Capacités numériques en chimie

I) Tracé de graphes

1) Généralités

Bibliothèques importées :

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

La bibliothèque numpy sert surtout à l'utilisation de la commande linspace :

```
np.linspace(borne_inferieure , borne_superieure , nombre de points)
```

Utilisation :

Pour chaque courbe, il est possible d'imposer une couleur, un marqueur, une épaisseur, une étiquette sous la forme :

```
plt.plot(X,Y,'CouleurMarqueur')
```

- Couleurs :

blue	red	green	black	yellow
b	r	g	k	y

- Marqueurs :

+	x	.	o	-	--
Croix verticale	Croix oblique	point	Rond plein	Trait plein	Pointillé

- Ajouts :

Numérotation de la figure	`plt.figure(1)`
Ajout d'un **titre**	`plt.title('Titre')`
Ajout de **texte**	`plt.text(x,y,'blabla')`
Ajout d'une **grille**	`plt.grid()`
Ajout d'une **étiquette d'abscisse**	`plt.xlabel('Titre abscisse')`
Contrôle des **bornes de tracé** en abscisse	`plt.xlim(borne_inf,borne_sup)`
Ajout d'une **légende**	`plt.legend()`
Afficher la figure	`plt.show()`

2) Utilisation

Pour un TP de suivi cinétique avec des espèces colorées, on peut par exemple mesurer l'absorbance de la solution au cours du temps. Voici le tableau de mesures :

t (min)	0	2	4	6	8	10	12	14	16
A	1,417	0,982	0,722	0,538	0,401	0,293	0,216	0,159	0,118

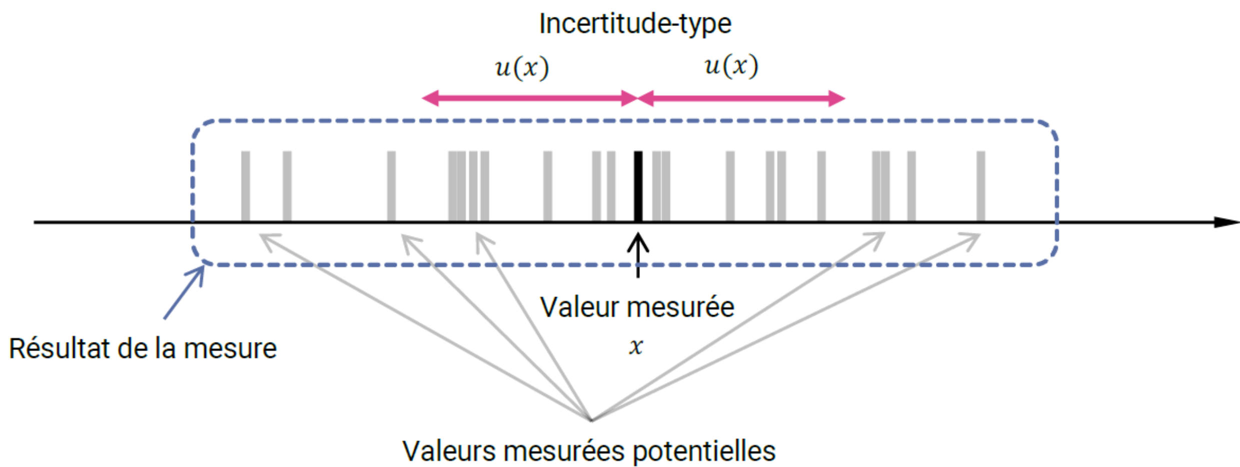
Tracer l'absorbance en fonction du temps.

```
import matplotlib.pyplot as plt
```

II) Les incertitudes

1) Définitions

Incertitude-type : estimation à l'aide d'un d'écart-type, de la dispersion des valeurs raisonnablement attribuables à la grandeur mesurée. La valeur mesurée est une de ces valeurs.



Incertitudes de type A : elles sont évaluées lorsqu'on procède à plusieurs fois la même mesure. La variabilité correspond alors à leur dispersion. Pour évaluer, l'incertitude-type *u(x),* on utilise

l'écart-type *s(x)* de l'ensemble des observations réalisées. Mathématiquement, on a donc :

$$u(\bar{x}) = \frac{s_x}{\sqrt{N}},$$

où *N* est le nombre d'observations utilisées pour le calcul de la valeur moyenne de x.

Incertitudes de type B : elles sont évaluées pour une observation unique. Pour calculer l'inceritititude-type u(x) de l'ensemble des valeurs comprises dans l'intervalle défini, on fait l'hypothèse que ces valeurs y sont equi-réparties autour d'une valeur moyenne m selon $\pm a$:

$$u(x) = \frac{a}{\sqrt{3}} \quad .$$

2) Incertitudes-types composées

Cas	Relation	Incertitude
1	$X = \lambda Y$ (λ constante)	$u(X) = \lvert\lambda\rvert \cdot u(Y)$
2	$X = Y + Z$ ou $X = Y - Z$	$u(X) = \sqrt{u(Y)^2 + u(Z)^2}$
3	$X = Y/Z$ ou $X = Y \cdot Z$	$u(X) = \lvert X\rvert \sqrt{\left(\frac{u(Y)}{Y}\right)^2 + \left(\frac{u(Z)}{Z}\right)^2}$
4	$X = \lambda Y^a Z^b$	$u(X) = \lvert X\rvert \sqrt{a^2\left(\frac{u(Y)}{Y}\right)^2 + b^2\left(\frac{u(Z)}{Z}\right)^2}$

3) Méthode de Monte-Carlo

Principe

La méthode de Monte-Carlo consiste à simuler numériquement la répétition de l'expérience. L'écart-type de l'ensemble des valeurs obtenues lors des ces répétitions fournit l'incertitude-type recherchée.

Etapes de la méthode

- Lister les grandeurs expérimentales utiles et associer à chacune, un intervalle au sein duquel on peut raisonnablement penser que celle-ci appartient.
- Faire procéder à un tirage au sort aléatoire d'un jeu de valeurs pour chaque grandeur expérimentale.
- Stocker la valeur dans une liste de résultats,
- Calculer la moyenne des valeurs de la grandeur mesurée
- Calculer l'écart-type de l'ensemble des valeurs : elle correspond à l'incertitude-type associée à la grandeur mesurée.

Fonctions pour réaliser le tirage au sort

La bibliothèque **numpy** est ici utilisée pour simuler un processus aléatoire :

```
numpy.random
```

Pour réaliser le tirage au sort :

Distribution rectangulaire	numpy.random.uniform(borne_inf , borne_sup, size)
Distribution normale	numpy.random.normal(valeur centrale , incertitude-type, size)

La bibliothèque **matplotlib.pyplot** permet d'utiliser la fonction **hist** pour représenter les résultats d'un ensemble de tirages d'une variable aléatoire :

```
plt.hist(données, bins='rice', range=[borne_inf , borne_sup])
```

4) Représenter le résultat d'un ensemble de tirages aléatoires

Soit un prélèvement de 10 mL avec une burette jaugée où il est précisée une tolérance de 0,03 mL. Proposer un programme pour afficher les deux types de distributions possibles.

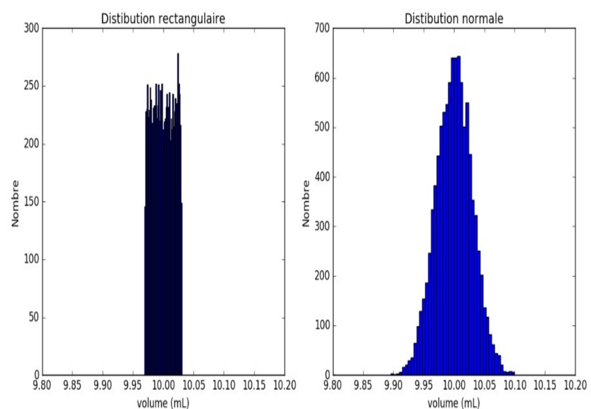
Remarque : on peut afficher plusieurs histogrammes avec la commande **subplot** de **matplotlib.pyplot** :

```
subplot(nrows, ncols, index)
```

```
import numpy as np

import matplotlib.pyplot as plt

Nsim = 10000
data = np.random.uniform(9.97, 10.03, Nsim)
data2 =np.random.normal(10,0.03, Nsim)
plt.subplot(1,2,1)
plt.hist(data, bins='rice', range=[9.8, 10.2])
plt.xlabel('volume (mL)')
plt.ylabel('Nombre')
plt.title('Distibution rectangulaire')
plt.subplot(1,2,2)
plt.hist(data2, bins='rice', range=[9.8, 10.2])
plt.xlabel('volume (mL)')
plt.ylabel('Nombre')
plt.title('Distibution normale')
plt.show()
```



5) **Calcul de l'incertitude type d'un titrage**

Soit un dosage de A par B avec :

- [B] = 0,10 mol/L, avec une incertitude-type de 0,001 mol/L,
- V_{eq} = 10 mL avec une incertitude-type de 0,5 mL,
- V_0 = 10 mL avec une incertitude-type de 0,03 mL.

Proposer un programme pour déterminer [A] ainsi que son incertitude-type.

Remarques :

- Pour réaliser une moyenne, on peut utiliser la fonction **mean** de **numpy**.
- La fonction **numpy.std(L, ddof=1)** permet d'avoir l'écart-type sur la liste L (ddof : Delta Degrees of Freedom, on choisit ddof=1 pour le calcul de la variance)

```
import numpy as np

N=1000
A = []
for k in range(N) :
```

III) Régressions linéaires

1) Méthode

L'objectif est de tester la validité d'une loi physique par le tracé d'un nuage de points et sa modélisation par une droite. On utilise les bibliothèques **numpy** et **matplotlib.pyplot**.

On utilise la fonction **polyfit** de la bibliothèque numpy pour vérifier si le nuage de points (x,y) est modélisable par une fonction polynomiale de degré deg :

```
numpy.polyfit(x,y,deg)
```

Pour une régression linéaire (deg=1), la fonction numpy.polyfit renvoie une liste de deux informations :

[pente, ordonnée à l'origine].

Le coefficient de corrélation R peut être obtenu à l'aide de la fonction **corcoeff** de **numpy** :

```
numpy.corrcoef(x, y)
```

Il renvoie une matrice avec les coefficients de corrélation :

R_{xx}	R_{xy}
R_{yx}	R_{yy}

2) Barres d'erreur

Le tracé d'une droite de modélisation avec les barres d'erreur se fait à l'aide de la fonction **errobar** de **matplotlib.pyplot**.

```
matplotlib.pyplot.errorbar(x, y, yerr=None, xerr=None, fmt=' ', )
```

On précise les valeurs des barres d'erreur l'abscisse (xerr) et l'ordonnée (yerr).

3) **Vérification d'un modèle linéaire**

On souhaite connaître la quantité de colorant alimentaire E131 dans un bonbon gélifié. Pour cela, on souhaite mesurer l'absorbance d'un solution dans laquelle le bonbon aura été dissout. On prépare cinq solutions étalonnées en concentration de E131.

C (mol/L)	2.10^{-5}	$1,5.10^{-5}$	1.10^{-5}	5.10^{-6}	2.10^{-6}
A	1,719	1,372	0,768	0,442	0,201

Vérifier que la loi de Beer-Lambert est vérifiée.

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

c=np.array([2*10**-5 , 1.5*10**-5 , 1*10**-5 , 5*10**-6 , 2*10**-6])
A=[1.719 , 1.372 , 0.768 , 0.442 , 0.201]
```

IV) Méthode de la dichotomie

1) Principe

On considère une fonction f : I = [a, b] → R, continue et strictement monotone sur l'intervalle [a, b]. De plus, on suppose que f change de signe sur l'intervalle.
On souhaite déterminer une valeur approchée de c tel que $f(c) = 0$.
On cherche la solution approchée à l'équation $f(x) = 0$ sur l'intervalle [a, b], où f est une fonction strictement monotone.

On se place au milieu de l'intervalle [a, b] ($c = (a + b)/2$) et on compare le signe de $f(a)$ avec le signe de $f(c)$ en regardant le signe de $f(a).f(c)$.

- Si $f(a).f(c) < 0$, alors f s'annule sur l'intervalle [a, c]. Et on procède de la même façon sur ce nouvel intervalle.
- Sinon, f s'annule sur l'intervalle [c, b]. Et on procède de la même façon sur ce nouvel intervalle.

On procède ainsi jusqu'à un certain critère imposé, qui peut porter soit sur la largeur de l'intervalle de recherche (on cherche jusqu'à ce que $|b - a|$ devienne inférieur à un certain ε), soit sur la proximité de f à 0 (on cherche jusqu'à ce que $|f(c)|$ devienne inférieur à un certain ε).

Comment choisir l'intervalle [a, b] de recherche ?

- Représenter la fonction f dont on cherche l'annulation ;
- Déterminer un intervalle pertinent, sur lequel f est strictement monotone et s'annule .

2) **Recherche de l'avancement à l'équilibre**

Soit une solution à 0,1 mol/L d'acide éthanoïque. Déterminer, par dichotomie, l'avancement de la réaction. On donne pKa = 4,8.

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

def f(x):
    return (0.1-x)*10**-4.8-x**2

x = np.linspace(0,0.1,1000)

plt.plot(x,f(x),'b-')
plt.ylim(-0.00001,0.000001)
plt.xlim(0,0.01)
plt.show()

def dichot(f,a,b,epsilon):
```

3) Utilisation de la bibliothèque scipy.optimize

Fonction bisect

Le module **optimize** de la bibliothèque **scipy** offre plusieurs fonction pour rechercher les racines d'une fonction. D'autre part, selon les besoins, des fonctions de la bibliothèque math peuvent s'avérer utiles.

La commande **bisect** opère par dichotomie. Sa syntaxe s'écrit :

```
bisect(fonction,borne_inf,borne_sup)
```

Pour que celle-ci fonctionne, il est nécessaire de :

- définir préalablement une fonction f(x),
- préciser un intervalle [a,b] à l'intérieur duquel rechercher la racine de f,
- s'assurer que f(a) et f(b) sont de signes opposés.

Application

Soit une solution à 0,1 mol/L d'acide éthanoïque. Déterminer, à l'aide de la commande bisect, l'avancement de la réaction. On donne pKa = 4,8.

```
import scipy.optimize as op
import numpy as np

def f(x) :
    return 10**-4.8*(0.1-x)-x**2
print('Solution par la fonction bisect : x =', op.bisect(f,0,0.1))
```

V) Méthode d'Euler

1) Principe

L'équation différentielle d'ordre 1 à résoudre est du type :

$$\frac{d\,y(t)}{d\,t}+a\,.\,y(t)=b$$, a et b étant des constantes.

On suppose que la valeur initiale y(0) = y_o est connue.

La méthode d'Euler consiste à déterminer les valeurs $y(t_i)$ de la fonction pour des dates t_i successives en utilisant l'approximation :

$$(1)\frac{d\,y(t_i)}{d\,t}=\frac{y(t_{i+1})-y(t_i)}{t_{i+1}-t_i}\,.$$

A partir de la valeur $y(t_i)$ de la fonction à une date t_i :

- on calcule $\frac{d\,y(t_i)}{d\,t}$ à partir de l'équation différentielle
- on en déduit $y(t_{i+1})$ à partir de la formule (1)

Dans le programme , on définit donc :

- les conditions initiales
- la durée t_{max} sur laquelle on veut tracer la solution
- l'expression de chaque valeur $y(t_{i+1})$
- le tracé de la courbe

2) Application à la cinétique

Soit la désintégration nucléaire d'un atome de ^{14}C selon une cinétique d'ordre 1. Le temps de demi vie est de 5730 ans. Tracer l'évolution du nombre d'atomes de carbone 14, N, en fonction du temps.

```
import numpy as np
import matplotlib.pyplot as plt
T = 5730
k = np.log(2)/T
N_0=100
N=N_0
dt = 100
t = 0
tmax = 40000
while (t<tmax) :
```

3) Utilisation de odeint

Principe

L'équation différentielle d'ordre 1 à résoudre est du type :

$$\frac{d\,y(t)}{d\,t} + a\,.\,y(t) = b$$, a et b étant des constantes.

Le principe est de définir une fonction dérivée F(y,t) = b-a.y(t) et la fonction **odeint** du package **scipy.integrate** va résoudre tout ça :

```
from scipy.integrate import odeint
odeint(func, y0, t)
```

On aura besoin d'utiliser une liste de temps t à l'aide de la fonction **linspace** de **numpy**.

Application à la cinétique

Soit la désintégration nucléaire d'un atome de ^{14}C selon une cinétique d'ordre 1. Le temps de demi vie est de 5730 ans. Tracer l'évolution du nombre d'atomes de carbone 14, N, en fonction du temps.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
T = 5730
k = np.log(2)/T
t = np.linspace(0,80000,100)
def F(N,t):
    return -k*N
y = odeint(F,100,t)
plt.plot(t,y,"bo")
plt.show()
```