Pilote de turboréacteur

Cette note présente une documentation succincte de la classe externe Thrust qui est utilisée comme pilote d'un modèle de turboréacteur à simple flux.

Un pilote permet de coordonner les recalculs d'un projet de Thermoptim selon des règles particulières. Pour notre modèle, qui est présenté dans la fiche-guide de TD sur les turboréacteurs, le pilote permet d'assurer la coordination des mises à jour du diffuseur, de la tuyère et du rapport de compression dans l'ensemble du projet, de calculer les valeurs de la poussée spécifique et de la consommation par unité de poussée, qui ne sont pas directement fournies par Thermoptim, et d'effectuer des études de sensibilité en sauvegardant les résultats dans un fichier. En opérant de la sorte, on facilite beaucoup l'utilisation du modèle.

Comme le tome 3 du manuel de référence l'explique de manière détaillée, il y a deux moyens de piloter Thermoptim : soit totalement à partir d'une application externe qui instancie le progiciel et le paramètre, soit partiellement, pour un projet donné. C'est dans ce dernier contexte que nous nous situons.

Dans ce cas, on associe une classe de pilotage spécifique au modèle, et cette classe est instanciée lors du chargement du projet. Pour pouvoir l'associer au projet, il faut commencer par en faire une classe externe pour qu'elle soit chargée dans Thermoptim lors de son lancement. Une fois le projet ouvert, la sélection de la classe de pilotage qui lui est associée se fait grâce à la ligne "Ecran de pilotage" du menu Special du simulateur, comme indiqué à la fin de cette note. Lors de l'écriture du fichier de projet, le nom de la classe de pilotage est sauvegardé pour qu'elle puisse être instanciée lors d'un chargement ultérieur du projet.

Rappel succinct des caractéristiques du modèle à piloter

La figure 1 présente un synoptique du modèle de turboréacteur que l'on désire piloter. Il met en jeu un diffuseur d'entrée qui permet de créer une surpression dynamique en entrée de compresseur lorsque l'avion est en vol, un générateur de gaz composé d'un compresseur, d'une chambre de combustion et d'une turbine, et enfin une tuyère qui assure la propulsion de l'avion.

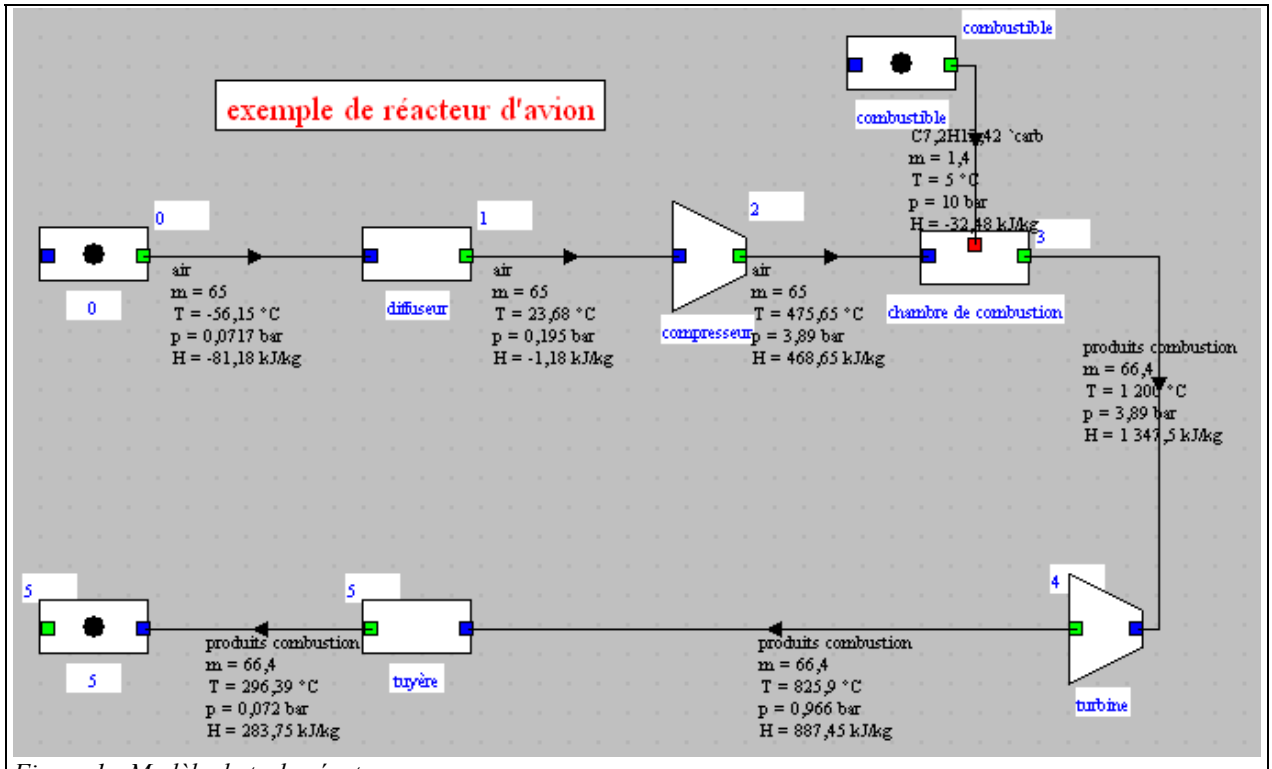

Figure 1 : Modèle de turboréacteur

Pour calculer un point de fonctionnement, il faut opérer de la manière suivante :

- entrer dans l'écran du diffuseur les valeurs des conditions extérieures, c'est-à-dire de la vitesse de l'avion (m/s), de la pression et de la température ambiante
- entrer dans l'écran de la tuyère la valeur de la pression ambiante
- paramétrer le rapport de compression
- entrer dans la chambre de combustion la température d'entrée turbine
- recalculer l'ensemble du projet avec ce paramétrage
- récupérer dans les écrans du diffuseur et de la tuyère les valeurs des vitesses de l'avion C_0 et de sortie des gaz C_5 ainsi que des débits aspiré $\dot{m}_0$ et rejeté $\dot{m}_5$, afin de calculer la poussée spécifique et la consommation par unité de poussée comme indiqué ci-dessous

L'expression de la poussée est $F = \dot{m}_0 C_0 - \dot{m}_5 C_5$, et la poussée spécifique vaut $F/\dot{m}_0$.

Le débit-masse de carburant consommé est $\dot{m}_c = \dot{m}_5 - \dot{m}_0$, et la consommation par unité de poussée vaut $\dot{m}_c /F$.

Un tel enchaînement d'opérations est fastidieux à répéter lorsque l'on désire faire des études de sensibilité, et comporte donc des risques d'erreur. La réalisation d'un pilote est parfaitement justifiée dans un tel cas.

Présentation de la classe externe

La classe de pilotage est une extension de extThopt.ExtPilot, qui dérive de rg.thopt.PilotFrame. La figure 2 montre l'écran du pilote dont le code est présenté ici, avec en haut la définition des conditions extérieures à l'avion, au milieu la saisie d'une valeur du rapport de compression pour effectuer un seul calcul, et en bas la saisie des bornes du rapport de compression pour effectuer 10 calculs et sauvegarder les résultats.

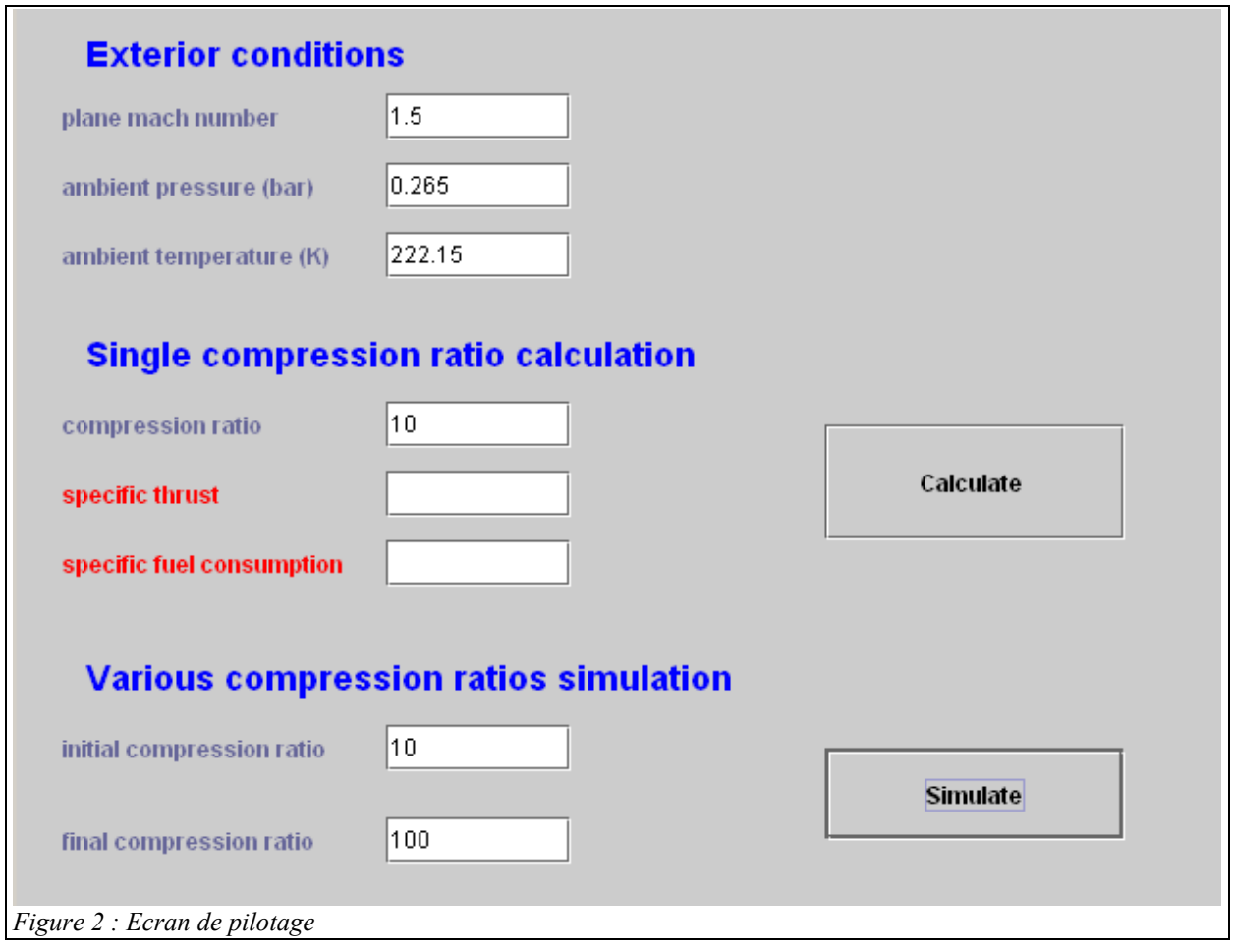

Figure 2 : Ecran de pilotage

Nous ne présenterons pas ici la construction de l'interface graphique, qui ne pose aucun problème particulier et dont le code est tout à fait classique.

L'enchaînement des initialisations et des calculs est le suivant :

1) on recherche des instances des classes externes mises en jeu. Les classes Diffuser et Nozzle ont été déclarées de manière globale, Nozzle comme un tableau de dimension 2 afin que le code puisse être facilement modifié pour pouvoir servir de pilote à un turboréacteur à double flux. Un test de cohérence est fait sur le nombre d'instances des classes. Il serait possible de l'améliorer.

```
    void setupProject(){
        //on récupère ici les instances des transfos externes dont on a besoin
        //afin d'accéder à leurs différents paramètres pour les calculs ultérieurs
        nozzle=new Nozzle[2];
        nomNozz=new String[2];
        nomNozz[0]="";
        nomNozz[1]="";
        Vector vExt=proj.getExternalClassInstances();//Vector contenant les classes externes
        int k=0,j=0;
        for(int i=0;i<vExt.size();i++){
            Object[] obj=new Object[6];
            obj=(Object[])vExt.elementAt(i);
            ExtProcess ep=(ExtProcess)obj[1];
            if(ep instanceof Diffuser){
                diff=(Diffuser)ep;
                nomDiff=diff.getName();
                j++;
            }
            if(ep instanceof Nozzle){
                nozzle[k]=(Nozzle)ep;
                nomNozz[k]=nozzle[k].getName();
                k++;
            }
        }
        if(j*k==1)isBuilt=true;//test de cohérence du pilote par rapport au modèle
    }
```

2) les autres initialisations font appel aux méthodes getProject() et proj.getEditorComponentList() présentées dans le tome 3 du manuel de référence, qui fournissent la référence du projet (proj) et la liste des composants de l'éditeur de schémas (la méthode setupProject() est présentée au point 1 ci-dessus).

```
    public void init(){
        isInitialized=true;
        proj=getProjet();
        setupProject();
        //On récupère la liste et le type des composants présents dans l'éditeur de schémas
        String[]listComp=proj.getEditorComponentList();
        composant=new String[listComp.length];
        nomComposant=new String[listComp.length];

        //on en extrait les noms des transfos du noyau dont on a besoin
        //à savoir le compresseur et la chambre de combustion
        for(int i=0;i<listComp.length;i++){
            composant[i]=Util.extr_value(listComp[i], "type");
            nomComposant[i]=Util.extr_value(listComp[i], "name");
            if(composant[i].equals("Combustion"))combustionChamberName=nomComposant[i];
            if(composant[i].equals("Compression"))compressorName=nomComposant[i];
        }

        //test de cohérence (des messages d'erreur plus précis seraient souhaitables)
        if((!combustionChamberName.equals(""))&&(!compressorName.equals(""))&& isBuilt)isBuilt=true;
        if(isBuilt)show();//on n'ouvre le pilote que si sa structure est correcte
        //initialisations pour la simulation (calculs pour 10 rapports de compression)
        thrust=new double[10];
        conso=new double[10];
        taux=new double[10];
    }
```

3) initialisation et calcul pour un seul rapport de compression (le code permettant d'effectuer la simulation pour 10 rapports de compression est tout à fait analogue, avec en plus la sauvegarde des résultats dans un fichier). Attention : le reparamétrage de la compression se fait en modifiant la pression du point aval, ce qui impose que la compression soit effectuée avec un rapport "calculé" et non "imposé" comme dans le projet sans pilote.

```
void bCalculate_actionPerformed(java.awt.event.ActionEvent event){
    Pext=Util.lit_d(ambiantP_value.getText());//lecture de la pression extérieure (bar)
    Text=Util.lit_d(ambientT_value.getText());//lecture de la température extérieure (K)
    diff.setAmbientConditions(Text, Pext);//mise à jour du point amont du diffuseur
    nozzle[0].setExternalPressure(Pext);//mise à jour de la pression en aval de la tuyère
    Ma=Util.lit_d(planeMach_value.getText());//lecture du nombre de Mach de l'avion
    double C=diff.getSpeed(Ma, Text, Pext);//mise à jour de la vitesse en amont du diffuseur
    diff.setInletVelocity(C);
    double comprRatio=Util.lit_d(comprRatio_value.getText());//lecture du rapport de compression
    for(int i=0;i<3;i++){//on effectue 3 recalculs pour garantir la convergence
        if(!combustionChamberName.equals("")){//mise à jour du compresseur
            String[] args=new String[2];
            args[0]="process";
            args[1]=compressorName;
            Vector vProp=proj.getProperties(args);
            String amont=(String)vProp.elementAt(1);
            String aval=(String)vProp.elementAt(2);
            args[0]="point";
            args[1]=amont;
            vProp=proj.getProperties(args);
            Double f=(Double)vProp.elementAt(3);
            double Pamont=f.doubleValue();

            double Paval=Pamont*comprRatio;//mise à jour du point aval du compresseur
            Vector vPoint=new Vector();
            vPoint.addElement(aval);
            vPoint.addElement("false");//update temperature
            vPoint.addElement("0");
            vPoint.addElement("true");//update pressure
            vPoint.addElement(Util.aff_d(Paval));
            vPoint.addElement("false");
            vPoint.addElement("1");
            proj.updatePoint(vPoint);
        }
        proj.calcThopt();
    }
    //calcul des performances globales du moteur
    double Cin=diff.getInletVelocity();
    double Cout0=nozzle[0].getOutletVelocity();
    double airFlow=diff.getFlow();
    double combustionFlow=airFlow;
    if(!combustionChamberName.equals("")){
        String[] args=new String[2];
        args[0]="process";
        args[1]=combustionChamberName;
        Vector vProp=proj.getProperties(args);
        Double f=(Double)vProp.elementAt(3);
        combustionFlow=f.doubleValue();
    }
    double fAR=combustionFlow/airFlow-1;//fuel air ratio
    double specThrust=((1+fAR)*Cout0-Cin)/1000;
    thrust_value.setText(Util.aff_d(specThrust, 3));
    double specFuelCons=fAR/specThrust;
    fuel_value.setText(Util.aff_d(specFuelCons, 5));
}
```

Chargement du pilote

Le chargement du pilote se fait en activant la ligne "Ecran de pilotage " du menu "Spécial" du simulateur. Un menu déroulant propose alors la liste des pilotes disponibles (figure 3). Sélectionnez celui que vous désirez charger (ici "thrust"), puis validez.

Lorsqu'un pilote est déjà chargé, la ligne "Ecran de pilotage " du menu "Spécial" du simulateur vous propose l'écran de la figure 4, qui vous permet de revenir au pilote chargé (ligne avec son nom, d'en sélectionner un autre, ou de supprimer l'existant sans le remplacer). Un seul pilote peut être associé à un projet.

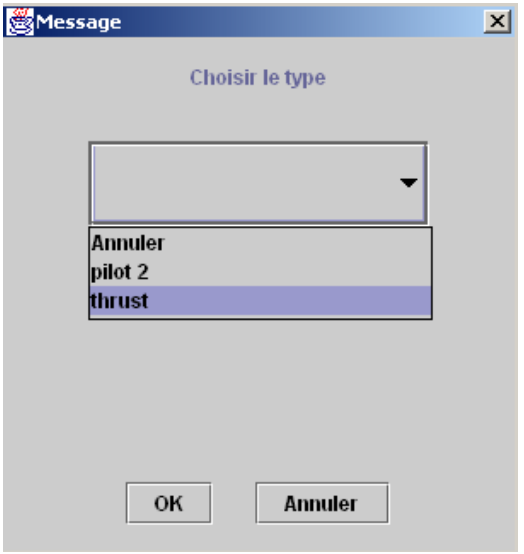

Figure 3 : Choix du pilote

Figure 4 : Suppression du pilote