T H E R M O P T I M ®

OFF-DESIGN OPERATION

DRIVER FOR REFRIGERATION CYCLE

WITH PRESSURE DROPS AND

SET REFRIGERANT CHARGE

RESOLUTION WITH MINPACK 1

VERSIONS JAVA 1.7 OR 2.7

© R. GICQUEL AUGUST 2009

CONTENTS

© R. GICQUEL 2009. All Rights Reserved. This document may not be reproduced in part or in whole without the author's express written consent, except for the licensee's personal use and solely in accordance with the contractual terms indicated in the software license agreement.

Information in this document is subject to change without notice and does not represent a commitment on the part of the author.

1 Introduction - prerequisite

The purpose of this notice is to allow a developer to become familiar with writing a driver for studying the off-design behavior of a refrigeration cycle with Thermoptim. We assume you already know well Thermoptim and its external class mechanism, and that you are acquainted with all four volumes of the software reference manual. We recommend that you also refer to Part 5 of the book Energy Systems[1] for further developments on the issue of technological design and off-design operation.

The compressor used will be first displacement (driver class PiloteFrigoAuxChargeVolum) and then centrifugal (driver class PiloteAuxFrigoChargeTurbo with mapping defined in the file dataCentrifrRefR134a.txt). In both cases, the refrigerant R134a is considered, allowing one to compare the results.

2 Principle of component calculation

The technological design of components requires to refine the phenomenological models used in Thermoptim's core, supplementing them to reflect the off-design operation mechanisms if any. The software has been equipped with new screens for this, called technological design, that define the geometric characteristics representative of the different technologies used and the parameters needed to calculate their performance.

This new environment, developed as external classes must be able to work both complementary to the core components, and at the same time fully consistent with them. At times, the calculations are actually performed by the software package kernel, or made by the technological design classes, the driver ensuring synchronization between the two modes.

2.1 Calculation of exchangers in off-design mode

The NUT method can be presented as follows:

By definition, NTU is defined as the ratio of the UA product to the minimum heat capacity rate.

$$NTU = \frac{UA}{(\dot{m} \cdot c_p)_{min}} \quad (1)$$

We call R the ratio (less than 1) of heat capacity rates:

$$R = \frac{(\dot{m} c_p)_{min}}{(\dot{m} c_p)_{max}} \leq 1 \quad (2)$$

and **ε** the **effectiveness of the exchanger, defined as the ratio of the heat flux actually transferred to the maximum possible flux**:

$$\varepsilon = \frac{\phi}{\phi_{max}} \quad (3)$$

With these definitions, it is possible to show that there is a general relation of type:

ε = f(NTU, R, flow pattern)

In **design mode**, if we know the flow of both fluids, their inlet temperatures and the heat flux transferred, the procedure is as follows:

- we start by determining the outlet temperatures of fluids;
- we deduce the fluid heat capacity rates $\dot{m}\ c_p$ and their ratio R;
- the effectiveness ε is calculated from equation (3);
- the value of NTU is determined from the appropriate (NTU, ε) relationship;

[1] GICQUEL R., Energy Systems : A New Approach to Engineering Thermodynamics, CRC Press, October 2011.

- UA is calculated from equation (1).

In **off-design mode**, we know the inlet temperatures and flow rates of both fluids, the area A of the exchanger and its geometry (flow patterns and technological parameters), calculation is done in three steps:

- determining U by correlations depending on the exchanger flow pattern and geometry;
- calculating UA, product of U and A, and then NTU by (1);
- determining effectiveness ε of the exchanger by the NTU method, and calculating the hot and cold fluid outlet temperatures by (3) and balance equations.

Knowing Thi, Tci, mh, mc and U, it is possible to calculate R and NTU to deduce ε and determine outlet temperatures Tco and Tho.

Knowing Tci, Tco, mh, mc and U, it is possible to calculate R and NTU to deduce ε, and determine temperatures Tho and Thi.

Recall that the NTU method assumes that the thermophysical properties of the fluid are constant in the heat exchanger, while this is true only in first approximation. If we consider U variable, depending as it does on both inlet and outlet temperatures, we obtain an implicit system of equations very difficult to solve, especially if the exchangers are multi-zone as evaporators or condensers.

In practice, however, we can often assume that U does only vary in the second order, and seek an approximate solution by considering U constant and recalculate its value for the new operating conditions, and iterate until we get a reasonable accuracy. In particular it is necessary to operate in this manner when the exchanger is multi-zone, because only the total area is known, not its distribution among the different areas. It is precisely in this way that we operate.

The calculations in the simulator are done using the NTU method, the UA being an unknown intermediate, while the calculations in the technological screens are made in details, leading for a set of inlet and outlet values of a given exchanger and taking into account the corresponding U value, to an estimate of the required area. Consistency between the two calculations is provided when the value of UA is such that the total exchange area is equal to that of sizing.

2.2 Calculation of displacement compressors in off-design mode

A displacement compressor is defined geometrically by its displacement and technological parameters allowing one to calculate its volumetric and isentropic efficiencies, according to its rotation speed and the conditions of suction and discharge

Models implemented in Thermoptim are based on the assumption that the behavior of volumetric compressors can be represented with reasonable accuracy by two parameters: volumetric efficiency λ which characterizes the actual swept volume (4), and classical isentropic efficiency η_s (5).

$$\lambda = a_0 - a_1 \frac{P_{ref}}{P_{asp}} \qquad (4)$$

$$\eta_s = K_1 + K_2 \cdot \frac{P_{asp}}{P_{ref}} + K3 \cdot \left(\frac{P_{asp}}{P_{ref}} \right)^2 \qquad (5)$$

Note that (5) is expressed linearly as a function of the inverse of compression ratio and its square.

Calculation of a compressor is made as follows:

- isentropic and volumetric efficiencies are calculated from equations (3.1.1) and (3.1.2);
- if we know rotation speed, swept volume and volumetric efficiency, volumetric flow can be calculated as:

$$\dot{V} = \frac{\lambda \, N \, V_s}{60} \qquad (1.2.2)$$

- knowing the suction volume v, we deduce mass flow:

$$\dot{m} = \frac{\dot{V}}{v} = \frac{\lambda N V_s}{60\, v}$$

In design mode, the rotation speed or the swept volume required to provide the desired flow is determined. Calculation is done taking into account the inlet and outlet pressures, and the flow value entered in the compressor flow field.

In off-design mode, the compression ratio allows one to determine λ and η_s, which sets the compressor flow and outlet temperature.

3 Selecting and setting technological screens

Suppose we want to size a simple refrigeration cycle (Figure 1) to provide a cooling capacity of 130 kilowatts for an outside temperature of 30 °C, the temperature of the brine being equal to -10 °C at the evaporator inlet.

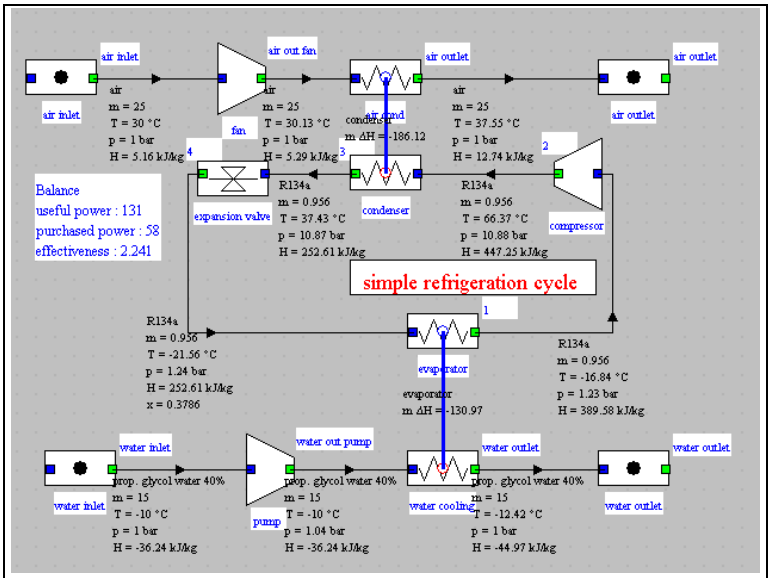

Figure 1: Model of refrigeration cycle

For this cycle, the cold fluid is 40% by volume propylene glycol, available in the external substances. Evaporating temperature must of course be lower than that of the brine. We retained approximately -21.6 °C, that is to say a pressure of 1.24 bar for R134a.

We assume that the isentropic efficiency of the compressor is about 0.8 at design point. To simplify somewhat the model we assume that the value of the evaporation superheating remains constant (5 K). The condensation sub-cooling, however, is determined on the basis of the overall mass of refrigerant, which remains constant in the machine.

Evaporation superheating T_{surch} is 5 K, which for a flow of refrigerant 0.955 kg/s sets the cooling capacity at about 131 kilowatts. With a brine flow of 15 kg/s, this leads to a cooling of 2.42 ° C.

For the condenser, the air temperature is 30 °C and its flow 25 kg/s. Condensing temperature is estimated at 42 °C, which, with an initial sub-cooling ΔT_{ssrefr} of 5 K, corresponds to a condensation pressure of about 10.9 bar. The COP of the machine is 2.24 under these conditions.

Taking into account the refrigerant pressure drops takes some precautions to avoid difficulties in calculating the phases of condensation and evaporation. We chose to assign them only to points where the condition is single phase: in full downstream of the evaporator, and half upstream and downstream of the condenser.

The pressure drop on air and brine are calculated in the technological screens of exchangers and set as an over-pressure upstream of the exchanger. Fans and pump brine are here modeled by compression standard processes, without taking into account their detailed characteristics. It would be possible to do so, but at the cost of increased complexity that is not really justified.

The design is done in two distinct steps, the first is the classic cycle setting in Thermoptim, while the second is made from the technological screen. It should be noted, and this is very important, that the second step can be performed only when the first has been completed and has resulted in a fully consistent model. If this is not the case, the technological design made during the second step may be aberrant and cause great difficulties for convergence.

We will not detail here, for simplicity, how to build the cycle Thermoptim model corresponding to the first step. If it does not exist, you should build it first, then the method would be the same. This refrigeration cycle is similar to those presented in the Getting Started guides the software, the main difference being that the condenser is modeled by a single multi-zone heat exchanger, the desuperheater not being dissociated from the condensation itself in the diagram editor.

3.1 Creation of technological screens

The creation of technological screens can be performed using either the generic driver, or a particular driver, which is necessary when you want to do off-design simulations as is the case here.

In this example, these screens are created as follows: the first step in building the driver is the instantiation of some PointThopt[2] that will allow you to access the simulator points (one for each point of the cycle), as well as the TechnoDesign.

```
amontEvap=new PointThopt(proj,"4");
avalEvap=new PointThopt(proj,"1");
amontCond=new PointThopt(proj,"2");
avalCond=new PointThopt(proj,"3");
waterInlet=new PointThopt(proj,"water inlet");
waterOutlet=new PointThopt(proj,"water outlet");
inAir=new PointThopt(proj,"air inlet");
outAir=new PointThopt(proj,"air outlet");
avalCond.getProperties();
refrig=(rg.corps.Corps)avalCond.lecorps;

 waterPumpOutlet=new PointThopt(proj,"water out pump");
 waterPumpOutlet.getProperties();
 outFanAir=new PointThopt(proj,"air out fan");
 outFanAir.getProperties();

evaporatorName="evaporator";
 condenserName="condenser";
compressorName="compressor";
```

The TechnoDesign are of type TechnoEvaporator, TechnoCondensor and VolumCompr, three classes created specifically for this type of components. The first two are used to calculate the evaporator and condenser as multi-zone heat exchangers, according to the equations given in Appendix 1. The third implements the equations (4) to (7). We will not just explain how to use them, referring the reader to the code of their classes for details.

```
technoEvap=new TechnoEvaporator(proj, evaporatorName, waterInlet, waterOutlet, amontEvap, avalEvap);
addTechnoVector(technoEvap);
technoCond=new TechnoCondensor(proj, condenserName, amontCond, avalCond, inAir, outAir);
addTechnoVector(technoCond);
technoCompr=new VolumCompr(proj, compressorName, avalEvap, amontCond);
addTechnoVector(technoCompr);
setupTechnoDesigns(vTechno);
```

The last line of code above allows you to transfer these technological screens in the core of the software. The values of the displacement and the rotation speed of the TechnoDesign are then displayed on the screen.

```
VsValue=Util.lit_d(technoCompr.Vs_value.getText());
Vs_value.setText(Util.aff_d(VsValue,8));
N_value=Util.lit_d(technoCompr.N_value.getText());
Nref_value.setText(Util.aff_d(N_value,4));
```

[2] This class instances are like clones of the core Thermoptim points, which allow for easy access to their values. It provides more comfort and clarity than does the use of methods getProperties() and updatePoint() of Project, documented in Volume 3 of the reference manual.

3.2 Driver screen

The upper part of the driver screen is given in Figure 2.

It allows you to change on the one hand the exchanger surfaces, and also the length of the liquid line (see Section 6), air temperature, rotation speed or displacement, and has options for the algorithm guidance that will be specified later.

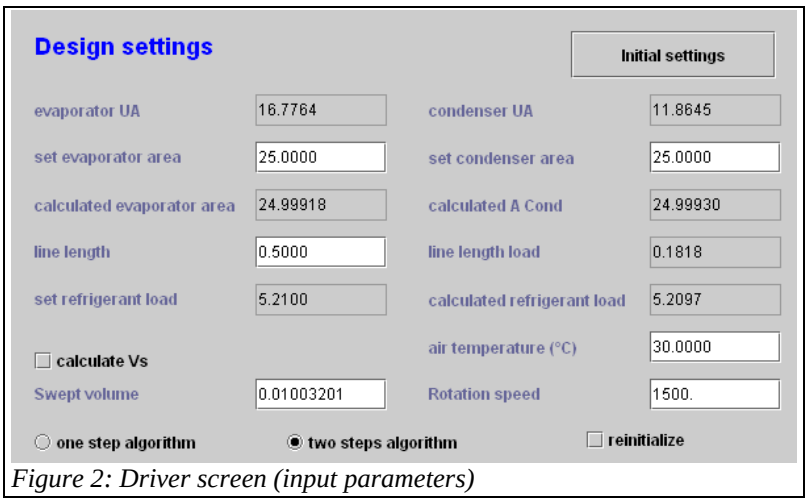

Figure 2: Driver screen (input parameters)

By default, the rotation speed is calculated for the value of the displacement entered in the screen.
If you select "Calculate Vs" the displacement is calculated for the value of the speed entered.

Start by clicking "Initial settings" to instantiate the TechnoDesigns and make an initial technological design, in this case calculate the rotation speed or displacement of the compressor and the surfaces of the two exchangers corresponding to the project file setup, on the basis of default values for TechnoDesign parameters.

3.3 Technological screen setting

Once the technological screens have been created, you set them and size them. This must be done carefully because it involves making a series of choices about the internal configurations, the geometric dimensions...

To access the technological screens, do so from the tables of the general simulator screen (Ctrl T) or from the "tech. design" buttons of component conventional screens.

3.3.1 Evaporator

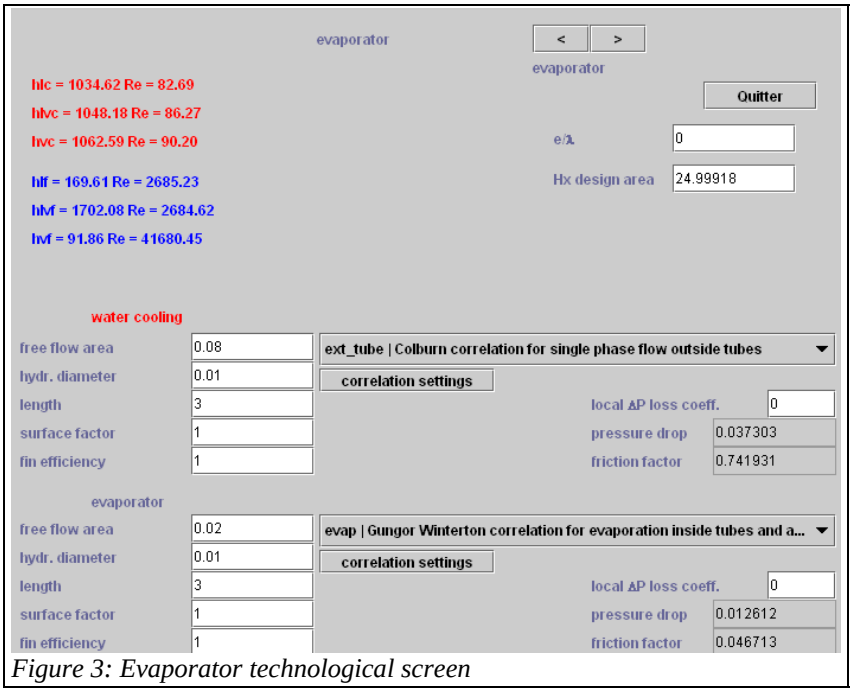

Figure 3: Evaporator technological screen

We consider that the evaporator is of shell and tube type and has water side a fluid flow area of 8 dm2 and a hydraulic diameter of 1 cm, and refrigerant side a flow area of 2 dm 2 and a hydraulic diameter of 1 cm, the tubing length being 3 m. Choose the type of configuration ("evap Gungor Winterton" for evaporation inside the tubes for the "evaporator" (refrigerant), and "ext tube Colburn correlation" for the brine, and set the screen as shown Figure 3.

3.3.2 Condenser

The condenser is of the finned coil type. Air side it has a fluid flow area of 1 m2 and a hydraulic diameter of 1 cm with an extended surface factor of 20, with a length of 15 cm (thickness of the air condenser), and refrigerant side a fluid flow area of 0.5 dm2 and a hydraulic diameter of 2 cm, the tubing length being 3 m.

Choose the type of configuration ("cond Shah correlation" for condensation inside the tubes for the "condenser" (refrigerant), and "air coil Morizot correlation" for air, and set the screen as shown Figure 4.

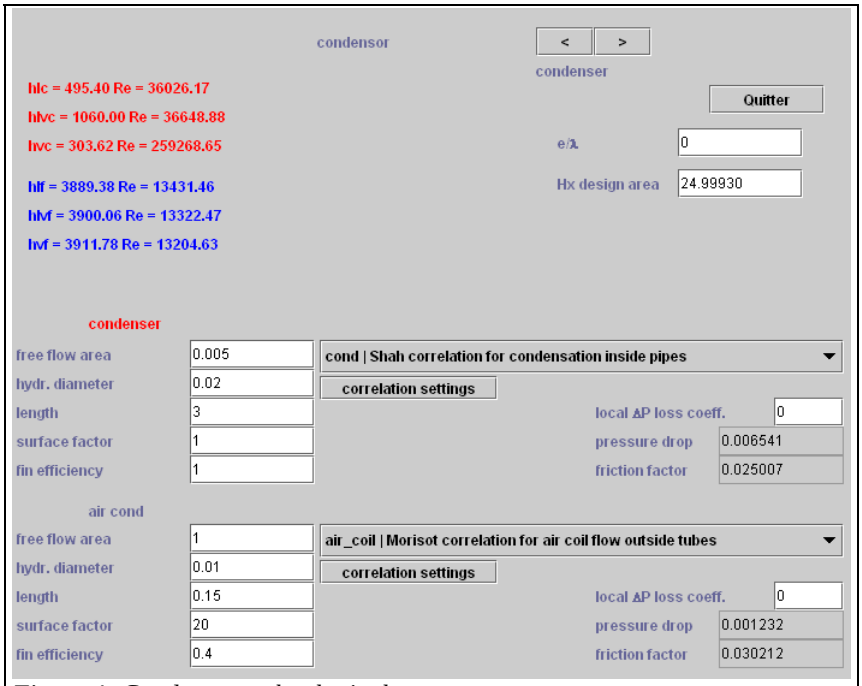

Figure 4: Condenser technological screen

3.3.3 Compressor

For the compressor (Figure 5), we must provide on the one hand the displacement Vs, and also the values of the parameters involved in the equations of isentropic and volumetric efficiency.

As we use the equation with three parameters, the parameters R1 and R2, unused, are equal to 0.

To achieve the technological design once the screens filled, click again on the "Initial settings" screen of the driver (Figure 2).

The results are displayed in the technological screen: exchange surfaces of 25 m2 for the evaporator and condenser; rotation speed of 1500 rpm and displacement of about 0.01 m3 for the compressor.

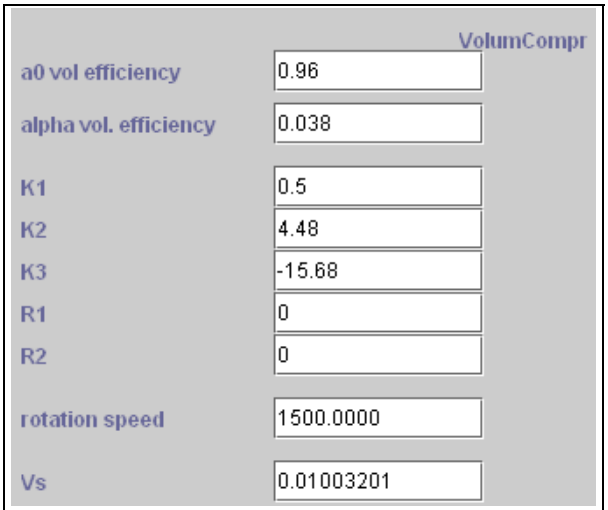

Figure 5: Compressor technological screen

Various calculation results are displayed on the technological screen, such as, for heat exchangers, pressure drop and the values of the Reynolds number Re and the local exchange coefficients.

4 Sizing initializations at design point

The setting of technological screens allows you to determine the exchange surfaces, the total mass of refrigerant and the compressor rotation speed, which are used to set a number of values from those of Thermoptim project, such as initial super-heating and sub-cooling ΔT.

The precise calculation of multi-zone heat exchangers is in turn performed by the method makeDesign() of the exchanger TechnoDesigns, while the total value of UA is obtained in a conventional manner, through the phenomenological models.

Let us explain to begin the initialization of the condenser, that of the evaporator being similar.

The first lines of code make it possible, using the method getProperties() of the Thermoptim project (proj), knowing the name of the exchanger (condenserName), to recover the value of UAcond and the cold fluid name, then the enthalpy DeltaH into play.

```
if(!condenserName.equals("")){//initialisation du Condenseur / condenser initialization
 String[] args=new String[2];
 args[0]="heatEx";
 args[1]=condenserName;
 Vector vProp=proj.getProperties(args);
 Double f=(Double)vProp.elementAt(15);
 UAcond=f.doubleValue();
 String fluideFroid=(String)vProp.elementAt(1);
 args[0]="process";
 args[1]=fluideFroid;
 vProp=proj.getProperties(args);
 f=(Double)vProp.elementAt(4);
 double DeltaH=f.doubleValue();
```

Methods getProperties() of PointThopt directly provide the complete thermodynamic state of the condenser upstream and downstream points, which is used to initialize the sub-cooling.

```
 avalCond.getProperties();
 amontCond.getProperties();
 DTssrefr=avalCond.DTsat;
```

Similarly, the value of the air temperature is obtained from the simulator and displayed in the driver screen.

```
 inAir.getProperties();
 outAir.getProperties();
 Tair=inAir.T;
 Ta_value.setText(Util.aff_d(Tair-273.15,4));
```

These values are used to initialize the values of the condenser heat capacities, which will be used later.

```
 mCpCalopCond=DeltaH/(outAir.T-outFanAir.T);
 mCpRefrigCond=DeltaH/(amontCond.T-avalCond.T);
 UAcond_value.setText(Util.aff_d(UAcond,4));
```

The TechnoDesign is then initialized, making it possible to know the exchange surface necessary given the design done, then the pressure drop is updated.

```
 //initializations of the TechnoDesign
 technoCond.makeDesign();
 AcondReel=Util.lit_d(technoCond.ADesign_value.getText());
 AcalculatedCond_value.setText(technoCond.ADesign_value.getText());

//calcul des pertes de charge / Calculation of pressure drops
 dPcond=technoCond.techc.getPressureDrop()*dPmult;
```

The initial charge of refrigerant is then calculated taking into account the geometric dimensions and the thermodynamic state of the fluid.

```
 //calcul de la charge de frigorigène / Calculation of the refrigerant charge
 mEvap=technoEvap.techf.getFluidLoad();
 mCond=technoCond.techc.getFluidLoad();

 lineLength=Util.lit_d(lineLength_value.getText());
 double dh=Util.lit_d(technoCond.techc.Dh_value.getText());
 volLigne=Math.PI*dh*dh/4.*lineLength;
```

```
 mLigne=volLigne/avalCond.V;
 mLine_value.setText(Util.aff_d(mLigne,4));

 mTot=mCond+mEvap+mLigne;
 mTot_value.setText(Util.aff_d(mTot,4));
}
```

The initialization of the compressor allows you to know the flow rate and determine the corresponding compressor speed, which is displayed in the driver screen.

```
if(!compressorName.equals("")){//initialisation du compresseur / initialization of the compressor
 String[] args=new String[2];
 args[0]="process";
 args[1]=compressorName;
 Vector vProp=proj.getProperties(args);
 String amont=(String)vProp.elementAt(1);
 String aval=(String)vProp.elementAt(2);
 Double f=(Double)vProp.elementAt(3);
 massFlow=f.doubleValue();
 lambdaVol=technoCompr.getLambdaVol();
 if(!jCheckVs.isSelected()){//calcul de la vitesse de rotation / rotation speed calculation
 N_value=massFlow*60*avalEvap.V/VsValue/lambdaVol;
 technoCompr.setN(N_value);
 Nref_value.setText(Util.aff_d(N_value,4));
 }
 else{//calcul de la cylindrée / calculation of the displacement
 N_value=Util.lit_d(Nref_value.getText());
 technoCompr.setN(N_value);
 VsValue=massFlow*60*avalEvap.V/N_value/lambdaVol;
 technoCompr.setVs(VsValue);
 Vs_value.setText(Util.aff_d(VsValue,8));
 }
}
```

The consumption of auxiliaries is then recalculated:

```
//mise à jour des auxiliaires / Updates the auxiliary

 outFanAir.P=inAir.P+dPair;
 outFanAir.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
 updateprocess(fanName, "Compression",RECALCULATE,!IS_SET_FLOW, !UPDATE_FLOW, 0,
!UPDATE_ETA, 0);
 outFanAir.getProperties();

 waterPumpOutlet.P=waterInlet.P+dPwater;
 waterPumpOutlet.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
 updateprocess(pumpName, "Compression",RECALCULATE,!IS_SET_FLOW, !UPDATE_FLOW, 0,
!UPDATE_ETA, 0);
 waterPumpOutlet.getProperties();
```

5 Resolution of the off-design system of equations of the refrigeration machine

In any study of off-design behavior, we must identify what are the independent variables of the system considered, distinguishing them from variables that are deduced.

In this example, it is the quadruple (refrigerant flow, evaporating temperature, condensing temperature, sub-cooling), the variables being the fluid pressures and the thermal and mechanical capacities. To these four "natural" variables must be added the two intermediate variables UAevap and UAcond (Section 2.1).

The search for this sextuplet corresponds to the solution of a set of relatively complex nonlinear equations which involves those we have presented and others that will be detailed section 6.

The solution we have adopted is to build an external driver that provides the resolution of this system of equations and updates Thermoptim once the solution found.

5.1 Method of resolution

We use the Marquardt Levendberg method implemented in algorithms developed in Fortran as the minPack 1 package, and translated in Java. This method combines the Gauss-Newton method and gradient descent. Its main interest is to be very robust and to require as initialization an approximate solution.

Its implementation in Java is done using an interface called optimization.Lmdif_fcn, which forces the calling class (in this case our driver) to have a function called fcn ().

This function fcn () receives as a main argument an array x [n] [3] containing the variables and an array fvec [m] returning residues of the functions that we seek to set to zero. Their numbers may exceed that of the variables, but in our case it will be the same.

Guiding the algorithm is done by playing on two criteria of accuracy, one on the sum of the residues, and the other on the accuracy of the partial derivatives, estimated by finite differences. Remember that we are trying to solve a system of six nonlinear equations in six unknowns, which can be numerically difficult. In practice, it was interesting to propose several options for calculating.

First, the "Reinitialize" option offers the ability to reset the evaporation and condensation temperature values according to those of the brine and the ambient air, to avoid temperature crossing in exchangers.

Then, two exclusive options are available: either run the algorithm in one step, for intermediate values of accuracy of the convergence criteria ("one-step algorithm"), or run it in two steps, first to a coarse convergence and the second more accurate ("two steps algorithm").

The user can choose one or the other, depending on the numerical difficulties encountered. An indicator of accuracy, corresponding to the L2 norm of residuals, is shown in the result part of the driver screen (Figure 6).

If he starts the calculations from the General screen of technological screens, the user can furthermore, if he wishes, interrupt the calculations properly by clicking the "Stop" button, which allows him to change options. Be careful, because this way of working can lead to errors.

5.1.1 Array of variables

The array of variables here is as follows:

x[1] = Tcond;
x[2] = Tevap;
x[3] = massFlow;
x[4] = UAevap;
x[5] = UAcond;
x[6] = DTssrefr;

5.1.2 Initialization and call to the resolution algorithm

The initializations are made on the basis of the values displayed in the driver screen for the evaporator and condenser exchange surfaces, the rotation speed and the displacement of the compressor and the outdoor temperature. Other values are those of the simulator screens.

```
//lecture à l'écran du pilote des paramètres et variables, éventuellement modifiés après initialisation
```

[3] Attention: in order to maintain the same indices as in Fortran, the Java implementation declares n+1 dimensional arrays, instead of a n, the index 0 not being used

```
// Reads in the driver screen the parameters and variables possibly changed after initialization
 AdesignEvap=Util.lit_d(AdesignEvap_value.getText());//surface de l'évaporateur / evaporator surface
 AdesignCond=Util.lit_d(AdesignCond_value.getText());//surface du condenseur /condenser surface

N_value=Util.lit_d(Nref_value.getText());//vitesse de rotation du compresseur /compressor speed
technoCompr.setN(N_value);
VsValue=Util.lit_d(Vs_value.getText());//cylindrée du compresseur /compressor displacement
 technoCompr.setVs(VsValue);

 Tair=Util.lit_d(Ta_value.getText())+273.15;
 inAir.T=Tair;
 inAir.update(UPDATE_T,!UPDATE_P,!UPDATE_X);
 inAir.getProperties();

 amontEvap.getProperties();
 avalEvap.getProperties();
 amontCond.getProperties();
 avalCond.getProperties();
 DTsurch=avalEvap.DTsat;
 DTssrefr=avalCond.DTsat;

algorithmResults.setText("");
```

As noted above, depending on whether or not the "Reinitialize" option is checked, the change of state temperatures are those of the simulator or determined from the temperature of the coolant (air) and the secondary refrigerant (brine). These initializations allow one avoiding temperature inversions in the heat exchangers when the research is far from the starting state.

```
 //initialisation des températures de changement d'état
// Initialization of the state change temperature
 if(jcheckReInitialize.isSelected()){//si l'option "reinitialize" est cochée / if "Reinitialize" is checked
 Tcond= Tair+10;
 Tevap = Tf-10;
}
else{
 Tcond=refrig.getSatTemperature(avalCond.P, 1);
 Tevap=refrig.getSatTemperature(amontEvap.P, 1);
}
```

The call for the resolution algorithm is, once it is initialized:

```
int m = 6; int n = 6;

double fvec[] = new double[m+1];
double x[] = new double[n+1];
 int info[] = new int[2];
int iflag[] = new int[2];

x[1] = Tcond;
x[2] = Tevap;
x[3] = massFlow;
x[4] = UAevap;
x[5] = UAcond;
x[6] = DTssrefr;

double residu0;
double residu1;

 fcn(m,n,x,fvec,iflag);
```

```
residu0 = optimization.Minpack_f77.enorm_f77(m,fvec);

nfev2 = 0; njev2 = 0;

double epsi=0.0005;//précision globale demandée sur la convergence / overall accuracy required on the
convergence
double epsfcn = 1.e-6;//précision calcul des différences finies / precision calculus of finite differences

 if(twoStepAlgorithm.isSelected()){
 epsi=0.01;
 }

 //appel modifié de lmdiff avec modification précision calcul des différences finies
// modified call to lmdiff with precision change in the calculus of finite differences
optimization.Minpack_f77.lmdif2_f77(this, m, n, x, fvec, epsi, epsfcn, info);
residu1 = optimization.Minpack_f77.enorm_f77(m,fvec);

if(twoStepAlgorithm.isSelected()){
 epsi=0.00005;
 epsfcn = 1.e-9;//précision calcul des différences finies / precision calculus of finite differences

 //appel modifié de lmdiff avec modification précision calcul des différences finies
// modified call to lmdiff with precision change in the calculus of finite differences
 optimization.Minpack_f77.lmdif2_f77(this, m, n, x, fvec, epsi, epsfcn, info);
 residu1 = optimization.Minpack_f77.enorm_f77(m,fvec);
}
```

5.1.3 Function fcn()

To estimate the residuals, it is necessary, as shown in the code below, to start by updating the Thermoptim variables corresponding to the array x, and also to calculate the other variables.

As indicated in fcn(), functions fvec are six methods resEvap(),recCond(), resFlow(), resAevap(), resAcond() and resLoad() defined below.

The call to the first three is done before recalculation of the simulator and theTechnoDesign, and that to the other afterwards.

```
public void fcn(int m, int n, double x[], double fvec[], int iflag[]) {

if (iflag[1]==1) this.nfev++;
if (iflag[1]==2) this.njev++;

//mise à jour des variables "physiques" pour une meilleure compréhension du code
 // Update of "physical" variables for a better understanding of the code
Tcond=x[1];
 Tevap=x[2];
 massFlow=x[3];
 UAevap=x[4];
 UAcond=x[5];
 DTssrefr=x[6];
```

The first step is to update all points and processes of the model so that the calculation of the residuals are made on the basis of the values of array x. The pressure drops are allocated half to the upstream point of the condenser and half to its downstream point, and fully to the evaporator downstream point.

```
//mise à jour du point aval du condenseur (avec modification du sous-refroidissement)
// Update of point downstream of the condenser (with modification of sub-cooling)
 Pcond=refrig.getSatPressure(Tcond, 1);
 avalCond.T=Tcond+DTssrefr;// DTssrefr est negative / DTssrefr is negative
```

```
 avalCond.P=Pcond-dPcond/2;
 avalCond.T=refrig.getSatTemperature(avalCond.P,1)+DTssrefr;// DTssrefr est negative / DTssrefr is negative
 avalCond.DTsat=DTssrefr;
 avalCond.update(UPDATE_T, UPDATE_P, !UPDATE_X, false, UPDATE_DTSAT, false, "", 0.);
 avalCond.getProperties();

 //mise à jour du point aval de l'évaporateur (avec modification de la surchauffe)
// Update of the point downstream of the evaporator (with change of superheating)
 Pevap=refrig.getSatPressure(Tevap, 1);
 avalEvap.P=Pevap-dPevap;
 avalEvap.T=refrig.getSatTemperature(Pevap-dPevap,1)+DTsurch;
 avalEvap.DTsat=DTsurch;
 avalEvap.update(UPDATE_T, UPDATE_P, !UPDATE_X, false, UPDATE_DTSAT, false, "", 0.);
 avalEvap.getProperties();

 //mise à jour de la pression du point aval du compresseur
// Update of the point pressure downstream of the compressor
 Pcond=refrig.getSatPressure(Tcond, 1);
 amontCond.P=Pcond;
 amontCond.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
 amontCond.getProperties();

//mise à jour de l'état du point amont de l'évaporateur
 //le recalcul du titre est fait après convergence
// Update of the state of the point upstream of the evaporator
 // The recalculation of the quality is made after convergence
 amontEvap.P=Pevap;
 amontEvap.T=Tevap;
 amontEvap.update(UPDATE_T,UPDATE_P,!UPDATE_X);

 //mise à jour des variables liées / Update of other variables
 DeltaHevap=getDeltaHEvap();//directement déterminé à partir de x / directly determined from x
 DeltaHcond=DeltaHevap+getTauCompr();//modifie la température de sortie compresseur / changes the
compressor outlet temperature
```

Once these updates made, the first three residuals corresponding to the balance of the refrigeration cycle are calculated.

```
 //calcul des premiers résidus / Calculation of the first residuals
fvec[1] = resEvap();
fvec[2] = resCond();
fvec[3] = resFlow();
```

The cycle being balanced, the project is recalculated a number of times, as well as heat exchangers:

```
//mises à jour du simulateur avant recalcul des TechnoDesign
//on les exécute 3 fois par sécurité, mais ce n'est pas optimisé
// Updates the simulator before recalculating the TechnoDesign
// It is running three times as a safety, but it is not optimized
 for(int j=0;j<3;j++)proj.calcThopt();
 updateHx(evaporatorName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
 updateHx(condenserName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
 updateHx(evaporatorName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
 updateHx(condenserName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
 updateHx(evaporatorName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
 updateHx(condenserName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0, !UPDATE_DTMIN, 0);
```

All PointThopt are updated, and auxiliary updated:

```
 amontEvap.getProperties();
```

```
 avalEvap.getProperties();
 amontCond.getProperties();
 avalCond.getProperties();
 waterInlet.getProperties();
 waterOutlet.getProperties();
 inAir.getProperties();
 outAir.getProperties();

 //mise à jour des auxiliaries / Update of the auxiliary

outFanAir.P=inAir.P+dPair;
 outFanAir.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
 updateprocess(fanName, "Compression",RECALCULATE,!IS_SET_FLOW, !UPDATE_FLOW, 0,
!UPDATE_ETA, 0);
 outFanAir.getProperties();

 waterPumpOutlet.P=waterInlet.P+dPwater;
 waterPumpOutlet.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
 updateprocess(pumpName, "Compression",RECALCULATE,!IS_SET_FLOW, !UPDATE_FLOW, 0,
!UPDATE_ETA, 0);
 waterPumpOutlet.getProperties();
```

The exchanger TechnoDesign are then recalculated, which updates the pressure drops and allows us to estimate the exchange surfaces and the refrigerant charge corresponding to this new state:

```
 technoEvap.makeDesign();
 technoCond.makeDesign();
 AcalculatedEvap_value.setText(technoEvap.ADesign_value.getText());
 AcalculatedCond_value.setText(technoCond.ADesign_value.getText());

//calcul des pertes de charge / Calculation of pressure drops
dPevap=technoEvap.techf.getPressureDrop()*dPmult;
 dPcond=technoCond.tech.getPressureDrop()*dPmult;
```

Finally, the last three residuals are estimated:

```
 //calcul des derniers résidus après recalcul des TechnoDesign (adimensionnés)
// Calculates the last residuals after recalculating the TechnoDesign (dimensionless)
fvec[4] = resAevap();
fvec[5] = resAcond();
fvec[6] = resLoad();

return;
}
```

5.1.4 Residual functions

We just give two examples of residuals, on the condenser balance and the calculation of the exchange surface. The others are presented Section 6. In both cases, the residual was normalized to balance the weight of the different functions of deviation, using a simple method, but valid only if the target value is not zero, which is always the case here.

```
 double resCond(){//equation 17.4.5
 //renvoie la différence entre la température de condensation
 //recalculée à partir de la méthode du NUT et Tcond=x[1]
// Returns the difference between the condensation temperature
 // recalculated using the NTU method and Tcond = x [1]

 mCpRefrigCond=DeltaHcond/(amontCond.T-(Tcond+DTssrefr));
```

```
 double[]res=Util.epsi_NUT(mCpRefrigCond,mCpCalopCond,UAcond);
 epsilon=res[0];
 mCpmin=res[1];

 double Tentree_refrig=outFanAir.T+DeltaHcond/epsilon/mCpmin;
 double Tsortie_refrig=Tentree_refrig-DeltaHcond/mCpRefrigCond;

 //le résidu est l'écart entre les deux températures de condensation
// The residual is the difference between the two condensing temperatures
 double z= Tcond-(Tsortie_refrig-DTssrefr);// DTssrefr est negative / DTssrefr is negative
 z= 2*z/(Tcond+Tsortie_refrig-DTssrefr);
 return z;
}

double resAcond(){//renvoie le résidu de la surface du condenseur / returns the residual of the condenser surface
 UAcond_value.setText(Util.aff_d(UAcond,4));
 AcalculatedCond_value.setText(Util.aff_d(AcondReel,4));
 AcondReel=technoCond.A;
 double z= AcondReel-AdesignCond;

 z= 2*z/(AcondReel+AdesignCond);
 return z;
}
```

5.2 Display of the driver results after calculation in off-design mode

The accuracy of the solution is written in the text file "output.txt", and the driver screen is updated.

```
 System.out.println();
 System.out.println(" Initial L2 norm of the residuals: " + residu0);
 System.out.println("Final L2 norm of the residuals: " + residu1);
 System.out.println("Number of function evaluations: " + nfev);
 System.out.println("Number of Jacobian evaluations: " + njev);
 System.out.println("Info value: " + info[1]);
 System.out.println("Final approximate solution: " + x[1] + ", " + x[2]+ ", " + x[3] );
 System.out.println(); /**/

algorithmResults.setText("Algorithm precision: " + Util.aff_d(residu1,8));

 UAevap_value.setText(Util.aff_d(UAevap,4));
 UAcond_value.setText(Util.aff_d(UAcond,4));
 AcalculatedEvap_value.setText(technoEvap.ADesign_value.getText());
 AcalculatedCond_value.setText(technoCond.ADesign_value.getText());
 COP_value.setText(Util.aff_d(DeltaHevap/tauCompr,4));
 DeltaHevap_value.setText(Util.aff_d(DeltaHevap,4));
 massFlow_value.setText(Util.aff_d(massFlow,4));
 DeltaHcond_value.setText(Util.aff_d(DeltaHcond,4));
 Pevap_value.setText(Util.aff_d(Pevap,4));
 Pcond_value.setText(Util.aff_d(avalCond.P,4));
 tauCompr_value.setText(Util.aff_d(tauCompr,4));
```

6 Equations of the refrigeration machine in off-design mode

6.1 Evaporator balance

The evaporating temperature is set by the thermal equilibrium of the evaporator, which depends mainly on the one hand on the temperature T_f and the flow of secondary refrigerant (brine in this example), and also the flow of refrigerant.

$$\Delta H_{evap} = \dot{m} (h(T_e + \Delta T_{surch}, P_e) - h(T_c - \Delta T_{ssrefr}, P_c)) = U_e A_e \Delta T_{ml\_ef} \quad (8)$$

ΔH_{evap} is calculated by:

```
double getDeltaHEvap(){//calcule la puissance thermique de l'évaporateur / calculates the evaporator thermal capacity
 return massFlow*(avalEvap.H-avalCond.H);
}
```

Balancing the heat exchanger is made by recalculating the evaporation temperature by the NTU method:

```
 double resEvap(){//équation 17.4.3
 //renvoie la différence entre la température d'évaporation,
 //recalculée à partir de la méthode du NUT et Tevap=x[2]
// Returns the difference between the evaporation temperature,
 // recalculated using the NTU method and Tevap = x [2]

 mCpRefrigEvap=DeltaHevap/DTsurch;

 double[]res=Util.epsi_NUT(mCpRefrigEvap,mCpCalopEvap,UAevap);
 epsilon=res[0];
 mCpmin=res[1];

 double z=Tevap-waterPumpOutlet.T+DeltaHevap/epsilon/mCpmin;//résidu / residual

 z= 2*z/(Tevap+Tf-DeltaHevap/epsilon/mCpmin);
 return z;
 }
```

6.2 Compressor balance

The balance of the compressor expresses that the compression work equal to the product of the mass flow by the isentropic compression work divided by the isentropic efficiency. It also depends on several variables and is given by:

$$\tau = \dot{m} \frac{\Delta h_s (T_e, P_e, P_c)}{\eta_s (P_c/P_e)} \quad (9)$$

It is calculated by:

```
 double getTauCompr(){//equation 17.4.4
// recalcul du compresseur / Recalculates the compressor
 double eta_is=technoCompr.getRisentr();
 updateprocess(compressorName, "Compression",RECALCULATE,IS_SET_FLOW, UPDATE_FLOW, massFlow, UPDATE_ETA, eta_is);
 amontCond.getProperties();

 //ce qui permet de connaître la puissance consommée / Which allows to know the power consumption
 String[] args=new String[2];
 args[0]="process";
 args[1]=compressorName;
 Vector vProp=proj.getProperties(args);
 Double f=(Double)vProp.elementAt(4);
 tauCompr=f.doubleValue();

 return tauCompr;
 }
```

6.3 First law

The first law reads:

$$\Delta H_{cond} = \tau + \Delta H_{evap} \quad (10)$$

It is written in fcn () as:

```
DeltaHcond=DeltaHevap+getTauCompr();//modifie la température de sortie compresseur / changes the compressor outlet temperature
```

6.4 Condenser balance

Similarly to what we presented for the evaporator, the condensation temperature is set by the thermal balance of the condenser, which depends mainly on the one hand on the temperature and flow of cooling air, on the other hand on the refrigerant flow, and finally on the condenser outlet temperature.

The condenser outlet temperature depends on the compression ratio and compressor isentropic efficiency which itself also depends on this ratio.

$$\Delta H_{cond} = U_c A_c \Delta T_{ml\_ac} \quad (11)$$

Balancing the heat exchanger is made by recalculating the condensation temperature by the NTU method. The code has already been given section 5.1.4.

6.5 Conservation of mass flow

The solution of equation (7) is done by comparing the value it provides (implemented in the compressor TechnoDesign) with that of massFlow = x [3]. The calculation of λ is done in theTechnoDesign.

$$\dot{m} = \frac{\dot{V}}{v} = \frac{\lambda N V_s}{60\, v} \quad (7)$$

```
double resFlow(){//equations 17.4.1 and 17.4.2
//renvoie le résidu du débit-masse / Returns the residual of the mass flow

Vevap=avalEvap.V;
double rCompr=amontCond.P/avalEvap.P;
double y=technoCompr.getMassFlow(Vevap, rCompr);
double z= massFlow-y;
z= 2*z/(massFlow+y);
return z;
}
```

6.6 Conservation of the refrigerant charge

The calculation of the mass of refrigerant in the two-phase exchangers requires special care because we do not know the exact distribution of liquid and vapor phases.

In design mode, selecting a reference subcooling (eg 5 K) we deduce the mass of refrigerant in the circuit: evaporator, condenser and liquid line (upstream the expansion valve). We can neglect the mass of refrigerant in the compressor and between the evaporator and compressor.

In off-design mode, the subcooling is calculated so that the mass of refrigerant circuit is conserved.

The void fraction ε is defined by the section occupied by the vapor divided by the total section of the exchanger. Its knowledge is essential to predict the charge of the refrigeration system. Indeed the total mass in a heat exchanger can be expressed in terms of the average density:

$$\rho_m = \varepsilon\rho_v + (1-\varepsilon)\rho_l \qquad (12)$$

ε, function of a constant Kh and the slip S of the liquid and vapor phases, is given by:

$$\varepsilon = \frac{K_H}{1 + S\frac{1-x}{x}\frac{\rho_v}{\rho_l}} \qquad (13)$$

The calculation of the average density is done by integrating (13) between the inlet and oulet qualities in the exchanger.

The mass contained in the liquid line is itself determined by considering a tube length of the same section as that of the condenser, which is entered in the TechnoDesign screen of Figure 4. The code is given below:

```
 //mises à jour pour la charge de frigorigène
// Updates for the refrigerant charge
 mEvap=technoEvap.techf.getFluidLoad();
 mCond=technoCond.techc.getFluidLoad();
 mLigne=volLigne/avalCond.V;
 mLine_value.setText(Util.aff_d(mLigne,4));
double load=mCond+mEvap+mLigne;


String [] args = new String [2];
 args [0] = "process";
 args [1] = compressorName;
 Vector vProp proj.getProperties = (args);
 Double f = (Double) vProp.elementAt (4);
 tauCompr f.doubleValue = ();

 tauCompr return;
 }
```

7 Use of the driver

The full screen of the driver is given in Figure 6. It allows you to change the exchanger surfaces and the length of the liquid line, as well as the air temperature, displacement or rotation speed.

Enter the air temperature or the speed you want to change, and either click "Calculate" or open the TechnoDesign screen from the simulator, then click "Calculate the driver." The second way is preferable because it allows you to track the convergence while keeping hands on Thermoptim to display intermediate values or modify the calculation of the driver.

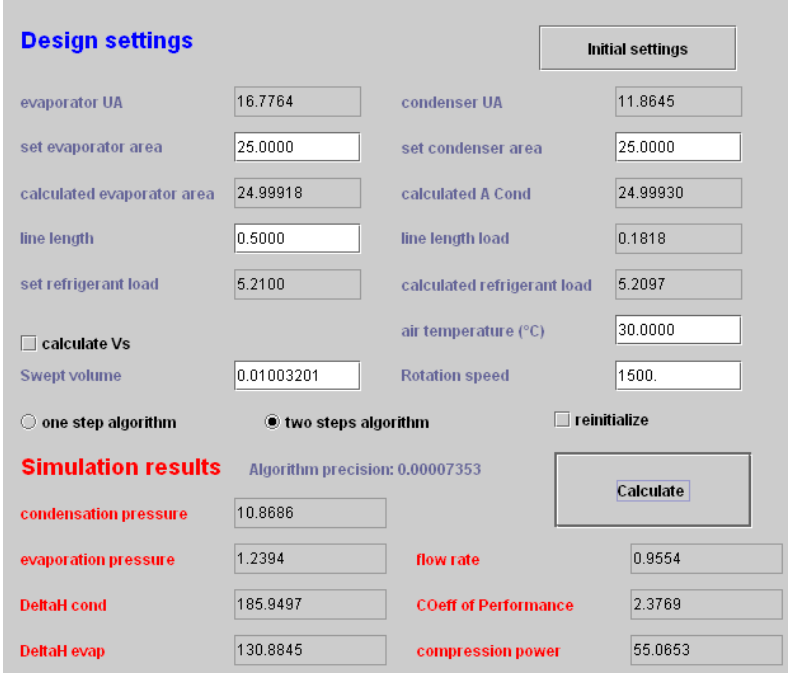

Figure 6: Driver screen

The results are displayed on the screen once the convergence obtained. If the values you enter are very different from those of design,

Thermoptim calculation errors can be generated with messages. If necessary, choose a value to recalculate closer to the initial value.

Figure 7 shows the simulation results obtained when we vary the temperature of the cooling air for the technological screen setting selected above. The influence of the rotation speed is given in Figure 8.

In this example, we have varied only two variables, but it would be easy to study the influence of air and brine flows, or temperature of the latter, either by modifying their values in the simulator screens before recalculation with this driver, or by modifying it so that these values appear in the driver screen and then be automatically updated.

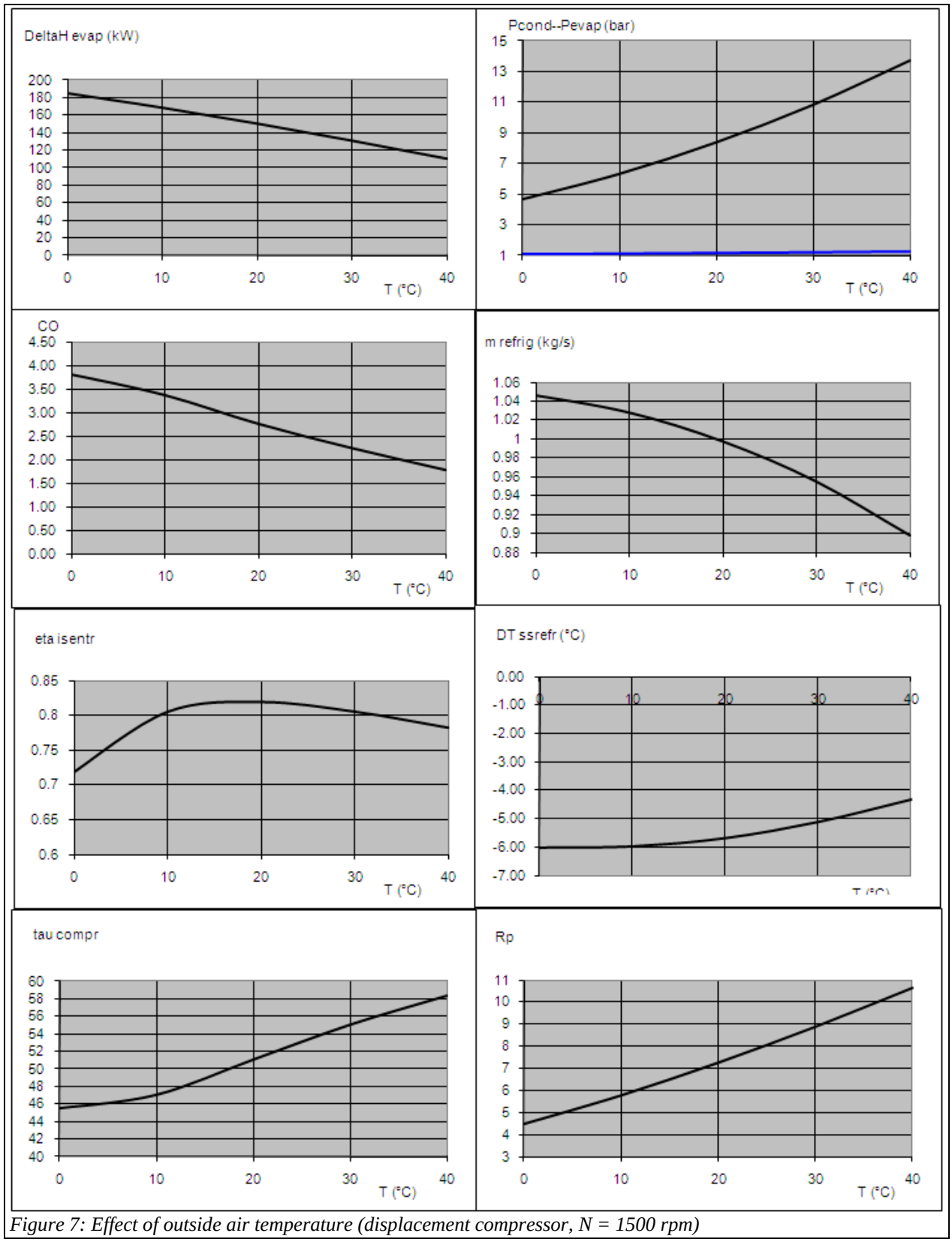

Figure 7: Effect of outside air temperature (displacement compressor, N = 1500 rpm)

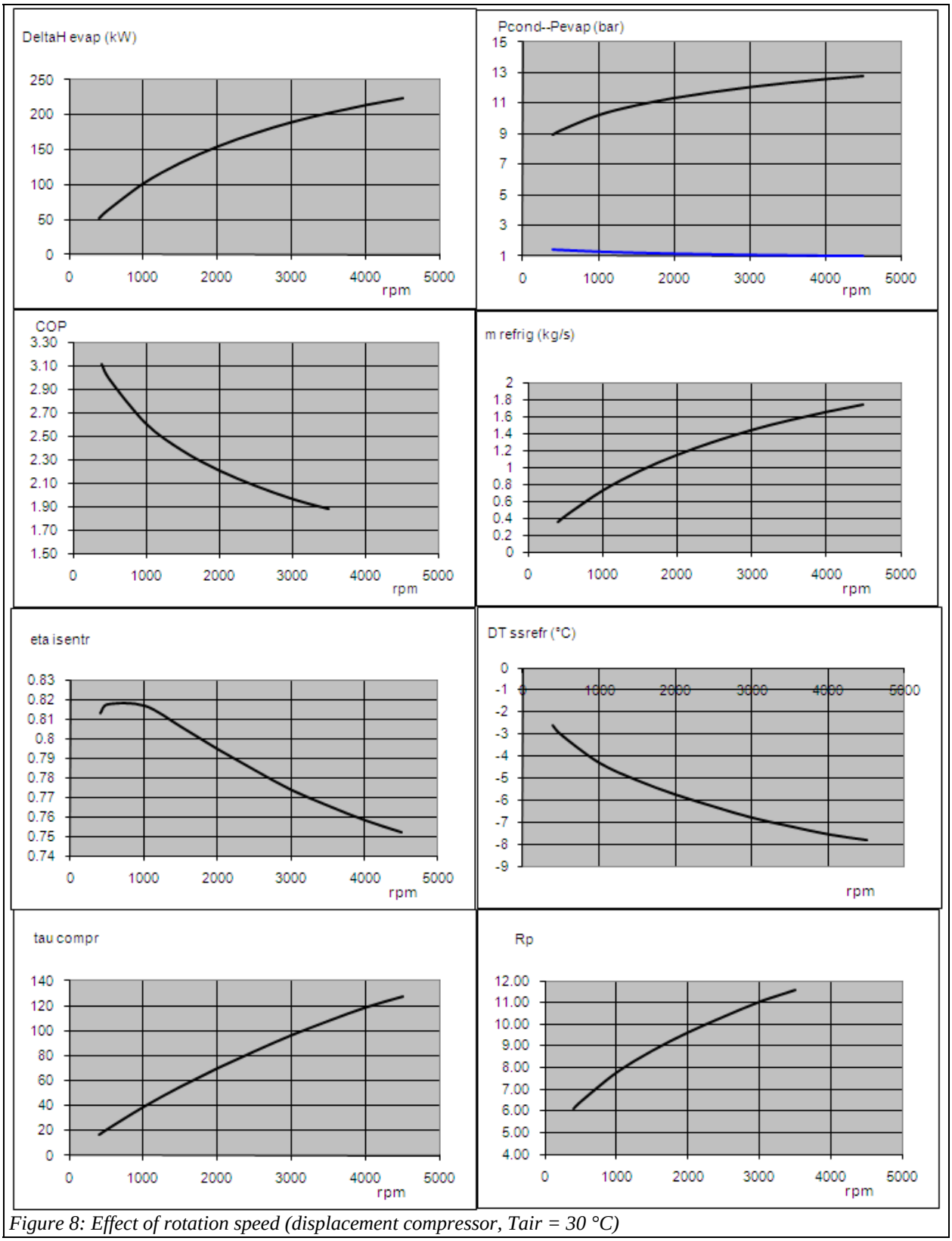

Figure 8: Effect of rotation speed (displacement compressor, Tair = 30 °C)

8 Cycle with dynamic compressor

8.1 Calculation of dynamic compressors in off-design mode

When the compressor used is no longer displacement, but centrifugal, equations (4) to (7) are no longer valid, and must be replaced by an accurate representation of the dynamic compressor mapping.
The characteristics of the dynamic compressor are given by equations (14) and (15):

$$Rp = \frac{P_r}{P_a} = f(\dot{m}_c) \qquad (14)$$

$$\eta_s = f(\dot{m}_c) \qquad (15)$$

These equations are written in dimensionless form and expressed numerically as explained in Volume 4 of the Thermoptim reference manual (see Figures 9 and 10).

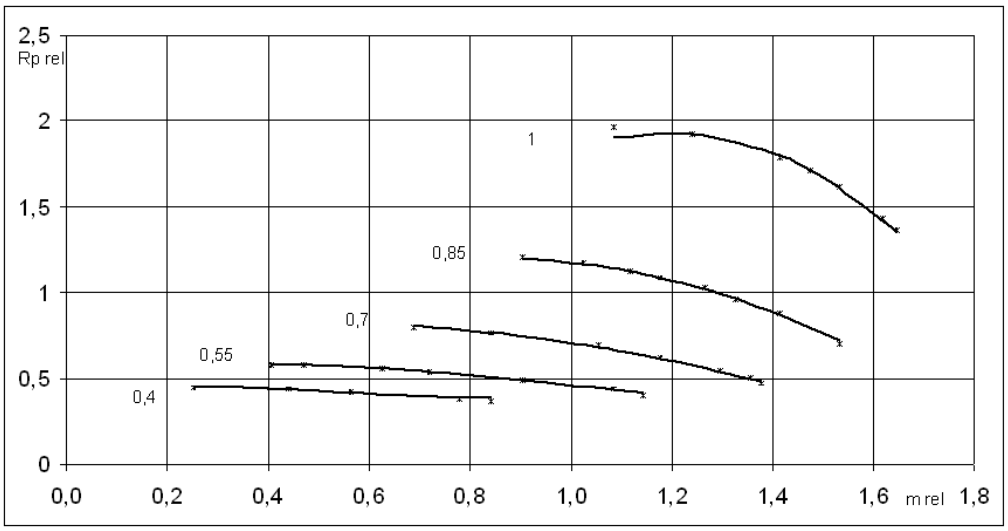

Figure 9: Dimensionless characteristics (compression ratio, flow)

The mass flow rate depends on the thermodynamic state at the suction and compression ratio.

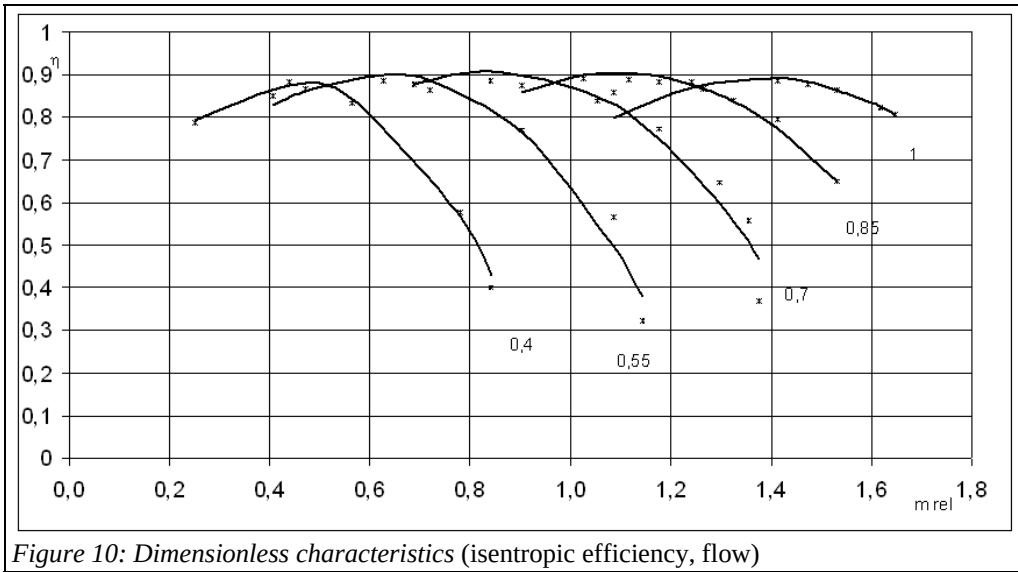

Figure 10: Dimensionless characteristics (isentropic efficiency, flow)

8.2 Code changes

The changes to the driver code are minor, most of the methods being implemented in the external classes MappedTurboCompr (TechnoDesign) and TurboComprMapDataFrame (mapping). The instantiation of the TechnoDesign is changed accordingly.

technoCompr=new MappedTurboCompr(proj, compressorName, avalEvap, amontCond);

8.2.1 Initializions for compressor sizing

When initializing the compressor we load the appropriate mapping, then search the rotation speed leading to nominal compression ratio, given the inlet conditions:

```
 technoCompr.setDataFile(dataFile);
 technoCompr.makeDesign();

 //recherche de la vitesse de rotation correspondant à Rp et massFlow
// Search for the rotation speed corresponding to Rp and massFlow
 double Ninit=technoCompr.getNfromRpAndFlow(Rp,
massFlow)/technoCompr.racT0*Math.pow(avalEvap.T,0.5)*technoCompr.Nref;
 if(Ninit!=0){//si on est en dehors de la cartographie, on ne fait rien en dehors du message d'information
// if we are outside the map, nothing is done outside the informational message
 technoCompr.setN(Ninit/technoCompr.Nref);
 technoCompr.setNparameters();//calcule les paramètres qui dépendent de N / calculates the parameters that
depend on N
 N_value=Ninit;
 Nref_value.setText(Util.aff_d(Ninit,4));
 }
```

8.2.2 Initialization of off-design calculations

Before starting the calculations, we read on the screen the value of the speed selected, and update the TechnoDesign in dimensionless form:

```
 N_value=Util.lit_d(Nref_value.getText())/technoCompr.Nref;
 technoCompr.setDesignValues();
 technoCompr.setN(N_value);
 technoCompr.setNparameters();
```

8.2.3 Update of the reduced speed

During updates that take place in the function fcn (), the reduced speed is updated once the new suction conditions known:

```
 //mise à jour de la vitesse de rotation corrigée du compresseur
/ / Update of the corrected compressor speed
 technoCompr.updateN();
 technoCompr.setNparameters();
```

8.2.4 Verification of adaptation to the mapping of the reduced speed

Once the calculations are complete, we check whether the reduced speed remains within allowable limits for the mapping chosen. Otherwise the user is warned.

```
 technoCompr.checkMapValidity();
```

8.3 Model Results

The model results are quite consistent with those obtained with the displacement compressor with regard to the influence of outside temperature, the only differences being due to the change of characteristics (see Figure 11).

That of the rotation speed is in turn quite different from that obtained with the displacement compressor (see Figure 12).

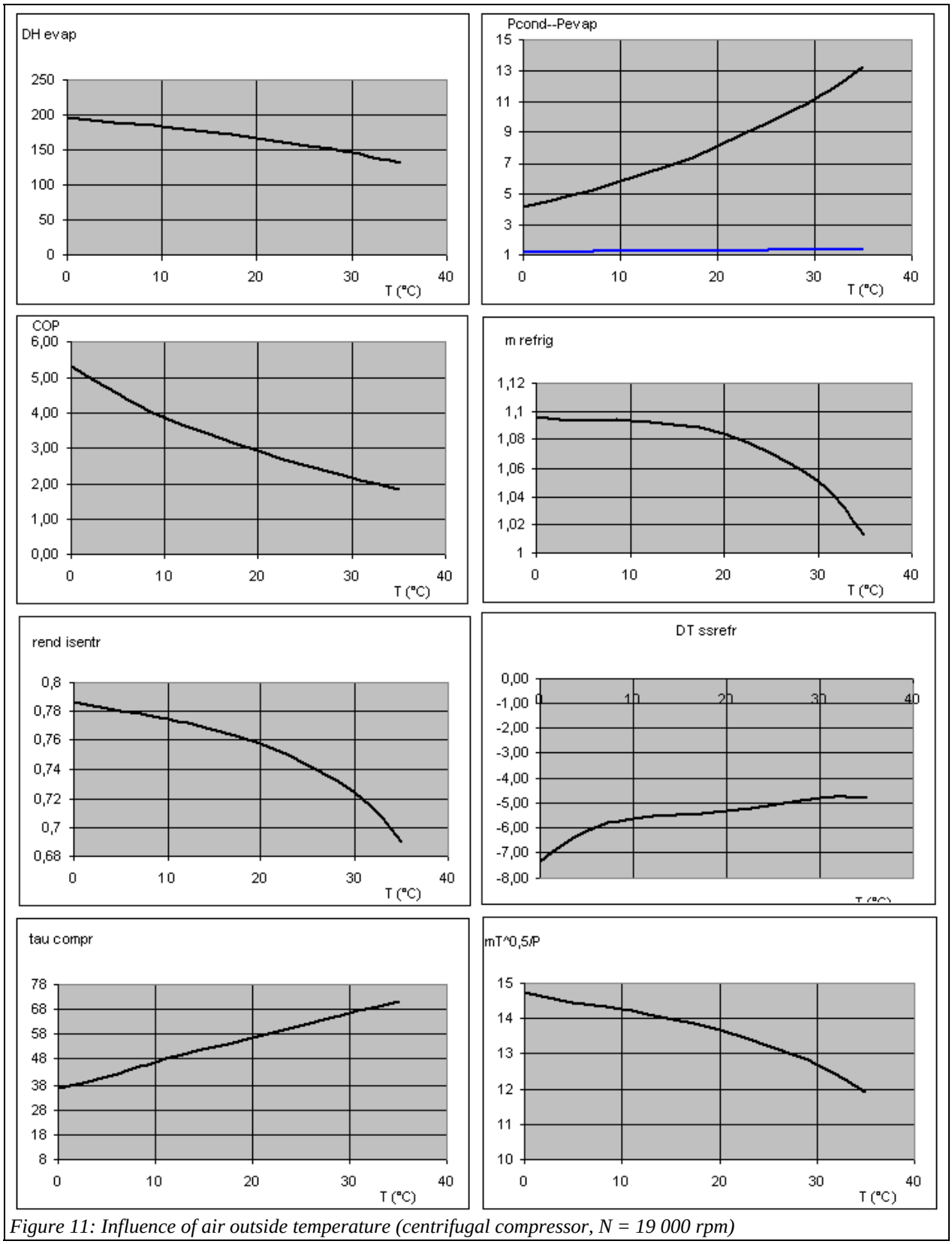

Figure 11: Influence of air outside temperature (centrifugal compressor, N = 19 000 rpm)

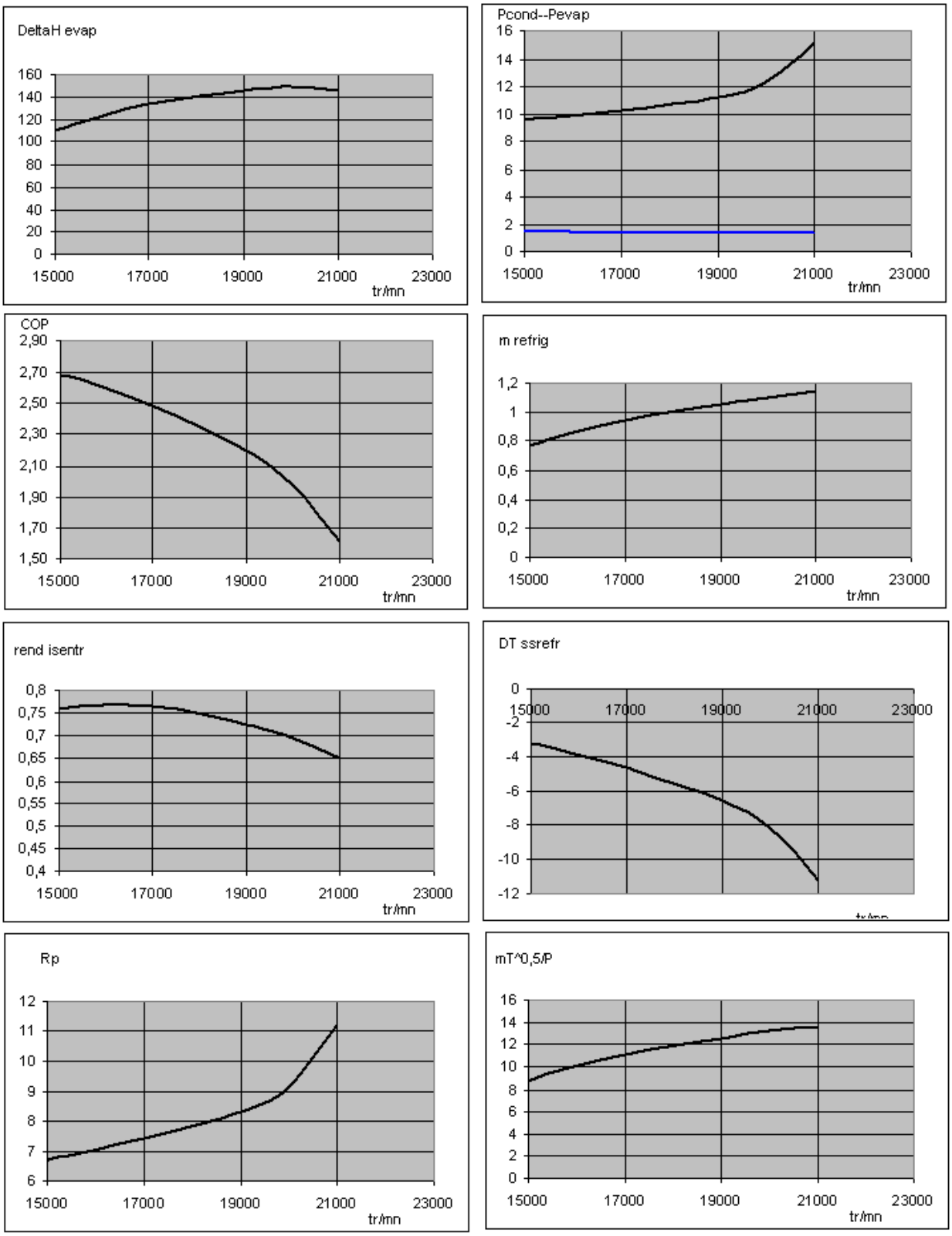

Figure 12: Influence of rotational speed (centrifugal compressor, Tair = 30 °C)

Annex 1: Principle of multi-zone heat exchanger calculation

Evaporators: two-phase cold fluid and hot fluid sensible heat

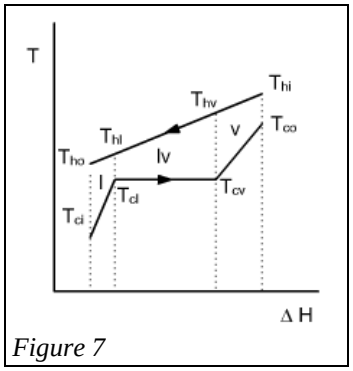

Figure 7

The equations are as follows (Figure 7):

The equations are as follows (figure 2.2.6):
$m_h Cp_h (T_{hi} - T_{hv}) = m_c Cp_{cv} (T_{co} - T_{cv})$
$m_h Cp_h (T_{hv} - T_{hl}) = m_c Cp_{clv} (T_{cv} - T_{cl}) = m_c L_c$
$m_h Cp_h (T_{hl} - T_{ho}) = m_c Cp_{cl} (T_{cl} - T_c i)$
Relations giving epsilon and R are then:

$$\varepsilon_v = \frac{T_{co} - T_{cv}}{T_{hi} - T_{cv}} \qquad R_v = \frac{m_c Cp_{cv}}{m_h Cp_h}$$

$$\varepsilon_{lv} = \frac{T_{hv} - T_{hl}}{T_{hv} - T_{ci}} \qquad R_{lv} = \frac{m_h Cp_h}{m_c Cp_{clv}}$$

$$\varepsilon_l = \frac{T_{cl} - T_{ci}}{T_{hl} - T_c i} \qquad R_l = \frac{m_c Cp_{cl}}{m_h Cp_h}$$

If the fluid enters the evaporator in the two-phase state, the equations are slightly different.

Vapor inlet condensers: two-phase hot fluid and cold fluid sensible heat

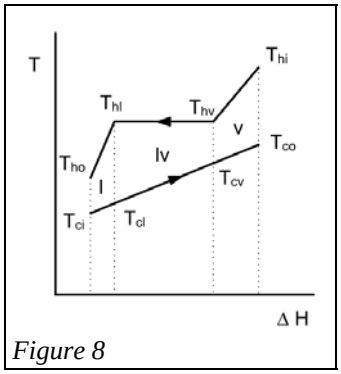

Figure 8

The equations are as follows (Figure 8):

$m_h Cp_{hv} (T_{hi} - T_{hv}) = m_c Cp_c (T_{co} - T_{cv})$
$m_h Cp_{hlv} (T_{hv} - T_{hl}) = m_c Cp_c (T_{cv} - T_{cl}) = m_h L_h$
$m_h Cp_{hl} (T_{hl} - T_{ho}) = m_c Cp_c (T_{cl} - T_{ci})$
Relations giving epsilon and R are then:

$$\varepsilon_v = \frac{T_{hi} - T_{hv}}{T_{hi} - T_{cv}} \qquad R_v = \frac{m_h Cp_{hv}}{m_c Cp_c}$$

$$\varepsilon_{lv} = \frac{T_{cv} - T_{cl}}{T_{hv} - T_{cl}} \qquad R_{lv} = \frac{m_c Cp_c}{m_h Cp_{hlv}}$$

$$\varepsilon_l = \frac{T_{hv} - T_{ho}}{T_{hv} - T_{ci}} \qquad R_l = \frac{m_h Cp_{hl}}{m_c Cp_c}$$

Two-phase inlet condensers: two-phase hot fluid and cold fluid sensible heat

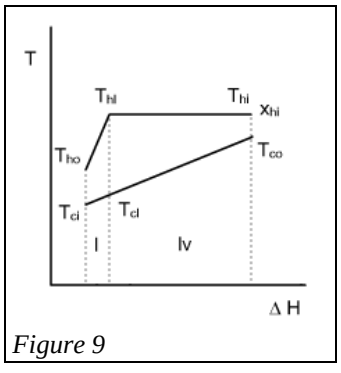

Figure 9

The equations are as follows (Figure 9):

$m_h Cp_{hlv} (T_{hi} - T_{hl}) = m_c Cp_c (T_{co} - T_{cl}) = m_h L_h x_{hi}$
$m_h Cp_{hl} (T_{hl} - T_{ho}) = m_c Cp_c (T_{cl} - T_{ci})$
Relations giving epsilon and R are then:

$$\varepsilon_{lv} = \frac{T_{co} - T_{cl}}{T_{hi} - T_{cl}} \qquad R_{lv} = \frac{m_c Cp_c}{m_h Cp_{hlv}}$$

$$\varepsilon_l = \frac{T_{hi} - T_{ho}}{T_{hi} - T_{ci}} \qquad R_l = \frac{m_h Cp_{hl}}{m_c Cp_c}$$