NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for naturvitenskap og teknologi
Institutt for kjemisk prosessteknologi

SPECIALIZATION PROJECT 2012

TKP 4550

PROJECT TITLE:
Refrigeration Cycle CO_2

By

Nils Arne Susort

Supervisor for the project:
Professor Sigurd Skogestad

Date:
07.12.12

Abstract

A simple refrigeration cycle based on CO_2 which operates trans-critical has been modeled in the equation based, object-oriented language Modelica. The modeling methodology, how the final model of the cycle is carried out, is described in the report. The modeled cycle turned out to differ from a reference model, but the differences was minor. A feasible steady-state solution to the Modelica model was obtained, but not considered as optimal.

Preface

This report is reflecting the work done and achieved results from the specialization project TKP4550. The credit of the project is 15 SP. The head of the project is incorporated inside the process systems engineering group at the department of chemical engineering as a compulsory part of the 5-year Master program. Many thanks to my supervisor Sigurd Skogestad and my co-supervisors Vladimiros Minasidis and Johannes Jäschke.

Trondheim
December 7, 2012

Nils Arne Susort

Contents

List of Figures

List of Tables

1 Introduction

Vapor compression cycle is the most common process utilized in refrigeration systems. Today, refrigeration cycles are of great importance in the household (e.g. refrigerator) , automotive vehicles (e.g. AC) and industry (e.g. LNG). Modeling and optimization of the mentioned processes is advantageous when speaking of saving nature (choice of refrigerant) and possible COP enhancement. The decision of choosing CO_2 as refrigerant reflects the worlds increasing interest and request for non-toxic, non-hazardous working medium.

This project was motivated by first of all reproducing parts of the work J. B. Jensen [1] did in his PhD thesis, which was done in MATLAB®, and implement a close to similar model in Modelica®. The intention of using Modelica was to avoid the causality which dominates many programming languages and take advantage of the benefits Modelica provides. Once a model is written, the Modelica engine compiles the code into C-code or XML whereby almost anything can be further analyzed (like optimization). Several books, articles and internet forums have been frequently employed to gain knowledge about Modelica usage, but the book by Fritzson [2] have been used as base material to build a model.

The refrigeration cycle studied in this project utilizes CO_2 as cooling medium. The cycle is kept at a simple level and the steady state behavior has been of most interest, although dynamics is included in the evaporator. In contrast to the model simplicity, a quite precise thermodynamic equation of state routine by Span and Wagner [3] is implemented to the model. The equation of state is in the form of a fundamental equation explicit in the Helmholtz free energy valid in the region $216\,\mathrm{K} \leq T \leq 1100\,\mathrm{K}$ and $0\,\mathrm{MPa} \leq p \leq 800\,\mathrm{MPa}$. For instance the uncertainty in density in the region up to temperatures and pressures of 523 K and 30 MPa, respectively ranges from $\pm 0.03\,\%$ to $\pm 0.05\,\%$. Only the trans-critical case is considered, meaning that the low pressure side of the process is sub-critical and the high pressure side is super-critical.

2 Process Description

The refrigeration cycle is depicted in Figure 2.1 and typical process variables[1] are announced in Table 2.2. The design conditions is given in Table 2.1. The evaporator operates in the two-phase region where the gas and the liquid is in equilibrium. Saturated gas is ensured due to the accumulator tank after the evaporator. The compressor is 75 % close to isentropic and compresses the gas from the sub-critical region to the super-critical region. The gas cooler condenses the gas and the heat flow entering the ambient air ($\dot{Q}_{GCO}$) at steady state is equal to the heat flow entering the evaporator ($\dot{Q}_{EVP}$) plus the shaft work to the compressor ($\dot{W}_{shaft}$). At steady state also the heat loss between the room and the ambient ($\dot{Q}_{loss}$) has to be equal to the heat entering the evaporator ($\dot{Q}_{EVP}$) to maintain constant room temperature. The internal heat exchanger is included by the fact that some studies have proven increased efficiency of sub-cooling the high pressure gas before the valve inlet and super-heating the gas from the evaporator outlet. This reduces the expansion loss through the valve, even though a super-heated gas will increase the compressor power compared to saturated gas. The valve is modeled as pure isenthalpic.

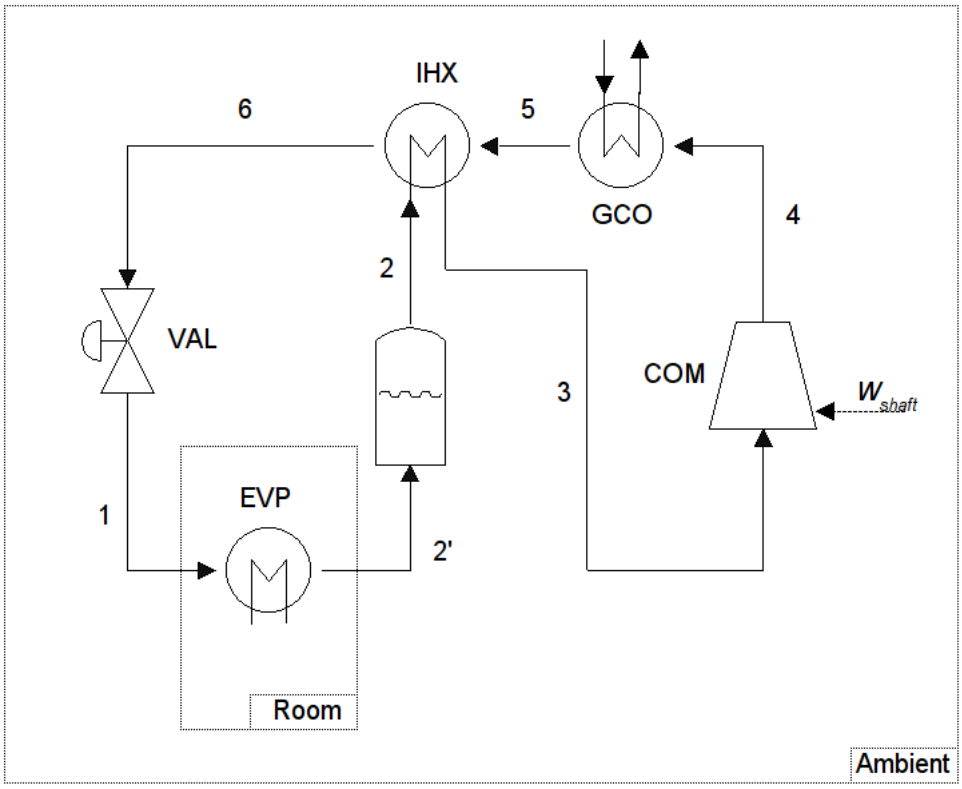

Figure 2.1: The refrigeration cycle process includes an evaporator (EVP), an internal heat exchanger (IHX), a compressor (COM), a gas cooler (GCO) and a valve (VAL).

[1]The variables presented are inherited from the exact same refrigeration cycle defended by J. B. Jensen's PhD thesis [1]

Table 2.1: The design conditions for the refrigeration cycle

Process unit / lumped region	**Condition Description**	**Symbol**	**Value**
Evaporator	Thermal conductance	$\dot{U}A_{EVP}$ [kW K^{-1}]	0.800
Valve	Choke	Cv [m^2]	1.21×10^{-6}
Internal HEX	Thermal conductance	$\dot{U}A_{IHX}$ [kW K^{-1}]	0.154
	Control volumes	n_{IHX}	6
Gas Cooler	Thermal conductance	$\dot{U}A_{GCO}$ [kW K^{-1}]	0.794
	Control volumes	n_{GCO}	6
Compressor	Isentropic efficiency	η	0.75
Room	Thermal conductance	$\dot{U}A_{loss}$ [kW K^{-1}]	0.400
	Temperature	T^{air} [K]	293.15
Ambient air	Temperature	T^{air} [K]	303.15
	Mass flow rate	$\dot{m}^{air}$ [kg s^{-1}]	0.25
	Heat capacity at constant pressure	C_p^{air} [kJ kg^{-1} K^{-1}]	1.00

Table 2.2: Typical variables which are optimized by J. B. Jensen

Variable	**Symbol**	**Value**
Compressor power	$\dot{W}_{shaft}$ [kW]	0.958
Valve opening	z	0.340
High pressure side	p_4, p_5 and p_6 [kPa]	9761
Low pressure side	p_1, $p_{2'}$, p_2 and p_3 [kPa]	5083
Heat flow leaving the gas cooler	$\dot{Q}_{GCO}$ [kW]	4.958
Heat flow entering the evaporator	$\dot{Q}_{EVP}$ [kW]	4.000
Heat flow exchanged in the internal HEX	$\dot{Q}_{IHX}$ [kW]	0.889
Mass flow rate circulating	$\dot{m}$ [kg s^{-1}]	0.0250
Temperatures	T_1 [K]	288.2
	T_3 [K]	304.4
	T_4 [K]	362.8
	T_6 [K]	298.7

3 Computer tools and thermodynamics

3.1 OpenModelica

The modeling and simulations are performed in the open-source environment OpenModelica by use of the OMC compiler. The user friendly environment is suited for both modeling, simulation and plotting of results. The OMC compiler handles most of the features present in the Modelica language, but there happen to be a couple of bugs.

3.2 CoolProp

Necessary lookup tables for the thermodynamics are created by the open-source database CoolProp. CoolProp is easily accessed inside the Python interpreter and uses the fundamental equation of state developed by Span and Wagner [3] for calculation of the CO_2 properties. As long as the thermodynamic tables is finely discretized there should be no difference between interpolation in lookup tables and calculating directly from the equations, confirmed by Andresen [4]. The model setup uses lookup tables in the evaporator due to it's simplicity when gas and liquid phase is in equilibrium. The saturation table is only dependent of one single thermodynamic property, e.g. temperature or pressure, to calculate rest of the unknown properties, which makes it simple. All the other units in the refrigeration cycle is modeled by solving the equation of state explicitly.

3.3 Thermodynamic considerations

The thermodynamic equation of state given in equation (3.1) is a fundamental equation expressed in form of Helmholtz energy.

$$\phi(\delta, \tau) = \phi^0(\delta, \tau) + \phi^r(\delta, \tau) \tag{3.1}$$

Where:

$$\begin{aligned}
\phi &: \text{ dimensionless Helmholtz energy } \left(\frac{A(\rho,T)}{RT}\right) \\
\phi^0 &: \text{ dimensionless ideal part of Helmholtz energy } \left(\frac{A^0(\rho,T)}{RT}\right) \\
\phi^r &: \text{ dimensionless residual part of Helmholtz energy } \left(\frac{A^r(\rho,T)}{RT}\right) \\
\delta &: \text{ dimensionless density } \left(\frac{\rho}{\rho_c}\right) \\
\rho_c &: \text{ critical density } (467.6\,\mathrm{kg\,m^{-3}}) \\
\tau &: \text{ dimensionless temperature } \left(\frac{T_c}{T}\right) \\
T_c &: \text{ critical temperature } (304.1282\,\mathrm{K}) \\
R &: \text{ gas constant } (0.188\,924\,1\,\mathrm{kJ\,kg^{-1}\,K^{-1}})
\end{aligned}$$

The expressions for $\phi^0(\delta,\tau)$ and $\phi^r(\delta,\tau)$ are given in equation (3.2) and (3.3), respectively. The coefficients, the exponents and the derivatives of the dimensionless Helmholtz energy are not included in this section, due to the extensiveness, but can be investigated either in the Modelica code [5] or in the original paper by Span and Wagner [3]. Also a sample on how the functions are implemented in the Modelica model is included in appendix B.1.

$$\phi^0(\delta,\tau) = \ln(\delta) + a_1^0 + a_2^0\tau + a_3^0\ln(\tau) + \sum_{i=4}^{8} a_i^0 \ln[1 - e^{-\tau\theta_i^0}] \tag{3.2}$$

Where:

$$\begin{aligned}
a_i^0 &: \text{ coefficient for number } i \\
\theta_i^0 &: \text{ exponent for number } i
\end{aligned}$$

$$\begin{aligned}
\phi^r(\delta,\tau) = &\sum_{i=1}^{7} n_i \delta^{d_i}\tau^{t_i} + \sum_{i=8}^{34} n_i \delta^{d_i}\tau^{t_i} e^{-\delta^{c_i}} + \sum_{i=35}^{39} n_i \delta^{d_i}\tau^{t_i} e^{-\alpha_i(\delta-\epsilon_i)^2-\beta_i(\tau-\gamma_i)^2} \\
&+ \sum_{i=40}^{42} n_i \Delta^{b_i}\delta e^{-C_i(\delta-1)^2-D_i(\tau-1)^2}
\end{aligned} \tag{3.3}$$

Where:

$$\begin{aligned}
n_i &: \text{ coefficient for number } i \\
d_i, t_i, c_i, \alpha_i, \beta_i, \gamma_i, \epsilon_i, a_i, b_i, A_i, B_i, C_i, D_i &: \text{ exponents for number } i
\end{aligned}$$

Calculation of thermodynamic properties is intuitive from the Helmholtz energy. How this is done for pressure, entropy, specific internal energy and specific enthalpy, which are utilized in the Modelica model, are shown in equation (3.4a), (3.4b), (3.4c) and (3.4d), respectively.

$$\frac{p(\delta,\tau)}{\rho RT} = 1 + \delta\phi_\delta^r \tag{3.4a}$$

$$\frac{s(\delta,\tau)}{R} = \tau(\phi_\tau^0 + \phi_\tau^r) - \phi^0 - \phi^r \tag{3.4b}$$

$$\frac{u(\delta,\tau)}{RT} = \tau(\phi_\tau^0 + \phi_\tau^r) \tag{3.4c}$$

$$\frac{h(\delta,\tau)}{RT} = 1 + \tau(\phi_\tau^0 + \phi_\tau^r) + \delta\phi_\delta^r \tag{3.4d}$$

The refrigeration cycle is modeled in terms of enthalpy and pressure. The thermodynamic equation of state is based on ideal and residual parts of the Helmholtz energy. To make the model handle enthalpies and pressures as function inputs, the temperatures and densities have to be explicitly calculated by an iteration algorithm. To do so, a Newton-Raphson method, equation (3.7), is implemented to the model. The necessary equations for the iteration strategy, the residual function and the Jacobian, is given in the equation (3.5) and (3.6), respectively. In the compressor model, also the outlet pressure needs to be calculated when only enthalpy and entropy is known from the inputs, hence the same iteration procedure apply for that case.

$$\mathbf{f} = \begin{bmatrix} f_1(\delta,\tau) \\ f_2(\delta,\tau) \end{bmatrix} \tag{3.5}$$

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial f_1(\delta,\tau)}{\partial\tau} & \frac{\partial f_1(\delta,\tau)}{\partial\delta} \\ \frac{\partial f_2(\delta,\tau)}{\partial\tau} & \frac{\partial f_2(\delta,\tau)}{\partial\delta} \end{bmatrix} \tag{3.6}$$

Where:

$$\begin{aligned} f_i &: \text{ residual function } i \text{ for desirable state} \\ \tfrac{\partial f_i}{\partial\tau} &: \text{ partial derivative of residual function } i \text{ with respect to } \tau \\ \tfrac{\partial f_i}{\partial\delta} &: \text{ partial derivative of residual function } i \text{ with respect to } \delta \end{aligned}$$

$$\begin{bmatrix} \tau \\ \delta \end{bmatrix}^{(k+1)} = \begin{bmatrix} \tau \\ \delta \end{bmatrix}^{(k)} - \mathbf{J_f}^{-1(k)} \mathbf{f}^{(k)} \tag{3.7}$$

Where:

$$k : \text{ iteration counter}$$

For instance, when density and temperature need to be solved explicitly in terms of enthalpy and pressure, the residual function $f_1^{(k)}$ is the difference between the pressure at iteration k

and the desirable pressure. Likewise is the residual function $f_2^{(k)}$ the difference between the enthalpy at iteration k and the desirable enthalpy. The iteration terminates when a certain confidence error is reached. Although the Newton-Raphson method is effective, a reasonable guess value has to be selected at startup. If not, it may iterate for ever and won't find a solution. A sample on how the iteration method is implemented in the Modelica model is attached to appendix B.2.

4 Modeling methodology

The cycle in Figure 2.1 is considered when discussing the modeling approach below. The Modelica model code is attached in Appendix A and available at the internet [5] along with the thermodynamic code. How the model calculates can in principle be done in several ways, however the model code following this report is only concerned about one single procedure. The applied simulation procedure is inspired by the standard calculation procedure in CSIM, covered by T. Andresen [4]. This procedure is illustrated in Figure 4.1. Initialization takes place at the starting point (evaporator outlet), where temperature (T_{out}) and mass flow rate ($\dot{m}_{out}$) are the only initialization variables. Basically, there are no differences whether the starting point is set at the inlet or at the outlet since the temperature is constant throughout the evaporator. The compressor do also need to be initialized at the same time requiring power (W_{shaft}) and a guess value for the temporary enthalpy (h_{in}) at the inlet. The temporary enthalpy is only for initializing the compressor such that the substance properties inside the gas cooler and the internal heat exchanger successfully can be calculated. After the first simulation interval, this enthalpy is updated (at the secondary comparison point) from the internal heat exchanger outlet, every time instant behind the present. At every time interval is the enthalpy at the valve inlet and at the valve outlet compared (at the primary comparison point) and the compressor power is manipulated if there are inconsistencies. This scheme controls the conservation of the energy of the cycle.

Complete model equations is given in equations (4.1) trough (4.5). Transport phenomenas and hence friction is not taken into account. The evaporator is the only unit that has dynamic terms since it is alone operating in the two-phase region. The holdups are nowhere calculated since specific properties dominate the model.

4.1 Evaporator

The equations that are used to model the evaporator is given in the equation set (4.1). The evaporator operates sub-critical and the outlet flow is connected to a tank ensuring saturated conditions for the gas. The tank is also indirectly controlling the active charge in the process. The steady state scenario assumes constant room temperature of 20 °C and fixed load (fixed UA-value). Constant room temperature is achieved if the "heat loss" out of the room is equal to the heat load transfered to the evaporator. The design specifications from Table 2.1 shows that this steady-state condition is achieved when the heat load is 4 kW and the evaporator temperature is 15 °C. The Modelica model is constructed to use temperature and mass flow rate as input and linear interpolation in the lookup table to find saturated pressure, density and enthalpy. The internal energy is also calculated from the equation of state.

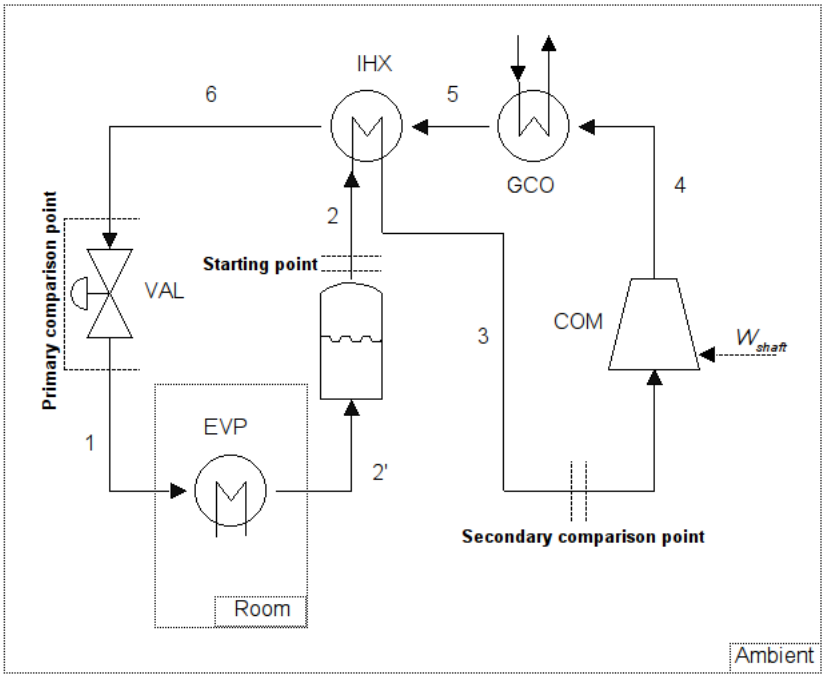

Figure 4.1: Utilized simulation scheme in the Modelica model. The starting point is where the simulation is initialized. A starting value for the enthalpy entering the inlet of the compressor is initialized at the secondary comparison point and updated by the enthalpy from the internal heat exchanger outlet every interval time. The enthalpy balance is controlled at the primary comparison point.

$$\frac{dm}{dt} = \dot{m}_{in} - \dot{m}_{out} \tag{4.1a}$$

$$\frac{dU}{dt} = \dot{m}_{in}h_{in} - \dot{m}_{out}h_{out} + \dot{Q} \tag{4.1b}$$

$$\frac{dU}{dt} = u\frac{dm}{dt} \tag{4.1c}$$

$$\dot{Q} = \dot{U}A(T^{air} - T_{out}) \tag{4.1d}$$

$$q = \frac{h_{in} - h_{liq}}{h_{vap} - h_{liq}} \tag{4.1e}$$

$$\dot{m}_{evap} = \frac{\dot{Q}}{h_{vap} - h_{liq}} \tag{4.1f}$$

$$0 = q\dot{m}_{in} - \dot{m}_{out} + \dot{m}_{evap} \tag{4.1g}$$

$$h_{out} = h_{vap} \tag{4.1h}$$

$$p_{in} = p_{out} \tag{4.1i}$$

$$(p_{out}, \rho_{vap}, h_{vap}) = f(T_{out}) \tag{4.1j}$$

$$(p_{out}, \rho_{liq}, h_{liq}) = f(T_{out}) \tag{4.1k}$$

$$u = f(\rho_{liq}, T_{out}) \tag{4.1l}$$

Where:

m : mass inventory
$\dot{m}$: mass flow rate
U : internal energy
h : specific enthalpy
$\dot{Q}$: heat flow rate
u : specific internal energy
$\dot{U}A$: thermal conductance
T : temperature
q : vapor quality
p : pressure
ρ : density
$_{in}$: inlet
$_{out}$: outlet
$_{liq}$: saturated liquid
$_{vap}$: saturated vapor
$_{evap}$: evaporated liquid

4.2 Compressor

The equation set (4.2) describes the compressor model. It is assumed no holdup and a constant isentropic efficiency. The gas entering the inlet is superheated. It is not straight forward to implement the compressor to the cycle simulation cause the compressor power determines the thermodynamic properties to the rest of the process. To contradict this issue the inlet enthalpy is updated every time instant from the internal heat exchanger outlet and the compressor power is controlled such that the energy balance of the cycle is accomplished.

$$0 = \dot{m}_{in} - \dot{m}_{out} \tag{4.2a}$$
$$0 = \dot{W}_{shaft} + \dot{m}_{in} h_{in} - \dot{m}_{out} h_{out} \tag{4.2b}$$
$$s_s = s_{in} \tag{4.2c}$$
$$p_s = p_{out} \tag{4.2d}$$
$$\eta = \frac{h_s - h_{in}}{h_{out} - h_{in}} \tag{4.2e}$$
$$p_s = f(\rho_s, T_s) \tag{4.2f}$$
$$s_{in} = f(\rho_{in}, T_{in}) \tag{4.2g}$$
$$(\rho_s, T_s) = f(h_s, s_s) \tag{4.2h}$$
$$(\rho_{in}, T_{in}) = f(h_{in}, p_{in}) \tag{4.2i}$$
$$(\rho_{out}, T_{out}) = f(h_{out}, p_{out}) \tag{4.2j}$$

Where:

$\dot{m}$: mass flow rate
$\dot{W}$: Power
h : specific enthalpy
s : specific entropy
T : temperature
p : pressure
η : isentropic efficiency
ρ : density
$_{in}$: inlet
$_{out}$: outlet
$_{s}$: isentropic
$_{shaft}$: shaft work

4.3 Gas cooler

The associated gas cooler model is given in the equation set (4.3). The thermal conductance and ambient air properties are given as inputs, thus this condenser is easy to model. The gas cooler is divided into 6 control volumes for consistency with the MATLAB model, although this number is easy to alter.

$$0 = \dot{m}_{in} - \dot{m}_{out} \tag{4.3a}$$
$$0 = \dot{m}^{air}_{in} - \dot{m}^{air}_{out} \tag{4.3b}$$
$$0 = \dot{m}_{in} h_{in} - \dot{m}_{out} h_{out} - \dot{Q} \tag{4.3c}$$
$$\dot{Q} = \dot{m}^{air} C^{air}_{p} (T^{air}_{out} - T^{air}_{in}) \tag{4.3d}$$
$$\dot{Q} = \dot{U}A (T_{out} - T^{air}_{out}) \tag{4.3e}$$
$$p_{in} = p_{out} \tag{4.3f}$$
$$(\rho_{out}, T_{out}) = f(h_{out}, p_{out}) \tag{4.3g}$$

Where:

$\dot{m}$: mass flow rate
h : specific enthalpy
$\dot{U}A$: thermal conductance
$\dot{Q}$: heat flow rate
T : temperature
p : pressure
C_p : heat capacity at constant pressure
ρ : density
$_{in}$: inlet
$_{out}$: outlet
^{air} : ambient air

4.4 Internal heat exchanger

The equations used to model the internal heat exchanger is given in the equation set (4.4). The high pressure stream from the gas cooler outlet is heat exchanged with the low pressure stream from the evaporator outlet. Also the internal heat exchanger is divided into 6 control volumes for consistency with the MATLAB model, although this number is easy to alter.

$$0 = \dot{m}_{H,in} - \dot{m}_{H,out} \tag{4.4a}$$
$$0 = \dot{m}_{C,in} - \dot{m}_{C,out} \tag{4.4b}$$
$$0 = \dot{m}_{H,in} h_{H,in} - \dot{m}_{H,out} h_{H,out} - \dot{Q} \tag{4.4c}$$
$$0 = \dot{m}_{C,in} h_{C,in} - \dot{m}_{C,out} h_{C,out} + \dot{Q} \tag{4.4d}$$
$$\dot{Q} = \dot{U}A(T_{H,out} - T_{C,out}) \tag{4.4e}$$
$$p_{H,in} = p_{H,out} \tag{4.4f}$$
$$p_{C,in} = p_{C,out} \tag{4.4g}$$
$$(\rho_{H,out}, T_{H,out}) = f(h_{H,out}, p_{H,out}) \tag{4.4h}$$
$$(\rho_{C,out}, T_{C,out}) = f(h_{C,out}, p_{C,out}) \tag{4.4i}$$

Where:

$\dot{m}$: mass flow rate
h : specific enthalpy
$\dot{Q}$: heat flow rate
$\dot{U}A$: thermal conductance
T : temperature
p : pressure
ρ : density
$_{in}$: inlet
$_{out}$: outlet
$_{H}$: hot side
$_{C}$: cold side

4.5 Valve

The equations for modeling the valve are given in the equation set (4.5). The isenthalpic valve is quite easy to implement into the simulation model if the outlet mass flow rate, pressure drop, inlet pressure and inlet enthalpy are known inputs, which is the setup of the Modelica model. As mentioned earlier under the compressor model section, the enthalpy connecting the valve and the evaporator is indirectly controlled ensuring feasible energy transfer throughout the process by adjusting the compressor power ($\dot{W}_{shaft}$) which satisfies the hard constrained isentropic efficiency (η). In practice preferably the valve opening (z) is directly manipulated rather than the mass flow rate.

$$\dot{m}_{out} = Cvz\sqrt{(p_{in} - p_{out})\rho_{in} \cdot 1 \times 10^3\,\mathrm{Pa\,kPa^{-1}}} \tag{4.5a}$$

$$h_{in} = h_{out} \tag{4.5b}$$

$$(\rho_{in}, T_{in}) = f(h_{in}, p_{in}) \tag{4.5c}$$

Where:

$\dot{m}$: mass flow rate
Cv : cross section
z : valve opening
h : specific enthalpy
T : temperature
p : pressure
ρ : density
$_{in}$: inlet
$_{out}$: outlet

5 Results

5.1 The Modelica model construction

A model is not directly a result, but most of the time spended has been used to construct a model from scratch. Therefore the final model is assumed to take a part of the results.

The simulations are modified in one single class, the class at the highest level in the model hierarchy, where all parameters and initiate variables are set. This class is depicted below where all predetermined specifications and supposed optimal variables are specified. At a glance every process units are connected, as they should be, except for the specific enthalpy connecting the valve and the evaporator (line 24), and the specific enthalpy connecting the cold side of the internal heat exchanger and the compressor (line 33). The latter is in a form connected since the specific enthalpy at the compressor inlet is equal to the previous (last simulation interval) specific enthalpy at the internal heat exchanger outlet, hence it works for steady-state studies. The connection between the valve and the evaporator is somewhat difficult to keep balanced (respect to energy balance), since this is dependent on the compressor power. Solving this issue has been accomplished by utilization of a PI-controller (instance at line 8-9) to control the compressor power (line 42) for matching enthalpies between the valve outlet and the evaporator inlet.

```
1  model RefrigerationCycleCO2Simulation
2    Compressor COM(eta = 0.75);
3    GasCooler GCO(n = 6, Cp_air = 1.0);
4    InternalHEX IHX(n = 6);
5    Valve VAL(Cv = 0.000001205151054618652);
6    Evaporator EVP;
7    //Instances of the model units
8    PIController CtrWs(K = -0.0001, T = 0.0005, val = pre(VAL.hOut.h),
9    ref = EVP.hIn.h);
10   //Controller to satisfy the enthalpy balance throughout the valve
11   //(primary comparison point)
12 equation
13   connect(COM.mOut,GCO.HEX[1].mIn);
14   connect(COM.pOut,GCO.HEX[1].pIn);
15   connect(COM.hOut,GCO.HEX[1].hIn);
16   connect(GCO.HEX[GCO.n].mOut,IHX.HEX[1].mInH);
17   connect(GCO.HEX[GCO.n].pOut,IHX.HEX[1].pInH);
18   connect(GCO.HEX[GCO.n].hOut,IHX.HEX[1].hInH);
19   connect(IHX.HEX[IHX.n].mOutH,VAL.mIn);
20   connect(IHX.HEX[IHX.n].pOutH,VAL.pIn);
21   connect(IHX.HEX[IHX.n].hOutH,VAL.hIn);
22   connect(VAL.mOut,EVP.mIn);
23   connect(VAL.pOut,EVP.pIn);
24   //connect(VAL.hOut,EVP.hIn) not in use since it is controlled by the
```

```
  //compressor power;
  connect(EVP.mOut,IHX.HEX[IHX.n].mInC);
  connect(EVP.pOut,IHX.HEX[IHX.n].pInC);
  connect(EVP.hOut,IHX.HEX[IHX.n].hInC);
  connect(IHX.HEX[1].mOutC,COM.mIn);
  connect(IHX.HEX[1].pOutC,COM.pIn);
  //Connecting the mass flow rates, pressures and specific enthalpies between
  //all units
  COM.hIn.h = if initial() then -54.432 else pre(IHX.HEX[1].hOutC.h);
  //Connecting the specific enthalpy between internal heat exchanger and
  //compressor (Secondary comparison point)
  EVP.mIn.m_flow = 0.0254;
  //Mass flow rate
  EVP.T_air = 20 + 273.15;
  //Room temperature
  EVP.T_C = 15 + 273.15;
  //Evaporator temperature
  COM.Ws = 0.958268 + CtrWs.outCtr;
  //Compressor power
  GCO.HEX[GCO.n].mairIn.m_flow = 0.25;
  //Mass flow rate ambient air
  GCO.HEX[GCO.n].TairIn.T = 30 + 273.15;
  //Inlet ambient air temperature
  EVP.UA = 0.7999999999999875;
  //UA-value evaporator
  GCO.UA_tot = 0.7936318367211835;
  //UA-value gas cooler
  IHX.UA_tot = 0.1538926355428347;
  //UA-value internal heat exchanger
end RefrigerationCycleCO2Simulation;
```

The models which are instances in the simulation model above are not depicted here, but can be investigated at the internet [5] or in the appendix A.

5.2 Validation of the results

The Modelica model was made as an exact copy of the MATLAB model by Jensen. Despite identical models, there are two differences between the two models that may cause deviations. First, the Modelica model is based on mass unit, while the MATLAB model is based on molar unit. Second, the Modelica model are using the complete equation of state while the MATLAB model are using a modified equation of state. Nevertheless, both models refer to the same publisher of the thermodynamic equations, Span and Wagner, which are based on mass as the primary unit. From this fact should the Modelica model in principle be more precise compared to the MATLAB model, due to the absence of modified equations. How the two models conduct can be seen in Figure 5.1. From this p-h chart it is clear that the two models are not matching complete, which means that the Modelica model is infeasible with the pressure and

mass flow rate settings from the MATLAB model. The infeasibility is outlined in the compressor that is 138 % more effective to an ideal isentropic compressor. There are a major enthalpy gap of $-18.48\,\mathrm{kJ\,kg^{-1}}$ at the compressor outlet and two minor enthalpy gaps at the valve and at the compressor inlet of $-1.27\,\mathrm{kJ\,kg^{-1}}$ and $-0.99\,\mathrm{kJ\,kg^{-1}}$, respectively. The deviations are relative to the MATLAB model.

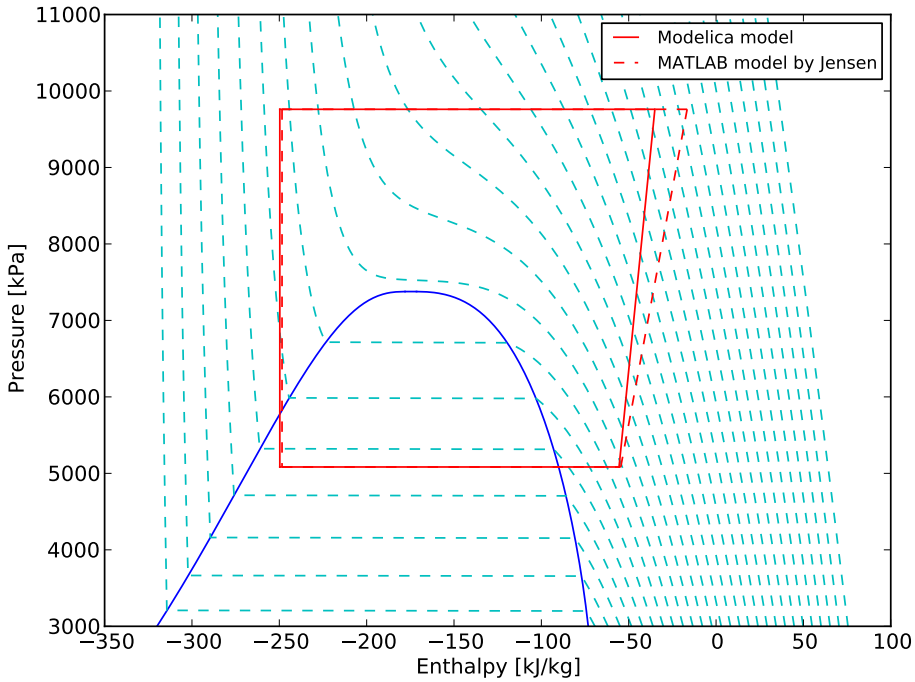

Figure 5.1: Pressure-enthalpy diagram which directly compares the Modelica model against the MATLAB model by Jensen when applying the same pressure and mass flow rate values. In this case the Modelica solution is infeasible since the isentropic efficiency in the compressor becomes 1.38. Isotherms are included in the chart raging from 270 K to 400 K displaced by 5 K.

All process units in Modelica calculates approximately the same values found in the MATLAB model except from the gas cooler where the relative deviations becomes significant and the feasibility breaks. This is seen in Figure 5.2. One plausible reason for this departed behavior is that the modified equation of state, which is used in the MATLAB model, cannot produce high accuracy results in the highly non-linear region above the critical point.

A feasible solution to the Modelica model is achieved by keeping the temperature in the evaporator constant at 15 °C (hence steady state) and adjusting the mass flow rate and the compressor power until convergence. Since the MATLAB model is optimized, then the optimal operation of the Modelica model would be somewhere close. The supposed optimal operation of the Modelica model is depicted in Figure 5.3. For this feasible model the new deviations are small and are hard to visualize in the figure. The new supposed optimal variables are listed in Table 5.1.

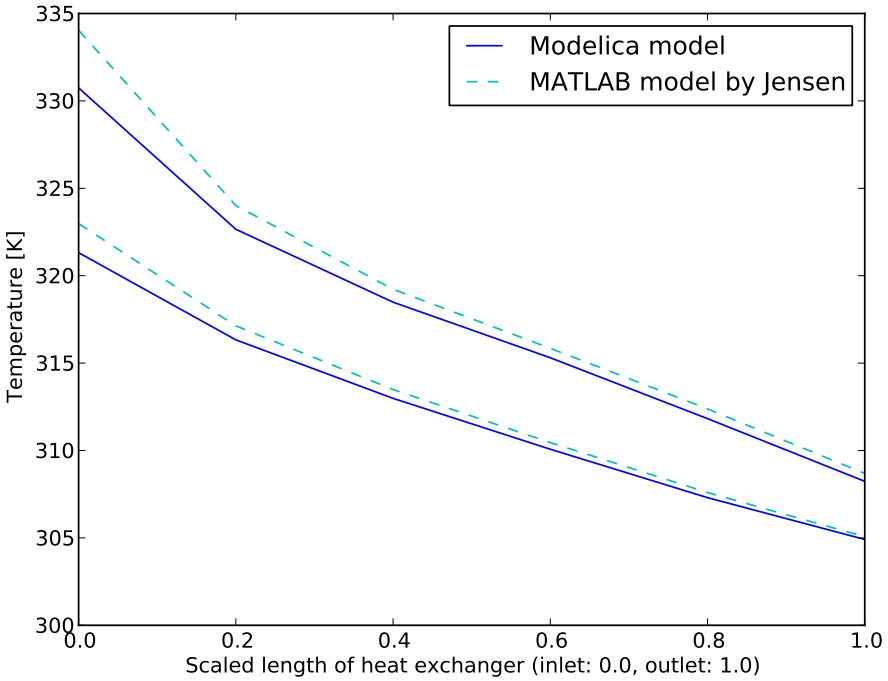

Figure 5.2: The temperature profile inside the gas cooler for both Modelica and MATLAB model is compared against each other when the exactly same fluid properties from the MATLAB model is implemented to the Modelica model. It is important to notice that this simulation for the Modelica model is infeasible, but the figure is meant to illustrate how the two models differ in calculations.

Table 5.1: New supposed optimized variables

Variable	**Symbol**	**Value**
Compressor power	$\dot{W}_{shaft}$ [kW]	0.958
Valve opening	z	0.345
High pressure side	p_4, p_5 and p_6 [kPa]	9714
Low pressure side	p_1, $p_{2'}$, p_2 and p_3 [kPa]	5087
Heat flow leaving the gas cooler	$\dot{Q}_{GCO}$ [kW]	4.958
Heat flow entering the evaporator	$\dot{Q}_{EVP}$ [kW]	4.000
Heat flow exchanged in the internal HEX	$\dot{Q}_{IHX}$ [kW]	0.907
Mass flow rate circulating	$\dot{m}$ [kg s^{-1}]	0.0254
Temperatures	T_1 [K]	288.15
	T_3 [K]	304.58
	T_4 [K]	362.60
	T_6 [K]	298.84

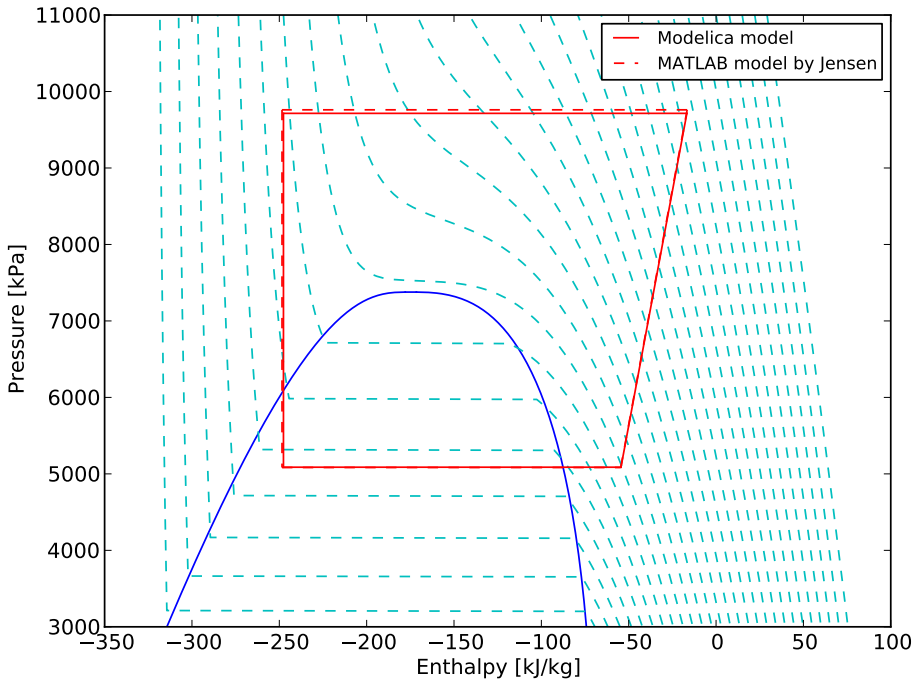

Figure 5.3: Pressure-enthalpy diagram which compares a feasible Modelica model against the MATLAB model by Jensen. Isotherms are included in the chart raging from 270 K to 400 K displaced by 5 K.

6 Discussion

Due to the fact that the Modelica model and the MATLAB model are not completely alike, there will be differences on how they behave. This was seen in the result section, Figure 5.2, where all process units calculated about the same values except for the gas cooler. A strategy could be to clone the original MATLAB model into the Modelica model, but that wouldn't be much of interest, besides, it is better to stay closest to the primary sources as possible.

The acausality Modelica provides has not been entirely adopted to the model. The acausal strategy means only use of equations (not assignments), connecting flow and potential variables, making partial models, inheritance of reusable models for multiple purposes etc. This CO_2 refrigeration cycle is more like a semi-causal type. It has connections between model components, but every model (i.e. class) are build in the causal way, therefore they cannot always be reused in other applications, e.g. there is not an easy way to replace the thermodynamic package if an other substance is desirable. Reusability is an important feature in Modelica, which means that large models shall depend on smaller models or other class types that are independent. Developing libraries that contain such reusable components are of great interest when fundamental investigations are to be made and besides, reconfigurations of complete cycles are then made simpler due to the connection principle in Modelica. Papers that presents the benefits, opportunities and development of implementing Modelica libraries for simulation of refrigeration systems is given by Pfafferott and Schmitz [6] and Schmitz et al. [7].

The ideal modeling approach is to code the governing equations, both for the process units and the thermodynamics, the acausal way. This method was also tried, but turned unsuccessful. The main problem is the equation of state, which has a lot of terms and is highly non-linear, including the derivatives. An another approach to meet the thermodynamic requirement is to develop lookup tables. The simulation time will then drastic decrease, due to the function calls and iterations avoidance. The lookup table approach have been tried and was successful. Anyhow the final model ended using the thermodynamic equation of state by solving density and temperature explicitly. The reason for this was a failure attempt trying to optimize the cycle when it was using lookup tables. Lookup tables was not directly the source of failure, but use of external functions was only partially supported by the JModelica[2] compiler and the interpolation in the lookup tables was performed by use of an external C-function. Hence, the lookup tables was rejected and replaced by the solution of using the EOS solving density and temperature explicitly. Yet another opportunity to deal with the thermodynamics is to use existing, freely available libraries like ThermoFluid [8], which also uses the high accuracy CO_2 EOS from Span and Wagner [3]. The downside is that ThermoFluid is no longer active developed, but it could be worth a try anyway.

The intention was to optimize the final model by use of the JModelica platform, but some

[2]Extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems

constraints were met. The model has to be rewritten before an attempt to optimization can be done. The reason is that JModelica has limitations to the features available in the Modelica language. Some of these non-supporting features are while- and if-statements, string arguments in functions, built in functions like initial() and pre() etc.

6.1 Suggestions for future work

- Rewrite the model code in a more acausal structure and make it simpler to simulate. The simulation difficulties so far are the two comparison points that have to be controlled every simulation interval to make sure the system is conserved.
- Implement another approach respect to the current thermodynamics. Could there be a opportunity to implement the equation of state the acausal way and not by causal functions (assignments)? If that is possible, and stable off course, this solution would certainly improve simulation flexibility.
- Keep the thermodynamics as implemented, but develop a reliable guess routine for the density and the temperature in all phases.
- Optimizing the model when the first bulletin in this list is solved.
- Derive applicable control structures for self-optimizing control.

7 Conclusion

A refrigeration cycle based on CO_2, which operates trans-critical, has been modeled in the equation based, object-oriented language Modelica. The cycle was based on the same model presented in the PhD thesis by Jensen [1]. The modeled cycle turned out to differ from the model by Jensen, but the differences was minor. A test simulation with the exact same values of the fluid properties (mass flow rate, high and low pressure) revealed a non-consistency relationship between the modified EOS used in the model by Jensen and the full EOS used in the Modelica model. This could be seen in Figure 5.1 and Figure 5.2 where the latter shows accumulating deviations of magnificence. A feasible steady-state solution to the Modelica model was obtained, but not considered as optimal. The model has to be rewritten before it can support all the dependencies in the JModelica compiler and, from then, be optimized.

List of symbols and abbreviations

Symbol	Dimension	Description
Uppercase		
$\mathbf{J_f}$	-	Jacobian matrix for function vector **f**
A	-	Exponent in EOS
B	-	Exponent in EOS
C	-	Exponent in EOS
C	[kJ kg^{-1} K^{-1}]	Heat capacity
Cv	[m^2]	Choke valve
D	-	Exponent in EOS
$\dot{Q}$	[kW]	Heat flow rate
R	[kJ kg^{-1} K^{-1}]	Gas constant
T	[K]	Temperature
U	[kJ]	Internal energy
$\dot{U}A$	[kW K^{-1}]	Thermal conductance
$\dot{W}$	[kW]	Power
Lowercase		
f	-	Vector of functions
a	-	Coefficient in EOS
a	-	Exponent in EOS
b	-	Exponent in EOS
c	-	Exponent in EOS
d	-	Exponent in EOS
d	-	Differential
f	-	A function
h	[kJ kg^{-1}]	Specific enthalpy
m	[kg]	Mass
$\dot{m}$	[kg s^{-1}]	Mass flow rate
n	-	Coefficient in EOS
n	-	Number of control volumes
p	[kPa]	Pressure
q	-	Vapor quality
s	[kJ kg^{-1} K^{-1}]	Specific entropy
t	-	Exponent in EOS
t	[s]	Time
u	[kJ kg^{-1}]	Specific internal energy
z	[m^2]	Valve opening
Greek letters		

Continued on next page

Continued from previous page

Symbol	Dimension	Description
α	-	Exponent in EOS
β	-	Exponent in EOS
γ	-	Exponent in EOS
δ	-	Dimensionless density
ϵ	-	Exponent in EOS
η	-	Isentropic efficiency
θ	-	Exponent in EOS
ρ	$[\mathrm{kg\,m^{-3}}]$	Density
τ	-	Dimensionless temperature
ϕ	-	Dimensionless Helmholtz energy
Superscript		
air	-	Either room or ambient air
k	-	Iteration index
r	-	Residual part
0	-	Ideal part
Subscript		
C	-	Cold side
COM	-	Compressor
EVP	-	Evaporator
GCO	-	Gas cooler (condenser)
H	-	Hot side
IHX	-	Internal heat exchanger
VAL	-	Valve
$evap$	-	Evaporated
i	-	Integer
in	-	Inlet
liq	-	Liquid
$loss$	-	Loss (i.e. heat loss in the room-ambient interphase)
out	-	Outlet
p	-	Constant pressure
s	-	Isentropic
$shaft$	-	Shaft work (compressor power)
vap	-	Vapor
δ	-	Partial derivative with respect to dimensionless density
τ	-	Partial derivative with respect to dimensionless temperature
1	-	Connection point between valve and evaporator

Continued on next page

Continued from previous page

Symbol	Dimension	Description
1	-	Function index
2	-	Connection point between evaporator and internal heat exchanger
2	-	Function index
3	-	Connection point between internal heat exchanger and compressor
4	-	Connection point between compressor and gas cooler
5	-	Connection point between gas cooler and internal heat exchanger
6	-	Connection point between internal heat exchanger and valve

References

[1] J. B. Jensen, *Optimal Operation of Refrigeration Cycles*. PhD thesis, Norwegian University of Science and Technology, May 2008.

[2] P. Fritzson, *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. A John Wiley & Sons, Inc., Publication, 2011.

[3] R. Span and W. Wagner, "A new equation of state for carbon dioxide covering the fluid region from the triple-point temperature to 1100 k at pressures up to 800 mpa," *J. Phys. Chem. Ref. Data*, vol. 25, no. 6, p. 88, 1996.

[4] T. Andresen, *Mathematical modeling of CO2 based heat pumping systems*. PhD thesis, Norwegian University of Science and Technology, September 2009.

[5] N. A. Susort, "Refrigeration cycle co2 - modelica model code." Internet, December 2012. http://folk.ntnu.no/nilsarsu/Projects/TKP4550/.

[6] *Modelling and transient simulation of CO2-refrigeration systems with Modelica*, vol. 27, Elsevier Ltd and IIR, 2004.

[7] G. Schmitz, ed., *System Simulation of Automotive Refrigeration Cycles*, 4th, 2005.

[8] F. W. J. Eborn, H. Tummescheit, *ThermoFluid A Thermo-Hydraulic Library in Modelica*, 2001.

A Refrigeration cycle CO_2 model code written in Modelica

A.1 Connectors

```
connector ConnectFlow
  import RefrigerationCycleCO2.Units.MassFlowRate;
  MassFlowRate m_flow;
end ConnectFlow;

connector ConnectPressure
  import RefrigerationCycleCO2.Units.Pressure;
  Pressure p;
end ConnectPressure;

connector ConnectEnthalpy
  import RefrigerationCycleCO2.Units.SpecificEnthalpy;
  SpecificEnthalpy h;
end ConnectEnthalpy;

connector ConnectTemperature
  import RefrigerationCycleCO2.Units.Temperature;
  Temperature T;
end ConnectTemperature;
```

A.2 Units used in the models

```
package Units
  type Area = Real(final quantity = "Area", final unit = "m2", min = 0);
  type CrossSection = Real(final quantity = "Area", final unit = "m2", min = 0);
  type Density = Real(final quantity = "Density", final unit = "kg/m3",
                  min = 0.000001, max = 30000.0);
  type Efficiency = Real(final quantity = "Efficiency", final unit = "1",
                      min = 0, max = 1);
  type HeatFlowRate = Real(final quantity = "HeatFlowRate", final unit = "kW");
  type InternalEnergy = Real(final quantity = "Energy", final unit = "kJ");
  type Mass = Real(quantity = "Mass", final unit = "kg", min = 0);
  type MassFlowRate = Real(quantity = "MassFlowRate", final unit = "kg/s");
  type Pressure = Real(final quantity = "Pressure", final unit = "kPa", min = 0,
                   max = 1000000.0, nominal = 100.0, start = 100.0);
  type PressureDrop = Real(final quantity = "PressureDrop", final unit = "kPa",
                         min = 0);
  type Power = Real(final quantity = "Power", final unit = "kW");
  type Quality = Real(final quantity = "Quality", final unit = "1", min = 0,
                  max = 1);
  type SpecificEnthalpy = Real(final quantity = "SpecificEnthalpy",
                           final unit = "kJ/kg");
```

```
  type SpecificEntropy = Real(final quantity = "SpecificEntropy",
                              final unit = "kJ/(kg.K)");
  type SpecificHeatCapacity = Real(final quantity = "SpecificHeatCapacity",
                                   final unit = "kJ/(kg.K)");
  type SpecificInternalEnergy = Real(final quantity = "SpecificEnergy",
                                     final unit = "kJ/kg");
  type Temperature = Real(final quantity = "Temperature", final unit = "K",
                          min = 1, max = 6000, start = 288.15, nominal = 300);
  type ThermalConductance = Real(final quantity = "ThermalConductance",
                                 final unit = "kW/K");
  type ValveOpening = Real(final quantity = "ValveOpening", final unit = "1",
                           min = 0, max = 1);
  type Volume = Real(final quantity = "Volume", final unit = "m3", min = 0);
end Units;
```

A.3 Evaporator

```
model Evaporator
  import RefrigerationCycleCO2.Units.*;
  RefrigerationCycleCO2.ConnectFlow mIn,mOut;
  //Mass flow rate instances for inlet and outlet
  RefrigerationCycleCO2.ConnectPressure pIn,pOut;
  //Pressure instances for inlet and outlet
  RefrigerationCycleCO2.ConnectEnthalpy hIn,hOut;
  //Specific enthalpy instances for inlet and outlet
  Density rho_l "Liquid density";
  Density rho_v "Vapor density";
  HeatFlowRate Q "Heat transferred from room to evaporator";
  InternalEnergy U "Internal energy";
  Mass m "Mass inventory";
  MassFlowRate m_flow_vap "Liquid evaporation rate";
  Pressure p_l "Liquid pressure";
  Pressure p_v "Vapor pressure";
  SpecificEnthalpy h_l "Liquid specific enthalpy";
  SpecificEnthalpy h_v "Vapor specific enthalpy";
  SpecificInternalEnergy u "Specific internal energy";
  Temperature T_C "Temperature at the cold side (the CO2 side)";
  Temperature T_air "Room temperature (the hot side)";
  ThermalConductance UA "Heat transfer rate per temperature difference";
  Quality q "Vapor quality";
equation
  der(m) = mIn.m_flow - mOut.m_flow;
  //Mass balance
  m_flow_vap = Q / (h_v - h_l);
  //Liquid evaporation calculated from enthalpy of vaporization
  0 = q * mIn.m_flow - mOut.m_flow + m_flow_vap;
  //All gas is leaving the evaporator
  der(U) = Q + mIn.m_flow * hIn.h - mOut.m_flow * hOut.h;
```

```
  //Energy balance
  der(U) = der(m) * u;
  //Relation between accumulated mass and accumulated internal energy
  q = (hIn.h - h_l) / (h_v - h_l);
  //Definition of vapor quality
  Q = UA * (T_air - T_C);
  //Heat transferred
  pIn.p = pOut.p;
  //No pressure drop
  pOut.p = p_v;
  //Oulet pressure equal to vapor pressure
  hOut.h = h_v;
  //Outlet specific enthalpy equal to specific enthalpy for saturated gas
  (p_v,rho_v,h_v) = RefrigerationCycleCO2.SpanWagnerEOS.SatPropGivenpLUT(T =
                    T_C, phase = "v");
  //Saturation properties for gas given temperature
  (p_l,rho_l,h_l) = RefrigerationCycleCO2.SpanWagnerEOS.SatPropGivenpLUT(T =
                    T_C, phase = "l");
  //Saturation properties for liquid given temperature
  u = RefrigerationCycleCO2.SpanWagnerEOS.InternalEnergy(rho = rho_l, T = T_C);
  //Specific internal energy given liquid density and temperature
  assert(q > 0, "Vapor quality is below zero");
  //The model is considering only two phase fluid at the inlet
end Evaporator;
```

A.4 Valve

```
model Valve
  import RefrigerationCycleCO2.Units.*;
  RefrigerationCycleCO2.ConnectFlow mIn,mOut;
  //Mass flow rate instances for inlet and outlet
  RefrigerationCycleCO2.ConnectPressure pIn,pOut;
  //Pressure instances for inlet and outlet
  RefrigerationCycleCO2.ConnectEnthalpy hIn,hOut;
  //Specific enthalpy instances for inlet and outlet
  parameter CrossSection Cv "Cross section of valve";
  Density rho_in "Inlet density";
  PressureDrop Dp "Inlet pressure subtracted for outlet pressure";
  Temperature T_in "Inlet temperature";
  ValveOpening z "Manipulated variable";
equation
  mOut.m_flow = Cv * z * sqrt(Dp * rho_in * 1000);
  //Mass balance
  Dp = pIn.p - pOut.p;
  //Pressure drop
  hIn.h = hOut.h;
  //Enthalpic
  (rho_in,T_in) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hIn.h,
```

```
                    p = pIn.p, rho0 = 815, T0 = 298);
  //Inlet density and temperature given inlet specific enthalpy and pressure
end Valve;
```

A.5 Internal heat exchanger

```
model PartialInternalHEX
  import RefrigerationCycleCO2.Units.*;
  RefrigerationCycleCO2.ConnectFlow mInH,mOutH,mInC,mOutC;
  //Mass flow rate instances for inlet and outlet, hot and cold side
  RefrigerationCycleCO2.ConnectPressure pInH,pOutH,pInC,pOutC;
  //Pressure instances for inlet and outlet, hot and cold side
  RefrigerationCycleCO2.ConnectEnthalpy hInH,hOutH,hInC,hOutC;
  //Specific enthalpy instances for inlet and outlet, hot and cold side
  Density rho_C "Outlet density cold side";
  Density rho_H "Outlet density hot side";
  HeatFlowRate Q "Heat flow rate from hot to cold side";
  Temperature T_H "Outlet temperature hot side";
  Temperature T_C "Outlet temperature cold side";
  ThermalConductance UA "UA-value for one single control volume";
equation
  0 = mInH.m_flow - mOutH.m_flow;
  //Mass balance hot side
  0 = mInC.m_flow - mOutC.m_flow;
  //Mass balance cold side
  0 = -Q + mInH.m_flow * hInH.h - mOutH.m_flow * hOutH.h;
  //Energy balance hot side
  0 = Q + mInC.m_flow * hInC.h - mOutC.m_flow * hOutC.h;
  //Energy balance cold side
  Q = UA * (T_H - T_C);
  //Heat transfer balance
  pInH.p = pOutH.p;
  //No pressure drop hot side
  pInC.p = pOutC.p;
  //No pressure drop cold side
  (rho_H,T_H) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hOutH.h,
                p = pOutH.p, rho0 = 745, T0 = 305);
  //Outlet density and temperature for the hot side given specific enthalpy and pressure
  (rho_C,T_C) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hOutC.h,
                p = pOutC.p, rho0 = 140, T0 = 298);
  //Outlet density and temperature for the cold side given specific enthalpy and pressure
end PartialInternalHEX;
```

```
model InternalHEX
  import RefrigerationCycleCO2.Units.*;
  PartialInternalHEX HEX[n](each UA = UA_tot / n);
  //Instances of the internal heat exchanger model
```

```
  parameter Integer n "Number of control volumes";
  ThermalConductance UA_tot "Total UA-value for the entire heat exchanger";
equation
  for i in 1:n - 1 loop
  connect(HEX[i].mOutH,HEX[i + 1].mInH);
  connect(HEX[i].pOutH,HEX[i + 1].pInH);
  connect(HEX[i].hOutH,HEX[i + 1].hInH);
  connect(HEX[i].mInC,HEX[i + 1].mOutC);
  connect(HEX[i].pInC,HEX[i + 1].pOutC);
  connect(HEX[i].hInC,HEX[i + 1].hOutC);

  end for;
  /*Connecting all mass flow rates, pressures and specific enthalpies, both hot
  and cold side, between the control volumes inside the internal heat exchanger */
end InternalHEX;
```

A.6 Gas cooler (condenser)

```
model PartialGasCooler
  import RefrigerationCycleCO2.Units.*;
  RefrigerationCycleCO2.ConnectFlow mIn,mOut,mairIn,mairOut;
  //Mass flow rate instances for inlet and outlet, hot and cold side
  RefrigerationCycleCO2.ConnectPressure pIn,pOut;
  //Pressure instances for inlet and outlet, only hot side
  RefrigerationCycleCO2.ConnectEnthalpy hIn,hOut;
  //Specific enthalpy instances for inlet and outlet, only hot side
  RefrigerationCycleCO2.ConnectTemperature TairIn,TairOut;
  //Temperature instances for inlet and outlet, only cold side
  parameter SpecificHeatCapacity Cp_air "Heat capacity ambient air";
  Density rho_H "Outlet density hot side";
  HeatFlowRate Q "Heat flow rate from hot to cold side";
  Temperature T_H "Outlet temperature hot side";
  ThermalConductance UA "UA-value for one single control volume";
equation
  0 = mIn.m_flow - mOut.m_flow;
  //Mass balance hot side
  0 = mairIn.m_flow - mairOut.m_flow;
  //Mass balance cold side
  0 = -Q - mOut.m_flow * hOut.h + mIn.m_flow * hIn.h;
  //Energy balance hot side
  Q = mairIn.m_flow * Cp_air * (TairOut.T - TairIn.T);
  //Energy balance cold side
  Q = UA * (T_H - TairOut.T);
  //Heat transfer balance
  pIn.p = pOut.p;
  //No pressure drop
  (rho_H,T_H) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hOut.h,
                p = pOut.p, rho0 = 558, T0 = 315);
```

```
  //Outlet density and temperature hot side given specific enthalpy and pressure
end PartialGasCooler;
```

```
model GasCooler
  import RefrigerationCycleCO2.Units.*;
  PartialGasCooler HEX[n](each UA = UA_tot / n, each Cp_air = Cp_air);
  //Instances of the gas cooler model
  parameter Integer n "Number of control volumes";
  parameter SpecificHeatCapacity Cp_air "Heat capacity for ambient air";
  ThermalConductance UA_tot "Total UA-value for the entire heat exchanger";
equation
  for i in 1:n - 1 loop
  connect(HEX[i].mOut,HEX[i + 1].mIn);
  connect(HEX[i].pOut,HEX[i + 1].pIn);
  connect(HEX[i].hOut,HEX[i + 1].hIn);
  connect(HEX[i].mairIn,HEX[i + 1].mairOut);
  connect(HEX[i].TairIn,HEX[i + 1].TairOut);

  end for;
  /*Connecting all mass flow rates, pressures and specific enthalpies, both hot
  and cold side, between the control volumes inside the gas cooler */
end GasCooler;
```

A.7 Compressor

```
model Compressor
  import RefrigerationCycleCO2.Units.*;
  RefrigerationCycleCO2.ConnectFlow mIn,mOut;
  //Mass flow rate instances for inlet and outlet
  RefrigerationCycleCO2.ConnectPressure pIn,pOut;
  //Pressure instances for inlet and outlet
  RefrigerationCycleCO2.ConnectEnthalpy hIn,hOut;
  //Specific enthalpy instances for inlet and outlet
  parameter Efficiency eta "Isentropic efficiency";
  Density rho_in "Inlet density";
  Density rho_s "Outlet density for isentropic compressor";
  Density rho_out "Outlet density";
  Power Ws "Compressor power";
  Pressure p_s "Outlet pressure for isentropic compressor";
  SpecificEnthalpy h_s "Outlet specific enthalpy for isentropic compressor";
  SpecificEntropy s_in "Inlet specific entropy";
  SpecificEntropy s_s "Outlet specific entropy for isentropic compressor";
  Temperature T_in "Inlet temperature";
  Temperature T_s "Outlet temperature for isentropic compressor";
  Temperature T_out "Outlet temperature";
equation
  0 = mIn.m_flow - mOut.m_flow;
```

```
  //Mass balance
  0 = Ws + mIn.m_flow * hIn.h - mOut.m_flow * hOut.h;
  //Energy balance
  eta * (mOut.m_flow * hOut.h - mIn.m_flow * hIn.h) = mOut.m_flow * h_s
                                                      - mIn.m_flow * hIn.h;
  //Isentropic efficiency in terms of enthalpies
  s_s = s_in;
  //Isentropic compressor
  pOut.p = p_s;
  //Outlet pressure is equal for both isentropic case and real case
  p_s = RefrigerationCycleCO2.SpanWagnerEOS.Pressure(rho = rho_s, T = T_s);
  //Outlet pressure given density and temperature
  s_in = RefrigerationCycleCO2.SpanWagnerEOS.Entropy(rho = rho_in, T = T_in);
  //Inlet specific entropy given density and temperature
  (rho_s,T_s) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhs(h = h_s,
                s = s_s, rho0 = 229, T0 = 346);
  //Outlet density and temperature for isentropic compressor given specific enthalpy and spe
  (rho_out,T_out) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hOut.h,
                    p = pOut.p, rho0 = 196, T0 = 363);
  //Outlet density and temperature given specific enthalpy and pressure
  (rho_in,T_in) = RefrigerationCycleCO2.SpanWagnerEOS.PropGivenhp(h = hIn.h,
                  p = pIn.p, rho0 = 125, T0 = 305);
  //Inlet density and temperature given specific enthalpy and pressure
end Compressor;
```

A.8 Refrigeration Cycle CO_2

```
model RefrigerationCycleCO2Simulation
  Compressor COM(eta = 0.75);
  GasCooler GCO(n = 6, Cp_air = 1.0);
  InternalHEX IHX(n = 6);
  Valve VAL(Cv = 0.00000120515105461865 2);
  Evaporator EVP;
  //Instances of the model units
  PIController CtrWs(K = -0.0001, T = 0.0005, val = pre(VAL.hOut.h),
  ref = EVP.hIn.h);
  //Controller to satisfy the enthalpy balance throughout the valve
  //(primary comparison point)
equation
  connect(COM.mOut,GCO.HEX[1].mIn);
  connect(COM.pOut,GCO.HEX[1].pIn);
  connect(COM.hOut,GCO.HEX[1].hIn);
  connect(GCO.HEX[GCO.n].mOut,IHX.HEX[1].mInH);
  connect(GCO.HEX[GCO.n].pOut,IHX.HEX[1].pInH);
  connect(GCO.HEX[GCO.n].hOut,IHX.HEX[1].hInH);
  connect(IHX.HEX[IHX.n].mOutH,VAL.mIn);
  connect(IHX.HEX[IHX.n].pOutH,VAL.pIn);
  connect(IHX.HEX[IHX.n].hOutH,VAL.hIn);
```

```
  connect(VAL.mOut,EVP.mIn);
  connect(VAL.pOut,EVP.pIn);
  //connect(VAL.hOut,EVP.hIn) not in use since it is controlled by the
  //compressor power;
  connect(EVP.mOut,IHX.HEX[IHX.n].mInC);
  connect(EVP.pOut,IHX.HEX[IHX.n].pInC);
  connect(EVP.hOut,IHX.HEX[IHX.n].hInC);
  connect(IHX.HEX[1].mOutC,COM.mIn);
  connect(IHX.HEX[1].pOutC,COM.pIn);
  //Connecting the mass flow rates, pressures and specific enthalpies between
  //all units
  COM.hIn.h = if initial() then -54.432 else pre(IHX.HEX[1].hOutC.h);
  //Connecting the specific enthalpy between internal heat exchanger and
  //compressor (Secondary comparison point)
  EVP.mIn.m_flow = 0.0254;
  //Mass flow rate
  EVP.T_air = 20 + 273.15;
  //Room temperature
  EVP.T_C = 15 + 273.15;
  //Evaporator temperature
  COM.Ws = 0.958268 + CtrWs.outCtr;
  //Compressor power
  GCO.HEX[GCO.n].mairIn.m_flow = 0.25;
  //Mass flow rate ambient air
  GCO.HEX[GCO.n].TairIn.T = 30 + 273.15;
  //Inlet ambient air temperature
  EVP.UA = 0.7999999999999875;
  //UA-value evaporator
  GCO.UA_tot = 0.7936318367211835;
  //UA-value gas cooler
  IHX.UA_tot = 0.1538926355428347;
  //UA-value internal heat exchanger
end RefrigerationCycleCO2Simulation;
```

B Thermodynamic considerations

B.1 Modelica sample for the dimensionless Helmholtz function

The ideal and residual Helmholtz functions, including the derivatives, are too extensive to include in the appendix, but they are available at the internet [5]. Only a representative sample on how the functions are written is given in the script below for the residual of the dimensionless Helmholtz energy (ϕ^r).

```
function phir
  input Real rho;
  input Real T;
  output Real phir;
```

```
protected
  constant Real rhoc = 467.606;
  constant Real Tc = 304.1282;
  parameter Real[42] n =
{0.38856823203161,2.938547594274,-5.5867188534934,-0.76753199592477,
0.31729005580416,0.54803315897767,0.12279411220335,2.165896154322,1.
5841735109724,-0.23132705405503,0.058116916431436,-0.55369137205382,
0.48946615909422,-0.024275739843501,0.062494790501678,-0.12175860225246,
-0.37055685270086,-0.016775879700426,-0.11960736637987,-0.045619362508778,
0.035612789270346,-0.0074427727132052,-0.0017395704902432,-0.021810121289527,
0.024332166559236,-0.037440133423463,0.14338715756878,-0.13491969083286,
-0.02315122505348,0.012363125492901,0.002105832197294,-0.00033958519026368,
0.0055993651771592,-0.00030335118055646,-213.6548868832,26641.569149272,
-24027.212204557,-283.41603423999,212.47284400179,-0.66642276540751,
0.72608632349897,0.055068668612842};
  parameter Real[39] d = {1,1,1,1,2,2,3,1,2,4,5,5,5,6,6,6,1,1,4,4,4,7,8,
                          2,3,3,5,5,6,7,8,10,4,8,2,2,2,3,3};
  parameter Real[39] t =
{0.0,0.75,1.0,2.0,0.75,2.0,0.75,1.5,1.5,2.5,0.0,1.5,2.0,0.0,1.0,2.0,3.0,
6.0,3.0,6.0,8.0,6.0,0.0,7.0,12.0,16.0,22.0,24.0,16.0,24.0,8.0,2.0,28.0,
14.0,1.0,0.0,1.0,3.0,3.0};
  parameter Real[27] c = {1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,4,4,4,4,4,4,5,6};
  parameter Real[5] alpha = {25.0,25.0,25.0,15.0,20.0};
  parameter Real[5] beta = {325.0,300.0,300.0,275.0,275.0};
  parameter Real[5] gamma = {1.16,1.19,1.19,1.25,1.22};
  parameter Real[5] epsilon = {1.0,1.0,1.0,1.0,1.0};
  parameter Real[3] a = {3.5,3.5,3.0};
  parameter Real[3] b = {0.875,0.925,0.875};
  parameter Real[3] BETA = {0.3,0.3,0.3};
  parameter Real[3] A = {0.7,0.7,0.7};
  parameter Real[3] B = {0.3,0.3,1.0};
  parameter Real[3] C = {10.0,10.0,12.5};
  parameter Real[3] D = {275.0,275.0,275.0};
  Real delta;
  Real tau;
  Real sum1;
  Real sum2;
  Real sum3;
  Real sum4;
  Real PSI;
  Real THETA;
  Real DELTA;
  Real dummy;
algorithm
  delta:=rho / rhoc;
  tau:=Tc / T;
  sum1:=0;
  sum2:=0;
  sum3:=0;
  sum4:=0;
```

```
  for i in 1:7 loop
      sum1:=sum1 + n[i] * delta ^ d[i] * tau ^ t[i];
  end for;
  for i in 8:34 loop
      sum2:=sum2 + n[i] * delta ^ d[i] * tau ^ t[i] * exp(-delta ^ c[i - 7]);
  end for;
  for i in 35:39 loop
      sum3:=sum3 + n[i] * delta ^ d[i] * tau ^ t[i] * exp(-alpha[i - 34]
           * (delta - epsilon[i - 34]) ^ 2 - beta[i - 34]
* (tau - gamma[i - 34]) ^ 2);
  end for;
  for i in 40:42 loop
      PSI:=exp(-C[i - 39] * (delta - 1) ^ 2 - D[i - 39] * (tau - 1) ^ 2);
    dummy:=(delta - 1) ^ 2;
    THETA:=1 - tau + A[i - 39] * dummy ^ (1 / (2 * BETA[i - 39]));
    DELTA:=THETA ^ 2 + B[i - 39] * (delta - 1) ^ (2 * a[i - 39]);
    sum4:=sum4 + n[i] * DELTA ^ b[i - 39] * delta * PSI;
  end for;
  phir:=sum1 + sum2 + sum3 + sum4;
end phir;
```

B.2 Thermodynamic properties as function of specific enthalpy and pressure

Below is the script calculating density and temperature as a function of specific enthalpy and pressure included. The same scripting is done for the entropy-enthalpy case, but is not included in appendix.

```
function PropGivenhp
  input Real h;
  input Real p;
  input Real rho0;
  input Real T0;
  output Real rho;
  output Real T;
protected
  constant Real rhoc = 467.606;
  constant Real Tc = 304.1282;
  constant Real R = 0.1889241;
  parameter Real tol = 0.000001;
  Real f1;
  Real f2;
  Real f1tau;
  Real f2tau;
  Real f1delta;
  Real f2delta;
  Real delta;
  Real tau;
```

```
  Real phirdelta;
  Real phi0tau;
  Real phirtau;
  Real phirdeltatau;
  Real phirdeltadelta;
  Real phi0tautau;
  Real phirtautau;
  Real phi0deltatau;
  Real det;
  Real rhoi;
  Real Ti;
algorithm
  rhoi:=rho0;
  Ti:=T0;
  delta:=rho0 / rhoc;
  tau:=Tc / T0;
  f1:=1;
  f2:=1;
  while (abs(f1) > tol and abs(f2) > tol) loop
      phirdelta:=RefrigerationCycleCO2.SpanWagnerEOS.phirdelta(rho = rhoi,
                 T = Ti);
    phi0tau:=RefrigerationCycleCO2.SpanWagnerEOS.phi0tau(T = Ti);
    phirtau:=RefrigerationCycleCO2.SpanWagnerEOS.phirtau(rho = rhoi, T = Ti);
    phirdeltatau:=RefrigerationCycleCO2.SpanWagnerEOS.phirdeltatau(rho = rhoi,
                  T = Ti);
    phirdeltadelta:=RefrigerationCycleCO2.SpanWagnerEOS.phirdeltadelta(rho =
                    rhoi, T = Ti);
    phi0tautau:=RefrigerationCycleCO2.SpanWagnerEOS.phi0tautau(T = Ti);
    phirtautau:=RefrigerationCycleCO2.SpanWagnerEOS.phirtautau(rho = rhoi,
                T = Ti);
    phi0deltatau:=RefrigerationCycleCO2.SpanWagnerEOS.phi0deltatau();
    //Calculating the necessary terms
    f1:=delta / tau * (1 + delta * phirdelta) - p / (rhoc * R * Tc);
    f2:=1 + delta * phirdelta + tau * (phi0tau + phirtau) - (tau * h) / (R * Tc);
    //Residual functions
    rho:=delta * rhoc;
    T:=Tc / tau;
    //The returned output
    f1tau:=((1 + delta * phirdelta) * (-delta)) / tau ^ 2 + delta / tau * delta
           * phirdeltatau;
    f1delta:=(1 + 2 * delta * phirdelta + delta ^ 2 * phirdeltadelta) / tau;
    f2tau:=delta * phirdeltatau + phi0tau + phirtau + tau * (phi0tautau
           + phirtautau) - h / (R * Tc);
    f2delta:=phirdelta + delta * phirdeltadelta + tau * (phi0deltatau
             + phirdeltatau);
    //Derivatives inside the Jacobian matrix
    det:=f1tau * f2delta - f1delta * f2tau;
    //Determinant of the Jacobian matrix
    tau:=tau - 1 / det * (f1 * f2delta - f2 * f1delta);
    delta:=delta - 1 / det * (f2 * f1tau - f1 * f2tau);
```

```
71     //One step in Newton-Raphson method
72     rhoi:=delta * rhoc;
73     Ti:=Tc / tau;
74     //Update the iteration variables
75   end while;
76 end PropGivenhp;
```