T H E R M O P T I M ®

CALCULATION OF MOIST GAS

FROM

EXTERNAL CLASSES

VERSION JAVA 1.5

CONTENTS

CALCULATIONS OF MOIST GAS IN THERMOPTIM

We will assume in what follows the reader is sufficiently familiar to both the use of external classes and thermodynamics of moist gas. The definitions, notations and equations are those of Volume 1 of the book Energy Systems.

INTRODUCTION

We call moist mix, and by abuse of language **moist gas**, a mixture of a gas that does not condense, which we call the **dry gas**, and **water** that could condense.

Since its inception, Thermoptim has functions for calculating properties of moist gases and points, but these are generally used for particular calculations, decoupled from standard thermodynamic cycle calculations, such as for air conditioning treatment. It is for this reason that moist processes have no symbol in the diagram editor.

We assume in what follows that the reader is sufficiently familiar with both external class use and moist gas thermodynamics.

For convenience we use the term **standard** when we refer to Thermoptim environment except moist calculations.

The standard Thermoptim cycle computing environment and that of moist calculations are not directly compatible for two reasons:

- following use in moist calculations, the values of thermodynamic functions are generally referred to the dry gas, whose composition is invariant, whereas the standard calculations performed in Thermoptim are relative to the actual composition of the gas. The values to which they lead are called specific, to distinguish them from others;
- furthermore, reference temperatures and pressures not being the same in both environments, it is necessary to do conversions when switching from one to another.

However, a number of thermodynamic cycles involve changes in gas moisture, and it was unfortunate not to be able to model them easily with Thermoptim. That is why the Thermoptim functions for calculating properties of moist gases and points have been made available from external classes. This section explains how to use them.

A moist gas can be represented in Thermoptim in two equivalent ways, either directly as a compound substance comprising at least two components: either H_2O and another gas, pure or compound, or as a dry gas of known specific humidity. The first way has the advantage that the composition of moist gas is available at any time. However, it implies, for the same dry gas, creating a new substance for each value of moist moisture. The second presentation is itself much more concise, since it only uses the invariant gas and the humidity value. This is why it is used by moist gas calculation functions, while the first is the rule in the standard Thermoptim cycle computing environment.

Let us recall that we call **relative humidity ε** the ratio of the partial pressure of water vapor divided by its saturation vapor pressure at the temperature of the mixture, and that, by definition, index gs corresponding to the dry gas, **absolute or specific humidity** w is equal to the ratio of the mass of water contained in a given volume of moist mixture to the mass of dry gas contained in this volume, namely:

$$w = \frac{y_{H_2O}}{y_{gs}} = \frac{M_{H_2O} x_{H_2O}}{M_{gs} x_{gs}} = \frac{18\, x_{H2O}}{M_{gs}\,(1 - x_{H2O})}$$

This relationship allows the calculation of w when we know the moist gas composition.

METHODS AVAILABLE IN THE EXTERNAL CLASSES

Generally, moist gas calculations are made (from external classes) at one point, but a method allows to directly modify the humidity of a gas. The gas humidity is entered giving either its absolute humidity w or its relative humidity ε. Both methods are defined in class ExtProcess, of which derive all external processes and nodes.
Method **updatepoint("pointName",** ...) can do moist calculations while method **getPointProperties("pointName")** recovers in addition to the standard properties (P, T, h, s ...), a point moist properties with the following values: Wpoint for absolute humidity w, Epsipoint for relative humidity ε, Qprimepoint for specific enthalpy q', Tprimepoint for adiabatic temperature t' (°C), Trpoint for dew temperature tr (°C), VPrimepoint for specific volume vs, Condpoint for condensates, and M_secpoint for molar mass of dry gas.

Search of a point humidity

When the point state has been calculated, w is known and method **getPointproperties("nomPoint")** allows to access it directly.

When the gas composition is determined by programming, it may be necessary to recalculate w, which can be done by the following formula:

// $w = \frac{M_{H_2O} x_{H_2O}}{M_{gs} x_{gs}}$

/ / Calculate the gas absolute humidity

double inlet_w=18.01528*fractH2OFuel/fuelM/(1-fractH2OFuel);

Updating a point moist properties

Method updatepoint(String name, boolean updateT, double T, boolean updateP, double P updateX boolean, double x, boolean melHum, String task, double value) is a generic method for update and recalculation of a point state variables, which has been generalized to allow for moist calculations:

```
public void updatepoint(String name, boolean updateT, double T,
    boolean updateP, double P, boolean updateX, double x,
    boolean melHum, String task, double value){
          String[] args=new String[2];
     Vector vPoint=new Vector();
     vPoint.addElement(name);
     vPoint.addElement(Util.aff_b(updateT));
     vPoint.addElement(Util.aff_d(T));
     vPoint.addElement(Util.aff_b(updateP));
     vPoint.addElement(Util.aff_d(P));
     vPoint.addElement(Util.aff_b(updateX));
     vPoint.addElement(Util.aff_d(x));
     vPoint.addElement(Util.aff_b(melHum));
     vPoint.addElement(task);
     vPoint.addElement(Util.aff_d(value));
     proj.updatePoint(vPoint);

          }
```

As shown in this code, it builds a Vector and then calls on the Project method updatePoint ().

If boolean melHum is false, the point update is for T, P, or x, depending on whether booleans updateT, updateP updateX are true or false: this is a standard property update without moist calculations.

If boolean melHum is "true", only moist calculations are made, even if updateT, and updateP updateX are "true".

These calculations are defined by two parameters **task** and **value**.

task is a String specifying the type of calculations to perform, and **value a** double providing the value of the variable to modify.

1) Calculations without changing the gas composition

The calculations being performed with respect to dry gas, the gas composition is not changed.

Set specific humidity w

If task ="setW and calculate all", Thermoptim sets w (passed in value) and calculates all moist properties.

When the temperature of a point is high, convergence problems may arise in calculating the wet bulb temperature t'. To circumvent this difficulty, the setting below only calculates specific enthalpy.

If task ="setW and calculate q'", Thermoptim sets w (passed in value) and calculates all moist properties except t'.

If task = "calcWsat" Thermoptim calculates wsat and all moist saturation properties except t'.

Set the relative humidity ε

If task = "setEpsi" Thermoptim sets ε (passed in value).

If task = "setEpsi and calculate", Thermoptim sets ε (passed in value) and calculates all moist properties except t'.

2) Changing the gas composition

2.1 by operating indirectly from a point

Method updatePoint() allows to alter the composition of a gas, with the following settings:

If task = "modHum" Thermoptim changes the composition of the gas so that its moisture equals Wpoint (there is then no need to pass a value).

If task = "setGasHum" Thermoptim changes the composition of the gas so that its moisture is equal to w (passed in value).

2.2 by operating directly on the gas

It is also possible to change the humidity of a gas regardless of a point state, using method **updateGasComp()** of GazIdeal, accessible by **public void updateGasComposition(Vector Vcomp)** of Corps: if the first Vcomp element is an Integer of negative value, a particular treatment is made. The absolute humidity passed as third Vcomp element is set on the gas.

```
else{//modifications gaz humides
   String task=(String)vComp.elementAt(1);
   String value=(String)vComp.elementAt(2);
   if(task.equals("setGasHum")){//sets the gas humidity
      double w=Util.lit_d(value);
      setGasHum(w);
   }
}
```

The example below, from class BiomassCombustion shows how to change the composition of a dry gas to match the moist gas whose water mole fraction is fractH2Ofuel (see previous section):

```
 // Shaping the Vector
```

```
Vector vComp=new Vector();
 vComp.addElement(new Integer(-1));
 vComp.addElement("setGasHum");
 vComp.addElement(Util.aff_d(inlet_w));
// Change the gas composition
NewFuelSubstance.updateGasComposition(vComp);
```

DISPLAY IN SPECIFIC UNITS

To give the possibility to change the reference system, a display option in specific units was added in the global Thermoptim settings screen. For moist gases, the flow rate shown is that of the dry gas, which is invariant, and the enthalpy H is replaced by the specific enthalpy q '.

USAGE EXAMPLE: SATURATOR MODEL

In a humid air gas turbine, the capacity of the machine and the cycle efficiency are increased by humidifying the air in a saturator before entering the combustion chamber. The cycle is quite complex to optimize heat recovery, but the saturation model is relatively simple (Figure 1): water at a temperature of about 280 °C enters the saturator, where it is contacted with a stream of hot (200 °C) and relatively dry air leaving the compressor. A portion of the water is vaporized and used to increase the humidity of the air coming out close to being saturated. The remaining water is recycled. It is assumed here that the saturator is adiabatic and that water and air exit at the same temperature.

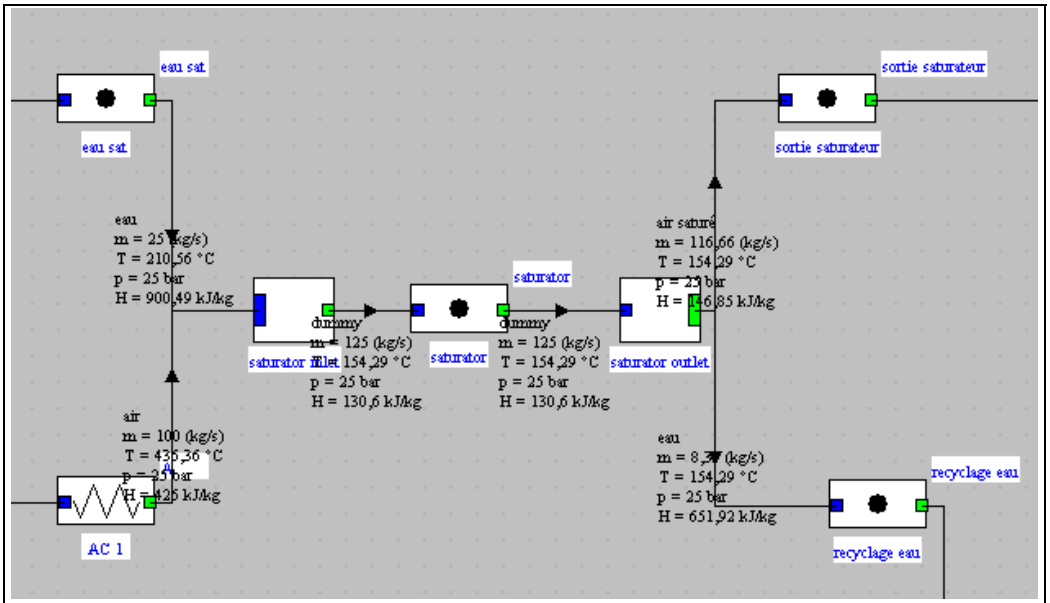

Figure 1: Saturator

The class code is as follows:

1) we begin by getting the incoming gas composition, and we update the composition of the outlet gas, a precaution in case the two dry gases would be different.

```
//Propriétés humides du gaz entrant
args[0]="process";//type of the element (see method getProperties(String[] args))
args[1]=wq.gasProcess;//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
String amont=(String)vProp.elementAt(2);//gets the downstream point name
getPointProperties(amont);//direct parsing of point property vector
getSubstProperties(nomCorps);
double M=Msubst;
Vector vComp=lecorps.getGasComposition();
outletGasSubstance.updateGasComposition(vComp);//on met à jour la composition du gaz en sortie
//cette ligne de code permet d'utiliser le saturateur avec n'importe quel gaz
```

2) we calculate the inlet absolute humidity with the definition formula, to avoid, given the high temperature of gas, having to estimate the saturation conditions.

```
//on calcule w inlet par la formule de définition, pour éviter, compte tenu de la
//température élevée du gaz, d'avoir à estimer les conditions de saturation
double fractH2O=Util.molarComp(vComp,"H2O");//fraction molaire de H2O
double inlet_w=fractH2O*18.01528/M_secpoint/(1-fractH2O);//(M_H2O)(x_H2O)/(M_gs)/(x_gs)
getPointProperties(amont);
double inletT=Tpoint;
```

3) the dry gas flow and the inlet gas specific enthalpy are determined, using methods updatepoint() and getPointProperties():

```
//estimation du débit d'air sec
Double f=(Double)vProp.elementAt(3);
double flow=f.doubleValue();//débit massique de gaz humide
flow_as=flow/(1+inlet_w);//débit massique de gaz sec

updatepoint(wq.gasPoint, false, 0,  //T
    false, 0, false, 0,//P,x
    true, "setW and calculate q'", inlet_w);
getPointProperties(wq.gasPoint);
qPrimeAmont=QPrimepoint;//enthalpie spécifique de l'air entrant dans le saturateur
```

4) At this stage, the upstream moist gas properties are perfectly calculated. We must now determine the saturator exit temperature, solving simultaneously:

- water balance (the flow of water consumed is equal to the product of the dry gas flow by the gas moisture variation);
- enthalpy balance (the sum of incoming enthalpy flows (specific units for the moist gas) is equal to the sum of outgoing enthalpy flows. Since Ts is unknown, we do a solution search by dichotomy, using generic function Util.dicho_T(), which uses f_dicho(). The code operates as follows:
- humidity is passed as input argument, instead of the pressure, otherwise known;
- we first calculate the enthalpy heau (Ts) of water at the outlet;
- we change the gas outlet temperature, then its moisture from the value read on the screen, and we get the values of its absolute humidity and specific enthalpy;
- we calculate the flow of water left (although we should do a test to make sure it remains positive);
- we write that the enthalpy lost by the water ends up in the air, and calculate residue diff;
- temperature Ts is determined when diff = 0.

```
//       double T=Util.dicho_T(this, 0, inlet_w, "saturator", 373.15, 450., 0.01);
      if (fonc.equals("saturator")){
          double diff;
          double w_dicho=P;
          //enthalpie de l'eau en sortie
          updatepoint(waterPoint, true, T,  //T
              false, 0, false, 0,//P,x
              false, "", 0);
          getPointProperties(waterPoint);//état de l'eau en sortie
          double hEau=Hpoint;

          //enthalpie spécifique et humidité spécifique de l'air en sortie
          updatepoint(gasPoint, true, T,  //T
              false, 0, false, 0,//P,x
              false, "", 0);
          updatepoint(gasPoint, false, 0,  //T
              false, 0, false, 0,//P,x
              true, "setEpsi and calculate", Util.lit_d(outletEpsi_value.getText()));
          getPointProperties(gasPoint);//propriétés humides
          waterFlow=wq.waterFlow -(Wpoint-w_dicho)*flow_as;//débit d'eau restant
          //on écrit que l'enthalpie perdue par l'eau se retrouve dans l'air
          diff=wq.waterFlow*wq.waterH-hEau*waterFlow+flow_as*(qPrimeAmont-QPrimepoint);
          return diff;
      }
```

5) Ts being determined, we change the composition of moist gas at the outlet

```
//modification de la composition du gaz
outletT=T;
updatepoint(gasPoint, false, 0,  //T
        false, 0, false, 0,//P,x
        true, "modHum", 0);
```

6) Consistency checks

Using the values that appear on the diagram of Figure 1.3.1, we can build the apparent saturator balance:

	kW
total inlet enthalpy flow	65,012
total output enthalpy flow	22,570
apparent discrepancy	42,443
difference per kg/s of water consumed	Leau = 2,547

Everything happens as if 42.4 MW of heat disappeared but, as shown in the last row of the table, this value corresponds exactly to the enthalpy of vaporization of water in the moist gas, which is not recognized in the values displayed by Thermoptim given the conventions adopted for ideal gases (zero enthalpy at 25 °C and 1 bar).

Obviously, if we sufficiently cooled exhaust gas for the water they contain to condense, this enthalpy would appear again (with the addition of the water formed in the combustion chamber).

APPENDIX: PROJECT METHODS ACCESSIBLE FROM THE EXTERNAL CLASSES

The Project method updatePoint() allows one to programmatically change the state of a point and recalculate it, including its moist properties.

```
public void updatePoint(Vector properties){
    String nomPoint=(String)properties.elementAt(0);
    PointCorps point=getPoint(nomPoint);
    if(point!=null){
      String test=(String)properties.elementAt(1);
      boolean updateT=Util.lit_b(test);
      String value=(String)properties.elementAt(2);
      double T=Util.lit_d(value);
      test=(String)properties.elementAt(3);
      boolean updateP=Util.lit_b(test);
      value=(String)properties.elementAt(4);
      double P=Util.lit_d(value);
      test=(String)properties.elementAt(5);
      boolean updateX=Util.lit_b(test);
      value=(String)properties.elementAt(6);
      double x=Util.lit_d(value);

      //pour mélanges humides for moist mixtures
      if(properties.size()>7){
        test=(String)properties.elementAt(7);
        boolean melHum=Util.lit_b(test);

        if(!melHum){//calculs à effectuer dans le cas général
          if(updateT)point.setT(T);
          if(updateP)point.setP(P);
          if(updateX)point.setX(x);
          point.CalculeUnPoint();
        }
        else{//calculs humides

          String task=(String)properties.elementAt(8);
          value=(String)properties.elementAt(9);

          if(task.equals("setW and calculate all")){//sets w and calculates moist properties
            double w=Util.lit_d(value);
            point.setW(w);
            point.calcHum();
          }
          if(task.equals("setW and calculate q'")){//sets w and calculates moist properties except t'
            double w=Util.lit_d(value);
            point.setW(w);
            point.calcQprime();
          }
         if(task.equals("setEpsi")){//sets epsilon
            double epsi=Util.lit_d(value);
            point.setEpsi(epsi);
            point.impHumRel();
          }
          if(task.equals("setEpsi and calculate")){//sets epsilon and calculates moist properties
            double epsi=Util.lit_d(value);
            point.setEpsi(epsi);
            point.impHumRel();
            point.calcQprime();
          }
          if(task.equals("calcWsat")){//calculates saturation properties and moist properties except t'
```

```
            T=Util.lit_d(value);
            double wsat=point.wsat(T);
            point.setW(wsat);
            point.calcQprime();
          }
          if(task.equals("modHum")){//modifies the gas composition
            point.modGasHum(false);
          }
        }
      }
      else{//calculs à effectuer dans le cas général
        if(updateT)point.setT(T);
        if(updateP)point.setP(P);
        if(updateX)point.setX(x);
        point.CalculeUnPoint();
      }
    }
```

The method getProperties() is then used to retrieve specific values, knowing that the method getPointProperties("nomPoint") of ExtProcess can load these values directly in the following two: Wpoint for absolute humidity w, Epsipoint for relative humidity ε, Qprimepoint for the specific enthalpy q', Tprimepoint for the adiabatic temperature t' (° C), Trpoint to the dew point tr (° C), VPrimepoint for the specific volume vs, Condpoint for condensate, M_secpoint and for the molar mass of dry gas

```
public Vector getProperties(String[] args){ (partiel)
else if(type.equals("point")){
      PointCorps pt=getPoint(nomType);
      if(pt!=null){
        vProp.addElement(pt.lecorps);//Substance
        vProp.addElement(pt.lecorps.getNom());//Substance name
        vProp.addElement(new Double(pt.getT()));//Temperature
        vProp.addElement(new Double(pt.getP()));//Pressure
        vProp.addElement(new Double(pt.getXx()));//Quality
        vProp.addElement(new Double(pt.getV()));//Volume
        vProp.addElement(new Double(pt.getU()));//Internal energy
        vProp.addElement(new Double(pt.getH()));//Enthalpy
        vProp.addElement(new Double(pt.getS()));//Entropy
        String setTsat="set_Tsat="+Util.aff_b(pt.JCheckSetTsat.isSelected());
        vProp.addElement(setTsat);//setTsat
        vProp.addElement(new Double(pt.dTsat_value.getValue()));//DTsat
        String setpsat="set_psat="+Util.aff_b(pt.JCheckSetPsat.isSelected());
        vProp.addElement(setpsat);//setpsat

        //wet gas values
        vProp.addElement(new Double(pt.w_value.getValue()));//specific humidity
        vProp.addElement(new Double(pt.epsi_value.getValue()));//relative humidity
        vProp.addElement(new Double(pt.qprime_value.getValue()));//specific enthalpy
        vProp.addElement(new Double(pt.tprime_value.getValue()));//adiabatic temperature
        vProp.addElement(new Double(pt.tr_value.getValue()));//dew point temperature
        vProp.addElement(new Double(pt.v_spec_value.getValue()));//specific volume
        vProp.addElement(new Double(pt.cond_value.getValue()));//condensates
        vProp.addElement(new Double(pt.lecorps.M_sec));//Dry gas molar mass
      }
```