```
# -*- coding: iso-8859-15 -*-

import itertools as It
import numpy as np
import scipy.sparse as Sp
from GVar_Stirling import *
from openopt import *

def TT_2_TetaDT(TM,Tm) :
    """ fonction permettant de passer du formalisme TM,Tm à Teta,DT """
    teta=TM/Tm
    DT=TM-Tm
    return [teta,DT]


def TetaDT_2_TT(teta,DT) :
    """ fonction permettant de passer du formalisme Teta,DT à TM,Tm"""
    TM=teta*DT/(teta-1.)
    Tm=DT/(teta-1.)
    return [TM,Tm]

def ParseResults(P,R,Bool) :
    return [Sp.csr_matrix(np.where(Bool,P,0.)),Sp.csr_matrix(np.where(Bool,R,0.)
)]


def Pow_Eff(teta,DT,epsi,gama,alpha,Kh,Kc,Kl,Kreg,R,Tc,Th) :
    """ Calcul de la puissance et du rendement d'un moteur Stirling
  C'est une relation vectorielle et les variables :
    * teta et DT sont des vecteurs.
    * les autres parmètres sont des scalaires
  Exemple d'utillisation :
  [mat_P,mat_R]=Pow_Eff(vec_teta,vec_DT,epsi,gama,alpha,Kh,Kc,Kl,Kreg,R,Tc,Th)
    * mat_P et mat_R sont des matrices creuses """
    teta=np.atleast_1d(np.asarray(teta,dtype=float))
    DT=np.atleast_1d(np.asarray(DT,dtype=float))
    M_T=np.tile(teta,(DT.size,1))
    M_DT=np.transpose(np.tile(DT,(teta.size,1)))
    C1=(gama-1.)*(M_T-R)/(M_T-1.)*np.log(epsi)
    C2=np.log((M_T-alpha*(M_T-1.))*((1.+alpha*(M_T-1.))/M_T)**R)/(M_T-1.)
    C3=(np.log(epsi)*(gama-1.)+(1.-alpha)*(M_T-1))/(Kc*(M_DT-Tc*(M_T-1.)))
    C4=(M_T*np.log(epsi)*(gama-1.)+(1.-alpha)*(M_T-1))/(Kh*(Th*(M_T-1.)-M_T*M_DT
))
    C5=2./Kreg/M_DT
    A1=C1+C2
    A2=C3+C4+C5
    Pow=np.squeeze(A1/A2/1000.) # Puissance en kW
    Eff=np.squeeze(A1/(M_T/(M_T-1.)*(gama-1.)*np.log(epsi)+(1.-alpha)+Kl*(Th-Tc)
*A2))
    if np.size(Eff)==1 and np.size(Pow)==1:
        return [Pow.astype('float'),Eff.astype('float')]
    else :
        try :
            return ParseResults(Pow,Eff,Bool)
        except :
            print "cas 2"
            print "recalcul du masque"
            X,Y=np.meshgrid(teta,DT)
            Bool1=(Y<Th*(X-1.)/X)
            Bool2=Y>Tc*(X-1.)
            Bool=Bool1*Bool2
            return ParseResults(Pow,Eff,Bool)


def Optim_func(Mat) :
    ncase=np.size(Mat,0)
```



```
    nList=np.size(Mat,1)
    Mat=np.c_[Mat,np.zeros((ncase,12))]
    print ncase,nList
    for nlig,case in It.izip(It.count(0),Mat[:,] ) :
        alpha,Kh,Kc,Kl,Kreg=case[:nList]
        Pref=(((Kc*Kh)/(Kc**0.5+Kh**0.5)**2)*(Th**0.5-Tc**0.5)**2)/1000.
        Rref=1.-(Tc/Th)**0.5
        # ================================================
        # On cherche à travailler sur la puissance Optimale
        # ================================================

        fopt_P=lambda x : Pref-Pow_Eff(x[0],x[1],epsi,gama,alpha,Kh,Kc,Kl,Kreg,R
,Tc,Th)[0]
        p = NLP(fopt_P,[2.5,300], iprint = -1, maxIter = 1e4,plot=False,name="Pu
iss. Optimale")
        p.c = lambda x: [Tc*(x[0]-1.)-x[1],x[0]*x[1]-Th*(x[0]-1.)]
        p.lb[0],p.ub[0] = teta_min,teta_max
        p.lb[1], p.ub[1] = DT_min,DT_max
        sol_p = p.solve('ralg')
        T_DT_Popt=sol_p.xf
        Popt,R_Popt=Pow_Eff(T_DT_Popt[0],T_DT_Popt[1],epsi,gama,alpha,Kh,Kc,Kl,K
reg,R,Tc,Th)
        print " case N = %g / %g "%(nlig,ncase)
        print " Puissance Optimale %g" %(-sol_p.ff)," kW "
        TM,Tm=TetaDT_2_TT(sol_p.xf[0],sol_p.xf[1])
        print " ------------------------ "

        # ================================================
        # On cherche à travailler sur le rendement Optimal
        # ================================================

        fopt_R=lambda x : Rref-Pow_Eff(x[0],x[1],epsi,gama,alpha,Kh,Kc,Kl,Kreg,R
,Tc,Th)[1]
        r=NLP(fopt_R,T_DT_Popt, iprint = -1, maxIter = 1e4,plot=False,name="Rend.
Optimal")
        r.c = lambda x: [Tc*(x[0]-1.)-x[1],x[0]*x[1]-Th*(x[0]-1.)]
        r.lb[0],r.ub[0] = teta_min,teta_max
        r.lb[1], r.ub[1] = DT_min,DT_max
        sol_r= r.solve('ralg')
        T_DT_Ropt=sol_r.xf
        P_Ropt,Ropt=Pow_Eff(T_DT_Ropt[0],T_DT_Ropt[1],epsi,gama,alpha,Kh,Kc,Kl,K
reg,R,Tc,Th)
        print " Rendement Optimal %g -" %(-sol_r.ff)
        TM,Tm=TetaDT_2_TT(sol_r.xf[0],sol_r.xf[1])
        print " ------------------------ "

        # ================================================
        # On cherche à travailler sur un moyen terme
        # Distance minimale par rapport à Popt et Ropt
        # ================================================

        #~ fopt_PR=lambda x : sum((Pow_Eff(x[0],x[1],epsi,gama,alpha,Kh,Kc,Kl,Kr
eg,R,Tc,Th)\
            #~ /np.array([Popt,Ropt])-np.array([1.,1.]))**2)**0.5
        fopt_PR=lambda x : sum((Pow_Eff(x[0],x[1],epsi,gama,alpha,Kh,Kc,Kl,Kreg,
R,Tc,Th)\
            /np.array([Pref,Rref])-np.array([1.,1.]))**2)**0.5

        first_guess=(sol_p.xf+sol_r.xf)*0.5
        pr=NLP(fopt_PR,first_guess, iprint = -1, maxIter = 1e4,plot=False,name="
Pow. Rend. Moyens")
        pr.c = lambda x: [Tc*(x[0]-1.)-x[1],x[0]*x[1]-Th*(x[0]-1.)]
        pr.lb[0],pr.ub[0] = teta_min,teta_max
        pr.lb[1], pr.ub[1] = DT_min,DT_max
        sol_pr= pr.solve('ralg')
        T_DT_PRmoy=sol_pr.xf
```

```
            Pmoy,Rmoy=Pow_Eff(T_DT_PRmoy[0],T_DT_PRmoy[1],epsi,gama,alpha,Kh,Kc,Kl,K
    reg,R,Tc,Th)
            print " Puissance Rendement moyens %g kW : %g -" %(Pmoy,Rmoy)
            TM,Tm=TetaDT_2_TT(sol_pr.xf[0],sol_pr.xf[1])
            print "========================="
            #~ stockage des resultats
            Mat[nlig,nList::]=np.r_[T_DT_Popt,Popt,R_Popt,T_DT_Ropt,P_Ropt,Ropt,T_DT
    _PRmoy,Pmoy,Rmoy]
        #~ renvoi des resultats
        return Mat
```