem.bkp

Inputs

Block	Variable	Strom	Wert	Einheit
Directly read out values				
B1	N_{C1}		44	
	Feed stage		20	
B2	N_{C2}		64	
	Feed stage	6	30	
B3	N_{C3}		61	
	Feed stage		34	
B1	Feed stream	FEED	100	
B1	Distillate Stream Flow	A	25.1	mol/h
B2	Distillate Stream Flow	B	24.7	mol/h
B3	Distillate Stream Flow	C	24.9	mol/h
B1	Reboiler Duty		1.01	kW
B2	Reboiler Duty		2.26	kW
B3	Reboiler Duty		1.84	kW
Resulting configuration				
	N_{C11}		20	
	N_{C12}		24	
	N_{C21}		30	
	N_{C22}		34	
	N_{C31}		34	
	N_{C32}		27	

Outputs

Product stream composition

	Flow [mol/h]	$x_{Ethanol}$	$x_{Propanol}$	$x_{Isobutanol}$	$x_{Butanol}$
A	25.1	99.47%	0.53%	0.00%	0.00%
B	24.7	0.33%	99.28%	0.39%	0.00%
C	24.9	0.00%	1.41%	98.19%	0.40%
D	25.3	0.00%	0.00%	1.62%	98.38%

Internal stream outputs

Evaporated Vapor B1	88.5	mol/h
Evaporated Vapor B2	201.5	mol/h
Evaporated Vapor B3	154.3	mol/h

	Flow [mol/h]	$x_{Ethanol}$	$x_{Propanol}$	$x_{Isobutanol}$	$x_{Butanol}$
FeedB2	75	0.11%	33.18%	33.36%	33.36%
FeedB3	50.26	0.00%	0.70%	49.55%	49.74%

Run Status	Error message	Error Count
9345	No Errors	0
Success		
9345		
Success		

Figure 3.5: Layout of the Interface in Excel for DSS-DSS: Worksheet "FlowsheetAccess"

3.1.3.1 Read Aspen Plus Variables

For the sub "CheckSimulationConvergence" see section 3.2.6.

```
Sub GetSimulationVariables()

If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot␣access␣simulation␣Variables" & Chr(13) & _
"because␣no␣Aspen␣Plus␣file␣is␣loaded." & Chr(13) & _
"Load␣a␣RunID␣first.")
End
End If

Dim firstTime As Boolean: firstTime = True

CheckSimulationConvergence

PullVariables:
On Error GoTo ErrorHandler

Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim Root As IHNode: Set Root = AspenPlusRunID.Tree
Dim Streams As IHNode: Set Streams = Root.Data.Streams
Dim UnitOps As IHNode: Set UnitOps = Root.Data.Blocks
Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
Dim count As Integer: count = 0

If SimulationErrorCount = 11 Then
GoTo ErrorHandler
Else
Dim ComponentA As String: ComponentA = "ETHANOL" 'Adapt according to
    ↪ system
Dim ComponentB As String: ComponentB = "PROPANOL" 'Adapt according to
    ↪ system
Dim ComponentC As String: ComponentC = "ISOBUTAN" 'Adapt according to
    ↪ system
Dim ComponentD As String: ComponentD = "NBUTAN" 'Adapt according to
    ↪ system

' ------- Inputs --------
Dim NC1 As Integer: NC1 = UnitOps.B1.Input.NSTAGE.Value
Dim NC2 As Integer: NC2 = UnitOps.B2.Input.NSTAGE.Value
Dim NC3 As Integer: NC3 = UnitOps.B3.Input.NSTAGE.Value
Dim NfC1 As Integer: NfC1 = UnitOps.B1.Input.FEED_STAGE.FEED.Value
```

```
Dim NfC2 As Integer: NfC2 = UnitOps.B2.Input.FEED_STAGE.FEEDB2.Value
Dim NfC3 As Integer: NfC3 = UnitOps.B3.Input.FEED_STAGE.FEEDB3.Value

Dim DistStreamC1 As Double: DistStreamC1 = UnitOps.B1.Input.Elements("
    ↪ BASIS_D").Value
Dim DistStreamC2 As Double: DistStreamC2 = UnitOps.B2.Input.Elements("
    ↪ BASIS_D").Value
Dim DistStreamC3 As Double: DistStreamC3 = UnitOps.B3.Input.Elements("
    ↪ BASIS_D").Value
Dim QC1 As Double: QC1 = UnitOps.B1.Input.Elements("QN").Value
Dim QC2 As Double: QC2 = UnitOps.B2.Input.Elements("QN").Value
Dim QC3 As Double: QC3 = UnitOps.B3.Input.Elements("QN").Value



' Transform stage setup
Dim NC11 As Integer: NC11 = NfC1
Dim NC12 As Integer: NC12 = NC1 - NfC1
Dim NC21 As Integer: NC21 = NfC2
Dim NC22 As Integer: NC22 = NC2 - NfC2
Dim NC31 As Integer: NC31 = NfC3
Dim NC32 As Integer: NC32 = NC3 - NfC3

fs.Cells(9, 4) = NC1
fs.Cells(10, 4) = NfC1
fs.Cells(11, 4) = NC2
fs.Cells(12, 4) = NfC2
fs.Cells(13, 4) = NC3
fs.Cells(14, 4) = NfC3

fs.Cells(15, 4) = Streams.FEED.Input.TOTFLOW.MIXED.Value
fs.Cells(16, 4) = DistStreamC1
fs.Cells(17, 4) = DistStreamC2
fs.Cells(18, 4) = DistStreamC3
fs.Cells(19, 4) = QC1
fs.Cells(20, 4) = QC2
fs.Cells(21, 4) = QC3

'Resulting column stages in each section
fs.Cells(24, 4).Value = NC11
fs.Cells(25, 4).Value = NC12
fs.Cells(26, 4).Value = NC21
fs.Cells(27, 4).Value = NC22
fs.Cells(28, 4).Value = NC31
fs.Cells(29, 4).Value = NC32
```

```
' −−−−−−− Outputs −−−−−−−−

' Product purities
fs.Cells(10, 8) = UnitOps.B1.Input.BASIS_D.Value
fs.Cells(11, 8) = UnitOps.B2.Input.BASIS_D.Value
fs.Cells(12, 8) = UnitOps.B3.Input.BASIS_D.Value

fs.Cells(10, 9) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(10, 10) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(10, 11) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(10, 12) = Streams.A.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(11, 9) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(11, 10) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(11, 11) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(11, 12) = Streams.B.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(12, 9) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(12, 10) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(12, 11) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(12, 12) = Streams.C.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value
fs.Cells(13, 9) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentA).Value
fs.Cells(13, 10) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentB).Value
fs.Cells(13, 11) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentC).Value
fs.Cells(13, 12) = Streams.D.Output.STR_MAIN.MOLEFRAC.MIXED.Elements(
    ↪ ComponentD).Value

' Evaporated Vapor
fs.Cells(17, 8) = UnitOps.B1.Output.VAP_FLOW.Elements(CStr(NC1)).Value
fs.Cells(18, 8) = UnitOps.B2.Output.VAP_FLOW.Elements(CStr(NC2)).Value
```

```
fs.Cells(19, 8) = UnitOps.B3.Output.VAP_FLOW.Elements(CStr(NC3)).Value

' Feed Stream compositions
fs.Cells(23, 8) = Streams.FEEDB2.Output.STR_MAIN.MOLEFLMX.MIXED.Value
fs.Cells(23, 9) = Streams.FEEDB2.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentA).Value
fs.Cells(23, 10) = Streams.FEEDB2.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentB).Value
fs.Cells(23, 11) = Streams.FEEDB2.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentC).Value
fs.Cells(23, 12) = Streams.FEEDB2.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentD).Value

fs.Cells(24, 8) = Streams.FEEDB3.Output.STR_MAIN.MOLEFLMX.MIXED.Value
fs.Cells(24, 9) = Streams.FEEDB3.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentA).Value
fs.Cells(24, 10) = Streams.FEEDB3.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentB).Value
fs.Cells(24, 11) = Streams.FEEDB3.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentC).Value
fs.Cells(24, 12) = Streams.FEEDB3.Output.STR_MAIN.MOLEFRAC.MIXED.
    ↪ Elements(ComponentD).Value

If Worksheets("FlowsheetAccess").Cells(13, 2).Value = 0# Then
If firstTime Then
firstTime = False
'FailSave
'GoTo PullVariables
End If
End If

End If
Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
'MsgBox "Error in GetSimulationVariables"

fs.Cells(7, 15).Value = "Error in GetSimulationVariables"
If SimulationErrorCount = 11 Then
fs.Cells(8, 15).Value = "Data was not updated, too many fails!"
SimulationErrorCount = 0
End If
ReloadAspenPlusRunIDWithoutVariables
```

```
Exit Sub

End Sub
```

3.1.3.2 Send variables to Aspen Plus

The following code is used to send variables to Aspen Plus.

```
Sub SendVariablesToAspen()
If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot␣access␣simulation␣Variables" & Chr(13) & _
"because␣no␣Aspen␣Plus␣file␣is␣loaded." & Chr(13) & _
"Load␣a␣RunID␣first.")
End
End If

Dim fs As Worksheet
Dim Root As IHNode
Dim Streams As IHNode
Dim UnitOps As IHNode

PushVariables:
On Error GoTo ErrorHandler

Set fs = Worksheets("FlowsheetAccess")
Set Root = AspenPlusRunID.Tree
Set Streams = Root.Data.Streams
Set UnitOps = Root.Data.Blocks

'−−−−−−−− Transform variables to Aspen Plus notation −−−−−−−−−−
' Read new inputs from Excel workbook
Dim NC11 As Integer: NC11 = fs.Cells(24, 4).Value
Dim NC12 As Integer: NC12 = fs.Cells(25, 4).Value
Dim NC21 As Integer: NC21 = fs.Cells(26, 4).Value
Dim NC22 As Integer: NC22 = fs.Cells(27, 4).Value
Dim NC31 As Integer: NC31 = fs.Cells(28, 4).Value
Dim NC32 As Integer: NC32 = fs.Cells(29, 4).Value

Dim DistC1out As Double: DistC1out = fs.Cells(16, 4).Value
Dim DistC2out As Double: DistC2out = fs.Cells(17, 4).Value
Dim DistC3out As Double: DistC3out = fs.Cells(18, 4).Value



Dim QC1out As Double: QC1out = fs.Cells(19, 4).Value
```

```
Dim QC2out As Double: QC2out = fs.Cells(20, 4).Value
Dim QC3out As Double: QC3out = fs.Cells(21, 4).Value

' Calculate resulting stages and splits for Aspen
Dim NC3out As Integer: NC3out = NC31 + NC32
Dim NC2out As Integer: NC2out = NC21 + NC22
Dim NC1out As Integer: NC1out = NC11 + NC12

fs.Cells(9, 4).Value = NC1out
fs.Cells(10, 4).Value = NC11
fs.Cells(11, 4).Value = NC2out
fs.Cells(12, 4).Value = NC21
fs.Cells(13, 4).Value = NC3out
fs.Cells(14, 4).Value = NC31


'-------- Send variables to Aspen Plus --------
' Remove all stage imputs first (otherwise errors occur)...
UnitOps.FindNode("\B1\Input\NSTAGE").Value = Nothing
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = Nothing
UnitOps.FindNode("\B2\Input\NSTAGE").Value = Nothing
UnitOps.FindNode("\B2\Input\FEED_STAGE\FEEDB2").Value = Nothing
UnitOps.FindNode("\B3\Input\NSTAGE").Value = Nothing
UnitOps.FindNode("\B3\Input\FEED_STAGE\FEEDB3").Value = Nothing

' ...then specify new
' Column specs
UnitOps.FindNode("\B1\Input\NSTAGE").Value = NC1out
UnitOps.FindNode("\B1\Input\FEED_STAGE\FEED").Value = NC11
UnitOps.FindNode("\B2\Input\NSTAGE").Value = NC2out
UnitOps.FindNode("\B2\Input\FEED_STAGE\FEEDB2").Value = NC21
UnitOps.FindNode("\B3\Input\NSTAGE").Value = NC3out
UnitOps.FindNode("\B3\Input\FEED_STAGE\FEEDB3").Value = NC31

UnitOps.FindNode("\B1\Input\BASIS_D").Value = DistC1out
UnitOps.FindNode("\B2\Input\BASIS_D").Value = DistC2out
UnitOps.FindNode("\B3\Input\BASIS_D").Value = DistC3out

UnitOps.FindNode("\B1\Input\QN").Value = QC1out
UnitOps.FindNode("\B2\Input\QN").Value = QC2out
UnitOps.FindNode("\B3\Input\QN").Value = QC3out

Exit Sub
```

```
ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
MsgBox "Error in SendVariablesToAspen"
ReloadAspenPlusRunIDWithoutVariables
fs.Cells(8, 14) = "Error in Send Variables"
fs.Cells(8, 16) = AspenErrorCounter
Resume PushVariables

End Sub
```

3.2 General Procedures

In this section all general procedures which are required independently of the column model in Aspen Plus are summarized.

3.2.1 Load simulation

Note that the Sub "GetSimulationVariables" is column specific. More details can be found in Section 3.1.

```
Sub LoadAspenPlusRunID()
If AspenPlusRunID Is Nothing Then
Else
UnloadAspenPlusRunID
End If

Dim RunIDPath As String
Dim fd As Office.FileDialog
Set fd = Application.FileDialog(msoFileDialogFilePicker)

With fd
.AllowMultiSelect = False
.Title = "Select Aspen Plus backup file ..."

.Filters.Clear
.Filters.Add "Aspen Plus backup files", "*.bkp"
.Filters.Add "All Files", "*.*"

' Show the dialog box. If the .Show method returns True, the
' user picked at least one file. If the .Show method returns
' False, the user clicked Cancel.
If .Show = True Then
```

```
RunIDPath = .SelectedItems(1)
Else
Exit Sub
End If
End With

Set AspenPlusRunID = GetObject(RunIDPath)
Worksheets("FlowsheetAccess").Cells(5, 1).Value = RunIDPath
GetSimulationVariables

If AspenPlusRunID Is Nothing Then
msg = "Wasn't able to load RunID."
MsgBox msg
End
End If
End Sub
```

3.2.2 Unload simulation

```
Sub UnloadAspenPlusRunID()

Set AspenPlusRunID = Nothing
Worksheets("FlowsheetAccess").Cells(5, 1).Clear

End Sub
```

3.2.3 Reload simulation

Note that the sub "GetSimulationVariables" is column specific. More details can be found in Section 3.1.

```
Sub ReloadAspenPlusRunID()
Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
UnloadAspenPlusRunID
Set AspenPlusRunID = GetObject(RunIDName)
Worksheets("FlowsheetAccess").Cells(5, 1).Value = RunIDName
GetSimulationVariables

If AspenPlusRunID Is Nothing Then
msg = "Wasn't able to reload RunID."
MsgBox msg
```

```
End
End If
End Sub
```

```
Sub ReloadAspenPlusRunIDWithoutVariables()

Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
Set AspenPlusRunID = Nothing
Application.Wait (Now + TimeValue("0:00:05"))
Set AspenPlusRunID = GetObject(RunIDName)

If AspenPlusRunID Is Nothing Then
msg = "Wasn't able to reload RunID."
MsgBox msg
End
End If

End Sub
```

3.2.4 Run simulations

Note that the sub "SendVariablesToAspen" is column specific. More details can be found in Section 3.1.

```
Sub RunSimulation()
If AspenPlusRunID Is Nothing Then
msg = "Load a job first."
MsgBox msg
End
End If

SendVariablesToAspen

On Error GoTo ErrorHandler
AspenPlusRunID.Run
AspenPlusRunID.Run

On Error GoTo 0
GetSimulationVariables

Exit Sub
```

```
ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
ReloadAspenPlusRunIDWithoutVariables
Exit Sub

End Sub
```

3.2.5 Save simulation

```
Sub SaveAspenPlusSimulation()

Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
Set AspenPlusRunID = GetObject(RunIDName)

Call AspenPlusRunID.SaveAs(RunIDName)

End Sub
```

3.2.6 Error Handler

Note that the cells in which the convergence outputs are typed can vary with the column sequence used.

```
Sub CheckSimulationConvergence()

If AspenPlusRunID Is Nothing Then
MsgBox ("Cannot␣access␣simulation␣Variables" & Chr(13) & _
"because␣no␣Aspen␣Plus␣file␣is␣loaded." & Chr(13) & _
"Load␣a␣RunID␣first.")
End
End If

Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim Root As IHNode: Set Root = AspenPlusRunID.Tree
Dim Streams As IHNode: Set Streams = Root.Data.Streams
Dim UnitOps As IHNode: Set UnitOps = Root.Data.Blocks
Dim ErrorCheck As Integer
Dim SimulationStatusNumber As Long
Dim StreamErrorHandler As Double
Dim AlgorithmUsed As String
Dim RunIDName As String: RunIDName = Worksheets("FlowsheetAccess").Cells(5,
    ↪ 1).Value
```

```
fs.Range("M7:O10").Clear
On Error GoTo ErrorHandler
SimulationStatusNumber = Root.Data.AttributeValue(HAP_COMPSTATUS)
SimulationStatusMessage = StatusMessage(SimulationStatusNumber)

fs.Cells(7, 13) = SimulationStatusNumber
fs.Cells(8, 13) = SimulationStatusMessage

If SimulationStatusMessage = "Problem not run" Then
AspenPlusRunID.Run
AspenPlusRunID.Run
SimulationStatusNumber = Root.Data.AttributeValue(HAP_COMPSTATUS)
SimulationStatusMessage = StatusMessage(SimulationStatusNumber)
fs.Cells(9, 13) = SimulationStatusNumber
fs.Cells(10, 13) = SimulationStatusMessage
End If

If SimulationStatusMessage = "Errors" Then
ErrorCheck = 1
Else
ErrorCheck = 0
End If

ReachConvergence:
On Error GoTo ErrorHandler

If ErrorCheck = 1 Then

Do While ErrorCheck = 1

fs.Cells(7, 14).Value = "Error in Simulation Run"
SimulationErrorCount = SimulationErrorCount + 1 'Global variable
fs.Cells(7, 15).Value = SimulationErrorCount



If SimulationErrorCount <= 5 Then
StreamErrorHandler = UnitOps.B1.Subobjects.Elements("Connect Streams").
    ↪ Elements("4").Input.Elements("BASIS_CSFLOW").Elements("4").Value
StreamErrorHandler = StreamErrorHandler * 1.001
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\BASIS_CSFLOW
    ↪ \4").Value = StreamErrorHandler
AspenPlusRunID.Run
AspenPlusRunID.Run
```

```
StreamErrorHandler = StreamErrorHandler / 1.001
UnitOps.FindNode("\B1\Subobjects\Connect Streams\4\Input\BASIS_CSFLOW
    ↪ \4").Value = StreamErrorHandler
AspenPlusRunID.Run
AspenPlusRunID.Run
ElseIf SimulationErrorCount > 5 And SimulationErrorCount <= 10 Then
AlgorithmUsed = UnitOps.B1.Input.ALGORITHM.Value
If AlgorithmUsed = "SUM−RATES" Then
AlgorithmUsed = "NEWTON"
UnitOps.B1.Input.ALGORITHM.Value = "NEWTON"
ElseIf AlgorithmUsed = "NEWTON" Then
AlgorithmUsed = "STANDARD"
UnitOps.B1.Input.ALGORITHM.Value = "STANDARD"
ElseIf AlgorithmUsed = "STANDARD" Then
AlgorithmUsed = "SUM−RATES"
UnitOps.B1.Input.ALGORITHM.Value = "SUM−RATES"
End If
AspenPlusRunID.Run
AspenPlusRunID.Run
ElseIf SimulationErrorCount > 10 Then
GoTo ErrorHandler
End If

SimulationStatusNumber = Root.Data.AttributeValue(HAP_COMPSTATUS)
SimulationStatusMessage = StatusMessage(SimulationStatusNumber)
If SimulationStatusMessage = "Errors" Then
ErrorCheck = 1
Else
ErrorCheck = 0
End If
Loop
End If

If ErrorCheck = 0 Then
SimulationStatusNumber = Root.Data.AttributeValue(HAP_COMPSTATUS)
SimulationStatusMessage = StatusMessage(SimulationStatusNumber)
fs.Cells(9, 13) = SimulationStatusNumber
fs.Cells(10, 13) = SimulationStatusMessage
fs.Cells(7, 14) = "No Errors"
fs.Cells(7, 15) = SimulationErrorCount
SimulationErrorCount = 0
End If
```

```
Exit Sub

ErrorHandler:
AspenErrorCounter = AspenErrorCounter + 1
'MsgBox "Error in CheckSimulationConvergence"

fs.Cells(7, 14).Value = "Error in CheckSimulationConvergence"
ReloadAspenPlusRunIDWithoutVariables
Exit Sub

End Sub
```

```
Function StatusMessage(CompStat As Long) As String
' This function interprets a status variable and returns a string

If ((CompStat And HAP_RESULTS_ERRORS) = HAP_RESULTS_ERRORS)
    ↪ Then
StatusMessage = "Errors"
ElseIf ((CompStat And HAP_RESULTS_WARNINGS) =
    ↪ HAP_RESULTS_WARNINGS) Then
StatusMessage = "Warnings"
ElseIf ((CompStat And HAP_NORESULTS) = HAP_NORESULTS) Then
StatusMessage = "No results"
ElseIf ((CompStat And HAP_RESULTS_INCOMPAT) =
    ↪ HAP_RESULTS_INCOMPAT) Then
StatusMessage = "Incompatible with input"
ElseIf ((CompStat And HAP_RESULTS_INACCESS) =
    ↪ HAP_RESULTS_INACCESS) Then
StatusMessage = "In access"
ElseIf ((CompStat And HAP_NOT_RUN) = HAP_NOT_RUN) Then
StatusMessage = "Problem not run"
ElseIf ((CompStat And HAP_RESULTS_SUCCESS) =
    ↪ HAP_RESULTS_SUCCESS) Then
StatusMessage = "Success"
End If
End Function
```

4 Screening (VBA)

4.1 Simple dividing wall column

For a better understanding consider Section 3.2.4 and Figure 3.2 in section 3.1.1.

```
Sub Sensitivity()
Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim sens As Worksheet: Set sens = Worksheets("SensitivityAnalysis")
Dim Vsplit As Double
Dim Lsplit As Double
Dim Q As Double

sens.Range("A3:R10000") = Empty
sens.Cells(1, 4).Clear

LoadAspenPlusRunID
k = 1

noOfSteps = 12

minValSplitV = 0.15
maxValSplitV = 0.9
stepSizeSplitV = WorksheetFunction.Floor_Precise((maxValSplitV - minValSplitV)
    ↪ / noOfSteps, 0.000001)

minValSplitL = 0.2
maxValSplitL = 0.9
stepSizeSplitL = WorksheetFunction.Floor_Precise((maxValSplitL - minValSplitL) /
    ↪  noOfSteps, 0.000001)

minValQ = 50
maxValQ = 80
stepSizeQ = WorksheetFunction.Floor_Precise((maxValQ - minValQ) / noOfSteps,
    ↪ 0.000001)

Vsplit = minValSplitV
Do While Vsplit <= maxValSplitV

Q = minValQ
Do While Q <= maxValQ

Lsplit = minValSplitL
```

```
Do While Lsplit <= maxValSplitL

fs.Cells(20, 4) = Vsplit
fs.Cells(21, 4) = Lsplit
fs.Cells(17, 4) = Q

RunSimulation

sens.Cells(1, 4) = k
sens.Cells(k + 2, 1) = k

sens.Cells(k + 2, 2) = Q
sens.Cells(k + 2, 3) = Vsplit
sens.Cells(k + 2, 4) = Lsplit

sens.Cells(k + 2, 5) = fs.Cells(17, 10).Value
sens.Cells(k + 2, 6) = fs.Cells(18, 10).Value

sens.Cells(k + 2, 7) = fs.Cells(17, 8).Value
sens.Cells(k + 2, 8) = fs.Cells(18, 8).Value

sens.Cells(k + 2, 9) = fs.Cells(10, 9).Value
sens.Cells(k + 2, 10) = fs.Cells(11, 10).Value
sens.Cells(k + 2, 11) = fs.Cells(12, 11).Value

sens.Cells(k + 2, 12) = fs.Cells(7, 15).Value

Lsplit = Lsplit + stepSizeSplitL

k = k + 1
Loop

Q = Q + stepSizeQ
Loop

Vsplit = Vsplit + stepSizeSplitV
Loop

End Sub
```

After the screening is completely calculated, the data is filtered according to the desired product specifications. Either the data can then be visualized in SOPPS or they are processed further for a visualization of vapor over liquid split. For the creation of SOPPs see Section 5. In order to enable a two dimensional plot first a new workbook is

created in the Excel file with the title "Pur95" (adaptation to other purities is obviously possible) in which the results are copied to line A1. Then the following code is run.

```
Sub DetermineSplitRange()

Dim sens As Worksheet: Set sens = Worksheets("Pur95")
Dim ranges As Worksheet
Dim Vsplit As Double
Dim VsplitAvg As Double
Dim Lsplit As Double
Dim lRow As Long
Dim Q As Double
Dim count As Integer
Dim QVArray() As Variant
Dim VArray() As Variant
Dim LArray() As Variant
Dim length As Integer

On Error GoTo ErrorHandler

'Clear error messages
sens.Cells(1, 15).Clear
sens.Cells(2, 15).Clear

CreateRangeWorksheet
Set ranges = Worksheets("SplitRanges")
ranges.Range("A3:D1000").Clear

' Find the last non-blank cell in Pur95
lRow = sens.Cells(Rows.count, 1).End(xlUp).row

' Copy variable range from screening here
noOfSteps = 12

minValSplits = 0.15
maxValSplits = 0.9
stepSizeSplits = WorksheetFunction.Floor_Precise((maxValSplits - minValSplits) /
    ↪ noOfSteps, 0.000001)



minValQ = 50
maxValQ = 80
stepSizeQ = WorksheetFunction.Floor_Precise((maxValQ - minValQ) / noOfSteps,
    ↪ 0.000001)
```

```
count = 0
Q = minValQ
Vsplit = minValSplits
Do While Q <= maxValQ
Do While Vsplit <= maxValSplits
QVArray = Find(CStr(Q), CStr(Vsplit), lRow)
If QVArray(0, 0) = -1 Then
Else
length = UBound(QVArray, 2) - LBound(QVArray, 2) + 1
ranges.Cells(count + 3, 1) = Q
If length = 1 Then
ranges.Cells(count + 3, 2) = QVArray(2, 0)
ranges.Cells(count + 3, 3) = QVArray(3, 0)
ranges.Cells(count + 3, 4) = QVArray(3, 0)
Else
'Calculate Average real Vsplit
ReDim VArray(0 To length - 1)

For k = 0 To length - 1
VArray(k) = QVArray(2, k)
Next k
k = Empty

VsplitAvg = CalculateAverage(VArray, length)
ranges.Cells(count + 3, 2) = VsplitAvg
'Calculate Min of LSplit
ReDim LArray(0 To length - 1)
For k = 0 To length - 1
LArray(k) = QVArray(3, k)
Next k
k = Empty

ranges.Cells(count + 3, 3) = WorksheetFunction.Min(LArray)
ranges.Cells(count + 3, 4) = WorksheetFunction.Max(LArray)
End If
Erase QVArray
Erase VArray
Erase LArray
count = count + 1
End If
Vsplit = Vsplit + stepSizeSplits
Loop
```

```
'Erase QArray
Vsplit = minValSplits
Q = Q + stepSizeQ
Loop

Exit Sub

ErrorHandler:
sens.Cells(1, 15) = "Error, calculation stopped"
Exit Sub

End Sub
```

```
Function Find(aVal As String, bVal As String, maxRow As Long) As Variant
'Dim maxRow As Long
'maxRow = Range("A65536").End(xlUp).row
Dim row As Long
Dim sens As Worksheet: Set sens = Worksheets("Pur95")
Dim Q As Double
Dim Vsplit As Double
Dim RowsSpec() As Variant
Dim count As Integer: count = 0

On Error GoTo ErrorHandler

For row = 2 To maxRow
Q = sens.Cells(row, 2).Value
Vsplit = sens.Cells(row, 3).Value
If CStr(Q) = aVal And CStr(Vsplit) = bVal Then
If count = 0 Then
ReDim RowsSpec(0 To 3, 0 To 20)
Else
End If
RowsSpec(0, count) = row
RowsSpec(1, count) = Q
RowsSpec(2, count) = sens.Cells(row, 5).Value 'Vsplit real
RowsSpec(3, count) = sens.Cells(row, 6).Value 'Lsplit real
count = count + 1
'Exit Function
End If
Next row



If count = 0 Then
```

```
ReDim RowsSpec(0 To 0, 0 To 0)
RowsSpec(0, 0) = −1
Find = RowsSpec
Else
ReDim Preserve RowsSpec(0 To 3, 0 To count − 1)
Find = RowsSpec
End If



Exit Function

ErrorHandler:
sens.Cells(2, 15) = "Error in 'Find'"
Exit Function

End Function
```

```
Function CalculateAverage(DataSet As Variant, length As Integer) As Double

Dim k As Integer
Dim sum As Double
Dim sens As Worksheet: Set sens = Worksheets("Pur95")

On Error GoTo ErrorHandler
For k = 0 To length − 1
sum = sum + DataSet(k)
Next k

CalculateAverage = sum / length
Exit Function

ErrorHandler:
sens.Cells(2, 15) = "Error in 'CalculateAverage'"
Exit Function
End Function
Sub CreateRangeWorksheet()

On Error Resume Next
Sheets.Add(After:=Worksheets(Worksheets.count)).Name = "SplitRanges"
Dim ranges As Worksheet
Set ranges = Worksheets("SplitRanges")
```

```
ranges.Cells(1, 1) = "Range for each Q"
Range("A1:D1").Merge
Range("A1:D1").Interior.ColorIndex = 16
ranges.Cells(2, 1) = "Q [kW] ="
ranges.Cells(2, 2) = "Averaged Vapor Split Real"
ranges.Cells(2, 3) = "Liquid Split Real, min"
ranges.Cells(2, 4) = "Liquid Split Real, max"
```

4.2 Simplified multiple dividing wall column

The screening of a simplified multiple dividing wall column is performed with the following code. For a better understanding consider Figure 3.4 in Section 3.1.2 and Section 3.2.4.

```
Sub Sensitivity()
Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim SensAnalysis As Worksheet: Set SensAnalysis = Worksheets("Sensitivity
    ↪ Analysis")

SensAnalysis.Range("A3:R10000") = Empty
SensAnalysis.Cells(1, 4).Clear

LoadAspenPlusRunID
I = 1
j = 3
noOfSteps = 8



minValSplitV1 = 0.15
maxValSplitV1 = 0.9
stepSizeSplitV1 = WorksheetFunction.Floor_Precise((maxValSplitV1 -
    ↪ minValSplitV1) / noOfSteps, 0.000001)

minValSplitV2 = 0.15
maxValSplitV2 = 0.9
stepSizeSplitV2 = WorksheetFunction.Floor_Precise((maxValSplitV2 -
    ↪ minValSplitV2) / noOfSteps, 0.000001)

minValSplitL1 = 0.2
maxValSplitL1 = 0.9
stepSizeSplitL1 = WorksheetFunction.Floor_Precise((maxValSplitL1 -
    ↪ minValSplitL1) / noOfSteps, 0.000001)
```

```
minValSplitL2 = 0.2
maxValSplitL2 = 0.9
stepSizeSplitL2 = WorksheetFunction.Floor_Precise((maxValSplitL2 −
    ↪ minValSplitL2) / noOfSteps, 0.000001)


Vsplit1 = minValSplitV1
Do While Vsplit1 < maxValSplitV1

Vsplit2 = minValSplitV2
Do While Vsplit2 < maxValSplitV2

Lsplit1 = minValSplitL1
Do While Lsplit1 < maxValSplitL1

Lsplit2 = minValSplitL2
Do While Lsplit2 < maxValSplitL2

fs.Cells(27, 4) = Vsplit1
fs.Cells(28, 4) = Vsplit2
fs.Cells(29, 4) = Lsplit1
fs.Cells(30, 4) = Lsplit2

RunSimulation

SensAnalysis.Cells(1, 4) = I
SensAnalysis.Cells(I + 2, 1) = I
' Estimated splits
SensAnalysis.Cells(I + 2, 2) = Vsplit1
SensAnalysis.Cells(I + 2, 3) = Vsplit2
SensAnalysis.Cells(I + 2, 4) = Lsplit1
SensAnalysis.Cells(I + 2, 5) = Lsplit2
' Real splits
SensAnalysis.Cells(I + 2, 6) = fs.Cells(18, 10)
SensAnalysis.Cells(I + 2, 7) = fs.Cells(19, 10)
SensAnalysis.Cells(I + 2, 8) = fs.Cells(20, 10)
SensAnalysis.Cells(I + 2, 9) = fs.Cells(21, 10)
' Total stream to split
SensAnalysis.Cells(I + 2, 10) = fs.Cells(18, 8)
SensAnalysis.Cells(I + 2, 11) = fs.Cells(19, 8)
SensAnalysis.Cells(I + 2, 12) = fs.Cells(20, 8)
SensAnalysis.Cells(I + 2, 13) = fs.Cells(21, 8)
' Purities
SensAnalysis.Cells(I + 2, 14) = fs.Cells(10, 9)
```

```
SensAnalysis.Cells(I + 2, 15) = fs.Cells(11, 10)
SensAnalysis.Cells(I + 2, 16) = fs.Cells(12, 11)
SensAnalysis.Cells(I + 2, 17) = fs.Cells(13, 12)
SensAnalysis.Cells(I + 2, 18) = fs.Cells(7, 16)

I = I + 1
Lsplit2 = Lsplit2 + stepSizeSplitL2

Loop

Lsplit1 = Lsplit1 + stepSizeSplitL1
Loop

Vsplit2 = Vsplit2 + stepSizeSplitV2
Loop

Vsplit1 = Vsplit1 + stepSizeSplitV1
Loop


End Sub
```

Afterwards the procedure is analogue as for the simple dividing wall column. The data is filtered according to the desired product specifications. Either the data can then be visualized in SOPPS or they are processed further for a visualization of vapor over liquid split. For the creation of SOPPs see Section 5. In order to enable a two dimensional plot first a new workbook is created in the Excel file with the title "Pur99" (adaptation to other purities is obviously possible) in which the results are copied to line A1. Then the following code is run.

```
Sub DetermineSplitRange()

Dim sens As Worksheet: Set sens = Worksheets("Pur98")
Dim fs As Worksheet: Set fs = Worksheets("FlowsheetAccess")
Dim ranges As Worksheet
Dim VsplitC1C2 As Double
Dim VsplitC1C3 As Double
Dim VsplitC1C2Avg As Double
Dim VsplitC1C3Avg As Double
Dim LsplitC1C2 As Double
Dim LsplitC2C3 As Double
Dim LsplitC2C3ad As Double
Dim lRow As Long
Dim count As Integer
```

```
Dim FindArray() As Variant
Dim VC1C2Array() As Variant
Dim VC1C3Array() As Variant
Dim LC1C2Array() As Variant
Dim LC2C3Array() As Variant
Dim LC2C3adArray() As Variant
Dim length As Integer
Dim Column As String

On Error GoTo ErrorHandler
' copy here variable range from screening
noOfSteps = 8

minValVSplitC1C3 = 0.15
maxValVSplitC1C3 = 0.9
stepSizeVSplitC1C3 = WorksheetFunction.Floor_Precise((maxValVSplitC1C3 −
    ↪ minValVSplitC1C3) / noOfSteps, 0.000001)

minValVSplitC1C2 = 0.15
maxValVSplitC1C2 = 0.9
stepSizeVSplitC1C2 = WorksheetFunction.Floor_Precise((maxValVSplitC1C2 −
    ↪ minValVSplitC1C2) / noOfSteps, 0.000001)
'Clear error messages
sens.Cells(1, 26).Clear
sens.Cells(2, 26).Clear

CreateRangeWorksheet
Set ranges = Worksheets("SplitRanges")
ranges.Range("A4:U1000").Clear

'Find the last non−blank cell in Pur98
lRow = sens.Cells(Rows.count, 1).End(xlUp).row

'Add left related splits in pur98
sens.Cells(1, 20) = "Vapor␣Split␣C1C3␣left"
sens.Cells(1, 21) = "Vapor␣Split␣C1C2␣left"
sens.Cells(1, 22) = "Liquid␣Split␣C1C2␣left"
sens.Cells(1, 23) = "Liquid␣Split␣C2C3␣left"
sens.Cells(1, 24) = "Liquid␣Split␣C2C3␣left␣ad"
sens.Range("T1:X1").WrapText = True

For k = 1 To lRow − 1
sens.Cells(k + 1, 20) = 1 − sens.Cells(k + 1, 6)
sens.Cells(k + 1, 21) = 1 − sens.Cells(k + 1, 7)
```

```
sens.Cells(k + 1, 22) = 1 − sens.Cells(k + 1, 8)
sens.Cells(k + 1, 23) = 1 − sens.Cells(k + 1, 9)
sens.Cells(k + 1, 24) = sens.Cells(k + 1, 22) ∗ sens.Cells(k + 1, 23)
Next k

' Find all values with same V C1C2
count = 0

VsplitC1C2 = minValVSplitC1C2
Do While VsplitC1C2 <= maxValVSplitC1C2
Column = "C1C2"
FindArray = Find(CStr(VsplitC1C2), lRow, Column)
If FindArray(0, 0) = −1 Then
Else
length = UBound(FindArray, 2) − LBound(FindArray, 2) + 1
ranges.Cells(count + 4, 1) = fs.Cells(22, 4)
If length = 1 Then
ranges.Cells(count + 4, 2) = FindArray(2, 0)
ranges.Cells(count + 4, 3) = FindArray(3, 0)
ranges.Cells(count + 4, 4) = FindArray(3, 0)
ranges.Cells(count + 4, 5) = FindArray(1, 0)
ranges.Cells(count + 4, 6) = FindArray(1, 0)
ranges.Cells(count + 4, 7) = FindArray(4, 0)
ranges.Cells(count + 4, 8) = FindArray(4, 0)
ranges.Cells(count + 4, 9) = FindArray(5, 0)
ranges.Cells(count + 4, 10) = FindArray(5, 0)
Else
'Calculate Average real Vsplit C1C2
ReDim VC1C2Array(0 To length − 1)
For k = 0 To length − 1
VC1C2Array(k) = FindArray(2, k)
Next k
k = Empty
VsplitAvg = CalculateAverage(VC1C2Array, length)
ranges.Cells(count + 4, 2) = VsplitAvg

' Calculate Range of Vsplit C1C3
ReDim VC1C3Array(0 To length − 1)
For k = 0 To length − 1
VC1C3Array(k) = FindArray(1, k)
Next k
k = Empty
ranges.Cells(count + 4, 5) = WorksheetFunction.Min(VC1C3Array)
ranges.Cells(count + 4, 6) = WorksheetFunction.Max(VC1C3Array)
```

```
'Calculate Min of LSplit C1C2
ReDim LC1C2Array(0 To length − 1)
For k = 0 To length − 1
LC1C2Array(k) = FindArray(3, k)
Next k
k = Empty

ranges.Cells(count + 4, 3) = WorksheetFunction.Min(LC1C2Array)
ranges.Cells(count + 4, 4) = WorksheetFunction.Max(LC1C2Array)

'Calculate Min of LSplit C2C3
ReDim LC2C3Array(0 To length − 1)
For k = 0 To length − 1
LC2C3Array(k) = FindArray(4, k)
Next k
k = Empty

ranges.Cells(count + 4, 7) = WorksheetFunction.Min(LC2C3Array)
ranges.Cells(count + 4, 8) = WorksheetFunction.Max(LC2C3Array)

'Calculate Min of LSplit C2C3 ad
ReDim LC2C3adArray(0 To length − 1)
For k = 0 To length − 1
LC2C3adArray(k) = FindArray(5, k)
Next k
k = Empty

ranges.Cells(count + 4, 9) = WorksheetFunction.Min(LC2C3adArray)
ranges.Cells(count + 4, 10) = WorksheetFunction.Max(LC2C3adArray)

End If
Erase FindArray
Erase VC1C2Array
Erase VC1C3Array
Erase LC1C2Array
Erase LC2C3Array
Erase LC2C3adArray
count = count + 1
End If
VsplitC1C2 = VsplitC1C2 + stepSizeVSplitC1C2
Loop
'Loop for CaC3 fix
count = 0
```

```
VsplitC1C3 = minValVSplitC1C3
Do While VsplitC1C3 <= maxValVSplitC1C3
Column = "C1C3"
FindArray = Find(CStr(VsplitC1C3), lRow, Column)
If FindArray(0, 0) = −1 Then
Else
length = UBound(FindArray, 2) − LBound(FindArray, 2) + 1
ranges.Cells(count + 4, 12) = fs.Cells(22, 4)
If length = 1 Then
ranges.Cells(count + 4, 13) = FindArray(2, 0)
ranges.Cells(count + 4, 14) = FindArray(2, 0)
ranges.Cells(count + 4, 15) = FindArray(3, 0)
ranges.Cells(count + 4, 16) = FindArray(3, 0)
ranges.Cells(count + 4, 17) = FindArray(1, 0)
ranges.Cells(count + 4, 18) = FindArray(4, 0)
ranges.Cells(count + 4, 19) = FindArray(4, 0)
ranges.Cells(count + 4, 20) = FindArray(5, 0)
ranges.Cells(count + 4, 21) = FindArray(5, 0)
Else
'Calculate Average real Vsplit C1C3
ReDim VC1C3Array(0 To length − 1)
For k = 0 To length − 1
VC1C3Array(k) = FindArray(1, k)
Next k
k = Empty
VsplitAvg = CalculateAverage(VC1C3Array, length)
ranges.Cells(count + 4, 17) = VsplitAvg

'Calculate Range of Vsplit C1C2
ReDim VC1C2Array(0 To length − 1)
For k = 0 To length − 1
VC1C2Array(k) = FindArray(2, k)
Next k
k = Empty
ranges.Cells(count + 4, 13) = WorksheetFunction.Min(VC1C2Array)
ranges.Cells(count + 4, 14) = WorksheetFunction.Max(VC1C2Array)

'Calculate Min of LSplit C1C2
ReDim LC1C2Array(0 To length − 1)
For k = 0 To length − 1
LC1C2Array(k) = FindArray(3, k)
Next k
k = Empty
```

```
ranges.Cells(count + 4, 15) = WorksheetFunction.Min(LC1C2Array)
ranges.Cells(count + 4, 16) = WorksheetFunction.Max(LC1C2Array)

'Calculate Min of LSplit C2C3
ReDim LC2C3Array(0 To length − 1)
For k = 0 To length − 1
LC2C3Array(k) = FindArray(4, k)
Next k
k = Empty

ranges.Cells(count + 4, 18) = WorksheetFunction.Min(LC2C3Array)
ranges.Cells(count + 4, 19) = WorksheetFunction.Max(LC2C3Array)

'Calculate Min of LSplit C2C3 ad
ReDim LC2C3adArray(0 To length − 1)
For k = 0 To length − 1
LC2C3adArray(k) = FindArray(5, k)
Next k
k = Empty

ranges.Cells(count + 4, 20) = WorksheetFunction.Min(LC2C3adArray)
ranges.Cells(count + 4, 21) = WorksheetFunction.Max(LC2C3adArray)

End If
Erase FindArray
Erase VC1C2Array
Erase VC1C3Array
Erase LC1C2Array
Erase LC2C3Array
Erase LC2C3adArray
count = count + 1
End If
VsplitC1C3 = VsplitC1C3 + stepSizeVSplitC1C3
Loop

Exit Sub

ErrorHandler:
sens.Cells(1, 26) = "Error, calculation stopped"
Exit Sub

End Sub
```

```
Function Find(bVal As String, maxRow As Long, Column As String) As Variant
```

```
Dim row As Long
Dim sens As Worksheet: Set sens = Worksheets("Pur98")
Dim Q As Double
Dim Vsplit As Double
Dim RowsSpec() As Variant
Dim count As Integer: count = 0

On Error GoTo ErrorHandler

For row = 2 To maxRow
'Q = sens.Cells(row, 2).Value
If Column = "C1C2" Then
Vsplit = sens.Cells(row, 3).Value
ElseIf Column = "C1C3" Then
Vsplit = sens.Cells(row, 2).Value
End If
If CStr(Vsplit) = bVal Then
If count = 0 Then
ReDim RowsSpec(0 To 5, 0 To 30)
Else
End If
RowsSpec(0, count) = row
RowsSpec(1, count) = sens.Cells(row, 20).Value 'Vsplit C1C3 real
RowsSpec(2, count) = sens.Cells(row, 21).Value 'Vsplit C1C2 real
RowsSpec(3, count) = sens.Cells(row, 22).Value 'Lsplit C1C2 real
RowsSpec(4, count) = sens.Cells(row, 23).Value 'Lsplit C2C3 real
RowsSpec(5, count) = sens.Cells(row, 24).Value 'Lsplit C2C3 real
count = count + 1
'Exit Function
End If
Next row

If count = 0 Then
ReDim RowsSpec(0 To 0, 0 To 0)
RowsSpec(0, 0) = -1
Find = RowsSpec
Else
ReDim Preserve RowsSpec(0 To 5, 0 To count - 1)
Find = RowsSpec
End If

Exit Function

ErrorHandler:
```

```
sens.Cells(2, 26) = "Error in 'Find'"
Exit Function

End Function
```

```
Function CalculateAverage(DataSet As Variant, length As Integer) As Double

Dim k As Integer
Dim sum As Double
Dim sens As Worksheet: Set sens = Worksheets("Pur98")

On Error GoTo ErrorHandler
For k = 0 To length − 1
sum = sum + DataSet(k)
Next k

CalculateAverage = sum / length
Exit Function

ErrorHandler:
sens.Cells(2, 26) = "Error in 'CalculateAverage'"
Exit Function
End Function
```

```
Sub CreateRangeWorksheet()

On Error Resume Next
Dim ranges As Worksheet: Set ranges = Worksheets("SplitRanges")
If ranges Is Nothing Then
Sheets.Add(After:=Worksheets(Worksheets.count)).Name = "SplitRanges"
Set ranges = Worksheets("SplitRanges")
Else
End If

ranges.Cells(1, 1) = "Range for each Q, Vsplit C1C2,l set"
ranges.Range("A1:J1").Merge
ranges.Range("A1:J1").Interior.ColorIndex = 16
ranges.Cells(2, 1) = "Q [kW] ="
ranges.Range("A2:A3").Merge
ranges.Cells(2, 2) = "Column C2"
ranges.Range("B2:D2").Merge
ranges.Cells(3, 2) = "Averaged VSplit C1C2,l Real"
ranges.Cells(3, 2).Interior.ColorIndex = 43
ranges.Cells(3, 3) = "LSplit C1C2, l Real, min"
```

```
ranges.Cells(3, 4) = "LSplit␣C1C2,␣l␣Real,␣max"
ranges.Cells(2, 5) = "Column␣C3"
ranges.Range("E2:J2").Merge
ranges.Cells(3, 5) = "VSplit␣C1C3,l␣Real,␣min"
ranges.Cells(3, 6) = "VSplit␣C1C3,l␣Real,␣max"
ranges.Cells(3, 7) = "LSplit␣C2C3,␣l␣Real,␣min"
ranges.Cells(3, 8) = "LSplit␣C2C3,␣l␣Real,␣max"
ranges.Cells(3, 9) = "LSplit␣C2C3,␣l␣(ad)␣Real,␣min"
ranges.Cells(3, 10) = "LSplit␣C2C3,␣l␣(ad)␣Real,␣max"

ranges.Cells(1, 12) = "Range␣for␣each␣Q,␣Vsplit␣C1C3,l␣set"
ranges.Range("L1:U1").Merge
ranges.Range("L1:U1").Interior.ColorIndex = 16
ranges.Cells(2, 12) = "Q␣[kW]␣="
ranges.Range("L2:L3").Merge
ranges.Cells(2, 13) = "Column␣C2"
ranges.Range("M2:P2").Merge
ranges.Cells(3, 13) = "VSplit␣C1C2,l␣Real,␣min"
ranges.Cells(3, 14) = "VSplit␣C1C2,l␣Real,␣max"
ranges.Cells(3, 15) = "LSplit␣C1C2,␣l␣Real,␣min"
ranges.Cells(3, 16) = "LSplit␣C1C2,␣l␣Real,␣max"
ranges.Cells(2, 17) = "Column␣C3"
ranges.Range("Q2:U2").Merge
ranges.Cells(3, 17) = "Averaged␣VSplit␣C1C3,l␣Real"
ranges.Cells(3, 17).Interior.ColorIndex = 43
ranges.Cells(3, 18) = "LSplit␣C2C3,␣l␣Real,␣min"
ranges.Cells(3, 19) = "LSplit␣C2C3,␣l␣Real,␣max"
ranges.Cells(3, 20) = "LSplit␣C2C3,␣l␣(ad)␣Real,␣min"
ranges.Cells(3, 21) = "LSplit␣C2C3,␣l␣(ad)␣Real,␣max"

ranges.Range("A1:J3").WrapText = True
ranges.Range("L1:U3").WrapText = True
ranges.Range("A1:J3").HorizontalAlignment = xlCenterAcrossSelection
ranges.Range("L1:U3").HorizontalAlignment = xlCenterAcrossSelection

End Sub
```

5 Self-organizing patch plots (SOPPs) (Matlab)

For the calculation of SOPPs a Matlab code is used. Generally, the transformation from the matrix data to the SOPP consists of the following steps [11]:

1.) Train self-organizing maps (SOMs) with the high-dimensional results

2.) Find best matching units (BMU)

3.) Create Voronoi patches

4.) Color patches

A user interface is developed to create and explore high dimensional data sets in the SOPPs. SOPPs can either be used to visualize optimization or screening results. Here an example is given for the exploration of a data set obtained from an optimization of a simple dividing wall column. However, the code is similar for plotting screening results. Figure 5.1 shows the user interface to create SOPPs. From the user interface,

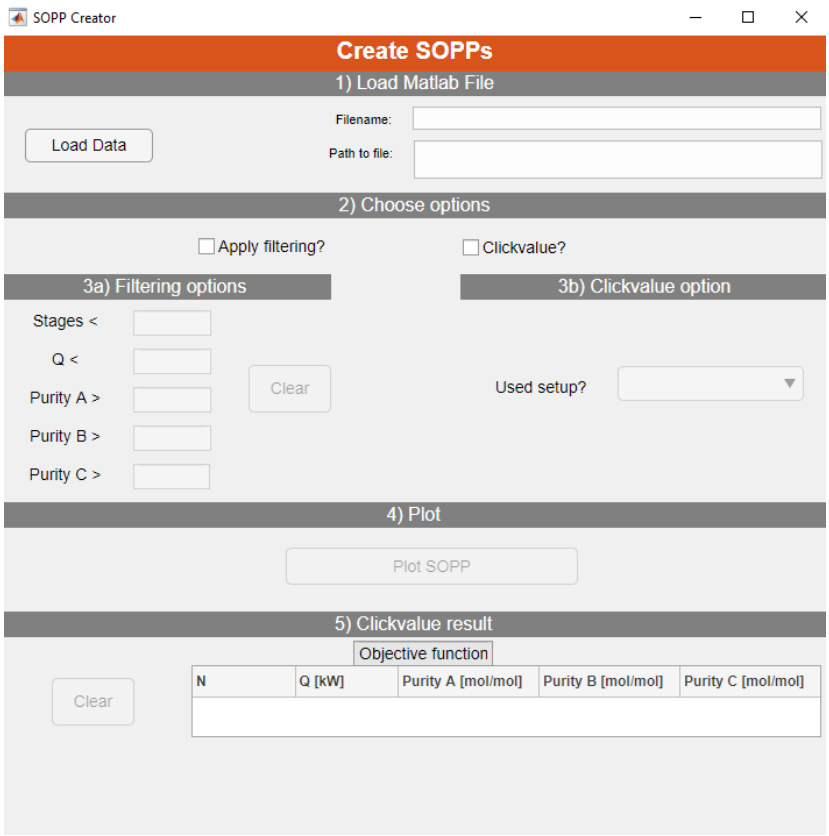

Figure 5.1: Interface for the creation and filtering of the SOPPs. As additional tool, one can click on one patch and get the underlying data as output.

the following function gets the input if and how data filtration should be applied and if the user wants to get the data set behind a patch on which he clicked as output.

```
function [ClickPoint]=MCO_BTX_DWC_EH2_NPQ_run12(FilterYesNo,
    ↪ parameterRestriction, ClickValueYesNo)
tic
%% Large, large matrix with results from optimization
DataForSOM = [];

%% Plot properties
n_feat=size(DataForSOM,2);
cax = [20 120; 20 200; 0.5 1; 0.5 1;0.5 1; 0 1; ...
0 1; 0 1;0 1;0 1;0 1;0 1;0 1]; %N, Q, purities, others
cmap = 'jet';
fontsize = 28;

[inputVoronoi] = getProjectionSOM(DataForSOM,60,30,0.1,0.08,30,0.05,0); %
    ↪ Probably these parameters have to be adapted for another data set
patches = getVoronoiPatches(inputVoronoi);

[subplotDist] = plotSOPP_DWC(patches,inputVoronoi,cmap,fontsize, cax, n_feat);
    ↪ %plots diagram and gives subplot distrubution

if FilterYesNo == 1
SOPP_filter(inputVoronoi, patches, n_feat, parameterRestriction, subplotDist);
end
if ClickValueYesNo == 1
[ClickPoint] = SOPP_ClickValue(inputVoronoi);
end
toc
end
```

The following function getProjectionSOM normalizes the data set to values between zero and one, calls the function to calculate the SOMs and the one to determine the BMU inside the resulting maps. Note that part of the following code is taken from [11].

```
function[inputVoronoi] = getProjectionSOM(data,ndim,nepochs,eta0,etadecay,sgm0,
    ↪ sgmdecay,reallyRandom)
% standard input (nfeatures,60,30,0.1,0.05,30,0.05,0)
nfeatures = size(data,2);
neuronList = data;

maximum = ones(size(neuronList,1),1) * max(neuronList);
minimum = ones(size(neuronList,1),1) * min(neuronList);

neuronList = (neuronList - minimum)./(maximum - minimum);
```

```
som = SOMAbgewandelt(nfeatures,ndim,nepochs,neuronList,eta0,etadecay,sgm0,
    ↪ sgmdecay,reallyRandom); %normiert

inputVoronoi = findBMUData(som,data,ndim,nfeatures); %nicht mehr normiert

[~,uniqueIndices] = unique(inputVoronoi(:,1:2),'rows');
uniqueIndices = sortrows(uniqueIndices);
inputVoronoi = inputVoronoi(uniqueIndices,:);

end
```

The functions getVoronoiPatches, boundPatch and getNeighboringVorteces calculate the patches for the Voronoi diagram.

```
function[patches] = getVoronoiPatches(inputVoronoi)

[v,c] = voronoin(inputVoronoi(:,1:2),{'Qbb','Qz'}); %voronoin(inputVoronoi(:,1:2))
[ex,ey] = voronoiInfinitePoints(inputVoronoi(:,1),inputVoronoi(:,2));

patches = cell(size(inputVoronoi,1),2);

for i = 1:size(inputVoronoi,1)
% if length(c{i}) == 2
% continue
% else
if all(c{i}~=1)
patches{i,1} = v(c{i},1);
patches{i,2} = v(c{i},2);
else % If at least one of the indices is 1,then it is an open
% region and we cannot yet patch that.
boundedPatch = boundPatch(v,c{i},ex,ey);
patches{i,1} = boundedPatch(:,1);
patches{i,2} = boundedPatch(:,2);
end
end
end


function[indexArrayPatch] = boundPatch(v,indexArrayPatch,ex,ey)
positionInf = find(indexArrayPatch==1);
if positionInf == 1
startPointIndex = indexArrayPatch(end);
endPointIndex = indexArrayPatch(2);
```

```
[vortex1,vortex2] = getNeighboringVorteces(v,ex,ey,startPointIndex,endPointIndex);
indexArrayPatch = [vortex1,vortex2,v(indexArrayPatch(2:length(indexArrayPatch))
    ↪ ,:)'];
elseif positionInf == length(indexArrayPatch)
startPointIndex = indexArrayPatch(end−1);
endPointIndex = indexArrayPatch(1);
[vortex1,vortex2] = getNeighboringVorteces(v,ex,ey,startPointIndex,endPointIndex);
indexArrayPatch = [vortex2,v(indexArrayPatch(1:end−1),:)',vortex1]';
else
startPointIndex = indexArrayPatch(positionInf−1);
endPointIndex = indexArrayPatch(positionInf+1);
[vortex1,vortex2] = getNeighboringVorteces(v,ex,ey,startPointIndex,endPointIndex);
indexArrayPatch = [v(indexArrayPatch(1:positionInf−1),:)',vortex1,vortex2,v(
    ↪ indexArrayPatch(positionInf+1:length(indexArrayPatch)),:)'];
end
end


function[vortex1,vortex2] = getNeighboringVorteces(v,ex,ey,startPointIndex,
    ↪ endPointIndex)
% output of voronoi and voronoin not excactly the same
precision = 1e10;
ex = round(ex * precision)/precision;
ey = round(ey * precision)/precision;
v = round(v * precision)/precision;

% get inner and outer vertices
innerPoints = [ex(1,:) ; ey(1,:)]';
outerPoints = [ex(2,:) ; ey(2,:)]';

% get vertices lying next to 'Inf'
startPoint = v(startPointIndex,:);
endPoint = v(endPointIndex,:);

% find the indices to the voronoi edges to 'Inf'
indexStart = ismember(innerPoints,startPoint,'rows');
indexEnd = ismember(innerPoints,endPoint,'rows');

% get coordinates representing 'Inf'
vortex1 = outerPoints(indexStart,:)';
vortex2 = outerPoints(indexEnd,:)';
end
```

The following function SOMAbgewandelt is based on the code of Azzopardi [12] and is the one that calculates the SOM itself by training it with the optimization results.

```
function som = SOMAbgewandelt(nfeatures, ndim, nepochs, trainingData, eta0,
    ↪ etadecay, sgm0, sgmdecay, reallyRandom)
%SOMSimple Simple demonstration of a Self−Organizing Map that was proposed by
    ↪ Kohonen.
% sommap = SOMSimple(nfeatures, ndim, nepochs, ntrainingvectors, eta0, neta,
    ↪ sgm0, nsgm, showMode)
% trains a self−organizing map with the following parameters
% nfeatures − dimension size of the training feature vectors
% ndim − width of a square SOM map
% nepochs − number of epochs used for training
% ntrainingvectors − number of training vectors that are randomly generated
% eta0 − initial learning rate
% etadecay − exponential decay rate of the learning rate
% sgm0 − initial variance of a Gaussian function that
% is used to determine the neighbours of the best
% matching unit (BMU)
% sgmdecay − exponential decay rate of the Gaussian variance
% showMode − 0: do not show output,
% 1: show the initially randomly generated SOM map
% and the trained SOM map,
% 2: show the trained SOM map after each update
%
% For example: A demonstration of an SOM map that is trained by RGB values
%
% som = SOMSimple(3,60,10,100,0.1,0.05,20,0.05,2);
% % It uses:
% % 3 : dimensions for training vectors, such as RGB values
% % 60x60: neurons
% % 10 : epochs
% % 100 : training vectors
% % 0.1 : initial learning rate
% % 0.05 : exponential decay rate of the learning rate
% % 20 : initial Gaussian variance
% % 0.05 : exponential decay rate of the Gaussian variance
% % 2 : Display the som map after every update

nrows = ndim;
ncols = ndim;

%% maybe fixed random numbers
if reallyRandom == 0
```

```
rng('default');
rng(1);
end

som = rand(nrows,ncols,nfeatures);

% Generate random training data
%trainingData = rand(ntrainingvectors,nfeatures);
ntrainingvectors = size(trainingData,1);
% Generate coordinate system
[x,y] = meshgrid(1:ncols,1:nrows);

for t = 1:nepochs
% Compute the learning rate for the current epoch
eta = eta0 * exp(-t*etadecay);

% Compute the variance of the Gaussian (Neighbourhood) function for the current
    ↪ epoch
sgm = sgm0 * exp(-t*sgmdecay);

% Consider the width of the Gaussian function as 3 sigma
width = ceil(sgm*3);

for ntraining = 1:ntrainingvectors
% Get current training vector
trainingVector = trainingData(ntraining,:);

% Compute the Euclidean distance between the training vector and
% each neuron in the SOM map
dist = getEuclideanDistance(trainingVector, som, nrows, ncols, nfeatures);

% Find the best matching unit (bmu)
[~, bmuindex] = min(dist);

% transform the bmu index into 2D
[bmurow,bmucol] = ind2sub([nrows ncols],bmuindex);

% Generate a Gaussian function centered on the location of the bmu
g = exp(-(((x - bmucol).^2) + ((y - bmurow).^2)) / (2*sgm*sgm));

% Determine the boundary of the local neighbourhood
fromrow = max(1,bmurow - width);
torow = min(bmurow + width,nrows);
fromcol = max(1,bmucol - width);
```

```
tocol = min(bmucol + width,ncols);

% Get the neighbouring neurons and determine the size of the neighbourhood
neighbourNeurons = som(fromrow:torow,fromcol:tocol,:);
sz = size(neighbourNeurons);

% Transform the training vector and the Gaussian function into
% multi−dimensional to facilitate the computation of the neuron weights update
T = reshape(repmat(trainingVector,sz(1)*sz(2),1),sz(1),sz(2),nfeatures);
G = repmat(g(fromrow:torow,fromcol:tocol),[1 1 nfeatures]);

% Update the weights of the neurons that are in the neighbourhood of the bmu
neighbourNeurons = neighbourNeurons + eta .* G .* (T − neighbourNeurons);

% Put the new weights of the BMU neighbouring neurons back to the
% entire SOM map
som(fromrow:torow,fromcol:tocol,:) = neighbourNeurons;

end
end
```

getEuclideanDistance calculates the eucledian distance that is needed for the calculation of the SOM.

```
function ed = getEuclideanDistance(trainingVector, sommap, nrows, ncols,
    ↪ nfeatures)

% Transform the 3D representation of neurons into 2D
neuronList = reshape(sommap,nrows*ncols,nfeatures);

% Initialize Euclidean Distance
ed = 0;
for n = 1:size(neuronList,2)
ed = ed + (trainingVector(n)−neuronList(:,n)).^2;
end
ed = sqrt(ed);
```

This function calculates the BMU data for the transformation of an SOM to an SOPP.

```
function[inputVoronoi] = findBMUData(sommap,data,ndim,nfeatures)
nrows = ndim;
ncols = ndim;

inputVoronoi = zeros(size(data,1),size(data,2)+2);
inputVoronoi(:,3:size(data,2)+2) = data;
```

```
maximum = max(data(:,1:nfeatures));
minimum = min(data(:,1:nfeatures));

for n = 1:size(data,1)
trainingVector = (data(n,1:nfeatures)−minimum)./(maximum−minimum);
dist = getEuclideanDistance(trainingVector, sommap(:,:,1:nfeatures), nrows, ncols,
    ↪ nfeatures);

% Find the best matching unit (bmu)
[~, bmuindex] = min(dist);

% transform the bmu index into 2D
[bmurow,bmucol] = ind2sub([nrows ncols],bmuindex);

% assign Pareto efficient solutions to indices
inputVoronoi(n,1) = bmucol;
inputVoronoi(n,2) = bmurow;
end
```

The following code is specific for plotting the optimization results of a DWC and has to be adapted for a different data set.

```
function [subplotDist] = plotSOPP_DWC(patches,inputVoronoi,cmap,fontsize, cax,
    ↪ n_feat)

figure('DefaultTextFontSize',fontsize, 'units','normalized','outerposition',[0 0 1 1]);

tags = {'\Sigma{\it␣N}','{\it␣Q}␣[kW]',...
        {'Purity␣A', '[mol/mol]'},{'Purity␣B', '[mol/mol]'},{'Purity␣C', '[mol/mol]'
            ↪ },...
        'Vapor␣split','Liquid␣split',...
        '{\it␣D/F}', '{\it␣S/F}',...
        {'Feed␣stage', 'height␣ratio'},{'Side␣stream', 'height␣ratio'},...
        {'Dividing␣wall', 'height␣ratio'},{'Vertical␣dividing', 'wall␣position'}};

[subplotDist] =[3, ceil(n_feat/3),10, 15]; % subplot properties n x m and subplot 10
    ↪ is empty

for k = 1:n_feat
if (1<=k) && (k<=9)
subplot(subplotDist(1),subplotDist(2),k)
elseif (9<k) && (k<=n_feat)
subplot(subplotDist(1),subplotDist(2),k+1)
```

```
end
tag=tags{k};
plotOneFeatureInVoronoiPatches(patches,inputVoronoi,k,tag,cmap,fontsize)
caxis(cax(k, :));
set(colorbar,'YTick',cax(k,1):(cax(k,2)−cax(k,1))/2:cax(k,2), 'FontSize', fontsize);

end

end
```

```
function[] = plotOneFeatureInVoronoiPatches(patches,inputVoronoi,ifeature,tag,
    ↪ cmap,fontsize)

%set(gca,'Fontsize',fontsize,'YDir','normal');

hold off

for i = 1:size(inputVoronoi,1)
patch(patches{i,1},patches{i,2},inputVoronoi(i,2+ifeature));
end

hold on
setPlotOptions(inputVoronoi,ifeature,fontsize,tag,cmap);

end

function[] = setPlotOptions(inputVoronoi,ifeature,fontsize,tag,cmap)

xlim([min(inputVoronoi(:,1)) max(inputVoronoi(:,1))]);
ylim([min(inputVoronoi(:,2)) max(inputVoronoi(:,2))]);

colormap(cmap);

colorbar('Fontsize',fontsize, 'TickLabelInterpreter', 'latex' );
caxis([min(inputVoronoi(:,2+ifeature)) max(inputVoronoi(:,2+ifeature))]);

title(tag,'Fontsize',fontsize, 'Interpreter', 'latex');

axis square;
set(gca,'XTick',[],'YTick',[]);
box on
end
```

The two following functions are used to explore the SOPP. First, it is possible to filter the patches according to the input of the user interface. As a result, non-suited patches are colored black.

```
function [] = SOPP_filter(inputVoronoi, patches, n_feat, parameterRestriction,
    ↪ subplotDist)

number_plots = subplotDist(1) * subplotDist(2);

for m = 1:size(parameterRestriction,2)
if isnan(parameterRestriction(m))== 1 %if there is no restriction this loop is
    ↪ skipped
else
for k = 1:size(inputVoronoi,1)
if m >=1 && m <= 2
if inputVoronoi(k,m+2)>parameterRestriction(m)
subplot(subplotDist(1),subplotDist(2),m);
patch(patches{k,1},patches{k,2},'black');
for j=setdiff(1:number_plots, subplotDist(3:end))
subplot(subplotDist(1),subplotDist(2),j);
patch(patches{k,1},patches{k,2},'black');
end
end
elseif m > 2 && m <= size(parameterRestriction,2)
if inputVoronoi(k,m+2)<parameterRestriction(m)
subplot(subplotDist(1),subplotDist(2),m);
patch(patches{k,1},patches{k,2},'black');
for j=setdiff(1:number_plots, subplotDist(3:end))
subplot(subplotDist(1),subplotDist(2),j);
patch(patches{k,1},patches{k,2},'black');
end
end
end
end
end
end
end
```

The second function enables to click on one patch and get the corresponding data as output at the bottom of the user interface.

```
function [SOPP_ClickValue] = SOPP_ClickValue(inputVoronoi)

% Load clickpoint tool
[x_coord, y_coord] = ginput(1);
```

```
% Eucledian distances of clickpoint to all seed points
n_SeedPoints = size(inputVoronoi,1);
EucDistance = zeros(1,n_SeedPoints);

for n = 1:n_SeedPoints
EucDistance(1,n) = sqrt(((x_coord−inputVoronoi(n,1))^2)+((y_coord−
    ↪ inputVoronoi(n,2))^2));
end

% Seed point corresponding to clickpoint has minimum distance
[minEucDistance, pos] = min(EucDistance);

SOPP_ClickValue = inputVoronoi(pos,3:size(inputVoronoi,2));
end
```

Bibliography

[1] A. J. V. Underwood: *Fractional Distillation of Ternary Mixtures Part I.* Journal of the Institute of Petroleum, pages 111–118, 1945.

[2] A. J. V. Underwood: *Fractional Diatillation of Ternary Mixtures Part II.* Journal of the Institute of Petroleum, pages 198–613, 1946.

[3] A. J. V. Underwood: *Fractional Distillation of Multicomponent Mixtures - Calculation of Minimum Reflux Ratio.* Journal of the Institute of Petroleum, pages 614–626, 1946.

[4] A. J. V. Underwood: *Fractional Distillation of Multicomponent Mixtures.* Industrial & Engineering Chemistry, 41(12):2844–2847, 1949.

[5] I. J. Halvorsen and S. Skogestad: *Minimum Energy Consumption in Multicomponent Distillation. 1. Vmin Diagram for a Two-Product Column.* Industrial & Engineering Chemistry Research, 42(3):596–604, 2003.

[6] I. J. Halvorsen and S. Skogestad: *Minimum Energy Consumption in Multicomponent Distillation. 2. Three-Product Petlyuk Arrangements.* Industrial & Engineering Chemistry Research, 42(3):605–615, 2003.

[7] I. J. Halvorsen and S. Skogestad: *Minimum Energy Consumption in Multicomponent Distillation. 3. More Than Three Products and Generalized Petlyuk Arrangements.* Industrial & Engineering Chemistry Research, 42(3):616–629, 2003.

[8] L.-M. Ränger, U. Preissinger and T. Grützner: *Robust Initialization of Rigorous Process Simulations of Multiple Dividing Wall Columns via Vmin Diagrams.* ChemEngineering, 2(2):25, 2018.

[9] J. Koehler, P. Poellmann and E. Blass: *A Review on Minimum Energy Calculations for Ideal and Nonideal Distillations.* Industrial & Engineering Chemistry Research, 34(4):1003–1020, 1995.

[10] M. Mazzotti: *Underwood's equations: derivation: Multicomponent distillation column design.*

[11] K. Stöbener, P. Klein, M. Horsch, K.-H. Küfer and H. Hasse: *Parametrization of two-center Lennard-Jones plus point-quadrupole force field models by multicriteria optimization.* Fluid Phase Equilibria, 411:33–42, 2016.

[12] G. Azzopardi: *Self-Organizing Map - Simple demonstration*, 2013.