TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

A MASTER'S THESIS

Development of a Steam Generator Simulation Model

prepared for the purpose of obtaining an academic degree of
Master of Science in Mechanical Engineering

under the guidance of
Projektass. Dipl.-Ing. Johannes Widhalm
Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Andreas Werner

submitted to Technical University of Vienna,
Institute for Energy Systems and Thermodynamics by

Hakkı ÜNER
1027827
Linzer Straße 429
1140 Wien

Vienna, in November 2013

Confidentiality

This master's thesis is neither completely nor partially permitted without the written consent of the author, as well as the Chair of the TU Vienna, to be reproduced, published or made accessible to third parties .

The submitted work is the property of the Chair as an exam documentation and remains there under lock and key.

I hereby declare and confirm that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references. I hereby agree to the storage of it permanently in the Library of Chairs at TU Vienna.

Vienna, den November 2, 2013 _____

Abstract

To keep up with today's new technologies & work environment, engineers & researchers have to use their time and work in an efficient way. To achieve results in a quick fashion, use of engineering simulation programs & software is getting more and more important everyday. In the energy industrial sector, engineers face similar challenges. Complex calculations for thermodynamics, heat transfer, fluid mechanics applications lead engineers & researchers to use up-to-date simulation and modeling tools. One of them is *Simulation Software IPSEpro* which is used in this master's thesis.

The subject of this master's thesis is the development of a steam generator simulation model in terms of convection and radiation calculations. It comprises of six chapters.

The first chapter gives an overview of steam generators, introduces a water-tube boiler concept, a summary of available simulation software in the market.

In the second chapter, Simulation Software IPSEpro is defined. Its Model Development Kit and Process Simulation Environment are mentioned; some significant points such as model-connections, systems analysis, numerical simulation methods are described.

The third chapter consists of two sections. In the first section, considerable theoretical approaches and formulas are given. To complement this, the second section introduces models developed in this work, the practical side and visual diagrams of sub-models.

The fourth chapter enlightens the reader about the structure of the developed model, the path that should be followed for the convergence method, the features used within the simulation software IPSEpro and the parameters considered for the simulated process.

The fifth chapter presents the test results of an entire steam generator process and a duct with different bundles. The influence of coatings for exterior walls and tube types on system, is evaluated. Regarding steam generator process, simulation results are shown and validated with a practical example.

The sixth chapter is the conclusions section. The aim of this master's thesis is briefly mentioned, literature survey is given. Advices and recommendations are given to engineers or researchers who want to study this subject further.

Contents

List of Figures

List of Tables

1. Introduction

This chapter describes the generation of steam, introduces a typical water-tube boiler concept, gives an overview of simulation software.

Steam generators or hot-water boilers produce hot water or steam for different sectors. There are different boiler types such as Fire-Tube Boiler, Water-Tube Boiler, Fluidized Bed Combustion Boiler, Pulverized Fuel Boiler, Waste Heat Boiler etc. The most common ones used in the industry are Water & Fire Tube Boilers.

The following figure 1.1 shows a LaMont Boiler which consists of a radiation chamber and some convective parts. The feed water flows through the economizer in the steam drum. The steam production is done by a forced circulation system (steam drum, circulating pump, radiation section & convective tube section where steam is generated, steam drum). The saturated steam from the drum flows through the Superheater (get superheated) and then to the turbine.

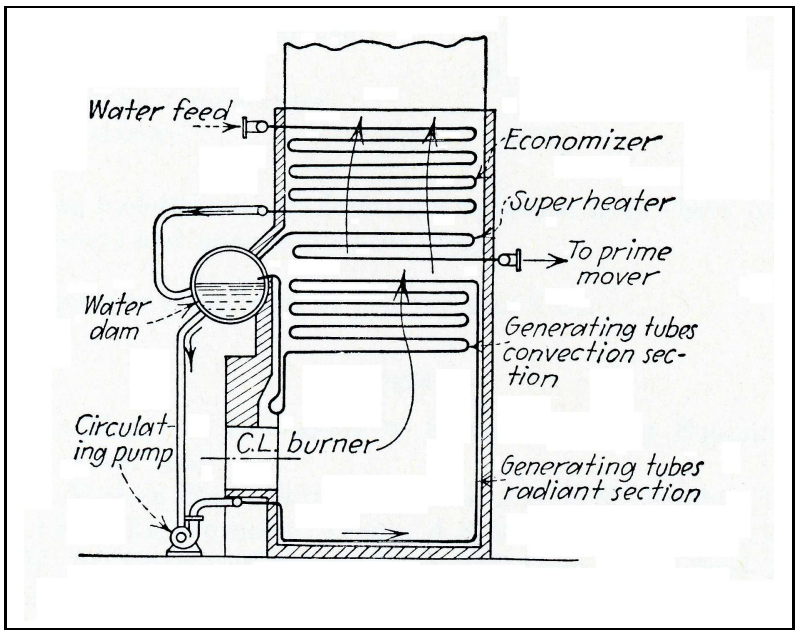


Figure 1.1.: LaMont Boiler [4]

In this master's thesis, the simulation model of Steam Generator is developed in Model Development Kit Environment of IPSEpro software. Different calculation methods are researched and the most suitable ones are embedded into the program. Moreover to validate new model an example problem [7] which gives a hand calculation of results, is tested with the same input parameters. Validation was successful.

There are also other steady state or dynamic simulation software such as Apros, Ebsilon, Enbipro, Dymola etc. which provide accurate results for users in a quick way. Furthermore, it is possible to combine generated models . This action allow users to see entire process in all respects. At the same time detection of a problem is easy.

2. IpsePro

In this chapter, Simulation Software IPSEpro is defined. Its Model Development Kit and Process Simulation Environment are mentioned; some significant points such as model-connections, systems analysis, numerical simulation methods are described. To attain more knowledge, IPSEpro's documentation files such as [8], [9] can be read.

'IPSEpro is a highly flexible and comprehensive environment for modeling and analyzing processes in energy engineering, chemical engineering, and many other related areas.' [8]

Modules of IPSEpro

- MDK
- PSE
- PSSolver
- PSValidate
- PSEasy
- PSServer
- PSOptimize

In this thesis two modules of IPSEpro have been used:

-MDK (Model Development Kit)
'MDK is IPSEpro's Model Development Kit for building up new component models or modifying existing ones in a library.' [8]

-PSE Process Simulation Environment
'With PSE you create a process model based on components from a library. PSE provides a user-friendly flowsheet editor, where you are able to build process models by selecting the components from a menu. You place the components in the project window, specify the data interactively, and connect the components according to your requirements.' [9]

Connections

Free equations method is preferred to build up connection of variables or parameters among different models.
'A free equation is very similar to a model equation except that it does not belong to a single object.' [9]

Items

'IPSEpro defines the following types of items that are used to describe a model mathematically:

- variables
- parameters
- curves/tables
- switches
- built-in functions and operators

- external functions

The first four items represent data that you provide when you build a process model.' [8] As an input, a realistic value must be given to the item which is defined as a parameter or a variable. Thus, there are only a few iterations necessary until the results converge.

Solution Method

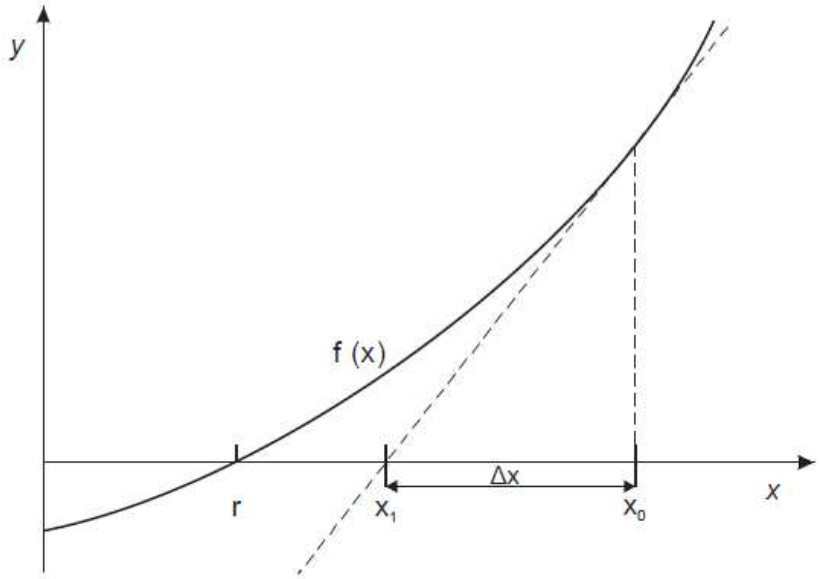
IPSEpro uses Newton-Raphson method to make convergences.

'To solve a system, PSE adopts a two-phase approach:

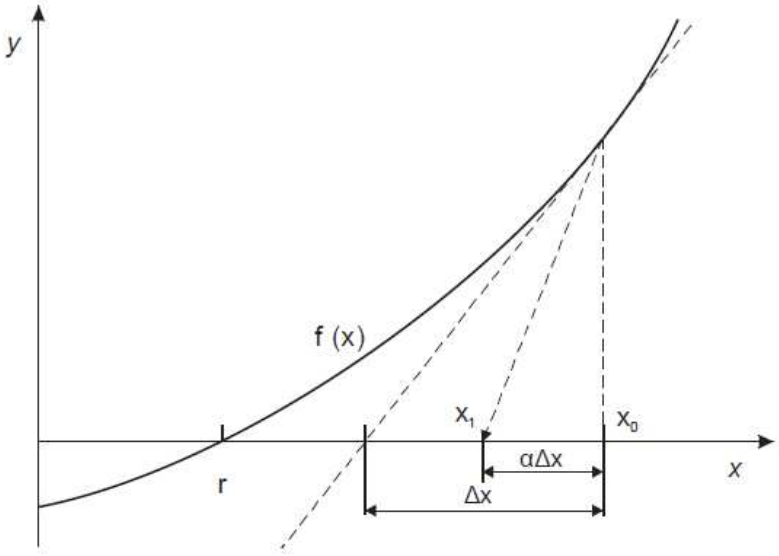
- System analysis *In the analysis phase PSE determines the optimum solution method for the equation system. It analyses the order in which it can treat the variables of a model. If PSE must solve several equations simultaneously, it combines them into groups. During the analysis phase PSE also chooses the optimum numerical method for each group.*

- Numerical solution *When calculating the numerical solution of the system, PSE solves the equations in the order pre-established by the analysis, and uses the numerical methods that also have been chosen in the analysis phase. For the numerical solution PSE uses the Newton-Raphson method for finding solutions. The Newton-Raphson method starts from starting values and approximates the system functions by their linearization at starting value. Using the linearized system a new starting value is calculated. This process is repeated until a sufficiently accurate solution is reached.*

Two variants of the Newton-Raphson method are available' as shown in figure 2.1a & 2.1b. [9]



(a) Undamped Newton-Raphson Method [9]



(b) Damped Newton-Raphson Method [9]

Figure 2.1.: Newton-Raphson Method

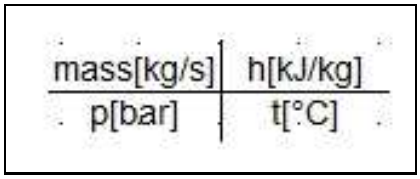


Figure 2.2.: Reference Cross

SI Unit & the Symbol of results are seen as reference cross in figure 2.2.

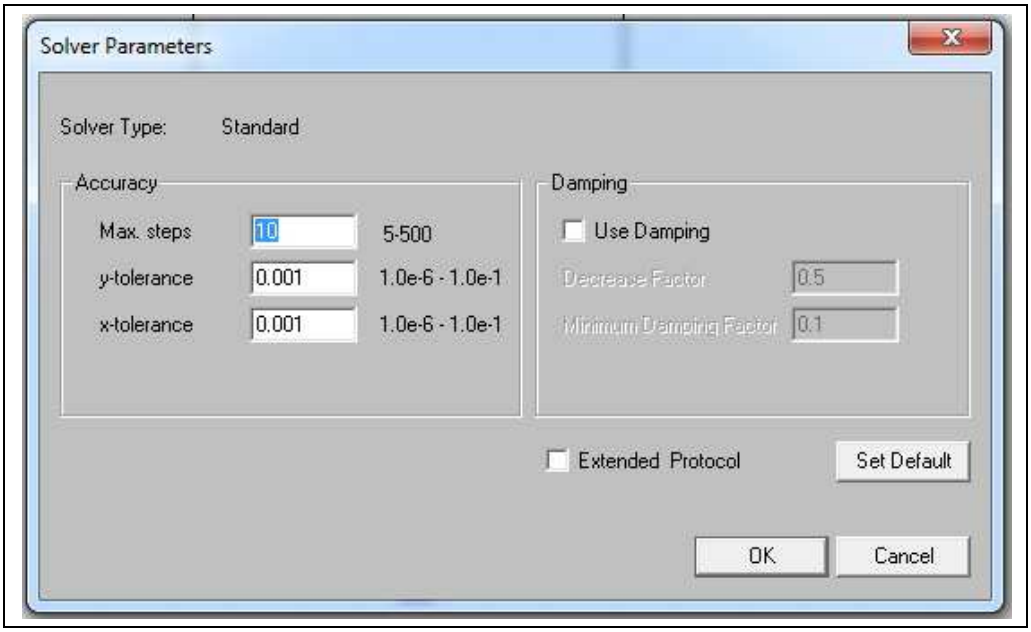


Figure 2.3.: Solver Parameters

When the value of 'Maximum Iteration Steps' is insufficient in number, user would see a warning on the screen such as 'Newton: Group not converged' 2.4. To avoid this error, user might increase the value of 'Maximum Iteration Steps' in number. That action would let software make numerically much more convergences.

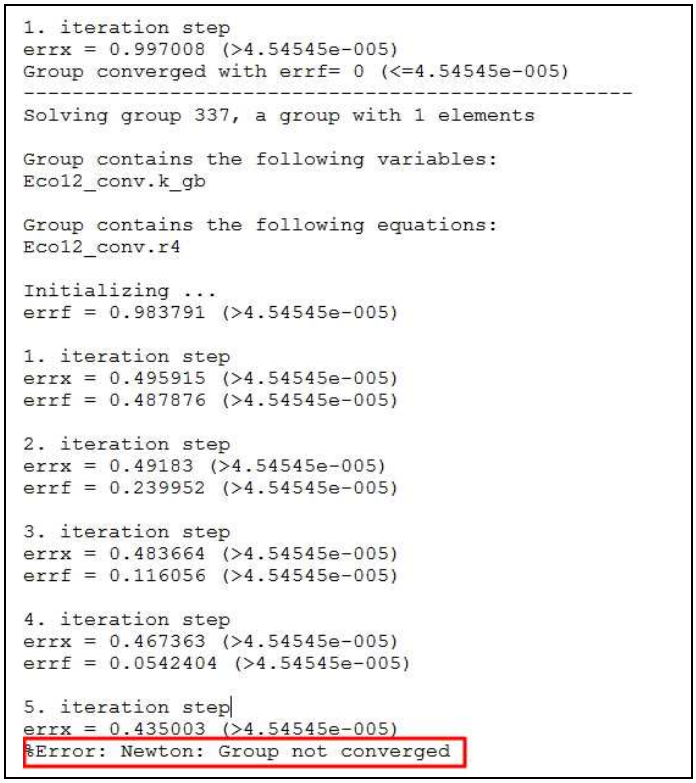


Figure 2.4.: Error: Newton: Group not converged

For instance; calculation ends at fifth iteration step as seen in figure 2.4. This means that the value entered to 'Max. Steps' shown in figure 2.3 was '5'. Changing this value from '5' to '10' would solve the problem.

In order to change 'Maximum Iteration Steps', user may follow instructions below;
Calculation → Iteration parameters → Max. Steps

3. Theory

This chapter consists of two sections. In the first section, considerable theoretical approaches and formulas are given. To complement this, the second section introduces models developed in this work, the practical side and visual diagrams of sub-models.

3.1. Single Sub-Models

Heat Balance of the system is as shown in figure 3.1

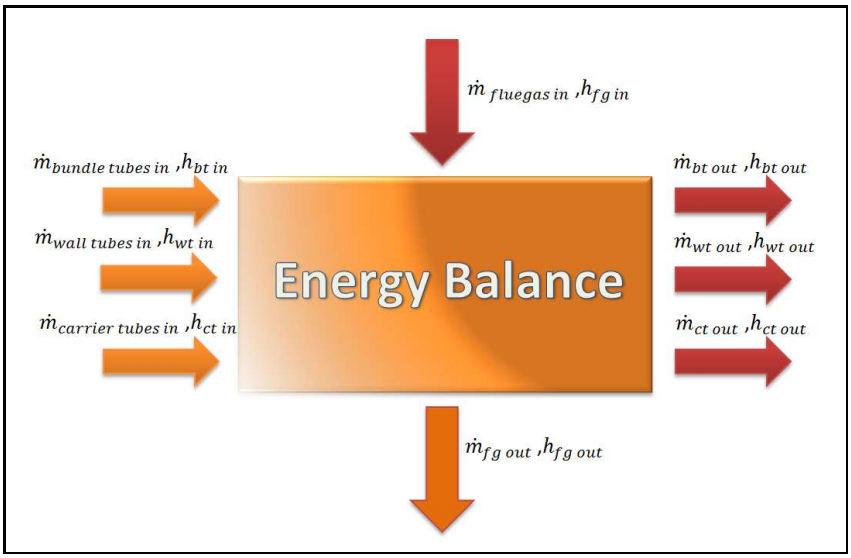


Figure 3.1.: Heat Balance

$$Q_{fluegas,tot} = Q_{fluegas,rad} + Q_{fluegas,conv} \tag{3.1}$$

Sum of heat transfer both by convection & radiation equals to the total heat transfer (from flue gas to the system). In the following subsections; significant approaches are mentioned.

3.1.1. Geometry

Inputs of geometric characteristics for each model are the outputs of 'Geometry' model. There is a bridge between two wall tubes as seen in figure 3.2. This bridge is welded on them to achieve a safe construction.

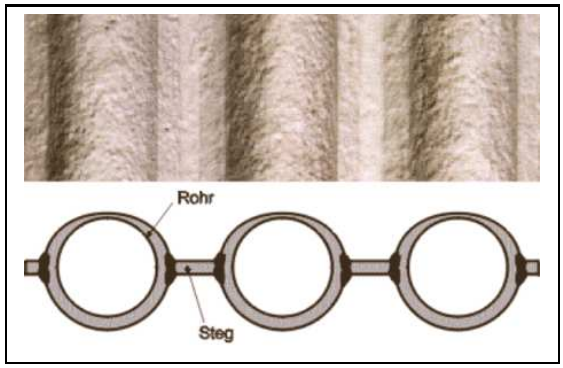


Figure 3.2.: One Side Heated Membrane Wall, Tube - Fin - Tube [6]

In Figure 3.3, geometric characteristics are represented by their symbols used to describe the design requirements of the features.

Assumptions of geometric characteristics for radiation calculations are;

-Surface area of carrier tubes, is included in surface area of bundle tubes. Both are considered together.

-Geometry of bundle is taken as cubic. 3.4

-Wall sides are presumed as plain surfaces.

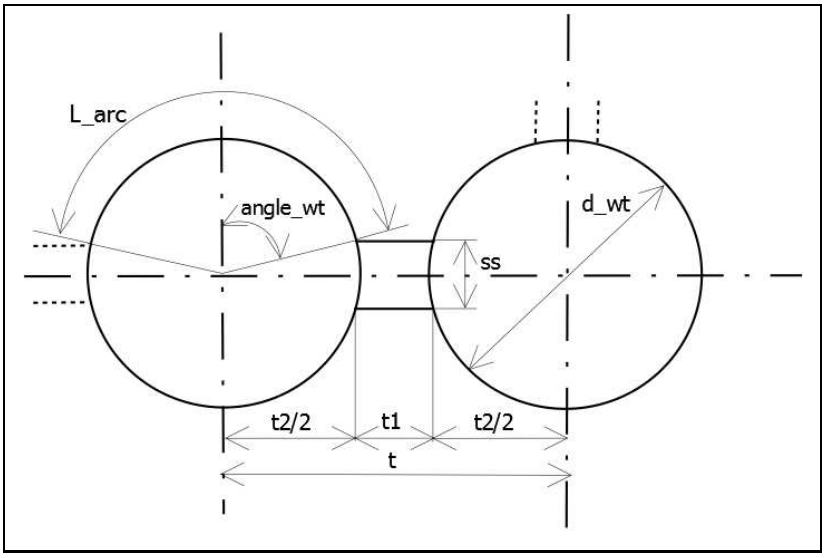


Figure 3.3.: Top-view of two connected wall tubes [1]

3.1.2. Convection

$$Q_{fluegas,conv} = Q_{gw,conv} + Q_{gc,conv} + Q_{gb,conv} + Q_{loss,conv} \tag{3.2}$$

Convective heat transfer is the sum of heat transfer from gas to bundle tubes (Eq.3.3 or Eq.3.4), -carrier tubes (Eq.3.5), -wall tubes (Eq.3.6) and the heat loss from system to the environment (Eq.3.7).

$$Q_{gb,conv} = k_{gb}.A_{bt}.LMTD_{gb}.f[1] \tag{3.3}$$

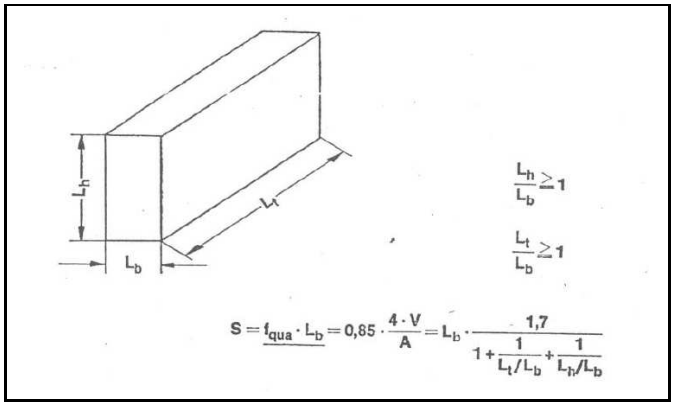


Figure 3.4.: Determination of equivalent layer thickness for cubic shapes [1]

$$Q_{gb,conv} = k_{gb}.a.L_{fin}.LMTD_{gb}[1] \tag{3.4}$$

If the bundle tubes are finned, then Eq. 3.4 is used instead of Eq. 3.3 .

$$Q_{gc,conv} = k_{gc}.A_{ct}.LMTD_{gc}.f[1] \tag{3.5}$$

$$Q_{gw,conv} = k_{gw}.A_{wt}.LMTD_{gw}.f[1] \tag{3.6}$$

$$Q_{loss,conv} = k_{we}.A_{wt}.LMTD_{we}.f[1] \tag{3.7}$$

System can be assumed as an adiabatic process which means there will be no leakage of heat energy to the environment ($k_{we} = 0 \rightarrow Q_{loss,conv} = 0$), or a non-adiabatic (without heat boundary) process which means heat transfer from wall surfaces to the environment must be considered 3.7.

Determination of k values;

$$\frac{1}{k_{gb}} = \frac{t_{bt}}{\lambda_{st,bt}} + \frac{1}{\alpha_{in,bt}} + \frac{1}{\alpha_{conv,gb}}[1] \tag{3.8}$$

$$\frac{1}{k_{gb}} = \frac{dG}{\alpha_{in,bt}.(d_{bt} - 2.t_{bt})} + \frac{dG}{2.\lambda_{fin}}.ln(\frac{d_{bt}}{d_{bt} - 2.t_{bt}}) + \frac{1}{\alpha_{conv,gb}}[1] \tag{3.9}$$

If the bundle tubes are finned, then 3.9 operates. Otherwise 3.8 is valid.

$$\frac{1}{k_{gc}} = \frac{1}{\alpha_{conv,gc}} + \frac{d_{ct}}{2.\lambda_{st,ct}}.ln(\frac{d_{ct}}{d_{ct} - 2.t_{ct}}) + \frac{d_{ct}}{\alpha_{in,ct}.(d_{ct} - 2.t_{ct})}[1] \tag{3.10}$$

$$\frac{1}{k_{gw}} = \frac{1}{\alpha_{conv,gw}} + \frac{d_{wt}}{2.\lambda_{st,wt}}.ln(\frac{d_{wt}}{d_{wt} - 2.t_{wt}}) + \frac{d_{wt}}{\alpha_{in,wt}.(d_wt - 2.t_{wt})}[1] \tag{3.11}$$

In case of non-adiabatic conditions, the information for walls of being coated or non-coated, must be defined in the model. 4.5.2.1 Eq. 3.12 operates when the walls doesn't have any coating. Eq. 3.13 operates when the walls are surrounded by one coating. Eq. 3.14 operates when the walls are surrounded by two coatings. Eq. 3.15 operates when the walls are surrounded by three coatings.

$$\frac{1}{k_{we}} = \frac{1}{\alpha_{conv,we}} + \frac{t_{wt}}{\lambda_{st,wt}} + \frac{1}{\alpha_{in,wt}}[1] \tag{3.12}$$

$$\frac{1}{k_{we}} = \frac{1}{\alpha_{conv,we}} + \frac{t_{wt}}{\lambda_{st,wt}} + \frac{t_{iso,1}}{\lambda_{iso,1}} + \frac{1}{\alpha_{in,wt}}[1] \tag{3.13}$$

$$\frac{1}{k_{we}} = \frac{1}{\alpha_{conv,we}} + \frac{t_{wt}}{\lambda_{st,wt}} + \frac{t_{iso,1}}{\lambda_{iso,1}} + \frac{t_{iso,2}}{\lambda_{iso,2}} + \frac{1}{\alpha_{in,wt}}[1] \tag{3.14}$$

$$\frac{1}{k_{we}} = \frac{1}{\alpha_{conv,we}} + \frac{t_{wt}}{\lambda_{st,wt}} + \frac{t_{iso,1}}{\lambda_{iso,1}} + \frac{t_{iso,2}}{\lambda_{iso,2}} + \frac{t_{iso,3}}{\lambda_{iso,3}} + \frac{1}{\alpha_{in,wt}}[1] \tag{3.15}$$

3.1.2.1. Dimensionless numbers

Reynolds Number;

$$Re = \frac{\rho.w.d}{\eta}[1] \tag{3.16}$$

Prandtl Number;

$$Pr = \frac{c_p.\eta}{\lambda}[1] \tag{3.17}$$

Nusselt Number;

$$Nu_{turb} = \frac{0.037.Re^{0.8}.Pr}{1 + 2.443.Re^{-0.1}.(Pr^{\frac{2}{3}} - 1)}[5] \tag{3.18}$$

$$Nu_{lam} = 0.664.\sqrt{Re}.\sqrt[3]{Pr}[5] \tag{3.19}$$

$$Nu = \sqrt{Nu_{lam}^2 + Nu_{turb}^2}[5] \tag{3.20}$$

The calculation of heat transfer coefficients inside tubes, are determined by following equations;

$$\alpha_{i\infty} = 0.02.\lambda^{0.58}.\eta^{-0.38}.c_p^{0.42}.\Phi^{0.8}.d^{-0.2}[1] \tag{3.21}$$

$$\alpha_i = f_1.\alpha_{i\infty}[1] \tag{3.22}$$

$$\alpha_i = f_1.f_2.\alpha_{i\infty}[1] \tag{3.23}$$

When Re $<$ 10000, equation 3.23, otherwise equation 3.22 is being used.

$$f_1 = 1 + (\frac{d}{L})^{\frac{2}{3}}[1] \tag{3.24}$$

$$f_2 = \frac{\alpha_{i\infty}(Hausen)}{\alpha_{i\infty}(Approximation)} = 1.85.(Re^{-0.05} - 180.Re^{-0.8})[1] \tag{3.25}$$

To calculate $\alpha_{conv,gb}$ for finned tubes, needed equations are utilized from Mr. Euler's Bachelor Project named 'Einbindung der Berechnungsvorschriften für Rippenrohre in ein Wärmetauschermodul' [3]

$$\alpha_{conv,gw} = \frac{Nu_{gw}.\lambda_{wt}}{L_{wt}}[12] \tag{3.26}$$

$$\alpha_{conv,we} = \frac{Nu_{we}.\lambda_{wt}}{L_{wt}}[12] \tag{3.27}$$

To determine k_ct, k_gb was multiplied by a presumed factor in the sample project of Mr. Plank [7]. Instead of doing that, some other approaches are used in this master's thesis. These are such as the determination of Thermal Diffusivity 3.29 and Rayleigh Number 3.28.

Determination of Nusselt number is different when there is a flow along the outside surface of a vertical tube;

$$Ra_{ct} = \frac{\beta.g.\Delta T.L^3}{\nu.\kappa}[11] \tag{3.28}$$

$$\kappa = \frac{\lambda}{\rho.c_p}[10] \tag{3.29}$$

$$\beta = \frac{1}{\rho}.\left(\frac{\partial \rho}{\partial T}\right)_p [11] \tag{3.30}$$

If the flue gas is an ideal gas, β can be approximated by $\frac{1}{T_*}$ [11]

For external flow $T_* = \frac{T_s + T_\infty}{2}$ [11]

Next step is the determination of Nusselt Number;

'The function $f_1(Pr)$ allows for the effect of the Prandtl number in the range 0.001 < Pr < ∞'; [12]

$$f_1(Pr) = [1 + (\frac{0.492}{Pr})^{9/16}]^{-16/9}[12] \tag{3.31}$$

'The average dimensionless heat transfer coefficient for laminar and turbulent flows near a vertical surface in the range from Ra = 10^{-1} to Ra = 10^{12} is defined by'; [12]

$$Nu_{plate,ct} = (0.825 + 0.387.[Ra.f_1(Pr)]^{1/6})^2[12] \tag{3.32}$$

$$Nu_{ct} = Nu_{plate,ct} + 0.97.\frac{L_{ct}}{d_{ct}}[12] \tag{3.33}$$

Thus, convective heat transfer coefficient for vertical pipes such as carrier tubes can be determined by the following equation;

$$\alpha_{conv,gc\perp} = \frac{Nu_{ct}.\lambda_{ct}}{L_{ct}}[12] \tag{3.34}$$

3.1.3. Radiation

' *Hottel and Egbert have suggested the following equations for the determination of the emissivity 3.35 and absorptance of mixtures 3.36 consisting of carbon dioxide, water vapor, and non-radiant components.*' [2]

$$\varepsilon_g = \varepsilon_{H_2O} + \varepsilon_{CO_2} - (\Delta\varepsilon)_g[2] \tag{3.35}$$

$$A_v = A_{vH_2O} + A_{vCO_2} - (\Delta\varepsilon)_w[2] \tag{3.36}$$

The way for radiation on wall tubes, is considered as three different directions. Owing to three dimension environment; first dimension is towards wall sides which are along x-axis, second one is towards wall sides which are along z-axis and the third one is along y-axis.

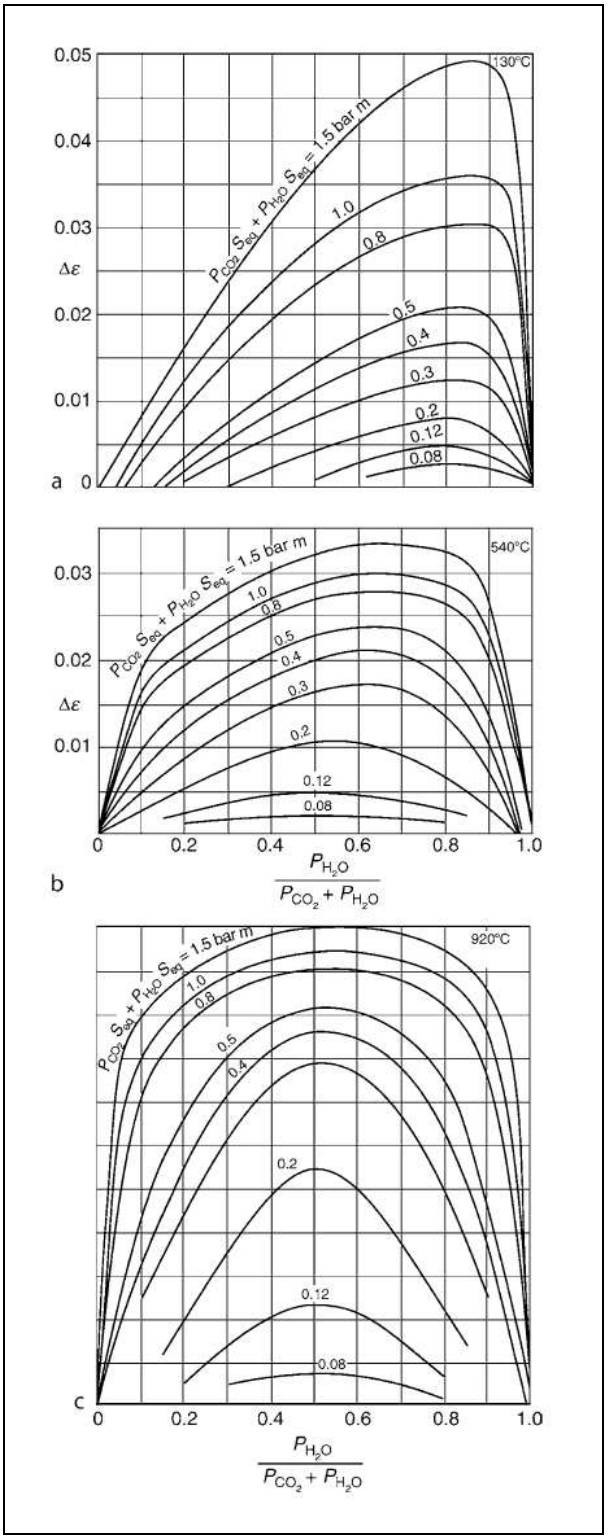


Figure 3.5.: Correction $\Delta\varepsilon$ for mixtures of CO_2 and H_2O at (a) 130 ℃, (b) 540 ℃, (c) 920 ℃ and above [2]

Q for the third one is neglected because it is already carried by flue gas from previous section to the next section.

$\Delta\varepsilon$ values are represented in figure 3.5 according to different temperatures.

Determination of a_i (needed in the calculation of ε_{CO_2}) is shown in Eq. 3.37.

$$a_i = b_{1,i} + b_{2,i}.\frac{T_g}{1000K} + b_{3,i}.(\frac{T_g}{1000K})^2[2] \tag{3.37}$$

i	b_{1i}	b_{2i}	b_{3i}	k_i (1/mbar)
1	0.1074	−0.10705	0.072727	0.036
2	0.027237	0.10127	−0.043773	0.3586
3	0.058438	−0.001208	0.0006558	3.06
4	0.019078	0.037609	−0.015424	14.76
5	0.056993	−0.025412	0.0026167	102.28
6	0.0028014	0.038826	−0.020198	770.6

Figure 3.6.: The values of the coefficients in Eq. 3.37 [2]

There are two methods in order to determine emissivities; Either graphically or analytically. Analytical calculation would be much more apposite for the radiation applications within a software program. Besides that would give more accurate results instead of graphical method. In this master's thesis, analytical method is taken into account.

'For computer-based calculations the total emissivities and absorptivities of real gases can be determined using the 'gray-and-clear gas approximation' (also called 'weighted sum of gray gases model'). For carbon dioxide CO_2 at a total gas pressure of $p = 1$ bar and a partial pressure of 0.01 mbar $< p_{CO_2}.s_{eq} <$ 10 mbar and at temperatures between 300 K $< T_g <$ 1800 K, we have'; Eq. 3.38 [2]

Calculation of ε_{CO_2} is shown in Eq. 3.38. To determine Z value, Eq. 3.39 is used.

$$\varepsilon_{CO_2} = Z - \sum_{i=1}^{6} a_i.exp(-k_i.p_{CO_2}.s_{eq})[2] \tag{3.38}$$

$$Z = c_1 + c_2.\frac{T_g}{1000K} + c_3.(\frac{T_g}{1000K})^2[2] \tag{3.39}$$

c1, c2, c3 values in Eq. 3.39 are respectively as 0.27769, 0.03869, $1.4249.10^{-5}$ given at [2]

'For water vapor H_2O at a total gas pressure of p = 1 bar, temperatures between 700 K $< T_g <$ 1500 K, and partial pressures between 0.05 mbar $< p_{H_2O}.s_{eq} <$ 0.5 mbar or 0.5 mbar $< p_{H_2O}.s_{eq} <$ 2 mbar we have'; Figure 3.7 [2]

'For a mixture of H_2O and CO_2 at a total pressure of p = 1 bar for p_{H_2O} / p_{CO_2} = 1, temperatures between 1100 K $< T_g <$ 1800 K, and equivalent layer thickness between 0.2 m $< s_{eq} <$ 6 m we have;' Figure 3.8 [2]

Calculation of ε_{H_2O} is shown in Eq. 3.40. To determine Z and a values, Eq. 3.41 and Eq. 3.42 are used.

$$\varepsilon_{H_2O} = Z - a.exp(-k.p_{H_2O}.s_{eq})[2] \tag{3.40}$$

$$Z = b_1 + b_2.\frac{T_g}{1000K}[2] \tag{3.41}$$

$$a = b_3 + b_4.\frac{T_g}{1000K}[2] \tag{3.42}$$

$p_{H_2O}s_{eq}$	0.05 . . . 0.5	0.5 . . . 2
b_1	0.43265	0.66439
b_2	−0.1089	−0.17389
b_3	0.3273	0.4572
b_4	−0.043821	−0.1317
k (1/mbar)	3.5829	0.84652

Figure 3.7.: The values of the coefficients in Eq. 3.40, 3.41 and 3.42 [2]

Calculation of $\varepsilon_{H_2O+CO_2}$ is shown in Eq. 3.43. To determine a_i value, Eq. 3.44 is used.

$$\varepsilon_{H_2O+CO_2} = \sum_{i=1}^{3} a_i.[1 - exp(-k_i.(p_{H_2O} + p_{CO_2}).s_{eq})][2] \tag{3.43}$$

$$a_i = b_{1,i} + b_{2,i}.\frac{T_g}{1000K}[2] \tag{3.44}$$

i	b_{1i}	b_{2i}	k_i (1/mbar)
1	0.130	0.265	0
2	0.595	−0.15	0.824
3	0.275	−0.115	25.91

Figure 3.8.: The values of the coefficients in Eq. 3.43 and 3.44 [2]

3.1.4. Pressure Loss

$$\Delta p = \zeta.\frac{\Phi^2.\nu}{2}[1] \tag{3.45}$$

3.1.4.1. Pressure Loss on the Gas Side

$$\zeta = 0.8.f_e.f_{sa}.f_z.Z[1] \tag{3.46}$$

f_{sa} (f_sa) & f_z (f_z) values are represented in figure 3.9 and 3.10. The equation which is shown in figure 3.9 is used in order to find f_z (f_z). f_e (f_e) value is also represented in figure 3.11.

Correction factor is often as '0.8' chosen for practical clean bundle tubes. In case of fouling, user is free to assign a bigger numerical data to fouling_factor depending on experience. [1]

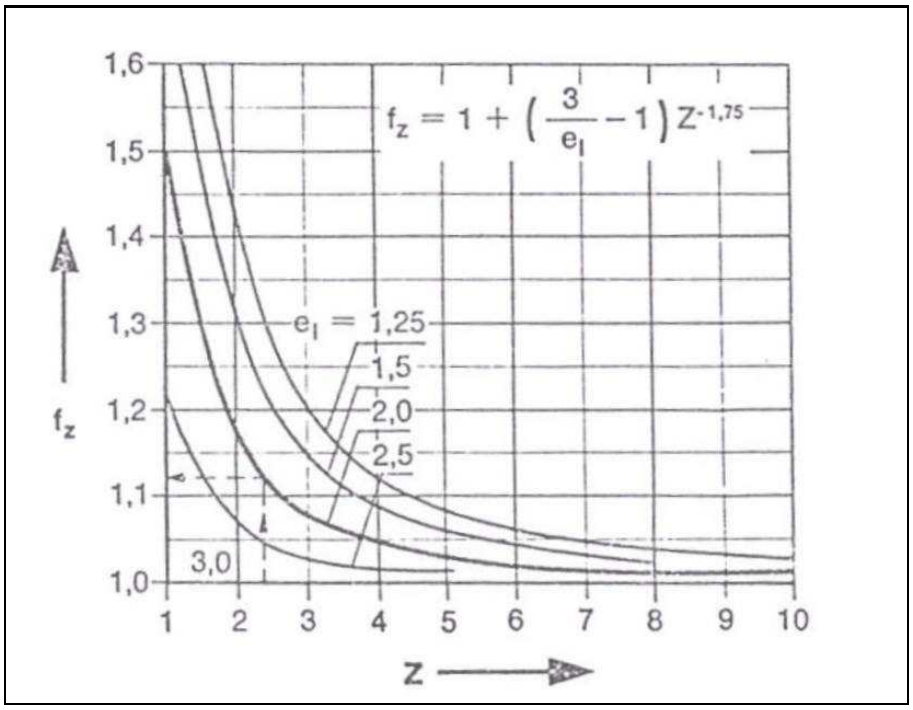


Figure 3.9.: Determination of f_z [1]

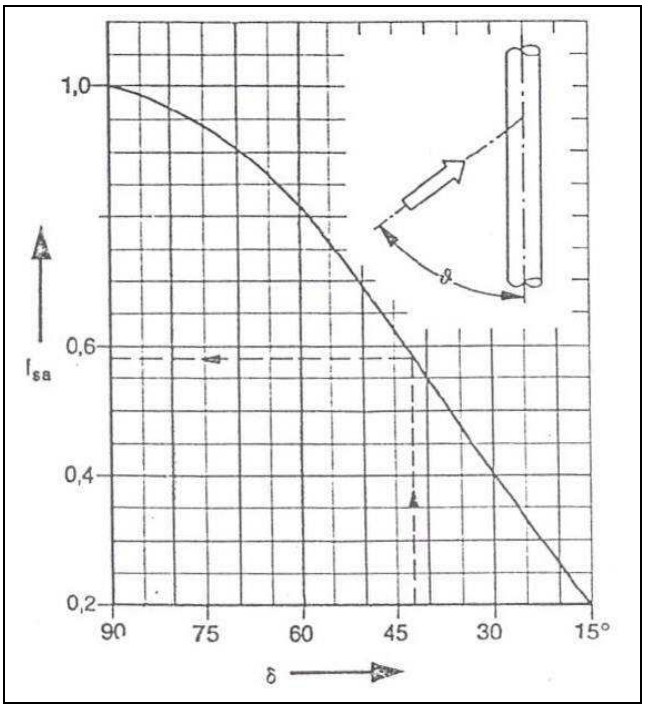


Figure 3.10.: Determination of f_{sa} for inclined flow [1]

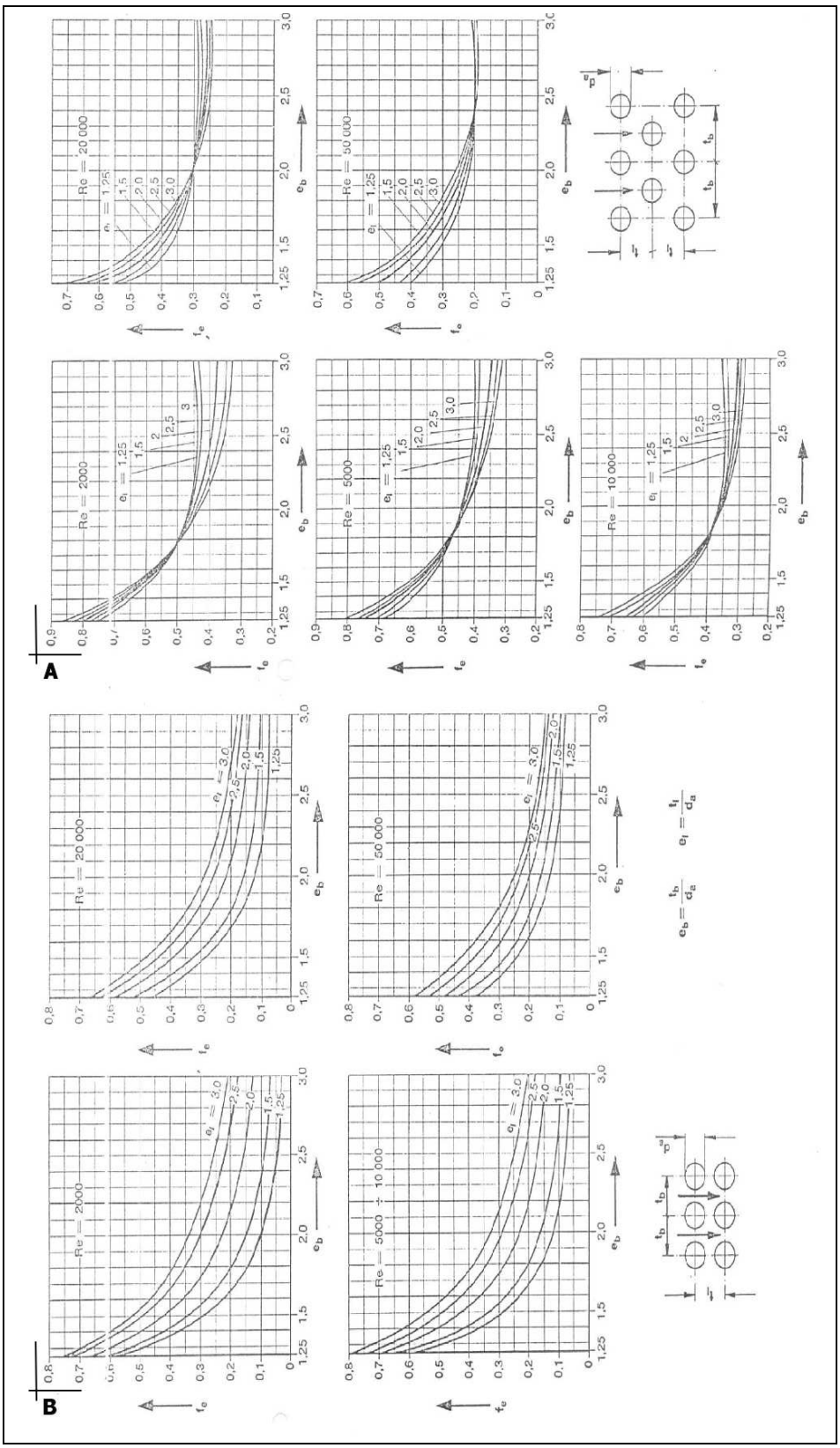


Figure 3.11.: Determination of f_e, A. Arrangement factor for staggered order B. Arrangement factor for in-line order [1]

Z in Eq. 3.46, is the number of bundle tubes towards Y-axis and its equivalent is n_bt_y in the model. If Z is bigger than 10 then f_z equals to f_z_. [1] Otherwise the equation in figure 3.9 is valid.

Φ, is the mass flow density, here based on the narrowest cross-section in the tube bundle.

3.1.4.2. Pressure Loss inside Tubes

$$\zeta_{tot} = \zeta_{re} + \zeta_{qu} + \zeta_{um} + \zeta_{vz} + \zeta_{ei}[1] \tag{3.47}$$

All the resistance coefficients seen in Eq. 3.47 affects the pressure loss inside tubes.
To determine each resistance coefficient, user may look at section 9.1.2 within FDBR Handbook [1].

3.1.5. Injection Cooling

In conventional steam power plants, injection cooling process is used for the control of steam temperature and the protection of downstream temperature-sensitive components. With this process, the temperature value of running steam should decrease in number. In order to achieve that, feed water which is colder than the flowing steam, must be injected into the system. A simple balance equation is illustrated below. 3.48

$$m_{w,out}.h_{w,out} - m_{ic}.h_{ic} = (m_{w,out} - m_{ic}).h_{w,in} \tag{3.48}$$

[1] User should assign a numerical data to f_z_ at first.

3.2. Models

3.2.1. Single Models

Figure 3.12 shows the 'Duct Section' model which includes wall tubes, carrier tubes and bundle tubes.

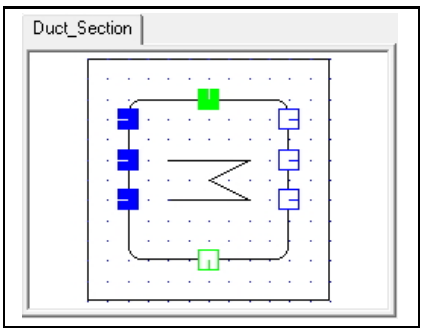


Figure 3.12.: 'Duct Section' Model

Figure 3.13 shows the 'Duct section without heat exchanger' model which includes wall tubes and carrier tubes. Model in figure 3.13 can be placed between heat exchangers ('Duct Section' Models) along the canal.

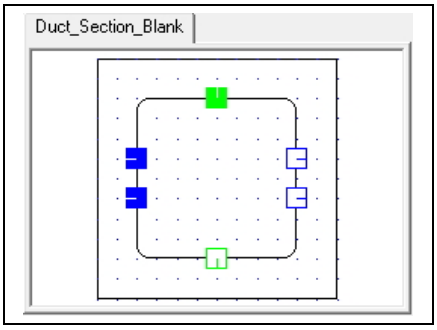


Figure 3.13.: 'Duct Section Without Heat Exchanger' Model

An economizer behaves such as a 'Duct Section' model as seen in figure 3.12. In case of a duct without wall & carrier tubes, an alternative 'Economizer' model (which is generated in this master's thesis) as seen in figure 3.14 can be included in process.

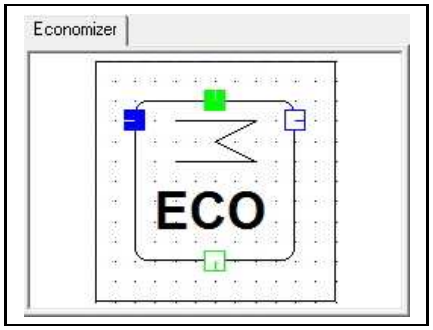


Figure 3.14.: 'Economizer' Model

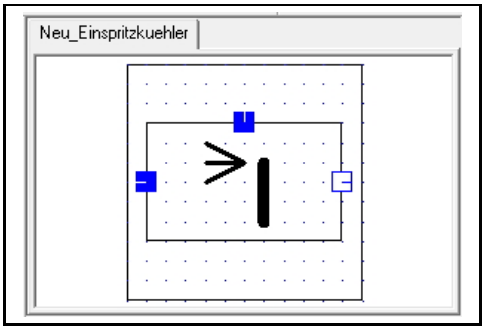


Figure 3.15.: 'Injection Cooler' Model

3.2.2. Combination of Models

In the following figures, single sub projects of Economizer, Injection Cooler, Blank & Duct_Section, Pressure Loss are shown in PSE environment.
Figure 3.16 represents the process flow sheet with an integrated 'Economizer' models with some results of the simulation.

A - Geometry Model,

B - Convection Model,

C - Radiation Model,

D - Model Pressure Loss along Canal,

E - Model Pressure Loss along Pipes of Superheater.

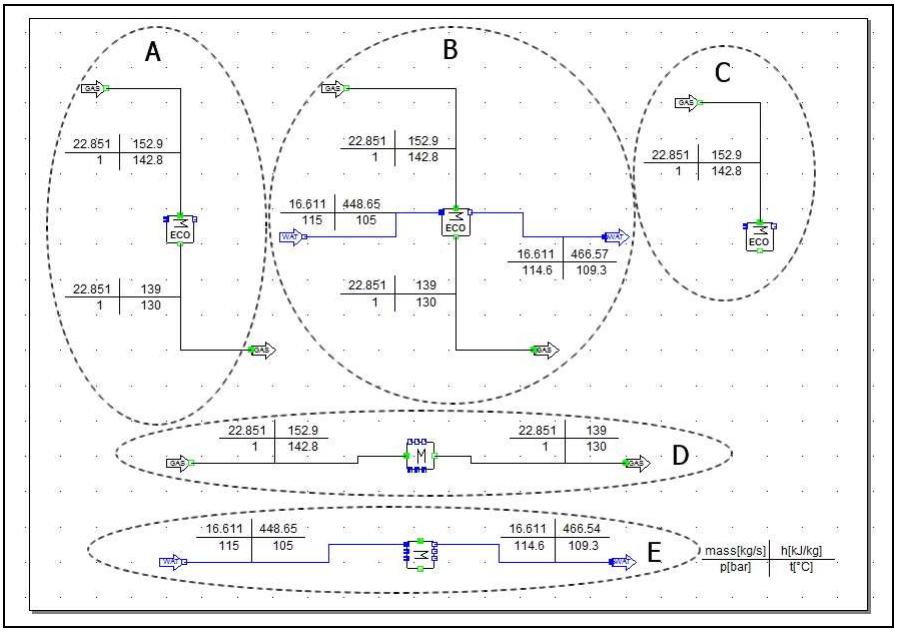


Figure 3.16.: Economizer-1

Figure 3.17 represents the process flow sheet with an integrated blank area models with some results of the simulation.

Figure 3.18 represents the process flow sheet with an integrated 'Superheater' models with some results of the simulation.

Figure 3.19 represents the process flow sheet with an integrated 'Pressure Loss' models with some results of the simulation.

Figure 3.20 & 3.21 represents simulation of the injection cooling system. In the simulation model, 3.20 is placed between Superheater-1 & Superheater-2 and 3.21 is placed between Superheater-2 & Superheater-3.

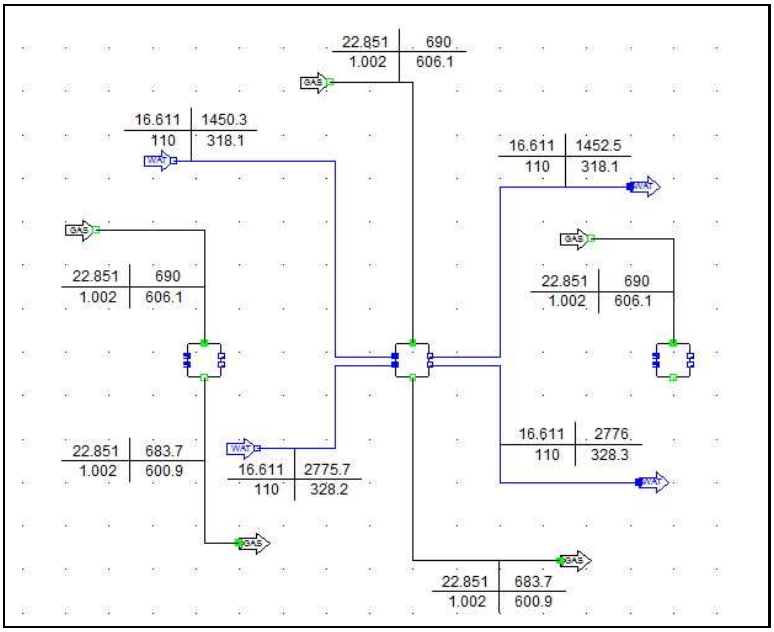


Figure 3.17.: Canal between Economizer-2 and Superheater-1

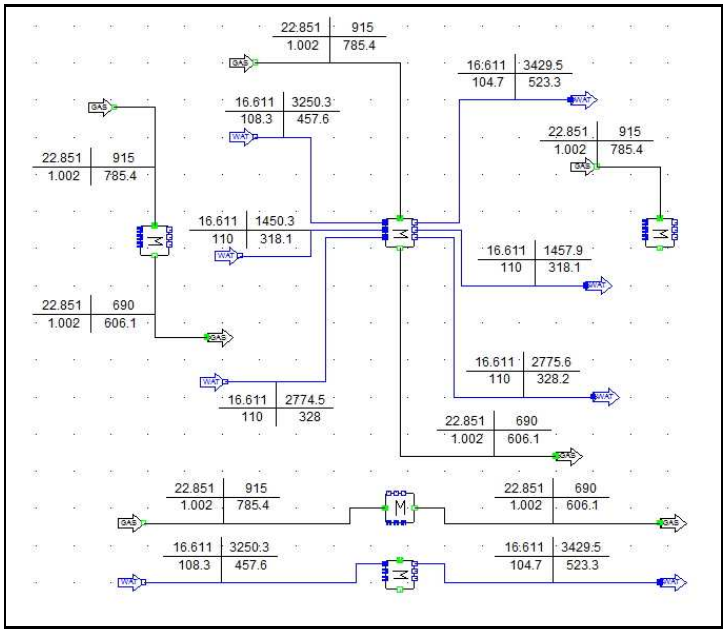


Figure 3.18.: The 'Duct Section' model used for Superheater-1

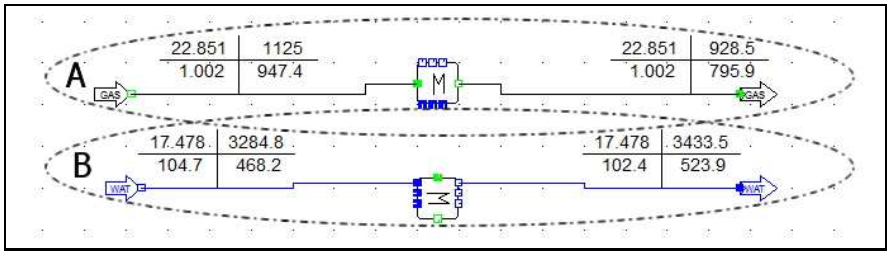


Figure 3.19.: A - Gas Side Pressure Loss of Superheater-2, B - Bundle Water Side Pressure Loss of Superheater-2

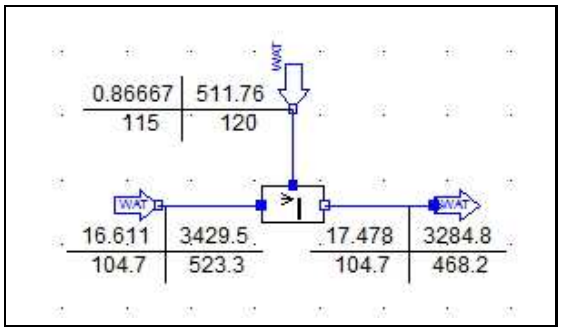


Figure 3.20.: Injection Cooler-1

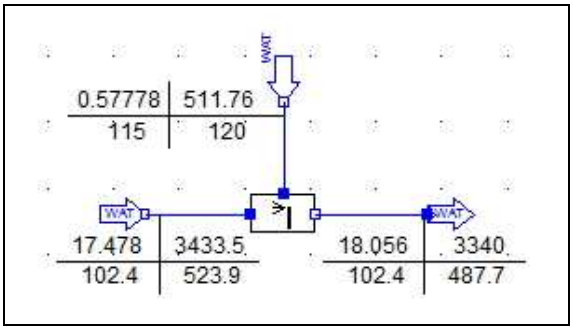


Figure 3.21.: Injection Cooler-2

Generation of an entire steam generator simulation is possible with combination of mentioned models.

4. Solution of the Problem

This chapter enlightens the reader about the structure of the developed model, the path that should be followed for the convergence method, the features used within the simulation software IPSEpro and the parameters considered for the simulated process.

4.1. Structure of the Developed Model

There are five sub-models within the main model 'Duct_Section'. These are:

- Geometry
- Convection
- Radiation
- Pressure_Loss_Gas_Side
- Pressure_Loss_Water_Side

During the development of model in MDK environment, it is frequently tested in PSE environment. Thus, potential errors are avoided.

4.2. Method of Approach

If two different equations are defined based on the same result, an error would occur. That is a conflict for the software. For this reason, two different results are defined as seen in Figure 4.1.

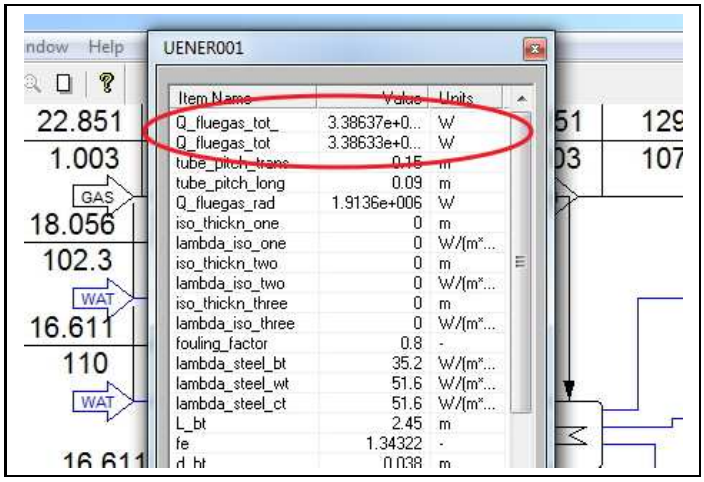


Figure 4.1.: Method of Approach

After the calculation is over, the user must check whether $Q_{fg\_}$ and Q_{fg} approach one another. If they don't approach each other, the user must follow instructions below;

1- Set needed variables & parameters ,

2- Set inlet and outlet temperatures, pressures , (Give realistic values which would be the starting values of iteration [1])

3- Click 'Run' button ,

4- User gets;

[1]The closest value which user estimates, would decrease the step of iteration in number.

$Q_{conv,fg}$, $Q_{conv,gb}$, $Q_{conv,gc}$, $Q_{conv,gw}$

&

$Q_{conv,fg\_}$, $Q_{conv,gb\_}$, $Q_{conv,gc\_}$, $Q_{conv,gw\_}$

&

Other results,

5- Q values and Q values end with _ (for instance: $Q_{conv,fg}$ and $Q_{conv,fg\_}$) must converge each other,

*Because;

$$Q\_ = \dot{m}.(h_{out} - h_{in})[1] \tag{4.1}$$

$$Q = k.A_s.LMTD.f[1] \tag{4.2}$$

Calculated Q value is significant for user to find the actual (outlet or inlet) temperature in Eq. 4.1 ,

6- Switch outlet or inlet temperatures, pressures (depends on which one you want to estimate) from 'set' to 'estimate' ,

7- Take (copy) Q results and set (paste) them to Q_ results ,

8- Click 'Run' button ,

9- Keep doing step '7' & '8' unless Q' and Q_' converge each other.

In this process, the following points have to be considered:

a) Keep setting new delta_p_fluegas and delta_p_bundle values which are calculated in 'Pressure Loss' sub-models. (delta_p_fluegas and delta_p_bundle are in connection [2] with 'Convection' sub-model)

b) Keep setting new Q_radiation value which is calculated in 'Radiation' sub-model. (Q_radiation is also in connection with 'Convection' sub-model)

c) Keep setting some new physical values which are calculated in 'Geometry' sub-model. (Physical values are also in connection with 'Convection' sub-model)

Not to do these actions by hand each time, free_equation feature 4.3 can be used.

The less Q values differ, the more estimations are accurate.

4.3. Steady-State Calculation

Figure 4.2 was captured during the calculation. There are totally 4658 variables on screen. It has to end up at 100%, if user wants to get results at each group.

4.4. Free Equations

'Free equations' mean independent equations in other words. They can easily be embedded into PSE environment. In this thesis, the purpose of its use is to connect models (inputs and outputs) each other. A screen shot of 'Free Equations' frame is seen in Figure 4.3.

[2] means; the model uses some outputs of other model as its input.

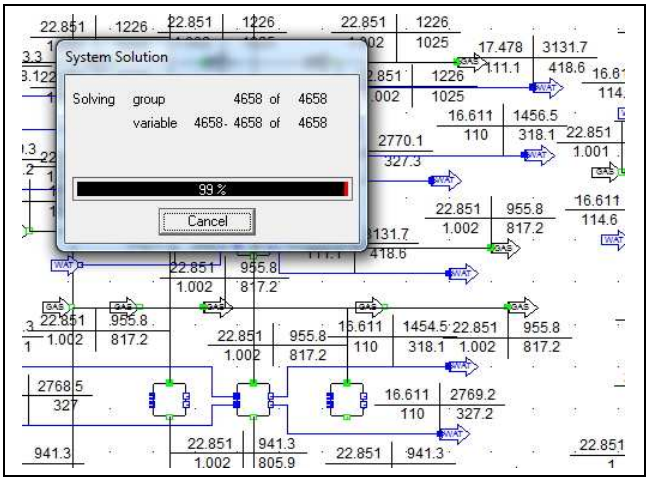


Figure 4.2.: Screen shot of a Steady-State Calculation

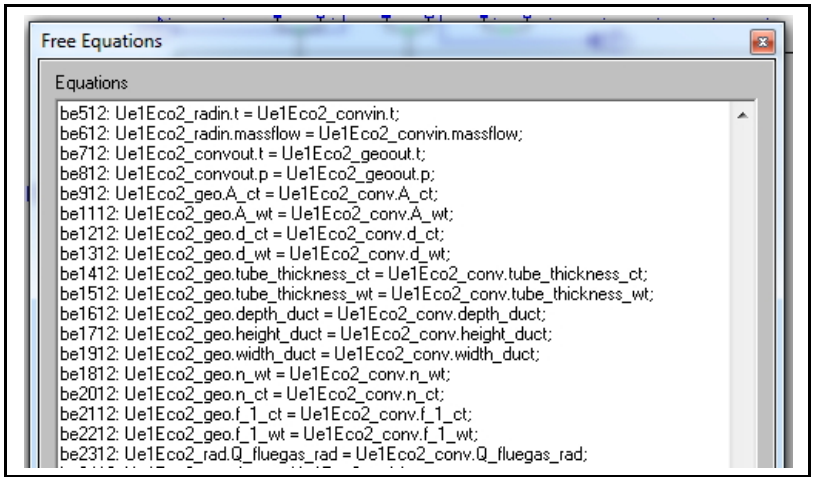


Figure 4.3.: Free Equations

4.5. Inputs

This section presents the input values given into the models.

4.5.1. Geometry

Before the calculation of physical properties, user must select some parameters as seen in figure 4.4.

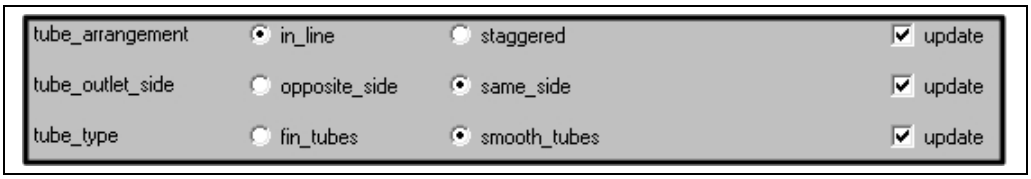


Figure 4.4.: Switch Buttons for Geometry

4.5.1.1. Tube Arrangement

The selecting of the parameter tube arrangement would affect the result of variable fe. The user must decide if the order of bundle tubes will be staggered or in_line.

4.5.1.2. Tube Outlet Side

According to front-view, if water enters heat exchanger from left side of the canal and exits from right side, then user must select 'opposite_side'. If both input and output is on the same side, then user must select 'same_side'. In case of same_side selection 's' variable in 'Geometry' Model would be 1, otherwise it would be 2. In conclusion, that would affect heat transfer surface area.

4.5.1.3. Tube Type

Here, user decides the type of bundle tubes. Therefore, this change would have an effect on efficiency of the process.

If the heat exchanger has finned [3] bundle tubes, then user must follow the instructions below;

- First step; Select 'fin_tubes'. 4.4
- Second step; Set 'tube_fin_number', 'fin_thickness' and 'fin_height' variables in 'Geometry' sub-model as shown in 4.5.
- Third step; Set lambda_fin, L_fin, 'fin_thickness', 'fin_height' variables & copy here 'a', 'a_r', 'a_ko', 'a_ro', 'dG' outputs which was determined in 'Geometry' sub-model, as shown in 4.6.

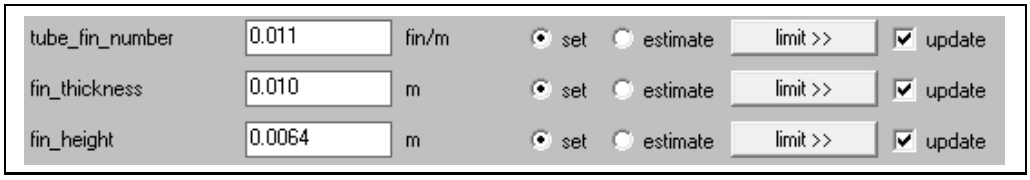


Figure 4.5.: Step 2 for finned tubes

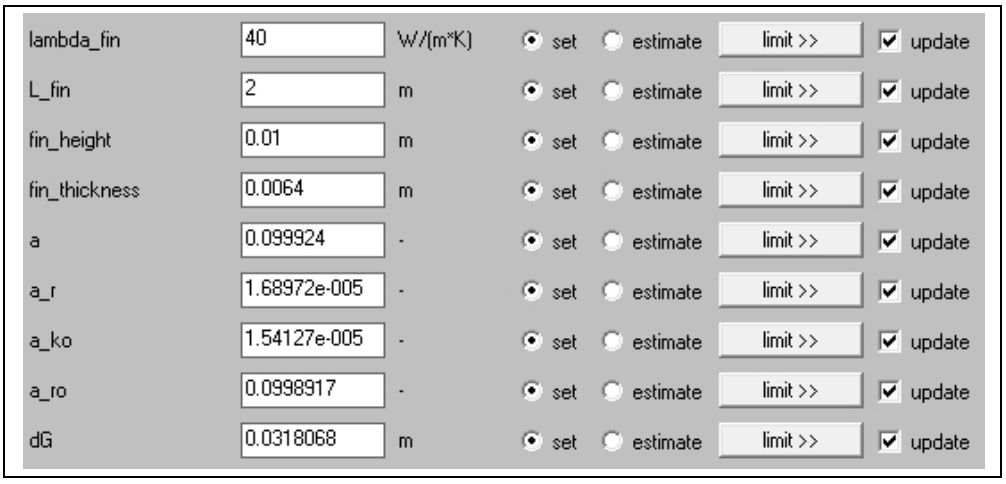


Figure 4.6.: Step 3 for finned tubes

[3] Source Codes for finned tubes was modified (for instance; some parameter and variable names are changed) based on Nikolaus Euler-Rolle's bachelor work [3]

4.5.2. Convection

Before the calculation of convective heat transfer, user has to select some parameters according to the physical property of structure. There are two common items of 'Geometry' sub-model & 'Convection' sub-model such as 'tube_type' and 'tube_arrangement'. That's why, they are not mentioned in this section again.

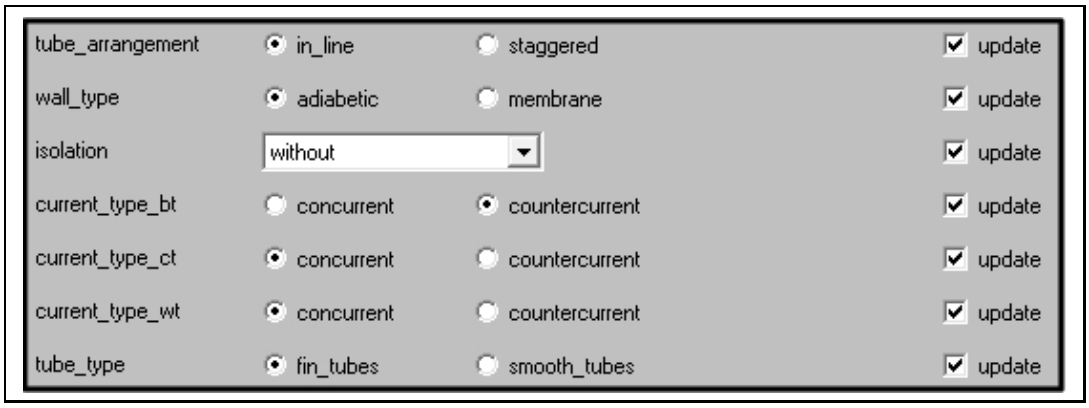


Figure 4.7.: Switch Buttons of 'Convection' sub-model

4.5.2.1. Wall Type

The system might be 'adiabatic' or 'membrane'. If user wants to ignore the heat energy loss from wall tubes to the ambient, then the selection must be 'adiabatic' button. In case of selecting 'membrane', the instructions are such as 4.5.2.2;

-In case of selecting 'ADIABATIC', it means;

There will be no energy loss from walls to the ambient. So Q_loss_conv would be '0'.

-In case of selecting 'MEMBRANE', it means;

There would be an energy loss from walls to the ambient. So Q_loss_conv will be calculated and get a numerical value. At the same time Q_fluegas_conv would increase in number. As a result, the convergence between Q_fluegas_conv and Q_fluegas_conv_ values would be broken. Instructions in section 'Method of Approach' must be followed again.

4.5.2.2. Isolation

For the purpose of the insulation, the designer might think of coating. So during the development of this model, this possibility is also taken into account.
As shown in 4.8, user is able to select coatings range from one to three;

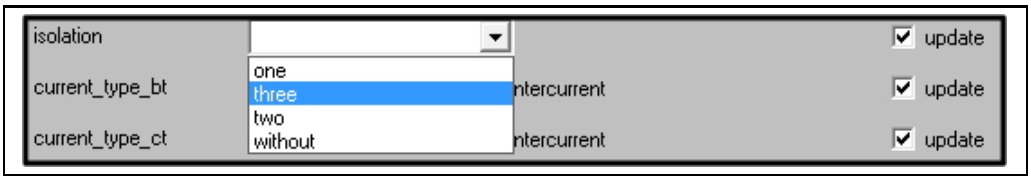


Figure 4.8.: Switch Buttons for Isolation

In case of selecting 'one', user must give a thickness and a thermal conductivity value of a Coating. In case of selecting 'two', user must give two thickness and two thermal conductivity values of two Coatings. In case of selecting 'three', user must give three thickness and three thermal conductivity values of three Coatings. In case of selecting 'without', user doesn't have to give any value. That means steam generator doesn't have any Coating. Other actions apart from these would give an error.

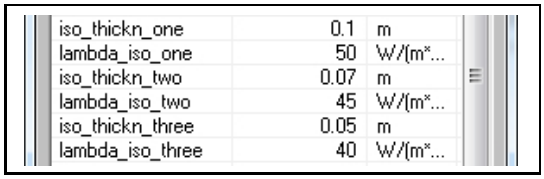


Figure 4.9.: Coating Thickness & Thermal Conductivity Inputs

4.5.2.3. Current Types

If water flows in the tubes and gas flows in the canal in opposite direction, the user must click 'countercurrent' button. But, if they are flowing in the same direction, then user must click 'concurrent' at the time of choosing 'current_type'.

4.5.3. Radiation

If the flue gas doesn't include CO_2 gas component, then user has to select 'water_vapor'. If the flue gas doesn't include H_2O gas component, then user has to select 'carbondioxide'. If the flue gas includes both CO_2 & H_2O gas components, then user has to select 'mixture' at the time of choosing 'fluegas'.

Figure 4.10.: Switch Buttons for Radiation

5. Test and Runs

This chapter presents the test results of an entire steam generator process and a duct with different bundles. The influence of coatings for exterior walls and tube types on system, is evaluated. Regarding steam generator process, simulation results are shown and validated with a practical example.

5.1. A Duct Section

Table 5.1 represents the effect of Coatings on the process simulation. Bundle tubes are considered as smooth tubes.

Table 5.2 also represents the effect of Coatings on the process simulation. Bundle tubes are considered as finned tubes.

In case of an adiabatic process, heat energy loss would be zero. Due this fact, less heat energy would be absorbed in the process. As a result, ΔT between $T_{fg,in}$ & $T_{fg,out}$ has the minimum value.

In case of a comparison between membrane walls (Whether they have a different number of coatings), ΔT between $T_{fg,in}$ & $T_{fg,out}$ has the minimum value when there are three coatings on the membrane wall. Because the heat energy loss is there at the minimum level.

	Adiabatic	**Without Coating**	**1 Coating**	**2 Coatings**	**3 Coatings**
Q_fluegas_tot (kW)	7300.20	7419.56	7333.58	7315.81	7305.52
Q_fluegas_tot_ (kW)	7308.54	7419.21	7333.77	7317.14	7305.79
Q_loss_conv (kW)	0	66.5819	20.7257	11.4492	5.7210
feed_fluegas.t (℃)	805.922	809.701	806.784	806.216	805.828
drain.fluegas.t (℃)	550.024	550.024	550.024	550.024	550.024
iso_thickn_one (m)	-	0	0.1	0.1	0.1
lambda_iso_one (m)	-	0	0.5	0.5	0.5
iso_thickn_two (m)	-	0	0	0.2	0.2
lambda_iso_two (m)	-	0	0	0.85	0.85
iso_thickn_three (m)	-	0	0	0	0.5
lambda_iso_three (m)	-	0	0	0	0.95

Table 5.1.: Test Results for a duct section with Smooth Bundle Tubes

	Adiabatic	**Without Coating**	**1 Coating**	**2 Coatings**	**3 Coatings**
Q_fluegas_tot (kW)	1088.06	1167.85	1112.78	1101.61	1094.75
Q_fluegas_tot_ (kW)	1088.93	1168.00	1112.88	1101.53	1094.75
Q_loss_conv (kW)	0	66.5819	20.7257	11.4492	5.72118
feed_fluegas.t (℃)	589.081	591.903	589.936	589.531	589.289
drain.fluegas.t (℃)	550.024	550.024	550.024	550.024	550.024

Table 5.2.: Test Results for a duct section with Finned Bundle Tubes

-Increase in Coating Number,

-Increase in Coating Thickness,

-Decrease in Thermal Conductivity of the Coating

would reduce heat energy loss. Thus, much more heat energy would remain inside the system boundary relatively.

When the tubes are finned, ΔT between $T_{fg,in}$ & $T_{fg,out}$ may remain at much less value in comparison to ΔT between $T_{fg,in}$ & $T_{fg,out}$ when the tubes are smooth.

5.2. Entire Steam Generator

In Figure 5.1, name of the models are mentioned such as;

A - Superheater - 3,

B - Blank between Superheater - 2 & - 3,

C - Superheater - 2,

D - Blank between Superheater - 1 & - 2,

E - Superheater - 1,

F - Blank between Economizer - 2 & Superheater - 1,

G - Economizer - 2,

H - Economizer - 1 Part 1,

I - Economizer - 1 Part 2,

J - Economizer - 1 Part 3.

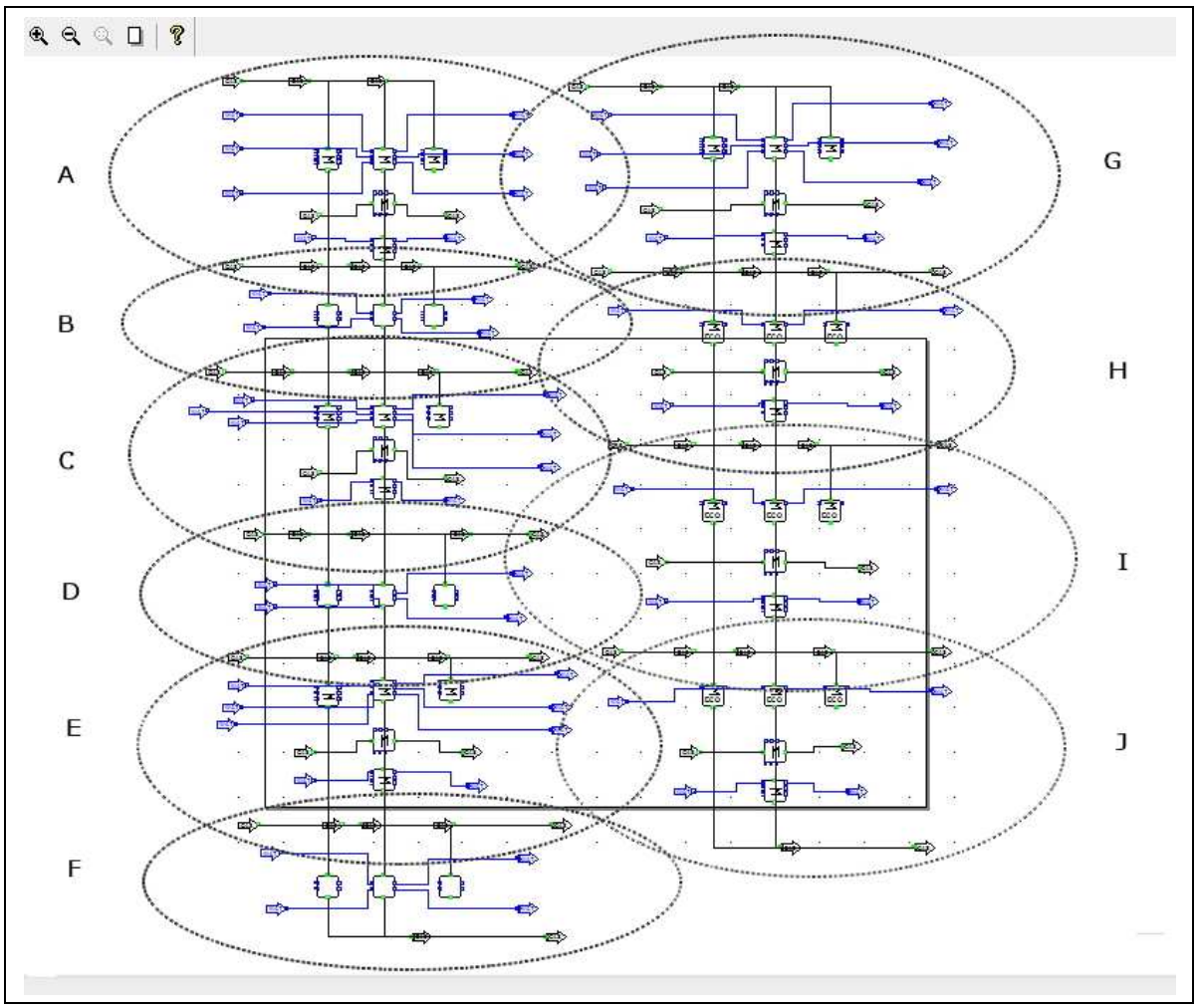


Figure 5.1.: Screen Shot of the Steam Generator Simulation

	Sample Results (°C)	**IPSEpro Results (°C)**
T_{in} SuperHeater 3	1074.534314	1074.534314
T_{out} SH 3	979.6676	964.356
T_{in} Blank SH 3 - SH 2	-	964.356
T_{out} Blank SH 3 - SH 2	-	947.386
T_{in} SH 2	979.6676	947.386
T_{out} SH 2	797.141	795.934
T_{in} Blank SH 2 - SH 1	-	795.934
T_{out} Blank SH 2 - SH 1	-	785.373
T_{in} SH 1	797.141	785.373
T_{out} SH 1	533.4975	606.065
T_{in} Blank SH 1 - Eco2	-	606.065
T_{out} Blank SH 1 - Eco2	-	600.926
T_{in} Eco 2	533.4975	600.926
T_{out} Eco 2	438.14	$460.751 > 183.955$
T_{in} Eco 1 Part 1	438.14	$460.751 > 183.955$
T_{out} Eco 1 Part 1	-	160.104
T_{in} Eco 1 Part 2	-	160.104
T_{out} Eco 1 Part 2	-	142.756
T_{in} Eco 1 Part 3	-	142.756
T_{out} Eco 1 Part 3	130	130

Table 5.3.: Test Results for Entire Steam Generator System

Table 5.3 represents the comparison of temperature results between the sample project and its simulation in IPSEpro.

The point is that the output temperature results differ from each other. One of the reasons that causes this situation is different assumptions. In the sample project; k_{bt} was multiplied by 0.6 in order to determine k_{ct},. Moreover, k_{bt} was multiplied by 0.4 in order to determine k_{wt}. In this master's thesis, mentioned coefficients (k_{ct} & k_{wt}) are determined by their specific equations (Look at Chapter 3). On the other hand, the modeled connection on the flue gas side between 'Superheater' and 'Economizer-2' is also taken into account in terms of radiation and convection calculations. This addition would also make slight changes in results. [1]

In sample project, main input values for the entire process are given in table A.5. These and also other values which doesn't exist in table A.5 are being set, in order to design an efficient steam generator. Subsequently, remaining values are calculated in the model.

Inlet temperature of 'Superheater-3' was already determined in sample project. This value is the outlet temperature of the combustion chamber [2]. So it is set in simulation program.

According to the sample project, $T_{fg,out}$ is 130 ℃, $T_{H_2O,in}$ is 105 ℃. So process calculation starts at 'Economizer-1 Part 3' model and proceed respectively to 'Economizer-1 Part 2', 'Economizer-1 Part 1', 'Economizer-2' models.

At the same time; live steam temperature is 515 ℃, live steam pressure is 100 bar according to the sample project and $T_{H_2O,in}$ of Superheater-3 already exists. Also process calculation starts at 'Superheater-3' model and proceed respectively to Injection Cooler 2, blank between 'Superheater-3 & -2', 'Superheater-2', 'Injection Cooler 1', blank between 'Superheater-2 & -1', 'Superheater-1', blank between 'Superheater-1' & 'Economizer-2', 'Economizer-2' models.

[1] The space between 'Economizer-2' & 'Economizer-1 Part 1' is neglected.

[2] This master's thesis does not include the calculation of combustion chamber.

As seen above, both process calculations end and intersect at 'Economizer-2' model. According to the results, the calculated $T_{fg,out}$ of 'Economizer-2' (460.751 ℃) is bigger than the calculated $T_{fg,in}$ of 'Economizer-1 Part 3' (183.955 ℃). Thereby this equation shows that there isn't any obstacle to leave set values the same. To benefit from this temperature difference, also surface area can be increased; For instance, an another economizer may be added or number of pipes may be increased in number. Any changes are possible, in order to improve the heat absorption within the boundary during process.

The pressure of drum was as 110 bar given in sample project. So the pressure of bundle tubes which leave drum, must be less than the pressure of drum. Besides the pressure of bundle tubes which enter drum, must be more bigger than the pressure of drum. After calculations, the pressure of incoming bundle tubes is as 113.891 bar (p_{out} of 'Economizer-2') calculated and the pressure of outgoing bundle tubes is as 108.334 bar (p_{in} of 'Superheater-1') calculated. Consequently, it is verified that there isn't any mistake in incoming & outgoing bundle tubes which are connected to the drum, in terms of pressure. The same situation is valid for wall and carrier tubes. [3]

During H_2O flow along wall & carrier tubes, small differences in temperature & pressure occur. Due to this fact, their pressure value is as same as the drum's pressure (110 bar) taken into account in this master's thesis.

[3]To see the installation of tubes, please look at Figure A.1

6. Conclusion

This chapter is the conclusions section. The aim of this master's thesis is briefly mentioned, literature survey is given. Advices and recommendations are given to engineers or researchers who want to study this subject further.

In general, the aim of this master's thesis is ;

-To demonstrate the calculations of steam generator in terms of radiation and convection.
-To show the development process of a simulation model.
-To point out how simulation work is achieved within a specific software.

Concerning theory; the equations used are based on VDI Heat Atlas 2010 [2], [5], [10], [11], [12] and FDBR Handbook 1980 [1]. Regarding source codes of finned tubes, I benefit from Nikolaus Euler-Rolle's bachelor work [3]. To determine gas properties, a *.dll [1] file which is Dipl. Ing. Dr. Techn. Thomas Gröbl's work, was embedded into the model.

Engineers or researchers who want to start learning IPSEpro, related documentations such as [8], [9] that comes with software CD, are advised. Previous theses upon this subject can also be read. This software allows user to make steady state calculations. It

[1]Dynamic Link Library - is a library that contains codes & datas which can be used by more than one program at the same time.

may not satisfy researchers in some respects. So they can try other dynamic simulation software such as ones which are previously mentioned in introduction chapter 1.

Researchers can go further on radiation subject. For instance; radiation on carrier tubes. There isn't so many researches related to this subject. In this project work, surface area of carrier tubes and bundle tubes are evaluated together.

In conclusion, this work is achieved as a master's thesis in Technical University of Vienna. IPSEpro library of TU Vienna is much more improved. Hopefully, it will be a useful literature for B.Sc., M.Sc., Ph.D. candidates studying in energy field. At the same time, it has benefits for steam generator designers, engineers or researchers who work on process simulation.

Bibliography

[1] *FDBR-Handbuch Wärme- und Strömungstechnik*. Fachverband Dampfkessel-, Behälter- und Rohrleitungsbau, 1980. – Ringbuch, ca. 800 Seiten

[2] Dieter Vortmeyer ; Stephan Kabelac ; Chemieingenieurwesen, VDI-Gesellschaft V. (Hrsg.): *VDI HEAT ATLAS*. Second Edition. Springer - Verlag Berlin Heidelberg, 2010. – 979 – 988 S.

[3] Euler-Rolle, Nikolaus: *Einbindung der Berechnungsvorschriften für Rippenrohre in ein Wärmetauschermodul*. Wurmstraße 15, 4020 Linz, Technische Universität Wien, Bachelor's Thesis, December 2009

[4] GAFFERT, G.A.: *Steam Power Stations ... Second Edition. [With Plans.]*. McGraw-Hill Book Company, 1940 `http://books.google.at/books?id=SkP2MgEACAAJ`

[5] Gnielinski, Volker ; Chemieingenieurwesen, VDI-Gesellschaft V. (Hrsg.): *VDI HEAT ATLAS*. Second Edition. Springer - Verlag Berlin Heidelberg, 2010. – 713–715 S.

[6] K.-J. Matthes ; Thomas Kohler ; Sören Heitz: Eine neuartige vollmechanisierte Schweißanlage für die Sanierung von Heizflächen in Kraftwerks- und Müllverbrennungsanlagen / DH Schweißtechnologie & Service Hohenthurm GmbH & Co. KG, Technischen Universität Chemnitz. `http://www.qucosa.de/fileadmin/data/qucosa/documents/4256/data/index.html`. – Forschungsbericht. – Bild 1-1

[7] Kurt, Plank: *Konstruktionsübungen aus Dampferzeugerbau*. Wien, 1995

[8] SimTech: *MDK Documentation*. 1991-2011

[9] SimTech: *PSE Documentation*. 1991-2011

[10] Stephan, Peter ; Chemieingenieurwesen, VDI-Gesellschaft V. (Hrsg.): *VDI HEAT ATLAS*. Second Edition. Springer - Verlag Berlin Heidelberg, 2010. – 17–29 S.

[11] Thess, Andre´ ; Chemieingenieurwesen, VDI-Gesellschaft V. (Hrsg.): *VDI HEAT ATLAS*. Second Edition. Springer - Verlag Berlin Heidelberg, 2010. – 663 – 665 S.

[12] Werner Kast ; Herbert Klan ; Chemieingenieurwesen, VDI-Gesellschaft V. (Hrsg.): *VDI HEAT ATLAS*. Second Edition. Springer - Verlag Berlin Heidelberg, 2010. – 667 – 672 S. – Revised by Andre´ Thess

A. Appendix

A.1. Nomenclature

Nomenclature

Table A.1.: Nomenclature

Symbol	**Nomenclature**
d	Diameter
f	Fouling factor
g	Acceleration of gravity
h	Height or Enthalpy
k	Overall heat transfer coefficient
p	Pressure
s	Layer Thickness
t	Tube Thickness
w	Speed
A	Area
L	Characteristic length
Q	Heat transfer
T	Temperature
V	Volume
Nu	Nusselt Number
Pr	Prandtl Number
Ra	Rayleigh Number

Continued on next page

Table A.1 – continued from previous page

Symbol	**Nomenclature**
LMTD	Logarithmic mean temperature difference
A_ν	Absorptance of mixtures
α	Heat transfer coefficient
β	Isobaric volume expansion coefficient
ε	Emissivity
κ	Thermal diffusivity
ξ	Resistance Coefficient
λ	Thermal conductivity
ν	Kinematic viscosity
ρ	Density
ΔT	Change in Temperature
Δp	Change in Pressure
c_p	Constant pressure specific heat.
T_*	Reference temperature
$\dot{m}$	Mass flow per second
Φ	Mass flow density

A.2. Indications

Indications

Table A.2.: Indications

Symbol	**Indication**
g	gas
s	surface
w	water
bt	bundle tubes
ct	carrier tubes
ei	Components
eq	equivalent
fg	flue gas
ic	injection cooler
in	inlet
qu	Change in cross-section
re	Friction
st	steel
um	Change in flow-direction
vz	Branching
wt	wall tubes
iso	isolation

Continued on next page

Table A.2 – continued from previous page

Symbol	**Indication**
lam	laminar
out	outlet
qua	cuboidal
rad	radiation
tot	total
conv	convection
plate	vertical plate
turb	turbulent
*	reference
∞	ambient
$\perp$	vertical

A.3. Definition of Items

Definition of Items

Table A.3.: Definitions and Units for Items

Item	Unit	Definition
GLOBAL		
Ambient	-	Geographical and meteorological conditions
SWITCH		
current_type_bt	-	Current direction along bundle tubes has to be chosen as concurrent or countercurrent
current_type_ct	-	Current direction along carrier tubes has to be chosen as concurrent or countercurrent
current_type_wt	-	Current direction along wall tubes has to be chosen as concurrent or countercurrent
fluegas	-	Choose if flue gas contains CO_2 or H_2O or both of them
isolation	-	Choose if there is 1 or 2 or 3 or no Coating for exterior walls

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
tube_arrangement	-	Choose the arrangement of tubes if they are in_line or staggered
tube_outlet_side	-	The side which heat exchanger's tube exits duct (relative to its entry side) has to be chosen as opposite_side or same_side
tube_type	-	Tube type has to be chosen as fin_tubes or smooth_tubes
wall_type	-	Wall type has to be chosen as adiabatic or membrane
FUNCTIONS		
dynViscosity	$\mu Pa.s$	Dynamic Viscosity for Gas
f_cp_pt	kJ/(kg.K)	Isobaric Heat Capacity for Water from Pressure[bar] and Temperature[°C]
f_eta_pt	$\mu Pa.s$	Dynamic Viscosity for Water from Pressure[bar] and Temperature[°C]
f_lambda_pt	W/(m.K)	Thermal Conductivity for Water from Pressure[bar] and Temperature[°C]
heatCapacity	kJ/(kg.K)	Heat Capacity for Gas
heatConductivity	W/(m.K)	Heat Conductivity for Gas
PARAMETERS		
d_emissivity_(wx,wy,wz,b)	-	Correction for mixtures of CO_2 and H_2O 3.5
fouling_factor	-	Fouling Factor
radiation_coefficient	W/($m^2.K^4$)	Radiation Coefficient

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
VARIABLES		
a1-a6	-	Calculated with taken values from 3.6 to determine g_tot, gw_totx, gw_toty, gw_totz
am_tube	m	Logaritmic Mean Value of a Bundle Tube's Inner & Outer Perimeter
angle_wt	^{o}	Illustrated in figure 3.3
alpha_air	$W/(m^2.K)$	Heat Transfer Coefficient for Convection of Air
alpha_in_bt	$W/(m^2.K)$	Heat Transfer Coefficient for Convection inside bundle tubes
alpha_in_ct	$W/(m^2.K)$	Heat Transfer Coefficient for Convection inside carrier tubes
alpha_in_wt	$W/(m^2.K)$	Heat Transfer Coefficient for Convection inside wall tubes
alpha_conv_gb	$W/(m^2.K)$	Heat Transfer Coefficient for Convection from gas to bundle tubes
alpha_conv_gc	$W/(m^2.K)$	Heat Transfer Coefficient for Convection from gas to carrier tubes
alpha_conv_gw	$W/(m^2.K)$	Heat Transfer Coefficient for Convection from gas to wall tubes
alpha_conv_we	$W/(m^2.K)$	Heat Transfer Coefficient for Convection from wall tubes to environment
alpha_conv_fin	$W/(m^2.K)$	Heat Transfer Coefficient for Convection of Fins

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
alpha_rad_gw_x	$W/(m^2.K)$	Heat Transfer Coefficient for Radiation from gas to Wall Tubes towards X-Direction
alpha_rad_gw_y	$W/(m^2.K)$	Heat Transfer Coefficient for Radiation from gas to Wall Tubes towards Y-Direction
alpha_rad_gw_z	$W/(m^2.K)$	Heat Transfer Coefficient for Radiation from gas to Wall Tubes towards Z-Direction
alpha_rad_gb_gc	$W/(m^2.K)$	Heat Transfer Coefficient for Radiation from gas to Bundle & Carrier Tubes
a_r, a_ro, a_ko ,a	-	Variables for fins to determine dG
A1	m/m	tube_pitch_trans / d_bt
A_bt	m^2	Heat transfer surface area of Bundle tubes
A_ct	m^2	Heat transfer surface area of Carrier tubes
A_wt	m^2	Heat transfer surface area of Wall tubes
A_proj	m^2	Projected area of Bundle tubes along the Duct
A_v_b	-	Absorptance of Flue Gas (Final Result) towards Bundle Tubes
A_v_x	-	Absorptance of Flue Gas (Final Result) towards X-Direction

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
A_v_y	-	Absorptance of Flue Gas (Final Result) towards Y-Direction
A_v_z	-	Absorptance of Flue Gas (Final Result) towards Z-Direction
A_v_carbondioxide_b	-	Absorptance of CO_2 towards Bundle Tubes
A_v_carbondioxide_x	-	Absorptance of CO_2 towards X-Direction
A_v_carbondioxide_y	-	Absorptance of CO_2 towards Y-Direction
A_v_carbondioxide_z	-	Absorptance of CO_2 towards Z-Direction
A_v_water_vapor_b	-	Absorptance of H_2O towards Bundle Tubes
A_v_water_vapor_x	-	Absorptance of H_2O towards X-Direction
A_v_water_vapor_y	-	Absorptance of H_2O towards Y-Direction
A_v_water_vapor_z	-	Absorptance of H_2O towards Z-Direction
A, B	-	Calculated variables to determine fpco2_(x,y,z,b)
B1	m/m	tube_pitch_long / d_bt
cp_air	J/(kg.K)	Specific Heat Capacity of Air
cp_fluegas	J/(kg.K)	Specific Heat Capacity of Flue gas

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
cp_water_bt	J/(kg.K)	Specific Heat Capacity of Water inside Bundle Tubes
cp_water_ct	J/(kg.K)	Specific Heat Capacity of Water inside Carrier Tubes
cp_water_wt	J/(kg.K)	Specific Heat Capacity of Water inside Wall Tubes
C	-	A constant used to determine Nusselt number for finned tubes (for In_line 0.3, for Staggered 0.45)
C1	m/m	$\sqrt{(A1/2)^2 + B1^2}$
density_air	kg/m^3	Density of Air
density_fluegas	kg/m^3	Density of Flue gas
density_water_bt	kg/m^3	Density of Water in Bundle Tubes
density_water_ct	kg/m^3	Density of Water in Carrier Tubes
density_water_wt	kg/m^3	Density of Water in Wall Tubes
dyn_visc_air	Pa.s	Dynamic Viscosity of Air
dyn_visc_water_bt	Pa.s	Dynamic Viscosity of Water inside Bundle Tubes
dyn_visc_water_ct	Pa.s	Dynamic Viscosity of Water inside Carrier Tubes
dyn_visc_water_wt	Pa.s	Dynamic Viscosity of Water inside Wall Tubes
dyn_visc_fluegas	Pa.s	Dynamic Viscosity of Fluegas
d_bt	m	Outer diameter of a Bundle Tube
d_ct	m	Outer diameter of a Carrier Tube

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
d_wt	m	Outer diameter of a Wall Tube
d_G	m	Outer diameter of a finned tube
delta_p_fluegas	bar	Pressure Drop of Flue Gas along the Flue gas Canal
delta_p_bundle	bar	Pressure Drop of Water along Bundle Tubes
delta_p_wt	bar	Pressure Drop of Water along Wall Tubes
delta_p_ct	bar	Pressure Drop of Water along Carrier Tubes
depth_bundle	m	Depth of the Bundle
depth_duct	m	Depth of the Canal
dt_in_gb	℃	Temperature Difference between Gas & Water in Bundle Tubes [1]
dt_in_gc	℃	Temperature Difference between Gas & Water in Carrier Tubes
dt_in_gw	℃	Temperature Difference between Gas & Water in Wall Tubes
dt_in_we	℃	Temperature Difference between Air & Water in Wall Tubes
dt_log_gb	K	Logarithmic Temperature Difference between Gas & Water in Bundle Tubes
dt_log_gc	K	Logarithmic Temperature Difference between Gas & Water in Carrier Tubes

Continued on next page

[1] dt_in & dt_out items vary due to Switch current_type

Table A.3 – continued from previous page

Item	Unit	Definition
dt_log_gw	K	Logarithmic Temperature Difference between Gas & Water in Wall Tubes
dt_log_we	K	Logarithmic Temperature Difference between Air & Water in Wall Tubes
dt_out_gb	℃	Temperature Difference between Gas & Water in Bundle Tubes
dt_out_gc	℃	Temperature Difference between Gas & Water in Carrier Tubes
dt_out_gw	℃	Temperature Difference between Gas & Water in Wall Tubes
dt_out_we	℃	Temperature Difference between Air & Water in Wall Tubes
emiss_wall	-	Wall Emissivity
emiss_H2O_b	-	Emissivity of H_2O towards Bundle Tubes
emiss_H2O_x	-	Emissivity of H_2O towards X-Direction
emiss_H2O_y	-	Emissivity of H_2O towards Y-Direction
emiss_H2O_z	-	Emissivity of H_2O towards Z-Direction
emissivity_co2_b	-	Emissivity of CO_2 towards Bundle Tubes
emissivity_co2_x	-	Emissivity of CO_2 towards X-Direction
emissivity_co2_y	-	Emissivity of CO_2 towards Y-Direction
emissivity_co2_z	-	Emissivity of CO_2 towards Z-Direction
emissivity_gas_b	-	Emissivity of Flue Gas (Final Result) towards Bundle Tubes

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
emissivity_gas_mix_b	-	Emissivity of Gas Mixture towards Bundle Tubes
emissivity_gas_x	-	Emissivity of Flue Gas (Final Result) towards X-Direction
emissivity_gas_y	-	Emissivity of Flue Gas (Final Result) towards Y-Direction
emissivity_gas_z	-	Emissivity of Flue Gas (Final Result) towards Z-Direction
f1	-	A Factor calculated to determine fe_
fe	-	Calculated to Determine Tube Arrangement Factor
fG	-	Material Values Correction Factor
fq	-	A factor calculated to determine phi_fluegas in the presence of fins
f1pr_ct	-	The effect of the Prandtl number
fpco2_x, fpco2_y, fpco2_z, fpco2_b	-	Pressure Correction Factors
fin_efficiency	-	Efficiency of the fins
fin_height	m	Height of a Fin
fin_pitch	m	1/tube_fin_number
fin_thickness	m	Thickness of a Fin
fe_ & f_e	-	Tube Arrangement Factor
f_1, f_3	-	Factors calculated to determine fin_efficiency
f_1_bt, f_1_wt, f_1_ct	-	Length Correction Factor

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
f_2_bt, f_2_wt, f_2_ct	-	Correction Factors
f_sa	-	Factor for Oblique Flow
f_z	-	Correction Factor for Tube Rows
f_z_	-	Set value, if n_bt_y $>$ 10 then f_z = f_z_
gw_totx	-	Nominated as the formula which is subtracted from Z to find emiss_co2_x Eq. 3.38
gw_toty	-	Nominated as the formula which is subtracted from Z to find emiss_co2_y Eq. 3.38
gw_totz	-	Nominated as the formula which is subtracted from Z to find emiss_co2_z Eq. 3.38
g_tot	-	Nominated as the formula which is subtracted from Z to find emiss_co2_b Eq. 3.38
height_bundle	m	Height of the Bundle
height_duct	m	Height of the Canal
iso_thickn_one	m	Thickness of first layer which surrounds the wall tubes
iso_thickn_two	m	Thickness of second layer
iso_thickn_three	m	Thickness of third layer
k1-k6	-	These unknowns are nominated as exp(-$k_i.p_{co2}.s_{eq}$), look at Eq. 3.38

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
kw1x-kw6x	-	k1-k6 towards x-axis (s_eq_wt_x entered instead of s_eq_b)
kw1y-kw6y	-	k1-k6 towards y-axis (s_eq_wt_y entered instead of s_eq_b)
kw1z-kw6z	-	k1-k6 towards z-axis (s_eq_wt_z entered instead of s_eq_b)
ksi	-	Resistance Coefficient for Pressure Loss at Gas side
ksi_45, ksi_90, ksi_180	-	Resistance Coefficient for the Direction Changes according to Angles
ksi_collector	-	Resistance Coefficient for Collector
ksi_combining	-	Resistance Coefficient for Combining
ksi_friction	-	Resistance Coefficient for the Friction
ksi_roughness	-	Resistance Coefficient for the Roughness
ksi_tot	-	Total Resistance Coefficient for Pressure Loss at Water side
k_gb	W/(m^2.K)	Overall Convective Heat Transfer Coefficient from gas to bundle tubes
k_gc	W/(m^2.K)	Overall Convective Heat Transfer Coefficient from gas to carrier tubes
k_gw	W/(m^2.K)	Overall Convective Heat Transfer Coefficient from gas to wall tubes
k_we	W/(m^2.K)	Overall Convective Heat Transfer Coefficient from wall tubes to environment

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
l	m	Characteristic Length
lambda_air	W/(m.K)	Thermal Conductivity of Air
lambda_fin	W/(m.K)	Thermal Conductivity of Fins
lambda_fluegas	W/(m.K)	Thermal Conductivity of Flue gas
lambda_steel_bt	W/(m.K)	Thermal Conductivity of bundle tubes
lambda_steel_ct	W/(m.K)	Thermal Conductivity of carrier tubes
lambda_steel_wt	W/(m.K)	Thermal Conductivity of wall tubes
lambda_water_bt	W/(m.K)	Thermal Conductivity of Water in Bundle Tubes
lambda_water_ct	W/(m.K)	Thermal Conductivity of Water in Carrier Tubes
lambda_water_wt	W/(m.K)	Thermal Conductivity of Water in Wall Tubes
lambda_iso_one	W/(m.K)	Thermal Conductivity of first layer
lambda_iso_two	W/(m.K)	Thermal Conductivity of second layer
lambda_iso_three	W/(m.K)	Thermal Conductivity of third layer
L_bt	m	Length of a Bundle Tube
L_ct	m	Length of a Carrier Tube
L_wt	m	Length of a Wall Tube
L_fin	m	Length of finned part for a tube
L_bt_tot	m	Total Length of Bundle Tubes
L_ct_tot	m	Total Length of Carrier Tubes
L_wt_tot	m	Total Length of Wall Tubes
n_45, n_90, n_180	-	Number of the Direction Changes

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
n_bt	-	Total number of Bundle tubes
n_bt_y	-	Total number of Bundle tubes along y axis
n_bt_z	-	Total number of Bundle tubes along z axis
n_collector	-	Number for Collector
n_combining	-	Number for Combining
n_ct	-	Total number of Carrier tubes
n_ct_x	-	Total number of Carrier tubes along x axis
n_ct_z	-	Total number of Carrier tubes along z axis
n_wt	-	Total number of Wall tubes
n_wt_x	-	Total number of Wall tubes along x axis
n_wt_z	-	Total number of Wall tubes along z axis
Nu_we_0	-	Nusselt Number for Gas Side Flow along outside Walls
Nu_we_lam	-	Nusselt Number for Laminar Flow to find Nu_we_0
Nu_we_turb	-	Nusselt Number for Turbulent Flow to find Nu_we_0
Nu_wt_0	-	Nusselt Number for Gas Side Flow along Wall Tubes
Nu_wt_lam	-	Nusselt Number for Laminar Flow to find Nu_wt_0

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
Nu_wt_turb	-	Nusselt Number for Turbulent Flow to find Nu_wt_0
Nus_ct_out	-	Nusselt Number for Gas Side Flow along Carrier Tubes
phi_fluegas	kg/(m^2.sec)	Mass Flow Density
porosity	-	Porosity at Bundle Tubes
pm_fluegas	bar	Mean Pressure of Flue gas
pm_water_bt	bar	Mean Pressure of Water Inside Bundle Tubes
pm_water_ct	bar	Mean Pressure of Water Inside Carrier Tubes
pm_water_wt	bar	Mean Pressure of Water Inside Wall Tubes
pr_air	-	Prandtl Number of Air
pr_fluegas	-	Prandtl Number of Flue gas
ps_x, ps_y, ps_z, ps_b	bar.m	p_H2O.s_eq_wt_(x,y,z,b respectively)
p_H2O	bar	Partial Pressure of H_2O
p_CO2	bar	Partial Pressure of CO_2
Q_fluegas_conv	W	Total Convective Heat Transfer from flue gas to the system
Q_fluegas_rad	W	Total Radiative Heat Transfer from flue gas to the system
Q_fluegas_tot_	W	Total Heat Transfer obtained from the formula 4.1

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
Q_fluegas_tot	W	Total Heat Transfer obtained from the formula 4.2
Q_gb_conv	W	Convective Heat Transfer from flue gas to Bundle Tubes
Q_gc_conv	W	Convective Heat Transfer from flue gas to Carrier Tubes
Q_gw_conv	W	Convective Heat Transfer from flue gas to Wall Tubes
Q_gb_conv_	W	$m_{bt}.(h_{bt,2} - h_{bt,1})$
Q_gc_conv_	W	$m_{ct}.(h_{ct,2} - h_{ct,1})$
Q_gw_conv_	W	$m_{wt}.(h_{wt,2} - h_{wt,1})$
Q_gb_gc_rad	W	Radiative Heat Transfer from flue gas to Bundle & Carrier Tubes
Q_gw_rad_x	W	Radiative Heat Transfer from flue gas to Wall Tubes towards X-Direction
Q_gw_rad_y	W	Radiative Heat Transfer from flue gas to Wall Tubes towards Y-Direction
Q_gw_rad_z	W	Radiative Heat Transfer from flue gas to Wall Tubes towards Z-Direction
Q_loss_conv	W	Convective Heat Transfer from Wall Tubes to Environment
roughness	-	Roughness Value
Ra_ct	-	Rayleigh Number for the outer surface of Carrier Tubes

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
Re_bt_in	-	Reynolds Number for Water Side Flow along Bundle Tubes
Re_ct_in	-	Reynolds Number for Water Side Flow along Carrier Tubes
Re_wt_in	-	Reynolds Number for Water Side Flow along Wall Tubes
Re_wt	-	Reynolds Number found to determine alpha_conv_gw
Re_we	-	Reynolds Number found to determine alpha_conv_we
ss	m	Thickness of the welded bridge that connects two Wall tubes each other
s_eq_b	m	Layer thickness of Bundle and Carrier tubes
s_eq_wt_x	m	Layer thickness of Wall tubes on y-z plane
s_eq_wt_y	m	Layer thickness of Wall tubes on x-z plane
s_eq_wt_z	m	Layer thickness of Wall tubes on x-y plane
t1, t2, t	m	Illustrated in figure 3.3
temp_bt_surface	℃	Surface Temperature of Bundle Tubes (Gas Side)
temp_ct_surface	℃	Surface Temperature of Carrier Tubes (Gas Side)

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
temp_we_surface	℃	Surface Temperature of Wall Tubes (Air Side)
temp_wt_surface	℃	Surface Temperature of Wall Tubes (Gas Side)
therm_diff_ct	m^2/sec	Thermal Diffusivity for the outer surface of Carrier Tubes
tm_fluegas	℃	Mean Temperature of Flue gas
tm_water_bt	℃	Mean Temperature of Water Inside Bundle Tubes
tm_water_wt	℃	Mean Temperature of Water Inside Wall Tubes
tm_water_ct	℃	Mean Temperature of Water Inside Carrier Tubes
tube_fin_number	-/m	Number of Fins per Meter
tube_pitch_trans	m	Tube pitch transversely
tube_pitch_long	m	Tube pitch longitudinally
tube_thickness_bt	m	Tube Thickness of a Bundle Tube
tube_thickness_wt	m	Tube Thickness of a Wall Tube
tube_thickness_ct	m	Tube Thickness of a Carrier Tube
vm_bt	m^3/kg	Mean Kinematic Viscosity of Water inside Bundle Tubes
vm_ct	m^3/kg	Mean Kinematic Viscosity of Water inside Carrier Tubes
vm_fluegas	m^3/kg	Mean Kinematic Viscosity of Flue gas

Continued on next page

Table A.3 – continued from previous page

Item	Unit	Definition
vm_wt	m^3/kg	Mean Kinematic Viscosity of Water inside Wall Tubes
V_duct	m^3	Volume of the Canal
V_tubes	m^3	Volume of the Bundle & Carrier tubes
width_bundle	m	Width of the Bundle
width_duct	m	Width of the Canal
w_air	m/sec	Velocity of Air outside Walls
w_bt	m/sec	Velocity of Water along inside Bundle Tubes
w_ct	m/sec	Velocity of Water along inside Carrier Tubes
w_fluegas	m/sec	Velocity of Flue gas
w_wt	m/sec	Velocity of Water along inside Wall Tubes
Z	-	A determined value to reach emissivity_co2_b

A.4. Samples

A.4.1. Sample 1 - Duct Section

Table A.4 represents sample datas which are taken as a reference for 'A Duct Section' simulation. [2]

Table A.4.: Sample-1 Datas [7]

Name	Data
tube_outlet_side	opposite side
tube_arrangement	in_line
current_type_bt	countercurrent
current_type_wt	concurrent
current_type_ct	concurrent
t	0.057 m
ss	0 m
tube_fin_number	0.011 fin/m
fin_thickness	0.01 m
fin_height	0.0064 m
tube_pitch_trans	0.075 m
tube_pitch_long	0.09 m
fouling_factor	0.8
lambda_steel_bt	35.2 W/(m*K)
lambda_steel_wt	51.6 W/(m*K)
lambda_steel_ct	51.6 W/(m*K)
L_bt	2.45 m
L_fin	2 m
d_bt	0.0318 m

Continued on next page

[2]Regarding smooth tubes, fin values are zero in table A.4.

Table A.4 – continued from previous page

Name	Data
d_wt	0.057 m
d_ct	0.03 m
tube_thickness_bt	0.0036 m
tube_thickness_wt	0.0045 m
tube_thickness_ct	0.0045 m
n_bt	1134
n_wt	246
n_ct	54
lambda_fin	40 W/(m*K)
m_{fg_in}	22.851 kg/s
m_{bt_in}	16.6111 kg/s
m_{wt_in}	16.6111 kg/s
m_{ct_in}	16.6111 kg/s
T_{fg_in}	estimated
T_{fg_out}	550.024 ℃
T_{bt_in}	327.584 ℃
T_{bt_out}	estimated
T_{wt_in}	318.081 ℃
T_{wt_out}	estimated
T_{ct_in}	326.855 ℃
T_{ct_out}	estimated
p_{fg_in}	estimated
p_{fg_out}	1.00159 bar
p_{bt_in}	115 bar

Continued on next page

Table A.4 – continued from previous page

Name	Data
p_{bt_out}	estimated
p_{wt_in}	110 bar
p_{wt_out}	estimated
p_{ct_in}	110 bar
p_{ct_out}	estimated

A.4.2. Sample 2 - Entire System

Table A.5 represents sample datas which are taken as a reference for 'Entire Steam Generator' simulation.

Name	**Data**
Boiler	Two Path
Boiler Type	Natural Circulation
Boiler Design	Self-Supporting
Fuel	Natural Gas
Fresh Steam Quantitative	65 $\frac{t}{h}$
Fresh Steam Pressure	100 bar
Fresh Steam Temperature	515 ℃
Temperature Control	2 Injection Coolers
Injection Cooler Inlet Temperature	120 ℃
Feed Water Inlet Temperature	105 ℃
Ambient Temperature	20 ℃
Flue gas outlet temperature	130 ℃
Minimum Calorific Value	49500 $\frac{kJ}{kg}$

Table A.5.: Sample-2 Datas [7]

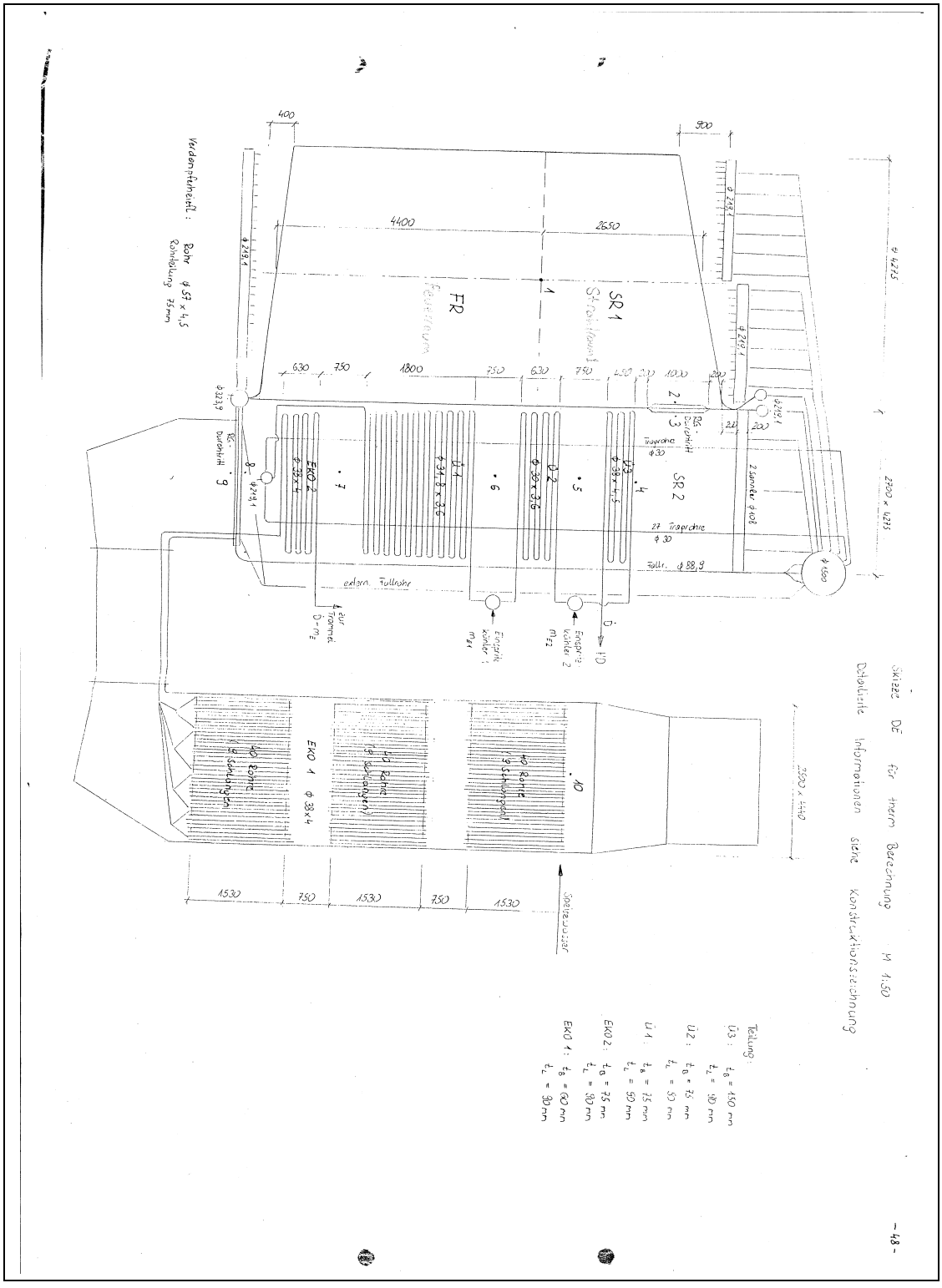


Figure A.1.: Installation of the entire system [7]

A.5. Source Code

UENER - GEOMETRY

```
########################
#    BASIC EQUATIONS    #
########################

er42:   feed_fluegas.massflow  = drain_fluegas.massflow;

########################
#          DUCT        #
########################

rt:       L_wt          =       height_duct;
zetz:     L_ct          =       height_duct;
n_wth:    n_wt          =       2*n_wt_x+2*(n_wt_z-2);
depth_duc4t:  depth_duct      =       (n_wt_z-1)*(d_wt+t1);
width_duct:    width_duct     =       (n_wt_x-1)*(d_wt+t1);
height_ductb: height_duct     =       height_bundle;

########################
#       BUNDLE TUBES   #
########################

width_bundleg: width_bundle   =  L_bt;
heighjs:       height_bundle  = (n_bt_y-1)*tube_pitch_long;
deptgsf:       depth_bundle   = tube_pitch_trans*n_bt_z;
A_btb:     A_bt = (L_bt*n_bt_y+(L_bt-
s)*tube_pitch_long*0.5*3.14159265359)*n_bt_z * (d_bt *
3.14159265359) ;
Lbtto5t:   L_bt_tot    = (L_bt*n_bt_y+(L_bt-
s)*tube_pitch_long*0.5*3.14159265359)*n_bt_z;
nbtsf:     n_bt        = n_bt_y*n_bt_z;
h11:               A1 * d_bt    = tube_pitch_trans;
h12a:              B1 * d_bt    = tube_pitch_long;
p4e:               C1           = sqrt( sqr(A1/2) + sqr(B1));
am_tube:           am_tube      = ((d_bt *  3.14159265359)-((d_bt-
2*tube_thickness_bt) * 3.14159265359)) / ln((d_bt *  3.14159265359)
/(d_bt-2*tube_thickness_bt) * 3.14159265359);
# FDBR  6.1.2 - 2/4

ifl     tube_outlet_side == same_side        then
hsdf5:  s = 1;
endifl

ifl     tube_outlet_side == opposite_side    then
hsdf6:  s = 2;
endifl

########################
#     CARRIER TUBES    #
########################

L_ct_tot:      L_ct_tot       =       L_ct*n_ct;
n_ct:   n_ct                  =       n_ct_x*n_ct_z;
A_ct:   A_ct                  =       3.14159265359*d_ct*L_ct*n_ct;

########################
#        WALL TUBES    ## FDBR 196 - 2
########################

tert:   asin(ss/d_wt)         =       90-angle_wt;
tog:    t1                    =       t - t2;
t82:    t2                    =       sqrt( sqr(d_wt) - sqr(ss) );

A_wtjhk:       A_wt           =       2*(depth_duct+width_duct)*L_wt;
L_wt_tot:      L_wt_tot       =       n_wt * L_wt;


# In case of finned bundle tubes

ifl tube_type == fin_tubes    then

gdc:     dG             =       a/3.14159265359;
gdgg4:  fins_pitch      =       1/tube_fin_number;
gg6:    phi_fluegas     =       feed_fluegas.massflow /
(width_duct*depth_duct*fq-A_proj);

endifl

ifl tube_type == smooth_tubes then

gdc2:          dG                     = d_bt;
gg5:           phi_fluegas            =feed_fluegas.massflow /
(depth_duct*width_duct-A_proj);
b4f:           tube_fin_number        = 0;
b3c:           fins_pitch             = 0;
b3e:           fin_height             = 0;
b4e:           fin_thickness  = 0;

endifl

hdh8:   fq = 1-
(d_bt+2*tube_fin_number*fin_height*fin_thickness)/tube_pitch_trans;#
FDBR 7.2.2 - 18/4
gg7:    a_r     =
2*3.14159265359*tube_fin_number*fin_height*(d_bt+fin_height);
# FDBR 7.2.2 - 18/9
gg8:    a_ko    =
3.14159265359*tube_fin_number*fin_thickness*(d_bt+2*fin_height);
```

```
# FDBR 7.2.2 - 18/10
gg9:    a_ro    = 3.14159265359*tube_fin_number*d_bt*(fins_pitch-
fin_thickness);
# FDBR 7.2.2 - 18/11
gg20:   a       = a_r+a_ko+a_ro;
# FDBR 7.2.2 - 18/1

# Determination of f_1_(bt,wt,ct)

f_1_bt: f_1_bt = 1+((d_bt-2*tube_thickness_bt)/L_bt)^(2/3);
f_1_wt: f_1_wt = 1+((d_wt-2*tube_thickness_wt)/L_wt)^(2/3);
f_1_ct: f_1_ct = 1+((d_ct-2*tube_thickness_ct)/L_ct)^(2/3);

gee:    porosity        =       (depth_duct*width_duct*height_bundle-
3.14159265359*0.25*d_bt^2*L_bt_tot)/(depth_duct*width_duct*height_bu
ndle);

h14:            l       = 3.14159265359*0.5*d_bt;

f14:            f1      = 4/3.14159265359*A1*porosity;

# Determination of fe FDBR 7.2.2 - 9/3 & 9/4

ifl tube_arrangement == in_line then

g5h:    fe_     =       1+(1.9-1.8/B1)*(1/(f1-0.4));

endifl

ifl tube_arrangement == staggered then

gdf5:   if B1<5 && f1>=3.04 then   fe_= 1+1.53/(f1-0.4);

        else fe_= 1+(1.87-1.7/B1)*(1/(f1-0.4));

endifl

fe4:    fe      =       fe_*porosity^-0.6;


# Projected Areas and Volumes for Radiation
#-----------------------------------------------------------------

V_duct: V_duct  =       height_duct*depth_duct*width_duct;
V_tubes:        V_tubes         =       L_bt_tot*       3.14159265359 *
sqr(d_bt) /4 + 3.14159265359 * sqr(d_ct) /4*L_ct_tot ;
b8a:            A_proj          =       n_bt_z*L_bt*dG;

# Determination of s_eq  FDBR 7.1.1 - 6/1
# During the Determination of s_eq for Carrier and Bundle Tubes,
they are evaluated together

# f3=0.85 for both bundle and wall(Cuboid)


s_eq1:          s_eq_wt_x       =       0.85*4*(V_duct-V_tubes)
/(2*L_wt*depth_duct);
s_eq2:          s_eq_wt_z       =       0.85*4*(V_duct-V_tubes)/
(2*L_wt*width_duct);
s_eq3:          s_eq_wt_y       =       0.85*4*(V_duct - V_tubes)/
(2*depth_duct*width_duct);
s_eq_b:         s_eq_b          =       0.85*4*(V_duct - V_tubes) / (
L_bt_tot * d_bt * 3.14159265359 + L_ct_tot * d_ct * 3.14159265359 )
;

########################
#         TESTS        #
########################

ztg6:   test(t>=d_wt)           error   "t must be >= d_wt";
h44g:   test(ss<=d_wt)          error   "ss must be <= d_wt";
hgfsd8: test(t>t1)              error   "t must be > t1";
uzu_:   test(n_ct_x*d_ct < L_bt)                error "Choose d_ct or
n_ct_x smaller";
f4sdgh: test(depth_duct > depth_bundle)         error "must be
depth_duct > depth_bundle";
t5:     test (tube_pitch_long*0.5 > fin_height) error " You must
change the number of gilles, or gilles high";
hd:     test (dG>0);
hfd:    test (d_bt>0);
gdfg:   test(B1<6)                      warning "B1 must be < 6";
g5f5:   test(fe_<=2+f1)                 warning "fe_ must be <=2+f1";
#g7fg:  test(fe_>=1.58)                 warning "fe_ must be >=1.58";
```

UENER - CONVECTION

```
########################
#    BASIC EQUATIONS    #
########################

# Mass Balance Equations

e1:     feed_bundle.massflow   = drain_bundle.massflow;
er42:   feed_fluegas.massflow  = drain_fluegas.massflow;
e3:     feed_ct.massflow       = drain_ct.massflow;
e4:     feed_wt.massflow       = drain_wt.massflow;

# Pressure Drops

f3:     feed_bundle.p - delta_p_bundle       = drain_bundle.p;
f4:     feed_fluegas.p - delta_p_fluegas     = drain_fluegas.p;
f63:    feed_wt.p - delta_p_wt               = drain_wt.p;
f46:    feed_ct.p - delta_p_ct               = drain_ct.p;

hdg:    L_wt= height_duct;
hdh:    L_ct= height_duct;

# Determination of Temperature Differences
# FDBR 6.2.2 - 5/1

ifl     current_type_bt == concurrent        then
f7co:   abs(feed_fluegas.t- feed_bundle.t)   =
        dt_in_gb;
f8_co:  abs(drain_fluegas.t- drain_bundle.t) =
        dt_out_gb ;
endifl

ifl     current_type_bt == countercurrent    then
fl7_counter:   abs(drain_fluegas.t- feed_bundle.t)   =
        dt_in_gb ;
f8r_counter:   abs(feed_fluegas.t- drain_bundle.t)   =
        dt_out_gb;
endifl

ifl     current_type_wt == concurrent        then
f7_co:  abs(feed_fluegas.t- tm_water_wt)     =              dt_in_gw
;
f8co:   abs(drain_fluegas.t- tm_water_wt)    =
        dt_out_gw ;
endifl

ifl     current_type_wt == countercurrent    then
f7jcounter:    abs(drain_fluegas.t- tm_water_wt)     =
        dt_in_gw;
f8e_counter:   abs(feed_fluegas.t- tm_water_wt)      =
        dt_out_gw;
endifl

ifl     current_type_ct == concurrent        then
f74_co:        abs(feed_fluegas.t- feed_ct.t)        =
        dt_in_gc;
f8_4co:        abs(drain_fluegas.t-drain_ct.t)       =
        dt_out_gc;
endifl

ifl     current_type_ct == countercurrent    then
f7_counter:    abs(drain_fluegas.t- feed_ct.t)       =
        dt_in_gc;
f8h_counter:   abs(feed_fluegas.t- drain_ct.t)       =
        dt_out_gc;
endifl

# Logarithmic Temperature Differences
# FDBR 6.2.2 - 5/6

dt_in_we: dt_in_we     = feed_wt.t-Ambient.t;
dt_out_we: dt_out_we   = drain_wt.t-Ambient.t;
dt_log_gb : dt_log_gb  = (dt_in_gb-dt_out_gb) /
ln(dt_in_gb/dt_out_gb);
dt_log_gw: dt_log_gw   =(dt_in_gw-dt_out_gw) / ln(dt_in_gw/dt_out_gw)
;
dt_log_gc: dt_log_gc   = (dt_in_gc-dt_out_gc) /
ln(dt_in_gc/dt_out_gc) ;
dt_log_we: dt_log_we   =(dt_in_we-dt_out_we) /
        ln(dt_in_we/dt_out_we) ;

# Energy Equations
#-----------------

iflref(feed_ct) &&ref(drain_ct) then

hh18:   Q_gc_conv=k_gc * A_ct * dt_log_gc * fouling_factor ;

endifl

ifl!ref(feed_ct) &&!ref(drain_ct) then

u6el:   Q_gc_conv      =       0.0;

endifl

iflref(feed_wt) &&ref(drain_wt) then

hh118:  Q_gw_conv=k_gw * A_wt * dt_log_gw * fouling_factor ;
```

```
endifl

ifl!ref(feed_wt) &&!ref(drain_wt) then

u36l:   Q_gw_conv       =       0.0;

endifl

ifl tube_type == smooth_tubes then

h21h218: Q_gb_conv=k_gb * A_bt * dt_log_gb * fouling_factor ;
L_fin:    L_fin = 0;

endifl

ifl tube_type == fin_tubes then

gsh4:   Q_gb_conv=k_gb * a * L_fin * dt_log_gb;

endifl

ifl!ref(feed_bundle) &&!ref(drain_bundle) then

u26l:   Q_gb_conv       =       0.0;

endifl

iflref(feed_fluegas) &&ref(drain_fluegas) then

hh128:  Q_fluegas_conv =Q_gw_conv + Q_gc_conv + Q_gb_conv +
Q_loss_conv;

endifl

ifl!ref(feed_fluegas) &&!ref(drain_fluegas) then

u6fl:    Q_fluegas_conv         =       0.0;
u6h6l:   Q_gc_conv              =       0.0;
u34d6l: Q_gb_conv               =       0.0;
uk6l:    Q_gw_conv              =       0.0;

endifl

jjg5:   Q_gc_conv_ = feed_ct.massflow*(drain_ct.h-feed_ct.h)*1000;
hfgi53: Q_gw_conv_ = feed_wt.massflow*(drain_wt.h-feed_wt.h)*1000;
j7jg5:  Q_gb_conv_ = feed_bundle.massflow*(drain_bundle.h-
feed_bundle.h)*1000;

ifl     wall_type == membrane           then

Q_lossa: Q_loss_conv   =k_we * A_wt * dt_log_we * fouling_factor ;

#fouling factor: FDBR  6.1.3 - 2/2
endifl

ifl     wall_type == adiabetic          then

Q_lossb: Q_loss_conv            =       0;

endifl

# Surface Temperatures
# VDI 2010 C2 - Eq6

# Bundle Tubes
#--------------

temp_bt_surfa3ce: temp_bt_surface =  feed_fluegas.t - k_gb *
dt_log_gb / alpha_conv_gb;

# Wall Tubes
#------------

temp_wt_surfac6e: temp_wt_surface  =  feed_fluegas.t - k_gw *
dt_log_gw / alpha_conv_gw  ;

# Carrier Tubes
#---------------

temp_ct_surfacke: temp_ct_surface  =  feed_fluegas.t - k_gc *
dt_log_gc / alpha_conv_gc;

# Wall-Environment Surface
#-------------------------

temp_we_surfac6e: temp_we_surface =   Ambient.t  + k_we * dt_log_we
/ alpha_conv_we;

########################
#        K Values        #  FDBR 7.2.2 - 19/6
########################

# Determination of k_gw

k_gw: 1/k_gw   = 1/alpha_conv_gw + d_wt*0.5 / lambda_steel_wt
* ln( d_wt / (d_wt-2*tube_thickness_wt) ) + d_wt/alpha_in_wt/(d_wt-
2*tube_thickness_wt);

# Determination of k_gb

ifl tube_type == fin_tubes then
```

```
tehfs: 1/k_gb  = 1/alpha_conv_gb + (dG * ln(d_bt / (d_bt -
2*tube_thickness_bt))/lambda_fin/2)
+ dG / (d_bt - 2*tube_thickness_bt) / alpha_in_bt ;

endifl

ifl tube_type == smooth_tubes then

r4:1/k_gb= 1/alpha_in_bt + 1/alpha_conv_gb + tube_thickness_bt /
lambda_steel_bt;

endifl

# Determination of k_we

ifl     isolation == without   then

k_we1: 1/k_we =
1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+1/alpha_in_wt;
gd8:            iso_thickn_one          =       0;
gd9:            iso_thickn_two          =       0;
gd10:           iso_thickn_three        =       0;
gd11:           lambda_iso_one          =       0;
gd12:           lambda_iso_two          =       0;
gd13:           lambda_iso_three        =       0;

endifl

ifl     isolation == one       then

k_we2: 1/k_we  =
1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+iso_thickn_one/lam
bda_iso_one+1/alpha_in_wt;
gd14:           iso_thickn_two          =       0;
gd15:           iso_thickn_three        =       0;
gd16:           lambda_iso_two          =       0;
gd17:           lambda_iso_three        =       0;

endifl

ifl     isolation == two       then

k_we3: 1/k_we = 1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt
+ iso_thickn_one/lambda_iso_one+
iso_thickn_two/lambda_iso_two+1/alpha_in_wt;
gd18:           iso_thickn_three                =       0;
gd19:           lambda_iso_three                =       0;

endifl

ifl     isolation == three     then

k_we4: 1/k_we  = 1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt
+ iso_thickn_one/lambda_iso_one+
iso_thickn_two/lambda_iso_two+iso_thickn_three/lambda_iso_three+1/al
pha_in_wt;

endifl

# Determination of k_gc

k_gc:   1/k_gc  = 1/alpha_conv_gc + d_ct*0.5 / lambda_steel_ct * ln(
d_ct / (d_ct-2*tube_thickness_ct) ) + d_ct/alpha_in_ct/(d_ct-
2*tube_thickness_ct);

########################
#        DENSITY       #
########################

g4df:           density_water_wt        =       1/vm_wt;
gduf:           density_water_ct        =       1/vm_ct;
gdf:            density_water_bt        =       1/vm_bt;
jhud:           density_air             =       Ambient.density_air;
wrw:            density_fluegas         =       1/vm_fluegas;

########################
#       VELOCITY       #
########################

vjfg:   n_wt*( sqr(d_wt-2*tube_thickness_wt) * 3.14159265359 /4) *
w_wt    = feed_wt.massflow *vm_wt ;
v2kj:   n_ct*( sqr(d_ct-2*tube_thickness_ct) * 3.14159265359 /4) *
w_ct    = feed_ct.massflow *vm_ct ;
v2:     n_bt*( sqr(d_bt-2*tube_thickness_bt) * 3.14159265359 /4) *
w_bt    = feed_bundle.massflow *vm_bt ;
v3:     (width_duct*depth_duct - A_proj)       * w_fluegas =
feed_fluegas.massflow *vm_fluegas;
ujrt:   w_air = 0.65 * sqrt( 9.81 * L_wt * (drain_wt.t-Ambient.t) /
(273 + Ambient.t) );
#temp_we_surface is as drain_wt.t assumed due to conflict in program

########################
#     AVERAGE VALUES     #
########################

a1:     tm_fluegas      = (feed_fluegas.t+drain_fluegas.t)*0.5;
a2:     tm_water_bt     = (feed_bundle.t+drain_bundle.t)*0.5;
ar2:    tm_water_ct     = (feed_ct.t+drain_ct.t)*0.5;
au2:    tm_water_wt     = (feed_wt.t+drain_wt.t)*0.5;

a4:     vm_fluegas      = (feed_fluegas.v+drain_fluegas.v)*0.5;
a3:     vm_bt           = (feed_bundle.v+drain_bundle.v  )*0.5;
```

```
ahz3:   vm_ct          = (feed_ct.v+drain_ct.v  )*0.5;
ae3:    vm_wt          = (feed_wt.v+drain_wt.v  )*0.5;

dgi:    pm_fluegas     = (feed_fluegas.p+drain_fluegas.p)*0.5;
dgi1:   pm_water_bt    = (feed_bundle.p+drain_bundle.p)*0.5;
dgi2:   pm_water_ct    = (feed_ct.p+drain_ct.p)*0.5;
dgi3:   pm_water_wt    = (feed_wt.p+drain_wt.p)*0.5;

#jr:    u_mean_bt      = w_bt;
#kr_:  u_mean_ct       = w_ct; #Presumptions
#rz:    u_mean_wt      = w_wt;

# Determination of Specific Heat Capacity, Dynamic Viscosity and
Thermal Conductivity

# Bundle Tubes
cp_1:   cp_water_bt =
10^3*f_cp_pt(pm_water_bt,tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0,0);
visc_water1:   dyn_visc_water_bt = 10^-
6*f_eta_pt(pm_water_bt,tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water1: lambda_water_bt = f_lambda_pt(pm_water_bt,
tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0,0);

# Wall Tubes
cp_8:   cp_water_wt = 10^3*f_cp_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);
visc_water3:   dyn_visc_water_wt = 10^-6*f_eta_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water3: lambda_water_wt = f_lambda_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);

# Carrier Tubes
cp_7:   cp_water_ct = 10^3*f_cp_pt(pm_water_ct,
tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);
visc_water2: dyn_visc_water_ct = 10^-
6*f_eta_pt(pm_water_ct,tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water2: lambda_water_ct = f_lambda_pt(pm_water_ct,
tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);

# Determination of pr_fluegas
# FDBR 5 - 2/1
Prandtl_fluegas: pr_fluegas =
cp_fluegas*dyn_visc_fluegas/lambda_fluegas;

cp_3: cp_fluegas = 10^3*heatCapacity(
tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yAr)/2,(feed_flue
gas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluegas.Gas.yC2H6+dra
in_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain_fluegas.Gas.yC
3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)/2,(feed_fluega
s.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.yCO2+drain_flue
gas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.Gas.yH2)/2,(feed
_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_fluegas.Gas.yH2S+d
rain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain_fluegas.Gas.yH
Cl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/2,(feed_fluegas
.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.yN2O+drain_flueg
as.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.Gas.yNH3)/2,(fee
d_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_fluegas.Gas.yO2+dra
in_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fluegas.Gas.ySO2)
/2);
visc_fluegas:  dyn_visc_fluegas = 10^-
6*dynViscosity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yA
r)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluega
s.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain
_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)
/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.
yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.G
as.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_flu
egas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain
_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/
2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.y
N2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.G
as.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_flue
gas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fl
uegas.Gas.ySO2)/2);
Lambda_fluegas: lambda_fluegas =
heatConductivity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.
yAr)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_flue
gas.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+dra
in_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH
4)/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Ga
s.yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas
.Gas.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_f
luegas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+dra
in_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN
)/2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas
.yN2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas
.Gas.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_fl
uegas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_
fluegas.Gas.ySO2)/2);

# Determination of pr_air
pr_air:  pr_air  = cp_air*dyn_visc_air/lambda_air/100;
cp_2: cp_air =
10^3*heatCapacity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.0397,0.0
00055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
visc_air: dyn_visc_air  = 10^-
6*dynViscosity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.0397,0.0000
55,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
Lambda_air: lambda_air        =
       heatConductivity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.03
97,0.000055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
```

```
# Determination of alpha_in_ct
# FDBR 7.2.2 - 2/1 & - 2/2
# Determination of f_2_ct according to Re_ct_in
# FDBR 7.2.2 - 1/4

gdf55g:  f_2_ct        =       1.85*(Re_ct_in^-0.05 - 180*Re_ct_in^-
0.8);
Re_ct_in: Re_ct_in     =       density_water_ct*w_ct*(d_ct-
2*tube_thickness_ct)/dyn_visc_water_ct;
alpha_in5_ct:
if      Re_ct_in < 10000       then
alpha_in_ct=f_1_ct*f_2_ct*0.02*(lambda_water_ct^0.58)*(dyn_visc_wate
r_ct^(-0.38))*(cp_water_ct^0.42)*((feed_ct.massflow / (3.14159265359
*n_ct* sqr(d_ct-2*tube_thickness_ct) /4))^0.8)*(d_ct-
2*tube_thickness_ct)^(-0.2);
# Additionaly multiplied by f_2_ct
else
alpha_in_ct =
f_1_ct*0.02*(lambda_water_ct^0.58)*(dyn_visc_water_ct^(-
0.38))*(cp_water_ct^0.42)*((feed_ct.massflow / (3.14159265359 *n_ct*
sqr(d_ct-2*tube_thickness_ct) /4))^0.8)*(d_ct-
2*tube_thickness_ct)^(-0.2);
# alpha_i_8    FDBR 7.2.2 - 1/5
# f_1_ct       FDBR fig. 7.2.2 - 2/1

# Determination of Nus_ct_out
# VDI 2010 F2 - Eq1
nusselt_ct_out: Nus_ct_out =
sqr(0.825+0.387*(Ra_ct*f1pr_ct)^(1/6))+0.97*L_ct/d_ct;
#Re FDBR 5 - 1/1
#0.97*h/d added for vertical cylinder,
#VDI 2010 F2 - Eq3
f1pr_ct: f1pr_ct = (1+(0.492/pr_fluegas)^(9/16))^(-16/9);
# VDI 2010 F2 - Eq2
Ra_ct:  Ra_ct = 9.81*(L_ct^3)*(1/(feed_fluegas.t+273.15))*
(feed_fluegas.t-
drain_ct.t)/(dyn_visc_fluegas/density_fluegas)/therm_diff_ct;
# temp_ct_surface is as drain_ct.t assumed due to conflict in
program
therm_diff_ct: therm_diff_ct =
lambda_fluegas/cp_fluegas/density_fluegas;

# Determination of alpha_conv_gc

alpha_conv_gc: alpha_conv_gc  = Nus_ct_out*lambda_fluegas/L_ct;

# Determination of alpha_in_wt
# FDBR 7.2.2 - 2/1 & - 2/2
# Determination of f_2_wt according to Re_wt_in
# FDBR 7.2.2 - 1/4

gdf5u5g: f_2_wt        =       1.85*(Re_wt_in^-0.05 - 180*Re_wt_in^-
0.8);
reynold_wt: Re_wt_in   =       density_water_wt*w_wt*(d_wt-
2*tube_thickness_wt)/dyn_visc_water_wt;
alpha_in_wt:
if      Re_wt_in < 10000
then
alpha_in_wt
=f_1_wt*f_2_wt*0.02*(lambda_water_wt^0.58)*(dyn_visc_water_wt^-
0.38)*(cp_water_wt^0.42)*((feed_wt.massflow / (3.14159265359 *n_wt*
sqr(d_wt-2*tube_thickness_wt) /4))^0.8)*((d_wt-
2*tube_thickness_wt)^-0.2);
else
alpha_in_wt =
f_1_wt*0.02*(lambda_water_wt^0.58)*(dyn_visc_water_wt^-
0.38)*(cp_water_wt^0.42)*((feed_wt.massflow / (3.14159265359 *n_wt*
sqr(d_wt-2*tube_thickness_wt) /4))^0.8)*((d_wt-
2*tube_thickness_wt)^-0.2);

# Determination of Nus_we_0

Nu_wt_s0:              Nu_we_0        =       ( sqr(Nu_we_turb)+
sqr(Nu_we_lam))^0.5;
@#VDI G4 - 4 - eq9
Nu_wt_elam:            Nu_we_lam      =       0.664* (Re_we^0.5 *
pr_air/3);
@#VDI G4 - 2.1 - eq2
Nu_wtk_turb:           Nu_we_turb     =
       0.037*(Re_we^0.8)*pr_air / (1+(pr_air^(2/3)-1)*2.443*Re_we^-
0.1);
@#VDI G4 - 4 - Eq8
re_wht: Re_we          =       w_air*L_wt/(dyn_visc_air/density_air);

# Determination of alpha_conv_we

alpha_conv_we: alpha_conv_we  =Nus_we_0*lambda_air/L_wt;

# Determination of Nu_wt_0
# VDI 2010 G4

Nu_wt_0: Nu_wt_0       =( sqr(Nu_wt_turb)+ sqr(Nu_wt_lam))^0.5;

# VDI 2010 G4 - Eq9
Nu_wt_lam: Nu_wt_lam   =0.664* (Re_wt)^0.5 * pr_fluegas^(1/3);

# VDI 2010 G4 - Eq2
Nu_wt_turb: Nu_wt_turb =0.037*(Re_wt^0.8)*pr_fluegas /
(1+(pr_fluegas^(2/3)-1)*2.443*Re_wt^-0.1);
# VDI 2010 G4 - Eq8
re_wt:  Re_wt   =w_fluegas*L_wt/(dyn_visc_fluegas/density_fluegas);
```

```
# Determination of alpha_conv_gw

alpha_conv_gw: alpha_conv_gw  =Nu_wt_0*lambda_fluegas/L_wt;

# Determination of alpha_in_bt
# FDBR 7.2.2 - 2/1 & - 2/2
# Determination of f_2_bt according to Re_bt_in
# FDBR 7.2.2 - 1/4

reynold_bt_in: Re_bt_in = density_water_bt*w_bt*(d_bt-
2*tube_thickness_bt)/dyn_visc_water_bt;
gdf5g:  f_2_bt  = 1.85*(Re_bt_in^-0.05 - 180*Re_bt_in^-0.8);

alpha_in_bt:
if      Re_bt_in < 10000
then
alpha_in_bt    =
f_1_bt*f_2_bt*0.02*lambda_water_bt^0.58*dyn_visc_water_bt^-
0.38*cp_water_bt^0.42*(feed_bundle.massflow / (3.14159265359 *n_bt*
(d_bt-2*tube_thickness_bt)^2 /4))^0.8*(d_bt-2*tube_thickness_bt)^-
0.2;
else
alpha_in_bt    =
f_1_bt*0.02*lambda_water_bt^0.58*dyn_visc_water_bt^-
0.38*cp_water_bt^0.42*(feed_bundle.massflow / (3.14159265359 *n_bt*
(d_bt-2*tube_thickness_bt)^2 /4) )^0.8*(d_bt-2*tube_thickness_bt)^-
0.2;

ifl     tube_arrangement == staggered then

gggsg:  C= 0.45;

endifl

ifl     tube_arrangement == in_line then

gggsg3: C= 0.3;

endifl

ifl tube_type == smooth_tubes then

gsdf7: lambda_fin                    = 0 ;
hdfg: alpha_conv_fin                 = 0;
jf64:   fin_thickness                = 0;
hg:     fin_height                   = 0;
gr7:    dG                           = d_bt;
fe3:    a_r                          = 0;
ge2:    a_ko                         = 0;
gr1:    a_ro                         = 0;
gr:     a                            = 0;

hhh54e:  fG =
(lambda_fluegas/lambda_air)^0.636*(cp_fluegas/cp_air)^0.364*(dyn_vis
c_fluegas/dyn_visc_air)^-0.236;
alpha_5: alpha_conv_gb = alpha_air*fG*fe;
t8tut:  alpha_air             =
0.287*lambda_air^0.636*cp_air^0.364*dyn_visc_air^-
0.236*phi_fluegas^0.6*d_bt^-0.4;
hhhi: f_1                     = 0;
tzi: f_3                      = 0;
hh9yyy: fin_efficiency        = 0;

endifl

ifl tube_type == fin_tubes then

hhh546: fG =
(lambda_fluegas/lambda_air)^0.667*(cp_fluegas/cp_air)^0.333*(dyn_vis
c_fluegas/dyn_visc_air)^-0.292;
alpha_6: alpha_conv_fin=      alpha_air*fG;
gggg000: alpha_conv_gb =
(alpha_conv_fin*a_ro+fin_efficiency*alpha_conv_fin*(a_r+a_ko))/a;
gggg5:  alpha_air=     C*lambda_air^0.667*cp_air^0.333*dyn_visc_air^-
0.292*phi_fluegas^0.625*dG^-0.375;
f_1z5:
       f_1=(2*alpha_conv_gb/fin_thickness/lambda_fin)^0.5*fin_height
;
f_3gs:  f_3=    fin_thickness/fin_height;
hhyyy:  fin_efficiency = ( tanh (f_1)+0.5*f_1*f_3) / f_1 /
(1+0.5*f_3) / (1+0.5*f_1*f_3* tanh (f_1));
t_fin_efficiency:
test (fin_efficiency <=1) error "fin efficiency must be less than
1.0";

endifl

am_tube: am_tube = ((d_bt *  3.14159265359)-((d_bt-
2*tube_thickness_bt) * 3.14159265359)) / ln((d_bt *  3.14159265359)
/(d_bt-2*tube_thickness_bt) * 3.14159265359);
# FDBR  6.1.2 - 2/4

terju:  l       =       3.14159265359*0.5*d_bt;

########################
#         TESTS        #
########################

t4:     test (Re_wt< 10^7)     error   "it has to be Re_wt<10^7";
t3:     test (Re_wt> 10)       error   "it has to be Re_wt>10";
jt4:    test (Re_we< 10^7)     error   "it has to be Re_we<10^7";
t3t:    test (Re_we> 10)       error   "it has to be Re_we>10";
tt1t:   test (dt_in_we>0.0)            warning "negative";
```

```
t2t9:   test (dt_out_we>0.0)          warning "negative";
t1t9:   test (dt_in_gw>0.0)           warning "negative";
t2t7:   test (dt_out_gw>0.0)          warning "negative";
t1t7:   test (dt_in_gc>0.0)           warning "negative";
t26:    test (dt_out_gc>0.0)          warning "negative";
t15:    test (dt_in_gb>0.0)           warning "negative";
t32:    test (dt_out_gb>0.0)          warning "negative";
t23:    test (Q_gc_conv>0.0)          warning "negative";
t263:   test (Q_gb_conv>0.0)          warning "negative";
tj3:    test (Q_gw_conv>0.0)          warning "negative";
kt3:    test (Q_loss_conv>=0.0)       warning "negative";
tk3:    test (Q_fluegas_conv>0.0)     warning "negative";

# Testing Rayleigh Numbers
#VDI 2010 F2 -1

op4pp: test( Ra_ct >= 10^-1) warning "has to be Ra_ct >= 10^-1";
zuuuu: test( Ra_ct <= 10^12) warning "has to be Ra_ct <= 10^12";

# Testing Prandtl Numbers

klg:test( pr_air > 0.001)     warning "has to be pr_air>0.001";
t1:test(0.5< pr_fluegas)      error   "it has to be
0.5<prandtl_fluegas";
t42:    test(2000> pr_fluegas)error   "it has to be
2000>prandtl_fluegas";
#kllg:  test( pr_fluegas > 0.001)warning "has to be
pr_fluegas>0.001";

zetue:  Q_fluegas_tot  =      Q_fluegas_rad+Q_fluegas_conv;
sege:   Q_fluegas_tot_ =      feed_fluegas.massflow*(feed_fluegas.h-
drain_fluegas.h)*1000;

# kinematic viscosity  = dyn_visc_fluegas/density_fluegas
```

UENER - RADIATION

```
########################
#   Partial Pressures   #
########################

pH2O1: p_H2O   =      feed_fluegas.p * feed_fluegas.Gas.yH2O;
pCO21: p_CO2   =      feed_fluegas.p * feed_fluegas.Gas.yCO2;

ps_x:   ps_x   =      p_H2O * s_eq_wt_x;
ps_y:   ps_y   =      p_H2O * s_eq_wt_y;
ps_z:   ps_z   =      p_H2O * s_eq_wt_z;

#(0.05 < p_H2O*s_eq_wt_x)  && (p_H2O*s_eq_wt_x < 0.5)

tr1f1: if      ps_x < 0.5     then

emiss_H2O_x    =      (0.43265 -0.1089 * (feed_fluegas.t+273.15) /
1000) - (0.3273 -0.043821 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-3.5829 * p_H2O*s_eq_wt_x);

else

emiss_H2O_x    =      (0.66439 -0.17389 * (feed_fluegas.t+273.15) /
1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-0.84652 * p_H2O*s_eq_wt_x);

tf1: if        ps_y < 0.5     then

emiss_H2O_y    =      (0.43265 -0.1089 * (feed_fluegas.t+273.15) /
1000) - (0.3273 -0.043821 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-3.5829 * p_H2O*s_eq_wt_y);

else

emiss_H2O_y    =      (0.66439 -0.17389 * (feed_fluegas.t+273.15) /
1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-0.84652 * p_H2O*s_eq_wt_y);

trh11: if              ps_z < 0.5     then

emiss_H2O_z    =      (0.43265 -0.1089 * (feed_fluegas.t+273.15) /
1000) - (0.3273 -0.043821 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-3.5829 * p_H2O*s_eq_wt_z);

else

emiss_H2O_z    =      (0.66439 -0.17389 * (feed_fluegas.t+273.15) /
1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-0.84652 * p_H2O*s_eq_wt_z);

ifl fluegas == water_vapor then

emissivity_wv_x:       emissivity_gas_x       =       emiss_H2O_x;
emissivity_wv_y:       emissivity_gas_y       =       emiss_H2O_y;
emissivity_wv_z:       emissivity_gas_z       =       emiss_H2O_z;

A_v_water_x:   A_v_x   =       emiss_H2O_x *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.45;
A_v_water_y:   A_v_y   =       emiss_H2O_y *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.45;
A_v_water_z:   A_v_z   =       emiss_H2O_z *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.45;

tt7:    test (feed_fluegas.t < (1500-273.15) )       error
"feed_fluegas.t has to be between 700 and 1500 K";
tt8:    test (feed_fluegas.t > (700-273.15) )        error
"feed_fluegas.t has to be between 700 and 1500 K";
ttgo:   test (p_H2O*s_eq_wt_x < 0.5 )                error
"p_H2Oxs_eq_wt_x has to be between 0.05 and 0.5 mbar or 0.5-2";
tjtz:   test (p_H2O*s_eq_wt_x >0.05 )                error
"p_H2Oxs_eq_wt_x has to be between 0.05 and 0.5 mbar or 0.5-2";
tt5to:  test (p_H2O*s_eq_wt_x < 2 )                  error
"p_H2Oxs_eq_wt_x has to be between 0.05 and 0.5 mbar or 0.5-2";
tt2zk:  test (p_H2O*s_eq_wt_x >0.5 )                 error
"p_H2Oxs_eq_wt_x has to be between 0.05 and 0.5 mbar or 0.5-2";
ttgo1:  test (p_H2O*s_eq_wt_y < 0.5 )                error
"p_H2Oxs_eq_wt_y has to be between 0.05 and 0.5 mbar or 0.5-2";
tjtz1:  test (p_H2O*s_eq_wt_y >0.05 )                error
"p_H2Oxs_eq_wt_y has to be between 0.05 and 0.5 mbar or 0.5-2";
tt5to1: test (p_H2O*s_eq_wt_y < 2 )                  error
"p_H2Oxs_eq_wt_y has to be between 0.05 and 0.5 mbar or 0.5-2";
tt2zk1: test (p_H2O*s_eq_wt_y >0.5 )                 error
"p_H2Oxs_eq_wt_y has to be between 0.05 and 0.5 mbar or 0.5-2";
ttgo2:  test (p_H2O*s_eq_wt_z < 0.5 )                error
"p_H2Oxs_eq_wt_z has to be between 0.05 and 0.5 mbar or 0.5-2";
tjtz2:  test (p_H2O*s_eq_wt_z >0.05 )                error
"p_H2Oxs_eq_wt_z has to be between 0.05 and 0.5 mbar or 0.5-2";
tt5to2: test (p_H2O*s_eq_wt_z < 2 )                  error
"p_H2Oxs_eq_wt_z has to be between 0.05 and 0.5 mbar or 0.5-2";
tt2zk2: test (p_H2O*s_eq_wt_z >0.5 )                 error
"p_H2Oxs_eq_wt_z has to be between 0.05 and 0.5 mbar or 0.5-2";

endifl

# Analytical calculation of the emissivities of water vapor and
carbondioxide and their mixtures VDI 2010 K3 - 5
```

```
emissivitgfd:   emissivity_co2_x = Z - gw_totx;
emisshd2_x:     emissivity_co2_y = Z - gw_toty;
emissividfg2_x:  emissivity_co2_z = Z - gw_totz;

fgdg:   fpco2_x =1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_wt_x)));
fgh:    fpco2_y =1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_wt_y)));
kf:     fpco2_z =1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_wt_z)));

ifl fluegas == carbondioxide then

emissivity_gasco2x: emissivity_gas_x  =emissivity_co2_x;
emissivity_gasco2y: emissivity_gas_y  =emissivity_co2_y;
emissivity_gasco2z: emissivity_gas_z  =emissivity_co2_z;

A_v_carbondioxidex: A_v_x     =       fpco2_x*emissivity_co2_x *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;
A_v_carbondioxidey: A_v_y     =       fpco2_y*emissivity_co2_y *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;
A_v_carbondioxidez: A_v_z     =       fpco2_z*emissivity_co2_z *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;

tt3:            test (feed_fluegas.t >(300 - 273.15) )       error
"feed_fluegas.t has to be between 300 and 1800 K";
tt4:            test (feed_fluegas.t <(1800-273.15) )        error
"feed_fluegas.t has to be between 300 and 1800 K";
tt5:            test (p_CO2*s_eq_wt_x < 10 )                 error
"p_CO2xs_eq_x has to be between 0.01 and 10 mbar";
tt6:            test (p_CO2*s_eq_wt_x >0.01 )                error
"p_CO2xs_eq_x has to be between 0.01 and 10 mbar";
tt51:           test (p_CO2*s_eq_wt_y < 10 )                 error
"p_CO2xs_eq_y has to be between 0.01 and 10 mbar";
tt64:           test (p_CO2*s_eq_wt_y >0.01 )                error
"p_CO2xs_eq_y has to be between 0.01 and 10 mbar";
tt55:           test (p_CO2*s_eq_wt_z < 10 )                 error
"p_CO2xs_eq_z has to be between 0.01 and 10 mbar";
tt66:           test (p_CO2*s_eq_wt_z >0.01 )                error
"p_CO2xs_eq_z has to be between 0.01 and 10 mbar";

endifl

k78k1:  A_v_water_vapor_x      =       emiss_H2O_x * (
(feed_fluegas.t+273.15) / (temp_wt_surface+273.15))^0.45;
kk81:   A_v_water_vapor_y      =       emiss_H2O_y * (
(feed_fluegas.t+273.15) / (temp_wt_surface+273.15))^0.45;
kk01:   A_v_water_vapor_z      =       emiss_H2O_z * (
(feed_fluegas.t+273.15) / (temp_wt_surface+273.15))^0.45;
# emiss_H2O can be readable from VDI 2010 K3 - Fig. 5b

k8k2:   A_v_carbondioxide_x    =       fpco2_x*emissivity_co2_x *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;
kk62:   A_v_carbondioxide_y    =       fpco2_y*emissivity_co2_y *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;
kk42:   A_v_carbondioxide_z    =       fpco2_z*emissivity_co2_z *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;
# emissivity_co2 can be readable from VDI 2010 K3 - Fig. 4

ifl fluegas == mixture  then

zteg:   emissivity_gas_x       = (0.13 + 0.265 * (feed_fluegas.t +
273.15) / 1000) * (1- ( 2.718281828^(0*(p_H2O+p_CO2)*s_eq_wt_x))) +
(0.595 -0.15 * (feed_fluegas.t + 273.15) / 1000) * (1 - (
2.718281828^(-0.824*(p_H2O+p_CO2)*s_eq_wt_x))) + (0.275 -0.115 *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_wt_x)));
jfgd:           emissivity_gas_y       = (0.13 + 0.265 *
(feed_fluegas.t + 273.15) / 1000) * (1- (
2.718281828^(0*(p_H2O+p_CO2)*s_eq_wt_y))) + (0.595 -0.15 *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
0.824*(p_H2O+p_CO2)*s_eq_wt_y))) + (0.275 -0.115 * (feed_fluegas.t +
273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_wt_y)));
ethdf:  emissivity_gas_z       = (0.13 + 0.265 * (feed_fluegas.t +
273.15) / 1000) * (1- ( 2.718281828^(0*(p_H2O+p_CO2)*s_eq_wt_z))) +
(0.595 -0.15 * (feed_fluegas.t + 273.15) / 1000) * (1 - (
2.718281828^(-0.824*(p_H2O+p_CO2)*s_eq_wt_z))) + (0.275 -0.115 *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_wt_z)));

A_vx:   A_v_x                  = A_v_water_vapor_x +
A_v_carbondioxide_x - d_emissivity_wx;
A_vy:   A_v_y                  = A_v_water_vapor_y +
A_v_carbondioxide_y - d_emissivity_wy;
A_vz:   A_v_z                  = A_v_water_vapor_z +
A_v_carbondioxide_z - d_emissivity_wz;
# d_emissivity for mixtures VDI 2010 K3 - Fig. 9(a-c)

#tt9:   test (feed_fluegas.t <(1800-273.15) ) error  "feed_fluegas.t
has to be between 1800 and 1100 K";
#tt10:  test (feed_fluegas.t >(1100-273.15) ) error  "feed_fluegas.t
has to be between 1800 and 1100 K";
tt13z1: test (s_eq_wt_x > 0.2 )                          error
"s_eq_wt_x has to be between 0.2 and 6 m";
tt12z2: test (s_eq_wt_x < 6 )                            error
"s_eq_wt_x has to be between 0.2 and 6 m";
tt15z1: test (s_eq_wt_y > 0.2 )                          error
"s_eq_wt_y has to be between 0.2 and 6 m";
tt128z2:        test (s_eq_wt_y < 6 )                    error
"s_eq_wt_y has to be between 0.2 and 6 m";
```

```
tt1z81: test (s_eq_wt_z > 0.2 )                    error
"s_eq_wt_z has to be between 0.2 and 6 m";
#tt1z22:       test (s_eq_wt_z < 6 )               error
"s_eq_wt_z has to be between 0.2 and 6 m";
@#hzd5gt:      test (p_H2O/p_CO2 == 1 )            warning
"p_H2O/p_CO2 has to be = 1 for gray- and -clear gas approximation";

endifl


# Total gas pressure 1 bar

# Determination of emissivity (radiative exchange between gas and
wall )  !!! for bundle !!!
# Determination of A_v (Absorption Behaviour)
# VDI 2010 K3 - Eq(20,21,23b)

#0.05 < p_H2O*s_eq_b  && p_H2O*s_eq_b < 0.5

t8sttt: if     feed_fluegas.t >= (700-273.15) then
B       =      0.225* sqr((feed_fluegas.t+273.15)/1000);
       else
B       =      0.054* sqr((feed_fluegas.t+273.15)/1000);
# VDI 2010 K3 - Eq17
ag1:    A              =
       (((0.1*((feed_fluegas.t+273.15)/1000)^-
1.45+1)*feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23)/(0.1*(fee
d_fluegas.p/1000)^-
1.45+feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23));
# VDI 2010 K3 - Eq16
jwes:   fpco2_b =      1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_b)));

ps_b:    ps_b          =      p_H2O*s_eq_b;


trhp11: if             ps_b < 0.5     then

       emiss_H2O_b    = (0.43265 -0.1089 * (feed_fluegas.t+273.15) /
1000) - (0.3273 -0.043821 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^( -3.5829 * p_H2O*s_eq_b);

else

       emiss_H2O_b    = (0.66439 -0.17389 * (feed_fluegas.t+273.15)
/ 1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^( -0.84652 * p_H2O*s_eq_b);


ifl fluegas == water_vapor then
```

```
emissivity_wv: emissivity_gas_b      =      emiss_H2O_b;
A_v_water:     A_v_b                 =      emiss_H2O_b *
((feed_fluegas.t+273.15)/((temp_bt_surface+temp_ct_surface)/2+273.15
))^0.45;
#emiss_H2O_b can be readable from VDI 2010 - K3 Fig 5b

ttrt:          test (0.05 < p_H2O*s_eq_b)           error
"ph20xs_eq must be between 0.05 and 2";
trt1:          test (p_H2O*s_eq_b < 2)              error
"ph20xs_eq must be between 0.05 and 2";
ttz7:          test (feed_fluegas.t <(1500-273.15) )error
"feed_fluegas.t has to be between 700 and 1500 K";
ttz8:          test (feed_fluegas.t >(700-273.15) ) error
"feed_fluegas.t has to be between 700 and 1500 K";
tto:           test (p_H2O*s_eq_b < 0.5 )                    error
"p_H2Oxs_eq has to be between 0.05 and 0.5 mbar or 0.5-2";
ttz:           test (p_H2O*s_eq_b >0.05 )                    error
"p_H2Oxs_eq has to be between 0.05 and 0.5 mbar or 0.5-2";
ttto:          test (p_H2O*s_eq_b < 2 )                      error
"p_H2Oxs_eq has to be between 0.05 and 0.5 mbar or 0.5-2";
ttzk:          test (p_H2O*s_eq_b >0.5 )                     error
"p_H2Oxs_eq has to be between 0.05 and 0.5 mbar or 0.5-2";

endifl

# Analytical calculation of the emissivities of water vapor and
carbondioxide and their mixtures VDI 2010 K3 - 5

hhet: emiss_co2_b = Z - g_tot;
gsg:    Z  = 0.27769 + 0.03869 * (feed_fluegas.t+273.15) / 1000 +
1.4249 * 10^-5 * ((feed_fluegas.t+273.15) / 1000)^2;

#-------------------------

ifl fluegas == carbondioxide then

emissihtgsco2: emissivity_gas_b      =      emiss_co2_b;
A_ghrg:        A_v_b          =      fpco2_b*emiss_co2_b *
((feed_fluegas.t+273.15)/((temp_bt_surface+temp_ct_surface)/2+273.15
))^0.65;
# emissivity_co2_b can be readable from VDI 2010 K3 - Fig. 4


ttk3:          test (feed_fluegas.t >(300 - 273.15))error
"feed_fluegas.t has to be between 300 and 1800 K";
tkt4:          test (feed_fluegas.t <(1800-273.15) )error
"feed_fluegas.t has to be between 300 and 1800 K";
ttk5:          test (p_CO2*s_eq_b < 10 )             error
"p_CO2xs_eq_b has to be between 0.01 and 10 mbar";
tkt6:          test (p_CO2*s_eq_b >0.01 )            error
"p_CO2xs_eq_b has to be between 0.01 and 10 mbar";
```

```
endifl

kk3:    A_v_water_vapor_b      =       emiss_H2O_b *
((feed_fluegas.t+273.15)/((temp_bt_surface+temp_ct_surface)/2+273.15
))^0.45;
kk4:    A_v_carbondioxide_b    =       fpco2_b*emiss_co2_b *
((feed_fluegas.t+273.15)/((temp_bt_surface+temp_ct_surface)/2+273.15
))^0.65;

emige5: emissivity_gas_mix_b   =       (0.13 + 0.265 * (feed_fluegas.t
+ 273.15) / 1000) * (1- ( 2.718281828^(-1*0*(p_H2O+p_CO2)*s_eq_b)))
+ (0.595 + (-0.15) * (feed_fluegas.t + 273.15) / 1000) * (1 - (
2.718281828^(-0.824*(p_H2O+p_CO2)*s_eq_b))) + (0.275 + (-0.115) *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_b)));

ifl fluegas == mixture  then

emissivity_gas_b:emissivity_gas_b     =       emissivity_gas_mix_b;
A_v:            A_v_b                 =       A_v_water_vapor_b +
A_v_carbondioxide_b - d_emissivity_b;
# delta_emiss_steam_b   VDI 2010 - K3 Fig. 9(a-c)

#tzt9:          test (feed_fluegas.t <(1800-273.15) ) error
"feed_fluegas.t has to be between 1800 and 1100 K";
#ttz10:         test (feed_fluegas.t >(1100-273.15) ) error
"feed_fluegas.t has to be between 1800 and 1100 K";
#tt1z1:         test (s_eq_b > 0.2 )                  error  "s_eq_b
has to be between 0.2 and 6 m";
#tt1z2:         test (s_eq_b < 6 )                    error  "s_eq_b
has to be between 0.2 and 6 m";

endifl

# Absorption behaviour of wall is taken as 0.85
# alpha_rad formulas FDBR 7.1.3 - 1/2

alpha_rad_gw_x:         alpha_rad_gw_x        =       (2*0.85*1 /
(1+0.85*1)) * radiation_coefficient*10^8*(
emissivity_gas_x*((feed_fluegas.t+273.15)/100)^4 -
A_v_x*((temp_wt_surface+273.15)/100)^4 )/(feed_fluegas.t -
temp_wt_surface);
alpha_rad_gw_y:         alpha_rad_gw_y        =       (2*0.85*1 /
(1+0.85*1)) * radiation_coefficient*10^8*(
emissivity_gas_y*((feed_fluegas.t+273.15)/100)^4 -
A_v_y*((temp_wt_surface+273.15)/100)^4 )/(feed_fluegas.t -
temp_wt_surface);
alpha_rad_gw_z:         alpha_rad_gw_z        =       (2*0.85*1 /
(1+0.85*1)) * radiation_coefficient*10^8*(
emissivity_gas_z*((feed_fluegas.t+273.15)/100)^4 -
A_v_z*((temp_wt_surface+273.15)/100)^4 )/(feed_fluegas.t -
temp_wt_surface);

#Radiation coefficient is multiplied by 10^8

# Determination of alpha_rad_gb_gc

alpha8_rad_gb: alpha_rad_gb_gc        =       (2*0.85*1 / (1+0.85*1))
*
radiation_coefficient*10^8*(emissivity_gas_b*((feed_fluegas.t+273.15
)/100)^4 -
A_v_b*(((temp_bt_surface+temp_ct_surface)/2+273.15)/100)^4
)/(feed_fluegas.t - (temp_bt_surface+temp_ct_surface)/2);

# Determination of Q_gb_gc_rad
# FDBR 7.1.3 - 1/1

hdfg:   Q_gb_gc_rad    =alpha_rad_gb_gc*(A_bt+A_ct)*(feed_fluegas.t-
(temp_bt_surface+temp_ct_surface)/2);

# Determination of Q_gw_rad_(x,y,z)
# VDI 2010 K3 - Eq18

drt:    Q_gw_rad_x
        =radiation_coefficient*depth_duct*height_duct*2*emiss_wall*(e
missivity_gas_x*(feed_fluegas.t+273.15)^4-
A_v_x*(temp_wt_surface+273.15)^4)/(1-(1-emiss_wall)*(1-A_v_x));
drt2:   Q_gw_rad_y      =
        radiation_coefficient*depth_duct*width_duct*2*emiss_wall*(emi
ssivity_gas_y*(feed_fluegas.t+273.15)^4-
A_v_y*(temp_wt_surface+273.15)^4)/(1-(1-emiss_wall)*(1-A_v_y));
drt3:   Q_gw_rad_z      =
        radiation_coefficient*width_duct*height_duct*2*emiss_wall*(em
issivity_gas_z*(feed_fluegas.t+273.15)^4-
A_v_z*(temp_wt_surface+273.15)^4)/(1-(1-emiss_wall)*(1-A_v_z));

jhfgh:  Q_fluegas_rad  =       Q_gb_gc_rad+Q_gw_rad_x+Q_gw_rad_z;

#Q_gw_rad_y passes to the next section

gfg5:    a1 = 0.1074 - 0.10705 * (feed_fluegas.t+273.15) / 1000 +
0.072727 * ((feed_fluegas.t+273.15) / 1000)^2;
jet:     a2 = 0.027237 + 0.10127 * (feed_fluegas.t+273.15) / 1000 -
0.043773 * ((feed_fluegas.t+273.15) / 1000)^2;
jfz6:    a3 = 0.058438 - 0.001208 * (feed_fluegas.t+273.15) / 1000 +
0.0006558 * ((feed_fluegas.t+273.15) / 1000)^2;
i75:     a4 = 0.019078 + 0.037609 * (feed_fluegas.t+273.15) / 1000 -
0.015424 * ((feed_fluegas.t+273.15) / 1000)^2;
hfgh6: a5 = 0.056993 - 0.025412 * (feed_fluegas.t+273.15) / 1000 +
0.0026167 * ((feed_fluegas.t+273.15) / 1000)^2;
```

```
jghj8: a6 = 0.0028014 + 0.038826 * (feed_fluegas.t+273.15) / 1000 -
0.020198 * ((feed_fluegas.t+273.15) / 1000)^2;


gh4:    k1 = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
h5:     k2 = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
hh6:    k3 = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
jf7:    k4 = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
oi7:    k5 = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
pf8:    k6 = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);


gh41:   kw1x = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);
h52:    kw2x = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);
hh61:   kw3x = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);
jf71:   kw4x = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);
oi71:   kw5x = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);
pf81:   kw6x = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_x);#

gh42:   kw1y = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);
h53:    kw2y = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);#
hh62:   kw3y = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);
jf72:   kw4y = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);
oi72:   kw5y = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);
pf82:   kw6y = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_y);

gh43:   kw1z = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);
h54:    kw2z = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);
hh63:   kw3z = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);
jf73:   kw4z = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);
oi73:   kw5z = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);
pf38:   kw6z = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt_z);

jfg:  g_tot   = (k1*a1)+(a2*k2)+(a3*k3)+(a4*k4)+(a5*k5)+(a6*k6);
jfg2: gw_totx = a1*kw1x+a2*kw2x+a3*kw3x+a4*kw4x+a5*kw5x+a6*kw6x;
jfg3: gw_toty = a1*kw1y+a2*kw2y+a3*kw3y+a4*kw4y+a5*kw5y+a6*kw6y;
jfg4: gw_totz = a1*kw1z+a2*kw2z+a3*kw3z+a4*kw4z+a5*kw5z+a6*kw6z;


########################
#         TESTS        #
########################

t263:   test(Q_gb_gc_rad>0.0)         warning "negative";
tj3:    test(Q_gw_rad_x>0.0)          warning "negative";
tj4:    test(Q_gw_rad_y>0.0)          warning "negative";
tj5:    test(Q_gw_rad_z>0.0)          warning "negative";
tk3:    test(Q_fluegas_rad>0.0)       warning "negative";
ttgrt:  test(s_eq_wt_x>0)             warning "negative";
ttjrt:  test(s_eq_wt_y>0)             warning "negative";
twtrt:  test(s_eq_wt_z>0)             warning "negative";
t7trt:  test(s_eq_b>0)                warning "negative";
```

UENER - PRESSURE LOSS GAS SIDE

```
########################
#    BASIC EQUATIONS    #
########################

# Mass Balance Equations

er42:   feed_fluegas.massflow = drain_fluegas.massflow;

# Pressure Drops

f4:     feed_fluegas.p - delta_p_fluegas  = drain_fluegas.p;

# Determination of Pressure Loss

j7f:    if n_bt_y > 10 then
# FDBR 9.1.3 - Fig. 1/2
                f_z             = f_z_;
        else
                f_z             = 1+(3/B1-1)*n_bt_y^-1.75;

urz:    ksi                     = fouling_factor*f_e*n_bt_y*f_z*f_sa;
# FDBR 9.1.3 - 1/2
urzu:   delta_p_fluegas = phi_fluegas^2*ksi*0.5/density_fluegas/10^5;
# FDBR 9.1.3 - 1/1
```

UENER - PRESSURE LOSS WATER SIDE

```
########################
#    BASIC EQUATIONS    #
########################

# Mass Balance Equations

e1:     feed_bundle.massflow   = drain_bundle.massflow;

# Pressure Drops

f3:     feed_bundle.p - delta_p_bundle = drain_bundle.p;

# Determination of Pressure Loss

gdf: ksi_tot   =
n_180*ksi_180+n_90*ksi_90+n_45*ksi_45+n_collector*ksi_collector+ksi_
friction+n_combining*ksi_combining;
hrtz: ksi_friction  = ksi_roughness * l_bt / (d_bt-
2*tube_thickness_bt);
# FDBR 9.1.2 - 1/2
hsrg: phi_bt    = feed_bundle.massflow / (3.14159265359*n_bt_z*
sqr(d_bt-2*tube_thickness_bt)*0.25 );
hdh:    Re      = phi_bt*(d_bt-
2*tube_thickness_bt)/dyn_visc_water_bt;
jf:     delta_p_bundle = vm_bt*ksi_tot* sqr(phi_bt) *0.5 / 10^5;
# FDBR 9.1.1 - 2/1
```

UENER_BLANK - GEOMETRY

```
########################
#    BASIC EQUATIONS    #
########################

er42:   feed_fluegas.massflow         = drain_fluegas.massflow;

#-----------------------------------------------------------------

A_ct:          A_ct           = 3.14159265359*d_ct*L_ct*n_ct;
A_wtjhk:       A_wt           = 2*(depth_duct+width_duct)*L_wt;

ert:           L_ct           = height_duct;
grst:          L_wt           = height_duct;
L_wt_tot:      L_wt_tot       = n_wt * L_wt;

f_1_ct: f_1_ct         = 1+((d_ct-2*tube_thickness_ct)/L_ct)^(2/3);
f_1_wt: f_1_wt         = 1+((d_wt-2*tube_thickness_wt)/L_wt)^(2/3);

# Determination of Duct Volume for Radiation

V_duct: V_duct         = height_duct*depth_duct*width_duct;

# Determination of s_eq
# During the determination of s_eq forCarrier and Bundle Tubes they
are evaluated together
# f3=0.85 for both bundles and walls (cuboid)
#FDBR  7.1.1 - 6/1

s_eq1:  s_eq_wt        =       0.85 * 4 * V_duct / A_wt;
```

UENER_BLANK - CONVECTION

```
########################
#    BASIC EQUATIONS    #
########################

# Mass Balance Equations

er42:   feed_fluegas.massflow  = drain_fluegas.massflow;
e3:     feed_ct.massflow       = drain_ct.massflow;
e4:     feed_wt.massflow       = drain_wt.massflow;

# Pressure Drops

f4:     feed_fluegas.p - delta_p_fluegas    = drain_fluegas.p;
f63:    feed_wt.p -delta_p_wt                 = drain_wt.p;
f46:    feed_ct.p -delta_p_ct                 = drain_ct.p;

ert:    L_ct = height_duct;
grst:   L_wt = height_duct;


ifl     current_type_wt == concurrent        then
        f7_co:          abs(feed_fluegas.t- tm_water_wt)      =
        dt_in_gw ;
        f8co:           abs(drain_fluegas.t- tm_water_wt)     =
        dt_out_gw ;
endifl

ifl     current_type_wt == countercurrent           then
        f7jcounter:     abs(drain_fluegas.t- tm_water_wt )    =
        dt_in_gw;
        f8e_counter:    abs(feed_fluegas.t- tm_water_wt )     =
        dt_out_gw;
endifl

ifl     current_type_ct == concurrent        then
        f74_co:         abs(feed_fluegas.t- feed_ct.t )       =
        dt_in_gc;
        f8_4co:         abs(drain_fluegas.t-drain_ct.t )      =
        dt_out_gc;
endifl

ifl     current_type_ct == countercurrent           then
        f7_counter:     abs(drain_fluegas.t- feed_ct.t)       =
        dt_in_gc;
        f8h_counter:    abs(feed_fluegas.t- drain_ct.t)       =
        dt_out_gc;
endifl
```

```
dt_in_we:       feed_wt.t-Ambient.t    =              dt_in_we;
dt_out_we:      drain_wt.t-Ambient.t   =              dt_out_we;

# Logarithmic Temperature Differences (LMTD)
# FDBR  6.2.2 - 5/6


dt_log_gw:      dt_log_gw      =       (dt_in_gw-dt_out_gw) /
        ln(dt_in_gw/dt_out_gw) ;
dt_log_gc:      dt_log_gc      =       (dt_in_gc-dt_out_gc) /
        ln(dt_in_gc/dt_out_gc) ;
dt_log_we:      dt_log_we      =       (dt_in_we-dt_out_we) /
        ln(dt_in_we/dt_out_we) ;


# Energy Equations

jjg5:   Q_gc_conv_ = feed_ct.massflow*(drain_ct.h-feed_ct.h)*1000;
h53:    Q_gw_conv_ = feed_wt.massflow*(drain_wt.h-feed_wt.h)*1000;

iflref(feed_ct) &&ref(drain_ct) then

hh18:    Q_gc_conv=k_gc * A_ct * dt_log_gc*fouling_factor ;

endifl

ifl!ref(feed_ct) &&!ref(drain_ct) then


u6el:    Q_gc_conv=0.0;

endifl

iflref(feed_wt) &&ref(drain_wt) then

hh118:  Q_gw_conv=k_gw * A_wt * dt_log_gw *fouling_factor ;

endifl

ifl!ref(feed_wt) &&!ref(drain_wt) then

u361:    Q_gw_conv=0.0;

endifl

iflref(feed_fluegas) &&ref(drain_fluegas) then

hh128: Q_fluegas_conv  =       Q_gw_conv+Q_gc_conv+Q_loss_conv;

endifl
```

```
ifl!ref(feed_fluegas) &&!ref(drain_fluegas) then

u6fl:    Q_fluegas_conv        =       0.0;
u6h6l: Q_gc_conv               =       0.0;
uk6l:    Q_gw_conv             =       0.0;

endifl

ifl     wall_type == membrane  then

Q_lossa: Q_loss_conv   =       k_we * A_wt *dt_log_we*fouling_factor
;
# fouling factor FDBR 6.1.3 - 2/2

endifl

ifl     wall_type == adiabetic then

Q_lossb: Q_loss_conv   =       0;

endifl

# Surface Temperatures
# VDI 2010 C2 - Eq6

# Wall Tubes
#-------------------------

temp_wt_surfac6e:      temp_wt_surface    =    feed_fluegas.t - k_gw *
dt_log_gw / alpha_conv_gw  ;

# Carrier Tubes
#-------------------------

temp_ct_surfacke:      temp_ct_surface  =     feed_fluegas.t - k_gc *
dt_log_gc / alpha_conv_gc;

# Wall-Environment Surface
#-------------------------

temp_we_surfac6e:      temp_we_surface =      Ambient.t  + k_we *
dt_log_we / alpha_conv_we;


########################
#       K VALUES    #
########################

# Determination of k_gw
# FDBR 7.2.2 - 19/6

k_gw:   1/k_gw         =       1/alpha_conv_gw + d_wt*0.5 /
lambda_steel_wt * ln( d_wt / (d_wt-2*tube_thickness_wt) ) +
d_wt/alpha_in_wt/(d_wt-2*tube_thickness_wt);

# Determination of k_we

ifl     isolation == without   then

k_we1:  1/k_we
        =1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+1/alpha_in
_wt;

gd8:                    iso_thickn_one =       0;
gd9:                    iso_thickn_two =       0;
gd10:                   iso_thickn_three=      0;
gd11:                   lambda_iso_one =       0;
gd12:                   lambda_iso_two =       0;
gd13:                   lambda_iso_three=      0;

endifl

ifl     isolation == one       then

k_we2:  1/k_we         =
        1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+
iso_thickn_one/lambda_iso_one+1/alpha_in_wt;
gd14:                   iso_thickn_two         =       0;
gd15:   iso_thickn_three       =       0;
gd16:   lambda_iso_two         =       0;
gd17:   lambda_iso_three       =       0;

endifl

ifl     isolation == two       then

k_we3:  1/k_we  =1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+
iso_thickn_one/lambda_iso_one+
iso_thickn_two/lambda_iso_two+1/alpha_in_wt;
gd18:   iso_thickn_three       =       0;
gd19:   lambda_iso_three       =       0;

endifl

ifl     isolation == three     then

k_we4:  1/k_we         =
        1/alpha_conv_we+tube_thickness_wt/lambda_steel_wt+
iso_thickn_one/lambda_iso_one+
iso_thickn_two/lambda_iso_two+iso_thickn_three/lambda_iso_three+1/al
pha_in_wt;
```

```
endifl

# Determination of k_gc

k_gc:   1/k_gc          =       1/alpha_conv_gc + d_ct*0.5 /
lambda_steel_ct * ln( d_ct / (d_ct-2*tube_thickness_ct) ) +
d_ct/alpha_in_ct/(d_ct-2*tube_thickness_ct);

########################
#        DENSITY       #
########################

g4df:   density_water_wt        =       1/vm_wt;
gduf:   density_water_ct        =       1/vm_ct;
jhud:   density_air             =Ambient.density_air;
wrw:    density_fluegas         =       1/vm_fluegas;

########################
#        VELOCITY      #
########################

vjfg:    n_wt*( sqr(d_wt-2*tube_thickness_wt) * 3.14159265359 /4) *
w_wt    = feed_wt.massflow *vm_wt ;
v2kj:    n_ct*( sqr(d_ct-2*tube_thickness_ct) * 3.14159265359 /4) *
w_ct    = feed_ct.massflow *vm_ct ;
v4:      (width_duct*depth_duct - 0)*w_fluegas
        = feed_fluegas.massflow * vm_fluegas;
ujrt:   w_air = 0.65 * sqrt( 9.81 * L_wt * (drain_wt.t-Ambient.t) /
(273 + Ambient.t) );
#A_proj = 0

########################
#     AVERAGE VALUES   #
########################

a1:     tm_fluegas      = (feed_fluegas.t+drain_fluegas.t)*0.5;
ar2:    tm_water_ct     = (feed_ct.t+drain_ct.t)*0.5;
au2:    tm_water_wt     = (feed_wt.t+drain_wt.t)*0.5;

dgi:    pm_fluegas      = (feed_fluegas.p+drain_fluegas.p)*0.5;
dgi2:   pm_water_ct     = (feed_ct.p+drain_ct.p)*0.5;
dgi3:   pm_water_wt      = (feed_wt.p+drain_wt.p)*0.5;

a4:     vm_fluegas      = (feed_fluegas.v+drain_fluegas.v)*0.5;
ahz3:   vm_ct           = (feed_ct.v+drain_ct.v  )*0.5;
ae3:    vm_wt           = (feed_wt.v+drain_wt.v  )*0.5;
#jr:    u_mean_bt               = w_bt;
#kr_:   u_mean_ct               = w_ct; # Presumptions
#rz:    u_mean_wt               = w_wt;

# Wall Tubes
cp_8:   cp_water_wt     =       10^3*f_cp_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);
visc_water3:    dyn_visc_water_wt       = 10^-6*f_eta_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water3: lambda_water_wt          = f_lambda_pt(pm_water_wt,
tm_water_wt,1,0,0,0,0,0,0,0,0,0,0,0,0);

# Carrier Tubes
cp_7:   cp_water_ct     =       10^3*f_cp_pt(pm_water_ct,
tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);
visc_water2:    dyn_visc_water_ct       = 10^-
6*f_eta_pt(pm_water_ct,tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water2: lambda_water_ct          = f_lambda_pt(pm_water_ct,
tm_water_ct,1,0,0,0,0,0,0,0,0,0,0,0,0);


# Determination of pr_fluegas
# FDBR 5 - 2/1
Prandtl_fluegas:pr_fluegas
        =cp_fluegas*dyn_visc_fluegas/lambda_fluegas;

cp_3:           cp_fluegas      =       10^3*heatCapacity(
tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yAr)/2,(feed_flue
gas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluegas.Gas.yC2H6+dra
in_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain_fluegas.Gas.yC
3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)/2,(feed_fluega
s.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.yCO2+drain_flue
gas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.Gas.yH2)/2,(feed
_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_fluegas.Gas.yH2S+d
rain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain_fluegas.Gas.yH
Cl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/2,(feed_fluegas
.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.yN2O+drain_flueg
as.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.Gas.yNH3)/2,(fee
d_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_fluegas.Gas.yO2+dra
in_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fluegas.Gas.ySO2)
/2);
visc_fluegas:   dyn_visc_fluegas        =               10^-
6*dynViscosity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yA
r)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluega
s.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain
_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)
/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.
yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.G
as.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_flu
egas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain
_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/
2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.y
N2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.G
as.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_flue
```

```
gas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fl
uegas.Gas.ySO2)/2);
Lambda_fluegas:       lambda_fluegas        =
       heatConductivity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_flueg
as.Gas.yAr)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(fe
ed_fluegas.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC
3H8+drain_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.
Gas.yCH4)/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_flu
egas.Gas.yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_
fluegas.Gas.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,
(feed_fluegas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.y
HCl+drain_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.G
as.yHCN)/2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_flue
gas.Gas.yN2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_
fluegas.Gas.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(
feed_fluegas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2
+drain_fluegas.Gas.ySO2)/2);

# Determination of pr_air
# FDBR 5 - 2/1
pr_air: pr_air  =             cp_air*dyn_visc_air/lambda_air/100;

# This pr formula is for gas mixtures
cp_2:   cp_air
       =10^3*heatCapacity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.
0397,0.000055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
visc_air:             dyn_visc_air          =             10^-
6*dynViscosity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.0397,0.0000
55,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
Lambda_air:           lambda_air                  =
       heatConductivity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.03
97,0.000055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);

# Determination of alpha_in_ct
# FDBR 7.2.2 - 2/2
# Determination of f_2_ct according to Re_ct_in
#FDBR 7.2.2 - 1/4

gdf55g:       f_2_ct        =      1.85*(Re_ct_in^-0.05 -
180*Re_ct_in^-0.8);
Re_ct_in:     Re_ct_in      =      density_water_ct*w_ct*(d_ct-
2*tube_thickness_ct)/dyn_visc_water_ct;

alpha_in5_ct:
if      Re_ct_in < 10000      then
alpha_in_ct
       =f_1_ct*f_2_ct*0.02*(lambda_water_ct^0.58)*(dyn_visc_water_ct
^(-0.38))*(cp_water_ct^0.42)*((feed_ct.massflow / (3.14159265359
*n_ct* sqr(d_ct-2*tube_thickness_ct) /4))^0.8)*(d_ct-
2*tube_thickness_ct)^(-0.2);
# additionaly multiplied by f_2_ct
# FDBR 7.2.2 - 2/2
                                                        else
alpha_in_ct    =
       f_1_ct*0.02*(lambda_water_ct^0.58)*(dyn_visc_water_ct^(-
0.38))*(cp_water_ct^0.42)*((feed_ct.massflow / (3.14159265359 *n_ct*
sqr(d_ct-2*tube_thickness_ct) /4))^0.8)*(d_ct-
2*tube_thickness_ct)^(-0.2);
# Determination of alpha_i_8
# FDBR 7.2.2 - 1/5

# Determination of Nus_ct_out
#VDI 2010 F2 Eq3

nusselt_ct_out:       Nus_ct_out            =
       sqr(0.825+0.387*(Ra_ct*f1pr_ct)^(1/6))+0.97*L_ct/d_ct;
#0.97*h/d added for vertical cylinder
f1pr_ct:f1pr_ct=      (1+(0.492/pr_fluegas)^(9/16))^(-16/9);
Ra_ct:  Ra_ct   =     9.81*(L_ct^3)*(1/(feed_fluegas.t+273.15))*
(feed_fluegas.t-
drain_ct.t)/(dyn_visc_fluegas/density_fluegas)/therm_diff_ct;
#temp_ct_surface as drain_ct.t assumed due to conflict in program
therm_diff_ct: therm_diff_ct  =
       lambda_fluegas/cp_fluegas/density_fluegas;

# Determination of Nu
# VDI 2010 F2 - Eq3

# Determination of alpha_conv_gc

alpha_conv_gc: alpha_conv_gc  =Nus_ct_out*lambda_fluegas/L_ct;

# Determination of alpha_in_wt
# FDBR  7.2.2 - 2/1 & 2/2
# Determination of f_2_wt according to Re_wt_in
# FDBR  7.2.2 - 1/4

gdf5u5g:      f_2_wt        =      1.85*(Re_wt_in^-0.05 -
180*Re_wt_in^-0.8);
reynold_wt:   Re_wt_in      =      density_water_wt*w_wt*(d_wt-
2*tube_thickness_wt)/dyn_visc_water_wt;

alpha_in_wt:
if      Re_wt_in < 10000      then
alpha_in_wt    =
       f_1_wt*f_2_wt*0.02*(lambda_water_wt^0.58)*(dyn_visc_water_wt^
-0.38)*(cp_water_wt^0.42)*((feed_wt.massflow / (3.14159265359 *n_wt*
sqr(d_wt-2*tube_thickness_wt) /4))^0.8)*((d_wt-
2*tube_thickness_wt)^-0.2);
                                    else
alpha_in_wt    =
       f_1_wt*0.02*(lambda_water_wt^0.58)*(dyn_visc_water_wt^-
```

```
0.38)*(cp_water_wt^0.42)*((feed_wt.massflow / (3.14159265359 *n_wt*
sqr(d_wt-2*tube_thickness_wt) /4))^0.8)*((d_wt-
2*tube_thickness_wt)^-0.2);

# Determination of Nus_we_0

Nu_wt_s0:      Nu_we_0        =       ( sqr(Nu_we_turb)+
sqr(Nu_we_lam))^0.5;
Nu_wt_elam:    Nu_we_lam      =       0.664* (Re_we^0.5 * pr_air/3);

Nu_wtk_turb:   Nu_we_turb     =       0.037*(Re_we^0.8)*pr_air /
(1+(pr_air^(2/3)-1)*2.443*Re_we^-0.1);
re_wht:        Re_we   = w_fluegas*L_wt/(dyn_visc_air/density_air);

# Determination of alpha_conv_we

alpha_conv_we: alpha_conv_we  =       Nus_we_0*lambda_air/L_wt;

# Determination of Nu_wt_0
#VDI 2010 G4 - Eq9

Nu_wt_0:       Nu_wt_0        =       ( sqr(Nu_wt_turb)+
sqr(Nu_wt_lam))^0.5;
# VDI 2010 G4 - Eq9
Nu_wt_lam:     Nu_wt_lam      =       0.664* (Re_wt)^0.5 *
pr_fluegas^(1/3);
#VDI 2010 G4 - Eq3
Nu_wt_turb:    Nu_wt_turb     =       0.037*(Re_wt^0.8)*pr_fluegas /
(1+(pr_fluegas^(2/3)-1)*2.443*Re_wt^-0.1);
#VDI 2010 G4 -Eq8
# Determination of Re
# VDI 2010 G4 - 2.1 - Eq1
re_wt:  Re_wt   =w_fluegas*L_wt/(dyn_visc_fluegas/density_fluegas);

# Determination of Nu
# VDI 2010 F2 - Eq3

# Determination of alpha_conv_gw

alpha_conv_gw: alpha_conv_gw  =Nu_wt_0*lambda_fluegas/L_wt;

zetue:  Q_fluegas_tot  =Q_fluegas_rad+Q_fluegas_conv;
sege:   Q_fluegas_tot_ =feed_fluegas.massflow*(feed_fluegas.h-
drain_fluegas.h)*1000;

########################
#        TESTS         #
########################

t4:     test (Re_wt< 10^7)     error   "it has to be Re_wt<10^7";
t3:     test (Re_wt> 10)       error   "it has to be Re_wt>10";

jt4:    test (Re_we< 10^7)     error   "it has to be Re_we<10^7";
t3t:    test (Re_we> 10)       error   "it has to be Re_we>10";
tt1t:   test (dt_in_we>0.0)            warning "negative";
t2t9:   test (dt_out_we>0.0)           warning "negative";
t1t9:   test (dt_in_gw>0.0)            warning "negative";
t2t7:   test (dt_out_gw>0.0)           warning "negative";
t1t7:   test (dt_in_gc>0.0)            warning "negative";
t26:    test (dt_out_gc>0.0)           warning "negative";
t23:    test (Q_gc_conv>0.0)           warning "negative";
tj3:    test (Q_gw_conv>0.0)           warning "negative";
kt3:    test (Q_loss_conv>=0.0)        warning "negative";
tk3:    test (Q_fluegas_conv>0.0)      warning "negative";


# Testing Rayleigh Numbers
#VDI 2010 F2 -1

op4pp:  test( Ra_ct >= 10^-1)  warning "has to be Ra_ct >= 10^-1";
zuuuu:  test( Ra_ct <= 10^12)  warning "has to be Ra_ct <= 10^12";

# Testing Prandtl Numbers

t1:     test(0.5< pr_fluegas)  error   "it has to be
0.5<prandtl_fluegas";
t42:    test(2000> pr_fluegas) error   "it has to be
2000>prandtl_fluegas";
klg:    test( pr_air > 0.001)  warning "has to be pr_air>0.001";
#kl1g:  test( pr_fluegas > 0.001)     warning "has to be
pr_fluegas>0.001";
```

UENER_BLANK - RADIATION

```
########################
#   Partial Pressures   #
########################

pH2O1: p_H2O    =       feed_fluegas.p * feed_fluegas.Gas.yH2O;
pCO21: p_CO2    =       feed_fluegas.p * feed_fluegas.Gas.yCO2;

##-------------------------

ps_x:    ps     =       p_H2O * s_eq_wt;

tr1f1: if               ps < 0.5        then

emiss_H2O       =       (0.43265 -0.1089 * (feed_fluegas.t+273.15) /
1000) - (0.3273 -0.043821 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-3.5829 * p_H2O*s_eq_wt);

else


emiss_H2O       =       (0.66439 -0.17389 * (feed_fluegas.t+273.15) /
1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^(-0.84652 * p_H2O*s_eq_wt);

ifl fluegas == water_vapor then

emissivity_wv_x:        emissivity_gas          =       emiss_H2O;
A_v_water_x:    A_v                             =       emiss_H2O *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.45;

tt7:            test (feed_fluegas.t < (1500-273.15) )        error
"feed_fluegas.t has to be between 700 and 1500 K";
tt8:            test (feed_fluegas.t > (700-273.15) )         error
"feed_fluegas.t has to be between 700 and 1500 K";
ttgo:           test (p_H2O*s_eq_wt < 0.5 )                   error
"p_H2Oxs_eq_wt has to be between 0.05 and 0.5 mbar or 0.5-2";
tjtz:           test (p_H2O*s_eq_wt >0.05 )                   error
"p_H2Oxs_eq_wt has to be between 0.05 and 0.5 mbar or 0.5-2";
tt5to:          test (p_H2O*s_eq_wt < 2 )                     error
"p_H2Oxs_eq_wt has to be between 0.05 and 0.5 mbar or 0.5-2";
tt2zk:          test (p_H2O*s_eq_wt >0.5 )                    error
"p_H2Oxs_eq_wt has to be between 0.05 and 0.5 mbar or 0.5-2";

endifl

# Analytical calculation of the emissivities of water vapor and
carbondioxide and their mixtures
# VDI 2010 K3 - 5

emissivitgfd: emissivity_co2 = Z - gw_tot;
gsg:    Z = 0.27769 + 0.03869 * (feed_fluegas.t+273.15) / 1000 +
1.4249 * 10^-5 * ((feed_fluegas.t+273.15) / 1000)^2;

fgdg:   fpco2           =       1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_wt)));

ifl fluegas == carbondioxide then

emissivity_gasco2x: emissivity_gas    =       emissivity_co2;
A_v_carbondioxidex: A_v               =       fpco2*emissivity_co2 *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;

tt3:            test (feed_fluegas.t >(300 - 273.15) )        error
"feed_fluegas.t has to be between 300 and 1800 K";
tt4:            test (feed_fluegas.t <(1800-273.15) )         error
"feed_fluegas.t has to be between 300 and 1800 K";
tt5:            test (p_CO2*s_eq_wt < 10 )                    error
"p_CO2xs_eq_wt has to be between 0.01 and 10 mbar";
tt6:            test (p_CO2*s_eq_wt >0.01 )                   error
"p_CO2xs_eq_wt has to be between 0.01 and 10 mbar";

endifl

k78k1:  A_v_water_vapor         =       emiss_H2O * (
(feed_fluegas.t+273.15) / (temp_wt_surface+273.15))^0.45;

k8k2:   A_v_carbondioxide       =       fpco2*emissivity_co2 *
((feed_fluegas.t+273.15)/(temp_wt_surface+273.15))^0.65;

ifl fluegas == mixture  then

zteg:   emissivity_gas          =       (0.13 + 0.265 * (feed_fluegas.t
+ 273.15) / 1000) * (1- ( 2.718281828^(0*(p_H2O+p_CO2)*s_eq_wt))) +
(0.595 -0.15 * (feed_fluegas.t + 273.15) / 1000) * (1 - (
2.718281828^(-0.824*(p_H2O+p_CO2)*s_eq_wt))) + (0.275 -0.115 *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_wt)));

A_vx:   A_v                     =       A_v_water_vapor +
A_v_carbondioxide - d_emissivity_w;

#tt9:           test (feed_fluegas.t <(1800-273.15) ) error
"feed_fluegas.t has to be between 1800 and 1100 K";
#tt10:  test (feed_fluegas.t >(1100-273.15) ) error   "feed_fluegas.t
has to be between 1800 and 1100 K";
tt13z1: test (s_eq_wt > 0.2 )   error   "s_eq_wt has to be between 0.2
and 6 m";
tt12z2: test (s_eq_wt < 6 )     error   "s_eq_wt has to be between 0.2
and 6 m";
```

```
@#hzd5gt:test (p_H2O/p_CO2 == 1 )     warning  "p_H2O/p_CO2 has to be
= 1 for gray- and -clear gas approximation";

endifl

# Determination of emissivity (radiative exchange between gas and
wall)          !!! for bundle !!!
# Determination of A_v (Absorption Behaviour)


t8sttt: if     feed_fluegas.t >= (700-273.15) then
B              =       0.225* sqr((feed_fluegas.t+273.15)/1000);
       else
B              =       0.054* sqr((feed_fluegas.t+273.15)/1000);

ag1:    A              =
       (((0.1*((feed_fluegas.t+273.15)/1000)^-
1.45+1)*feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23)/(0.1*(fee
d_fluegas.p/1000)^-
1.45+feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23));

# Analytical calculation of the emissivities of water vapor and
carbondioxide and their mixtures
# VDI 2010 K3 5

# Determination of alpha_rad x,y,z for wall ( wall pipes are as
'cuboidal gas body' taken)
# FDBR Fig. 7.1.1 - 8 for s_eq_wt_x _y _z


# Absorption Behaviour of Wall is taken as 0.85
#alpha_rad formulas FDBR 7.1.3 - 1/2
alpha_rad_gw_x:        alpha_rad_gw          =       (2*0.85*1 /
(1+0.85*1)) * radiation_coefficient*10^8*(
emissivity_gas*((feed_fluegas.t+273.15)/100)^4 -
A_v*((temp_wt_surface+273.15)/100)^4 )/(feed_fluegas.t -
temp_wt_surface);
# Radiation coefficient is multiplied by 10^8

# Due to assumption of bundle and carrier tubes together for
radiation calculations, in this model radiation calculatedjust for
wall tubes.(Note: Bundle tubes doesn't exist in the blank area)

# VDI 2010 K3 Eq18
drt:    Q_gw_rad              =
       radiation_coefficient*A_wt*emiss_wall*(emissivity_gas*(feed_f
luegas.t+273.15)^4-A_v*(temp_wt_surface+273.15)^4)/(1-(1-
emiss_wall)*(1-A_v));

jhfgh:  Q_fluegas_rad  =       Q_gw_rad;
```

```
########################
#         TESTS#
########################


tj3:    test(Q_gw_rad>0.0)            warning "negative";
tk3:    test(Q_fluegas_rad>0.0)       warning "negative";
ttgrt: test(s_eq_wt>0)        warning "negative";

gfg5:    a1 = 0.1074 - 0.10705 * (feed_fluegas.t+273.15) / 1000 +
0.072727 * ((feed_fluegas.t+273.15) / 1000)^2;
jet:     a2 = 0.027237 + 0.10127 * (feed_fluegas.t+273.15) / 1000 -
0.043773 * ((feed_fluegas.t+273.15) / 1000)^2;
jfz6:    a3 = 0.058438 - 0.001208 * (feed_fluegas.t+273.15) / 1000 +
0.0006558 * ((feed_fluegas.t+273.15) / 1000)^2;
i75:     a4 = 0.019078 + 0.037609 * (feed_fluegas.t+273.15) / 1000 -
0.015424 * ((feed_fluegas.t+273.15) / 1000)^2;
hfgh6: a5 = 0.056993 - 0.025412 * (feed_fluegas.t+273.15) / 1000 +
0.0026167 * ((feed_fluegas.t+273.15) / 1000)^2;
jghj8: a6 = 0.0028014 + 0.038826 * (feed_fluegas.t+273.15) / 1000 -
0.020198 * ((feed_fluegas.t+273.15) / 1000)^2;

gh4:     k1 = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);
h5:      k2 = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);
hh6:     k3 = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);
jf7:     k4 = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);
oi7:     k5 = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);
pf8:     k6 = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_wt);

jfg2: gw_tot = a1*k1+a2*k2+a3*k3+a4*k4+a5*k5+a6*k6;
```

UENER_ECONOMIZER - GEOMETRY

```
########################
#    BASIC EQUATIONS    #
########################

er42:   feed_fluegas.massflow         = drain_fluegas.massflow;

########################
#       BUNDLE TUBES   #
########################

ifl     tube_outlet_side == same_side        then
hsdf5:  s = 1;
endifl

ifl     tube_outlet_side == opposite_side    then
hsdf6:  s = 2;
endifl

A_btb:  A_bt           =      (L_bt*n_bt_y+(L_bt-
s)*tube_pitch_long*0.5*3.14159265359)*n_bt_z * (d_bt *
3.14159265359) ;
Lbtto5t:       L_bt_tot               =      (L_bt*n_bt_y+(L_bt-
s)*tube_pitch_long*0.5*3.14159265359)*n_bt_z;
nbtsf:  n_bt                          =       n_bt_y*n_bt_z;
erthzte:       height_bundle          = height_duct;


ifl tube_type == fin_tubes    then

gdc:           dG = a/3.14159265359;
gdgg4:         fins_pitch= 1/tube_fin_number;
gg6:           phi_fluegas    =feed_fluegas.massflow /
(width_duct*depth_duct*fq-A_proj);

endifl

ifl tube_type == smooth_tubes then

gdc2:          dG = d_bt;
gg5:           phi_fluegas    =      feed_fluegas.massflow /
(depth_duct*width_duct-A_proj);
b4f:           tube_fin_number = 0;
b3c:           fins_pitch = 0;
b3e:           fin_height = 0;
b4e:           fin_thickness = 0;

endifl
```

```
hdh8:   fq = 1-
(d_bt+2*tube_fin_number*fin_height*fin_thickness)/tube_pitch_trans;
gg7:    a_r =
2*3.14159265359*tube_fin_number*fin_height*(d_bt+fin_height);
gg8:    a_ko =
3.14159265359*tube_fin_number*fin_thickness*(d_bt+2*fin_height);
gg9:    a_ro = 3.14159265359*tube_fin_number*d_bt*(fins_pitch-
fin_thickness);
gg20:   a = a_r+a_ko+a_ro;

h11:    A1 * d_bt      = tube_pitch_trans;
h12a:   B1 * d_bt      = tube_pitch_long;
p4e:    C1             = sqrt( sqr(A1/2) + sqr(B1));

am_tube:       am_tube =       ((d_bt *  3.14159265359)-((d_bt-
2*tube_thickness_bt) * 3.14159265359)) / ln((d_bt *  3.14159265359)
/(d_bt-2*tube_thickness_bt) * 3.14159265359);
# FDBR 6.1.2 - 2/4

f_1_bt: f_1_bt         =      1+((d_bt-
2*tube_thickness_bt)/L_bt)^(2/3);

gee:    porosity       =      (depth_duct*width_duct*height_bundle-
3.14159265359*0.25*d_bt^2*L_bt_tot)/(depth_duct*width_duct*height_bu
ndle);

h14:    l              = 3.14159265359*0.5*d_bt;

f14:    f1             = 4/3.14159265359*A1*porosity;

ifl tube_arrangement == in_line then

g5h:    fe_            =1+(1.9-1.8/B1)*(1/(f1-0.4));

endifl

ifl tube_arrangement == staggered then

gdf5:   if B1<5 && f1>=3.04 then   fe_= 1+1.53/(f1-0.4);

        else fe_= 1+(1.87-1.7/B1)*(1/(f1-0.4));

endifl

gdfg:   test(B1<6)     warning "B1 must be < 6";
g5f5:   test(fe_<=2+f1)   warning "fe_ must be <=2+f1";
#g7fg:  test(fe_>=1.58)       warning "fe_ must be >=1.58";

fe4:    fe     =       fe_*porosity^-0.6;

# Projected Areas and Volumes for Radiation
```

```
#-----------------------------------------

V_duct: V_duct  =height_duct*depth_duct*width_duct;
V_tubes:V_tubes        =       L_bt_tot*3.14159265359*sqr(d_bt) /4  ;
b8a:    A_proj         =       n_bt_z*L_bt*dG;

# Determination of s_eq
# During the determination of s_eq for Carrier and Bundle Tubes,
they are evaluated together
#f3=0.85 for both bundles and walls (cuboid)
# FDBR7.1.1 - 6/1
s_eq_b: s_eq_b         =       0.85*4*(V_duct - V_tubes) / ( L_bt_tot
* d_bt * 3.14159265359 ) ;


########################
#          TESTS #
########################

t5:     test (tube_pitch_long*0.5 > fin_height) error " You must
change the number of gilles, or gilles high";
hd:     test(dG>0);
hfd:    test(d_bt>0);
```

UENER_ECONOMIZER - CONVECTION

```
########################
#    BASIC EQUATIONS    #
########################

# Mass Balance Equations
e1:     feed_bundle.massflow   = drain_bundle.massflow;
er42:   feed_fluegas.massflow  = drain_fluegas.massflow;


# Pressure Drops
f3:     feed_bundle.p  -delta_p_bundle       = drain_bundle.p;
f4:     feed_fluegas.p -delta_p_fluegas      = drain_fluegas.p;

# Temperature Differences according to Current Directions
# FDBR 6.2.2 - 5/1

ifl     current_type_bt == concurrent        then
f7co:           abs(feed_fluegas.t- feed_bundle.t)   =
        dt_in_gb;
f8_co:          abs(drain_fluegas.t- drain_bundle.t) =
        dt_out_gb ;

endifl

ifl     current_type_bt == countercurrent    then

fl7_counter:    abs(drain_fluegas.t- feed_bundle.t)    =
        dt_in_gb ;
f8r_counter:    abs(feed_fluegas.t- drain_bundle.t)  =
        dt_out_gb;

endifl

# Logarithmic Temperature Differences
# FDBR 6.2.2 - 5/6

dt_log_gb :     dt_log_gb       =       (dt_in_gb-dt_out_gb) /
        ln(dt_in_gb/dt_out_gb);

# Energy Equations

j7jg5: Q_gb_conv_ = feed_bundle.massflow*(drain_bundle.h-
feed_bundle.h)*1000;

ifl tube_type == smooth_tubes then

h21h218:        Q_gb_conv       =       k_gb * A_bt * dt_log_gb
*fouling_factor ;

L_fin:  L_fin = 0;

endifl

ifl tube_type == fin_tubes then

gsh4:           Q_gb_conv       =       k_gb * a * L_fin * dt_log_gb;

endifl

ifl!ref(feed_bundle) &&!ref(drain_bundle) then

u261:           Q_gb_conv       =       0.0;

endifl

iflref(feed_fluegas) &&ref(drain_fluegas) then

hh128:  Q_fluegas_conv          =       Q_gb_conv+Q_loss_conv;

endifl

ifl!ref(feed_fluegas) &&!ref(drain_fluegas) then


u6fl:   Q_fluegas_conv          =       0.0;

u34d61: Q_gb_conv               =       0.0;

endifl

# Surface Temperatures
#VDI 2010 C2 - Eq6

# Bundle Tubes

temp_bt_surfa3ce:       temp_bt_surface  =  feed_fluegas.t - k_gb *
dt_log_gb / alpha_conv_gb;

########################
#        K Values #
########################

# Determination of k_gb

ifl tube_type == fin_tubes then

tehfs: 1/k_gb  = 1/alpha_conv_gb + (dG * ln(d_bt / (d_bt -
2*tube_thickness_bt))/lambda_fin/2) + dG / (d_bt -
2*tube_thickness_bt) / alpha_in_bt ;
```

```
endifl

ifl tube_type == smooth_tubes then

r4:     1/k_gb
        =1/alpha_in_bt+1/alpha_conv_gb+tube_thickness_bt/lambda_steel
_bt;

endifl

########################
#        DENSITY       #
########################

gdf:            density_water_bt        =       1/vm_bt;
jhud:           density_air             =       Ambient.density_air;
wrw:            density_fluegas         =       1/vm_fluegas;

########################
#        VELOCITY      #
########################

v2:      n_bt*( sqr(d_bt-2*tube_thickness_bt) * 3.14159265359 /4) *
w_bt    = feed_bundle.massflow *vm_bt ;
v3:      (width_duct*depth_duct - A_proj) * w_fluegas
        = feed_fluegas.massflow *vm_fluegas;

########################
#  AVERAGE VALUES      #
########################

a1:     tm_fluegas      = (feed_fluegas.t+drain_fluegas.t)*0.5;
a2:     tm_water_bt     = (feed_bundle.t+drain_bundle.t)*0.5;
a4:     vm_fluegas      = (feed_fluegas.v+drain_fluegas.v)*0.5;
a3:     vm_bt           = (feed_bundle.v+drain_bundle.v  )*0.5;
dgi:    pm_fluegas      = (feed_fluegas.p+drain_fluegas.p)*0.5;
dgi1:   pm_water_bt     = (feed_bundle.p+drain_bundle.p)*0.5;

# Bundle Tubes

cp_1:   cp_water_bt
        =10^3*f_cp_pt(pm_water_bt,tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0
,0);
visc_water1:    dyn_visc_water_bt= 10^-
6*f_eta_pt(pm_water_bt,tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0,0);
Lambda_water1: lambda_water_bt=       f_lambda_pt(pm_water_bt,
tm_water_bt,1,0,0,0,0,0,0,0,0,0,0,0,0);

# Determination of pr_fluegas
# FDBR 5 - 2/1

Prandtl_fluegas:        pr_fluegas      =
        cp_fluegas*dyn_visc_fluegas/lambda_fluegas;

cp_3:   cp_fluegas      =               10^3*heatCapacity(
tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yAr)/2,(feed_flue
gas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluegas.Gas.yC2H6+dra
in_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain_fluegas.Gas.yC
3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)/2,(feed_fluega
s.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.yCO2+drain_flue
gas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.Gas.yH2)/2,(feed
_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_fluegas.Gas.yH2S+d
rain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain_fluegas.Gas.yH
Cl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/2,(feed_fluegas
.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.yN2O+drain_flueg
as.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.Gas.yNH3)/2,(fee
d_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_fluegas.Gas.yO2+dra
in_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fluegas.Gas.ySO2)
/2);
visc_fluegas:  dyn_visc_fluegas                 =               10^-
6*dynViscosity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_fluegas.Gas.yA
r)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(feed_fluega
s.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC3H8+drain
_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.Gas.yCH4)
/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_fluegas.Gas.
yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_fluegas.G
as.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,(feed_flu
egas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.yHCl+drain
_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.Gas.yHCN)/
2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_fluegas.Gas.y
N2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_fluegas.G
as.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(feed_flue
gas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2+drain_fl
uegas.Gas.ySO2)/2);
Lambda_fluegas:         lambda_fluegas          =
        heatConductivity(tm_fluegas,(feed_fluegas.Gas.yAr+drain_flueg
as.Gas.yAr)/2,(feed_fluegas.Gas.yC2H4+drain_fluegas.Gas.yC2H4)/2,(fe
ed_fluegas.Gas.yC2H6+drain_fluegas.Gas.yC2H6)/2,(feed_fluegas.Gas.yC
3H8+drain_fluegas.Gas.yC3H8)/2,(feed_fluegas.Gas.yCH4+drain_fluegas.
Gas.yCH4)/2,(feed_fluegas.Gas.yCO+drain_fluegas.Gas.yCO)/2,(feed_flu
egas.Gas.yCO2+drain_fluegas.Gas.yCO2)/2,(feed_fluegas.Gas.yH2+drain_
fluegas.Gas.yH2)/2,(feed_fluegas.Gas.yH2O+drain_fluegas.Gas.yH2O)/2,
(feed_fluegas.Gas.yH2S+drain_fluegas.Gas.yH2S)/2,(feed_fluegas.Gas.y
HCl+drain_fluegas.Gas.yHCl)/2,(feed_fluegas.Gas.yHCN+drain_fluegas.G
as.yHCN)/2,(feed_fluegas.Gas.yN2+drain_fluegas.Gas.yN2)/2,(feed_flue
gas.Gas.yN2O+drain_fluegas.Gas.yN2O)/2,(feed_fluegas.Gas.yNH3+drain_
fluegas.Gas.yNH3)/2,(feed_fluegas.Gas.yNO+drain_fluegas.Gas.yNO)/2,(
feed_fluegas.Gas.yO2+drain_fluegas.Gas.yO2)/2,(feed_fluegas.Gas.ySO2
+drain_fluegas.Gas.ySO2)/2);

# Determination of pr_air
```

```
pr_air:        pr_air                =
       cp_air*dyn_visc_air/lambda_air/100;
cp_2:          cp_air                =
       10^3*heatCapacity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.0
397,0.000055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
visc_air:      dyn_visc_air          =                10^-
6*dynViscosity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.0397,0.0000
55,0,0,0,0,78.084,0.0000325,0,0,20.946,0);
Lambda_air:    lambda_air            =
       heatConductivity(Ambient.t,0.9340,0,0,0,0.000179,0.00001,0.03
97,0.000055,0,0,0,0,78.084,0.0000325,0,0,20.946,0);

# Determination of alpha_in_bt
reynold_bt_in: Re_bt_in              =
       density_water_bt*w_bt*(d_bt-
2*tube_thickness_bt)/dyn_visc_water_bt;
gdf5g:         f_2_bt         =      1.85*(Re_bt_in^-0.05 -
180*Re_bt_in^-0.8);

alpha_in_bt:   if      Re_bt_in < 10000
then
alpha_in_bt
       =f_1_bt*f_2_bt*0.02*lambda_water_bt^0.58*dyn_visc_water_bt^-
0.38*cp_water_bt^0.42*(feed_bundle.massflow / (3.14159265359 *n_bt*
(d_bt-2*tube_thickness_bt)^2 /4))^0.8*(d_bt-2*tube_thickness_bt)^-
0.2;
else
alpha_in_bt    =f_1_bt*0.02*lambda_water_bt^0.58*dyn_visc_water_bt^-
0.38*cp_water_bt^0.42*(feed_bundle.massflow / (3.14159265359 *n_bt*
(d_bt-2*tube_thickness_bt)^2 /4) )^0.8*(d_bt-2*tube_thickness_bt)^-
0.2;

ifl    tube_arrangement == staggered then

gggsg:  C= 0.45;
endifl

ifl    tube_arrangement == in_line then

gggsg3: C= 0.3;
endifl

# Determination of alpha_conv_gb

ifl tube_type == smooth_tubes then

gsdf7: lambda_fin = 0 ;
hdfg: alpha_conv_fin = 0;
jf64:   fin_thickness = 0;
hg:     fin_height = 0;
gr7:    dG = d_bt;
fe3:    a_r     = 0;
ge2:    a_ko    = 0;
gr1:    a_ro    = 0;
gr:     a       = 0;
hhh54e: fG      =
(lambda_fluegas/lambda_air)^0.636*(cp_fluegas/cp_air)^0.364*(dyn_vis
c_fluegas/dyn_visc_air)^-0.236;
alpha_5:       alpha_conv_gb  = alpha_air*fG*fe;
t8tut:  alpha_air             =
0.287*lambda_air^0.636*cp_air^0.364*dyn_visc_air^-
0.236*phi_fluegas^0.6*d_bt^-0.4;
hhhi:          f_1            = 0;
tzi:           f_3            = 0;
hh9yyy:        fin_efficiency = 0;
Nus_egasg:     Nus_fluegas    =
0.287*Re_fluegas^0.6*pr_fluegas^0.364*fe;
regf0s: Re_fluegas             = phi_fluegas*d_bt/dyn_visc_fluegas;

endifl

ifl tube_type == fin_tubes then

hhh546: fG
       =(lambda_fluegas/lambda_air)^0.667*(cp_fluegas/cp_air)^0.333*
(dyn_visc_fluegas/dyn_visc_air)^-0.292;
alpha_6:       alpha_conv_fin =alpha_air*fG;
gggg000:       alpha_conv_gb
       =(alpha_conv_fin*a_ro+fin_efficiency*alpha_conv_fin*(a_r+a_ko
))/a;
gggg5:  alpha_air  =   C*lambda_air^0.667*cp_air^0.333*dyn_visc_air^-
0.292*phi_fluegas^0.625*dG^-0.375;
f_1z5:  f_1
       =(2*alpha_conv_gb/fin_thickness/lambda_fin)^0.5*fin_height;
f_3gs:  f_3     =      fin_thickness/fin_height;
hhyyy:  fin_efficiency =       ( tanh (f_1)+0.5*f_1*f_3) / f_1 /
(1+0.5*f_3) / (1+0.5*f_1*f_3* tanh (f_1))    ;
Nus_uegasg: Nus_fluegas        =
       alpha_conv_gb*dG/lambda_fluegas;
regfs:  Re_fluegas             =
       phi_fluegas*dG/dyn_visc_fluegas;

t_fin_efficiency:      test (fin_efficiency <=1)
       error "fin efficiency must be less than 1.0";

endifl

am_tube:       am_tube        =      ((d_bt *  3.14159265359)-
((d_bt-2*tube_thickness_bt) * 3.14159265359)) / ln((d_bt *
3.14159265359) /(d_bt-2*tube_thickness_bt) * 3.14159265359);

# FDBR 6.1.2 - 2/4
```

```
terju:  l               =       3.14159265359*0.5*d_bt;

########################
#          TESTS       #
########################

t15:    test (dt_in_gb>0.0)           warning "negative";
t32:    test (dt_out_gb>0.0)          warning "negative";
t263:   test (Q_gb_conv>0.0)          warning "negative";
tk3:    test (Q_fluegas_conv>0.0)     warning "negative";

# Testing Prandtl Numbers

klg:    test( pr_air > 0.001)         warning "has to be
pr_air>0.001";
kllg:   test( pr_fluegas > 0.001)     warning "has to be
pr_fluegas>0.001";
t1:     test(0.5< pr_fluegas)         error   "it has to be
0.5<prandtl_fluegas";
t42:    test(2000> pr_fluegas)        error   "it has to be
2000>prandtl_fluegas";

zetue:  Q_fluegas_tot  =      Q_fluegas_rad+Q_fluegas_conv;
sege:   Q_fluegas_tot_ =      feed_fluegas.massflow*
abs(feed_fluegas.h-drain_fluegas.h)*1000;
```

UENER_ECONOMIZER - RADIATION

```
########################
#   Partial Pressures   #
########################

pH2O1: p_H2O    =       feed_fluegas.p * feed_fluegas.Gas.yH2O;
pCO21: p_CO2    =       feed_fluegas.p * feed_fluegas.Gas.yCO2;


t8sttt: if      feed_fluegas.t >= (700-273.15) then
B       =       0.225* sqr((feed_fluegas.t+273.15)/1000);
else
B       =       0.054* sqr((feed_fluegas.t+273.15)/1000);

ag1:    A = (((0.1*((feed_fluegas.t+273.15)/1000)^-
1.45+1)*feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23)/(0.1*(fee
d_fluegas.p/1000)^-
1.45+feed_fluegas.p*(1+0.28*p_CO2/feed_fluegas.p)+0.23));

jwes:   fpco2_b =1 + (A-1) * 2.718281828^(-0.5* sqr(
log(B/100/p_CO2*s_eq_b)));

ps_b:    ps_b   =       p_H2O*s_eq_b;

trhp11: if              ps_b < 0.5      then

emiss_H2O_b             =       (0.43265 -0.1089 *
(feed_fluegas.t+273.15) / 1000) - (0.3273 -0.043821 *
(feed_fluegas.t+273.15) / 1000) * 2.718281828^( -3.5829 *
p_H2O*s_eq_b);

else

emiss_H2O_b     =       (0.66439 -0.17389 * (feed_fluegas.t+273.15) /
1000) - (0.4572 -0.1317 * (feed_fluegas.t+273.15) / 1000) *
2.718281828^( -0.84652 * p_H2O*s_eq_b);


ifl fluegas == water_vapor then

emissivity_wv:          emissivity_gas_b                =
        emiss_H2O_b;
A_v_water:              A_v_b                           =
        emiss_H2O_b *
((feed_fluegas.t+273.15)/((temp_bt_surface)+273.15))^0.45;
#emiss_H2O_b  can be readable from VDI 2010 - K3 - Fig 5b

ttrt:   test(0.05 < p_H2O*s_eq_b)                       error
        "ph20xs_eq must be between 0.05 and 2";

trt1:   test(p_H2O*s_eq_b < 2)                          error
        "ph20xs_eq must be between 0.05 and 2";

endifl

# Analytical calculation of the emissivities of water vapor and
carbondioxide and their mixtures
# VDI 2010 - 5
emihhet: emiss_co2_b = Z - g_tot;
gsg:    Z = 0.27769 + 0.03869 * (feed_fluegas.t+273.15) / 1000 +
1.4249 * 10^-5 * ((feed_fluegas.t+273.15) / 1000)^2;


#-------------------------

ifl fluegas == carbondioxide then

emissihtgsco2: emissivity_gas_b         =       emiss_co2_b;
A_ghrg:         A_v_b                   =       fpco2_b*emiss_co2_b *
((feed_fluegas.t+273.15)/((temp_bt_surface)+273.15))^0.65;
# emissivity_co2_b  can be readable from VDI 2010 - K3 - Fig 4

ttk3:   test (feed_fluegas.t >(300 - 273.15)) error  "feed_fluegas.t
has to be between 300 and 1800 K";
tkt4:   test (feed_fluegas.t <(1800-273.15) ) error  "feed_fluegas.t
has to be between 300 and 1800 K";
ttk5:   test (p_CO2*s_eq_b < 10 )             error  "p_CO2xs_eq_b
has to be between 0.01 and 10 mbar";
tkt6:   test (p_CO2*s_eq_b >0.01 )            error  "p_CO2xs_eq_b
has to be between 0.01 and 10 mbar";

endifl

kk3:    A_v_water_vapor_b       =       emiss_H2O_b *
((feed_fluegas.t+273.15)/((temp_bt_surface)+273.15))^0.45;
kk4:    A_v_carbondioxide_b     =       fpco2_b*emiss_co2_b *
((feed_fluegas.t+273.15)/((temp_bt_surface)+273.15))^0.65;

emige5: emissivity_gas_mix_b    =       (0.13 + 0.265 * (feed_fluegas.t
+ 273.15) / 1000) * (1- ( 2.718281828^(-1*0*(p_H2O+p_CO2)*s_eq_b)))
+ (0.595 + (-0.15) * (feed_fluegas.t + 273.15) / 1000) * (1 - (
2.718281828^(-0.824*(p_H2O+p_CO2)*s_eq_b))) + (0.275 + (-0.115) *
(feed_fluegas.t + 273.15) / 1000) * (1 - ( 2.718281828^(-
25.91*(p_H2O+p_CO2)*s_eq_b)));


ifl fluegas == mixture  then

emissivity_gas_b:emissivity_gas_b       =emissivity_gas_mix_b;
A_v:            A_v_b                   =A_v_water_vapor_b +
A_v_carbondioxide_b - d_emissivity_b;
```

```
# delta_emiss_steam_b VDI 2010 - K3 - Fig. 9a-c.

#tzt9:  test (feed_fluegas.t <(1800-273.15) ) error  "feed_fluegas.t
has to be between 1800 and 1100 K";
#ttz10: test (feed_fluegas.t >(1100-273.15) ) error  "feed_fluegas.t
has to be between 1800 and 1100 K";
#tt1z1: test (s_eq_b > 0.2 )                         error  "s_eq_b
has to be between 0.2 and 6 m";
#tt1z2: test (s_eq_b < 6 )                           error  "s_eq_b
has to be between 0.2 and 6 m";
@#hzdj5gt:test (p_H2O/p_CO2 == 1 )                   warning
"p_H2O/p_CO2 has to be = 1 for gray- and -clear gas approximation";
endifl

# Determination of alpha_rad_gb

alpha8_rad_gb: alpha_rad_gb   =(2*0.85*1 / (1+0.85*1)) *
radiation_coefficient*10^8*(
emissivity_gas_b*((feed_fluegas.t+273.15)/100)^4 -
A_v_b*(((temp_bt_surface)+273.15)/100)^4 )/(feed_fluegas.t -
(temp_bt_surface));
# FDBR 7.1.3 - 1/2

hdfg:  Q_gb_rad       =      alpha_rad_gb*A_bt* abs(feed_fluegas.t-
temp_bt_surface);

jhfgh: Q_fluegas_rad  =      Q_gb_rad;


########################
#         TESTS        #
########################


t263:  test(Q_gb_rad>=0.0)           warning "negative";
tk3:   test(Q_fluegas_rad>0.0)       warning "negative";
t7trt: test(s_eq_b>0)                warning "negative";


gfg5:   a1 = 0.1074    - 0.10705  * (feed_fluegas.t+273.15) / 1000 +
0.072727  * ((feed_fluegas.t+273.15) / 1000)^2;
jet:    a2 = 0.027237  + 0.10127  * (feed_fluegas.t+273.15) / 1000 -
0.043773  * ((feed_fluegas.t+273.15) / 1000)^2;
jfz6:   a3 = 0.058438  - 0.001208 * (feed_fluegas.t+273.15) / 1000 +
0.0006558 * ((feed_fluegas.t+273.15) / 1000)^2;
i75:    a4 = 0.019078  + 0.037609 * (feed_fluegas.t+273.15) / 1000 -
0.015424  * ((feed_fluegas.t+273.15) / 1000)^2;
hfgh6: a5 = 0.056993  - 0.025412 * (feed_fluegas.t+273.15) / 1000 +
0.0026167 * ((feed_fluegas.t+273.15) / 1000)^2;
jghj8: a6 = 0.0028014 + 0.038826 * (feed_fluegas.t+273.15) / 1000 -
0.020198  * ((feed_fluegas.t+273.15) / 1000)^2;
```

```
gh4:    k1 = (1/2.718281828) ^ (0.036 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
h5:     k2 = (1/2.718281828) ^ (0.3586 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
hh6:    k3 = (1/2.718281828) ^ (3.06 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
jf7:    k4 = (1/2.718281828) ^ (14.76 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
oi7:    k5 = (1/2.718281828) ^ (102.280 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);
pf8:    k6 = (1/2.718281828) ^ (770.60 * feed_fluegas.p *
feed_fluegas.Gas.yCO2 * s_eq_b);

jfg:   g_tot = (k1*a1)+(a2*k2)+(a3*k3)+(a4*k4)+(a5*k5)+(a6*k6);
```

UENER_INJECTION COOLER

```
########################
#    BASIC EQUATIONS    #
########################

feehdh:feed_water.massflow = drain_water.massflow-
feed_einspritzwasser.massflow;
hfdg:   feed_water.p =drain_water.p;
zdt:    drain_water.h*drain_water.massflow-
feed_einspritzwasser.massflow*feed_einspritzwasser.h
= (drain_water.massflow-feed_einspritzwasser.massflow)*feed_water_h;
```