```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 16 20:57:24 2022
@author: Anne
"""

#Etude de la réaction de synthèse de l'ammoniac : influence de T et P

#Importation des biblothèques
import numpy as np
import matplotlib.pyplot as plt


plt.clf()        #Nettoyage figure précédente
fig=plt.figure(figsize=(6,6))     #Création nouvelle figure
ax=fig.add_subplot(111)
plt.grid(True)   # quadrillage
#Titre et légendes
plt.title("Synthèse NH3 : Influence de T sur le taux d'avancement de la
réaction")

#On travaillera pour une gamme de température de 300 à 600 K => gamme des
abscisses
ax.set_xlim()
#Un taux d'avancement est forcément compris entre 0 et 1 => gamme des
ordonnées
ax.set_ylim()

# Etiquettes des axes :
plt.ylabel("taux d'avancement à l'équilibre")
plt.xlabel("température (K)")

#coefficients stoechiométriques de la réaction (valeurs absolues )
c_N2,c_H2,c_NH3=

#Les conditions initiales sont stoechiométriques dans cette étude , mais on
prévoit éventuellement de pouvoir étudier des conditions NON
stoechiométriques : Voir partie détermination du réactif limitant.

no_N2=
no_H2=
no_NH3=

#Liste des pressions étudiées -> autant de courbes
LP=[0.1,1,10,20,50,100,200]

#Bornes de la gamme de températures étudiées ( -> abscisses ). On utilisera
un range entre 2 valeurs, avec un pas de 1, poour parcourir toutes les
températures de 300 à 600 K, comme prévu à l'axe de abscisses. Ne pas oublier
que le dernier terme n'est pas étudié.

T_min=

T_max=

#Caractéristiques thermodynamiques de la réaction en kJ/mol pour DrH° (H) et
J/K/mol pour DrS° ( S)

H=-92.4    #réaction exothermique
S=-199     #réaction qui crée de l'ordre
```

```
#On se propose de calculer un taux d'avancement = avancement / avancement max
.
# Or l'avancement max est donné par la disparition du réactif limitant.
Détermination du réactif limitant, de l'avancement maximal ( nécessaire si on
n'est plus dans les conditions stoechiométriques )

#On initialise la valeur de x_max (avancement maximal) à la valeur minimale
possible soit 0.
x_max=0
#On calcule x_max si N2 est limitant :
x_max_N2=
#On calcule x_max si H2 est limitant :
x_max_H2=
#On les compare pour retenir le plus petit des 2 :
if x_max_N2<=x_max_H2:
    x_max=
else:
    x_max=



#Création de 2 boucles imbriquées: à une pression donnée ( 1° boucle ), pour
une série de T variable d'abscisses ( 2° boucle ) on calcule K°(T) et on
résoud la relation à l'équilibre K°(T) = Q, qui donne l'avancement à T. On
change de pression ( en utilisant la 1° boucle ) pour recommencer un nouveau
tracé à une autre pression, dans la liste des pressions choisies.


#Demarrage de la première boucle, pour chaque valeur de P dans la liste
arbitrairement choisie, donc grâce à un "for P in ... " :
for P in    :


#A chaque pression, on va tracer une courbe taux d'avancement ( à calculer)
fonction de la température ( qui change K°(T) ) =>
# préparation liste des abscisses températures et des ordonnées taux
d'avancement, remise à 0 à chaque nouvelle Pression => nouvelle courbe. On
définit des listes VIDES de températures (abscisse ) et taux d'avancement
(ordonnée)

    Abscisses_T=

    Ordonnees_taux=
#Demarrage de la deuxième boucle : à chaque P,  pour chaque valeur de T (
"for T in range () "), il faudra calculer la constante d'équilibre valide à
cette température
    for         :  # Attention T_max exclus, calcul de K°(T)
        K=



    #Définition de la fontion f équation K(T)-Q qu'on résoudra = 0
    #Ne pas la laisser sous forme de la fraction K-n(x)/d(x) car pb de d(x)=0
(?????)
        def f(x):
            return
```

```
    #Résolution par dichotomie de l'équation, à une pression P en cours de la 1° boucle, à la température en cours  T dans la 2° boucle.
    #On cherche l'avancment dans un segment donnée, dont les les valeurs bornes sont 0 ( réaction qui n'avance pas ) et l'avancement maximal fixé par le réactif limitant ;
    #la dichotomie rétrécit l'intervalle jusqu'à une limite qu'on se choisit arbitrairement, qui donne la "précision" du résultat : 10^-5 est courant, permet des calculs rapides. Pas < 10^-16

        a,b,lim=

#On étudie d'abord le cas où il existe une solution

        if f(a)*f(b)    :

#On définit 2 bornes qui seront amenées à varier au fur et à mesure de la dichotomie, mais dont les valeurs initiales sont celles de l'intervalle d'étude choisi au départ .

            u,v=

# La boucle de dichotomie doit tourner tant que ( => while )  l'intervalle [u,v] est plus grand que la limite choisi :
            while      :
# On crée une nouvelle borne possible au Milieu de l'intervalle, valeur m
                m=
#On étudie dans quel demi intervalle se trouve la solution, par un if : si le produit borne inférieure / milieu est négatif, alors la solution est dans cet intervalle => on n'étudiera que cet intervalle moitié dans la suite : la borne supérieure devient m. Sinon, c'est que la solution est dans l'autre moitié de l'intervalle : c'est la borne inférieure qui devient m.
                if :
                    #si <=0 => borne sup = m
                else:
                    #si > 0 => borne inf = m

#La boucle s'arrêtera quand l'intervalle d'étude atteint la limite choisie : m est alors la solution de l'équation, soit dans notre cas, l'avancement : on peut calculer le taux d'avancement à l'aide de cette valeur de m, à la fin de ce "while" :
            taux=



        #Incrémentation des points, abscisses, ordonnées. On vient de finir UN calcul, à une température T, qu'il faut mettre comme n ième terme des abscisses, associé au n ième terme des ordonnées ( même numéro d'ordre )

            Abscisses_T.append( )

            Ordonnees_taux.append(    )

#Ne pas oublier le cas( impossible en réalité, sauf errreur grave de votre part dans le choix de l'intervalle... ), où il n'y aurait pas de solution  . En principe , ce else, indentation à la hauteur du 1° if sera inutile !
        else :
            print ("pas de solution")
```

```
#La 2° boucle, à la même pression, va repartir pour la tempértaure suivante.
#quand l'ensemble des temératures aura été étudié, on aura notre collection
d'abscisses et d'ordonnées associées, à la première pression P de la première
boucle : on peut faire tracer la courbe ( indentation au niveau du "for des
T" )

    #Représentation graphique : prevoir une étiquette par "label" pour chaque
courbe tracée à une pression donnée : rappel plt.plot(liste d'abscisses,
liste-même longueur- d'ordonnées, label=...). Ne pas oublier de dire
d'afficher les légendes ( dont titre, étiquettes, etc..., par plt.legend() )


    plt.
    plt.

#Le programme repart au niveau du 1° "for des P", boucle des pressions, et va
donc créer un graphe par pression. Quand cette première boucle sera achevée,
on demande de montrer TOUTES les courbes ( 1 par valeur de P )  gardées en
mémoire par plt.show() .
#Indentation primaire, fin du programme

plt.
```