Outils informatiques pour la chimie

Utilisation de Python en chimie :

1) Comment tracer une courbe :

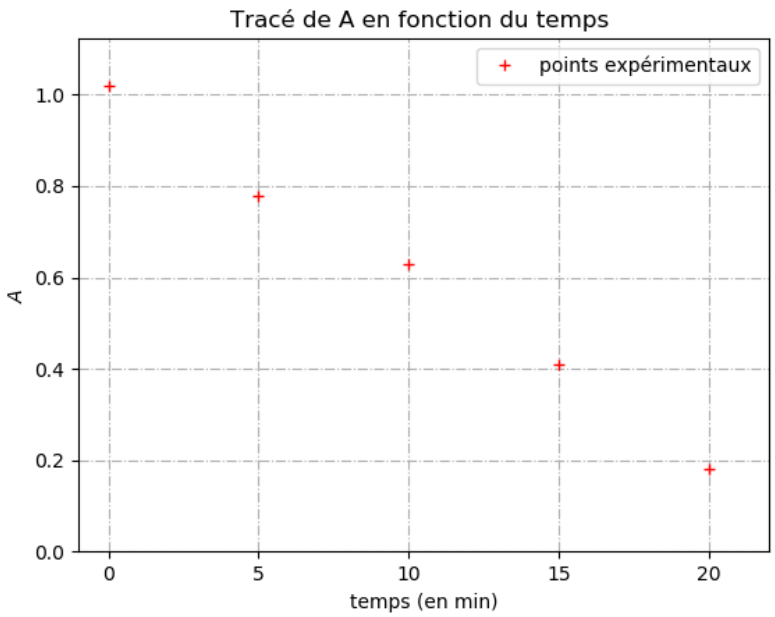
Pour un tutoriel en vidéo : https://www.youtube.com/watch?v=d7iKOkLVvEw

Application :

Absorbance A	1.02	0.78	0.63	0.41	0.18
temps (en min)	0	5	10	15	20

Tracer A = f(t).

On doit obtenir :



Solution :

```
import matplotlib.pyplot as plt
import numpy as np

A = np.array([1.02, 0.78, 0.63, 0.41, 0.18]) # valeurs d'absorbance
t = np.array([0, 5, 10, 15, 20]) # valeurs de t

# Affichage graphique :
plt.figure(1)
plt.title('Tracé de A en fonction du temps ')
plt.xlabel('temps (en min)')
plt.ylabel('A')
plt.xlim(-1, 1.1*np.max(t)),
plt.ylim(0,1.1*np.max(A))
plt.plot(t, A,'r+', label='points expérimentaux')
plt.grid(linestyle="-.")
plt.legend()
plt.show()
```

2) Comment réaliser une régression linéaire :

Pour un tutoriel en vidéo : https://www.youtube.com/watch?v=4cNWBBOEck0

Si le nuage de points est modélisable par une fonction polynomiale, on peut utiliser la fonction **np.polyfit(x , y , deg)** de la bibliothèque numpy as np.

Modélisons maintenant la courbe obtenue par une droite (polynôme de degré 1) et affichons les coefficients (modele[0] correspond au coefficient directeur et modele[1] correspond à l'ordonnée à l'origine) :

```
Modele = np.polyfit(X, Y, 1)
print(modele[0], modele[1])
```

Créons l'équation de la droite modélisée grâce à ces coefficients :

```
modele1 = ("Y = "+ format(modele[0])+"x X +"+ format(modele[1]))   # on fabrique l'équation de la droite de régression
```

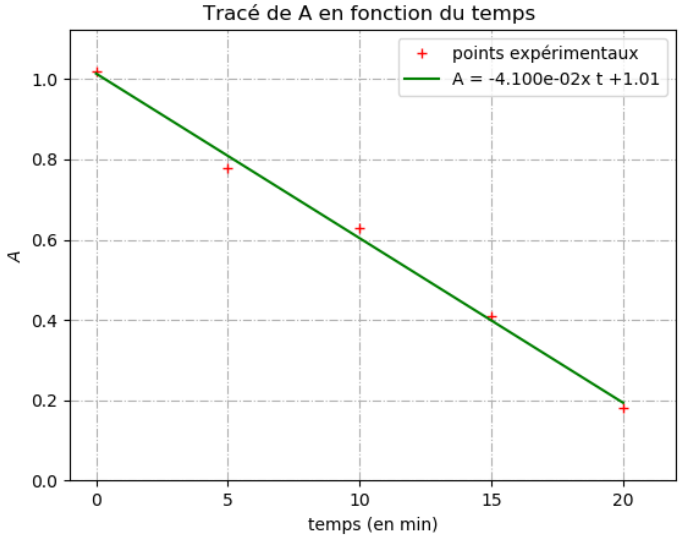
Affichons maintenant les points expérimentaux et la droite modélisée sur le même graphique :

```
plt.plot(X, Y,'r+', label='points expérimentaux')
plt.plot(X,np.polyval(modele,X),color='green',label=modele1)
```

Application :

Reprendre les données (A, t) précédentes et réaliser la régression linéaire.

On doit obtenir :



Solution :

```
import matplotlib.pyplot as plt
import numpy as np

A = np.array([1.02, 0.78, 0.63, 0.41, 0.18]) # valeurs d'absorbance
t = np.array([0, 5, 10, 15, 20]) # valeurs de t

# Régression linéaire :
p1 = np.polyfit(t, A, 1)                    #Régression linéaire de A en fonction de t
modele1=("A = "+ format(p1[0], "#.3e")+"x t +"+ format(p1[1], "#.3g"))   # on fabrique l'équation de la droite de régression

# Affichage graphique :
plt.figure(1)
plt.plot(t, A,'r+', label='points expérimentaux')
plt.plot(t,np.polyval(p1,t),color='green',label=modele1)
plt.title('Tracé de A en fonction du temps ')
plt.xlabel('temps (en min)')
plt.ylabel('A')
plt.xlim(-1, 1.1*np.max(t)),
plt.ylim(0,1.1*np.max(A))
plt.grid(linestyle="-.")
plt.legend()
plt.show()
```

3) Comment ajouter les barres d'incertitude dans une régression linéaire

Pour un tutoriel en vidéo : https://www.youtube.com/watch?v=4cNWBBOEck0 vers 10 '.

Il faut utiliser plt.errorbar, après avoir défini l'incertitude associée à chaque mesure en ordonnée.

```
plt.errorbar(X, Y, yerr = u_Y, fmt='.r')
```

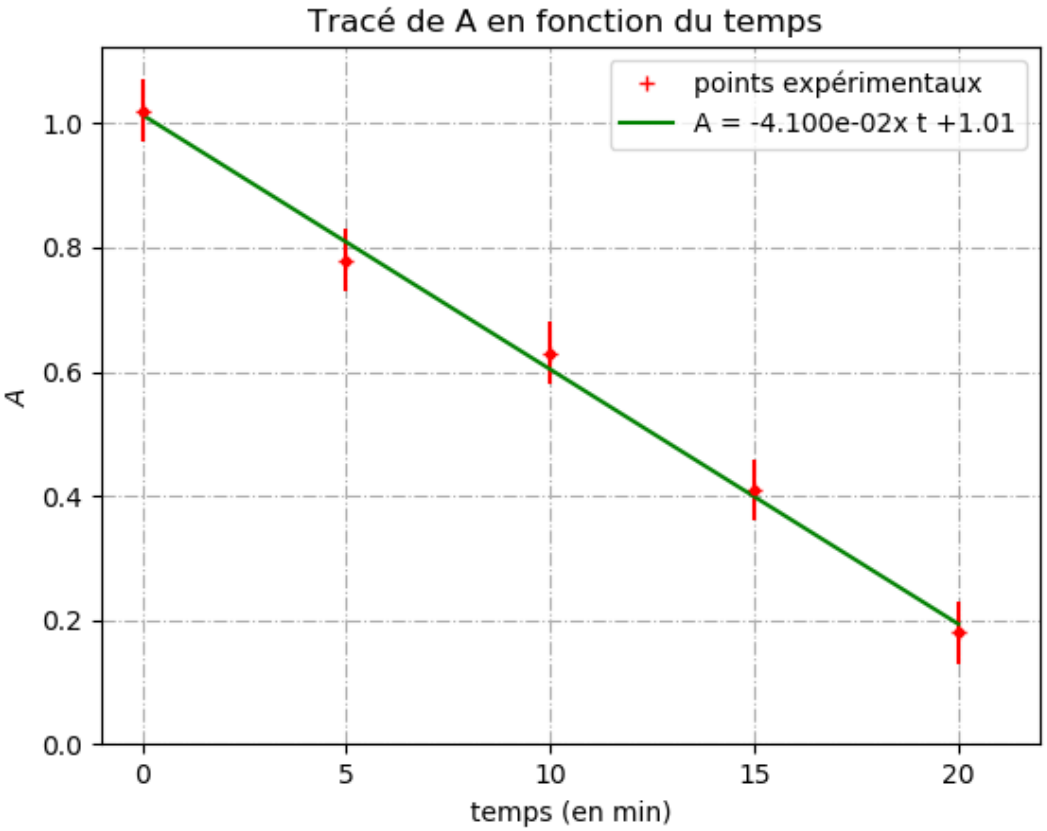
Application :

Reprendre la régression linéaire précédente et ajouter les barres d'incertitude.

On choisira ici u(A) = 0,05. Peut-on valider que les points expérimentaux sont alignés ?

Pour l'incertitude sur A, on n'est pas obligé de choisir une constante, ça peut aussi être un pourcentage, par exemple u(A) = 2/100*A. Réessayez. Quelle différence constatez-vous ?

On doit obtenir :



Solution :

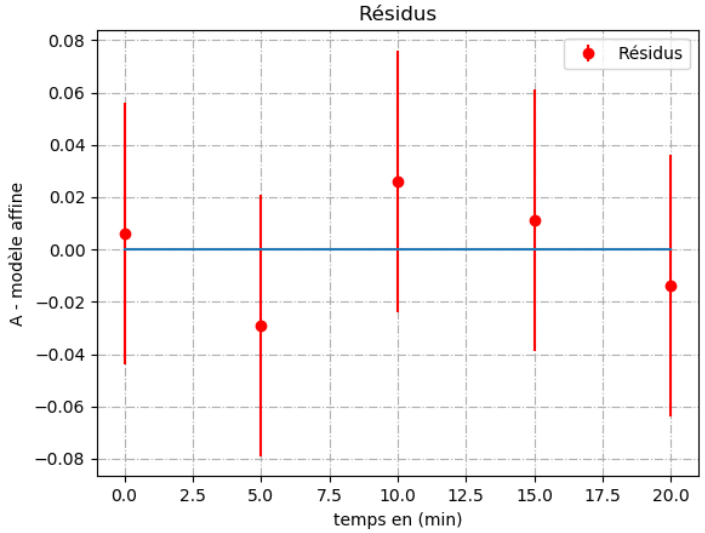
```
import matplotlib.pyplot as plt
import numpy as np

A = np.array([1.02, 0.78, 0.63, 0.41, 0.18]) # valeurs d'absorbance
t = np.array([0, 5, 10, 15, 20]) # valeurs de t

u_A = 0.05 # incertitude - type par rapport à l'absorbance

# Régression linéaire :
p1 = np.polyfit(t, A, 1)                    #Régression linéaire de A en fonction de t
modele1=("A = "+ format(p1[0], "#.3e")+"x t +"+ format(p1[1], "#.3g"))   # on fabrique l'équation de la droite de régression

# Affichage graphique :
plt.figure(1)
plt.plot(t, A,'r+', label='points expérimentaux')
plt.plot(t,np.polyval(p1,t),color='green',label=modele1)
plt.errorbar(t, A, yerr=u_A, fmt='.r')
```

```
plt.title('Tracé de A en fonction du temps ')
plt.xlabel('temps (en min)')
plt.ylabel('A')
plt.xlim(-1, 1.1*np.max(t)),
plt.ylim(0,1.1*np.max(A))
plt.grid(linestyle="-.")
plt.legend()
plt.show()
```

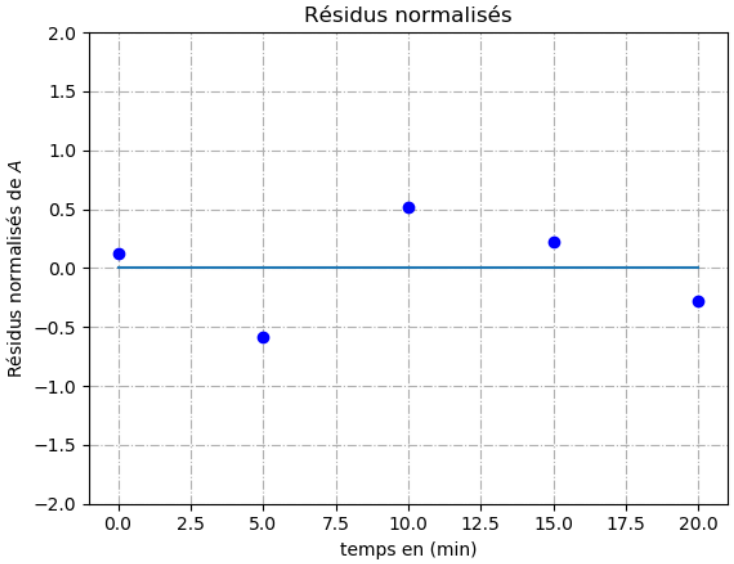
4) Comment tracer les résidus normalisés pour valider ou non un modèle linéaire

Pour s'assurer efficacement que la droite de régression passe par les barres d'incertitude, on peut tracer :

- les résidus avec les barres d'incertitude : il faut pour cela calculer l'écart entre les points expérimentaux et la droite de régression. On obtient :



- les résidus normalisés directement : il faut pour cela diviser les résidus obtenus ci-dessus par l'incertitude u(A). On obtient :



Notre critère de décision est le dépassement, en valeur absolue, de la valeur 2 pour l'écart normalisé. Que conclurait-on pour ce nuage de résidus normalisés ?

Solution :

```
import matplotlib.pyplot as plt
import numpy as np

A = np.array([1.02, 0.78, 0.63, 0.41, 0.18]) # valeurs d'absorbance
t = np.array([0, 5, 10, 15, 20]) # valeurs de t

u_A = 0.05 # incertitude - type par rapport à l'absorbance

# Régression linéaire :
p1 = np.polyfit(t, A, 1)                    #Régression linéaire de A en fonction de t
modele1=("A = "+ format(p1[0], "#.3e")+"x t +"+ format(p1[1], "#.3g"))   # on fabrique l'équation de la droite de régression

# Affichage graphique :
plt.figure(1)
plt.plot(t, A,'r+', label='points expérimentaux')
plt.plot(t,np.polyval(p1,t),color='green',label=modele1)
plt.errorbar(t, A, yerr=u_A, fmt='.r')
plt.title('Tracé de A en fonction du temps ')
plt.xlabel('temps (en min)')
plt.ylabel('A')
plt.xlim(-1, 1.1*np.max(t)),
plt.ylim(0,1.1*np.max(A))
plt.grid(linestyle="-.")
plt.legend()
plt.show()

#Tracé des résidus
plt.figure(2)
plt.title("Résidus")
plt.ylabel('A - modèle affine')
plt.xlabel('temps en (min)')
plt.errorbar(t,A-np.polyval(p1,t), yerr = u_A, fmt = 'ro',label="Résidus")
plt.plot((np.min(t), np.max(t)), (0, 0) )
plt.grid(linestyle="-.")
plt.legend()
plt.show()
# Tracé des résidus normalisés z
plt.figure(3)
plt.title('Résidus normalisés')
z = (A-np.polyval(p1,t))/(u_A)
plt.plot(t, z,'bo')
plt.xlabel('temps en (min)')
plt.ylabel('Résidus normalisés de A' )
plt.plot((np.min(t), np.max(t)), (0, 0) )
plt.ylim(-2,2)
plt.grid(linestyle="-.")
plt.show()
```

5) Comment simuler de multiples valeurs avec la méthode de Monte-Carlo

Pour un tutoriel en vidéo sur les tirages aléatoires et le tracé d'histogrammes :

https://www.youtube.com/watch?v=dK1XqloyGNU

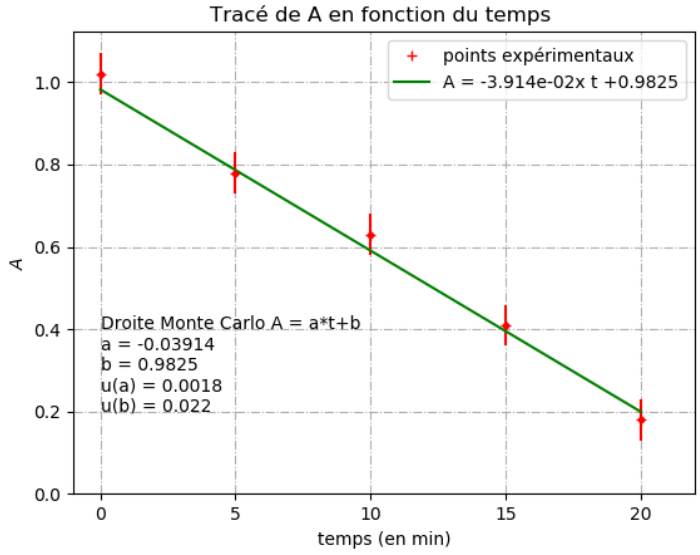
Pour un tutoriel en vidéo expliquant comment utiliser une simulation de Monte-Carlo dans le cadre d'une régression linéaire :
https://www.youtube.com/watch?v=ilN6HwNkeu0

Il faut importer la bibliothèque numpy.random
Il existe différentes sortes de distribution

- rectangulaire : chaque valeur de l'intervalle a autant de chance d'être sélectionné. On utilise la commande numpy.random.uniform(borne_inf , borne_sup). C'est le cas des burettes.
- triangulaire : la valeur centrale est plus probable que les valeurs latérales de l'intervalle. On utilise la commande numpy.random.triangular(borne_inf , centre , borne_sup). C'est le cas des fioles jaugées, des pipettes jaugées deux traits.
- normale (gaussienne) : si l'incertitude-type sur la grandeur est fournie/connue, ce qui est rare. On utilise la commande numpy.random.normal(valeur centrale, incertitude-type)

Application :

Reprendre la régression linéaire précédente avec les barres d'incertitude.
Réaliser 100 000 tirages de lots de valeurs de A +/- u(A)
et calculer le coefficients directeur moyen, l'ordonnée à l'origine moyenne, ainsi que les incertitudes associées.
On choisira une distribution rectangulaire.
On doit obtenir :



Solution :

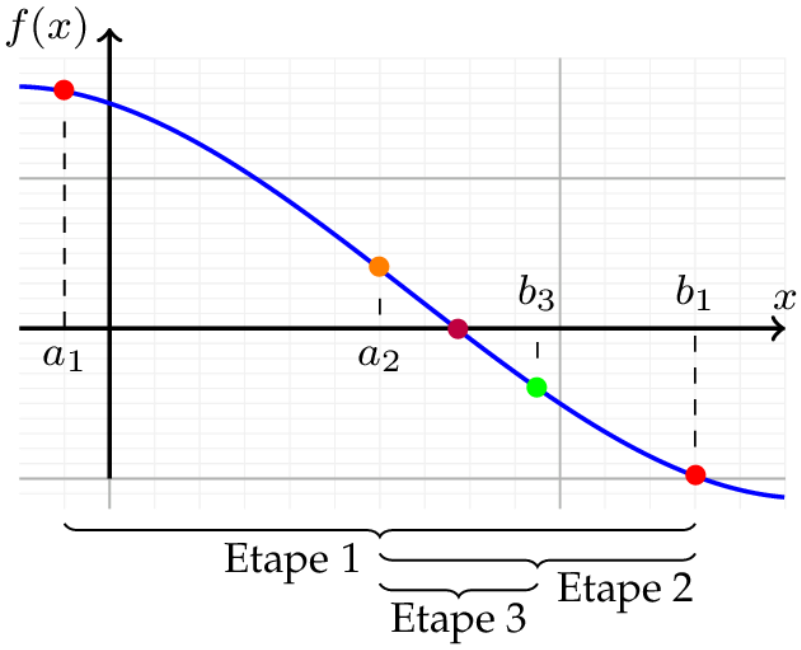
```
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rd

A = np.array([1.02, 0.78, 0.63, 0.41, 0.18]) # valeurs d'absorbance
t = np.array([0, 5, 10, 15, 20]) # valeurs de t

u_A = 0.05 # incertitude - type par rapport à l'absorbance

# Régression linéaire :
N_MC = 100000       #Nombres de simulations
a = []              #Liste vide de coefficient directeur
b = []              #Liste vide d'ordonnée à l'origine
for i in range (0, N_MC):
    A_MC = A + rd.uniform(-u_A, u_A,len(A))        #Tirage de 5 valeurs de A
    p1 = np.polyfit(t, A_MC, 1)                    #Régression linéaire de A en fonction de t
    a.append(p1[0])         # ajout du coefficient directeur trouvé pour ces 5 points dans la liste a
    b.append(p1[1])         # ajout de l'ordonnée à l'origine trouvée pour ces 5 points dans la liste b

a_moy = np.mean(a)   #Calcul du coefficient directeur moyen
u_a = np.std(a)      #Calcul de l'incertitude type sur le coefficient directeur

b_moy = np.mean(b)
u_b = np.std(b)

modele1=("A = "+ format(a_moy, "#.3e")+"x t +"+ format(b_moy, "#.4g"))   # on fabrique
l'équation de la droite de régression moyenne

# Affichage graphique :
plt.figure(1)
plt.plot(t, A,'r+', label='points expérimentaux')
plt.plot(t,np.polyval(p1,t),color='green',label=modele1)
plt.errorbar(t, A, yerr=u_A, fmt='.r')
plt.title('Tracé de A en fonction du temps ')
plt.xlabel('temps (en min)')
plt.ylabel('A')
plt.xlim(-1, 1.1*np.max(t)),
plt.ylim(0,1.1*np.max(A))
plt.text(0,0.4,'Droite Monte Carlo A = a*t+b')    # ça permet de positionner le texte où on veut
plt.text(0,0.35,'a = '+'{:.4}'.format(a_moy))
plt.text(0,0.3,'b = '+'{:.4}'.format(b_moy)
plt.text(0,0.25,'u(a) = '+'{:.2}'.format(u_a))
plt.text(0,0.2,'u(b) = '+'{:.2}'.format(u_b))
plt.grid(linestyle="-.")
plt.legend()
plt.show()
```

7) Comment résoudre une équation simple par la méthode de dichotomie

La méthode de recherche par dichotomie permet d'approcher la solution d'une fonction f(x)=0.

Principe de la méthode

Soit deux valeurs a et b et la fonction f continue sur l'intervalle [a,b].
L'encadrement par a et b est tel que f(a) et f(b) sont de signes opposés. On recherche dans cet intervalle la ou les valeurs de x telles que f(x) = 0. Pour trouver la solution, on divise l'intervalle en deux parties égales avec comme milieu m = (a+b)/2. Si f(a)×f(m) > 0, la solution se trouve alors dans l'encadrement [m,b] sinon elle se trouve dans l'encadrement [a,m]. On réitère alors la recherche dans le nouvel encadrement jusqu'à ce que a−b soit inférieur à la précision voulue.



Code associé pour utiliser la dichotomie :

```
def dichotomie(f, a, b, epsilon):
    try:
        if f(a)*f(b) > 0:  # On vérifie l'encadrement de la fonction
            raise Exception("Mauvais choix de a ou b.")
        else:
            m = (a + b)/2.
            while abs(a - b) > epsilon:
                if f(m) == 0.:
                    return m
                elif f(a)*f(m) > 0:
                    a = m
```

```
            else:
                b = m
            m = (a + b)/2
        return m
    except:
        print("Mauvais choix de l'encadrement")
        return 0
```

Il reste désormais à définir la fonction f en question et à résoudre l'équation.

Application :

Soit la réaction d'estérification suivante pour laquelle K° = 2,5 à 298 K.

$CH_3COOH_{(l)} + CH_3CH_2OH_{(l)} = CH_3COOCH_2CH_{3(l)} + H_2O_{(l)}$

Etat initial :	3	2	1	0	en moles

Tous les constituants ont pour activité $a_i = x_i$.
Déterminer la valeur de l'avancement molaire à l'équilibre $\xi_{éq}$ à l'aide de la méthode de la dichotomie.

Solution :

Il faut déjà établir l'équation à résoudre pour trouver la valeur de $\xi_{éq}$:

$CH_3COOH_{(l)} + CH_3CH_2OH_{(l)} = CH_3COOCH_2CH_{3(l)} + H_2O_{(l)}$

Etat initial :	3	2	1	0	en moles
Etat d'équilibre :	$3 - \xi_{éq}$	$2 - \xi_{éq}$	$1 + \xi_{éq}$	$\xi_{éq}$	

A l'équilibre, on a la relation de Guldberg-Waage qui est vérifiée :

$$K° = Q_{r,éq} = \frac{\left(x_{CH_3COOC_2H_5}\right)\left(x_{H_2O}\right)}{\left(x_{CH_3COOH}\right)\left(x_{C_2H_5OH}\right)} = \frac{\left(n_{CH_3COOC_2H_5}/n_{total}\right)\left(n_{H_2O}/n_{total}\right)}{\left(n_{CH_3COOH}/n_{total}\right)\left(n_{C_2H_5OH}/n_{total}\right)} = \frac{n_{CH_3COOC_2H_5}.n_{H_2O}}{n_{CH_3COOH}.n_{C_2H_5OH}}$$

Soit $K° = \dfrac{(1+\xi_{éq})\xi_{éq}}{(3-\xi_{éq})(2-\xi_{éq})}$

On la réécrit sous la forme $f(\xi_{éq}) = 0$:

$$K°(3-\xi_{éq})(2-\xi_{éq}) - (1+\xi_{éq})\xi_{éq} = 0$$

On peut désormais coder :

```
import numpy as np

K = 2.5       # valeur de K° à 298 K
#dans la suite, je nomme ksi avec la lettre x

def f(x):
   return K*(3-x)*(2-x)-(1+x)*x

a = 0         # valeur minimale de x
b = 2         # valeur maximale de x
precision = 0.001

#programme de dichotomie pour trouver x
def dichot(f,a,b,precision) :
   try:
      if f(a)*f(b) > 0:  # On vérifie l'encadrement de la fonction
         raise Exception("Mauvais choix de a ou b.")
      else:
         m = (a + b)/2
         while abs(a - b) > precision:
            if f(m) == 0.:
               return m
            elif f(a)*f(m) > 0:
               a = m
            else:
               b = m
            m = (a + b)/2
         return m
   except:
      print("Mauvais choix de l'encadrement")
      return 0

x1 = dichot(f, 0, 2, 0.001)
print("La valeur de ksi est:", x1,"mol")
```

On trouve 1,29 mol.

Alternative proposée par un collègue utilisant la fonction bisect (qui évite de taper soi-même le programme sur la dichotomie) :
https://jlamerenx.fr/wp-content/uploads/2020/11/R%c3%a9solution_dune_%c3%a9quation_%c3%a0_une_inconnue.html

8) Comment résoudre un système d'équation différentielle par la méthode d'Euler

Pour un tutoriel en vidéo : https://www.youtube.com/watch?v=-d7qrNkPDtQ
On va commencer avec un exemple simple où il y a une seule équation différentielle.

Exemple :
Soit un mécanisme réactionnel constitué d'un seul acte élémentaire :

- A→B de constante de vitesse k_1

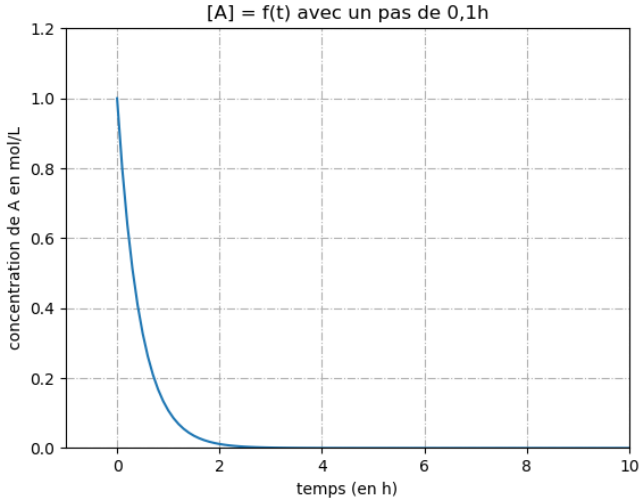
Données : $k_1 = 2\ h^{-1}$; $[A]_0 = 1\ mol.L^{-1}$ et $[B]_0 = 0\ mol.L^{-1}$

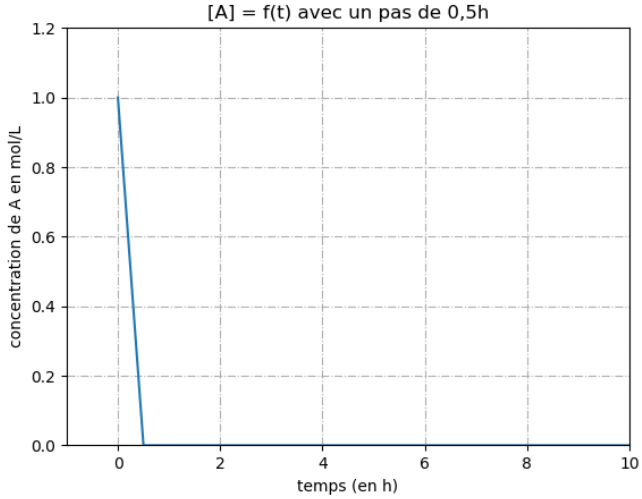
Etablissement de l'équation différentielle :

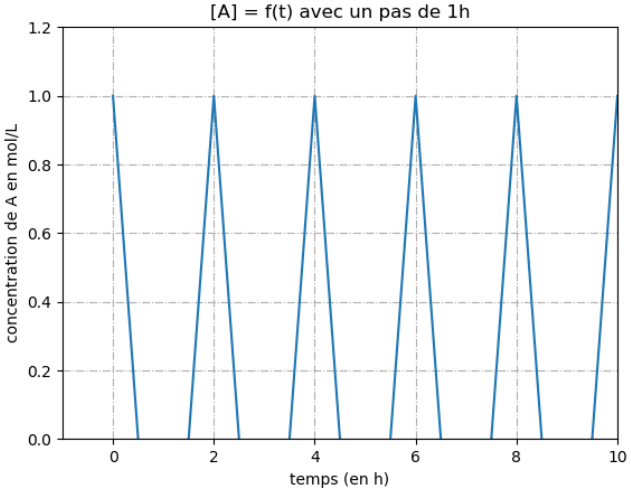
$$\frac{d[A]}{dt} = -v_1 = -k_1[A]$$

Code associé : avec un pas de 0,1 h

```
import matplotlib.pyplot as plt
import numpy as np

k1 = 2  #h-1
t_i = 0 #h
t_f = 10 #h
A0 = 1 #mol/L

def F1(A,t):
    return -k1*A

def euler(F, A0, dt, t_f):
    ts = [0]  #initialisation de t
    As = [A0] #initialisation de A
    while ts[-1] < t_f:
        t = ts[-1] + dt
        A = As[-1] + F(As[-1], ts[-1])*dt
        ts.append(t)
        As.append(A)
    return ts, As

#Appel de la fonction Euler et utilisation :
lt, lA = euler(F1,1,0.1,10)
plt.plot(lt,lA)
plt.xlabel('temps (en h)')
plt.ylabel('concentration de A en mol/L')
plt.title('[A] = f(t) avec un pas de 0,1h')
plt.xlim(-1, 10),
plt.ylim(0,1.2)
plt.grid(linestyle="-.")
plt.show()
```







Que concluez-vous sur le choix du pas ?

Autre exemple avec un système d'équations différentielles :

Soit un mécanisme réactionnel constitué de deux réactions successives :

- A→B de constante de vitesse k_1
- B→C de constante de vitesse k_2.

Données : $k_1 = 2\ h^{-1}$; $k_2 = 0{,}4\ h^{-1}$ et $[A]_0 = 1\ mol.L^{-1}$ et $[B]_0 = [C]_0 = 0\ mol.L^{-1}$

Application :

Etablir le système de trois équations différentielles vérifiées par [A], [B] et [C].

Résoudre ce système à l'aide de la méthode d'Euler sur une durée de 10h.

Solution :

Etablissement du système d'équations différentielles :

$$\frac{d[A]}{dt} = -v_1 = -k_1[A]$$

$$\frac{d[B]}{dt} = v_1 - v_2 = k_1[A] - k_2[B]$$

$$\frac{d[C]}{dt} = v_2 = k_2[B]$$

Code associé :

```
# 1) Recherche des bibliothèques
import matplotlib.pyplot as plt
import numpy as np

# 2) Saisie des données
k1 = 2 #h-1
k2 = 0.4  #h-1
a0, b0, c0 = 1,0,0  #mol.L-1

# 3) Définition du système d'équations différentielles
t_i = 0  #h
t_f = 10 #h
h=0.01
n = int((t_f-t_i)/h)

# Définition de F telle que dY/dt = F(Y)
def F(Y):
```

```
  return np.array([-k1*Y[0],k1*Y[0] - k2*Y[1], k2*Y[1]])

# Initialisation de YY tableau à 3 lignes et n colonnes (3,n)
YY = np.zeros((3,n))

# Initialisation des temps
T = np.zeros(n)

# Calcul de YY et de T
YY[:,0]=[a0,0,0] #fixe la valeur a0 au premier élément de la liste associée à A
T[0]=t_i
for i in range(n-1):
  YY[:,i+1] = YY[:,i]+h*F(YY[:,i])
  T[i+1]=T[i]+h

# 4) Tracé des courbes

plt.plot(T,YY[0],'k')
plt.plot(T,YY[1],'r',linestyle='--')
plt.plot(T,YY[2],'b',linestyle='-')
plt.grid()
plt.legend(["[A]","[B]","[C]"])
plt.xlabel('t en h')
plt.ylabel('[] en mol.L-1')
plt.show()
```

Alternative proposée par un collègue :
https://jlamerenx.fr/wp-content/uploads/2020/11/BCPST1_Cin%c3%a9tique_microscopique.html