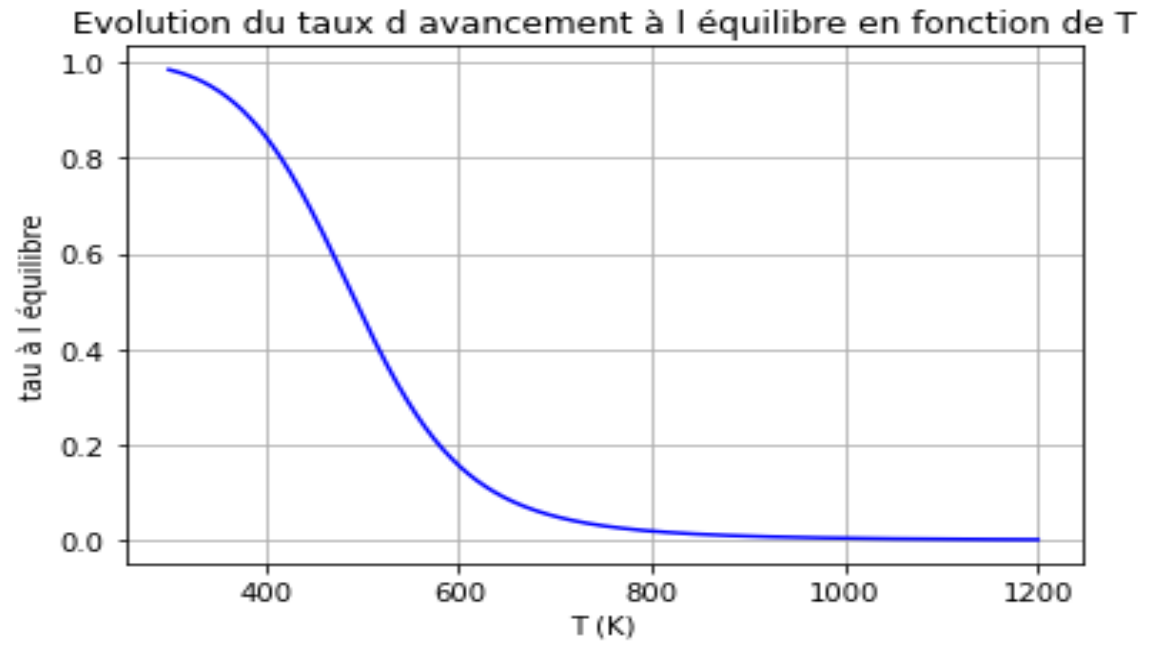
CN2

- **Capacité numérique :** tracer, à l'aide d'un langage de programmation, le taux d'avancement à l'équilibre en fonction de la température pour un système siège d'une transformation chimique modélisée par une seule réaction.



Préparation théorique

- a) Déterminer l'évolution de ΔrG° en fonction de T
  - ΔrG° = ΔrH° - T× ΔrS°

```
21    # 3- fonction DrG0
22    def DrG0(T):
23        return DrH0 - T*DrS0
```

- b) Rappeler la définition de K en fonction de ΔrG° et T
  - K(T) = exp(- ΔrG°(T)/(RT))

```
25    #3- fonction K
26    def K(T):
27        return np.exp(-(DrG0(T))/(R*T))
```

- c) Exprimer le quotient de réaction en fonction du taux d'avancement
  - Tableau d'avancement

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

  - expression de Q :

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

- 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

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

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

Programme python

- L'idée est de trouver les zéros d'une fonction issue de l'équation obtenue question d). Pour ce faire 4 possibilités au programme :
  - **la dichotomie** (il faut au préalable déterminer, en s'appuyant sur une représentation graphique, un intervalle adapté à la recherche numérique d'une racine par la méthode dichotomique.)
- Utiliser une commande toute prête de python grâce au sous-module scipy.optimize (alias op) de la bibliothèque NumPy. On a 3 possibilités :
  - **op.bisect(f,a,b)** qui permet de résoudre l'équation f(x) = 0 sur le segment [a ; b] par la méthode de dichotomie
  - **op.newton(f,xo)** qui permet de résoudre l'équation f(x) = 0 par la méthode de Newton en précisant la valeur initiale de l'algorithme
  - **op.fsolve(f,xo)** qui permet de résoudre l'équation f(x) = 0, xo est la valeur initiale de l'algorithme
- PB : f ne doit dépendre que d'1 variable !!! Or nous elle dépend de 3 variables : x, T et P

- 3- liste des abscisses
  - ☞ commande np.linspace(a,b,N)

```
35    #3- liste des abscisses
36    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

```
38    #3-recherche des tau_eq pour chaque T
39    tau_eq=[]
40    for i in range(len(T_abs)):
41        def g(x):
42            return f(x,T_abs[i],Ptot)
43        tau_eq.append(op.newton(g,0.0001))
```

- 4-tracé de la courbe tau=f(t)

```
45  #4-tracé de la courbe tau=f(t)
46  plt.plot(T_abs,tau_eq,'-b')
47  plt.title('Evolution du taux d avancement à l équilibre en fonction de T')
48  plt.xlabel('T (K)')
49  plt.ylabel('tau à l équilibre')
50  plt.grid()
51  plt.show()
```

Pour aller plus loin

Avec la dichotomie

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

```
20  tau_min=0.0001 #mol
21  tau_max=0.9999 #mol
22
23  # 3- création de la fonction dichotomie
24  def dicho(a,b,f,precision):
25      while abs(b-a) > precision :
26          m=(a+b)/2
27          if f(a)*f(m) < 0:
28              b=m
29          else:
30              a=m
31      return (a+b)/2
```

-

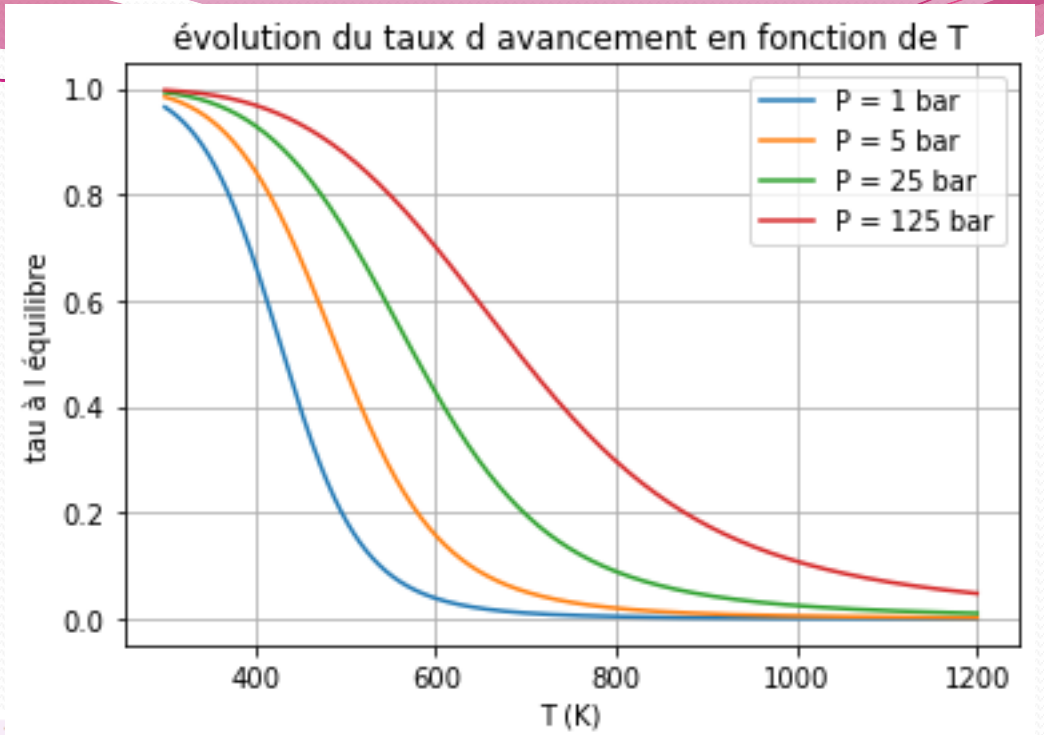
```
#3-recherche des tau_eq pour chaque T
tau_eq=[]
for i in range(len(T_abs)):
    def g(x):
        return f(x,T_abs[i],Ptot)
    tau_eq.append(dicho(tau_min,tau_max,g,0.001))
```

```
#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))
```

Influence de P et T

- ☞ double boucle sur P et T

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




```
39  #3-recherche des tau_eq pour chaque
40  for j in range(len(P)):
41      tau_eq=[]
42      for i in range(len(T_abs)):
43          def g(x):
44              return f(x,T_abs[i],P[j])
45          tau_eq.append(op.newton(g,0.0001))
46      titre = 'P = '+str(P[j])+' bar'
47      #4-tracé de la courbe tau=f(t)
48      plt.plot(T_abs,tau_eq,label=titre)
49      plt.title('évolution du taux d avancement en fonction de T')
50      plt.xlabel('T (K)')
51      plt.ylabel('tau à l équilibre')
52      plt.legend()
53  plt.grid()
54  plt.show()
```


That's all Folks!