1) α(réactif i) = $\alpha_i = \frac{n(Réactif_i)_{ayant\ réagi}}{n(Réactif_i)_0} = \frac{n(Réactif_i)_0 - n(réactif_i)_{eq}}{n(Réactif_i)_0}$ et $\tau = \frac{\xi}{\xi_{max}}$

α(réactif i) = τ lorsque le réactif i est le réactif limitant.

2) tableau d'avancement :

	4 HCl(g)	+	*O_2(g)*	=	*2 Cl_2(g)*	+	*2 H_2O(g)*	*n_{tot}^{gaz}*
EI	1		1		0		0	2
EF	1 - 4ξ		1 - ξ		2ξ		2ξ	2 - ξ

3) Préparation théorique :

a) Déterminer l'évolution de ΔrG° en fonction de T

- ΔrG° = ΔrH° - T× ΔrS°
-
```
# 3- fonction DrG0
def DrG0(T):
    return DrH0 - T*DrS0
```

b) Rappeler la définition de K en fonction de ΔrG° et T

- K(T) = exp(- ΔrG°(T)/(RT))
-
```
#3- fonction K
def K(T):
    return np.exp(-(DrG0(T))/(R*T))
```

c) Exprimer le quotient de réaction en fonction du taux d'avancement τ.

	N_2	+	3 H_2	→	2 NH_3	n_{tot}^{gaz}
EI	n_1		$3n_1$			$4n_1$
Eeq	n_1 - ξ		$3n_1$ – 3 ξ		2 ξ	$4n_1$ – 2 ξ
Eeq	$n_1(1-\tau)$		$3n_1(1-\tau)$		$2\ n_1\tau$	$n_1(4-2\tau)$

- $Q = \frac{P(NH_3)^2 P^{o2}}{P(N_2)P(H_2)^3} = \frac{n(NH_3)^2 n_{tot}^{gaz^2} P^{o2}}{n(N_2)n(H_2)^3 P^2} = \frac{4(\tau)^2(4-2\tau)^2 P^{o2}}{27(1-\tau)^4 P^2}$

d) Donner l'équation qui permet de déterminer τ_{eq} à l'aide de K.

- $K(T) = \frac{4(\tau)^2(4-2\tau)^2 P^{o2}}{27(1-\tau)^4 P^2}$
- algorithme efficace pour trouver les zéros de fonction f(x) sous forme polynomiale pour éviter les divisions par zéro

$$4(\tau)^2(4-2\tau)^2 P^{o2} - (27(1-\tau)^4 P^2)K(T) = 0$$

```
#on va résoudre  4*(tau**2)*((4-2*tau)**2)*(P0**2)-K(T)*(27*(1-tau)
def f(x,T,P):
    return 4*(x**2)*((4-2*x)**2)*(P0**2)-K(T)*(27*(1-x)**4*(P)**2)
```

4) **Programme python**

a) trouver un zéro :

- dichotomie • **op.bisect(f,a,b)** • **op.newton(f,xo)** • **op.fsolve(f,xo)**

PB : f ne doit dépendre que d'1 variable !!! Or nous elle dépend de 3 variables : x, T et P

b) structure du programme (cf. ci-dessus pour les fonctions)

- 3- liste des abscisses

☞ commande np.linspace(a,b,N)

```
#3- liste des abscisses
T_abs=np.linspace(Tmin,Tmax,901)
```

- 3- recherche des tau_eq pour chaque T, en utilisant op.newton

☞ faire une boucle !

☞ introduire une nouvelle fonction g qui ne dépend que de x, dont on cherche les zéros

```
#3-recherche des tau_eq pour chaque T
tau_eq=[]
for i in range(len(T_abs)):
    def g(x):
        return f(x,T_abs[i],Ptot)
    tau_eq.append(op.newton(g,0.0001))
```

- 4-tracé de la courbe tau=f(t)

```
#4-tracé de la courbe tau=f(t)
plt.plot(T_abs,tau_eq,'-b')
plt.title('Evolution du taux d avancement à l équilibre en fonction de T')
plt.xlabel('T (K)')
plt.ylabel('tau à l équilibre')
plt.grid()
plt.show()
```

POUR ALLER PLUS LOIN :

- Avec la dichotomie

☞ créer une fonction dichotomie que l'on utilisera à la place de Newton

```
tau_min=0.0001 #mol
tau_max=0.9999 #mol

# 3- création de la fonction dichotomie
def dicho(a,b,f,precision):
    while abs(b-a) > precision :
        m=(a+b)/2
        if f(a)*f(m) < 0:
            b=m
        else:
            a=m
    return (a+b)/2
```

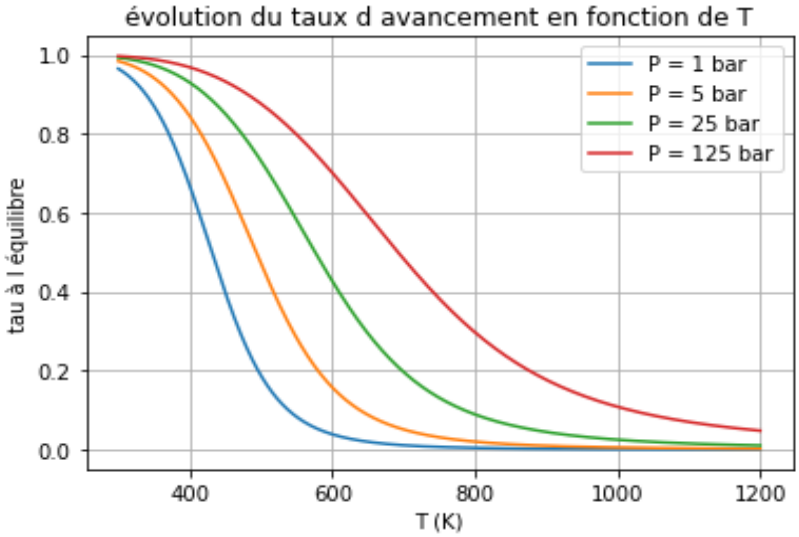
```
#3-recherche des tau_eq pour chaque T
tau_eq=[]
for i in range(len(T_abs)):
    def g(x):
        return f(x,T_abs[i],Ptot)
    tau_eq.append(dicho(tau_min,tau_max,g,0.001))
```

- Influence de P et T

☞ double boucle sur P et T

```
P=[1, 5, 25, 125]
```

```
#3-recherche des tau_eq pour chaque T et chaque P
for j in range(len(P)):
    tau_eq=[]
    for i in range(len(T_abs)):
        def g(x):
            return f(x,T_abs[i],P[j])
        tau_eq.append(op.newton(g,0.0001))
    titre = 'P = '+str(P[j])+' bar'
    #4-tracé de la courbe tau=f(t)
    plt.plot(T_abs,tau_eq,label=titre)
    plt.title('évolution du taux d avancement en fonction de T')
    plt.xlabel('T (K)')
    plt.ylabel('tau à l équilibre')
    plt.legend()
plt.grid()
plt.show()
```


évolution du taux d avancement en fonction de T
tau à l équilibre
T (K)
P = 1 bar
P = 5 bar
P = 25 bar
P = 125 bar
0.0
0.2
0.4
0.6
0.8
1.0
400
600
800
1000
1200