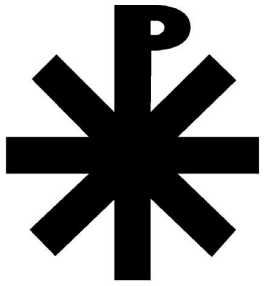


P E TECHNIKON

Research Dissertation

for

A Generic Simulation of Energy Consumption of Automobile Air Conditioning Systems

by

Dipl.-Ing.(FH) M. Konz

for the qualification

M Tech Eng

(Magister Technologiae: Engineering (Mechanical))

Promoters: Prof. Dr.-Ing. H. Holdack-Janssen
Mr. J. Maczek

Copyright statement

Contents

Acknowledgements

I would like to thank the staff of the Faculty of Electrical and Mechanical Engineering, in particular, Joe Maczek for his great support and Professor Jeffery, for making me feel welcome and assisting me in every way possible.

Furthermore, I would like to thank Professor Holdack-Janssen who was all this time a mentor for me with great ideas and critical discussions.

Author’s declaration

At no time during the registration for the Magister Technologiae Degree has the author been registered for any other Technikon or University degree.

Signed .....

Date .....

Glossary

Block-orientated Each component in the simulation is represented by a rectangle (a block) on the screen. It can be placed there with drag & drop technology. These components can have input and output pins and special parameters which can be entered in pop-up menues.

CFD Computational Fluid Dynamics. A mathematical method to simulate fluid flows in great detail. Very demanding in time and computer power.

C-MEX-Files A special kind of subsystem in MATLAB/SIMULINK® which uses a high speed C-Compiler to reduce computing time.

Conduction Energy transferred between two objects in contact.

Convection Energy carried from one object to another by a fluid.

Heat The transfer of energy between two objects due to temperature differences.

Heat Tranfer Heat may transfer across the boundaries of a system, either to or from the system. It occurs only when there is a temperature difference between the system and surroundings. Heat transfer changes the internal energy of the system. Heat is transferred by *conduction*, *convection* and *radiation*, which may occur separately or in combination.

Latent Heat The quantity of heat absorbed or released when a substance changes its physical phase at constant temperature.

MATLAB® A commercial computer program widely used for mathematical manipulation, simulation, programming and visualisation.

M-Files A special kind of subsystem in MATLAB/SIMULINK® that enters commands in the MATLAB® language.

NEDC	The New European Driving Cycle. This is a standardized driving cycle that is used to compare fuel and emmision consumptions of cars.
Radiation	Energy transferred as electromagnetic waves.
Sensible Heat	Heat that can be sensed, or detected, by a change in the temperature of the system.
SIMULINK®	A block-orientated graphical user interface for MATLAB® which allows the rapid modeling of physical systems.
Thermal Comfort	It is defined in the ISO 7730 standard as being ”That condition of mind which expresses satisfaction with the thermal environment”.

List of Tables

List of Figures

Nomenclature

α	heat transmission coeff.
β	window angle
δ	declination
δ_E	angle of incidence
$\dot{m}$	Mass flow rate
$\dot{Q}$	Heat
η	efficiency
γ	latitude
ω	hour angle
ψ	solar altitude
σ	window orientation angle
τ	time intervall
τ_S	solar azimuth
φ	relative humidity or driving direction
ρ	density
a	radiation absorbtion coeff.
c_p	heat capacity coefficient
d	radiation transm. coeff.
h	specific enthalpy
I	radiation
k	heat transfer coeff.
m	Mass
P	power
p	pressure
s	Specific entropy
s	thickness
T	Temperature
t	Time
w	velocity
x	absolute or specific humidity
x_w	deviation
Y	control output

Indices

$\perp$	perpendicular
$''$	saturated liquid
$'$	saturated vapour

a	outside	*L*	air
ab	leaving	*LL*	Leaking air
B	bottom	*M*	engine
Bt	compartment parts	*max*	maximum
c	condenser	*N*	normal
c	critical	*PP*	proportional
comp	compressor	*ref*	refrigerant
D	roof	*S*	solar
D	vapour	*s*	sides
e	evaporator	*SS*	side window
f	humid	*Str*	radiation
FS	front window	*sub*	sub-cooled
Gl	glass	*sup*	superheated
HS	rear window	*Tr*	transmission
i	inside	*w*	water
IR	integral	*z*	entering
is	isentropic	*zu*	entering

Chapter 1

Introduction

The air conditioning system in a car is, in addition to the heating system, installed to increase the comfort in the passenger compartment. Comfort is not the only reason for automotive air conditioning systems, as road safety also improves with the comfort of the driver, as a pleasant environment reduces driver fatigue.

The rising environmental problems and, hence, resulting stringent legislation are forcing the automobile industry to develop cars with ever decreasing fuel consumptions. The question of better fuel consumption and energy utilisation does not stop with the engine and aerodynamics, but is required of the air-conditioning system as well. Thus, incessantly innovative technologies are developed to decrease the energy required by the air-conditioning systems. The interaction of the refrigerant cycle components and the rapidly changing operating conditions of the car (speed, revolutions per minute, etc.) places extensive demands on the control system. In addition, the air-conditioning system is also designed for high ambient temperatures (cool down), but is mostly used in fairly moderate conditions. This operation allows for energy saving control strategies such as externally controlled compressors, blower motor control, etc.

The experimental comparison of different air-conditioning systems, components or control strategies is very time consuming and extensive, and the use of an air-conditioned wind tunnel is inevitable when experiments need to be done with reproducible ambient conditions.

This, combined with the high costs of installation and operation of a wind tunnel is a major problem. Furthermore, the effect of component or control strategy enhancements should be available as soon as possible in the early stages of design.

The above considerations have prompted the rapid development of new powerful simulation tools, but in most cases the simulation tools are focused on one specific component or problem only. A more holistic approach would be to combine the calculations of two or more programs. This implies the adaptation of the model to more programs which leads to a lack of transparency.

Obviously, the entire development work cannot be done entirely by simulation, especially in the later phases of the development where it would still be necessary to build prototypes to evaluate the done work experimentally. However, in the early stages of development, it would be advantagous to work without expensive prototypes.

1.1 Statement of the Problem

This Research is based on the Main Research Project ”Generic Simulation of Energy Consumption of Automobile Air Conditioning Systems”, which is divided into two main problems. Firstly, the simulation of the car interior and secondly the simulation of the refrigerant cycle.

These two problems cannot be examined completely separately because of their direct interaction. The passenger compartment rules the ambient conditions being the heating or cooling load with the cooling having to be generated by the refrigerant cycle. However, the cooling capacity of the refrigerant cycle is influenced by the ambient conditions as well. This greatly increases the complexity of the problem and only through critical detection of all influencing factors can lead to a realistic simulation.

1.2 Hypothesis

This research describes a simulation environment to calculate and analyze energy utilization in automotive air conditioning systems, and with this tool a model of the passenger compartment and the refrigerant cycle would be realized. The temperature distribution in the cabin, the heating rate and the control characteristics can be examined, which allows an energetic evaluation of the air conditioning system following potential enhancements that may be made. In addition, the intuitive, block-orientated environment of SIMULINK allows rapid variations of parameters which is moreover open-ended with script-files. Reproducibility is guaranteed with the use of standard driving cycles and performed reference measurements.

1.3 Subproblems

The following subproblems apply to this research:

1. *Description of thermal and mechanical properties of the refrigerant.* The thermal properties and the mechanical properties of the refrigerant have to be described mathematically and implemented into MATLAB/SIMULINK for further calculations.

2. *Design of the refrigerant cycle.* The basic design of the refrigerant cycle with the boundary conditions has to be developed. The basic thermodynamic rules should lead to a stationary simulation of the refrigerant cycle.

3. *Transient modelling of the components and the whole refrigerant cycle.* The transient modelling of the components is important because of the dynamic operation mode of the compressor and the changing ambient conditions during a regular driving cycle. The following components have to be described as being transient:

    - compressor
    - condenser
    - expansion device

- vaporator
- tubes.

The combination of the components concludes to the transient description of the whole refrigerant cycle.

4. *Implementation into* MATLAB/SIMULINK. This problem deals with the implementation of the theoretical model to an adequate MATLAB/SIMULINK environment which allows fast and easy refrigerant cycle modifications as well as boundary and ambient condition alterations. Furthermore, an easy visualisation method for the results has to be developed.

1.4 Delimitations

The following delimitations apply to this research:

- The simulation has to be block orientated, which leads to components that can only be described in a few subgroups in opposition to CFD-simulations.
- The simulation has to be implemented in MATLAB/SIMULINK because the interior model was developed in this environment as well.
- The simulation is limited to vapour compression refrigerant cycles. No other refrigerant cycles are relevant for the use in car air-conditioning systems.
- The influence of the circulating lubricant in the refrigerant cycle is neglected. This delimitation is usually made for refrigerant cycle simulations and should mainly result in differences in suction density and enthalpy[1].

1.5 Instationary simulation with MATLAB/SIMULINK

The above mentioned criterions for the simulation shows the importance of the choice of the *right* simulation environment. The following advantages of MATLAB/SIMULINK motivated the selection of the software over other software:

- Industry standard for general purpose simulations
- Highly sophisticated Computer Algebra System
- Open structure: new numeric algorithms can be implemented
- Graphical possibilities
- Intuitive use of the block orientated tool SIMULINK

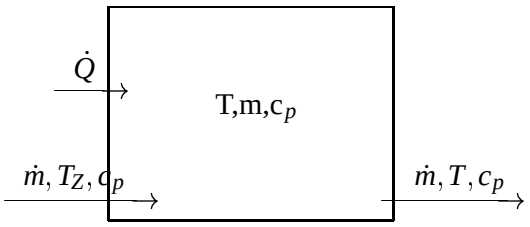
The slow built-in interpreter must be mentioned as one disadvantage, but this is offset by the open structure. It is possible to implement *dll's*[1] which implies the use of a very fast, modern Compiler language (e.g. C/C++).

The following example of a simple transient passenger compartment model demonstrates the differences and advantages between the algebraic solution, the MATLAB- and the SIMULINK-solution.

For the calculation of the transient temperature development the following simplified model may be used:

- homogeneous temperature distribution in the cabin,
- inlet mass flow rate = outlet mass flow rate,
- outlet air temperature = cabin air temperature,
- given external heat.

[1] dynamic linked libraries



$\dot{Q}$ = Heat load [W]
$\dot{m}$ = mass flow rate [kg/s]
m = cabin air mass [kg]
c_p = spec. heat capacity coefficient [kJ/(kg·K)]
T_Z = temperature of the air entering [K]
T = temperature in the cabin [K]

Figure 1.1: *Simplified passenger cabin model*

The energy analysis can be expressed as:

$$m \cdot c_p \cdot \frac{dT}{dt} = \dot{m} \cdot c_p \cdot T_Z - \dot{m} \cdot c_p \cdot T + \dot{Q} \tag{1.1}$$

Transforming the equation to

$$\frac{dT}{dt} = \frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot T_Z - \frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot T + \frac{\dot{Q}}{m \cdot c_p} \tag{1.2}$$

results in a standard differential equation of the type

$$\dot{T} = a \cdot T(t) + b \cdot u(t) \tag{1.3}$$

with the coefficients

$$a = -\frac{\dot{m} \cdot c_p}{m \cdot c_p} \qquad \text{and} \qquad b = \frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot T_Z + \frac{\dot{Q}}{m \cdot c_p}$$

The general solution of this equation is given by the finding the homogeneous and particular solution of the differential equation. The homogeneous differential equation

$$\dot{T} = a \cdot T(t), \qquad T(0) = T_0 \tag{1.4}$$

can be solved with

$$T(t) = k\, e^{\lambda t}$$

$$\dot{T} = k\lambda e^{\lambda t}$$

After substituting in Eq.1.4

$$k\lambda e^{\lambda t} = ak e^{\lambda t}$$

and

$$k\lambda = ak$$

so $\lambda = k$ and the solution for the homogeneous differential equation is

$$T(t) = k e^{at}. \tag{1.5}$$

For the solution of the particular differential equation the *variation of the constants* method is advisable. The constant k in Eq.1.5 is substituted with the time-function $k(t)$. The solution is then:

$$T(t) = k(t) e^{at}.$$

The derivation

$$\dot{T} = ak(t) e^{at}$$

substituted into Eq.1.3 results in

$$ak(t) e^{at} + \dot{k} e^{at} = ak(t) e^{at} + bu(t).$$

Then $\dot{k}$ can be calculated with

$$\dot{k} = bu(t) \cdot e^{-at}.$$

The integration over the interval $0 < \tau < t$ leads to

$$\int_0^t \dot{k}(\tau)d\tau = k(t) - k(0) = \int_0^t bu(\tau)\, e^{-at} d\tau$$

and hence

$$T(t) = k(0)\, e^{at} + \int_0^t bu(\tau)\, e^{-at} d\tau \tag{1.6}$$

Where $k(0)$ at present time is an unknown constant, which can be calculated considering the boundary conditions

$$k(0) = T(0)$$

The complete solution of the whole differential equation now becomes:

$$T(t) = T_0\, e^{at} + \int_0^t bu(\tau)\, e^{a(t-\tau)} d\tau \tag{1.7}$$

or with substituted coefficients

$$T(t) = T_0\, e^{-\frac{\dot{m}\cdot c_p}{m\cdot c_p}\cdot t} + \int_0^t \left[\left(\frac{\dot{m}\cdot c_p}{m\cdot c_p} \cdot T_Z + \frac{\dot{Q}}{m\cdot c_p} \right) e^{-\frac{\dot{m}\cdot c_p}{m\cdot c_p}\cdot(t-\tau)} \right] d\tau \tag{1.8}$$

The solution consists of two parts. The homogeneous and the particular solution of the differential equation. In a technical matter the homogeneous solution

$$T_{hom}(t) = T_0\, e^{at}$$

describes the transient temperature in the adiabatic cabin without external heat influences due to the initial excursion T_0. The particular solution

$$T_{part}(t) = \int_0^t bu(\tau)\, e^{a(t-\tau)} d\tau$$

describes the superposition of the "free" adiabatic cabin with the additional external stimulation caused by the external heat $\dot{Q}$. In the case of linear systems, the solution is given by the addition of the homogeneous and the particular solutions. In general the solution for $T(t)$ with $T(0) = T_0$ is:

$$T(t) = \begin{cases} T_0 \cdot e^{at} + \dfrac{b}{a}\left(e^{at} - 1\right) & \text{for } a \neq 0 \\ T_0 + bt & \text{for } a = 0 \end{cases} \tag{1.9}$$

The re-substituted coefficients lead to the solution for $a \neq 0$:

$$T(t) = T_0 \cdot e^{-\frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot t} + \frac{\frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot T_Z + \frac{\dot{Q}}{m \cdot c_p}}{-\frac{\dot{m} \cdot c_p}{m \cdot c_p}} \left(e^{-\frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot t} - 1 \right) \tag{1.10}$$

and for $a = 0$ (which represents a mass flow rate $\dot{m} = 0$):

$$T(t) = T_0 + \left(\frac{\dot{m} \cdot c_p}{m \cdot c_p} \cdot T_Z + \frac{\dot{Q}}{m \cdot c_p} \right) \cdot t \tag{1.11}$$

A transposition of this solution in MATLAB can be done as shown in Fig.1.2.

```
%--------------------------------inst.m----------------------------
%
%This Script shows a closed transient simulation of a simplified
%passanger compartment with a homogeneous temperature distribution,
%constant ventilation mass flow rates and a constant ventilation air
%temperature with constant heat flow rate.

m = 5;                    %Air mass in the cabin in kg
t0 =40;                   %Starting air cabin temperaturen in °C
tz = 6;                   %Ventilation air temperature in °C
mzu = 6/60;               %Ventilation mass flow rate in kg/s
c = 1;                    %spec. heat capacity in kJ/(kg*K)
Q = 1;                    %Heat flow rate in kW
tau = 0:1:300;            %Time in s
a = -mzu*c/(m*c);
b = mzu*c*tz/(m*c)+Q/(m*c);
T = t0*exp(a*tau)+b/a*(exp(a*tau)-1);  %Solution of the differential eq.
plot(tau,T)
xlabel('Time in s');
ylabel('Cabin Air Temperature in °C');
title('Transitory Cabin Air Temperature');
grid
```

Figure 1.2: MATLAB *m-File*

This model can be transfered to SIMULINK as shown in Fig.1.3. The green blocks are used for input data and the red ones are used for output data. The gray blocks are other mathematical operations. On comparing the calculated temperatures for both simulations in Fig.1.4 and Fig.1.5, it becomes obvious that the solutions are identical.

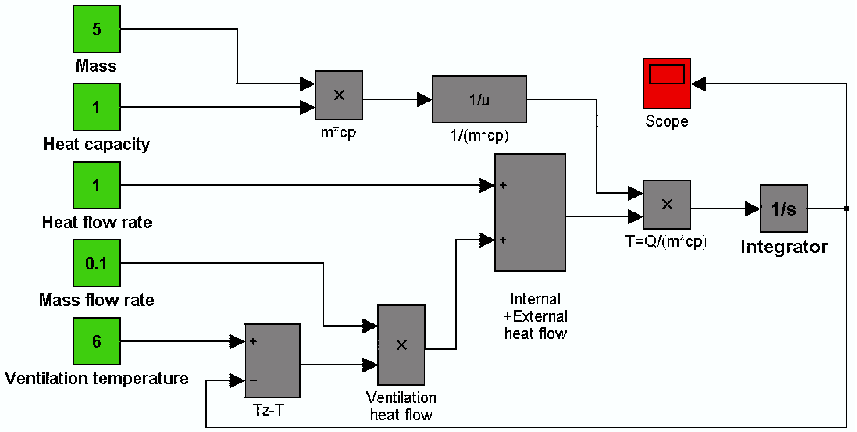


Figure 1.3: SIMULINK-*model*

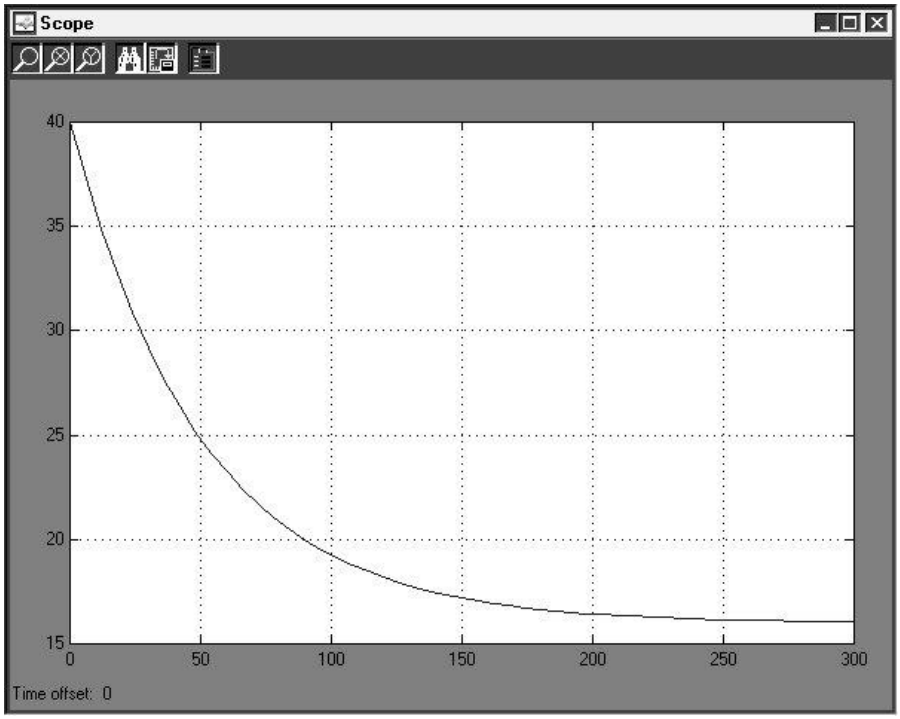


Figure 1.4: *Solution of the* SIMULINK*-calculation*

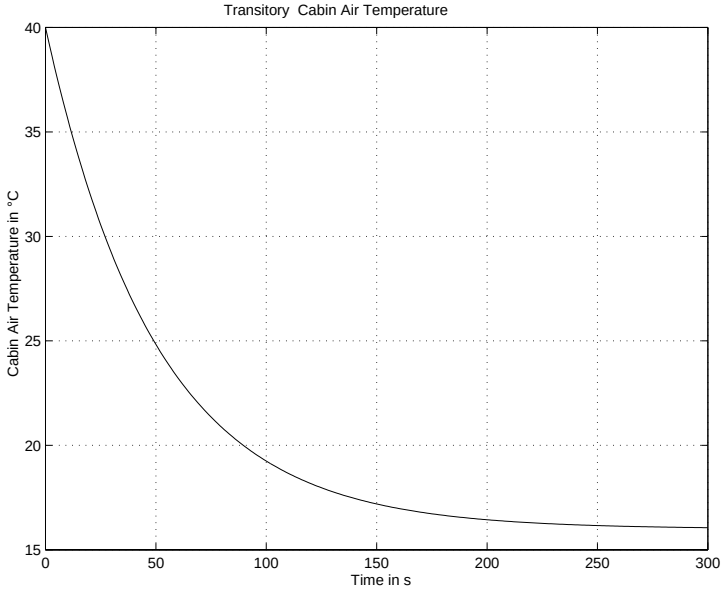


Figure 1.5: *Temperature graph calculated with* MATLAB

1.6 Conclusion

The model of a very simple dynamic system, as shown above, leads to differential equations, which have to be solved to make statements of the systems behavior.

With the use of the graphical user interface (GUI) SIMULINK to MATLAB it is possible to simulate systems by only knowing the differential equations, not their solution.

With the GUI shown in fig. 1.3 it is possible to calculate the system's behavior by only changing the parameters in the constant blocks and re-running the simulation (e.g. changing the mass flow rate). This allows for rapid and intuitive parameter analysis, which was a major reason for the use of simulation tools.

The following chapters analyze and discuss the actual systems involved, which are more complex. These systems are compositions of differential equations, algebraic expressions and tabular data. They are programmed with drag and drop methods on the desktop and are linked with Input/Output-blocks to complete the user interface.

Chapter 2

The Interior model

As stated in chapter 1, the simulation of the air conditioning system can be split into two parts, namely, the simulation of the car interior and the refrigerant cycle. This chapter describes the modeling of the transient behavior of the passenger compartment.

The interior model is used to calculate the air temperature and the cooling load or the heating load respectively under user specified climatic and driving conditions.

2.1 Function of the interior model

The Interior model should represent the climatic properties of the passenger compartment for cooling conditions as well as for heating conditions. Due to the many variables involved, several useful simplifications have to be made. These simplifications will be explained in the respective sections.

The definitive property of the passenger compartment model is the interior temperature at any time. This temperature can be used to control the *heating core* or the *fan*. Moreover, the interior temperature is also useful for other considerations such as cool down time or defrosting. The following influences were taken into account for the interior model:

- Solar radiation through the windows,
- Heat convection and transmission through the body,

- Air leakage,
- Passengers,
- Heat capacity of the interior and
- Ventilation.

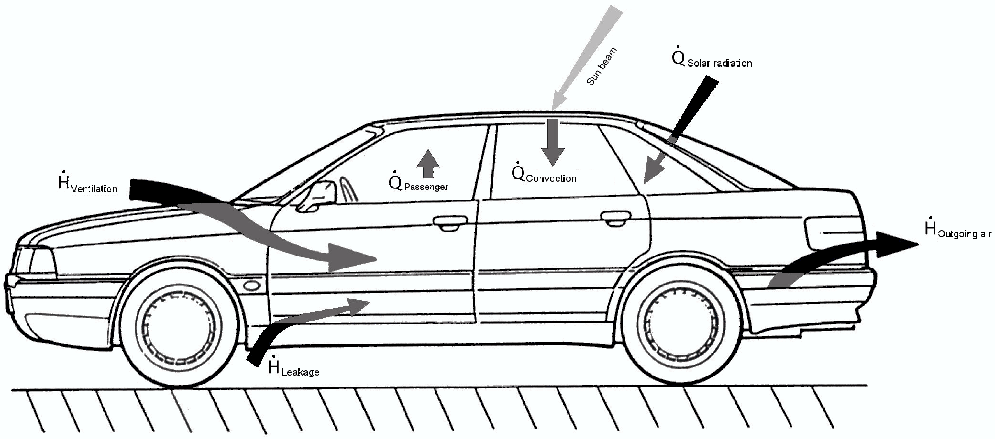


Figure 2.1: *Heat flow through a passenger cabin*

The simulation considers both thermodynamic equilibrium and transient conditions and, hence, cool down and heat up processes are taken into account.

Several blocks were used to define system constants with the interior model defined by different *subsystems*, where the seperate heat loads and temperatures are calculated. These subsystems are as follows:

- Interior
- Passengers
- Air leakage
- Solar radiation
- Transmission
- Air temperature

The subsystems are either block- or command orientated (as *S-Function*)[1]. The following sections deal with the function and structure of the subsystems.

2.1.1 Heat generated by the occupants

The purpose of this subsystem is the calculation of the cooling load required as a result of the thermal radiation of the passengers. The higher activity of the driver leads to a higher cooling load than for the other passengers. According to Shimizu [2], the driver and passangers provide 220W and 102W, respectively. The cooling load generated by the passengers is constant and not time dependent.

2.1.2 Cooling load from air leakage

The air leakage is defined as air entering and leaving the passenger cabin resulting from structural leakages. The cooling and heating load may be represented by the following energy balance:

$$\begin{aligned} \dot{Q}_{LL} &= \dot{H}_{LL,zu} - \dot{H}_{LL,ab} \\ \dot{Q}_{LL} &= \dot{m}_{LL} \cdot h_a - \dot{m}_{LL} \cdot h_i \\ \dot{Q}_{LL} &= \dot{m}_{LL} \cdot (h_a - h_i) \end{aligned} \tag{2.1}$$

$\dot{Q}_{LL}$	Cooling load caused by leakage
$\dot{H}_{LL,zu}$	Additional enthalpy by leaking entering air
$\dot{H}_{LL,ab}$	Leaving enthalpy by leaking air leaving the cabin
h_a	specific enthalpy of the outside air
h_i	specific enthalpy of the internal air
$\dot{m}_{LL}$	mass flow rate of the leakage air

[1] S-Functions are C-compiled dll's

For the calculation of the enthalpy, the specific humidity x has to be determined:

$$x_L = 0,622 \cdot \frac{\varphi_L \cdot p_S}{p_{Lf} - \varphi_L \cdot p_S} \tag{2.2}$$

where the saturated vapour pressure p_S is calculated using:

$$p_S = a \cdot \left(b + \frac{t_L}{100°\mathrm{C}} \right)^n \tag{2.3}$$

x_L absolute or specific humidity of the air (kg of water/kg of dry air)

φ_L relative humidity of the air (%)

p_{Lf} total or absolute pressure of the humid air (Pa)

p_S saturated vapour pressure of the moisture in the air (Pa)

t_L dry bulb temperature of the air (°C)

The constants used are as follows [3]:

$t_L < 0°\mathrm{C}:$	$t_L \geq 0°\mathrm{C}:$
$a = 4,689\,\mathrm{Pa}$	$a = 288,68\,\mathrm{Pa}$
$b = 1,486$	$b = 1,098$
$n = 12,36$	$n = 8,02$

The specific enthalpy is calculated as:

$$h_{1+x} = c_{pL} \cdot t_L + x_L \cdot (c_{pD} \cdot t_L + r) \tag{2.4}$$

h_{1+x}	specific enthalpy of the humid air in terms of dry air
c_{pL}	specific heat capacity of the dry air at constant pressure
c_{pD}	specific heat capacity of the vapour at constant pressure
r	specific enthalpy of water vapour (=2501kJ/kg at 0°C)

The air mass flow rate by leakage depends strongly on the driving speed. In addition, the modes of *fresh air* and *recirculated air* have to be computed. Results of investigations by the *Audi*® company, showed that the relationship of mass flow rate of the leaking air to the driving speed is linear and may be determined as follows [4]:

Fresh air mode:

$$\dot{m}_{LL,Freshairmode} \approx \left(\frac{w}{100}\right) \cdot 0,25 \frac{\text{kg}}{\text{min}} \tag{2.5}$$

Recirculation mode:

$$\dot{m}_{LL,Recirculation} \approx \left(\frac{w}{100}\right) \cdot 1 \frac{\text{kg}}{\text{min}} \tag{2.6}$$

Now, simplifying:

$$\dot{m}_{LL} \approx \left(\frac{w}{100}\right) \cdot (0,75 \cdot Recirculated\ fraction + 0,25) \frac{\text{kg}}{\text{min}} \tag{2.7}$$

The driving speed w is given as a constant. Alternatively, it is possible to use a *driving cycle*[1]. The data is either in tabular form in a file or a MATLAB variable and can be read by the simulation.

[1]The driving cycle used was the New European Driving Cycle (NEDC), but other driving cycles may be substituted.

2.1.3 Cooling load by solar radiation

Solar radiation is one of the most significant cooling loads in the model. The subsystem *Solar radiation* calculates the transmitted portion of the radiation. It doesn't result in an immediate change in the interior temperature but is absorbed by the interior at first, following which, the heated interior emits the heat by convection and radiation. The portion of the radiation that is *absorbed* in the windows and the total heat transmitted is also treated in the subsystem *Transmission*.

2.1.3.1 Angle of incidence

To analyse the solar radiation to an area, the perpendicular portion of the beam is required. For this reason it is necessary to determine the angle between the window and the sun beam for each window. The following coordinate system, wher the angles between the unit vectors of the windows and sun beam are specified, is essential in this analysis.

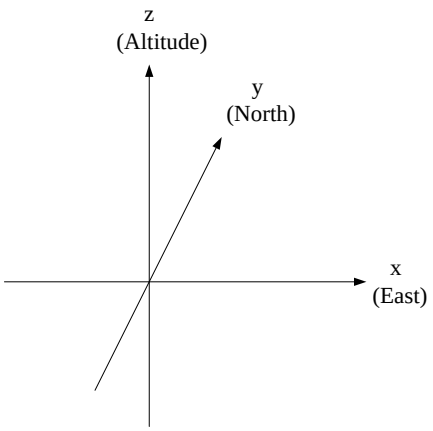


Figure 2.2: *3D-Coordinate system*

Unit vector of the solar beams. The position of the sun is given by the solar azimuth (angle between the north direction and the direction of the sun, τ_S) and the solar altitude (angle between the horizon and the sun, ψ), as illustrated in Fig. 2.4. To determine these angles the declination (angle between the connection sun – earth and the celestial equator plain, δ), illustrated in Fig. 2.3 is required.

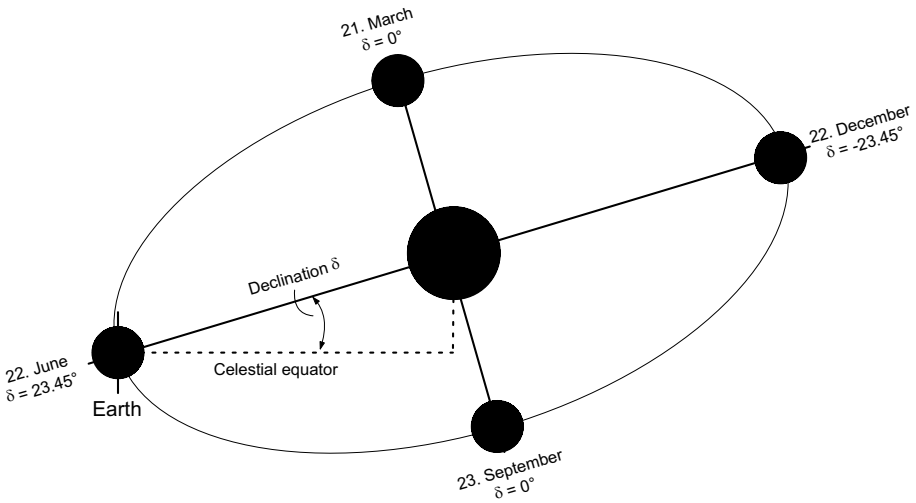


Figure 2.3: *Declination*

The eccentricity of the earth orbit is neglected and the declination can be expressed as follows:

$$\delta \approx -23,45° \cdot \cos \frac{360 \cdot (z+10)}{365} \tag{2.8}$$

Where *z* is the day of the year (1 through 365). To calculate the declination, the date must be provided in the corresponding constant blocks of the simulation. With the declination, the solar altitude can be determined with Duffies [5] simplified equation corresponding to Benford and Bock:

$$\psi = \arcsin\left(\sin\delta \cdot \sin\gamma - \cos\delta \cdot \cos\gamma \cdot \cos\omega\right) \tag{2.9}$$

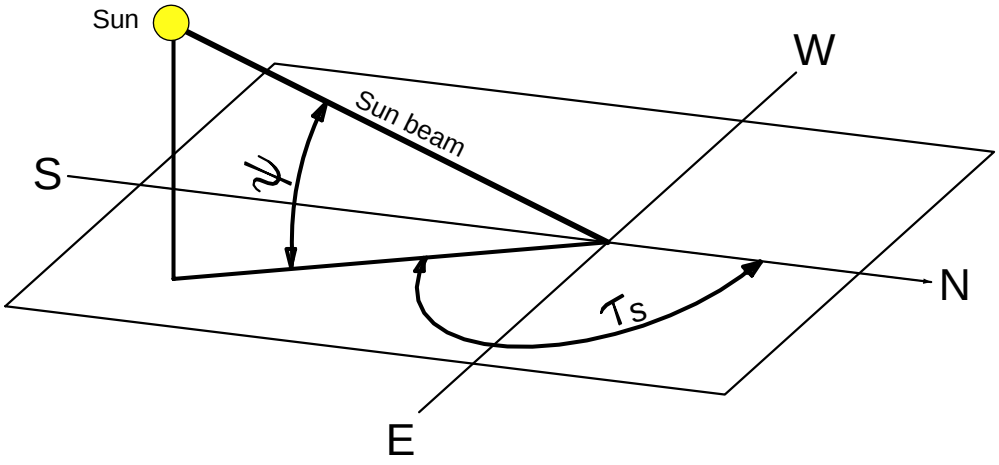


Figure 2.4: *Solar altitude and azimuth*

The latitude is provided in a constant block. The hour angle (the time of the day represented as an angle) is given by the equation

$$\omega = \frac{t_{Std}}{24\,\mathrm{h}} \cdot 360° \tag{2.10}$$

The solar azimuth, τ_S is determined using:

$$\sin(\tau_S) = \frac{\cos\delta \cdot \sin\omega}{\cos\psi} \tag{2.11}$$

ψ	solar altitude (°)
δ	declination (°)
γ	latitude (north: positive, south: negative) (°)
ω	hour angle (°)
t_{Std}	time of the day in hours (0 to 24h)

With the range of the solar azimuth being 0 through 360°and the arccos-function, with only values of -90° to +90° are possible, a distinction of cases is needed. For this reason a second azimuth τ_{S2} is introduced, which is slightly higher in the hour angle ω. With this method

increasing or decreasing function values can be determined.

I. quadrant:	$\tau_{S1} \geq 0$ AND $\tau_{S2} \geq \tau_{S1}$	$\Rightarrow \tau_S = \tau_{S1}$
II. quadrant:	$\tau_{S1} \geq 0$ AND $\tau_{S2} < \tau_{S1}$	$\Rightarrow \tau_S = 180° - \lvert\tau_{S1}\rvert$
III. quadrant:	$\tau_{S1} < 0$ AND $\tau_{S2} \leq \tau_{S1}$	$\Rightarrow \tau_S = 180° + \lvert\tau_{S1}\rvert$
IV. quadrant:	$\tau_{S1} < 0$ AND $\tau_{S2} > \tau_{S1}$	$\Rightarrow \tau_S = 360° - \lvert\tau_{S1}\rvert$

These cases are correct for the northern hemisphere only. For the southern hemisphere (with negative latitude), the following conversion rule applies:

$$\tau_{S,south.hemisphere} = 180° - \tau_{S,north.hemisphere} \tag{2.12}$$

Now, the unit vector of the solar beam simplifies to:

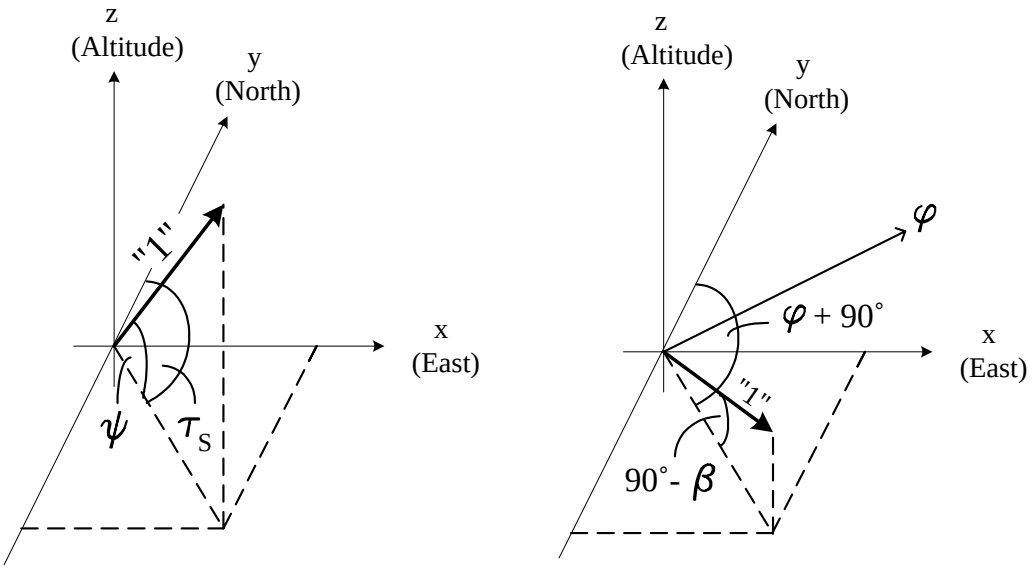
$$\vec{e}_S = \begin{pmatrix} \cos\psi \cdot \sin\tau_S \\ \cos\psi \cdot \cos\tau_S \\ \sin\psi \end{pmatrix} \tag{2.13}$$

Fig. 2.5 shows the position of the angles.

Unit vectors of the windows. For simplification, the surfaces of the windows are assumed as plain. The position of the surfaces in the space is described by the window angle β (angle between the horizon and window)and the driving direction φ, which is parallel to the side of the car. These postions are expressed in unit vectors for easier calculations.

To consider the orientation of the car window an angle σ is added to the driving direction φ:

Front window	$\sigma_F = 0°$
Right windows	$\sigma_R = 90°$
Rear window	$\sigma_H = 180°$
Left windows	$\sigma_L = 270°$



(a) Solar beam

(b) Right window

Figure 2.5: *Unit vectors*

The driving direction can either be specified in a constant block or is calculated in a *rotation*-block (subsystem). For this case the car rotates continuously during the time of simulation.

The normal-vectors of the windows are calculated component-wise:

$$\begin{aligned}\vec{x} &= \cos(90° - \beta) \cdot \sin(\varphi + \sigma) \\ &= \sin\beta \cdot \sin(\varphi + \sigma)\end{aligned} \tag{2.14}$$

$$\begin{aligned}\vec{y} &= \cos(90° - \beta) \cdot \cos(\varphi + \sigma) \\ &= \sin\beta \cdot \cos(\varphi + \sigma)\end{aligned} \tag{2.15}$$

$$\begin{aligned}\vec{z} &= \sin(90° - \beta) \\ &= \cos\beta\end{aligned} \tag{2.16}$$

The *unit vectors* of the windows are:

$$\vec{e}_N = \begin{pmatrix} \sin\beta \cdot \sin(\varphi + \sigma) \\ \sin\beta \cdot \cos(\varphi + \sigma) \\ \cos\beta \end{pmatrix} \tag{2.17}$$

The angles used to determine the normal vectors and their positions are shown in Fig. 2.5.

Determination of the angle of incidence. When the vector of the window and sun beam are known, their intermediate angle can be determined. To do this, it must be noted that the vector of the sun beam leads from the window to the sun and the window vector stands on the outer side of the window.

$$\delta_E = \arccos \frac{\vec{e}_N \cdot \vec{e}_S}{|\vec{e}_N| \cdot |\vec{e}_S|} \qquad \text{for} \qquad 0° \leq \delta_E \leq 180° \tag{2.18}$$

δ_E Angle between window and sun beam

For $\delta_E \geq 90°$ there is no direct radiation.

2.1.3.2 Value of the cooling load

If there is direct radiation the value of the perpendicular component can be calculated:

$$I_{S\perp} = I_{max} \cdot \cos \delta_E \tag{2.19}$$

$I_{S\perp}$ perpendicular component of the radiation

I_{max} total radiation (prescribed in the constant block)

If the perpendicular component of the direct radiation is lower than the diffuse radiation then only the diffuse radiation is taken into account. The entering heat load depends on the surface area of the window and the transmission coefficient d_S:

$$\dot{Q}_{Str} = I_{S\perp} \cdot A_{Gl} \cdot d_S \tag{2.20}$$

$\dot{Q}_{Str}$ Radiated heat flow rate through the windows (W)

A_{Gl} surface area of the window (m^2)

d_S radiation transmission coefficient of the glass

This calculation is performed for all sides of the car with transparent surfaces. The sum of all calculated heat loads is presented to the subsystem of the interior. It is assumed that the entire radiated energy remains in the interior and is absorbed entirely.

2.1.4 Transmission and convection

The calculation of the heat flow rate through the body which occurs through the roof, the bottom, the side surfaces and the windows needs to be considered separately, especially in the case of the roof and the windows where the general equation for heat transfer cannot be used. The direct solar irradiated surfaces absorb the radiation and heat up, transfering heat to

the interior and to the outside environment by convection. The heat flow rate moving into the passenger compartment by convection and transmission through the body can be expressed as:

$$\dot{Q}_{Tr} = \dot{Q}_B + \dot{Q}_S + \dot{Q}_M + \dot{Q}_D + \dot{Q}_{Gl} \tag{2.21}$$

$\dot{Q}_{Tr}$ Heat flow by convection and transmission

$\dot{Q}_B$ Heat flow through the bottom to the cabin

$\dot{Q}_S$ Heat flow through the sides

$\dot{Q}_M$ Heat flow through the engine compartment to the cabin

$\dot{Q}_D$ Heat flow through the roof

$\dot{Q}_{Gl}$ Heat flow through the windows

2.1.4.1 Bottom, sides and engine compartment, $\dot{Q}_B$, $\dot{Q}_S$ and $\dot{Q}_M$, respectively

Because the bottom is not exposed to direct solar radiation the heat flow $\dot{Q}_B$ may be calculated using the general equation for heat transfer:

$$\dot{Q}_B = k_B \cdot A_B \cdot (t_a - t_i) \tag{2.22}$$

k_B Overall heat transfer coefficient bottom $(\frac{W}{m^2 \cdot K})$

A_B Surface of the bottom (m^2)

t_a Outside air temperature (°C)

t_i Inside air temperature (°C)

The heat flow is dominated by the thermal conduction because the heat transmission resistance inside and outside is low in comparison to the conductibility of the multilayer of underseal, metal, insulation and carpet, hence the heat transfer coefficient is assumed constant at [6],[7]:

$$k_B = 2,3 \frac{\mathrm{W}}{\mathrm{m^2K}}$$

For non-transparent side surfaces it is assumed that they are rear-ventilated. In the determination of the heat flow through the side surfaces, $\dot{Q}_S$, it is important to know the temperature in the cavities. The calculation is analogous to Eq. 2.22 with the simplified temperature difference $t_a - t_i$. The overall heat transfer coefficient is:

$$k_S = 2,3 \frac{\mathrm{W}}{\mathrm{m^2K}}$$

For the heat flow from the engine compartment to the cabin, Shimizu [2] provides a suitable overall heat transfer coefficient as:

$$kA_M = 7 \frac{\mathrm{W}}{\mathrm{K}}$$

Hence,

$$\dot{Q}_M = kA_M \cdot (t_M - t_i) \tag{2.23}$$

t_M Temperature in the engine compartment

The temperature in the engine compartment is assumed constant at 100°C.

2.1.4.2 Roof and windows

The roof and the windows are exposed to direct solar radiation. The general heat transfer equation cannot be used because of the radiated heat. The heat flow rate is calculated by the convection at the inside surface of the component parts. Due to the relatively thin materials a homogeneous temperature distribution may be assumed. Then the heat flow rate into the cabin is:

$$\dot{Q}_{Bt} = \alpha_{i,Bt} \cdot A_{Bt} \cdot (t_{Bt} - t_i) \tag{2.24}$$

$\dot{Q}_{Bt}$ Heat flow rate through the component part (W)

$\alpha_{i,Bt}$ Heat transmission coefficient of the component part inside $(\frac{W}{m^2 \cdot K})$

A_{Bt} surface of the component part (m^2)

t_{Bt} component temperature (°C)

The temperature of the corresponding body part may be determined by applying an energy balance to determine the temperature difference for each time step:

$$\frac{\mathrm{d}t}{\mathrm{d}\tau} = \frac{I_{S\perp} \cdot A_{Bt} \cdot a_{Bt} + \alpha_{a,Bt} \cdot A_{Bt} \cdot (t_a - t_{Bt}) + \alpha_{i,Bt} \cdot A_{Bt} \cdot (t_i - t_{Bt})}{m_{Bt} \cdot c_{Bt}} \tag{2.25}$$

with $m = \rho \cdot V = \rho \cdot s_{Bt} \cdot A$:

$$\frac{\mathrm{d}t}{\mathrm{d}\tau} = \frac{I_{S\perp} \cdot a_{Bt} + \alpha_{a,Bt} \cdot (t_a - t_{Bt}) + \alpha_{i,Bt} \cdot (t_i - t_{Bt})}{\rho_{Bt} \cdot s_{Bt} \cdot c_{Bt}} \tag{2.26}$$

$I_{S\perp}$	Perpendicular portion of solar radiation
a_{Bt}	Radiation absorption coefficient of the component
$\alpha_{a,Bt}$	heat transmission coefficient of the component outside
m_{Bt}	Mass of the component
c_{Bt}	Specific heat capacity of the component
ρ_{Bt}	Density of the component
s_{Bt}	Thickness of the component

The temperature change for each simulation step is transfered to an integrator block whose output signal is the current temperature for the corresponding time. The initial condition for this integrator block is the initial condition of the passenger compartment air temperature. On establishing each component's temperature it is possible to determine the heat flow rates for the front, side and rear windows and for the roof.

The view on the radiation from each component in Eq. 2.25 and Eq. 2.26 which is emitted from the surfaces is cut out. Instead, an overall heat transmission coefficient is used which takes the energy flux caused by convection and the radiation into account [8].

Absorbed solar radiation. The intermediate angle between the sun beam and the surface window vector is determined as per section 2.1.3.1. The solar radiation can be expressed as:

$$I_{S\perp} = I_{max} \cdot \cos\delta_E \tag{2.27}$$

Outside heat transmission coefficients. The outside heat transmission coefficients are strongly dependend on the driving speed. The typical minimum heat transmission coefficient for a stationary vehicle is [2]:

$$\alpha_a = 25 \frac{\mathrm{W}}{\mathrm{m^2K}}$$

This is the minimum value that is to be used even if it is higher than the following calculated values for the single components [2]:

$$\alpha_{a,FS} = 3,79 \cdot w^{0,8} \quad \text{Front window} \tag{2.28}$$

$$\alpha_{a,SS} = 7,21 \cdot w^{0,8} \quad \text{Side windows} \tag{2.29}$$

$$\alpha_{a,HS} = 4,65 \cdot w^{0,8} \quad \text{Rear window} \tag{2.30}$$

$$\alpha_{a,D} = 4,41 \cdot w^{0,8} \quad \text{roof} \tag{2.31}$$

with $[w] = \frac{\text{m}}{\text{s}}$

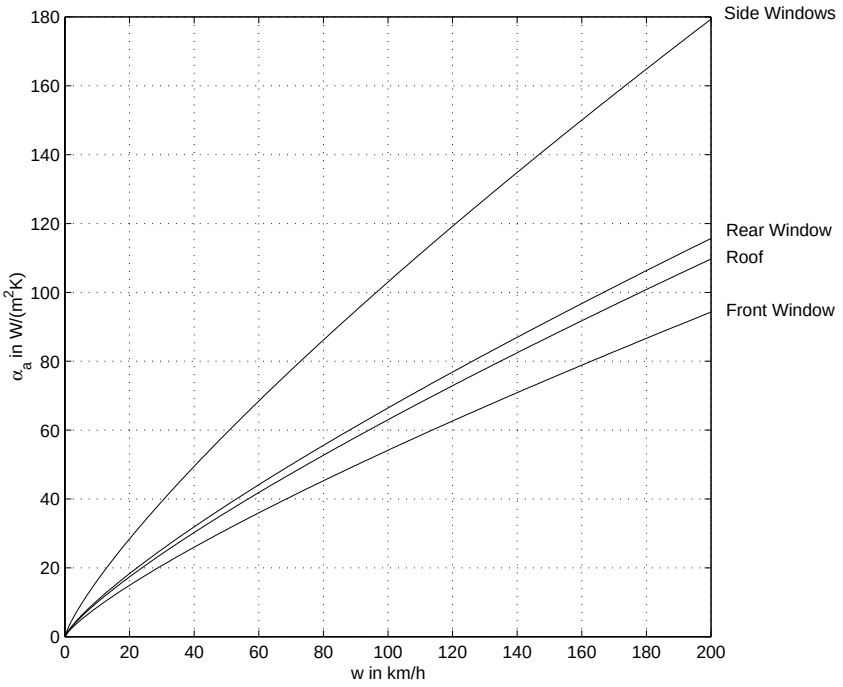


Figure 2.6: *Outside heat transmission coefficient corresponding to Eq. 2.28 through Eq. 2.31*

Internal heat transmission coefficient. The convective heat transfer from the windows to the cabin is dependent on the ventilated air volume flow rate, $\dot{V}_L$. If $V_L = 0$ the minimum

used heat coefficient is:

$$\alpha_i = 7 \frac{\mathrm{W}}{\mathrm{m^2K}}$$

This value is used even if it is less than the following calculated value for the single components [2]:

$$\alpha_{i,FS} = 0,584 \cdot \sqrt{\dot{V}_L} \qquad \text{Front window} \tag{2.32}$$

$$\alpha_{i,SS} = 0,495 \cdot \sqrt{\dot{V}_L} \qquad \text{Side windows} \tag{2.33}$$

$$\alpha_{i,HS} = 0,700 \cdot \sqrt{\dot{V}_L} \qquad \text{Rear window} \tag{2.34}$$

$$\text{with} \quad \left[\dot{V}_L\right] = \frac{\mathrm{m^3}}{\mathrm{h}}$$

The roof is assumed to have a stationary air layer between the metal sheet and the ceiling. The heat transmission coefficient from the metal sheet to the air layer is assumed as [2]:

$$\alpha_{i,D} = 2,3 \frac{\mathrm{W}}{\mathrm{m^2K}} \qquad \text{for the roof}$$

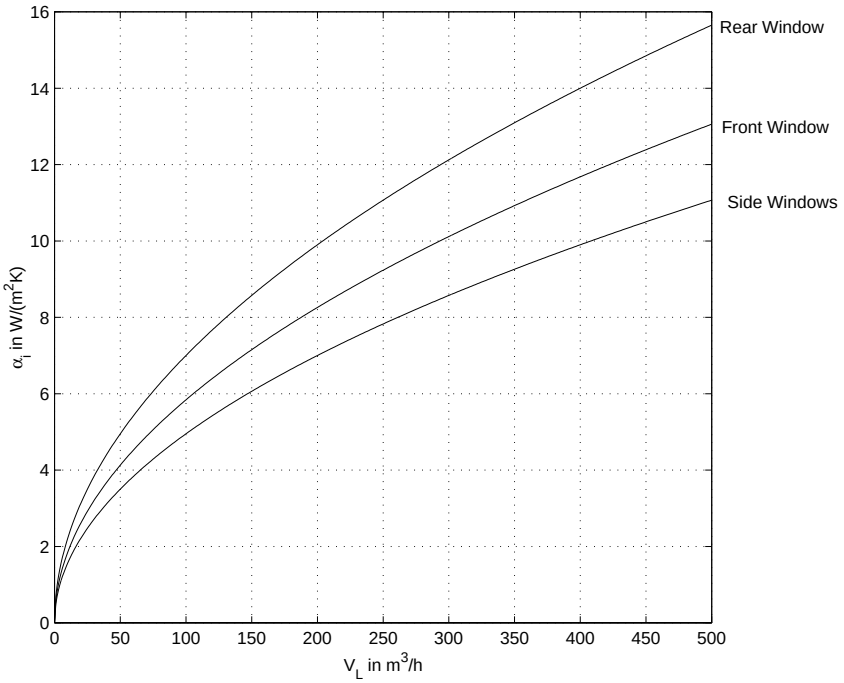


Figure 2.7: *Internal heat transmission coefficient corresponding to Eq. 2.32 through Eq. 2.34*

Density, thickness and specific heat capacity. The following assumptions are made for the values[1]:

Assumptions for the windows:

- Density of the glass $\rho_{Gl} \approx 2500 \frac{\text{kg}}{\text{m}^3}$
- Thickness of the glass $s_{Gl} \approx 4\,\text{mm}$
- Spec. heat capacity of the glass $c_{Gl} \approx 0,8 \frac{\text{kJ}}{\text{kg}\cdot\text{K}}$

Assumptions for the sheet metal body:

- Density steel $\rho_{St} \approx 7850 \frac{\text{kg}}{\text{m}^3}$
- Thickness of the sheet $s_{Bl} \approx 0,8\,\text{mm}$
- Spec. heat capacity of steel $c_{St} \approx 0,465 \frac{\text{kJ}}{\text{kg}\cdot\text{K}}$

[1] Based on measurements and Volkswagen® information

2.1.5 Heat capacity effect of the interior

The subsystem of the interior components is designed to determine the effect of the heat capacity. As soon as there is a temperature difference between the interior components and the cabin air temperature a convective heat transfer mode is created between them. The interior components are also exposed to solar radiation and, hence, the energy balance for the interior components is as follows:

$$\dot{Q}_{Eb} = \dot{Q}_{Str} + \alpha_{Eb} \cdot A_{Eb} \cdot (t_i - t_{Eb}) \tag{2.35}$$

$\dot{Q}_{Eb}$ Heat flow on the interior components (W)

α_{Eb} Heat transmission coefficient of the interior components $(\frac{W}{m^2 \cdot K})$

t_{Eb} Temperature of the interior components (K)

A_{Eb} Surface area of the interior components(m^2)

The air temperature in the passenger compartment is determined in the subsystem *air temperature*. The following temperature change relationship has to be determined to calculate the temperature of the interior components:

$$\frac{\mathrm{d}t}{\mathrm{d}\tau} = \frac{\dot{Q}_{Eb}}{m_{Eb} \cdot c_{Eb}} \tag{2.36}$$

m_{Eb} Mass of the interior components (kg)

c_{Eb} Specific heat capacity of the interior components $(\frac{J}{kg \cdot K})$

Commencing with a given start condition the temperature change is integrated over the time. The resulting actual temperature of the interior components, which is valid for the respective simulation step, is again the input value for the next calculation of the convective heat transfer between the interior components and the cabin air.

The values used for the parameters, as e.g. specific heat capacity, mass or surface area, are extremely difficult to estimate hence, it is nessecary to enter and alter them in the constant blocks manually.

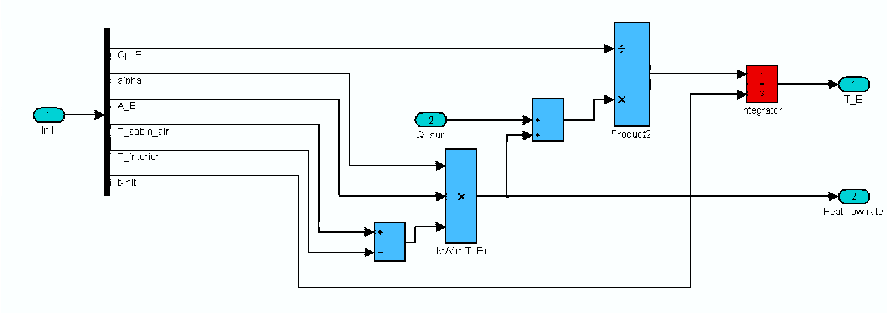


Figure 2.8: *Subsystem "Interior"*

2.1.6 Cabin air temperature

2.1.6.1 Interaction with other subsystems

Thus far the subsystems introduced, represent the cooling load required for the passenger compartment. Their outputs, which are the single heat flow rates, are the input values for the simulation "core" of the subsystem *Cabin air temperature*. Besides the loads already discussed, there is also the ventilation enthalpy flow to contend with which occurs in this subsystem. This enthalpy flow is required to control the air temperature in the cabin which means maintaining a comfortable value (stationary set point) or achieving a comfortable value (transient operation). The enthalpy flow is defined by the ventilation mass flow rate $\dot{m}_{zu}$ and the entering air temperature t_{zu}, which in turn is dependent on the refrigerant cycle and the engine coolant temperature. These are the first of two interfaces between the passenger compartment and the refrigerant cycle.

The subsystem, *Cabin air temperature* computes the air temperature which is dependent on several variables. Firstly, the cabin air temperature is important in the determination of the heat/cooling loads and, on the other hand, the air temperature is the physical dimension that has to be controlled itself. The air temperature, together with the temperature set point, inputs to the control unit. The control unit controls the single components of the air

conditioning/heating system to achieve the set point, so that the control unit is a coupling device for the *passanger compartment model* and *refrigerant cycle model* HVAC-box (Heating Ventilating Air Conditioning box), illustrated in Fig. 2.9.

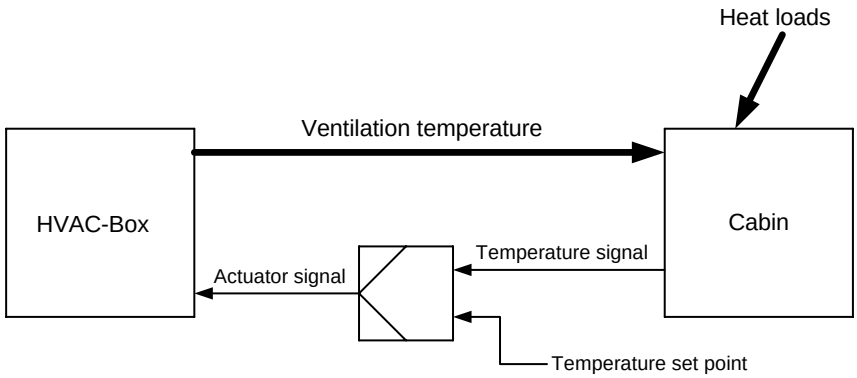


Figure 2.9: *Coupling of cabin and HVAC-box*

2.1.6.2 Function of the subsystem "cabin air temperature"

The function of this subsystem is quite similar to that of the *interior*. An energy balance is drawn up with the single loads not treated indiviually but combined into a single heat flow rate with,

$$c_{pL} \approx 1 \frac{\text{kJ}}{\text{kgK}} = \text{constant.}$$

Hence, is the derivative of the enthalpy with the change in time:

$$\frac{\mathrm{d}H}{\mathrm{d}\tau} = \dot{Q}_L + \dot{m}_{zu} \cdot c_{pL} \cdot t_{zu} - \dot{m}_{ab} \cdot c_{pL} \cdot t_{ab} \tag{2.37}$$

with $\dot{m}_{zu} = \dot{m}_{ab}$ and $t_{ab} \approx t_i$ it follows that:

$$\frac{\mathrm{d}H}{\mathrm{d}\tau} = \dot{Q}_L + \dot{m}_{zu} \cdot c_{pL} \cdot (t_{zu} - t_{ab})$$

The temperature change for each time step is then:

$$\frac{\mathrm{d}t}{\mathrm{d}\tau} = \frac{\dot{Q}_L + \dot{m}_{zu} \cdot c_{pL} \cdot (t_{zu} - t_i)}{m_L \cdot c_{pL}} \tag{2.38}$$

$\dot{Q}_L$ sum of the heat loads (W)

$\dot{m}_{zu}$ Ventilating air mass flow rate $(\frac{kg}{s})$

$\dot{m}_{ab}$ Leaving ventilation air mass flow rate $(\frac{kg}{s})$

m_L Mass of the air in the cabin (kg)

The temperature change becomes the input value for the integrator block at each time step. An initial start value, for e.g. the outside air temperature, needs to be specified to the integrator block to commence the calculation process.

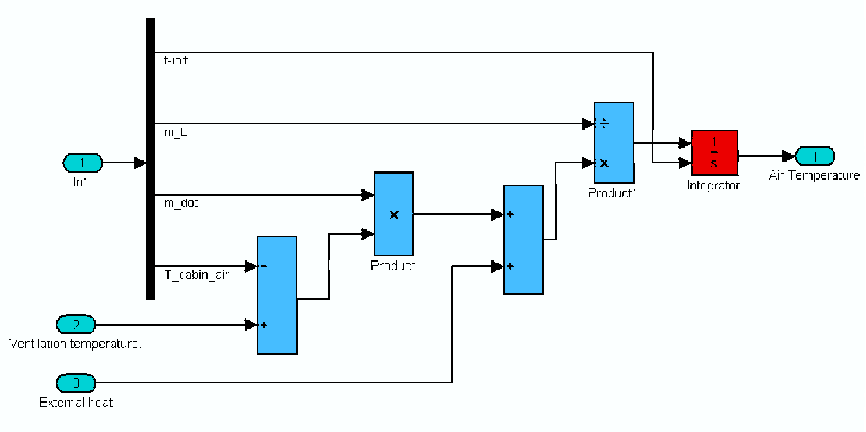


Figure 2.10: *Subsystem "Cabin air temperature"*

2.1.7 Controlling the cabin temperature

As mentioned earlier the control unit is a coupling device between the passenger compartment model and the refrigeration cycle model. It switches the refrigerant cycle on and off and controls the evaporator fan, the heater core valve and the bypass flap automatically. These were the control outputs in this case, which can be altered easily. The automatic control of the passenger compartment temperature is also used to control the refrigerant cycle and

the heating device through the deviation of the inside air temperature and set point of the compressor in the refrigerant cycle (x_w). Otherwise the air conditioning system works in a *re-heat*-mode. This is when the compressor is running a fixed temperature set point (usually $\approx$3°C) and, hence, the cold air has to be heated up again to fit to the passengers comfort.

The automatic control is implemented in the subsystem *control*. This control guarantees a flexible structure which allows for changes in the control strategy with ease. The Proportional-Integral (PI)- controller controls the heater core and the fan in sequence. This controller should not be an automatic temperature control, but an imitation of a user controlling a manual airconditioning (a/c) system. The structure of the control is shown in Fig. 2.11.

When heating (with a deviation $x_w < 0$), the refrigerant cycle remains obviously in the "off" status. With increasing deviation x_w the heating valve or the the bypass flap of the heater core closes. When the valve is completely opened or alternatively the bypass is completely closed, the fan speed increases. It can be adjusted from 25% to 100% of the maximum air flow rate.

For cooling conditions ($x_w > 0$) the refrigerant cycle switches to "on" status. The control strategy represents the clasical reheat mode, which results in an opened heating valve or closed bypass flap for small deviations with small air mass flow rates. With increasing deviations the heating valve closes or alternatively the bypass flap opens. Not until these devices are fully closed/opened would the mass flow rate be increased for further increasing deviations. See Fig. 2.12.

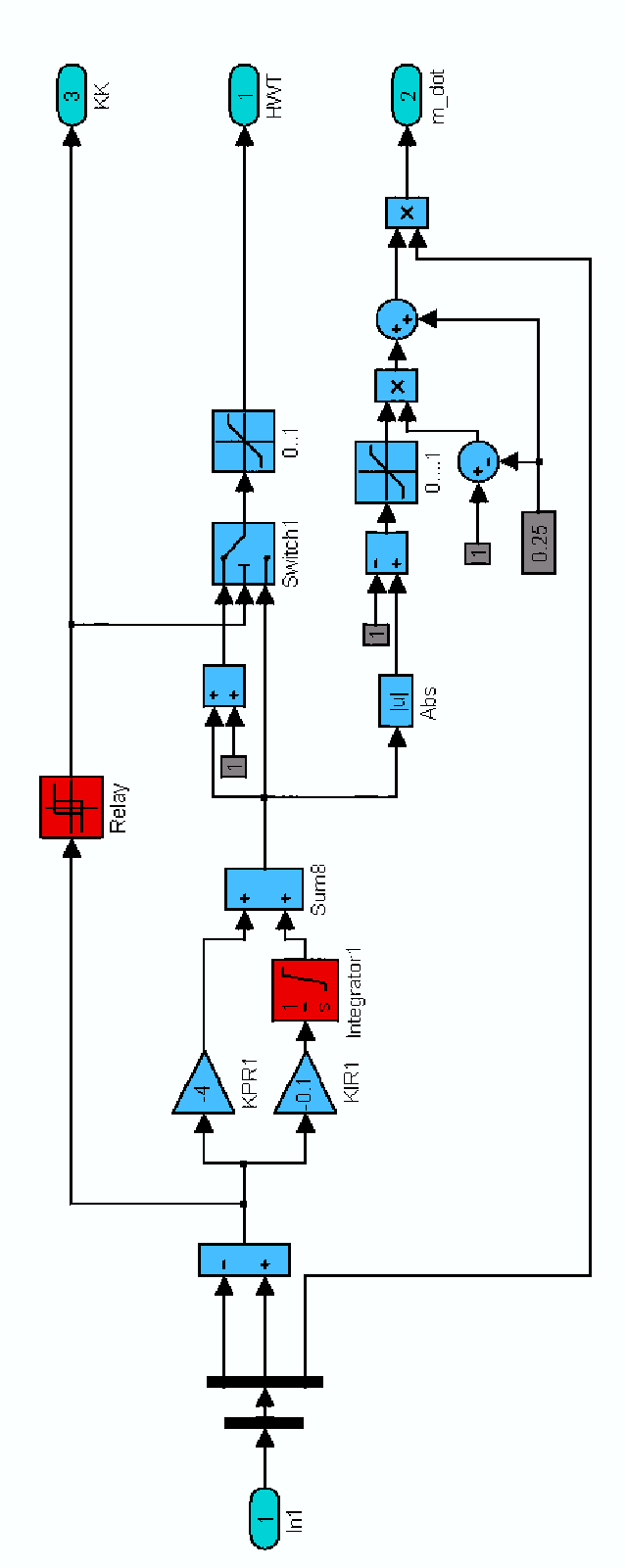


Figure 2.11: *Structure of the subsystem "Control"*

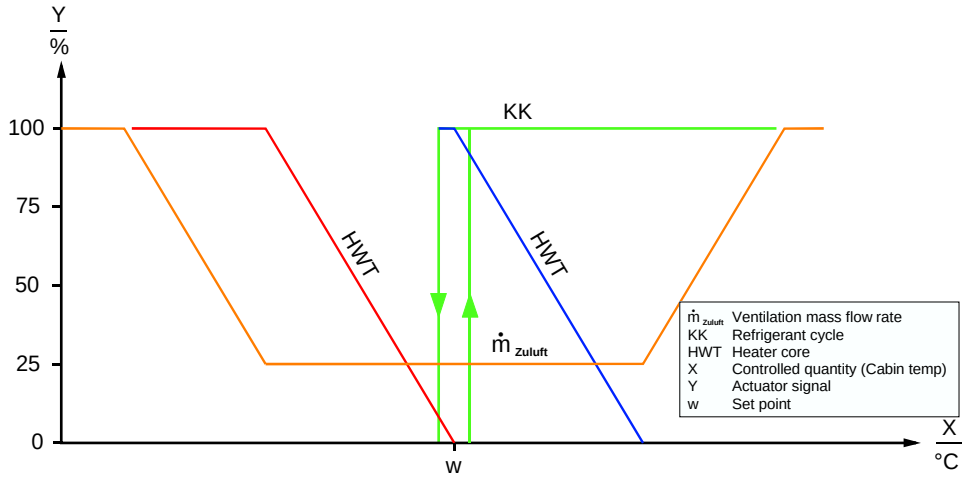


Figure 2.12: *Control sequence*

To realize the control sequence it is important to establish whether the refrigerant cycle is in the “on” or “off” mode. The “off” mode is the linear control function for the heater core at the set point zero and, in the case of cooling, the set point is 100%. On comparing the power required for heating and cooling, the power required for heating is much greater and would lead to a greater deviation as the heater core operates in the lower region of the control function. Fortunately, the *Integration* portion of the controler compensates for the deviation. The deviation for the PI controler is

$$Y = x_w \cdot (K_{\mathrm{PR}} + K_{\mathrm{IR}} \cdot \Delta\tau) \qquad \text{Heating} \tag{2.39}$$

$$Y = x_w \cdot (K_{\mathrm{PR}} + K_{\mathrm{IR}} \cdot \Delta\tau) + 1 \qquad \text{Cooling} \tag{2.40}$$

K_{PR} Proportional coefficient

K_{IR} Integration coefficient

x_w deviation $\hat{=} X - w$

Y control output $(0 \ldots 1)$

$\Delta\tau$ time interval

Useful values for these parameters were found by own experiments:

$$K_{\mathrm{PR}} = -4\,\frac{1}{\mathrm{K}}$$

and.

$$K_{\mathrm{IR}} = -0,1\,\frac{1}{\mathrm{K}\cdot\mathrm{s}}$$

2.2 “Cool down” experiment (validation of parameters)

As with any simulation model, adjusting and validating the chosen parameters is necessary to check and establish its accuracy. For this reason a “cool down” scenario in a real car was performed. The car was equiped with internal and external air temperature sensors to capture the data required for a graph illustrating the process. A well heated car (at low ambient temperature conditions) was the base for this experiment. The measured temperatures were recorded by a data aquisition device and analysed later. A time period was chosen that provided the condition for high temperature changes, but with no heat transmission from the engine or heater core and the passenger compartment. In the simulation the same start and boundary conditions were chosen. The overall heat transfer coefficients for the sides and the bottom, the values for the heat transmission coefficients, and the heat capacity of the interior components $m_{Eb} \cdot c_{Eb}$ were adjusted until the temperature graphs of the simulation closely matched the experimental results. The “cool down” graphs are illustrated in Fig. 2.13. Further experimental measurements for other specific scenarios such as summer scenarios, soak conditions and defrosting could also be undertaken.

2.3 Conclusion

The passenger compartment model facilitates the analysis of the temperature in the cabin during user defined scenarios, including different driving cycles, well. Hence, improvements

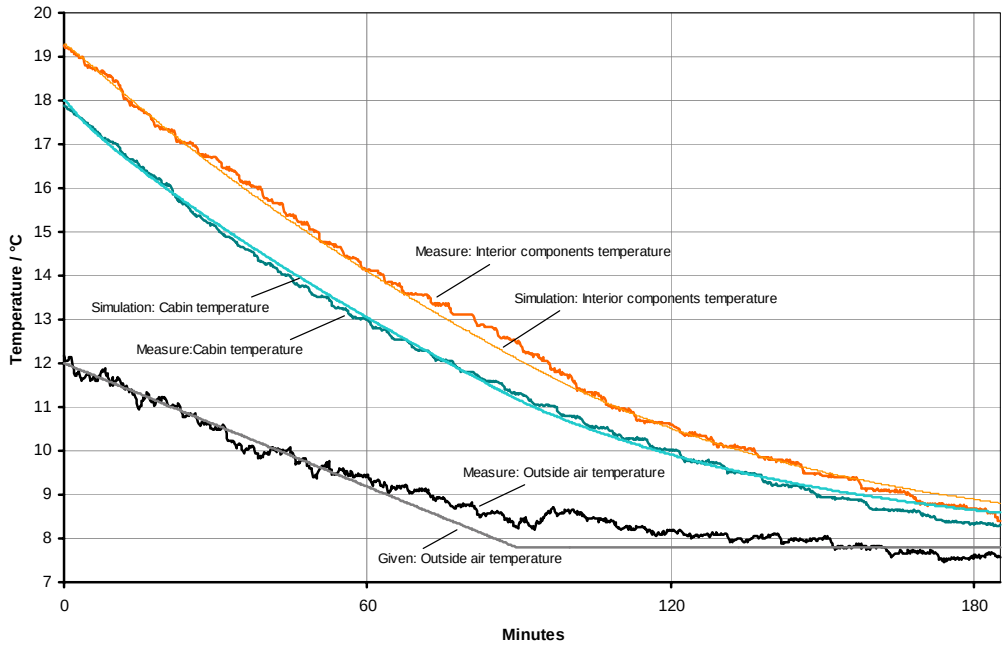


Figure 2.13: *Temperature graphs for the "cool down" experiments*

to the control system can be made and innovative technologies (e.g. smart recirculation air modes, etc.) may be investigated confidently and with ease during the design refinement stage of the design procedure.

This may all be done in a user friendly environment without writing any source-code. The environment is shown in Fig.2.14.

Unfortunately, several parameters such as lacquer and windows area have to be provided and are not always available. In most cases these variables have to be established through experimentation. A list of input variables for this simulation are stipulated in appendix A.

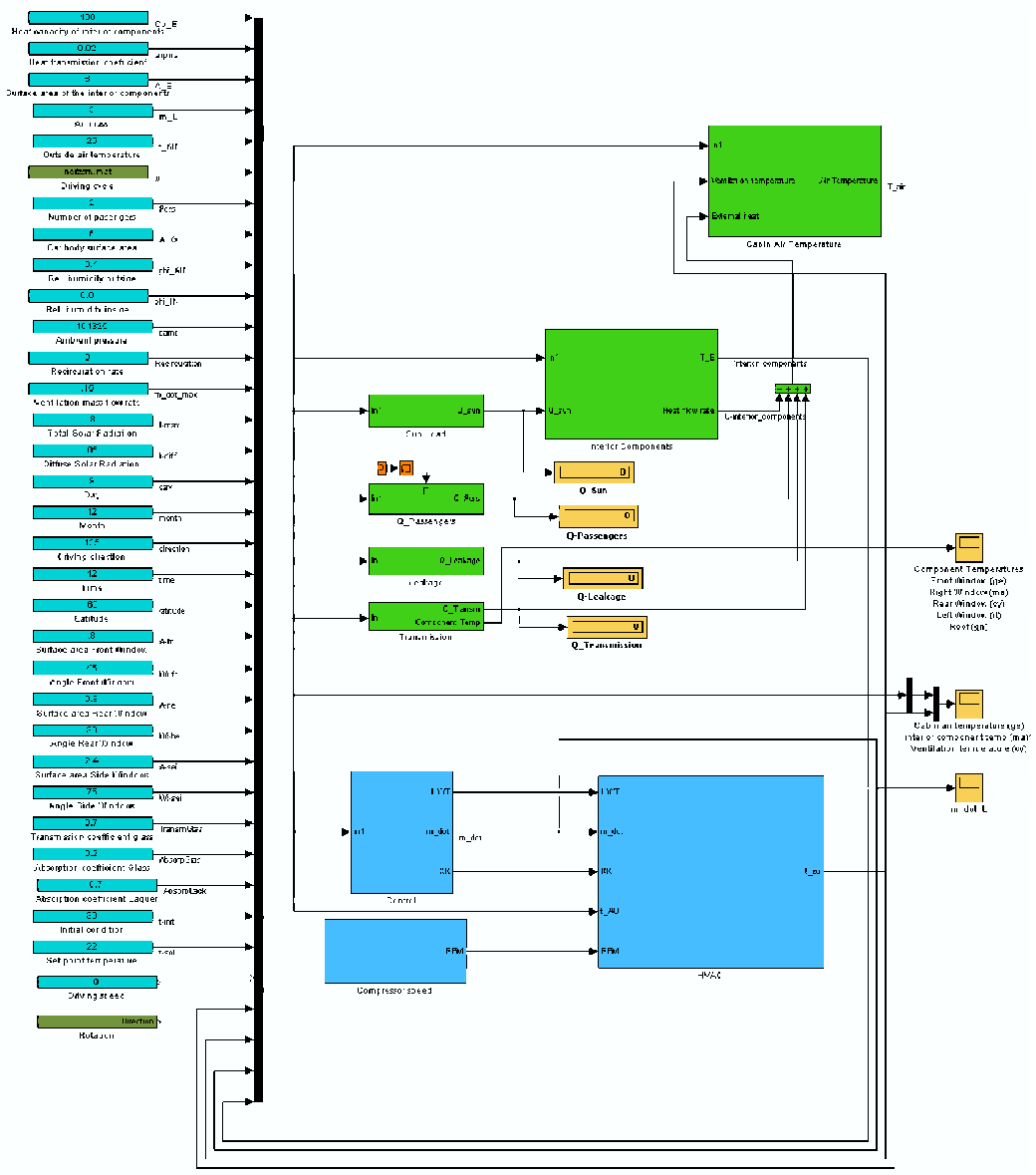


Figure 2.14: *Interior model*

Chapter 3

The Refrigerant cycle

This chapter describes how to simulate the refrigerant cycle to accomplish the cooling loads and comfort requirements in the passenger compartment.

Firstly, the calculation of the thermodynamic properties of the chosen refrigerant, R134a, is described, followed by the refrigerant cycle calculation itself, including the cooling performance and efficiency.

3.1 The refrigerant

The refrigerant selected, was R134a, because it is currently the most popular refrigerant used in car air conditioning. R134a is a transitional refrigerant which superseded R12 which has a high ozone layer depletion potential. Although R134a is ozone friendly it still has a green house effect potential factor and should be replaced itself, sometime in the future[1].

Some counries, such as Australia, already use alternative, but flammable refrigerants with a lower green house effect potential than R134a with no recorded accidents, but the refrigerant for the future may probably be, carbon dioxide (CO_2).

BAEHR, TILLNER-ROTH [9] published working equations for several alternative refrigerants. This set of equations is used in this research. These equations are in the same form for

[1]The *Kyoto*-protocol controls the emission of green house gases.

all hydrocarbons, like R134a, but simply have different constants for each refrigerant and, hence, provide the flexibility to accomodate other refrigerants in the simulation by simply inputing the appropriate constants (Appendix C.3).

Unfortunately, carbondioxide is different and the properties cannot be calculated with the same set of equations because the refrigerant cycle operates above the critical point and the properties are completely different to those of hydrocarbons, hence this research limits itself to hydrocarbons.

For R 134a, the following general equations and constants are valid: $\Theta := 1 - \frac{T}{T_C}$, $\tau := \frac{T_C}{T}$, $\delta := \frac{\rho}{\rho_C}$, $R = 81,488856\,J/(kg\,K)$, $T_C = 374,18\,K$, $\rho_C = 508\,kg/m^3$

3.1.1 Equations for the vapour

The valid limits for the scope are: $195\,K \leq T \leq 455\,K$ and $p \leq p_s(T) < 3,5\,MPa$.

- *pressure* $p(\tau,\delta)$

$$\begin{aligned}\frac{p(\tau,\delta)}{\rho RT} &= 1 + \delta\left(b_1\tau^{-1/2} + b_2\tau^{7/4} + b_3\tau^4 + b_4\tau^{12}\right) \\ &\quad + 2\delta^2\left(c_1\tau^{-1/4} + c_2\tau^{5/2}\right) + 3\delta^3\left(d_1\tau^{7/2} + d_2\tau^{12} + d_3\tau^{20}\right)\end{aligned} \tag{3.1}$$

- *Enthalpy* $h(\tau,\delta)$

$$\begin{aligned}\frac{h(\tau,\delta)}{RT} &= a_1^*\tau + m_1 + \frac{2}{3}m_2\tau^{-1/2} + \frac{4}{7}m_3\tau^{-3/4} \\ &\quad + \delta\left(\frac{1}{2}b_1\tau^{-1/2} + \frac{11}{4}b_2\tau^{7/4} + 5b_3\tau^4 + 13b_4\tau^{12}\right) \\ &\quad + \delta^2\left(\frac{7}{4}c_1\tau^{-1/4} + \frac{9}{2}c_2\tau^{5/2}\right) \\ &\quad + \delta^3\left(\frac{13}{2}d_1\tau^{7/2} + 15d_2\tau^{12} + 23d_3\tau^{20}\right)\end{aligned} \tag{3.2}$$

- *Entropy* $s(\tau,\delta)$

$$\begin{aligned}\frac{s(\tau,\delta)}{R} &= a_0^* + (m_1-1)(1-\ln\tau) - \ln\delta + 2m_2\tau^{-1/2} + \frac{4}{3}m_3\tau^{-3/4} \\ &+ \delta\left(-\frac{3}{2}b_1\tau^{-1/2} + \frac{3}{4}b_2\tau^{7/4} + 3b_3\tau^4 + 11b_4\tau^{12}\right) \\ &+ \delta^2\left(-\frac{5}{4}c_1\tau^{-1/4} + \frac{3}{2}c_2\tau^{5/2}\right) \\ &+ \delta^3\left(\frac{5}{2}d_1\tau^{7/2} + 11d_2\tau^{12} + 19d_3\tau^{20}\right)\end{aligned} \tag{3.3}$$

- *Spec. heat capacity* $c_p^0(\tau)$

$$\frac{c_p^0(\tau)}{R} = m_1 + m_2\tau^{-1/2} + m_3\tau^{-3/4} \tag{3.4}$$

with the coefficients:

b_1=0,22012860	b_2=-1,26740200	b_3=-0,30849910	b_4=-3,546135 $\cdot 10^{-4}$
c_1=0,01387313	c_2=0,33531550	d_1=-0,06356633	d_2=0,08120353
d_3=-0,03699965	a_0^*=1,019535	a_1^*=9,047135	
m_1=-0,629789	m_2=7,292937	m_3=5,154411	

3.1.2 Equations for saturated vapour

- *vapour pressure* $p_s(T)$

  valid for $169,85K \leq T \leq 374,18K$ with $p_0 = 4,056318MPa$

$$\ln\frac{p_s(\Theta)}{p_0} = \frac{1}{1-\Theta}\left[a_1\Theta + a_2\Theta^{3/2} + a_3\Theta^2 + a_4\Theta^4\right], \tag{3.5}$$

  with the coefficients:

  $a_1 = -7,7057291, a_2 = 2,4186313, a_3 = -2,1848312, a_4 = -3,4530733.$

- *density of saturated vapour pressure* $\rho'(\Theta)$ (liquid)

valid for $185\,K \leq T \leq 373\,K$

$$\frac{\rho'(\Theta)}{kg/m^3} = q_0 + q_1\Theta^{1/3} + q_2\Theta^{2/3} + q_3\Theta^{13/4}, \tag{3.6}$$

with the coefficients

$q_0 = 518,236, q_1 = 885,538, q_2 = 482,517, q_3 = 192,157.$

- *The saturated liquid density* $\rho''(\Theta)$

  can be determined with an iterative procedure where $p = p(\tau,\delta)$ *(Eq.3.1)*.

- *Saturated vapour enthalpy* $h'(\Theta)$

  valid for $185\,K \leq T \leq 373\,K$ with $h_0 = 384,07 kJ/kg$

$$\ln\frac{h'(\Theta)}{h_0} = n_1\Theta^{1/2} + n_2\Theta^{5/4} + n_3\Theta^3 + n_4\Theta^4 + n_5\Theta^{10}, \tag{3.7}$$

with the coefficients

$n_1 = -0,534728, n_2 = -1,757777, n_3 = -0,928326, n_4 = -2,666564,$

$n_5 = -30,82570.$

3.1.2.1 Numerical methods

The instant determination is only possible for the following properties, with the given independent property combinations:

- $p = p(T,\rho)$
- $h = h(T,\rho)$
- $s = s(T,\rho)$
- $ps_s = p_s(T)$
- $\rho' = \rho'(T)$
- $h' = h'(T)$

The solution for any other combination of dependent and independent properties can be achieved using a numerical method. In a case with limited scope, the dichotomizing or binary search is useful. With this method the equation is equated to zero and the starting points for the search have to be assumed with one point being a zero point. The determination of these starting points can be difficult in that there may be more than one zero point in the range of the interval of the starting points, by definition of the dichotomizing algorithm. Hence, these starting points were found by examining the results of the calculations within the stated limits for the equations 3.1 to 3.7. Using these two starting points the marked interval can be reduced so long as the remaining error is greater than the error limit.

The implementation into MATLAB occurs with *C-MEX*-files, which are compiled and hence translated into dll's. Dll's are superior in speed compared to the MATLAB-Interpreter.

property	command	independent parameters
$p = p(T, \rho)$	p_T_rho	T, ρ
$h = h(T, \rho)$	h_T_rho	T, ρ
$s = s(T, \rho)$	s_T_rho	T, ρ
$p_s = p_s(T)$	ps_T	T
$h' = h'(T)$	hstrich_T	T
iterative calculation		
$\rho = \rho(T, p)$	rho_T_p	T, p
$\rho = \rho(T, s)$	rho_T_s	T, s
$h'' = h''(T)$	hzweistrich_T	T
$T = T(h, p)$	T_h_p	h, p
$T = T(p, s)$	T_p_s	p, s
$T = T(p_s)$	T_ps	p_s

Table 3.1: *Properties and there command in* MATLAB

Table 3.1 shows the implemented properties and the required interconnection parameters, in SI units.

The following example illustrates the ease of use:

What's the temperature of superheated vapour for refrigerant R134a at a pressure of 20 bar and an enthalpy of 430 kJ/kg?

Input:

```
T_h_p(430e3,20e5)
```

results in:

```
ans =
341.7815
```

Hence, the temperature is *341,7815K - 273,15K = 68,63°C*

3.1.3 log**p,h-diagram**

The variations of properties during phase-change processes are best studied and understood with the help of property diagrams. A frequently used diagram in the analysis of vapour-compression refrigeration cycles is the logp,h-diagram, shown in fig. 3.1.

This property diagram is also extremely useful in analysing the thermodynamic processes, heat transferred and efficiencies, for the cycle. With this diagram, the heat transferred is simply proportional to the enthalphy change across the corresponding process. Alternatively, a T,s-diagram may be used for this analysis, but the heat transferred is more difficult to determine, in that the heat transferred is equal to the area under the corresponding process.

$$Q_{12} = \int_2^1 T\,ds \tag{3.8}$$

For this reason, the log*p*,h-diagram was selected for this research.

With the described property equations it is possible to create a logp,h-diagram and the generation of the saturation lines and iso-lines with MATLAB script-files.

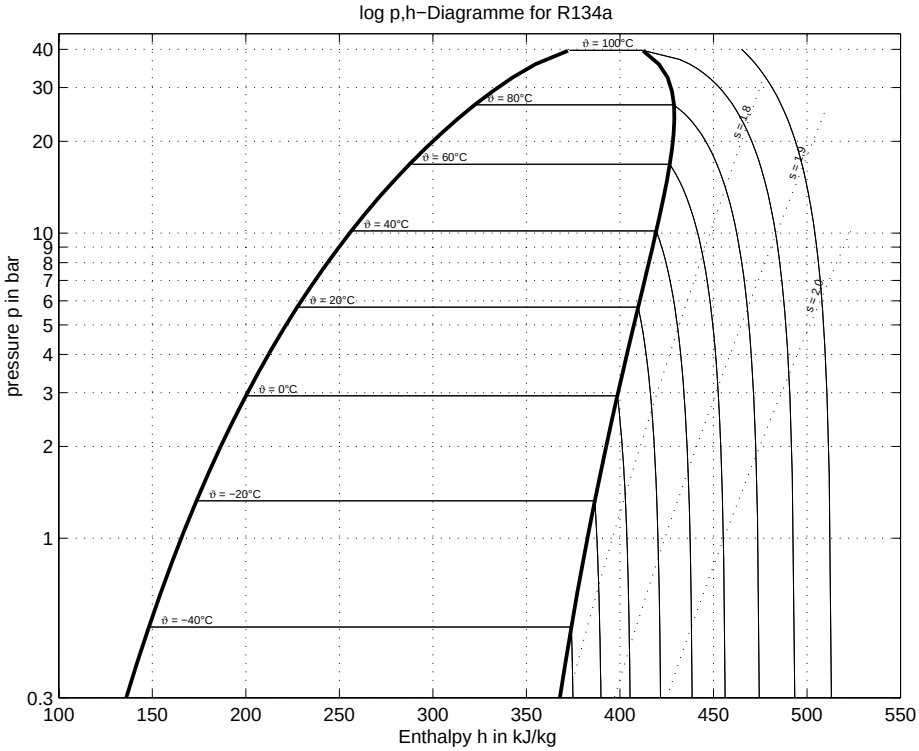


Figure 3.1: log*p,h-diagram for R134a*

Fig. 3.1 shows the diagram that was created with the script-file `lgph_Diagramm.m` (see Appendix C.4).

Firstly, the liquid saturation line is determined for the log*p*,h-diagram. To do this the liquid saturated enthalpy $h' = h'(\Theta)$ *(using Eq.3.7)* is calculated for a given temperature range *T*. From this, the saturated vapour line and the saturated vapour pressure is determined first. With this pressure the boiling density ρ'' is computed iteratively and, hence, using these parameters the enthalpy h'' can be determined.

The isothermals are calculated for each temperature by providing the pressure for sufficient supporting points (e. g. 0.3 bar–ps(T)). At the pressure and supporting points, the density is computed iteratively. These properties of pressure and density leads to the enthalpy. The calculation of the other isolines is analogous.

For the dual-phase region, the pressure was assumed to be constant and lying on the isothermals, such that $p'=p''$ with T=const.

3.2 The refrigerant cycle

Figure 3.2 shows the structure of the refrigerant cycle with its main parts: compressor, condenser, expansion valve and the evaporator. The following sections will describe the behavior of the refrigerant cycle and the implementation of the single parts in MATLAB/SIMULINK.

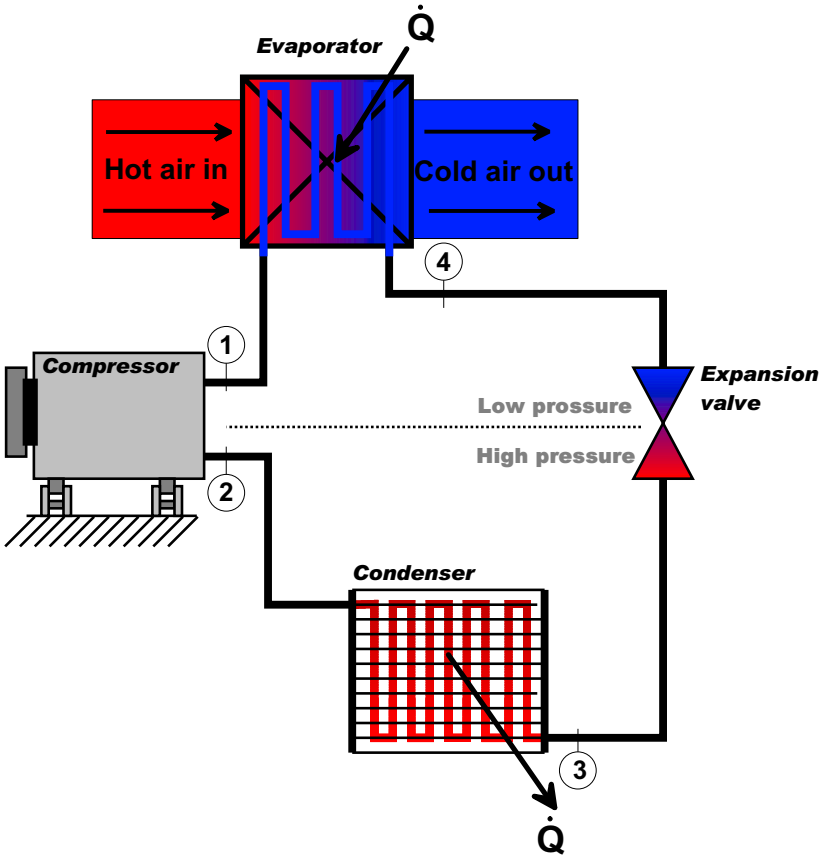


Figure 3.2: *Refrigerant cycle*

On the log$p$,h-diagram, three of the four processes appear as straight lines (see fig. 3.3 for the ideal cycle), and the heat transfer in the condenser and evaporator is proportional to the lengths of the corresponding process lines.

3.2.1 The stationary ideal refrigerant cycle

The stationary refrigerant cycle does not take any variable property changes with time into account. Furthermore, the system is estimated as being in thermodynamic equilibrium, such that:

$d\dot{Q}_i/dt = 0$, and hence $d\dot{m}_{R134a}/dt = 0$.

The following assumptions are made for the ideal refrigerant cycle:

- Isentropic compression ($s = constant$)
- Isobaric condensation ($p = constant$)
- Isenthalpic expansion ($h = constant$)
- Isobaric evaporation ($p = constant$)
- The compressor receives the refrigerant as saturated vapour (*liquid fraction = 0 or dryness fraction =1*)
- The expansion or the refrigerant begins as fully saturated liquid (*liquid fraction=1 or dryness fraction =0*)

3.2.1.1 Calculation

To calculate the ideal refrigerant cycle the input variables for the *compressor power*, and the *evaporator* and *condenser pressures* are inputted to the simulation. Using this data and the thermodynamic properties it is possible to calculate the set points of the refrigerant cycle. The condenser heating rate, evaporator cooling rate, refrigerant mass flow rate and the cop (*coefficient of performance*) can be determined.

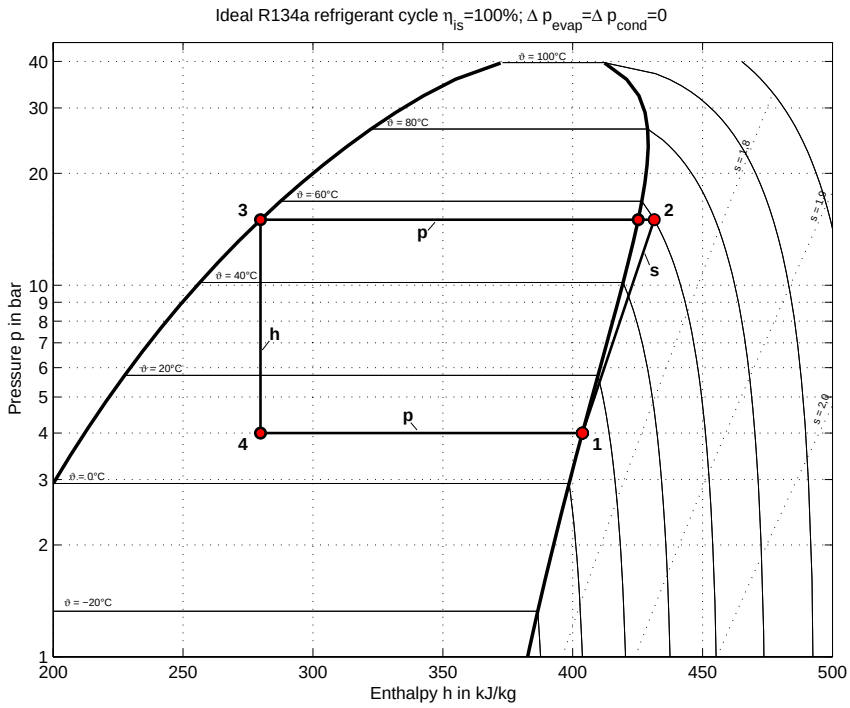


Figure 3.3: *Ideal refrigerant cycle*

Procedure:

- **Point 1** (suction state or inlet of compressor)

  $T_1 = T(p_s)$ with $p_s = p_0$

  $s_1 = s(T, \rho)$ with $T = T_1, \rho = \rho(T_1, p_0)$

  $h_1 = h''(T_1)$

- **Point 2** (Entry to the condenser)

  $T_2 = T(p_c, s_1)$ because $\eta_{is} = 1$ is $s_2 = s_1$

  $h_2 = h(T_2, \rho_2)$ with $\rho_2 = \rho(T_2, p_c)$

- **Point 3** (Entry to the expansion device without sub-cooling)

  $h_3 = h'(T_3)$ with $T_3 = T(p_s)$ and $p_s = p_c$ because $T = const.$ in the two-phase region.

- **Point 4** (Entry to the evaporator)

  $h_4 = h_3$

  $T_4 = T_1$

- **refrigerant mass flow rate**

  $\dot{m}_{ref} = \frac{P_{comp}}{h_2 - h_1}$

- **Condenser power**

  $Q_c = \dot{m}_{ref} \cdot (h_3 - h_2)$

- **Evaporator power**

  $Q_e = \dot{m}_{ref} \cdot (h_1 - h_3)$

- **cop**

  $cop = \frac{Q_e}{P_{comp}}$

3.2.2 The real stationary refrigerant cycle

The real refrigerant cycle differs from the ideal in the following ways:

- The isentropic efficiency of the compressor $\eta_{is} < 1$.
- The pressure losses in the heat exchangers $\Delta p_c > 0$ and $\Delta p_e > 0$.
- The temperature of the refrigerant entering the compressor (suction temperature) $\neq$ temperature of the saturated refrigerant at the pressure of the saturated vapour pressure ($T_1 \neq T_{1''}$). The refrigerant in the real refrigerant cycle is super-heated on entering the compressor.
- The temperature of the refrigerant at entry to the expansion device $<$ temperature of the saturated liquid at the same pressure. The refrigerant is normally sub-cooled after condensation.

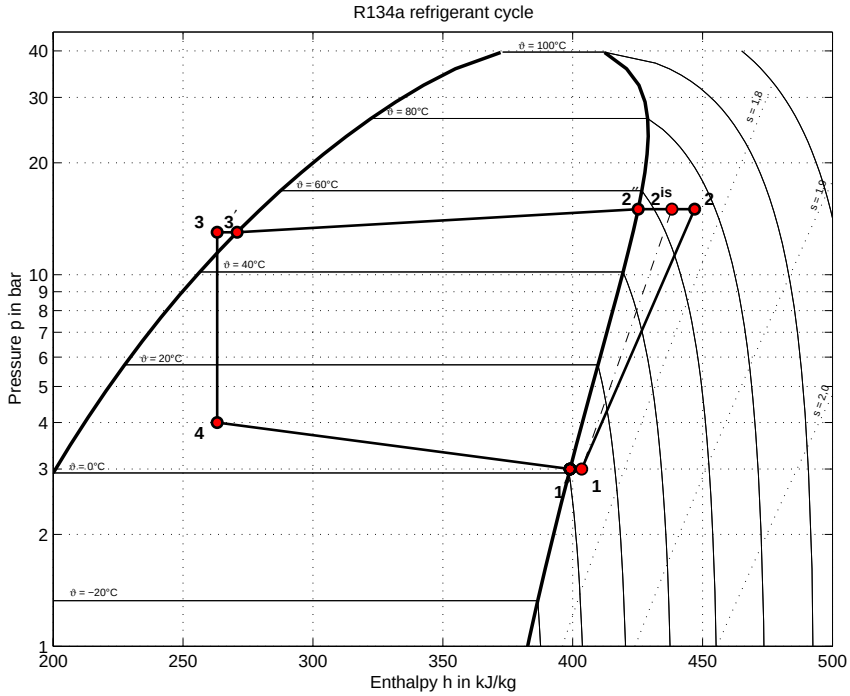


Figure 3.4: *Refrigerant cycle*

To calculate the real refrigerant cycle variables and properties, the functions used are the same as for the ideal cycle (section 3.2.1.1). The values for the *sub-cooling* (T_{sub}), *super-heating* (T_{sup}), the *isentropic efficiency* of the compressor (η_{is}), and the *pressure losses in the condenser and evaporator* (Δp_c and Δp_0) are necessary, and the procedure is as follows:

- **Point 1** (suction state)

  $p_1 = p_0 - \Delta p_0$ (Evaporation pressure is defined as the refrigerant pressure when *entering* the evaporator).

  $T_1 = T(p_s) + T_{sup}$ with $p_s = p_1$

  $s_1 = s(T_1, \rho_1)$ with $\rho_1 = \rho(T_1, p_1)$

  $h_1 = h(T_1, \rho_1)$

- **Point 2**^{is} (isentropic compression)

  $T_{2^{is}} = T(p_c, s_1)$

  $h_{2^{is}} = h(T_{1^{is}}, \rho_{2^{is}})$ with $\rho_{2^{is}} = \rho(T_{2^{is}}, p_c)$

- **Point 2** (real compression)

  $h_2 = \frac{h_{2^{is}} - h_1}{\eta_{is}} + h_1$

  $T_2 = T(h_2, p_c)$

  $s_2 = s(T_2, \rho_2)$ with $\rho_2 = \rho(T_2, p_c)$

- **Refrigerant mass flow rate**

  $\dot{m}_{ref} = \frac{P_{comp}}{h_2 - h_1}$

- **Condenser power**

  $Q_c = \dot{m}_{ref} \cdot (h_3 - h_2)$

- **Evaporator power**

  $Q_e = \dot{m}_{ref} \cdot (h_1 - h_3)$

- **cop**

  $cop = \frac{Q_e}{P_{comp}}$

The real refrigerant cycle:

With the above considerations it is possible to calculate the values for the condenser power, evaporator power, refrigerant mass flow rate as well as the cop for real refrigerant cycles. A MATLAB-script using the above relationships was utilised for this task and a typical set of parameters is provided below, to illustrate this:

- Evaporation pressure p_0=4bar,
- Condensation pressure p_c=15bar,
- Compressor power 2kW,
- Isentropic efficiency η_{isen}=80%,

- Sub-cooling by the condenser T_{sub}=5K,
- Overheating or superheating in the evaporator T_{sup}=5K,
- Pressure losses in the condenser Δp_c=2bar and
- Pressure losses in the evaporator Δp_e=1bar.

A MATLAB-script determined the following results:

- Condenser power -8,46kW,
- Evaporator power 6,46kW,
- Refrigerant mass flow rate $\dot{m}_{R134a}$=0,0461kg/h,
- cop= 3,23.

Furthermore, a graph of the results is generated as shown in Fig. 3.5.

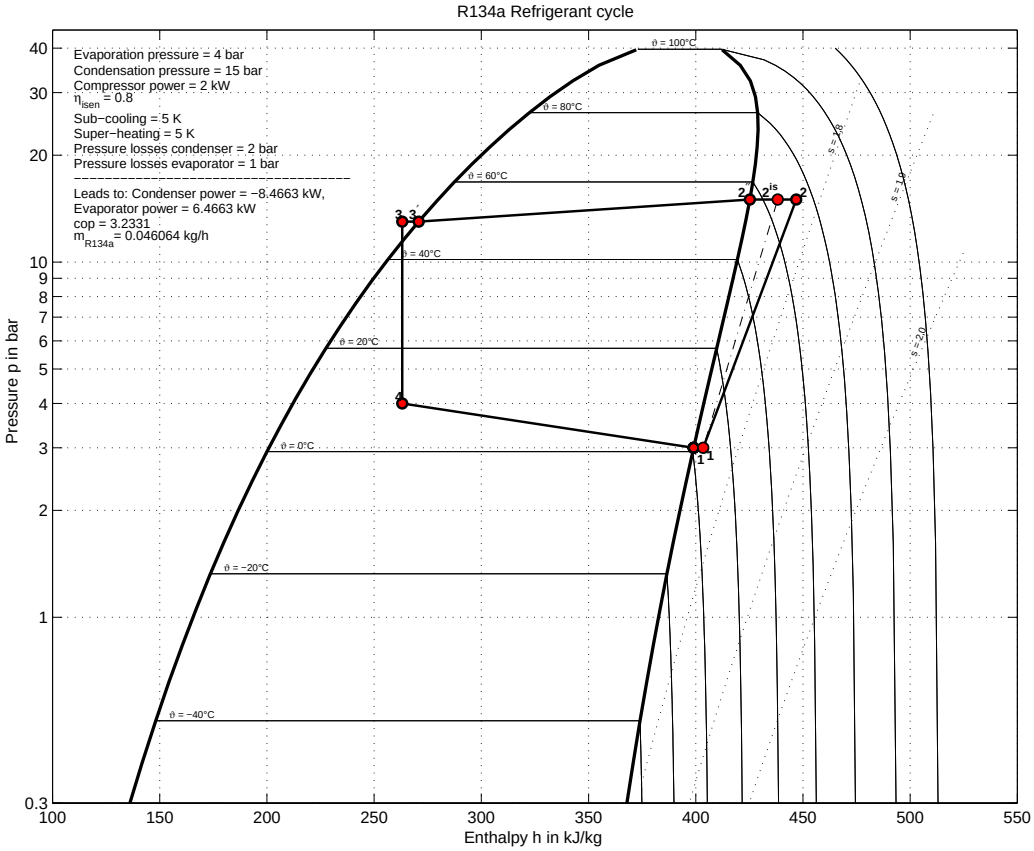


Figure 3.5: *Plot of a real refrigerant cycle simulation*

3.3 Conclusion

To examine the behaviour of the a/c system it is necessary to simulate the interface to the environment (e. g. the driving speed, ambient conditions, etc.) and to the passenger compartment.

The interface of the a/c system and the passenger compartment is the evaporator. It is, hence, essential that the refrigerant cycle is critically analysed. It too, has many variables and processes that are inter-related and the best and most convenient method of analysis is done using graphical methods.

The log p,h-diagram serves this end well. Also, slight modifications in the MATLAB-script-files would facilitate different scenarios with ease making the simulation adaptable and flexible.

The interfaces are described in the following chapters and will show the use of SIMULINK, instead of MATLAB-script-files, to adapt scenarios.

Chapter 4

Coupling of passenger compartment and refrigerant cycle model

The previous chapters described the function of the single models. Both, the passenger compartment model and the refrigerant cycle model can be handled as stand-alone applications, without any feedback from the other model.

To determine the energy consumption of an automotive a/c system it is essential to use a closed simulation for the passenger compartment and the refrigerant cycle and, hence, any inconsistencies between different simulations and transmission errors are ruled out. The link between the two models, the passenger compartment and the refrigerant cycle, is the evaporator as this is where the ventilation air is cooled by the refrigerant cycle.

This chapter describes the coupling of the two models via the evaporator. A simple algorithm is introduced to calculate the air temperature that enters the passenger compartment.

Furthermore, the use of a driving cycle for this research is also shown in this chapter because the driving speed influences the passenger compartment (i.e. the heat transfer coeficient) and the evaporator power of the refrigerant cycle, as the driving speed rules the compressor speed.

4.1 The Effectiveness-Number of Transfer Units (ε-NTU) method

To calculate the the air temperature at the evaporator outlet it is advisable to use the ε-NTU method, as oppossed to the logarithmic mean temperature difference (LMTD) method, as it is possible to calculate the outlet temperatures and transfered heat with the knowledge of the heat exchanger geometry and the inlet temperatures.

This method is based upon the dimensionless parameter ε, the *heat transfer efficiency*, which is defined as:

$$\varepsilon = \frac{\dot{Q}}{\dot{Q}_{max}} = \frac{\text{actual heat transfer rate}}{\text{maximum possible heat transfer rate}} \tag{4.1}$$

The actual heat transfer rate can be determined from an energy balance of the cold or hot fluid:

$$\dot{Q} = \dot{m}_c \cdot c_{p,c} \left(T_{c,out} - T_{c,in}\right) = \dot{m}_h \cdot c_{p,h} \left(T_{h,in} - T_{h,out}\right) \tag{4.2}$$

To calculate the maximum heat tranfer rate the maximum temperature difference is used, which in this case would be the difference of the inlet temperatures:

$$\Delta T_{max} = T_{h,in} - T_{c,in} \tag{4.3}$$

The maximum possible heat transfer rate is reached when the cold fluid is heated up to the inlet temperature of the hot fluid, or the hot fluid is cooled down to the inlet temperature of the cold fluid. These limiting conditions will not be reached simultaneously except in the case where the heat capacities of the hot and the cold fluid are identical ($C_h = \dot{m}_h \cdot c_{ph} = C_c = \dot{m}_c \cdot c_{pc}$), otherwise $C_h \neq C_c$. In most cases, the temperature difference is *greater* for the fluid with the *lower* heat capacity. It then follows that the fluid with the lower heat capacity rate reaches the maximum temperature change first at which point the heat transfer ceases.

Hence, the maximum possible heat transfer rate is:

$$\dot{Q}_{max} = C_{min} \cdot \left(T_{h,in} - T_{c,in}\right) \tag{4.4}$$

To calculate $\dot{Q}_{max}$ it is necessary to know the inlet temperature as well as the mass flow rate. With the aid of effectiveness, ε, it is possible to calculate the actual heat transfer rate $\dot{Q}$:

$$\dot{Q} = \varepsilon \cdot \dot{Q}_{max} = \varepsilon \cdot C_{min} \cdot \left(T_{h,in} - T_{c,in}\right) \tag{4.5}$$

The effectiveness, ε, of a heat exchanger is determined by it's *geometry* and the *fluid flow arrangement*. This means that different types of heat exchangers have different values for their effectiveness, ε. The effectiveness of different types of heat exchangers may be found in many popular heat transfer texts. References [11], [12] and [6] were used in this research. In the case of a parallel flow heat exchanger, as a simple example, it is expressed as:

$$\varepsilon_{\text{parallel-flow heat exchanger}} = \frac{1 - \mathrm{e}^{-\frac{k \cdot A}{C_{min}}\left(1+\frac{C_{min}}{C_{max}}\right)}}{1 + \frac{C_{min}}{C_{max}}} \tag{4.6}$$

C_{min} is the lower heat capacity flow rate and C_{max} is the higher one. It makes no difference which represents the cold fluid and the hot fluid.

The effectiveness relations of different heat exchangers include the dimensionless fraction $k \cdot A / C_{min}$. This value is called the **number of transfer units (NTU)** and is expressed as

$$NTU = \frac{k \cdot A}{C_{min}} = \frac{k \cdot A}{(\dot{m} \cdot c)_{min}} \tag{4.7}$$

Where, k is the heat transfer coefficient and A the surface area of the heat exchanger. Due to the proportional relationship of A and NTU it is obvious that the larger NTU, the larger is the heat exchanger.

Furthermore, it is convenient to introduce another dimensionless value for the ratio of the heat capacity flow rates:

$$C = \frac{C_{min}}{C_{max}} \tag{4.8}$$

The effectiviness of the heat exchanger is then determined with the dimensionless variables *NTU* and *C* as follows:

$$\varepsilon = f(NTU, C) \tag{4.9}$$

and for the parallel flow heat exchanger results in:

$$\varepsilon = \frac{1 - e^{-\mathrm{NTU}\cdot(1-\mathrm{C})}}{1+\mathrm{C}} \tag{4.10}$$

The analysis of the function leads to the following points:

1. The value for the effectiveness, ε, could be in a range between 0 and 1. The gradient is steep for growing NTU's until NTU reaches a value of approx. 1,5.

2. For given NTU and C-values the effectiveness is greatest for the counter-flow heat exchanger, followed closely by the cross-flow.

3. The effectiveness of a heat exchanger is independent of the capacity ratio for NTU values $\leq$ approximate 0,3.

4. The values for C are between 0 and 1. For a given NTU value, the effectiviness becomes a maximum when C=0 and a minimum when C=1. When C=1, this corresponds to equal heat capacity flow rates.

   One of the most important considerations with regard to the refrigerant cycle is the case when $C = 0$. This occurs when $C_{max} \to \infty$, which occurs when the media does not experience a temperature change during a phase change. This typically occurs during evaporating and condensing heat transfer. Hence, the relation for effectiveness for all

heat exchangers with $C = 0$, which is the case for all heat exchangers considered by this research, may be expressed as:

$$\varepsilon = \varepsilon_{max} = 1 - e^{-NTU} \tag{4.11}$$

This relationship is illustrated in Fig. 4.1

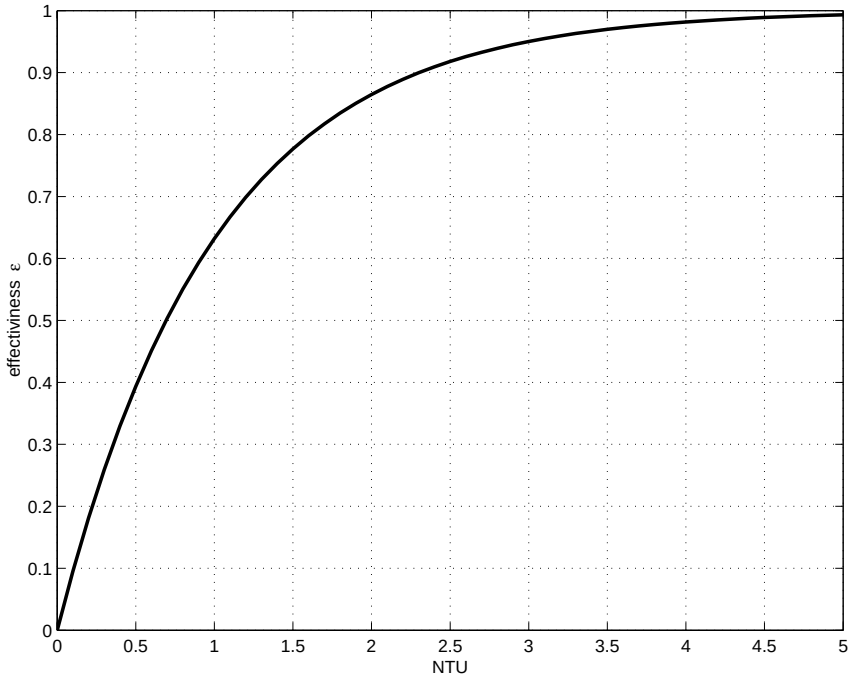


Figure 4.1: *ε-NTU relationship for* **all** *condensers and evaporators*

With this equation it is possible to calculate the heat transfer rate for known heat exchanger geometries.

Example:

Given is a typical evaporator with a surface area A of 2,15m^2 and a heat transfer coefficient k of 70 J/(kg·K) at a refrigerant mass flow rate of 0,08 kg/s and with an evaporation temperature of 3°C. The ambient temperature should be 40°C.

The maximum heat transfer rate $\dot{Q}_{max}$ is 3,85 kW using Eq. 4.4.

Eq. 4.7 gives an NTU value of 1,45 and Eq. 4.11 an effectiviness, ε, of 0,765. Hence, the heat transfer rate $\dot{Q}$ for this evaporator is 2,95 kW.

4.2 The driving cycle

To achieve reproducable simulation results it is necessary to use standarized parameters. The parameters required to simulate the driving conditions are given by the **N**ew **E**uropean **D**riving **C**yle (NEDC). The NEDC provides the values for the driving speed and selected gear.

The NEDC is a combination of four repeating cycles with low driving speeds (city cycles) and one with higher driving speed (transurban cycle). The city cylce also includes some “stop and go”-elements. Given the NEDC specifications for the selected gear and the specific gear ratio, it is possible to calculate the engine speed and hence the compressor speed.

Fig. B.5 and Fig. 4.2 illustrate a typical NEDC test for an *Audi A4 Avant quattro* while driving the NEDC.

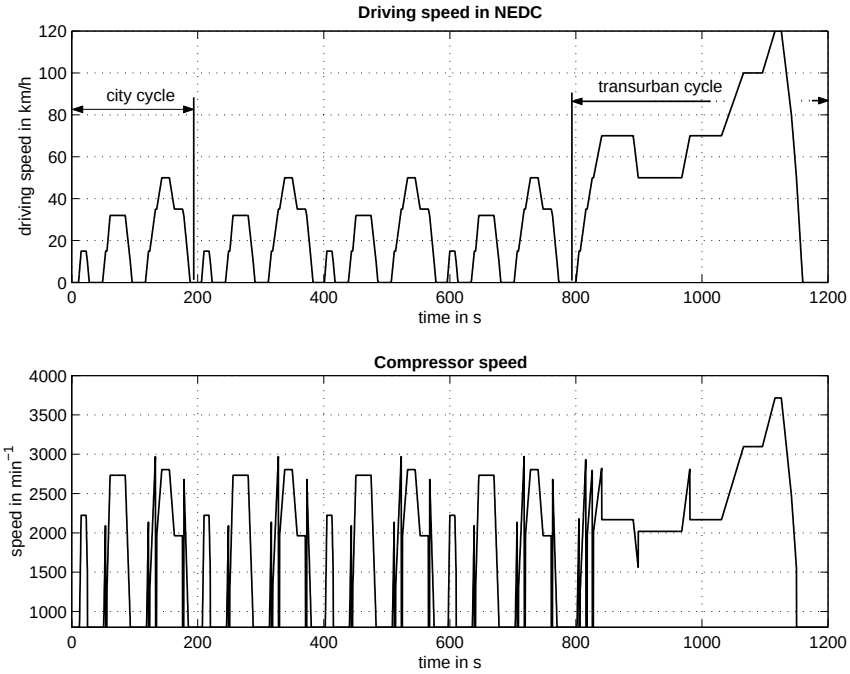


Figure 4.2: *Driving speed and engine speed during NEDC*

The compressor speed allows for the determination of the the evaporator power and, hence, with the ε-NTU-method it is possible to determine the air temperature leaving the evaporator.

Chapter 5

Results

To judge a mathematical model of a physical system it is necessary to compare the calculations with some results achieved experimentally. As the simulation is divided into two parts, the passenger compartment and the refrigerant cycle, the justification can be done seperately for each part.

As mentioned in the Introduction (Chapter 1), it is very time consuming and costly to install a test bench to make all the measurements that are requiered to fulfill a complete validation of *all* simulated values. As this was not part of this research, only some measurements were done to demonstrate the accuracy of the mathematical models.

5.1 The Passenger compartment

A first check of the passenger compartment model was established with the experimental comparison to a real car. Temperature sensors were placed in the car to record the transient temperatures of the air and the interior components. An additional outside air temperature sensor was applied. This temperature was the input value for the simulation. Which means that the outside air temperature followed the measured outside air temperature in the simulation. As stated in earlier sections of this research, some parameters used in the simulation, such as

- The surface area of the interior components.
- The mass of the interior components.
- The heat transmission coefficient of the interior components.
- The heat capacity of the interior components.

are very difficult to determine and, hence, the initial values for these parameters were estimated and adjusted within physical meaningful ranges until the transitory temperature of the simulation matched the measured one. This is illustrated in Fig. 5.1

The results indicate that a high degree of accuracy is achievable with the given parameters, which indicates that the simulation model for the passenger compartment is a good representation of the actual passenger compartment.

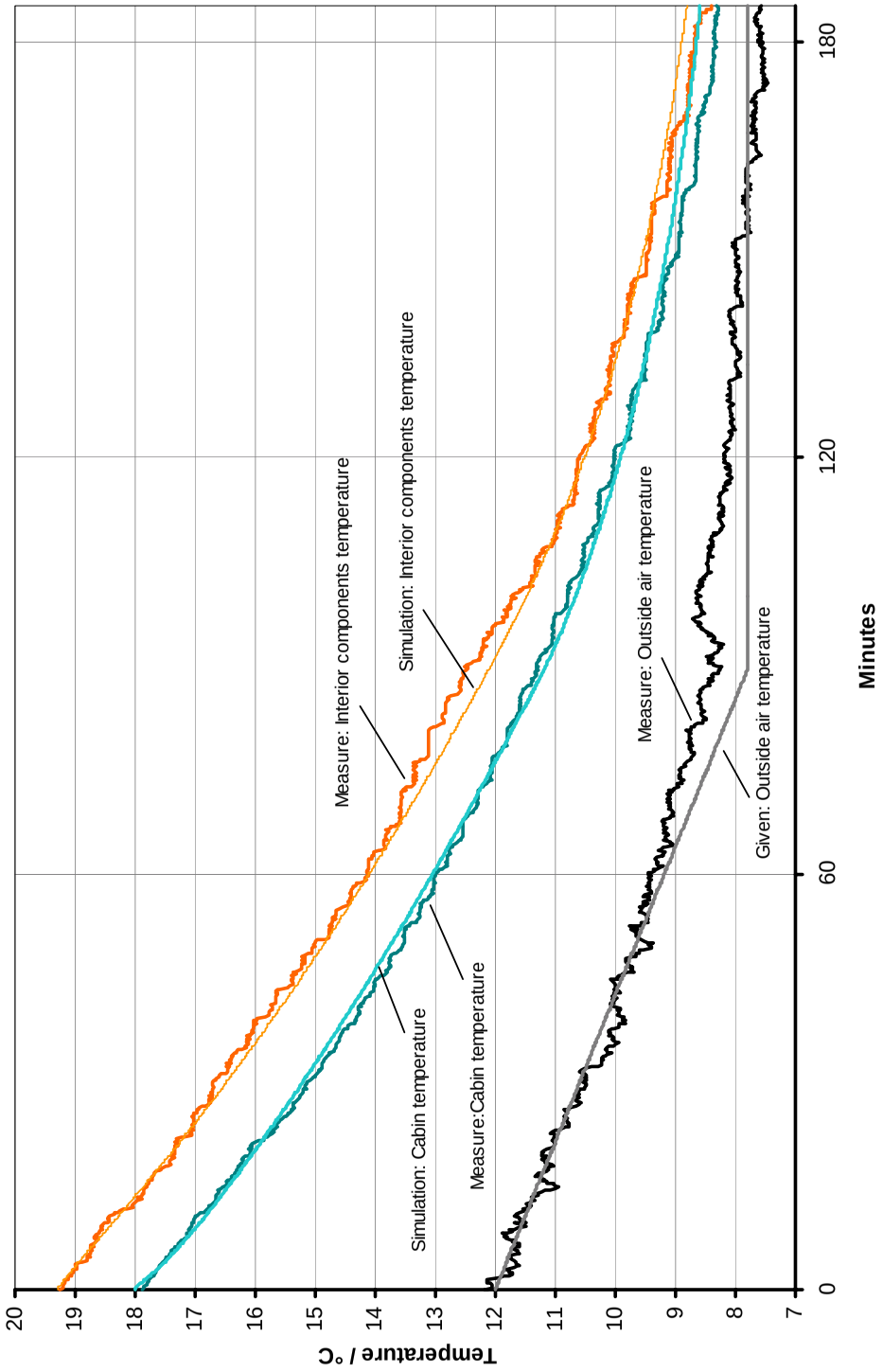


Figure 5.1: *Measured and simulated results*

5.2 The instationary simulation NEDC

The following explanatory calculations give an overview of the thermal analytic possibilities with the simulation. The temperatures of several body parts, windows and, of course, the passenger compartment air temperature would be calculated while completing a driving cycle. Additionaly, the ventilation mass flow rate is shown, to illustrate the control stategy (see Fig. 2.12).

Scenario 1:

The result of a simulation is shown with following parameters driving the NEDC:

- Environmental temperature, ϑ_a =27°C
- Starting temperature inside: 40°C (soaked)
- Target temperature: 20°C
- Relative humidity, outside: 40%
- Max. mass flow rate: 9kg/min
- With 800W direct solar radiation
- and 50W diffuse radiation
- Driving speed and engine speed corresponding to NEDC and specific car parameters (The vehicle used was a Audi A4 Avant quattro 1,8l)
- Sun beam from behind

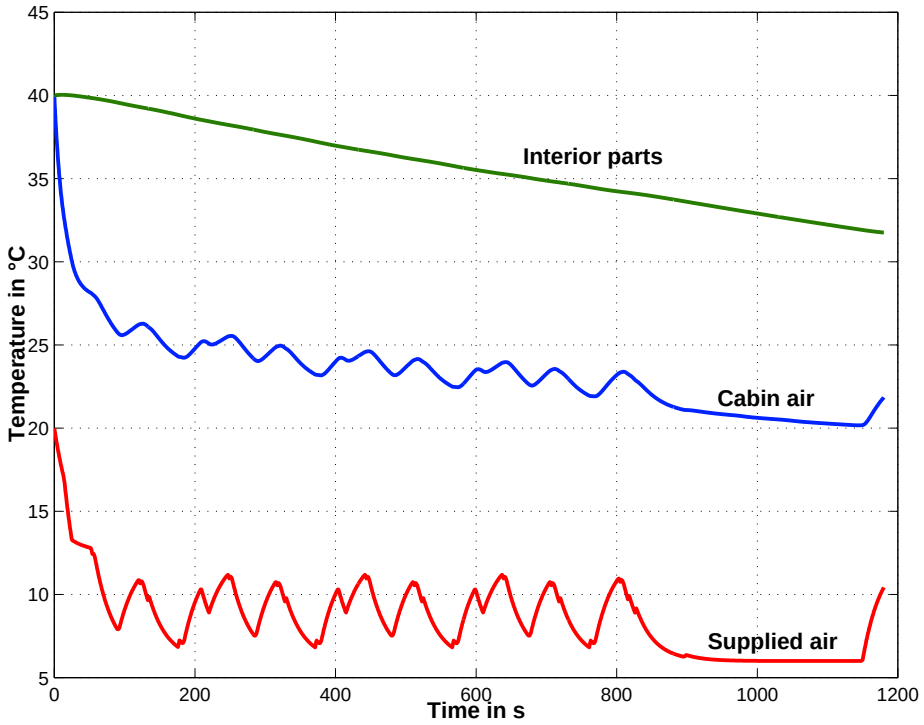


Figure 5.2: *Air cabin temperature and ventilation air temperature at ϑ_a=27°C driving the NEDC*

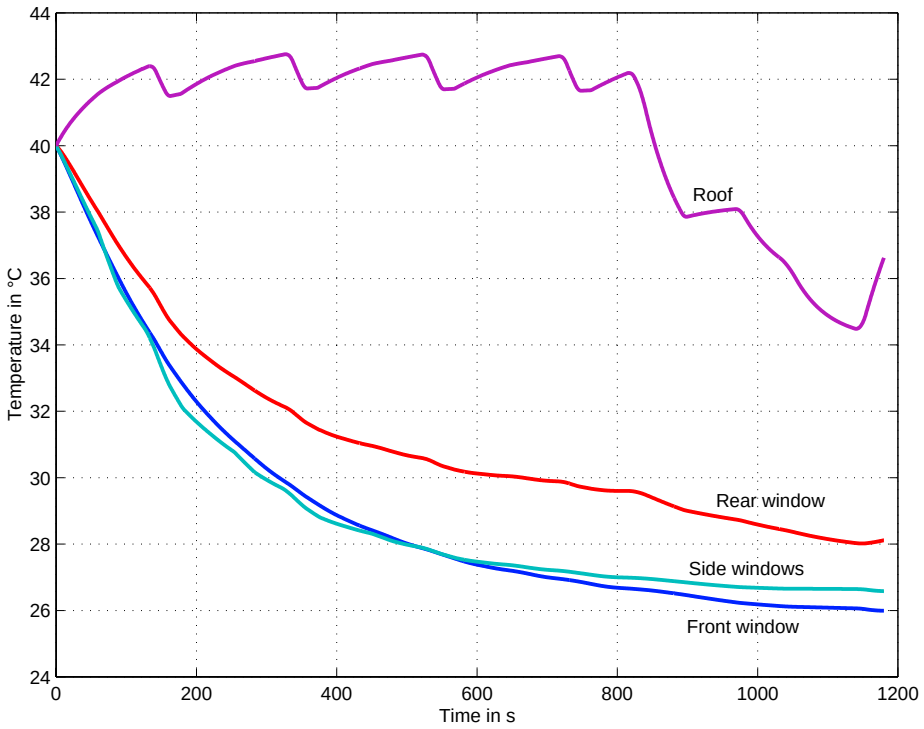


Figure 5.3: *Surface temperatures of the interior components at ϑ_a=27°C driving the NEDC*

Fig. 5.2 shows the cabin air temperature and the ventilation air temperature while driving the NEDC. The temperatures of the body and the windows is shown in Fig. 5.3.

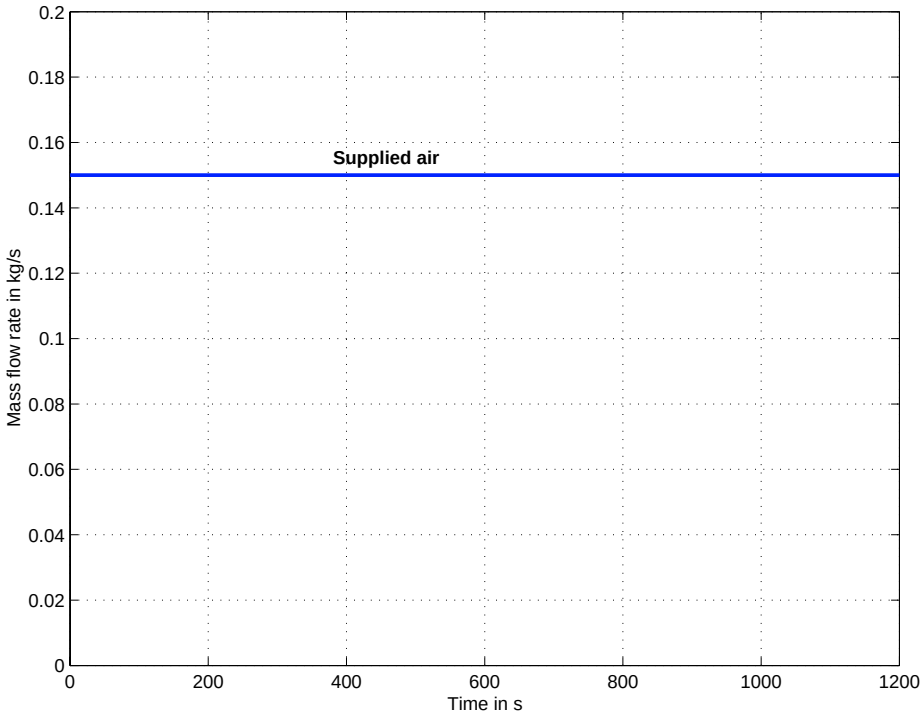


Figure 5.4: *Mass flow rate of ventilation air at ϑ_a=27°C driving the NEDC*

As the cabin air temperature is controlled by the ventilation mass flow rate it's corresponding values are shown in Fig. 5.4. It is obvious from the graph that the mass flow rate never sinks below the maximum of 0,15 kg/s (9 kg/min). This means that the entire cooling capacity produced, is required to reach the target cabin temperature of 20°C, otherwise the heater valve would open and the control would be as mentioned in section 2.1.7.

Figure 5.2 shows some peaks in the curve for the cabin air temperature. They are direct results of the peaks in the curve for the supplied air temperature. Compared with the driving cycle (see Fig. 4.2) one can see that the compressor speed affects the cooling capacity and hence the temperature of the supplied air, as the cooling capacity is directly proportional to the mass flow rate and the temperature difference, typically:

$$\dot{Q} = k \cdot \dot{m} \cdot \Delta T = k \cdot \dot{m} \cdot (T_1 - T_2) \tag{5.1}$$

Scenario 2:

The characteristic of the control can be analysed with slightly modified values for some parameters compared to scenario 1:

- Outside air temperature, ϑ_a =25°C
- Starting temperature inside: 30°C
- Target temperature: 22°C
- Sun beam from left-behind.

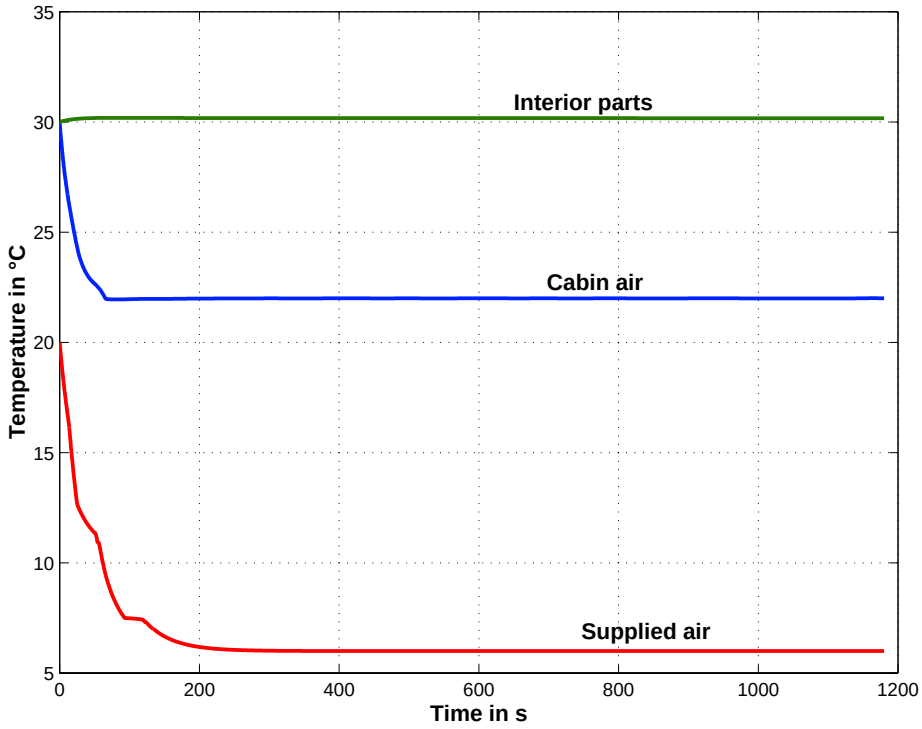


Figure 5.5: *Cabin temperatures and ventilation air temperature at ϑ_a=25°C driving the NEDC*

The temperatures of the body and the windows are shown in Fig. 5.6.

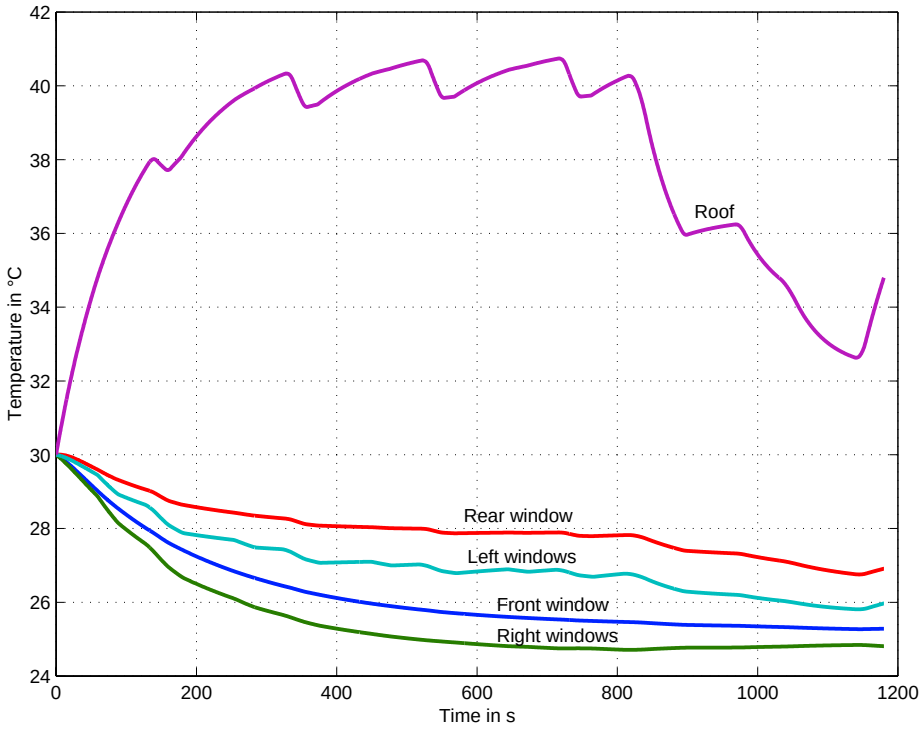


Figure 5.6: *Surface temperatures at ϑ_a=25°C driving the NEDC*

Fig. 5.7 shows the ventilation mass flow rate at a outside air temperature of 24°C.

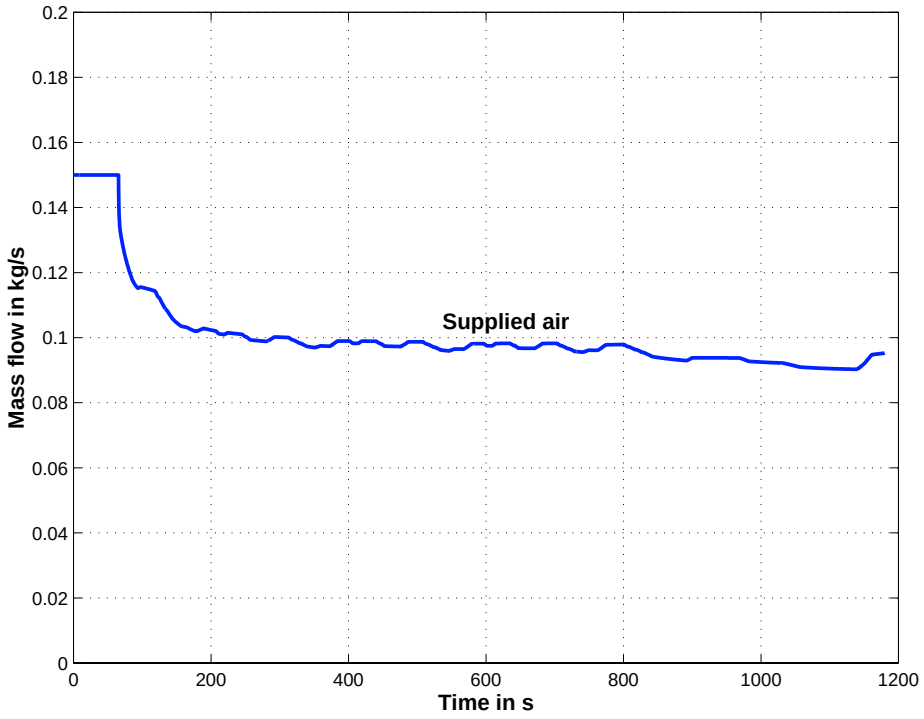


Figure 5.7: *Ventilation mass flow rate at ϑ_a=25°C driving the NEDC*

Fig. 5.5 illustrates that the installed cooling capacity is sufficient for this scenario. The temperature of the supplied air reaches the set point of 6°C after approximately 5 minutes. The cabin air reaches the target temperature of 22°C after approximately 3 minutes. Now, from fig. 5.7, it is obvious that the ventilation mass flow rate is the variable that controls the cabin air temperature.

5.3 Conclusion

The established simulation model for automotive a/c systems achieved good results in the few comparisons made with measured systems. The results indicate the ability to adapt the model to real systems.

The parameter variations illustrate the ability to analyze the sytem behaviour and control characteristics and, hence, energy and comfort judgment is possible.

Chapter 6

Conclusion

The simulation environment represents an optimal base for energy and comfort analysis in automotive air conditioning systems by facilitating adjustments to the environmental conditions or to operate them via *script*-files, therefore, making it an ideal basis for the comparision of simulation results with experimental results using identical environmental conditions.

With this kind of analysis it is essential to establish useful values for parameters empirically. This means that several values that are difficult to measure, like the heat capacity of the interior components, have to be estimated and, perhaps, adjusted to fit the experimental results. This is especially important in the case of transitory behavior. However, on establishing the set of parameters, all further conditions can be determined.

While the simulation is user friendly and intuitive due to SIMULINK, it is a complex system to change, e.g. changes to the physical behaviour of the components are only possible with modifications in one or more of the many C-files. The use of such C-files is, unfortunately, necessary due to the complexity of the real system which leads to a mathematical model that is too large for the built-in interpreter of MATLAB. To accomodate the demands for a one-dimensional simulation (rapid parameter variation and fast results) it is necessary to use compiler-languages. All the described simulations were performed with C-files and never took longer than 5-10 minutes for the NEDC simulation using a Pentium II 350MHz, 128Mb memory and MS Windows NT 4.0.

The system is comprehensive and provides a major advantage in fast and accurate model enhancements and refinement.

The above mentioned complexity of the system leads to huge elbowroom for model enhancements and refinement. This means more realistic modeling of the refrigerant cycle components or a subdivided passenger compartment. The implementation of innovative control concepts and systems, e.g. external controlled compressors, internal heat exchangers, controlled recirulation air modes, can be done to perform energy optimizations for the a/c system. This simulation environment establishes a solid foundation for such investigations.

With the extension of the program library, with characteristic charts of real components, it should be possible to use the simulation environment for the development and configuration of new components to compute energy consumptions under realistic conditions.

As typical a/c systems are responsible for 7-8% of the overall fuel consumption of a motor vehicle [13], tools such as this become invaluable in evaluating, predicting and refining the performance of components and variables influencing the energy consumption of a vehicle, without compromising the comfort and safety of the occupants.

Bibliography

[1] W. Eichsleder et al. *Optimierung des Wärmemanagements von Kraftfahrzeugen mit Hilfe von Simulationswerkzeugen*, pages 161–186. Expert Verlag, 1998.

[2] S. Shimizu, H. Hara, et al. Analysis on air-conditioning heat load of passenger vehicle. In *JSAE Review*, Nov. 1982.

[3] Arbeitskreis der Dozenten der Klimatechnik, editor. *Handbuch der Klimatechnik*, volume 1. Verlag C.F. Müller, Karlsruhe, 1988.

[4] W.-H. Hucho. *Aerodynamik des Automobils*. VDI Verlag, Düsseldorf, 1994.

[5] J. A. Duffie and W. A. Beckman. Sonnenenergie: Thermische Prozesse, 1976. München.

[6] VDI Wärmeatlas, 1997.

[7] H. Recknagel and E.-R. Schramek. *Taschenbuch für Heizung und Klimatechnik*. Schramek, E.-R., München, Oldenbourg, 1999.

[8] H. J. Cerbe and G. Hoffman. *Einführung in die Thermodynamik*. Carl Hanser Verlag, München Wien, 1994.

[9] Baehr and Tillner-Roth. *Thermodynamische Eigenschaften umweltvertäglicher Kältemittel*. Springer Verlag, 1995.

[10] K. Beetz and F. Bohlender. Elektrische Zuheizsysteme. In *Wärmemanagement des Kraftfahrzeugs*.

[11] Y. Cengel. *Heat transfer: A practical approach*. McGraw-Hill, 1998.

[12] W. M. Kays and A. L. London. *Compact Heat Exchangers*. Krieger Pblishing Company, 3 edition, 1994.

[13] B. Taxis-Reischl. Energieverbrauch von Klimaanlagen und Wege zur Verbrauchsreduzierung. *ATZ*, 99(9):7–10, 1997.

[14] B. Adiprasito. Simulation des instationären Verhaltens einer Pkw-Klimaanlage mit CO_2 als Kältemittel. Technical report, VDI Fortschrittsberichte, Reihe 12, Nr. 346, 1998.

[15] H. Bossel. *Modellbildung und Simulation*. Vieweg Verlag, Braunschweig/Wiesbaden, 1992.

[16] J. Currle. Einsatz der Strömungsberechnung für Fragestellungen der Fahrzeugklimatisierung und der Aeroakustik bei Mercedes-Benz. Technical report, Mercedes-Benz AG, Stuttgart.

[17] N. Deussen. Instationärmodell für das Fahrzeugkühlsystem als EXCEL-Software. *ATZ*, 98(6):358–361, 1996.

[18] A. Dick and B. Winter. Simulationsmodelle für die Auslegung von Heizungs- und Klimaanlagen in Pkw. *VDI Berichte*, (816):355–366, 1990.

[19] W. Eichsleder, J. Hager, et al. Auslegung von Kühlsystemen mittels Simulationsrechnung. *ATZ*, 99(10):638–646, 1997.

[20] U. Essers, H. Kruse, et al. Numerische Simulation des Motorkühlsystems von klimatisierten Automobilen. *ATZ*, 96(2):108–115, 1994.

[21] O. Gaveau and D. Clodic. Test bench for measuring the energy consumption of an automotive air conditioning system. In *Automotive Climate Control*, number SP-1347, pages 81–89. SAE, 1998.

[22] O. Gehl. Energieeinsparung durch kontrollierte Umluftregelung in Fahrzeugklimaanlagen. Master's thesis, Fachhochschule Braunschweig/Wolfenbüttel, 1997.

[23] M. Giradet. Strömungsberechnung im Automobilbau, Stand der Technik und Perspektiven. Technical report, BMW AG, München, 1996.

[24] *Technische Thermodynamik: Einführung und Anwendung*. Addison-Wesley, Bonn, 1993.

[25] J. Himmelsbach, P. Dilgen, et al. Zum Kraftstoffeinsparpotential bei Minimierung der Wärmeverluste des Motors an die Umgebung. In *Wärmemanagement des Kraftfahrzeugs*. Haus der Technik Essen, 22.-23. Sep. 1998.

[26] H. Kampf, J. Maggioncalda, et al. Pkw-Klimaanlagen: Kraftstoffersparnis durch Regelung der Verdampfertemperatur. *ATZ*, 99(7):442–445, 1997.

[27] Y. Khamsi and C. Petitjean. Modeling of automotive passenger compartment and its air conditioning system. In *Automotive Climate Control*, pages 35–43. SAE.

[28] J. Köhler. *Wärme- und Stoffübertragung in Zweiphasenströmungen*. Vieweg Verlag, Braunschweig/Wiesbaden, 1996.

[29] H. Lutz and W. Wendt. *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, Frankfurt am Main, 1998.

[30] J. Maué, J. Brucker, et al. Numerische Simulation transienter Betriebszustände von Kraftfahrzeug- Motorkühl- und Klimatisierungssystemen. Technical Report 816, VDI Berichte, 1990.

[31] E. Pott. Energieflußmanagement zur Kraftstoffverbrauchs,- Emissions- und Heizkomfortoptimierung. *ATZ*, 100(8):480–488, 1998.

[32] D. Schlenz and G. Seider. Simulation des sommerlichen Fahrzeugklimas durch Kopplung von Teilkreisläufen. In *Wärmemanagement des Kraftfahrzeugs*. Haus der Technik Essen, 22.-23. Sep. 1998.

[33] *Kälte- und Klimatechnik*. Solkane™- 1997.

[34] F. Werner, S. Fell, et al. Strömungsberechnung: Ein integraler Bestandteil der Motoren-, Karosserie- und Klimasystementwicklung. In *Numerische Strömungsberechnung in der Automobilindustrie*. Haus der Technik Essen, 21.-22. Nov. 1996.

[35] B. Upmeier. Untersuchung des instationären Verhaltens von Wärmepumpenkreisläufen. In *Forschungsberichte des DKV*. 1996.

[36] H. D. Baehr and K. Stephan. *Wärme- und Stoffübertragung*. Springer Verlag, 1994.

Appendix A

Stipulated values for the SIMULINK-model

The most important parameters for the best flexibility are not placed in the subsystems and functions, but in the constant blocks available on the main level of the simulation environment. These parameters are tabled below:

Interior components	
Heat capacity of the interior components	Relevant heat capacity of all interior components in $\frac{kJ}{K}$.
Heat transmission coefficient	Average heat transmission coefficient of all interior components in $\frac{W}{m^2 \cdot K}$.
Surface	Surface area of the interior components that are in contact with the air, in m^2.
air mass	Mass of the air in the passenger compartment in kg.

Properties of the air	
Outside air temperature	Temperature of the outside air in °C.
Outside humidity	Relative humidity of the outside air: $0 \leq \varphi \leq 1$.
Temperature set point	Set point of the inside air temperature in °C.
Humidity inside	Relative humidity of the cabin air: $0 \leq \varphi \leq 1$.
Initial temperature	Starting temperature for inside air, interior components, windows and body in °C.
Ambient pressure	Ambient pressure in Pa.

Mass flow rates	
Maximum ventilation mass flow rate	The maximum ventilation mass flow rate supplied by the ventilation pipes in $\frac{kg}{s}$.
Fraction of recirculated air	Part of the recirculated air on the overall ventilation mass flow rate with $0 \leq$ *recirculated air fraction* ≤ 1.

Solar radiation	
Total irradiance	The surface density of the radiant flux for a surface that is perpendicular to the beam in $\frac{\mathrm{kW}}{\mathrm{m}^2}$.
Diffuse irradiance	The surface density of the radiant flux for a surface that is diffuse radiated in $\frac{\mathrm{kW}}{\mathrm{m}^2}$.
Day/ month/ location/ - time	Day and month of the simulation date. For the local time should the *true* local time be used. The *standard* local time can be used for an approximation. The indication is in hours. Decimals can be entered.
Driving direction	The direction (cardinal point) the car is headed to. Indicate in degrees: $0° \leq \varphi \leq 360°$.
Latitude	The geographic latitude of the location of the simulation in degrees. For the northern hemisphere the latitude is positive and for the southern hemisphere negative.
Body surface	Surface area of the body that is surrounded by the passenger compartment in m^2.
Window surface	Surface Area of the front, rear and side windows in m^2. The area of the left and the right windows must be entered for the side windows.
Angle of windows	Angle between horizon and inside ´surface of front, side and rear windows. Indicated in degrees.
Radiation transmission coefficient of the glass	The portion of the radiation that is transmitted through the glass: $0 \leq d_S \leq 1$.
Absorbtion coefficient of the glass	The portion of the radiation that is absorbed by the glass: $0 \leq a_{Glas} \leq 1$.
Absorption coefficient of the lacquer	The portion of the radiation absorbed by the lacquer on the body: $0 \leq a_{Lack} \leq 1$.

Miscellaneous	
Driving speed	So long as there is no driving cycle the driving speed can be entered in km/h.
Driving direction	The driving direction in degrees: $0 \leq \varphi \leq 360°$. Alternatively a rotating driving direction may be specified.
Number of passengers	Number of persons in the cabin.

Appendix B

Subsystems of the SIMULINK passenger compartment model

B.1 Interior components

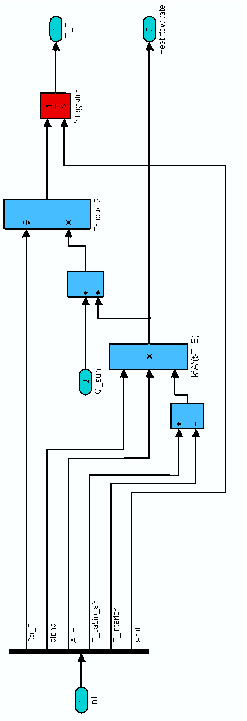


Figure B.1: *Subsystem of the interior components*

B.2 Cabin air temperature

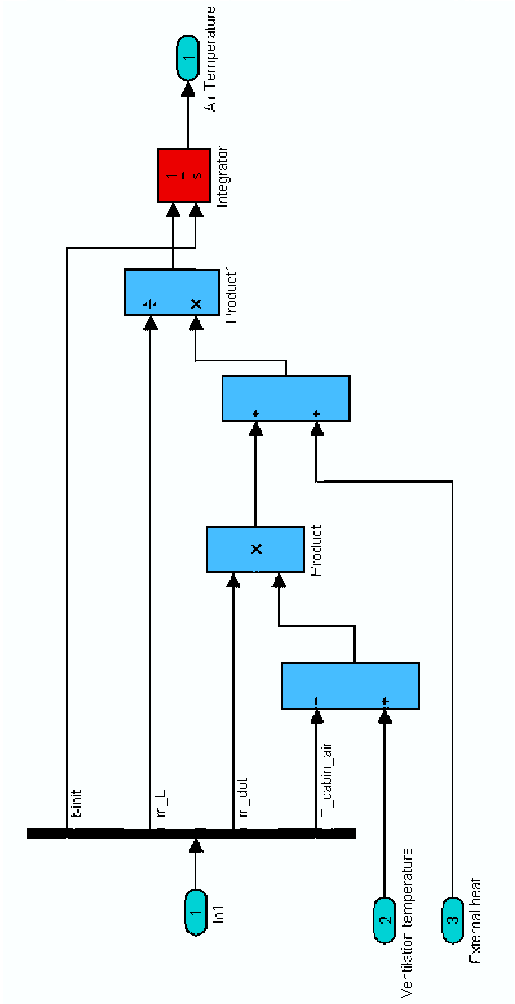


Figure B.2: *Subsystem for the cabin air temperature*

B.3 Control system for the ventilation air temperature

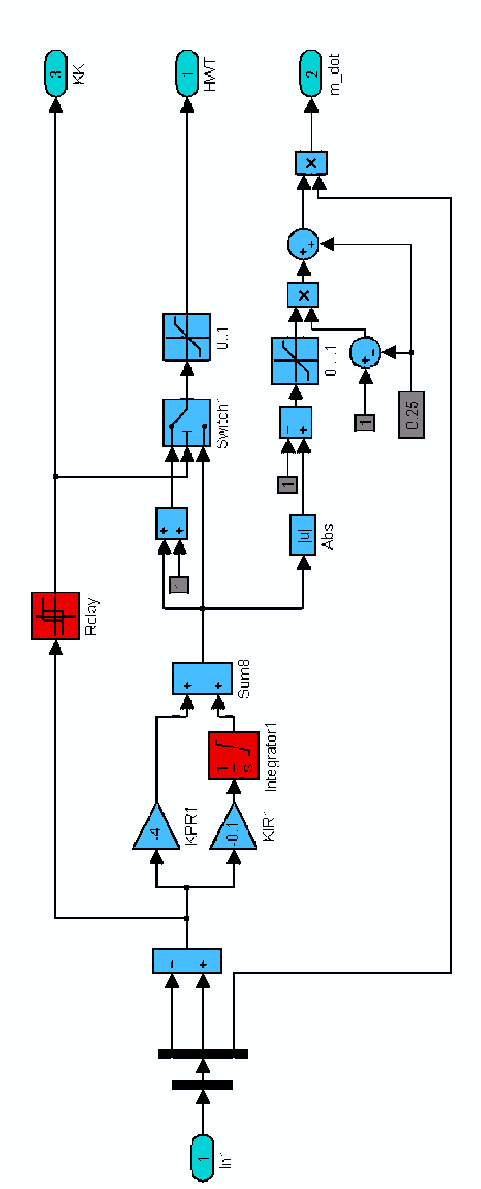


Figure B.3: *Control system for the ventilation air temperature*

B.4 Passengers

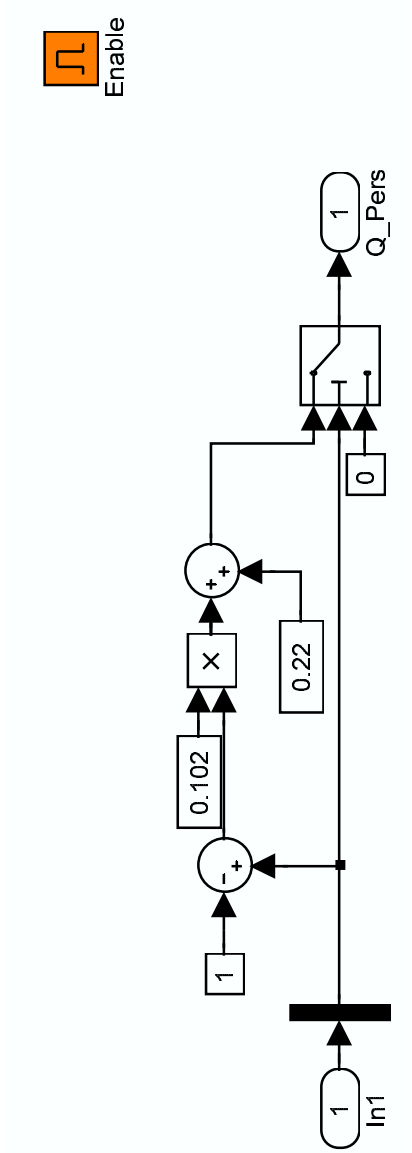


Figure B.4: *Subsystem of the passengers*

B.5 Compressor speed

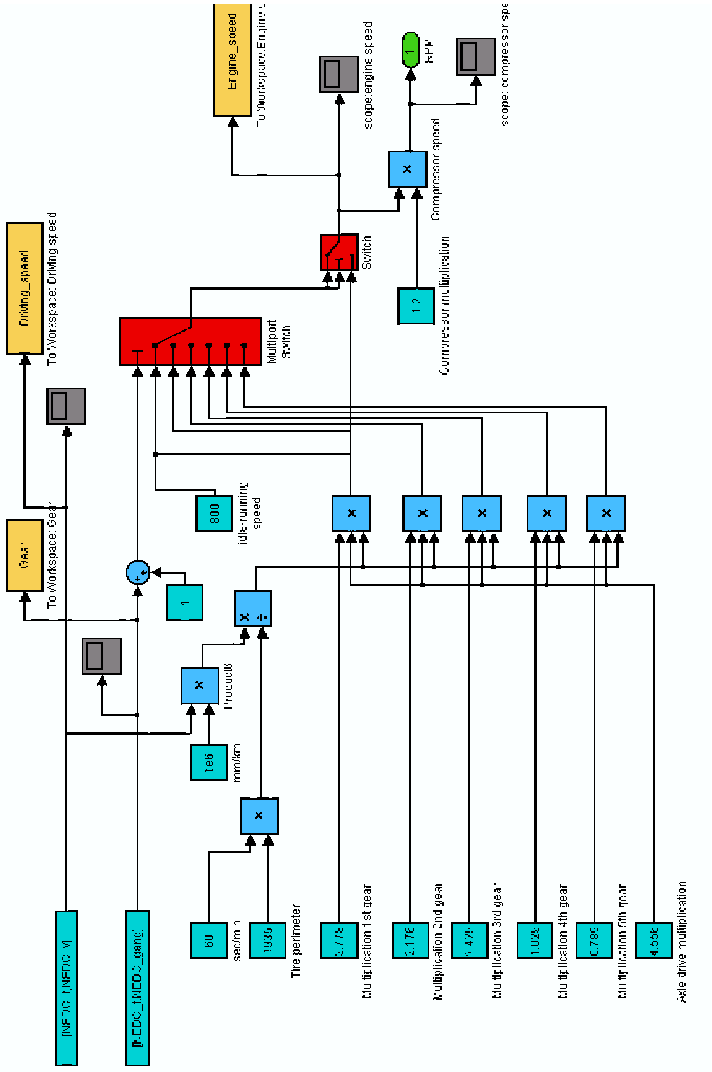


Figure B.5: *Subsystem to calculate the compressor speed*

Appendix C

Source-Code

C.1 Properties of air

C.1.1 x(t,φ)

This function calculates the humidity of air in kg/kg at a given temperature in °C and rel. humidity as a %.
Function call:

x_t_phi(temperature in °C, rel. humidity in °C), e. g.

```
>>x_t_phi(25,50)
ans =
0.0059
```

Sourcecode:

```
function x_t_phi = x_t_phi(t,phi);
%--------------------------------------------------------------
% --------------------- x_t_phi.m ----------------------------
% --------------------------------------------------------------
% This function calculates the humidity of air in ------------
% kg/kg at a given temperature in° C and rel. humidity in %.--

x_t_phi=0.622*phi/100*ps_t(t)./(101325-phi/100*ps_t(t));
```

C.1.2 p_S(x)

This function calculates the pressure of water saturated air in Pa at a given absolute humidity in kg/kg.
Function call:

ps_x(humidity in kg/kg), e. g.

```
>>ps_x(10e-3)
ans =
1.6032e+003
```

Sourcecode:

```
function ps_x = ps_x(x);
%------------------------------------------------------------------
% ------------------------ ps_x.m ----------------------------
% ------------------------------------------------------------------
% calculates the pressure of water saturated air in Pa at a --
% given absolute humidity in kg/kg.

ps_x=x.*101325./((0.622+x));
```

C.1.3 $p_S(t)$

This function calculates the pressure of water saturated air in Pa at a given temperature in °C.
Function call:

ps_t(temperature in °C), e. g.

```
>>ps_t(25)
ans =
3.1653e+003
```

Sourcecode:

```
function ps_t = ps_t(t);
%------------------------------------------------------------------
% ------------------------ ps_t.m ----------------------------
% ------------------------------------------------------------------
% calculates the pressure of water saturated air in Pa at a --
% given temperature in° C. ----------------------------------

ps_t=288.6*(1.098+t./100).^8.02;
```

C.1.4 x(t,φ)

This function calculates the absolute humidity of air in kg/kg at a given temperature in °C and rel. humidity as a %.

This function calculates the pressure of water saturated air in Pa at a given absolute humidity in kg/kg.
Function call:

x_t_phi(temperature in °C, humidity in %), e. g.

```
>>x_t_phi(25,50)
ans =
0.0099
```

Sourcecode:

```
function x_t_phi = x_t_phi(t,phi);
%-------------------------------------------------------------
% ------------------------ x_t_phi.m -------------------------
% -------------------------------------------------------------
% calculates the absolute humidity of air in kg/kg at a given
% temperature in° C and rel. humidity in %. ------------------

x_t_phi=0.622*phi/100*ps_t(t)./(101325-phi/100*ps_t(t));
```

C.1.5 h(t,φ)

This function calculates the specific enthalpy for humid air in kJ/kg at a a given temperature in °C and rel. humidity as a %.
Function call:

> **h_t_phi**(temperature in °C, rel. humidity in °C), e. g.
> `>>h_t_phi(25,50)`
> `ans =`
> `50.2982`

Sourcecode:

```
function h_t_phi = h_t_phi(t,phi);
%-------------------------------------------------------------
% ------------------------ h_t_phi.m -------------------------
% -------------------------------------------------------------
% This function calculates the specific enthalpy for humid air
% in kJ/kg at a given temperature in° C and rel. humidity in %.

x=0.622.*phi./100*ps_t(t)./(101325-phi./100.*ps_t(t));
h_t_phi=1.006.*t+x.*(1.86.*t+2501.6);
```

C.2 R-134a properties (Matlab)

The following functions are written in the MATLAB-language. This is an intuitive laguage which uses the MATLAB- interpreter.

C.2.1 T(p_s)

This function calculates the temperature for R134a in K at a given pressure in Pa on the line of saturated vapour.
Function call:

t_ps(pressure in Pa), e. g.

```
>>t_ps(10e5)
ans =
312.5360
```

Sourcecode:

```
function T_ps = T_ps(ps);
%-------------------------------------------------------------
% ------------------------- t_ps.m ----------------------------
% -------------------------------------------------------------
% This function calculates the temperature for R134a in K at a
% given pressure in Pa on the line of saturated vapour.-------

aeins = -7.7057291;
azwei = 2.4186313;
adrei = -2.1848312;
avier = -3.4530733;

r =   81.488856;
tc =  374.18;
rhoc = 508;

pnull=4.056318*10^6;

for j = 1:size(ps,2)
  theta = 1-(300/tc);
  thetaeins = 1-(240/tc);
  thetazwei = 0;

  for i = 1:30
    thetadrei = thetazwei + 0.5*(thetaeins-thetazwei);
    fvontheta_eins = (exp(1./(1-thetaeins).*(aeins.*...
      thetaeins + azwei.*thetaeins.^1.5 + adrei.*...
      thetaeins.^2 + avier.*thetaeins.^4))*pnull)-ps(j);
    fvontheta_zwei = (exp(1./(1-thetazwei).*(aeins.*...
      thetazwei + azwei.*thetazwei.^1.5 + adrei.*...
```

```
      thetazwei.^2 + avier.*thetazwei.^4))*pnull)-ps(j);
    fvontheta_drei = (exp(1./(1-thetadrei).*(aeins.*...
      thetadrei + azwei.*thetadrei.^1.5 + adrei.*...
      thetadrei.^2 + avier.*thetadrei.^4))*pnull)-ps(j);
    if fvontheta_zwei.*fvontheta_drei >= 0
      thetazwei = thetadrei;
    else
      thetaeins = thetazwei;
      thetazwei = thetadrei;
  end;
end;
  T_ps(j) = (1 - thetazwei)*tc;
end;
```

C.2.2 h′(T)

This function calculates the specific enthalpy for R134a in J/kg at a given temperature in K on the line of saturated liquid.
Function call:

hstrich_T(temperature in K), e. g.

```
>>hstrich_T(333)
ans =
2.8726e+005
```

Sourcecode:

```
function hstrich_T = hstrich_T(T);
%-------------------------------------------------------------
% ----------------------hstrich_T.m ------------------------
% -------------------------------------------------------------
% This function calculates the specific enthalpy for R134a ---
% J/kg at a given temperature in K on the line of saturated --
% liquid.-----------------------------------------------------

neins = -0.534728;
nzwei = -1.757777;
ndrei = -0.928326;
nvier = -2.666564;
nfuenf = -30.82570;

hnull = 384.07e3;
tc =  374.18;
rhoc = 508;

tau = tc./T;
theta = 1- T./tc;
```

```
hstrich_T = hnull*exp(neins*theta.^(1/2) + nzwei*...
    theta.^(5/4) + ndrei*theta.^3 + nvier*theta.^4...
    + nfuenf*theta.^10);
```

C.2.3 h″(T)

This function calculates the specific enthalpy for R134a in J/kg at a given temperature in K on the line of saturated vapour.
Function call:

hzweistrich_T(temperature in K), e. g.

```
>>hzweistrich_T(333)
ans =
4.2659e+005
```

Sourcecode:

```
function hzweistrich_T = hzweistrich_T(T);
%-------------------------------------------------------------
% -------------------- hzweistrich_T.m ----------------------
% -------------------------------------------------------------
% This function calculates the specific enthalpy for R134a ---
% J/kg at a given temperature in K on the line of saturated --
% vapour.-----------------------------------------------------

hzweistrich_T = h_T_rho(T,rho_T_p(T,ps_T(T)));
```

C.2.4 h(t,ρ)

This function calculates the specific enthalpy for R134a in J/kg at a given temperature in K and density in kg/m^3 in the *overheated gaseous phase*.
Function call:

h_t_rho(temperature in K, density in kg/m^3), e. g.

```
>>h_t_rho(333,43.3)
ans =
4.4138e+005
```

Sourcecode:

```
function h_T_rho = h_T_rho(T,rho);
%-------------------------------------------------------------
% ----------------------- h_t_rho.m -------------------------
% -------------------------------------------------------------
% This function calculates the specific enthalpy for R134a in-
% J/kg at a given temperature in K and density in kg/m^3 in --
```

```
% the overheated gaseous phase}.-----------------------------

aeins = -7.7057291;
azwei = 2.4186313;
adrei = -2.1848312;
avier = -3.4530733;
beins = 0.2201286;
bzwei = -1.267402;
bdrei = -0.3084991;
bvier = -3.546135E-04;
ceins = 0.01387313;
czwei = 0.3353155;
deins = -0.06356633;
dzwei = 0.08120353;
ddrei = -0.03699965;
anullstern = 1.019535;
aeinsstern = 9.047135;
meins = -0.629789;
mzwei = 7.292937;
mdrei = 5.154411;
r =   81.488856;
tc =  374.18;
rhoc = 508;

tau = tc./T;
delta = rho ./ rhoc;
h_T_rho =r*T.*(aeinsstern*tau + meins + 2/3*mzwei...
  * tau.^(-1/2) + 4/7*mdrei*tau.^(-3/4) + delta.*...
  (1/2*beins*tau.^(-1/2) + 11/4*bzwei*tau.^(7/4)...
  + 5*bdrei*tau.^4 + 13*bvier*tau.^12)...
  + delta.^2.*(7/4*ceins*tau.^(-1/4) + 9/2...
  * czwei*tau.^(5/2)) + delta.^3.*(13/2*deins...
  * tau.^(7/2) + 15*dzwei*tau.^12 + 23*ddrei*tau.^20));
```

C.2.5 p(t,ρ)

This function calculates the pressure for R134a in Pa at a given temperature in K and density in kg/m^3 in the *overheated gaseous phase*.
Function call:

p_t_rho(temperature in K, density in kg/m^3), e. g.

```
>>p_t_rho(333,43.3)
ans =

9.9829e+005
```

Sourcecode:

```
function p_T_rho = p_T_rho(T,rho);
%-------------------------------------------------------------
% ------------------------ p_t_rho.m ------------------------
% -------------------------------------------------------------
% This function calculates the pressure for R134a in Pa at a--
% given temperature in K and density in kg/m^3 in the---------
% overheated gaseous phase}.---------------------------------

beins = 0.2201286;
bzwei = -1.267402;
bdrei = -0.3084991;
bvier = -3.55E-04;
ceins = 0.01387313;
czwei = 0.3353155;
deins = -0.06356633;
dzwei = 0.08120353;
ddrei = -0.03699965;
anullstern = 1.019535;
aeinsstern = 9.047135;
meins = -0.629789;
mzwei = 7.292937;
mdrei = 5.154411;
r =   81.488856;
tc =  374.18;
rhoc = 508;

tau = tc./T;
delta = rho./rhoc;

  p_T_rho = (1+delta.*(beins*tau.^(-1/2)+bzwei*tau.^(7/4)...
    + bdrei*tau.^4+bvier*tau.^12) + 2*delta.^2.*...
    (ceins*tau.^(-1/4)+czwei*tau.^(5/2)) + 3*delta.^3.*...
    (deins*tau.^(7/2) + dzwei*tau.^12+ddrei*...
    tau.^20)).*rho*r.*T;
```

C.2.6 p_S(T)

This function calculates the pressure for R134a in Pa at a given temperature in K for the saturated vapour.
Function call:

ps_t(temperature in K), e. g.

```
>>ps_t(333)
ans =
9.9829e+005
```

Sourcecode:

```
% ------------------------------------------------------------
% ----------------------- ps_T.m -----------------------------
% ------------------------------------------------------------
% This function calculates the saturation pressure for R134a -
% in Pa at a given temperature in K --------------------------

aeins = -7.7057291;
azwei = 2.4186313;
adrei = -2.1848312;
avier = -3.4530733;

r =   81.488856;
tc =  374.18;
rhoc = 508;

pnull=4.056318*10^6; theta=1-(T./tc);
ps_T=exp(1./(1-theta).*(aeins.*theta + azwei.*theta.^1.5 +...
  adrei.*theta.^2 + avier.*theta.^4))*pnull;
```

C.2.7 Evaporator temperature

This function calculates the air temperature behind the evaporator in °C for the compressor *Zexel®DKS-16H*.
Function call:

> **Verdampferaustrittstemperatur**(air mass flow rate in kg/s, outside air temperature in °C, rel. humidity in %, compressor revolutions in min^{-1}), e.g.
>
> ```
> >>Verdampferaustrittstemperatur(0.2,32,70,3000)
> ans =
>
> 7.5285
> ```

Sourcecode:

```
function Verdampferaustrittstemperatur = ...
      Verdampferaustrittstemperatur((mL,...
      Aussentemperatur,phi,rev)
%------------------------------------------------------------
% ------------- Verdampferaustrittstemp.m -------------------
% ------------------------------------------------------------
% calculates the air temperature behind the evaporator -------
% for given outside air temperatures (Aussentemperatur in° C),
% mass rate (mL in kg/s), rel. humidity (phi in %) and comp.--
% revolutions (rev in min^-1).--------------------------------
% ----------------- Martin Konz 03/2001 ----------------------

P=[-0.00000044183109 0.00400435535565 -0.00929561290026];
```

```
% ------------------------------------------------------------
% Compressor power ZEXEL DKS-16H ---------------------
% ------------------------------------------------------------

Q_max=polyval(P,rev)*1000
Aussentemperatur=Aussentemperatur+273.15;
k=70;
A=2.15;
cpL=1000;
mR=0.08;
T_R=276;
cpR=1300;
CR=mR*cpR;
CL=mL.*cpL;
NTU_calc=ones(size(mL));
NTU_calc=k*A./CL;
  %Q_max=mL.*cpL*(Aussentemperatur-T_R)
epsilon=1-exp(-NTU_calc);
Q_tats=epsilon.*Q_max
H_trocken=Q_tats/mL
h_ein=h_t_phi(Aussentemperatur-273.15,phi)*1000;
x_ein=x_t_phi(Aussentemperatur-273.15,phi);
t=-25:0.5:50;
xs=xs_t_phi(t,100);
t_satt=interp1(xs,t,x_ein,'cubic');
h_satt=h_t_phi(t_satt,100)*1000;
if H_trocken <= (h_ein-h_satt)
  Verdampferaustrittstemperatur=Aussentemperatur-Q_tats./...
    (mL.*cpL)-273.15;
  if Verdampferaustrittstemperatur<6
    Verdampferaustrittstemperatur=6;
  end;
else
  h_aus=h_ein-H_trocken;
  if h_aus<0
    h_aus=0;
  end;
  t=-25:1:40;
  h_2=h_t_phi(t,100).*1000;
  t_aus=interp1(h_2,t,h_aus,'cubic');
  Verdampferaustrittstemperatur=t_aus;
  if Verdampferaustrittstemperatur<6
    Verdampferaustrittstemperatur=6;
  end;
end;
```

C.3 R-134a properties (C MEX-Functions)

The following functions are written in C/C++ and are called *C MEX-Functions*. Once compiled with the MATLAB contributed C-compiler they can be used directly in MATLAB like an ordinary M-function.

The use of this function-type consumes much less calculation time than the MATLAB M-functions do. This is very important if one needs to know the value for properties that can't be detected directly. Iterations have to be used to approximate the value.

The structure of the source code is identical for all functions. First, there is the *computational routine*, which is the calculation algorithm for the problem. Second, there has to be a *gateway routine*, which represents the interface between the MATLAB environment and the calculation routine.

C.3.1 ρ(T,p)

This function calculates the specific density of R134a in kg/m^3 at a given Temperature in K and pressure in Pa in the *overheated gaseous phase*.

Function call:

rho_T_p(Temperature in K, pressure in °C, e.g.

```
>>rho_T_p(353,10e5)
ans =

39.4039
```

Sourcecode:

```
/*
-----------------------------------------------------------------
--------------------rho_T_p.c, rho_T_p.dll-----------------
-----------------------------------------------------------------
*/
#include <math.h>
#include "mex.h"
/*
-----------------------------------------------------------------
------------------COMPUTATIONAL ROUTINE--------------------
-----------------------------------------------------------------
*/
void rho_T_p(double *y, double *T, double *p)
{
  double beins =   0.2201286;
  double bzwei =   -1.267402;
  double bdrei =   -0.3084991;
  double bvier =   -3.55E-04;
  double ceins =   0.01387313;
```

```
20  double czwei =   0.3353155;
    double deins =   -0.06356633;
    double dzwei =   0.08120353;
    double ddrei =   -0.03699965;
    double anullstern = 1.019535;
25  double aeinsstern = 9.047135;
    double meins =   -0.629789;
    double mzwei =   7.292937;
    double mdrei =   5.154411;
    double r = 81.488856;
30  double tc = 374.18;
    double rhoc = 508;
    int i;
    double rho, rhoeins, rhozwei, rhodrei;
    double fvonrho_eins, fvonrho_zwei, fvonrho_drei;
35  double deltaeins, deltazwei, deltadrei;
    double tau;

        tau = tc/ *T;
        rho = *p/ (r* *T);
40      rhoeins = rho* 0.2;
        rhozwei = rho* 3;
        deltaeins = rhoeins/ rhoc;
        deltazwei = rhozwei/ rhoc;

45   for (i=1;i<30;i++)
      {
      rhodrei = rhozwei+ 0.5* (rhoeins- rhozwei);
      deltadrei = rhodrei/ rhoc;

50    fvonrho_eins = (1+ deltaeins* (beins* pow(tau, (-0.5))
        + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
        + bvier* pow(tau, 12))+ 2* pow(deltaeins, 2)
        * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
        + 3* pow(deltaeins, 3) * (deins* pow(tau, (3.5))
55      + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
        * rhoeins* r* *T- *p;

      fvonrho_zwei = (1+ deltazwei* (beins* pow(tau, (-0.5))
        + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
60      + bvier* pow(tau, 12))+ 2* pow(deltazwei, 2)
        * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
        + 3* pow(deltazwei, 3) * (deins* pow(tau, (3.5))
        + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
        * rhozwei* r* *T- *p;
65
      fvonrho_drei = (1+ deltadrei* (beins* pow(tau, (-0.5))
```

```
      + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
      + bvier* pow(tau, 12))+ 2* pow(deltadrei, 2)
      * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
      + 3* pow(deltadrei, 3)* (deins* pow(tau, (3.5))
      + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
      * rhodrei* r* *T- *p;


    if (fvonrho_zwei* fvonrho_drei >= 0)
      {
      rhozwei = rhodrei;
      deltazwei = rhozwei/ rhoc;
      }
    else
      {
      rhoeins = rhozwei;
      deltaeins = rhoeins/ rhoc;
      rhozwei = rhodrei;
      deltazwei = rhozwei/ rhoc;
      }
  }
  *y = rhozwei;
}
/*
-----------------------------------------------------------------
----------------------GATEWAY ROUTINE--------------------------
-----------------------------------------------------------------
*/
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
  double *y, *T, *p;
  int mrows, ncols, orows, pcols;

/* checks if there is only one right handed argument */
  if (nrhs =! 2 )
  {
    mexErrMsgTxt("Two␣input␣arguments␣required.");
  }
  else if (nlhs>2)
  {
   mexErrMsgTxt("Too␣many␣output␣arguments");
  }
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  orows = mxGetM(prhs[1]);
  pcols = mxGetN(prhs[1]);
```

```
/* checks if input arguments are noncomplex scalar double */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
     (!(mrows==1 && ncols==1)) ||
    (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
     (!(orows==1 && pcols==1))))
  {
    mexErrMsgTxt("Input must be a noncomplex scalar double");
     }

/* Set Pointer to right handed arguments */
  T = mxGetPr(prhs[0]);
  p = mxGetPr(prhs[1]);


/* Set pointer to created output Matrix */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* set pointer to first left handed argument */
  y = mxGetPr(plhs[0]);

/* call computational routine */
  rho_T_p(y,T,p);
}
```

C.3.2 ρ(T,s)

This function calculates the specific density of R134a in kg/m^3 at a given Temperature in K and specific entropy in J/(kg K) in the *overheated gaseous phase*.

Function call:

rho_T_s(Temperature in `K`, entropy in `J/(kg K)`, e.g.

```
>>rho_T_s(353,1800)
ans =

58.8809
```

Sourcecode:

```
double ceins =   0.01387313;
double czwei =   0.3353155;
double deins =   -0.06356633;
double dzwei =   0.08120353;
double ddrei =   -0.03699965;
double anullstern = 1.019535;
```

```
    double aeinsstern = 9.047135;
    double meins =   -0.629789;
    double mzwei =   7.292937;
    double mdrei =   5.154411;
    double r = 81.488856;
    double tc = 374.18;
    double rhoc = 508;
    double tau, delta;
    double deltaeins, deltazwei, deltadrei;
    double fvonrho_eins, fvonrho_zwei, fvonrho_drei;
    double rho, rhoeins, rhozwei, rhodrei;
    int i;



    tau = tc/ *T;
    rho = 30;
    rhoeins = rho* 0.02;
    rhozwei = rho* 6;

    deltaeins = rhoeins/ rhoc;
    deltazwei = rhozwei/ rhoc;

    for (i=1;i<30;i++)
      {
       rhodrei = rhozwei+ 0.5* (rhoeins- rhozwei);
       deltadrei = rhodrei/ rhoc;

       fvonrho_eins = r* (anullstern+ (meins- 1)
         * (1- log(tau))- log(deltaeins)+ 2
         * mzwei* pow(tau,-0.5)+ 4.0/ 3.0 * mdrei
         * pow(tau, -0.75) + deltaeins* (-1.5* beins
         * pow(tau, -0.5)+ 0.75* bzwei* pow(tau, 1.75)+3
         * bdrei* pow(tau, 4)+ 11* bvier* pow(tau, 12))
         + pow(deltaeins, 2)* (-1.25* ceins* pow(tau, -0.25)
         + 1.5* ceins* pow(tau, 2.5)) + pow(deltaeins, 3)
         * (2.5* deins* pow(tau, 6.5)+ 11* dzwei* pow(tau, 12)
         + 19* ddrei* pow(tau, 20)))- *s;

       fvonrho_zwei = r* (anullstern+ (meins- 1)
         * (1- log(tau))- log(deltazwei)+ 2* mzwei
         * pow(tau,-0.5)+ 4.0/ 3.0* mdrei* pow(tau, -0.75)
         + deltazwei* (-1.5* beins* pow(tau, -0.5)+ 0.75
         * bzwei* pow(tau, 1.75)+3 * bdrei* pow(tau, 4)+ 11
         * bvier* pow(tau, 12)) + pow(deltazwei, 2)* (-1.25
         * ceins* pow(tau, -0.25)+ 1.5* ceins* pow(tau, 2.5))
         + pow(deltazwei, 3)* (2.5* deins* pow(tau, 6.5)+ 11
```

```
      * dzwei* pow(tau, 12)+ 19* ddrei* pow(tau, 20)))- *s;

    fvonrho_drei = r* (anullstern+ (meins- 1)
      * (1- log(tau))- log(deltadrei)+ 2* mzwei
      * pow(tau,-0.5)+ 4.0/ 3.0* mdrei* pow(tau, -0.75)
      + deltadrei* (-1.5* beins* pow(tau, -0.5)+ 0.75
      * bzwei* pow(tau, 1.75)+3 * bdrei* pow(tau, 4)+ 11
      * bvier* pow(tau, 12)) + pow(deltadrei, 2)
      * (-1.25* ceins* pow(tau, -0.25)+ 1.5* ceins
      * pow(tau, 2.5)) + pow(deltadrei, 3)* (2.5* deins
      * pow(tau, 6.5)+ 11* dzwei* pow(tau, 12)+ 19
      * ddrei* pow(tau, 20)))- *s;

    if (fvonrho_zwei* fvonrho_drei >= 0)
      {
        rhozwei = rhodrei;
        deltazwei = rhozwei/ rhoc;
      }
    else
      {
        rhoeins = rhozwei;
        deltaeins = rhoeins/ rhoc;
        rhozwei = rhodrei;
        deltazwei = rhozwei/ rhoc;
      }
   }
  *y = rhozwei;
}
/*
------------------------------------------------------------
----------------------GATEWAY ROUTINE-----------------------
------------------------------------------------------------
*/
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
  double *y, *T, *s;
  int mrows, ncols, orows, pcols;

/* checks if there is only one right handed argument */
  if (nrhs =! 2 )
  {
    mexErrMsgTxt("Two␣input␣arguments␣required.");
  }
  else if (nlhs>2)
  {
   mexErrMsgTxt("Too␣many␣output␣arguments");
```

```
  }
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  orows = mxGetM(prhs[1]);
  pcols = mxGetN(prhs[1]);

/* checks if input arguments are noncomplex scalar double */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
     (!(mrows==1 && ncols==1)) ||
    (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
     (!(orows==1 && pcols==1))))
  {
    mexErrMsgTxt("Input must be a noncomplex scalar double");
     }

/* Set Pointer to right handed arguments */
  T = mxGetPr(prhs[0]);
  s = mxGetPr(prhs[1]);


/* Set pointer to created output Matrix */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* set pointer to first left handed argument */
  y = mxGetPr(plhs[0]);

/* call computational routine */
  rho_T_s(y,T,s);
}
```

C.3.3 s(T,ρ)

This function calculates the specific entropy of R134a in J/(kg K)^3 at a given Temperature in K and density in the *overheated gaseous phase*.

Function call:

s_T_rho(Temperature in `K`, density in `kg/m`^{3}, e.g.

```
>>s_T_rho(353,58.9)
ans =

1.8000e3
```

Sourcecode:

```
/*
-----------------------------------------------------------------
```

```
-------------- s_T_rho.c bzw. s_T_rho.dll------------------
------------------------------------------------------------
*/
#include <math.h>
#include "mex.h"
/*
------------------------------------------------------------
------------------COMPUTATIONAL ROUTINE---------------------
------------------------------------------------------------
*/
void s_T_rho(double *y, double *T, double *rho)
{

  double aeins = -7.7057291;
  double azwei = 2.4186313;
  double adrei = -2.1848312;
  double avier = -3.4530733;


  double beins =   0.2201286;
  double bzwei =   -1.267402;
  double bdrei =   -0.3084991;
  double bvier =   -3.546135E-04;
  double ceins =   0.01387313;
  double czwei =   0.3353155;
  double deins =   -0.06356633;
  double dzwei =   0.08120353;
  double ddrei =   -0.03699965;
  double anullstern = 1.019535;
  double aeinsstern = 9.047135;
  double meins =   -0.629789;
  double mzwei =   7.292937;
  double mdrei =   5.154411;
  double r = 81.488856;
  double tc = 374.18;
  double rhoc = 508;
  double tau, delta;

  tau = tc/ *T;
  delta = *rho/ rhoc;

  *y = r* (anullstern+ (meins- 1)* (1- log(tau))- log(delta)
    + 2* mzwei* pow(tau,-0.5)+ 4.0/ 3.0* mdrei
    * pow(tau, -0.75) + delta* (-1.5* beins* pow(tau, -0.5)
    + 0.75* bzwei* pow(tau, 1.75)+3 * bdrei* pow(tau, 4)+ 11
    * bvier* pow(tau, 12)) + pow(delta, 2)* (-1.25* ceins
    * pow(tau, -0.25)+ 1.5* ceins* pow(tau, 2.5))
```

```
    + pow(delta, 3)* (2.5* deins* pow(tau, 6.5)+ 11* dzwei
    * pow(tau, 12)+ 19* ddrei* pow(tau, 20)));

}
/*
-----------------------------------------------------------------
----------------------GATEWAY ROUTINE-------------------------
-----------------------------------------------------------------
*/
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
  double *y, *T, *rho;
  int mrows, ncols, orows, pcols;

/* checks if there is only one right handed argument */
  if (nrhs =! 2 )
  {
    mexErrMsgTxt("Two input arguments required.");
  }
  else if (nlhs>2)
  {
   mexErrMsgTxt("Too many output arguments");
  }
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  orows = mxGetM(prhs[1]);
  pcols = mxGetN(prhs[1]);

/* checks if input arguments are noncomplex scalar double */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
     (!(mrows==1 && ncols==1)) ||
    (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
     (!(orows==1 && pcols==1))))
  {
    mexErrMsgTxt("Input must be a noncomplex scalar double");
     }

/* Set Pointer to right handed arguments */
  T = mxGetPr(prhs[0]);
  rho = mxGetPr(prhs[1]);


/* Set pointer to created output Matrix */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* set pointer to first left handed argument */
```

```
  y = mxGetPr(plhs[0]);

/* call computational routine */
  s_T_rho(y,T,rho);
}
```

C.3.4 T(h,p)

This function calculates the Temperature of R134a in K at a given specific enthalpy in J/kg and pressure in Pa in the *overheated gaseous phase*.

Function call:

T_h_p(specific enthalpy in `J/kg`, pressure in `Pa`, e.g.

```
>>T_h_p(450e3,10e5)
ans =

341.0338
```

Sourcecode:

```
------------------------------------------------------------------
--------------------- T_h_p.c, T_h_p.dll --------------------------
------------------------------------------------------------------
*/
#include <math.h>
#include "mex.h"
/*
------------------------------------------------------------------
------------------COMPUTATIONAL ROUTINE---------------------------
------------------------------------------------------------------
*/
void h_T_rho(double *y, double *h, double *p)
{

  double aeins = -7.7057291;
  double azwei = 2.4186313;
  double adrei = -2.1848312;
  double avier = -3.4530733;

  double beins =  0.2201286;
  double bzwei =  -1.267402;
  double bdrei =  -0.3084991;
  double bvier =  -3.546135E-04;
  double ceins =  0.01387313;
  double czwei =  0.3353155;
  double deins =  -0.06356633;
```

```
  double dzwei =   0.08120353;
  double ddrei =   -0.03699965;
  double anullstern = 1.019535;
  double aeinsstern = 9.047135;
  double meins =   -0.629789;
  double mzwei =   7.292937;
  double mdrei =   5.154411;
  double r = 81.488856;
  double tc = 374.18;
  double rhoc = 508;
  double pnull = 4.056318e6;
  double Teins, Tzwei, Tdrei;

  int i, j;
  double arho, arhoeins, arhozwei, arhodrei;
  double rhoeins, rhozwei, rhodrei;
  double fvonrho_eins, fvonrho_zwei, fvonrho_drei;
  double deltaeins, deltazwei, deltadrei;
  double Tdeltaeins, Tdeltazwei, Tdeltadrei;
  double tau, taueins, tauzwei, taudrei;
  double fvonTeins, fvonTzwei, fvonTdrei;
  double theta, thetaeins, thetazwei, thetadrei;
  double fvontheta_eins, fvontheta_zwei, fvontheta_drei;
  double Ts;

  Teins = 273;
  Tzwei = 403;


  theta = 1-(300/tc);
  thetaeins = 1-(240/tc);
  thetazwei = 0;

  for (j=1;j<30;j++)
   {
    thetadrei = thetazwei + 0.5* (thetaeins- thetazwei);
    fvontheta_eins = (exp(1.0/ (1- thetaeins)
      * (aeins* thetaeins+ azwei* pow(thetaeins, 1.5)
      + adrei* pow(thetaeins, 2)+ avier* pow(thetaeins, 4)))
      * pnull)- *p;

    fvontheta_zwei = (exp(1.0/ (1- thetazwei)* (aeins
      * thetazwei+ azwei* pow(thetazwei, 1.5) + adrei
      * pow(thetazwei, 2)+ avier* pow(thetazwei, 4)))
      * pnull)- *p;

   fvontheta_drei = (exp(1.0/ (1- thetadrei)* (aeins
```

```
      * thetadrei+ azwei* pow(thetadrei, 1.5) + adrei
      * pow(thetadrei, 2)+ avier* pow(thetadrei, 4)))
      * pnull)- *p;

    if (fvontheta_zwei* fvontheta_drei >= 0)
      {
       thetazwei = thetadrei;
      }
    else
      {
       thetaeins = thetazwei;
       thetazwei = thetadrei;
      }
   }
  Ts = (1 - thetazwei)*tc;
  if (Teins < Ts)
   {
    Teins = Ts;
   }

  //------------------------------------------------
  //----------- rhoeins=rho_T_p(Teins,p); ----------
  //------------------------------------------------

  tau = tc/ Teins;
  arho = *p/ (r* Teins);
  arhoeins = arho* 0.2;
  arhozwei = arho* 3;
  deltaeins = arhoeins/ rhoc;
  deltazwei = arhozwei/ rhoc;

  while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
   {
    arhodrei = arhozwei+ 0.5* (arhoeins- arhozwei);
    deltadrei = arhodrei/ rhoc;

    fvonrho_eins = (1+ deltaeins* (beins* pow(tau,
     (-0.5))+ bzwei* pow(tau, (1.75))
     + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))+ 2
     * pow(deltaeins, 2)* (ceins* pow(tau, (-0.25))
     + czwei* pow(tau,(2.5)))+ 3* pow(deltaeins, 3)
     * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
     + ddrei* pow(tau, 20))) * arhoeins* r* Teins- *p;

   fvonrho_zwei = (1+ deltazwei* (beins* pow(tau,
     (-0.5))+ bzwei* pow(tau, (1.75))
     + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))+ 2
```

```
      * pow(deltazwei, 2) * (ceins* pow(tau, (-0.25))
      + czwei* pow(tau,(2.5)))+ 3* pow(deltazwei, 3)
      * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
      + ddrei* pow(tau, 20))) * arhozwei* r* Teins- *p;

   fvonrho_drei = (1+ deltadrei* (beins* pow(tau,
     (-0.5))+ bzwei* pow(tau, (1.75))
     + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))+ 2
     * pow(deltadrei, 2)* (ceins* pow(tau, (-0.25))
     + czwei* pow(tau,(2.5)))+ 3* pow(deltadrei, 3)
     * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
     + ddrei* pow(tau, 20))) * arhodrei* r* Teins- *p;


   if (fvonrho_zwei* fvonrho_drei >= 0)
     {
      arhozwei = arhodrei;
      deltazwei = arhozwei/ rhoc;
     }
   else
     {
      arhoeins = arhozwei;
      deltaeins = arhoeins/ rhoc;
      arhozwei = arhodrei;
      deltazwei = arhozwei/ rhoc;
     }
  }
rhoeins = arhozwei;

//-------------------------------------------------
//----------- rhozwei=rho_T_p(Tzwei,p); ----------
//-------------------------------------------------
tau = tc/ Tzwei;
arho = *p/ (r* Tzwei);
arhoeins = arho* 0.2;
arhozwei = arho* 3;
deltaeins = arhoeins/ rhoc;
deltazwei = arhozwei/ rhoc;

while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
 {
  arhodrei = arhozwei+ 0.5* (arhoeins- arhozwei);
  deltadrei = arhodrei/ rhoc;

  fvonrho_eins = (1+ deltaeins* (beins* pow(tau, (-0.5))
    + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
    + bvier* pow(tau, 12))+ 2* pow(deltaeins, 2)
```

```
      * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
      + 3* pow(deltaeins, 3) * (deins* pow(tau, (3.5))
      + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
      * arhoeins* r* Tzwei- *p;

    fvonrho_zwei = (1+ deltazwei* (beins* pow(tau, (-0.5))
      + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
      + bvier* pow(tau, 12))+ 2* pow(deltazwei, 2)
      * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
      + 3* pow(deltazwei, 3) * (deins* pow(tau, (3.5))
      + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
      * arhozwei* r* Tzwei- *p;

    fvonrho_drei = (1+ deltadrei* (beins* pow(tau, (-0.5))
      + bzwei* pow(tau, (1.75)) + bdrei* pow(tau, 4)
      + bvier* pow(tau, 12))+ 2* pow(deltadrei, 2)
      * (ceins* pow(tau, (-0.25))+ czwei* pow(tau,(2.5)))
      + 3* pow(deltadrei, 3) * (deins* pow(tau, (3.5))
      + dzwei* pow(tau, 12)+ ddrei* pow(tau, 20)))
      * arhodrei* r* Tzwei- *p;

    if (fvonrho_zwei* fvonrho_drei >= 0)
      {
       arhozwei = arhodrei;
       deltazwei = arhozwei/ rhoc;
      }
    else
      {
       arhoeins = arhozwei;
       deltaeins = arhoeins/ rhoc;
       arhozwei = arhodrei;
       deltazwei = arhozwei/ rhoc;
      }
   }
  rhozwei = arhozwei;

  //-----------------------------------------------------------
  Tdeltaeins = rhoeins/ rhoc;
  Tdeltazwei = rhozwei/ rhoc;
  taueins = tc/ Teins;
  tauzwei = tc/ Tzwei;
  while (abs(Teins-Tzwei) > 0.00001*Tzwei)
   for (i=1;i<=20;i++)
    {
     Tdrei = Tzwei + 0.5*(Teins-Tzwei);
     //-------------------------------------------------
     //----------- rhodrei=rho_T_p(Tdrei,p); ----------
```

```
215       //-------------------------------------------------
          tau = tc/ Tdrei;
          arho = *p/ (r* Tdrei);
          arhoeins = arho* 0.2;
          arhozwei = arho* 3;
220       deltaeins = arhoeins/ rhoc;
          deltazwei = arhozwei/ rhoc;

          while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
            //     for (j=1;j<30;j++)
225         {
             arhodrei = arhozwei+ 0.5* (arhoeins- arhozwei);
             deltadrei = arhodrei/ rhoc;

             fvonrho_eins = (1+ deltaeins* (beins* pow(tau,
230            (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
               * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
               * pow(deltaeins, 2) * (ceins* pow(tau, (-0.25))
               + czwei* pow(tau,(2.5)))+ 3* pow(deltaeins, 3)
               * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
235            + ddrei* pow(tau, 20)))
               * arhoeins* r* Tdrei- *p;

             fvonrho_zwei = (1+ deltazwei* (beins* pow(tau,
               (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
240            * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
               * pow(deltazwei, 2) * (ceins* pow(tau, (-0.25))
               + czwei* pow(tau,(2.5)))+ 3* pow(deltazwei, 3)
               * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
               + ddrei* pow(tau, 20)))
245            * arhozwei* r* Tdrei- *p;

             fvonrho_drei = (1+ deltadrei* (beins* pow(tau,
               (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
               * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
250            * pow(deltadrei, 2) * (ceins* pow(tau, (-0.25))
               + czwei* pow(tau,(2.5)))+ 3* pow(deltadrei, 3)
               * (deins* pow(tau, (3.5))+ dzwei* pow(tau, 12)
               + ddrei* pow(tau, 20)))
            * arhodrei* r* Tdrei- *p;
255
             if (fvonrho_zwei* fvonrho_drei >= 0)
               {
                arhozwei = arhodrei;
                deltazwei = arhozwei/ rhoc;
260            }
             else
```

```
        {
          arhoeins = arhozwei;
          deltaeins = arhoeins/ rhoc;
          arhozwei = arhodrei;
          deltazwei = arhozwei/ rhoc;
        }
      }
    rhodrei = arhozwei;

    Tdeltadrei = rhodrei/rhoc;
    taudrei = tc/ Tdrei;

    fvonTeins = r* Teins* (aeinsstern* taueins+ meins
      + 2.0/ 3.0* mzwei* pow(taueins,-0.5)+ 4.0/ 7.0
      * mdrei* pow(taueins,-0.75) + Tdeltaeins* (0.5
      * beins* pow(taueins, -0.5)+ 11.0/ 4.0* bzwei
      * pow(taueins, 1.75)+ 5* bdrei* pow(taueins, 4)
      + 13* bvier* pow(taueins, 12)) + pow(Tdeltaeins, 2)
      * (7.0/ 4.0* ceins* pow(taueins, -0.25)+ 9.0/ 2.0
      * czwei* pow(taueins, 2.5)) + pow(Tdeltaeins, 3)
      * (6.5* deins* pow(taueins, 3.5)+ 15* dzwei
      * pow(taueins, 12)+ 23* ddrei
      * pow(taueins, 20))) - *h;

    fvonTzwei = r* Tzwei* (aeinsstern* tauzwei+ meins
      + 2.0/ 3.0* mzwei* pow(tauzwei,-0.5)+ 4.0/ 7.0
      * mdrei* pow(tauzwei,-0.75) + Tdeltazwei* (0.5
      * beins* pow(tauzwei, -0.5)+ 11.0/ 4.0* bzwei
      * pow(tauzwei, 1.75)+ 5* bdrei* pow(tauzwei, 4)
      + 13* bvier* pow(tauzwei, 12)) + pow(Tdeltazwei, 2)
      * (7.0/ 4.0* ceins* pow(tauzwei, -0.25)+ 9.0/ 2.0
      * czwei* pow(tauzwei, 2.5)) + pow(Tdeltazwei, 3)
      * (6.5* deins* pow(tauzwei, 3.5)+ 15* dzwei
      * pow(tauzwei, 12)+ 23* ddrei
      * pow(tauzwei, 20))) - *h;

    fvonTdrei = r* Tdrei* (aeinsstern* taudrei+ meins
      + 2.0/ 3.0* mzwei* pow(taudrei,-0.5)+ 4.0/ 7.0
      * mdrei* pow(taudrei,-0.75) + Tdeltadrei* (0.5
      * beins* pow(taudrei, -0.5)+ 11.0/ 4.0* bzwei
      * pow(taudrei, 1.75)+ 5* bdrei* pow(taudrei, 4)
      + 13* bvier* pow(taudrei, 12)) + pow(Tdeltadrei, 2)
      * (7.0/ 4.0* ceins* pow(taudrei, -0.25)+ 9.0/ 2.0
      * czwei* pow(taudrei, 2.5)) + pow(Tdeltadrei, 3)
      * (6.5* deins* pow(taudrei, 3.5)+ 15* dzwei
      * pow(taudrei, 12)+ 23* ddrei
      * pow(taudrei, 20))) - *h;
```

```
310        if (fvonTzwei* fvonTdrei >= 0)
             {
               Tzwei = Tdrei;
               //-------------------------------------------------
               //----------- rhozwei=rho_T_p(Tzwei,p); ----------
315            //-------------------------------------------------
               tau = tc/ Tzwei;
               arho = *p/ (r* Tzwei);
               arhoeins = arho* 0.2;
               arhozwei = arho* 3;
320            deltaeins = arhoeins/ rhoc;
               deltazwei = arhozwei/ rhoc;

               while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
                 {
325                arhodrei = arhozwei+ 0.5
                    * (arhoeins- arhozwei);
                   deltadrei = arhodrei/ rhoc;

                   fvonrho_eins = (1+ deltaeins* (beins* pow(tau,
330                  (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
                     * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
                     * pow(deltaeins, 2) * (ceins* pow(tau,
                     (-0.25))+ czwei* pow(tau,(2.5)))+ 3
                     * pow(deltaeins, 3) * (deins* pow(tau,
335                  (3.5))+ dzwei* pow(tau, 12)+ ddrei
                     * pow(tau, 20))) * arhoeins* r* Tzwei- *p;

                   fvonrho_zwei = (1+ deltazwei* (beins* pow(tau,
                     (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
340                  * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
                     * pow(deltazwei, 2) * (ceins* pow(tau,
                     (-0.25))+ czwei* pow(tau,(2.5)))+ 3
                     * pow(deltazwei, 3) * (deins* pow(tau,
                     (3.5))+ dzwei* pow(tau, 12)+ ddrei
345                  * pow(tau, 20))) * arhozwei* r* Tzwei- *p;

                   fvonrho_drei = (1+ deltadrei* (beins* pow(tau,
                     (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
                     * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
350                  * pow(deltadrei, 2) * (ceins* pow(tau,
                     (-0.25))+ czwei* pow(tau,(2.5)))+ 3
                     * pow(deltadrei, 3) * (deins* pow(tau,
                    (3.5))+ dzwei* pow(tau, 12)+ ddrei
                    * pow(tau, 20))) * arhodrei* r* Tzwei- *p;
355
```

```
            if (fvonrho_zwei* fvonrho_drei >= 0)
              {
                arhozwei = arhodrei;
360             deltazwei = arhozwei/ rhoc;
              }
            else
            {
              arhoeins = arhozwei;
365           deltaeins = arhoeins/ rhoc;
              arhozwei = arhodrei;
              deltazwei = arhozwei/ rhoc;
            }
           }
370        rhozwei = arhozwei;
           Tdeltazwei = rhozwei/ rhoc;
           tauzwei=tc/ Tzwei;
         }
       else
375     {
           Teins = Tzwei;
           //-------------------------------------------------
           //----------- rhoeins=rho_T_p(Teins,p); ----------
           //-------------------------------------------------
380        tau = tc/ Teins;
           arho = *p/ (r* Teins);
           arhoeins = arho* 0.2;
           arhozwei = arho* 3;
           deltaeins = arhoeins/ rhoc;
385        deltazwei = arhozwei/ rhoc;

           while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
             {
               arhodrei = arhozwei+ 0.5* (arhoeins- arhozwei);
390            deltadrei = arhodrei/ rhoc;

               fvonrho_eins = (1+ deltaeins* (beins* pow(tau,
                 (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
                 * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
395              * pow(deltaeins, 2) * (ceins* pow(tau,
                 (-0.25))+ czwei* pow(tau,(2.5)))+ 3
                 * pow(deltaeins, 3) * (deins* pow(tau,
                 (3.5))+ dzwei* pow(tau, 12)+ ddrei
                 * pow(tau, 20))) * arhoeins* r* Teins- *p;
400
               fvonrho_zwei = (1+ deltazwei* (beins* pow(tau,
                 (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
```

```
                * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
                * pow(deltazwei, 2) * (ceins* pow(tau,
405             (-0.25))+ czwei* pow(tau,(2.5)))+ 3
                * pow(deltazwei, 3) * (deins* pow(tau,
                (3.5))+ dzwei* pow(tau, 12)+ ddrei
                * pow(tau, 20))) * arhozwei* r* Teins- *p;

410           fvonrho_drei = (1+ deltadrei* (beins* pow(tau,
                (-0.5))+ bzwei* pow(tau, (1.75)) + bdrei
                * pow(tau, 4)+ bvier* pow(tau, 12))+ 2
                * pow(deltadrei, 2) * (ceins* pow(tau,
                (-0.25))+ czwei* pow(tau,(2.5)))+ 3
415             * pow(deltadrei, 3) * (deins* pow(tau,
                (3.5))+ dzwei* pow(tau, 12)+ ddrei
                * pow(tau, 20))) * arhodrei* r* Teins- *p;

              if (fvonrho_zwei* fvonrho_drei >= 0)
420             {
                  arhozwei = arhodrei;
                  deltazwei = arhozwei/ rhoc;
                }
              else
425             {
                  arhoeins = arhozwei;
                  deltaeins = arhoeins/ rhoc;
                  arhozwei = arhodrei;
                  deltazwei = arhozwei/ rhoc;
430             }
            }
          rhoeins = arhozwei;
          //-----------------------------------------
          deltaeins = rhoeins/ rhoc;
435       taueins=tc/ Teins;
          Tzwei = Tdrei;
          //-----------------------------------------
          //------- rhozwei=rho_T_p(Tzwei,p); ----------
          //-----------------------------------------
440       tau = tc/ Tzwei;
          arho = *p/ (r* Tzwei);
          arhoeins = arho* 0.2;
          arhozwei = arho* 3;
          deltaeins = arhoeins/ rhoc;
445       deltazwei = arhozwei/ rhoc;

          while (abs(arhoeins-arhozwei) > 0.00001*arhoeins)
            {
              arhodrei = arhozwei+ 0.5* (arhoeins- arhozwei);
```

```
            deltadrei = arhodrei/ rhoc;

            fvonrho_eins = (1+ deltaeins* (beins
              * pow(tau, (-0.5))+ bzwei* pow(tau, (1.75))
              + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))
              + 2* pow(deltaeins, 2) * (ceins* pow(tau,
              (-0.25))+ czwei* pow(tau,(2.5)))+ 3
              * pow(deltaeins, 3) * (deins* pow(tau,
              (3.5))+ dzwei* pow(tau, 12)+ ddrei
              * pow(tau, 20))) * arhoeins* r* Tzwei- *p;

            fvonrho_zwei = (1+ deltazwei* (beins
              * pow(tau, (-0.5))+ bzwei* pow(tau, (1.75))
              + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))
              + 2 * pow(deltazwei, 2) * (ceins* pow(tau,
              (-0.25))+ czwei* pow(tau,(2.5)))+ 3
              * pow(deltazwei, 3) * (deins* pow(tau,
              (3.5))+ dzwei* pow(tau, 12)+ ddrei
              * pow(tau, 20))) * arhozwei* r* Tzwei- *p;

            fvonrho_drei = (1+ deltadrei* (beins
              * pow(tau, (-0.5))+ bzwei* pow(tau, (1.75))
              + bdrei* pow(tau, 4)+ bvier* pow(tau, 12))
              + 2* pow(deltadrei, 2) * (ceins* pow(tau,
              (-0.25))+ czwei* pow(tau,(2.5)))+ 3
              * pow(deltadrei, 3) * (deins* pow(tau,
              (3.5))+ dzwei* pow(tau, 12)+ ddrei
              * pow(tau, 20))) * arhodrei* r* Tzwei- *p;

            if (fvonrho_zwei* fvonrho_drei >= 0)
              {
                arhozwei = arhodrei;
                deltazwei = arhozwei/ rhoc;
              }
            else
              {
                arhoeins = arhozwei;
                deltaeins = arhoeins/ rhoc;
                arhozwei = arhodrei;
                deltazwei = arhozwei/ rhoc;
              }
          }
        rhozwei = arhozwei;
        Tdeltazwei = rhozwei/rhoc;
        tauzwei=tc/Tzwei;
      }
```

```
      if (abs(fvonTzwei)<=abs(fvonTeins))
        {
         *y = Tzwei;
        }
      else
        {
         *y = Teins;
        }
    }
}
/*
-----------------------------------------------------------------
----------------------GATEWAY ROUTINE--------------------------
-----------------------------------------------------------------
*/

void mexFunction( int nlhs, mxArray *plhs[],
              int nrhs, const mxArray *prhs[] )
{
  double *y, *h, *p;
  int mrows, ncols, orows, pcols;

/* checks if there is only one right handed argument */
  if (nrhs =! 2 )
  {
    mexErrMsgTxt("Two input arguments required.");
  }
  else if (nlhs>2)
  {
   mexErrMsgTxt("Too many output arguments");
  }
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  orows = mxGetM(prhs[1]);
  pcols = mxGetN(prhs[1]);

/* checks if input arguments are noncomplex scalar double */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      (!(mrows==1 && ncols==1)) ||
     (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
      (!(orows==1 && pcols==1))))
  {
    mexErrMsgTxt("Input must be a noncomplex scalar double");
     }

/* Set Pointer to right handed arguments */
  h = mxGetPr(prhs[0]);
```

```
  p = mxGetPr(prhs[1]);

/* Set pointer to created output Matrix */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* set pointer to first left handed argument */
  y = mxGetPr(plhs[0]);

/* call computational routine */
  h_T_rho(y,h,p);
}
```

C.3.5 T(p,s)

This function calculates the Temperature in K of R134a at a given pressure in Pa and specific entropy in J/(kg K) in the *overheated gaseous phase*.

Function call:

T_p_s(pressure in `Pa`, specific entropy in `J/(kg K)`, e.g.

```
>>T_p_s(10e5,1.8e3)
ans =

339.4592
```

Sourcecode:

```
/*
------------------------------------------------------------
---------------------T_p_s.c bzw. T_p_s.dll-----------------
------------------------------------------------------------
*/
#include <math.h>
#include "mex.h"
/*
------------------------------------------------------------
------------------COMPUTATIONAL ROUTINE---------------------
------------------------------------------------------------
*/
void T_p_s(double *y, double *p, double *s)
{

  double aeins = -7.7057291;
  double azwei = 2.4186313;
  double adrei = -2.1848312;
  double avier = -3.4530733;
```

```
20

    double beins =   0.2201286;
    double bzwei =   -1.267402;
    double bdrei =   -0.3084991;
25  double bvier =   -3.546135E-04;
    double ceins =   0.01387313;
    double czwei =   0.3353155;
    double deins =   -0.06356633;
    double dzwei =   0.08120353;
30  double ddrei =   -0.03699965;
    double anullstern = 1.019535;
    double aeinsstern = 9.047135;
    double meins =   -0.629789;
    double mzwei =   7.292937;
35  double mdrei =   5.154411;
    double r = 81.488856;
    double tc = 374.18;
    double rhoc = 508;
    double tau, delta;
40  double deltaeins, deltazwei, deltadrei;
    double fvonrho_eins, fvonrho_zwei, fvonrho_drei;
    double rho, rhoeins, rhozwei, rhodrei;
    double Temp[120];
    double Dichte_rho[120];
45  double Druck_p[120];
    int i,j,count = 0;
    for (i=0;i<120;i++)
      {
       Temp[i]=272+ i+1;
50    }

    for (j=0;j<120;j++)
      {
       tau = tc/ Temp[j];
55     rho = 30;
       rhoeins = rho* 0.02;
       rhozwei = rho* 6;

      deltaeins = rhoeins/ rhoc;
60    deltazwei = rhozwei/ rhoc;

      for (i=1;i<30;i++)
        {
          rhodrei = rhozwei+ 0.5* (rhoeins- rhozwei);
65        deltadrei = rhodrei/ rhoc;
          fvonrho_eins = r* (anullstern+ (meins- 1)* (1
```

```
  - log(tau))- log(deltaeins)+ 2* mzwei
  * pow(tau,-0.5)+ 4.0/ 3.0* mdrei
  * pow(tau, -0.75) + deltaeins* (-1.5* beins
  * pow(tau, -0.5)+ 0.75* bzwei* pow(tau, 1.75) + 3
  * bdrei* pow(tau, 4)+ 11* bvier* pow(tau, 12))
  + pow(deltaeins, 2)* (-1.25* ceins
  * pow(tau, -0.25)+ 1.5* ceins* pow(tau, 2.5))
  + pow(deltaeins, 3)* (2.5* deins* pow(tau, 6.5)
  + 11* dzwei* pow(tau, 12)+ 19
  * ddrei* pow(tau, 20)))- *s;

fvonrho_zwei = r* (anullstern+ (meins- 1)* (1
  - log(tau))- log(deltazwei)+ 2* mzwei
  * pow(tau,-0.5)+ 4.0/ 3.0* mdrei
  * pow(tau, -0.75) + deltazwei* (-1.5* beins
  * pow(tau, -0.5)+ 0.75* bzwei* pow(tau, 1.75)+3
  * bdrei* pow(tau, 4)+ 11* bvier* pow(tau, 12))
  + pow(deltazwei, 2)* (-1.25* ceins
  * pow(tau, -0.25)+ 1.5* ceins* pow(tau, 2.5))
  + pow(deltazwei, 3)* (2.5* deins* pow(tau, 6.5)
  + 11* dzwei* pow(tau, 12)+ 19
  * ddrei* pow(tau, 20)))- *s;

fvonrho_drei = r* (anullstern+ (meins- 1)* (1
  - log(tau))- log(deltadrei)+ 2* mzwei
  * pow(tau,-0.5)+ 4.0/ 3.0* mdrei
  * pow(tau, -0.75) + deltadrei* (-1.5* beins
  * pow(tau, -0.5)+ 0.75* bzwei* pow(tau, 1.75) +3
  * bdrei* pow(tau, 4)+ 11* bvier* pow(tau, 12))
  + pow(deltadrei, 2)* (-1.25* ceins
  * pow(tau, -0.25)+ 1.5* ceins* pow(tau, 2.5))
  + pow(deltadrei, 3)* (2.5* deins
  * pow(tau, 6.5)+ 11* dzwei* pow(tau, 12)
  + 19* ddrei* pow(tau, 20)))- *s;

if (fvonrho_zwei* fvonrho_drei >= 0)
  {
    rhozwei = rhodrei;
    deltazwei = rhozwei/ rhoc;
  }
else
  {
    rhoeins = rhozwei;
    deltaeins = rhoeins/ rhoc;
    rhozwei = rhodrei;
    deltazwei = rhozwei/ rhoc;
  }
```

```
      }
    Dichte_rho[j] = rhozwei;
   }
  for(i=0;i<120;i++)
   {
    tau = tc/ Temp[i];
    delta = Dichte_rho[i]/ rhoc;

    Druck_p[i] = (1+ delta* (beins* pow(tau, (-0.5))+ bzwei
     * pow(tau, (1.75)) + bdrei* pow(tau, 4)+ bvier
     * pow(tau, 12))+ 2* pow(delta, 2) * (ceins
     * pow(tau, (-0.25))+ czwei* pow(tau, (2.5)))+ 3
     * pow(delta, 3) * (deins* pow(tau, (3.5))+ dzwei
     * pow(tau,12)+ ddrei* pow(tau, 20)))
     * Dichte_rho[i]* r* Temp[i];
   }
  count=0;
  while ((Druck_p[count] < *p) && (count < 10000))
   {
    count++;
   }
  *y = Temp[count- 1]+ (*p- Druck_p[count- 1])
    / (Druck_p[count] - Druck_p[count- 1])
    * (Temp[count]- Temp[count- 1]);
}
/*
------------------------------------------------------------
----------------------GATEWAY ROUTINE-----------------------
------------------------------------------------------------
*/
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
  double *y, *p, *s;
  int mrows, ncols, orows, pcols;

/* checks if there is only one right handed argument */
  if (nrhs =! 2 )
  {
    mexErrMsgTxt("Two␣input␣arguments␣required.");
  }
  else if (nlhs>2)
  {
   mexErrMsgTxt("Too␣many␣output␣arguments");
  }
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
```

```
  orows = mxGetM(prhs[1]);
  pcols = mxGetN(prhs[1]);

/* checks if input arguments are noncomplex scalar double */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
     (!(mrows==1 && ncols==1)) ||
    (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
     (!(orows==1 && pcols==1))))
  {
    mexErrMsgTxt("Input must be a noncomplex scalar double");
     }

/* Set Pointer to right handed arguments */
  p = mxGetPr(prhs[0]);
  s = mxGetPr(prhs[1]);


/* Set pointer to created output Matrix */
  plhs[0] = mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/* set pointer to first left handed argument */
  y = mxGetPr(plhs[0]);

/* call computational routine */
  T_p_s(y,p,s);
}
```

C.4 log*p*,h-diagram

The following MATLAB-script plots a log*p*,h-diagram:

```
set(0,'DefaultAxesColorOrder',[0,0,0])
% Zeichnen der Grenzkurve
T = 223:5:374;
n = size(T,2);
a = hstrich_T(T)/1000;
b = ones(1,n);
for i=1:n
  b(i) = hzweistrich_T(T(i))/1000;
end;
p = ps_T(T);
p=p*10^(-5);
semilogy(a,p,b,p,'LineWidth',2);
hold on;

% Ermittlung einiger isothermen°
%--- -40C Isotherme ----------
tm40= 233.15;
pm40= linspace(0.3e5,ps_T(tm40),n);
pm40(n-1)=pm40(n);
hm40= ones(1,n);
for i=1:n
  hm40(i)=h_T_rho(tm40,rho_T_p(tm40,pm40(i)))/1000;
end;
hm40(n-1)=hm40(n-1);
hm40(n)=hstrich_T(tm40)/1000;
pm40= pm40/1e5;°

%--- -20C Isotherme ----------
tm20= 253.15;
pm20= linspace(0.3e5,ps_T(tm20),n);
pm20(n-1)=pm20(n);
hm20= ones(1,n);
for i=1:n
  hm20(i)=h_T_rho(tm20,rho_T_p(tm20,pm20(i)))/1000;
end;
hm20(n-1)=hm20(n-1);
hm20(n)=hstrich_T(tm20)/1000;
pm20= pm20/1e5;°

%--- 0C Isotherme ----------
t0= 273.15;
p0= linspace(0.3e5,ps_T(t0),n);
p0(n-1)=p0(n);
h0= ones(1,n);
```

```
for i=1:n
  h0(i)=h_T_rho(t0,rho_T_p(t0,p0(i)))/1000;
end;
h0(n-1)=h0(n-1);
h0(n)=hstrich_T(t0)/1000;
p0= p0/1e5;°

%--- 20C Isotherme ----------
t20= 293.15;
p20= linspace(0.3e5,ps_T(t20),n);
p20(n-1)=p20(n);
h20= ones(1,n);
for i=1:n
  h20(i)=h_T_rho(t20,rho_T_p(t20,p20(i)))/1000;
end;
h20(n-1)=h20(n-1);
h20(n)=hstrich_T(t20)/1000;
p20= p20/1e5;°

%--- 40C Isotherme ----------
t40= 313.15;
p40= linspace(0.3e5,ps_T(t40),n);
p40(n-1)=p40(n);
h40= ones(1,n);
for i=1:n
  h40(i)=h_T_rho(t40,rho_T_p(t40,p40(i)))/1000;
end;
h40(n-1)=h40(n-1);
h40(n)=hstrich_T(t40)/1000;
p40= p40/1e5;°

%--- 60C Isotherme ----------
t60= 333.15;
p60= linspace(0.3e5,ps_T(t60),n);
p60(n-1)=p60(n);
h60= ones(1,n);
for i=1:n
  h60(i)=h_T_rho(t60,rho_T_p(t60,p60(i)))/1000;
end;
h60(n-1)=h60(n-1);
h60(n)=hstrich_T(t60)/1000;
p60= p60/1e5;°

%--- 80C Isotherme ----------
t80= 353.15;
p80= linspace(0.3e5,ps_T(t80),n);
p80(n-1)=p80(n);
```

```
h80= ones(1,n);
for i=1:n
  h80(i)=h_T_rho(t80,rho_T_p(t80,p80(i)))/1000;
end;
h80(n-1)=h80(n-1);
h80(n)=hstrich_T(t80)/1000;
p80= p80/1e5;°

%--- 100C Isotherme ----------
t100= 373.15;
p100= linspace(0.3e5,ps_T(t100),n);
p100(n-1)=p100(n);
h100= ones(1,n);
for i=1:n
  h100(i)=h_T_rho(t100,rho_T_p(t100,p100(i)))/1000;
end;
h100(n-1)=h100(n-1);
h100(n)=hstrich_T(t100)/1000;
p100= p100/1e5;

t120=393.15;
p120=linspace(0.3e5,40e5,n);
h120= ones(1,n);
for i=1:n
  h120(i)=h_T_rho(t120,rho_T_p(t120,p120(i)))/1000;
end;
p120=p120/1e5;

%--- Einzeichnen der Isothermen ---
semilogy(hm40,pm40,hm20,pm20,h0,p0,h20,p20,h40,p40,h60,p60,...
  h80,p80,h100,p100,h120,p120);
set(gca,'ytick',[0.3 1 2 3 4 5 6 7 8 9 10 20 30 40]);

semilogy(hm40,pm40,hm20,pm20,h0,p0,h20,p20,h40,p40,h60,p60,...
  h80,p80,h100,p100,h120,p120);

text(hm40(n)+5,pm40(n),'␣\vartheta°␣=␣-40C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(hm20(n)+5,pm20(n),'␣\vartheta°␣=␣-20C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(h0(n)+5,p0(n),'␣\vartheta°␣=␣0C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
```

```
  'FontSize',6);
text(h20(n)+5,p20(n),' \vartheta° = 20C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(h40(n)+5,p40(n),' \vartheta° = 40C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(h60(n)+5,p60(n),' \vartheta° = 60C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(h80(n)+5,p80(n),' \vartheta° = 80C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);
text(h100(n)+5,p100(n),' \vartheta° = 100C',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'FontSize',6);

%-- Isentropen berechnen und einzeichnen --
T=233:5:393;
n=size(T,2);
s18 = 1.8e3;
rho18 = ones(1,n);
p18=ones(1,n);
h18=ones(1,n);
for i=1:n
  rho18(i)=rho_T_s(T(i),s18);
  p18(i)=p_T_rho(T(i),rho18(i))/1e5;
  h18(i)=h_T_rho(T(i),rho18(i))/1e3;
end;
plot(h18,p18,':');
text(h18(n-5),p18(n-5),' s = 1,8',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'Rotation',70,...
  'FontSize',6);

T=233:5:413;
n=size(T,2);
s19 = 1.9e3;
rho19 = ones(1,n);
p19=ones(1,n);
h19=ones(1,n);
```

```
for i=1:n
  rho19(i)=rho_T_s(T(i),s19);
  p19(i)=p_T_rho(T(i),rho19(i))/1e5;
  h19(i)=h_T_rho(T(i),rho19(i))/1e3;
end;
plot(h19,p19,':');
text(h19(n-5),p19(n-5),' s = 1,9',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'Rotation',70,...
  'FontSize',6);

T=233:5:413;
n=size(T,2);
s2 = 2e3;
rho2 = ones(1,n);
p2=ones(1,n);
h2=ones(1,n);
for i=1:n
  rho2(i)=rho_T_s(T(i),s2);
  p2(i)=p_T_rho(T(i),rho2(i))/1e5;
  h2(i)=h_T_rho(T(i),rho2(i))/1e3;
end;
plot(h2,p2,':');
text(h2(n-5),p2(n-5),' s = 2,0',...
  'HorizontalAlignment','left',...
  'VerticalAlignment','bottom',...
  'Rotation',70,...
  'FontSize',6);

%-- Gitternetz, Beschriftung, etc. --
grid on;
axis([1e2,5.5e2,0.3,45]);
xlabel('Enthalpie h in kJ/kg');
ylabel('p in bar');
title('log p,h-Diagramm üfr R134a');

hold off;
```

C.5 SIMULINK C MEX-S-Functions

The calculation of some values in the model for the passenger compartment are done with special functions in SIMULINK, the *C MEX-S-Functions*. These functions are similar to the *C MEX-Functions*. They compensate for the time handicap of the MATLAB interpreted commands by using C-compiled files.

C.5.1 Cooling load by Transmission

This function calculates the cooling load by Transmission occuring in the passenger compartment.

Sourcecode:

```
#define S_FUNCTION_NAME Transm

#include "simstruc.h"
#include <math.h>

static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumSFcnParams(S, 0);
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    return;
  }

  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);

  if (!ssSetNumInputPorts(S, 1)) return;
  ssSetInputPortWidth(S, 0, 26);
  ssSetInputPortDirectFeedThrough(S, 0, 1);

  if (!ssSetNumOutputPorts(S, 1)) return;
  ssSetOutputPortWidth(S, 0, 6);

  ssSetNumSampleTimes(S, 1);
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);

  ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);

}

static void mdlInitializeConditions(SimStruct *S)
{
}
#endif
#define MDL_START
#if defined(MDL_START)

static void mdlStart(SimStruct *S)
{
}
#endif

static void mdlOutputs(SimStruct *S, int_T tid)
{
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
  real_T        *y   = ssGetOutputPortRealSignal(S,0);

  double WF,WS,WH,FR,gamma,delta,omega,psi,tau,Inf,
    Inr,Inh,Inl,Ind;
  double as,as1,s[3],cf[3],cr[3],ch[3],cl[3],cd[3],
    n,aaf,aas,aah,aad,aif,ais,aih,aid,aib;
  double dtf,dtr,dth,dtl,dtd,Q;
  int mon[12],i;

  WF = 2*M_PI/360*(*uPtrs[9]);
  WS = 2*M_PI/360*(*uPtrs[13]);
  WH = 2*M_PI/360*(*uPtrs[11]);
  FR = 2*M_PI/360*(*uPtrs[1]);

  mon[0] = 0;
  mon[1] = 31;
  mon[2] = 59;
  mon[3] = 90;
  mon[4] = 120;
  mon[5] = 151;
  mon[6] = 181;
  mon[7] = 212;
  mon[8] = 243;
  mon[9] = 273;
  mon[10] = 304;
  mon[11] = 334;
```

```
    gamma = (*uPtrs[0])/180*M_PI;

    i = (*uPtrs[3])-1;
    delta = (-0.4093*cos(2*M_PI*(mon[i]+*uPtrs[4]+10)/365));
    omega = 0.2618*(*uPtrs[2]);
    psi=asin(sin(delta)*sin(gamma)-cos(delta)
           *cos(gamma)*cos(omega));

    Inf=0;
    Inr=0;
    Inh=0;
    Inl=0;
    Ind=0;

    if (psi > 0)
      {
       as = asin(cos(delta)*sin(omega)/cos(psi));
       as1 = asin(cos(delta)*sin(omega+0.00002)/cos(psi));

       if (as >= 0 && as1 < as)
        as=M_PI-fabs(as);
       else
        {
         if (as < 0 && as1 <= as)
          as=M_PI+fabs(as);
         else
          {
           if (as < 0 && as1 > as)
            as=2*M_PI-fabs(as);
          }
        }
       if (gamma < 0)
        as=M_PI-as;
       if (as < 0)
        as=2*M_PI+as;

       s[0] = cos(psi)*sin(as);
       s[1] = cos(psi)*cos(as);
       s[2] = sin(psi);

       cf[0] = sin(WF)*sin(FR);
       cf[1] = sin(WF)*cos(FR);
       cf[2] = cos(WF);

       cr[0] = sin(WS)*sin(FR+.5*M_PI);
       cr[1] = sin(WS)*cos(FR+.5*M_PI);
       cr[2] = cos(WS);
```

```
130       ch[0] = sin(WH)*sin(FR+M_PI);
          ch[1] = sin(WH)*cos(FR+M_PI);
          ch[2] = cos(WH);

          cl[0] = sin(WS)*sin(FR+1.5*M_PI);
135       cl[1] = sin(WS)*cos(FR+1.5*M_PI);
          cl[2] = cos(WS);

          cd[0] = 0;
          cd[1] = 0;
140       cd[2] = 1;


          Inf = (*uPtrs[6])*(s[0]*cf[0]+s[1]*cf[1]+s[2]*cf[2]);
          if (Inf < (*uPtrs[7]))
145         Inf = (*uPtrs[7]);
          Inf = (*uPtrs[16])*Inf;

          Inr = (*uPtrs[6])*(s[0]*cr[0]+s[1]*cr[1]+s[2]*cr[2]);
          if (Inr < (*uPtrs[7]))
150         Inr = (*uPtrs[7]);
          Inr = (*uPtrs[16])*Inr;

          Inh = (*uPtrs[6])*(s[0]*ch[0]+s[1]*ch[1]+s[2]*ch[2]);
          if (Inh < (*uPtrs[7]))
155         Inh = (*uPtrs[7]);
          Inh = (*uPtrs[16])*Inh;

          Inl = (*uPtrs[6])*(s[0]*cl[0]+s[1]*cl[1]+s[2]*cl[2]);
          if (Inl < (*uPtrs[7]))
160         Inl = (*uPtrs[7]);
          Inl = (*uPtrs[16])*Inl;

          Ind = (*uPtrs[6])*(s[0]*cd[0]+s[1]*cd[1]+s[2]*cd[2]);
          if (Ind < (*uPtrs[7]))
165         Ind = (*uPtrs[7]);
          Ind = (*uPtrs[15])*Ind;
        }

      n = pow(((*uPtrs[17])/3.6),0.8);
170   aaf = 3.79e-3*n;
      if (aaf < 25e-3)
        aaf = 25e-3;

      aas = 7.21e-3*n;
175   if (aas < 25e-3)
```

```
  aas = 25e-3;

aah = 4.65e-3*n;
if (aah < 25e-3)
  aah = 25e-3;

aad = 4.41e-3*n;
if (aad < 25e-3)
  aad = 25e-3;

n = sqrt((*uPtrs[18])*3000);

aif = 0.584e-3*n;
if (aif < 7e-3)
  aif = 7e-3;

ais = 0.495e-3*n;
if (ais < 7e-3)
  ais = 7e-3;

aih = 0.700e-3*n;
if (aih < 7e-3)
  aih = 7e-3;

aid = 4.6e-3;

aib = 4.6e-3;

dtf = (Inf+aaf*((*uPtrs[19])-(*uPtrs[21]))
      +aif*((*uPtrs[20])-(*uPtrs[21])))/8;
dtr = (Inr+aas*((*uPtrs[19])-(*uPtrs[22]))
      +ais*((*uPtrs[20])-(*uPtrs[22])))/8;
dth = (Inh+aah*((*uPtrs[19])-(*uPtrs[23]))
      +aih*((*uPtrs[20])-(*uPtrs[23])))/8;
dtl = (Inl+aas*((*uPtrs[19])-(*uPtrs[24]))
      +ais*((*uPtrs[20])-(*uPtrs[24])))/8;
dtd = (Ind+aad*((*uPtrs[19])-(*uPtrs[25]))
      +aid*((*uPtrs[20])-(*uPtrs[25])))/2.92;

Q = aif*(*uPtrs[8])*((*uPtrs[21])-(*uPtrs[20]))
  + ais*.5*(*uPtrs[12])*((*uPtrs[22])-(*uPtrs[20]))
  + aih*(*uPtrs[10])*((*uPtrs[23])-(*uPtrs[20]))
  + ais*.5*(*uPtrs[12])*((*uPtrs[24])-(*uPtrs[20]))
  + aid*.3*(*uPtrs[5])*((*uPtrs[25])-(*uPtrs[20]))
  + aib*.7*(*uPtrs[5])*((*uPtrs[19])-(*uPtrs[20]))
  + 7e-3*(100-(*uPtrs[20]));
```

```
  y[0] = dtf;
  y[1] = dtr;
  y[2] = dth;
  y[3] = dtl;
  y[4] = dtd;
  y[5] = Q;


}



#define MDL_UPDATE
#if defined(MDL_UPDATE)
static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif
#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
static void mdlDerivatives(SimStruct *S)
{
}
#endif

static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

C.5.2 Cooling load by air leakage

This function calculates the cooling load by air leakage occuring in the passenger compartment.

Sourcecode:

```
#include <math.h>

static void mdlInitializeSizes(SimStruct *S)
{

  ssSetNumSFcnParams(S, 0);
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
```

```
    return;
  }

  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);

  if (!ssSetNumInputPorts(S, 1)) return;
  ssSetInputPortWidth(S, 0, 8);
  ssSetInputPortDirectFeedThrough(S, 0, 1);

  if (!ssSetNumOutputPorts(S, 1)) return;
  ssSetOutputPortWidth(S, 0, 1);

  ssSetNumSampleTimes(S, 1);
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);

  ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);

}

#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)
static void mdlInitializeConditions(SimStruct *S)
{
}
#endif



#define MDL_START
#if defined(MDL_START)

static void mdlStart(SimStruct *S)
{
}
#endif
```

```
55 static void mdlOutputs(SimStruct *S, int_T tid)
   {
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T        *y   = ssGetOutputPortRealSignal(S,0);
    real_T        mll,ps_au,ps_in,x_au,x_in,h_au,h_in;
60
    mll = (*uPtrs[7])/6000*(0.75*(*uPtrs[6])+0.25);

    if ((*uPtrs[4]) < 0)
     ps_au = 4.689*pow(1.468+(*uPtrs[4])/100,12.3);
65  else
     ps_au = 288.68*pow(1.098+(*uPtrs[4])/100,8.02);

    if ((*uPtrs[2]) < 0)
     ps_in = 4.689*pow(1.468+(*uPtrs[2])/100,12.3);
70  else
     ps_in = 288.68*pow(1.098+(*uPtrs[2])/100,8.02);

    x_au = 0.622*(*uPtrs[1])*ps_au/((*uPtrs[5])
                              -(*uPtrs[1])*ps_au);
75  x_in = 0.622*(*uPtrs[3])*ps_au/((*uPtrs[5])
                              -(*uPtrs[3])*ps_au);
    h_au = 1.006*(*uPtrs[4])
     +x_au*(2501.6+1.86*(*uPtrs[4]));
    h_in = 1.006*(*uPtrs[2])
80   +x_in*(2501.6+1.86*(*uPtrs[2]));

    y[0] = mll*(h_au-h_in);
   }

85 #define MDL_UPDATE
   #if defined(MDL_UPDATE)

    static void mdlUpdate(SimStruct *S, int_T tid)
    {
90  }
   #endif
   #define MDL_DERIVATIVES
   #if defined(MDL_DERIVATIVES)

95 static void mdlDerivatives(SimStruct *S)
   {
   }
   #endif

100 static void mdlTerminate(SimStruct *S)
   {
```

```
}
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

C.5.3 Rotation of the vehicle

This short function allows the investigated vehicle to be rotated by 360 degrees to compensate for the dependencies of the driving direction.

Sourcecode:

```
#define S_FUNCTION_NAME Rotation

#include "simstruc.h"
#include <math.h>

static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return;
   }

   ssSetNumContStates(S, 0);
   ssSetNumDiscStates(S, 0);

   if (!ssSetNumInputPorts(S, 1)) return;
   ssSetInputPortWidth(S, 0, 1);
   ssSetInputPortDirectFeedThrough(S, 0, 1);

   if (!ssSetNumOutputPorts(S, 1)) return;
   ssSetOutputPortWidth(S, 0, 1);

   ssSetNumSampleTimes(S, 1);
   ssSetNumRWork(S, 0);
   ssSetNumIWork(S, 0);
   ssSetNumPWork(S, 0);
   ssSetNumModes(S, 0);
   ssSetNumNonsampledZCs(S, 0);

   ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
35    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
      ssSetOffsetTime(S, 0, 0.0);

  }
  #define MDL_INITIALIZE_CONDITIONS
40 #if defined(MDL_INITIALIZE_CONDITIONS)

    static void mdlInitializeConditions(SimStruct *S)
    {
    }
45 #endif
  #define MDL_START
  #if defined(MDL_START)

    static void mdlStart(SimStruct *S)
50  {
    }
  #endif

  static void mdlOutputs(SimStruct *S, int_T tid)
55 {
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T        *y  = ssGetOutputPortRealSignal(S,0);
    int     winkel;

60  winkel = (*uPtrs[0]);
    winkel = winkel % 360;
    y[0] = winkel;
  }
  #define MDL_UPDATE
65 #if defined(MDL_UPDATE)

    static void mdlUpdate(SimStruct *S, int_T tid)
    {
    }
70 #endif
  #define MDL_DERIVATIVES
  #if defined(MDL_DERIVATIVES)
    static void mdlDerivatives(SimStruct *S)
    {
75  }
  #endif

  static void mdlTerminate(SimStruct *S)
  {
80 }
  #ifdef MATLAB_MEX_FILE
```

```
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

C.5.4 Solar radiation

This function calculates the solar irradiation depending on the location and time.

Sourcecode:

```
#define S_FUNCTION_NAME Sonne

#include "simstruc.h"
#include <math.h>

static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumSFcnParams(S, 0);
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    return;
  }

  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);

  if (!ssSetNumInputPorts(S, 1)) return;
  ssSetInputPortWidth(S, 0, 14);
  ssSetInputPortDirectFeedThrough(S, 0, 1);

  if (!ssSetNumOutputPorts(S, 1)) return;
  ssSetOutputPortWidth(S, 0, 1);

  ssSetNumSampleTimes(S, 1);
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);

  ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
```

```
}
#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)

static void mdlInitializeConditions(SimStruct *S)
{
}
#endif
#define MDL_START
#if defined(MDL_START)

static void mdlStart(SimStruct *S)
{
}
#endif

static void mdlOutputs(SimStruct *S, int_T tid)
{
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
  real_T       *y   = ssGetOutputPortRealSignal(S,0);

  double WF,WS,WH,FR,gamma,delta,omega,psi,Qges,as,as1;
  double s[3],cf[3],cr[3],ch[3],cl[3],Qf,Qr,Qh,Ql;
  int  i,mon[12];

  WF = 2*M_PI/360*(*uPtrs[9]);
  WS = 2*M_PI/360*(*uPtrs[13]);
  WH = 2*M_PI/360*(*uPtrs[11]);
  FR = 2*M_PI/360*(*uPtrs[1]);

  mon[0] = 0;
  mon[1] = 31;
  mon[2] = 59;
  mon[3] = 90;
  mon[4] = 120;
  mon[5] = 151;
  mon[6] = 181;
  mon[7] = 212;
  mon[8] = 243;
  mon[9] = 273;
  mon[10] = 304;
  mon[11] = 334;

  gamma = *uPtrs[0]/180*M_PI;
  i    = *uPtrs[3]-1;
  delta = (-0.4093*cos(2*M_PI*(mon[i]
        +(*uPtrs[4])+10)/365));
```

```
  omega = 0.2618*(*uPtrs[2]);
  psi  = asin(sin(delta)*sin(gamma)
         -cos(delta)*cos(gamma)*cos(omega));

  Qges = 0;

  if (psi > 0)
    {
     as = asin(cos(delta)*sin(omega)/cos(psi));
     as1 = asin(cos(delta)*sin(omega+0.00002)/cos(psi));

     if (as >= 0 && as1 < as)
       as=M_PI-fabs(as);
     else
       {
         if (as < 0 && as1 <= as)
           as=M_PI+fabs(as);
         else
           {
           if (as < 0 && as1 > as)
             as=2*M_PI-fabs(as);
           }
       }
     if (gamma < 0)
       as=M_PI-as;
     if (as < 0)
       as=2*M_PI+as;
     s[0] = cos(psi)*sin(as);
     s[1] = cos(psi)*cos(as);
     s[2] = sin(psi);

     cf[0] = sin(WF)*sin(FR);
     cf[1] = sin(WF)*cos(FR);
     cf[2] = cos(WF);

     cr[0] = sin(WS)*sin(FR+.5*M_PI);
     cr[1] = sin(WS)*cos(FR+.5*M_PI);
     cr[2] = cos(WS);

     ch[0] = sin(WH)*sin(FR+M_PI);
     ch[1] = sin(WH)*cos(FR+M_PI);
     ch[2] = cos(WH);

     cl[0] = sin(WS)*sin(FR+1.5*M_PI);
     cl[1] = sin(WS)*cos(FR+1.5*M_PI);
     cl[2] = cos(WS);
```

```
      Qf = (*uPtrs[6])*(s[0]*cf[0]+s[1]*cf[1]+s[2]*cf[2]);
      if (Qf < (*uPtrs[7]))
        Qf = (*uPtrs[7]);
      Qf = Qf*(*uPtrs[8]);

      Qr = (*uPtrs[6])*(s[0]*cr[0]+s[1]*cr[1]+s[2]*cr[2]);
      if (Qr < (*uPtrs[7]))
        Qr = (*uPtrs[7]);
      Qr = Qr*(*uPtrs[12])/2;

      Qh = (*uPtrs[6])*(s[0]*ch[0]+s[1]*ch[1]+s[2]*ch[2]);
      if (Qh < (*uPtrs[7]))
        Qh = (*uPtrs[7]);
      Qh = Qh*(*uPtrs[10]);

      Ql = (*uPtrs[6])*(s[0]*cl[0]+s[1]*cl[1]+s[2]*cl[2]);
      if (Ql < (*uPtrs[7]))
        Ql = (*uPtrs[7]);
      Ql = Ql*(*uPtrs[12])/2;

      Qges = (*uPtrs[5])*(Qf + Qr + Qh + Ql);
    }
  *y = Qges;
}
#define MDL_UPDATE
#if defined(MDL_UPDATE)

static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif
#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)

static void mdlDerivatives(SimStruct *S)
{
}
#endif

static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

C.5.5 Air temperature controller

This function realizes a P-controller for the air temperature.

Sourcecode:

```
#define S_FUNCTION_NAME Zuluftregler

#include "simstruc.h"
#include <math.h>

static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumSFcnParams(S, 0);
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    return;
  }

  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);

  if (!ssSetNumInputPorts(S, 1)) return;
  ssSetInputPortWidth(S, 0, 2);
  ssSetInputPortDirectFeedThrough(S, 0, 1);

  if (!ssSetNumOutputPorts(S, 1)) return;
  ssSetOutputPortWidth(S, 0, 1);

  ssSetNumSampleTimes(S, 1);
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);

  ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)

static void mdlInitializeConditions(SimStruct *S)
{
}
```

```
#endif
#define MDL_START
#if defined(MDL_START)

static void mdlStart(SimStruct *S)
{
}
#endif

static void mdlOutputs(SimStruct *S, int_T tid)
{
 InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
 real_T        *y   = ssGetOutputPortRealSignal(S,0);
 real_T        tluft,tsoll,xw,Kpr,tzusoll;

 tluft = (*uPtrs[0]);
 tsoll = (*uPtrs[1]);
 Kpr = -16;

 xw = tluft-tsoll;

 tzusoll = xw*Kpr+tsoll;
 if (tzusoll < 6)
  tzusoll = 6;
 if (tzusoll > 70)
  tzusoll =70;

 *y = tzusoll;
}
#define MDL_UPDATE
#if defined(MDL_UPDATE)

static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif
#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)

static void mdlDerivatives(SimStruct *S)
{
}
#endif

static void mdlTerminate(SimStruct *S)
{
}
```

```
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```