T H E R M O P T I M ®

OFF-DESIGN

DRIVER FOR SIMPLE STEAMCYCLE

RESOLUTION WITH MINPACK 1

JAVA VERSION 1.7 OR 2.7

SOMMAIRE

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 steam 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 turbine used will be initially modeled by a law of Stodola (driver class SimpleSteamPlantDriverOptim), then (driver class MappedTurbineSteamPlantDriver), with a mapping defined in the file dataSingleStageTurbine.txt).

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.

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

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

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

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

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

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

$\varepsilon = f(NTU, R, \text{flow pattern})$

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 turbines in off-design mode

The turbine model that we consider here is based on those described section 4.5 of Part 2. It assumes that the behavior of adiabatic turbines can be represented with reasonable accuracy by two parameters: the classic isentropic efficiency η_s, and a magnitude K_0 called cone or Stodola constant, which characterizes the flow rate.

For the isentropic efficiency, characteristics may be representated with good accuracy by polynomial curves based on the expansion ratio, for example a very simple equation to identify the type $y = a + bx + c\,x^2$ (4), or a more complex (5) analogous to the law of Dehausse often used for positive displacement compressors.

$$\eta_s = K_1 + K_2 \cdot \frac{P_r}{P_a} + K_3 \cdot \left(\frac{P_r}{P_a}\right)^2 \qquad (4)$$

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

The so-called cone rule, due to Stodola, is expressed as (6):

$$\frac{\dot{m}\sqrt{T_{in}}}{P_{in}} = K_0 \sqrt{1 - \left[\frac{P_{out}}{P_{in}}\right]^{(k+1)/k}} \qquad (6)$$

It is only valid as long as the flow remains below a limit value, which is reached when the sonic conditions are established in the stator nozzle throat.

For a turbine stage, the critical ratio is given (for the isentropic) by equation (7). For n stages, it is raised to the power n.

$$\frac{P_{in}}{P_{out}} = \left[\frac{2}{\gamma + 1}\right]^{(\gamma + 1)/2(\gamma - 1)} \qquad (7)$$

In design mode, we determine the rotation speed that provides the desired flow. The calculation is performed taking into account the inlet and outlet stagnation pressures and the flow value of the turbine process in the simulator.

In off-design mode, the expansion ratio determines the flow and η_s, which sets the turbine outlet temperature.

3 Selecting and setting technological screens

Suppose we want to size a simple steam cycle (Figure 1) to provide a power of about 660 MW for a cooling temperature of 8 °C, a maximum cycle pressure of 165 bar and a superheating temperature of 560 °C, the isentropic efficiency of the turbine being close to 0.85.

For the condenser, the cooling flow is 18 t/s.

The efficiency of the machine is 0.336 under these conditions.

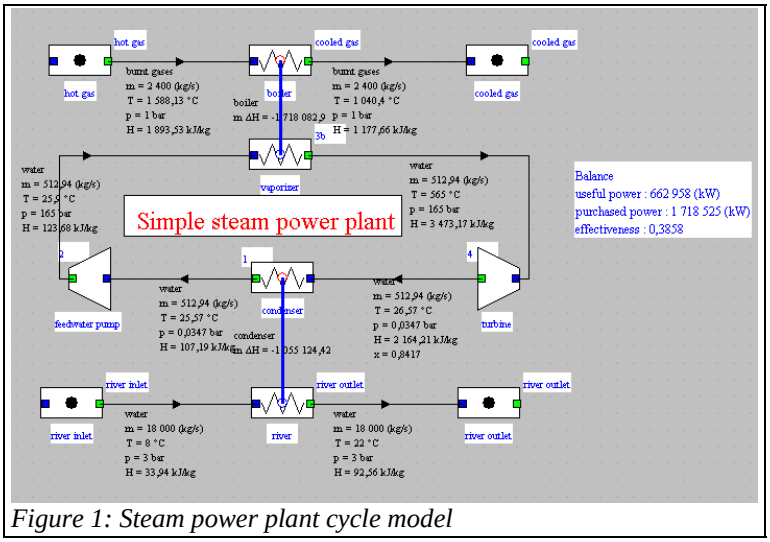

Figure 1: Steam power plant cycle model

The pressure drops will be neglected here, although they are calculated, and the residual velocity losses at the turbine outlet will be considered only at a later time (see Section 8).

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 steam cycle is similar to those presented in the Getting Started guides the software, the main difference being that the boiler is modeled by a single multi-zone heat exchanger, the economizer, the vaporizer and the superheater not being dissociated 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.

[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.

```
//initialisations des PointThopt et des TechnoDesign
//initializations for simulation
boilerUpstream=new PointThopt(proj,"2");
boilerDownstream=new PointThopt(proj,"3b");
condUptream=new PointThopt(proj,"4");
condDownstream=new PointThopt(proj,"1");
hotGases=new PointThopt(proj,"hot gas");
coldGases=new PointThopt(proj,"cooled gas");
inRiver=new PointThopt(proj,"river inlet");
outRiver=new PointThopt(proj,"river outlet");
condDownstream.getProperties();
subst=(rg.corps.Corps)condDownstream.lecorps;

//noms des composants / component names
boilerName="boiler";
condenserName="condenser";
turbineName="turbine";
```

The TechnoDesign are of type TechnoEvaporator, TechnoCondensor and TechnoSimpleTurb, three classes created specifically for this type of components. The first two are used to calculate the boiler and condenser as multi-zone heat exchangers, according to the equations given in Appendix 1. The third implements the equations (4) to (6). We will just explain how to use them, referring the reader to the code of their classes for details.

```
//instanciation des TechnoDesign dans les classes externes
//instanciation of TechnoDesign in external classes
technoBoiler=new TechnoEvaporator(proj, boilerName, hotGases, coldGases,
boilerUpstream, boilerDownstream);
addTechnoVector(technoBoiler);
technoCond=new TechnoCondensor(proj, condenserName, condUptream,
condDownstream, inRiver, outRiver);
addTechnoVector(technoCond);

technoTurbine= new TechnoMultiStageTurb(proj, turbineName,
boilerDownstream, condUptream);
addTechnoVector(technoTurbine);

//initialisation des TechnoDesign dans Thermoptim / TechnoDesign
initialisation
setupTechnoDesigns(vTechno);
```

The last line of code above allows you to transfer these technological screens in the core of the software.

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 Stodola constant, the maximum cycle temperature and pressure, and the cooling fluid temperature. It has options for the algorithm guidance that will be specified later.

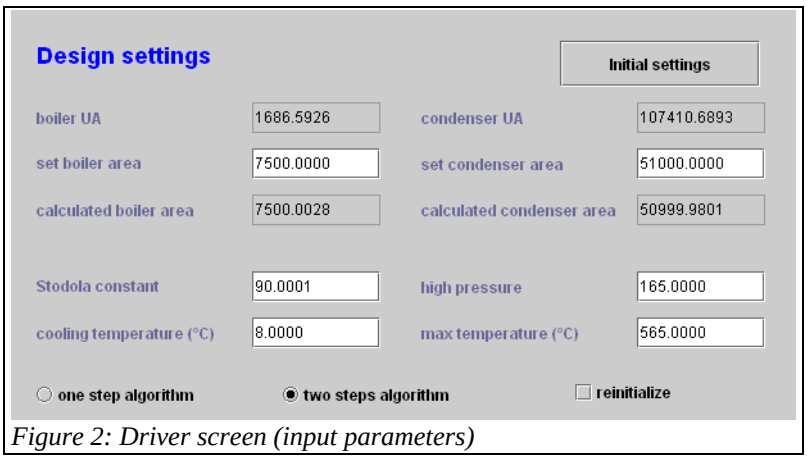

Figure 2: Driver screen (input parameters)

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.

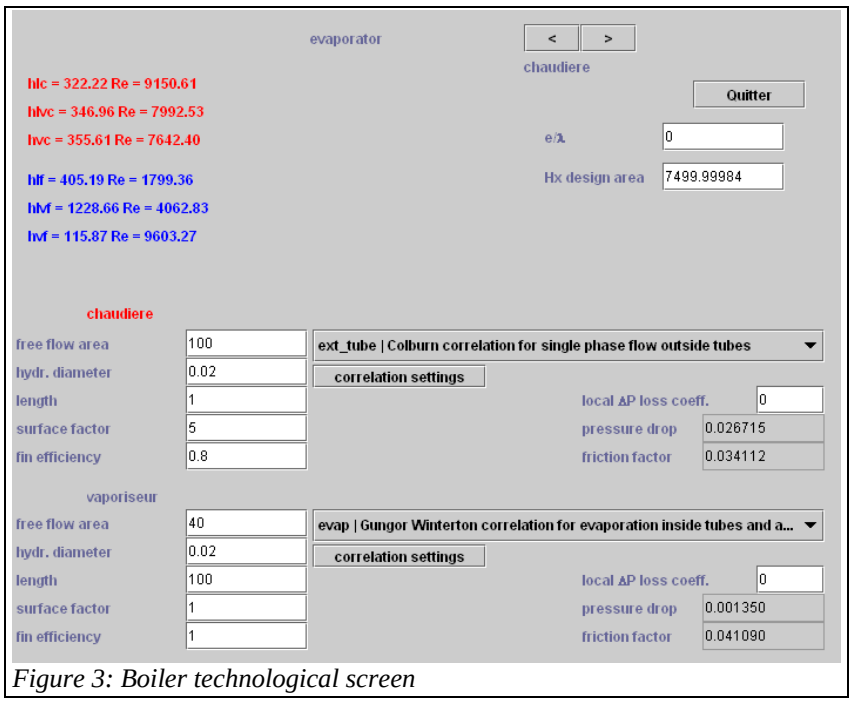

Figure 3: Boiler technological screen

3.3.1 Boiler

We consider that the boiler has a free flow area of 40 m^2 steam side and a hydraulic diameter of 2 cm with a tubing length of 100 m, and gas side a free flow area of 100 m^2 and a hydraulic diameter of 2 cm, for a finned tubes length of 5 m, with a surface factor equal to 5 and a fin efficiency of 0.8. Choose the type of configuration ("evap Gungor Winterton" for evaporation inside the tubes for the "vaporizer" (steam), and "ext tube Colburn correlation" for smoke), and set the screen as shown in Figure 3.

3.3.2 Condenseur

Steam side, the condenser has a free flow area of 123.6 m^2 and a hydraulic diameter of 2 cm, a length of 10 cm, and cooler side a free flow area of 25.5 m^2 and a hydraulic diameter of 1.6 cm, a length of tubing 1 m.

Choose the type of configuration ("cond_ext_hor Levy correlation" for condensation inside the tubes for the "condenser" (steam), and "Mac Adams int_tube correlation" for cooling water), and set the screen as shown in Figure 4.

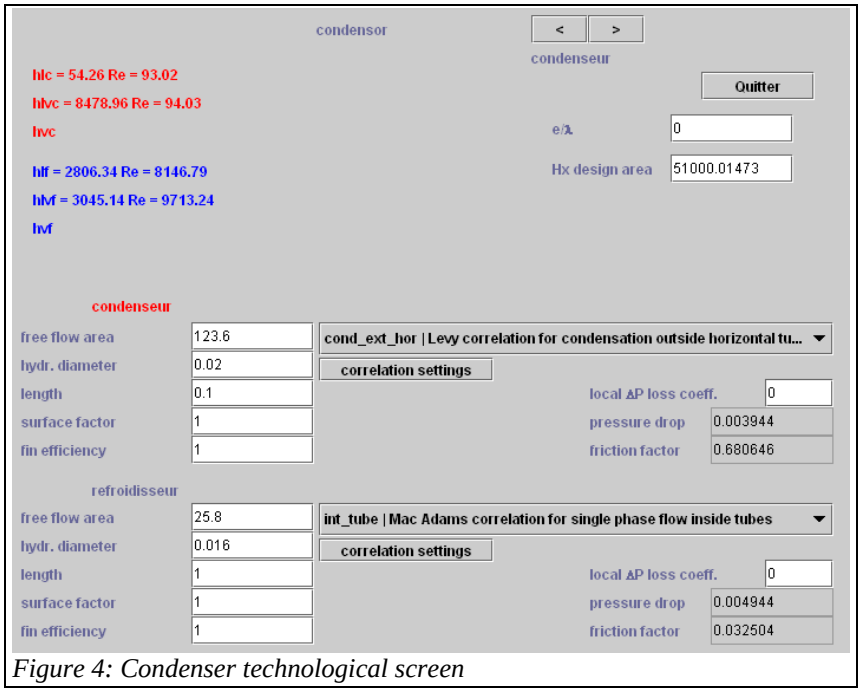

Figure 4: Condenser technological screen

3.3.3 Turbine

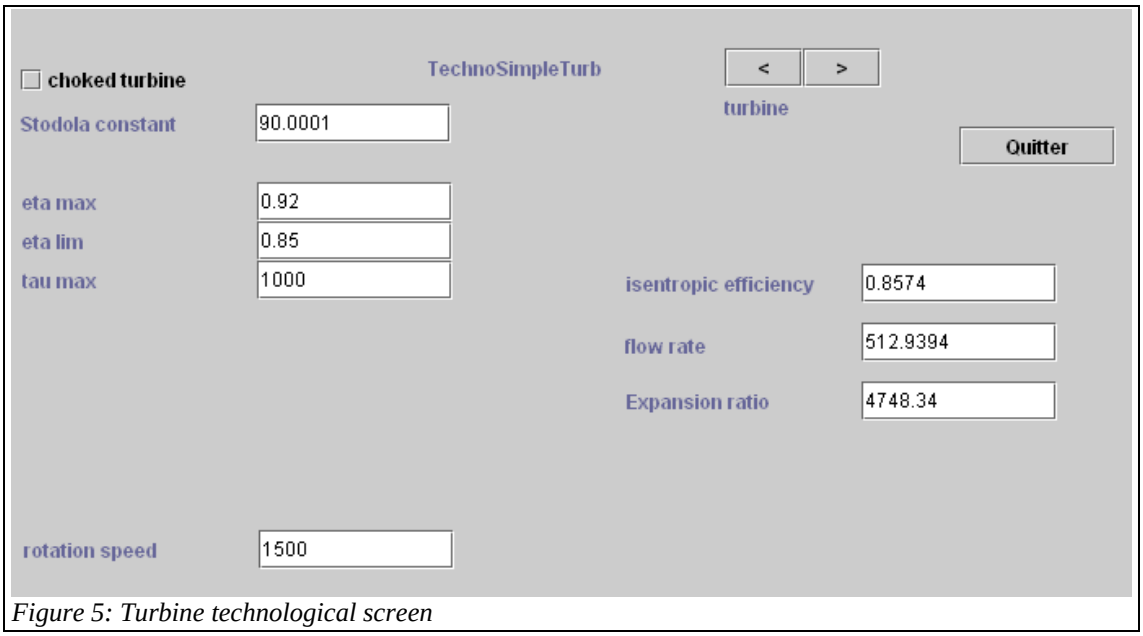

Figure 5: Turbine technological screen

For the turbine (Figure 5), provide the values of the parameters appearing in the isentropic efficiency equations. The rotation speed is one of them, but it is not included in this first model.

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

The results are displayed in the technological screen: exchange surfaces of 7500 and 51 000 m^2 for the boiler and condenser; Stodola constant of 90 for the turbine.

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 turbine Stodola constant, which are used to set a number of values from those of Thermoptim project, such as initial superheating 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 boiler 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
args=new String[2];
args[0]="heatEx";
args[1]=condenserName;
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.

```
condDownstream.getProperties();
condUptream.getProperties();
DTssrefr=condDownstream.DTsat;
```

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

```
inRiver.getProperties();
outRiver.getProperties();
Twater=inRiver.T;
Teau_value.setText(Util.aff_d(Twater-273.15,4));
```

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

```
mCpCalopCond=DeltaH/(outRiver.T-inRiver.T);
mCpRefrigCond=DeltaH/(condUptream.T-condDownstream.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.

```
//initialisations du TechnoDesign
technoCond.makeDesign();
AcondReel=Util.lit_d(technoCond.ADesign_value.getText());
//Ucond sera utilisé ultérieurement pour actualiser UAcond
Ucond=technoCond.UA/AcondReel;
AcalculatedCond_value.setText(technoCond.ADesign_value.getText());
```

The initialization of the turbine allows to know the flow rate and determine the corresponding Stodola constant, which is displayed in the driver screen.

```
 if(!turbineName.equals("")){//initialization of the steam turbine
args=new String[2];
args[0]="process";
args[1]=turbineName;
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();
f=(Double)vProp.elementAt(11);
eta_is=f.doubleValue();

Kh=technoTurbine.getStodolaConstant();
Kh_value.setText(Util.aff_d(Kh,4));
RVL=0;
 }
```

5 Resolution of the off-design system of equations of the steam power plant

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 couple (steam flow, condensing temperature), the other variables being the fluid pressures and the thermal and mechanical capacities. To these two "natural" variables must be added the two intermediate variables UAevap and UAcond (Section 2.1).

The search for this quadruple 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 only 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] = massFlow;
x[3] = UAevap;
x[4] = UAcond;
```

5.1.2 Initialization and call to the resolution algorithm

[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

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.

```
  //parameters and variables are read on the driver screen
AdesignBoiler=Util.lit_d(AdesignBoiler_value.getText());
AdesignCond=Util.lit_d(AdesignCond_value.getText());
Kh=Util.lit_d(Kh_value.getText());
technoTurbine.setKh(Kh);
UAevap=Util.lit_d(UAevap_value.getText());
UAcond=Util.lit_d(UAcond_value.getText());

Pevap=Util.lit_d(Pevap_value.getText());
boilerDownstream.P=Pevap;
boilerDownstream.T=Util.lit_d(Tmax_value.getText())+273.15;
boilerDownstream.update(UPDATE_T,UPDATE_P,!UPDATE_X);
boilerDownstream.getProperties();

boilerUpstream.P=Pevap;
boilerUpstream.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
boilerUpstream.getProperties();
Twater=Util.lit_d(Teau_value.getText())+273.15;
```

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 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 phase change temperatures
if(jcheckReInitialize.isSelected()){//if "reinitialize" is checked
    Tcond= condDownstream.T+Twater-inRiver.T;
}
else{
    Tcond=subst.getSatTemperature(condDownstream.P, 1);
}

inRiver.T=Twater;
inRiver.update(UPDATE_T,!UPDATE_P,!UPDATE_X);
inRiver.getProperties();

condUptream.getProperties();
condDownstream.getProperties();
DTsurch=boilerDownstream.DTsat;
DTssrefr=condDownstream.DTsat;

algorithmResults.setText("");
```

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

```
int m = 4;
int n = 4;

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] = massFlow;
x[3] = UAevap;
```

```
x[4] = UAcond;

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 finies /
precision of finite differences calculus

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 of
finite differences calculus

//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 Fonction 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 four methods resCond(), resFlow(), resAevap() and resAcond defined below.

The call to the first two 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];
massFlow=x[2];
UAevap=x[3];
UAcond=x[4];

Vector vProp;
Double f;
```

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.

```
//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=subst.getSatPressure(Tcond, 1);
condDownstream.T=Tcond+DTssrefr;// DTssrefr est négatif / DTssrefr is negative
condDownstream.P=Pcond;
condDownstream.T=subst.getSatTemperature(condDownstream.P,1)+DTssrefr;// DTssrefr est négatif
condDownstream.DTsat=DTssrefr;
condDownstream.update(UPDATE_T, UPDATE_P, !UPDATE_X, false, UPDATE_DTSAT, false, "", 0.);
condDownstream.getProperties();

double Hl=condDownstream.H;
DHcompr=condDownstream.V*(boilerDownstream.P-condDownstream.P)*100;//prise en compte de la compression liquide / liquid compression
Hl=Hl+DHcompr;
DHcompr=DHcompr*massFlow;

condUptream.P=Pcond;
condUptream.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
condUptream.getProperties();

//new DeltaHevap
DeltaHevap=massFlow*(boilerDownstream.H-Hl);
mCpRefrigBoiler=DeltaHevap/(boilerDownstream.T-condDownstream.T);
double[]res=Util.epsi_NUT(mCpRefrigBoiler,mCpCalopBoiler,UAevap);
epsilon=res[0];
mCpmin=res[1];
Tf=boilerUpstream.T+DeltaHevap/epsilon/mCpmin;

hotGases.T=Tf;
hotGases.update(UPDATE_T,!UPDATE_P,!UPDATE_X);
coldGases.T=Tf-DeltaHevap/gasFlow;
coldGases.update(UPDATE_T,!UPDATE_P,!UPDATE_X);

hotGases.getProperties();
coldGases.getProperties();

//calcul de la turbine / calculation of the turbine
DeltaHcond=DeltaHevap+getTurbinePower();
condUptream.getProperties();
```

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

```
//calcul des premiers résidus / calculation of first residuals
fvec[1] = resCond();
fvec[2] = 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(boilerName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
updateHx(condenserName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
updateHx(boilerName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
updateHx(condenserName, RECALCULATE, !UPDATE_UA, 0, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
updateHx(boilerName, 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:

```
boilerUpstream.getProperties();
boilerDownstream.getProperties();
condUptream.getProperties();
condDownstream.getProperties();
hotGases.getProperties();
coldGases.getProperties();
inRiver.getProperties();
outRiver.getProperties();
```

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

```
technoBoiler.makeDesign();
AcalculatedBoiler_value.setText(technoBoiler.ADesign_value.getText());

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

Finally, the last two residuals are estimated:

```
//calcul des derniers résidus après recalcul des TechnoDesign
(adimensionnés)
// Calculates the last residuals after recalculating the TechnoDesign
(dimensionless)
fvec[3] = resAevap();
fvec[4] = resAcond();
}
```

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(){
//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]

double Toutlet_refrig=0;
mCpRefrigCond=DeltaHcond/(condUptream.T-(Tcond+DTssrefr));
double[] res=Util.epsi_NUT(mCpRefrigCond,mCpCalopCond,UAcond);
epsilon=res[0];
mCpmin=res[1];
double Tinlet_refrig=Twater+DeltaHcond/epsilon/mCpmin;
Toutlet_refrig=Tinlet_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-(Toutlet_refrig-DTssrefr);
System.out.println("resCond: " + z + ", "+ Tcond + ", " + Tevap+ ", "  +
massFlow );
z= 2*z/(Tcond+Toutlet_refrig-DTssrefr);
return z;
 }
      double resAevap(){//renvoie le résidu de la surface de l'évaporateur
UAevap_value.setText(Util.aff_d(UAevap,4));
AcalculatedBoiler_value.setText(Util.aff_d(AevapReal,4));
AevapReal=technoBoiler.A;
double z= AevapReal-AdesignBoiler;
System.out.println("resAevap: " + z + ", "+ UAevap + ", " + UAcond);
System.out.println();

z= 2*z/(AevapReal+AdesignBoiler);
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));
}
eff_value.setText(Util.aff_d(-tauTurb/DeltaHevap,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(condDownstream.P,4));
tauTurb_value.setText(Util.aff_d(tauTurb,4));
DeltaHcompr_value.setText(Util.aff_d(DHcompr,4));
```

6 Equations of the steam power plant in off-design mode

6.1 Boiler balance

The coolant temperature is set by the thermal equilibrium of the boiler, which depends mainly on the one hand on the vaporization temperature and the coolant flow of (smoke in this example), and also the steam flow.

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

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

```
//new DeltaHevap
DeltaHevap=massFlow*(boilerDownstream.H-Hl);
mCpRefrigBoiler=DeltaHevap/(boilerDownstream.T-condDownstream.T);
double[]res=Util.epsi_NUT(mCpRefrigBoiler,mCpCalopBoiler,UAevap);
epsilon=res[0];
mCpmin=res[1];
Tf=boilerUpstream.T+DeltaHevap/epsilon/mCpmin;
```

6.2 Turbine balance

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

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

It is calculated by:

```
 double getTurbinePower(){
Vector vProp;
Double f;

//recalcul de la turbine / Recalculation of the turbine
double eta_is=technoTurbine.getRisentr();
updateprocess(turbineName, "Expansion",RECALCULATE,IS_SET_FLOW,
UPDATE_FLOW, massFlow,  UPDATE_ETA, eta_is);

//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]=turbineName;
vProp=proj.getProperties(args);
f=(Double)vProp.elementAt(4);
tauTurb=f.doubleValue();
double tauIs=tauTurb/eta_is;
return tauTurb;
 }
```

6.3 First law

τ_c being the pump compression work, the first law reads:

$$\Delta H_{cond} = \tau_t + \tau_c + \Delta H_{evap} \qquad (10)$$

It is written in fcn () as:

```
//calcul de la turbine / calculation of the turbine
DeltaHcond=DeltaHevap+getTurbinePower();
condUptream.getProperties();
```

6.4 Condenser balance

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

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

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

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 turbine TechnoDesign) with that of massFlow = x [2]. The calculation of λ is done in theTechnoDesign.

```
 double resFlow(){
//renvoie le résidu du débit-masse
//recalcul du débit d'eau
// Returns the remainder of the mass flow
 // Recalculate the flow of water
double y=technoTurbine.getMassFlow();
double z= massFlow-y;
System.out.println("resFlow: " + z + ", "+ Tcond + ", " + Tevap+ ", "  +
massFlow );
z= 2*z/(massFlow+y);
return z;
}
```

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 Stodola constant, as well as the cycle maximum temperature and pressure, and the coolant temperature.

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.

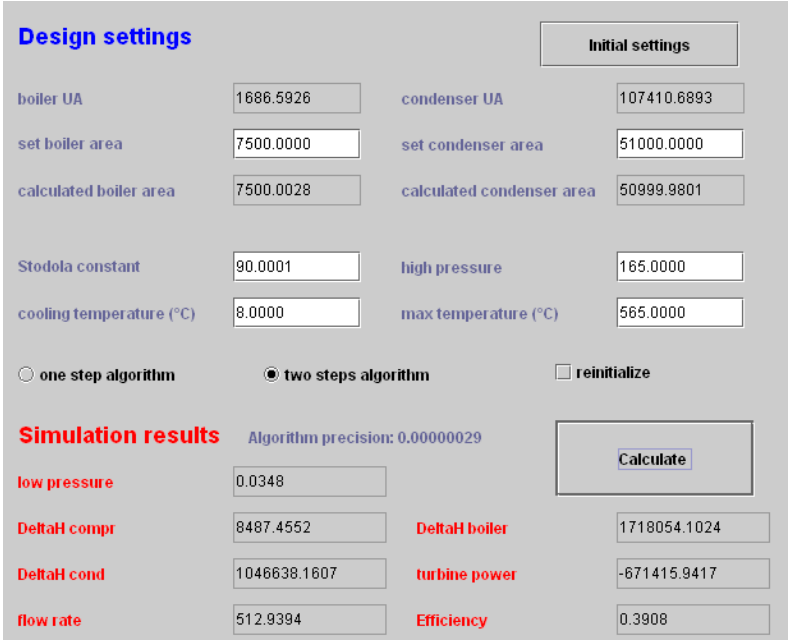

Figure 6: Driver screen

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 coolant for the technological screen setting selected above. The influence of the cycle maximum pressure is given in Figure 8.

In this example, we have varied only two variables, but it would be easy to study the influence of others, 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.

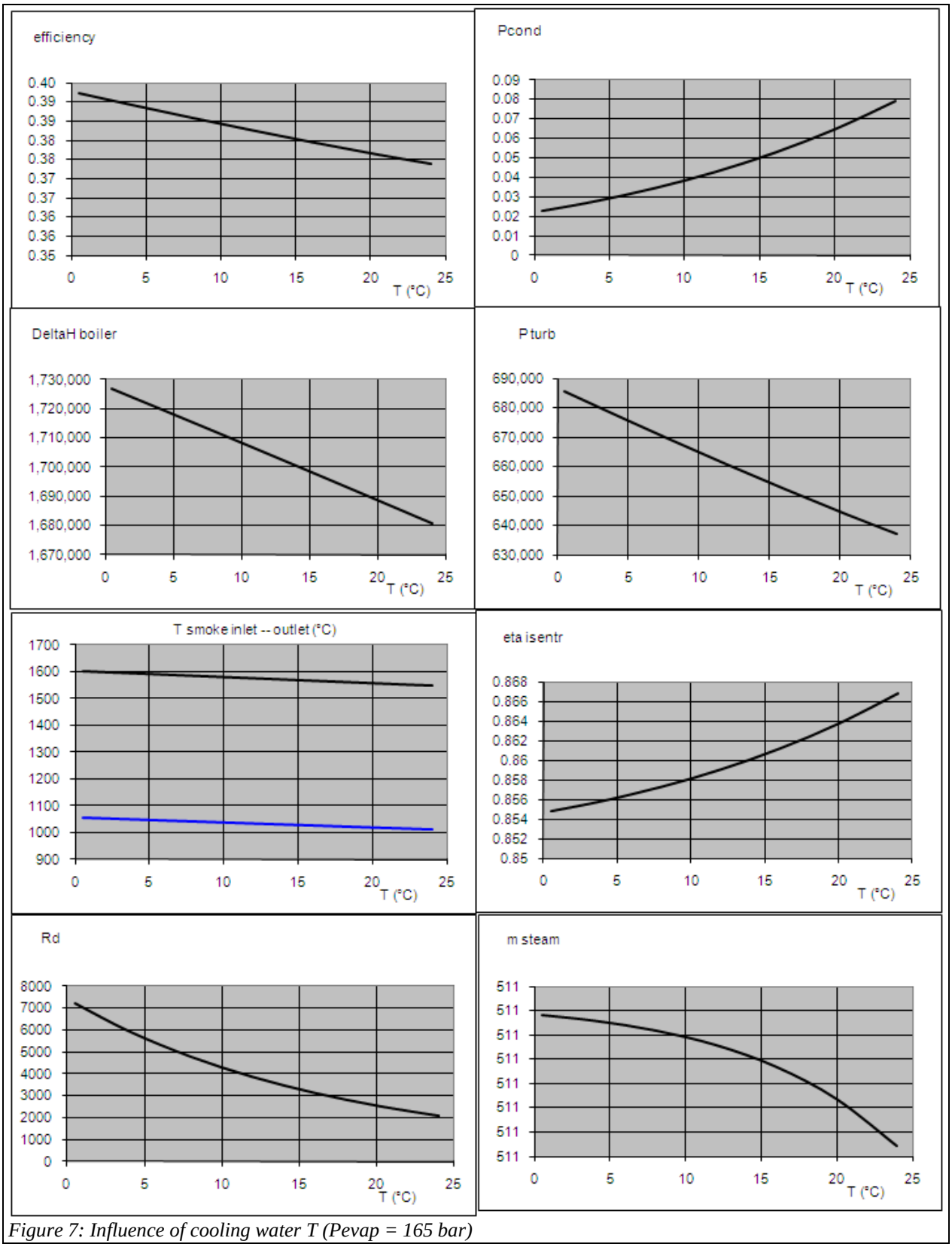

Figure 7: Influence of cooling water T (Pevap = 165 bar)

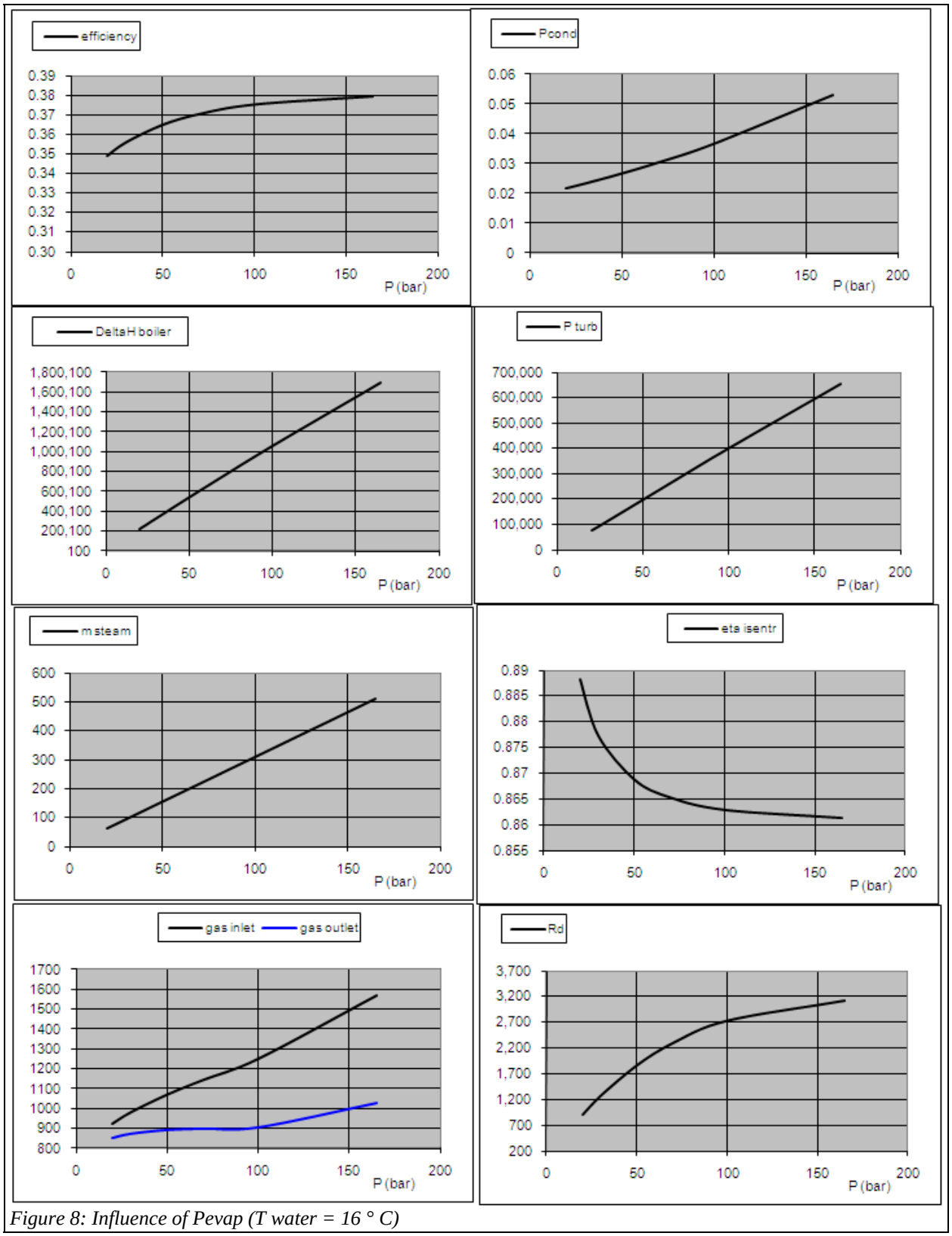

Figure 8: Influence of Pevap (T water = 16 ° C)

8 Cycle taking into account residual velocity losses

When the downstream pressure drops, for a given free flow area, the flow velocity increases, so that the kinetic energy lost may not be negligible. Physically, this means that some of the available power is not recovered on the turbine shaft, and is converted downstream in heat.

The corresponding loss assessment can be made on the basis of the velocity triangle, which assumes the downstream flow section is known, also the circumferential speed U and outlet angle β. In the equation below, C_2 represents the flow velocity Cf with our usual notation.

These losses are equal to $C_2^2/2$, with (figure 9):

$C_t = Cr \cot g\ \beta - U$

$C_2^2 = Cr^2 + Ct^2 = Cr^2 + (Cr \cot g\ \beta - U)^2$

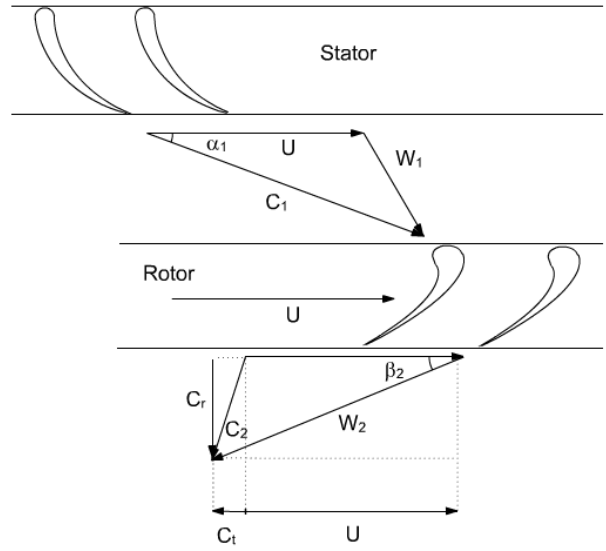

Figure 9: Turbine velocity triangle

In practical terms, if one does not want to modify the isentropic efficiency expression, the calculation of residual velocity losses (RVL) can be performed as follows.

- we first determine the turbine outlet conditions in the absence of RVL;
- knowledge of the specific volume and the exhaust section lets us know Cr and so RVL;
- work done by the turbine is equal to that in the absence of RVL minus RVL. Knowing the isentropic work (in the absence of RVL because we assume the machine is perfect), we deduce a new value of isentropic efficiency;
- recalculation of the turbine with this isentropic efficiency value gives the exit point state taking into account RVL.

8.1 Code changes

The turbine technological class now is TechnoTurb, and getTurbinePower()code is simply modified as follows:

```
RVL=0;
if(technoTurbine.isPVR()){//if we take into account residual velocity
losses / s'il y a prise en compte des pertes par vitesse restante
condUptream.getProperties();
double v=condUptream.V;
double volFlow=massFlow*v;
RVL=technoTurbine.getPVR(volFlow);
tauTurb=tauTurb+RVL*massFlow;
eta_is=tauTurb/tauIs;
if(eta_is<0.2)eta_is=0.2;//pour éviter des valeurs négatives éventuelles /
to avoid any negative value
technoTurbine.risentr_value.setText(Util.aff_d(eta_is,4));
updateprocess(turbineName, "Expansion",RECALCULATE,IS_SET_FLOW,
UPDATE_FLOW, massFlow,  UPDATE_ETA, eta_is);
}
```

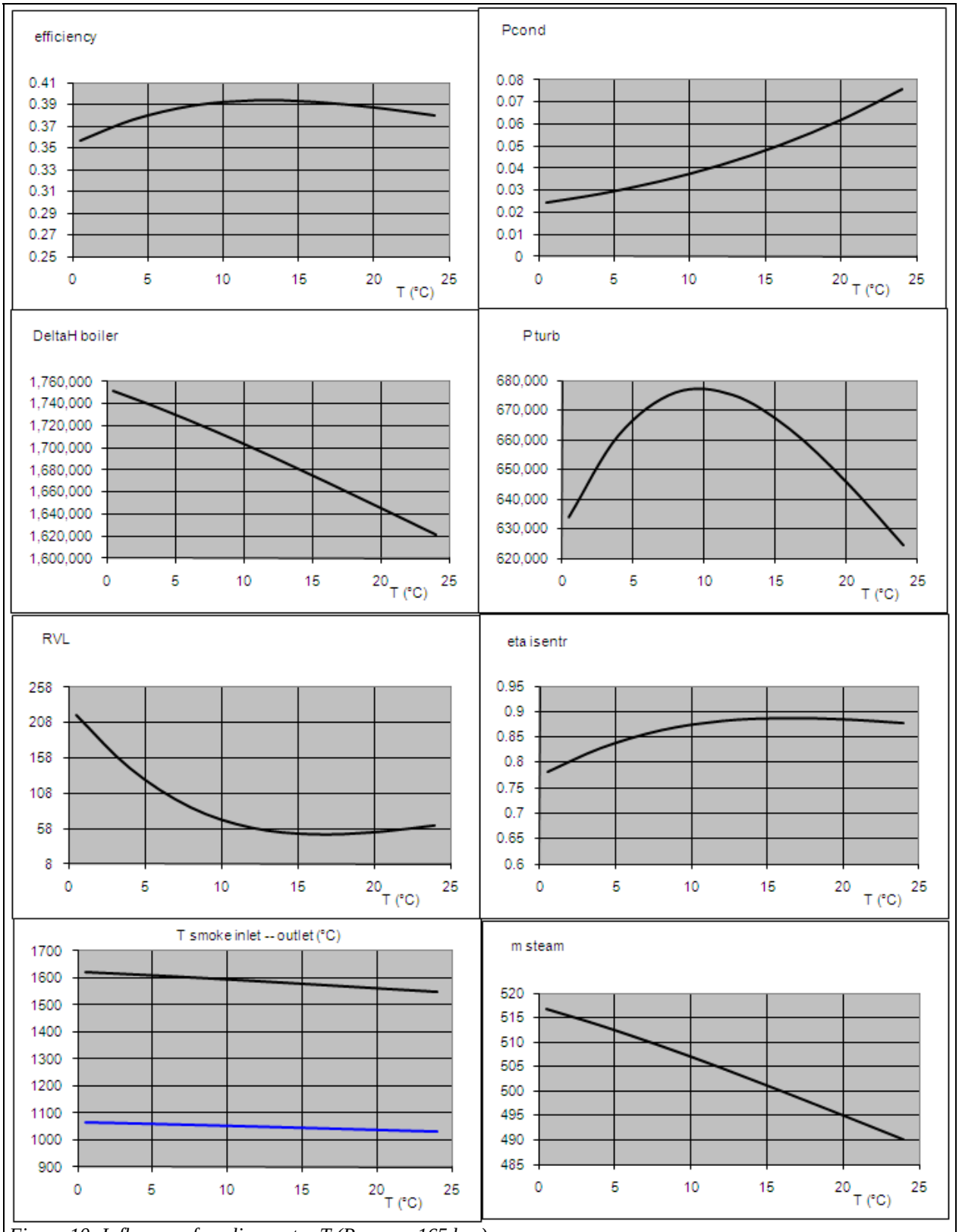

Figure 10: Influence of cooling water T (Pevap = 165 bar)

8.1 New results

With these assumptions, the influence of the cooling temperature is quite different from the previous case, mainly because the residual velocity losses are penalizing the cycle when the cooling temperature is low (Figure 10). It is less sensitive to the influence of pressure (figure 11).

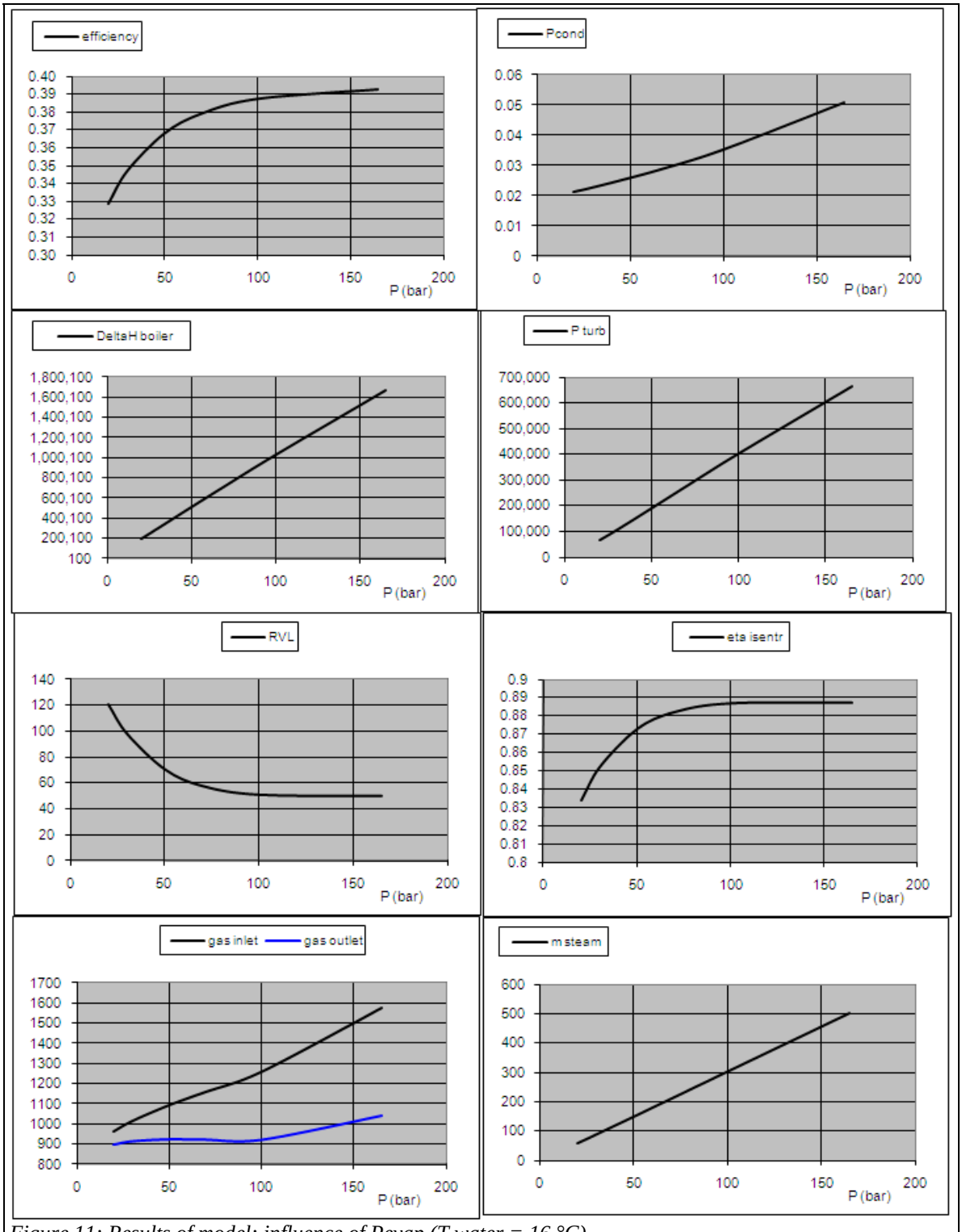

Figure 11: Results of model: influence of Pevap (T water = 16 °C)

9 Modeling with turbine mapping

9.1 Mapping of turbines

In the case of a turbine, although it is also possible to retain the same reference as for a dynamic compressor, the pressure ratio is usually used as abscissa. In ordinate, we find the corrected mass flow or isentropic efficiency of the machine.

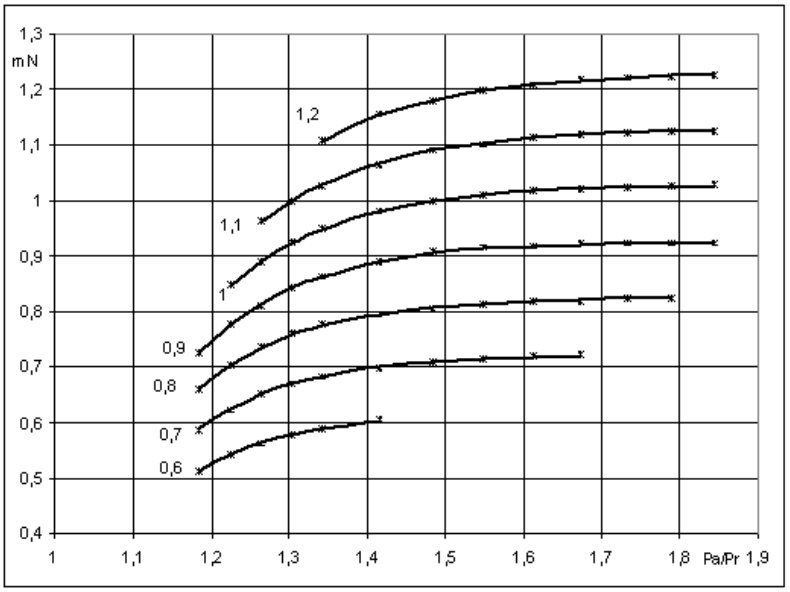

Figure 12: Characteristics (expansion ratio, flow)

The curve parameter is still the corrected rotation speed, which plays a secondary role: efficiency tends to deteriorate only when we want to dramatically reduce the pressure ratio or the speed. This flexibility is particularly due to the flow stability in the blades due to the gradient of pressure therein.

Given the combination of curves, the characteristics (expansion ratio, flow) are not readable, and it is interesting to change the ordinate for better visibility. Since the useful area of the performance map is very small, we can represent them using as abscissa the product of corrected mass flow and velocity, leading to the plots in Figure 12.

These equations are written in dimensionless form and expressed numerically as explained in Volume 4 of Thermoptim reference manual

The mass flow rate depends on the thermodynamic state at the inlet and the expansion ratio.

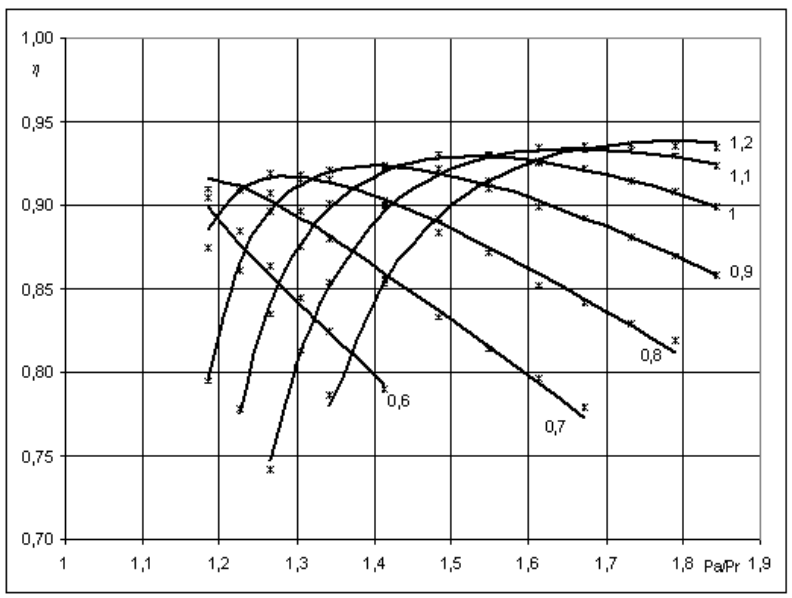

Figure 13: Characteristics (isentropic efficiency, speed)

9.2 Code Changes

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

```
technoTurbine= new MultiStageMappedTurbine(proj, turbineName,
boilerDownstream, condUptream);
```

9.2.1 Initialization of the turbine design

During the turbine initialization, we load the appropriate map, then research, given the inlet conditions, either the speed leading to the nominal expansion ratio, or the number of stages compatible with the selected rotation speed:

```
technoTurbine.setDataFile(technoTurbine.getDataFile());
```

```
technoTurbine.makeDesign();
if(JCheckRotationSpeed.isSelected()){
    //recherche de la vitesse de rotation correspondant à Rp et massFlow
double Ninit=technoTurbine.getNfromRpAndFlow(Rd,
massFlow)/technoTurbine.racT0*Math.pow(boilerDownstream.T,0.5)*technoTurbin
e.Nref;
if(Ninit!=0){//si on est en dehors de la cartographie, on ne fait rien en
dehors du message d'information
technoTurbine.setN(Ninit/technoTurbine.Nref);
technoTurbine.setNparameters();//calcule les paramètres qui dépendent de N
N_value=Ninit;
Nref_value.setText(Util.aff_d(Ninit,4));
}
}
else{
N_value=Util.lit_d(Nref_value.getText())/technoTurbine.Nref;
technoTurbine.setN(N_value);
int nStage=technoTurbine.getStageNumber(N_value,massFlow);
//prévoir test sur valeur de j
nStage_value.setText(Util.aff_i(nStage));
technoTurbine.setStageNumber(nStage);
}
double v=condUptream.V;
double volFlow=massFlow*v;
technoTurbine.getPVR(volFlow);
```

9.2.2 Initialization of off-design calculations

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

```
Kh=Util.lit_d(nStage_value.getText());
technoTurbine.setKh(Kh);
N_value=Util.lit_d(Nref_value.getText())/technoTurbine.Nref;
technoTurbine.setDesignValues();
technoTurbine.setN(N_value);
technoTurbine.setNparameters();
technoTurbine.setStageNumber();
```

9.2.3 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 de la turbine
//update of the turbine corrected speed
technoTurbine.updateN();
technoTurbine.setNparameters();
```

9.2.4 Verification of adaptation of the reduced speed to the mapping

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

```
//vérifie si N réduit est dans les limites de la cartographie
//checks if N reduced is within the map limits
//ce test n'est fait qu'à la fin / test done at the end
technoTurbine.checkMapValidity();
```

9.3 Model Results

The model results are quite consistent with those obtained previously with regard to the influence of the cooling temperature, the only differences being due to the change of characteristics (see Figure 14).

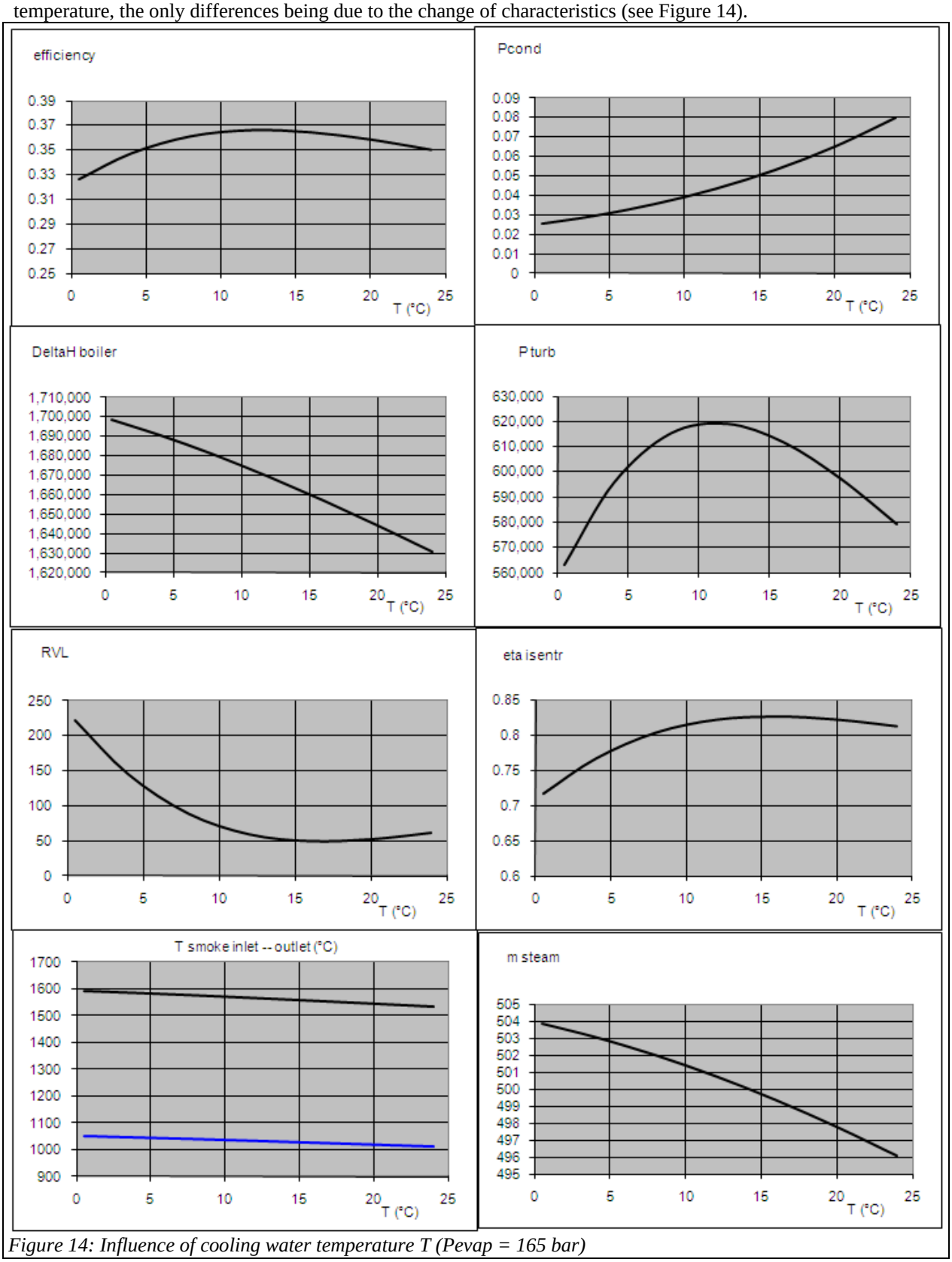

Figure 14: Influence of cooling water temperature T (Pevap = 165 bar)

That of the maximum pressure of the cycle is given in Figure 15.

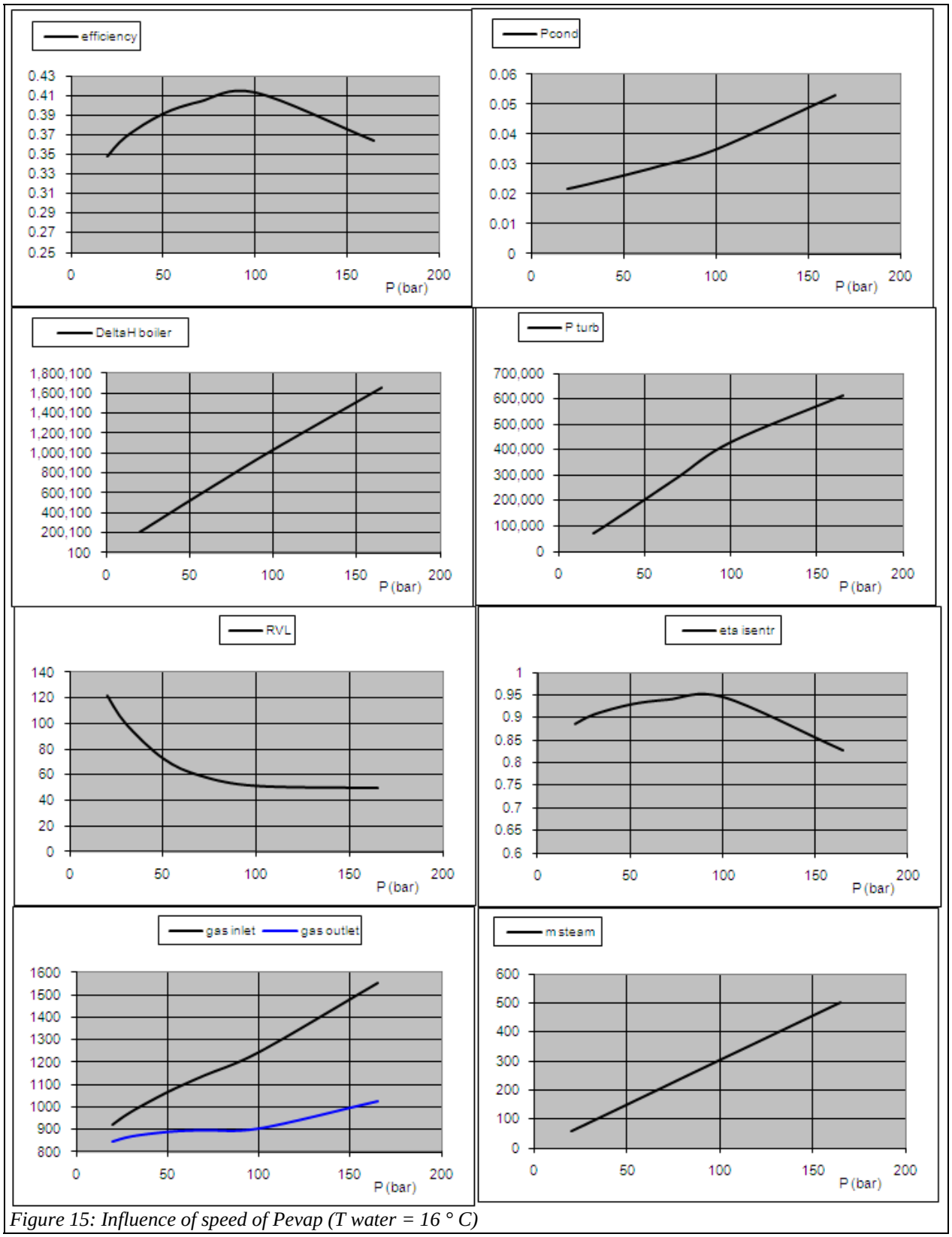

Figure 15: Influence of speed of Pevap (T water = 16 ° C)

Annex 1: Principle of multi-zone heat exchanger calculation

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

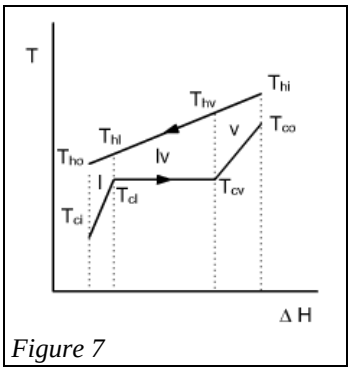

Figure 7

The equations are as follows (Figure 7):

$m_h Cp_h (T_{hi} - T_{hv}) = m_c Cp_{cv} (T_{co} - T_{cv})$
$m_h Cp_h (T_{hv} - T_{hl}) = m_c Cp_{clv} (T_{cv} - T_{cl}) = m_c L_c$
$m_h Cp_h (T_{hl} - T_{ho}) = m_c Cp_{cl} (T_{cl} - T_c i)$
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

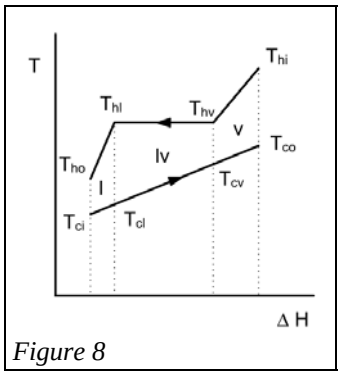

Figure 8

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

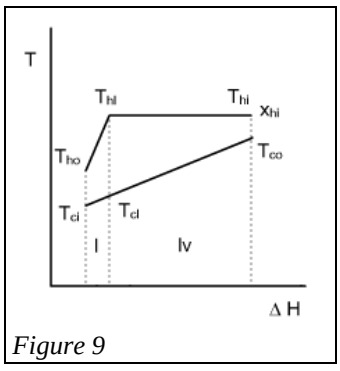

Figure 9

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:

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

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