T H E R M O P T I M ®

OFF-DESIGN

DRIVER FOR PRESSURIZED WATER REACTOR STEAM GENERATOR

RESOLUTION WITH MINPACK 1

JAVA VERSION 2.72 OR 2.82

© R. GICQUEL DECEMBER 2021

TABLE OF CONTENTS

© R. GICQUEL 2021. 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 pressurized water reactor steam generator 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.

2 Principle of 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.

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:

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;
- 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

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

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.

3 Selecting and setting technological screens

Suppose we want to size a PWR steam generator (Figure 1) to provide a power of about 1000 MW.

The processes of the primary circuit are placed in the upper part of the diagram, and those of the secondary circuit below.

A pressurized water flow at 155 bar and 325 °C enters the GV, and exits at about 294 °C.

The feedwater from the secondary circuit enters the GV at 47 bar and 150 °C, and leaves as superheated steam at 298 °C.

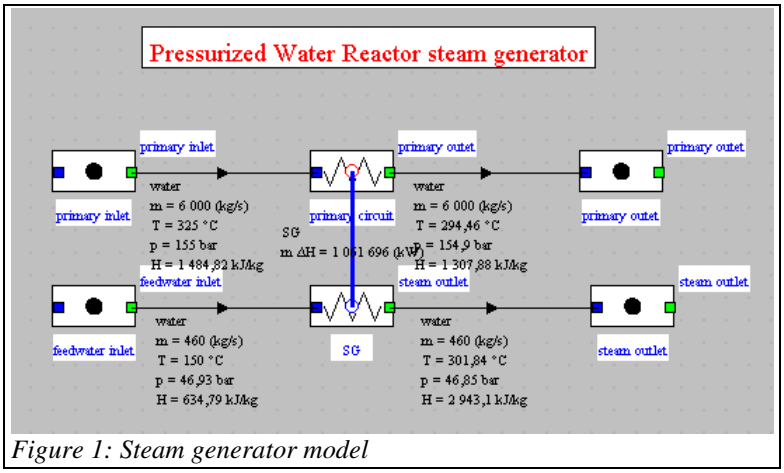

Figure 1: Steam generator model

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.

```
       //initialisations des PointThopt et des TechnoDesign
        //attention : les noms des points et composants doivent être
exacts, sous peine de générer une erreur empêchant le chargement des
TechnoDesign
        amontChaudiere=new PointThopt(proj,"feedwater inlet");
        avalChaudiere=new PointThopt(proj,"steam outlet");
        entreePrimaire=new PointThopt(proj,"primary inlet");
        sortiePrimaire=new PointThopt(proj,"primary outet");
        fluideThermo=(rg.corps.Corps)avalChaudiere.lecorps;

        //noms des composants
          boilerName="SG";
```

The TechnoDesign is of type TechnoSteamGenerator, class created specifically for this type of component. It is used to calculate the boiler as multi-zone heat exchanger, according to the equations given in Appendix 1.

```
          //instanciation des TechnoDesign dans les classes externes
            technoChaudiere=new TechnoSteamGenerator(proj, boilerName,
entreePrimaire, sortiePrimaire, amontChaudiere, avalChaudiere);
          addTechnoVector(technoChaudiere);

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

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

[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 driver screen is given in Figure 2.

It makes it possible to modify on the one hand the surface of the exchanger, and on the other hand the temperature and flow of the primary circuit, as well as the pressure and flow of the secondary circuit. It has options for the algorithm guidance that will be specified later.

Start by clicking "Technological sizing" to instantiate the TechnoDesigns and make an initial technological design, in this case calculate surface of the steam generator corresponding to the project file setup, on the basis of default values for TechnoDesign parameters.

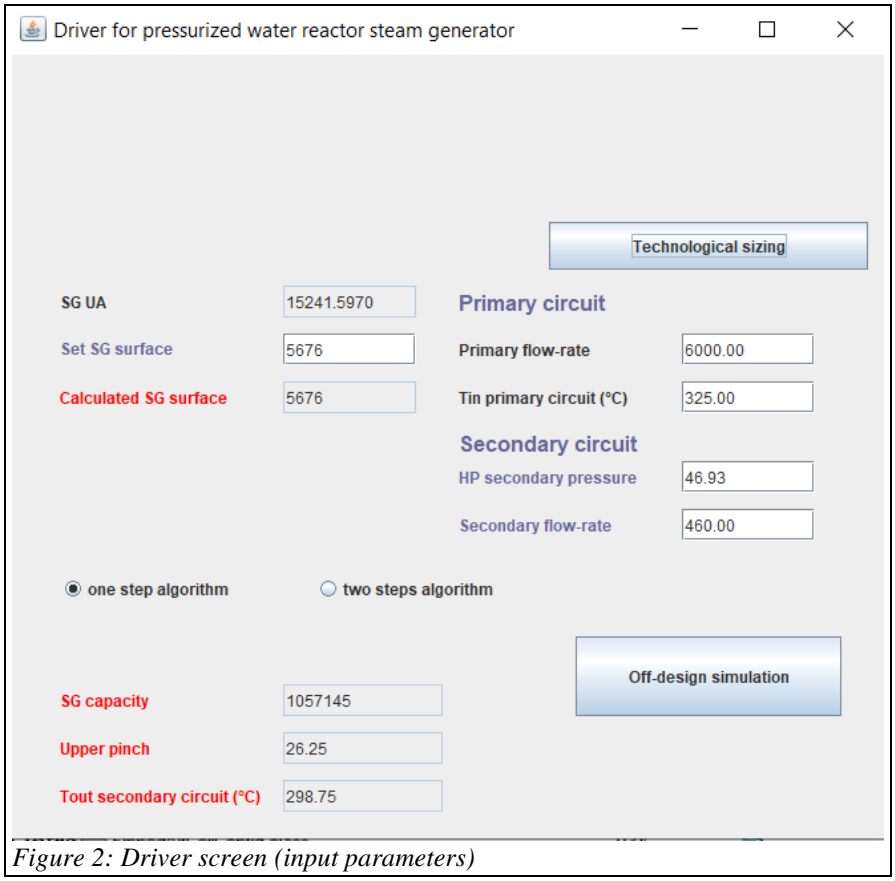

Figure 2: Driver screen (input parameters)

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.

We consider that the boiler has a steam free flow area of 3 m^2 steam side and a hydraulic diameter of 2 cm with a tubing length of 22 m, and in the primary circuit side a free flow area of 5 m^2 and a hydraulic diameter of 5 cm, for a finned tubes length of 22 m. Choose the type of configuration ("ext tube Colburn correlation" for the primary circuit, and "Boiling | Techno.." for evaporation inside the tubes for the "SG" (steam), plus “Saitoh & al.” for NTU correlation and “Sun & Mishima” for pressure drop correlation), and set the screen as shown in Figure 3.

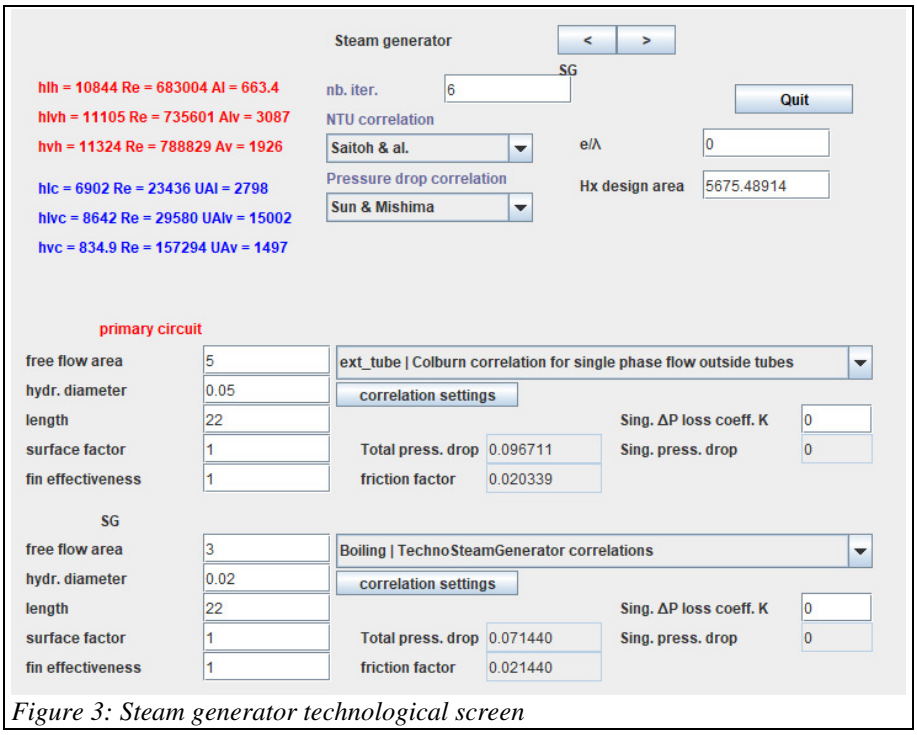

Figure 3: Steam generator technological screen

The results are displayed in the technological screen: exchange surface of 5,340 m2 for the boiler.

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 surface from the initializations corresponding to the design point, which are used to set a number of values from those of Thermoptim project.

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 boiler.

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 get the value of UAcond and the cold fluid name, then the enthalpy DeltaH into play.

```
if(!boilerName.equals("")){//initialisation de la chaudière
  args=new String[2];
  args[0]="heatEx";
  args[1]=boilerName;
  vProp=proj.getProperties(args);
  Double f=(Double)vProp.elementAt(15);
  UAevap=f.doubleValue();
  String fluideChaud=(String)vProp.elementAt(0);
  args[0]="process";
  args[1]=fluideChaud;
  vProp=proj.getProperties(args);
  f=(Double)vProp.elementAt(4);
  double DeltaH=-f.doubleValue();
  f=(Double)vProp.elementAt(3);
  primaryFlow=f.doubleValue();
  primaryFlow_value.setText(Util.aff_d(primaryFlow,2));
  Tinprimaire_value.setText(Util.aff_d(entreePrimaire.T-273.15,2));

  amontChaudiere.getProperties();
  avalChaudiere.getProperties();
  DTsurch=avalChaudiere.DTsat;
  entreePrimaire.getProperties();
  sortiePrimaire.getProperties();
     Tf=entreePrimaire.T;

  mCpCalopChaudiere=DeltaH/(entreePrimaire.T-sortiePrimaire.T);
  mCpRefrigChaudiere=DeltaH/(avalChaudiere.T-amontChaudiere.T);
  DeltaHevap=DeltaH;
```

The getProperties() methods of the PointThopt directly provide the complete thermodynamic state of the upstream and downstream points of the SG, which makes it possible to initialize the superheat.

```
  amontChaudiere.getProperties();
  avalChaudiere.getProperties();
```

```
DTsurch=avalChaudiere.DTsat;
```

In the same way, the value of the primary water temperature is obtained from the simulator and displayed in the driver screen.

```
entreePrimaire.getProperties();
sortiePrimaire.getProperties();
  Tf=entreePrimaire.T;
```

These values are used to initialize the values of the SG heat flows, which will be used later.

```
mCpCalopChaudiere=DeltaH/(entreePrimaire.T-sortiePrimaire.T);
 mCpRefrigChaudiere=DeltaH/(avalChaudiere.T-amontChaudiere.T);
 DeltaHevap=DeltaH;
 UAevap_value.setText(Util.aff_d(UAevap,4));
```

The TechnoDesign is then initialized in turn, which makes it possible to know the necessary exchange surface given the sizing carried out, then the pressure drops are updated.

```
//initialisations du TechnoDesign
    //initialisations du TechnoDesign
    technoChaudiere.makeDesign();
    AevapReel=Util.lit_d(technoChaudiere.ADesign_value.getText());
    Uevap=technoChaudiere.UA/AevapReel;
    AcalculatedChaudiere_value.setText(Util.aff_d(technoChaudiere.A,0));
```

The value of the pinch inside the SG is also an important parameter to follow:

```
    pinch_value.setText(Util.aff_d(technoChaudiere.DTmin,2));
```

5 Resolution of the off-design system of equations of the steam generator

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, we will retain the primary return temperature, the related variable being the thermal power involved.

The search for this temperature corresponds to that of the solution of a relatively complex set of nonlinear equations that involves those we have just 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

To solve the problem, we operate in two stages: we start by performing the technological sizing for the nominal point, then we look for off-design mode the setting allowing us to obtain the same technological sizing for inlet conditions and constraints different from those of the nominal regime.

One of the difficulties of the problem is that it is necessary to reconcile two methods of calculation:
- that of Thermoptim's conventional (phenomenological) screens, which do not know the technological sizing parameters and carry out their calculations without taking them into account
- that of TechnoDesign, which take into account the parameters of technological sizing.

In practice, this means that the driver performs the calculations in off-design mode and determines how the phenomenological screens must be reconfigured to keep the whole thing consistent.

In the specific case of our SG, the driver is looking for a solution such that the sizing carried out for a state other than the nominal one leads to a surface of the same value as that retained by the user. It varies the primary return

temperature until the correct surface is obtained, the U values depending on the flow rates and temperatures, and that of UA of the balance of the exchanger.

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").

Users 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 they start the calculations from the General screen of technological screens, users can furthermore, if they wish, interrupt the calculations properly by clicking the "Stop" button, which allows them 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] = sortiePrimaire.T;
```

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 exchange surface, the temperature and flow of the primary circuit, as well as the pressure and flow of the secondary circuit. Other values are those of the simulator screens.

In order to facilitate the convergence of the algorithm, we set up a ramp between the initial values and those we are trying to reach.

```
        //mise en place d'une rampe de variation des valeurs

        avalChaudiere.getProperties();
    entreePrimaire.getProperties();
        double AdesignChaudiereOld=AdesignChaudiere;
        double PevapOld=avalChaudiere.P;
        double massFlowOld=massFlow;
        double entreePrimaireTOld=entreePrimaire.T;
        double primaryFlowOld=primaryFlow;
```

[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

```
            UAevap=Util.lit_d(UAevap_value.getText());
            double
AdesignChaudiereTarget=Util.lit_d(AdesignChaudiere_value.getText());
            double PevapTarget=Util.lit_d(Pevap_value.getText());
            double massFlowTarget=Util.lit_d(massFlow_value.getText());
            double
entreePrimaireTTarget=Util.lit_d(Tinprimaire_value.getText())+273.15;
            double primaryFlowTarget=Util.lit_d(primaryFlow_value.getText());
            double nRampe=20;
```

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

```
                    int m = 1;
                    int n = 1;

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

                    x[1] = sortiePrimaire.T;

                    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
                double epsfcn = 1.e-6;//précision calcul des différences finies

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

                //appel modifié de lmdiff avec modification précision calcul des
différences finies
                    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 residual, 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 is a method resAevap() defined below.

The call is done before recalculation of the simulator and theTechnoDesign.

```
      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
            sortiePrimaire.T=x[1];
            sortiePrimaire.update(UPDATE_T,!UPDATE_P,!UPDATE_X);
            sortiePrimaire.getProperties();

        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.

```
       DeltaHevap= (entreePrimaire.H-sortiePrimaire.H)*primaryFlow;
        avalChaudiere.H=amontChaudiere.H+DeltaHevap/massFlow;

avalChaudiere.T=avalChaudiere.corps.lecorps.getT_from_hP(avalChaudiere.H,avalChaud
iere.P);
        // pour bien faire, il faudrait calculer X, ce qui peut se faire assez
facilement
        //et afficher un message si X<=1
        avalChaudiere.update(UPDATE_T,!UPDATE_P,UPDATE_X);
        avalChaudiere.getProperties();

        mCpRefrigChaudiere=DeltaHevap/(avalChaudiere.T-amontChaudiere.T);
        mCpCalopChaudiere=DeltaHevap/(entreePrimaire.T-sortiePrimaire.T);
```

The second step is to calculate the exchanger by the NUT method, which provides UAevap.

```
        double R=mCpRefrigChaudiere/mCpCalopChaudiere;
        if(mCpRefrigChaudiere<mCpCalopChaudiere) {
             mCpmin=mCpRefrigChaudiere;
        }
        else {
             mCpmin=mCpCalopChaudiere;
             R=mCpCalopChaudiere/mCpRefrigChaudiere;
        }
        epsilon=(avalChaudiere.T-amontChaudiere.T)/(entreePrimaire.T-
amontChaudiere.T);

        double NUT=Util.NUT_epsi(epsilon,R);
        UAevap=NUT*mCpmin;

        if(debug)System.out.println("\nDeltaHevap: " + DeltaHevap
                   +"  mCpRefrigChaudiere: " + mCpRefrigChaudiere
                   +"  mCpCalopChaudiere: " + mCpCalopChaudiere);
        if(debug)System.out.println("UAevap: " + UAevap+"  epsilon: " + epsilon+"
mCpmin: " + mCpmin
                   +"  NUT: " + NUT+"  R: " + R
                   );
```

The third step is to recalculate the simulator so that all its screens are updated. This is done several times so that the calculations stabilize. We then recalculate the TechnoDesign, then estimate the residue.

```
            //mises à jour du simulateur avant recalcul du TechnoDesign
            //on les exécute 3 fois par sécurité, mais ce n'est pas optimisé
          for(int j=0;j<3;j++)proj.calcThopt();
        updateHx(boilerName, RECALCULATE, UPDATE_UA, UAevap, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
```

```
        updateHx(boilerName, RECALCULATE, UPDATE_UA, UAevap, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);
        updateHx(boilerName, RECALCULATE, UPDATE_UA, UAevap, !UPDATE_EPSI, 0,
!UPDATE_DTMIN, 0);

        amontChaudiere.getProperties();
        avalChaudiere.getProperties();
        entreePrimaire.getProperties();
        sortiePrimaire.getProperties();

          technoChaudiere.makeDesign();
          AcalculatedChaudiere_value.setText(Util.aff_d(technoChaudiere.A,0));

          //calcul du résidu
            fvec[1] = resAevap();
}
```

5.1.4 Residual function

In this case, the residue has been standardized to balance the weights of the different gap functions, using a simple method, but valid only if the value to be reached is not zero, which is always the case here.

```
       double resAevap(){//renvoie le résidu de la surface de l'évaporateur
         UAevap_value.setText(Util.aff_d(UAevap,4));
           AevapReel=technoChaudiere.A;
           AcalculatedChaudiere_value.setText(Util.aff_d(AevapReel,0));
           double z= AevapReel-AdesignChaudiere;
           z= 2*z/(AevapReel+AdesignChaudiere);
           if(debug)System.out.println("AevapReel: " + AevapReel + ",
AdesignChaudiere "+   AdesignChaudiere);
           if(debug)System.out.println("resAevap: " + z + ", UAevap "+  UAevap);
           if(debug)System.out.println();

        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] );
                    System.out.println();

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

               if(twoStepAlgorithm.isSelected()){
                   epsi=0.00005;
                   epsfcn = 1.e-9;//précision calcul des différences finies
```

```
                    //appel modifié de lmdiff avec modification précision calcul des
différences finies
                        optimization.Minpack_f77.lmdif2_f77(this, m, n, x, fvec,
epsi, epsfcn, info);
                        residu1 = optimization.Minpack_f77.enorm_f77(m,fvec);

                        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]  );
                        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,0));
                massFlow_value.setText(Util.aff_d(massFlow,4));
                Pevap_value.setText(Util.aff_d(Pevap,4));
                pinch_value.setText(Util.aff_d(technoChaudiere.DTmin,2));
                Tmax_value.setText(Util.aff_d(avalChaudiere.T-273.15,2));
            }
```

6 Use of the driver

6.1 Off-design simulation

The full screen of the driver is given in Figure 4.

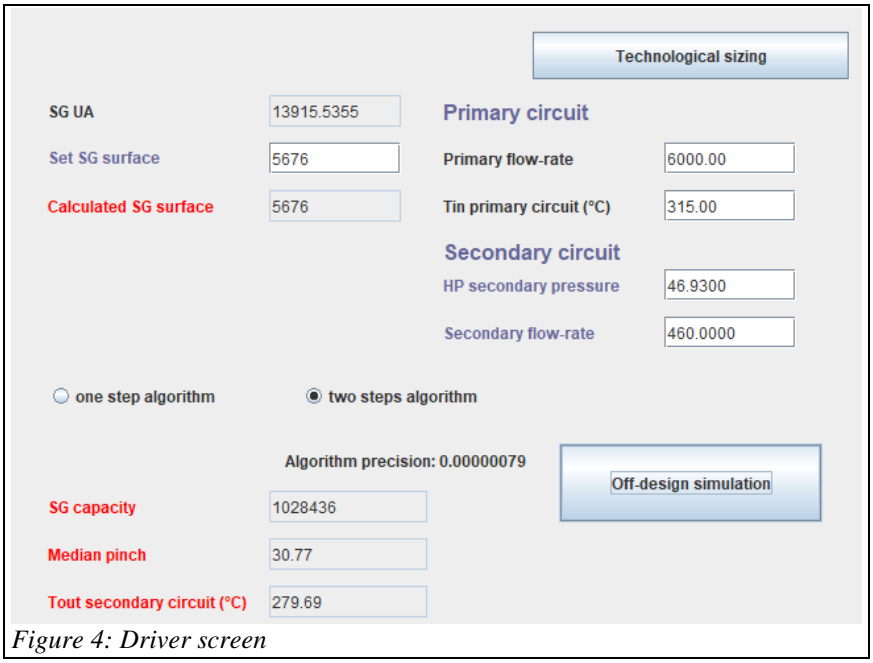

Figure 4: Driver screen

If this has not been done, start by clicking on "Technology Sizing" to initialize the driver.

Then enter the values you want to change (here the temperature of the primary circuit, which goes to 315 °C instead of 325), being careful not to deviate too quickly from the starting point, because convergence difficulties may appear and force you to stop Thermoptim. So gradually increment the variables you want to modify, and make intermediate backups that you can start again in case of problems.

The results are displayed on the screen once convergence is achieved. You can use Thermoptim's post-processing macro to use the resulting project files as results.

The value of the internal pinch in the SG is displayed. The phenomenological screen of the exchanger does not see this pinch, while the TechnoDesign chosen in the driver (a TechnoSteamGenerator) performs its calculations by distinguishing the three internal areas and therefore takes into account the pinch.

Monitor its value. If it becomes smaller than 5°C, you are likely to encounter numerical problems that reflect the fact that the SG can no longer work in practice.

For the same reason, it must be taken in mind that the value of the LMTD indicated in the screen of the exchanger is misleading because the SG is represented by a single exchanger. The apparent LMTD remains roughly unchanged around 70°C, while the pinch increases from 26.25 to 30.7°C.

The SG capacity has decreased from 1.057 MW to 1.028 MW.

6.2 Model limits

This model has only one component, the steam generator. In a real cycle, there would of course be others, in particular an HP turbine crossed by a flow rate that varies according to its conditions of admission, for example by a law of the Stodola type. The flow rate of the secondary circuit would therefore vary instead of being set as in this driver.

In addition, we used the available TechnologySteamGenerator, but it would be possible to model the exchanges in the SG by a custom-built TechnoDesign. In the same way, the correlations chosen are those available in the external classes distributed with Thermoptim, but it would be more accurate to select a specific one.

Annex 1: Principle of multi-zone heat exchanger calculation

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

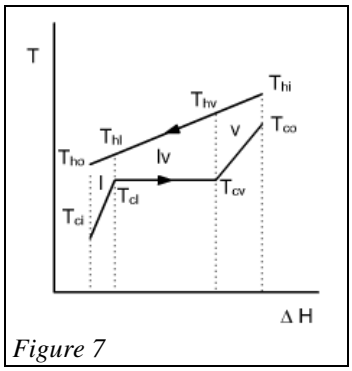

Figure 7

The equations are as follows (Figure 7):

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

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

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

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

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

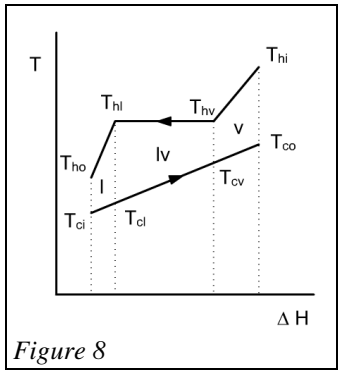

Figure 8

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

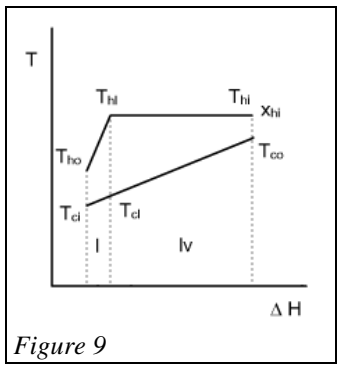

Figure 9

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:

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

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